JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
DI Andreas Hinterreiter

Angefertigt am
Institut für Computergrafik

Erstbetreuer
Univ.-Prof. Dr. Marc Streit

Zweitbetreuer
Prof. Dr. Bernhard Kainz

November 2022

Visual Explanations of High-dimensional and Temporal Processes

Dissertation
zur Erlangung des akademischen Grads
Doktor der Technischen Wissenschaften
im Doktorratsstudium der
Technischen Wissenschaften

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, November 2022

Andreas Hinterreiter

ABSTRACT

Visualization and machine learning research are both driven by a desire to extract insights from data. However, the means to this end differ substantially between the two fields. While machine learning typically tries to automate decisions, visualization focuses on the human in the loop. A combination of these disparate approaches can help users to acquire insights from data more effectively. This thesis compiles results from five studies in which visualization and machine learning were brought together with a focus on temporal and/or high-dimensional processes. These works span the range from visualizations for model analysis to data processing for visualization. *ConfusionFlow* and *InstanceFlow* are two interactive visualization systems that let machine learning developers analyze the temporal progression of the training of classification models. A more general analysis of high-dimensional, temporal processes is possible with the *Projection Path Explorer*, which visualizes processes as trajectories in a low-dimensional embedding space. The Projection Path Explorer makes use of unsupervised machine learning for nonlinear dimensionality reduction. *Projective Latent Interventions* show how these unsupervised techniques can be adapted to give users more control over, and a better understanding of, the latent representations of classification models. To this end, parametric extensions of dimensionality reduction techniques are introduced, which allow users to manipulate the embeddings in such a way that changes can be propagated back to the original classification model. Finally, *ParaDime* is a framework for specifying such parametric dimensionality reduction routines in a flexible and customizable way. ParaDime unifies existing techniques and facilitates experimentation with new embedding methods for visualization. These five works illustrate the variety of possible combinations of machine learning and visualization, and showcase how such combined approaches can help users to better understand high-dimensional, temporal processes.

KURZFASSUNG

Erkenntnisgewinn aus Daten ist die Hauptmotivation sowohl für Forschung im Bereich der Datenvisualisierung als auch im maschinellen Lernen. Die Mittel, mit denen dieses Ziel erreicht werden soll, unterscheiden sich jedoch erheblich zwischen diesen Feldern. Während das maschinelle Lernen in der Regel versucht Entscheidungen zu automatisieren, rückt die Visualisierung den Menschen in den Mittelpunkt. Eine Kombination dieser unterschiedlichen Ansätze kann Nutzer:innen dabei unterstützen, effektive Einblicke in Daten zu erhalten. Diese Dissertation fasst die Ergebnisse aus fünf Arbeiten zusammen, in denen Visualisierung und maschinelles Lernen kombiniert werden – mit Fokus auf zeitliche und/oder hochdimensionale Prozesse. Dabei spannen diese Arbeiten den Bogen von Visualisierungen zur Modellanalyse bis hin zur Datenverarbeitung für Visualisierung. *ConfusionFlow* und *InstanceFlow* sind zwei interaktive Visualisierungssysteme, mit denen Entwickler:innen den zeitlichen Verlauf des Lernprozesses von Klassifikationsmodellen untersuchen können. Eine allgemeinere Analyse von hochdimensionalen, zeitlichen Prozessen ermöglicht der *Projection Path Explorer*, in dem Prozesse als Trajektorien in einem niedrigdimensionalen Einbettungsraum dargestellt werden. Der Projection Path Explorer nutzt Modelle des unüberwachten maschinellen Lernens zur nichtlinearen Dimensionalitätsreduktion. *Projective Latent Interventions* zeigen auf, wie diese unüberwachten Techniken adaptiert werden können, um Benutzer:innen mehr Kontrolle über – und ein besseres Verständnis für – die latenten Repräsentationen von Klassifikationsmodellen zu gewähren. Hierfür werden parametrisierte Erweiterungen von Dimensionalitätsreduktionstechniken eingeführt, dank derer Änderungen, die Benutzer:innen in der niedrigdimensionalen Darstellung vornehmen, auf das ursprüngliche Klassifikationsmodell zurückgeführt werden können. *ParaDime* ist schlussendlich ein Programmiergerüst, mit dem solche parametrisierten Dimensionalitätsreduktionsroutinen auf flexible und anpassbare Weise spezifiziert werden können. ParaDime vereinheitlicht bestehende Techniken und erleichtert Experimente mit neuen Projektionsmethoden, welche für Visualisierungen verwendet werden können. Diese fünf Arbeiten veranschaulichen die Bandbreite möglicher Kombinationen des maschinellen Lernens mit Visualisierung und zeigen, wie solche kombinierten Ansätze Benutzer:innen dabei helfen können, hochdimensionale und zeitliche Prozesse besser zu verstehen.

"... there is much pleasure to be gained from 'useless' knowledge."

— *A. Montagu & E. Darling,*
paraphrasing Bertrand Russell

ACKNOWLEDGEMENTS

First of all I would like to thank my supervisor Marc Streit. Marc took a leap of faith when he accepted me as a PhD student coming from a completely different field. Thanks to his expertise and excellent guidance, this endeavor turned out a success. Marc has fostered a great atmosphere for research in his lab and I thoroughly enjoyed my time as a PhD student under his supervision. I would also like to thank my co-supervisor Bernhard Kainz for his ideas and for broadening my research horizon. I wish my stay in London would have been under different circumstances, but the time I spent in his group was inspiring nonetheless.

Special thanks go to my fellow PhD students Patrick Adelberger, Vaishali Dhanoa, Klaus Eckelt, Christina Humer, Christian Steinparz, and Conny Walchshofer. Between the scientific discussions, the—sometimes "slightly" extended—coffee breaks, and our bouldering sessions they have become true friends. I am thankful for the time we have spent together and I hope it will continue this way. Additionally, I would like to thank all my collaborators who I have not yet mentioned, in particular Jürgen Bernard, Moritz Heckman, Benedikt Leichtmann, Martina Mara, Michael Pühringer, Holger Stitz, and Kai Xu, as well as all my colleagues at the ICG. I would also like to express my gratitude to all my old physics colleagues who were involved in one way or another in preparing me for my scientific journey.

Next, I want to thank my brothers Matthias and Michael. Usually I would try to think of something funny to put here, but I guess it is time for a heartfelt "Thank you!" I am truly happy to have them as companions. I am also indebted to Tia Truglas. She gave me the confidence to take this step and I will forever remain grateful for her support. Finally, I owe the most special thanks to my parents Barbara and Peter. They instilled in me the desire for knowledge and the ability to think critically. Thanks to their trust, love, and generosity I could pursue my path free from most worries.

This PhD was carried out within the *Human-Interpretable Machine Learning* project, a collaboration between Johannes Kepler University Linz and Imperial College London funded by the State of Upper Austria. Additional financial support by Johannes Kepler University Linz, the Linz Institute of Technology (LIT), the State of Upper Austria, and the Federal Ministry of Education, Science and Research (LIT-2019-7-SEE-117), by the Austrian Science Fund (FWF P27975-NBL and FWF DFH 23–N), by the Austrian Research Promotion Agency (FFG 851460 and FFG 854184), by the Wellcome Trust (IEH 102431 and EPSRC EP/S013687/1.), and by the Boehringer Ingelheim Regional Center Vienna is gratefully acknowledged.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1

1.1 VISUALIZATION & MACHINE LEARNING

In his 1962 landmark paper *The Future of Data Analysis*, statistician John Tukey wrote:

> All in all, I have come to feel that my central interest is in *data analysis*, which I take to include, among other things: procedures for analyzing data, techniques for interpreting the results of such procedures, [...] and all the machinery and results of (mathematical) statistics which apply to analyzing data. (Tukey, 1962, p. 2)

Tukey's multi-faceted vision of data analysis was one of the catalysts that spurred the "rebirth of data visualization". Under this slogan, psychologist Michael Friendly summed up the time between 1950 and 1975, when data visualization rose from a decade-long slumber (Friendly, 2008). Interestingly, the period that saw the "rebirth of data visualization" also coincides with several major milestones in the early development of machine learning (ML): Rosenblatt introduced the first perceptron in 1958 (Rosenblatt, 1958) and Werbos pioneered backpropagation in 1974 (Werbos, 1974). Arguably, the developments in both data visualization and early machine learning research around this time were driven by a joint goal—to better understand how to transform raw data into useful insights.

Both fields have since come a long way, but the desire to obtain insights from data has remained a common denominator of the fields of visualization ("vis") and machine learning. The strategies for insight acquisition, however, vary substantially between the two fields. Data visualization provides "visual representations of datasets designed to help *people* carry out tasks more effectively" (Munzner, 2014, p. 1, emphasis added). Machine learning, in contrast, "focuses primarily on learning (predictive) models from large sets of collected data, with the common goal of *automatizing* a certain task" (Ngo et al., 2022, p. 1, emphasis added).

Over the last decades it has become increasingly obvious that the visual, human-centered approach of vis and the automation-focused one of ML do not have to be competitors, but can benefit from each other. Works emerging from the mutual influence of visualization and ML have been categorized into *ML4Vis* and *Vis4ML*, depending on the main direction of influence.

1.1.1 *Vis4ML*

The recent surge in applications of deep learning has been met with an increased desire for model interpretability, and visualization has been identified as a vital tool in this context (Hohman et al., 2018). In addition to interpretability, Hohman et al. (2018) identified model comparison, model improvement, and teaching/education as the main motivations for Vis4ML

(*Why* in their interrogative classification). The use of visualizations for machine learning has proliferated to such an extent, that the body of work on the topic by now suffices for a survey of survey papers (Chatzimparmpas et al., 2020). In their meta-analysis of works covering mostly the last two decades, Chatzimparmpas et al. (2020) used latent Dirichlet allocation to generate 10 major topics related to the use of visualization for ML. Most pertinent to the content of this thesis are the topics of "model and clustering visualization for time-series data" and "dimensionality reduction (DR) and projections visualization". These topics roughly align with the modes "line charts for temporal metrics" and "dimensionality reduction *&* scatter plots" identified as *How* by Hohman et al. (2018). The importance of DR and temporality in the context of this thesis is outlined further below.

1.1.2 *ML4Vis*

Q. Wang et al. (2021, p.1) defined ML4Vis as the application of "ML techniques to solve visualization-related problems using the knowledge extracted from data". They identified seven visualization processes that can be supported by ML. In addition to processes that help with the automatic creation of figures or with analyzing user behavior, they highlight the role of "data processing for vis" (Q. Wang et al., 2021, p. 2). C. Wang and Han (2022) surveyed deep learning techniques for scientific visualization and categorized the works along various axes. For the research-task axis, they identified "data generation" and "feature learning and extraction" as two of the five main categories. Endert et al. (2017) had previously introduced a 2D categorization of research works related to integrating machine learning in visual analytics. They divided their algorithm axis into dimensionality reduction, clustering, classification and regression. Most of these subtasks introduced by Endert et al., in particular DR, are also part of the definitions of "data processing for vis" by Q. Wang et al. (2021) and of "data generation" and "feature learning and extraction" by C. Wang and Han (2022).

1.1.3 *The Special Role of Dimensionality Reduction*

As stated above, dimensionality reduction has been adduced both as an example for Vis4ML and ML4Vis. This makes DR a particularly interesting showcase of the cross-pollination between vis and ML. Dimensionality reduction has become such an integral part of both fields, that in its case the direction of influence between the two fields is no longer obvious. Ngo et al. (2022) recently reflected on their experience working at the intersection of vis and ML. As an example, they introduce a tool for visual parameter space analysis for dimensionality reduction. Users interact with a "meta map" that shows each DR technique with a certain hyperparameter setting as a point in 2D. Here, a visualization of data created with an ML algorithm (i.e., the meta map) helps users to pick another ML algorithm for later use in visualization or in yet another ML pipeline. This example shows how deeply connected the two fields, vis and ML, have become, and what special role dimensionality reduction can play in this connection.

1.1.4 *The Special Role of Temporality*

The importance of interaction for modern visualization systems has long been recognized (Pike et al., 2009; Thomas & Cook, 2005). Interacting with visualizations allows a user to progress from a static role as a viewer to an active participant in a feedback loop. Endert et al. (2017) ground their discussion of integrating ML with visual analytics on such formalized feedback loops. In particular, they refer to the sensemaking loop by Pirolli and Card (2005) and the human-computer knowledge generation model of Sacha et al. (2014). In a similar vein, S. Liu et al. (2017) and Sacha et al. (2017) introduced iterative pipelines specifically for working with DR. In all these formalized loops, a user's mental model is continually updated and new insights are fed back into the pipeline. Conceptually, this optimization of mental models is strikingly similar to the optimizations of algorithmic models that are ubiquitous in ML. A holistic analysis of outcomes of iterative ML processes—within sensemaking processes which are themselves iterative—is only possible by recognizing the special importance of sequential and/or temporal data created in these loops (Hinterreiter, Ruch, et al., 2022).

1.1.5 *The Model Analysis Process*

Figure 1.1 shows an adapted version of the visual analytics process model by Keim et al. (2010). In this general workflow combining vis and ML, raw data is first fed into a model. The model is optimized, creating a temporal sequence of derived data and/or insights (e.g., predictions or low-dimensional data representations). The derived data (potentially together with raw data) is presented to the user in an interactive visualization. Based on insights gained from the visualization, the user can go back and update the model and/or refine the original data. These updates can either be triggered via the visualization or performed directly by the user.

Note that the importance of temporality discussed above manifests itself in the explicit inner loop (i.e., the model optimization). The data created in this inner loop is typically high-dimensional, and, as explained above, dimensionality reduction is a well-established approach for tackling high-dimensional data both in ML and vis. The interaction model depicted in Figure 1.1 is tightly connected to the overarching research question of this thesis:

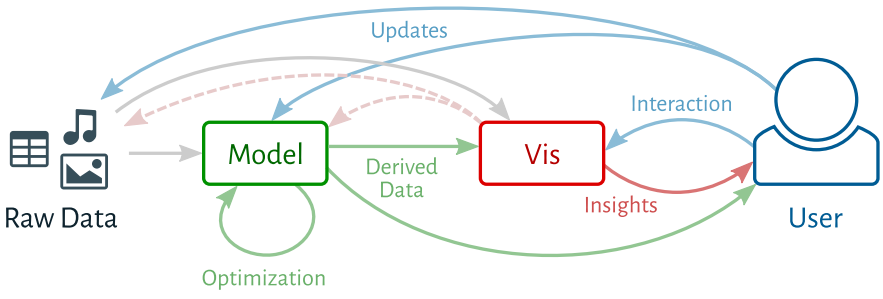


Figure 1.1: A model for the analysis of iterative processes with interactive visualizations, based on the visual analytics process model by Keim et al. (2010).

How can visualization, machine learning, and dimensionality reduction come together to provide users with a better understanding of high-dimensional, temporal processes?

In the following section, variations of this question, combined with the interaction, serve as the basis for a categorization of the individual contributions of this thesis.

1.2 ABOUT THIS THESIS

The core of this thesis is formed by several publications to which I contributed in a leading role (✒). This core is augmented with content extracted from other publications, to which I contributed as a co-author (✒). In this section, I list all these publications, summarize their content, detail my contribution to them, and explain how they fit into the bigger frame of the thesis. To this end, I refer back to the interaction model (Figure 1.1) and the overarching research question introduced in the previous section. Finally, I describe how the content of the publications relates to the chapters and sections of this thesis.

1.2.1 *Main Contributions*

The following publications form the core of this thesis. Each publication corresponds to a chapter of the thesis, as outlined in Section 1.2.3. The publications are listed here in the same order in which they appear in the thesis, and this order is mostly chronological. Note, however, that the publication dates do not necessarily reflect the chronological order due to differences in review times.

✒ **Hinterreiter**, **A.**, Ruch, P., Stitz, P., Ennemoser, M., Bernard, J., Strobelt, H., & Streit, M. (2022). ConfusionFlow: A model-agnostic visualization for temporal analysis of classifier confusion. *IEEE Transactions on Visualization and Computer Graphics*, *28*(2), 1222–1236. https://doi.org/10.1109/TVCG.2020.3012063.

ConfusionFlow is an interactive visualization tool for the combined temporal and comparative analysis of the performance of classification models. It extends the traditional class confusion matrices to allow a visualization of performance characteristics ("Derived Data" in Figure 1.1) over time. This way, ConfusionFlow acknowledges the temporal nature of the model optimization loop in the analysis process model. ConfusionFlow is model-agnostic and can be used to compare performances for different model types, model architectures, and/or training and test datasets. We apply ConfusionFlow in an active learning scenario and in the context of neural network pruning.

My contributions: preparation of manuscript; task analysis; study of related work; use cases for scalability and pruning.

✒ Pühringer, M., **Hinterreiter**, **A.**, & Streit, M. (2020). InstanceFlow: Visualizing the evolution of classifier confusion at the instance level. *2020*

IEEE Visualization Conference –- Short Papers. https://doi.org/10.1109/VIS47514.2020.00065.

Similar to ConfusionFlow, InstanceFlow allows the temporal analysis of the performance of classification models over time. However, it focuses on the level of individual instances rather than the level of class confusion. InstanceFlow presents instance-level data through glyphs and trajectories, while a Sankey-diagram visualizes class-level information. In a short case study, we show how a model developer might use InstanceFlow to determine and fix problems in the data distribution (see "Updates" edge in Figure 1.1) based on an interactive investigation of the sequential performance data.

My contributions: conceptual input; preparation of manuscript.

Hinterreiter, **A.**, Steinparz, C. A., Schöfl, M., Stitz, H., & Streit, M. (2021). Projection Path Explorer: Exploring visual patterns in projected decision-making paths. *ACM Transactions on Interactive Intelligent Systems*, *11*(3–4), Article 22. https://doi.org/10.1145/3387165.

The Projection Path Explorer resulted from the idea of creating a more generalized visualization of high-dimensional, temporal data derived from optimization loops. To this end, we employ collections of Time Curves (Bach et al., 2016)—trajectories through points in a low-dimensional embedding space. We discuss which patterns can emerge from drawing many such trajectories together, and we relate those patterns to the underlying high-dimensional processes. Using several examples from diverse application areas, we show that the Projection Path Explorer is not limited to data from optimization processes, but generalizes to a wide variety of high-dimensional data.

My contributions: conception and implementation of examples; description of patterns; mathematical description; preparation of manuscript.

Hinterreiter, **A.**, Streit, M., & Kainz, B. (2020). Projective Latent Interventions for understanding and fine-tuning classifiers. In J. Cardoso et al. (Eds.), *Interpretable and annotation-efficient learning for medical image computing. Proceedings of the 3rd workshop on interpretability of machine intelligence in medical image computing (iMIMIC 2020)* (pp. 13–22). Springer. https://doi.org/10.1007/978-3-030-61166-8_2. Best Paper Award at iMIMIC 2020.

Our experience with analyzing low-dimensional embeddings prompted an interesting research question: Can these low-dimensional representations be used to manipulate the underlying high-dimensional processes? This idea led to the development of Projective Latent Interventions (PLIs). PLIs allow ML model developers to retrain classification models by back-propagating manual changes made to low-dimensional embeddings of the latent space. The backpropagation is enabled via parametric extensions to existing DR techniques. With PLIs, users indirectly modify a machine

learning model by interacting with a visualization (central dashed edge in Figure 1.1).

My contributions: concept; implementation; preparation of manuscript.

Hinterreiter, **A.**, Humer, C., Kainz, B., & Streit, M. (2022). ParaDime: A framework for parametric dimensionality reduction. *arXiv: 2210.04582*. https://doi.org/10.48550/arXiv.2210.04582. To be submitted to *EuroVis '23*.

ParaDime is a framework for parametric dimensionality reduction (DR). It builds on the idea that the objective functions of several modern DR techniques result from transformed inter-item relationships. ParaDime provides a common interface for specifying these relations and transformations, thereby unifying parametric versions of several existing DR techniques. In ParaDime, each aspect of the DR is fully customizable, which makes it easy to experiment with novel DR techniques. With respect to the interaction model in Figure 1.1, ParaDime focuses on user control of the DR model, which may then be applied further in visualization or ML pipelines.

My contributions: concept; implementation of framework; documentation; preparation of manuscript.

1.2.2 *Secondary Contributions*

The following is a list of additional publications that are related to this thesis. I refer to these publications in different ways throughout the chapters. For details about where each publication is referenced, see Section 1.2.3. The publications are again listed in the order in which they appear in the thesis.

Steinparz, C. A., **Hinterreiter**, **A.**, Stitz, H., & Streit, M. (2019). Visualization of Rubik's cube solution algorithms. In T. v. Landesberger & C. Turkay (Eds.), *EuroVis workshop on visual analytics (EuroVA '19)*. The Eurographics Association. https://doi.org/10.2312/eurova.20191119.

Here we discuss visualizations of solution strategies for Rubik's cube. The visualization approach is the same one as that used in the Projection Space Explorer.

My contributions: preparation of manuscript; conceptual input.

Walchshofer, C., **Hinterreiter**, **A.**, Xu, K., Stitz, H., & Streit, M. (2021). Provectories: Embedding-based analysis of interaction provenance data. *IEEE Transactions on Visualization and Computer Graphics (Early Access)*. https://doi.org/10.1109/TVCG.2021.3135697.

Provectories are trajectory-based visualizations of user interaction data, which are analyzed within the Projection Path Explorer. We apply Provectories to data from two user studies to better understand the users' interaction histories. In one of the studies, users interacted with an AI-assisted visual analytics tool.

My contributions: conceptual input; implementation of data processing pipeline; preparation of parts of the manuscript.

Eckelt, K., **Hinterreiter**, **A.**, Adelberger, P., Walchshofer, C., Dhanoa, V., Humer, C., Heckmann, M., Steinparz, C. A., *&* Streit, M. (2022). Visual exploration of relationships and structure in low-dimensional embeddings. *IEEE Transactions on Visualization and Computer Graphics (Early Access).* https://doi.org/10.1109/TVCG.2022.3156760.

This work extends the Projection Path Explorer with additional visual encodings and interactions for exploring hierarchical structures.

My contributions: conceptual input; description of item/group relationships; preparation of parts of the manuscript.

Leichtmann, B., Humer, C., **Hinterreiter**, **A.**, Streit, M., *&* Mara, M. (2023). Effects of explainable artificial intelligence on trust and human behavior in a high-risk decision task. *Computers in Human Behavior*, *139*, 107539. https://doi.org/10.1016/j.chb.2022.107539.

In this collaboration with colleagues from the JKU Robopsychology Lab, we performed an online experiment to better understand how visual explanations of AI predictions influence trust and performance in a high-risk decision task.

My contributions: conceptual input; app design; preparation of parts of the manuscript.

Leichtmann, B., **Hinterreiter**, **A.**, Humer, C., Streit, M., *&* Mara, M. (2022). Explainable artificial intelligence improves human decision-making: Results from a mushroom picking experiment at a public art festival. *OSF Preprint.* https://doi.org/10.31219/osf.io/68emr. To be submitted to the *International Journal of Human–Computer Interaction.*

After the online study mentioned above, we conducted a replication study in an immersive, artificial forest environment within the context of a public art exhibition.

My contributions: conceptual input; app design; draft for game design; supervision of study and installation; preparation of parts of the manuscript.

Humer, C., **Hinterreiter**, **A.**, Leichtmann, B., Mara, M., *&* Streit, M. (2022). Comparing effects of attribution-based, example-based, and feature-based explanation methods on AI-assisted decision-making. *OSF Preprint.* https://doi.org/10.31219/osf.io/h6dwz. Submitted to the *27th Annual Conference on Intelligent User Interfaces (IUI '23).*

Finally, we conducted a second online study to compare the effects of three different explanation methods, reusing some of the gamification elements developed for the replication study.

My contributions: conceptual input; app design; draft for game design; preparation of parts of the manuscript.

1.2.3 *Structure*

The structure of this thesis is, for the most part, based on the publications listed above. In Chapter 2 I summarize existing research work that is related to several of the publications that form the core of this thesis. Additional related work that is specific to the individual contributions is part of the respective chapters. Chapters 3 and 4 discuss ConfusionFlow (Hinterreiter, Ruch, et al., 2022) and InstanceFlow (Pühringer et al., 2020), respectively. Chapter 5 introduces the Projection Path Explorer (Hinterreiter et al., 2021). Two of the application scenarios discussed in Section 5.4 of this chapter coincide with secondary contributions mentioned above: the visualization of Rubik's cube solutions (Steinparz et al., 2019) is the core of Section 5.4.1, while Section 5.4.4 is based on Provectories (Walchshofer et al., 2021). Section 5.5.5 of this chapter's discussion is based on Eckelt et al. (2022). Projective Latent Interventions are introduced in Chapter 6. Chapter 7 forms the basis of the ParaDime preprint (Hinterreiter, Humer, et al., 2022). Finally, Chapter 8 concludes the thesis. Here I discuss future challenges and I also reflect on experiences from interdisciplinary research collaborations with psychologists (Leichtmann et al., 2022; Leichtmann et al., 2023).

RELATED WORK

2

As outlined in Section 1.1, the analysis of temporal and high-dimensional data plays a major role at the intersection of ML and visualization. This chapter summarizes related work on time series visualization and dimensionality reduction that is relevant in the bigger frame of this thesis. More specific work related to the individual contributions is discussed in the respective chapters.

2.1 TIME SERIES VISUALIZATION

In data mining, ML, and visualization research, principal goals of temporal data analysis are the exploration, identification, and localization of temporal patterns (Fayyad et al., 1996; Fu, 2011; Miksch & Aigner, 2014). Task characterizations in the visualization community include the localization of single (i.e., atomic) values as the finest granularity of temporal analysis (Aigner et al., 2011; Andrienko & Andrienko, 2006). Building on these atomic localization tasks, most visualization techniques support the assessment of multiple points in time, enabling users to grasp higher-level temporal patterns. Depending on the application context, such patterns include trends, outliers, or periodic patterns (Bernard, 2015).

In this thesis, time series visualization is applied to the training of machine learning models, where the identification of trends plays a crucial role. A model's performance tends to increase throughout the learning process until it reaches a saturation value. Other frequently investigated patterns include outliers and anomalies (Bernard, 2015; Blázquez-García et al., 2022). The assessment of anomalies in the context of ML models is highly relevant but challenging. In contrast to trends and outliers, periodic patterns and cycles hardly play a role in the temporal analysis of classifier training.

2.1.1 *Trajectories in Time Series Visualization*

Visualizing how processes progress through different states (e.g., during model optimization) is equivalent to visualizing multivariate time series data. A dataset with two quantitative attributes that change over time can be readily visualized as a trajectory in the 2D data space, with time varying implicitly along the line. The DimpVis system, for instance, demonstrated the use of trajectories as an important factor in interactive displays of time series data (Kondo & Collins, 2014).

Trajectories naturally play an important role in visual analytics of movement (Andrienko & Andrienko, 2013). When many trajectories (also known as trails) are shown at once, visual clutter can be a problem. To address this issue, edge-bundling techniques (Ersoy et al., 2011; Holten & Wijk, 2009) have been adapted to trail visualization (Du et al., 2015; Hurter et al., 2013;

Peysakhovich et al., 2015). In order to gain additional insights from trajectories, numerous data mining techniques have been developed (Zheng, 2015).

Often the data to be visualized as trajectories has more than two dimensions. While some data attributes can be encoded in additional channels (e.g., line color, line width, or marker shape), dimensionality reduction becomes inevitable for more than just a few data dimensions.

2.1.2 *Similarity of Time Series*

Assessing the similarity between multiple time series has been studied extensively for various encodings of one-dimensional temporal data. This includes line charts, heatmaps/colorfields and horizon graphs (Gogolou et al., 2019), and image encodings (Z. Wang & Oates, 2015). However, these encodings of one-dimensional data cannot exhibit many of the interesting patterns that arise from trajectories. Such patterns are cycles, parallel segments, and/or points visited multiple times, and they result from the fact that—in case of the trajectory encoding—time varies implicitly along the path through multiple dimensions. From these patterns a notion of self-similarity follows (Bach et al., 2016), which is arguably one of the strengths of the trajectory encoding. Plotting multiple trajectories together enables the assessment of both similarity *between* time series and *self*-similarity of individual time series.

2.2 DIMENSIONALITY REDUCTION

A multitude of dimensionality reduction techniques has been developed, and their usefulness for data visualization has been studied (Engel et al., 2012; Espadoto et al., 2019; van der Maaten et al., 2009). Dimensionality reduction can be categorized broadly into linear and nonlinear techniques. Parametric extensions of the latter play an important role in the context of this thesis.

2.2.1 *Linear Techniques*

The oldest linear technique, principal component analysis (PCA), has been known for over a century, tracing back to Pearson (1901) or even Cauchy (Abdi & Williams, 2010). The survey by Cunningham and Ghahramani (2015) provides an excellent overview of the many linear techniques that have since been developed. Besides PCA, another "traditional" technique is multidimensional scaling (MDS) (Torgerson, 1952). Classical MDS is an eigenvalue problem with a close relationship to PCA (Cox & Cox, 2008; Williams, 2002). In contrast, *metric* MDS is a more general approach for finding a low-dimensional configuration of points whose pairwise distances best match those of the high-dimensional data.

2.2.2 *Non-linear Techniques*

Metric MDS, with its principle of comparing pairwise distances, is the intellectual predecessor of many modern, non-linear techniques, such as Isomap (Tenenbaum, 2000), SNE (G. E. Hinton & Roweis, 2002), t-SNE (van der Maaten

& Hinton, 2008), and UMAP (McInnes et al., 2018). Isomap tries to find a low-dimensional configuration based on geodesic (i.e., shortest-path) distances computed on a high-dimensional neighborhood graph (Tenenbaum, 2000). In SNE, Gaussian kernels are used to transform pairwise distances into neighborhood probability distributions for both the high- and low-dimensional data. These probability distributions are then compared using the Kullback–Leibler (KL) divergence (G. E. Hinton & Roweis, 2002). To avoid the so-called crowding problem in the resultant embeddings, t-SNE instead uses the more fat-tailed Student's t-distribution for computing the probabilities in the low-dimensional space (van der Maaten & Hinton, 2008). Finally, UMAP replaces the t-distribution with a modified Cauchy distribution and uses a cross entropy loss instead of the KL divergence (McInnes et al., 2018; Sainburg et al., 2021). The conceptual similarities of these non-linear DR techniques were highlighted in different contexts by Bengio et al. (2004), Böhm et al. (2022), and Agrawal et al. (2021). Recently, the relationship between t-SNE and UMAP has been subject to a lot of debate (Becht et al., 2019; Damrich et al., 2022; Damrich & Hamprecht, 2021; Kobak & Linderman, 2021).

2.2.3 *Parametric Techniques*

Apart from their conceptual similarities, many non-linear DR techniques share a practical limitation: they involve the calculation of pairwise distances. Adding new points to existing embeddings—a problem known as out-of-sample extension—usually requires recomputing the whole embedding. This drawback has been addressed by approximating embeddings with parametric functions, using kernel-based approaches (Bengio et al., 2004; Gisbrecht et al., 2012; Gisbrecht et al., 2015), mixture models and data imputation (de Bodt et al., 2019), or neural networks (Min et al., 2010; Sainburg et al., 2021; van der Maaten, 2009). Parametric versions of t-SNE and UMAP are examples of manifold learning (Bengio et al., 2013), a subfield of representation learning. The general idea of using neural networks to reduce data dimensionality, in particular with autoencoders (G. E. Hinton, 2006; G. E. Hinton & Zemel, 1993), predates the techniques mentioned above. Additionally, parametric, non-linear DR techniques based on neighborhood information are related to metric learning (Bellet et al., 2013; Kulis, 2012), where representations are created by learning features and distance metrics conjointly.

CONFUSIONFLOW

3

Classification is one of the most frequent machine learning (ML) tasks. Many important problems from diverse domains, such as image processing (K. He et al., 2016; Krizhevsky et al., 2017), natural language processing (Glorot et al., 2011; Nogueira dos Santos & Gatti, 2014; Socher et al., 2013), or drug target prediction (Mayr et al., 2018), can be framed as classification tasks. Sophisticated models, such as neural networks, have been proven to be effective, but building and applying these models is difficult. This is especially the case for multiclass classifiers, which can predict one out of several classes (as opposed to binary classifiers, which predict one out of two).

During the development of classifiers, data scientists are confronted with a series of challenges. They need to observe how the model performance develops over time, where the notion of *time* can be twofold: on the one hand, the general workflow in ML development is incremental and iterative, typically consisting of many sequential experiments with different models; on the other hand, the actual (algorithmic) training of a classifier is itself an optimization problem, involving different model states over time (see Figure 1.1). In the first case, comparative analysis helps the data scientists gauge whether they are on the right track. In the latter case, temporal analysis helps to find the right time to stop the training, so that the model generalizes to unseen samples not represented in the training data.

Model behavior can depend strongly on the choices of hyperparameters, optimizer, or loss function. It is usually not obvious how these choices affect the overall model performance. It is even less obvious how these choices might affect the behavior on a more detailed level, such as commonly "confused" pairs of classes. However, knowledge about the class-wise performance of models can help data scientists make more informed decisions.

To cope with these challenges, data scientists employ three different types of approaches. One, they assess single value performance measures such as accuracy, typically by looking at temporal line charts. This approach is suitable for comparing learning behavior, but it inherently lacks information at the more detailed class level. Two, data scientists use tools for comparing the performances of classifiers. However, these tools typically suffer from the same lack of class-level information, or they are not particularly suited for temporal analysis. Three, data scientists assess class-level performance from the class-confusion matrix (Sokolova & Lapalme, 2009). Unfortunately, the rigid layout of the classic confusion matrix does not lend itself well to model comparison or temporal analysis.

So far, few tools have focused on classification analysis from a combined temporal, model-comparative, and class-level perspective. However, gaining insights from all three points of view in a single tool can (1) serve as a starting point for interpreting model performances, (2) facilitate the navigation

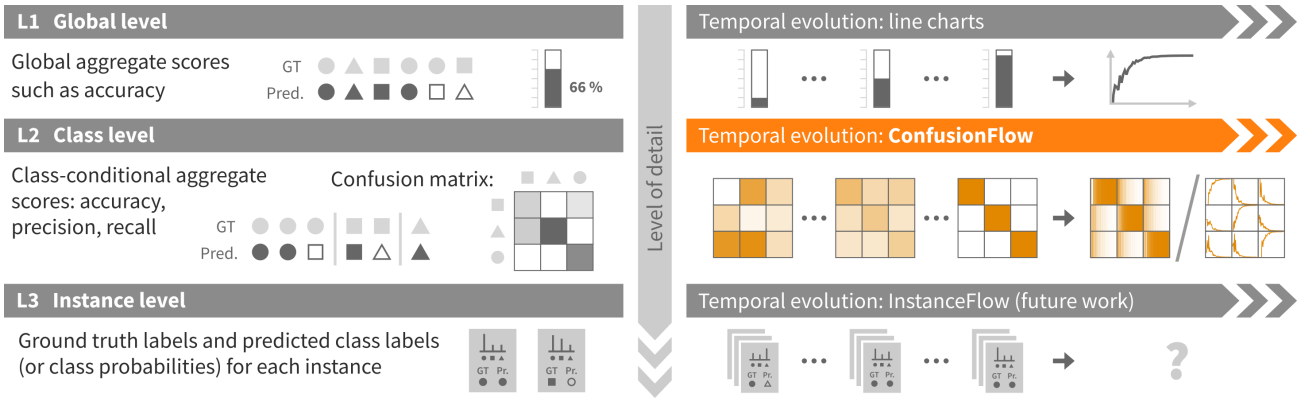


Figure 3.1: A classifier's performance can be evaluated on three levels of detail: global aggregate scores (L1); class-conditional scores and class confusion (L2); and detailed, instance-wise information (L3). ConfusionFlow operates on the class level L2, enabling a temporal analysis of the learning behavior.

through the space of model adaptations, and (3) lead to a better understanding of the interactions between a model and the underlying data.

