Improving Interaction in Visual Analytics using Machine Learning

Chaoran Fan

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2021

UNIVERSITY OF BERGEN

Improving Interaction in Visual Analytics using Machine Learning

Chaoran Fan

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 05.11.2021

Year: 2021

Title: Improving Interaction in Visual Analytics using Machine Learning

Name: Chaoran Fan

Print: Skipnes Kommunikasjon / University of Bergen

Scientific environment

The work presented in this thesis was conducted as a part of my PhD studies at the Department of Informatics, University of Bergen. In addition, I have been enrolled in the ICT Research School at the Department of Informatics, University of Bergen. During the research on my thesis, parts of my work have been done in the context of CEDAS, Center for Data Science, at the University of Bergen and the Computer Graphics and Visualization group (CGV) at the Delft University of Technology.

UNIVERSITY OF BERGEN

UNIVERSITAS BERGENSIS

Research School In
Information and Communication Technology
ICT

CEDAS
Center for Data Science

TU Delft
Delft University of Technology

Acknowledgements

Time flies, there are so many thanks I would like to say to people who helped and accompanied me in the past 4.5 years.

First and foremost, I would like to express my deep gratitude to my advisor Prof. Helwig Hauser for the constant support during my PhD study. He has offered me valuable ideas, guidance and suggestions with his profound knowledge and rich research experience. In my eyes, he is an encyclopedia and it is always beneficial and enjoyable to talk with him. In addition, I do appreciate his patience and painstaking efforts to help me revise and polish paper drafts for the past few years. Truly, I could not imagine that the completion of the present thesis is possible without his tremendous assistance, he always let me feel that I am not alone in my Ph.D journey. I am also very thankful to my co-supervisor Krešimir Matković, whose incisive comments and valuable suggestions have greatly improved my submissions.

My appreciation also extends to my lovely colleagues in Bergen Visualization group: Andreas Lind, Eric Mörth, Fabian Bolte, Fourough Gharbalchi, Ivan Kolesar, Jan Byška, Julius Parulek, Juraj Pálenik, Laura Garrison and Oli, M. Eduard Gröller, Noeska Smit, Sergej Stoppel, Sherin Sugathan, Stefan Bruckner, Thomas Trautner, Veronika Šoltészová, Yngve Sekse Kristiansen, Åsmund Birkeland. We come from different countries with different cultural backgrounds, the talks with you made me a growth of knowledge and greatly broadened my horizon. I do enjoy the time that we have meals together and share our life and experience, treating each other just like a whole family. I am so grateful to have you working around me in the office.

My thanks are also due to my friends who I spent the most time with in my spare time: Aksel Heitman Olsen, Bowen Sun, Chengcheng Wang, Dan Zhang, Guyu Peng, Hua Dong, Hui Huang, Jie Liu, Junjie Cai, Kui Xiang, Kaiqing Yang, Miao Teng, Morten Meland, Rui Li, Runxi Niu, Sisi Zheng, Shihao Wei, Tsai-Ming Lu, Wanmei Zhang, Xianglian Hu, Xiaoshuang Li, Xiaokang Zhang, Xiaozheng Liu, Xi Lan, Yuejia Wang, Yue Gao, Yufei Yuan. We travel, cook, drink, game, play sports and talk together, those beautiful memories I will never forget.

In addition, I also want to thank everyone I met during the 1 year exchange in Computer Graphics and Visualization group at TU Delft: Ahmad Nasikun, Anna Vilanova, Changgong Zhang, Christopher Brandt, Elmar Eisemann, Jerry Guo, Jingtang Liao, Klaus Hildebrandt, Leonardo Scandolo, Markus Billeter, Nestor Salamon, Nicola Pezzotti, Niels de Hoon, Peiteng Shi, Rafael Bidarra, Thomas Höllt, Thomas Kroes, Timothy R. Kol, Victor Petitjean. I often miss about the carefree life there where the tulips were in full bloom and the sails of the windmill were wheeling round.

Special thanks should go to my family, I am indebted to my beloved parents for their encouragement and unconditional support.

Above ground, thanks fate, lets us be acquainted in the boundless crowds.

Abstract

Interaction is one of the most fundamental components in visual analytical systems, which transforms people from mere viewers to active participants in the process of analyzing and understanding data. Therefore, fast and accurate interaction techniques are key to establishing a successful human-computer dialogue, enabling a smooth visual data exploration. Machine learning is a branch of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. It has been utilized in a wide variety of fields, where it is not straightforward to develop a conventional algorithm for effectively performing a task. Inspired by this, we see the opportunity to improve the current interactions in visual analytics by using machine learning methods.

In this thesis, we address the need for interaction techniques that are both fast, enabling a fluid interaction in visual data exploration and analysis, and also accurate, i.e., enabling the user to effectively select specific data subsets. First, we present a new, fast and accurate brushing technique for scatterplots, based on the Mahalanobis brush, which we have optimized using data from a user study. Further, we present a new solution for a near-perfect sketch-based brushing technique, where we exploit a convolutional neural network (CNN) for estimating the intended data selection from a fast and simple click-and-drag interaction and from the data distribution in the visualization. Next, we propose an innovative framework which offers the user opportunities to improve the brushing technique while using it. We tested this framework with CNN-based brushing and the result shows that the underlying model can be refined (better performance in terms of accuracy) and personalized by very little time of retraining. Besides, in order to investigate to which degree the human should be involved into the model design and how good the empirical model can be with a more careful design, we extended our Mahalanobis brush (the best current empirical model in terms of accuracy for brushing points in a scatterplot) by further incorporating the data distribution information, captured by kernel density estimation (KDE). Based on this work, we then provide a detailed comparison between empirical modeling and implicit modeling by machine learning (deep learning). Lastly, we introduce a new, machine learning based approach that enables the fast and accurate querying of time series data based on a swift sketching interaction. To achieve this, we build upon existing LSTM technology (long short-term memory) to encode both the sketch and the time series data in two networks with shared parameters.

All the proposed interaction techniques in this thesis were demonstrated by application examples and evaluated via user studies. The integration of machine learning knowledge into visualization opens further possible research directions.

List of papers

This thesis is based on the following publications:

(A) **Chaoran Fan** and Helwig Hauser. **User-study based optimization of fast and accurate Mahalanobis brushing in scatterplots**. In *Proc. Vision, Modeling, and Visualization (VMV 2017)*, pages 77–84, 2017.

(B) **Chaoran Fan** and Helwig Hauser. **Fast and Accurate CNN-based Brushing in Scatterplots**. *Computer Graphics Forum (Eurovis 2018)*, 37 (3): 111–120, 2018.

(C) **Chaoran Fan** and Helwig Hauser. **Personalized Sketch-Based Brushing in Scatterplots**. *IEEE Computer Graphics and Applications*, 39 (4): 28–39, 2019.

(D) **Chaoran Fan** and Helwig Hauser. **On sketch-based selections from scatterplots using KDE, compared to Mahalanobis and CNN brushing**. *IEEE Computer Graphics and Applications*, 41 (5): 67–78, 2021.

(E) **Chaoran Fan**, Krešimir Matković and Helwig Hauser. **Sketch-based fast and accurate querying of time series using parameter-sharing LSTM networks**. *IEEE Transaction on Visualization and Computer Graphics*, Early Access, 2020.

The following paper is also related to this thesis:

(1) **Chaoran Fan** and Helwig Hauser. **On KDE-based brushing in scatterplots and how it compares to CNN-based brushing**. In *Proc. Machine Learning Methods in Visualisation for Big Data (MLVis 2019)*, 2019.

All the related publications from A to E were written during the Ph.D. studies and the author of the thesis is the main author of them. All papers were written in collaboration with Helwig Hauser, who is the main supervisor of the main author. He contributed with guidance, advice and fruitful discussion to the realization and publication of the scientific work. Paper E was co-authored with Krešimir Matković, who is the co-supervisor of the thesis author and he provided some valuable suggestions to improve the paper.

Contents

Part I

Overview

Chapter 1

Introduction

With the advance of new data acquisition and generation technologies, our society is becoming increasingly information-driven. Thus, understanding the information in large and complex data sets has been in the focus of several research fields such as statistics, data mining, machine learning, and visualization. The first three fields predominantly rely on computational power while visualization is dependent mainly on our human perceptual and cognitive capabilities for extracting information. Visualization is a popular way to explore and communicate data via the use of interactive visual encodings. The purpose of visualization is to help people better understand the data. For example, plotting a graph instead of inspecting a table with numbers is one of the simplest examples of an effective visualization solution.

Visual analytics is an emerging field for dealing with the complexity of an ever increasing information space by combining the processing power and storage capacity of computers with the intellectual strengths of humans through interactive visual interfaces. It is a powerful tool for problems where the size and complexity of the data requires human input along with machine analysis. In visual analytics, a tight feedback loop of computation/visualization and user interaction is commonly used to facilitate knowledge discovery in complex datasets. Humans are engaged with analysis models through interactive visualization with the underlying data, making them capable of applying domain knowledge to iteratively refine models. This user-in-the-loop methodology allows the user to explore deeper into the data to continuously build and apply knowledge.

Machine learning is a pervasive science which we use dozens of times each day without realizing it. The main goal of machine learning is to provide a system that can automatically obtain deep insights, recognize unknown patterns, and create high performing predictive models from data without being explicitly programmed. In the past decade, many machine learning-based applications have appeared in our daily life, such as self-driving cars, speech recognition and translation, web search and even the human genome project. Inspired by the great success, making use of machine learning to achieve more efficient and effective visual analytics solutions is becoming a new trend recently.

1.1 Problem Statement

Interaction between humans and computers is at the heart of modern visual analytics. It enables user to develop and understand relationships within datasets through foraging and synthesis.

As we know, machine learning has been applied to a wide range of fields and achieved a remarkable success over the years. Inspired by this, the research presented in this thesis is to exploit machine leaning to improve the traditional interaction techniques in visual analytics. More specifically, we are aiming to develop advanced interaction mechanisms that incorporate the power of machine learning-based methods to make the data exploration more efficient and effective.

In this thesis, we focus on improving two commonly used interaction techniques in visual analytics—brushing and sketch-based visual querying. Generally, for designing a interaction technique, two particularly important criteria should be taken into account:

- *efficiency*—is the interaction fast enough (including the interaction and all computation) to enable a fluid data exploration/analysis [20, 98]?
- *accuracy*—how accurately does the interaction lead to a result, which the user wished to achieve?

Linking and brushing is a prevalent interaction technique for data exploration and analysis in coordinated multiple views. The concept was first defined by Becker and Cleveland [2] as an interactive method for selecting data points in a visualization by drawing simple geometries onto it. A key functionality in coordinated multiple views is that brushing leads to a consistent highlighting of the selected data in all linked views. This results in the most common form of focus+context visualization [35], enabling the fast and effective exploration of data relations, which are too challenging to show in just one view.

Despite the rich variation of existing brushing tools, we rarely see a solution that combines both criteria really well: Many brushing techniques are indeed fast, as clicking on one point, for example, or drawing simple geometries—also sketched brushes are fast, requiring only a simple gesture as interaction and thus enabling a swift user–computer dialogue during the exploration/analysis [8]. A common disadvantage of fast techniques, however, is that it can be difficult to accurately brush a particular data subset according to the user's intention.

On the other hand, we certainly find brushing techniques, that are straight-forward for accurately selecting subsets of interest, such as lassoing and the logical combination of simple brushes. This benefit of being accurate, however, commonly comes at the cost of reduced efficiency—specifying a lasso, for example, easily becomes a unit task by itself [8], potentially interrupting the exploration/analysis process. In our work, we aim to integrate both criteria in one technique as good as possible to improve the current brushing techniques.

A visual query system is designed to find patterns of interest in data as it is easier for human users to visually describe patterns than to express them textually or programmatically. For the exploration of large time series data, it is almost impossible for the analysts to visually identify specific patterns efficiently. To overcome this issue, a visual query system is commonly used to bridge the gap between the user and the computer, in which the metaphor of freehand sketching is frequently employed as an efficient means of visual communication. Often, a carefully designed empirical model is adopted for estimating the similarity relation between the sketching interaction and the intended pattern selection. However, the often resulting non-optimal efficiency and low accuracy lead to the limited deployment of sketch-based visual query systems for real-world visual analytics problems.

More specifically in terms of the efficiency, most of these empirical methods are based on local characteristics and a sliding window is used to compute the best match or similarity ranking, which easily leads to a time-consuming procedure and makes the interactive exploration a tedious task. On the other hand, sketches are artistic depictions from humans. Due to the ambiguity and distortion existing in sketches, the empirical model is usually far from being stable to interpret their underlying semantics conveyed by the user. In this situation, the matching algorithms may fail to produce good similarity rankings when "goodness" is directly evaluated by the user [71].

Overall, to improve the current sketch-based querying, we see two main directions. First, in order to realize a fluent data exploration, we aim at a faster computation of the matching procedure. Second, we have to develop a better model which is able to understand the meaning of the user sketch, making the querying result as close as possible to the user's real desire.

1.2 Scope and Contributions

In this thesis, we make several serious attempts to take advantage of machine learning in order to improve interaction in visual analytics. The research presented in this thesis is motivated by successful machine learning-based applications in a wide range of fields as well as the challenges arising from model design for visualization techniques. More specifically, the main contributions of this thesis can be summarized as follows:

1. We present a new Mahalanobis brush, which we have extended and further optimized using data from a user study with 50 participants. This attempt contributes an improvement to a central procedure in many modern visual analytics solutions, i.e., to brushing (scatterplots). The user study-based optimization of visualization parameters is too little seen in the visualization literature. We could demonstrate quantitatively that we significantly improve the accuracy of the original Mahalanobis brushing while still using a very fast interaction technique.

2. By exploiting deep learning, we present a CNN-based technique for brushing in scatterplots, which provides a solution that is able to brush also nonlinear shapes of data subsets and significantly reduces the error rate when compared to our improved Mahalanobis brush. To the best of our knowledge, this is the first study to report the successful application of a structured regression model, realized by a convolutional neural network, to improve a central user interaction technique in visual analytics.

3. In order to optimize a brush tool for every single user, we present a personalized CNN-based brushing solution which takes the user in the loop to iteratively

refine the brushing model with additional data that the user provides while using the brushing technique. By refining a first general model for estimating data selections from simple click-and-drag interactions incrementally with the additional data from a particular user and leveraging the existing parameterization, we achieve a solution which is able to turn the general model based on people's average brushing preference to a tailored model for the specific user. In addition, the retraining time cost is largely reduced, thus only a short break is needed if the users want to improve their brushing models.

4. To investigate how much an empirical model can be furher improved with a more sophisticated design and whether it can outperform the deep learning approach, we present our attempt to construct a best-possible empirical model by further extending the Mahalanobis brush, incorporating kernel density estimation (KDE) with the goal to figure out the influence of human expertise during model design. The main contribution of this work includes our extension of the empirical model for brushing points in a scatterplot and a threefold comparison between empirical brushing models (Mahalanobis brushing and KDE brushing) and our deep learning-based brushing model (CNN brushing) as well as an according discussion.

5. We present an LSTM-based solution to improve visual querying of time series data in visual analytics. More specially, we make use of the long short-term memory (LSTM) to encode the user sketch and the time series data respectively in two networks with sharing parameters in order to learn the similarity function between them for matching purpose. To the best of our knowledge, this is the first time that deep learning is used to learn the matching relation between a human sketch and time series data, outperforming two state-of-the-art models (Qetch and DTW) in terms of accuracy and efficiency.

1.3 Thesis Structure

This thesis is composed of two main parts. The first part provides an overview of the research carried out within the course of this thesis. The second part contains individual publications, presented verbatim with only adjusted formatting to fit the layout of this thesis. Furthermore, the bibliographies of the individual publications and Part I were merged to a single unified bibliography.

The first part, namely the overview, is structured as follows: After the introduction (Chapter 1), a structured overview of related work is presented in Chapter 2, in which we discuss previous research work concerned with machine learning methods utilized in visualization as well as work on brushing techniques and visual query systems. In Chapter 3, the contributions of this thesis are outlined in more detail (for get more details, we refer to the paper in Part II). We evaluate the proposed ideas and methods in Chapter 4. Finally, we conclude the first part of this thesis in Chapter 5.

The second part of the thesis consists of five papers, providing further details on the contributions of this PhD work.

Chapter 2

Related work

During the last few years, substantial work has been done to integrate machine learning into visual analytics, leveraging the strengths from both sides to help users extract valuable information from the data. In this chapter, we outline the state of the art of the combination between machine learning and visualization. Then, we provide an overview of a central interaction technique used in visual analytics—brushing. Furthermore, we review the visual query systems for time series data as well as the relevant pattern matching algorithms.

2.1 Integrating machine learning into visual analytics

Visual analytics (VA) is a process which incorporates automatic and visual analysis methods with a tight coupling through human interaction in order to discover knowledge from data. Visual analytics can be seen as an integral approach combining visualization, human factors, and data analysis, where the user plays an important role in the communication between the human and the computer, as well as in the decision-making process. Figure 2.1 shows an overview of the different stages (represented by rectangles) and their transitions (arrows) in visual analytics process [50]. First, the typically heterogeneous data has to be preprocessed and transformed to suitable representations for further exploration via data cleaning, normalization, grouping and so on. Afterwards, visual or automatic analysis methods are applied and the analyst is allowed to evaluate and refine the models by interaction with the visualization (for example using zooming in on different data areas or considering different visual views on the data). The whole process can lead to a continuous refinement and verification of preliminary results and knowledge (insightful information) can be gained as a feedback to support the future analysis.

In the data-driven era, data is becoming more complicated and difficult as scales increase, driving the need for systems that enable to draw valid conclusions from data while maintaining trustworthy and interpretable results. Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. As the focus of machine learning and visualization is on algorithms and interfaces/interaction respectively, making use of the complementing strengths from both communities is becoming a trend to address the current challenges in visual data exploration. Over the years, much work has been

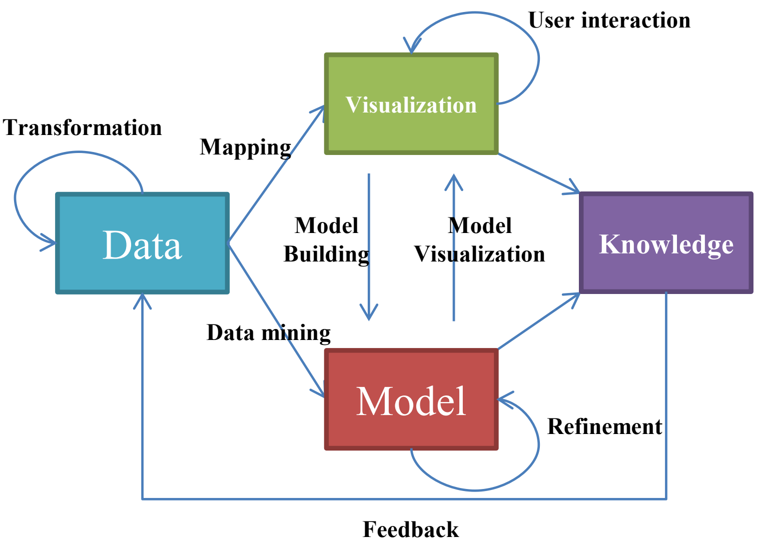


Figure 2.1: Overview of the visual analytics process [50].

done to integrate machine learning into visual analytics, in which the utilized machine learning methods can be roughly classified into two types: traditional machine learning and deep learning. In the following, we review important works in both categories as well as the popular machine learning frameworks applied in visual analytics.

2.1.1 Traditional machine learning for visualization

Fundamentally, traditional machine learning is using algorithms to extract information from raw data and represent it by constructing a certain type of model. The traditional machine learning community has achieved conspicuous progress in various kinds of tasks (such as clustering, regression or classification) over the past decades by bringing in the knowledge from statistics, data analysis and processing. As machine learning and visualization have the same goal, which is to help people get insight from the data, we have seen a mentionable number of work combining traditional machine learning with visual analytics solutions. Endert et al. [21] summarized the conventional machine learning techniques that have been integrated into VA applications, dividing them into 4 categories:

Dimension reduction—the analyst can use it to compress a large set of features into a new feature space of lower dimensionality without losing the most important information. This way, conventional visualization methods for moderate dimentionality can be employed. Williams and Munzner presented MDSteer (shown in Figure 2.2 (A)) [106] which is a steerable multidimensional scaling (MDS) system that can progressively compute an MDS layout and handle huge datasets. To achieve this, the high dimentional points are projected into a hierarchical decomposition of rectangular screen-space regions. Then the system allows user to interactively steer the computation to the regions where more precision is needed. Another classical example is

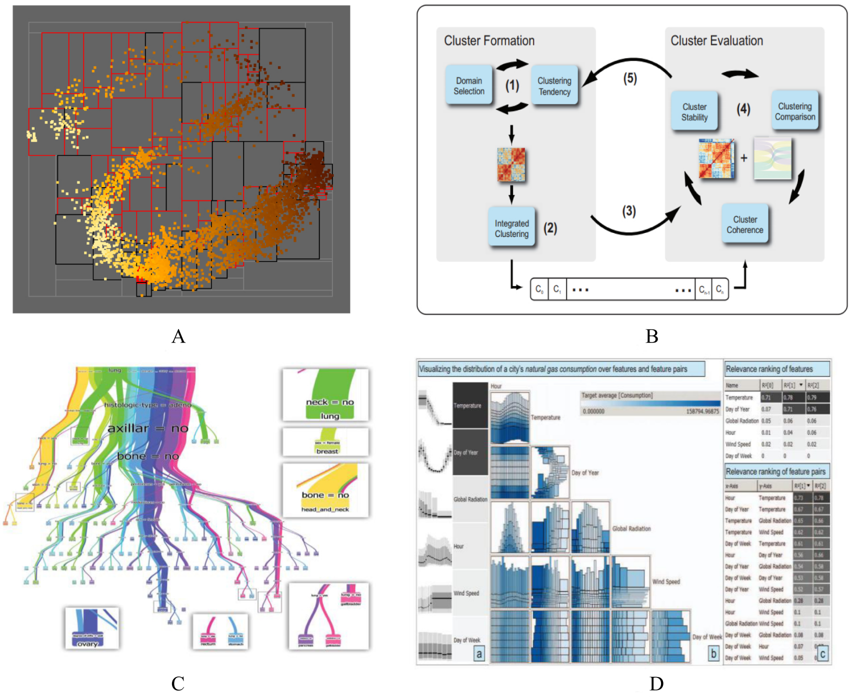


Figure 2.2: Examples of traditional machine learning methods utilized in visual analytics solutions: (A) MDSteer, a system enables the user to steer the computation to where it is needed for dimension reduction [106]. (B) Interactive system for allowing the user to refine the clustering criteria [99]. (C) Baobabview, enables the interactive construction and analysis of decision trees [100]. (D) An example of a visual analytics application which allows the user to refine the regression models by integrating domain knowledge [75].

iPCA (interactive PCA) [47] that visualizes the results of principle component analysis using multiple coordinated views (data view, correlation view, eigenvector view and projection view) and a rich set of user interactions, in order to assist the user in better understanding and utilizing PCA. User interactions in one view are immediately reflected in the others so that the user can easily identify a data item or a data dimension in the original data space and its counterpart in eigenspace. In addition, Johansson and Johansson [48] presented a visually guided system that allows the user to interactively reduce the data dimensionality with the help of user-defined and weighted quality metrics.

Clustering—A task of dividing the data into groups such that data in the same group are more similar to other data in the same group and dissimilar to the data in other groups. Rasmussen and Karypis [85] presented gCLUTO, which is an early example where multiple clustering algorithms are integrated to facilitate clustering-driven analysis of large datasets. The user can find clusters based on a number of analysis, reporting, and visualization tools, and the clustering results with different characteristics can be visually inspected. Turkay et al. [99] presented an interactive system (shown in Figure 2.2 (B)) that addresses both the generation and evaluation stages within the clustering process and provides interactive control to users to refine grouping criteria through investigations of measures of clustering quality. In addition, Hossain et al. [43] made use of a scattergather technique and iteratively introduce grouping constraints by breaking up or merging clusters and the results are user-optimized through interaction.

Classification, which is to identify the category a new observation belongs to, based on a training set of data containing observations whose category membership is known. van den Elzen and van Wijk [100] developed a system (shown in Figure 2.2 (C)) for the interactive construction and analysis of decision trees that takes advantage of the specific knowledge from the domain experts. More specially, the domain experts are supported to grow, prune (reducing search space), optimize and analyze decision trees. Krause et al. [58] focused on the feature selection within predictive model building process and presented a system that enables the analysts to interactively decide which particular features should be taken into account for a classification model. Moreover, Behrisch et al. [3] introduced a feedback-driven view exploration framework by integrating the users' relevance feedback which is approximated by a classification system. In the presented system, an iterative dialogue between the user and the algorithm is built, in which users communicate known/expected/wrong classification results back to the algorithm and the model then can iteratively learn the users' preference and find new interesting views to recommend.

Regression/correlation analysis methods are commonly used to investigate relations between features in the data and to understand/generate causal links to explain phenomena. Mühlbacher and Piringer [75] presented a framework (shown in Figure 2.2 (D)) which integrates domain knowledge to improve the process of building regression models. The framework is a combination of visualizing relationship structures in a qualitative analysis and a quantification of relevance for ranking any number of features. Malik et al. [70] developed a visual analytics solution for interactive auto-correlation, which enables users to discover correlations and explore potentially causal or predictive links at different spatiotemporal aggregation levels among the datasets. Matković et al. [73] realized a successful prototyping environment that tightly couples steering loop, integrating new simulation technology and interactive visualization for designing an injection system. The control variables of the simulation can be visually explored by the expert and then the simulation models can be assessed whether they are feasible or needed to be refined. A regression model is incorporated within this process to further optimize the simulation results based on users' interactive inputs.

The literature mentioned above introduces two types of user involvement in the combination of machine learning and visual analytics: (1.) the parameters and the settings of an algorithm are explicitly modified by the user via interaction and the users are allowed to steer the computational domain to which the algorithm is applied. (2.) users can apply their relevant knowledge to instruct or correct the algorithm to meet their expectation via directly communicating with the output. While progress has been made on incorporating machine learning into visual analytics to improve sensemaking and knowledge discovery, the smart and effective combination of these two fields are still under-explored, for example, decomposing the tasks for human and machine in a balanced way.

2.1.2 Deep learning for visualization

Deep learning (DL) is a subfield of machine learning, in which computational models with multiple processing layers are adopted to extract features from raw data and to discover the hierarchical representations needed for different kinds of tasks. In recent years, deep learning methods have been achieving breakthroughs in various major artificial intelligence tasks, especially image processing and natural language processing, attracting a lot of attention. In these tasks, deep neural networks reached a level of accuracy comparable to or even better than humans' performance. In general, the techniques used in deep learning can be categorized by their architecture, such as deep neural networks (DNNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs) and deep belief networks (DBNs). Among them, CNNs and RNNs are the most widely used and have achieved numerous state-of-the-art results.

A convolutional neural network (CNN) is a deep learning architecture, which is inspired by the connectivity pattern between neurons and their organization in the visual cortex [45]. The concept of a neocognitron, proposed by Fukushima [28], is widely considered a as the fundamental basis of modern CNNs. LeCun et al. [61, 62] established the framework of CNNs by developing a multi-layer artificial neural network called LeNet-5, which was applied successfully to image classification problems. With the emergence of big data and the development of according computing infrastructure, the structure of some CNNs has become very deep. A solution by Krizhevsky et al. [59] was able to classify about 1.2 million images into 1000 classes, i.e., a record-breaking result in the ImageNet Large Scale Visual Recognition Challenge. Often, the impressive success of image processing CNNs is attributed to their ability to learn rich mid-level image patterns as opposed to hand-designed low-level features used in more traditional methods.

Recurrent Neural Networks (RNNs) are another special type of network with a loop, which are usually used for handling sequential data. A standard RNN is an extension of the traditional feedforward neural network which is able to store relevant parts of the input and use this information to predict the output in the future. Although vanilla RNN performs well in capturing nonlinearity in time series problems, it was observed that backpropagation dynamics caused the gradients in an RNN to either vanish or explode while training to capture the long-term dependencies [4]. To overcome this issue, the advanced version of RNN—LSTM (long short-term memory) [39] was proposed by Hochreiter and Schmidhuber to address the difficulties of training RNNs [4]. More specifically, LSTM units include a "memory cell" at each time step, which can choose to read from, write to, or reset the cell using explicit gating mechanisms. This architecture lets them capture potential longer-term dependencies.

Understanding deep neural networks is challenging due to their complicated inner workings. In the visualization area, research focuses mostly on helping with the design, training, diagnosis and refinement of deep learning models [41, 66, 109] (one example shown in Figure 2.3 (A)). Work that applies deep learning to solve visualization tasks is still rare. Han et al. [33] presented FlowNet, an approach based on an autoencoder for improving the clustering and selection of streamlines and stream surfaces (shown in Figure 2.3 (C)). Kim and Günther [53] proposed a robust reference frame extraction method based on convolutional neural network (CNN) that is able to extract a steady reference frame from a given unsteady 2D vector field. Wang et al. [103] proposed

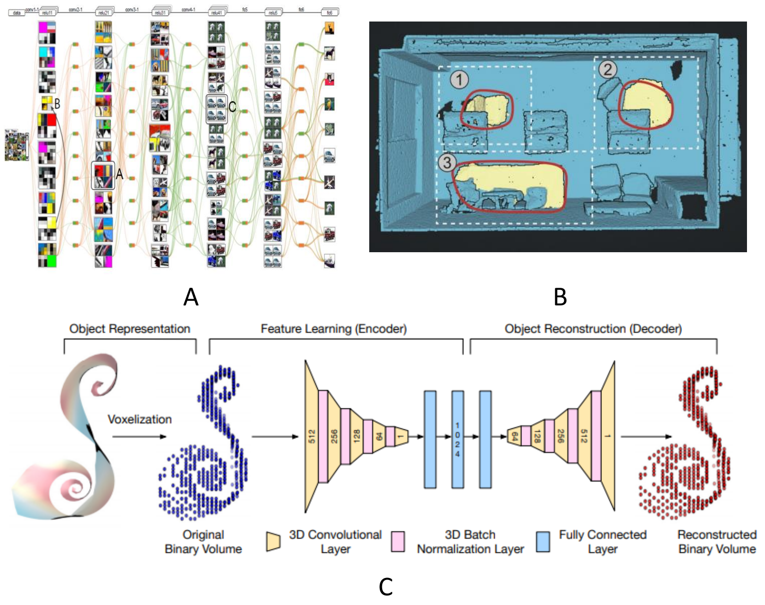


Figure 2.3: Examples of deep learning knowledge used in visualization: (A) CNNVis, a visual analytics approach to understanding and diagnosing convolutional neural networks (CNNs) [66]. (B) Lassonet: lasso-selection of 3D point clouds based on deep neural networks [10]. (C) FlowNet, a deep learning framework for clustering and selection of streamlines and stream surfaces [33].

an LSTM-based approach to facilitate the network exploration by directly mapping network structures to graph drawings. Hu et al. [44] introduced VizML that predicts visualization design choices from a large corpus of datasets using neural networks. Data2Vis [14] made use of recurrent neural networks to generate Vega-lite visualization specifications from JSON-encoded datasets. To improve the interaction technique in visual analytics, Chen et al. [10] developed a learning-based approach of lasso selection for 3D point clouds. In this approach, the lasso selection is modelled as a latent mapping from viewpoint and lasso to point cloud regions (shown in Figure 2.3 (B)).

The limited deployment of DL-based systems for real-world visual analytics solutions are likely based on three reasons: (1.) Understanding and explaining a deep neural network is challenging due to its complex “black box” nature. (2.) Usually, high accuracy of DL-based prediction requires large amounts of training data, which often is very difficult to acquire. (3.) There is no established common understanding of how to determine the right DL solution as knowledge of topology, training method and required hyperparameters. Consequently, it is often difficult to efficiently make good use of deep learning—especially, when non-standard tasks are to be supported.

2.1.3 Machine learning frameworks

As we know, we humans are able to learn and apply relevant knowledge from previous learning when encountering new tasks. Most of traditional machine learning algorithms are designed to address single tasks. In contrast, transfer learning offers the opportunity to bring the power of state-of-the-art models to new domains where insufficient data and time/cost constraints might otherwise prevent their use [81]. Transfer learning with CNNs has been also explored and demonstrates that the intermediate activations learned with deep CNNs pre-trained on large datasets such as ImageNet and GoogLeNet, can be transferred to many other recognition tasks with limited training data [93].

In general, labeling the datasets is an important prerequisite for machine learning tasks. Active learning (AL) is a special type of semi-supervised machine learning and able to interactively queries the user to obtain the label information for new data. As labeling manually is expensive and time-consuming, AL has been successfully applied to the situations where abundant data are unlabeled. The goal of AL is to improve the training performance of a classifier at the lowest possible annotation cost by intelligently picking the best examples to label. In order to combine the potentials of humans and machines to make labeling more efficient, Bernard et al. [5] proposed a visual-interactive labeling framework which enables users play an active role in the process of labeling.

2.2 Brushing techniques

Brushing is one of the most important interactions in visual analytics, where elements are selected (and highlighted) in one display, and concurrently the same information is also highlighted in any other linked display. Figure 2.4 shows an example of linking and brushing, where the rectangular brush on the left leads to the highlighting in two separate views on the right, solving the problem of showing data relations in just one view. Many techniques for brushing have been developed, each with its own strengths and weaknesses—for example, in terms of their ease of use and the degree of control that the user has and variants can be categorized into:

- *brushing using simple geometries*—the most commonly used brushing solutions include rectangular or circular brushing on scatterplots, line-brushing on data graphs [56], etc.
- *lassoing*—the user selects subsets by drawing a geometrically detailed lasso around the target group of item representations.
- *logical combinations of simple brushes*—the user makes use of multiple brushes and combines them using logical operators to refine the data selection [17, 72].
- *sketch-based brushing*—the user sketches a shape onto a visualization and a selection heuristic is used to determine which data are selected [22, 76, 84].

Brushing is intrinsically based on the interaction between the user and the system, often a combination of mouse/cursor motions and button clicks. Less usual methods, based on eye/head tracking, for example, or gestures in a virtual reality environment, have also been proposed [107].

Brushing in scatterplots is often based on the use of simple geometric shapes such as a rectangle or circle. Alternatively, users can use a lasso to specify the selection more accurately. Several extensions to simple brushing have been published, including techniques to formulate more complex brushes by combining multiple brushes using logical

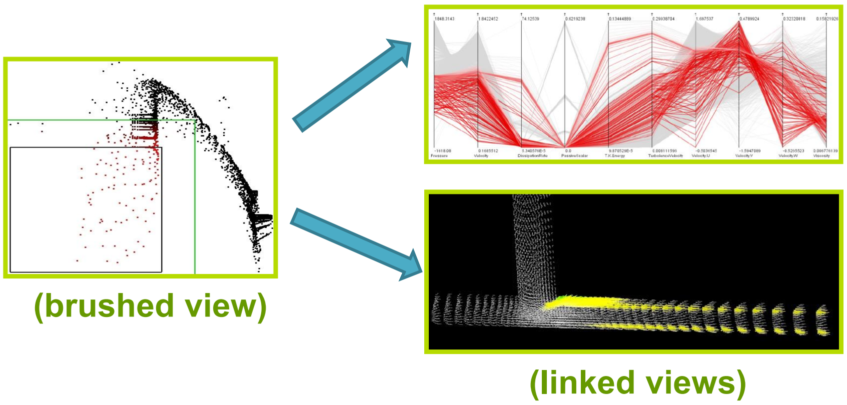


Figure 2.4: Example of linking and brushing in visualization [18]. With the rectangular brush (on the left), we get corresponding highlighting in two other visualizations (on the right).

operators. Martin and Ward [72], for example, enable the user to configure composite brushes by applying logical combinations of brushes, including unions, intersections, negations, and exclusive or operations.

In addition, advanced brushing mechanisms have been integrated into visualization solutions, which offer interactive formula editor and execute additional functions to specialize the behavior of a brush. For example, Hauser et al. [36] developed angular brushing of parallel coordinates to select only data points whose representation on the display form lines with angles similar to that specified in the brush. In this way, data points that are well correlated with each other between a given pair of dimensions can be readily isolated.

Koytek et al. [57] created MyBrush, which extended the popular brushing and linking technique by incorporating personal agency. It offers users the flexibility to configure the source, link, and target of multiple brushes. Hurter et al. [46] developed a semantic lens which selects a specific spatial and attribute-related data range and it is applicable for scenarios requiring a mixed selection of the zones of interest.

Similarity brushing [76, 80] is a typical example of sketch-based brushing, which is based on a fast and simple sketching interaction—the user uses a swift and approximate gesture (for example, drawing an approximate shape that the data should follow) and then a similarity measure (target function) is defined to identify, which data items actually are brushed. This way, the interaction is fast, but likely not 100% accurate.

Recently, the Mahalanobis brush was presented as an interesting alternative for brushing scatterplots [84]. The user simply clicks into the center of a coherent data subset to be selected. The link between the interaction and the actual selection is realized on the basis of an analysis of the underlying data (a local covariance matrix indicates the overall shape and orientation of the data to be brushed, forming then the basis for a local Mahalanobis metric, which is then used as a distance measure to select the data). While this technique is giving quite good results, it still has limitations, including a non-optimized selection of the local context for the Mahalanobis computation and one off-screen parameter for the brush size.

Although various brushing techniques have been introduced over the years and applied to different visual analytical tasks, it is rare to see the work which incorporates user's perspective into brushing model building and optimization. In our eyes, the user interaction logs contain rich information about how they use the technique and the interest of the datasets they explore. Therefore, we see this as a highly interesting chance for relevant innovation.

2.3 Time series data analysis

A time series is a series of data points indexed in time. Time series are usually visualized via line charts and widely used in any domain which involves temporal measurements such as applied science and engineering, economics, statistics, etc. In order to extract meaningful statistics and understand the underlying context of the time series, various methods are incorporated in time series data analysis. In our work, we focus on the time series matching which is one of the most important tasks in time series analysis. In general, time series matching refers to a scenario that a user enters a time series and the system finds "similar" time series. In the following, we provide a brief introduction to common time series similarity matching algorithms, followed by a detailed overview of prior work related to visual query systems for time series data.

2.3.1 Time series data similarity

In the visualization and data-mining literature, the Euclidean distance (ED) and Dynamic Time Warping (DTW) are the most commonly used distance measures. Squared ED is defined as the sum of the squared differences of values between two time series at n sampled points. The basic ED can be improved with data normalization, often z-normalization, which considers the variation of similar patterns in amplitude and y-offset [30]. However, since distances are computed point-wise and the mapping of a query point to a data point is fixed, ED is sensitive to noise and local time misalignments.

DTW overcomes ED's inability to handle local time misalignments (or warps) by allowing the horizontal stretching or compression of a time series when searching for similar ones. Therefore, DTW is considered to yield better fits for shape matching, especially when the similar shapes are not aligned along time.

For matching the pattern between the sketched query and the time series data, both ED and DTW require a sliding window with size equals to the query length to compute the similarity along time. In addition, Ding et al. [16] conclude in their survey that on small datasets, DTW can be significantly more accurate than ED, but the relatively simple and straightforward ED can be competitive with the DTW when the size of the dataset increases.

2.3.2 Visual query system for time series data

In visual query systems, the visual components are used to drive the formation of queries. TimeSearcher [37] is a pioneering information visualization tool based on the use of timeboxes to query time series data (shown in Figure 2.5 (A)). To use the timeboxes, the analyst is asked to draw a rectangular region which can specify the extent

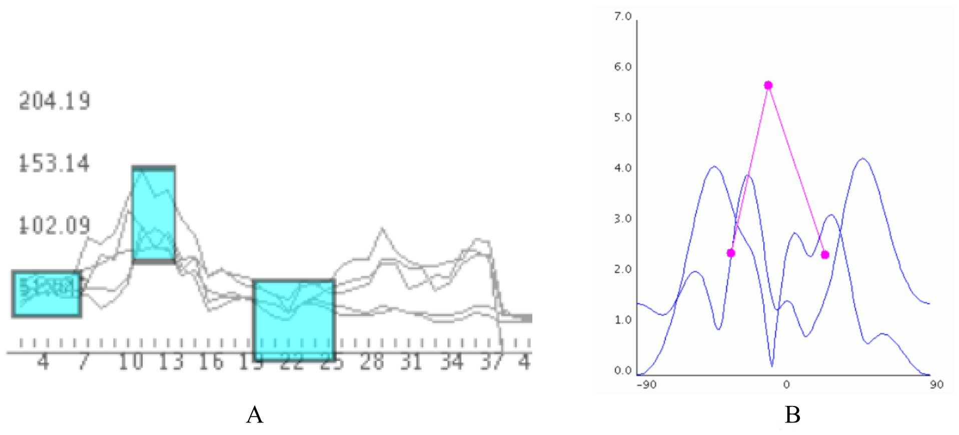


Figure 2.5: Examples of visual query systems. (A) TimeSearcher, an interactive temporal query system based on timeboxes [37]. (B) Querylines, a flexible visual query tool that allows users to form queries consisting of soft constraints and preferences [89].

of time points on the horizontal axis and the range of values on the vertical axis. The time series data then can be highlighted while passed through by the timeboxes. Later, some extended versions have been proposed to improve the basic timeboxes by incorporating the variable (fuzziness in the boundaries) [51], angular queries and slopes to search ranges of differentials [38] and supporting more flexibility with options to adjust the query [7]. Overall, timeboxes are powerful value-based widgets and widely used in many visual query systems. However, it is troublesome to specify a shape-based query, for example, a head-and-shoulders pattern with timeboxes.

The Querylines system [89] is another typical type of a filter-based approach to visual querying (shown in Figure 2.5 (B)). It offers the user the opportunity to specify the constraints by using line segments. The analyst can qualify these line segments as hard or soft constraints based on their preference. If the query gets over-constrained, feedback from the system enables the users to quickly and continuously refine their query specification.

An alternative technique for constructing visual queries is to identify the most common shapes such as spikes, sinks, rise, drop, plateau and valley, then build queries using these basic shapes as pattern templates [32].

The concept of a sketch-based visual query system was first proposed by Wattenberg [104]. In the system, the analyst can sketch an approximate pattern on the same display where also the data is visualized for searching similar patterns and the similarity to the time series data is calculated as simple Euclidean distance. The system is straightforward for the user to use, but the quality of query result relies strongly on little scaling errors and well defined time and amplitude ranges of the sketch, which is not easy for the user to handle.

To improve the flexibility and tolerance in their sketch-based visual query system, Holz and Feiner [42] provided a relaxed selection technique which allows the user to implicitly indicate a level of similarity that can vary across a search patterns during sketching. Specifically, the mouse speed is used to inform the tolerance of points spatially and temporally in the sketched query.

In order to know the human visual perception between their sketches and the corresponding patterns in their mind, Eichmann and Zgraggen made a comparison of rankings of pattern matches produced by algorithms against human-annotated results [19]. They found that human annotated rankings can differ drastically from algorithmically generated rankings and concluded that the meaning of sketching is too diverse to be captured in one algorithm or metric.

As a multitude of queries can be targeted by the same sketch, Correll and Gleicher [13] investigated these ambiguities of sketch-based query system in time series data and define a set of “invariants", enabling the user to choose the properties of data to ignore while sketching. In addition, they adapted different matching algorithms to support different invariants correspondingly. The main drawback of this approach is that it is not easy and straight forward for the user to think about the invariant while doing the data exploration which may not be suitable for non-experts.

Muthumanickam et al. [78] outlined important perceptual features for effective shape matching and define a grammar to express time series approximately by considering the data as a combination of basic elementary shapes positioned across different amplitudes. These basic shapes are represented by using a ratio value and then a symbolic approximation can be achieved by performing binning on ratio values. However, the major problem of this method is the limited query expressiveness, along with the black-box nature of query execution with each shape often having its own processing or matching steps.

Research on human visual perception suggests that humans mentally decompose complex shapes into visually salient parts such as piecewise upward or downward lines, peaks and troughs [40, 52, 55]. Based on this research, Mannino and Abouzied presented Qetch [71], a tool where users freely sketch patterns on a scale-less canvas to query time series data and get rid of specifying query length or amplitude. To achieve this, the curvature was used to segment the time series data and the sketch, and the matching was then based on the segments rather than time slices. The proposed matching algorithm was based on the local distortion and the shape errors, which they identified from a user study by analyzing the human sketch and their sketching goal. This method claims its advantage (dealing with the scale-less sketch) over the traditional matching algorithms—ED and DTW. However, the query result is very sensitive to the smoothing level of the time series data as the query length is based on the salient parts (constructed by extrema and inflection points) of the data. In addition, the use of a sliding window to do the computation in different level of smoothing is too time-consuming for a large amount of time series data.

Prior work in visual query system research suggests a strong need for modeling technology which is able to properly capture the semantics of user’s sketching intention for querying time series data. While a lot effort has been invested, all the current methods mentioned above are not good enough to achieve a satisfactory result and enable a real-time interaction at the same time. Inspired by the remarkable success achieved by deep learning, we see an opportunity to take advantage of the deep learning knowledge to improve the current situation and contribute a new way to cover the missing solutions in this field.

2

Chapter 3

Contributions

In this chapter, we outline the contributions of the research done during this PhD project. First, we introduce the use of machine learning methods to improve brushing in visual analytics. More specially, we exploit traditional machine learning and develop an improved Mahalanobis brush and a new KDE-based brush. In addition, we make use of deep learning and implement CNN-based brush and its improved version—a personalized CNN-based brush. Based on this, we also make a quantitative comparison between the empirical brushing models (Mahalanobis brush and KDE-based brush) and deep learning-based brushing (CNN-based brush) in order to investigate the human influence in the model design. Furthermore, we pay attention to visual query systems and present an application based on the LSTM network to improve the accuracy and efficiency of sketch-based querying of time series data. Related evaluation and demonstration cases are presented in Chapter 4.

3.1 Improving brushing by machine learning

Linking and brushing is a central and well-established interaction technique for relating data aspects across coordinated multiple views [77, 87]. In our research, we have studied the question of how close we can get to successfully integrating efficiency and accuracy in one technique. In the following, we present the solutions we have achieved based on different machine learning models (we chose the example of brushing in scatterplots as our study case—we think, however, that our principle approach is extensible to other views and according brushes).

3.1.1 New Mahalanobis brush

Our first solution is an extended version of the previously published Mahalanobis brush [84], which we have extended and further optimized using data from a user study with 50 participants. In order to get as close as possible to our goal (fast and accurate), we used the following principal approach (also illustrated in Figure 3.1):

In order to be fast, any technique which requires the user to do multiple basic interactions in order to define just one brush (like a lasso, for example) is excluded. We also wished that the users could get rid of off-screen parameters adjustment as it can potentially interrupt their explorative/analyical procedure.

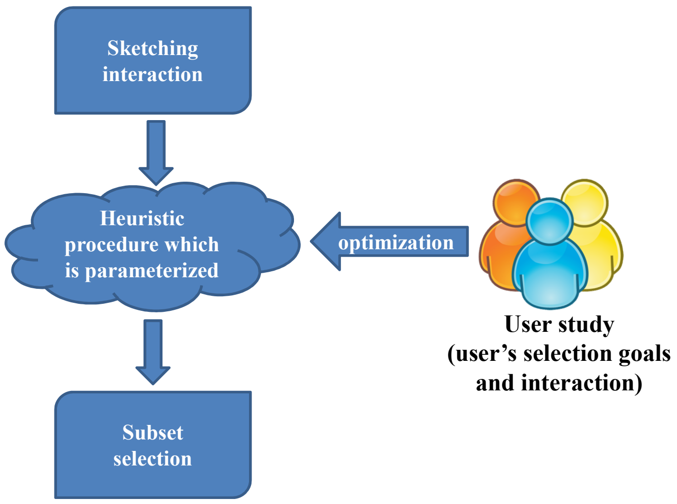

Sketching interaction
Heuristic procedure which is parameterized
optimization
User study (user's selection goals and interaction)
Subset selection

Figure 3.1: Illustration of our principal approach: To be fast, we use sketching as interaction; to derive which data to actually brush, we use a heuristic with parameters that we optimize using data from a user study.

In order to be accurate, we aimed to outperform the use of simple geometries—mostly due to their limited abilities to accurately select specific data subsets, in particular in "crowded" regions of a data visualization. Accordingly, we concluded that a sketching interaction, combined with a carefully modeled selection heuristic, would be the right principal approach in our case.

Typically, the heuristic, which determines the data subset to be brushed, based on a simple sketching interaction, is parameterized and different parameters will lead to different brushing results, even when the user interaction (the sketch) is exactly the same. As our goal was to develop a technique, which does not require any adjustment of technique parameters by the user such that the user can focus on the fast and accurate interaction with the data, we therefore optimize the relevant parameters of our selection heuristic based on the data acquired from a user study (including the information of which dataset subset did the users actually wish to brush and the according gestures would the users do). This optimization procedure is done only once, which means that for any new selection with our technique, using the same optimized parameters, we then also expect a similar accuracy as achieved during training.

Figure 3.2 provides an overview of the new Mahalanobis brushing algorithm. We use a simple click-and-drag interaction for sketching the data subset to brush (click into the middle of the targeted data subset and drag the pointer to the boundary of the subset). The click-point $\mathbf{s}$ and the end-point $\mathbf{e}$ of this interaction provides us a first hint concerning the size of the data subset (scaled by parameter α), which the user wishes to brush. Similarly to the original Mahalanobis brushing technique [84], we also consider a circular data subset, centered around the click-point $\mathbf{s}$ of the interaction, and estimate the shape and orientation of the data in this region by looking at the local covariance information. As an improvement, we then start an iteration (influenced by parameter β—jittering size to avoid a singular covariance matrix), until convergence,

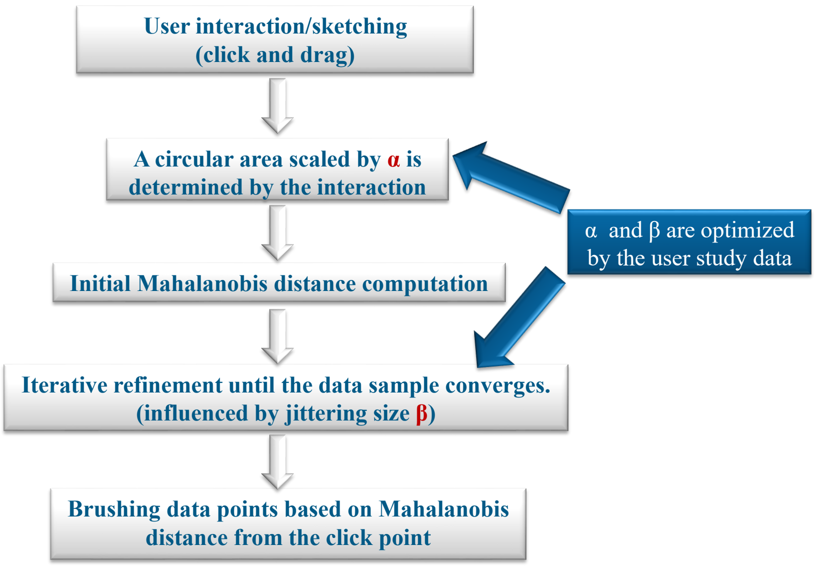

User interaction/sketching (click and drag)
A circular area scaled by α is determined by the interaction
Initial Mahalanobis distance computation
Iterative refinement until the data sample converges. (influenced by jittering size β)
Brushing data points based on Mahalanobis distance from the click point
α and β are optimized by the user study data

Figure 3.2: Overview of our fast and accurate brushing technique: the user clicks into the middle of the data subset to be selected and drags the pointer to the border of the subset (sketching interaction); iteratively, a selection of points around the click-point is chosen, based on local covariance information, until convergence; a selection is made based on the Mahalanobis distance from the click-point. Two parameters, α and β, related to the sample size, before iterating, and to some jittering, stabilizing the technique, influence the performance and we optimize them using our user study.

that refines this data subset selection, based on the local covariance information. After convergence, we eventually make a selection of data points, based on the Mahalanobis distance, taking the local covariance information into account.

Since the Mahalanobis distance is a central concept in our technique, be briefly review it first. The Mahalanobis distance is introduced by P. C. Mahalanobis [69] and based on the correlation between data variables. The Mahalanobis distance between vectors **a** and **b** can be defined as

$$d_\Sigma(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^\top \Sigma^{-1} (\mathbf{a}-\mathbf{b})} \tag{3.1}$$

where Σ is the covariance matrix of the sample (its diagonal elements consist of the variance of each variable and the off-diagonals are the mutual covariances). The location of equal Mahalanobis distances forms an ellipse around the sample mean (in 2D).

As our technique is based on using the local covariance structure of a data subset around the click-point **s**, it is important to determine, which data subset should be used for this computation and we do this in two steps.

Initially, we consider a circular area with radius $\alpha \cdot d_E(\mathbf{s},\mathbf{e})$, where α is a weighting factor and $d_E(\mathbf{s},\mathbf{e})$ is the Euclidean distance between **s** and **e**. All data points within this circle are used to compute the first instance of the local covariance information, Σ_1.

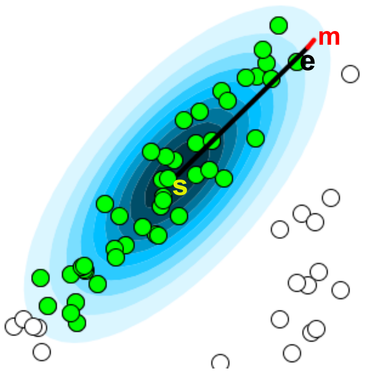


Figure 3.3: Selecting data points, based on the local, weighted covariance information: **s** and **e** denote the start- and end-points of the click-and-drag interaction; the ellipses illustrate Σ_w. **m** lies on the Mahalanobis ellipse which acts as the eventual selector.

Next, we consider all points within a Mahalanobis ellipse, based on Σ_1 and sized according to $d_{\Sigma_1}(\mathbf{s},\mathbf{e})$. Usually, this leads to a new data subset, which is similar to the data subset as determined by the initial circle, but more closely following the underlying data structure. To obtain an even better sample, we refine the sample iteratively by replacing it with the points in the Mahalanobis ellipse that is updated every iteration according to the covariance of the samples in last iteration. While this process usually converges quite quickly, we observe that it sometimes can lead to small fluctuations, including/excluding a few data points in consecutive iterations.

In order to stabilize and secure the convergence of the covariance matrix optimization, we enable the partial consideration of data points for the computation, leading to a solution that is based on the weighted covariance matrix [31]. During the iteration, the weight of each point is updated and the points that are stable in the Mahalanobis ellipse are assigned a higher weight than less stable points. This minimizes the inaccuracy caused by the initial samples, resulting in a more reasonable converged covariance matrix. More details of this procedure (weight function design, singular matrix handling, etc.) are described in Paper A (A.4.2).

The selector used to determine the actually brushed data points is based on the weighted covariance matrix Σ_w: We use the Mahalanobis ellipse, according to Σ_w, that corresponds to point $\mathbf{m} = \mathbf{s} + \alpha(\mathbf{e} - \mathbf{s})$ ($\mathbf{s}$ and $\mathbf{e}$ are the start and end point of the interaction respectively). Accordingly, the set of all brushed points is defined as

$$\{\,\mathbf{x}_i \mid d_{\Sigma_w}(\mathbf{s},\mathbf{x}_i) \leq d_{\Sigma_w}(\mathbf{s},\mathbf{m})\,\} \tag{3.2}$$

Figure 3.3 shows contours of the selector, selecting all green points within the Mahalanobis ellipse, which corresponds to click-point **s** and location **m**.

As described so far, our brushing model has two not-yet-optimized parameters: α (size of the initial selection, determining the context of the local data shape analysis) and β (jittering size). In order to achieve an as accurate as possible brushing result, we conducted a user study to get information about how users would use our technique to brush and what they actually wanted to select from the dataset (ground truth). In the user study, we collected 600 selections, of which we randomly chose 400 as training

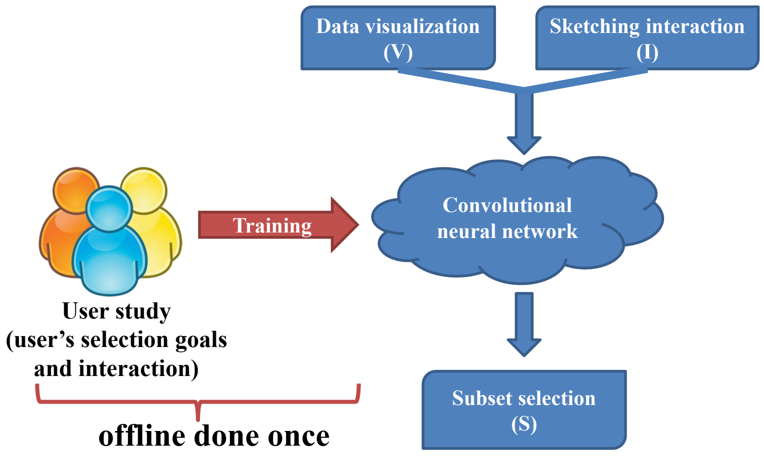

Data visualization (V)
Sketching interaction (I)
Training
Convolutional neural network
User study (user's selection goals and interaction)
Subset selection (S)
offline done once

Figure 3.4: Illustration of our principal approach: To be fast, we use sketching as interaction; to estimate which data to actually brush, we use a CNN trained with data from two user studies.

data, leaving 200 selections for the validation. Based on this information, we then did an optimization of α and β: the Dice coefficient is used as a cost function to compare the similarity between the selection goal by the user and the corresponding results by our technique (see Paper A (A.6)). After the parameter optimization, we obtained the optimal value of α and β for our brushing technique. Based on this, we did a quantitative accuracy comparison with the previously published Mahalanobis brush [84] using the interaction information from our user study (illustrated in section 4.1).

In this work, we have described and exercised an approach, which is all-too-little seen in the visualization literature, i.e., a user study-based optimization of visualization parameters. We see the potential that this work can motivate others to follow a similar approach in their visualization research, i.e., to take advantage of the user interaction information to do an automatic optimization of visualization parameters.

3.1.2 CNN-based brush

Recently, deep learning based methods especially convolutional neural networks (CNN), which exploit the deep architecture to learn the hierarchical discriminative features, have been utilized successfully in a wide range of fields such as natural language processing, object detection and image processing. As brushing is mainly used to select some coherent and structured subsets, which is similar to detect the distinguishing patterns in image analysis, we are inspired to make use of the CNN knowledge and develop a new CNN-based technique for brushing in scatterplots. To achieve this, we used the following approach (also illustrated in Figure 3.4):

Usually, users brush subsets, which are spatially coherent in the visualization. Thus, we assume that we can estimate the brushing goal from both the actual brushing interaction and the data distribution in the visualization near the interaction. In our approach, we aimed to create a computational link between the fast and simple interaction and the selection of a non-trivially delimited subset, estimating the visual structure that the user identified as the brushing target. More specifically, we deemed the combination of an basic sketching interaction I, indicating the location, size, and orientation of the subset to brush, with a computational estimation function S, determining which subset to actually select, based on its visualization V near sketch I, to be a useful framework for modeling our solution. In previous work [76, 84] and the new Mahalanobis brush introduced above, the estimation function S was carefully modeled according to meaningful heuristics, based, for example, on a geometric similarity function in visualization space.

Since S amounts to interpreting the data visualization in terms of which spatially coherent subset best possibly relates to the sketching interaction, we found it promising to exploit CNN for our solution. We expected that the increased flexibility of this approach also helps to overcome limitations of brushing nonlinear structures and data subsets, in particular in "crowded" regions of a visualization. Thus, we constructed our solution around a convolutional neural network (CNN) and the training data is from two user studies.

The first user study we used is introduced above for optimizing the parameters of Mahalanobis brush, providing information about both the brushing goals (which dataset subset did the users wish to brush) and the according interaction (which gesture would the user do to actually select the targeted data subset) from 50 users. In the second user study (details are in section B.7), we examined the variation information of the user's interaction in order to use this for modeling an extension of the training data for the CNN.

Figure 3.5 provides an overview of our new brushing algorithm. Since our goal is to estimate the selection information S from both the input sketch I as well as from the data visualization V, we need to efficiently and effectively consider these two heterogeneous parts of input information. As a result, we handle I and V individually, using the CNN only for the interpretation of V. The critical input from I (click-and-drag sketch), i.e., the click point $\mathbf{s}$ (center of the interaction) as well as the length r and the angle ϕ of the drag component, is first used to locate, scale, and orient the receptive field of the CNN. This way, we "normalize" the network's operation with respect to I by a simple linear transformation such that we can easily "undo" this normalization after the network's estimation process. Accordingly, the network's task is then to interpret the 2D data distribution in the appropriately located, scaled, and rotated region of the visualization. In order to predict which data subset to select, we model this step as an image processing operation: on the input side, we let the network see a 2D histogram of the data in the targeted area; on the output side we expect a measure p per bin of the histogram, indicating a "degree of selection" such that a simple thresholding at $\overline{p} = 0.5$ can identify the region within the target area corresponding to the selected data subset. We carefully experimented with many different layouts/settings of the CNN model, varying the size and number of the convolution filters, the number of convolutional and fully-connected layers, and the number of neurons in the fully-connected layers. As a result, we found a model which fits our scenario well. In our design, we deviate from the conventional CNN layout by replacing the last layer (classifier) with a structured regression layer to encode the output information from which the actual data subset selection can be derived in a subsequent step.

Altogether, we propose a model with two convolutional (C), two max-pooling (M), and two fully-connected layers (F). Figure 3.6 shows this architecture and the associ-

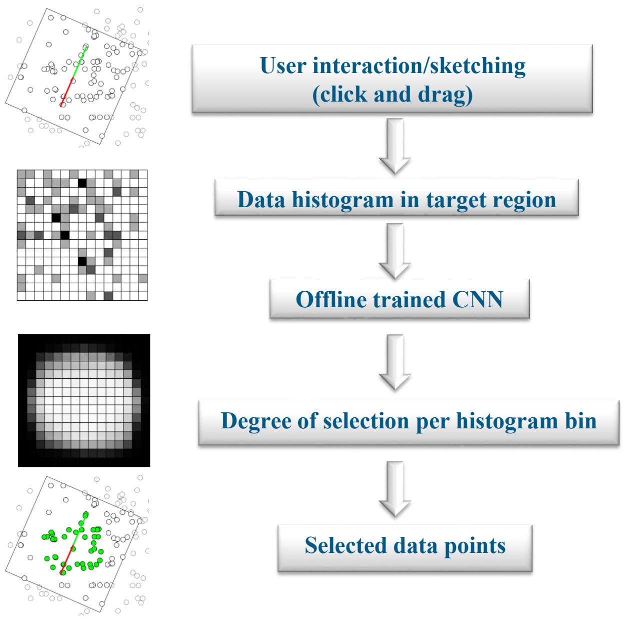

User interaction/sketching (click and drag)
Data histogram in target region
Offline trained CNN
Degree of selection per histogram bin
Selected data points

Figure 3.5: Overview of our fast and accurate brushing technique: For sketching, the user clicks into the middle of the data subset to be selected and drags the pointer to the border of the subset; The CNN then sees the data distribution near the interaction as a 2D histogram. It delivers a degree-of-selection value per histogram bin, from which we can compute, which data subset is selected.

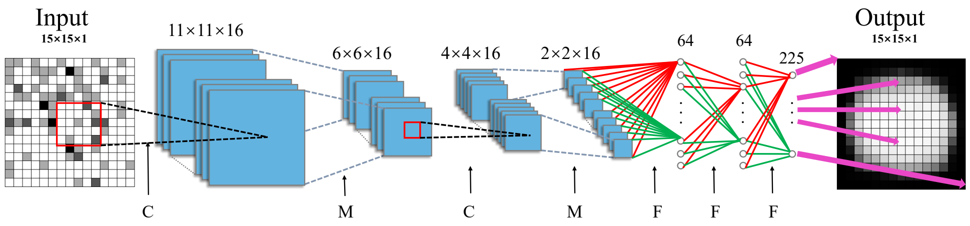


Figure 3.6: The proposed CNN model. C, M and F represent the convolutional layers, max-pooling layers, and fully connected layers, respectively. The purple arrows from the last layer illustrate the association between the final layer's outputs and histogram-aligned grid of degree-of-selection values.

ation between the last layer and the histogram-aligned grid of *p*-values. In detail, our model is configured as Input(15×15×1), C(11×11×16), M(6×6×16), C(4×4×16), M(2×2×16), F(64), F(64), and F(225). The sizes of the C and M layers are defined as width×height×depth, where width×height determines the extent of each feature map and depth represents the number of maps (filters). The details about the training parameters setting, data preprocessing and training data synthesis are illustrated in Paper B (B.5,B.8). To evaluate the proposed CNN-based brush, we did a quantitative accuracy and efficiency (computation cost) comparison with Mahalanobis brush using the

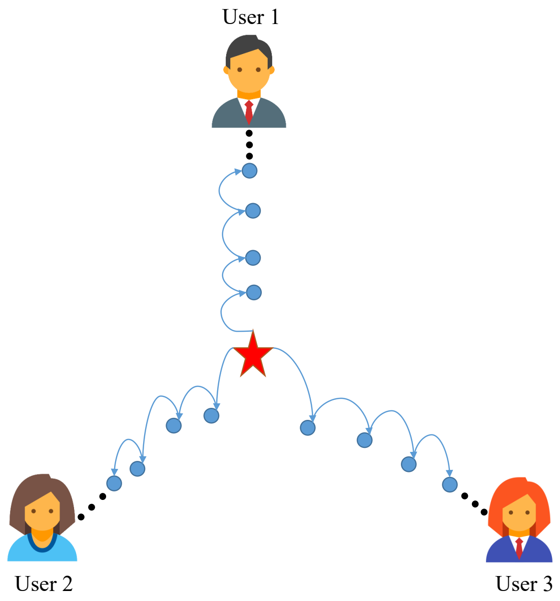


Figure 3.7: Illustration of the core idea: the general model (red star) is progressively refined and getting closer and closer to a single user model in the high dimensional parameter space (here 2D in this illustration).

user study data (details are in section 4.2).

In this work, we demonstrate how deep learning can be used to further improve the central operation of brushing in visual analytics. By learning the relation between the data subset to be selected and a click-and-drag sketch by the user to do the selection, we contribute an innovative application of a structured regression model, realized by a convolutional neural network, to improve brushing in scatterplots. Based on this successful attempt, we believe that it can be beneficial a lot for experts in visualization to incorporate the advances in deep learning into visualization tasks.

3.1.3 Personalized CNN-based brush

As mentioned above, our CNN-based brush for scatterplots achieves the highest accuracy compared to all current sketch-based methods with a fast interaction. While this method has made great progress in learning the user's mind while brushing, it suffers from two limitations: First, the model is trained off-line once by the data from different users and what the model learns is the average brushing preference across several users, leading to a general model which is obviously not optimized to every single user. Second, while it of course is possible to retrain a new model for a single user from scratch, this procedure is time-consuming and requires sufficient training data which is difficult to get from a single user.

To address the limitations of this CNN-based brushing solution, we propose an innovative framework which is able to iteratively refine the brushing model for a single user with additional data that he/she provides while using the brushing technique. This idea is inspired by active learning (AL), which is a special case of semi-supervised

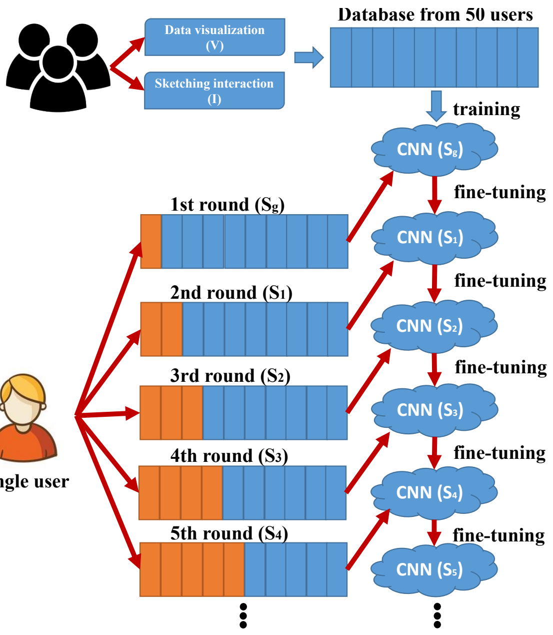


Figure 3.8: Principal approach: the general model is fine-tuned during continued use by new data from a specific user.

machine learning that can incrementally improve the existing model by interactively querying the user for additional input. In addition, we exploit knowledge from transfer learning and leverage the parameterization of a well-trained model instead of learning the user's brushing behavior from scratch, largely reducing the time cost of the retraining procedure while maintaining a focus on avoiding overfitting.

The overall goal of our research was to improve the general CNN-based brushing model [23] and to make it suitable for those users whose brushing preference is deviating from the average. The schematic in Figure 3.7 shows the core idea of our proposed framework where the general model is indicated by a red star. To find the optimized model for a single user, we take advantage of the well-trained general model as an initial point in the model's high dimensional parameter space and progressively adapt it step by step to a personalized model. In this way, the prior parameterization is taken into account and we can get rid of the costly general (global) search in the parameter space, largely reducing the time spent and making the results more stable.

Figure 3.8 provides an overview of the proposed framework to illustrate our adaptive brushing model. In the following, we first describe the initial CNN model we used, before we then describe the construction of our solution in detail.

Following the basic definition of the general CNN-based brushing technique mentioned above, we keep using click-and-drag as the sketching interaction I and a computational function for estimating the brushing result, denoted as S, which is based on the visualization V near sketch I. The general model S_g based on the CNN is trained off-line once by 500 basic selections (augmented to 8000 in actual training via sampling natural variation of user interactions) from 50 different users (the data is obtained via a user study described in section 4.1).

In our proposed framework, we establish an interactive scenario to improve the brushing accuracy for a single analyst during data exploration, where the user can actively give some new input when he/she does not like the result generated by the current brushing model. This additional data is used to gradually adapt the general model to a personalized one and we expect, with more data from the user, the brushing result is more accurate for this user.

For learning a single user's brushing behavior, we choose to leverage the existing CNN parametrization of the general model instead of retraining a model from scratch. This is because the data used for training the general model is from 50 different users and it is not practical to get a similar size data from a single user. To avoid the overfitting issue caused by the limited new data, the new data is chosen to replace the most similar data in the original dataset, composing a new training dataset rather than treating the additional data individually.

Our retraining procedure is based on the transfer learning. Transfer learning strategies depend on various factors, but the two most important ones are the size of the new dataset (relatively small or big), and its similarity to the original dataset. As we apply the replacement on the old data, the new training data is the same size and of high similarity (90%) to the old training data. Therefore, we can fine-tune the weights of the pre-trained current network via backpropagation with less of a chance to be overfitted.

The original data for training the CNN-based brushing model is based on 500 selections. In order to find the appropriate data to be replaced among the 500, we formulate a similarity metric based on the extracted features of the input data. The reason for picking up the most similar cases to be replaced is—we hope the retrained model learns the user's specific brushing preference and also keeps itself as general as possible at the same time. Therefore, the similar cases from the original data can be considered as replicated data which should be replaced. The algorithm for the dataset replacement is based on 18 indicators and the details of the computation are explained in Paper C (C.3.1).

To evaluate this framework, we organized a user study where a single user is asked to participate the model refinement procedure 5 times, and each time the user needs to provide 10% additional data by doing brushing in new datasets. Then this user's personal brushing preference data can be contributed to updating the current model in retraining. The details are in section 4.3.

In this work, we present a user-centric framework which allows the user to iteratively improve a brushing technique in scatterplots while using it. The main contribution of this work is that we achieve a solution which is able to turn the general model based on people's average brushing preference to a tailored model for the specific user with a very short time training cost ($\approx$3mins) by leveraging the existing paremeterization.

3.1.4 KDE-based brush

As mentioned above, Mahalanobis brush and CNN-based brush are two principally different approaches which can be categorized as empirical modeling (based on reasoning) and implicit modeling (based on deep learning) respectively. Since CNN brushing resulted in much higher accuracy, and since Mahalanobis brushing is not really based

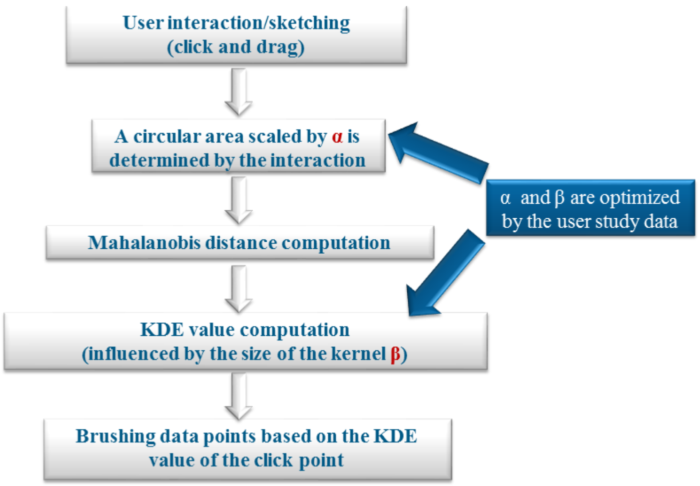

User interaction/sketching (click and drag)
A circular area scaled by α is determined by the interaction
Mahalanobis distance computation
KDE value computation (influenced by the size of the kernel β)
Brushing data points based on the KDE value of the click point
α and β are optimized by the user study data

Figure 3.9: Overview of our KDE-based brushing technique: the user clicks into the middle of the data subset to be selected and drags the pointer to the border of the subset (sketching interaction); then a selection of points around the click-point is determined, based on the KDE values of the data. Two parameters, α and β, related to the sample size, and the size of the KDE bandwidth, influence the results and we optimize them using a user study (50 participants).

on any advanced distance metric – it's "just" linear, after all –, we were interested in studying to which degree empirical modeling can compete with machine learning in this context.

To find out, we report our attempt to construct a best-possible empirical model by further extending the Mahalanobis brush, incorporating kernel density estimation (KDE) [82], and informing a clustering step that returns one of the clusters as the data selection. The reasons for choosing KDE are based on three assumptions: (1.) Actual data-driven density information, captured by KDE, may improve the brushing results (over the simple assumption of a linear model as in Mahalanobis brushing). (2.) A non-linear model may be able to select also non-linear shapes. (3.) The modes of a 2D KDE, at the right scale, could represent clusters of data points to be selected. In the following, we will have a short introduction of our KDE-based brushing model design.

Figure 3.9 shows an overview of the new, KDE-based brushing technique. We keep the simple click-and-drag interaction for sketching the data subset (click into the middle of the targeted data subset and drag the pointer to the outer boundary of the subset). The click-point **s** and the end-point **e** of the drag-interaction provide us with a first hint concerning the size of the data subset, which the user wishes to brush. Similarly to the Mahalanobis brush, we first consider a circular data subset (scaled by parameter α which needs to be optimized), centered around the start-point of the interaction, and estimate the shape and orientation of the data in this region by looking at the local covariance information. We then start a short iteration that refines this data subset selection, based on the local covariance information. This process is same as the Mahalanobis brush, the details of which have been introduced above. After a sufficiently close convergence of this iteration, we make a selection of data points,

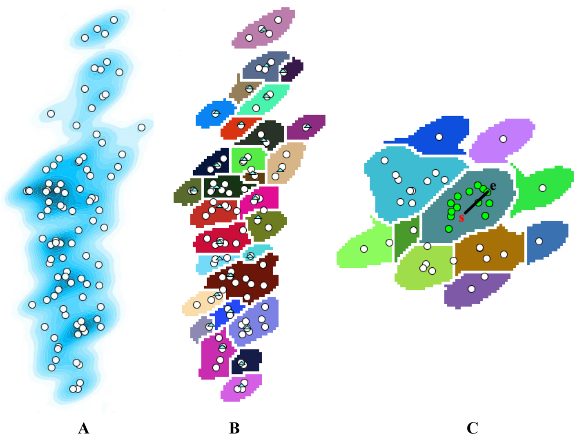


Figure 3.10: A: KDE of a dataset (relatively small kernel). B: Clustering related to the modes of the KDE, indicated by the small blue triangles. C: The one cluster, which corresponds to the KDE mode near to **s** determines, which data points are selected (indicated as green points).

based on a kernel density estimation of the data, using the local covariance information as a basis for specifying the kernel.

Kernel density estimation (KDE) is a popular method for data analysis in the field of statistics, which was introduced by Rosenblatt [88] and Parzen [82]. It is a non-parametric way to estimate the probability density function of a random variable. KDE can be used, for example, to make inferences about data, based on a finite sample.

Assuming that $\{\mathbf{x}_i\}_{1 \le i \le n}$ are n samples of d-dimensional vectors drawn from a common distribution, described by a particular density function, then KDE can be used to estimate this density function as

$$f_{\mathbf{H}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x} - \mathbf{x}_i) \tag{3.3}$$

with $\mathbf{H}$ being the $d \times d$ kernel matrix (symmetric and positive definite). The choice of matrix $\mathbf{H}$ is the single most important factor affecting the main characteristics of $f_{\mathbf{H}}$ [102]. Since we wish to consider the local data distribution when modeling an appropriate kernel matrix $\mathbf{H}$, we can make direct use of the converged covariance matrix Σ_w (introduced in section 3.1.1), leading to the following (anisotropic) kernel function [96]

$$K_{\mathbf{H}}(\mathbf{x}) = \frac{e^{-\frac{1}{2}\mathbf{x}^\top \Sigma_w^{-1} \mathbf{x}}}{\sqrt{(2\pi)^d |\Sigma_w|}} \tag{3.4}$$

In order to realize an appropriate scaling of our kernel, we make use of an eigendecomposition of $\Sigma_w = \mathbf{V}\Lambda\mathbf{V}^\top$ with eigenvectors $\mathbf{V}$ and eigenvalues Λ. This leads to the following, scaled versions of $|\mathbf{H}|^{-\frac{1}{2}}$ and $\mathbf{H}^{-\frac{1}{2}}$:

$$|\mathbf{H}|^{-\frac{1}{2}} = |\beta\phi\Lambda|^{-\frac{1}{2}} \quad \& \quad \mathbf{H}^{-\frac{1}{2}} = \mathbf{V}(\beta\phi\Lambda)^{-\frac{1}{2}}\mathbf{V}^\top \tag{3.5}$$

Used with an isotropic kernel function $K(\mathbf{x}) = (2\pi)^{-\frac{d}{2}} e^{-\frac{1}{2}\mathbf{x}^\top\mathbf{x}}$, this amounts to a KDE with an accordingly scaled kernel matrix. We find the best possible scaling of $\mathbf{H}$ by choosing the two scaling parameters ϕ and β based on two separate solutions: On the one hand, we use a data-driven approach to determine ϕ (see Paper D, D.3.4). On the other hand, we optimize β as a general parameter using the data from the user study. The details of our optimization procedure regarding α and β are in D.5.

The selector used for selecting the data subset is based on a clustering process. The modes of the KDE represent groups of data items (at the scale determined by the size of $\mathbf{H}$). We use clustering (each mode leading to one cluster) to identify the one group of data items, which is associated with the click-and-drag interaction, and select it.

For clustering, we use a watershed algorithm [6, 26, 27, 79]: Starting with the mode with the highest KDE value, we iteratively include neighboring locations into the corresponding cluster, lowering the KDE threshold for doing so iteratively. For every new threshold, we either join a neighboring location to an existing cluster, or create a new one, if the new location is not adjacent to an existing cluster. Figure 3.10 (B) shows an according clustering result for a KDE with a relatively small kernel (shown in Figure 3.10 (A)) – the different clusters are shown in different colors and the corresponding KDE modes are located by small blue triangles. Figure 3.10 (C) shows an example of how data points are then selected (the points in the same cluster, corresponding to click-point $\mathbf{s}$, are selected and highlighted in green).

To evaluate the KDE-based brush and understand the difference between empirical modeling and implicit modeling (deep learning), we did an in-depth comparison between the Mahalanobis brush, the CNN-based brush, and the KDE brush in terms of accuracy, efficiency, generality and interpretability, using the interaction information from two user studies (mentioned in section 4.1 and 4.4 respectively). The details of the comparison are in section 4.4.

The main contribution of this paper includes our extension of the empirical model for brushing points in a scatterplot, and a thorough comparison between the Mahalanobis brush, the CNN-based brush, and the KDE brush with the goal to investigate the influence of human expertise during model design.

3.2 Improving sketch-based visual querying of time series by machine learning

Time series are sequences of data points listed in time order. A time series query refers to finding from a set of time series that satisfy a given search criteria. In general, it is easier to visually describe patterns in time series data than to express them textually or procedurally. Therefore, visual query systems are a convenient user interface with freehand sketching as an efficient means for visual communication.

As the current visual query systems for time series data suffer from the most important efficiency and accuracy problems, the overall goal of our research was to achieve a solution with a fast interaction and an accurate query result. In addition, our solution is designed to be friendly to non-expert users and easy to use with minimal training. Figure 3.11 shows an illustration of the principal approach. To achieve a swift interaction, we enable freehand sketching as the querying input. Our goal is to find a similarity function (S) which is capable of interpreting the relation between the imperfect human sketching interaction (I) and the matching goal within the time series data (V) as ac-

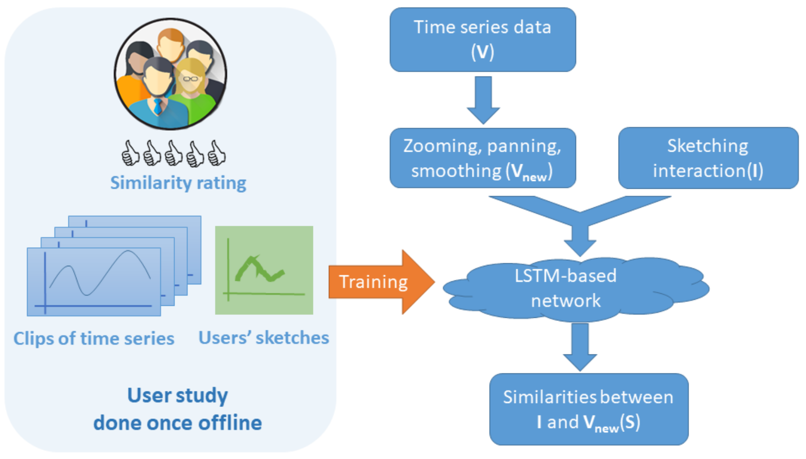


Figure 3.11: Illustration of our principal approach: users specify the scale of the time series patterns of interest by zooming, panning or smoothing, then freely sketch an approximate pattern on the sketching panel. Then a similarity rank between the user sketch and the processed time series data is computed by the proposed parameter sharing LSTM networks. The network is trained only once offline based on user study data.

curately as possible. In addition, we are aiming at a real-time system involving user interaction, which means that the computational cost should also be minimal.

As discussed before, all existing empirical models are not really good at estimating the semantics of a human sketch. Thus, we found it promising to exploit the recent successes of deep learning in similarity comparison for our solution. As a result, we found that recurrent neural networks (RNNs) can be a reasonable way to encode time series data and do the matching computation for two reasons: 1. RNNs have a memory which allows the model to store information about its past computations. This enables RNNs to exhibit dynamic temporal behavior which is naturally fitting to sequential data like time series data. 2. An advanced version of RNN—Long short-term memory (LSTM), can be trained to remember the information from a specific length of past times steps (details are in E.4.2). This mechanism can be used to mimic a sliding window while doing the matching computation along the time series data. At the same time, it avoids reading the same data repeatedly, leading to a relatively low computation cost.

To construct the network structure, we used a pair of LSTM networks with shared parameters to encode the sketch and the time series data, respectively. The sharing of the network parameters was beneficial because of the high similarity between the sketch and the time series in structure. The reduced overall number of parameters accelerates the training procedure and minimizes overfitting. This design is inspired by the "Siamese" network-based solution for sentence similarity [74]. Figure 3.12 provides an overview of our proposed network structure for the user sketch and time series data similarity learning and the detailed description of our network is given in section E.4.3 of Paper E. To train this network, we collected training data from two user studies. For the first user study (details are in E.5.1), we gathered the ground truth about how different users sketch patterns and how they rate similarities based on their

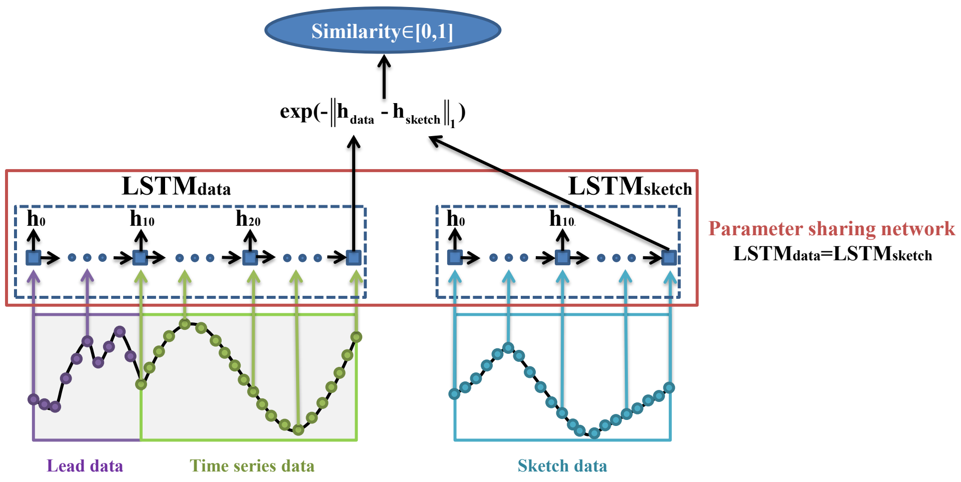


Figure 3.12: The entire structure of our proposed network: the time series data (colored in green) with its lead data (colored in purple) on the left and the sketch data (colored in blue) are encoded by two LSTM networks respectively. This network is trained against a distance metric based on the Manhattan distance and the two LSTM networks share their parameters.

visual perception of their sketches and several clips of the time series data that we offered. In the second user study (details are in E.5.2), we examined the variation of the user's sketches in order to use this for modeling an extension of the training data for a more stable training, the according training data augmentation procedure is elaborated in E.4.5. In the following, we introduce four basic steps of the data exploration workflow and the user interface of our proposed sketching query system for the four steps is shown in Figure 3.13 .

Step 1. Data preprocessing. For most time series data, smoothing is necessary to capture the key patterns of the data, while leaving out noise and micro-patterns. In our design, cubic splines are used to smooth the data. Instead of asking the user to specify the smoothing level, we offer a default smoothing level after loading the dataset based on the number of salient parts. We count the salient parts by segmenting the time series data at extrema and inflection points. To obtain the default smoothing level for each dataset, we adjust the smoothing level until we are satisfied with the number of salient parts that were enough to represent the time series data in advance and this smoothing level is then chosen as the default smoothing setting in the beginning.

Step 2. Interactive Scaling and Smoothing. Choosing a scale (and a smoothing level) for data exploration is a crucial user-side task – meaningful questions may be asked about time series data at multiple scales, depending on the user task. Instead of iterating through all possible scales and smoothing levels while matching, we allow the user to interact with the data via zooming and panning (and/or adjusting a slider to specify the targeted scale and smoothing level), after initially estimating a proper scale automatically.

Step 3. Sketching. After the user specifies the scale and smoothing of the data, a square sketch panel is provided for the user to do free sketching. The empty sketch panel is located on the center of the canvas which can assist the user to sketch with a

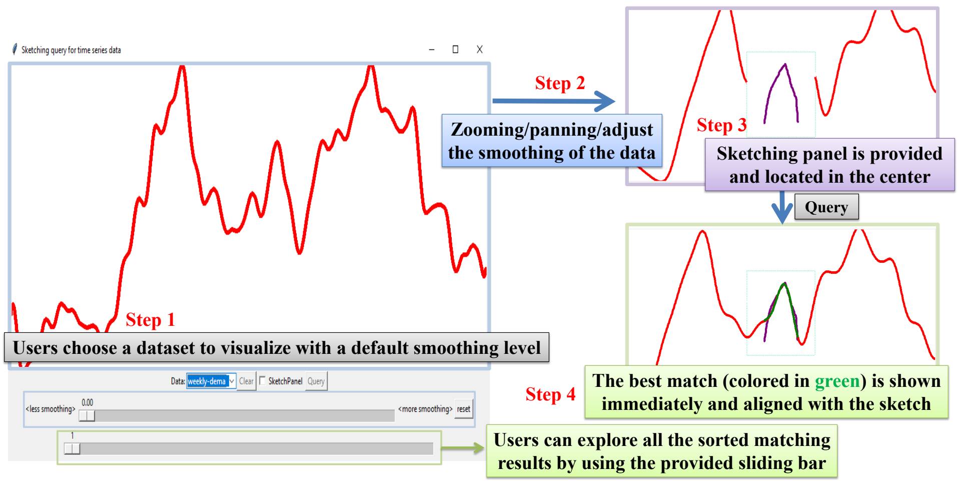

Sketching query for time series data
Step 1
Users choose a dataset to visualize with a default smoothing level
Step 2
Zooming/panning/adjust the smoothing of the data
Step 3
Sketching panel is provided and located in the center
Query
Step 4
The best match (colored in green) is shown immediately and aligned with the sketch
Users can explore all the sorted matching results by using the provided sliding bar
Data: weekly-dema
Clear
SketchPanel
Query
<less smoothing>
0.00
<more smoothing>
reset
1

Figure 3.13: The interface of our proposed sketching query system for time series data is shown on the left. Data exploration is done as: Users choose the data to visualize and interact with the data to specify the desired scale and smoothing by zooming and panning. Then our system allows the users to freely draw a sketch. Based on this, a matching rank is computed and users can explore all results.

similar scale by referring to the scaled time series data in the background. Sketches are slightly smoothed in order to remove hand jitter and the query length is determined by the sketch length.

Step 4. Query and explore the matching results. The proposed parameter sharing LSTM networks then execute the matching computation and an ordered set of similarities between the sketch and the time series data is obtained. The best match is immediately highlighted in green color after the computation and the time series data is then shifted by aligning the best matching part with the sketch. Moreover, a slider is provided to explore all the other results from the ranked list of matching results.

Overall, the main contributions of our work are:

- **A data-driven method for the sketch-based querying of time series data.** To the best of our knowledge, this is the first time that deep learning is used to learn the matching relation between a human sketch and time series data, outperforming two state-of-the-art models in terms of accuracy and efficiency (illustrated in section 4.5).

- **A sketch-based querying system.** We present a prototype of a sketch-based querying system for time series data. We offer the user an opportunity to use a freehand sketch to explore the time series data interactively without the need to set any offline parameter.

Chapter 4

Evaluation and demonstration

In order to demonstrate the usefulness of our proposed techniques, we have developed a set of prototypes to evaluate and illustrate the utility of our contributions. Following the order of the techniques presented in Chapter 3, we demonstrate results for the new Mahalanobis brushing first, followed by CNN-based brushing and personalized CNN-based brushing. Thereafter, we present a comparison of empirical brushing models and deep learning methods, including KDE-based brushing. In the end, we present the LSTM-based visual query system for time series data, where the model is evaluated directly by the user in a real application and compared with two state-of-the-art models—DTW and Qetch algorithm. More details of the demonstration can be found in related papers, which are attached as Part II of this thesis.

4.1 New Mahalanobis brushing in scatterplots

To test our proposed approach, we developed a prototype, which is implemented in *Processing*. *Processing* is a Java-based visual programming language, which is frequently used for the electronic arts, new media art, and in the visual design communities.

For the evaluation, we compared our model with a previously published Mahalanobis brushing technique [84] in terms of accuracy, and with manual Lasso regarding the time spent, based on the user study data. In the following, a short introduction is given about the user study we did before we go through the details of the evaluation.

In the user study, 50 individuals, all students or employees from the University of Bergen, Norway, participated in our study. Each one was asked to do 12 selections. In every case, a particular scatterplot (one out of six) and a particular request (choose a large cluster/a small cluster/an elongated cluster) were given. We formulated two questions for each dataset in advance based on our perception of the datasets (Figure 4.1 shows an overview of the user study data). Then the participant was instructed to choose a target data subset to select (ground truth, reported by the participants using a lasso tool), then also providing the corresponding click-and-drag interaction, which this participant would use to select the target group. We recorded all points selected by the lasso (the brushing goal), the sketching interaction (i.e., the start point and the end point of our new brushing technique), and the time spent on the interaction (excluding thinking time) during both of these two techniques. Accordingly, we collected 600 selections (including 252400 points), of which we randomly chose 400 as training data, leaving 200 selections for the validation.

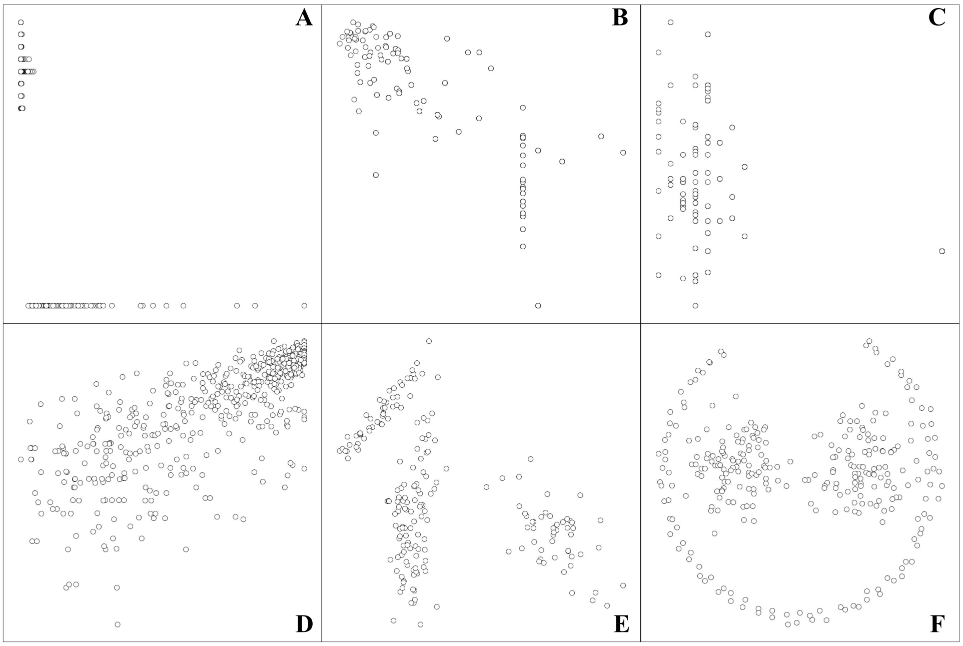


Figure 4.1: Overview of the six datasets that we used in the user study: A–D show Boston housing data (with as different scagnostics as possible); E shows Gaussian clusters and F shows non-linear path-based spectral clusters (as a particuarly difficult case).

Based on this, we did a quantitative accuracy comparison with the previously published Mahalanobis brush [84] using the interaction information from our user study. Figure 4.2 shows a Venn-diagram-like visualization of this comparison in terms of true positives, false positives, etc.

The area surrounded by the dashed line represents the user goal, accumulated over all selections. The area surrounded by a solid line represents the brushing results by our technique while the dotted line surrounds the results by the old Mahalanobis technique. The area size is proportional to the numbers of brushed data points.

We calculated the percentages of how many data points fall in each of the eight possible overlap regions between the user's goal, our brushing, and the original brush after accumulating over all cases (the all-negative region corresponding to the overall context of points outside of all selections was left out from the visualization). The colors used in this visualization correspond also to the colors of points in the other scatterplots in this discussion section:

- least interesting are yellow points (both brushing technique succeed to select the point correctly (both true positive), purple points (both brushing techniques fail to select), and pink points (both techniques select falsely).
- more interesting are green points (the new technique succeeds, while the original fails), blue points (the original technique selected falsely, while the new one does not), orange points (the new technique fails to select, while the original did), and red points (the new technique selects falsely, while the original did not)—

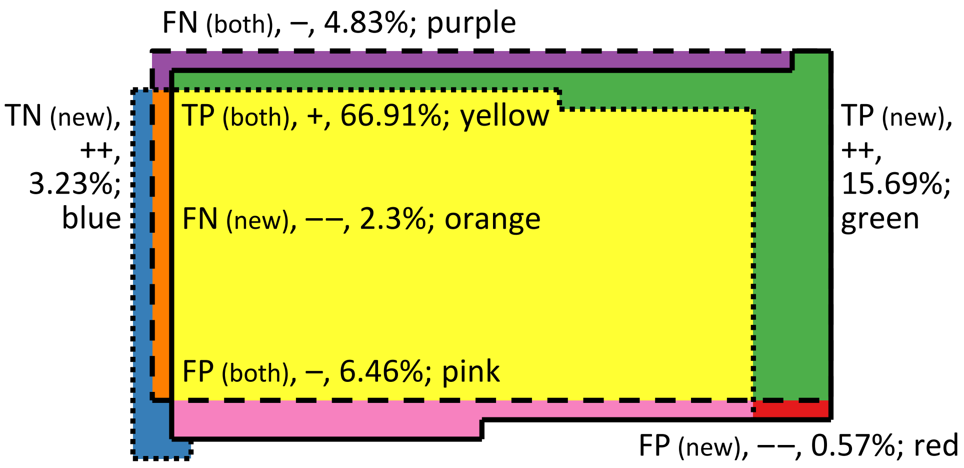

FN (both), –, 4.83%; purple
TN (new), ++, 3.23%; blue
TP (both), +, 66.91%; yellow
FN (new), ––, 2.3%; orange
FP (both), –, 6.46%; pink
TP (new), ++, 15.69%; green
FP (new), ––, 0.57%; red

Figure 4.2: Statistics of the comparison between our technique (solid line) and the original Mahalanobis brushing (dotted line), based on the user's goal (dashed line).

assuming the perspective of this work, green and blue points are very good (better than the original)!