The **primary contribution** of this chapter is *ConfusionFlow*, a precision- and recall-based visualization that enables temporal, comparative, and class-level performance analysis *at the same time*. To this end, we introduce a temporal adaptation of the traditional confusion matrix.

As **secondary contributions** we present (1) a thorough problem space characterization of classifier performance analysis, including a three-level granularity scheme; (2) a case study showing how ConfusionFlow can be applied to analyze labeling strategies in active learning; (3) an evaluation of ConfusionFlow's scalability; and (4) a usage scenario in the context of neural network pruning.

3.1 PROBLEM SPACE CHARACTERIZATION

The development of new classification models or the adaptation of an existing model to a new application domain are highly experimental tasks. A user is confronted with many different design decisions, such as choosing an architecture, a suitable optimization method, and a set of hyperparameters. All of these choices affect the learning behavior considerably, and influence the quality of the final classifier. Consequently, to obtain satisfying results, multiple classifiers based on different models or configurations need to be trained and evaluated in an iterative workflow. Here, we chose the term *configuration* to refer to the set of model, optimization techniques, hyperparameters, and input data. This design process requires the user to compare the learning behaviors and performances of different models or configurations across multiple training iterations. To this end, model developers typically inspect performance measures such as precision and recall.[1] Depending on the measures used, the analysis can be carried out on three levels of detail.

[1] For detailed definitions of the most common performance metrics, see Appendix A.1.

3.1.1 *Analysis Granularity*

Based on our reflection of related works (see Section 3.2), most performance analysis tasks for classifiers can be carried out on three levels of detail (see left part of Figure 3.1):

L1 *Global level*— At the global level, a classifier's performance is judged by aggregate scores that sum up the results for the entire dataset in a single number. The overall accuracy is a typical example for a global performance measure. For showing trends across multiple training iterations, global aggregate scores can be represented in simple line charts.

L2 *Class level*— At the class level, performance measures are derived from subsets of the results based on specific class labels. Typical performance measures at the class level are class-wise accuracy, precision, or recall. Like for the global level, the temporal evolution of these measures throughout training can be visualized as line charts. More detailed class-level information is contained in the confusion matrix. This work addresses the problem of visualizing the confusion matrix across multiple training iterations.

L3 *Instance level*— At the instance level, quality assessment is based on individual ground truth labels and predicted labels (or predicted class probabilities). This allows picking out problematic input data. Strategies for how to further analyze these instances vary strongly between different models and data types. Depending on the specific problem, interesting information may be contained in input images or feature vectors, network outputs, neuron activations, and/or more advanced concepts such as saliency maps (Simonyan et al., 2014).

These three levels are different degrees of aggregation of individual instance predictions. The right part of Figure 3.1 shows schematically how data at these levels can be visualized across iterations to enable an analysis of the training progress.

ConfusionFlow aims at a temporal analysis of per-class aggregate scores, introducing a visualization of the confusion matrix across training iterations. ConfusionFlow thus operates on the second of the three levels (L2).

3.1.2 *Analysis Tasks*

ConfusionFlow is designed for experts in data science and ML, ranging from model developers to model users. Building upon the reflection of these user groups from related work (see Section 3.2) and the characterization of the working practices of our collaborators, we break-down user goals and intents into a task characterization as follows. The principal structure of tasks is along two axes, which correspond to the two high-level goals of *comparative* analysis (G1) and *temporal* analysis (G2).

The comparative axis differentiates *within*-classification comparisons from *between*-classification comparisons. This frequently used within/between dichotomy accounts for analyses that are conducted with a single classification result (within), as well as those relevant for multiple results (between). The notion of within-comparison also alludes to the different levels of detail discussed in Section 3.1.1. All between-classification analyses share the principle of comparing multiple classification results. In general, users have a series of classification configurations at hand, aiming at identifying common-

alities and differences. According to our collaborators, multiple classification configurations can result from (a) different (hyper-)parameter choices of the same model, (b) different classification models, or (c) different datasets or dataset folds used for classifier training and validation. While the differentiation between these three different types of classification configurations is interesting from a methodological perspective, the requirements to tools for quality assessment are very similar.

The second axis structures the temporal analysis tasks (G2). Along this axis we differentiate between tasks as they are typically applied in time series analysis (Aigner et al., 2011): looking up values, assessing trends, and finding anomalies.

The complete crosscut of these two axes leads to six primary analysis tasks, T1 to T6, which are listed in Table 3.1. Fore each task, an exemplary scenario is given, which explains how the abstract low-level task relates to the ML analysis procedure.

In Section 3.1.1, we already hinted at the fact that existing tools mostly support temporal analysis (G2) only on a global level (L1). As our literature survey below shows, comparison between classifiers (G1B) is rarely supported on a class level. The main novelty of ConfusionFlow is enabling precision- and recall-based class-level analysis in a comparative *and* temporal way.

Figure 3.2 illustrates all possible scenarios of performing the high level tasks G1 and G2 at the same time, with special focus on between-classification comparison (G1B). This schematic shows the most general case, where the models ($C \Leftrightarrow D$) *and* the datasets ($X \Leftrightarrow Y$) are different, and the datasets additionally change over time ($X_i \neq X_j$). Many specialized comparison and/or temporal analysis tasks can be derived from the general case depicted in Figure 3.2, when either the dataset or the model are kept constant:

- In the simple case of observing how a single model is trained on a constant data set ($X_t = X$ for all t), the user is interested only in the sequence $C(\theta_1)(X) \cdots C(\theta_T)(X)$. This corresponds to the performance measuring (T1) and progress measuring tasks (T2).

Table 3.1: Analysis tasks relevant to the design of ConfusionFlow. The space of low-level tasks T1–T6 is generated by two axes of high-level goals, comparative (G1) and temporal analysis (G2), respectively.

	G1 Comparative analysis			
	A Within-classific. comparison	**B** Between-classification comparison		
G2 Temporal analysis		Different models	Different hyperparameters	Different data
A Lookup values	**T1** Measure performance	**T4** Compare performances		
	Read off quality measure at certain epoch	Assess final and intermediate model suitability	Relate final performance to hyperparameter space	Estimate final generalization capabilities
B Assess trends	**T2** Measure learning progress	**T5** Compare learning progress		
	Assess characteristic saturation curve for learning process	Compare learning speeds for different models	Relate learning speeds to hyperparameter choices	Identify over- or underfitting
C Find anomalies	**T3** Detect temporal problems	**T6** Relate temporal problems		
	Identify performance spikes and drops	Relate anomalies to model	Relate learning failure to parameter choice	Identify problematic instance sampling

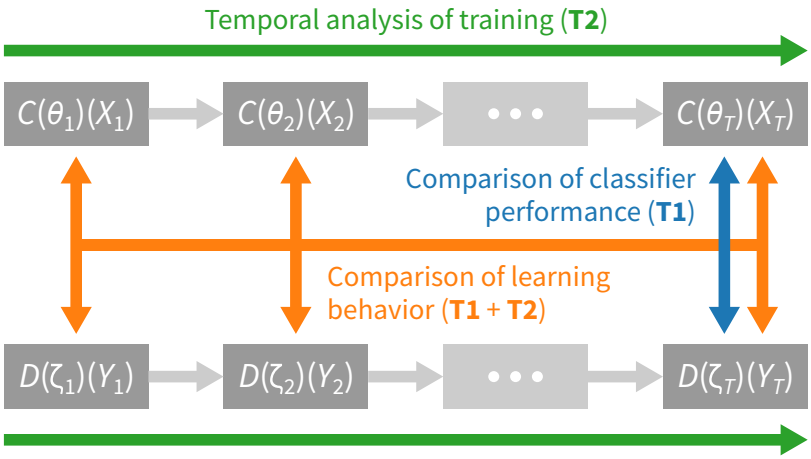


Figure 3.2: Principle working practice as observed with our collaborators. The two main goals of comparative and/or temporal analysis of classifiers led us to define the two axes that help to structure the analysis tasks. This schematic shows the most general case of different models and different, changing datasets.

- For comparing the final performances of two classification models, C and D, acting on the same test set X, the user analyzes the pair $C(\theta_T)(X)$ vs. $D(\zeta_T)(X)$. This is a typical realization of task T4.
- Often, the performances of a classifier on two different dataset folds, such as training and test folds, need to be compared temporally. This scenario implies $C = D$, and $X_t = X$ and $Y_t = Y$ for all t, but $X \neq Y$. The user now needs to compare the sequence $C(\theta_1)(X) \cdots C(\theta_T)(X)$ with the sequence $C(\theta_1)(Y) \cdots C(\theta_T)(Y)$. This analysis includes a comparative assessment of trends and anomalies (tasks T5 and T6).
- A more complex example is the comparison of two classifiers during active learning (see Section 3.4.1). In this case, both models are trained on the same dataset, but the dataset changes over time. The user compares the sequence $C(\theta_1)(X_1) \cdots C(\theta_T)(X_T)$ with the sequence $D(\zeta_1)(X_1) \cdots D(\zeta_T)(X_T)$. All fine-grained tasks T1 to T6 may be relevant in this complex example.

We deliberately kept all example scenarios in Table 3.1 level-agnostic. The tasks T1 to T6 are equally relevant on any of the three levels of detail. ConfusionFlow focuses on enabling users to perform the tasks on the class level L2, but also includes some global information deducible from the confusion matrix.

3.2 RELATED WORK ON MODEL ANALYSIS

The recent resurgence of ML in general and the increasing popularity of deep learning in particular have led to an increased demand for ML development and monitoring tools, but also to an increased desire to better understand existing techniques. This interplay between algorithm design on the one hand, and the challenge of making ML algorithms explainable or interpretable on the other hand, has spawned high activity in the field of visualization for ML. The survey of survey papers by Chatzimparmpas et al. (2020) gives a comprehensive overview of the recent research activity.

In the following, we discuss approaches that (1) target the user goal of comparison across models and/or configurations (see goal G1B); (2) enable a temporal analysis of the learning behavior (see goal G2); and/or (3) operate

on the class level (L2) as defined in our contextualization in Section 3.1.1. Table 3.2 summarizes which of these three aspects is covered by each of our selected publications. We also briefly review previous work on time series visualization and comparison, since ConfusionFlow is a small multiples approach, that should support users in performing typical temporal analysis tasks.

Our literature survey shows that hardly any tool so far has focused on simultaneously addressing—on the class level of detail (L2)—the two high-level goals of comparison (especially between models, G1B) and temporal analysis (G2).

3.2.1 *Model Comparison*

Gleicher et al. (2011) structured the design space for visual comparisons into three main categories: juxtaposition, superposition, and explicit representation. Most ML model comparison visualizations use superposition for plots of single-number performance metrics, and juxtaposition for comparison of multidimensional (e.g., vectors or matrices) or unstructured data (e.g., images or text).

One of the most well-known visualization systems for developing, debugging, and evaluating neural networks is TensorFlow's TensorBoard (Abadi et al., 2016; Wongsuphasawat et al., 2018). It combines a visualization of the computation graph with a display of various performance metrics, but it is not designed for comparing multiple ML models in the same view. TensorFlow also includes Google's What-If Tool (Wexler et al., 2019), which enables comparison of a model with a changed version of itself upon hypothetical changes to the dataset.

For certain types of model architecture, tools with specialized comparison features have been developed: RNNVis by Ming et al. (2017) for recurrent neural networks, GANViz by J. Wang et al. (2018) for generative adversarial networks, and CNNComparator by Zeng et al. (2017) for convolutional neural networks. RNNVis features a main view with a glyph-based sentence visualization. On demand, two models can be compared side by side. GANViz focuses on the comparison of the outputs of the generative network with those of the discriminative network that together make up the GAN. CNNComparator consists of a histogram of parameter values for a selected network layer, a matrix visualization of the convolution operation, as well as an instance-level side-by-side comparison of two selected networks' performances. It allows comparison of two different configurations or of two different model states for the same configuration, but does not feature immediate access to class confusion measures. ShapeShop by Hohman et al. (2017) is aimed at non-experts, and enables comparison of the performances of convolutional neural networks. It is designed to give the user a basic understanding of what the network learns, rather than provide in-depth evaluation functionality.

Zhang et al. (2019) presented Manifold, a model-agnostic framework for interpreting, comparing, and diagnosing ML models. Small multiples of

scatter plots visualize how two different models generate different class outputs. Color coding gives a sense of each model's class confusion, but there is no option to track the models' learning behaviors.

Comparison of two models cannot only be used to select which model performs better on its own. It can also be part of a workflow to construct new ensemble models or adapt models interactively. In BaobabView by van den Elzen and van Wijk (2011), decision trees can be pruned interactively, and the performances of the resulting tree can be compared to the initial one, for example, by looking at the confusion matrices. EnsembleMatrix by Talbot et al. (2009) displays confusion matrices for different classifiers, allowing the user to construct a weighted combination from among them. The resulting ensemble model can be evaluated, again in terms of class confusion.

Each of these techniques enables the user to compare the performances of multiple models or model states in some way (addressing goal G1B), but misses either the temporal aspect (G2), or does not yield class confusion information (L2).

3.2.2 *Temporal Analysis of Training*

TensorBoard (Abadi et al., 2016) and GanViz (J. Wang et al., 2018) augment their main visualization with line charts of accuracy or other performance scores. Similarly, Chung et al. (2016) show temporal training statistics in an extra window of their ReVACNN system for real-time analysis of convolutional neural networks. In CNNComparator (Zeng et al., 2017), limited temporal information is accessible by comparing two model states from different training epochs.

DeepEyes by Pezzotti et al. (2018) is a progressive visualization tool that combines curves for loss and accuracy with perplexity histograms and activation maps. Progressive line charts of loss during the training are also used in educational tools for interactive exploration, such as TensorFlow Playground (Smilkov et al., 2017) or GAN Lab (Kahng et al., 2019).

DeepTracker by D. Liu et al. (2019) displays performance data in a cube-style visualization, where training epochs progress along one of the three axes. A different approach to enable inspection of the learning behavior is a selector or slider which is linked to a main visualization or multiple visualizations in a dashboard and allows accessing individual iterations. Chae et al. (2017) made use of this technique in their visualization of classification outcomes, as did J. Wang et al. (2019) in DQNViz , a tool for understanding Deep Q-networks. In one of the views in ML-o-scope Bruckner (2014) an epoch slider is tied to a confusion matrix, in which cells can be interactively augmented with example instances. The Blocks system by Alsallakh et al. (2018) also features a confusion matrix bound to an epoch slider. Blocks supports investigation of a potential class hierarchy learned by neural networks, which requires the visualization to be scalable to many classes.

Of all the tools for exploration of the learning behavior (G2) mentioned above, none focuses on class confusion (L2) while also providing comparison functionality (G1B).

Table 3.2: Publications related to ConfusionFlow, classified by whether they allow between/classification comparison (G1B), offer temporal information (G2), and/or operate at the class level (L2).

System	Publication	G1B	G2	L2
RNNVis	Ming et al., 2017	✓		N/A
CNNComparator	Zeng et al., 2017	✓	✓	
GANViz	J. Wang et al., 2018	✓	✓	✓
DQNViz	J. Wang et al., 2019	✓	✓	N/A
TensorBoard	Abadi et al., 2016	✓	✓	
ReVACNN	Chung et al., 2016		✓	
DeepEyes	Pezzotti et al., 2018		✓	
DeepTracker	D. Liu et al., 2019		✓	✓
unnamed	Chae et al., 2017	✓	✓	✓
Blocks	Alsallakh et al., 2018		✓	✓
ML-o-scope	Bruckner, 2014		✓	✓
Confusion wheel	Alsallakh et al., 2014			✓
ManiMatrix	Kapoor et al., 2010			✓
Squares	Ren et al., 2017	✓		✓
BaobabView	van den Elzen and van Wijk, 2011	✓		✓
EnsembleMatrix	Talbot et al., 2009	✓		✓
Manifold	Zhang et al., 2019	✓		✓

✓ Covered ✓ Partly covered N/A Not applicable

3.2.3 *Class Confusion*

When evaluating the output of classifiers at level L2, class confusion can be interpreted in two ways. Typically, it describes the aggregate scores used in the individual cells of the confusion matrix. However, the term "between-class confusion" is sometimes also used to describe high probability values for more than one class in a classifier's output for an individual instance. In order to avoid ambiguity, we call this notion "per-instance classification uncertainty" in our discussion.

Of the works mentioned so far, BaobabView (van den Elzen & van Wijk, 2011), EnsembleMatrix (Talbot et al., 2009), ML-o-scope (Bruckner, 2014), and Blocks (Alsallakh et al., 2018) all allow, at least partially, performance analysis on the class level (L2). In these tools, this is realized by visualizing class confusion in terms of standard confusion matrices, either for the final classifier or for one training step at a time.

The confusion matrix is also the heart of the ManiMatrix tool by Kapoor et al. (2010), where it is used to interactively modify classification boundaries. This lets the user explore how constraining the confusion for one pair of classes affects the other pairs, aiming at class-level model optimization and interpretability.

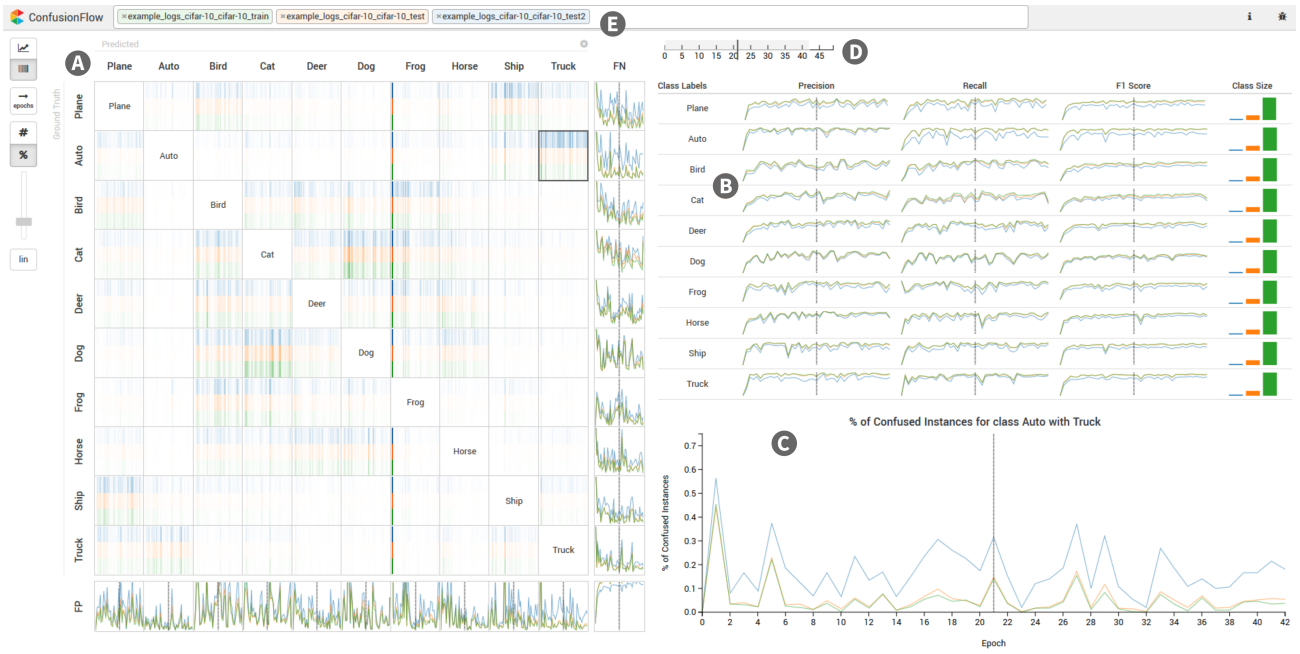


Figure 3.3: The ConfusionFlow matrix (A) visualizes confusion of classifiers across training iterations. Performance data for multiple classifiers can be loaded (E) and compared with each other. Additionally, class-wise performance measures and class distributions are displayed in a second view (B). The timeline (D) allows interactive exploration and selection of temporal regions of interest. On demand, plots can be expanded to the detail view (C). Here, we compare the performance of a neural network classifying images from the train set (■) and test set (■) of CIFAR-10 (Krizhevsky, 2009), and a recently proposed alternative test set (■) from CIFAR-10.1 (Recht et al., 2018), respectively. The line chart (C) shows that the relative number of misclassified images for the selected classes *auto* and *truck* deviates notably between the original and the new test set. For the remaining classes the classifier performs similarly on the new test set and the original CIFAR-10 test set.

Next to the confusion matrix, some alternative ways of evaluating classifier performance on level L2 have been proposed. Alsallakh et al. (2014) introduced the confusion wheel. It consists of a circular chord diagram, in which pairs of classes with high confusion counts are connected with thicker chords. On the outside, ring charts encode FN, FP, TP, and TN distributions for each class. Squares by Ren et al. (2017) is focused on visualizing per-instance classification uncertainty. Histograms of prediction scores can be unfolded to access individual instances, whose predictions are then encoded using parallel coordinates. Additionally, sparklines for each class give an impression of aggregate class confusion. Squares allows a hybrid-level (L2 and L3) confusion analysis.

None of the existing tools for class-level performance analysis (L2) provide an immediate, temporal representation of the learning behavior (G2), and most are relatively ill-suited for between-classification comparison (G1B).

3.3 CONFUSIONFLOW TECHNIQUE

The ConfusionFlow interface consists of three views, as illustrated in Figure 3.3: (A) the *ConfusionFlow matrix* presenting the confusion of one or more classifier(s) over time; (B) the *class performance and distribution view*, including plots of precision, recall, and F_1-score, as well as visualizations of the instances' class distributions; and (C) the *detail view* showing magnified plots of interactively selected confusion or performance curves. Additionally, ConfusionFlow features (D) a *timeline* for selecting the range of training steps that are used for exploration; and (E) an input field for loading datasets, which also serves as a legend for the whole visualization.

Figure 3.3 shows how ConfusionFlow can be used to compare the image classification performance of a neural network on different datasets. For this

example, we have loaded confusion data for a neural network image classifier trained on the training set (■) of CIFAR-10 (Krizhevsky, 2009), and evaluated on the images from the corresponding test set (■), as well as on a recently proposed new test set (■) from CIFAR-10.1 (Recht et al., 2018), respectively.

3.3.1 *ConfusionFlow Matrix*

The ConfusionFlow matrix, shown in Figures 3.3-A and 3.4, is a visualization of classification errors that supports the within- and between model comparison (G1) as well as temporal analysis (G2). In the classic confusion matrix, cell (i, j) lists the number of instances with ground truth label of class i that are classified as class j. While the classic confusion matrix is limited to showing confusion data for a single model at one specific time step, the ConfusionFlow matrix visualizes the class confusion for *multiple classifiers* over *multiple training steps* (see Figure 3.1). As described in Section 3.1.2, the different classifiers can come from ML experiments with different models or ML experiments with different datasets or dataset folds. The ConfusionFlow matrix is a small multiples approach: for each cell, the single value of the classic confusion matrix is replaced with a plot of the values for a selected time range (see Figure 3.4).

The ConfusionFlow matrix should enable temporal analysis (G2) and comparison (G1B) at the same time, while conserving the familiar layout of the confusion matrix. This means that the confusion visualization for each classification model should only take up little space, but should still be able to show a fine-grained temporal resolution. At the same time, individual temporal progressions for different models should be easily distinguishable to enable users to perform tasks T2 and T5. Accordingly, we chose the heatmap idiom for the visualization of class confusions (Bernard et al., 2019). One-dimensional heatmaps, sometimes called colorfields, have been shown to support the task of time series comparison well, particularly in terms of task completion time (Gogolou et al., 2019). The thumbnail at the top right of this paragraph shows how temporal confusion values for a single cell of

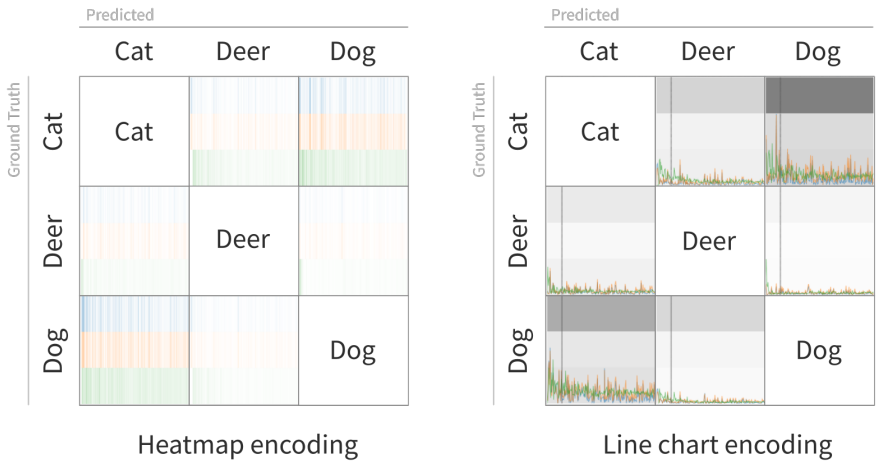


Figure 3.4: The ConfusionFlow matrix with its two different encoding options. Left: Stacked heatmap encoding (lasagna plot; Swihart et al., 2010). Right: Superimposed line charts with background heatmap corresponding to the selected iteration.

the matrix are encoded in this idiom. Each matrix cell contains one time series per loaded model. In a line chart encoding, multiple time series are plotted together in the same chart (superposition strategy, cf. Gleicher et al. (2011)). If each line is instead transferred to a one-dimensional heatmap, the heatmaps can be stacked for comparison without any overplotting issues (juxtaposition strategy (Gleicher et al., 2011)). These stacked heatmaps are sometimes called "lasagna plots" (Swihart et al., 2010). The confusion value is encoded as brightness, and a unique hue is automatically assigned to each classifier. To ensure visual consistency, the hue for each classifier is kept constant across all linked views. The left part of Figure 3.4 shows a matrix made up of such stacked heatmaps in a real example.

Users can interactively switch from the heatmap to a line chart encoding of class confusion. This option is available for two reasons. First, we found that ML users are particularly used to viewing temporal performance measures as line charts. Second, the line charts can facilitate reading off and comparing absolute values, either for individual epochs of interest (T1) or between models (T4).

If the line chart encoding is selected and the user additionally selects a single iteration (see Section 3.3.4 for information on the timeline selector), then a stacked heatmap of the confusion values for the selected iteration is plotted as background for the line charts. An example of this encoding is shown in the right part of Figure 3.4. The gray background heatmaps correspond to the confusion values for the three models at the time step indicated by the dashed vertical line. This additional background heatmap increases the visual saliency of problematic class-pairs for the line chart encoding This increased saliency is already inherent in the heatmap encoding. The heatmap/line chart switch, as well as several other controls for visual encoding options, can be seen to the left of the ConfusionFlow matrix in Figure 3.3.

To facilitate comparison of cells for a given predicted class, time by default progresses left-to-right, regardless of the encoding choice. This choice lines up with most users' expectations for plots of time series data. On demand, users can rotate the cell contents by 90° for easier comparison along a given ground truth class, if the heatmap encoding is selected.

The diagonal elements of the classic confusion matrix list the numbers of correctly classified instances for each class. For well-functioning classifiers, these numbers are typically much higher than the confusion counts. To keep the user's focus on the exploration of the error behavior and to keep lookup of confusion values feasible (T1 and T4), we decided to replace the diagonal cells in the ConfusionFlow matrix by class labels. In this way, we retain high visual contrast in the off-diagonal cells, and facilitate navigating through the matrix.

To let users assess the overall performance of individual classes, we show temporal plots of false negatives for each class in an additional column to the right, slightly offset from the matrix. Likewise, an additional row at the bottom shows the false positives for each class. We use the diagonal element at the intersection of the additional row and column to show the temporal

progression of the overall accuracy of the classifier(s). This allows the user to perform all analysis tasks T1 to T6 on a global level (L1)—especially when this chart is brought to the detail view (see Section 3.3.3).

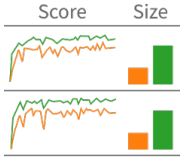
To enable performance comparison between datasets of different sizes (T4 to T6), such as training versus test sets, ConfusionFlow includes an option to switch from absolute to relative performance values. To obtain the relative performance scores, the confusion counts are simply divided by the total number of classified instances.

In order to address anomaly detection and comparison tasks (T3 and T6), peak values for problematic pairs of classes or training iterations should be visible and salient by default. However, these particularly high confusion counts can sometimes hide potentially interesting findings in other cells. To address this issue, we let users toggle between linear and logarithmic scales for the plots. Using an exponential scaling slider, lower values can be further accentuated. In the heatmap encoding, this corresponds to increasing the contrast.

If users are only interested in a subset of the class alphabet, they can narrow down the number of selected classes in a *class selection* dialog. In order to maintain a meaningful confusion matrix, a minimum of two classes need to be selected at all times. The number of displayed classes is not limited in terms of implementation, but there are some practical limitations which we discuss in depth in Section 3.4.2. *Class aggregation* is a second strategy to reduce the number of displayed classes. This strategy is currently not supported within ConfusionFlow and is only relevant for datasets with a distinct class/superclass hierarchy.

In case of the CIFAR-10 example shown in Figure 3.3, the ConfusionFlow matrix reveals that the confusion between classes *auto* and *truck* is considerably higher for the CIFAR-10.1 test set. Due to ConfusionFlow's focus on temporal analysis, it is immediately visible that this error is consistent across all training epochs (cf. tasks T4 and T5). For all other pairs of classes, the network generalizes well. In those cells, this can be seen by the similar brightness values for all three selected datasets. Without these class-level observations, the reason for the decrease in overall accuracy would remain unclear; by performing a comparative analysis with ConfusionFlow, the performance decrease can be traced back to changes in the underlying data distribution.

3.3.2 *Class Performance & Distribution View*



A thorough, class-level (L2) analysis of a classifier's performance should not only focus on pairs of classes, but also include an assessment of the general performance for *individual* classes. To this end, ConfusionFlow provides temporal (G2) line charts of precision, recall, and F_1-score for all selected classes (see Figure 3.3-B). In contrast to the ConfusionFlow matrix, horizontal space is not as limited for these plots, and visual clutter is thus less of an issue even in case of noisy data. For comparison between multiple classifiers (G1B), we superimpose the line charts. Again hue is used to differentiate

between different classifier results, and the hues are consistent with those chosen for the ConfusionFlow matrix. To let users assess the size and class distributions for each dataset, bar charts encode the number of instances per class. The bars are placed next to the per-class performance metrics and are colored consistently with all other views. A mouseover action reveals the number of instances for each bar.

In case of the CIFAR example (Figure 3.3), the class distribution charts reveal that the updated test set from CIFAR-10.1 is considerably smaller than that from CIFAR-10. The precision, recall, and F_1-score charts confirm that the recall for class *auto* and the precision for class *truck* are particularly bad for the new test set, but they also suggest that the classifier could not generalize well for *plane* instances from CIFAR-10.1. This assessment of generalization capabilities over multiple epochs is an example of task T5. While the class performance charts can lead to interesting insights on their own, they should also lead the user to re-examining the ConfusionFlow matrix.

3.3.3 *Detail View*

All cells in the ConfusionFlow matrix, as well as all per-class performance charts can be selected to be shown in greater detail in a separate view (see Figure 3.3-C). We visualize the temporal development of selected cells as line charts and superimpose the curves for multiple classifiers, keeping the hue for each model/configuration consistent. The detail view particularly addresses the tasks of pin-pointed temporal identification of problematic instances (T3 and T6) as well as reading off and comparing numeric values (T1 and T4), as space in the ConfusionFlow matrix is rather limited. Upon loading a new performance dataset, by default the overall (global) accuracy is shown in the detail view, as users are accustomed to this plot from many other performance analysis tools.

For the CIFAR example, the detail view confirms that, for a typical iteration, the confusion value of interest (*auto* vs. *truck*) is about twice as high for the updated CIFAR-10.1 test set than for the CIFAR-10 test and train sets.

3.3.4 *Timeline*

ConfusionFlow should aid users in exploring the error behavior of classifiers at different temporal granularities, involving single-time step tasks (T1 and T4) and multiple-time-step tasks (T2, T3, T5, and T6). Moving from a temporally rough to a more fine-grained analysis is facilitated by the timeline shown in Figure 3.3-D.

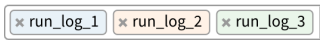
By default, the whole range of available iterations is selected upon loading performance data. Users can select a subset of iterations by dragging the left and right boundaries of the highlighted range in the timeline. All linked views, such as the ConfusionFlow matrix or the detail view, are updated automatically according to the selection. This range selection corresponds to a temporal zoom-in.

To support users in locating and comparing values for a specific time

step across multiple views, they can additionally select a single training step by clicking the iteration number below the range selector. A black line in the timeline marks the currently selected single iteration. The selected iteration is then dynamically highlighted by a vertical marker in all linked components. If the line chart encoding is selected for the ConfusionFlow matrix, the background heatmap is also updated, as described in Section 3.3.1.

The performance data for the example in Figure 3.3 spans 50 epochs. The user has selected the epoch range 0 to 42, and found an interesting peak for the confusion *auto* vs. *truck* at epoch 22 in the detail view.

3.3.5 *Dataset Selection*



As stated above, a unique hue is automatically assigned to each classification run upon loading the respective performance data. An input field with dynamic drop-down suggestions lets the user select from pre-loaded performance data for a number of classification configurations (see Figure 3.3-E). After the user made the selection, the input field serves as a legend for the visualization, representing each run with a colored box.

ConfusionFlow is a model-agnostic visualization technique. This means that the kind of data on which ConfusionFlow relies does not depend on the nature of the model. During training, only the classifier output for each instance needs to be logged after each iteration, and stored along with the ground truth labels. Along with our prototype implementation (see Section 3.3.6 below) we provide Python code examples for logging and exporting data for the commonly used ML frameworks TensorFlow and PyTorch.

In Figure 3.3, the input field serves as a legend to remind the user that performance data for the training (■) and test set (■) of CIFAR-10 (Krizhevsky, 2009), as well as the recently proposed new test set (■) from CIFAR-10.1 (Recht et al., 2018), have been loaded.