Based on the percentages as presented in Figure 4.2, we can calculate the overall accuracy for the original technique to be $\approx$65% and for the new technique to be $\approx$92% (the very positive areas, green and blue, are significantly larger than the very negative results, orange and red). In addition, we also did a case study by looking at few representative good and bad cases in terms of the performance when compared with the old Mahalanobis brush and the ground truth.

4.1.1 Good case analysis

Figure 4.3 shows a typical situation—our method performs very well, based on the weighted covariance information, but the original Mahalanobis brush results in a clearly worse selection (note the many blue points, i.e., points, which the original technique falsely selects, while the new technique does not).

4.1.2 Bad cases

Figure 4.4 (left) shows a scatterplot with statistical information for each selection result (with respect to the ratio of false negatives to our brushing result and false positives to the goal). Most results lie in the bottom-left corner (the good corner), only a few results show significant numbers of false negatives / false positives. We choose six cases, highlighted by red points in the scatterplot for a detailed analysis, shown also in detail on the right in Figure 4.4:

Case 1: Details of the user's interaction have a big influence when selecting very small subsets (here, the start point of the user interaction deviates a bit from the center point of the target cluster, leading to a bad performance in this case).

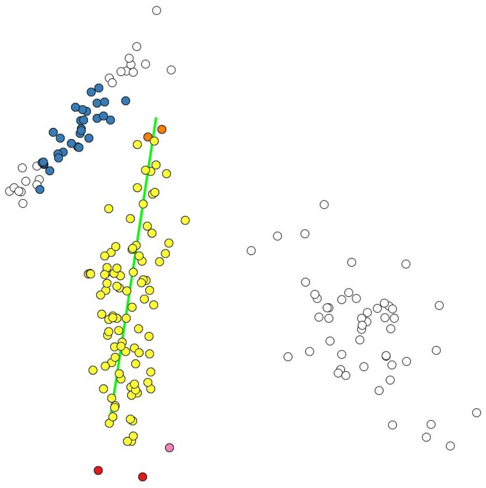


Figure 4.3: An example of a good match between the user's goal and the new brushing technique.

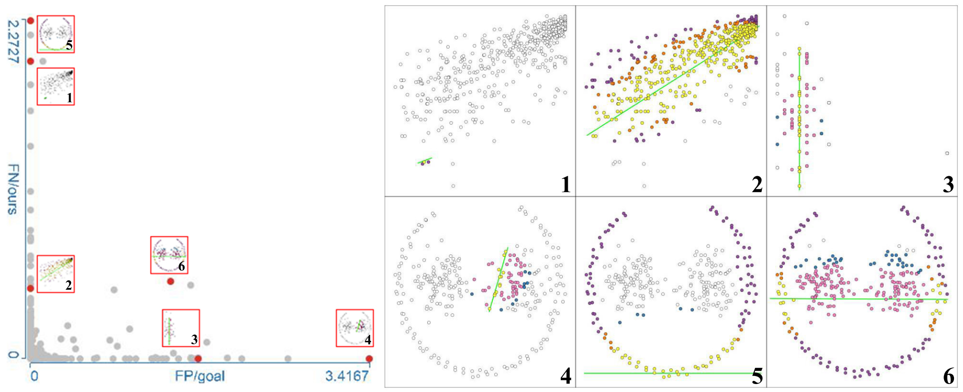


Figure 4.4: Left: A visualization of how certain selected cases from the user study deviated in terms of accuracy. Right: Six (relatively extreme) cases of suboptimal matches between the user's goal and the new brushing technique (details in the text).

Table 4.1: Statistics of the cross validation and training times

Group (validation)	Size factor of training data		
	1	4	16
G_1	95.73%	96.33%	**96.65%**
G_2	96.43%	**97.74%**	97.72%
G_3	98.39%	98.50%	**98.60%**
G_4	97.11%	97.13%	**97.24%**
G_5	95.00%	96.02%	**96.82%**
G_6	95.37%	97.17%	**97.46%**
Mean	96.34%	97.15%	**97.42%**
Variance	$1.3 \cdot 10^{-4}$	$6.9 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$
Time	$\approx$**5mins**	$\approx$20mins	$\approx$80mins

Case 2: Here, the new technique is too conservative and selects too few points (the old technique tends to select more circular regions).

Case 3: For scatterplots with linear structures that also are close to each other, our technique selects wider clusters than what users seemingly wish (in this data, several users wished to select individual “lines” of data points).

Case 4: Here, we think that it is close to impossible to correctly predict the user goal computationally.

Cases 5 and 6: In both cases, the user wished to select the outer ring—something, which is by design impossible with our (linear) selection technique (the click-and-drag interaction gives too little information to correctly select such “advanced” clusters).

In conclusion, we could demonstrate, quantitatively, that we significantly improve the accuracy of Mahalanobis brushing from ≈65% to ≈92%, while still using a very fast interaction technique (click-and-drag, the average time spent is only 41% of Lasso in our user study). In addition, in terms of efficiency, it only costs 20ms for the computation of brushing 2000 points, which enables the user obtain the brushing result in real time.

4.2 CNN-based brushing in scatterplots

We implemented the network and executed its training in Keras [11] which provides useful GPU acceleration. For the training and testing, we used a PC with an Intel Xeon E5-1650 CPU and an NVIDIA GeForce GTX 1080 GPU. The details of the parameter setting (regularizer, filters, dropout and so on) for the network are explained in Paper B (B.5.2).

For evaluating the new method—CNN-based brush, we used k-fold cross-validation [54]. In our evaluation, we set $k = 6$ and split the original 600 selections (from the user study presented in section 4.1) into six evenly sized groups G_i. For training the network, we use five groups as such (500 selections), or the extended training sets (see section B.8 for the data synthesis) with 2000 (size factor=4) or 8000 (size factor=16) selections, respectively.

Table 4.1 shows the results of the cross validation in terms of accuracy for the three training set sizes. With the extended training data, the trained model is more stable and has a higher accuracy. We also see that the performance of our model, when using the 16 times larger training set, is only slightly better as when trained with the four times bigger training set, with an overall accuracy 97.42%.

Based on the trained CNN model, we did a quantitative accuracy comparison with the new Mahalanobis brush using the 600 selections. For each point in each of the cases, we test whether the Mahalanobis brush selects it, whether our new technique selects it, and whether it should be selected (ground truth), looking at 252,400 points altogether. We use different colors (an accordingly colored Venn diagram is embedded in Figures 4.5 as color legend) in the visualization to represent the comparison result:

- yellow points (both brushing techniques succeed to select the point correctly; both true positive), purple points (both brushing techniques fail to select, i.e., both false negative), and pink points (both techniques select falsely; both false positive)—purple and pink points (both techniques fail) amount to about 4.57% of all cases, where at least one technique fails.
- green points (the CNN-based brush succeeds, while the Mahalanobis fails) and blue points (the original method selected falsely, while the new one does not)—these points represent the cases, where our new technique improves the so far best results and ≈89.3% of all cases, where at least one method fails, fall into this category!
- orange points (the CNN-based brush fails to select, while the Mahalanobis did) and red points (the new method selects falsely, while the old did not)—these points represent cases, where the CNN-based brush is worse than the Mahalanobis brush (only about 6.13% of all cases, where at least one method fails, amount to this category).

In total, when using the dice coefficient [15] to assess how well both techniques agree with the ground truth, we get excellent 99% for our new technique, as compared to 91% for the Mahalanobis brush. In terms of efficiency, the new technique is similarly fast as the previous Mahalanobis brush for small subsets; when brushing 2000 points, for example, it takes around 20ms. But when it comes to larger datasets, our method takes only 180ms when brushing 1 million points, while the Mahalanobis brush takes very long 110s for 100000 points, which is orders of magnitude too slow for a fluid interaction with large data. We also did a case analysis to look at the good and bad cases respectively.

4.2.1 Good cases

Figure 4.5 shows two typical situations, when our method performs very well, while the Mahalanobis brush results in a worse selection. On the left of Figure 4.5 we see many green points which the Mahalanobis brush would need to select, but actually does not, while our technique selects them correctly. Besides, we see many blue points on the right of Figure 4.5: by design, the Mahalanobis brush brushes an elliptical area that more often than the CNN has troubles in selecting elongated, skinny groups.

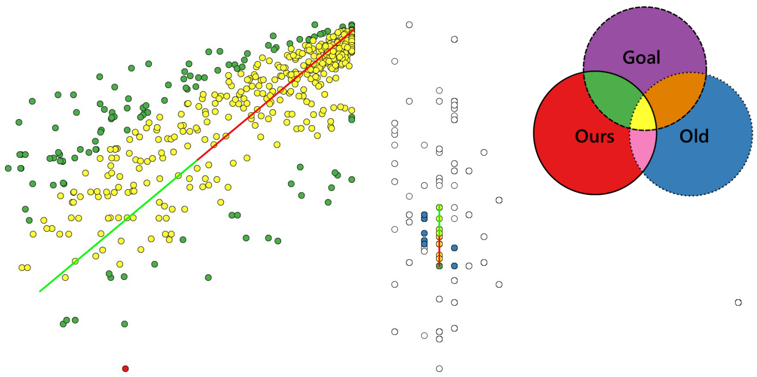


Figure 4.5: Two typical examples of a good match between the user's goal and the CNN-based brushing technique.

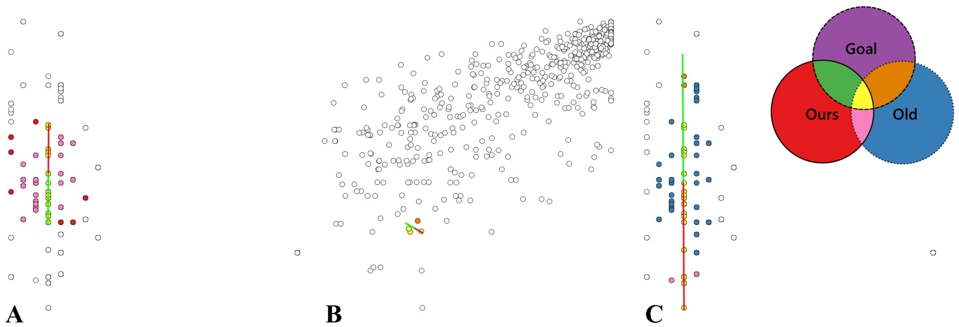


Figure 4.6: Three (most extreme) cases of suboptimal matches between the user's goal and the new brushing technique.

4.2.2 Worst cases

The worst cases, shown in Figure 4.6, are selected based on the ratio of false negatives to our brushing result (FN/ours) and false positives to the goal (FP/goal). Comparably large values in either of these measures identify our worst cases.

Case A, highest value of FN/ours: our technique has a problem to differentiate whether the user's goal is a circular region or an elongated group in some very similar regions.

Case B, highest value of FP/goal: the user's interaction has a big influence when selecting very small subsets (here, the start point of the user interaction deviates a bit from the center point of the target cluster, leading to a bad performance in this case).

Case C: Actually, this is not a very bad example, but we chose to show it here, because it performs relatively badly both in FP/goal and FN/ours (a relatively bad case that isn't really bad, after all).

To further substantiate the evaluation of our approach, we organized a new user study and invited ten users to test our CNN model on new data. In this study, we followed the previously established procedure of the user study mentioned in section 4.1, but provided 6 completely new datasets for the users to brush, which were not used in any way in the construction or training of our model. We got 120 new selections from this user study and the average accuracy is ≈95.3%, providing further evidence that our model is good at capturing relevant features of the user's brushing preference, rather than being biased by the training data.

4.3 Personalized CNN-based brushing in scatterplots

As users have their own brushing preference, it is important to explore how the user uses our brushing tool and test whether it is possible to allow the model to be customized. To evaluate our proposed framework with the CNN brushing for a single user, we developed a prototype with its interface written in Python and the CNN-based brushing model implemented in Keras. To gather the data for evaluation, we conducted a user study to collect the user's brushing data and then retrained the model to better fit it to a single user. In our user study, eight users were invited, all students or employees (seven from Delft University of Technology and one from the University of Bergen).

The user study was set up in two parts. In the first part, a scatterplot was provided to the users and then they could freely use the click-and-drag interaction to brush some data subset of their choice. In the second part, the brushing result based on the current CNN model was shown to the users immediately after they finished the interaction. Then they could think about whether the results were what they wanted originally. If not, they could use a lasso to do corrections (add and delete points, or directly specify the goal) until they were satisfied with the results. For a specific scatterplot, every user had to do 5 selections including the correction, if needed, before they would click the "next" button to switch to another scatterplot.

In order to investigate the influence of the retrained model based on the user's new brushing data, we asked each user to participate in the user study for 5 rounds and each scatterplot from 25 datasets showed up twice in total but in different rounds. For each round, the user needed to finish 5 selections in 10 different scatterplots, and these data were then applied to retrain a new, adapted brushing model which was then used for the

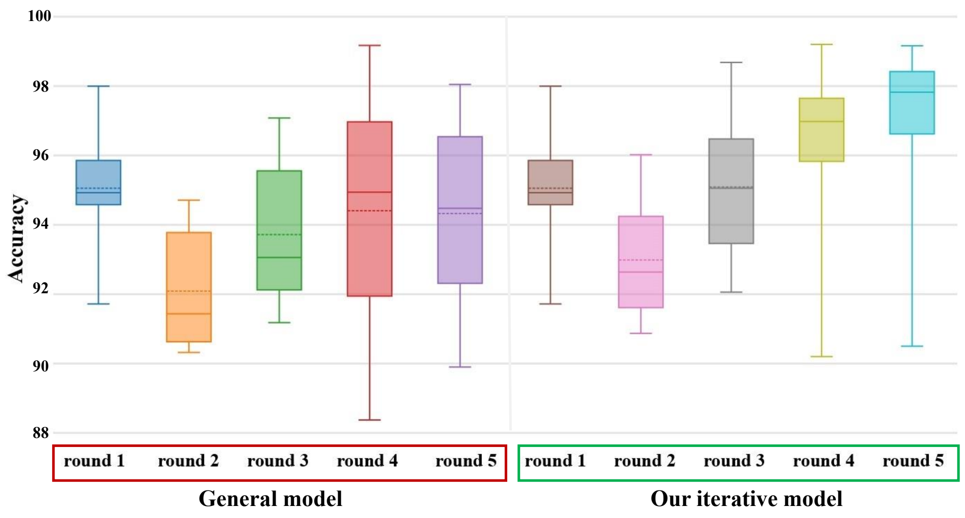


Figure 4.7: Boxplots of the brushing accuracy over rounds based on the general model (left) and our iterative model (right). The dash line and solid line in the box show the mean and median respectively.

next round. In the first round of the study, we directly used the already trained CNN brushing model (introduced in section 4.2) as the initial brushing model. Then this model will be retrained by the data obtained from the first round of the study. During the user study, the brushing results generated by the current model and the user's real goal and interaction were recorded.

For evaluating our proposed iterative model, we did a quantitative accuracy comparison with the general CNN-based brush mentioned above. Figure 4.7 is a statistical comparison with boxplots. The left five boxplots and right five boxplots show the brushing accuracy of 8 users in all five rounds based on the general model (left) and our iterative model (right) respectively. The accuracy of our iterative model in each round is calculated based on the retrained model from the last round. In the first round of the user study, the user's brushing results are computed by the general model without optimization based on the new data, thus the difference between the general model and our iterative model appears from the 2nd round to the 5th round. The dashed line in the box is the mean, if we compare the accuracy by pairs in terms of the round, we can see after iterative retraining, our model performs better than the general one (with higher median and mean). Besides, the size of the boxes on the right are smaller than the corresponding one on the left, this indicates our model has less variance and becomes more stable. Figure 4.8 shows an accuracy comparison with line plots, between the general model (top) and our iterative model (bottom). By looking at the accuracy variation altogether, we can obviously see a rising trend of accuracy over rounds on the bottom side while the accuracy of the general model is more distributed (no special trend except a decline from round 1 to round 2) on the top. This shows a strong indication that with additional data for retraining the model, it is able to gradually learn the brushing preference of a single user, which leads to a more personalized model from the general model. In

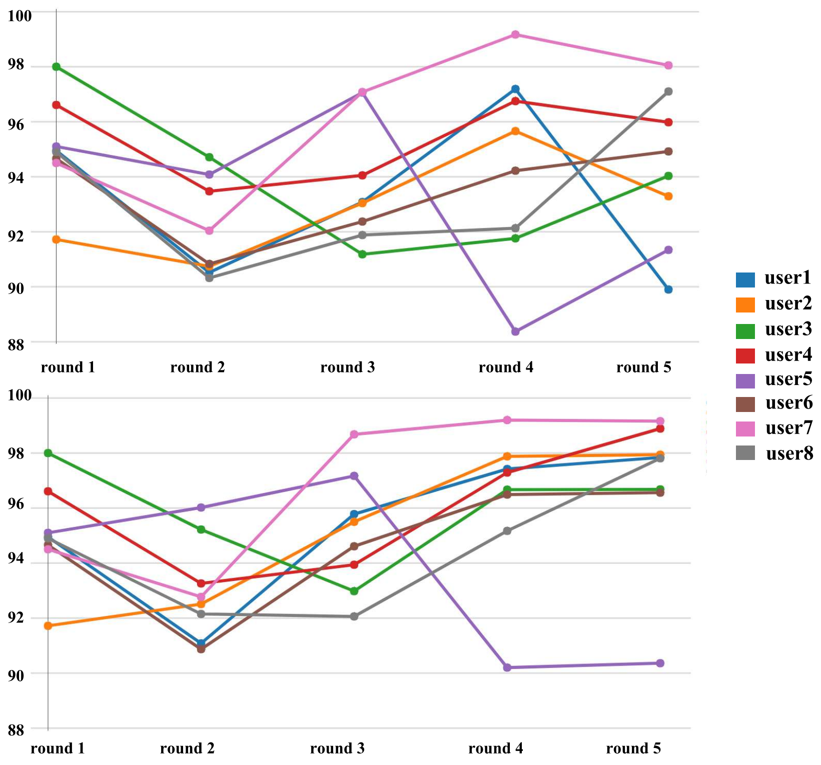


Figure 4.8: Line plots of the brushing accuracy over 5 rounds of 8 users based on the general model (top) and our iterative model (bottom).

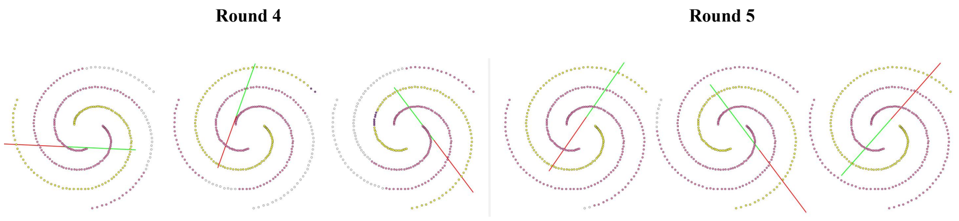


Figure 4.9: Worst performing cases of user 5 in round 4 (left) and round 5 (right).

addition, we also see an outlier (user 5) with a sharp accuracy decrease in the last two rounds. To investigate the reasons behind, we pick the 6 worst performing cases from user 5 in the 4th and 5th round and list them in Figure 4.9. The brushing results are compared to the user goal (encoded by color). The true positives (correctly brushed), true negatives (correctly not brushed), false positives (falsely brushed) and the false negatives (falsely left out) are colored in yellow, white, pink and purple, accordingly. We can see that the user tried to select a complicated spiral shape and this kind of selection does not exist in previous rounds, being almost impossible for the network to predict at first time. For the relatively low accuracy (actually 90.36% is still very good)

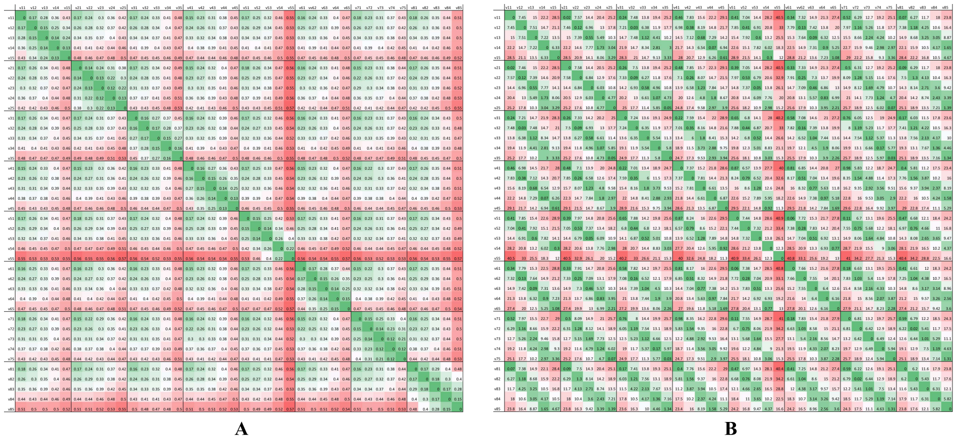


Figure 4.10: Vector similarity comparison of the CNN model parameters in each round of each user. Matrix A: angle between vectors. Matrix B: absolute length differences.

in round 5, as there are only 3 spiral-like cases from round 4 which contribute to the network retraining, so it is difficult to tweak the model to learn it.

During the retraining procedure in each round, the parameters of the network are updated. To understand what the network learns for each user and the relation between different user models, we extract the parameters of the CNN model in each round of each user, composing a vector with 35233 dimensions separately, which is denoted as v_{mn}, where m is the user ID and n is round number. In Figure 4.10, we compute the angle and absolute length differences to measure the similarities between these 40 different vectors, which are shown in two matrices with green (low) to red (high) as the color legend. In both matrix A and B, we see the outlier model in round 4 (v_{54}) and round 5 (v_{55}) of user 5 also have big difference with other users in this comparison. In addition, in matrix A, the angle between intra-user vectors are clearly smaller than inter-user's—this indicates that the model indeed approaches different user point in the parameter space. For matrix B, we see a clear pattern that the diagonals right next to the main diagonal are also very close to 0. This gives a strong impression that the model indeed adapts in small steps over the five rounds. In summary, the visualization of the CNN model parameters shows clear evidence that the iterative model learns reasonable parameters while being adapted instead of randomly search in the parameter space.

Overall, based on the quantitative performance of our iterative model in comparison to the general model, with the retraining procedure over time, we can see that the results in terms of accuracy show a clear rising trend compared to the general model: from 92.09% to 92.99% (+0.9%, with 10% new data), 93.72% to 95.09% (+1.37%, with 20% new data), 94.41% to 96.29% (+1.88%, with 30% new data) and 94.33% to 96.92% (+2.59%, with 40% new data). We assume the accuracy improvement can be even better with more user input. A practical advantage is that the retraining time cost is reduced to 4% of training a general model from scratch.

4.4 On KDE-based brush, compared to Mahalanobis and CNN-based brush

The KDE-based brushing model was implemented in *Processing* and we obtained the optimized parameter α (initial selection, determining the context of local data shape analysis; too small values of α lead to underselection while too large values to overselection) and β (overall scaling parameter, influencing the kernel size) after a optimization process (Details are introduced in D.5). Using the optimized parameters, we did an in-depth comparison between the Mahalanobis brush, the CNN-based brush, and the KDE brush, using the interaction information from the user study data (introduced in section 4.1).

Table 4.2 shows quantitative evaluation results for the three brushing techniques, according to a number of different measures [83]:

- **TP**: true positives (correctly brushed), total number and in percent
- **FP**: false positives (falsely brushed), total number and in percent
- **TN**: true negatives (correctly left out), total number and in percent
- **FN**: false negatives (falsely left out), total number and in percent
- **Accuracy**: correctly brushed or left out, compared to all, $\frac{\mathbf{TP}+\mathbf{TN}}{all}$ (the higher, the better)
- **Recall**: how much of the goal is brushed, $\frac{\mathbf{TP}}{\mathbf{TP}+\mathbf{FN}}$ (the higher, the less "underbrushing")
- **FPR (fall-out)**: how much of the non-goal is brushed, $\frac{\mathbf{FP}}{\mathbf{FP}+\mathbf{TN}}$ (the lower, the fewer **FP**)
- **FOR (false omissions)**: how much non-brushed is goal, $\frac{\mathbf{FN}}{\mathbf{FN}+\mathbf{TN}}$ (the lower, the fewer omission.)
- **TS (threat score)**: how much of $\cup(\text{brush}, \text{goal})$ is **TP**, $\frac{\mathbf{TP}}{\mathbf{TP}+\mathbf{FP}+\mathbf{FN}}$ (the higher, the better)
- **Precision**: how much of the brush is goal, $\frac{\mathbf{TP}}{\mathbf{TP}+\mathbf{FP}}$ (the higher, the less "overbrushing")
- **F$_1$ score**: combination of precision and recall (harmonic mean), $\frac{2\mathbf{TP}}{2\mathbf{TP}+\mathbf{FP}+\mathbf{FN}}$ (the higher, the better)
- **MCC (Matthews correlation coefficient)**: measuring the quality of binary classification,
$\frac{\mathbf{TP}\cdot\mathbf{TN}-\mathbf{FP}\cdot\mathbf{FN}}{\sqrt{(\mathbf{TP}+\mathbf{FP})(\mathbf{TP}+\mathbf{FN})(\mathbf{TN}+\mathbf{FP})(\mathbf{TN}+\mathbf{FN})}}$ (the higher, the better)

According to the quantitative evaluation in the top part of Table 4.2, CNN-based brush performs best with respect to **TP**, **FP**, **TN**, and **FN**. When comparing the two empirical models, KDE-based brush results in more **TP** than Mahalanobis brush (and in fewer **FN**), while it leads to more **FP** and fewer **TN**. This could indicate that KDE-based brush is better at recognizing the user's brushing goal, but at the cost of more false negatives (overbrushing). By looking at the bottom part of Table 4.2, showing eight

Table 4.2: Quantitative evaluation of the three brushing techniques, based on several measures and computed for all 600 selections

	TP	**TP** (%)	**FP**	**FP** (%)	**TN**	**TN** (%)	**FN**	**FN** (%)
Mah.	50 737	20.10%	5 189	2.06%	191 682	75.94%	4 792	1.90%
KDE	52 436	20.77%	9 583	3.80%	187 288	74.20%	3 093	1.23%
CNN	55 321	21.92%	929	0.37%	195 942	77.63%	208	0.08%

	accuracy	**recall**	**FPR**	**FOR**	**TS**	**precision**	**F$_1$**	**MCC**
Mah.	96.05%	91.37%	2.64%	2.44%	83.56%	90.72%	91.04%	88.51%
KDE	94.98%	94.43%	4.87%	1.62%	80.53%	84.55%	89.22%	86.18%
CNN	99.55%	99.63%	0.47%	0.11%	97.99%	98.35%	98.98%	98.70%

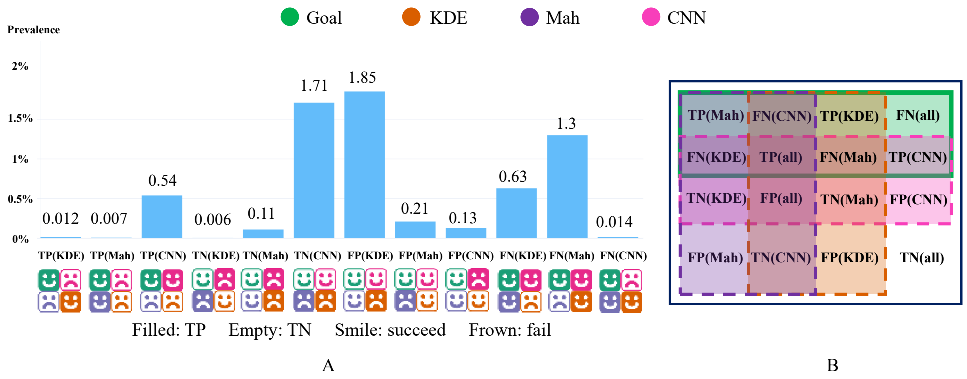


Figure 4.11: A three-way comparison of four sets (A): *actual goal* (green), *Mahalanobis brush* (violet), *KDE brush* (orange), *CNN brush* (pink). The cases of *all techniques correctly brush the goal* (19.45% of all), *all techniques leave out the non-goal correctly* (73.86%), *all techniques select falsely* (0.12%) and *all techniques fail to select* (0.05%) are not shown, focusing on those cases, where at least one technique has a problem (**FP** or **FN**) and at least one technique succeeds (**TP** or **TN**); Venn diagram, illustrating the relation between the sets (B).

different measures for judging the quality of the classification, CNN-based brush also outperforms the two empirical models (in all measures). Comparing the two empirical models, KDE-based brush seems to be better than Mahalanobis brush in **recall** (how much of the goal is brushed) and in the **false omission rate** (how much of the non-brushed view is actually goal), while Mahalanobis brush appears to be better in all other six measures.

In addition to comparing each method with the goal, we also did a threefold comparison to see the relation of the four related sets (actual goal, brushed by the Mahalanobis brush, the KDE brush, and the CNN brush), shown in Figure 4.11 (A). The Venn diagram in Figure 4.11 (B) is an illustration of the threefold comparison: the thick green line surrounds the actual goal, the dashed violet line surrounds all that's brushed by the Mahalanobis brush, the dashed orange line surrounds what the KDE brush selects, and the dashed pink link surrounds, what the CNN brush selects (in the shown schematic, the areas do not correspond to the proportions of the respective cases).

For each point in the 600 cases from the user study [22] – 252 400 points, altogether – we check whether it belongs to the brushing goal (green), whether the Mahalanobis brush selects it (violet), whether the KDE brush selects it (orange), and whether the CNN brush selects it (pink), leading to $2^4 = 16$ possible situations per point. The relative prevalence of these situations is shown in Fig. 4.11 (A), leaving out the dominating "good" cases of *all techniques brush a goal point*, **TP**(all), 19.45% of all, *all techniques leave out a non-goal point*, **TN**(all), 73.86%, *all techniques select falsely*, 0.12% of all cases, and *all techniques fail to select*, 0.05% of all cases, emphasizing the situations, where at least one brushing technique has a problem (**FP** or **FN**) and at least one technique succeeds (**TP** or **TN**). The label of each situation indicates its characteristics – if one technique has a problem, then this is indicated (for ex., "FN(Mah): 1.30%" indicates the situation, when only the Mahalanobis brush fails to select a goal point); when two techniques have a problem, the opposite is done (for ex., "TN(Mah): 0.11%" indicates the situation, when only Mahalanobis brushing leaves out a non-goal point, while both other techniques incorrectly select it). As an additional mark in Fig. 4.11 (A), a combination of emojis is used to indicate the situation: a frowny indicates a problem (color: technique, filled: **FP**, empty: **FN**), while a smiley shows that the technique succeeded (filled: **TP**, empty: **TN**). Below, we briefly address all cases:

- **TP**(all), **TN**(all): in most cases, all three techniques do the right thing (select a goal point, or leave a non-goal point out): 19.45% are consistently selected goal points, 73.86% are consistently left-out non-goal points – altogether, 93.31% of all cases are "good" in general!
- **FN**(KDE), **FN**(Mah), **FN**(CNN): the indicated brush is the only one to not select a goal point; of these, the case where the Mahalanobis brush underbrushes is most prevalent (1.30%, as compared to 0.63% and 0.014% of underbrushing by the KDE-based brush and CNN-based brush respectively).
- **FP**(KDE), **FP**(Mah), **FP**(CNN): the indicated brush is the only one to falsely select a non-goal point; of these, clearly the case where the KDE-based brush overbrushes is most prevalent (1.85%, as compared to 0.21% (Mah) and 0.13% (CNN)).
- **TP**(KDE), **TP**(Mah), **TP**(CNN): the one indicated brush (Mahalanobis, KDE,

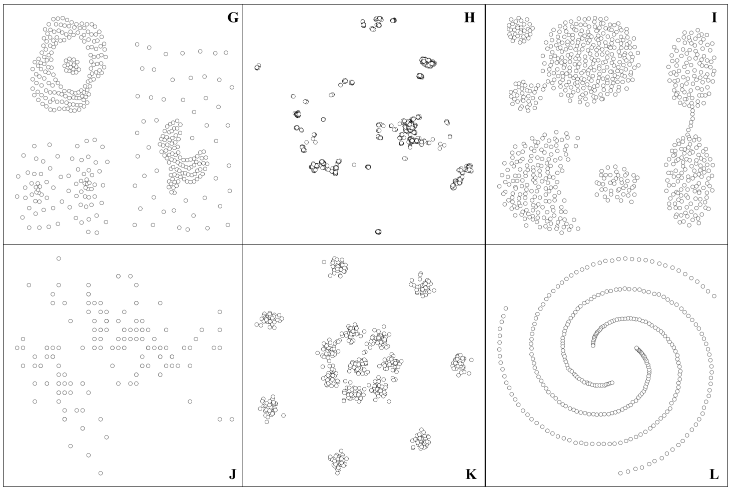


Figure 4.12: Overview of the six datasets in the follow-up user study.

CNN) succeeds to select the goal, while the two others fail; of these three, clearly the case where only the CNN-based brush succeeds is most prevalent (0.54%, as compared to 0.012% (KDE) and 0.007% (Mah)).

- **TN**(KDE), **TN**(Mah), **TN**(CNN): the one indicated brush succeeds in not selecting a non-goal point, while the other two select it falsely; of these, also the case where only the CNN-based brush is right is most prevalent (1.71%, as compared to 0.11% (Mah) and 0.006% (KDE)).
- **FP**(all), **FN**(all): in these rare cases, all the three techniques do the wrong thing (select falsely or fail to select), amounting to 0.12% and 0.05%, respectively.

To further substantiate the evaluation of the three techniques, we also tested the three methods on new data from another user study, which has not been used for training of the models. This user study used six completely new datasets (shown in Figure 4.12: G is compound data [108], H is personal hiking data, I is aggregation data [29], J is the omnipresent Iris data, K is R15 data [101], L are three spirals [9]) and 10 users provided 12 selections each, leading to 120 selections in total. The corresponding quantitative comparison is shown in Table 4.3 and Figure 4.13, considering 86 700 points in total.

Comparing the quantitative evaluation based on the two user studies, we see that CNN-based brush produces many more **FP** in the follow-up study (0.37% → 1.48%), leading also to a higher fall-out value (0.47% → 1.91%). Mahalanobis brush and KDE-based brush produce much less **FN** in the follow-up study (1.90% → 0.28% and 1.23% → 0.35%, respectively), while CNN produces more **FN** (0.08% → 0.15%). With respect to the other measures, KDE's **threat score** got better in the follow-up study (80.53% → 89.75%) and became more similar to the others. Mahalanobis's **threat score** improved also (83.56% → 91.39%), while CNN's **threat score** worsened (97.99% → 93.10%). Besides that, Mahalanobis' **recall** got better in the follow-up study (91.37% → 98.75%) and became more similar to the others (all methods are very good). CNN's **precision** and **accuracy** went down in the follow-up study (98.35%

Table 4.3: Quantitative evaluation of the three brushing techniques, based on several measures and computed for all 120 selections from the new user study:

	TP	**TP** (%)	**FP**	**FP** (%)	**TN**	**TN** (%)	**FN**	**FN** (%)
Mah.	18 989	21.90%	1 549	1.79%	65 921	76.03%	241	0.28%
KDE	18 927	21.83%	1 859	2.14%	65 611	75.68%	303	0.35%
CNN	**19 100**	**22.03%**	**1 286**	**1.48%**	**66 184**	**76.34%**	**130**	**0.15%**

	accuracy	**recall**	**FPR**	**FOR**	**TS**	**precision**	**F_1**	**MCC**
Mah.	97.94%	98.75%	2.30%	0.36%	91.39%	92.46%	95.50%	94.25%
KDE	97.51%	98.42%	2.76%	0.46%	89.75%	91.06%	94.60%	93.10%
CNN	**98.37%**	**99.32%**	**1.91%**	**0.20%**	**93.10%**	**93.69%**	**96.43%**	**95.44%**

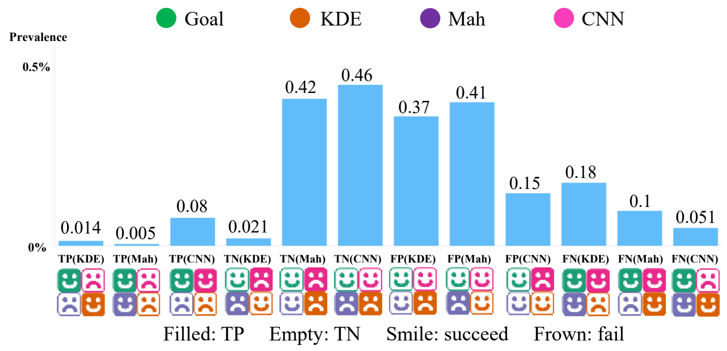


Figure 4.13: Threefold comparison between the Mahalanobis brush, KDE brush and the CNN brush based on the user's goal from a follow-up user study (details in the text). Note that all relative prevalences are below 0.5%.

→ 93.69%, 99.55% → 98.37%) and became more similar to the others. In addition, the **FPR** and **FOR** of KDE-based brush are both largely reduced (4.87% → 2.76%, 1.62% → 0.46%), becoming more similar to the other two methods. In terms of the **F_1 score** and **MCC**, both Mahalanobis brush and KDE-based brush improved (91.04% → 95.50%, 88.51% → 94.25%; 89.22% → 94.60%, 86.18% → 93.10%), while CNN's measures worsened (98.98% → 96.43%, 98.70% → 95.44%).

For the threefold comparison, we see a performance decline of CNN-based brush in all **TP**(CNN) (0.54% → 0.08%), **TN**(CNN) (1.71% → 0.46%), **FP**(CNN) (0.13% → 0.15%) and **FN**(CNN) (0.014% → 0.051%). For the empirical models, KDE achieves better results in all **TP**(KDE) (0.012% → 0.014%), **TN**(KDE) (0.006% → 0.021%), **FP**(KDE) (1.85% → 0.37%) and **FN**(KDE) (0.63% → 0.18%), while Mahalanobis brushing performs better in **TN**(Mah) (0.11% → 0.42%), and **FN**(Mah) (1.30% → 0.10%) but worse in **TP**(Mah) (0.007% → 0.005%), and **FP**(Mah) (0.21% → 0.41%).

While almost all measures got better for both Mahalanobis brush and for KDE-based brush in the second user study, they all got worse for CNN-based brush – at least a bit. It is important to see, however, that CNN-based brush still outperformed both other methods in all indicators (even though they are much more similar in the follow-up study). This could reveal one important disadvantage of the CNN brush, namely that it is less general, when compared with the empirical models.

Additionally, based on the comparison between KDE-based brush and Mahalanobis brushing, we could not see that KDE-based brush would outperform Mahalanobis brush, after all. This assumption was originally made, because we thought that more carefully considering the local data distribution should help to further improve the technique's accuracy (as a nonlinear method, KDE-based brush should have better abilities to adapt to nonlinear structures in the data). So far, we cannot rule out that we have overlooked another limitation when realizing the KDE-based approach – either a conceptual one, or a limitation of our implementation. Accordingly, we see it still possible that another solution could achieve a further improved accuracy.

As another point of this discussion, we note that empirical modeling comes with the advantage of an explainable result (for example, we know how different values of α and β influence the results), while the excellent performance of the DL model comes at the cost of a poor interpretability (including some uncertainty concerning the stability of its predictive power, see section D.6.4). This comparison leads to the interesting question of how much accuracy we are willing to sacrifice for a good interpretability.

4.5 LSTM-based visual query system

The prototype of our proposed visual query system was implemented in *Python* with the network training based on *Keras*. To evaluate the accuracy of our proposed matching model, the most direct way is to ask users to tell us whether the matching result is good or not. As a baseline for comparison, we chose two state-of-the-art techniques—the recently published Qetch algorithm [71] and DTW. The well-established DTW metric was chosen as one of the best options for distance measures in time series data [16], while the authors of Qetch claimed its strength over DTW for freehand sketch and matching for high-level task (in terms of time spent). In this user study, the users were asked to rate the matching results computed by Qetch, DTW, and our new method, leading to a quantitative, comparative evaluation reflecting the user's perspective.

For the user study, 10 users were invited. For each user, 8 new time series data (with lengths ranging from 40 to 1440, none of them used for training before) were provided in sequence to test the generality of the proposed model. The procedure of this user study consisted of two steps:

1. To start, a dataset with a useful default smoothing level (see above) was presented to the user. For each dataset, the user could interact with the interface and specify the targeted scale of the visualization by zooming with the mouse. In addition, users could also adjust the smoothing level by using a slider.

2. Then, the user was asked to sketch a pattern he/she wanted to look up from the time series data on the sketching panel. Based on the user' sketch, we computed the three best matching results from our new model, the Qetch algorithm, and using DTW, respectively. Then, we showed these three results (in random order and without telling which is which) to the user and asked them to rate the similarity between their sketch and these three results separately. As Qetch and DTW are highly competitive matching algorithms, capturing the difference between good and very good results requires more detail, so we offered a once refined nine-points range (also from 0 to 1, $s = 0, 0.125, 0.25, 0.375 \ldots 1.0$) for rating instead of the courser five-points Likert scale, delivering the required details for a proper comparison.

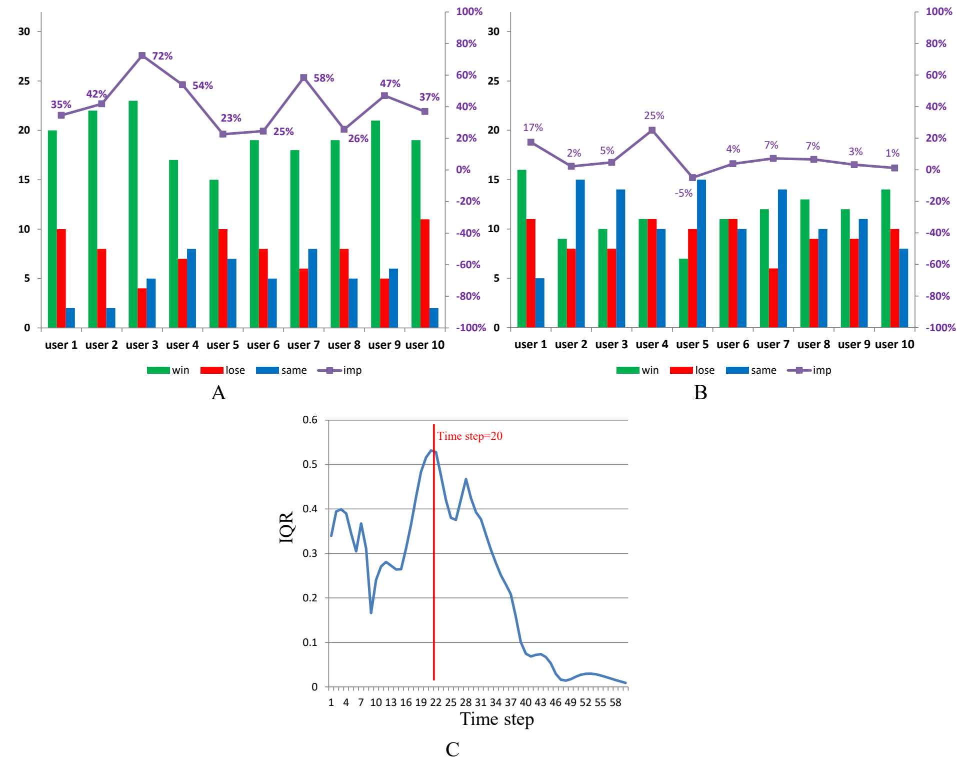


Figure 4.14: A: comparison between our method and the Qetch algorithm. B: comparison between our method and DTW. Green bars show how often our technique was preferred (red: how often Qetch/DTW was preferred; blue: tied) and the purple line indicates per user the improvement as achieved by our method (positive: average improvement due to our method). C: IQR values showing output variation between different time series from the 1st time step to the 60th, indicating the change of variation over time.

During this study, we recorded the similarity rating from each user for a subsequent quantitative analysis (see below). For each data, the user sketched four times and did four sets (our method, Qetch, and DTW) rating. Accordingly, we collected altogether 320 sets of rating results from 10 users. In addition, the time cost of computation were recorded for comparing efficiency, also. Before the user study started, every users was offered a training session to get familiar with the interface and the interaction of the user study.

Figure 4.14 (A) and Figure 4.14 (B) are statistical comparisons based on the rating results (32 pairs from each user) from the user study. The results are shown as bars (how often one technique was preferred) and as a line graph (average improvement). Green bars show (per user) the number of times that our method was rated with a higher similarity than Qetch (A) or DTW (B), while red bars represents the contrary and the tied rating is represented as blue bars. Further, we also compute the average improvement (denoted as *imp*) of our method as compared with Qetch/DTW in terms of the similarity value: $imp = \frac{s_{\text{our}} - s_{\text{qetch}}}{s_{\text{qetch}}}$ or $\frac{s_{\text{our}} - s_{\text{dtw}}}{s_{\text{dtw}}}$ (s_{our}, s_{qetch}, and s_{dtw} denote the average similarity rating of our method, the Qetch algorithm, and DTW, respectively) and show it (also per user) as line graph in purple.

By looking at the bar graph in Figure 4.14 (A), we clearly see that all users preferred our matching over the Qetch algorithm and that our method was preferred about 2.5 so often as the other way around (193:77). The line graph shows that all average improvements are positive, providing a clear evidence that our method is closer to the user's perception in terms of similarity. More specifically, the average improvement is $\approx 42\%$ as $\bar{s}_{\text{our}}$=0.64 and $\bar{s}_{\text{qetch}} = 0.45$, where $\bar{s}_{\text{our}}$ and $\bar{s}_{\text{qetch}}$ are the average similarity values of our method as compared to the Qetch algorithm. Based on this evaluation, we are confident to conclude that in general our method performs significantly better than the Qetch algorithm, when the matching results are directly judged by the users.

By looking at the chart in Figure 4.14 (B), 7 out of 10 users preferred our technique over DTW according to the bar graph, while the average improvements are positive (in our favor) for 9 out of 10 users, when looking at the line graph. The overall ratio of users preferring our method over DTM is around 1.2 (115:93) and the average improvement for each user is $\approx 7\%$ with $\bar{s}_{\text{dtw}} = 0.6$ ($\bar{s}_{\text{dtw}}$ is the average similarity of DTW rated by all the users). Overall, and even though the improvement numbers are clearly smaller than in the comparison with Qetch, this suggests that our method is slightly better than DTW in terms of accuracy (at least not wrose) when evaluated directly from the user's perception.

Furthermore, and since a swift user–computer dialogue in visual query systems is highly dependent on the efficiency of the involved interactions, we also compared the computation cost of the three methods. According to the user study, the average computation costs of our method, Qetch, and DTW, are 33ms, 132ms and 3.6s, respectively. Based on this data, we see clearly that our method is the most efficient one due to its linear complexity, while DTW is the slowest and unable to achieve a real-time interaction.

Besides the quantitative evaluation in terms of accuracy and efficiency, we also examined the output of each step of the LSTM network to check whether the network learns meaningful information. The reason for doing this was that the network was implicitly taught to only remember the information of a specific length of previous time steps (we set this to 40 in our experiment) with a goal to mimic a sliding window for matching. To investigate whether the network has successfully achieved this important feature, we collect the output (namely the hidden state **h**) of each time step of several time series data with different left lead data and compute the output variations over the time steps to analyze whether the network is able to discard long term dependencies. The details are illustrated below.

In our training, the time series data clips were sampled at 40 points with synthesized lead data "on the left" at 20 points. For looking into the information as learned by the network, we did an experiment that generated 20 time series snippets with length 60 (denoted as G_i, $i \in \{1, \ldots, 20\}$), where the trailing 40 entries for each time series were the same, but the first 20 varied according to our synthesis procedure. As a reference, we have another time series (denoted as ref) that has the same last 40 entries but with a real lead data that is different from all synthesized lead data G_i. We then iterated the time series and the reference time series over the 60 time steps by using the already trained model and obtained a set of outputs with format 20×1×60×20 and 1×60×20. We then computed the cosine similarity between all outputs of G_i and ref per time step, leading to a set of 20 cosine similarities. We compute the inter-quartile range (IQR), i.e., the difference between the 75%- and the 25%-percentile, as a robust indicator of the variation over the time steps.

After iterating over all the time steps from 1 to 60, we have 60 IQR values which represent the output variations over the 60 time steps and this information is shown in Figure 4.14 (C). As we mentioned, from the 1st time step to 20th time step, the network output corresponds to lead data – since all of the lead data was randomly synthesized the variation during this period is fluctuating at a relatively high level. After time step 20, however, we see a decline of the IQR values, correlated with the networking reading actual time series data. Towards the 60th time step, the IQR values approach 0, which meets our expectation that the output at the last time step almost only represents the information of the previous 40 time steps. In summary, this statistics gives us a strong indication that the LSTM network has been successfully trained to understand that only the information from a certain length of the previous time steps should be taken into account in each time step.

Overall, we can demonstrate, quantitatively, and in comparison with the recently published Qetch algorithm as well as the classical distance measure DTW, that our LSTM-based solution leads to an improvement in terms of the overall similarity ($\approx$42% to Qetch and $\approx$7% to DTW), rated by users in a user study, while enabling a fast interaction ($\approx$4 times faster than Qetch and $\approx$100 times faster than DTW).

In this chapter, we demonstrate how machine learning can be used to improve the user interaction in visual analytics by making use of the user's interaction data. To briefly summarize: the Mahalanobis brush and the KDE-based brush are empirically constructed by conventional machine learning algorithms and optimized by user study data, with the advantage of a better interpretability and stability of the results. While the CNN-based brush and the LSTM-based time series matching algorithm are implicitly modeled, based on user study data, achieving the best performance in terms of accuracy and efficiency, but with the disadvantage of being a "black box" mechanism. The quantitative results deliver a positive signal that the collaboration between ML and VA can be beneficial and promising to drive innovations. In addition, we hope our work can motivate people to pay more attention to the user feedback as this valuable data can be collected to optimize the existing models or even create new effective models.

Chapter 5

Conclusion and Future Work

Fast and accurate interaction techniques are critical for data analysts to achieve a seamless and intuitive visual communication with the visual analytical systems. More specifically, the analyst should be able to fully focus on the task at hand instead of being distracted by overly technical or complex user interaction. This practical demand drives the need for smart interaction tools which can intelligently take user's feedback into consideration and then require as little user input as possible after a proper learning procedure.

In this thesis, we demonstrate how machine learning can be utilized to further improve central interaction techniques in visual analytics. By learning the relation between the data subset to be selected and a swift sketch by the user to do the selection, we achieve multiple solutions, which are fast and also very accurate. Moreover, we also discuss the accuracy, practicability, robustness and interpretability between empirical modeling and deep learning-based modeling by taking the brushing techniques we implemented together as an example, in order to investigate the human influence in model design. This comparison also bring up an interesting debate about how to trade off different strengths when choosing/designing a model.

Based on our observation, there is a mentionable amount of work which has been done to integrate machine learning knowledge to visualization tasks. However, this combination targeted for improving user interaction is still rare to see. In addition, we notice that only a small subset of ML techniques are incorporated in visualization solutions and that the visualization community is relatively slow in catching up with advanced ML techniques. We hope the work done in this thesis can inspire people to increase more awareness and interest in the ML domain and motivate others to follow a similar approach in their visualization research, i.e., to do an automatic optimization of visualization parameters, based on data from a corresponding user study.

In the future, we see several opportunities to further extend our work, including

- allowing the user to improve the interaction technique instantaneously while using it.
- taking innovative advantage of both sides—empirical modeling *and* machine learning. For example, to automatically learn the kernel size or to design the deep learning input on the basis of the KDE.
- possible improvements of the additional training data synthesis as it plays a crucial role for the learning process. One possible way is to use the generative adversarial network (GAN), which has been successfully applied in image generation

and synthesis. We assume it can synthesize more realistic variations of the user interaction.

- the extension of our principal approach to other interactions in visual analytics.

Part II

Included papers

Paper A

User-study based optimization of fast and accurate Mahalanobis brushing in scatterplots

Chaoran Fan and Helwig Hauser

University of Bergen, Norway

Abstract

Brushing is at the heart of most modern visual analytics solutions with coordinated, multiple views and effective brushing is crucial for swift and efficient processes in data exploration and analysis. Given a certain data subset that the user wishes to brush in a data visualization, traditional brushes are usually either accurate (like the lasso) or fast (e.g., a simple geometry like a rectangle or circle). In this paper, we now present a new, fast and accurate brushing technique for scatterplots, based on the Mahalanobis brush, which we have extended and then optimized using data from a user study. We explain the principal, sketch-based model of our new brushing technique (based on a simple click-and-drag interaction), the details of the user study and the related parameter optimization, as well as a quantitative evaluation, considering efficiency, accuracy, and also a comparison with the original Mahalanobis brush.

This article was published in *Proc. Vision, Modeling, and Visualization (VMV 2017)*, pages 77–84, 2017

A.1 Introduction

In interactive visual data exploration and analysis, linking and brushing is a central and well-established interaction technique for relating data aspects across coordinated multiple views [77, 87]. The principles of brushing were first described by Becker and Cleveland [2], who defined brushing as an interactive method for painting a group or subsets of points with a square, circle, or a polygon, i.e., the brush. A key functionality in standard instances of coordinated multiple views is that brushing leads to a consistent highlighting of the selected data in all linked views, for example, by coloring them consistently. This amounts to one important form of focus+context visualization [35], enabling the fast and effective exploration of data relations, which are too challenging to show in just one view.

As popular and common as linking & brushing has become in modern visual analytics solutions, already many different techniques for brushing have been realized, including many variants from the following categories:

- *brushing using simple geometries*—examples of this most common approach include the rectangular or circular brushing on scatterplots, line-brushing on data graphs [56], etc.
- *lassoing*—the user selects data subsets by drawing a geometrically detailed lasso around a target group of item representations
- *logical combinations of simple brushes*—the user refines the data selection by using multiple brushes and combining them using logical operators [17, 72]
- *sketch-based brushing*—the user sketches a shape onto a visualization and some selection heuristic is used, usually exploiting a related similarity function, to determine which data are actually selected [76]

Each brushing technique can be discussed in terms of its advantages and disadvantages and two criteria are particularly important:

- *efficiency*—how fast is the brushing interaction; does it enable a fluid data exploration/analysis [20, 98]?
- *accuracy*—does the brushing interaction lead to a selection of exactly the data subset, which the user wished to select?

In many cases, there is an unfortunate competition between these criteria: Many brushing techniques are indeed fast—we think of a brushing technique to be fast, if only one click (or only very few atomic interactions of that kind) are needed to actually specify the brush, leading to a swift user–computer dialogue during the data exploration/analysis [8]. Classical examples include the use of simple brushing geometries (rectangles, circles, etc.) as well as sketched brushes, where only a quick gesture is used for brushing. A common disadvantage of all these fast techniques is that it can be difficult to accurately brush a particular data subset.

On the other hand, we certainly find brushing techniques that are fully accurate—likely with lassoing being the most prominent example besides others such as the logical combination of simple brushes. With these techniques, it is straight-forward to select subsets of interest accurately. This benefit, however, comes at the price of being slower, in general—specifying a lasso, point by point, for example, easily becomes a unit task by itself [8], potentially interrupting the exploration/analysis process.

In our research, we have studied the question of how close one can get to successfully integrating both criteria in one technique and in the following we present a successful solution (we chose the example of brushing in scatterplots as our study case—we think, however, that our principle approach is extensible to other views and according brushes). Our solution is based on an extended version of the previously published Mahalanobis brush [84], which we have extended and further optimized using data from a user study with 50 participants. Our quantitative evaluation shows that we significantly improved the brushing accuracy from ≈65% (original Mahalanobis brush) to ≈92% (our new technique). Our technique is as fast as a simple click-and-drag interaction—the original Mahalanobis brush required only one location (one click), but depended, in addition, on an off-screen size parameter (overall brush size).

Since brushing is central in most modern visual analytics systems, we see our research very relevant—optimizing at the heart of a common procedure has the strong potential of significant impact.

A.2 Related Work

In the following, we review a few pieces of important related work, before going into detail with respect to our new brushing technique. We first review, in short, some critical works concerning brushing for visual analytics, before we then discuss related work concerning the optimization of interaction techniques.

A.2.1 Brushing techniques

Many variations of brushing have been proposed over the years, each with its own strengths and weaknesses—for example, in terms of their ease of use and the degree of control which the user has. Brushing is intrinsically based on the interaction between the user and the system, often a combination of mouse/cursor motions and button clicks. Less usual methods, based on eye/head tracking, for example, or gestures in a virtual reality environment, have also been proposed [107].

Brushing in scatterplots is often based on the use of simple geometric shapes such as a rectangle or circle to select the data items, or using a lasso to specify the brush region more accurately.

Several extensions to simple brushing have been published, including techniques to formulate more complex brushes by combining multiple brushes using logical operators. Martin and Ward [72], for example, allow the user to configure composite brushes by applying logical combinations of brushes, including unions, intersections, negations, and exclusive or operations.

Similarity brushing [76, 80] is an interesting alternative in that it is based on a fast and simple sketching interaction—the user uses a swift and approximate gesture (for example, drawing an approximate shape that the data should follow) and then a similarity measure is used to identify, which data items actually are brushed by such an advanced brush. This way, the interaction is fast, but likely not 100% accurate.

Recently, the Mahalanobis brush was presented as an interesting alternative for brushing scatterplots [84]. The user uses a simple interaction (like a simple click into the center of some coherent data subset to be selected) for brushing the data. The link between the interaction and the actual selection is realized on the basis of a simple analysis of the underlying data (a local covariance matrix indicates the overall shape and orientation of the data to be brushed, forming then the basis for a local Mahalanobis metric, which is then used as a distance measure to select the data).

While this technique is already giving quite good results, it still has certain limitations, including: a non-optimized selection of the local context for the Mahalanobis computation (improved in our solution), at least one off-screen parameter for the brush size (no free parameter in our solution), and empirical parameters (we use the data from a user study to optimize the relevant parameters).

A.2.2 Optimization based on user data

Obviously, the user plays a key role during all sorts of interaction. Thus, efforts have been invested to take the user behavior into account, when improving the performance of interaction techniques.

The design of adaptive user interfaces, for example, is one of the most classical examples in this respect, enabling the interface to recognize user action plans by tracing and analyzing the user's action sequences [60, 65].

Lieberman et al. [63] developed the "Let's Browse" application to assist the user browsing websites by tracking the user's behavior and predicting items of interest, accordingly.

In the mobile phone architecture design area, Shye et al. [95] developed a logger application for mobile phones and released it "into the wild" to collect the traces of real users. Then, they used these traces for characterizing the power consumption on the mobile phones, eventually leading to the development of applications that optimize the consumption of battery power.

Considering visualization research, in particular, we cannot find a lot of related work (in terms of optimizing interaction techniques, based on user data). Instead, we see this as a highly interesting chance for interesting and relevant innovation.

A.3 The principal approach

The overall goal of our research was to devise a brushing technique, which is both fast and accurate. In order to get as close as possible to both goals, we used the following principal approach (also illustrated in Fig. A.1):

In order to be fast, we excluded techniques that would require the user to do multiple basic interactions in order to define just one brush (like a lasso, for example). We also wished that the users would not have to adjust any off-screen parameters, potentially interrupting their explorative/analyical procedure.

In order to be accurate, we decided to raise our expectations over the use of simple geometries—mostly due to their limited abilities to accurately select specific data subsets, in particular in "crowded" regions of a data visualization. Accordingly, we

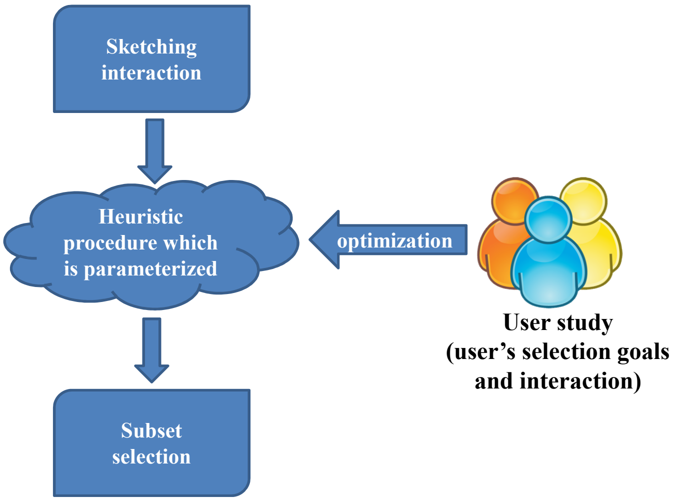

Sketching interaction
Heuristic procedure which is parameterized
optimization
User study (user's selection goals and interaction)
Subset selection

Figure A.1: Illustration of our principal approach: To be fast, we use sketching as interaction; to derive which data to actually brush, we use a heuristic with parameters that we optimize using data from a user study.

concluded that a sketching interaction, combined with a carefully modeled selection heuristic, would be the right principal approach in our case.

Typically, the heuristic, which determines the data subset to be brushed, based on a simple sketching interaction, is parameterized and different parameters will lead to different brushing results, even when the user interaction (the sketch) is exactly the same. One opportunity is, of course, to expose these parameters in the user interface, requiring the user to adjust parameters in order to achieve an expected result. Clearly, this is not, what we need. Instead, our goal was a technique, which does not require any adjustment of technique parameters by the user such that the user can concentrate on the fast and accurate interaction with the data.

In order to optimize the performance of our selection heuristic, we therefore conducted a user study with 50 participants, in which we collected information about both the brushing goals (which dataset subset did user wish to brush) as well as the associated interaction (which click-and-drag gesture would the user do to actually selected the targeted data subset). A subset of the acquired information from this user study (training data) was then used to optimize the relevant parameters of our selection heuristic.

A.4 Fast and accurate brushing in scatterplots

Figure A.2 provides an overview of the new brushing algorithm. We use a simple click-and-drag interaction for sketching the data subset to brush (click into the middle of the targeted data subset and drag the pointer to the boundary of the subset). The start- and end-point of this interaction provides us with a first hint concerning the size of the data subset, which the user wishes to brush. Similarly to the original Mahalanobis brushing technique [84], we also consider a circular data subset, centered around the start-point of the interaction, and estimate the shape and orientation of the data in this

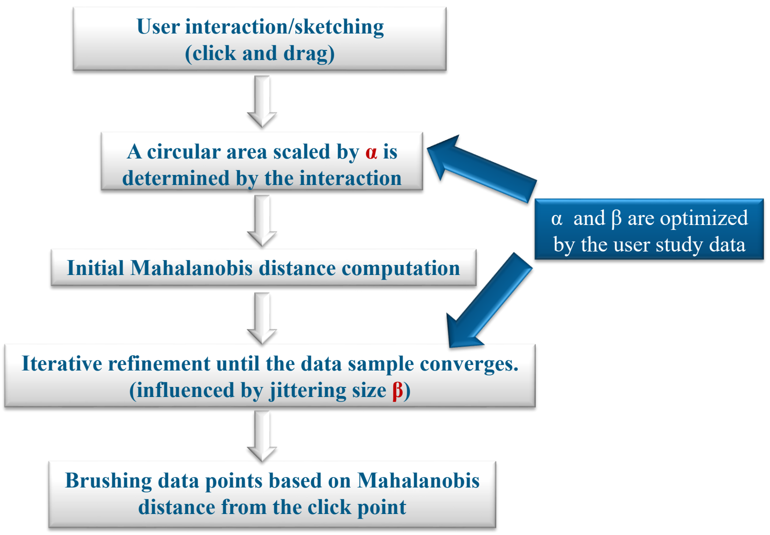

User interaction/sketching (click and drag)
A circular area scaled by α is determined by the interaction
Initial Mahalanobis distance computation
Iterative refinement until the data sample converges. (influenced by jittering size β)
Brushing data points based on Mahalanobis distance from the click point
α and β are optimized by the user study data

Figure A.2: Overview of our fast and accurate brushing technique: the user clicks into the middle of the data subset to be selected and drags the pointer to the border of the subset (sketching interaction); iteratively, a selection of points around the click-point is chosen, based on local covariance information, until convergence; a selection is made based on the Mahalanobis distance from the click-point. Two parameters, α and β, related to the sample size, before iterating, and to some jittering, stabilizing the technique, influence the performance and we optimize them using our user study.

region by looking at the local covariance information. As an improvement, we then start an iteration, until convergence, that refines this data subset selection, based on the local covariance information. After convergence, we eventually make a selection of data points, based on the Mahalanobis distance, taking the local covariance information into account. In the following, we go into more details with respect to the individual components of our solution.

Since the Mahalanobis distance, introduced by P. C. Mahalanobis in 1936[69], is central in our technique, we briefly review it here. It is based on the correlation between data variables and helps with the identification and analysis of patterns in the data. It is unit-less and scale-invariant, which is a useful way for determining the similarity of an unknown sample to a known one. It differs from the Euclidean distance in that it measures with respect to the available data. Mathematically, the Mahalanobis distance between vectors **a** and **b** is defined by

$$d_{\Sigma}(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^{\top}\Sigma^{-1}(\mathbf{a}-\mathbf{b})} \qquad \text{(A.1)}$$

where Σ is the covariance matrix of the sample (its diagonal elements consist of the variance of each variable and the off-diagonals are the mutual covariances). The location of equal Mahalanobis distances forms an ellipse around the sample mean (in 2D).

The click-and-drag interaction, which we use to sketch the data subset to be selected, provides two locations that subsequently are essential as input to our technique, i.e., the click-point $\mathbf{s} = (s_x, s_y)^\top$ and the end-point of the drag-interaction $\mathbf{e} = (e_x, e_y)^\top$.

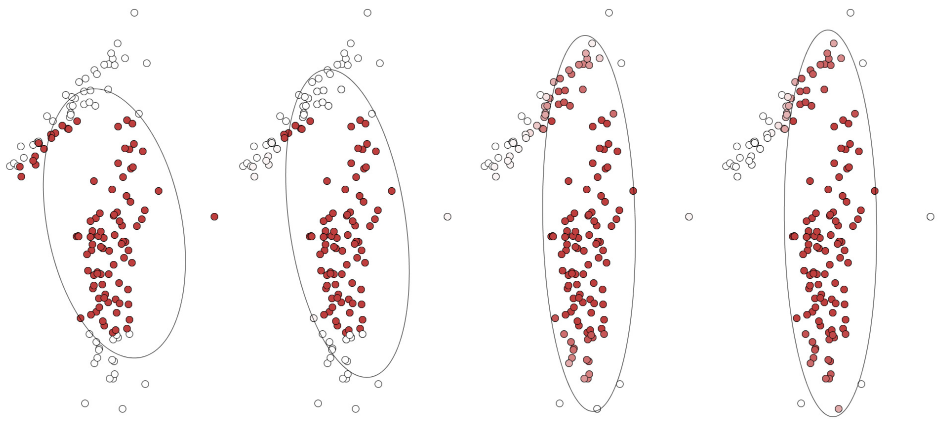


Figure A.3: Illustration of the convergence process: Initially, points in a circular neighborhood (red points on the left) are used to compute Σ_1 (illustrated by the ellipse); this leads to a new selection (red points in the 2nd panel) and the computation of Σ_2; the ellipse in panel #3 illustrates Σ_{10} and the one on the right Σ_{100} (converged). Shades of red show points which are weighted, accordingly.

A.4.1 Mahalanobis distance computation

Our technique is based on the approach to only consider the local covariance structure of a data subset around the click-point $\mathbf{s}$. It is an important part of our overall approach to determine, which data subset should be used for this computation. In our technique, we do two steps to get these sample points.

Initially, we consider a circular area with the radius $\alpha \cdot d_E(\mathbf{s}, \mathbf{e})$, where α is a weighting factor and $d_E(\mathbf{s}, \mathbf{e})$ is the Euclidean distance between $\mathbf{s}$ and $\mathbf{e}$. All points within this circle are used to compute the first instance of the local covariance information, Σ_1.

Next, we consider all points within a Mahalanobis ellipse, based on Σ_1 and sized according to $d_\Sigma(\mathbf{s}, \mathbf{e})$. This usually leads to a new data subset, which is similar but still different from the data subset as determined by the initial circle. Usually, this new subset is already a better approximation of the data subset to be brushed. To obtain an even more reasonable sample, we refine the sample iteratively by replacing them with the points in the Mahalanobis ellipse which is updated every iteration according to the samples in last iteration. Most often, we observe a quick convergence of this process. However, it can happen, that small fluctuations appear, for example, between two selections that replace each other, iteratively. Therefore, we stabilize the convergence by enabling the partial consideration of data points around the click-point, leading to a solution that is then based on a weighted covariance matrix[31].

A.4.2 Weighted covariance matrix

In a weighted sample, each vector $\mathbf{x}_i$ is assigned a weight $\omega_i \geq 0$. Without any loss of generality, we assume normalized weights:

$$\sum \omega_i = 1 \tag{A.2}$$

Then the weighted mean vector $\bar{\mathbf{x}}$ is given by

$$\bar{\mathbf{x}} = \sum \omega_i \mathbf{x}_i \tag{A.3}$$

and the elements Σ_{jk} of the weighted covariance matrix Σ are

$$\Sigma_{jk} = \frac{1}{1 - \sum_i \omega_i^2} \sum_i \omega_i (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \tag{A.4}$$

In our solution, we use an impact factor $\varepsilon_{i,0} = \varepsilon = 0.95$ that we apply to all the initial samples (as well as 0 to the remaining points). During the iteration, we update the impact factor for the points in the current Mahalanobis ellipse as follows:

$$\varepsilon_{i,n} = \varepsilon_{i,n-1} + \varepsilon^{n+1} \tag{A.5}$$

where n is the number of the iteration.

In order to obtain the weights, we normalize the impact factors:

$$\omega_i = \frac{\varepsilon_{i,n}}{\sum_j \varepsilon_{j,n}} \tag{A.6}$$

The above described mechanism achieves the following: with more iterations (growing n), the relative update of the impact factors (after normalization) decreases increasingly (the powers of ε^{n+1} drop below $1/3$ already after 20 iterations), suppressing any possible fluctuations and securing convergence at a high-quality result.

We considered to also optimize the value of ε, but found this unnecessary, because its specification could be determined by the number of iterations, which seemed to be more than sufficient to guarantee a good result (20 iterations is on the safe side). After convergence, the points with a positive impact factor are used to calculate the final, weighted covariance matrix Σ.

Figure A.3 shows the weights by using different shades of red. We can clearly see that the points which are stable in the Mahalanobis ellipse are shown in darker red.

In certain situations, the covariance matrix can be singular, also (in particular, when all sample points are along a line) and no 2D Mahalanobis distance can be computed in such a case. We thus add a small jittering (scaled according to β) to the sample points to avoid this situation. The random jitter used in our work is based on a Gaussian distribution with the mean of 0 and the standard deviation of 1 pixel.

A.4.3 Selecting a data subset using a selector

Based on the converged covariance matrix, a selector is used to determine the actually brushed data points. The selector is also based on the weighted covariance matrix

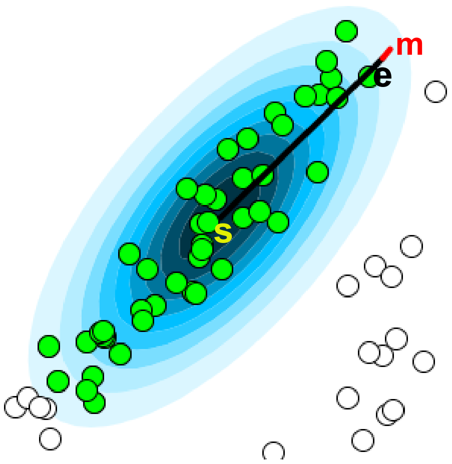


Figure A.4: Selecting data points, based on the local, weighted covariance information: **s** and **e** denote the start- and end-points of the click-and-drag interaction; the ellipses illustrate Σ_n. **m** lies on the Mahalanobis ellipse which acts as the eventual selector.

Σ_n: We use the Mahalanobis ellipse, according to Σ_n, that corresponds to point $\mathbf{m} = \mathbf{s} + \alpha(\mathbf{e} - \mathbf{s})$. Accordingly, the set of all brushed points is defined as

$$\{\, \mathbf{x}_i \,|\, d_{\Sigma_n}(\mathbf{s}, \mathbf{x}_i) \leq d_{\Sigma_n}(\mathbf{s}, \mathbf{m}) \,\} \tag{A.7}$$

Figure A.4 shows contours of the selector, selecting all green points based on **s** and **m** (all points within the Mahalanobis ellipse, which corresponds to location **m**).

A.5 User study

The new brushing technique has two not-yet-optimized parameters: α (the size of the circular area determining the initial sample, influencing also the selector) and β (jittering size). In order to achieve as accurate as possible brushing, we conducted a user study to get information about how users would use our technique to brush and what they actually wanted to select from the dataset (ground truth). Based on this information, we then did an optimization of α and β. In the following, we provide details about this user study.

A.5.1 Study datasets

In our user study, we used six representative datasets as shown in Figure A.5. In order to use as representative datasets as possible, we looked at a variety of sample data and according scagnostics. Scagnostics is short for Scatterplot Diagnostics, first mentioned by John and Paul Tukey [97] to help characterize scatterplots and find interesting structures according to density, skewness, shape, outliers, etc. Wilkinson et al. [105] revived the topic and implemented concrete measures in the R package scagnostics. We wished to choose sample datasets with mutually as different as possible scagnostics, aiming at

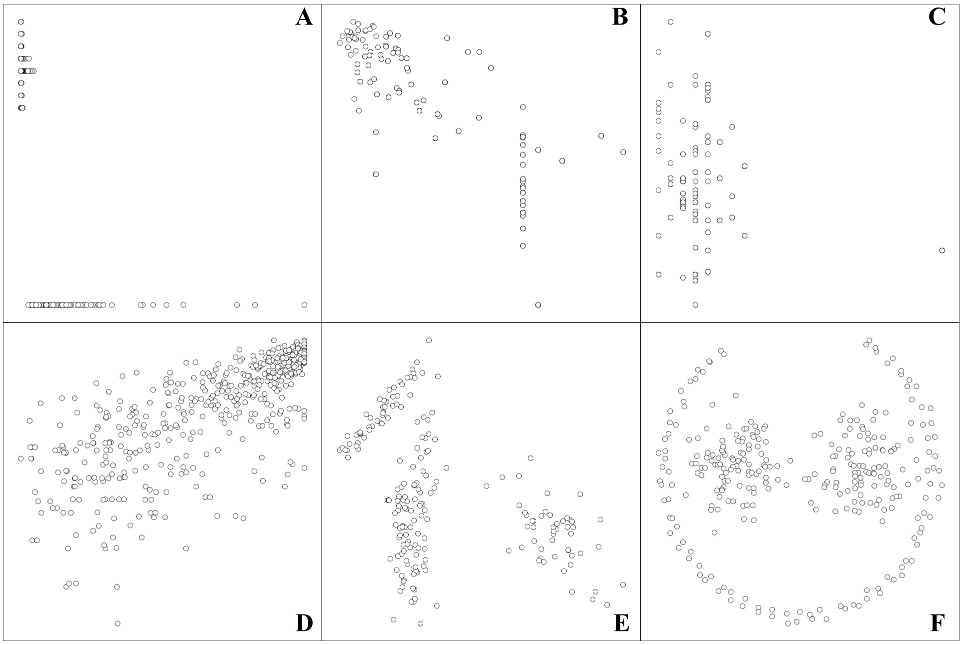


Figure A.5: Overview of the six datasets that we used in the user study: A–D show Boston housing data (with as different scagnostics as possible); E shows Gaussian clusters and F shows path-based spectral clusters (as a particuarly difficult case).

a healthy spread of scatterplots of different type. Accordingly, we chose four scatterplots (A, B, C, D) from the Boston Housing data [34], which consists of 14 variables and 91 different scatterplots, that had maximally different scagnostics from each other. We then complemented this set of four datasets with two additional ones: E shows Gauss-type clusters (standard case) and F is a path-based spectral clustering dataset (particularly difficult case due to the bent, elongated outer cluster).

A.5.2 User study process

Our user study consisted of three parts:

In the first part, all users were asked to look at a scatterplot. Then they were instructed to choose a particular data subset according to a question that was posted along with the scatterplot—we used one out of three questions in any case: choose a large cluster, choose a small cluster, and choose an elongated cluster.

Then, the users were asked to use a lasso to accurately select the points which they had choosen in the first step. This was done in order to record the user's brushing goal (later used as ground truth during the optimization).

In the last step, the users were required to use our new technique to select the same points which they had already selected in the second step. To do so, the users had to click on the center of the points to choose and then drag the pointer, while holding down the button of the mouse to the border of the points, and then release to finish the selection.

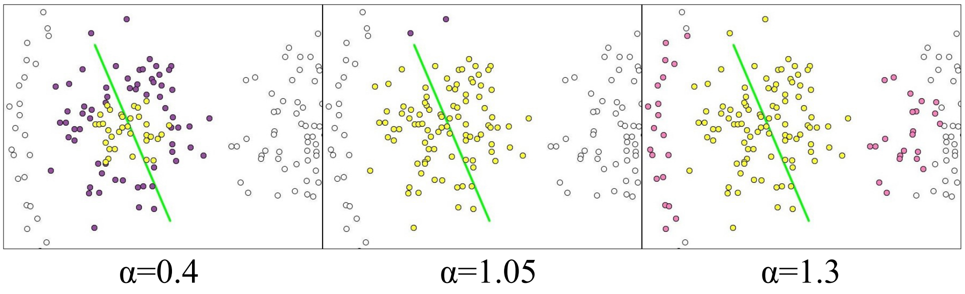


Figure A.6: Demonstration of the influence of different values of α: too small values of α will underselect, while too large values will overselect.

In the user study, we recorded all points selected by the lasso (the brushing goal), the sketching interaction (i.e., the start point and the end point of our new brushing technique), and the time spent on the interaction(think time is not included) during both of these two techniques. 50 individuals, all students or employees from the University of Bergen, Norway, participated in our study. Each one was asked to do 12 selections (six different datasets and two different questions each). We formulated two questions for each dataset in advance based on our perception of the datasets. Before the users were doing their selections, we presented our new Mahalanobis brushing in a training session, where we showed the main features by examples of brushing a test dataset. These sessions took approximately 10 minutes and the participants were free to interrupt for questions and to take over the software to experiment with the new brushing technique until they were comfortable to do the study.

A.6 Optimization

Figure A.6 demonstrates the influence of α on brushing results compared to the user goal (encoded by color). The true positives (correctly brushed), true negatives (correctly not brushed), false positives (falsely brushed) and the false negatives (falsely left out) are colored in yellow, white, pink and purple, accordingly. The green line is the diameter of the circular area determined by the user sketch. We can easily see that there are more false negatives when α is too small (left). Conversely, more false positives appear when the value of α is too big (right). Concerning β, we need a sufficiently large β (to avoid a singular covariance matrix), while also β is bound to be small: with steadily increasing values of β, the structure of the brushed data gets increasingly diluted (for really large values of β, the Mahalanobis distance basically degenerates to Eucledian distances.

In the user study we collected 600 selections, of which we randomly chose 400 as training data, leaving 200 selections for the validation. In order to compare the similarity between the selection goal by the user and the corresponding results by our technique, we used the Dice coefficient as a cost function. The Dice coefficient is a similarity measure related to the Jaccard index, developed by Lee Raymond Dice [15]. For sets X and Y, and the estimated parameters α and β, the coefficient can be defined

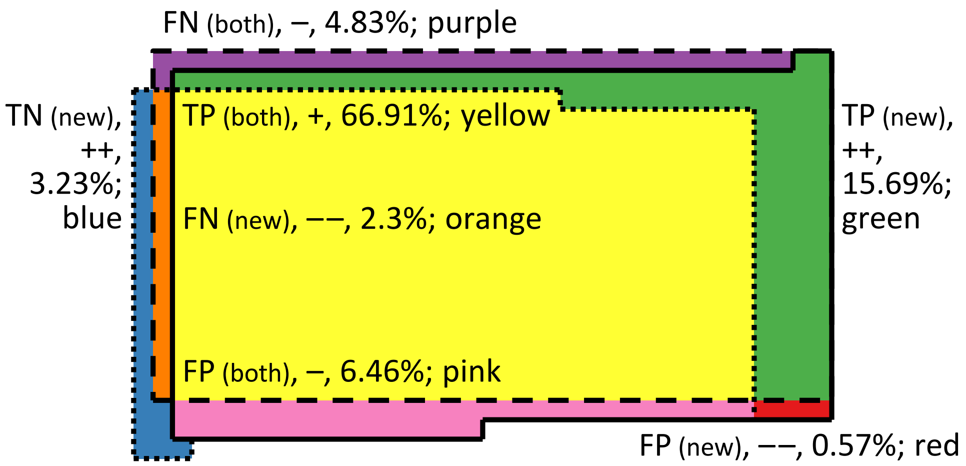

FN (both), –, 4.83%; purple
TN (new), ++, 3.23%; blue
TP (both), +, 66.91%; yellow
FN (new), ––, 2.3%; orange
FP (both), –, 6.46%; pink
TP (new), ++, 15.69%; green
FP (new), ––, 0.57%; red

Figure A.7: Statistics of the comparison between our technique and the original Mahalanobis brushing based on the user's goal (details in the text).

as

$$s(\alpha,\beta) = \frac{2\,|X \cap Y|}{|X| + |Y|}$$

where $|X|$ and $|Y|$ are the cardinalities of the two sets. In our data training, X is the result of our brushing technique and Y is the user goal. In the case of optimal agreement, i.e., X=Y, $s(\alpha,\beta)$ equals 1, while in the case, where X and Y do not overlap, $s(\alpha,\beta) = 0$.

After collecting the ground truth (lasso data) from the user study as well as the click- and release-points from the sketching interaction, we were able to conduct a numeric optimization of α and β according to the following procedure, not involving the users anymore: Based on a particular choice of α and β, we execute our selection heuristic, using the datasets from the user study and the recorded interaction data, leading to a particular X(α,β)—this was then straight-forward to compare to Y (always the same, of course), leading to $s(\alpha,\beta)$, accordingly. We started with a large matrix of systematically different combinations of the two parameters, covering a domain, which for sure was big enough. Inspecting the s-values for all these combination lead us to further examining a more detailed subset of the parameter space (basically, we refined our optimization hierarchically, doing the refinement manually). Eventually, we ended up with the following optimal values for both parameters when obtaining a highest overall accuracy of 400 training selections: $\alpha = 1.05$ and $\beta = 11$ (wrt. a view size of 800$\times$800).

A.7 Detailed discussion of accuracy

After the parameter optimization, we obtained the optimal value of α and β for our brushing technique ($\alpha = 1.05$ & $\beta = 11$). Based on this, we did a quantitative accuracy comparison with the previously published Mahalanobis brush [84] using the interaction

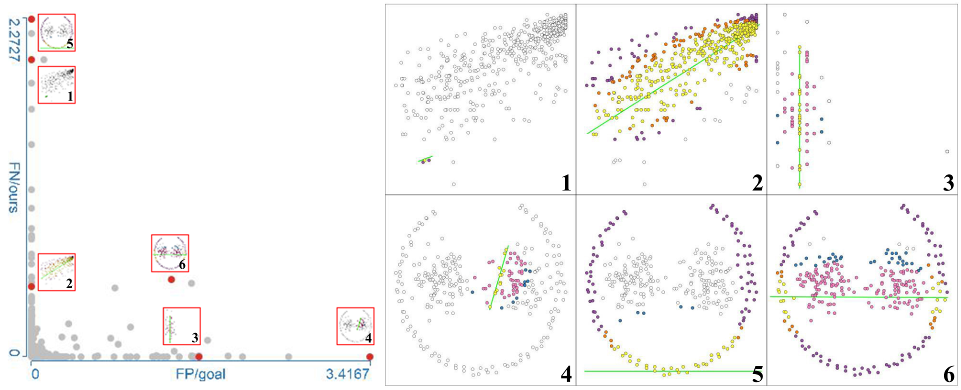


Figure A.8: Left: A visualization of how certain selected cases from the user study deviated in terms of accuracy. Right: Six (most extreme) cases of suboptimal matches between the user's goal and the new brushing technique (details in the text).

information from our user study. Figure A.7 shows a Venn-diagram-like visualization of this comparison.

The area surrounded by the dashed line represents the user goal, accumulated over all selections. The area surrounded by a solid line represents the brushing results by our technique while the dotted line surrounds the results by the old Mahalanobis technique. Same areas correspond to same numbers of brushed data points.

We calculated the percentages of how many data points fall in each of the eight possible overlap regions between the user's goal, our brushing, and the original brush after accumulating over all cases (the all-negative region corresponding to the overall context of points outside of all selections was left out from the visualization). The colors used in this visualization correspond also to the colors of points in the other scatterplots in this discussion section:

- least interesting are yellow points (both brushing technique succeed to select the point correctly (both true positive), purple points (both brushing techniques fail to select), and pink points (both techniques select falsely).

- more interesting are green points (the new technique succeeds, while the original fails), blue points (the original technique selected falsely, while the new one does not), orange points (the new technique fails to select, while the original did), and red points (the new technique selects falsely, while the original did not)—assuming the perspective of this paper, green and blue points are very good (better than the original)!

Based on the percentages as presented in Fig. A.7, we can calculate the overall accuracy for the original technique to be ≈65% and for the new technique to be ≈92% (the very positive areas, green and blue, are significantly larger than the very negative results, orange and red). Next, a few cases are discussed in detail.

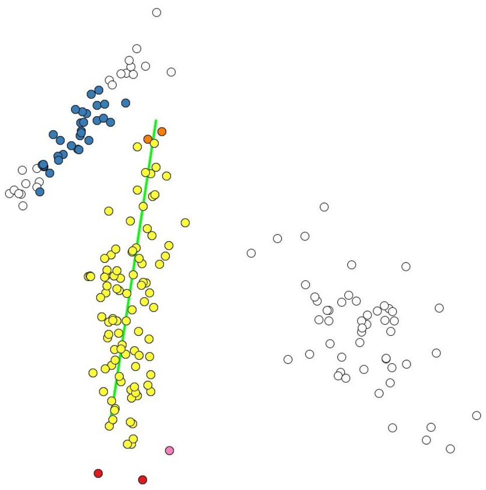


Figure A.9: An example of a good match between the user's goal and the new brushing technique.

A.7.1 Good case analysis

Figure A.9 shows a typical situation—our method performs very well, based on the weighted covariance information, but the original Mahalanobis brush results in a clearly worse selection (note the many blue points, i.e., points, which the original technique falsely selects, while the new technique does not).

A.7.2 Bad cases

Figure A.8 (left) shows a scatterplot with statistical information for each selection result (with respect to the ratio of false negatives to our brushing result and false positives to the goal). Most results lie in the bottom-left corner (the good corner), only a few results show significant numbers of false negatives / false positives. We choose six cases, highlighted by red points in the scatterplot for a detailed analysis, shown also in detail on the right in Figure A.8:

Case 1: Details of the user's interaction have a big influence when selecting very small subsets (here, the start point of the user interaction deviates a bit from the center point of the target cluster, leading to a bad performance in this case).

Case 2: Here, the new technique is too conservative and selects to few points (the old technique tends to select more circular regions).

Case 3: For scatterplots with linear structures that also are close to each other, our techniques selects wider clusters than what users seemingly wish (in this data, several users wished to select individual "lines" of data points).

Case 4: Here, we think that it is close to impossible to correctly predict the user goal computationally.

Cases 5 and 6: In both cases, the user wished to select the outer ring—something, which is by design impossible with our (linear) selection technique (the click-and-drag interaction gives too little information to correctly select such "advanced" clusters).

A.8 Discussion, conclusion, and future work

In this paper, we have made a serious attempt to contribute an improvement to a central procedure in many modern visual analytics solutions, i.e., to brushing (scatterplots). We have described and exercised an approach, which is all-too-little seen in the visualization literature, i.e., a user study based optimization of visualization parameters (here the crucial parameters of the new interaction technique). We could demonstrate, quantitatively, that we significantly improve the accuracy of Mahalanobis brushing from $\approx$65% to $\approx$92%, while still using a very fast interaction technique (click-and-drag, the average time spent is only 41% of Lasso in our user study).

After the completion of this study, we now also have a better understanding of the influence of the two optimized parameters, α and β, which consequently could lead to a further improved approach. With respect to α (scaling parameter for the selection), we clearly see the need for an optimization as we performed it, but it may be advisable to search for an improvement of the actual optimization technique in order to compensate the stochastic influence of the jittering on the results (without having worked through the complete cycle yet, we see indications that one should arrive at a very similar, optimal value of α). With respect to β (amount of jittering), we found out that β really needs to become substantially large (like 100, or so), before a measurable negative impact is clearly detectable—as long as a small, non-zero value of β avoids the singularity of the covariance matrix, the results are good.

In terms of efficiency, it only costs 20ms for the computation of brushing 2000 points, which enables the user obtain the brushing result in real time. We see the potential that this work can motivate others to follow a similar approach in their visualization research, i.e., to do an automatic optimization of visualization parameters, based on data from a corresponding user study.

Certainly, we see several opportunities for future work, including

- extending our principal approach to other views and according brushes
- further improving the selection heuristic by including kernel density estimation to even better delineate the sample for computing the selection [82, 88]

Acknowledgements

We thank the participants of our user study. We are also grateful to Krešimir Matković for valuable exchange related to the original Mahalanobis brush [84].

A

Paper B

Fast and accurate CNN-based brushing in scatterplots

Chaoran Fan and Helwig Hauser

University of Bergen, Norway

Abstract

Brushing plays a central role in most modern visual analytics solutions and effective and efficient techniques for data selection are key to establishing a successful human-computer dialogue. With this paper, we address the need for brushing techniques that are both *fast*, enabling a fluid interaction in visual data exploration and analysis, and also *accurate*, i.e., enabling the user to effectively select specific data subsets, even when their geometric delimination is non-trivial. We present a new solution for a near-perfect sketch-based brushing technique, where we exploit a convolutional neural network (CNN) for estimating the intended data selection from a fast and simple click-and-drag interaction and from the data distribution in the visualization. Our key contributions include a drastically reduced error rate—now below 3%, i.e., less than half of the so far best accuracy—and an extension to a larger variety of selected data subsets, going beyond previous limitations due to linear estimation models.

This article was published in *Computer Graphics Forum (Eurovis 2018)*, 37(3): 111–120, 2018

B.1 Introduction

Linking and brushing is useful for interactive visual data exploration and analysis in coordinated multiple views [77, 87]. Already 30 years ago, Becker and Cleveland [2] defined brushing as an interactive method for selecting data points in a visualization by drawing simple geometries onto it. A key functionality in coordinated multiple views is that brushing leads to a consistent highlighting of the selected data in all linked views. This results in the most common form of focus+context visualization [35], enabling the fast and effective exploration of data relations, which are too challenging to show in just one view. Many techniques for brushing have been developed and variants can be categorized into:

- *brushing using simple geometries*—the most commonly used brushing solutions include the rectangular or circular brushing on scatterplots, line-brushing on data graphs [56], etc.
- *lassoing*—the user selects subsets by drawing a geometrically detailed lasso around the target group of item representations
- *logical combinations of simple brushes*—the user makes use of multiple brushes and combines them using logical operators to refine the data selection [17, 72]
- *sketch-based brushing*—the user sketches a shape onto a visualization and a selection heuristic is used to determine which data are selected [22, 76, 84]

For designing a brushing technique, two particularly important criteria should be taken into account:

- *efficiency*—is the brushing is fast enough (including the interaction and all computation) to enable a fluid data exploration/analysis [20, 98]?
- *accuracy*—does the brushing interaction lead to a selection of exactly the data subset, which the user wished to select in the view (we refer to the most common form of brushing, where data points are selected due to their location in the visualization)?

Despite the rich variation of existing brushing tools, we rarely see a solution that combines both criteria really well: Many brushing techniques are indeed fast, as clicking on one point, for example, or drawing simple geometries—also sketched brushes are fast, requiring only a simple gesture as interaction and thus enabling a swift user–computer dialogue during the exploration/analysis [8]. A common disadvantage of fast techniques, however, is that it can be difficult to accurately brush a particular data subset.

On the other hand, we certainly find brushing techniques, that are straight-forward for accurately selecting subsets of interest, such as lassoing and the logical combination of simple brushes. This benefit, however, comes at the price of being slower—specifying a lasso, for example, easily becomes a unit task by itself [8], potentially interrupting the exploration/analysis process. In our work, we aim to integrate both criteria in one technique as good as possible.

Recently, deep learning methods, especially convolutional neural networks (CNN), have been used very successfully in a wide range of fields including natural language processing [49, 94], object detection [86] and image classification [59]. As brushing is mainly used to select spatially coherent data subsets, which is related to detecting patterns in images, we see the potential of exploiting deep learning to improve brushing even further.

Inspired by the impressive performance of CNNs in image processing, we developed a new CNN-based technique for brushing in scatterplots (we chose brushing in scatterplots as our study case, assuming that this approach is extensible to other views and according brushes, as well). Our quantitative evaluation shows that we reduce the brushing error rate from about 8% (Mahalanobis brush [22]) to about 2.5%. We also build on a fast and simple click-and-drag interaction, but provide a solution which is much faster and also more flexible in terms of which data subsets can be selected (not limited due to a linear model). Since brushing is central in most modern visual analytics systems, we see this result as potentially very relevant.

This paper is organized as follows: After reviewing related work (Sect. B.2), we first describe our principal approach (Sect. B.3), before we then present our technique in detail (Sect. B.4). The network training and evaluation are presented in sections B.5 and B.6, before we present details about our user studies (Sect. B.7) and the model of natural variation among click-and-drag sketches for brushing (Sect. B.8). We conclude and address future work in section B.9.

B.2 Related work

In the following, we first review some critical works concerning brushing for visual analytics, before we then discuss related work concerning applications of convolutional neural network.

B.2.1 Brushing techniques

Many variations of brushing have been proposed, each with its own strengths and weaknesses—for example, in terms of their ease of use and the degree of control that the user has. Brushing is intrinsically based on the interaction between the user and the system, often a combination of mouse/cursor motions and clicks. Less usual methods, based on eye/head tracking, for example, or gestures in a virtual reality environment, have also been proposed [107].

Brushing in scatterplots is often based on the use of simple geometric shapes such as a rectangle or circle. Alternatively, users can use a lasso to specify the selection more accurately. Several extensions to simple brushing have been published, including techniques to formulate more complex brushes by combining multiple brushes using logical operators. Martin and Ward [72], for example, enable the user to configure composite brushes by applying logical combinations of brushes, including unions, intersections, negations, and exclusive or operations.

Koytek et al. [57] created MyBrush, which extended the popular brushing and linking technique by incorporating personal agency. It offers users the flexibility to configure the source, link, and target of multiple brushes. Hurter et al. [46] developed a semantic lens which selects a specific spatial and attribute-related data range and it is applicable for scenarios requiring a mixed selection of the zones of interest.

Similarity brushing [76, 80] is a typical example of sketch-based brushing, which is based on a fast and simple sketching interaction—the user uses a swift and approximate gesture (for example, drawing an approximate shape that the data should follow) and then a similarity measure (target function) is defined to identify, which data items actually are brushed. This way, the interaction is fast, but likely not 100% accurate.

Recently, the Mahalanobis brush was presented as an interesting alternative for brushing scatterplots [84]. The user simply clicks into the center of a coherent data subset to be selected. The link between the interaction and the actual selection is realized on the basis of an analysis of the underlying data (a local covariance matrix indicates the overall shape and orientation of the data to be brushed, forming then the basis for a local Mahalanobis metric, which is then used as a distance measure to select the data).

While this technique is giving quite good results, it still has limitations, including a non-optimized selection of the local context for the Mahalanobis computation and one off-screen parameter for the brush size. Fan and Hauser [22] extended the Mahalanobis brush and improved the accuracy by optimizing the parameters based on a user study and getting rid of the off-line parameter. However, this improved solution is still linear and has difficulties with complex structures that would require a more flexible approach. Also, it is not really real-time for large datasets.

B.2.2 CNNs and visualization

A convolutional neural network (CNN) is a deep learning architecture, which is inspired by the connectivity pattern between neurons and their organization in the visual cortex [45]. The concept of a neocognitron, proposed by Fukushima [28], is widely considered a fundamental basis of modern CNNs. LeCun et al. [61, 62] established the framework of CNNs by developing a multi-layer artificial neural network called LeNet-5, which was applied successfully to image classification problems. With the emergence of big data and the development of computing infrastructure, the structure of some CNNs has become very deep. A solution by Krizhevsky et al. [59] was able to classify about 1.2 million images into 1000 classes, i.e., a record-breaking result in the ImageNet Large Scale Visual Recognition Challenge. Often, the impressive success of image processing CNNs is attributed to their ability to learn rich mid-level image patterns as opposed to hand-designed low-level features used in more traditional methods.

Considering an increasing number of successful applications of CNNs in many fields, we would expect according approaches also in visualization. While several interesting works look into the opportunity of visualization helping with the design, training, and analysis of CNNs [109], we do not yet see a mentionable number of visualization solutions that exploit CNNs, in particular not in interaction techniques in visualization. With our work, we also hope to inspire interesting new research in this direction.