3.3.6 *Implementation*

ConfusionFlow is a server-client application based on the Caleydo Phovea framework[2] and the Flask framework[3]. The server side is written in Python and the client side is written in TypeScript using D3.js (Bostock et al., 2011). The code for ConfusionFlow—including the logging tools mentioned above—is available on GitHub[4]. A deployed prototype of ConfusionFlow with several pre-loaded example datasets is available online[5].

[2] Caleydo Phovea: `https://github.com/phovea/`

[3] Flask: `http://flask.pocoo.org`

[4] Repository: `https://github.com/ConfusionFlow/confusionflow`

[5] Prototype: `https://confusionflow.caleydoapp.org`

3.4 EVALUATION

To evaluate the usefulness of ConfusionFlow we describe three scenarios. First, we describe a case study which we performed in collaboration with ML researchers. Our collaborators used ConfusionFlow for visually comparing labeling strategies in active learning. In a second evaluation scenario, we analyze how ConfusionFlow scales to datasets with many ($\gtrsim$ 10) classes. A third use case shows how ConfusionFlow can be applied to study the effects of different neural network pruning strategies.

Figure 3.5: Visual comparison of instance selection strategies for effective data labeling. In an experiment, our collaborators tested three different labeling strategies: Greedy (■), Smallest Margin (■), and Dense Areas First (■). Using ConfusionFlow, our collaborators made a series of findings regarding the overall performances (A, D1, D2) as well as the temporal progression of class confusions (B, C, D3) for the different strategies.

3.4.1 *Case Study: Effective Labeling in Active Learning*

Labeled data is a prerequisite for supervised ML tasks. The labeling process requires human supervision to attach semantics to data instances. The challenge addressed in this case study refers to the instance selection problem: which instance should be selected next for labeling, in order to (a) improve a classifier most effectively, and (b) keep the effort of human labelers at a minimum?

Within the field of active learning (Settles, 2012), *Visual-Interactive Labeling* (VIAL) (Bernard, Zeppelzauer, Sedlmair, et al., 2018) is a concept to combine the strengths of humans and algorithmic models to facilitate effective instance selection. To this end, VIAL combines active learning with visual-interactive interfaces which enable humans to explore and select instances (Bernard, Hutter, et al., 2018).

One model-based active learning strategy used in this study is *Smallest Margin* (Wu et al., 2006), which always selects the remaining unlabeled instances with the highest classifier uncertainty. In contrast, one instance selection strategy frequently applied by humans is *Dense Areas First*, reflecting the idea that humans tend to select instances in the dense areas of the data (Bernard, Zeppelzauer, Lehmann, et al., 2018).

Our collaborators aim at making the labeling process a more effective, efficient, and human-friendly endeavor. Given that recent experiments confirmed that human-based strategies and model-centered strategies have complementary strengths (Bernard, Zeppelzauer, Lehmann, et al., 2018), our collaborators are interested in further analyzing the differences between strategies.

ConfusionFlow allows our collaborators to compare the model-based *Smallest Margin* with the human-based *Dense Areas First* strategy. As a third

strategy, a *Greedy* algorithm based on ground truth information serves as the theoretical upper limit of performance. For their analyses, our collaborators chose the MNIST handwritten digits dataset (LeCun et al., 1998) as an intuitive and well-established dataset that does not require domain knowledge. Another advantage of MNIST over other datasets is the ability of users to label most instances unambiguously.

Our collaborators use ConfusionFlow's temporal analysis capability to analyze and compare the labeling process over time. Accordingly, each training epoch corresponds to one labeling iteration (consisting of instances selection, labeling, and model re-training). Figure 3.5 shows how the collaborators compared the labeling processes of the three strategies visually (Smallest Margin, Dense Areas First, and Greedy).

The primary comparison goal (G1) of our collaborators is between models, accompanied by more detailed analyses of individual within-model characteristics. Due to the exploratory nature of the analysis, all three temporal analysis goals (G2) are relevant. As a result, the information requirements of our collaborators include all six analysis tasks (T1 to T6, see Table 3.1).

The Greedy strategy (Figure 3.5, ■) shows a steep performance increase (T2) at the beginning (A1), leading to more than 50 percent accuracy after only 10 iterations (T1). As the theoretical upper performance limit, Greedy has the strongest upward trend compared to the other strategies. It converges earlier (T5) and at the highest level of accuracy (T4). With only 50 labels the Greedy strategy already achieves almost 80 percent accuracy (T1).

Our collaborators also identified a known anomaly pattern in the accuracy curve (T3), which happens after ten instances (i.e., when all labels have been visited exactly once; see Figure 3.5-A2). This pattern is unique to the Greedy strategy (T6): with the eleventh label the training set becomes unbalanced, leading to a significant temporal decrease of classification accuracy. With ConfusionFlow, our collaborators could relate this anomaly to increased confusions between the classes 0, 4, and 9 (T4). Future repetitions of the experiment will clarify whether this effect can be related to the semantics of the particular classes or can be explained by other influencing factors.

The Smallest Margin strategy (Figure 3.5, ■) starts with a short and very early peak (instances 3 to 6) (T3) a pattern that the other two strategies do not show (T6). Thereafter, an almost linear upwards trend continues until instance 50 (T2), where the Margin strategy has almost 60 percent accuracy (T1).

In the ConfusionFlow matrix, our collaborators identified considerably high confusion values of class 8 (T1) with almost any remaining class (Figure 3.5-B1). This poor performance for class 8 is also clearly visible in the precision curve. An interesting pattern was the significant decrease of confusion between classes 0 vs. 8, roughly beginning at the 35th time step (B2). It seems that sometimes a single labeled instance can make a difference and support the accuracy of a classifier. Additionally, up to around instance 50, confusion values for class 2 are relatively high (T3), leading to many false positives for this class (B3).

The Dense Areas First strategy (Figure 3.5, ■) exhibits a minor but steady

increase in the early phase of the labeling process (T2). After 50 labels, the strategy achieves almost 55 percent accuracy T1. At a glance, Dense Areas First and Smallest Margin have similar overall accuracy curves (T5).

By inspecting the ConfusionFlow matrix, the analysts gained a series of different insights. Some class confusions lasted for the entire labeling process (T2) (1 vs. 3, 5 vs. 3, 2 vs. 6; see Figure 3.5-C1). Conversely, some class confusions seemed to have different levels of intensity during the labeling process (T2) (2 vs. 4, 7 vs. 4, 9 vs. 4). One class confusion even increased during the process (T2) (7 vs. 9), visible both in the matrix and in the FP chart (C2). Some training steps introduced peaks of confusion (T3) (class 6, roughly at instance 10). Finally, some classes did not suffer from considerable confusions at all (0, 1, 5, and 7).

One of the most interesting findings—according to our collaborators—was that the confusion patters for some pairs of classes differed considerably over time and between strategies. In case of the classes 9 vs. 4, for example, the confusions of the model-based Smallest Margin and the human-based Dense Areas First strategy show a strongly contrasting behavior (see Figure 3.5-D3). This observation strengthens our collaborators' view that strategies have complementary strengths.

As a result of their analysis with ConfusionFlow, our collaborators are motivated to perform in-depth follow-up analyses. They are particularly interested in using their class-level insights (L2) gained with ConfusionFlow as a starting point for drilling down to the instance level (L3). This way, they will be able to confirm general class-related patterns or trace back performance changes to individual images.

3.4.2 *Use Case: Scalability to Many Classes*

The traditional confusion matrix—as a class-level aggregation technique—scales well to large datasets with many instances. However, the confusion matrix organizes the inter-class confusion counts across *all pairs of dataset classes*, so datasets with more classes result in larger confusion matrices. Doubling the number of classes in a dataset reduces the available area for each cell in the confusion matrix by a factor of four.

The ConfusionFlow visualization idiom inherits these scalability issues from the traditional confusion matrix. In its current implementation, ConfusionFlow works well for classification problems with up to around fifteen classes. This is sufficient to support the majority of freely available datasets for multiclass classification[6]: a query on `www.openml.org` (Vanschoren et al., 2014) in February 2020 revealed that out of 434 datasets for multiclass classification problems, 368 datasets have fewer than 15 classes.

Still, given the popularity of certain benchmark datasets for image classification with higher numbers of classes—such as ImageNet (Deng et al., 2009) (1000 classes), Caltech 101 (Li Fei-Fei et al., 2004) and 256 (Griffin et al., 2007) (101 and 256 classes, resp.), and CIFAR-100 (Krizhevsky, 2009) (100 classes)—it is worth evaluating ConfusionFlow's scalability. To this end, we assess the characteristics of the ConfusionFlow matrix in the context of two class reduc-

[6] ConfusionFlow can also be applied to *binary* classification tasks, but the analysis of binary classifiers does not benefit much from ConfusionFlow's focus on the class level (L2).

tion strategies, class *selection* and class *aggregation* (cf. Section 3.3.1), when applied to an image classifier trained on the CIFAR-100 dataset.

The one hundred classes of CIFAR-100 are semantically divided into twenty superclasses. Each superclass groups five classes, e.g., the classes *fox*, *porcupine*, *possum*, *raccoon*, and *skunk* together make up the superclass *medium-sized mammals*. We chose this dataset as it already includes a proposed class aggregation scheme, making it suitable for an unbiased comparison between the two approaches for reducing the dimensions of the confusion matrix.

We trained a simple convolutional neural network to classify images from the CIFAR-100 dataset into one of the 100 classes, logging confusion data for 100 epochs. After each epoch, we evaluated the network's performance on the train fold as well as the test fold of the whole dataset. We determined superclass confusion values from the class predictions.

We studied class selection based on two different selection criteria: F_1 scores and off-diagonal confusion matrix entries. We first selected the ten classes with the lowest F_1-scores on the test set in the last epoch. Among these classes were many representing furry, brown or gray animals (*bear*, *otter*, *rabbit*, *kangaroo*, and *seal*).

In the ConfusionFlow matrix, we saw that these classes were frequently confused with each other (see Figure A.4 in the appendix). The remaining classes with low F_1-score were related to images of young humans (i.e., *boy*, *girl*, and *baby*). ConfusionFlow's rescaling functions helped to show that the confusion values for these three human-related classes are much higher than for the animal classes. This means that the poor performance for animals is spread out over many more classes than in the case of humans, as the final F_1-scores were similarly bad for all classes (T1).

While these findings could have been made without looking at temporal data, ConfusionFlow reveals that all these classes suffer from severe overfitting (T4). This overfitting was apparent in the precision, recall, and F_1-score plots of all classes as well as in the overall accuracy (see Figure A.2 in the appendix).

However, we could not find particular pairs of classes in this submatrix for which the confusion score got significantly worse over the course of training as a result of the overfitting. Using only this class selection mechanism, we could not assess whether overfitting was a general issue or affected certain pairs of classes more severely than others.

We thus moved on to evaluate a second class selection mechanism. We searched for the ten largest off-diagonal values in the confusion matrix for the test set and for the final training epoch. In the pairs of classes describing these cells, 14 classes were represented (see Figure A.5 in the appendix). Among these 14 classes are three different types of trees and four different types of flowers. The classes *boy* and *girl* are part of the list again, which matched our previous findings. The temporal aspect of ConfusionFlow reveals that the performance of the tree-related classes does not suffer as badly from overfitting as most other classes (T5).

Had we not chosen the CIFAR-100 dataset for its superclass structure,

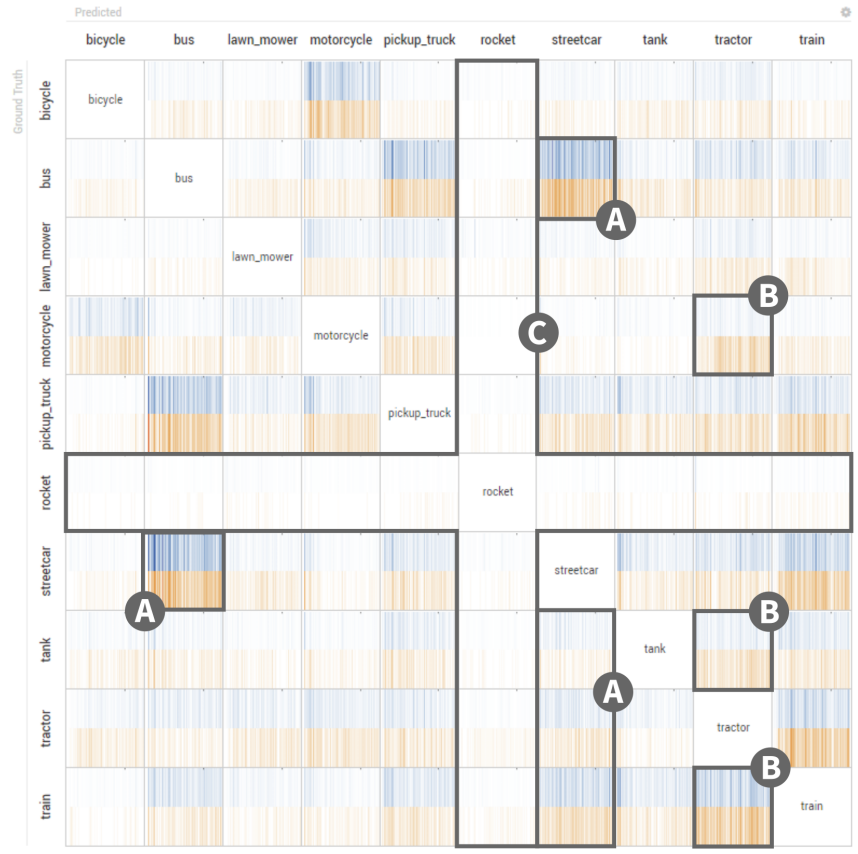


Figure 3.6: ConfusionFlow matrix for the ten classes making up the *vehicle 1* and *vehicle 2* superclasses for the train (■) and test folds (■) of CIFAR-100. *Streetcar* is more often confused with *bus* than with classes from its own vehicle superclass (a). Performance for *tractor* images suffers particularly from overfitting (b). Unsurprisingly, *rocket* seems to be hardly confused with any other vehicle (c), even though too long training causes some tractors and buses to be misclassified as rockets.

these results from the class selection strategy would have hinted strongly at the presence of a class hierarchy. While ConfusionFlow was not designed specifically for analyzing class hierarchies, it can help to build some intuition about their possible existence.

We resumed our evaluation by looking at the ConfusionFlow matrix for all 20 superclasses (see Figure A.3 in the appendix). We realized from the precision, recall, and F_1-charts that the performance on the test set for all superclasses except for *trees* gets worse compared to the performance on the training set (T5). This is in line with the results for the individual tree classes. It seems that the network has such a hard time to distinguish between different types of trees, that it is not capable of overfitting. From looking exclusively at the superclass confusion matrix, it would seem like this is an advantage, but obviously this behavior is only caused by generally high confusion *within* the superclass.

There are some superclass pairings that have large confusion values *between* superclasses with temporal characteristics that hint at overfitting problems. In particular, confusion values for the two vehicle classes *vehicles 1* and *vehicles 2* increase over time (T5).

With the insights gained from the class aggregation we could now go back and explore the fine-grained classes again. We looked at the ten classes making up the superclasses *vehicles 1* (*bicycle*, *bus*, *motorcycle*, *pickup truck*, and *train*) and *vehicles 2* (*lawn-mower*, *rocket*, *streetcar*, *tank*, and *tractor*). The ConfusionFlow visualization for these ten classes is shown in Figure 3.6.

The confusion values for the pairs of classes within *vehicles 1* are generally much higher than those for pairs of classes within *vehicles 2* and pairs of classes across the two superclasses. A strong exception to this rule is class pairings featuring the *streetcar* class (see Figure 3.6-A), which hints at a flawed definition of superclasses. It seems appropriate to swap the superclass memberships of *train* and *streetcar*.

Again, the temporal aspect of ConfusionFlow was helpful in assessing problems caused by overfitting. In particular, the performance of *tractor* gets worse over time (T5), which is most likely related to confusion with *train* and *tank* (see Figure 3.6-B. Interestingly, while the network is generally good at distinguishing *rockets* from other vehicles, too long training causes some *tractor* and *bus* images to be classified as *rockets* (Figure 3.6-C). These temporal anomalies (T3) would have been hard to detect from a single "snapshot" of the sparse confusion matrix.

3.4.3 *Use Case: Neural Network Pruning*

Neural networks are often heavily over-parameterized and contain millions of weights whose values are irrelevant for the network's performance. One technique for removing redundant parameters from a neural network is called *pruning*. During learning, connections in the network are successively removed (pruned) according to different selection strategies. Successful pruning results in a compressed model that retains its accuracy while requiring fewer computations and less memory. This facilitates deployment in embedded systems (Han et al., 2015) and sometimes even leads to faster learning and better generalization than the initial dense model (Frankle & Carbin, 2018; Z. Liu et al., 2018).

We examined the performance of several fully connected networks with different architectures trained on the Fashion-MNIST dataset (Xiao et al., 2017). This dataset consists of grayscale images of fashion objects organized in 10 classes (*trouser*, *pullover*, *sandal*, etc.). Specifically, we investigated the effects of pruning a neural network and re-initializing the resulting sparse network with the initial weights as described by Frankle and Carbin (2018). Using ConfusionFlow's temporal-comparative analysis features, we tried to get a better understanding of how removing certain weights affects the model's ability to distinguish between classes.

Figure 3.7 shows the ConfusionFlow visualization for three different networks trained on the Fashion-MNIST dataset. The original network (■) had 6-layers, each with 200 hidden units and ReLU activation functions. The learning rate was 0.012, with a batch size of 60. In the second network (■), 20 percent of the connections were removed at each epoch (this approach is called *online* pruning). The sparse network resulting after 15 epochs was then re-initialized with the same weights as the original dense network, and re-trained from scratch (■). In this network less than 4 percent of the original connections remain.

It is immediately obvious from the overall accuracy plot (Figure 3.7-A) that the training of the original model (■) fails completely after 10 epochs (T3).

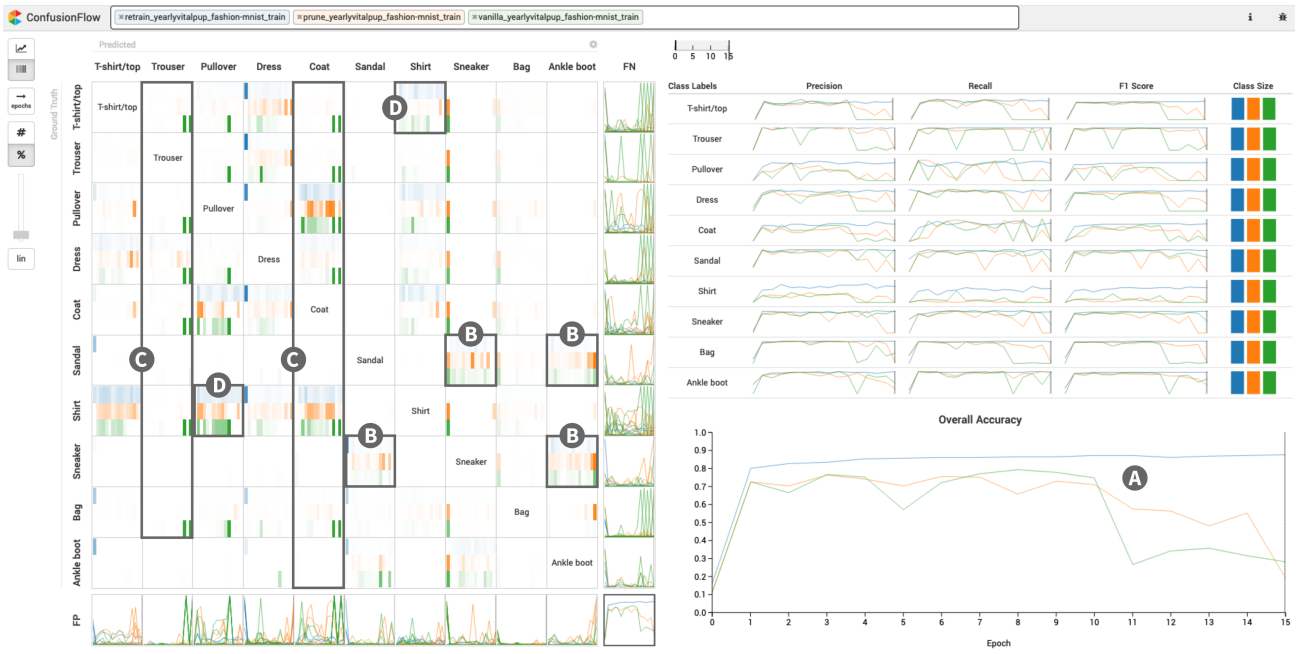


Figure 3.7: Visual comparison of different neural network pruning strategies. An original network (■), a pruned network (■), and a re-initialized sparse network (■) were trained to classify Fashion-MNIST images. ConfusionFlow reveals how the accuracy drop after 10 to 14 epochs (A) relates to confusions for different pairs of classes (B–D). The learning behavior of the re-initialized sparse network is much more stable compared to that of the other two models.

Training of the online-pruned network (■) fails slightly later (T6). The performance of the re-initialized sparse network (■), however, remains high. Remarkably, it even performs better than the other two networks right from the start (T5). ConfusionFlow allows relating this global (L1) accuracy improvement to pairwise class confusions.

Inspection of the ConfusionFlow matrix shows that the confusion counts for all shoe-related pairs of classes (*sneaker* vs. *sandal*, *ankle boot* vs. *sneaker*, etc.) increase considerably during later epochs for the non-pruned and online-pruned networks (Figure 3.7-B). The re-initialized sparse network, on the other hand, continues to learn to better distinguish between these classes (T5). Another reason for the complete failure of the original network seems to be related to the classes *trouser* and *coat* (see Figure 3.7-C), with extremely high FP values for these two classes in two of the later epochs (T6).

Even though the global accuracy plot showed a pronounced accuracy drop for the non-pruned and the online-pruned networks, both models retain an accuracy of about 30 percent (T1). The ConfusionFlow matrix reveals that this remaining accuracy in the later epochs is related to a better performance for the pairs *shirt* vs. *T-shirt/top* and *pullover* vs. *shirt* (Figure 3.7-D). The better generalization of the re-initialized sparse network across other classes comes at the cost of higher confusion values for images showing different kinds of upper body garments.

These findings demonstrate that ConfusionFlow allows a more nuanced evaluation of classifier performance, enabling the user to trace accuracy changes back to the class level.

3.5 DISCUSSION

In this section, we summarize additional findings from the use cases and insights about ConfusionFlow from discussion with our collaborators.

3.5.1 *Visual Design*

Our collaborators found ConfusionFlow easy to use and appreciated that its visual design incorporated two visualization idioms that they were already acquainted with: the confusion matrix and temporal line charts. The familiar layout of the confusion matrix along with the consistent color-coding of models helped our collaborators to navigate their analysis through the information-rich display. They mentioned, however, that in the case of more classes or more models they would welcome additional focus + context capabilities, perhaps similar to those provided by tools such as LiveRAC (McLachlan et al., 2008).

3.5.2 *Comparative & Temporal Analysis*

The different scenarios in our case study on active learning and our scalability evaluation revealed a general strength of the combined temporal and comparative analysis capabilities (G1 + G2) in ConfusionFlow, in particular with regards to the class level (L2). As classifiers learn, the off-diagonal elements of the confusion matrix tend to get sparse. Temporally fine-grained learning—such as model updates after each mini-batch or even after individual instance, as is the case in active learning—can result in confusion matrices of final classifiers that may not be representative of general patterns. Our collaborators appreciated ConfusionFlow's temporal-comparative approach, as it enabled them to identify temporally stable patterns. We found the same aspect useful in our scalability study, where the high number of classes leads to particularly sparse matrices. Furthermore, looking at the class level reveals how likely the model is to fluctuate. Global-level performance measures tend to hide this susceptibility of models to random or systematic changes during the training, as they average over all classes.

3.5.3 *Scalability*

Our scalability study with the CIFAR-100 dataset showed that ConfusionFlow can be used to analyze up to about 20 classes, although for more than 15 classes, screen resolution and label placement start to become limiting factors. It was still possible to derive insights from the 20-superclass confusion matrix, which could be used downstream in a subsequent class-level analysis.

3.5.4 *Bidirectional Analysis*

Typically, users apply ConfusionFlow after they already went through a reasoning process about a model's potential usefulness. With enough prior knowledge about the motivations behind the initial models, insights gained

with ConfusionFlow can be directly used to "go back" and optimize these models—resulting in a *bidirectional* analysis and development workflow.

However, in most cases true bidirectionality requires further tools. The three scenarios presented in this section showed that the specific requirements for a holistic analysis are domain-specific and/or model-specific (keep in mind that ConfusionFlow is fully model-agnostic). Additionally, in some cases instance-level information (L3) is required.

After the discussions with our collaborators, we see ConfusionFlow as one step in a bidirectional, iterative analysis workflow. We see its main strength in that it provides temporal and comparative insights which can serve as additional mental inputs for a more application-specific analysis. Examples of tools that could be used in conjunction with ConfusionFlow are LIME (Ribeiro et al., 2016) for instance-based explanations and Blocks by Alsallakh et al. (2018) for exploring class hierarchies. Chapter 4 introduces InstanceFlow (Pühringer et al., 2020), a temporal visualization that operates on the instance-level and that serves as an analysis springboard similar to ConfusionFlow. Our collaborators showed particular interest in such an instance-level tool for further continuing their analysis of selection strategies based on insights gained with ConfusionFlow.

3.6 CONCLUSION

In this chapter we introduced ConfusionFlow, a tool for visualizing and exploring the temporal progression of classifier confusion. ConfusionFlow combines a visualization of the confusion matrix over time with charts for global and per-class performance metrics. We evaluated the usefulness of ConfusionFlow's interactive exploration capabilities by means of a case study on instance selection strategies in active learning. Furthermore, we analyzed ConfusionFlow's scalability and presented a use case in the context of neural network pruning.

ConfusionFlow was not designed as a catch-all, standalone tool, but to be used in conjunction with other tools and visualization components. In particular, we plan to complement ConfusionFlow's class-level information with a novel visualization tool focused on temporal observation of instance-level confusion. However, by offering model comparison and temporal training analysis at the class level, ConfusionFlow can fill an important gap in an ML workflow towards understanding and interpreting classification models.

INSTANCEFLOW

4

As discussed above, the application of increasingly complex machine learning models to real-world problems has led to a growing interest in visualizations for post-hoc model explainability (Barredo Arrieta et al., 2020; Chatzimparmpas et al., 2020; Hohman et al., 2018). One of the most important supervised ML tasks, with a wide variety of application areas, is classification. In the previous chapter on ConfusionFlow (Hinterreiter, Ruch, et al., 2022), we established that the performance of classification models can be analyzed and visualized at three levels of detail globally (L1), at the class level (L2), and at the instance level (L3). We also discussed that existing approaches often focus on fully trained models and disregard the temporal evolution that led to this final model state. Tools that enable temporal performance analysis are typically limited to global, single-value performance measures (Ferri et al., 2009).

With ConfusionFlow, we argued that extending a temporal performance analysis to the class-level can lead to new insights (Hinterreiter, Ruch, et al., 2022). Similarly, a temporal drill-down to the *instance* level can help model developers to distinguish stable (mis)classification patterns from stochastic effects caused by the partially random training.

The main contribution of this chapter is InstanceFlow, a visualization that combines aggregated temporal information in a Sankey diagram with detailed traces of individually selected instances. Instances of interest can be located via a tabular view that allows users to rank and filter instances by several temporal difficulty measures. With this dual approach, InstanceFlow aims to bridge the gap between class- and instance-level analysis of the learning behaviors of classification models.

4.1 USER TASKS

InstanceFlow focuses on a temporal analysis of instance-level classification performance. Such an analysis can concentrate either on exploring instance-based properties of certain *epochs*, or analyzing the temporal characteristics of individual *instances*. Consequently, we organize the user tasks addressed by InstanceFlow according to whether they are epoch- or instance-focused (see Table 4.1).

We based the individual user tasks on a survey of existing instance-level visualizations (Section 4.2)—with a focus on filling gaps related to model agnosticism and temporal analysis—and on discussions with ML researchers in our previous work (Hinterreiter, Ruch, et al., 2022). InstanceFlow primarily addresses model developers and builders (cf. *Who?* in Hohman et al. (2018)) who seek to better understand the training process.

The instance-focused tasks (IT) are concerned with finding instances which are hard to classify correctly (IT1) or whose predictions evolve un-

Table 4.1: User tasks addressed by InstanceFlow, categorized by their focus on epochs (ET) or instances (IT).

Task	Description
IT1	Find *difficult* instances
IT2	Trace an instance's *classification history*
IT3	Analyze whether an instance *visits many or few* classes
IT4	Find instances *oscillating* between classes
ET1	Assess *class distributions* for a given epoch
ET2	Find momentarily *wrong and/or correct* instances
ET3	Find instances that *stay* in their class or *move* between classes between epochs

usually (IT2–IT4). This allows users to assess temporally (un)stable characteristics of the model or detect potentially mislabeled input data. The epoch-focused tasks (ET) are related to analyzing epoch-wise class distributions (ET1) or locating problematic epochs (ET2, ET3). Problematic epochs are those for which weight or parameter changes produce a non-beneficial outcome, such as increased confusion between two critical classes.

4.2 RELATED INSTANCE-LEVEL APPROACHES

Previous work on visualizing instance-level information in ML has focused mostly on model-dependent parameters such as the activation of neurons in deep neural networks in response to a given input instance. In many cases, the visualizations concentrate on the behavior of individual layers of the networks (Chung et al., 2016; Kahng et al., 2017; Pezzotti et al., 2018; Zhong et al., 2017), particularly the convolutional layers of CNNs (Bruckner, 2014; D. Liu et al., 2019; Zeng et al., 2017). Similar visualizations exist for GANs (J. Wang et al., 2018) and Deep Q-networks (J. Wang et al., 2019). Most of these model-specific approaches are further limited because they provide information for only one training iteration at a time.

Likewise, visualizations that focus on the performance analysis of classifiers typically do not truly enable temporal analysis. Chae et al. (2017) showed instance-wise predictions and aggregated distributions; Alsallakh et al. (2018) focused on class confusion with basic drill-down functionality to explore instances. In both cases, limited temporality is achieved via single-epoch selection sliders.

Squares by Ren et al. (2017) is closely related to our work in terms of visual design and the type of information shown. Users can switch between aggregated prediction distributions and a fine-grained instance-wise visualization that uses rectangular glyphs. However, Squares only shows the final model predictions.

InstanceFlow aims to enable a true temporal performance analysis at the instance level. It is a direct companion to ConfusionFlow (Hinterreiter, Ruch, et al., 2022), which uses an adaptation of the confusion matrix to enable

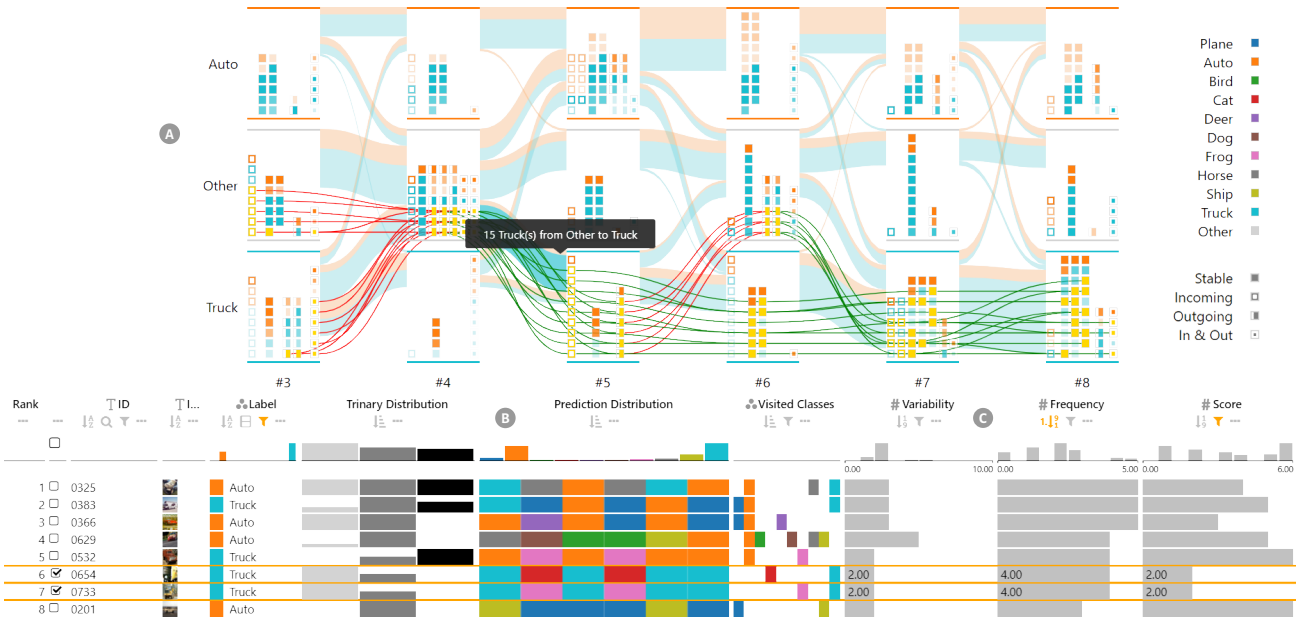


Figure 4.1: InstanceFlow visualizes the evolution of a classifier's predictions throughout the training process on an instance level. The *Flow View* (A) shows all instances and their corresponding class association as rectangular glyphs. A Sankey diagram shows the fractions of instances moving between classes. Additionally, the traces of single instances can be highlighted. The *Tabular View* (B) of the instance predictions over time along with custom performance scores (C) allows finding, ranking, and grouping instances.

temporal class-level analysis. Visually, InstanceFlow combines a multiform Sankey diagram like those used in VisBricks (Lex et al., 2011) and StratomeX (Lex et al., 2012) with a sortable, aggregatable tabular view (cf. Table Lens by Rao and Card (1994), LineUp by Gratzl et al. (2013), and Taggle by Furmanova et al. (2020)).

4.3 INSTANCEFLOW TECHNIQUE

The InstanceFlow interface consists of two main components, as illustrated in Figure 4.1: The *Flow View* (A) shows a Sankey diagram of the model's instance predictions throughout the selected training epochs; the *Tabular View* (B) lists detailed temporal instance information including performance scores (C).

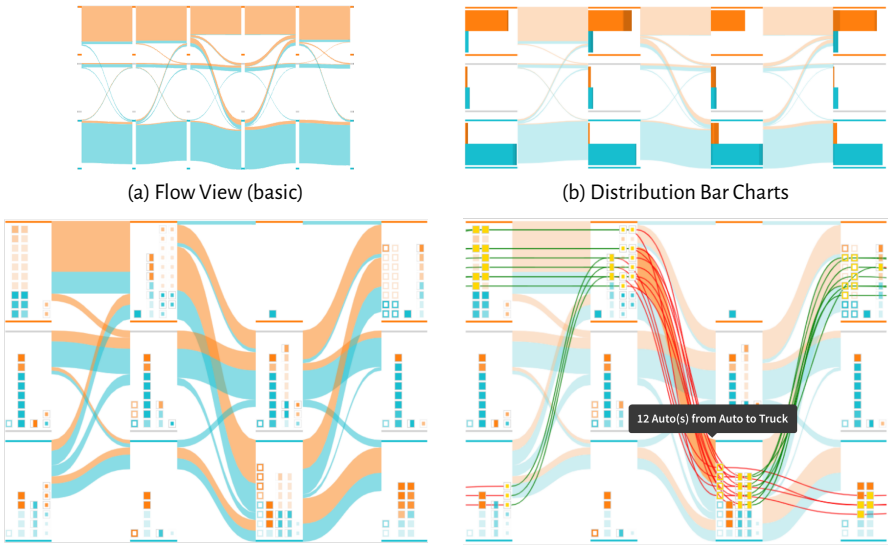
The Flow View supports different levels of granularity. In its basic form, the Flow View visualizes "class changers" in a Sankey diagram. *Distribution Bar Charts* emphasize the proportions of correctly versus incorrectly classified instances. At the finest granularity, *Instance Glyphs* encode each individual sample, with *Instance Traces* connecting the instances to reveal their classification history, that is, their "paths" through the epochs.

The Tabular View lists all instances along with their associated predictions over time and allows finding, ranking, and grouping instances via custom instance-level performance measures.

The Flow View and Tabular View are fully linked, such that traced or selected instances are highlighted in both views simultaneously.

4.3.1 *Flow View*

To visualize the overall flow of instances between classes, we use the well-established Sankey diagram. The flow visualization can be seen in Figure 4.2a,



(a) Flow View (basic)

(b) Distribution Bar Charts

(c) Instance Glyphs

(d) Instance Traces

Figure 4.2: Flow View: (a) the basic Flow View only consists of the Sankey diagram; (b) Distribution Bar Charts emphasize the class distribution; (c) Instance Glyphs show the underlying instances; and (d) Instance Traces reveal individual paths through epochs.

where the x-axis denotes the epoch and the y-axis denotes the predicted classes. The thickness of each band in the diagram encodes how many instances move from one class to another in the following epoch. The user selects classes of interest, and each class is assigned to a vertical region in the Sankey diagram. All non-selected classes are aggregated as "Other" and also assigned to a dedicated vertical region. The range of epochs to be visualized can be selected via an epoch slider. Hovering over a section of the Sankey diagram reveals the exact number of instances moving between the corresponding classes. Clicking on a section of the Sankey diagram selects those instances.

Distribution Charts To emphasize the class distributions in each epoch, horizontal Distribution Bar Charts, placed between the Flow visualizations, can be switched on (see Figure 4.2b). While this information is already implicitly encoded by the thicknesses at the borders of each band in the Flow View, the horizontal bar charts facilitate a quantitative comparison.

Instance Glyphs For a more detailed view, the individual instances can be represented by rectangular glyphs (see Figure 4.2c). The color of an Instance Glyph denotes the actual class of the instance (e.g., ■ and ■ in Figure 4.2). The shape indicates whether the instance prediction is temporally stable (■), coming from a different class (□), leaving for a different class (▣), or coming from and leaving for different classes (▮). The horizontal positions encode the same information, with glyphs for incoming instances placed at the left, outgoing ones at the right, and stable ones at the center. To visually rank the instances by their "importance", the opacity and the vertical position of each glyph together encode one of the calculated numerical difficulty measures described in Section 4.3.2.

Instance Traces To allow users to track the paths of specific instances throughout the training epochs, their traces can be visualized as lines con-

necting the corresponding instance glyphs (see Figure 4.2d). The color of an Instance Trace indicates whether the instance is moving to the correct (—) or incorrect (—) class. Instance Traces are only shown for instances selected by clicking on an Instance Glyph or a section of the Sankey diagram, or chosen from the Tabular View.

4.3.2 *Tabular View*

The per-class distribution flow is effective for finding anomalies in the learning process, but recognizing specific instances can be difficult due to the high information density. To facilitate the tasks of identifying problematic instances (IT1–IT4), all instances are organized in a sortable, filterable, and customizable table. The LineUp technique allows an interactive exploration of rankings based on multiple attributes of a tabular dataset (Gratzl et al., 2013). Each instance is a row in the LineUp table. By default, only instances with at least one incorrect classification are shown in InstanceFlow's Tabular View.

The columns include the input data (i.e., images in the case of image classification), the ground-truth class label, and several "difficulty" measures defined in Section 4.3.2. One column shows the class predictions over time as a colored heatmap (see sixth column in Figure 4.1), using a categorical color scheme to encode the sequence of predicted classes. An additional column shows a histogram of correct (■), incorrect (■), and other (■) predictions (see fifth column in Figure 4.1). Here, "incorrect" and "other" refer to predictions of the wrong class within and outside the selected subset of classes, respectively. The encodings in both of these columns can be switched between time-dependent heatmaps and summarizing histograms.

The LineUp technique includes a number of interactive features for exploring the instance predictions: (1) *Ranking:* instances can be sorted by each of the attributes in the columns or by user-defined combinations of attributes; (2) *Filtering:* Users can further filter the instances, either by the value of an individual column or by combining filters on multiple columns.

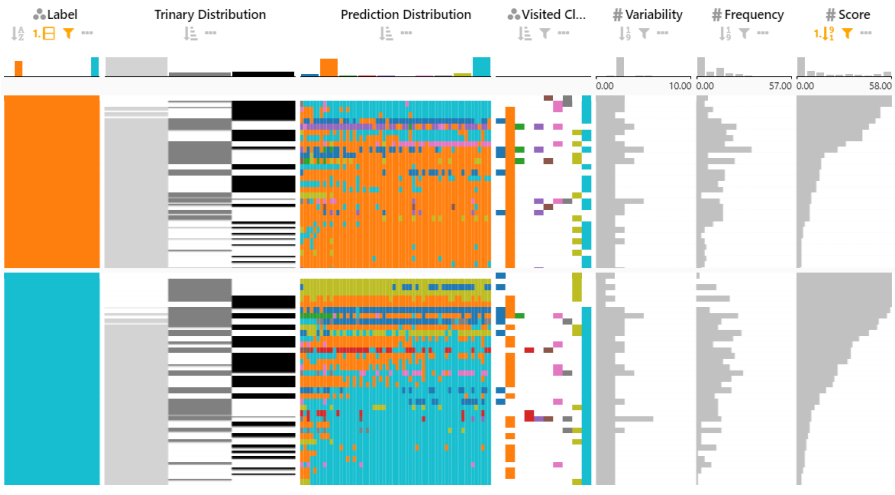


Figure 4.3: Condensed mode of all instances, revealing patterns of successful learning in the classification process.

Advanced filtering with respect to temporally changing attributes is possible via regular expressions. (3) *Grouping and Aggregating:* Users can gain an overview of the table by switching to a display mode in which the height of each row is reduced to a minimal height of a single pixel (see Figure 4.3). As a result, the previously individual heatmaps and bar charts now form a dense, two-dimensional table that reveals overall patterns, similar to visualizations produced by the Table Lens technique (Rao & Card, 1994). Users can further condense the display by using the group aggregation feature of LineUp, which shows only summary visualizations for the selected classes. Depending on the attribute type, classes are summarized using histograms or box plots (see Figure 4.4). The summary histograms for the prediction distributions encode the same information as a confusion matrix.

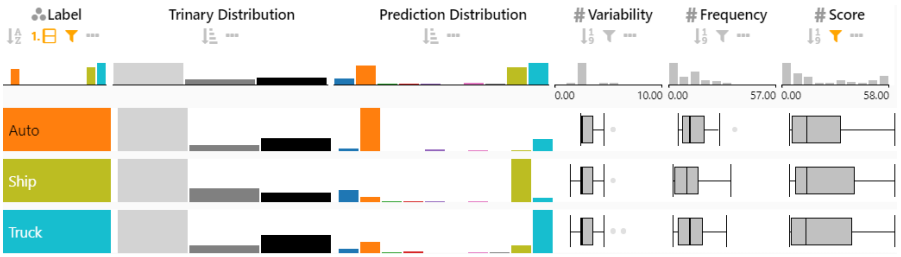


Figure 4.4: Summary mode of the Tabular View. The overview is similar to a confusion matrix, with correct classifications along the diagonal.

Difficulty Measures The ranking and filtering operations can help users to identify interesting instances when used in conjunction with measures that describe how difficult an instance is to classify. In this section, we describe three such measures.

Let m be the total number of instances, n the number of classes, and k the number of selected epochs. Let $C(i)$ be the actual class of instance i and $P(i, j)$ the prediction for instance i in epoch j.

The *misclassification score* S of an instance is the fraction of epochs in which it was assigned to the wrong class: $S(i) = (1/k)\sum_{j=1}^{k}[P(i,j) \neq C(i)]$. A misclassification score of 0 means the model predicted the correct class in every epoch, whereas a score of 1 means that the model never predicted the correct class.

The *variability* V is the fraction of class labels that the model predicted for an instance across all epochs: $V(i) = (1/n)|\{P(i,j)\}_{j\in\{1,\ldots,k\}}|$. A variability of $1/n$ means that the model predicted the same class in every epoch, whereas $V = 1$ means that the model predicted every possible class at least once.

The *frequency* F is the fraction of epoch transitions for which the model's prediction jumps between classes: $F(i) = 1/(k-1)\sum_{j=1}^{k-1}[P(i,j) \neq P(i,j+1)]$. A frequency of 0 means that an instance always stayed in the same class, whereas a frequency of 1 means that the prediction changed between any two epochs.

4.3.3 *Relationship between Views and Tasks*

The different levels of detail in the Flow View and the Tabular View with its different numerical measures have complementary strengths. Table 4.2

assigns the proposed user tasks from Table 4.1 to the various visualizations/measures, depending on whether the tasks are well supported (✔), partially supported (✔), or not supported. Instance-focused tasks (IT1–IT4) are enabled primarily by the Flow View at full detail, whereas the epoch-focused tasks (ET1–ET3) are better supported by the more aggregated visualizations. The Tabular View supports a wide range of tasks.

Table 4.2: Comparison of InstanceFlow visualization components & difficulty measures with respect to the user tasks introduced in Section 4.1.

Visualization / Metric	IT1	IT2	IT3	IT4	ET1	ET2	ET3
Flow View (basic)					✔	✔	✔
Distribution Bar Charts					✔	✔	✔
Instance Glyphs & Traces	✔	✔	✔	✔		✔	✔
Tabular View	✔	✔	✔	✔	✔	✔	✔
Misclassification Score	✔		✔	✔			
Variability	✔		✔	✔			
Frequency	✔		✔	✔			

For the instance-level analysis, the Flow View focuses primarily on the free exploration of a classifier's behavior, or on tracing individual instances once they have been located. Localizing interesting instances is enabled by the Tabular View with its ranking and filtering operations based on the difficulty measures. For epoch-level analysis, the aggregated Sankey visualization provides a good overview of the class distributions and overall flows.

4.3.4 *Implementation*

InstanceFlow is a client-side web application built using the React framework. The code for InstanceFlow is available on GitHub (`https://github.com/jku-vds-lab/InstanceFlow`). A deployed prototype of InstanceFlow with example datasets and the ability to upload new datasets is available at `https://instanceflow.pueh.xyz/`.

4.4 USAGE SCENARIO: CIFAR-10 IMAGE CLASSIFICATION

For this usage scenario, let us assume that a model developer has built a simple CNN to classify images from the CIFAR-10 dataset. This training and test set consists of 60,000 color images (32 × 32 px) divided into 10 different classes, such as *Auto*, *Truck*, *Cat*, and *Dog* (Krizhevsky, 2009). The model developer is satisfied with the overall classification performance, but notices errors for *Auto* and *Truck* instances. The developer wants to better understand what causes these errors and uses InstanceFlow to analyzes the training process.

1. The user trains the neural network to classify CIFAR-10 images, and loads the classification results into InstanceFlow.

2. In the Tabular View, the user groups instances by their actual class and enables the condensed mode with the predictions shown as histograms, revealing the class confusion over the epoch range selected.

3. The user notices that most classes are predicted correctly (with the bin for correct classification being by far the highest), but for the *Auto* class the user finds that the *Truck* bar is similarly high as the bar for the actual class (and vice versa). This is an immediate indicator of a high class confusion between *Auto* and *Truck*.

4. The user is now interested in why the neural network classifies *Auto* images incorrectly as *Truck*. To focus on this confusion, the user hides all other classes. Additionally, the user filters the instances to show only those classified as *Auto* or *Truck* at least once. Finally, the user switches from the condensed mode to the normal mode to gain access to the actual underlying instances. Sorting by a high misclassification score and a low variability reveals to the user that the most problematic instances are mainly *Auto* images classified as *Truck*.

5. Now the user notices a common pattern: the topmost images all show old and bulky cars (see Figure 4.5).

6. The user proceeds by investigating the flow of these images (see Figure 4.6). It becomes clear that all of them were classified correctly in early epochs, but then suddenly changed to *Truck* one after another.

7. The user checks the traces of random modern-looking cars and finds that in stark contrast to the previous instances, many of them are temporally stable and correctly classified *after an initial misclassification.* This leads the user to hypothesize that the network, over time, learns features that tend to prioritize modern cars over bulky, antique cars.

8. The user can use these new insights in the subsequent model development or refinement process, for example, by increasing the number of problematic instances in an attempt to improve the accuracy and temporal stability for *Auto* images.

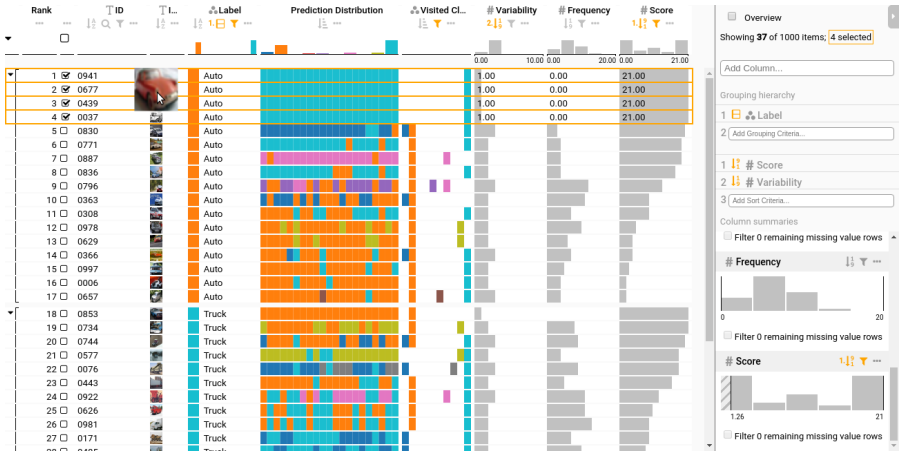


Figure 4.5: InstanceFlow showing *Auto* and *Truck* instances of CIFAR-10 sorted by a high misclassification score and a low variability, and grouped by the ground-truth label.

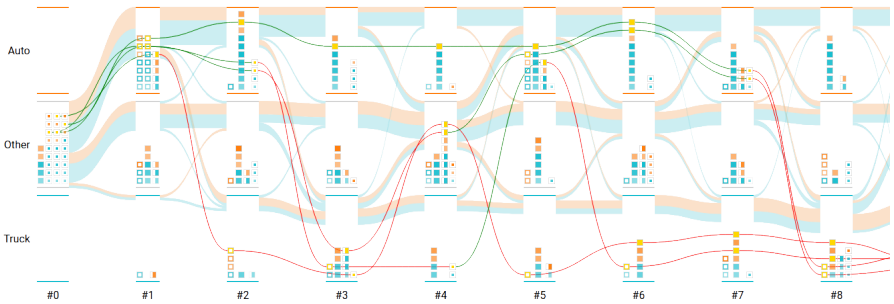


Figure 4.6: Instance Traces for several selected *Auto* images of bulky, antique cars. These images are correctly classified as *Auto* at the beginning, but tend to be classified consistently as *Truck* over time.

4.5 LIMITATIONS & FUTURE WORK

SCALABILITY The basic Flow View (without Instance Glyphs and Traces) and the Tabular View of InstanceFlow scale well to large datasets. However, for more than ~ 100 *selected* instances and at full detail, the InstanceFlow visualization can become cluttered. Additionally, with each selected class, the number of possible paths in the Sankey visualization increases. Thus, class aggregation or automatic class and/or instance selection mechanisms would be necessary for exploring datasets with many ($\gtrsim$ 15) classes.

COMPARISON OF DATASETS A comparison of multiple classification models can be helpful for evaluating the effectiveness of modifications applied during model development. We introduced a combined temporal-comparative approach for class-level analysis with ConfusionFlow (Hinterreiter, Ruch, et al., 2022). However, there is no straightforward, effective way to extend InstanceFlow to enable similar comparisons.

4.6 CONCLUSION

We have introduced InstanceFlow, a visualization of the evolution of instance classifications in machine learning. The Flow View visualizes the temporal progression of predicted class distributions. Detailed visualizations allow users to trace the predictions for individual instances over time. Interesting instances can be located effectively in the Tabular View, which allows ranking and filtering by numerical difficulty measures. With its various aggregation levels, InstanceFlow integrates class-level and instance-level performance evaluation and enables full temporal analysis of the training process.

PROJECTION PATH EXPLORER

5

The act of solving problems usually involves sequences of decisions. In most contexts, decisions can be viewed as transitions between states in a representation space. For example, solving logic puzzles such as the Rubik's cube requires decisions about how to transform an object from a random initial configuration to a solved final state. Strategy games can also be viewed as progressions through game states; each decision transforms the board state and leads closer to a player's victory or defeat. Even optimization processes, such as the training of neural networks, involve decisions about transitioning from one intermediate state to another in the hope that the process ultimately converges to a final solution.

In many application domains, the decisions determining a transition from one state to another are made consciously by human agents, often after a complex reasoning process. In other cases, the transitions between states can be based on decisions made automatically by algorithms. Examining patterns in the paths towards a problem solution can lead to a better understanding of approaches used by humans and machines to solve complex tasks.

For almost all real-world problems, the possible representation spaces are not only vast, but also high-dimensional. Many attributes are needed to accurately describe each intermediate state along the pathway to a solution. The high dimensionality of the state spaces makes assessing patterns in solution paths challenging. Previously, dimensionality reduction techniques have been used in conjunction with trajectory visualizations to explore high-dimensional sequences of states (Bach et al., 2016).

Research has so far focused mostly on discovering patterns in single solution paths. However, the exact path towards a solution may depend strongly on the initial state for that particular instance of the problem. For example, in the case of optimization algorithms, paths of convergence might depend on the chosen initialization. In other domains, for instance in strategy games such as chess, the initialization is always the same. Here, early decisions might force the solution path along one particular of many different tracks.

In both cases, it is not enough to view single solution paths without the context of many other paths. In fact, many interesting patterns emerge only from *multiple* paths. In this chapter, we present an extension of established trajectory techniques for visualizing high-dimensional paths through complex representation spaces. We identify patterns emerging from sets of projected solution trajectories. Using interactive visualization, we enable users to find such patterns and explore their relationships with the underlying real-world processes.

We present a general interactive prototype for the visualization of multiple high-dimensional solution trajectories and show how minimal adaptations can make this prototype useful in a diverse range of application domains.

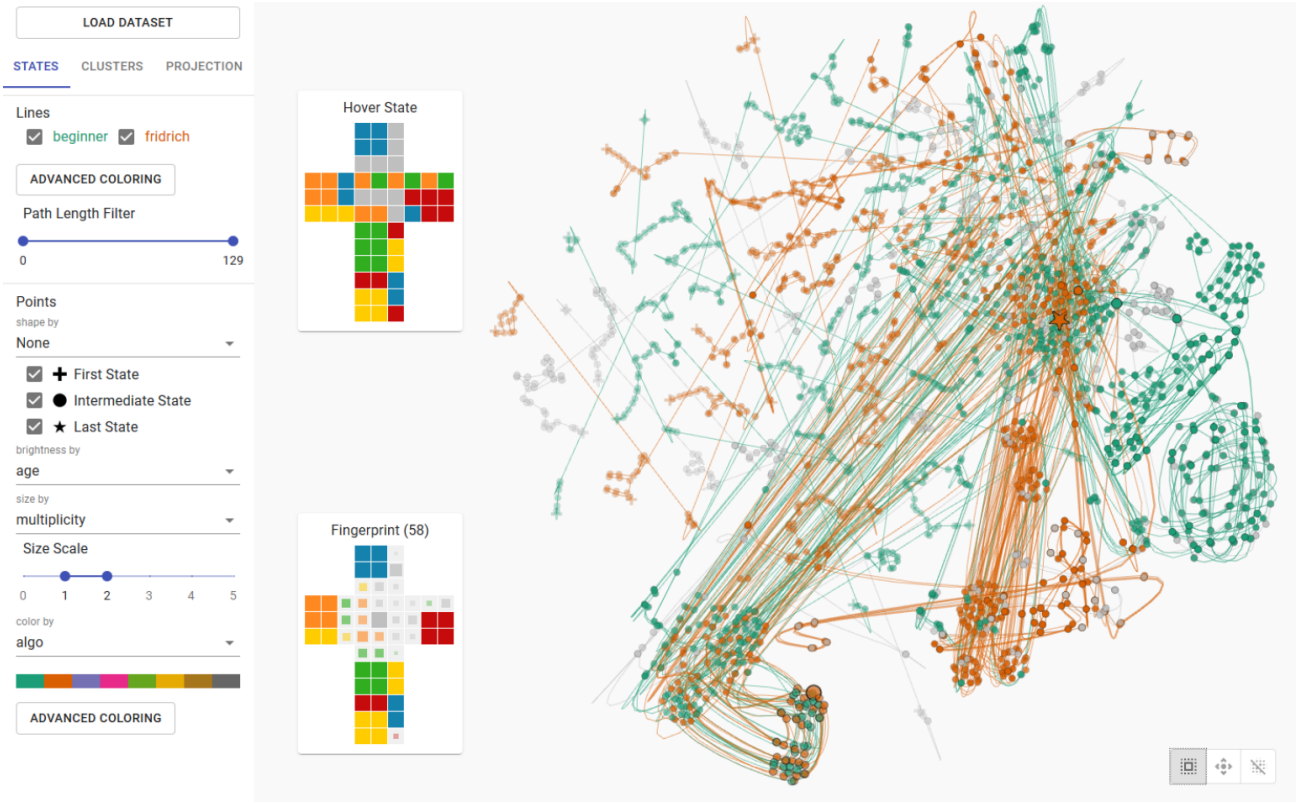


Figure 5.1: The Projection Path Explorer visualization prototype allows exploration of patterns in decision-making paths. Multiple series of high-dimensional states are visualized as trajectories through a joint embedding space.

Specifically, we present examples in which we compare Rubik's cube solution algorithms, find patterns in chess games, and assess meaningful representations of neural network training processes. These examples show how extending the self-similarity concept of single projected solution trajectories to include similarities between multiple trajectories can provide more insights into problem-solving processes.

This chapter is structured as follows. Section 5.1 gives an overview of publications related to our work. In Section 5.2 we introduce our technique and discuss how it extends existing work. In Section 5.2.3 we characterize the possible patterns emerging from our visualization technique. Section 5.3 is a description of our prototype implementation of Projection Path Explorer, which we used to analyze data from several application domains. In Section 5.4 we show the results of these analyses. The general insights gained from these applications and ideas for future work are discussed in Section 5.5. Section 5.6 concludes the chapter.

5.1 RELATED WORK ON PROJECTED SEQUENTIAL DATA

We study patterns emerging from multiple dimensionality-reduced trajectories through a problem state space. For a general discussion of the importance of trajectories for time series visualization, see Section 2.1. The most widely used dimensionality reduction techniques are reviewed in Section 2.2. Here, we discuss previous attempts at using trajectories for visualizing dimensionality-reduced data.

5.1.1 *Combining Dimensionality Reduction and Trajectories*

Without any additional encoding, trajectories can be readily used only to visualize time series with two attributes. For time series with more attributes, a combination of dimensionality reduction with trajectory visualization can be a powerful approach.

Schreck et al. (2007) used self-organizing maps (SOMs) to display projected trajectories of high-dimensional, time-resolved financial data. For document visualization, Mao et al. (2007) derived representations of n-gram data akin to phase-space trajectories used in theoretical physics. Ward and Guo (2011) generalized the notion of n-grams to non-textual data, yielding a shape-space representation of snippets of time series data. In TimeSeriesPath, data points of multivariate time series are projected using PCA, and shown as trajectories in 2D space (Bernard et al., 2012).

All these similar approaches are summarized in the Time Curve idiom by Bach et al. (2016). Time Curves are "based on the metaphor of folding a timeline visualization into itself so as to bring similar time points close to each other" (Bach et al., 2016, p. 559). The versatility of Time Curves is reflected in the diverse range of application domains: high-dimensional dynamic networks (Boz, 2019; van den Elzen et al., 2016), neural network training (Rauber et al., 2017), user-interaction data (Brown et al., 2018), trend and outlier detection (Cakmak et al., 2018), sport-game visualization (Zhu & Chen, 2016), and representation data (J. He & Chen, 2017).

5.1.2 *Context of This Work*

The usefulness of the Time Curves idiom is based on a number of characteristics of the projected trajectories, such as point density and irregularity, as well as patterns such as transitions, cycles, and oscillations (Bach et al., 2016). However, Bach et al. focused on the interpretation of single trajectories and only hinted at the power of constructing and interpreting multiple trajectories together. We build upon their classification, but focus on additional patterns emerging from *multiple* trajectories.

Other authors have demonstrated the potential of constructing multiple trajectories in the same embedding space in particular contexts. Zhu and Chen (2016) detected outliers among multiple basketball games visualized as trajectories. Brown et al. (2018) discovered differences in velocities along multiple trajectories from interaction data, and traced back these differences to varying user speeds. However, little work has focused on identifying general patterns emerging from multiple embedded trajectories throughout different application domains.

In this chapter we apply our approach to data from diverse application domains, including games and puzzles, neural network training, and user interaction data. Depending on the application context, the number of trajectories we plot conjointly in the same embedding space varies between a few and several hundred. We argue that general statements about the visualized decision-making processes can be made based on similarities between multiple trajectories.

5.2 TECHNIQUE

Prior to visualizing paths of problem solutions with trajectories through an embedding space, a number of important decisions have to be made. The visualization depends strongly on how the real-world states are represented as vectors. The distance metric for these vectors needs to be chosen carefully, as must the dimensionality reduction technique. Finally, a suitable visual encoding for the low-dimensional states and trajectories needs to be chosen. We discuss all these aspects in the following section. We also present a discussion of possible patterns emerging from multiple trajectories in the same embedding space.

5.2.1 *State-space Representation*

As previously mentioned, our visualization approach is similar to the Time Curve technique introduced by Bach et al. (2016). Time Curves are trajectories drawn through projections of high-dimensional data points. They are constructed by defining a distance metric in the high-dimensional domain and embedding the data points in two dimensions such that certain characteristics of the high-dimensional distance metric are preserved. The projected points are connected depending on the timestamps associated with each high-dimensional data point. Table 5.1 lists the key terms and definitions in the context of dimensionality-reduced trajectories.

The timestamps of the projected points are not relevant to their placement in the embedding space. Only the relative temporal ordering is used for connecting the embedded points. In most scenarios, projected trajectories are thus visualizations of sequences (or ordered sets) of states. This trajectory technique can therefore be applied to any process that involves an object undergoing several sequential states, and state indices can entirely replace actual timestamps. Bach et al. (2016) use the term "data snapshot" to refer to the individual high-dimensional data items that—together with the timestamps—make up the multivariate time series. They call the pairs of timestamps and data snapshots *time points*. In contrast, we refer to data snapshots as *states*, and we call the snapshot domain *state space*. This stresses our focus on sequential processes in which the actual time values are not necessarily of interest or even unavailable (e.g., as in the guiding example below).

An essential requirement for constructing a meaningful visualization of trajectories through an embedding space is the way in which the real-world object (that goes through the sequential steps) is "digitized". In our opinion, this is the first, and possibly most important decision when combining projection with trajectories, but it was not discussed in detail by Bach et al. (2016). We refer to this digitization as *state-space representation* and define it as a mapping from the "real world" $\mathscr{R}$ to the state space $\mathbb{S}$. Importantly, the state-space representation directly influences the meaning of any potential distance metrics.

A state-space representation incorporates one of possibly many selections of an object's attributes. All data that is not included is potentially interesting

Table 5.1: Key terms and definitions for the construction of Time Curves as given by Bach et al. (2016), and our own notation for state-space representations and dimensionality reduction.

Notation	Explanation
$t_i \in \mathbb{R}$	Timestamp
$\mathbb{S}$	Snapshot domain (state space)
$s_i \in \mathbb{S}$	Data snapshot (state)
$p_i = (t_i, s_i)$	Time point
$P = \{p_1, \dots, p_n\}$	Temporal dataset
$d\colon \mathbb{S}^2 \to \mathbb{R}^+$	Distance metric
$D = [d_{ij}] = [d(p_i, p_j)]$	Distance matrix
$P_D = \{P, D\}$	Temporal similarity dataset
$r\colon \mathscr{R} \to \mathbb{S}$	State-space representation
$f\colon \mathbb{S} \to \mathbb{R}^2$ or $f\colon \mathbb{S}^n \to (\mathbb{R}^2)^n$	Embedding*

* Domain and codomain of the embedding function depend on the dimensionality reduction technique chosen.

metadata, which is not used directly to calculate the trajectories' support points, but can be encoded additionally (e.g., by coloring trajectories or points categorically depending on the associated metadata). Thus, the actual value of the timestamp t_i of a time point (t_i, s_i) can also be treated as metadata.

As a guiding example throughout this section, we apply our visualization approach to data from two well-known sorting algorithms: bubble sort and quicksort. We first create all possible permutations of the list $(1, \dots, n)$ for a given length n. We apply each of the two algorithms to each permutation and record all intermediate steps. For visualizing the progression of the algorithms, we first need to convert the list objects from $\mathscr{R}$ to a suitably chosen state space $\mathbb{S}$ by means of the state-space representation r. We decide that we want to compare lists in terms of "how shuffled" they are. This similarity can be measured using an edit distance. There are now two approaches for obtaining meaningful distances:

- We can directly use the symmetric group of all permutations of lists with length n as our state space: $\mathbb{S} = \mathfrak{S}_n$. This allows us to simply use an edit distance of our choice as metric d, such as the Hamming distance or the Damerau–Levenshtein distance.

- Alternatively, we can use a representation of lists in which a distance of our choice (e.g., the squared euclidean distance) corresponds to an edit distance. In the case of our permuted lists, such a representation can be constructed with a one-hot encoding: $\mathrm{enc}((x_1, \dots, x_n)) = \mathrm{flat}((\mathrm{sp}_n(x_1), \dots, \mathrm{sp}_n(x_n)))$. Here, $\mathrm{flat}(M)$ flattens a matrix M to a vector. The vector $\mathrm{sp}_n(i) = (\delta_{1i}, \dots, \delta_{ni})$ is a vector of length n in which only the ith entry is set to 1 and all other entries are set to 0. In the formula for $\mathrm{sp}_n(i)$, δ represents the Kronecker delta. The resulting vectors have a length of n^2, and the state space is $\mathbb{S} = \{0, 1\}^{(n^2)}$. In this state space, $d(a, b) = \mathrm{Euclidean}(a, b)^2/2$ is exactly equivalent to the Hamming distance.

Since most dimensionality reduction techniques accept distance matrices as input, or can be readily adapted to accept non-numeric data, the first approach is usually preferable. However, the second approach can help to construct meaningful distances via intermediate encodings, when the "true" distance is computationally infeasible. This will become evident in the discussion of the distance metric we used to compare states of a Rubik's cube in Section 5.4.1.

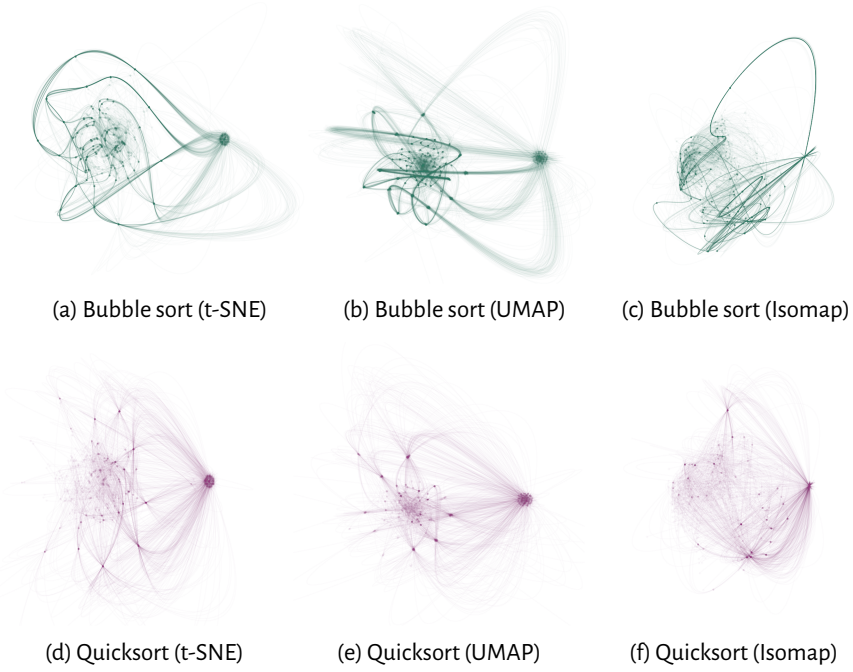
5.2.2 *Dimensionality Reduction*

In this work, we focus on visualizing *multiple* trajectories conjointly. This also affects the calculation of the embedding used for dimensionality reduction. In order to embed all trajectories in the same space, the individual states of all sequences must be projected in conjunction. This means that identical elements in the high-dimensional state space are more likely to occur multiple times. This is especially the case when the visualized real-world process is constrained to always start from the same initial state, always ends in the same final state, or has fixed intermediate states that must be traversed (see also the discussion of patterns in Section 5.2.3). In our guiding example, each application of either of the two sorting algorithms terminates in the same state, namely the sorted array $(1, \dots, n)$. This state will be present in our dataset a total of m times, where m is the number of different initial states multiplied by the number of sorting algorithms used.

In our experience, most implementations of t-SNE exhibit the following behavior: when an input set $X = \{x_1, x_2, \dots, x_n\}$, with some identical high-dimensional points $x_i = x_j$, is embedded, $y_i = y_j$ is not guaranteed to hold for the resulting embedding $f(X) = \{y_1, y_2, \dots, y_n\}$. Since the objective function of t-SNE is optimized for all points simultaneously, identical embedding coordinates for identical input values cannot be guaranteed—although the embedding coordinates will typically be very close together. Depending on the projection technique, keeping duplicates in the input set can thus offset, and thereby highlight, points that are visited more often than others. If exact identity is required, duplicates must be removed beforehand.

Figure 5.2 presents the sorting trajectories of all permutations of the list $(1, 2, 3, 4, 5, 6)$ for bubble sort and quicksort. It shows results for projecting the states by means of t-SNE, UMAP, and Isomap.