B.3 The principal approach

The overall goal of our research was to devise a brushing technique, which is both fast and accurate. In order to get as close as possible to both requirements, we used the

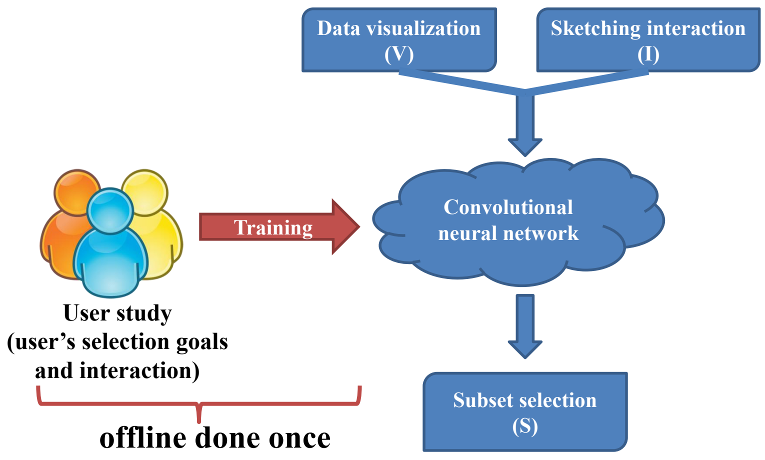

Data visualization (V)
Sketching interaction (I)
Training
Convolutional neural network
User study (user's selection goals and interaction)
offline done once
Subset selection (S)

Figure B.1: Illustration of our principal approach: To be fast, we use sketching as interaction; to estimate which data to actually brush, we use a CNN trained with data from two user studies.

following approach (also illustrated in Figure B.1):

In order to achieve a fast interaction and a fluid exploration, we excluded any technique that would require the user to do multiple basic interactions in order to define just one brush (like a lasso, for example). To be as accurate as possible, we had to go beyond simple geometries with their limited abilities to accurately select data subsets, in particular in "crowded" regions of a visualization. Therefore, we needed a computational link between the fast and simple interaction and the selection of a non-trivially delimited subset, estimating the visual structure that the user identified as the brushing target. Usually, users brush subsets, which are spatially coherent in the visualization. Thus, we assume that we can estimate the brushing goal from both the actual brushing interaction and the data distribution in the visualization near the interaction.

Following successful previous work [22, 76, 84], we deemed the combination of an basic sketching interaction I, indicating the location, size, and orientation of the subset to brush, with a computational estimation function S, determining which subset to actually select, based on its visualization V near sketch I, to be a useful framework for modeling our solution. In previous work [22, 76, 84], the estimation function S was carefully modeled according to meaningful heuristics, based, for example, on a geometric similarity function in visualization space.

Since S amounts to interpreting the data visualization in terms of which spatially coherent subset best possibly relates to the sketching interaction, we found it promising to exploit recent successes of deep learning in image processing for our solution. We expected that the increased flexibility of this approach also helps to overcome limitations in previous work with respect to the variety of shapes that such an interaction can address—all solutions S for sketch-based brushing of scatterplots, so far [22, 84], are limited to brushing structures that are described by linear models.

We also wished that the users would not have to adjust any off-screen parameters, interrupting their exploration/analysis (such that they can benefit from a fluid interaction with the data). Thus, we constructed our solution around a convolutional neural

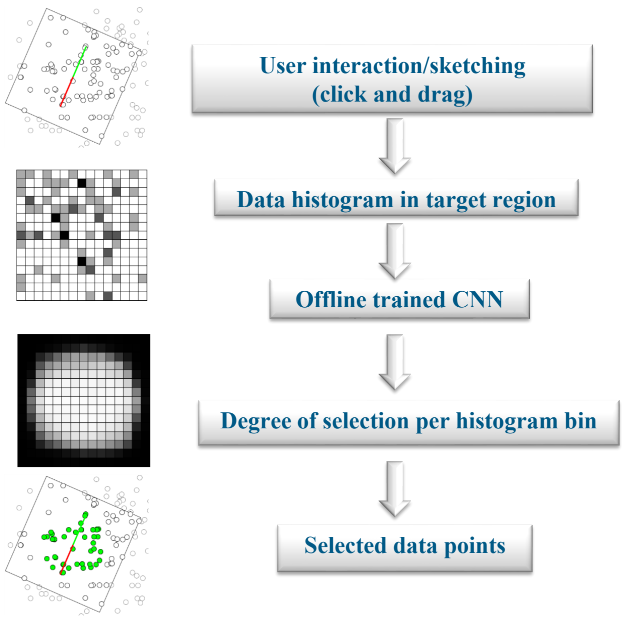

User interaction/sketching (click and drag)
Data histogram in target region
Offline trained CNN
Degree of selection per histogram bin
Selected data points

Figure B.2: Overview of our fast and accurate brushing technique: For sketching, the user clicks into the middle of the data subset to be selected and drags the pointer to the border of the subset; The CNN then sees the data distribution near the interaction as a 2D histogram. It delivers a degree-of-selection value per histogram bin, from which we can compute, which data subset is selected.

network (CNN) that we trained with data from two user studies.

The first user study, presented in more detail in another recent publication [22], provided information about both the brushing goals (which dataset subset did the users wish to brush) and the according interaction (which gesture would the user do to actually select the targeted data subset). In the second user study, presented further back in this paper, we examined the variation information of the user's interaction in order to use this for modeling an extension of the training data for the CNN.

B.4 The new brushing technique

Figure B.2 provides an overview of our new brushing algorithm. In the following, we first describe the overall construction of our solution, before we then describe the individual components in detail.

B.4.1 Technique at large

Since we aim at estimating the selection information S from both the input sketch I as well as from the data visualization V, we need to efficiently and effectively consider these two heterogeneous parts of input information. For mainly two reasons, we handle I and V individually, using the CNN only for the interpretation of V. The critical input from the click-and-drag sketch, i.e., the click point c (center of the interaction) as well as the length r and the angle ϕ of the drag component, is first used to locate, scale, and orient the receptive field of the CNN. This way, we "normalize" the network's operation with respect to I by a simple linear transformation such that we can easily "undo" this normalization after the network's estimation process. Accordingly, the network's task is then to interpret the 2D data distribution in the appropriately located, scaled, and rotated region of the visualization. In order to predict which data subset to select, we model this step as an image processing operation: on the input side, we let the network see a 2D histogram of the data in the targeted area; on the output side we expect a measure p per bin of the histogram, indicating a "degree of selection" such that a simple thresholding at $\overline{p} = 0.5$ can identify the region within the target area corresponding to the selected data subset.

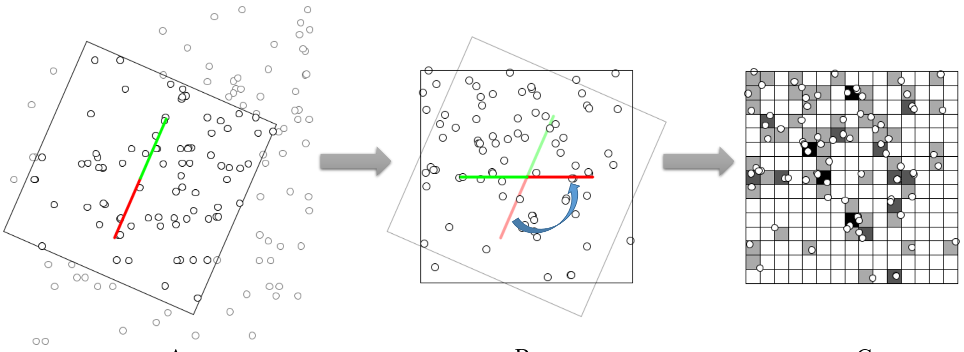


Figure B.3: Computing the input to the CNN: A, a square area is specified by the interaction (red line segment); B, after rotating to the horizontal; C, histogram of the local data distribution (CNN input).

B.4.2 Computing the input to the CNN

The interaction I by the user amounts to a line in the scatterplot. To focus on the related, local visualization V near I, we use a square area including the user's brushing goal, with side length $2 \cdot r \cdot \omega$ and r being the length of I. To choose ω, we balance two goals: On the one hand, we need to make the receptive field of the network large enough to indeed see the data subset, which is targeted by the user as the brushing goal. On the other hand, this area should be not bigger than necessary in order to optimize the CNN results with respect to the resolution of the input histogram. Examining the user study data [22], we found $\omega = 1.5$ to be a useful compromise. Figure B.3A shows the square area of the local visualization V related to I (red line segment).

In order to let the CNN see a normalized input (independent of I) as well as to interpret its output efficiently, the square area is rotated by $-\phi$ into a horizontal orientation as shown in Figure B.3B. To make use of the local data distribution, we divide the square into a grid with a specific resolution (15 by 15 in our experiments) and compute a histogram by counting how many data points show up in each bin of the grid, denoted by C_{ij} where $i, j \in [1, 15]$. We then normalize the value of each bin into $[0, 1]$

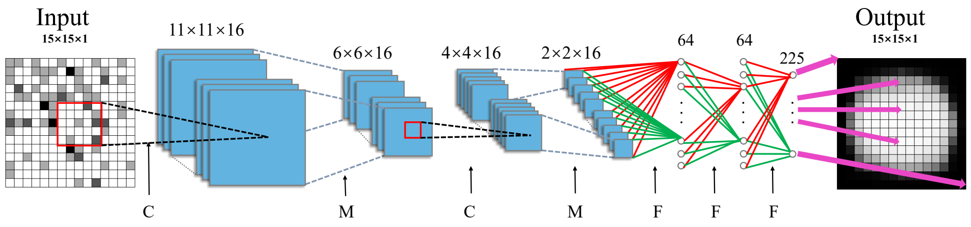


Figure B.4: The proposed CNN model. C, M and F represent the convolutional layers, max-pooling layers, and fully connected layers, respectively. The purple arrows from the last layer illustrate the association between the final layer's outputs and histogram-aligned grid of degree-of-selection values.

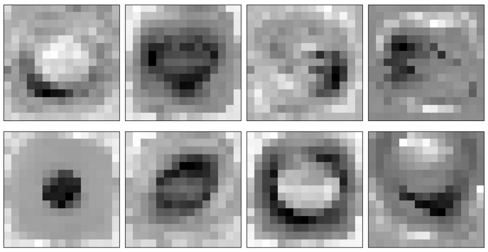


Figure B.5: Visualization of the weights of 8 selected neurons among 64 in the second fully connected layer of the CNN.

by $C_{ij}/max(C_{ij})$. Figure B.3C shows a visualization of the network input with darker bin colors representing larger C_{ij} values.

B.4.3 CNN design

A typical image processing CNN is composed of convolutional, pooling and dense layers [61]. The purpose of convolutional layers is to extract patterns in local regions of the input images. Pooling layers are also referred to as a downsampling layers, with maxpooling being the most popular choice. This serves two main purposes. First, the number of parameters gets limited, reducing both the computation cost and overfitting. Second, the network can this way "see" on different scales, including also larger structures (earlier layers usually see smaller structures with subsequent layers then focussing on larger patterns). Fully connected layers then connect every neuron in one layer to every neuron in the next layer. This is typically used in the last stages of the CNN.

For CNN design, the two most important goals are to avoid both overfitting and underfitting. Overfitting refers to when a model is overly tuned to the training data so that it does not generalize well. And if the model is too simple, with too few parameters, then this leads to underfitting, i.e., bad results (low accuracy, etc.).

We carefully experimented with many different layouts/settings of the CNN model, varying the size and number of the convolution filters, the number of convolutional and fully-connected layers, and the number of neurons in the fully-connected layers. As a result, we found a model which fits our scenario well. In our design, we deviate from the conventional CNN layout by replacing the last layer (classifier) with a structured regression layer to encode the output information from which the actual data subset selection can be derived in a subsequent step.

Altogether, we propose a model with two convolutional (C), two max-pooling (M), and two fully-connected layers (F). Figure B.4 shows this architecture and the association between the last layer and the histogram-aligned grid of p-values. In detail, our model is configured as Input(15×15×1), C(11×11×16), M(6×6×16), C(4×4×16), M(2×2×16), F(64), F(64), and F(225). The sizes of the C and M layers are defined as width×height×depth, where width×height determines the extent of each feature map and depth represents the number of maps (filters).

The activation function of the last F layer (regression) is chosen to be a sigmoid function so that the output values are from $[0, 1]$. To reduce the likelihood of vanishing gradients, ReLu is used for all the other F and C layers. The filter size is chosen to be 5×5 for the first C layer and 3×3 for the second one. The max pooling layer uses a window of size 2×2 with a stride of 2 in each direction.

In order to check that we have chosen a reasonable number of parameters and a useful structure, avoid overfitting as well as underfitting, we have also visualized the weights of the neurons in the second fully connected layer, the output of which are used to compose the overall results. The weights are useful to visualize, because well-trained networks usually display nice and smooth filters without any noisy patterns [109]. Noisy patterns can be an indicator of a network that has not been trained for long enough, or possibly a very low regularization strength that may have led to overfitting. Figure B.5 shows a weights visualization of 8 selected neurons among 64 in the second fully connected layer, showing that our model learns meaningful structures and patterns.

B.4.4 Interpreting the output of the CNN

The output of the CNN is a grid comprised of degree-of-selection values p per histogram bin and we threshold this information at $\overline{p} = 0.5$ to locate the selected data subset (Fig. B.6). We use the Marching Squares algorithm [68] with a threshold of 0.5 to generate the selection contour based on the two-dimensional network output and all points in the selection contour are selected to be the brushing result. Instead, one could also use the p values directly and select all data points that fall into a bin with $p > 0.5$. Since this would correspond to an unnatural selection contour, we prefer the smoother results as provided by the Marching Squares.

B.5 Training the CNN

We define the training data as $(x_i \in T_{in}, y_i \in T_{out})_{i=1}^{N}$ with pairs of input and expected output (N is the number of the training samples). We optimize the parameters of the network based on the training data using the mean-squared error as a loss function. The method of computing the network input x_i (appropriately located, scaled, and rotated histograms) has been described in section B.4.2. In the following, we explain the design of the reference output y_i, which the model is trained against, and the implementation of the CNN.

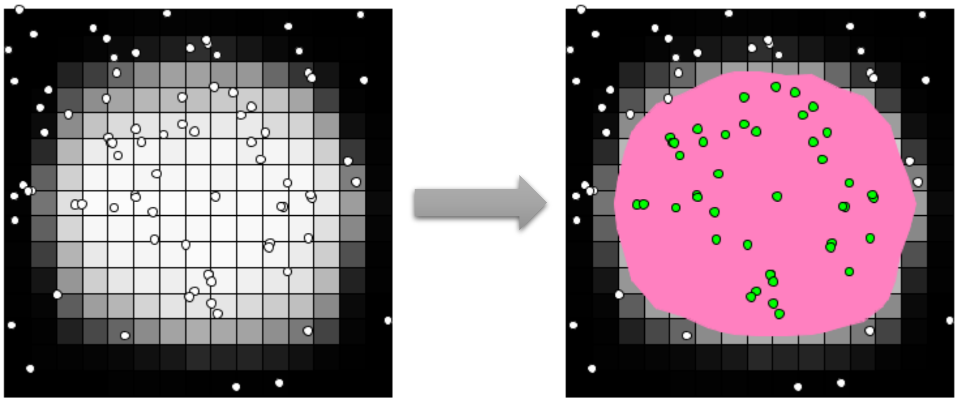


Figure B.6: Left: output of the CNN. Right: the points colored in green are the brushing result (inside the Marching Squares contour, surrounding the pink area).

B.5.1 Computing the reference output

For training the CNN, the information of the user goal (data items to select), which we have from the first user study, needs to be given to the training in an appropriate form that is compatible with the output layer of the CNN. For the specific square area that we consider for x_i, we know which points are the user's goal from the user study. On the left side of Figure B.7, for example, the yellow points are the user's brushing goal and the red points are not to be selected.

To extract the information that is needed for the CNN training, we convert this binary select-vs.-disregard information from the user study into an image of the same resolution as the CNN input, using an adapted form of the K-nearest neighbors algorithm [1]. For each bin of the grid, we estimate the degree-of-selection value p, that we then want the network to learn, by considering all data points in the bin and, if needed, nearby points. Further points are included from near to far, if there are less than K points in the center bin, stopping when at least K points are found. We give less weight to more distant points, based on the Euclidean distance d_k between the bin, where the point is located, and the center bin. We then estimate the degree-of-selection by

$$p = \frac{\sum_k i_k/(1+d_k)}{\sum_k 1/(1+d_k)} \tag{B.1}$$

where $p \in [0,1]$, k being the index of the points in the search area, and $i_k = 1$ if the point is a user goal, otherwise $i_k = 0$.

In general, the user goal is confined to the inside of the target area. Thus we ensure that the border bins have small values. Accordingly, we stop the search procedure, when an outside bin of the grid is searched. If the number of points found so far is then less than K, we synthesize the missing points right outside of the grid, and these points are labeled as not being a user goal.

A visualization of one according reference output is on the right of Figure B.7: the darker a bin is colored, the smaller the corresponding p value is. Even though also other methods for estimating p come to mind, we found that this simple approach gave

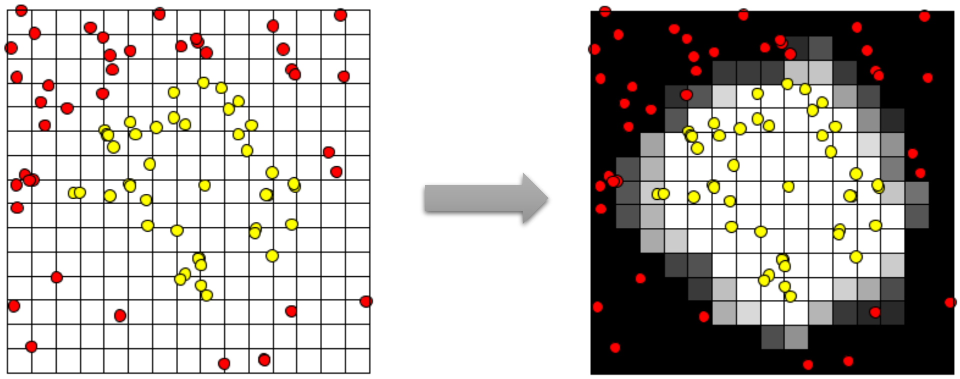


Figure B.7: Left: binary select (yellow) vs. disregard (red) information from the user study. Right: the image-based reference output computed with an adapted nearest neighbor algorithm.

very good results—most likely due to well-behaved data from the user study (all of the user goals from the study lead to selection geometries with substantially smooth boundaries). In our research, we experimented with different values of K and validated them against the user study data—we found that $K = 3$ achieved the smoothest boundaries that successfully separate the user goal.

B.5.2 Training details

Usually, high accuracy cannot be achieved unless enough training samples are provided. It is labor-intensive and time-consuming to invite a large number of users to provide large amounts of user data. Instead, we follow a common strategy and synthesize additional training data [59] from the already acquired training set based on a second user study that we did with the goal to study the variation of the user's interaction (in our sketch-based brushing context). We assumed that there would be a certain amount of natural variation in the users' sketching interaction (in terms of where exactly they click and how far and in which direction they drag). In the user study, we thus measured this variation, and modeled it in a statistically best-possible way, and then synthesized additional interaction sketches according to the resulting models as additional training data for the CNN. On the left of Figure B.8, we see several user interactions from the second user study that we used for the variation analysis, and on the right are correspondingly modeled synthetic variations, confirming that our model leads to meaningful new training data. More details about this user study and the modeling procedure are provided in sections B.7 and B.8, respectively.

We then used three datasets of different sizes to train the CNN and compared their performances. The smallest dataset consists of 500 (x_i, y_i)-pairs, based on 500 selections from the first user study. The larger and the largest training sets were generated by synthesizing additional 1500 and 7500 (x_i, y_i)-pairs, respectively.

We implemented the network and executed its training in Keras [11] which provides useful GPU acceleration. For the training and testing, we used a PC with an Intel Xeon E5-1650 CPU and an NVIDIA GeForce GTX 1080 GPU. We used regularizers in the

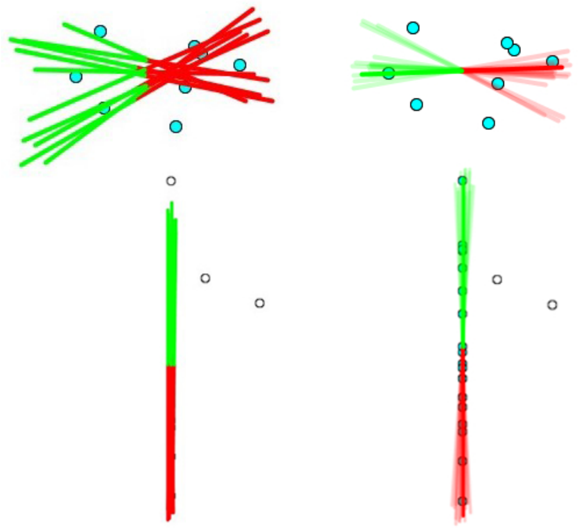


Figure B.8: Left: Two cases from the second user study, 10 interactions for a specific brushing target in each case. Right: 15 modeled interactions per case (semi-transparent) with the original user interaction shown as solid line.

convolutional layers and a dropout function with a drop rate of 0.2 to avoid overfitting. As the output of our model is related to a degree-of-selection instead of a binary matrix, an L2 regularizer is more suitable in our model, resulting in less sparse output as when using an L1 regularizer. The learning rate was set to be 10^{-3} and in order to obtain a good convergence towards a high-quality optimum, we ran 10000 epochs for the training with a full batch.

Table B.1: Statistics of the cross validation and training times

Group (validation)	Size factor of training data		
	1	4	16
G_1	95.73%	96.33%	**96.65%**
G_2	96.43%	**97.74%**	97.72%
G_3	98.39%	98.50%	**98.60%**
G_4	97.11%	97.13%	**97.24%**
G_5	95.00%	96.02%	**96.82%**
G_6	95.37%	97.17%	**97.46%**
Mean	96.34%	97.15%	**97.42%**
Variance	$1.3 \cdot 10^{-4}$	$6.9 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$
Time	≈**5mins**	≈20mins	≈80mins

B.6 Evaluation

For evaluating the new method, and in particular the trained CNN, we used k-fold cross-validation [54]. In k-fold cross-validation, the original sample is randomly partitioned into k equal sized sets. In each of the k folds, a single set is retained as the validation data for testing the model, and the remaining $k-1$ sets are used as training data. The cross-validation process is then repeated k times, with each of the k sets used exactly once for validation. The results from all k folds are then averaged to assess the model. In our evaluation, we set $k = 6$ and split the original 600 selections (from the first user study) into six evenly sized groups G_i. For training the network, we use five groups as such (500 selections), or the extended training sets (see section B.5.2) with 2000 or 8000 selections, respectively.

Table B.1 shows the results of the cross validation in terms of accuracy for the three training set sizes. With the extended training data, the trained model is more stable and has a higher accuracy. We also see that the performance of our model, when using the 16 times larger training set, is only slightly better as when trained with the four times bigger training set, with an overall accuracy 97.42%.

Based on the trained CNN model, we did a quantitative accuracy comparison with the previously published new Mahalanobis brush [22] using the same 600 selections as presented in their user study. For each point in each of the cases, we test whether the Mahalanobis brush selects it, whether our new technique selects it, and whether it should be selected (ground truth), looking at 252,400 points altogether. We use different colors in the visualization to represent the comparison result:

- yellow points (both brushing techniques succeed to select the point correctly; both true positive), purple points (both brushing techniques fail to select, i.e., both false negative), and pink points (both techniques select falsely; both false positive)—purple and pink points (both techniques fail) amount to about 4.57% of all cases, where at least one technique fails.
- green points (the new technique succeeds, while the old fails) and blue points (the original method selected falsely, while the new one does not)—these points represent the cases, where our new technique improves the so far best results and ≈89.3% of all cases, where at least one method fails, fall into this category!
- orange points (the new method fails to select, while the old did) and red points (the new method selects falsely, while the old did not)—these points represent cases, where the new technique is worse than the one (only about 6.13% of all cases, where at least one method fails, amount to this category).

To make this color-coding easier to follow, an accordingly colored Venn diagram is embedded in Figures B.9 and B.10 as color legend. The dashed circle on top represents the user goal (ground truth). The solid circle represents the brushing results by our new technique, while the dotted one surrounds the brushing results by the previously published Mahalanobis technique. We note that in the shown schematic, the areas do not correspond to the proportions of the respective cases—it's just a color legend.

In total, when using the dice coefficient [15] to assess how well both techniques agree with the ground truth, we get excellent 99% for our new technique, as compared to 91% for the reference method [22]. In terms of efficiency, the new technique is

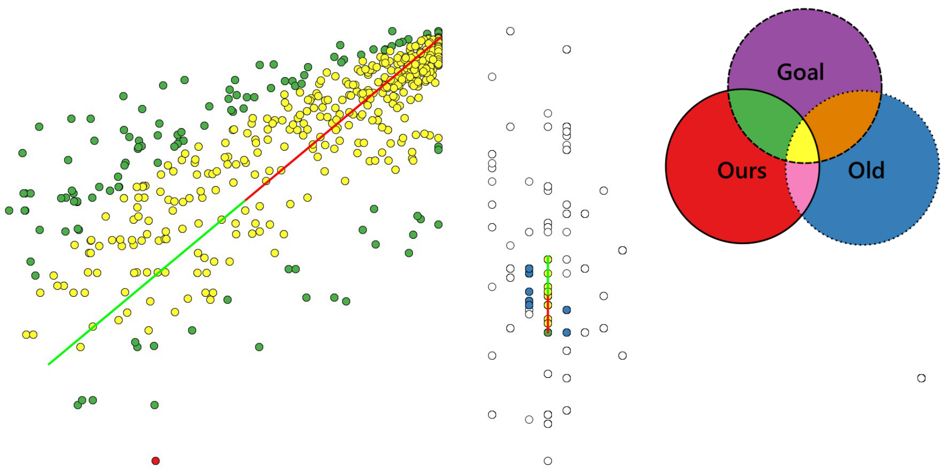


Figure B.9: Two typical examples of a good match between the user's goal and the CNN-based brushing technique.

similarly fast as the previous Mahalanobis brush for small subsets; when brushing 2000 points, for example, it takes around 20ms. But when it comes to larger datasets, our method takes only 180ms when brushing 1 million points, while the Mahalanobis brush takes very long 110s for 100000 points, which is orders of magnitude too slow for a fluid interaction with large data.

To further substantiate the evaluation of our new approach, we organized a new user study and invited ten users to test our CNN model on new data. In this study, we followed the established procedure of the first user study [22], but provided 6 completely new datasets (D7–D12 in the supplementary video) for the users to brush, which were not used in any way in the construction or training of our model. We got 120 new selections from this user study and the average accuracy is ≈95.3%, providing further evidence that our model is good at capturing relevant features of the user's brushing preference, rather than being biased by the training data.

B.6.1 Examples of good cases

Figure B.9 shows two typical situations, when our method performs very well, while the Mahalanobis brush [22] results in a worse selection. On the left of Figure B.9 we see many green points which the Mahalanobis brush would need to select, but actually does not, while our technique selects them correctly. Besides, we see many blue points on the right of Figure B.9: by design, the Mahalanobis brush brushes an elliptical area that more often than the CNN has troubles in selecting elongated, skinny groups.

B.6.2 Worst cases

We also did a worst case analysis, shown in Figure B.10, which we selected based on the ratio of false negatives to our brushing result (FN/ours) and false positives to the

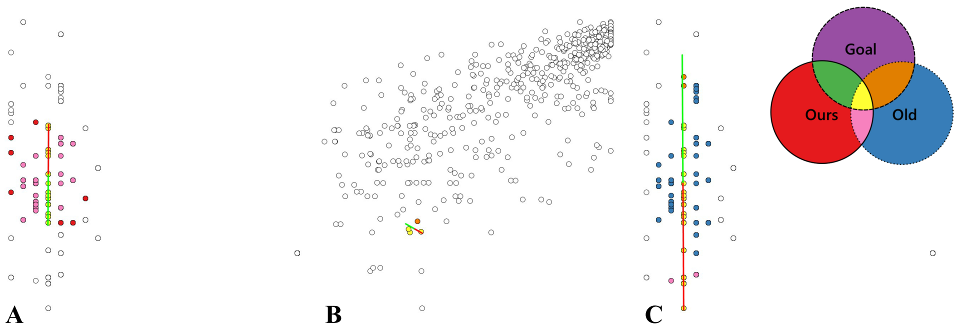


Figure B.10: Three (most extreme) cases of suboptimal matches between the user's goal and the new brushing technique.

goal (FP/goal). Comparably large values in either of these measures identify our worst cases.

Case A, highest value of FN/ours: our technique has a problem to differentiate whether the user's goal is a circular region or an elongated group in some very similar regions.

Case B, highest value of FP/goal: the user's interaction has a big influence when selecting very small subsets (here, the start point of the user interaction deviates a bit from the center point of the target cluster, leading to a bad performance in this case).

Case C: Actually, this is not a very bad example, but we chose to show it here, because it performs relatively badly both in FP/goal and FN/ours (a worst case that isn't really bad, after all).

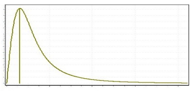
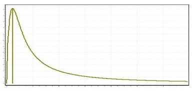
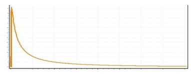
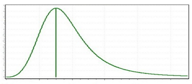
B.7 User studies

As the user plays a central role in brushing, we conducted a new user study to explore how the user uses our brushing tool in practice and analyzed the natural variation of the sketching interaction in order to prepare the synthesis of additional data for the training of the CNN. This new user study can be seen as a follow-up user study, based on the user study done earlier [22] (called the first user study in this paper). In the following, we first describe, what data we used from the earlier published user study, before then going into details about our new user study.

The previously published user study (50 participants) provided the following data which we also used for this work: Given a particular scatterplot (one out of six) and a particular request (one out of three), the study participant chose a target data subset to select (ground truth, reported by the participants using a lasso tool) and then also provided the corresponding click-and-drag interaction, which this participant would use to select the target group.

Naturally, for each user goal, the interaction done by the user will be at least a bit different every time. In order to understand the natural variation of the user's sketching interaction, when having the same brushing operation in mind, and to model this variation to enable the synthesis of additional data for the training of the CNN, we did this follow-up user study based on the first user study. In the new study, 10 individuals, all

Table B.2: Best-fit modeling of the observed variation in the executed sketches, when intending to brush the same data subset

Variance	Cases	Best-fit Probability Density Function	
$\frac{var(r)}{mean(r)}$	all	Burr	
		mode=0.30682	
		$k = 1.0164$	
		$\alpha = 1.9863$	
		$\beta = 0.53886$	$f(x) = \frac{\alpha k(\frac{x}{\beta})^{\alpha-1}}{\beta(1+(\frac{x}{\beta})^{\alpha})^{k+1}}$
$var(\phi)$	S∪B	Burr	
		mode=0.00112	
		$k = 0.31523$	
		$\alpha = 1.6348$	
		$\beta = 0.00191$	$f(x) = \frac{\alpha k(\frac{x}{\beta})^{\alpha-1}}{\beta(1+(\frac{x}{\beta})^{\alpha})^{k+1}}$
$var(\phi)$	E	Lognormal	
		mode=3.211E-5	
		$\sigma = 1.8456$	
		$\mu = -6.9401$	$f(x) = \frac{exp(-\frac{1}{2}(\frac{lnx-\mu}{\sigma})^2)}{x\sigma\sqrt{2\pi}}$
$var(c)$	all	Log-logistic	
		mode=-0.0241	
		$\alpha = 3.7167$	
		$\beta = 0.08711$	
		$\gamma = -0.09919$	$f(x) = \frac{(\alpha/\beta)(\frac{x-\gamma}{\beta})^{\alpha-1}}{(1+(\frac{x-\gamma}{\beta})^{\alpha})^2}$

students or employees from the Delft University of Technology, participated. We used 100 representative selections from the 600 selections in the first user study. For each user, 10 different selections were displayed only showing the user goal information.

The second user study then consisted of two parts: In the first part, the users were asked to look at the points, which they should brush, using the user goal information from the first study for each case. Then, the users were required to use our new technique to select the target points. To do so, the users had to click into the center of the target points and then drag the pointer, while holding down the button of the mouse, to the border of the target group (and then release to finish the selection). The users were asked to repeat this interaction 10 times in each case. The user interaction (the start point and end point) was recorded. This resulted in 1000 interactions which we then studied in a variation analysis.

Before the start of the study, we presented our new brushing in a training session, where we showed the new technique by brushing a test dataset. These sessions took approximately 10 minutes and the participants were free to interrupt for questions and to take over the software to experiment with the new brushing technique until they were comfortable to do the study.

As mentioned, we also performed a third study to collect additional evidence about whether the learned model would generalize to new data and new users, basically following the design of the first one [22]. We provided six completely new datasets and invited ten new users. In the supplementary material, we include more details about this third study and detailed information about the achieved accuracy.

B.8 Modeling the variation in sketching

In order to make the CNN training as successful as possible, we extended the training data with synthetic interaction data based on the second user study. In our observation, the variation of the user interaction consists of three parts that meaningfully are modeled separately: the start point of the interaction (denoted by c), as well as the length r and the angle ϕ of the drag-component.

If a user brushes a big group, the potential variation of the length of the drag-component will be larger than when brushing a small group. Therefore, when considering the variation of interaction length r, we normalize by the mean of the interaction length. Further, the users' interaction has much less angle variation when they brush an elongated, anisotropic group, compared with brushing more isotropic subsets. Our user study cases are related to a specific question, which was used to instruct the users before brushing. The questions "select a small cluster" (S) and "select a big cluster" (B) are used to expect a rather isotropic data subset from the user and the anisotropic cases are more related to the "select an elongated group" (E) question. Therefore, we compute the variation of angles separately according to the two different types of cases. We used the statistical tool EasyFit [91] to analyze which distribution fits our variation data best and the resulting function details (probability density functions) and their specific parameters are listed in Table B.2.

For synthesizing new training data, given a user interaction I with $c = (c_x, c_y)$, r, and ϕ, we compute a new user interaction I' based on random samples from the accordingly fitted PDFs. The entries in the first column of Table B.2 describe the random samples which we are drawing from the corresponding PDFs (with "var" referring to variance). The new I' is then given by c', r', and ϕ' according to $r' = r \pm \sqrt{\frac{var(r)}{mean(r)}} \cdot r$ and $\phi' = \phi \pm \sqrt{var(\phi)}$. To compute c', the sampling result $var(c)$ is considered to be an intermediate variation as the variation of the start point c is also related to the size of the brushing goal. Therefore, we normalized for this relation before fitting the PDFs. As a result, we need to "undo" this normalization after sampling the PDF and get $c' = c \pm (var(c) + 0.1) \cdot (v + 20)/10$, where v is the standard deviation of the user goal (in x or y direction, respectively).

B.9 Conclusion and future work

With this paper, we demonstrate how deep learning can be used to further improve the central operation of brushing in visual analytics. By learning the relation between the data subset to be selected and a click-and-drag sketch by the user to do the selection, we achieve a solution, which is both very fast and also very accurate. To the best of our knowledge, this is the first study to report the successful application of a structured regression model, realized by a convolutional neural network, to improve a central user interaction in visual analytics—in our case brushing in scatterplots. We demonstrate, quantitatively, and in comparison with the previously published Mahalanobis brush, that our CNN-based solution leads to a significant reduction of the error rate (from $\approx$8% to $\approx$2.5%), while enabling very fast interaction. In the future, we see several opportunities to further extend our work, including

- the design of a brushing tool which is tailored for a single user, using an appropriate method to learn a user's particular brushing behavior over time
- research possible improvements of both the network input (using a different method for capturing the data distribution, like KDE) and the reference output (alternative ways to estimate p from the binary user goal information)
- the extension of our principal approach to other views and according brushes

Acknowledgements

We thank the participants of the user studies and TU Delft for the hardware support (a fast graphics card). We are also grateful to E. Gröller and A. Lundervold for fruitful discussions about the CNN design. In addition, we highly appreciate the help of N. Pezzotti and A. Vilanova with optimizing the CNN architecture.

Paper C

Personalized sketch-based brushing in scatterplots

Chaoran Fan and Helwig Hauser

University of Bergen, Norway

Abstract

Brushing is at the heart of most modern visual analytics solutions and effective and efficient brushing is crucial for successful interactive data exploration and analysis. As the user plays a central role in brushing, several data-driven brushing tools have been designed that are based on predicting the user's brushing goal. All of these general brushing models learn the users' average brushing preference, which is not optimal for every single user. In this paper, we propose an innovative framework which offers the user opportunities to improve the brushing technique while using it. We realized this framework with a CNN-based brushing technique and the result shows that with additional data from a particular user, the model can be refined (better performance in terms of accuracy), eventually converging to a personalized model based on a moderate amount of retraining.

This article was published in *IEEE Computer Graphics and Applications*, 39 (4): 28–39, 2019.

C.1 Introduction

Nowadays, linking and brushing is becoming a prevalent interaction technique for data exploration and analysis in coordinated multiple views [87], widely integrated into many visualization tools such as Tableau and D3.js. Becker and Cleveland [2] were the first to describe the basic principles of brushing, which is interactively painting the visualization of subset of data points with usually simple geometries. In coordinated multiple views, the selected data are consistently highlighted in all linked views after brushing. This amounts to the most common form of focus+context visualization [35], enabling a fast and effective exploration of data relations, which are too challenging to show in just one view.

Due to the importance and prevalence of brushing in visualization solutions, many efforts have been invested in the improvement of brushing technique and brushing variants can be categorized into:

- *brushing using simple geometries*—the most commonly used brushing solutions include the rectangular or circular brushing on scatterplots, line-brushing on data graphs, etc.
- *lassoing*—the user draws a detailed shape around the target group of item representations to select the subset
- *logical combinations of simple brushes*—the user refines the data selection by making use of multiple brushes and combining them with logical operators [17, 72]
- *sketch-based brushing*—a sketch is drawn by the user onto a visualization and a selection heuristic is used to determine which data are selected [22, 76, 84]

Each brushing technique can be discussed in terms of pros and cons, by two particular important criteria:

- *efficiency*—whether the brushing is fast enough (including the interaction and all computation) to enable a fluid data exploration/analysis [20]?
- *accuracy*—does the brushing interaction lead to a selection of exactly the data subset, which the user wished to select in the view?

Despite the large variation of existing brushing tools, we rarely see a solution that combines both criteria really well: Many brushing techniques are indeed fast, as clicking on individual points, for example, or drawing simple geometries—also sketched brushes are fast, requiring only a simple gesture as interaction and thus enabling a swift user-computer dialogue during the exploration/analysis [8]. A common disadvantage of fast techniques, however, is that it can be difficult to accurately brush a particular data subset. On the other hand, we certainly find brushing techniques, that are straight-forward for accurately selecting subsets of interest, such as lassoing and the logical combination of simple brushes. This benefit, however, comes often at the price of the solution being slower—specifying a lasso, for example, easily becomes a unit task by itself [8], potentially interrupting the exploration/analysis process.

Recently, deep learning methods have become the state-of-the-art in a wide range of fields, including natural language processing, object detection and image classification.

Inspired by this, we have recently developed a CNN-based brushing technique for scatterplots and achieved the best accuracy compared to previously existing sketch-based methods with a fast interaction[23]. Although this method has made great progress in learning the user's intention while brushing, it suffers from two limitations: First, the model is trained off-line once by the data from different users and what the model learns is the average brushing preference across the users, leading to a general model which is obviously not optimized to every single user. Second, while it of course is possible to retrain a new model for a single user from scratch, this procedure is time-consuming and requires sufficient training data which is difficult to get from a single user in a short time.

To address the limitations of this CNN-based brushing solution, we here propose an innovative framework which is able to iteratively refine the brushing model for a single user with additional data that he/she provides while using the brushing technique. This idea is inspired by active learning (AL), which is a special case of semi-supervised machine learning that can incrementally improve the existing model by interactively querying the user for additional input. In addition, we exploit knowledge from transfer learning and leverage the parameterization of a well-trained model instead of learning the user's brushing behavior from scratch, largely reducing the time cost of the retraining procedure while maintaining a focus on avoiding overfitting.

To evaluate the usability of our proposed framework, we experimented with the previously published CNN-based brush in scatterplots to see relevant differences in comparison. Our quantitative evaluation shows that the iterative model based on our proposed framework achieves better accuracy and becomes a customized model for a single user as compared with the general model, and the time cost for the retraining procedure is only 3 minutes, which is efficient and acceptable for most data analysts to improve their brushing method even during a short break only.

C.2 Related work

In the following, we first review some critical works concerning brushing for visual analytics, before we then discuss related work concerning applications of convolutional neural networks, transfer learning and active learning.

C.2.1 Brushing techniques

Brushing is intrinsically based on the interaction between the user and the system, often a combination of mouse/cursor motions and clicks. Many variations of brushing have been proposed, each with its own strengths and weaknesses—for example, in terms of their ease of use and the degree of control that the user has. Brushing in scatterplots is often based on the use of simple geometric shapes such as a rectangle or circle. Alternatively, users can use a lasso to specify the selection more accurately.

Several extensions to simple brushing have been published, including techniques to formulate more complex brushes by combining multiple brushes using logical operators. Martin and Ward [72], for example, enable the user to configure composite brushes by applying logical combinations of brushes, including unions, intersections, negations, and exclusive or operations.

Similarity brushing [80] is a typical example of sketch-based brushing, which is based on a fast and simple sketching interaction—the user uses a swift and approximate gesture (for example, drawing an approximate shape that the data should follow) and then a similarity measure (target function) is defined to identify, which data items actually are brushed. This way, the interaction is fast, but likely not 100% accurate.

Recently, the Mahalanobis brush was presented as an interesting alternative for brushing scatterplots [84]. The user simply clicks into the center of a coherent data subset to be selected. The link between the interaction and the actual selection is realized on the basis of an analysis of the underlying data (a local covariance matrix indicates the overall shape and orientation of the data to be brushed, forming the basis for a local Mahalanobis metric, which is then used as a distance measure to select the data).

While this technique is giving quite good results, it still has limitations, including a non-optimized selection of the local context for the Mahalanobis computation and one off-screen parameter for the brush size. As a follow-up work, we extended the Mahalanobis brush and improved the accuracy by optimizing the parameters based on a user study and getting rid of the off-line parameter [22]. However, this improved solution is still linear and has difficulties with complex structures that would require a more flexible approach. Also, it is not really real-time for large datasets.

Later, we exploited deep learning and developed a CNN-based brushing in scatterplots [23]. This model achieves state-of-art accuracy while——as a general model——it only learns the average behavior from different users, and thus is not able to match every single user's brushing preferences in an optimal way.

Koytek et al. [57] created MyBrush, which extended the popular brushing and linking technique by incorporating personal agency. It offers users the flexibility to configure the source, link, and target of multiple brushes, which is able to adapt brushing and linking to preferences and needs. However, the user needs to spend some time to figure out all the possibilities of the configuration initially, and the brushing tool they provide is not based on a data-driven method which cannot be improved while being used.

C.2.2 CNN, transfer learning and active learning

A convolutional neural network (CNN) is a deep learning architecture, which is inspired by the connectivity pattern between neurons and their organization in the visual cortex [45]. The concept of a neocognitron, proposed by Fukushima [28], is widely considered as fundamental basis of modern CNNs. LeCun et al. [62] established the framework of CNNs by developing a multilayer artificial neural network called LeNet-5, which was applied successfully to image classification problems. With the emergence of big data and the development of computing infrastructure, the structure of some CNNs has become very deep. A solution by Krizhevsky et al. [59] was able to classify about 1.2 million images into 1000 classes, i.e., a record-breaking result in the ImageNet Large Scale Visual Recognition Challenge. Often, the impressive success of image processing CNNs is attributed to their ability to learn rich mid-level image patterns as opposed to hand-designed low-level features used in more traditional methods.

As humans we can learn and apply relevant knowledge from previous learning when encountering new tasks. Most of traditional machine learning algorithms are designed to address single tasks. In contrast, transfer learning allows us to bring the power of

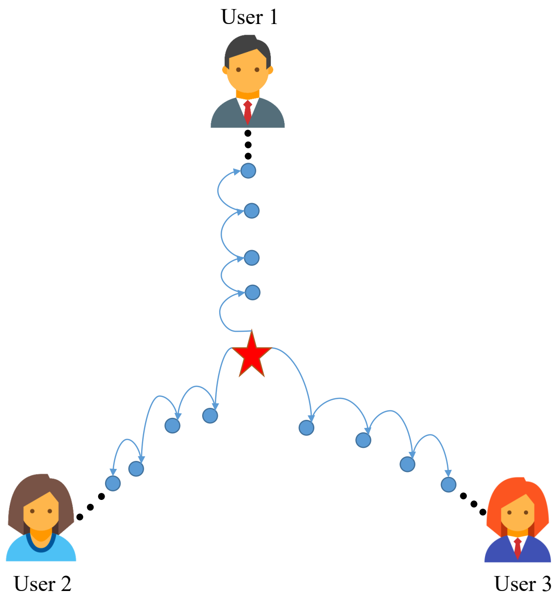


Figure C.1: Illustration of the core idea: the general model (red star) is progressively refined and getting closer and closer to a single user model in the high dimensional parameter space (here 2D in this illustration).

state-of-the-art models to new domains where insufficient data and time/cost constraints might otherwise prevent their use [81]. Transfer learning with CNNs has been also explored and demonstrates that the intermediate activations learned with pretrained deep CNNs on large datasets such as ImageNet and GoogLeNet, can be transferred to many other recognition tasks with limited training data.

Active learning (AL) is a special type of semi-supervised machine learning which incorporates the user into the loop to query label information. As labeling manually is expensive and time-consuming, AL has been successfully applied to the situations where large portions of data are unlabeled. The goal of AL is to improve the training performance of a classifier at the lowest possible annotation cost by intelligently picking the best examples to label [92].

While an increasing number of successful applications of deep learning methods are recognized in many fields, we do not yet see a mentionable number of visualization solutions, particularly in improving interaction techniques. With our work, benefitting from ML, we also hope to inspire interesting new research in this very interesting direction.

C.3 The principal approach

The overall goal of our research was to improve the general CNN-based brushing model [23] and to make it suitable for those users whose brushing preference is deviating from the average. The schematic in Figure C.1 shows the core idea of our proposed framework where the general model is indicated by a red star. To find the optimized model for a single user, we take advantage of the well-trained general model as an initial point in the model's high dimensional parameter space and progressively adapt it to a personalized model. In this way, the prior parameterization is taken into account and we can get rid of the costly general (global) search in the parameter space, largely reducing the time spent and making the results more stable.

Figure C.2 provides an overview of the proposed framework to illustrate our adaptive brushing model. In the following, we first describe the initial model we used from the earlier published paper [23], before we then describe the construction of our solution in detail.

Following the basic definition of the general CNN-based brushing technique [23], we use click-and-drag as the sketching interaction I and a computational function for estimating the brushing result, denoted as S, which is based on the visualization V near sketch I. The CNN structure is built with two convolutional layers, two max pooling layers and three fully connected layers. The input of the CNN is a histogram-like image that is based on the interaction I and the local visualization V near I, while the output of the network is also an image, of same resolution as the input, which contains a degree-of-selection information extracted from the ground truth (users' brushing goal). The general model S_g is trained off-line once based on 500 basic selections (augmented to 8000 in actual training via sampling the natural variation of user interactions), provided by the previously published user study where 50 different users were invited [23]. In the user study [23], a particular scatterplot (one out of six) and a particular request (one out of three) were given, and the participant chose a target data subset to select as ground truth by using a lasso tool, and then also provided the corresponding click-and-drag interaction that this participant would use to select the target group.

In our proposed framework, we establish an interactive scenario to improve the brushing accuracy for a single analyst during data exploration, where the user can actively give some new input when he/she does not like the result generated by the current brushing model. This additional data is used to gradually adapt the general model to a personalized one and we see that with more data from the user, the brushing result is more accurate for this user. To evaluate this framework, we organized a user study where single users were asked to participate in the model refinement procedure 5 times, and each time the user provided 10% new data by brushing new datasets. Then this user's personal brushing preference data contributed to updating the current model in retraining. The design of the user study intended to simulate a daily process of the data analyst while using the brushing tool during 5 working days. The time spent for participating in one round of the user study was around 20mins, which was comfortable for most users. We saw that 10% new data could be obtained in this time period and this amount of new data was reasonably enough to see the incremental effect of the retraining.

For learning a single user's brushing preference, we chose to leverage the existing CNN parametrization of the general model instead of retraining a model from scratch. This is because the data used for training the general model is from 50 different users and it is not practical to get a similar size data from a single user. To avoid overfitting by limited new data, the new data is chosen to replace the most similar data in the original dataset, composing a new training dataset rather than treating the additional data individually. Picking the most similar cases to be replaced is based on two reasons: 1. We aimed at the retrained model to learn the user's specific brushing preference and

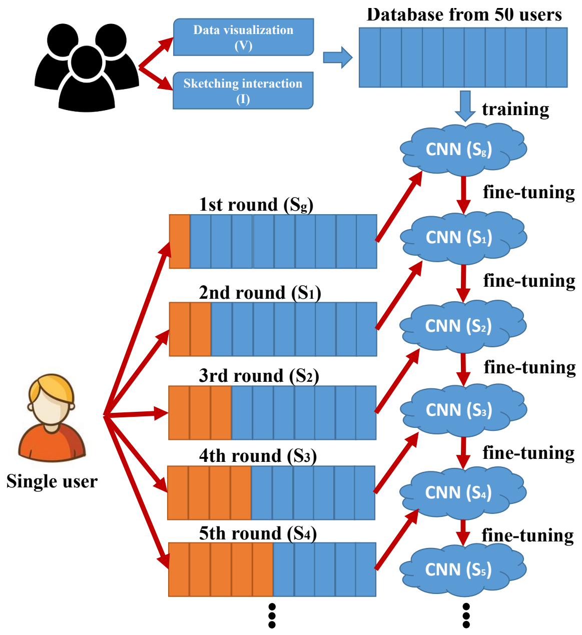


Figure C.2: Principal approach: the general model is fine-tuned during continued use by new data from a specific user.

also keep itself as general as possible (with respect to cases not yet seen by the new user) at the same time. Therefore, the similar cases from the original data can be considered as replicated data which should be replaced. 2. Instead of adding the new data to the retraining dataset, the data replacement strategy accelerates the convergence during retraining.