To ensure comparability between the sorting algorithms, the results of bubble sort and quicksort were projected together into a shared embedding space by using one of the three dimensionality reduction techniques. We optimized the hyperparameters of each technique to obtain visually appealing results. Regardless of the embedding routine, the visualizations for bubble sort (Figure 5.2 (a–c)) exhibit long and winding trajectory bundles. Many different sorting trajectories must go through the same few states before finally ending up in the sorted end state. For quicksort (Figure 5.2 (d–f)), the paths from the initial states to the sorted state are generally shorter and do not overlap as much as in the case of bubble sort. This reflects the fact that, on average, quicksort passes only 3.5 intermediate states from start to finish,



(a) Bubble sort (t-SNE) (b) Bubble sort (UMAP) (c) Bubble sort (Isomap)

(d) Quicksort (t-SNE) (e) Quicksort (UMAP) (f) Quicksort (Isomap)

Figure 5.2: Sorting trajectories for bubble sort (a–c) and quicksort (d–f), applied to all permutations of a list of length 6 with unique entries. Three different dimensionality reduction techniques are shown: (a, d) t-SNE with a perplexity of 100 and an exaggeration factor of 2 during the main optimization; (b, e) UMAP with 25 nearest neighbors and a minimum distance of 0.1; and (c, f) Isomap with 75 nearest neighbors. For each projection technique, data for quicksort and bubble sort was combined, to obtain a shared embedding space. Duplicates were not removed prior to the embedding. For t-SNE and UMAP, the duplicate sorted arrays form a pronounced cluster, and the shared intermediate points are more salient. All plots were rotated such that the sorted end states are located to the right.

while bubble sort takes 7.5 steps for the given lists of length 6. Furthermore, bubble sort is limited to simple swapping actions, which forces many sorting trajectories to take similar paths through the state space.

For all three embedding techniques, we did not remove duplicates prior to the embedding. For Isomap, duplicates in the input list make no difference, as duplicate high-dimensional points are mapped to to the same two-dimensional point. For t-SNE and UMAP, however, high-dimensional duplicates are embedded as clusters in two dimensions. This behavior of t-SNE and UMAP is especially relevant to the cluster of sorted end states near the right-hand side of the plots in Figure 5.2 (a), (b), (d), and (e). The clusters make the trajectory bundles thicker and more easily traceable.

One insight from the comparison in Figure 5.2 is that, with the right parameter choice, t-SNE and UMAP can yield very similar plots. This is the case when the chosen perplexity parameter of t-SNE is high enough for some global structure to be preserved. A typical argument in favor of UMAP is that t-SNE becomes inefficient for high perplexity values. However, this depends strongly on the size of the projected dataset, and on the chosen hyperparameters. In our guiding example, we projected sorting trajectories for 720 different initial states, which resulted in a total of 8,640 non-unique states (i.e., including duplicates). With the implementations we used (see Section 5.3.2), t-SNE with perplexity 100 was in fact slightly faster than UMAP with 25 nearest neighbors. However, for larger datasets, higher dimensionality, or different hyperparameters, UMAP may perform better by up to two orders of magnitude, as reported by Espadoto et al. (2019).

Note that the state space in the guiding example of sorting algorithms for lists of length 6 is small enough for all possible initial states to be tested easily, and the state space can thus be projected as a whole. For more complex problems, such as those described in Section 5.4, the state spaces are much

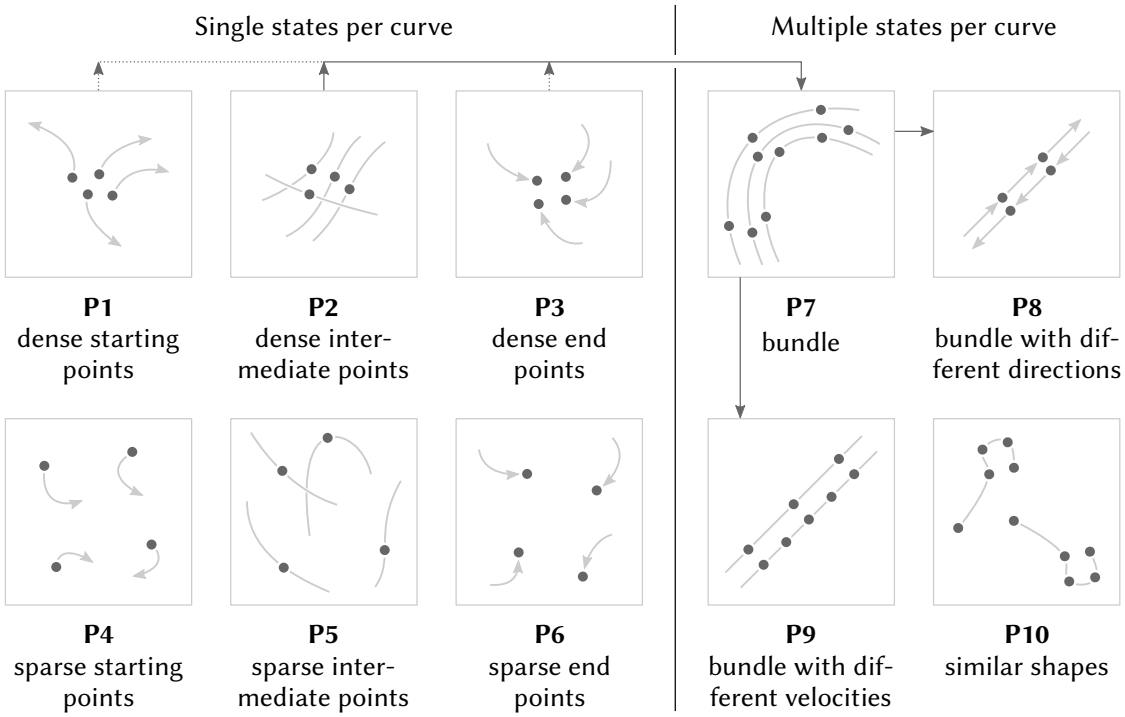


Figure 5.3: Possible patterns emerging from multiple trajectories. The different kinds of points—starting, intermediate, or end points–of several trajectories can be either sparsely distributed over large regions of the embedding space, or more densely packed. This results in six patterns: dense starting points (P1), dense intermediate points (P2), dense end points (P3), sparse starting points (P4), sparse intermediate points (P5), and sparse end points (P6). Several consecutive intermediate point clusters (possibly together with starting and/or end point clusters) lead to trajectory bundles (P7). Trajectory bundles can differ in terms of direction (P8), and/or state-space velocity (P9). The trajectories can also have similar shapes while populating entirely different regions of the embedding space (P10).

larger, rendering it impossible to project them as a whole. In these cases, only the subspace of actually visited states is projected.

5.2.3 *Patterns in Multiple Trajectories*

As can be seen in the guiding example in Figure 5.2, when multiple trajectories are constructed together, several different patterns can emerge. Figure 5.3 (a–f) shows the patterns we identified for start, intermediate, or end points, depending on whether they are sparsely distributed over large regions of the embedding space or form a dense cluster.

Obviously, tightly packed starting points (P1) can be observed when the visualized processes are constrained to start from the same or from very similar states. This would be the case, for instance, in games that always start from a fixed initial configuration. Likewise, many processes converge to the same or to similar end states, which results in a dense cluster of end points (P3). Examples of these processes are list sorting (see Figure 5.2), optimization routines that reach global optima, and solution paths of puzzles such as the Rubik's cube. As explained above, in most cases we chose not to remove duplicate entries in the state lists prior to the embedding, which—in the case of t-SNE or UMAP—leads to a tightly packed cluster rather than a single point in the embedding space even for exactly identical states.

Sparse clouds of starting points (P4) in the embedding space typically arise from randomly distributed initial states. Certain optimization algorithms, for instance, start with randomly initialized states. In high-dimensional space, the Euclidean distances between any two random states tend to be similar, with a variance limit of only 7/120 (Henry [https://math.stackexchange.

com/users/6460/henry], 2017). Most nonlinear dimensionality reduction techniques based on distances in high-dimensional space tend to spread out many similarly spaced high-dimensional points onto a disk in the embedding space. Thus, also sparse clouds of end points can emerge if the visualized processes can terminate at many different, but similarly distant, end states. The list-sorting example from Figure 5.2 is difficult to categorize in terms of patterns of starting points, since—due to the small state space—all possible states were used as starting points.

Perhaps the most interesting pattern among those described in Figure 5.3 (P1–P6) is the densely packed cluster of intermediate points (P2) of multiple trajectories through the embedding space. This pattern gives rise to *trajectory bundles* (P7)—groups of trajectories that share multiple consecutive clusters of intermediate points. Trajectory bundles also play an important role in the guiding example in Figure 5.2, as the two sorting algorithms differ mainly in whether or not distinct bundles are present. In trajectory data mining, such bundles and similar structures have been described as flocks, convoys, swarms, or gatherings (Zheng, 2015).

Figure 5.3 (P7) shows how a trajectory bundle results from consecutive clusters of intermediate states (possibly also involving clusters of start and/or end states). It also shows that the trajectories within a bundle can differ in terms of direction (P8) and/or velocity (P9). However, interpreting the velocity is not straightforward. As explained in Section 5.2.1, the timestamps (if present) are used only for joining points to form trajectories and not for the projection of states. This means that two points close to each other in the embedding space do not necessarily need to be close in a temporal sense. A direct connection between two points means that the states they represent were visited consecutively, but the viewer might still not know about the "sampling rate", that is, the actual time between two states. In our applications, we deal with indexed states that often do not have a timestamp at all.

State-space velocity can thus be interpreted as the number of state changes undergone between two non-consecutive points. Consider two trajectories, $a = \{a_1, \dots, a_n\}$ and $b = \{b_1, \dots, b_m\}$, that have two shared intermediate point clusters X and Y. Consider further that some $a_i \in X$ and $a_{i+1} \in Y$, which means that a progresses directly from X to Y. If now some $b_j \in X$ and $b_{j+2} \in Y$, but b_{j+1} lies somewhere between X and Y, then b has a *lower* state-space velocity near state j than a has at i, because b needs more intermediate steps to traverse from X to Y. Such a scenario is shown in Figure 5.3 (P9). A meaningful comparison of state-space velocities is only possible when two trajectories share at least two intermediate point clusters.

Another pattern that can emerge from plotting multiple trajectories together is shown in Figure 5.3 (P10). Two or more trajectories can have similar overall shapes, but lie in completely different regions of the embedding space. In the case of nonlinear embedding techniques, such as t-SNE and UMAP, such patterns must be interpreted with care. The actual transformation from high-dimensional to low-dimensional manifold may differ considerably for two separate state-space regions. The only true insight gained from such

patterns is that none of the states are similar across different trajectories. When the shapes of many, well-separated trajectories are very similar (as is the case in the neural network application discussed in Section 5.4.3), then a certain similarity between the paths in the high-dimensional space—and thus between behaviors of the real-world processes—can be assumed.

5.3 INTERACTIVE VISUALIZATION PROTOTYPE

To explore the possible patterns described in Section 5.2.3, we implemented an interactive visualization prototype. In this section, we describe the choices for the visual encoding and the implementation details. In order to adapt this prototype to new application domains, only minimal changes to one particular part of the encoding are necessary. While the general visualization principle will be discussed here, the domain-specific changes will be covered with each application scenario in Section 5.4.

5.3.1 *Visual Encoding and Interaction*

Projection Path Explorer is an interactive visualization prototype for exploring patterns in collections of projected decision making paths. The Projection Path Explorer user interface (see Figure 5.1) consists of two parts: a side panel with several controls and an interactive plot showing the projected trajectories along with two inset detail views.

In the Projection Path Explorer visualization, all states are represented by plot markers whose positions correspond to the results of some dimensionality reduction technique. A marker can be used to encode additional metadata: Its shape can encode categorical metadata, and its size and color can encode categorical or quantitative metadata.

Similar to Bach et al. (2016), we use Bézier interpolation for connecting projected states to form trajectories. The improved readability compared to straight lines is especially important, as Projection Path Explorer typically displays many trajectories in a shared embedding space. The color of the connecting lines of the trajectories can be used to encode additional categorical metadata for each path.

To allow full flexibility and rapid adaptation to new application domains, Projection Path Explorer can visualize data from CSV files with minimal requirements: only the coordinates of the projected states, the trajectory indices, and the relative ordering of states along trajectories must be given. Projection Path Explorer supports an unlimited number of additional metadata attributes (i.e., additional columns in the CSV files). Users can choose interactively how these metadata attributes are to be encoded in the different visual channels. Groups of paths and/or states can be filtered by categorical attributes, and, additionally, paths can be filtered by path length.

Users can hover over state markers to see detailed information in an inset ("Hover State" in Figure 5.1). Multiple states can be selected by drawing a lasso around them. A second detail view shows a *fingerprint* of the selected set of states. This fingerprint view is typically an adapted version of the detail view for single states, often based on a similarity or difference encod-

ing. These two insets—the detail view for single states and the fingerprint view for sets of states—are the only parts of the Projection Path Explorer visualization prototype that must be adapted to each application scenario. We discuss possible similarity encodings for each application scenario separately in Section 5.4, and we reflect on the design space of these encodings in Section 5.5.5.

Additionally, Projection Path Explorer allows interactive dimensionality reduction using t-SNE or UMAP. This feature is helpful if no precomputed projection has been supplied in the CSV, or if users want to explore different hyperparameters of the embedding techniques. Users can select which data attributes they want to consider when calculating the embedding. Since both t-SNE and UMAP are computationally expensive, the results are shown in a progressively updated animation, which can be terminated at any time.

5.3.2 *Implementation*

The Projection Path Explorer visualization prototype is implemented as a web application written in TypeScript. We used the three.js framework[1] for WebGL rendering of the trajectory plot. The user interface was implemented using the React library.[2] For the interactive dimensionality reduction we used JavaScript implementations of t-SNE (tsnejs[3]) and UMAP (umap-js[4]). Automatic cluster detection requires a local Python backend and uses the hdbscan[5] package. The Projection Path Explorer prototype can be accessed at `https://jku-vds-lab.at/projection-path-explorer/`.

While we used the Projection Path Explorer for exploring our datasets, the static visualizations in Section 5.4 were created in Python, based on openTSNE results (Poličar et al., 2019). For the examples in Figure 5.2 we used openTSNE, the scikit-learn (Pedregosa et al., 2011) implementation of Isomap, and the Python implementation of UMAP by McInnes et al. (2018).

[1] `https://threejs.org`

[2] `https://reactjs.org/`

[3] `https://github.com/karpathy/tsnejs`

[4] `https://github.com/PAIR-code/umap-js`

[5] `https://github.com/scikit-learn-contrib/hdbscan`

5.4 APPLICATIONS

To analyze the usefulness of the patterns described in Section 5.2.3, we applied the Projection Path Explorer visualization prototype (introduced in Section 5.3) to high-dimensional processes from four different application domains: Rubik's cube, chess games, neural network training, and user interaction data. For each application scenario, we first introduce the most important domain-specific concepts, then we describe the state-space representation (cf. Section 5.2.1), and finally we discuss how the different patterns relate to the real-world processes.

5.4.1 *Rubik's Cube Solution Algorithms*

Rubik's cube is a famous puzzle toy devised in 1974 by the Hungarian inventor and professor of architecture Ernő Rubik. The classic Rubik's cube has six faces, with each face being made up by a 3×3 grid of colored facets. These facets are the faces of smaller cubes, the so-called cubies. The cube consists of 26 such cubies: 8 corner cubies with three facets each, 12 edge cubies

with two facets each, and 6 center cubies with one facet each. The cube is considered solved when each of its faces shows only one color.

Rubik's cube is commonly known to be almost impossible to solve if no specific solution strategy or algorithm is applied. The closer a cube is to being solved and the more cubies are in the correct position, the more likely it is that any rotation made with the intention of solving another part of the cube will scramble already correctly placed cubies. Therefore, precisely considered sequences of rotations must be applied which ensure that only specific cubies are moved to their intended destinations. Many solution strategies for Rubik's cube have been developed. They differ in complexity and speed, depending on the number of special patterns and conditions that are detected and utilized in the solving process. Generally, the faster a solution algorithm is and the fewer rotations are needed, the more sub-algorithms must be learned and applied under the correct conditions. The classic beginners' method is highly inefficient, but has only a few sub-algorithms to be memorized. More advanced methods, such as Fridrich's CFOP method (Fridrich, 1997) and the Petrus method (Petrus, 1997), are harder to learn, but usually require significantly fewer moves. Generally, solution algorithms often use checkpoints: special points in the state space of the cube (e.g., having a yellow cross on the yellow side). This state space in which the solution algorithms act is high-dimensional and encompasses more than 4.3×10^{19} unique states.

State-space Representation The first step of visualizing the solution pathways is to transform the cube states into numerical representations. We based the encoding of the cube state on the data structure underlying a Python Rubik's cube API Liberacki and Brannan (2015), which we also used for calculating the solutions. Each face of the cube is represented by a 3×3 matrix with one entry for each facet. For the entries representing the facet colors, we chose a one-hot encoding (i.e., (0,0,0,0,0,1) for red, (0,0,0,0,1,0) for green, etc.). This encoding facilitates the definition of meaningful distance metrics for the dimensionality reduction. Flattening the resulting $6 \times (3 \times 3) \times 6$ tensor yielded a feature vector of length 324 for a single state.

Similarity in the original feature space is defined via a distance metric. We tried different metrics and ultimately chose the Euclidean distance. Euclidean distance in the high-dimensional state space does not take into account the number of operations (i.e., rotations of cube slices) necessary to go from one state to the next. Instead, we argue that applying the Euclidean distance to our choice of feature vectors yields a representation that is more in line with the intuitive judgment of how scrambled the cube is. In fact, for this one-hot encoding, the Euclidean distance is equal to the square root of the Hamming distance. The Euclidean distance thus corresponds to a "naive edit distance", that gives a measure of the number of cube facets with the wrong color, but not the number of moves required to fix them.

Implementation & Visualization Details The solutions were calculated from random initial states. Using a Python Rubik's cube API (Liberacki

& Brannan, 2015), the initial states were generated by performing a set of random rotations on an initially solved cube. This ensured that only physically possible cube states were used. We added a new solution algorithm and options for data export to the API. For each state, we exported (*i*) the high-dimensional encoding and (*ii*) whether that state is a checkpoint of the algorithm used. Finally, we projected the high-dimensional cube states using t-SNE. For the 200 solution trajectories we used a learning rate of 100 and a perplexity of 50. For the plots with only 2 trajectories we used a perplexity of 20.

The trajectories for different solution algorithms are visually encoded by hue. The marker shape encodes whether a state is an initial state, a final state, or a checkpoint of the algorithm. Disk markers for intermediate states between checkpoints can be displayed on demand. The markers' brightness encodes the progression through the solution trajectory, with bright colors corresponding to early states.

The single-state detail view (see Section 5.3) shows the colored facets of an unfolded cube. In the fingerprint view for multiple selected states, only those cube facets that have the same colors across all selected states are shown as full-sized, colored squares. For each remaining, non-constant facet, the color values are counted across all selected states. Each facet is given the color with the highest number of counts, that is, the most prevalent color for that facet. The non-constant facets are additionally shrunk and made transparent, with both the facet area and the alpha value being proportional to the count of the most prevalent color. This encoding ensures that constant facets across multiple states can be quickly identified, while information about the non-constant facets is preserved. An example of this similarity encoding can be seen in Figure 5.6.

A deployed version of our prototype implementation with pre-selected Rubik's cube solution data can be accessed at `https://jku-vds-lab.at/projection-path-explorer/?set=rubik`.

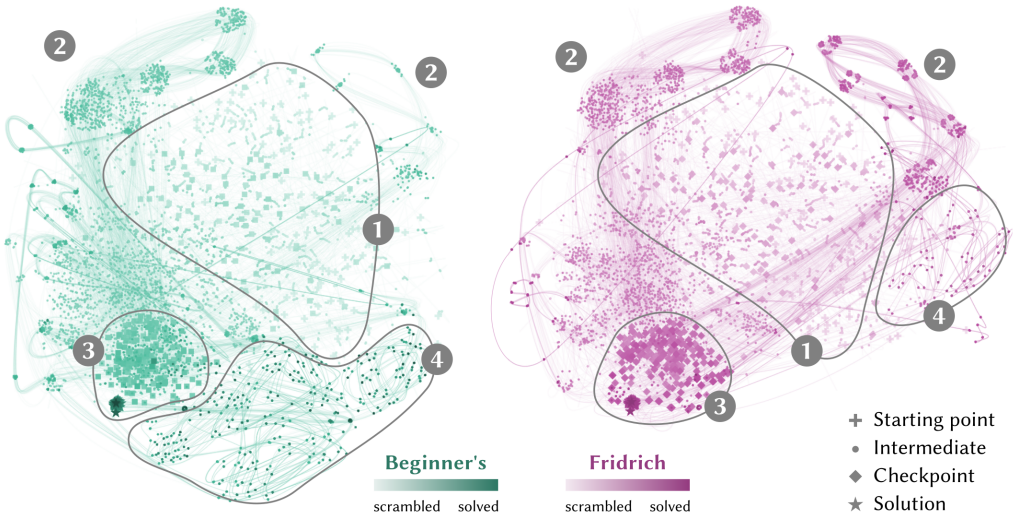


Figure 5.4: Projected solution pathways for 100 random Rubik's cubes solved with the beginner's method (left), and with Fridrich's method (right), respectively. Data for both algorithms was combined for the calculation of the t-SNE projection, but is shown in two individual visualizations for easier interpretation. The random initial states form a broad cluster (P4) near the center of the projected state space (1). Both solution algorithms take similar intermediate paths (2), and later checkpoints cluster densely (P2) near the final solution (3). Notably, Fridrich's algorithm avoids lengthy sequences of rotations to transform an almost solved cube into a solved cube (4).

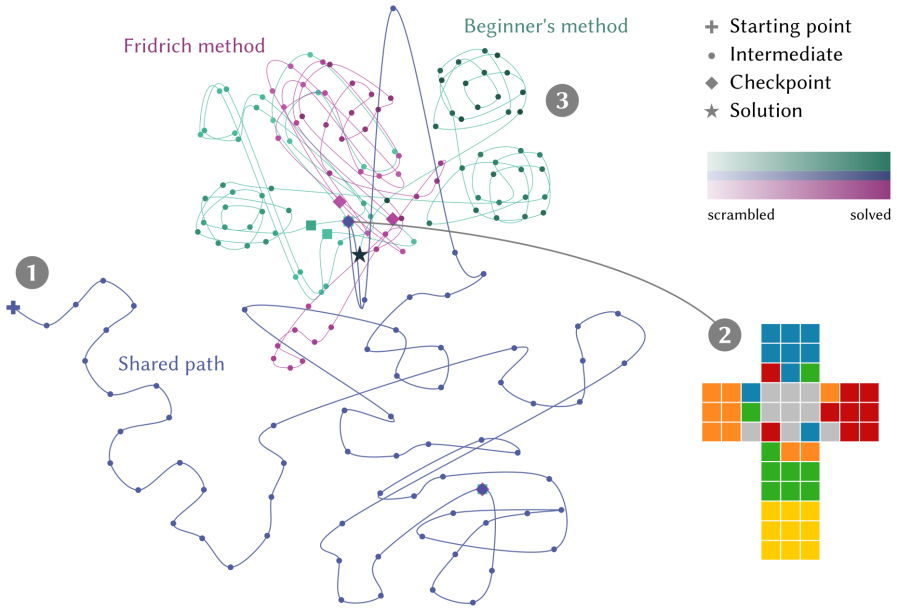


Figure 5.5: Projected solution trajectories of the same initial state (1) solved with the beginner's method and Fridrich's method, respectively. The algorithms share the same path up to the second checkpoint (2), at which two layers of the cube are already fully solved. Near the end, the beginner's method requires additional rotations, while Fridrich's method approaches the solution much faster.

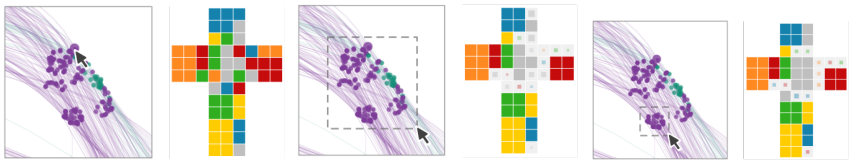
Results Figure 5.4 shows the projected solution trajectories for 100 randomly chosen initial cube states, both for the beginner's method and the more advanced Fridrich's method. Clearly, projecting the cube states with t-SNE leads to the formation of different clusters. Most prominently, later checkpoints form dense clusters close to the final solution (see (3) in Figure 5.4). Earlier checkpoints with fewer correctly positioned cube facets are much more spread out. These checkpoints share a wide, sparse region with the randomly selected initial states (1). For the checkpoints, we thus observe a correlation between the timestamp t_i (see Table 5.1) and the type of pattern: small t_i correspond to P5, while larger t_i correspond to P2. In later stages along the solution paths, intermediate states also tend to form dense clusters (P2), which leads to bundles of parallel trajectories (P7) between checkpoints (2).

Figure 5.5 shows solution trajectories for the beginner's method and Fridrich's method applied to the same initial state (1). Our choice of color-mixing and the use of opacity reveal that the solution trajectories overlap completely up to the second checkpoint (2). This almost perfect overlap—a special case of the bundle pattern (P7)—shows that even without removing duplicates, t-SNE does not always introduce significant spacing between identical points for certain hyperparameter choices (in this case perplexity 20).

As, upon user demand, the visualization provides detailed views of unfolded cubes for all intermediate states, it can be seen that two layers of the cube are already fully solved at the second checkpoint. We found this behavior to be general, that is, independent of the initial state. Afterwards, the more optimized Fridrich method uses a large number of different sub-algorithms. This avoids additional lengthy sequences of rotations close to the final solution. For the beginner's method, these rotations show up as characteristic coils (3).

When more cube solutions are projected, the differences between the strategies due to the use of sub-algorithms become even more apparent (see Figure 5.4). Our visualization shows large bundles of trajectories (P7) as a result of clusters of similar intermediate points (P2). Up to a certain point, these bundles appear similar for the two different techniques (2).

Inspection of multiple states by means of a lasso selection can help to understand the origin of state clusters and trajectory bundles. As an example, we look at one state cluster along the trajectory bundle on the upper right-hand side of Figure 5.4. The similarity encoding for the cluster is shown in Figure 5.6 (b), along with the standard detail view for an individual state in the cluster (a). The similarity encoding reveals that about 70 % of the facets have the same color across the states in the cluster. This particular cluster of intermediate points (P2) has a clearly defined sub-cluster, with four more white facets shared across all its states, as seen in Figure 5.6 (c). This more fine-grained analysis is made possible by the similarity encoding in the fingerprint view.



(a) Single state (b) Cluster (c) Sub-cluster

Figure 5.6: Analysis of clusters and sub-clusters along trajectory bundles. Selecting a single point of a state in a cluster of intermediate points reveals the corresponding cube in the standard detail view (a). Selecting the whole cluster via brushing updates the similarity detail view (b). About 70 % of the cube facets are the same across the whole cluster. Selecting only the small sub-cluster shows that four more facets are the same (white) inside this sub-cluster (c).

Further along the solution paths, the beginner's method's inefficient approach to correctly positioning the final cubies gives rise to a large cluster of many "avoidable" intermediate steps (P5, see (4) in the left part of Figure 5.4). In the case of Fridrich's method, many fewer steps are required during the final stages of solving the cube, leading to a much smaller and less populated cluster of intermediate points (see (4) in the right part of Figure 5.4).

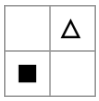
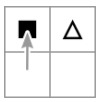
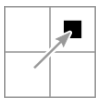
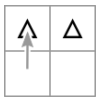
Thus, the scalability of our visualization approach to hundreds of cubes allows us to reliably detect differences between the beginner's method and Fridrich's method. These differences are most apparent in the different patterns of types P2 and P5. Furthermore, the detail views for single and multiple state selections help users understand the structure of the embedding space. Thereby, these views clarify how the different patterns in the embedding space relate to the real-world cubes.

To tighten this connection between patterns and real-world actions, we built an interactive physical Rubik's cube demonstrator. This demonstrator combines our visualization approach with a Bluetooth Rubik's cube and a Lego Mindstorms robot and is described in Appendix B.

5.4.2 *Chess Games*

The game of chess has a compact set of rules. For each piece—king, queen, rook, bishop, knight, and pawn—only a limited set of movements is allowed. Nevertheless, chess is extremely rich in tactics and strategy. Players need to continuously evaluate the positions of the pieces on the board and adapt

Table 5.2: State-space representations for board states before and after various kinds of moves on a simplified 2 × 2 chessboard with two different pieces. The pieces are encoded as ■ = (0, 1, 0) and △ = (0, 0, 1), and empty fields as (0, 0, 0). For simplicity of the representation, castling and promotion are performed across colors. The listed distances are Euclidean distances with respect to the initial state given in the first row.

Movement	Board	Tensor	Vector	Distance
None (initial)		$\begin{pmatrix} (0,0,0) & (0,0,1) \\ (0,1,0) & (0,0,0) \end{pmatrix}$	$(0,0,0,0,0,1,0,1,0,0,0,0)$	0
Simple		$\begin{pmatrix} (0,1,0) & (0,0,1) \\ (0,0,0) & (0,0,0) \end{pmatrix}$	$(0,1,0,0,0,1,0,0,0,0,0,0)$	$\sqrt{2}$
Capture		$\begin{pmatrix} (0,0,0) & (0,1,0) \\ (0,0,0) & (0,0,0) \end{pmatrix}$	$(0,0,0,0,1,0,0,0,0,0,0,0)$	$\sqrt{3}$
Castling		$\begin{pmatrix} (0,0,0) & (0,1,0) \\ (0,0,1) & (0,0,0) \end{pmatrix}$	$(0,0,0,0,1,0,0,0,1,0,0,0)$	2
Promotion		$\begin{pmatrix} (0,0,1) & (0,0,1) \\ (0,0,0) & (0,0,0) \end{pmatrix}$	$(0,0,1,0,0,1,0,0,0,0,0,0)$	$\sqrt{2}$

their future moves accordingly. To strengthen their skills, players typically study records of games, often with annotations from experts.

A game of chess can be roughly divided into the three stages of opening, middlegame, and endgame. In the opening phase, players aim to develop their pieces (i.e., move them to strategically relevant positions), take control of the center of the board, ensure safety of their king, and structure their pawns. The middlegame depends mostly on the openings chosen by the two players. It is typically the phase in which most captures occur, often as results from so-called combinations. Finally, the endgame is the phase of the game in which only few pieces remain on the board. Important goals in this stage are promotion of pawns, strategic positioning of the king, and forcing the opponent to make certain moves—a situation called zugzwang. Players win either by checkmate or resignation of their opponents. A game can also end in a draw for a number of reasons, including agreement or a stalemate.

Classic tools for computer-aided studying of chess games, such as Fritz 16 (Friedel, 2017), typically feature a simple chessboard visualization of one game state at a time. Instead of focusing on single snapshots of the game, Lu et al. (2014) visualize the evolution of entire games. They introduced the evolution graph: a decision-tree visualization that combines the actual moves performed by the players with alternative moves calculated by an AI. However, this approach remains limited to a single game.

Using our visualization approach, we explore possible similarities between *many* games, and analyze how the games proceed through the different phases. We apply our visualization approach to chess by viewing the chessboard as a state space, with each configuration of pieces(i.e., each so-called position) corresponding to one possible state. Each of the players' moves causes the game to proceed from one state to the next. The number of

reachable positions has been estimated to lie between 2×10^{43} and 1.8×10^{46} (Chinchalkar, 1996).

State-space Representation In order to represent the possible configurations of pieces numerically, we started by viewing the chessboard as an 8×8 matrix. Each field can be empty or populated by one of 12 different pieces (six different pieces each for black and white). We thus represented the state of one field by a one-hot code of length 13, which resulted in a $8 \times 8 \times 13$ tensor encoding the full board state. Flattening this tensor yielded a vector of length 832. As in the case of the Rubik's cube visualization, we chose to use the Euclidean distance. The results of this choice for the different kinds of moves are detailed in Table 5.2.

Using this encoding and the Euclidean distance has the consequence that moving a bishop diagonally by 1 or 4 fields, for instance, does not result in different state-space distances (as long as no captures occur). Both resulting states have a distance of $\sqrt{2}$ to the previous state, and the same is true for all simple movements and promotions (see first and last row of Table 5.2). The states resulting from castling, however, have distance 2 to the previous state, and captures result in a state-space distance of $\sqrt{3}$. The type of move (capture, castling, promotion, etc.) leading to each state can additionally be regarded as metadata.

Implementation & Visualization Details The chess game records used for the visualizations in this section are freely available at KingBase (Havard, 2019) in the Portable Game Notation (PGN) format. We parsed the PGN files using the chess module of the pgn2gif Python package (Deniz [`https://github.com/dn1z`], 2018). The resulting sequences of chessboard states were encoded as described above.

The trajectories for different opening moves are visually encoded by hue. Each intermediate state is represented by a marker. We visualize the initial state (standard chess setup) as crosses, and the final states of the games as stars. The detail view for a single state selection is simply the chessboard. The fingerprint for multiple selected states uses a similarity encoding comparable to that used in the Rubik's cube application. Fields with varying chess pieces across the selected states show only the most prevalent piece. The piece is made transparent, with an alpha value proportional to the counts for the most prevalent piece on that field across all selected states.

A deployed version of our prototype implementation with pre-selected chess game data can be accessed at `https://jku-vds-lab.at/projection-path-explorer/?set=chess`.

Results Figure 5.7 shows decision trajectories for 200 randomly selected chess games between players with Elo scores greater than 2,000. The games are colored depending on the first move by white, which was either the Queen's Pawn Game (d4) or the Zukertort Opening (Nf3). Together with the King's Pawn Game, these two moves are among the three most common openings.