Our retraining procedure is also based on transfer learning. Transfer learning strategies depend on various factors, but the two most important ones are the size of the new dataset (relatively small or big), and its similarity to the original dataset. As we replace old data, the new training data is the same size and of high similarity (90%) to the old training data. Therefore, we can fine-tune the weights of the pretrained current network via backpropagation with less of a chance to be overfitted. In the following, we introduce the algorithm for the dataset replacement in detail.

C.3.1 Training dataset replacement

The original data for training the CNN-based brushing model from our previous work [23] is based on 500 selections and the training data can be defined as $(x_i \in T_{in}, y_i \in T_{out})_{i=1}^{N}$ with pairs of input and expected output (N is the number of the training samples). In order to find the appropriate data to be replaced among the 500, we formulate a similarity metric based on the extracted features of the input data.

Figure C.3 shows the procedure of CNN input generation. The histogram-like image data used as input for training the network is converted from the local data visualization V and the related interaction I, the features we compute are mostly based on the

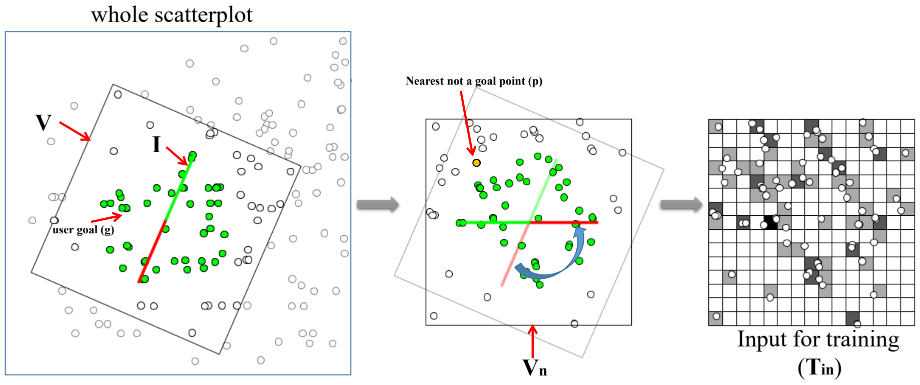


Figure C.3: Illustration of CNN input computation: Left, a square area is specified by the interaction (red line segment); Middle, after rotating to the horizontal; Right, histogram of the local data distribution (CNN input).

local visualization V after normalization (V_n) with respect to the interaction I by a linear transformation. Moreover, to represent the features, we have the number of points within V_n denoted as n_v and the number of user goal points and all points in scatterplot are denoted as n_g and n_c respectively. Besides, the area of V_n is represented as s_v while the area of the scatterplot canvas is s_c.

To extract important and meaningful features from V_n, the size of the user's brushing goal is obviously needed to be considered. We measure the influence of user goal size by calculating the ratios of the user goal size to all the points within V_n and the user goal size to all the points in the scatterplot, which can be denoted as $f_{pR} = \frac{n_g}{n_v}$ and $f_{gR} = \frac{n_g}{n_c}$ respectively.

To know whether the user focuses on a small or large area, we can compute the ratio of areas of V_n to the whole canvas of the scatterplot. This can be denoted by $f_{aR} = \frac{s_v}{s_c}$.

The geometrical shape of the user goal is another important feature which needs to be taken into account. We estimate this by finding the nearest point (denoted by p, not a user goal) to the start point (denoted by c) of I, then compute the ratio $f_{nR} = \frac{d_E(p,c)}{r}$, where $d_E(p,c)$ is the Euclidean distance between p and c, and r is the length of I.

As we know, the eigenvectors indicate the direction of points which are stretched by the transformation and the corresponding eigenvalue is the factor by which it is stretched. Therefore, the ratio of the eigenvalues is a proper way to measure the isotropy and anisotropy of the user goal, which can be defined as $f_{eR} = \frac{e_1}{e_2}, e_1 < e_2$, where the eigenvalues e_1 and e_2 are computed by eigendecomposition of the covariance matrix of the user goal.

In addition, we compute scagnostics to characterize the V_n. Scagnostics is short for Scatterplot Diagnostics, first mentioned by John and Paul Tukey [97] to identify interesting structures according to density, skewness, shape, outliers, etc. The nine Scagnostics measures are defined as $f_{outlier}$, f_{skewed}, f_{clumpy}, f_{sparse}, $f_{striated}$, f_{convex}, f_{skinny}, $f_{stringy}$ and $f_{monotonic}$ respectively and we use f_{sg} as a shorthand of these 9 features together.

Lastly, we look at the density of the histogram-like image (T_{in}) as it represents the data distribution. The value of each bin in T_{in} is normalized into $[0,1]$, we compute 4 indicators by counting the percentage of the value of each bin in $[0,0.25]$ (f_{q_1}), $(0.25,0.5]$ (f_{q_2}), $(0.5,0.75]$ (f_{q_3}) and $(0.75,1]$ (f_{q_4}) respectively.

Overall, we use 18 different indicators to represent the user's selection. To preprocess the indicators for an efficient comparison, we use principal component analysis (PCA) to reduce the dimension from 18 to 8 while keeping 92% variance of the original dataset. The dissimilarity function which is used to pick the most similar case among the old data, can be defined as, can be defined as

$$dis(i_{new}, i_n) = d_E(f_{pca}(i_{new}), f_{pca}(i_n)) \tag{C.1}$$

where i_{new} and i_n are the original features of new data and old data, respectively. d_E represents the Euclidean distance and f_{pca} are the features after PCA, $n \in$[1,500]. The most similar case has the minimum value in terms of the dissimilarity function.

C.4 Retraining the CNN

Usually, high accuracy cannot be achieved unless enough training samples are provided. As we replaced the old data by the new data, so the selection samples in the retraining data are still 500, which are not enough for retraining. Therefore, we synthesize 7500 additional training data from the already acquired training set based on the user's natural interaction variation. This data augmentation method is illustrated in our previously published paper [23]. Then we optimize the parameters of the network based on the training data (N=8000) using the mean-squared error as a loss function. In the following, we will illustrate the details of retraining the CNN.

C.4.1 Training details

In practice, it is rare to train an entire CNN from scratch with random initialization. This is because it is relatively difficult to have a dataset of sufficient size that is required for the depth of the network. Therefore, it is very often still beneficial to initialize with weights from a pretrained model.

Since the initial model is quite general and capturing the common brushing preference, we choose to fine-tune the network to make the initial model adapt well to the target dataset. As the CNN features are more generic in early layers (such as edges or local shapes) and later layers of the CNN become progressively more specific to the details of the information contained in the target dataset, an effective way for fine-tuning the pretrained parameters is to make the learning process happen only in the fully-connected layers, and keep the parameters of the convolutional layers frozen. To validate this approach, we also recomputed the brushing accuracy based on retraining the whole network and found that the results were worse (about 5%) than freezing the early layers of the network, which supports that our retraining strategy is appropriate.

In our user study, we iteratively retrain the model based on the pretrained model 5 times. We start the retraining after the first round of the user study, where we used the general model from our previously published paper [23] as a starting point for optimization towards to single user's tailored model.

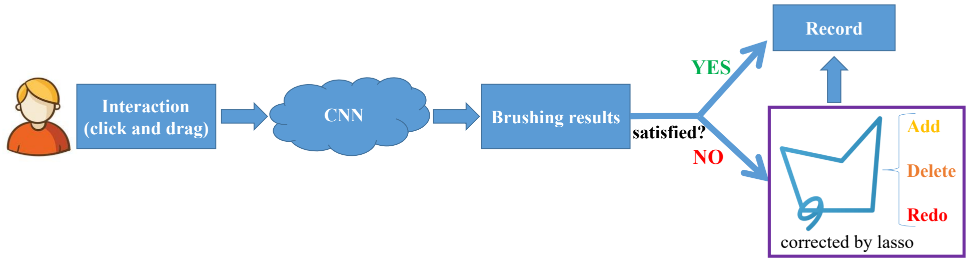


Figure C.4: Pipeline of the user study. The user brushes on the scatterplot based on the trained CNN model, and if he/she is not satisfied with the results, a lasso is offered to do corrections. In the end, the user's brushing goal, interaction and the brushing result are recorded.

For the experiment, we retrained the network in Keras with Tensorflow as the back-end, which provides useful GPU acceleration and coding flexibility. For the training and testing, we used a PC with an Intel Xeon E5-1650 CPU and an NVIDIA GeForce GTX 1080 GPU. During fine-tuning, as we expect the weights of the pretrained CNN to be relatively good, the learning rate was set to be 5×10^{-4}, which is half of the learning rate for training the initial model. In this way, the model will not be distorted too quickly and too much. And as we can make use of the existing parameterization, thus we only need 2000 epochs instead of 10000 (training from scratch) to obtain a good convergence towards a high-quality state. As a result, the retraining procedure for fine-tuning a model once only takes 3 minutes.

C.5 User study

As users have their own brushing preference, it is important to explore how the user uses our brushing tool and test whether it is possible to allow the model to be customized. To evaluate our proposed framework with the CNN-based brushing for a single user, we conducted a user study to collect the user's brushing data and then retrained the model to better fit a single user. In our user study, 8 users are invited who are students or employees (7 are from Delft University of Technology and 1 is from University of Bergen).

C.5.1 Study datasets

During the user study, we provided 25 different datasets for the user to brush, which all were not used for constructing the general model [23]. Besides, the users were encouraged to bring their own datasets to replace the provided datasets.

C.5.2 Study process

Our user study consisted of two parts:

In the first part, a scatterplot was provided to the users and then they could freely use the click-and-drag interaction to brush some data subset of their choice.

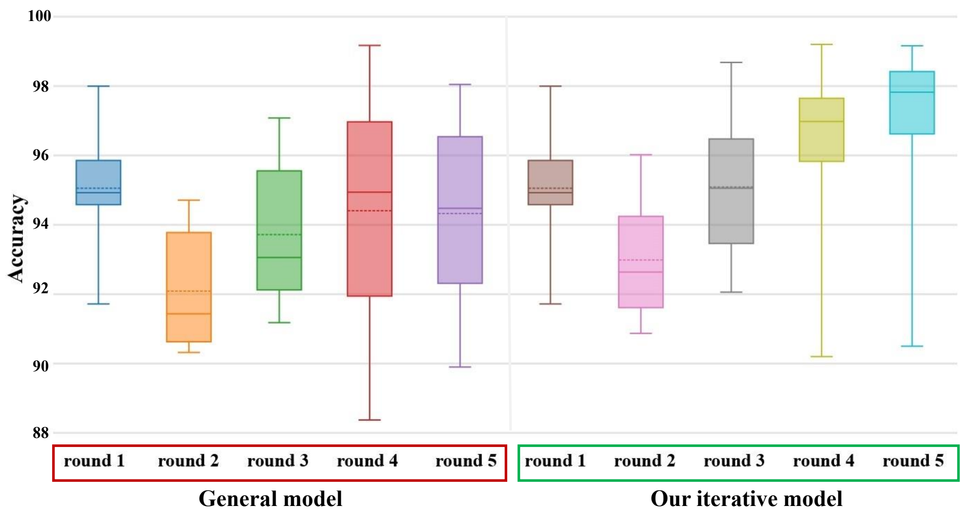


Figure C.5: Boxplots of the brushing accuracy over rounds based on the general model (left) and our iterative model (right). The dash line and solid line in the box show the mean and median respectively.

In the second part, the brushing result based on the current CNN model was shown to the users immediately after they finished the interaction. Then they could think about whether the results were what they wanted originally. If not, they could use a lasso to do corrections (add and delete points, or directly specify the goal) until they were satisfied with the results. For a specific scatterplot, every user had to do 5 selections including the correction, if needed, before they would click the "next" button to switch to another scatterplot.

In order to investigate the influence of the retrained model based on the user's new brushing data, we asked each user to participate in the user study for 5 rounds and each scatterplot from 25 datasets showed up twice in total but in different rounds. For each round, the user needed to finish 5 selections in 10 different scatterplots, and these data were then applied to retrain a new, adapted brushing model which was then used for the next round. In the first round of the study, we directly used the already trained model from our previously published paper [23] as the initial brushing model. Then this model will be retrained by the data obtained from the first round of the study.

As the retraining procedure takes approximately 3 minutes, the users finished a full user study (5 rounds) in 5 discontinuous time periods. During the user study, the brushing results generated by the current model and the user's real goal and interaction were recorded. The pipeline of the user study is shown in Figure C.4.

Before the start of the user study (only for the first time when the user joins), we presented our brushing technique in a training session, where we showed how the technique works and the interface operation in a test dataset. This session took approximately 10 minutes for each participant and the participants were free to interrupt for questions and to take over the software to experiment with the brushing technique (e.g. the click-and-drag interaction and correcting the results by lasso) until they felt ready

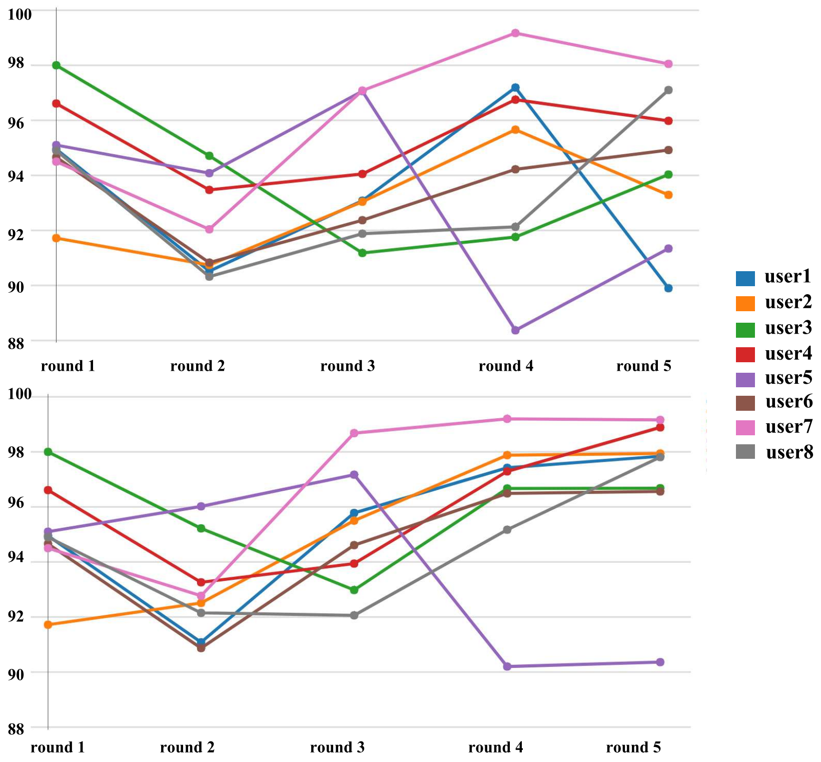


Figure C.6: Line plots of the brushing accuracy over 5 rounds of 8 users based on the general model (top) and our iterative model (bottom).

to do the study.

C.6 Evaluation

For evaluating our proposed iterative model, we did a quantitative accuracy comparison with the previously published general model [23]. Figure C.5 is a statistical comparison with boxplots. The left five boxplots and right five boxplots show the brushing accuracy of 8 users in all five rounds based on the general model (left) and our iterative model (right) respectively. The accuracy of our iterative model in each round is calculated based on the retrained model from the last round. In the first round of the user study, the user's brushing results are computed by the general model without optimization based on the new data, thus the difference between the general model and our iterative model appears from the 2nd round to the 5th round. The dashed line in the box is the mean, if we compare the accuracy by pairs in terms of the round, we can see after iterative retraining, our model performs better than the general one (with higher median and mean). Besides, the size of the boxes on the right are smaller than the corresponding one on the left, this indicates our model has less variance and becomes more stable. Figure C.6 shows an accuracy comparison with line plots, between the general model

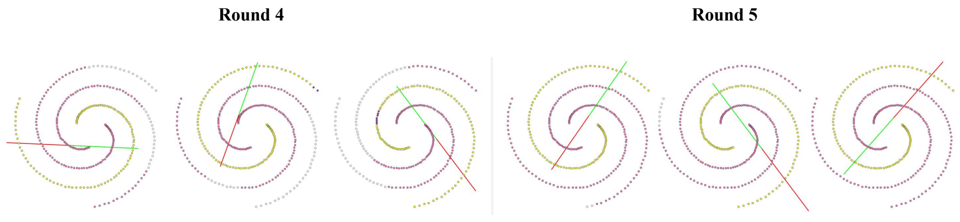
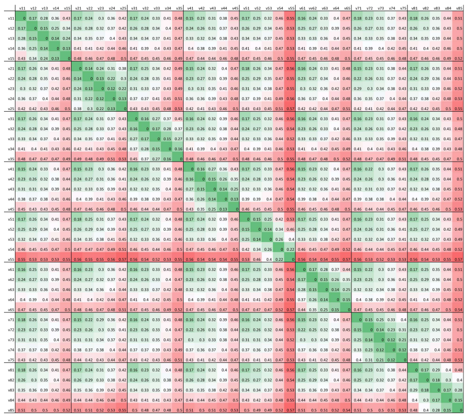


Figure C.7: Worst performing cases of user 5 in round 4 (left) and round 5 (right).


A

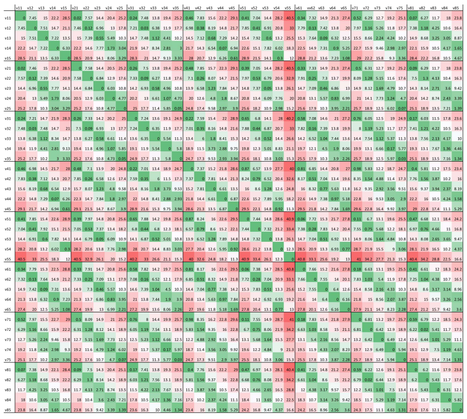

B

Figure C.8: Vector similarity comparison of the CNN model parameters in each round of each user. Matrix A: angle between vectors. Matrix B: absolute length differences.

(top) and our iterative model (bottom). By looking at the accuracy variation altogether, we can obviously see a rising trend of accuracy over rounds on the bottom side while the accuracy of the general model is more distributed (no special trend except a decline from round 1 to round 2) on the top. This shows a strong indication that with additional data for retraining the model, it is able to gradually learn the brushing preference of a single user, which leads to a more personalized model from the general model. In addition, we also see an outlier (user 5) with a sharp accuracy decrease in the last two rounds. To investigate the reasons behind, we pick the 6 worst performing cases from user 5 in the 4th and 5th round and list them in Figure C.7. The brushing results are compared to the user goal (encoded by color). The true positives (correctly brushed), true negatives (correctly not brushed), false positives (falsely brushed) and the false negatives (falsely left out) are colored in yellow, white, pink and purple, accordingly. We can see that the user tried to select a complicated spiral shape and this kind of selection does not exist in previous rounds, being almost impossible for the network to predict at first time. For the relatively low accuracy (actually 90.36% is still very good) in round 5, as there are only 3 spiral-like cases from round 4 which contribute to the network retraining, so it is difficult to tweak the model to learn it.

During the retraining procedure in each round, the parameters of the network are updated. To understand what the network learns for each user and the relation between different user models, we extract the parameters of the CNN model in each round of each user, composing a vector with 35233 dimensions, which is denoted as vmn, where m is the user ID and n is round number. In Figure C.8, we compute the angle and absolute length differences to measure the similarities between these 40 different vectors, which are shown in two matrices with green (low value) to red (high value) as the color legend. In both matrix A and B, we see the outlier model in round 4 (v54) and round 5 (v55) of user 5 also have big difference with other users in this comparison. In addition, in matrix A, the angle between intra-user vectors are clearly smaller than inter-user's—this indicates that the model indeed approaches different user point in the parameter space. For matrix B, we see a clear pattern that the diagonals right next to the main diagonal are also very close to 0. This gives a strong impression that the model indeed adapts in small steps over the five rounds. In summary, the visualization of the CNN model parameters shows clear evidence that our iterative model learns reasonable parameters while being adapted instead of random search in the parameter space.

C.7 Conclusion and future work

In this paper, we present a user-centric framework which takes the user in the loop to iteratively improve a brushing technique in scatterplots. By refining a general model for estimating data selections from simple click-and-drag interactions incrementally with the additional data from a single user and leveraging the existing parameterization, we achieve a solution which is able to turn the general model based on people's average brushing preference to a tailored model for the specific user with a very short time training cost (≈3mins). In addition, we examined the quantitative performance of our iterative model in comparison to the general model, with the retraining procedure over time, the results show a clear improving trend compared to the general model: from 92.09% to 92.99% (+0.9%, with 10% new data), 93.72% to 95.09% (+1.37%, with 20% new data), 94.41% to 96.29% (+1.88%, with 30% new data) and 94.33% to 96.92% (+2.59%, with 40% new data). We assume the accuracy improvement can be even better with more user input and the most practical thing is that the retraining time cost is reduced to 4% of training a general model from scratch.

In the current version of the data replacement algorithm, we entirely replace the most similar cases in the original dataset by the new data from the user. We also did a follow-up experiment to investigate the influence of an alternative replacement rule: we checked the dissimilarity value of the replaced cases and found that 80% of the replaced cases have a dissimilarity value less than 0.5. Based on this statistics, we recomputed the replacement with the most similar case replaced only if the dissimilarity value is less than 0.5 (threshold), in this way, we can keep 20% of the (relatively different) old data which otherwise would have been replaced also. The result shows, however, that the model accuracy is almost the same as before. Still we see the potential to find an optimal threshold to improve the model in the future.

In addition, we see several other opportunities to further extend our work, including

- the current retraining procedure is off-line, the ultimate goal is to make it real-time
- give users more flexibility to configure the brushing tool and even involve them to the brushing tool design

- explore other deep learning methods such as RNN and GAN to learn user's brushing behavior
- the extension of our principal approach to other views and according brushes

We hope that this work can inspire further related research, especially in visualization to offer the opportunity for users to improve the data-driven approach by their own knowledge and behavior.

C

Paper D

On sketch-based selections from scatterplots using KDE, compared to Mahalanobis and CNN brushing

Chaoran Fan and Helwig Hauser

University of Bergen, Norway

Abstract

Fast and accurate brushing is crucial in visual data exploration and sketch-based solutions are successful methods. In this paper, we detail a solution, based on kernel density estimation (KDE), which computes a data subset selection in a scatterplot from a simple click-and-drag interaction. We explain, how this technique relates to two alternative approaches, i.e., Mahalanobis brushing and CNN brushing. To study this relation, we conducted two user studies and present both a quantitative three-fold comparison as well as additional details about the prevalence of all possible cases in that each technique succeeds / fails. With this, we also provide a comparison between empirical modeling and implicit modeling by deep learning in terms of accuracy, efficiency, generality and interpretability.

This article was published in *IEEE Computer Graphics and Applications*, 41 (5): 67–78, 2021

D.1 Introduction

Linking and brushing is a widely adopted interaction technique for visual data exploration in coordinated multiple views [87]. Over 30 years ago, Becker and Cleveland [2] defined brushing as an interactive method to select data points by using simple geometries on a data visualization such as a square, circle, or a polygon. In coordinated multiple views, brushing usually leads to a consistent highlighting of the selected data in all linked views, amounting to an important form of focus+context visualization [35].

Since brushing is central to visual analytics, a substantial amount of research has been devoted to it. The many available forms of brushing can be categorized into four different types:

- *simple geometries*—this is the most common category, including the rectangular brush on scatterplots, line brushing on data graphs, etc.
- *lassoing*—the user selects a data subset by drawing a geometrically detailed lasso around the subset's visualization in a view.
- *logical combinations of simple brushes*—the user refines the data selection iteratively by using multiple brushes and combining them using logical operators such as AND and OR.
- *sketch-based brushing*—the user sketches a shape onto a visualization and a selection heuristic is used (often based on a similarity measure) to determine which data are selected.

To evaluate a brushing technique, the following two criteria are of particular importance:

- *efficiency*—how fast is the brushing algorithm and how much time does the users spend on the selection process; is the interaction fluid?
- *accuracy*—to which degree does the interaction lead to an accurate selection of the data subset as the user actually intended to select?

Clicking on a mark in a visualization to select the corresponding data point, for example, is highly efficient for selecting individual data points. High accuracy is possible, when using a geometrically detailed lasso to select data points in a scatterplot. In many cases, it is not trivial, or possible, to optimize for both criteria concurrently.

Many brushing techniques are indeed fast—we think of brushing to be fast, if only one click or only very few atomic interactions are needed to specify the brush, leading to a swift user–computer dialogue during exploration / analysis. The use of simple brushing geometries (rectangle, circle, etc.) and sketch-based brushes, where only a quick gesture is used for brushing, are examples of fast techniques. A common disadvantage of these methods is, however, that it can be difficult to accurately brush a targeted data subset.

Certainly, we also have brushing techniques that are accurate—likely with lassoing and the logical combination of simple brushes being the most prominent examples. With such a technique, it is straight-forward to accurately select a subset of interest. This benefit of being accurate, however, commonly comes at the cost of reduced efficiency—specifying a lasso point-by-point, for example, easily becomes a lengthy unit task by itself, interrupting the data exploration process.

To optimize both criteria for one technique as much as possible, data-driven methods are an interesting option. One typical kind of such a solution is based on sketch-based user interaction. The sketching of a simple shape, for example a line segment by a click-and-drag interaction, is complemented with a heuristic that estimates the actual data selection from the sketch. To achieve high accuracy, the parameters of such a technique can be optimized on the basis of data from a user study [22, 23].

The Mahalanobis brush [22, 84] is an example of a data-driven technique, using local covariance information as basis for a Mahalanobis metric which determines which points are closest to the sketch. The parameters of Mahalanobis brushing can be optimized based on data from a user study [22]. Quantitative evaluation shows that it can achieve ≈92% of accuracy, based on a fast click-and-drag interaction. Inspired by the success of machine learning, a brushing technique based on deep learning (DL) with a convolutional neural network (CNN) was developed [23] and achieved a substantially improved accuracy (≈97%). Both the interaction as well as the data visualization were represented as images to subject them as input to the CNN, and letting the network learn the model based on data from a user study.

With Mahalanobis brushing [22] and CNN brushing [23], two principally different approaches are given, i.e., representing the principles of empirical modeling (based on reasoning) vs. implicit modeling (based on deep learning). Since CNN brushing resulted in a significantly higher accuracy [23], and since Mahalanobis brushing is not based on any advanced distance metric, we were interested in studying to which degree empirical modeling can in fact compete with deep learning in this context.

In this paper, we now provide details of our attempt to construct an improved empirical model by further extending the Mahalanobis brush, incorporating kernel density estimation (KDE) [82], to inform a clustering step which then returns one of the clusters as the data selection [24]. Additionally, we contribute an in-depth, three-fold comparison between the Mahalanobis brush, the CNN brush, and the new KDE brush.

D.2 Related work

Selecting data subsets by brushing is one of the most common types of interaction in visual analytics, where often data points are selected in one display, while the same information is highlighted in linked views (linking and brushing) [87].

Often, a combination of mouse motions and button clicks are used to realize brushing [72]. Less common methods are based on eye / head tracking or gestures, as for example in virtual reality [107].

Several extensions to simple brushing have been proposed, including techniques to formulate complex brushes by combining simple brushes using logical operators, including the union, intersection, and negation of brushes, and enabling the user to iteratively refine the data selection.

Furthermore, brushes are often adapted to interact with a particular aspect of the visualization mapping. Hauser et al. [36], for example, developed angular brushing on parallel coordinates to select data points, whose representation include lines with a particular angle. MyBrush was suggested by Koytek et al. [57] in order to extend brushing and linking by incorporating personal agency, allowing to configure the source, link, and target of multiple brushes.

Sketching is a natural and intuitive way for users to identify data subsets of interest in a visualization. Similarity brushing [76] is an example of sketch-based brushing, which is based on a fast and simple sketching interaction—the user performs a swift and approximate gesture (for example, drawing an approximate shape that the data should follow) and then a similarity measure is used to identify, which data items are actually selected. The main advantage of sketch-based brushing is the fast interaction, which, however, is not perfectly accurate, usually.

As another sketch-based solution, the Mahalanobis brush was presented as an interesting option for brushing scatterplots [84]. In this technique, a simple sketch (a click near the center of the data subset that should be selected) is used to brush the data. The link between the interaction and the actual selection is computed on the basis of the underlying data (local covariance information is used to determine the overall shape and orientation of the selection): all data points near the click-point, measured according to a local Mahalanobis metric, are selected.

While this technique is giving good results ($\approx$65% accuracy [22]), it still has limitations, including a non-optimized selection of the local context for the Mahalanobis computation and one off-screen parameter for the brush size. To improve the accuracy of this approach, the Mahalanobis brush was extended by optimizing its parameters using data from a user study and getting rid of the off-line parameter [22]. In terms of efficiency, the average interaction (click-and-drag) time spent for the new Mahalanobis brushing is only 41% of Lasso [22]. However, this improved solution is still linear and has therefore difficulties with complex structures that would require a more flexible approach.

Inspired by the success of deep learning in a wide range of applications, especially in image processing, we then developed CNN brushing [23], using deep learning to establish the link between the user sketch and the actual data selection, achieving very high accuracy. However, as a general model, it learns the "average behavior" from different users, and thus is not able to match every single user's brushing preferences 100%. To address this issue, we further improved CNN brushing and achieved a solution which is able to turn the general model into a tailored model for a specific user [25]. To achieve this, we take the user into the loop to iteratively refine the brushing model with additional data which the user provides while using the brushing technique.

Despite the opportunity to achieve really good results with the help of deep learning, related approaches to solve interactive visualization tasks are still rare. The limited utilization of deep learning for visual analytics solutions is likely based on three reasons: (1.) Understanding a deep learning based model is challenging due to its "black box" nature. (2.) Usually, high accuracy of DL-based prediction requires large amounts of training data, which often is difficult to acquire. (3.) There is no established common understanding of how to determine the right DL solution as knowledge of topology, training method and required hyperparameters. Consequently, it is often difficult to efficiently make good use of deep learning—especially, when non-standard tasks are to be supported.

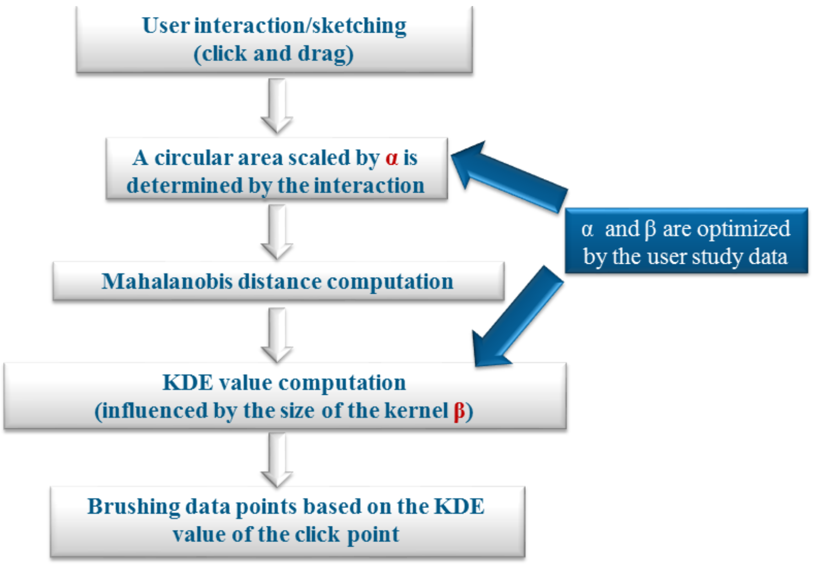

User interaction/sketching (click and drag)
A circular area scaled by α is determined by the interaction
Mahalanobis distance computation
KDE value computation (influenced by the size of the kernel β)
Brushing data points based on the KDE value of the click point
α and β are optimized by the user study data

Figure D.1: Overview of KDE brushing: the user clicks into the middle of the subset to be selected and drags the pointer to the border of the subset (sketching interaction); then a selection of points around the click-point is determined, based on the estimated density of the data; two parameters, α and β, related to the sample size and the size of the KDE bandwidth, influence the results and we optimize them based on a user study with 50 participants.

D.3 KDE brushing in scatterplots

Fig. D.1 shows an overview of the new KDE-based brushing technique. We keep the simple click-and-drag interaction for sketching the target data subset: click into the middle of the subset and drag the pointer swiftly to the outer boundary of it. The click point $\mathbf{s} = (s_x, s_y)^\top$ and the end point $\mathbf{e} = (e_x, e_y)^\top$ of the drag interaction indicate the size of the target subset that the user wishes to select. As with the Mahalanobis brush [22], we first consider a circular subset, centered around **s**, and estimate the shape and orientation of the data in this region by looking at the local covariance information. We then start a short iteration to refine this data subset selection, based on the local covariance information—ideally, we would like to directly use the resulting user selection as the data subset, of course, amounting to a chicken-and-egg type of problem, since the eventual selection is not known in advance. After a sufficiently close convergence of this iteration, we make a selection of data points, based on a kernel density estimation (KDE), using the local covariance information as a basis for specifying the kernel. The choice of KDE is based on three assumptions: (1.) Data-driven density information, captured by KDE, should improve results over the linear model in Mahalanobis brushing. (2.) A non-linear model should be suited to select general shapes. (3.) The modes (local maximum points) of a 2D KDE, at the right scale, can represent clusters of data points, corresponding to selection targets. In the following, we provide more details about the individual components of this approach.

D.3.1 Mahalanobis distance computation

Since the Mahalanobis distance is central to KDE brushing, we briefly review it first. Introduced by Mahalanobis in 1936 [69], it is based on the correlation between data variables to help with the identification and analysis of multivariate patterns. It is unitless and scale-invariant, which is a useful way for determining the similarity of an unknown sample to a known one. It differs from the Euclidean distance, which measures the distance with the available data. The Mahalanobis distance between vectors **a** and **b** is defined by

$$d_\Sigma(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})^\top \Sigma^{-1} (\mathbf{a}-\mathbf{b})} \tag{D.1}$$

where Σ is the covariance matrix of the sample. The locations of equal Mahalanobis distance from a central reference vector $\mathbf{x}$ form an ellipse around $\mathbf{x}$ in 2D.

In our proposed KDE brushing, the local covariance structure of a data subset around the click-point $\mathbf{s}$ is used as the basis of the kernel specification. Therefore, it is an important part of our approach to determine, which data subset should be used for this computation. We perform this in two steps.

Initially, we consider a circular area with radius $\alpha \cdot d_E(\mathbf{s},\mathbf{e})$, where α is a weighting factor and $d_E(\mathbf{s},\mathbf{e})$ is the Euclidean distance between $\mathbf{s}$ and $\mathbf{e}$. All data points within this circle are used to compute the first instance of the local covariance information, Σ_1.

Next, we consider all points in a Mahalanobis ellipse, based on Σ_1 and sized to $d_{\Sigma_1}(\mathbf{s},\mathbf{e})$. Usually, this leads to a new subset, which is similar to the initial one, but fitting the underlying data structure more closely. To obtain an even better sample, we refine the sample iteratively by replacing it with the points in the Mahalanobis ellipse that is updated every iteration according to the covariance of the samples in last iteration.

While this process usually converges quickly, we observed that it sometimes can lead to small fluctuations, including and excluding a few points in consecutive iterations. To stabilize the convergence of the covariance matrix optimization, we enable the partial consideration of data points during the computation, leading to a solution that is based on the weighted covariance matrix. The elements σ_{jk} of the weighted covariance matrix Σ_w can be defined as:

$$\sigma_{jk} = \frac{\sum_i \omega_i (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)}{1 - \sum_i \omega_i^2} \tag{D.2}$$

where ω_i is the normalized weight ($\omega_i \geq 0$) for vector $\mathbf{x}_i$ in a weighted sample with $\sum_i \omega_i = 1$ and $\bar{\mathbf{x}}$ being the weighted mean vector, given by $\sum_i \omega_i \mathbf{x}_i$. During the iteration, the weight of each point is updated and the points that are stable in the Mahalanobis ellipse are assigned a higher weight than less stable points. This process results in a well-converged covariance matrix after a few interations. More details of this procedure (weight function design, singular matrix handling, etc.) are described in an earlier publication [22].

D.3.2 Density estimation

Kernel density estimation (KDE) is a popular method for data analysis [82]. It is a non-parametric way to estimate the probability density function of a random variable.

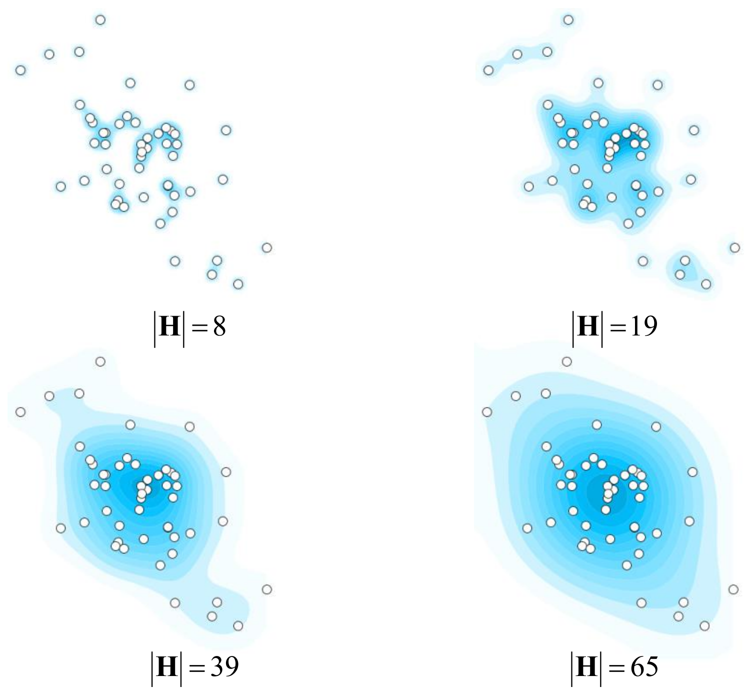


Figure D.2: Changing the kernel size in KDE: bigger kernels, i.e., larger $|\mathbf{H}|$, bring forth larger structures in the data, while smaller kernels represent details.

KDE can be used, for example, to make inferences about data, based on a finite sample, without imposing any a priori structure on the data.

Given that $\{\mathbf{x}_i\}_{1 \le i \le n}$ are n samples of d-dimensional vectors from a common distribution, KDE can be used to estimate their density as

$$f_{\mathbf{H}}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} K_{\mathbf{H}} \cdot (\mathbf{x} - \mathbf{x}_i) \tag{D.3}$$

with $\mathbf{H}$ being a $d \times d$ bandwidth matrix (symmetric and positive definite). The choice of matrix $\mathbf{H}$ is the most important factor, critically affecting the characteristics of $f_{\mathbf{H}}$. Fig. D.2 shows four results from 2D KDE with increasing determinant of $\mathbf{H}$, where $f_{\mathbf{H}}$ reveals details for small $|\mathbf{H}|$, while larger data structures dominate for larger $|\mathbf{H}|$.

$K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-\frac{1}{2}} K(\mathbf{H}^{-\frac{1}{2}}\mathbf{x})$ is the kernel function, with $K(\mathbf{x})$ being a symmetric multivariate density function with $K(\mathbf{x}) \ge 0$ and $\int K(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1$. A variety of kernels has been studied, including the uniform kernel, the triangle, normal / Gaussian, and Epanechnikov kernel, as well as others. The choice of the kernel function is actually not as important as the choice of the size (and shape) of $\mathbf{H}$. Being interested in the local mode of the data distribution, we use the normal kernel for KDE brushing.

To consider the local data distribution, when modeling kernel matrix $\mathbf{H}$, we make direct use of the converged covariance matrix Σ_w, leading to the following anisotropic

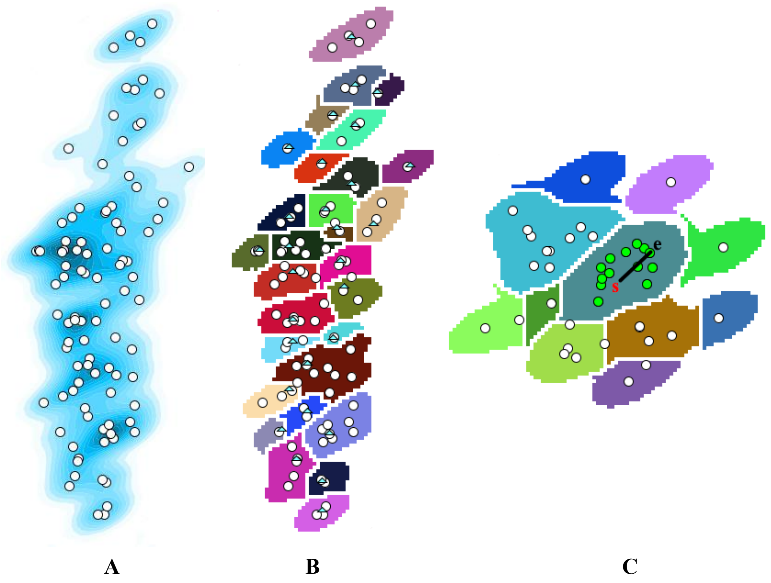


Figure D.3: A: KDE of a dataset (relatively small kernel). B: Clustering related to the modes of the KDE, indicated by the small blue triangles. C: The one cluster, which corresponds to the KDE mode near to **s** determines, which data points are selected (indicated as green points).

kernel function:

$$K_{\mathbf{H}}(\mathbf{x}) = \frac{e^{-\frac{1}{2}\mathbf{x}^\top \Sigma_w^{-1}\mathbf{x}}}{\sqrt{(2\pi)^d |\Sigma_w|}} \tag{D.4}$$

To realize an proper scaling of the kernel, we use the eigendecomposition of $\Sigma_w = \mathbf{V}\Lambda\mathbf{V}^\top$ with eigenvectors $\mathbf{V}$ and eigenvalues Λ_{ii}. This leads to the scaled versions of $|\mathbf{H}|^{-\frac{1}{2}} = |\beta\phi\Lambda|^{-\frac{1}{2}}$ and $\mathbf{H}^{-\frac{1}{2}} = \mathbf{V}(\beta\phi\Lambda)^{-\frac{1}{2}}\mathbf{V}^\top$.

Used with an isotropic kernel function $K(\mathbf{x}) = (2\pi)^{-\frac{d}{2}} e^{-\frac{1}{2}\mathbf{x}^\top\mathbf{x}}$, this corresponds to KDE with an accordingly scaled kernel matrix. We optimize the scaling of **H** by choosing the two scaling parameters ϕ and β by two separate methods: On the one hand, we use a data-driven approach to determine ϕ. On the other hand, we optimize β as a tuning parameter using the data from the user study.

D.3.3 Selecting a data subset using clustering

The modes of KDE represent groups of data items (at the scale determined by $|\mathbf{H}|$). We use clustering (each mode leading to one cluster) to identify the one group of data items, which is associated with the click-and-drag interaction, and select it.

The clustering is applied to a grid-based canvas where each grid has its KDE value. During the clustering, we use a simple watershed algorithm [26]: Starting with the mode with the highest KDE value, we iteratively include neighboring locations into the corresponding cluster, lowering the threshold iteratively. In every step, we either join a neighboring location to an existing cluster, or create a new cluster, if the new location is not adjacent to an existing cluster. In addition, for the clustering, we only apply the algorithm to the points in a square area with center being the start point **s** and side length being $2 \cdot r \cdot \omega$ (r is the length of interaction). Examining the user study data, we found $\omega = 1.5$ to be a reasonable value that the corresponding square area can cover all the user goals. Fig. D.3B shows an according clustering result for a KDE with a relatively small kernel (shown in Fig. D.3A) where the different clusters are shown in different colors and the corresponding KDE modes are located by small blue triangles. Fig. D.3C shows an example of how data points are then selected (the points in the same cluster, corresponding to click-point **s**, are selected and highlighted in green).

D.3.4 Optimizing the kernel size

The number of modes of a KDE is strongly related to the kernel size: the bigger the kernel, the fewer modes. Thus, there is a strong relation between the size of the kernel and the size of the cluster, which is used to select the data points. In the following, we describe a data-driven approach to determining an appropriate scaling factor ϕ.

Since we aim at a KDE that provides one cluster with the targeted data points, we optimize the size of the bandwidth kernel so that the size of the resulting cluster matches the size of the Mahalanobis ellipse around **s** and through **e** as closely as possible—as a measure of comparison, we are using the dice coefficient between the Mahalanobis ellipse **E** and the KDE cluster $\mathbf{C}(\phi)$:

$$s(\phi) = \frac{2\,|\mathbf{E} \cap \mathbf{C}(\phi)|}{|\mathbf{E}| + |\mathbf{C}(\phi)|} \tag{D.5}$$

where $|\mathbf{E}|$ and $|\mathbf{C}(\phi)|$ are the sizes of the Mahalanobis ellipse and the KDE cluster, respectively (evaluated grid-based). In our experiment, the searching domain for ϕ was $[\frac{1}{10}, 3]$.

The example in Fig. D.4 illustrates the influence of different kernel sizes (sensitivity wrt. β) on the resulting selection. True positives (correctly selected), true negatives (correctly omitted), false positives (falsely selected), and false negatives are colored in yellow, white, pink, and purple, respectively. There are more false negatives when the kernel size is too small (Fig. D.4, left) with a low similarity between the gray KDE cluster and the Mahalanobis ellipse. More false positives appear when the kernel size is too big (Fig. D.4, right).

D.4 Data for parameter optimization

KDE brushing, as described so far, has two not-yet-optimized parameters: α (size of the initial selection, determining the context of the local data shape analysis) and β (overall scaling parameter on top of ϕ, influencing the kernel size). To achieve an as accurate as possible brushing result, we need information about how users would use our technique to brush and what they actually intend to select as ground truth. As the interaction and brushing targets are the same as with Mahalanobis brushing [22], we can use the same user study data to optimize α and β.

In the user study [22], six diverse datasets were used. To select these datasets, we examined their scagnostics [97], aiming at a good spread of scatterplots with varying characteristics. Scagnostics are useful to characterize scatterplots and help with the

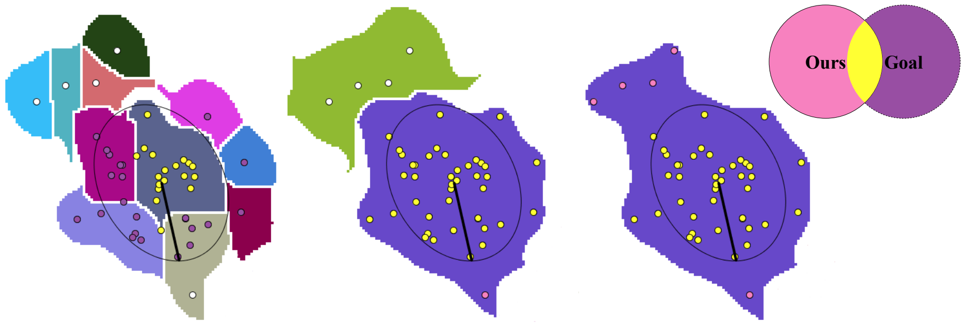


Figure D.4: Clustering based on different kernel sizes. Left: too small kernel, $s(\phi) = 0.63$; Middle: the optimal size of the kernel, $s(\phi) = 0.72$; Right: too big kernel, $s(\phi) = 0.64$.

identification of relevant structures due to density, skewness, shape, outliers, etc. For the user study, four scatterplots were chosen from Boston Housing data with 14 variables and 91 different scatterplots, so that their scagnostics were maximally different from each other [22]. The other two datasets were one with Gaussian clusters and one with path-based spectral clusters (difficult due to the bent, elongated outer cluster).

For the study, 50 participants were asked to do 12 selections each. In every case, one scatterplot (of the six) and a rather general request ("choose a large cluster", "choose a small cluster", or "choose an elongated cluster") were given to cover a reasonable variety of cases. The actual choice of what to select was left to the user. Including a request to select an elongated cluster, was also to provoke non-trivial cases, as common in real-world applications, on top of the more simple standard cases.

During the study, the user was instructed to select a target subset (as ground truth, reported by the participants using a lasso tool), before then also providing the corresponding click-and-drag interaction that this participant would use to select the target group. Accordingly, 600 brushing cases were collected from the user study (the details are attached in the supplementary material), of which we used 400 for the optimization (see below).

D.5 Optimization

To better understand the result of this parameter optimization, we also investigated the influence of parameter α with respect to the resulting selection (see Fig. D.5). The green line is the diameter of the circular area determined by the user sketch. We see that there are more false negatives (colored in purple as underselection) when α is too small. Conversely, more false positives (colored in pink when overselecting) appear, when the value of α is too big. The influence of the kernel size (scaled by β) is demonstrated in Fig. D.4. We found that α and β were the most critical parameters to optimize in our technique.