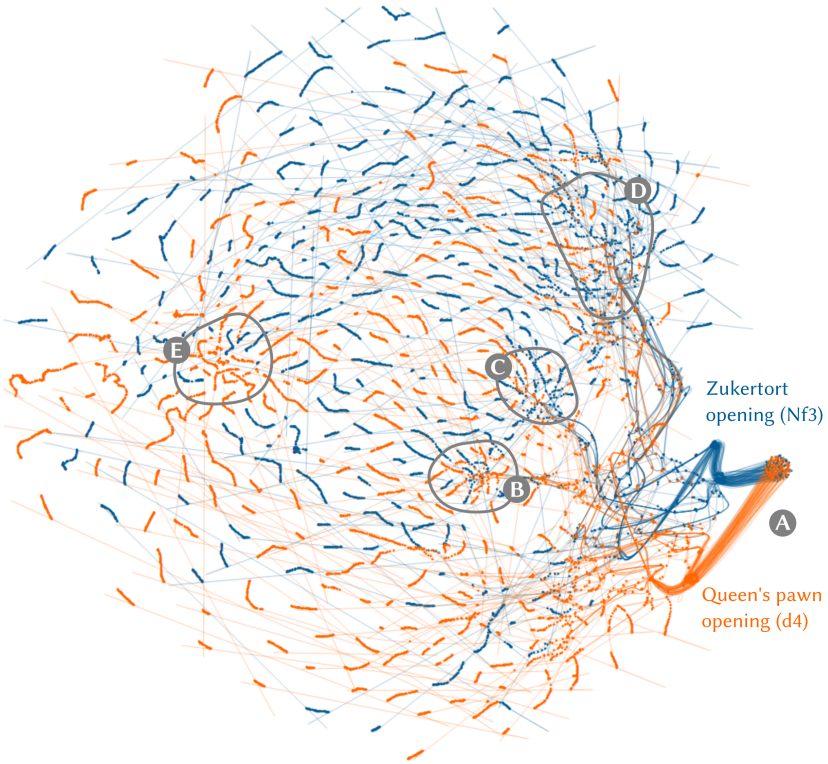


Figure 5.7: Visualization of 200 chess game trajectories for two different openings, Queen's Pawn Game (d4) and Zukertort Opening (Nf3), respectively. Several trajectory bundles (P7) emerge from the cluster of identical initial states (a). Two smaller clusters of intermediate game states (P2) are visible near the center of the plot. Both clusters consist of game states resulting from earlier castling moves. The lower cluster (b) has black pawns placed on c6 and d5. The upper cluster (c) has black pawns placed on c7 and d6, respectively. Game states resulting from a certain kind of pawn defense cluster towards the upper right side of the plot (d). Finally, endgames with only few pieces remaining form a more densely packed cluster (e), while other endgames are spread out evenly across the embedding space (P6).

Since each game starts from the same board state, a dense cluster of initial states (P1) is visible in Figure 5.7 (a). From this cluster, two trajectory bundles (P7) emerge—one for each of the two openings. Subsequently, the bundle resulting from the Queen's Pawn Game splits up into one narrow bundle going down and one wide bundle going up. These bundles reflect the first move performed by black (f5 and e6, respectively). A similar splitting can be observed for the games starting with the Zukertort opening. Even several states later, only a few trajectory bundles are visible in the lower right and right parts of Figure 5.7. For early timestamps, no sparse intermediate point patterns (P5) are visible. This makes sense, as for each ope ning there are only a few established reactions.

The later intermediate states, roughly between the opening phases and the middlegames, form three particular structures that are of type P2, but do not give rise to path bundles. Two clusters, seen in Figure 5.7 (b) and (c), are collections of game states resulting from earlier castling moves. The transparency encoding in the interactive detail view revealed that the main difference between these two clusters is the position of two black pawns. In the lower cluster (b), two of the black pawns are placed on c6 and d5. In the upper cluster (c), they are instead placed on c7 and d6. Interestingly, games for the two different openings are not represented equally in each cluster. The Queen's Pawn Game seems to be overrepresented in the cluster related to black pawn positions c6 and d5, while the Zukertort Opening is slightly overrepresented among games in the cluster related to black pawn positions c7 and d6.

Finally, at some point the trajectories tend to make large jumps to a distinct region in the state space where each game culminates in a sequence of densely threaded intermediate states. These dense sequences (P2) before the endstates are representations of endgames. Endgames in which only few pieces remain converge in a similar region of the state space, as seen in Figure 5.7 (e). The final states of all other games are sparsely distributed across large regions of the embedding space (P6).

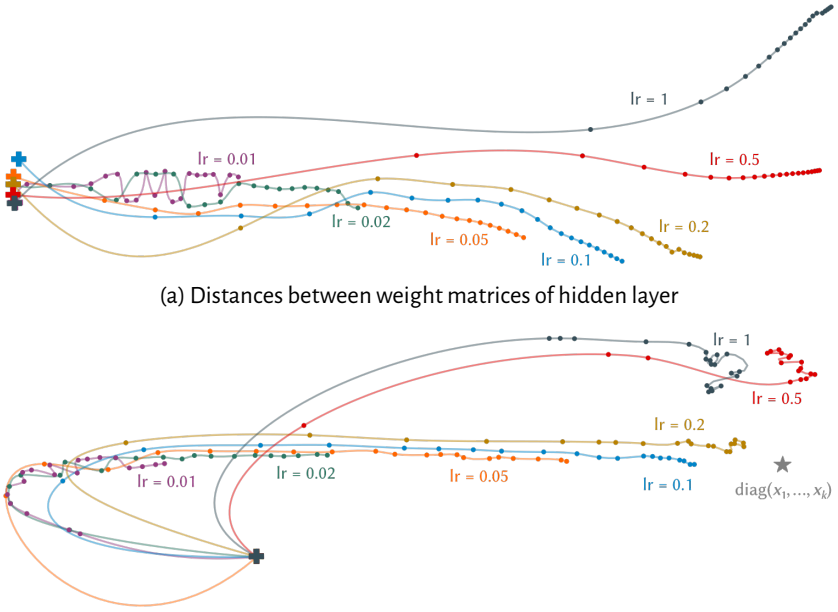
Considering the simple encoding of the chessboard we chose as our state-space representation, it is surprising to see actual gameplay patterns emerge from the embedded trajectories. These patterns are only possible when many trajectories are constructed together. Furthermore, the application of decision trajectories to chess games showed that interactive exploration is vital for relating visual patterns to the real-world meaning. As part of future work, we plan to cooperate with a professional chess player to explore how exactly the patterns correlate with strategic decisions made by players.

5.4.3 *Neural Network Training*

Deep neural networks are a class of powerful, nonlinear models that can learn complex representations of data. They have greatly improved the state of the art in a diverse range of application domains, including computer vision, speech recognition, drug discovery, and genomics (LeCun et al., 2015). However, the learning behavior of deep neural networks is difficult to interpret. It is usually not clear, how a certain choice of hyperparameters or a certain train/test split affect the final performance of the model. With the growing impact of deep learning on real-world decision-making, these difficulties have led to an increased demand for explainable or interpretable models. As outlined earlier, one possible way of analyzing, understanding, and communicating the processes during deep learning is visualization (Hohman et al., 2018).

The behavior of deep neural networks depends on many different aspects: training and test data, the network architecture, choice of optimization technique and hyperparameters, the actual learned weights, and the resulting activations for each input. Most of these aspects can be visualized, and almost all of them—even architecture and training data—may change over the course of the training. As such, the training of neural networks can be viewed as sequential steps through a high-dimensional state space. Furthermore, developing well-functioning deep models is a highly incremental and iterative process, that requires model builders as well as model users to compare the behavior for many different experimental configurations. This combination of "temporality" and the need for comparative analysis makes training of neural networks a perfect candidate for a visualization using multiple embedded trajectories.

While dimensionality reduction has been applied to visualize the representations learned by different network architecture (Aubry & Russell, 2015; Donahue et al., 2014; Hamel & Eck, 2010; Mnih et al., 2015; Mohamed et al., 2012), in most cases only the projections for one time step are shown—usually



(a) Distances between weight matrices of hidden layer

(b) Distances between confusion matrices

Figure 5.8: Learning processes of neural networks trained with different learning rates using a dimensionality reduced (PCA) version of the MNIST images as training data. The initialization was the same for all networks. Trajectories projected by t-SNE from two different state-space representations are shown: (a) based on the weight matrices of the hidden layer; and (b) on the confusion matrices for the test dataset. Both representations resulted in groups of trajectories with similar patterns (P1, P7, P9). The state space of the confusion matrix representation can be made interpretable by augmenting it with the confusion matrix of perfect classification, denoted by $\mathrm{diag}(x_1, \ldots, x_k)$.

for the final network state after termination of the training. Only few works combine dimensionality reduction techniques with the temporal progression of the training process. Rauber et al. (2017) visualized the inter-epoch evolution of neuron activations as trajectories in an embedding space. Even more closely related to our work is a visualization used by Erhan et al. (2010) in their study about pretraining neural networks. They show learning trajectories through function space, which is one particular state-space representation. We will show results for two other choices of representations, one of which also motivates why Erhan et al. chose their function space representation.

State-space Representation We chose two different state-space representations: a representation of the weight space and a representation of the confusion matrix.

The *weight representation* corresponds directly to the weights learned by the network. For a given layer, we flatten the associated weight matrix to a single vector. After each training epoch, the weights are updated and a new weight vector can be obtained. The length of the vectors, and thus the dimensionality of the state space for this representation, depends on the network architecture. It is equal to the number of units in that layer, times the number of units in the previous layer. We use the Euclidean distance as a metric for comparing weight vectors.

The *confusion matrix representation* is more closely connected to network performance. At each epoch, we let the network classify all test instances, and construct the resulting confusion matrix. Each cell (i, j) in the confusion matrix lists the number of instances with ground truth class label i and predicted class label j. This state-space representation is only suitable for supervised classification problems. The length of the flat confusion vector is k^2, where k is the number of different classes. We experimented with using

the Euclidean distance as well as the cosine distance for comparing confusion states, but found that the resulting plots look very similar for both metrics. Here, we only show trajectories constructed using the Euclidean distance.

Implementation & Visualization Details We used PyTorch (https://pytorch.org/) to train a simple neural network to classify MNIST (LeCun, 2005) images. We decided to use a preliminary dimensionality reduction from the number of weights down to 50 components by means of PCA. According to van der Maaten and Hinton (2008), this preliminary dimensionality reduction can reduce noise and speed up the computation of the subsequent embedding. We found that this preprocessing step reduces the computation time significantly, while affecting the plots only negligibly. We also chose 50 as the number of units in the hidden layer, resulting in 2,500 weights between the input and hidden layers. We used ReLU as activation function, stochastic gradient descent for optimization, and cross entropy loss. We tried different learning rates between 0.01 and 1. In most of our experiments, we trained the networks for 20 epochs, and used the standard MNIST train/test split.

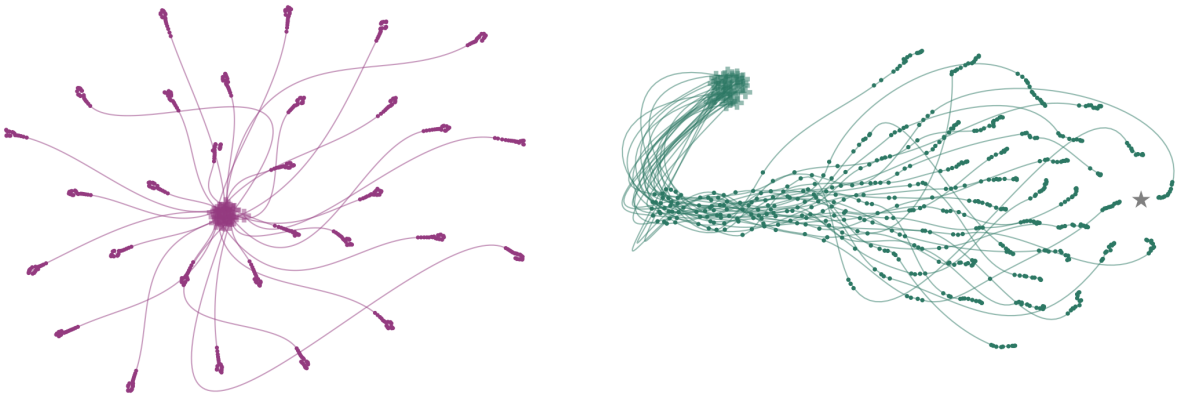
For our experiments with different learning rates, we colored the trajectories categorically by learning rate. The detail view for single state selection is a visualization of the confusion matrix of the network at this state. The confusion matrices were evaluated for the MNIST test sets. Since all networks learned relatively fast, a naive visualization of the confusion matrix would look similar to a diagonal matrix already after the first epoch. To instead direct the users' attention to the errors (i.e., the off-diagonal values) we left the diagonal blank and adjusted the color scale accordingly. This approach is similar to what we used in ConfusionFlow (Hinterreiter, Ruch, et al., 2022).

A deployed version of our prototype implementation with pre-selected learning trajectories can be accessed at https://jku-vds-lab.at/projection-path-explorer/?set=neural.

Results For our first experiments, we wanted to see whether the notion of "learning speed" is conserved in the embedded trajectories for neural network training. To this end, we used several different learning rates, and initialized all networks with the same (randomly chosen) initial weights. The corresponding trajectories are shown in Figure 5.8, where the top plot (a) shows trajectories constructed from the weight space representation, while the bottom plot (b) shows trajectories constructed from the confusion matrix representation. Different learning rates are encoded by hue, and the values are listed in colored insets.

In both plots, the equal initial states are represented by a cluster of dense states (P1). It can be seen that the dimensionality reduction technique—in this case t-SNE with perplexity 40—does not always behave in the same way with regards to clustering equal states, even with equal hyperparameters. In the weight space representation, the initial states are slightly spread apart, while in the confusion embedding space, they are extremely tightly packed. The overall trend, however, is the same for both representations.

Lower learning rates lead to smaller changes in the networks' weights, as



(a) Distances between weight matrices of hidden layer

(b) Distances between confusion matrices

Figure 5.9: Learning processes of neural networks trained with equal learning rates but different random initialization, using a dimensionality-reduced version (PCA) of the MNIST images as training data. Trajectories embedded using t-SNE, based on two different state-space representations, are shown: (a) based on the weight matrices of the hidden layer; and (b) on the confusion matrices for the test dataset. For the confusion matrix representation, the state space is augmented with the confusion matrix of perfect classification (★).

well as in their classification behavior. This slow progression through the state space is reflected in the trajectories by a high density of intermediate states, i.e., the learning rate is directly reflected in the different state-space velocities in the path bundle (P9). This is also the case near the end of the training, when the optimization starts to converge.

Figure 5.8 (a) shows that, in the weight space, the trajectories for learning rates between 0.01, 0.2 all move in the same direction. The trajectories for 0.5 and 1 move into a different region of the embedded state space. However, from the weight space representation, no insight about performance can be gained. Even from the confusion embedding, without any additional tricks, only insight about similarity of performances can be gained. Absolute performance, i.e., overall accuracy, cannot readily be determined from the embedding. To fix this shortcoming, we added additional structure to the confusion embedding space, by augmenting it with the projected confusion matrix for perfect classification. This single state was added to all other confusion matrices before performing the dimensionality reduction. It is shown as a gray star in Figure 5.8 (b), and denoted by $\mathrm{diag}(x_1, \ldots, x_k)$, since it is a diagonal matrix with the class distribution along its main diagonal. In this augmented state space it is possible to interpret intermediate state in terms of network performance, by assessing the distance to the added diagonal matrix. Calculating the classification accuracy for all learning rates after 20 epochs gives slightly lower values for 0.5 and 1. A learning rate of 0.2 yielded the best classifier in terms of accuracy, as would be expected from the embedded trajectories.

From Figure 5.8, one might draw the conclusion that the two representations—based on weights or confusion—carry mostly the same information. However, this is generally not the case! Figure 5.9 shows 30 networks trained on MNIST data, all with the same learning rate (chosen as 0.2 based on the insight from the previous example) and the same architecture. The only difference between the 30 networks is the random initialization of the weights. Starting from a dense cluster of initial states (P1) in the weight space embedding, all networks quickly move towards their own region, where they converge in a chain of densely packed intermediate states (P2). Near the end of this chain, almost all trajectories exhibit a similar, bulge-like shape (P10). This pattern

corresponds to the final convergence, when the differences between states become so small that their embedded representations start to form a ball rather than a chain.

The reason for the appearance of similarly shaped trajectories (P10) in different parts of the embedding space—rather than parallel path bundles (P7)—in Figure 5.9 (a) is a phenomenon known as weight space symmetries. Many different weight configurations can result in the exact same mapping function from inputs to outputs.

To address this issue, Erhan et al. (2010) chose their function space representation.

We chose a confusion matrix representation, which—as detailed above—in addition to revealing differences in the mapping functions also can be interpreted in terms of classification accuracy. In this embedding, shown in Figure 5.9 (b), it becomes clear that the networks are indeed much more similar with respect to their classification and learning behavior as one would expect from the weight space embedding. Especially in the early learning phases, distinct path bundles form (P7). However, regardless of whether the function or the weight space representation is chosen, care must be taken in interpreting the distances in the trajectory visualizations. In Figure 5.9 (b), absolute distances between final confusion matrices for differently initialized weights are artificially "blown up" by the projection technique trying to conserve relative distances within the chains of very similar matrices near the end of each trajectory. Nevertheless, this example demonstrates the power of using different state-space representations for showing different aspects of high-dimensional processes with embedded trajectories.

5.4.4 *User Interaction Data*

As outlined in Section 1.1, visual analytics tools promote users from mere observers to active agents in the data analysis process. The history of individual steps that users take in this iterative process is called *analytic provenance* (Ragan et al., 2016). Researchers have argued that recording and analyzing interaction provenance can yield insights about the users' sensemaking processes (Brown et al., 2014; Dou et al., 2009; Nguyen et al., 2016; Pohl et al., 2012; Setlur et al., 2016). This section summarizes our work on *Provectories* (Walchshofer et al., 2021), a meta-analysis approach for interaction provenance that makes use of the Projection Path Explorer.

State-space Representation We applied Provectories to two datasets from real user interactions. In one user study, we asked participants to interact with a version of the *Gapminder* visualization (Rosling & Zhang, 2011) that was extended with provenance tracking (Stitz et al., 2019). Each state of the Gapminder visualization can be fully captured with a small number of variables of different types. The variables include the selected year (numeric), axes attributes (categorical), and countries (set attribute). We vectorized each non-numeric attribute independently, using one-hot encodings for the categorical attributes and a simple list of binary values

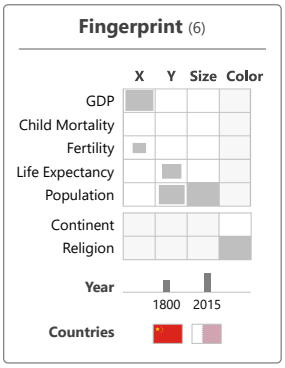


Figure 5.10: Fingerprint visualization used for the Gapminder interaction dataset (Walchshofer et al., 2021).

for the country selection. We then constructed a compound distance metric similar to Gower's distance (Gower, 1971).

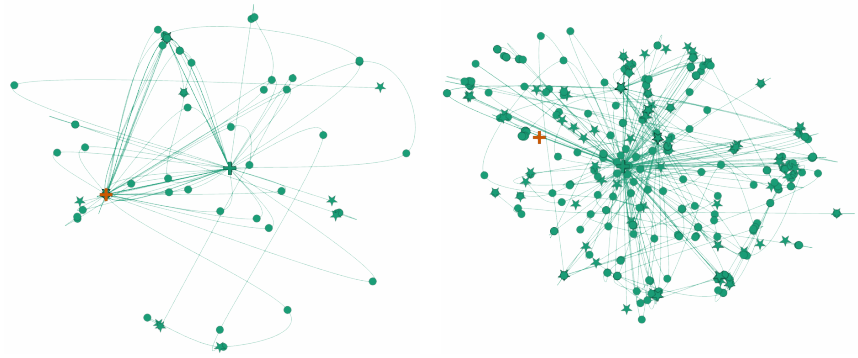
In our second use case, we applied Provectories to data from a previous study by Gadhave et al. (2021). Here, users interacted with a scatterplot and received AI suggestions for point selections based on predicted tasks. Interaction provenance was recorded with the Trrack library (Cutler et al., 2020), and each application state consisted of the user's point selection in the scatterplot. Here, in contrast to the Gapminder example, we constructed a more *semantic* representation space (Walchshofer et al., 2021). We divided the original scatterplot into a 10 by 10 grid and counted the number of selected points in each cell. We then used a flattened list of these counts as the vector representation, and compared these vectors using the cosine distance. This way, we made sure that it was not important that users selected exactly the same points in the plot for state-space representations to be similar. Instead, it was enough when users selected points in a similar region.

Implementation & Visualization Details We analyzed the data from both user studies in the Projection Path Explorer, using both t-SNE and UMAP embeddings. We compared these *attribute*-driven layouts with a *topology*-driven layout, where points are placed only based on direct connections in the sequences (see Walchshofer et al. (2021) for a detailed description). We used categorical user and task labels as metadata for color and shape encodings. We also introduced two specialized fingerprint visualizations. For the Gapminder data, we opted for a tabular view of the categorical attributes (see Figure 5.10). We used a size encoding to indicate the frequency of the settings within the selection. For countries, we listed the flags and indicated the frequencies with opacity. For the data from the study by Gadhave et al. (2021), we simply showed the scatterplots of the selected points corresponding to each state, with an opacity encoding to indicate frequency.

Results Before analyzing the results from the Gapminder user study, we produced artificial interaction data to understand how certain sequences of interactions would appear in the Provectories visualizations. For example, we created artificial interaction sequences in which only the year was changed incrementally. We then identified similar patterns to the ones from the artificial user data in the visualizations of the real user interactions. Furthermore, we found strong visual differences between the tasks that the users had to perform in the study. In particular, easy identification and lookup tasks could be readily distinguished from more challenging exploration tasks.

This observation was also true for the second study based on interactions with scatterplots. Figure 5.11 shows the Provectories visualizations for an easy and a difficult cluster selection task, as defined by Gadhave et al. (2021). For this dataset, the summary visualizations of the scatterplots also helped us to reveal that users tended to select points in outlier tasks in the typical Western reading direction (left-to-right, top-to-bottom).

For a more detailed discussion of all insights gained with Provectories, see Walchshofer et al. (2021). There, we also summarize the relative strengths



(a) Easy cluster selection task (b) Difficult cluster selection task

Figure 5.11: Provectories visualizations for an easy and a hard cluster identification task from the user intent study by Gadhave et al. (2021). The ground truth is indicated as +. Plots taken from Walchshofer et al. (2021).

and weaknesses of the different layout approaches.

5.5 DISCUSSION

In this section, we first summarize the validity and usefulness of the different patterns introduced in Section 5.2.3. We then discuss several challenges related to hyperparameters of embeddings, the drawing of trajectories, distance metrics, and the design of the fingerprint visualization. Finally, we discuss recent developments of the Projection Path Explorer and give an outlook on future work.

5.5.1 *Patterns in the Embedding Space*

Based on the results described in Section 5.4 and our experience with additional data from Go games and user interactions, we are confident that finding patterns (P1–P10, as described in Section 5.2.3)—in combination with interactive exploration of the detail and fingerprint views—can provide interesting insights. However, we want to stress the importance of verifying that the patterns found are "real" and not just the side effects of the dimensionality reduction technique.

In general, patterns which are stable across multiple different embeddings are most useful (here, embeddings can differ in terms of technique or hyperparameters). In the case of Rubik's cube, for instance, we found the shared trajectory bundles (P7) of both solution strategies to be stable. Likewise, the additional "coils" for lengthy rotation sequences appeared across multiple t-SNE and UMAP runs (see also Figure 5.1, which shows a different embedding than Figure 5.4).

Generally speaking, we found the dense point-cluster patterns (P1–P3) and the resulting path bundle-patterns (P7) to be most reliable across all application scenarios. Sparse patterns (P4–P6) are harder to relate to the real-world processes, as they cover many, often considerably different, high-dimensional states. Only when the association with the real-world process is straightforward (e.g., with random initialization of algorithms), have we found these patterns to be useful anchor points in the analysis.

In the case of the pattern P9 in the neural network example (Figure 5.8 in Section 5.4.3), we were surprised how well the state-space velocity correlated

with the "true" velocity (i.e., the learning rate). We think that this pattern has considerable potential in exploring processes that converge with different speeds.

Finally, the hypothetical pattern P8, which would arise from trajectory bundles with different directions, could not be found in any of the examples presented. However, we are currently investigating user-interaction data and expect to find such patterns when users backtrack to revisit earlier parts of their analysis.

5.5.2 *Dimensionality Reduction*

Clearly, when high-dimensional states are represented in a two-dimensional embedding space, it is impossible to preserve all original distance information. Additionally, hyperparameters of the projection techniques must be set appropriately. Depending on the dimensionality reduction technique, several challenges arise.

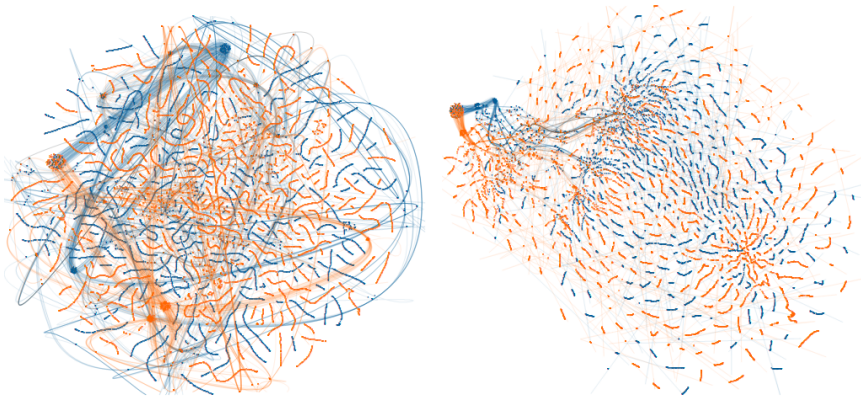
The most obvious challenge when using t-SNE is to set the perplexity. Figure 5.12 shows the same data as Figure 5.7, but embedded with perplexity values of (a) 5, and (b) 3000. Two types of patterns are stable across all perplexity values: the presence of path bundles at the early games stages and the chains of states towards the ends. However, in the case of perplexity 5 the path bundles appear more "messy", and regions of more densely packed intermediate states (as in Figure 5.7 (b–e)) cannot be made out. Both of these differences are a result of using a perplexity value that was too low to preserve global structure.

While van der Maaten (2020) stated that "typical values for the perplexity range between 5 and 50", we generally found higher perplexity values to be more useful for our purposes. Oskolkov (2019) suggested a square-root law, that is, choosing a perplexity of $\sqrt{N}$ for a dataset with N points. We suggest experimenting with even higher perplexity values. Plots often remain relatively stable in a wide perplexity range above $\sqrt{N}$. Higher perplexity values work around the fact that the objective function of t-SNE, as compared to that of UMAP, does not strictly penalize for two points with large high-dimensional distance to be embedded close together.

Another challenge related to t-SNE is that the result heavily depends on the initialization. In the t-SNE implementation that we used (Poličar et al., 2019), random initialization of low-dimensional positions can be replaced by initialization with PCA results. In our experience, this PCA initialization greatly improves comparability of multiple embeddings, even across different perplexity values.

Occasionally, we experienced some unexpected behavior with t-SNE: We noticed significant "jumps" between similar states. While t-SNE is known for sometimes putting points of high-dimensionally large distance close to each other in the embedding space (see above), it should not put high-dimensionally close points very far apart in the embedding. In experiments with user interaction-data, we noticed that UMAP did not produce these jumps as often. However, UMAP introduces other challenges related to

setting the minimal-distance parameter such that interesting patterns related to dense clusters are not hidden.



(a) t-SNE, perplexity 5

(b) t-SNE, perplexity 300

Figure 5.12: Embeddings (t-SNE) of chess games from Figure 5.7, re-run with different perplexity values.

5.5.3 *State-space Representations and Distance Metrics*

Choosing a meaningful combination of state-space representation and distance metric for constructing decision trajectories can be challenging. As already hinted at in Section 5.4.1 about Rubik's cube solution strategies, it might not always be possible to find a state-space representation that can accurately preserve the real-world notion of distances between states. For the Rubik's cube example, a combination of representation and metric would be ideal if it put two cube states at a distance of 1 if one state could be reached from the other in a single quarter-turn of a cube face. Even if such a representation existed, it would have to be based on finding the shortest path between two cube states in terms of rotations. An algorithm capable of solving Rubik's cube with the shortest possible sequence of moves has been termed God's Algorithm, and proving the lower bound of necessary moves alone took 35 CPU-years in 2010 (Rokicki et al., 2010). To the best of our knowledge, there is still no efficient implementation for finding the shortest path between two arbitrary states. Therefore, the Rubik's cube visualization shows that sometimes a simpler state-space representation must be used in the hope that the representation and metric preserve interesting features of the real-world processes.

Furthermore, for many applications certain symmetries could be exploited in the state space. In the case of the Rubik's cube, many cubes are equivalent with regards to solutions involving, for instance, the yellow or green cross as a first checkpoint. Likewise, chessboard states could be seen as equivalent if the white and black pieces are swapped. For chess, it could be argued that this symmetry is broken by the rule of white always performing the first move. Nonetheless, reflection-symmetrical states could be regarded as equivalent. In case of Rubik's cube, we broke the symmetries consciously by forcing each cube to pass the same checkpoint (yellow cross)—essentially disregarding more efficient solutions involving different checkpoints.

For both the chess and the Rubik's cube visualizations, disregarding the

symmetry still led to interpretable trajectory patterns. However, many applications could benefit from a state-space representation and/or distance metric that takes symmetries into account. In recent experiments with Go game data, we found that only symmetry-preserving distance metrics based on wavelet or Fourier transforms of the Go boards led to interpretable visualizations.

5.5.4 *Construction of Trajectories*

In order to keep visualizations with hundreds of trajectories readable, interpolation between points is extremely important. However, each interpolation technique may introduce artifacts, such as overshooting splines. An example of such artifacts can be seen close to the final chain of points for each game in Figure 5.7.

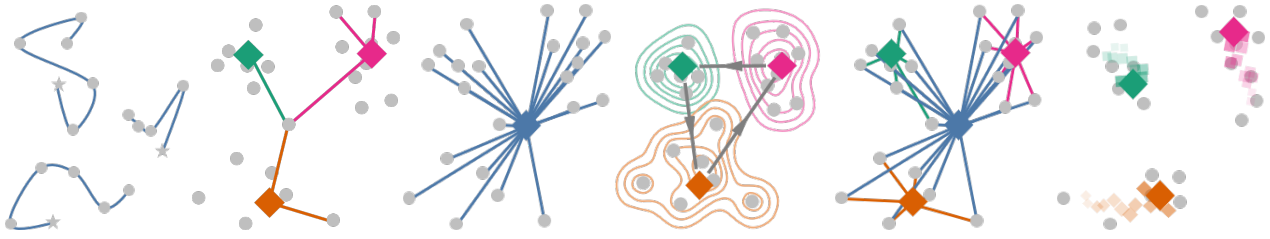
Additionally, not all interpolation techniques are equally suited for adding points to trajectories on the fly, which is necessary when our visualization technique is used for streaming data. Newly added points may change the shapes of trajectories through many of the previous points considerably. For our interactive physical Rubik's cube demonstrator (see Appendix B), we used cubic cardinal splines to make this addition as unobtrusive as possible. This spline interpolation technique makes sure that adding a point alters only the trajectory through the last three intermediate points.

Even with a suitably chosen interpolation technique, trajectories through the embedded state space may cause visual clutter. We plan to experiment with edge-bundling techniques to address this issue in future versions of our applications of decision trajectories. In addition to path bundling, smoothing of trajectories (e.g., by 1D Laplacian filtering) could help to reduce this clutter.

5.5.5 *Hierarchical Structures in Embeddings*

In a recent follow-up publication to this work (Eckelt et al., 2022), we introduced several extensions for the Projection Path Explorer. In particular, we focused on the analysis of different types of structures in embeddings by means of *layout enrichment* (Nonato & Aupetit, 2019). The different structures we considered were:

- individual points (i.e., "plain" scatterplots);
- item-to-item relationships, such as trajectories;
- item-to-group and group-to-item relationships;
- sequential group-to-group relationships; and



(a) Item to item (b) Item to groups (c) Group to items (d) Groups to groups (e) Groups within group (f) Reprojection

Figure 5.13: Visual encodings introduced to extend the Projection Path Explorer for analyzing different types of relationships in embeddings (Eckelt et al., 2022).

- hierarchical group-to-group relationships.

Here, groups can be defined in a variety of ways; they can be clusters based on high- or low-dimensional distances, via metadata or by user selections. We introduced different visual encodings and interactions to facilitate the exploration of the different relationships. Among the new encodings are centroid and density encodings for groups, star-shaped plots for group-to-item links, and tree-like visualizations for more complex hierarchies (see Figure 5.13a–e).

In addition to these new encodings for relationships, we also improved the functionality to explore different types of embeddings on the fly, addressing some of the challenges discussed above. New embeddings can be calculated either from scratch or using the previous positions as seeds. During the calculation of the embedding, the scatterplot is continually updated, and centroid traces (Figure 5.13f) are shown to help users keep track of where groups of interest are moving.

Because these new visual encodings generalize the Projection Path Explorer even further, we renamed the tool to Projection *Space* Explorer. We also constructed a default summary visualization for items and groups, that works with any combination of attributes. For a single point, all attributes are simply listed. For a group of points, histograms show the subset's value distributions compared to the whole set (similar to Stahnke et al., 2016). In the publication (Eckelt et al., 2022), we discuss design considerations for domain-specific summary and comparison visualizations, including one developed for a recent adaptation of the Projection Space Explorer for chemical datasets (Humer, Heberle, et al., 2022).

5.5.6 *Future Work*

As stated in Section 5.4.2, we plan to examine the correlation between players' strategies and patterns in trajectories of chess games in more detail. To this end, we are currently collaborating with an expert chess player. We have started to apply our visualization technique to games played by modern chess engines (see, e.g., Silver et al., 2018). For these embeddings, we switched from the simple state-space representation introduced in Section 5.4.2 to one based on the engine's internal representation (i.e., a neural network's latent space representation). Based on the expert's feedback, we are also paying special attention to the openings, where trajectory bundles (P7) are particularly important for the analysis. More general ideas for future work are discussed in Section 8.1.

5.6 CONCLUSION

In this chapter, we explored visual patterns in projected problem-solution paths. This was made possible by viewing the decisions—whether made by humans or resulting from the rules of an algorithm—as transitions between states in a high-dimensional representation space. To reveal and explore the patterns, we projected multiple paths through the high-dimensional state

space as trajectories in a shared, low-dimensional embedding space. We found this approach sufficiently general to be applied in various application domains: Rubik's cube solution algorithms, Chess games, neural network training, and meta-analysis of user interactions.

In all our applications, we used our interactive visualization prototype, Projection Path Explorer, to analyze the embedded trajectories. This general prototype can be adapted easily to new domains by defining new encodings for the detail view and the fingerprint encoding for state clusters. We hope that our work will inspire further studies of projected decision trajectories in a variety of interesting application domains.

PROJECTIVE LATENT INTERVENTIONS

6

6.1 INTRODUCTION

The interpretation of classification models is often difficult due to a high number of parameters and high-dimensional latent spaces. Dimensionality reduction techniques are commonly used to visualize and explain latent representations via low-dimensional embeddings. These embeddings are useful to identify problematic classes, to visualize the impact of architectural changes, and to compare new approaches to previous work. However, there is a lot of debate about how well such mappings represent the actual decision boundaries and the resulting model performance.

In this work, we aim to change the paradigm of passive observation of mappings to active interventions during the training process. We argue that such interventions can be useful to mentally connect the embedded latent space with the classification properties of a classifier. We show that in some situations, such as class-imbalanced problems, the manual interventions can also be used for fine-tuning and targeted performance gains. This means that practitioners can prioritize the decision boundary for certain classes over the others simply by manipulating the embedded latent space. The overall idea of our work is outlined in Figure 6.1. We use a neural-network-based parametric implementation of *t*-distributed stochastic neighborhood embeddings (*t*-SNE) (Min et al., 2010; van der Maaten, 2009; van der Maaten & Hinton, 2008) to inform the training process by back-propagating the manual manipulations of the embedded latent space through the classification network.