Of the 600 selections from the user study [22], we randomly chose 400 as training data. As a measure for how well the selection $\mathbf{S}(\alpha, \beta)$ agrees with the user goal $\mathbf{G}$, we

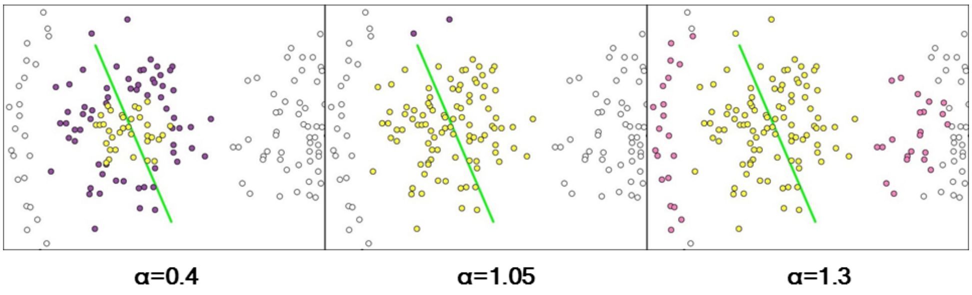


Figure D.5: The influence of different values of α: too small values lead to underselection (left; false negatives in purple), too large values lead to overselection (right; false positives in pink).

computed the dice coefficient for these two sets:

$$s(\alpha,\beta) = \frac{2|\mathbf{G} \cap \mathbf{S}(\alpha,\beta)|}{|\mathbf{G}| + |\mathbf{S}(\alpha,\beta)|} \tag{D.6}$$

If the computed selection $\mathbf{S}(\alpha,\beta)$ matches the user goal $\mathbf{G}$ perfectly, $s(\alpha,\beta) = 1$; in the case of a complete mismatch, $s(\alpha,\beta) = 0$.

Having collected the ground truth (lasso data) from the study and the click-and-release-points of the sketching interaction, we were able to conduct an optimization of α and β according to the following procedure (not involving users): Based on varying choices of α and β, we could execute our selection heuristic, using the datasets from the user study and the recorded interaction data, leading to a particular $\mathbf{S}(\alpha,\beta)$ per case—it was then straight-forward to compare $\mathbf{S}(\alpha,\beta)$ to $\mathbf{G}$ as collected during the user study, leading to a corresponding accuracy $s(\alpha,\beta)$. We started with a large matrix of different combinations of the two parameters, covering domain $[\frac{1}{10}, 10]$ that was certainly big enough. Inspecting the s-values for all these cases lead us to further examining a more detailed subset of the parameter space (basically, we refined our optimization hierarchically, doing the refinement manually). Eventually, we ended up with the following (near-optimal) values for both parameters: $\alpha = 1.05$ and $\beta = 1.05$. The optimization was done offline once.

D.6 Evaluation

Using the optimized parameters, we did an in-depth comparison in terms of accuracy, efficiency, generality and interpretability between the Mahalanobis brush [22], the CNN brush [23], and the KDE brush, using the interaction information from the user study [22] considering 252 400 points in all 600 selections.

D.6.1 Accuracy

Table D.1 shows quantitative evaluation results for the three brushing techniques, according to a number of different measures [83]:

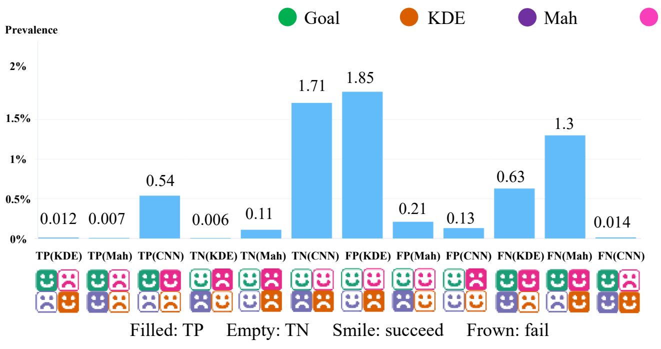
Table D.1: Quantitative evaluation of the three brushing techniques, based on a dozen measures, computed for all 600 selections from the original user study [22] (emphasizing the best result in bold):

	TP	**TP** (%)	**FP**	**FP** (%)	**TN**	**TN** (%)	**FN**	**FN** (%)
Mahalanobis	50 737	20.10%	5 189	2.06%	191 682	75.94%	4 792	1.90%
KDE	52 436	20.77%	9 583	3.80%	187 288	74.20%	3 093	1.23%
CNN	**55 321**	**21.92%**	**929**	**0.37%**	**195 942**	**77.63%**	**208**	**0.08%**

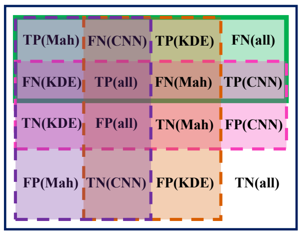
	accuracy	**recall**	**FPR**	**FOR**	**TS**	**precision**	**F_1**	**MCC**
Mahalanobis	96.05%	91.37%	2.64%	2.44%	83.56%	90.72%	91.04%	88.51%
KDE	94.98%	94.43%	4.87%	1.62%	80.53%	84.55%	89.22%	86.18%
CNN	**99.55%**	**99.63%**	**0.47%**	**0.11%**	**97.99%**	**98.35%**	**98.98%**	**98.70%**

- **TP**: true positives (correctly selected points), total number and in percent
- **FP**: false positives (falsely selected points), total number and in percent
- **TN**: true negatives (correctly left out points), total number and in percent
- **FN**: false negatives (falsely left out points), total number and in percent
- **Accuracy**: correctly selected or left out, relative to all, $(\mathbf{TP}+\mathbf{TN})/all$ (the higher, the better)
- **Recall**: how much of the goal is selected, $\mathbf{TP}/(\mathbf{TP}+\mathbf{FN})$ (higher $\leftrightarrow$ less underbrushing)
- **FPR (fall-out)**: how much of the non-goal is selected, $\mathbf{FP}/(\mathbf{FP}+\mathbf{TN})$ (lower $\leftrightarrow$ fewer **FP**)
- **FOR (false omissions)**: how much of the non-brushed was goal, $\mathbf{FN}/(\mathbf{FN}+\mathbf{TN})$ (lower $\leftrightarrow$ fewer omissions)
- **TS (threat score)**: how much of $brush \cup goal$ is **TP**, $\mathbf{TP}/(\mathbf{TP}+\mathbf{FP}+\mathbf{FN})$ (higher $\leftrightarrow$ better)
- **Precision**: how much of the selection is goal, $\mathbf{TP}/(\mathbf{TP}+\mathbf{FP})$ (higher $\leftrightarrow$ less overbrushing)
- **$\mathbf{F_1}$ score**: harmonic mean (precision, recall), $2\mathbf{TP}/(2\mathbf{TP}+\mathbf{FP}+\mathbf{FN})$ (higher $\leftrightarrow$ better)
- **MCC (Matthews correlation coefficient)**: measuring the quality of binary classification (the higher, the better), $(\mathbf{TP}\cdot\mathbf{TN} - \mathbf{FP}\cdot\mathbf{FN})/\sqrt{(\mathbf{TP}+\mathbf{FP})(\mathbf{TP}+\mathbf{FN})(\mathbf{TN}+\mathbf{FP})(\mathbf{TN}+\mathbf{FN})}$

According to the quantitative evaluation in the top part of Table D.1, CNN brushing performs best with respect to **TP**, **FP**, **TN**, and **FN**. When comparing the two empirical models, KDE brushing gives more **TP** than Mahalanobis brushing and fewer **FN**, while it leads to more **FP** and fewer **TN**. This could indicate that KDE brushing is better at recognizing the user's brushing goal, but at the cost of more false negatives (some overbrushing).



A



B

Figure D.6: A three-way comparison of four sets (A): *actual goal* (green), *Mahalanobis brush* (violet), *KDE brush* (orange), *CNN brush* (pink). The cases of *all techniques correctly brush the goal* (19.45% of all), *all techniques leave out the non-goal correctly* (73.86%), *all techniques select falsely* (0.12%) and *all techniques fail to select* (0.05%) are not shown, focusing on those cases, where at least one technique has a problem (**FP** or **FN**) and at least one technique succeeds (**TP** or **TN**); Venn diagram, illustrating the relation between the sets (B).

By looking at the bottom part of Table D.1, showing eight different measures for judging the quality of the classification, CNN brushing also outperforms the two empirical models in all measures. Comparing the two empirical models, KDE brushing seems to outperform Mahalanobis brushing in **recall** (how much of the goal is brushed) and in the **false omission rate** (how much of the non-brushed view is actually goal), while Mahalanobis brushing appears to be better in all other six measures.

In addition to comparing each method with the goal, we also did a threefold comparison to see the relation of the four related sets (actual goal, brushed by the Mahalanobis brush, the KDE brush, and the CNN brush), shown in Fig. D.6A. The Venn diagram in Fig. D.6B is an illustration of the threefold comparison: the thick green line surrounds the actual goal, the dashed violet line surrounds all that's brushed by the Mahalanobis brush, the dashed orange line surrounds what the KDE brush selects, and the dashed pink link surrounds, what the CNN brush selects (in the shown schematic, the areas do not correspond to the proportions of the respective cases).

For each point in the 600 cases from the user study [22] – 252 400 points, altogether – we check whether it belongs to the brushing goal (green), whether the Mahalanobis brush selects it (violet), whether the KDE brush selects it (orange), and whether the CNN brush selects it (pink), leading to $2^4 = 16$ possible situations per point. The relative prevalence of these situations is shown in Fig. D.6A, leaving out the dominating "good" cases of *all techniques brush a goal point*, **TP**(all), 19.45% of all, *all techniques leave out a non-goal point*, **TN**(all), 73.86%, *all techniques select falsely*, 0.12% of all cases, and *all techniques fail to select*, 0.05% of all cases, emphasizing the situations, where at least one brushing technique has a problem (**FP** or **FN**) and at least one technique succeeds (**TP** or **TN**). The label of each situation indicates its characteristics – if one technique has a problem, then this is indicated (for ex., "FN(Mah): 1.30%" indicates the situation, when only the Mahalanobis brush fails to select a goal point); when two techniques have a problem, the opposite is done (for ex., "TN(Mah): 0.11%" indicates the situation, when only Mahalanobis brushing leaves out a non-goal point, while both other techniques incorrectly select it). As an additional mark in Fig. D.6A, a combination of emojis is used to indicate the situation: a frowny indicates a problem (color: technique, filled: **FP**, empty: **FN**), while a smiley shows that the technique succeeded (filled: **TP**, empty: **TN**). Below, we briefly address all cases:

- **TP**(all), **TN**(all): in most cases, all three techniques do the right thing, i.e., select a goal point or leave out a non-goal point: 19.45% are consistently well-selected goal points, 73.86% are consistently left-out non-goal points; thus, altogether, 93.31% of all cases are "good" for all three techniques!
- **FN**(KDE), **FN**(Mah.), **FN**(CNN): the indicated brush is the only one failing to select a goal point; of these, the case where the Mahalanobis brush underselects is most prevalent: 1.30%, compared to 0.63% and 0.014% of underbrushing by the KDE and CNN brushes.
- **FP**(KDE), **FP**(Mah.), **FP**(CNN): the indicated brush is the only to falsely select a non-goal point; of these, clearly the case where the KDE brush overbrushes is most prevalent: 1.85%, compared to 0.21% (Mah.) and 0.13% (CNN).
- **TP**(KDE), **TP**(Mah.), **TP**(CNN): the indicated brush succeeds to select the goal, while the other two fail; of these three, clearly the case where only the CNN brush succeeds is most prevalent: 0.54%, compared to 0.012% (KDE) and 0.007% (Mah).
- **TN**(KDE), **TN**(Mah.), **TN**(CNN): the indicated brush succeeds in not selecting a non-goal point, while the other two select it falsely; of these, also the case where only the CNN brush is right is most prevalent: 1.71%, compared to 0.11% (Mah) and 0.006% (KDE).
- **FP**(all), **FN**(all): in these rare cases, all three techniques do the wrong thing (select falsely or fail to select), amounting to 0.12% and 0.05%, respectively.

D.6.2 Efficiency

We evaluate the actual efficiency of sketch-based brushing in two ways: the time users spend on the interaction and the computation cost of the method. The three methods adopt the same click-and-drag interaction and the average time spent is around 41% of using a lasso [22]. In terms of computation cost, CNN brushing and Mahalanobis brushing are similarly fast for small subsets. It takes, for example, around 20ms when brushing 2000 points, while the computation cost for KDE brushing is around 110ms. In the case of larger datasets, the CNN brush takes only 180ms when brushing one million points, while Mahalanobis brushing and KDE brushing take comparably long 110s and 300s for 100 000 points, respectively, which is too slow for a fluid interaction. Accordingly, all three methods enable a smooth and fluid interaction for small datasets while Mahalanobis brushing and KDE brushing are too slow for large data ($>\approx 100\,000$ points).

Table D.2: Quantitative evaluation of the three brushing techniques, based on a dozen measures, computed for all 120 selections from the new user study [23] (emphasizing the best result in bold):

	TP	**TP** (%)	**FP**	**FP** (%)	**TN**	**TN** (%)	**FN**	**FN** (%)
Mahalanobis	18 989	21.90%	1 549	1.79%	65 921	76.03%	241	0.28%
KDE	18 927	21.83%	1 859	2.14%	65 611	75.68%	303	0.35%
CNN	**19 100**	**22.03%**	**1 286**	**1.48%**	**66 184**	**76.34%**	**130**	**0.15%**

	accuracy	**recall**	**FPR**	**FOR**	**TS**	**precision**	**F$_1$**	**MCC**
Mahalanobis	97.94%	98.75%	2.30%	0.36%	91.39%	92.46%	95.50%	94.25%
KDE	97.51%	98.42%	2.76%	0.46%	89.75%	91.06%	94.60%	93.10%
CNN	**98.37%**	**99.32%**	**1.91%**	**0.20%**	**93.10%**	**93.69%**	**96.43%**	**95.44%**

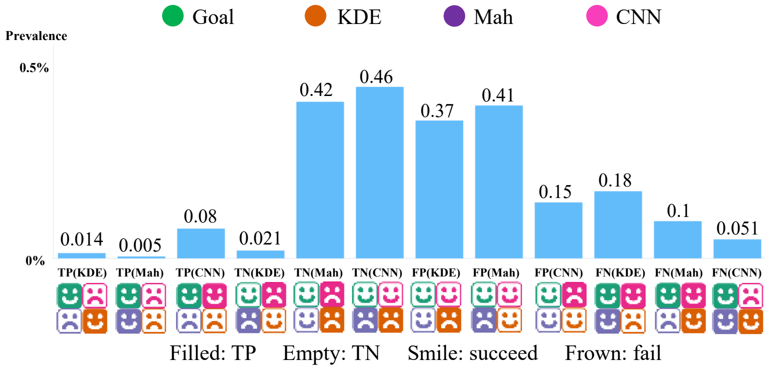


Figure D.7: Threefold comparison between the Mahalanobis brush, KDE brush and the CNN brush based on the user's goal from a follow-up user study (details in the text). Note that all relative prevalences are below 0.5%.

D.6.3 Generality

Generality is used to describe a model's ability to react to new and previously unseen data. To further substantiate the evaluation of the three techniques in terms of generality, we also tested the three methods on new data from another user study [23], which has not been used for training of the models at all. This user study used six new datasets (including compound data, personal hiking data, aggregation data, the omnipresent Iris data, R15 data and spirals shape data) and 10 users provided 12 selections each, leading to 120 selections in total (details in the supplementary material). The corresponding quantitative comparison is shown in Table D.2 and Fig. D.7, considering 86 700 points in total.

Comparing the quantitative evaluation based on the two user studies, we see that CNN brushing produces many more **FP** in the follow-up study (0.37% → 1.48%), leading also to a higher fall-out value (0.47% → 1.91%). Mahalanobis brushing and KDE brushing produce much fewer **FN** in the follow-up study (1.9% → 0.28% and 1.23% → 0.35%, respectively), while CNN produces more **FN** (0.08% → 0.15%). With respect to the other measures, KDE's **threat score** got better in the follow-up study (80.53%

D

$\rightarrow$ 89.75%) and became more similar to the others. Mahalanobis' **threat score** improved also (83.56% $\rightarrow$ 91.39%), while CNN's **threat score** worsened (97.99% $\rightarrow$ 93.1%). Besides that, Mahalanobis' **recall** got better in the follow-up study (91.37% $\rightarrow$ 98.75%) and became more similar to the others (all methods are very good). CNN's **precision** and **accuracy** went down in the follow-up study (98.35% $\rightarrow$ 93.69%, 99.55% $\rightarrow$ 98.37%) and became more similar to the others. In addition, the **FPR** and **FOR** of KDE brushing are both largely reduced (4.87% $\rightarrow$ 2.76%, 1.62% $\rightarrow$ 0.46%), becoming more similar to the other two methods. In terms of the **F_1 score** and **MCC**, both Mahalanobis and KDE brushing improved (91.04% $\rightarrow$ 95.5%, 88.51% $\rightarrow$ 94.25%; 89.22% $\rightarrow$ 94.6%, 86.18% $\rightarrow$ 93.1%), while CNN brushing got worse (98.98% $\rightarrow$ 96.43%, 98.7% $\rightarrow$ 95.44%).

For the threefold comparison, we see a performance decline of CNN brushing in **TP**(CNN) (0.54% $\rightarrow$ 0.08%), **TN**(CNN) (1.71% $\rightarrow$ 0.46%), **FP**(CNN) (0.13% $\rightarrow$ 0.15%) and **FN**(CNN) (0.14% $\rightarrow$ 0.51%). For the empirical models, KDE is better in **TP**(KDE) (0.12% $\rightarrow$ 0.14%), **TN**(KDE) (0.06% $\rightarrow$ 0.21%), **FP**(KDE) (1.85% $\rightarrow$ 0.37%) and **FN**(KDE) (0.63% $\rightarrow$ 0.18%), while Mahalanobis brushing is better in **TN**(Mah) (0.11% $\rightarrow$ 0.42%), and **FN**(Mah) (1.3% $\rightarrow$ 0.1%) but not in **TP**(Mah) (0.07% $\rightarrow$ 0.05%), and **FP**(Mah) (0.21% $\rightarrow$ 0.41%).

While almost all measures got better for both Mahalanobis brushing and for KDE brushing in the second user study, they all got worse for CNN brushing – at least a bit. It is important to see, however, that CNN brushing still outperformed both other methods in all indicators (even though they are much more similar in the follow-up study). This could reveal one disadvantage of the CNN brush, namely that it is less general, when compared with the empirical models.

D.6.4 Interpretability

Interpretability refers to how much a human can understand the model's process and result. Although the three brushing methods achieve all good results (>90% accuracy) in two studies, we were curious to see how the methods perform in particularly difficult cases (for example, when brushing the large ring shapes in the original study or the spiral shapes in the follow-up study). Therefore, we had a closer look at special cases and found that the average accuracy for brushing the ring shapes are around 53%, 51% and 95% for KDE brushing, Mahalanobis brushing and CNN brushing, respectively, while they achieve 54%, 50% and 41% accuracy when selecting the spiral shapes. We can see that KDE brushing is slightly better than Mahalanobis brushing in these cases due to its nonlinear model while the performance of CNN brushing falls in the follow-up cases, possibly because of issues related to overfitting.

In addition to the "bad cases" analysis, we observed that some prediction results of CNN brushing are unreasonable. CNN brushing can, for example, accurately find the border of a cluster while some scattered points inside the cluster are not selected. This situation is not possible in Mahalanobis brushing and KDE brushing, representing an increased uncertainty and low interpretability of the DL-based model.

D.6.5 Summary

In general, we could not see that KDE brushing would significantly outperform Mahalanobis brushing based on the comparison between KDE brushing and Mahalanobis brushing, even though we see slightly better results for KDE brushing in the follow-up study and in some nonlinear-shape cases. An according assumption was originally made, because we thought that more carefully considering the local data distribution should help to further improve the technique's accuracy (as a nonlinear method, KDE brushing should have much better abilities to adapt to nonlinear structures in the data). So far, we cannot rule out that we have overlooked another limitation when realizing the KDE-based approach – either a conceptual one, or a limitation of our implementation. Accordingly, we see it still possible that another solution could achieve a further improved accuracy. To outperform CNN brushing, however, seems like a tall order.

We note that empirical modeling comes with the advantage of an explainable result (for example, we know how different values of α and β influence the results), while the excellent performance of the DL-based model comes at the cost of a poor interpretability (including some uncertainty concerning the stability of its predictive power). This comparison leads to the interesting question of how much accuracy we are willing to sacrifice for a good interpretability.

D.7 Conclusion and future work

In this paper, we presented our attempt to improve Mahalanobis brushing by incorporating kernel density estimation to increase its accuracy. Although more information is taken into account for modeling the KDE-based model, we have not seen a significant improvement compared to the simpler Mahalanobis brush. Based on this result, we think that the increased cost of incorporating KDE could have come with an overdesign issue. When compared with deep learning, we found that its black-box nature results in a questionable interpretability (but with excellent accuracy), whereas the results based on the empirical model are explainable (even though not as good as the ones based on the learned model). Considering its reduced robustness, the DL-based method appears to be (a bit) less stable and a bit more unpredictable, even though it does have the best performance, after all. It is unclear, however, how to weigh in all factors, when comparing the overall performance for model selection.

In the future, we see several opportunities to further extend our work, including:

- Combining advantages of both sides, i.e., empirical modeling *and* deep learning. We imagine, for example, to automatically learn the kernel size or to design the deep learning input on the basis of the KDE.
- Investigating more closely, why KDE brushing did not outperform the Mahalanobis brush, and make a new attempt to further improve it.
- A sensitivity study with respect to the (optimized) parameters of the empirical models.
- Exploring other machine learning approaches to develop a new brushing technique which outperforms CNN brushing.

We hope, also, that this work can inspire further related research, especially in visualization for model design and model selection.

D.8 Acknowledgements

We thank the participants in the user studies for helping with our research. Parts of this work have been done in the context of CEDAS, Center for Data Science, at the University of Bergen, Norway.

Paper E

Sketch-based fast and accurate querying of time series using parameter-sharing LSTM networks

Chaoran Fan[1], Krešimir Matković[2] and Helwig Hauser[1]
[1] University of Bergen, Norway
[2] VRVis Research Center, Austria

Abstract

Sketching is one common approach to query time series data for patterns of interest. Most existing solutions for matching the data with the interaction are based on an empirically modeled similarity function between the user's sketch and the time series data with limited efficiency and accuracy. In this paper, we introduce a machine learning based solution for fast and accurate querying of time series data based on a swift sketching interaction. We build on existing LSTM technology (long short-term memory) to encode both the sketch and the time series data in a network with shared parameters. We use data from a user study to let the network learn a proper similarity function. We focus our approach on perceived similarities and achieve that the learned model also includes a user-side aspect. To the best of our knowledge, this is the first data-driven solution for querying time series data in visual analytics. Besides evaluating the accuracy and efficiency directly in a quantitative way, we also compare our solution to the recently published Qetch algorithm as well as the commonly used dynamic time warping (DTW) algorithm.

This article was published in IEEE Transaction On Visualization and Computer Graphics, Early Access, 2020

E.1 Introduction

In the emerging era of big data, extensive time series data are common in a large variety of application domains. The visualization of such data is often cluttered, especially when the trend is non-periodic and the data size is large. In the exploration of long time series data, it is often hard for the analysts to visually identify specific patterns efficiently. To overcome this issue, the topic of finding relevant parts of time series data has become popular in recent research.

In general, it is easier to visually describe patterns in time series data than to express them textually or procedurally. Therefore, visual query systems are a convenient user interface with freehand sketching as an efficient means for visual communication. The use of sketching enables the analyst to convey complex free-form patterns of interest, which are matched against the data to identify subsets of interest.

For matching sketches and data, usually a carefully designed, empirical model is adopted to estimating the similarity between the sketch and the time series data. Often, this approach comes with non-optimal efficiency and accuracy, having so far also resulted in a limited deployment of sketch-based visual query systems for real-world visual analytics applications. More specifically in terms of their limited efficiency, most of these empirical methods are based on local characteristics and a sliding window (of the same length as the sketching query) that is used to compute the best match or a similarity ranking, generally leading to a time-consuming comparison procedure that can hamper the interactive exploration. On the other hand, sketches are artistic expressions and due to ambiguity and inaccuracies in sketches, an empirical model is often quite far from robustly representing the underlying ideas and expectations of the user. This can lead to matching algorithms that fail to produce good similarity rankings, especially when "goodness" is evaluated by humans [71].

To improve sketch-based querying, we see two main directions. First, in order to secure a fluid data exploration, we aim at a fast computation of the matching procedure. Second, we need a better understanding of the user's intention given her/his sketch—only this way we can make the querying result as close as possible to what the user really needs. Due to great recent success, deep learning in computer vision [86], image classification [59], and natural language processing [74, 94] has attracted a lot of attention. As pattern matching in time series data is somehow similar to detecting patterns in images, we expected that deep learning would boost the performance of matching solutions.

In this paper, we now show a successful exploitation of the long short-term memory (LSTM) architecture [4] to encode the sketch and the time series data respectively in two networks with shared parameters. In principle, two LSTM networks with different parameters could be used to learn the representation of the sketch and the time series data. In our design, the two networks share the same parameters and this parameter sharing helps with accelerating training and limiting overfitting. The networks are trained based on perceived similarities from a user study. This way, we integrate the user's perception into the learned model. As no existing model is capable of fully capturing the complex semantics of a user's sketch, we saw a great potential to improve the situation by learning the matching model directly from users. We demonstrate the effectiveness of our method in comparison with two state-of-the-art matching models—the recently presented Qetch algorithm [71] and the seminal DTW technique (dynamic time warping) [90].

Overall, the main contributions of our paper are:

- **A data-driven method for sketch-based querying of time series data.** To the best of our knowledge, this is the first time that deep learning is used to learn the matching relation between a human sketch and time series data, outperforming two state-of-the-art models in terms of accuracy and efficiency.

- **A sketch-based querying system for time series data.** We present a prototype of a sketch-based querying system for time series data. We offer the user an opportunity to use a freehand sketch to explore the time series data interactively without the need to set any offline parameter.

E.2 Related work

Sketching is a natural and expressive type of interaction, which has been frequently used in the visualization area, especially as a brushing technique [22–25, 80] and in visual query systems [13, 42, 71, 78, 104].

In the following, we provide a brief introduction to common time series similarity matching algorithms, followed by a detailed overview of prior work related to visual query systems for time series data and visualization applications based on deep learning knowledge.

E.2.1 Time series data similarity

Among a variety of similarity measures, the Euclidean distance (ED) and dynamic time warping (DTW) [90] are the most commonly used measures with the squared ED being the sum of the point-wise squared differences of the two time series. The basic ED can be improved by data normalization, often standardization, which considers the variation of similar patterns in amplitude and y-offset[30]. Since ED is computed point-wise and the mapping of a query point to a data point is fixed, it is sensitive to noise and local time misalignments.

DTW overcomes ED's inability to handle local time misalignments (or warps) by allowing horizontal stretching (or compression) of a time series when searching for similar data subsets. Therefore, DTW is considered to yield better fits for shape matching, especially when the similar shapes are not aligned along time.

For matching a sketched query and time series data, both ED and DTW require a sliding window of size equal to the query length to compute the similarities over time. In their survey, Ding et al. [16] conclude that there is no distance measure that is systematically better than DTW, while the relatively simple and straight-forward ED can be computationally competitive with DTW, when the size of the data increases.

E.2.2 Visual query systems

In visual query systems, visual interface components are used to formulate the user's queries. TimeSearcher [37] was a pioneering information visualization tool using time-boxes to query time series data. The analyst draws a rectangular region to indicate time points of interest on the time axis and the range of interesting values on the value axis. Time series data is then highlighted while passing through the timeboxes. Later, extended versions have been proposed to improve the basic timeboxes by incorporating the variable (fuzziness in the boundaries) [51], angular queries and slopes to search ranges of differentials [38] and supporting more flexibility with options to adjust the query [7]. Overall, timeboxes are powerful value-based widgets and they are used in several visual query systems. Still, it is far from straight-forward to specify a shape-based query with timeboxes, for example, a head-and-shoulders pattern.

The Querylines system [89] realizes a filter-based approach to visual querying. It offers the user the opportunity to specify constraints by using line segments. The analyst can qualify these line segments as hard or soft constraints based on their preference. If the query gets over-constrained, feedback from the system enables the users to refine the query specification.

An alternative technique for constructing visual queries is to first identify common shapes such as a spike, sink, rise, drop, plateau, and valley, and then build queries using these basic shapes as pattern templates [32].

The concept of a sketch-based visual query system was first proposed by Wattenberg [104]. In his approach, the analyst sketches an approximate pattern on the same display where also the data is visualized for searching similar patterns. The similarity to the time series data is calculated as simple ED. The system is straight-forward to use, but the quality of matching relies strongly on details and well-defined time and amplitude ranges of the sketch, which is in general not easy for the user to handle.

To improve the flexibility and tolerance in their sketch-based visual query system, Holz and Feiner [42] provided a relaxed selection technique which allows the user to implicitly indicate a level of similarity that can vary across search patterns during sketching. Specifically, the mouse speed is used to inform the system about the spatial and temporal tolerance of points in the sketched query.

In order to study the human perception of correspondence between sketches and time series patterns, Eichmann and Zgraggen presented a comparison of rankings of computed pattern matches with human-annotated results [19]. They found that human-annotated rankings can differ drastically from algorithmically generated rankings and concluded that the meaning of sketching is too diverse to be captured in one algorithm or metric.

As a multitude of queries can be targeted by the same sketch, Correll and Gleicher [13] investigate the ambiguities of sketch-based query systems in time series data and define a set of “invariants”, enabling the user to choose the properties of data to ignore while sketching. In addition, they adapt different matching algorithms to support different invariants. The main drawback of this approach is that it is not easy and straight-forward for the user to think about the invariants while doing data exploration.

Muthumanickam et al. [78] outline important perceptual features for effective shape matching and define a grammar to express time series data approximately by considering the data as a combination of basic elementary shapes positioned across different amplitudes. These basic shapes are represented by using a ratio value and then a symbolic approximation can be achieved by performing binning on ratio values. The major problem of this method is the limited query expressiveness, along with the black-box nature of query execution with each shape often having its own processing or matching steps.

Research on visual perception suggests that we mentally decompose complex shapes into salient parts such as piece-wise upward or downward lines, peaks and troughs [40, 52, 55]. Based on this research, Mannino and Abouzied present Qetch [71], a tool where users freely sketch patterns on a scale-less canvas to query time series data and get rid of specifying query length or amplitude. This method claims its advantage (dealing with the scale-less sketch) over the traditional matching algorithms—ED and DTW. However, in our observation, the query result is very sensitive to the smoothing level of the time series data as the query length is based on the salient parts (constructed by extrema and inflection points) of the data.

E.2.3 Deep learning for visualization

In recent years, deep learning has become popular due to its successful application to a wide range of fields, especially in image processing and natural language processing. In the visualization area, according research focuses on helping with the design, training, diagnosis and refinement of deep learning models [41, 67, 109]. Using deep learning for solving visualization tasks, however, is still rare.

Han et al. [33] presented FlowNet, an approach based on an autoencoder, for improving clustering and the selection of streamlines and stream surfaces. Kim and Gunther [53] extract a robust reference frame based on a convolutional neural network (CNN) that is able to yield a steady reference frame for a given unsteady 2D vector field. Hu et al. [44] introduced VizML that predicts visualization design choices from a large corpus of datasets using neural networks. Data2Vis [14] makes use of recurrent neural networks to generate Vega-lite visualization specifications from JSON-encoded datasets.

Further, we see visualization solutions that leverage deep learning to improve techniques in visual analytics, for example interaction techniques. Fan and Hauser [23] exploited a CNN and modeled sketch-based brushing in scatterplots to predict the selected points. This method achieves state-of-art accuracy while providing a fast interaction. The model is trained on data from different users in a user study, leading to a general model that is thus not optimized to every single user. To address this issue, they presented a personalized CNN-based brushing technique that is able to iteratively refine the brushing model for a single user with additional data that he/she provides while using the brushing technique [25].

More recently, Chen et al. [10] developed a learning-based approach to realize a lasso selection of 3D points by modelling the selection as a latent mapping from viewpoint and lasso to point cloud regions.

E.3 The principal approach

The overall goal of our research was to design a visual query system for time series data with a fast interaction and an accurate query result in order to solve efficiency and accuracy problems of existing solutions. Also, the system was expected to be friendly to the non-expert and easy to use with limited training. Figure E.1 shows an illustration of the principal approach. To achieve a swift interaction, we use freehand sketching as the querying input and a similarity function S that is capable of interpreting the

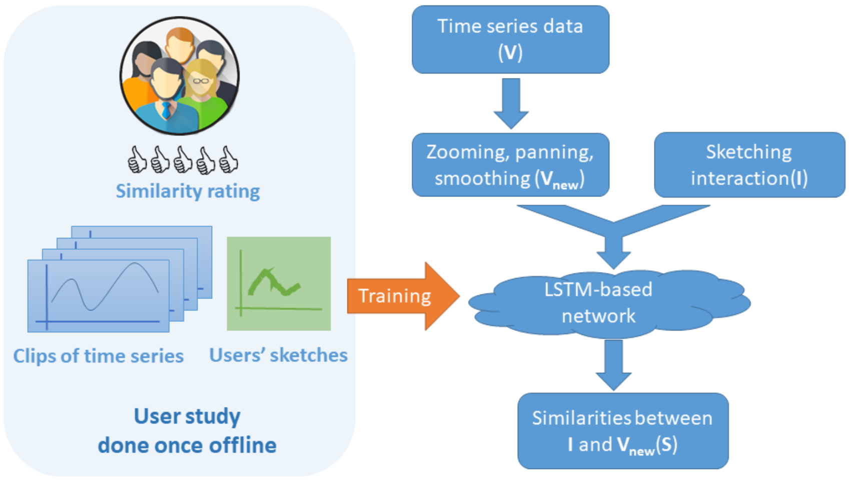


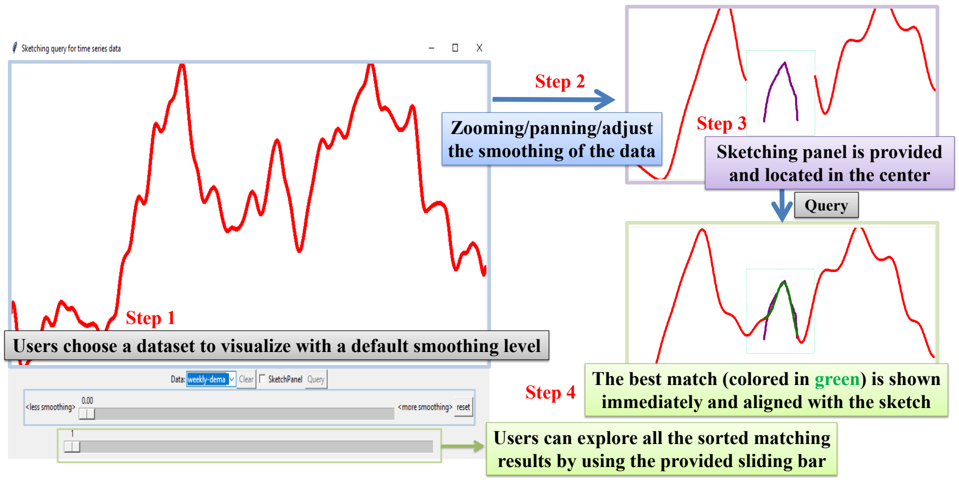
Figure E.1: Illustration of our principal approach: users specify the targeted scale of the time series data by zooming, panning or smoothing, then freely sketch an approximate pattern on the sketching panel. Then a similarity rank between the user sketch and the processed time series data is computed by the proposed parameter sharing LSTM networks. The network is trained only once offline based on user study data.

relation between the human sketch I and the matching goal in the time series data V as accurately as possible. Further, we aimed at a real-time system, meaning that the computational cost should be minimal, as well.

Based on our understanding that all empirical models have their limitation at estimating the intended meaning of a human sketch, we found it promising to exploit learn the needed similarity measure directly from users. Recurrent neural networks (RNNs) are a straight-forward solution for encoding time series data and to do the matching for two reasons: 1., RNNs have a memory which allows the model to keep information about its past computations. This enables RNNs to have dynamic temporal behavior, which naturally fits to sequential data like time series data. 2., An advanced version of RNNs, LSTM networks (long short-term memory), can be trained to remember the information from a specific length of past times steps. This mechanism can be used to mimic a sliding window while doing the matching computation along the time series data. At the same time, it avoids reading the same data repeatedly, leading to a relatively low computation cost.

To construct the network structure, we used a pair of LSTM networks with shared parameters to encode the sketch and the time series data, respectively. The sharing of the network parameters was beneficial because of the high similarity between the sketch and the time series. The thereby reduced overall number of parameters accelerates the training procedure and helps with preventing overfitting. This design is inspired by the "Siamese" network-based solution for sentence similarity [74]. Detailed description of our network is given in section E.4.3.

To train this pair of networks, we collected data from two user studies. For the first user study, we gathered the ground truth about how different users sketch patterns and



Sketching query for time series data
Step 1
Users choose a dataset to visualize with a default smoothing level
Data: weekly-dema
Clear
SketchPanel
Query
<less smoothing>
0.00
<more smoothing>
reset
1
Step 2
Zooming/panning/adjust the smoothing of the data
Step 3
Sketching panel is provided and located in the center
Query
Step 4
The best match (colored in green) is shown immediately and aligned with the sketch
Users can explore all the sorted matching results by using the provided sliding bar

Figure E.2: The interface of our sketching system for time series data. To explore, users choose data first and interact with the data then to specify the scale of interest by zooming and panning. Then the user sketches a pattern of interest. A matching rank is computed and results are explored.

how they rate similarities based on their visual perception of correspondences between their sketches and several clips of the time series data that we offered. In the second user study, we examined the variation of the user's sketches in order to use this for modeling an extension of the training data for a more stable training.

In the following, we introduce the four basic steps of our data exploration workflow (technical details are provided in section E.4.1). Figure E.2 illustrates the user interface of our proposed sketching system and the four steps to data exploration.

Step 1: Data Preprocessing. For most time series data, some smoothing is necessary to capture the key patterns of interest, leaving out noise and patterns on other scales. In our design, cubic splines are used to smooth the data. Instead of asking the user to specify the smoothing level, we offer a default smoothing level after loading the dataset.

Step 2: Interactive Scaling and Smoothing. Choosing a scale (and a smoothing level) for data exploration is a crucial user-side task – meaningful questions may be asked about time series data at multiple scales, depending on the user task. Instead of iterating through all possible scales and smoothing levels while matching, we allow the user to interact with the data via zooming and panning (and/or adjusting a slider to specify the targeted scale and smoothing level), after initially estimating a proper scale automatically.

Step 3: Sketching. Once the scale and smoothing is determined, a sketch panel is provided for the user to do free-hand sketching. The empty sketch panel is located at the center of the canvas to let the user sketch at the targeted scale, visually referring to the scaled time series data in the background. Sketches are then slightly smoothed in order to remove hand jitter and the query length is determined by the sketch length.

Step 4: Query and explore the matching results. The two parameter-sharing

LSTM networks are then executed to obtain an ordered set of similarities between the sketch and subsets of the time series data. The best match is immediately highlighted in green after the computation and the corresponding time series data is shifted by aligning the best matching part with the sketch. Moreover, a slider is provided to explore all the other results from the ranked list of matching results.

E.4 Technique in detail

In the following, we first go through the details of scaling and smoothing before we then describe the specification of the used RNN and the design of the proposed network.

E.4.1 Scaling and smoothing the data

As we mentioned, a default smoothing level (denoted by k_0) is computed for the data after loading, based on the number of salient parts (denoted by N_s). We count the salient parts by segmenting the time series data at extrema and inflection points. To obtain the default smoothing level for each dataset, we adjust the smoothing level until we are satisfied with the number of salient parts that were enough to represent the time series data in advance and this smoothing level is then chosen as the default smoothing setting in the beginning.

To represent the scale of the data, we use z with $z = 1$ in the beginning. We assume that the user wants to see more details when zooming in (and vice versa when zooming out). To automatically adjust the smoothing level during zooming, a linear function is used to adapt the smoothing according to the (logarithm of the) scale: $k(z) = k_0 - a \cdot \ln z$, where a is a coefficient that we obtained via a simple regression procedure. Specifically, we used that the levels of details are related to the number of salient parts. A correctly chosen a should lead to a stable number of salient parts while zooming in or out. To achieve this, we randomly choose 10 points in the time series data for a specific value of a and then compute the number of salient parts with 10 different scales. The results were used to fit a linear function with h as the coefficient ($N_s = h \cdot \ln z + d$). This procedure was repeated several times by trying different values of a. This way, we identified a well-working a by finding h which was closest to 0. In our work (10 different datasets), the value of a varied from 50 to 1371, while k_0 ranged from 9054 to 549923. Doing this offline in advance (finding a and k_0 for each dataset), we can minimize the operations that the users have to do and help them focus on the pattern searching in the time series data. To increase the flexibility, the user can also use a slider to adjust the smoothing level, if they are not satisfied with the suggested smoothing level.

E.4.2 Recurrent neural network (RNN)

An RNN is an extension of the traditional feed-forward neural network which is able to store relevant parts of the input and use this information to predict future outputs. More formally, at time step t, the memory cell's current hidden state $\mathbf{h}_t$, preserved by the RNN structure, is a function of the input at the current time step ($\mathbf{X}_t$) and the hidden state at the last time step ($\mathbf{h}_{t-1}$). The RNN updates its current state by computing

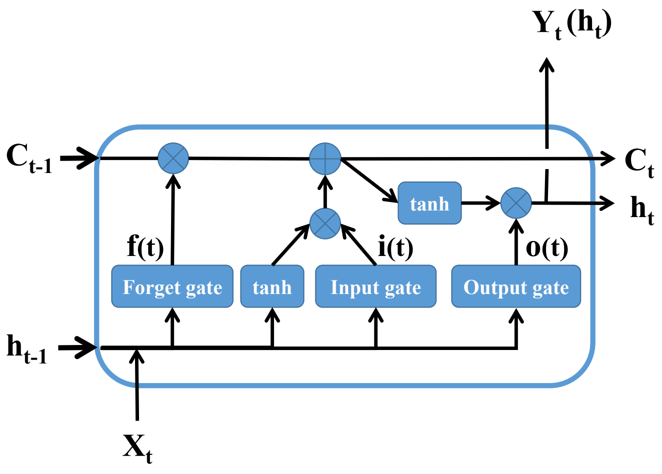


Figure E.3: Complete structure of the LSTM cell, which can process data sequentially and keep its hidden state through time.

$\mathbf{h}_t = \phi(\mathbf{h}_{t-1}, \mathbf{X}_t)$, where ϕ is a nonlinear function such as the composition of a logistic sigmoid with an affine transformation. Optionally, the output at time step t, denoted by $\mathbf{Y}_t$ is a function of the previous state and the current input, and it is the same as the hidden state $\mathbf{h}_t$ for basic cells.

Although a basic RNN performs well in capturing nonlinearity in time series data, it was observed that back-propagation dynamics caused the gradients in an RNN to either vanish or explode while training to capture the long-term dependencies [4].

To overcome this disadvantage, the LSTM (long short-term memory) architecture [39] was proposed by Hochreiter and Schmidhuber. As shown in Figure E.3, in addition to the hidden state vector $\mathbf{h}_t$, LSTMs also maintain a memory cell $\mathbf{c}_t$ at time t. At each time step, the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms. The memory cell $\mathbf{c}_t$ is updated by partially forgetting the existing memory and adding new memory content:

$$\mathbf{c}_t = f(t) \otimes \mathbf{c}_{t-1} + i(t) \otimes \tanh(\mathbf{W}_c \mathbf{X}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{E.1}$$

where the forget gate $f(t)$ controls the extent to which the existing memory should be erased while the input gate $i(t)$ is used to decide the degree to which the new memory content is added. The two gates are computed respectively by

$$f(t) = \sigma(\mathbf{W}_f \mathbf{X}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{E.2}$$

$$i(t) = \sigma(\mathbf{W}_i \mathbf{X}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{E.3}$$

Moreover, the output gate $o(t)$ controls the exposure of the memory content and it is computed by

$$o(t) = \sigma(\mathbf{W}_o \mathbf{X}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{E.4}$$

As a last step, the output of the LSTM unit ($\mathbf{Y}_t (= \mathbf{h}_t)$) at time step t is be obtained by

$$\mathbf{h}_t = o(t) \tanh(\mathbf{c}_t) \tag{E.5}$$

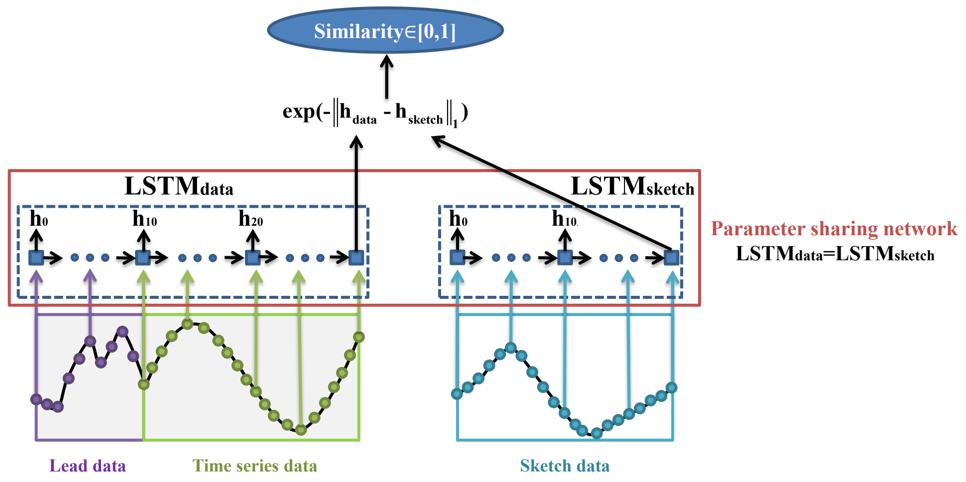


Figure E.4: The structure of our proposed double network: the time series data (green) with lead data (purple) on the left and the sketch (blue) are encoded by the two parameter-sharing LSTM networks, which are trained against a distance metric based on the Manhattan distance.

In all of the above, operator $\otimes$ is the Hadamard product (entry-wise product). $\mathbf{X}_t \in \mathbb{R}^d, f(t), i(t), o(t), \mathbf{h}_t, \mathbf{c}_t \in \mathbb{R}^h$. The weight matrices $\mathbf{W} \in \mathbb{R}^{h \times d}$ and $\mathbf{U} \in \mathbb{R}^{h \times h}$ and the bias vector $\mathbf{b} \in \mathbb{R}^h$ are learned during training. The dimensions d and h correspond to the number of input features and the number of hidden units, respectively.

Instead of overwriting its content at each time step, an LSTM unit is able to decide whether to store or retrieve the existing memory via the introduced gates. The activations of these gates are based on the sigmoid function and hence range smoothly from 0 to 1 (not at the least to keep the model differentiable). Intuitively, if the LSTM unit detects an important feature from an input sequence at an early stage, it carries this information (the existence of the feature) over multiple steps, capturing potential long-distance dependencies.

E.4.3 Network design

Figure E.4 provides an overview of our proposed network structure for estimating the similarity between the user's sketch and the time series data, composed of two LSTM networks: $LSTM_{\text{data}}$ and $LSTM_{\text{sketch}}$, sharing their parameters.

In each time step, the hidden state (h) has to be carried as an input to the next time step. For the similarity computation, we only consider the final representation of both the time series data and the sketch, encoded as h_{data} and h_{sketch}, respectively. To compute the similarity between these two vectors, we use a metric based on the Manhattan distance, which can be defined as $\exp(-\| h_{\text{data}} - h_{\text{sketch}} \|_1) \in [0,1]$. The reason to choose an L1 norm for the similarity computation is that an L2 norm can lead to undesirable plateaus in the overall objective function due to vanishing gradients of the Euclidean distance [12]. During the training, the network learns how the predicted similarity between h_{data} and h_{sketch} deviates from the user-annotated ground truth.

In order to compute the similarities along the time series data, we mimic a sliding window by making use of the special feature of the LSTM network that in each time step it can choose to forget a part of the information extracted from the previous time steps. In the training, which we explain in more detail further below, the network is trained to force the output of each time step to represent the information only for a specific number (L) of previous time steps. Figure E.4 shows a typical example, where the sketch data is sampled as 21 “blue” points, determining the size of the sliding window ($L = 21$). Therefore, after a proper training, h_{data} only contains the information of the 21 “green” points and the influence of the previous 10 “purple” points are forgotten. As the output of each time step can be trained to contain the information of a certain previous time steps, our method only needs to iterate the data once and then interpret the output of each time step for the matching computation, which is much more efficient than the traditional sliding window, which needs to access a data point several times while moving.

E.4.4 Training the network

We define the training data as $([L_i, R_i] \in T_{in}, y_i \in T_{out})_{i=1}^{N}$ as pairs of input and expected output (N is the number of training samples). L_i contains the time series data and its synthesized left lead data while R_i is the corresponding user sketch. The reference output y_i is the human annotated similarity between L_i and R_i, which the model is trained against. We optimize the parameters of the network based on the training data using the mean-squared error as a loss function.