6.2 CONTEXT & CONTRIBUTION

Low-dimensional representations of high-dimensional data have been subject to scientific research for many decades (McInnes et al., 2018; Mead, 1992; Tenenbaum, 2000; van der Maaten & Hinton, 2008; Wold et al., 1987). When used to visualize latent spaces of artificial neural networks, these methods are commonly treated as independent modules and applied to a selected part of the representation (e.g., the penultimate layer of a discriminator network). However, these embeddings are often spatially inconsistent during training

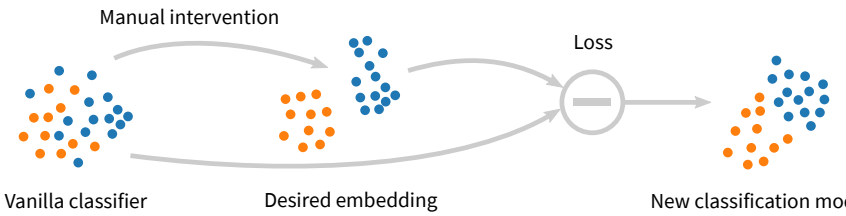


Figure 6.1: PLIs define a desired embedding, which is subsequently used to inform the training or fine-tuning process of a classification model in an end-to-end way.

from epoch to epoch and cannot inform the training process through back-propagation. (Min et al., 2010; van der Maaten, 2009) proposed to learn mappings through a neural network. This approach has the advantage that it can be directly integrated into an existing network architecture enabling end-to-end forward and backward updates. While unsupervised dimensionality reduction techniques have been used as part of deep learning workflows (Chen et al., 2018; Lee et al., 2015; Rusu et al., 2018; Tomar & Rose, 2014) and for visualizing latent spaces (Erhan et al., 2010; Rauber et al., 2017), we are not aware of any previous work that exploited parametric embeddings for a direct manipulation of learned representations. This shaping of the latent space relates our approach to metric learning (Bellet et al., 2013; Kulis, 2012). Metric learning makes use of specific loss functions to automatically constrain the latent space, but does not allow manual interventions. PLIs are general enough to be combined with concepts of metric learning.

We introduce Projective Latent Interventions (PLIs), a technique for (a) understanding the relationship between a classifier and its learned latent representation, and (b) facilitating targeted performance gains by improving latent space clustering. We discuss an application of PLIs in the context of anatomical standard plane classification during fetal ultrasound imaging.

6.3 METHOD

Projective Latent Interventions (PLIs) can be applied to any neural network classifier. Consider a dataset $X = \{x_1, \dots, x_N\}$ with N instances belonging to K classes. A neural network C was trained to predict the ground truth labels g_i of x_i, where $g_i \in \{y_1, \dots, y_K\}$. Let $C_l(x_i)$ be the activations of the network's lth layer, and let the network have L layers in total.

Given C, PLIs consist of three steps: (1) training of a secondary network $\tilde{E}$ that approximates a given non-linear embedding $E = \{y_1, \dots, y_N\}$ for the outputs $C_l(x_i)$ of layer l; (2) modifying the positions y_i of embedded points, yielding new positions y_i'; and (3) retraining C, such that $\tilde{E}(C_l(x_i)) \approx y_i'$. In the following sections, we will discuss these three steps in detail.

6.3.1 *Parametric Embeddings*

The embeddings used for PLIs are parametric approximations of t-SNE. For t-SNE, distances between high-dimensional points z_i and z_j are converted to probabilities of neighborhood p_{ij} via Gaussian kernels. The variance of each kernel is adjusted such that the perplexity of each distribution matches a given value. This perplexity value is a smooth measure for how many nearest neighbors are covered by the high-dimensional distributions. Then, a set of low-dimensional points is initialized and likewise converted to probabilities q_{ij}, this time via heavy-tailed t-distributions. The low-dimensional positions are then adjusted by minimizing the Kullback–Leibler divergence $\mathrm{KL}(p_{ij} \| q_{ij})$ between the two probability distributions.

Given a set of d-dimensional points $z_i \in \mathbb{R}^d$, t-SNE yields a set of d'-dimensional points $z' \in \mathbb{R}^{d'}$. However, it does not yield a general function $E : \mathbb{R}^d \to \mathbb{R}^{d'}$ defined for all $z \in \mathbb{R}^d$. It is thus impossible to add new points

to existing *t*-SNEs or to back-propagate gradients through the embeddings.

In order to allow out-of-sample extension, van der Maaten (2009) introduced the idea of approximating *t*-SNE with neural networks. We adapt van der Maaten's approach and introduce two important extensions, based on recent advancements related to *t*-SNE (Poličar et al., 2019): (1) *PCA initialization* to improve reproducibility across multiple runs and preserve global structure; and (2) *approximate nearest neighbors* (Dong et al., 2011) for a more efficient calculation of the distance matrix without noticeable effects on the embedding quality.

Our approach is an unsupervised learning workflow resulting in a neural network that approximates *t*-SNE for a set of input vectors $\{z_1, \dots, z_N\}$ given a perplexity value Perp. We only take into account the k approximate nearest neighbors, where $k = \min(3 \times \text{Perp}, N - 1)$. In contrast to the simple binary search used by van der Maaten (2009), we use Brent's method (Brent, 2002) for finding correct variances of the kernels. Optionally, we pretrain the network such that its 2D output matches the first two principal components of z_i. In the actual training phase, we calculate low-dimensional pairwise probabilities q_{ij} for each input batch, and use the KL-divergence $\text{KL}(p_{ij} \| q_{ij})$ as a loss function.

While van der Maaten (2009) used a network architecture with three hidden layers of sizes 500, 500, and 2000, we found that much smaller networks (e.g., two hidden layers of sizes 300 and 100) are more efficient and yield more reliable results. The *t*-SNE-approximating network can be connected to any complex neural network, such as a convolutional neural network (CNN) for medical image classification.

6.3.2 *Projective Latent Constraints*

Once the network $\tilde{E}$ has been trained to approximate the *t*-SNE, new constraints on the embedded latent space can be defined. This is most easily done by visualizing the embedded points, $y_i = E(C_l(x_i))$, in a scatter plot with points colored categorically by their ground truth labels g_i. For our applications, we chose only simple modifications of the embedding space: shifting of entire class clusters[1], and contraction of class clusters towards their centers of mass. The modified embedding positions y_i' are used as target values for the subsequent regression learning task.

[1] The class cluster for class γ_j is simply the set of points $y_i = E(C_l(x_i))$ for which $g_i = \gamma_j$.

In this work, we focus on class-level interventions because their effect can be directly measured via class-level performance metrics and they do not require domain-specific interactive tools that would lead to additional cognitive load. In principle, arbitrary alterations of the embedded latent space are possible within our technique.

6.3.3 *Retraining the Classifier*

In the final step, the original classifier is retrained with an adapted loss function $\mathscr{L}_{\text{PLIs}}$ based on the modified embedding:

$$\mathscr{L}_{\text{PLIs}}(x_i, g_i, y_i') = (1 - \lambda)\, \mathscr{L}_{\text{class}}(C_L(x_i), g_i) + \lambda\, \mathscr{L}_{\text{emb}}(\tilde{E}(C_l(x_i)), y_i'). \quad (6.1)$$

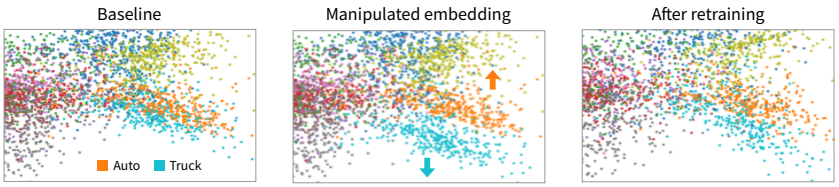


Figure 6.2: Detail views of the embedded latent space before (left), during (center) and after (right) Projective Latent Interventions for classification of CIFAR-10 images, focusing on the classes *Truck* and *Auto*.

The new loss function combines the original classification loss function $\mathscr{L}_{\text{class}}$, typically a cross-entropy term, with an additional term $\mathscr{L}_{\text{emb}}$. Minimization of $\mathscr{L}_{\text{emb}}$ causes the classifier to learn new activations that yield embedded points similar to y' (using the given embedding function $\tilde{E}$). As $\tilde{E}$ is simply a neural network, back-propagation of the loss is straightforward. In our experiments, we use the squared euclidean distance for $\mathscr{L}_{\text{emb}}$ and test different values for the weighting coefficient λ. We also experiment with only counting the embedding loss for instances of classes that were altered in the embedding.

6.4 EXPERIMENTS

6.4.1 *MNIST and CIFAR*

As a proof of concept, we applied PLIs to simple image classifiers: a small multilayer perceptron for MNIST (LeCun, 2005) images and a simple CNN for CIFAR-10 (Krizhevsky, 2009) images. For MNIST, the embedded latent space after retraining generally preserved the manipulations well, when class clusters were contracted and/or translated. The classification accuracy only changed insignificantly (within a few percent over wide ranges of λ). Typical results for the CIFAR-10 classifier are shown in Figure 6.2, where the goal of the Projective Latent Interventions was to reduce the model's confusion between the classes *Truck* and *Auto*, by separating the respective class clusters. When comparing a classifier trained for 5+4 epochs with $\mathscr{L}_{\text{class}}$ to one trained for 5 epochs $\mathscr{L}_{\text{class}}$ + 4 epochs $\mathscr{L}_{\text{PLIs}}$, the latter showed a relative increase of target-class-specific F_1-scores by around 5 %, with the overall accuracy improving or staying the same. The embeddings after retraining, as seen in Figure 6.2, reflected the manual interventions well, but not as closely as in the case of MNIST. We also found that, in the case of CNNs, using the activations of the final dense layer ($l = L$) yielded the best results.

6.4.2 *Standard Plane Detection in Ultrasound Images*

We tested our approach on a challenging diagnostic view plane classification task in fetal ultrasound screening. The dataset consists of about 12,000 2D fetal ultrasound images sampled from 2,694 patient examinations with gestational ages between 18 and 22 weeks. Eight different ultrasound systems of identical make and model (GE Voluson E8) were used for the acquisitions to eliminate as many unknown image acquisition parameters as possible. Anatomical standard plane image frames were labeled by expert sonographers as defined in the UK FASP handbook (NHS, 2015). We selected a subset of images that tend to be confused by established models (Baumgartner et al.,

2017): Four Chamber View (4CH), Abdominal, Femur, Spine, Left Ventricular Outflow Tract (LVOT), and Right Ventricular Outflow Tract (RVOT) and Three Vessel View (3VV). RVOT and 3VV were combined into a single class after clinical radiologists confirmed that they are identical. We split the resulting dataset into 4,777 training and 1,024 test images.

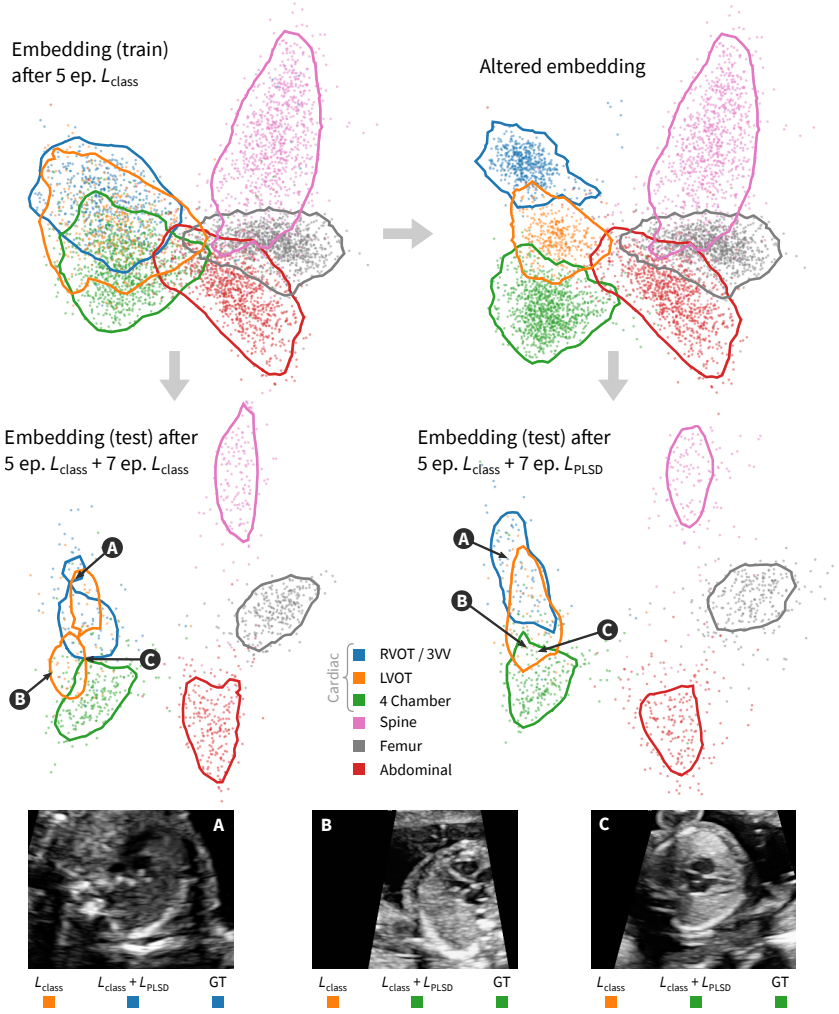


Figure 6.3: Projective Latent Interventions for standard plane classification in fetal ultrasound images. Top left: embedding of the baseline network's output (train) after 5 epochs of classification training ($\mathscr{L} = \mathscr{L}_{\text{class}}$). Top right: altered output embedding (train) with manually separated cardiac classes. Center left: Output embedding (test) after resuming standard classification training for 7 epochs ($\mathscr{L} = \mathscr{L}_{\text{class}}$), starting from the baseline classifier (top left). Center right: embedding (test) after resuming training with an updated loss function ($\mathscr{L} = \mathscr{L}_{\text{PLIs}} = 0.9\,\mathscr{L}_{\text{class}} + 0.1\,\mathscr{L}_{\text{emb}}$), starting again from the baseline classifier (top left). For easier comparability, class-specific contour lines at a density threshold of $1/N$ are shown, where N is the total number of train or test images, respectively. Performance measures for the classifiers are given in Table 6.1. Bottom: Three example images that were successfully classified after applying PLSD. For each image, the positions in both embeddings are indicated.

The architecture of our baseline classifier is SonoNet-64 (Baumgartner et al., 2017). The network was trained for 5 epochs with pure classification loss, i.e., $\mathscr{L} = \mathscr{L}_{\text{class}}$. We used Kaiming initialization, a batch size of 100, a learning rate of 0.1, and 0.9 Nesterov momentum. During these first five training epochs, we used random affine transformations for data augmentation (±15° rotation, ±0.1 shift, 0.7 to 1.3 zoom).

The 6-dimensional final-layer logits for the non-transformed training images were used as inputs for the training of the parametric t-SNE network. We used a fully connected network with two hidden layers of sizes 300 and 100. The t-SNE network was trained for 10 epochs with a learning rate of 0.01, a batch size of 500 and a perplexity of 50. We pretrained the network for 5 epochs to approximate a PCA initialization.

The ultrasound dataset is imbalanced, with 1,866 images in the three

Table 6.1: Global and class-specific performance measures for standard plane classification in fetal ultrasound images with and without PLIs, evaluated on the test set. The last two columns are weighted averages of the values for the three cardiac and the three non-cardiac classes, respectively.

		RVOT*	4CH	LVOT	Abd.	Femur	Spine	Cardiac	Other
Precision	Class. only	**0.82**	0.82	0.42	0.93	0.98	0.97	0.77	0.96
	PLIs	0.78	**0.85**	**0.61**	0.91	0.97	0.96	**0.80**	0.95
Recall	Class. only	0.38	0.94	**0.46**	0.96	0.97	0.94	0.76	0.96
	PLIs	**0.73**	0.94	0.28	0.96	0.97	0.94	**0.81**	0.96
F_1-score	Class. only	0.56	0.88	**0.44**	0.95	0.97	0.95	0.75	0.96
	PLIs	**0.76**	**0.89**	0.41	0.94	0.97	0.95	**0.80**	0.95

* The class labeled as RVOT also includes 3VV.

cardiac classes, and 2,911 images in the three non-cardiac classes. There are about twice as many 4CH images as RVOT/3VV, and three times as many 4CH images as LVOT. As a result, after five epochs of classification learning, our vanilla classifier could not properly distinguish between the three cardiac classes. This is apparent in the baseline embedding shown in Figure 6.3 (top left).

We experimented with PLIs to improve the performance for the cardiac classes, in particular for RVOT/3VV and LVOT. Figure 6.3 (top right) shows the case of contracting and shifting the class clusters of RVOT/3VV and LVOT.

After the latent interventions, training was resumed for 7 epochs with the mixed loss function defined in Equation 6.1. We experimented with different values for λ; all results given in this section are for $\lambda = 0.1$, which was found to be a suitable value in this application scenario. For a fair comparison, training of the baseline network was also resumed for 7 epochs with pure classification loss. In both cases, the remaining training epochs were performed without data augmentation, but with all other hyperparameters kept the same as for the vanilla classifier.

The outputs were then embedded with the parametric *t*-SNE learned on the baseline outputs (see Figure 6.3, center). By resuming the training with included embedding loss, the clusters for the three cardiac classes assume relative positions that are closer to those in the altered embedding. The contraction constraint also led to more convex clusters for the test outputs. Figure 6.3 (bottom) shows three exemplary images that were misclassified in case of the pure classification loss model, but correctly classified after applying PLIs. Further inspection showed that most of the images that were correctly classified after PLIs (but not before) had originally been embedded close to decision boundaries.

Table 6.1 lists the class-specific precision, recall, and F_1-scores for the two different networks. By applying PLIs, the average quality for the cardiac classes could be improved without negatively affecting the performance for the remaining classes. In some experiments, we observed much larger quality improvements for individual classes. For example, in one case the F_1-score

for LVOT improved by a factor of two. In these extreme cases, however, local improvements were often accompanied by significant performance drops for other classes.

6.5 DISCUSSION

The insights gained from PLIs about the relationship between a classifier and its latent space are based on an assessment of the model's response to the interventions. This response can be evaluated on two axes: the *embedding response* and the *performance response*.

Simple classifications tasks, for which the baseline classifier already works well (e.g., MNIST) often show a considerable embedding response with only a minor performance response. This means that the desired alterations of the latent space are well reflected after retraining without strong effects on the classification performance. Such classifiers are flexible enough to accommodate the latent manipulations, likely because they are overparameterized. In more complex cases, such as CIFAR, the embedding response is weaker, but often accompanied by a more pronounced class-specific performance increase. For these cases, the learned representation seems to be more rigidly connected with the classification performance. Finally, the standard plane detection experiments showed that sometimes a minimal change in the embedding is accompanied by a considerable performance increase for the targeted classes. Here, the overall structure of the embedding seems to be fixed, but the classification accuracy can be redistributed between classes by injecting additional domain knowledge while allowing non-targeted classes to move freely.

In general, we found that too severe alterations of the latent space cannot be preserved well since the embeddings are based on local information. Furthermore, seemingly obvious changes made in the embedding may contradict the original classification task due to the non-linearity of the embedding. The strength of PLIs is that a co-evaluation of the two components of the loss function can reveal these discrepancies. As a result, even when PLIs cannot be used for improving a classifier's performance, it can still lead to a better understanding of the flexibility of the model and/or the trustworthiness of the embedding.

In future work, we would like to experiment with parametric versions of different dimensionality reduction techniques and explore the potential of instance-level manipulations controlled via an interactive visualization.

6.6 CONCLUSION

We introduced Projective Latent Interventions, a promising technique to inject additional information into neural network classifiers by means of constraints derived from manual interventions in the embedded latent space. PLIs can help to get a better understanding of the relationship between the latent space and a classifier's performance. We applied PLIs successfully to obtain a targeted improvement in standard plane classification for ultrasound images without negatively affecting the overall performance.

PARADIME

7

Dimensionality reduction (DR) is one of the standard strategies for visualizing high-dimensional data. The general concepts of DR have been known and applied for over a century (Abdi & Williams, 2010) in the form of linear techniques such as principal component analysis (PCA). In the last decades, however, nonlinear DR techniques gained popularity. The most prominent modern techniques are t-distributed stochastic neighborhood embedding (t-SNE) (van der Maaten & Hinton, 2008) and uniform manifold approximation and projection (UMAP) (McInnes et al., 2018). Both t-SNE and UMAP rely on pairwise inter-item relationship information from high-dimensional data to construct embeddings in a low-dimensional space, with the goal of preserving key "structures" of the original data.

One shortcoming of such relationship-based DR techniques is that new items cannot readily be added to existing embeddings without having to re-compute all pairwise relationships. To address this shortcoming, researchers have developed *parametric* DR techniques. In parametric DR, embeddings are created by parameterized functions (e.g., neural networks) that are trained on the high-dimensional data. While several implementations of parametric DR exist, they are usually tailor-made variations of existing techniques, and they are often difficult to customize or extend. This "scattered" nature of existing parametric DR techniques is surprising considering the strong conceptual similarities between various nonlinear DR approaches (Böhm et al., 2022).

We believe that the potential of parametric DR is underexplored, and that both, the visualization and machine learning communities could benefit from a framework that makes it easy to experiment with such techniques. To fill this gap, we introduce *ParaDime*, a unifying framework for parametric DR. Our contribution with ParaDime is threefold:

- We introduce a generalizing grammar, which formalizes all the steps and building blocks necessary to specify a parametric DR routine.
- We show how this grammar can be used not only to reproduce existing

Figure 7.1: ParaDime is a framework for parametric dimensionality reduction. Left: Data flow in a single training phase of a ParaDime routine. Right: Parametric t-SNE trained on a subset of 5000 images from the MNIST dataset (LeCun, 2005) and applied to the whole dataset.

parametric DR techniques, but also to experiment with new ideas.

- We present an implementation of the grammar with a focus on usability and customization.

This chapter is structured as follows: in Section 7.1 we summarize related work on generalizing and constraining DR techniques; in Section 7.2 we explain how similarities observed in these techniques give rise to a grammar of parametric DR, and how ParaDime implements this grammar; we then show how ParaDime can be used to reproduce existing techniques (Section 7.3) and how it facilitates experimentation with new ideas (Section 7.4); in Section 7.5 we discuss design and implementation choices, non-parametric embeddings, limitations, and future work.

7.1 Related work on generalizing and constraining DR

An overview of various dimensionality reduction techniques is given in Section 2.2. In addition to these techniques, Minimum distortion embeddings (MDEs) (Agrawal et al., 2021) are most closely related to our ParaDime approach. MDEs and ParaDime are related in two ways: (*i*) conceptually both aim to unify several existing techniques in a common framework; and (*ii*) both use PyTorch's autograd functionality Paszke et al., 2017; Paszke et al., 2019 for the optimization. MDEs, however, are non-parametric and support out-of-sample-extension only via a combination of anchoring constraints and solving a new MDE sub-problem. In addition, MDE's strong focus on formalized distortions and penalty functions makes it challenging to map them to other techniques (see, e.g., the comparison of t-SNE and UMAP by Sainburg et al. (2021) vs. how Agrawal et al. (2021) relate penalties to UMAP). ParaDime instead focuses on (potentially transformed) pairwise relations between data items, which allows several existing techniques to be directly "translated" into its common framework. Furthermore, ParaDime uses neural networks to compute embeddings and additional well-established loss functions from other tasks, such as classification and reconstruction, can be readily included in ParaDime DR routines. This relates ParaDime to other techniques that add constraints to dimensionality reduction (Vu et al., 2022). In summary, ParaDime combines ideas from unifying nonlinear DR in general (Agrawal et al., 2021; Böhm et al., 2022) with parametric DR (Sainburg et al., 2021; van der Maaten, 2009), and it allows flexibility to include alternative learning paradigms.

7.2 The ParaDime grammar of parametric DR

The similarities between the different neighbor- and distance-based DR techniques outlined above inspired us to develop a unifying interface for specifying parametric dimensionality reduction *routines*. In ParaDime, routines are complete data processing pipelines that include all the specifications necessary to arrive from a given dataset at a trained parametric DR model. In this section, we describe how routines can be specified with the ParaDime

grammar of parametric DR. This approach follows the tradition of grammars and grammar-like structures in the visualization community, such as Vega (Satyanarayan et al., 2016), Vega-Lite (Satyanarayan et al., 2017) and Encodable (Wongsuphasawat, 2020) for general visualizations, Atom (Park et al., 2018) for unit visualizations, Gosling (L'Yi et al., 2022) for genome visualizations, and Neo (Görtler et al., 2022) for confusion matrices).

7.2.1 *Overview*

ParaDime generalizes parametric DR by breaking it down into several steps, outlined in the data flow graph in Figure 7.1. First, relations between all items in a given dataset are computed. Then, a batch of data is sampled in a training loop. The data batch is processed with a machine learning model, and new relations between all items in the processed batch are computed. The batch-wise relations are compared with an appropriate subset of the overall relations to compute an embedding loss. Additional losses may be added to the embedding loss. Finally, the losses are used to optimize the machine learning model via backpropagation.

The ParaDime grammar defines how the necessary building blocks for each of these steps are specified. We use YAML for these specifications due to its focus on readability (döt Net et al., 2021). A ParaDime specification requires the five base-level entries **`model`**, **`dataset`**, **`relations`**, **`losses`**, and **`training phases`**. In the following subsections, we explain each of these entries in detail.

7.2.2 *Model*

In parametric DR techniques, the embedding is performed by a machine learning model. During the training, the model is updated in such a way that a given objective function is minimized. In the ParaDime grammar, model specifications closely follow the idea behind PyTorch modules:

```
model:
  modules:
    - type: <module type>
      options: {...}
      submodules: [<modules>]
    - ...
  methods: [<method name>, ...]
```

PyTorch modules can contain other modules, allowing tree-like structures. Individual layers of neural networks are examples of leaf modules, where the module **`type`** would be `linear` or `conv2D`, with **`options`** specifying the layer dimensions. More important than the model structure, however, is a list of model methods that the model implements. These methods are later used by ParaDime to process data batches inside the training loops.

7.2.3 *Dataset*

The **`dataset`** in a ParaDime specification defines the training data for the routine. A dataset consists of any number of regular or derived entries.

Regular entries are simple, named data tensors. *Derived* entries are populated later, based on other data entries or data from relation computations. Details are as specified with **`data func`** and **`keys`**. This gives rise to the following general specification:

```
dataset:
  # regular entry
  - name: <attr name>
    data: <data array>
  # derived entry
  - name: <attr name>
    data func: <data function>
    keys:
      data: [<data attr name>, ...]
      rels: [<rel name>, ...]
  - ...
```

Our rephrasing of parametric UMAP in terms of ParaDime in Section 7.3.3 shows how derived entries can be used to set up complex initialization schemes. All entries in the dataset are named to allow them to be referenced later in the routine. Typical entries in a **`dataset`** are the original high-dimensional data or labels in supervised applications.

7.2.4 *Relations*

The **`relations`** entry of a ParaDime specification lists "recipes" for computing mutual relations between data items. Each relation recipe is specified either globally or on a batch level. As explained above, ParaDime computes `global` relations between all items in the dataset before any training begins; these are typically relations between the original, high-dimensional data points. In contrast, the computation of `batch`-wise relations is deferred to the actual training loops. The `batch`-wise relations are computed between items in a batch of data that has been processed by the model. A relation's **`type`** specifies how relations are computed; supported types are, for instance, exact pairwise distances (`pdist`) or approximate neighbor-based distances (`neighbor`). A relation's **`attr`** entry specifies which attribute of the dataset to use for computing relations. Possible **`options`** for relations are the distance function to be used (e.g., **`metric`**: `euclidean`) or algorithm-specific settings such as the number of nearest neighbors for `neighbor`-based relations. Finally, a list of **`transforms`** can be applied to the relations. Transforms can be used, for instance, to convert pairwise distances into perplexity-based probabilities of neighborhood as in t-SNE (van der Maaten & Hinton, 2008) (see Section 7.3.2). In total, the **`relations`** specification has the following structure:

```
relations:
  - name: <rel name>
    level: global | batch
    type: <rel type>
    options: { ... }
    attr: <data attr name>
    transforms:
      - type: <transform type>
        options: { ... }
      - ...
  - ...
```

Note that a routine can have any number of global or batch-wise relations. Like the dataset entries, they are named so that they can be referenced by the **losses** or in derived **dataset** entries.

7.2.5 *Losses*

Once ParaDime knows how to compute relations betwen data items, these relations can be used inside losses to construct objective functions that govern the training process. A ParaDime specification of a routine's **losses** has the following structure:

```
losses:
  - name: <loss name>
    type: <loss type>
    func: <loss function>
    keys:
      data: [<data attr name>, ...]
      rels: [<rel name>, ...]
      methods: [<model method>, ...]
  - ...
```

Each loss has a **type**, which defines how it behaves during training. A loss of type `relation` compares a subset of precomputed global relations to relations computed from a processed batch of data. Other types of losses are `classification`, which compares the model output for a data batch to given labels, or `reconstruction`, which compares the original input batch to an encoded and decoded version of the batch. To retain flexibility, each loss includes a specification of the **keys** that should be used to access the relevant model methods, attributes of the data, and/or the relations. Losses can be combined in the training by forming weighted compound losses, as explained in following subsection.

7.2.6 *Training Phases*

In ParaDime, the training of a routine is organized into **training phases**. Each training phase consists of **sampling** and **optimizer** specifications, a number of **epochs**, and a **loss** specification:

```
training phases:
  - epochs: <number of epochs>
    sampling:
      type: item | edge
      options: { ... }
    optimizer:
      type: <optim type>
      options: { ... }
    loss:
      components: [<loss name>, ...]
      weights: [<weight>, ...]
  - ...
```

The **sampling** type can be either item (simple sampling of batches of items) or edge (sampling of items based on relations between them). The edge-based sampling option allows ParaDime specifications of techniques based on negative edge-sampling (McInnes et al., 2018; Mikolov et al., 2013; Tang et al., 2016) or triplets (Chechik et al., 2010) (see example in Section 7.4.2). As already mentioned above, the **loss** in each training phase is a weighted compound loss, whose **components** are specified with the names of the **losses** defined earlier. Finally, the **optimizer** entry specifies which optimization technique to use (e.g., sgd (Bottou, 2010) and adam (Kingma & Ba, 2017)), along with options such as the learning rate or the momentum (Sutskever et al., 2013).

A ParaDime routine can have any number of training phases. Organizing the training into phases allows simple concepts, such as pretraining models to initialize embeddings, but also complex, multi-stage training schemes like in the example in Section 7.4.4.

7.2.7 *Implementation of the Grammar*

We implemented ParaDime in Python, choosing PyTorch (Paszke et al., 2019) as the machine learning framework. We discuss this choice briefly in Section 7.5.2. Each high-level component introduced in this section is realized as a class. ParaDime parses the specifications and sets up the corresponding Python objects. We provide a detailed documentation with examples and a less technical introduction of the building blocks of ParaDime routines online.[1] Paradime is pip-installable and the code is available on GitHub.[2]

[1] ParaDime documentation: https://paradime.readthedocs.io/en/latest/

[2] ParaDime code: https://github.com/einbandi/paradime

7.3 FRAMING EXISTING TECHNIQUES IN TERMS OF PARADIME

In this section, we show how (parametric extensions of) existing techniques can be specified in terms of the ParaDime grammar. To highlight the relevant parts of the specifications, we omit irrelevant ones for the **model** and **dataset**. Any model can be specified for all routines. If no **dataset** is specified, we assume that the dataset only contains a single high-dimensional data tensor that is accessed implicitly by the other specifications. Furthermore, we omit the **weights** list, since all examples in this section use only a single loss per training phase.

7.3.1 *Metric MDS*

Metric multidimensional scaling aims to find a configuration of points in low-dimensional space such that the pairwise distances match those of the high-dimensional data (Cunningham & Ghahramani, 2015). This can be specified with ParaDime through Euclidean pairwise distance relations and a mean square error loss between the two relations:

```yaml
relations:
  - name: dists hd
    level: global
    type: pairwise
    options:
      metric: euclidean
  - name: dists ld
    level: batch
    type: pairwise
    options:
      metric: euclidean
```

```yaml
losses:
  - name: mds
    type: relation
    func: mse
    keys:
      rels:
        - dists hd
        - dists ld
training phases:
  - loss:
      components: mds
      weights: 1
```

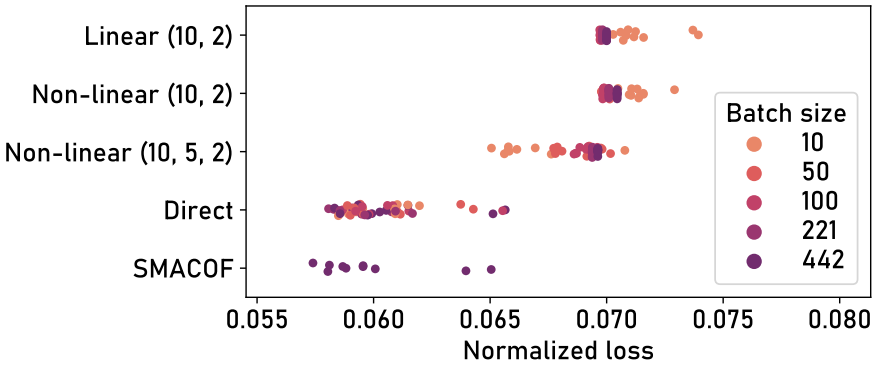


Figure 7.2: Normalized stress (Espadoto et al., 2019) for parametric versions of metric MDS compared with the non-parametric SMACOF implementation of scikit-learn (Pedregosa et al., 2011). All models were trained on the ten-dimensional diabetes dataset with 442 items (Efron et al., 2004).

Figure 7.2 shows the normalized stresses (Espadoto et al., 2019) for several ParaDime routines with different models and the above specification trained on a ten-dimensional diabetes dataset (Efron et al., 2004). The linear model was a simple matrix multiplication to map the ten-dimensional vectors to a two-dimensional embeddings space. The nonlinear models had an additional bias and used softplus as the activation function. The routine labeled "direct" is a non-parametric routine implemented with ParaDime (see discussion in Section 7.5.3). All ParaDime routines used the same optimizer (Adam; Kingma and Ba, 2017), learning rate (0.01) and number of epochs (500). The losses of the ParaDime routines are compared with that of the non-parametric scikit-learn implementation using the SMACOF algorithm (Kruskal, 1964). Note how the routines with linear and nonlinear models of size 10×2 performed almost identically. Adding another hidden layer of dimension five reduced the loss substantially, especially for smaller batch sizes. The average loss for batch size ten was less than 12 % above the average of the SMACOF baseline, despite the simplicity of the model and the fact that no hyperparameter tuning was performed. Interestingly, for the two models of size 10×2, smaller

batch sizes led to higher losses. The non-parametric implementation had losses similar to the SMACOF baseline.