During training, no hyper-parameter explicitly “tells” the network to learn the information from a certain length of previous time steps in each time step. The mechanism for the LSTM to forget earlier data is when the network finds that the information carried by certain previous time steps is important, while the information before that is not. To teach the network to “understand” this, we add some synthesized data of half the length of the query to the left of the time series data (for example the purple points in Figure E.4). The synthesized data is chosen from other parts of the same data, where the clip of the time series data is extracted from and then smoothly connected to the left of the time series data. For one pair of time series data and a sketch, we add 10 different synthesized lead data to the left of the time series data, which has been tested to make sure that the network can recognize the real data and force itself to forget the influence of the synthesized data. In this way, the network is trained to only remember a specific length of earlier data in each time step. Based on this, we can compute the similarity rank along the time series data while reading the time series only once.

High accuracy cannot be achieved without providing enough training data. It is labor-intensive and time-consuming to invite a large number of users to provide a large amount of user data. Instead, we follow a common strategy and synthesize additional training data from the already acquired training set by modeling the natural variation of user sketches. In the following, we describe how we synthesize the left lead data of the time series data and the variation of user sketches in detail.

E.4.5 Training data augmentation

In addition to the training data that we acquired by a user study, we employ two strategies to augment the training data: First, we synthesize lead data ahead of the actual

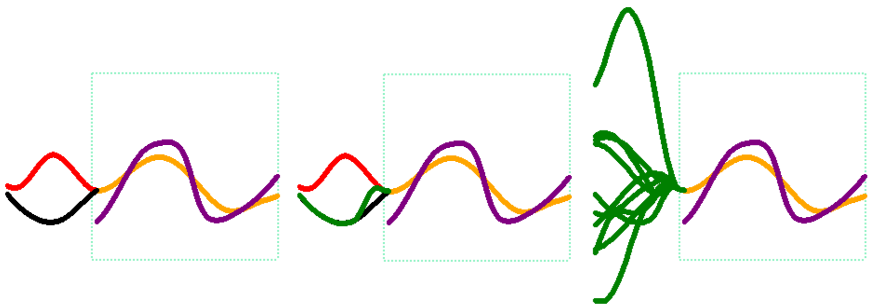


Figure E.5: Lead data synthesis. Left: time series data (orange) with actual lead data (red), and the user's sketch (purple). New lead data (black) is randomly chosen. Middle: the new lead data is smoothly attached (green) to the time series data (orange) by interpolation. Right: ten instances of synthesized lead data.

time series training data in order to teach the network the actual query length. Second, we generate variants of the sketch, based on a second user study that informed us about the natural variation of the user's sketching interaction.

Lead data synthesis

To preserve the character of the time series data, we initialize the newly synthesized lead data with randomly chosen snippets from the original time series data connected them to the left of the time series training data. Figure E.5 shows the whole procedure of lead data synthesis: The user's sketch is shown in purple and the corresponding time series data clip in orange with its actual lead data in red (denoted as $r(t)$). New lead data (black, denoted as $b(t)$) is chosen randomly from the time series data, forming the basis of the newly synthesized lead data.

To start, we randomly choose another part of the original time series data (with the targeted length). We do so to maintain the overall character of the time series data when synthesizing new lead data. Obviously, this usually leads to a non-smooth connection with the actual time series data (illustrated on the left in Figure E.5).

To achieve a natural, smooth concatenation, we designed a simple, fifth-order polynomial weighting function $w(t)$ to merge $r(t)$ and $b(t)$ by convex combination. We set up $w(t)$ to fulfill six constraints: $w(0) = 0$, $w(1) = 1$, $w'(0) = w'(1) = 0$, and $w''(0) = w''(1) = 0$, leading to

$$w(t) = 6t^5 - 15t^4 + 10t^3 . \tag{E.6}$$

Given that we wish to adapt the last $n+1$ values of the randomly chosen new lead data $b(t)$ (of length m) to smoothly connect to the following time series data, we compute

$$b_{\text{new}}[i] = (1 - w(t))\, b[i] + w(t)\, r[i] \tag{E.7}$$

for $t = \frac{i-(m-n)}{n}$ and $i \in [m-n, m]$, keeping $b_{\text{new}}[i] = b[i]$ for $i < m-n$. Adapted $b_{\text{new}}[i]$ then smoothly connects to the following time series data in $i = m$.

In our experiment, the randomly chosen part is sampled into $m = 20$ points and the last $n = 8$ points are merged with the real lead data. In the middle of Figure E.5, r is

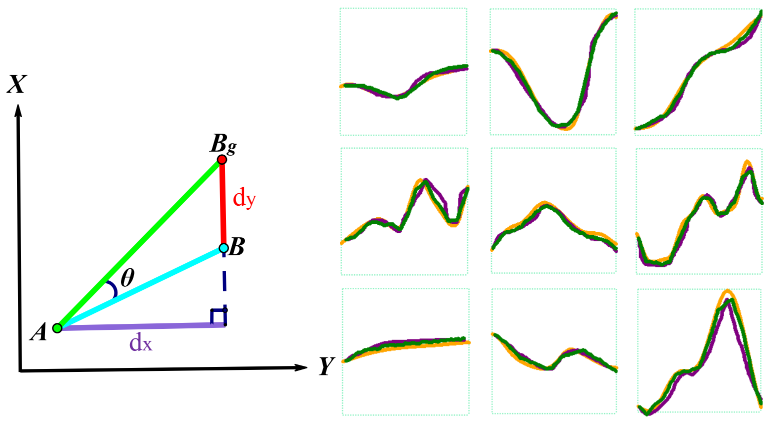


Figure E.6: Sketch synthesis. Left: illustration of the modeling base between two consecutive points A and B in the sketch and the corresponding goal point B_g. Right: 9 modeled interactions (colored in green) according to the specific querying target (orange curve) and the original user sketch (purple curve).

shown in red, b in black, and b_{new} in green. On the right of the Figure E.5, 10 different pieces of lead data are generated for one pair of sketch and time series data, all smoothly connected to the time series data (orange). This approach leads to a good variation of synthesized lead data, mimicking plausible cases for all time series data that we worked with. Synthesizing ten instances of substantially varying lead data allows the LSTM to learn that only the actual time series data (orange) is to be taken into account when matching with the sketch.

Sketch synthesis

Clearly, there is a certain amount of natural variation in the users' sketching interaction (even if intended, they would not repeat the same sketch twice – at least not exactly). In order to achieve a stable training result and to reduce overfitting as much as possible, we synthesize additional sketches based on the natural variation of human sketches. To collect information about natural sketch variation, we organized a user study in which we asked the users to repeat the same sketch several times for a specific matching goal. The details of this user study are presented in section E.5.2.

Figure E.6 is an illustration of how we consider the variation of sketches based on the user study, also showing nine sample synthesis results according to the fitted model. As the user sketch is recorded as discrete points, the variation is modeled point-wise and consists of two parts that are meaningfully modeled separately: the horizontal displacement d_x and the vertical displacement d_y. A and B are two consecutive points of the user sketch, while B_g is the corresponding goal point which is located in the time series data (the point which the user aimed for). The distance d_y is the vertical point-wise displacement between the user sketch and the goal. θ is the angle between AB_g and AB.

By examining the user study data, we found that θ is strongly correlated with d_y, meaning that larger angles lead also to larger vertical displacements. Based on this observation, we use a cubic polynomial model $m_1(\theta)$ to fit the relation between θ and d_y. As m_1 represents the central tendency, we can compute the absolute difference between $m_1(\theta)$ and d_y to model deviation information. Also here, we use a cubic polynomial to fit the relation between this difference and θ as $m_2(\theta)$. Eventually, we can sample d_y from a normal distribution $N(m_1(\theta), m_2(\theta))$ with $m_1(\theta)$ as the mean and $m_2(\theta)$ as the standard deviation. Additionally, we compute the horizontal difference d_x between two consecutive points for distribution fitting from the user study data. We found that the average of d_x is around 1.5 pixels, which is too small and detailed to observe any variation. Thus, we consider a larger interval by taking every 10 points into account instead of every point, which we think is more reasonable for distribution fitting. Based on the statistical data we gathered, we used the statistical tool EasyFit [91] to analyze which distribution fits our variation data best. As a result, the variance of d_x (denoted as $var(d_x)$) follows a logarithmic distribution $f(x) = \frac{-\alpha^x}{x\ln(1-\alpha)}$ with $\alpha = 0.8525$.

For synthesizing a new sketch, given a user sketch I and the querying target G, we compute a new user interaction I' based on random samples from the fitted PDFs. We start from the first point in I, denoted by I_1. We then have $I'_1 = I_1$ and θ_1 as the angle between I_1I_2 and $I_1I'_2$. Based on this, we can synthesize the next point I'_2 with $I'_{2x} = I'_{1x} + d_{x_1}$ and for I'_{2y} we have $I'_{2y} = G_{2y} - d_{y_1}$ when $G_{2y} > I_{2y}$ while $I'_{2y} = G_{2y} + d_{y_1}$ when $G_{2y} < I_{2y}$. For sampling, d_{x_i} is sampled from the logarithmic distribution $f(x)$ and d_{y_i} is sampled from the normal distribution $N((m_1(\theta_1), m_2(\theta_1)))$ respectively. All subsequent points of the new sketch I' are estimated in the same way.

On the right of Figure E.6, nine user interactions (purple) from the second user study are shown that we used for studying the variation of sketches and the correspondingly modelled synthetic variations are shown in green, confirming that our model is able to generate meaningful and realistically looking sketching variations.

E.4.6 Training details

The original training data collected from the first user study consists of 2200 pairs of sketches and time series data with similarity ratings by the users. We extended this data by synthesizing ten pieces of lead data for each time series data and adding four modeled sketches for each actually recorded sketch, resulting in 110 000 pairs of $([L_i, R_i], y_i)$ as the training data. As the queries were supposed to match independently of their horizontal and vertical position, we preprocessed the input data $([L_i, R_i])$ by removing the x-coordinate and replacing the y-coordinate by its first derivative. For our research, we implemented the network and executed the training in Keras with Tensorflow as the backend, providing powerful GPU acceleration and convenient coding flexibility. For training and testing, we used a PC with an Intel Xeon E5-1650 CPU and an NVIDIA GeForce GTX 1080 GPU. There are two main hyper-parameters of the network to be set: time step (i.e., sliding window length) and cell size (i.e., output dimension). As the sketching panel in our experiment is 200×200 pixels, we experimented with different values of the sampling frequency and found that 40 (sampling interval up to 5 pixels) is sufficient to represent the details of a sketch. Therefore, we set the time steps of the $LSTM_{\text{data}}$ to 60 and the one of $LSTM_{\text{sketch}}$ to 40, which means that the length of the sliding window that we mimic is 40 (L=40). The hidden feature of the LSTM cell was set to be of size 20. Besides, a dropout function with a drop rate of 0.2 (com-

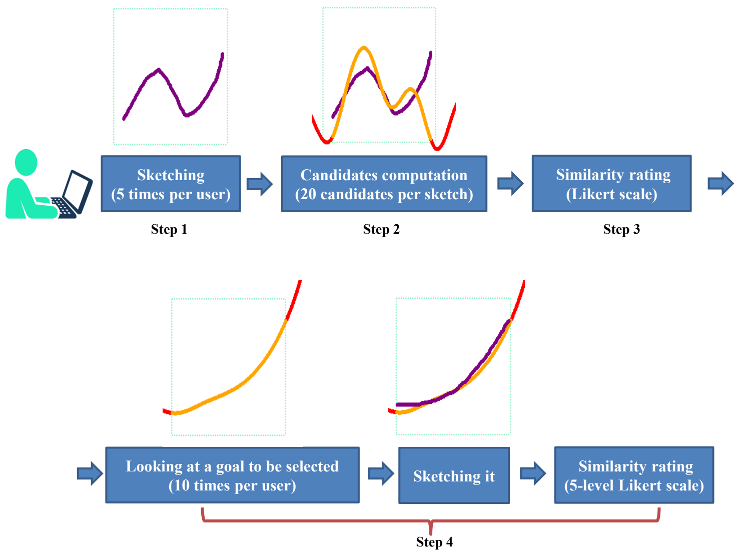


Figure E.7: Pipeline of the first user study. After sketching a pattern, we offer 20 candidates (computed by ED) per sketch and ask the user to rate similarity between them. Then, in order to enforce some high similarity data, we offer 10 target time series clips and ask the user to do the sketching aiming for them. In total, we can get 110 pairs of user sketches and time series data per user with corresponding similarity ratings.

mon value) was used to limit overfitting. We optimized the parameters (1760 in total) of the network based on the training data using the mean-squared error as a loss function. The Adam optimizer was used during the optimization and the learning rate was set to 10^{-3}. In total, we ran 2000 epochs for the training with a full batch.

E.5 User studies

Since the user plays a key role in visual query systems and to meet the user's expectation as close as possible regarding the matching, we conducted a user study to investigate how users sketch patterns and how they rate the similarities between their sketches and corresponding time series data. We then used this information to train our double network. Further, we also conducted a follow-up user study to explore how the user uses our sketching query in practice and analyzed the natural variation of their sketching interactions in order to prepare for the synthesis of additional data for a more stable training. Eventually, to evaluate our model, we also organized a third user study, in which we asked the users to rate the results from our model in comparison to the Qetch algorithm as well as to the DTW algorithm. Below, we provide more details about the three user studies.

E.5.1 User study for base training data

For the first user study, 10 different time series data with length ranges from 91 to 4774 were prepared as a basis. All of these data were carefully chosen in advance from datasets in finance, industry, medicine, and the labor market, with a healthy spread of characteristics (covering important and common patterns: head-and-shoulder, sharp rises/dips, upward or downward slopes, peaks and troughs, etc.). For each dataset, 5 new variants were produced based on 5 different scales and these 50 new time series data were used for the user study. The procedure of the first user study consists of four steps, illustrated in Figure E.7:

1. First, an empty canvas (900×400 pixels) was shown to the user and at the same time the user was asked to think about a pattern he/she wanted to look up from an instance of time series data. Then the user used the mouse to sketch this pattern on the sketch panel (200×200 pixels, located in the middle of the canvas). The reason for choosing a mouse as the input device was that we wanted as general as possible results and pen-based input is not generally available to many users.

2. Based on the user's sketch, we computed similarities by using a simple ED matching rule and a sliding window with step length of 20 pixels, over all the 50 variants of the prepared time series data, yielding a ranked list of matches. We then chose 20 candidates evenly from the matching results based on their similarity rank. The 20 candidates were distributed evenly around 0%, 25%, 50%, 75%, and 100%, where 0% means the most dissimilar and 100% amounting to the best match. The purpose of this procedure was to achieve training data within a healthy range of similarities so that the network could learn the similarity function properly.

3. In the next step, the user was asked to rate the similarities between their sketch and the 20 candidates we offered, one by one. The similarity rating was acquired on the Likert scale [64], which is commonly used to collect respondents' attitudes and opinions. We asked the users to choose their rating from five options: no match ($s = 0$), bad match ($s = 0.25$), half good / half bad match ($s = 0.5$), good match ($s = 0.75$), and excellent match ($s = 1$). All users were clearly informed about the correspondence between the similarities and the five rating options in the tutorial section of the user study. For the whole study, each user had to do five sketches and for each sketch he/she rated the similarity against 20 candidates based on their visual perception. In total, 100 pairs of time series data and sketch with rated similarities were recorded per user.

4. In the second step, we used simple ED to do an initial selection of candidates for rating, aiming at training data with balanced similarities. The actual situation, however, turned out to be so that only for very few candidates our users were satisfied enough to rate them as good match or even as excellent match. Therefore, in the fourth stage of this study, instead of asking the users to sketch a pattern in their mind, we showed them 10 additional candidates in sequence, asking them to sketch while aiming at the shown curve. Then, we asked them to rate how satisfied they were with their own drawing, again using the Likert scale. This way, we got some additional pairs of data with highly rated similarity, making sure that our training data covers a health range in terms of similarity.

In the first user study, 20 users were invited to participate, all students or employees from the University of Bergen, and in total 2200 pairs of data with perceived similarities were collected for training (the details of the collected pairs are shown in the supplementary material). Before the user study, every user was given a training session to get familiar with the interface and the mouse operations for sketching. In addition, we showed them 10 typical patterns in time series data such as rounded-bottom, head-and-shoulders, sharp rise and down, falling peaks, and so on. This procedure is to help those users, who were unfamiliar with time series data, to do the sketching meaningfully. During the training session, we answered any questions they had about the tool until they were ready for the study.

E.5.2 Studying the variation of sketches

Naturally, for the same matching goal, the sketch done by the user will be a bit different every time. In order to understand the natural variation of the user's sketches, when having the same matching goal and sketch operation in mind, and to model this variation for synthesizing additional data for the training, we did this follow-up user study based on the first user study. In the second study, 10 individuals, all students or employees from the University of Bergen, participated. We chose 100 representative clips of the time series data from the first user study. These 100 clips have an even distribution in terms of shapes and frequency, which forms a healthy base to investigate the variation of human sketches. For each user, 10 different time series clips were displayed as the matching goal for sketching. The second user study then consisted of two parts:

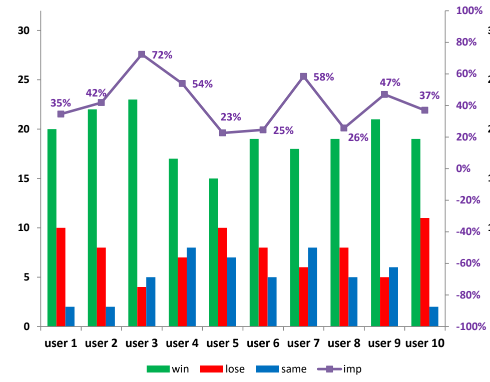
In the first part, the users were asked to look at a clip of the time series data that we provided. Then, the users were required to draw a sketch with the goal to match the shown clip. The users were asked to repeat this interaction 12 times in each case. The traces of all sketches were recorded during the study. Altogether, 1200 sketches were collected and used for modeling the variation among sketches.

E.5.3 Evaluation user study

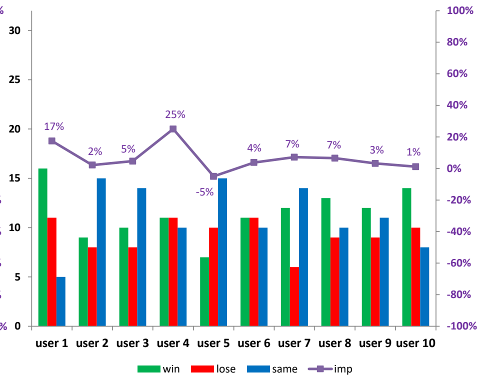
In order to evaluate our new model, learned by the double network, we conducted a third user study, asking users to tell whether the computed matching results were considered good, or not. As a baseline for comparison, we chose two state-of-the-art techniques, i.e., DTW and the recently published Qetch algorithm [71]. The well-established DTW metric was chosen as one of the best options for distance measures in time series data [16], while the authors of Qetch claimed its strength over DTW for freehand sketch and matching for high-level task (in terms of time spent). In this user study, the users were asked to rate the matching results computed by Qetch, DTW, and our new method, leading to a quantitative, comparative evaluation reflecting the user's perspective.

For this evaluation study, 10 users were invited. For each user, 8 new time series data (with lengths ranging from 40 to 1440, none of them used for training before) were provided in sequence to test the generality of the proposed model. The procedure of this user study consisted of two steps:

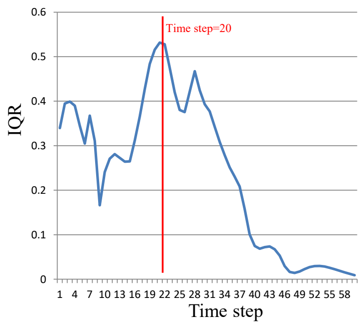
1. To start, a dataset with a useful default smoothing level (see above) was presented to the user. For each dataset, the user could interact with the interface and specify the targeted scale of the visualization by zooming with the mouse. In addition, users could also adjust the smoothing level by using a slider.



A



B



C

Figure E.8: A: comparison between our method and the Qetch algorithm. B: comparison between our method and DTW. Green bars show how often our technique was preferred (red: how often Qetch/DTW was preferred; blue: tied) and the purple line indicates per user the improvement as achieved by our method (positive: average improvement due to our method). C: IQR values showing output variation between different time series from the 1st time step to the 60th, indicating the change of variation over time.

2. Then, the user was asked to sketch a pattern he/she wanted to look up from the time series data on the sketching panel. Based on the user' sketch, we computed the three best matching results from our new model, the Qetch algorithm, and using DTW, respectively. Then, we showed these three results (in random order and without telling which is which) to the user and asked them to rate the similarity between their sketch and these three results separately. As Qetch and DTW are highly competitive matching algorithms, capturing the difference between good and very good results requires more detail, so we offered a once refined nine-points range (also from 0 to 1, $s = 0, 0.125, 0.25, 0.375 ... 1.0$) for rating instead of the courser five-points Likert scale, delivering the required details for a proper comparison.

During this study, we recorded the similarity rating from each user for a subsequent quantitative analysis (see below). For each data, the user sketched four times and did four sets (our method, Qetch, and DTW) rating. Accordingly, we collected altogether 320 sets of rating results from 10 users. In addition, the time cost of computation were recorded for comparing efficiency, also. Before the user study started, every users was offered a training session to get familiar with the interface and the interaction of the user study.

E.6 Evaluation results

For evaluating our matching model, we did a quantitative comparison with Qetch and DTW – shown in Figure E.8A and Figure E.8B respectively. The results (32 pairs from each user in the third study) are shown as bars (how often one technique was preferred) and as a line graph (average improvement). Green bars show (per user) the number of times that our method was rated with a higher similarity than Qetch (A) or DTW (B), while red bars represents the contrary (blue bars a tied rating). Further, we also compute the average improvement (denoted as *imp*) of our method as compared with Qetch/DTW in terms of the similarity value: $imp = \frac{s_{\text{our}} - s_{\text{qetch}}}{s_{\text{qetch}}}$ or $\frac{s_{\text{our}} - s_{\text{dtw}}}{s_{\text{dtw}}}$ (s_{our}, s_{qetch}, and s_{dtw} denote the average similarity rating of our method, the Qetch algorithm, and DTW, respectively) and show it (also per user) as line graph in purple.

By looking at the bar graph in Figure E.8A, we clearly see that all users preferred our matching over the Qetch algorithm and that our method was preferred about 2.5 so often as the other way around (193:77). The line graph shows that all average improvements are positive, providing a clear evidence that our method is closer to the user's perception in terms of similarity. More specifically, the average improvement is $\approx 42\%$ as $\bar{s}_{\text{our}}$=0.64 and $\bar{s}_{\text{qetch}} = 0.45$, where $\bar{s}_{\text{our}}$ and $\bar{s}_{\text{qetch}}$ are the average similarity values of our method as compared to the Qetch algorithm. Based on this evaluation, we are confident to conclude that in general our method performs significantly better than the Qetch algorithm, when the matching results are directly judged by the users.

By looking at the chart in Figure E.8B, 7 out of 10 users preferred our technique over DTW according to the bar graph, while the average improvements are positive (in our favor) for 9 out of 10 users, when looking at the line graph. The overall ratio of users preferring our method over DTM is around 1.2 (115:93) and the average improvement for each user is $\approx 7\%$ with $\bar{s}_{\text{dtw}} = 0.6$ ($\bar{s}_{\text{dtw}}$ is the average similarity of DTW rated by all the users). Overall, and even though the improvement numbers are clearly smaller than in the comparison with Qetch, this suggests that our method is slightly better than

DTW in terms of accuracy (at least not worse) when evaluated directly from the user's perception. The details of all user ratings are in the supplementary material.

Furthermore, and since a swift user–computer dialogue in visual query systems is highly dependent on the efficiency of the involved interactions, we also compared the computation cost of the three methods. According to the user study, the average computation costs of our method, Qetch, and DTW, are 33ms, 132ms and 3.6s, respectively. Based on this data, we see clearly that our method is the most efficient one due to its linear complexity, while DTW is the slowest and unable to achieve a real-time interaction.

Besides the quantitative evaluation in terms of accuracy and efficiency, we also examined the output of each step of the LSTM network to check whether the network learns meaningful information. The reason for doing this was that the network was implicitly taught to only remember the information of a specific length of previous time steps (we set this to 40 in our experiment) with a goal to mimic a sliding window for matching. To investigate whether the network has successfully achieved this important feature, we collect the output (namely the hidden state **h**) of each time step of several time series data with different left lead data and compute the output variations over the time steps to analyze whether the network is able to discard long term dependencies. The details are illustrated below.

In our training, the time series data clips were sampled at 40 points with synthesized lead data "on the left" at 20 points. For looking into the information as learned by the network, we did an experiment that generated 20 time series snippets with length 60 (denoted as G_i, $i \in \{1, \ldots, 20\}$), where the trailing 40 entries for each time series were the same, but the first 20 varied according to our synthesis procedure. As a reference, we have another time series (denoted as ref) that has the same last 40 entries but with a real lead data that is different from all synthesized lead data G_i. We then iterated the time series and the reference time series over the 60 time steps by using the already trained model and obtained a set of outputs with format $20 \times 1 \times 60 \times 20$ and $1 \times 60 \times 20$. We then computed the cosine similarity between all outputs of G_i and ref per time step, leading to a set of 20 cosine similarities. We compute the inter-quartile range (IQR), i.e., the difference between the 75%- and the 25%-percentile, as a robust indicator of the variation over the time steps.

After iterating over all the time steps from 1 to 60, we have 60 IQR values which represent the output variations over the 60 time steps and this information is shown in Figure E.8C. As we mentioned, from the 1st time step to 20th time step, the network output corresponds to lead data – since all of the lead data was randomly synthesized the variation during this period is fluctuating at a relatively high level. After time step 20, however, we see a decline of the IQR values, correlated with the networking reading actual time series data. Towards the 60th time step, the IQR values approach 0, which meets our expectation that the output at the last time step almost only represents the information of the previous 40 time steps. In summary, this statistics gives us a strong indication that the LSTM network has been successfully trained to understand that only the information from a certain length of the previous time steps should be taken into account in each time step.

E.7 Limitations and future work

Although our new model demonstrated its accuracy and efficiency over two state-of-the-art methods, there are still three limitations: 1. It cannot be excluded that scenarios exist that are not covered by the model due to limited training data. This is in general a known problem of deep learning approaches – high prediction accuracy requires tremendous amount of training data which is often not easy to obtain. 2. Deep learning models lack sufficient interpretability due to their black-box nature, especially when compared with the carefully crafted empirical models. 3. Using a fixed query size is not as flexible as the Qetch approach, which matches based on the salient parts in the time series data.

In the future, we see several opportunities to further extend our work, including:

- Further improving the synthesis of additional training data as this plays a crucial role for the learning process. One possible way may be to use a generative adversarial network (GAN) – similar to other successful applications in image generation and synthesis. We hypothesize that a GAN could synthesize more realistic variations of the user's sketch.
- The design of a matching algorithm, which is tailored for a particular user, using an appropriate method to learn this user's particular sketching behavior over time, would be interesting, as well.

E.8 Conclusion

In this paper, we have demonstrated how deep learning can be used to further improve a visual query system for exploring time series data in visual analytics. By learning the relation between the time series clip to be selected and a free-hand user sketch, we achieve a solution, which is both fast and accurate. To the best of our knowledge, this is the first study to report the successful application of a matching model, realized by a pair of parameter-sharing LSTM networks, to improve an important human interaction scenario in visual analytics. We demonstrate, quantitatively, and in comparison with the recently published Qetch algorithm as well as the classical distance measure DTW, that our LSTM-based solution leads to an improvement in terms of the overall similarity ($\approx$42% to Qetch and $\approx$7% to DTW), rated by users in a user study, while enabling a fast interaction ($\approx$4 times faster than Qetch and $\approx$100 times faster than DTW). The code of our prototype is available via github.com/reddestrabbit/LSTM-based-visual-query-system.git.

Acknowledgments

We thank all participants of the three user studies for taking their valuable time to support this research. We also thank the reviewers of an earlier version of this paper for their valuable input. Parts of this work were done in the context of CEDAS, UiB's Center for Data Science. VRVis is funded by BMVIT, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG.

Bibliography

[1] Altman, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician 46*, 3 (1992), 175–185. B.5.1

[2] Becker, R. A., and Cleveland, W. S. Brushing scatterplots. *Technometrics 29*, 2 (1987), 127–142. 1.1, A.1, B.1, C.1, D.1

[3] Behrisch, M., Korkmaz, F., Shao, L., and Schreck, T. Feedback-driven interactive exploration of large multidimensional data supported by visual classifier. In *IEEE Conference on Visual Analytics Science and Technology (VAST)* (2014), pp. 43–52. 2.1.1

[4] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks 5*, 2 (1994), 157–166. 2.1.2, E.1, E.4.2

[5] Bernard, J., Zeppelzauer, M., Sedlmair, M., and Aigner, W. VIAL: a unified process for visual interactive labeling. *The Visual Computer 34*, 9 (2018), 1189–1207. 2.1.3

[6] Beucher, S., and Meyer, F. The morphological approach to segmentation: the watershed transformation. *Optical Engineering 34* (1992), 433–433. 3.1.4

[7] Buono, P., Aris, A., Plaisant, C., Khella, A., and Shneiderman, B. Interactive pattern search in time series. In *Visualization and Data Analysis* (2005), vol. 5669, pp. 175–187. 2.3.2, E.2.2

[8] Card, S. K., Robertson, G. G., and Mackinlay, J. D. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1991), pp. 181–186. 1.1, A.1, B.1, C.1

[9] Chang, H., and Yeung, D.-Y. Robust path-based spectral clustering. *Pattern Recognition 41*, 1 (2008), 191–203. 4.4

[10] Chen, Z., Zeng, W., Yang, Z., Yu, L., Fu, C.-W., and Qu, H. LassoNet: Deep lasso-selection of 3D point clouds. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2019), 195–204. 2.3, 2.1.2, E.2.3

[11] Chollet, F., et al. Keras, 2015. 4.2, B.5.2

[12] Chopra, S., Hadsell, R., LeCun, Y., et al. Learning a similarity metric discriminatively, with application to face verification. In *Conference on Computer Vision and Pattern Recognition* (2005), pp. 539–546. E.4.3

[13] Correll, M., and Gleicher, M. The semantics of sketch: Flexibility in visual query systems for time series data. In *IEEE Conference on Visual Analytics Science and Technology (VAST)* (2016), pp. 131–140. 2.3.2, E.2, E.2.2

[14] Dibia, V., and Demiralp, Ç. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications 39*, 5 (2019), 33–46. 2.1.2, E.2.3

[15] Dice, L. R. Measures of the amount of ecologic association between species. *Ecology 26*, 3 (1945), 297–302. 4.2, A.6, B.6

[16] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment 1*, 2 (2008), 1542–1552. 2.3.1, 4.5, E.2.1, E.5.3

[17] Doleisch, H., Gasser, M., and Hauser, H. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the Symposium on Data Visualisation* (2003), pp. 239–248. 2.2, A.1, B.1, C.1

[18] Doleisch, H., Hauser, H., and Hauser, M. Smooth brushing for focus+context visualization of simulation data in 3D. *Journal of WSCG 1* (2002), 147–154. 2.4

[19] Eichmann, P., and Zgraggen, E. Evaluating subjective accuracy in time series pattern-matching using human-annotated rankings. In *Proceedings of the 20th International Conference on Intelligent User Interfaces* (2015), pp. 28–37. 2.3.2, E.2.2

[20] Elmqvist, N., Vande Moere, A., Jetter, H.-C., Cernea, D., Reiterer, H., and Jankun-Kelly, T. J. Fluid interaction for information visualization. *Information Visualization 10*, 4 (2011), 327–340. 1.1, A.1, B.1, C.1

[21] Endert, A., Ribarsky, W., Turkay, C., Wong, B. W., Nabney, I., Blanco, I. D., and Rossi, F. The state of the art in integrating machine learning into visual analytics. In *Computer Graphics Forum* (2017), vol. 36, pp. 458–486. 2.1.1

[22] Fan, C., and Hauser, H. User-study based optimization of fast and accurate mahalanobis brushing in scatterplots. In *Proceedings of the Conference on Vision, Modeling and Visualization* (2017), pp. 77–84. 2.2, 4.4, B.1, B.2.1, B.3, B.4.2, B.6, B.6.1, B.7, C.1, C.2.1, D.1, D.2, D.3, D.3.1, D.4, D.5, D.6, D.1, D.6.1, D.6.2, E.2

[23] Fan, C., and Hauser, H. Fast and accurate cnn-based brushing in scatterplots. In *Computer Graphics Forum* (2018), vol. 37, pp. 111–120. 3.1.3, C.1, C.2.1, C.3, C.3.1, C.4, C.4.1, C.5.1, C.5.2, C.6, D.1, D.2, D.6, D.2, D.6.3, E.2.3

[24] Fan, C., and Hauser, H. On KDE-based brushing in scatterplots and how it compares to CNN-based brushing. In *Machine Learning Methods in Visualisation for Big Data* (2019), The Eurographics Association. D.1

[25] Fan, C., and Hauser, H. Personalized sketch-based brushing in scatterplots. *IEEE Computer Graphics and Applications 39*, 4 (2019), 28–39. D.2, E.2, E.2.3

[26] Florek, M., and Hauser, H. Quantitative data visualization with interactive kde surfaces. In *Proceedings of the 26th Spring Conference on Computer Graphics* (2010), pp. 33–42. 3.1.4, D.3.3

[27] Florek, M., and Hauser, H. Interactive bivariate mode trees for visual structure analysis. In *Proceedings of the 27th Spring Conference on Computer Graphics* (2011), pp. 95–102. 3.1.4

[28] Fukushima, K., and Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*. Springer, 1982, pp. 267–285. 2.1.2, B.2.2, C.2.2

[29] Gionis, A., Mannila, H., and Tsaparas, P. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*, 1 (2007), 4. 4.4

[30] Goldin, D. Q., and Kanellakis, P. C. On similarity queries for time-series data: constraint specification and implementation. In *International Conference on Principles and Practice of Constraint Programming* (1995), pp. 137–153. 2.3.1, E.2.1

[31] Gough, B. *GNU scientific library reference manual*. Network Theory Ltd., 2009. 3.1.1, A.4.1

[32] Gregory, M., and Shneiderman, B. Shape identification in temporal data sets. In *Expanding the Frontiers of Visual Analytics and Visualization*. Springer, 2012, pp. 305–321. 2.3.2, E.2.2

[33] Han, J., Tao, J., and Wang, C. FlowNet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE Transactions on Visualization and Computer Graphics 26*, 4 (2020), 1732–1744. 2.1.2, 2.3, E.2.3

[34] Harrison, D., and Rubinfeld, D. L. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management 5*, 1 (1978), 81–102. A.5.1

[35] Hauser, H. Generalizing focus+context visualization. In *Scientific Visualization: The Visual Extraction of Knowledge from Data*. Springer, 2005, pp. 305–327. 1.1, A.1, B.1, C.1, D.1

[36] Hauser, H., Ledermann, F., and Doleisch, H. Angular brushing of extended parallel coordinates. In *IEEE Symposium on Information Visualization* (2002), pp. 127–130. 2.2, D.2

[37] HOCHHEISER, H., AND SHNEIDERMAN, B. Visual queries for finding patterns in time series data. *University of Maryland, Computer Science Dept. Tech Report, CS-TR 4365* (2002). 2.3.2, 2.5, E.2.2

[38] HOCHHEISER, H., AND SHNEIDERMAN, B. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization 3*, 1 (2004), 1–18. 2.3.2, E.2.2

[39] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780. 2.1.2, E.4.2

[40] HOFFMAN, D. D., AND SINGH, M. Salience of visual parts. *Cognition 63*, 1 (1997), 29–78. 2.3.2, E.2.2

[41] HOHMAN, F., KAHNG, M., PIENTA, R., AND CHAU, D. H. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics 25*, 8 (2018), 2674–2693. 2.1.2, E.2.3

[42] HOLZ, C., AND FEINER, S. Relaxed selection techniques for querying time-series graphs. In *Proceedings of the 22nd annual ACM Symposium on User Interface Software and Technology* (2009), pp. 213–222. 2.3.2, E.2, E.2.2

[43] HOSSAIN, M. S., OJILI, P. K. R., GRIMM, C., MÜLLER, R., WATSON, L. T., AND RAMAKRISHNAN, N. Scatter/gather clustering: Flexibly incorporating user feedback to steer clustering results. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2829–2838. 2.1.1

[44] HU, K., BAKKER, M. A., LI, S., KRASKA, T., AND HIDALGO, C. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)* (2019), pp. 1–12. 2.1.2, E.2.3

[45] HUBEL, D. H., AND WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology 160*, 1 (1962), 106–154. 2.1.2, B.2.2, C.2.2

[46] HURTER, C., TELEA, A., AND ERSOY, O. Moleview: An attribute and structure-based semantic lens for large element-based plots. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2600–2609. 2.2, B.2.1

[47] JEONG, D. H., ZIEMKIEWICZ, C., FISHER, B., RIBARSKY, W., AND CHANG, R. iPCA: An interactive system for PCA-based visual analytics. In *Computer Graphics Forum* (2009), vol. 28, pp. 767–774. 2.1.1

[48] JOHANSSON, S., AND JOHANSSON, J. Interactive dimensionality reduction through user-defined combinations of quality metrics. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 993–1000. 2.1.1

[49] KALCHBRENNER, N., GREFENSTETTE, E., AND BLUNSOM, P. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* (2014). B.1

[50] KEIM, D., ANDRIENKO, G., FEKETE, J.-D., GÖRG, C., KOHLHAMMER, J., AND MELANÇON, G. Visual analytics: Definition, process, and challenges. In *Information visualization*. Springer, 2008, pp. 154–175. 2.1, 2.1

[51] KEOGH, E., HOCHHEISER, H., AND SHNEIDERMAN, B. An augmented visual query mechanism for finding patterns in time series data. In *International Conference on Flexible Query Answering Systems* (2002), pp. 240–250. 2.3.2, E.2.2

[52] KEOGH, E., AND SMYTH, P. A probabilistic approach to fast pattern matching in time series databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)* (1997), pp. 24–30. 2.3.2, E.2.2

[53] KIM, B., AND GÜNTHER, T. Robust reference frame extraction from unsteady 2d vector fields with convolutional neural networks. In *Computer Graphics Forum* (2019), vol. 38, pp. 285–295. 2.1.2, E.2.3

[54] KOHAVI, R., ET AL. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *the 14th International Joint Conference on Artificial Intelligence* (1995), vol. 14, pp. 1137–1145. 4.2, B.6

[55] KONG, N., AND AGRAWALA, M. Perceptual interpretation of ink annotations on line charts. In *Proceedings of the 22nd annual ACM Symposium on User Interface Software and Technology* (2009), pp. 233–236. 2.3.2, E.2.2

[56] KONYHA, Z., MATKOVIĆ, K., GRAČANIN, D., JELOVIĆ, M., AND HAUSER, H. Interactive visual analysis of families of function graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 6 (2006), 1373–1385. 2.2, A.1, B.1

[57] KOYTEK, P., PERIN, C., VERMEULEN, J., ANDRÉ, E., AND CARPENDALE, S. MyBrush: Brushing and linking with personal agency. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2017), 605–615. 2.2, B.2.1, C.2.1, D.2

[58] KRAUSE, J., PERER, A., AND BERTINI, E. Infuse: interactive feature selection for predictive modeling of high dimensional data. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 1614–1623. 2.1.1

[59] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105. 2.1.2, B.1, B.2.2, B.5.2, C.2.2, E.1

[60] LANGLEY, P. Machine learning for adaptive user interfaces. In *Annual Conference on Artificial Intelligence* (1997), pp. 53–62. A.2.2

[61] LECUN, Y., BOSER, B. E., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W. E., AND JACKEL, L. D. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems* (1990), pp. 396–404. 2.1.2, B.2.2, B.4.3

[62] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324. 2.1.2, B.2.2, C.2.2

[63] LIEBERMAN, H., VAN DYKE, N., AND VIVACQUA, A. Let's browse: a collaborative browsing agent. *Knowledge-Based Systems 12*, 8 (1999), 427–431. A.2.2

[64] LIKERT, R. A technique for the measurement of attitudes. *Archives of psychology* (1932). E.5.1

[65] LIU, J., WONG, C. K., AND HUI, K. K. An adaptive user interface based on personalized learning. *IEEE Intelligent Systems 18*, 2 (2003), 52–57. A.2.2

[66] LIU, M., SHI, J., LI, Z., LI, C., ZHU, J., AND LIU, S. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2016), 91–100. 2.1.2, 2.3

[67] LIU, S., WANG, X., LIU, M., AND ZHU, J. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics 1*, 1 (2017), 48–56. E.2.3

[68] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH Computer Graphics* (1987), vol. 21, pp. 163–169. B.4.4

[69] MAHALANOBIS, P. C. On the generalised distance in statistics. In *Proceedings National Institute of Science, India* (1936), vol. 2, pp. 49–55. 3.1.1, A.4, D.3.1

[70] MALIK, A., MACIEJEWSKI, R., ELMQVIST, N., JANG, Y., EBERT, D. S., AND HUANG, W. A correlative analysis process in a visual analytics environment. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2012), pp. 33–42. 2.1.1

[71] MANNINO, M., AND ABOUZIED, A. Expressive time series querying with hand-drawn scale-free sketches. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)* (2018), pp. 1–13. 1.1, 2.3.2, 4.5, E.1, E.2, E.2.2, E.5.3

[72] MARTIN, A. R., AND WARD, M. O. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings of the 6th Conference on Visualization* (1995), IEEE Computer Society, pp. 271–278. 2.2, 2.2, A.1, A.2.1, B.1, B.2.1, C.1, C.2.1, D.2

[73] MATKOVIC, K., GRACANIN, D., JELOVIC, M., AND HAUSER, H. Interactive visual steering-rapid visual prototyping of a common rail injection system. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1699–1706. 2.1.1

[74] MUELLER, J., AND THYAGARAJAN, A. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016), p. 2786–2792. 3.2, E.1, E.3

[75] MÜHLBACHER, T., AND PIRINGER, H. A partition-based framework for building and validating regression models. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 1962–1971. 2.2, 2.1.1

[76] MUIGG, P., KEHRER, J., OELTZE, S., PIRINGER, H., DOLEISCH, H., PREIM, B., AND HAUSER, H. A four-level focus+context approach to interactive visual analysis of temporal features in large scientific data. *Computer Graphics Forum 27*, 3 (2008), 775–782. 2.2, 2.2, 3.1.2, A.1, A.2.1, B.1, B.2.1, B.3, C.1, D.2

[77] MUNZNER, T. *Visualization Analysis & Design*. CRC Press, 2014. 3.1, A.1, B.1

[78] MUTHUMANICKAM, P. K., VROTSOU, K., COOPER, M., AND JOHANSSON, J. Shape grammar extraction for efficient query-by-sketch pattern matching in long time series. In *IEEE Conference on Visual Analytics Science and Technology (VAST)* (2016), pp. 121–130. 2.3.2, E.2, E.2.2

[79] NOVOTNÝ, M., AND HAUSER, H. Outlier-preserving focus+ context visualization in parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 893–900. 3.1.4

[80] NOVOTNỲ, M., AND HAUSER, H. Similarity brushing for exploring multidimensional relations. In *Journal of WSCG* (2006), vol. 14, pp. 105–112. 2.2, A.2.1, B.2.1, C.2.1, E.2

[81] PAN, S. J., AND YANG, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359. 2.1.3, C.2.2

[82] PARZEN, E. On estimation of a probability density function and mode. *The annals of mathematical statistics 33*, 3 (1962), 1065–1076. 3.1.4, 3.1.4, A.8, D.1, D.3.2

[83] POWERS, D. M. Evaluation: From precision, recall and f-measure to roc, informedness, markedness correlation. *J. Mach. Learn. Technol 2* (01 2011), 2229–3981. 4.4, D.6.1

[84] RADOŠ, S., SPLECHTNA, R., MATKOVIĆ, K., ĐURAS, M., GRÖLLER, E., AND HAUSER, H. Towards quantitative visual analytics with structured brushing and linked statistics. In *Computer Graphics Forum* (2016), vol. 35, pp. 251–260. 2.2, 2.2, 3.1.1, 3.1.1, 3.1.1, 3.1.2, 4.1, 4.1, A.1, A.2.1, A.4, A.7, A.8, B.1, B.2.1, B.3, C.1, C.2.1, D.1, D.2

[85] RASMUSSEN, M., AND KARYPIS, G. gCLUTO: An interactive clustering, visualization, and analysis system. *UMN-CS TR-04 21*, 7 (2004). 2.1.1

[86] REN, S., HE, K., GIRSHICK, R., AND SUN, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* (2015), pp. 91–99. B.1, E.1

[87] ROBERTS, J. C. State of the art: Coordinated & multiple views in exploratory visualization. In *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization* (2007), pp. 61–71. 3.1, A.1, B.1, C.1, D.1, D.2

[88] ROSENBLATT, M., ET AL. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics 27*, 3 (1956), 832–837. 3.1.4, A.8

[89] RYALL, K., LESH, N., LANNING, T., LEIGH, D., MIYASHITA, H., AND MAKINO, S. Querylines: approximate query for visual browsing. In *Extended Abstracts on Human Factors in Computing Systems (CHI)* (2005), pp. 1765–1768. 2.5, 2.3.2, E.2.2

[90] SAKOE, H., AND CHIBA, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing 26*, 1 (1978), 43–49. E.1, E.2.1

[91] SCHITTKOWSKI, K. Easy-fit: a software system for data fitting in dynamical systems. *Structural and Multidisciplinary Optimization 23*, 2 (2002), 153–169. B.8, E.4.5

[92] SETTLES, B. Active learning literature survey. Tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2009. C.2.2

[93] SHARIF RAZAVIAN, A., AZIZPOUR, H., SULLIVAN, J., AND CARLSSON, S. CNN features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition workshops* (2014), pp. 806–813. 2.1.3

[94] SHEN, Y., HE, X., GAO, J., DENG, L., AND MESNIL, G. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web* (2014), pp. 373–374. B.1, E.1

[95] SHYE, A., SCHOLBROCK, B., AND MEMIK, G. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (2009), pp. 168–178. A.2.2

[96] TONG, Y. L. *The multivariate normal distribution*. Springer Science & Business Media, 2012. 3.1.4

[97] TUKEY, J. W., AND TUKEY, P. A. Computer graphics and exploratory data analysis: An introduction. *The Collected Works of John W. Tukey: Graphics: 1965-1985 5* (1988), 419. A.5.1, C.3.1, D.4

[98] TURKAY, C., KAYA, E., BALCISOY, S., AND HAUSER, H. Designing progressive and interactive analytics processes for high-dimensional data analysis. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 131–140. 1.1, A.1, B.1

[99] TURKAY, C., PARULEK, J., REUTER, N., AND HAUSER, H. Integrating cluster formation and cluster evaluation in interactive visual analysis. In *Proceedings of the 27th Spring Conference on Computer Graphics* (2011), pp. 77–86. 2.2, 2.1.1

[100] VAN DEN ELZEN, S., AND VAN WIJK, J. J. Baobabview: Interactive construction and analysis of decision trees. In *IEEE Conference on Visual Analytics Science and Technology (VAST)* (2011), pp. 151–160. 2.2, 2.1.1

[101] VEENMAN, C. J., REINDERS, M. J. T., AND BACKER, E. A maximum variance cluster algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 9 (2002), 1273–1280. 4.4

[102] WAND, M., AND JONES, M. Kernel smoothing, 1995. 3.1.4

[103] WANG, Y., JIN, Z., WANG, Q., CUI, W., MA, T., AND QU, H. Deepdrawing: A deep learning approach to graph drawing. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2019), 676–686. 2.1.2

[104] WATTENBERG, M. Sketching a graph to query a time-series database. In *Extended Abstracts on Human factors in Computing Systems (CHI)* (2001), pp. 381–382. 2.3.2, E.2, E.2.2

[105] WILKINSON, L., ANAND, A., AND GROSSMAN, R. Graph-theoretic scagnostics. In *IEEE Symposium on Information Visualization* (2005), IEEE, pp. 157–164. A.5.1

[106] WILLIAMS, M., AND MUNZNER, T. Steerable, progressive multidimensional scaling. In *IEEE Symposium on Information Visualization* (2004), pp. 57–64. 2.1.1, 2.2

[107] YANG, M.-H., AND AHUJA, N. *Face detection and gesture recognition for human-computer interaction*, vol. 1. Springer Science & Business Media, 2001. 2.2, A.2.1, B.2.1, D.2

[108] ZAHN, C. T. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers 100*, 1 (1971), 68–86. 4.4

[109] ZEILER, M. D., AND FERGUS, R. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision* (2014), pp. 818–833. 2.1.2, B.2.2, B.4.3, E.2.3

Graphic design: Communication Division, UIB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230846124 (print)
9788230856437 (PDF)