7.3.2 *t-SNE*

The ParaDime specification for parametric t-SNE is as follows:

```
dataset:
  - name: main
    data: [...]
  - name: pca
    data func: pca
    keys:
      data: [main]
relations:
  - name: p
    level: global
    type: neighbor
    options:
      metric: euclidean
    transforms:
      - type: perplexity
        options:
          perplexity: <p>
      - type: symmetrize
      - type: normalize
  - name: q
    level: batch
    type: pairwise
    options:
      metric: euclidean
    transforms:
      - type: t-dist
        options:
          alpha: 1.
      - type: normalize
losses:
  - name: init
    type: position
    func: mse
    keys:
      data: [main, pca]
  - name: emb
    type: relation
    func: kl div
    keys:
      rels: [p, q]
training phases:
  # pca initialization
  - loss:
      components: [init]
      sampling:
        type: item
  # main embedding
  - loss:
      components: [emb]
      sampling:
        type: item
```

The t-SNE algorithm begins with calculating pairwise distances that are transformed into normalized and symmetrized probabilities of high-dimensional neighborhood based on a perplexity hyperparameter (van der Maaten & Hinton, 2008). In low-dimensional space, probabilities of neighborhood are calculated by transforming Euclidean distances with a Student's t-distribution (van der Maaten & Hinton, 2008). It is straightforward to define these two relations in ParaDime by making use of **`transforms`**. Note that the global relation specification contains `neighbor` rather than `pdist` as **`type`**, which tells ParaDime to use approximate nearest-neighbor-based distances. The two probability matrices are compared using the KL divergence. Before this step, modern implementations perform an initialization of the embedding with PCA coordinates (Poličar et al., 2019). However, the embedding coordinates cannot be initialized directly in a parametric DR routine. Instead, the model is pretrained to output coordinates that match the initialization, based on a `position`-type loss with a mean squared error loss function. The PCA coordinates can be supplied with a deferred dataset entry.

An example of a parametric t-SNE routine implemented with ParaDime is shown in the right part of Figure 7.1. It was trained on a subset of 5000 images of the MNIST dataset of handwritten digits (LeCun, 2005) with a perplexity of 100 and a learning rate of 0.01. The model had hidden layer dimensions

of 100 and 50 and used softplus for all activation functions. Figure 7.1 also shows the result of applying the trained model to the whole dataset.

7.3.3 *UMAP*

As discussed in Section 7.1, UMAP has several conceptual similarities to TSNE. As a consequence, its ParaDime specification looks relatively similar to that of TSNE:

```
dataset:
  - ...
  - name: spectral
    data func: spectral
    keys:
      rels: [p]
relations:
  - ...
    transforms:
      - type: connect
        options:
          neighbors: <n>
      - type: symmetrize
        options:
          sub prod: true
      - type: normalize
  - ...
    transforms:
      - type: cauchy
        options:
          spread: <s>
          min dist: <md>
```

```
losses:
  - name: init
    type: position
    func: mse
    keys:
      data:
        - main
        - spectral
  - name: emb
    type: relation
    func: cross entropy
    keys:
      rels: [p, q]
training phases:
  # spectral init
  - ...
  # main embedding
  - loss:
      components: [emb]
      sampling:
        type: edge
```

Here, we omitted entries that are the same as in the ParaDime specification for parametric t-SNE. The main differences for UMAP are (see, e.g., Sainburg et al., 2021):

- a `spectral` embedding based on the global relations is used for initialization;
- a connectivity-based transform is used instead of the perplexity-based one for t-SNE;
- batch-wise relations are transformed with a modified Cauchy distribution rather than a Student's t-distribution;
- `cross entropy` replaces the KL divergence; and
- negative-edge sampling is used instead of regular item-based sampling.

7.3.4 *Additional Neighbor-based Techniques*

ParaDime includes implementations of all **`relations`**, **`transforms`**, and **`data func`** methods specified in the examples above. With these methods, it is also possible to specify LargeVis (Tang et al., 2016), which basically combines t-SNE's high-dimensional relations with negative-edge sampling. LargeVis is not restricted to a specific transform for the low-dimensional (i.e., `batch`-wise) relations; instead, the authors state that "many probabilistic functions can

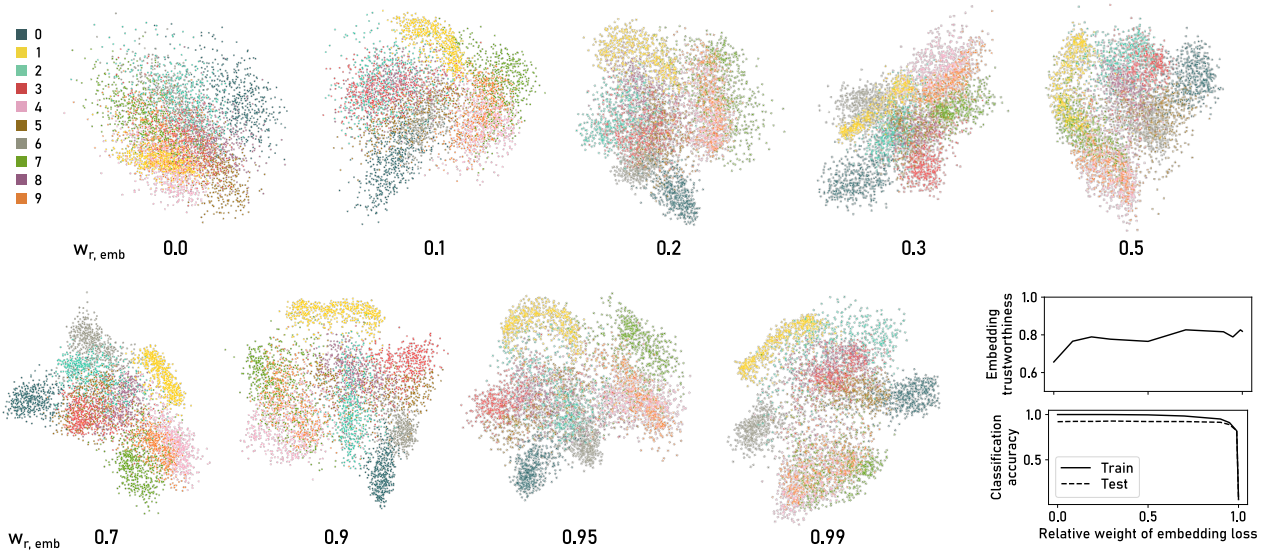


Figure 7.3: Embeddings of hybrid embedding/classification routines for the MNIST dataset (LeCun, 2005) created with ParaDime. The relative weight of the embedding loss component is indicated by $w_{r,\text{emb}}$, and the weight of the classification component was $1 - w_{r,\text{emb}}$. All embedding-related specification were the same as those of the ParaDime parametric UMAP routine. The routines were trained on a subset of 5000 randomly sampled MNIST images. The test accuracy was calculated on a different subset of 5000 images. Trustworthiness was calculated based on ten nearest neighbors.

be used" (Tang et al., 2016, p. 290). This aligns well with ParaDime's flexible concept of transforms.

Isomap is another neighbor-based technique, but it uses geodesic distances instead of probabilities of neighborhood (Tenenbaum, 2000). To specify Isomap with ParaDime, it suffices to implement either a new type of `relations` or a transform that converts Euclidean distances to geodesic distances.

7.3.5 *Classifiers & Autoencoders*

In addition to the `relation`-type loss used in all DR techniques discussed so far, ParaDime also provides losses for typical machine-learning tasks that are not limited to DR. In particular, the `classification` loss makes it straightforward to implement classification models if one of the dataset entries contains class labels:

```
dataset:
  - name: main
    data: [...]
  - name: labels
    data: [...]
```

```
losses:
  - type: classification
    func: cross entropy
    keys:
      data:
        - main
        - labels
```

Similarly, autoencoders can be concisely specified using the predefined `reconstruction` loss.

7.4 EXPERIMENTING WITH COMBINED TECHNIQUES

In this section, we present several application ideas for ParaDime. These example are supposed to show the versatility of the ParaDime specifications, and they should encourage experimentation with new ideas emerging from combinations of different losses.

7.4.1 *Hybrid UMAP for Embedding and Classification*

In Sections 7.3.3 and 7.3.5 we showed how to use ParaDime to specify a parametric version of UMAP and a simple classification model, respectively. In this section, we combine the two to create a hybrid embedding and classification routine, which uses a shared latent space for both tasks. We apply our multitask routine to the MNIST dataset of handwritten digits (LeCun, 2005).

As a model, we use a fully connected network with hidden-layer dimensions 100 and 50. The model has two output layers: one of dimension ten yielding the logits used for classification, and one of dimension two for the embedding. Both output layers are connected to the second hidden layer.

As explained above, UMAP uses edge-based sampling. When edge-based sampling is in ParaDime, each batch not only contains the pairs of vertices between the sampled edges, but also a simple list of unique data items suitable for other tasks such as classification. Therefore, losses that require item-based sampling can be readily added to routines that use negative-edge sampling. Below is the specification of our hybrid classification and embedding model, in which we omitted the losses and relations that were already defined earlier.

```
dataset:
  - <main data>
  - <labels>
relations:
  <UMAP rels>
losses:
  - <UMAP loss>
  - <classification loss>
```

```
training phases:
  - loss:
      components:
        - umap
        - class
      weights: <w>
```

Thanks to ParaDime's specification interface, the losses from above can simply be reused as components in a compound loss. Figure 7.3 shows nine embeddings created with different weights for the loss components. All routines were trained on the same subset of 5000 images from MNIST for 100 epochs and without any pretraining. Figure 7.3 also includes plots of the classification accuracy and the embedding trustworthiness (Espadoto et al., 2019; Venna & Kaski, 2001) as functions of the weight. Note that already a small weight on the embedding loss leads to a substantial class separation in the scatterplots. At the same time, the classification accuracy is not affected by the additional embedding task. The accuracy only suffers when the weight on the classification approaches zero. In particular weighting the embedding with values in the wide range of 0.5 to 0.95 produces visually "sensible" embeddings with relatively high trustworthiness and practically the same classification accuracy as the pure classifier. In fact, some of our experiments showed that the additional embedding loss can improve generalization of the classifier slightly. This observation is in line with the original motivation for multitask learning (Caruana, 1997).

Such a hybrid embedding and classification model could be the basis of a visualization tool in which users can add new points to existing embeddings. The predicted class labels could be used to visually encode the new data points and/or to inform users whether a new point lies in a region of the embedding where also other points of the same class are located.

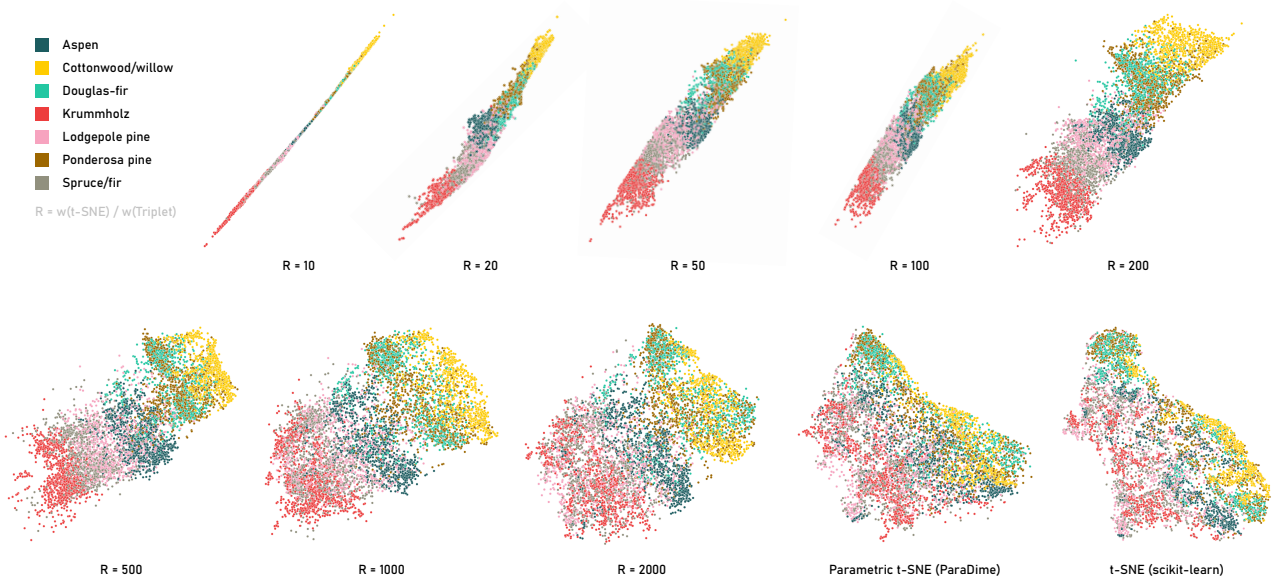


Figure 7.4: Supervised embeddings of a subset of the forest covertype dataset (Chechik et al., 2010). All embeddings labeled with R are supervised version of parametric t-SNE, where the supervision was included by means of a triplet loss based on the ground truth labels. R is the ratio of the weights of the t-SNE loss and the triplet loss. For comparison, embeddings created with scikit-learn's non-parametric t-SNE implementation and with a plain ParaDime t-SNE version (using item-based sampling and no triplet loss) are shown. The perplexity was 200 in all cases, and a class-balanced subset of 7000 items was used.

7.4.2 *Supervised t-SNE with Triplet Loss*

In this example we combine a parametric version of t-SNE (see Section 7.3.2) with a triplet loss (Chechik et al., 2010) to learn several supervised embeddings for the forest covertype dataset (Blackard & Dean, 1999). This is an example of a instance-level constraint as categorized by Vu et al. (2022).

The forest covertype dataset consists of 581 012 records with 54 attributes each. Each item corresponds to a 30 m × 30 m cell of a US region, and the attributes describes cartographic variables, such as elevation, slope, and distance to the next roadway. Each item is labeled with the ground truth value for the type of trees covering the cell (e.g., aspen, krummholz, and spruce/fir). The dataset is strongly imbalanced, with the prevalent being more than 100 times more frequent than the least. In this example, we use the first ten numerical attributes and sample an almost balanced subset of 7000 items.

Supervising t-SNE with an additional term based on triplets can be achieved easily thanks to the ParaDime interface. First, we use negative-edge sampling to construct triplets. In negative-edge sampling, batches of edges between items are sampled during training, rather than batches of individual items. In other applications of this sampling strategy (for example UMAP (McInnes et al., 2018)), a positive edge is sampled according to the probabilities of neighborhood of the two points (i.e., vertices). Then, a specified number of random negative edges for one of the two vertices are added. Negative edges are edges between two vertices for which the probability of neighborhood is zero. In this example, we instead create a probability matrix r_{ij} with:

$$r_{ij} = \begin{cases} 1 & g_i = g_j \\ 0 & g_i \neq g_j \end{cases}, \tag{7.1}$$

where g_i are the ground truth labels of the data. If we use this probability matrix for negative-edge sampling with a negative sampling rate of one,

we essentially sample one pair of vertices (a, b) with equal labels, and another pair (a, c) with different labels. The set of vertices a, b, c constitutes a triplet (Balntas et al., 2016; Chechik et al., 2010). We can then simply add an additional triplet loss.

This results in the following ParaDime specification, where `pairwise equal` stands for the global relation as defined by r_{ij} (we abridged the parts of the specification that match the t-SNE one from Section 7.3.2):

```
dataset:
  - name: main data
    data: [...]
  - name: labels
    data: [...]
  - <PCA>
relations:
  - <global t-SNE rel>
  - <batch t-SNE rel>
  - name: r
    level: global
    type: pairwise equal
losses:
  - <PCA init loss>
  - <t-SNE loss>
  - name: triplet
    type: triplet
    func: margin
    keys:
      data: [main data]
training phases:
  - <PCA init>
  - loss:
      components:
        - tsne
        - triplet
      weights: <w>
      sampling:
        type: edge
        options:
          rels: r
          rate: 1
```

Here, `margin` is the name of the following loss function that is applied to the triplets (Balntas et al., 2016; J. Wang et al., 2014):

$$L_{\text{triplet}}(a, b, c) = \max(d(a, b) - d(a, c) + m, 0), \tag{7.2}$$

where m is the margin hyperparameter.

Figure 7.4 shows eight version of embeddings specified this way, with different values for the loss weights. In all cases, the model was a fully connected neural network with hidden layer dimensions 100 and 50. Each embedding was initialized with a PCA-based pretraining for ten epochs with item sampling and a batch size of 500. As explained above, the main embedding phases used negative edge sampling, with 300 triplets being sampled in each batch. For comparison, Figure 7.4 includes a parametric t-SNE without the extra triplet loss and with regular item sampling. We also show the result of scikit-learn's non-parametric t-SNE. For all embeddings the perplexity value was set to 200.

For the triplet loss as defined above to be minimal, the distance along negative edges (i.e., between a pair of items with different labels) must be substantially larger than the distances along a positive edge. This pulls together items from the same class. If too much weight is put onto the triplet loss, this causes all items to condense on a single line, along which they are more or less ordered by their class label. As the weight of the triplet loss is reduced, the structure of the "pure" t-SNE is preserved increasingly, while classes are well separated (see, e.g., the embeddings for t-SNE/triplet loss weight ratios of 1000 in Figure 7.4). With vanishing weight on the triplet loss, the embedding still differs noticeably from the one that used item-based sampling; here the triplet sampling strategy might be disadvantageous, as it favors certain batch configurations over others.

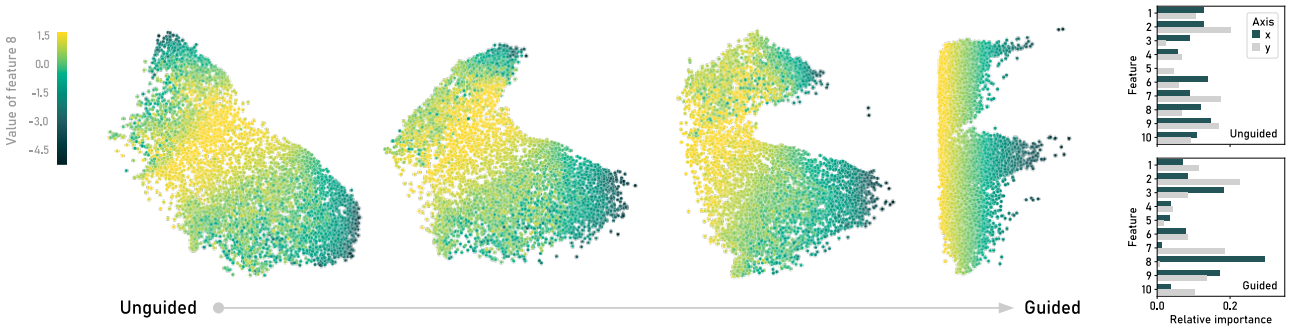


Figure 7.5: Attribute-guided embeddings of a subset of the forest covertype dataset (Chechik et al., 2010). Attribute guiding was implemented by combining t-SNE with a correlation loss which orders the data points along the x-axis by the value of the eighth feature (hillshade at noon). The weights for the embeddings shown are $(w_{\text{t-SNE}}, w_{\text{corr}}) = (1, 0)$, $(5000, 1)$, $(1000, 1)$, and $(100, 1)$, respectively. The bar chart on the right shows the feature importances for the learned embeddings, based on integrated gradients. Note the high importance of the eighth feature for the x-axis in the guided embedding.

One potential application idea for such supervised embeddings is an interactive visual interface for dataset exploration, where users can switch between a purely attribute-driven visualization (e.g., pure t-SNE) and the supervised one with more pronounced class separation. In the first mode, users could explore similarities and difference between all data points as usual, while the second mode would enable class-specific exploration without losing track of the overall structure.

7.4.3 *Attribute-guided Embeddings*

In this example, we again look at embeddings of the covertype dataset discussed in the previous section. This time, however, our primary interested is not in the class distribution, but in using specific attributes to guide the encodings. In particular, we use ParaDime to construct an embedding in which a specified direction correlates with one of the high-dimensional attributes. To this end, we define a new type of loss:

$$L_{\text{corr}}(a, b; i, j) = 1 - \left(\frac{\text{cov}(a_i, b_j)}{\sigma_{a_i} \sigma_{b_j}} \right)^2 . \tag{7.3}$$

Here, a and b are two data matrices with the same number of rows, and a_i and b_j refer to columns i and j, respectively; cov is the covariance, and σ is the standard deviation. This loss is equivalent to one minus the squared Pearson's correlation coefficient for the ith column of a and the jth column of b. During the training of our routine, a will be a batch of high-dimensional data and b the processed (i.e., embedded) two-dimensional batch.

Once a loss `corr` has been defined which uses the above function L_{corr} and applies it to the unprocessed and embedded versions of the input batch, we can simply construct a compound loss in a way equivalent to the other previous examples in this section. The loss components (t-SNE loss and correlation loss) can be weighted, and the dimensions that should correlate can be specified as options to the loss.

Figure 7.5 shows four examples of such attribute-guided embeddings with different weights. In all examples, i was set to eight and j to one, which means that the *Hillshade (noon)* attribute of the covertype dataset was constrained to correlate with the x-direction of the embedding. In the embeddings in Figure 7.5, the points are colored by the specified high-dimensional attribute

value. With increasing weight on the correlation loss, the embedding is distorted in such a way that the values decrease from left to right, while the remaining structure is preserved to some extent. Within a certain range of weights, the transition from unguided to strongly guided embeddings appears to be smooth, with the points "folding over" continuously to satisfy the constraints.

Because ParaDime models are neural networks, we can apply any existing explanation techniques to them. In this example, we want to verify that the importance of our specified attribute (feature eight, *Hillshade (noon)* for the resulting x value was actually increased. To this end, we applied a "vanilla" version of integrated gradients (Molnar, 2022) to our model. The resulting feature importance score are shown in the bar chart in Figure 7.5. Note that feature eight is indeed the most important for the x result by some margin, and that it does not contribute to y at all for the strongly attribute-guided embedding.

Attribute-guided embeddings are not only a showcase of how easily new techniques can be constructed with ParaDime. They might be useful in cases where users want to transition from purely unsupervised embeddings to ones were a specified attribute is of particular interest in the analysis.

7.4.4 *Shaping Latent Spaces through Embeddings*

We previously introduced Projective Latent Interventions (PLIs) (Chapter 6; Hinterreiter et al., 2020) as a way to influence the latent space of classification models through modifications in parametric embeddings. In PLIs, users are presented with a scatterplot of the low-dimensionally embedded activations of a classification network. They can then move points and/or clusters in the embedding to make the visualization of the latent space better match their idea of how the classifier should "see" the data. For example, users might separate class clusters from each other or make them more compact. The classifier is then retrained in such a way, that the embedded activations preserve these modifications. The backpropagation of the user-defined constraints was achieved through parametric embeddings. We showed that this technique can be used to improve the generalization of the classifier and we argued that it can also lead to a better understanding of how the model's latent representation relates to the user's mental model (Hinterreiter et al., 2020). In this section, we briefly show how ParaDime facilitates such constrained, multi-step classification routines.

PLIs can use any neural network as the classification model. This model is augmented with a second part that embeds the high-dimensional latent representation (similar to the example in Section 7.3). Thanks to ParaDime organization of the training into phases, the PLI process can be easily reproduced. In a first training phase, only the classification part of the model is trained with a `classification` loss. In a second training phase, only the embedding part is trained with a `relation` loss (leaving the classification part untouched). Now users view the resulting embedding and specify the desired modifications. The resulting modified dataset can be used in a third training

phase inside a `position` loss, which is paired with the original classification loss. This completes the PLI workflow.

We expect that ParaDime can be used in a similar fashion to reimplement existing multi-step techniques, while improving their customizability.

7.5 DISCUSSION

In this section we discuss some of the design choices related to the structure of the ParaDime grammar and its implementation. We also outline how ParaDime can be used for non-parametric embeddings. Finally, we reflect on limitations and future work.

7.5.1 *Structure of the Grammar*

The structure of the ParaDime grammar cannot be uniquely derived from the necessary building-blocks (dataset, relations, etc.), but depends on a number of choices. For example, in an earlier version of the specifications, losses were defined entirely within the training phases and their specification included a weight. However, this strongly limited the reusabilty of losses across phases. We thus opted for loss specifications at the base level, which required the introduction of the **`components`** and **`weights`** entry, and the use of loss names that could be referenced. Furthermore, we initially planned different base-level entries for lists of global and batch-wise relations. From a computational view, they are typically used at different times in the routines, and only the batch-wise relations must be differentiable. Nevertheless, we finally chose a flat list of relations with individual `level` entries to highlight the conceptual similarities between the global and batch-wise relations.

7.5.2 *Implementation Choices*

We considered PyTorch (Paszke et al., 2019), TensorFlow (Abadi et al., 2016), and JAXX (Frostig et al., 2018) as machine learning frameworks for ParaDime. Ultimately, we settled on PyTorch, because it has become the most popular framework for research purposes (O'Connor, 2021).

While we used YAML (döt Net et al., 2021) for the specifications in this work due to its focus on readability, ParaDime is also capable of parsing JSON specifications with the same structure. In addition to the specifications, which facilitate sharing and reproducibility, ParaDime also allows an object-oriented construction of routines. This way of specifying routines is particularly suitable for adapting existing routines or changing properties of routines dynamically.

7.5.3 *Non-parametric Routines*

We realize that—except for the `model` entry—each ParaDime specification could also be valid for a non-parametric DR routine. Still, each ParaDime routine currently requires a PyTorch module as **`model`**, because parametric routines were our main focus. The losses access the model's methods to compute the processed data batches. With some modifications, however, it

is possible to use a simple tensor of low-dimensional coordinates instead of the model. In this case, the coordinates are directly optimized in a non-parametric way. As shown in Section 7.3.1, this can lead to lower losses, but new points cannot be added to the resulting embeddings afterwards. We plan to improve this functionality in future work.

7.5.4 *Limitations & Future Work*

One major limitation when moving from traditional DR techniques to parametric embeddings is the increased number of hyperparameters. Users must select a suitable model architecture and set batch sizes, optimizers and learning rates in a way that the loss is properly minimized. For the predefined ParaDime routines, we provide defaults based on our own experiments. With new routines, however, finding suitable choices for hyperparameters can be challenging. The same is true for weights in compound losses. Different losses can differ by orders of magnitude, and weighing them equally may lead to vanishing effects of some of the losses. As a result, non-obvious weight ratios have to be tried out, as seen in some of the examples discussed in Section 7.4. However, due to ParaDime's focus on reusability and ease of specification, experiments with different weights should be straightforward. We also supply plotting utilities with ParaDime, so that users can rapidly check the embeddings with little to no boilerplate code.

A second limitation related to batch-wise training is that certain global constraints are difficult to implement. For example, global density-based measures such as the one used in densMAP (Narayan et al., 2021) are challenging to reproduce from small batches. In principle, the batch size in ParaDime can be set to the number of items in the dataset to allow computation of global measures during training. However, this might lead to problems with gradients for other losses. We plan to experiment with such globally constrained techniques in future work to provide ways of incorporating them.

Finally, we plan to include export utilities for the trained models, so that they can readily be used elsewhere. It would be particularly desirable to export models in a format that can be used directly within browsers. This way, visualizations implemented as web-apps could make use of pretrained ParaDime routines without the need for a backend.

7.6 CONCLUSION

In this chapter we introduced ParaDime, a framework for parametric dimensionality reduction. The ParaDime grammar allows users to specify DR routines in a declarative way. We showed how this approach enables parametric extensions of existing techniques. We also illustrated how ParaDime facilitates experimentation with new ideas. We hope that through our focus on flexibility and customization, ParaDime inspires further research about the potential of parametric dimensionality reduction.

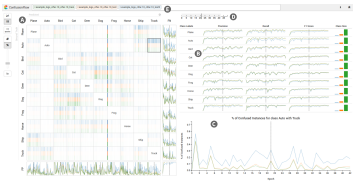
SUMMARY & OUTLOOK

8

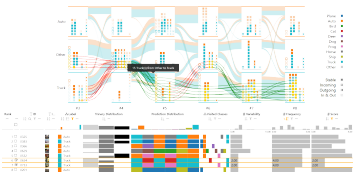
Visualization and machine learning are both driven by the goal of acquiring insights from data. However, the means to achieve this goal differ substantially between the two fields. While visualization typically focuses on the human in the loop, machine learning often prioritizes automation. With the increasing popularity of machine learning, interest has sparked in both communities to combine the complementary strengths of vis and ML. In this thesis, I have presented several works in which the two fields come together in various ways. Through this combination, these works provide users with a better understanding of high-dimensional, temporal processes.

ConfusionFlow (Chapter 3; Hinterreiter, Ruch, et al., 2022) and InstanceFlow (Chapter 4; Pühringer et al., 2020) are two visualizations that allow machine learning experts to analyze temporal information from the training process of classification models. They are both examples of the model-analysis branch of *Vis4ML*. The Projection Path Explorer (Chapter 5; Hinterreiter et al., 2021) is a more general visualization tool for high-dimensional sequences, based on unsupervised machine learning algorithms for dimensionality reduction. We applied the Projection Path Explorer to visualize similarities and differences in the convergence of neural networks during training. We also applied it to provenance data to better understand how users interact with AI-assisted visualization tools (Walchshofer et al., 2021). With Projective Latent Interventions (Chapter 6; Hinterreiter et al., 2020) we explored how low-dimensional embeddings can serve as an interface to give users control over the latent space of classification models. Finally, ParaDime (Chapter 7; Hinterreiter, Humer, et al., 2022) is a framework for neural-network-based dimensionality reduction, which allows machine learning and visualization researchers to experiment with novel embedding techniques. ParaDime is an example from the data-processing branch of *ML4Vis*.

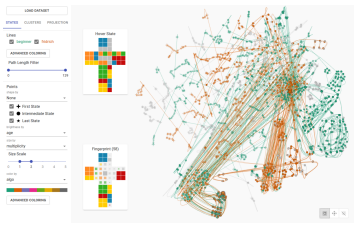
These works are part of the wide spectrum of research about the synergies between ML and vis. However, as this field is still rapidly evolving, there remain several challenges that offer potential for future work.



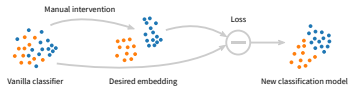
ConfusionFlow



InstanceFlow



Projection Path Explorer



Projective Latent Interventions

ParaDime

Figure 8.1: Visual summary of the contributions of this thesis.

8.1 AUTOMATION & GUIDANCE

One of the main motivations of using visualization in the context of machine learning is to improve the interpretability of complex, opaque models (Hohman et al., 2018). However, visualizations themselves can be difficult to interpret in an objective and reproducible way. This is particularly the case for non-linear dimensionality reduction, where distortions can hide interesting patterns or lead to "fake" patterns (Espadoto et al., 2019; Nonato & Aupetit, 2019). In our work, we have tackled this issue with various interaction and layout enrichment techniques that let users find and verify patterns (Eckelt et al., 2022; Hinterreiter et al., 2021). Still, this search for

interesting patterns remains a manual process that often requires a good understanding of the dataset and/or the dimensionality reduction technique used. Therefore, *guidance* might play an important role in the exploration of dimensionality-reduced data in future work.

Guidance has been defined as a "mixed-initiative process" that "comprises (1) the assistance a system gives to a user, and (2) [...] the feedback the user provides to the system in order to steer the analysis process" (Ceneda et al., 2019, p. 862). In the context of low-dimensional embeddings, this could be realized by scanning both the low- and the high-dimensional data for patterns such as clusters. A selection of these patterns could then be presented to the user, potentially ranked by conservation or distortion metrics. In the case of trajectory visualizations such as the Projection Path Explorer, temporal information can additionally be taken into account, for example to extract clusters that lie along path bundles. In this context, pattern-driven navigation with insets (Lekschas et al., 2020) might be a promising approach. In this technique, visual summaries of patterns are dynamically placed within the visualization. They adapt to the zooming level and can be used to navigate the visualization.

Most forms of guidance require algorithms for detecting patterns of interest. This may add another level of opaque data-processing to an already challenging visualization. Even when patterns can be automatically extracted from the data, it might be difficult for users to understand why these patterns have been identified and what makes them interesting. Future work will have to address the challenge of how guidance can be used with embeddings in an intuitive way that does not disrupt established analysis workflows.

8.2 THE TOOLING LANDSCAPE

Many visualization systems in the context of machine learning are developed as standalone tools (e.g., web applications). With ConfusionFlow (Hinterreiter, Ruch, et al., 2022) and InstanceFlow (Pühringer et al., 2020) we opted for browser-based applications ourselves. One advantage of deployed web apps is that they can be tried out with minimal setup. However, it can be challenging to include them efficiently in traditional machine learning workflows.

Machine learning researchers typically use Jupyter notebooks and Python scripts for developing and running their models. Analyzing these models in separate, browser-based tools disrupts this workflow. Data has to be transferred between systems, and the visualization part is often not easily accessible and/or customizable for data scientists. In addition, visualization tools may require different data formats, and it is often not clear how to move from one tool to another in a multi-step analysis while preserving the insights already gained.

A tighter integration of cutting-edge visualizations—especially within computational notebooks—could lead to a more widespread adoption of vis in ML. An integration of interactive visualization in notebooks is already possible with Altair's (VanderPlas et al., 2018) Python bindings for Vega-Lite

(Satyanarayan et al., 2017) or with other widget-based plotting packages such as bqplot (Cherukuri et al., 2022). We have recently adapted one of our standalone visualizations, an extension of Grad-CAM for image segmentation (Humer, Elharty, et al., 2022), to be usable directly within Jupyter notebooks.

However, even when visualizations are embedded in notebooks, another challenge remains with respect to reaching the whole machine learning community: the division based on different frameworks. TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019) are the two most popular frameworks for deep learning. While PyTorch has surpassed TensorFlow in terms of popularity in the research community, TensorFlow remains the industry favorite (O'Connor, 2021). A third framework, JAX (Frostig et al., 2018), has recently gained traction in the research community. When visualization and machine learning experts work together, the framework is often locked in at the beginning of a project. Since the frameworks are not compatible with each other, this means that only a subset of the community is ever reached. Future efforts are necessary to harmonize the tooling landscape and make visualizations easily accessible and customizable for more developers.

8.3 SOCIETAL IMPACT & OUTREACH

In their interrogative survey on visual analytics for deep learning, Hohman et al. (2018) identified three target audiences: model developers, model users, and non-experts. The vast majority of surveyed works was aimed at model developers. As machine learning models are used more frequently in everyday applications, *AI literacy* among laypersons becomes an increasingly important issue. Long and Magerko defined AI literacy as the "set of competencies that enables individuals to critically evaluate AI technologies; communicate and collaborate effectively with AI; and use AI as a tool online, at home, and in the workplace" (Long & Magerko, 2020, p. 2). In order to understand how visualizations can be used to boost these competencies, future collaborations will not only have to include machine learning researchers, but also experts from other fields such as psychology or human–computer interaction.

During my time as a PhD student, I was fortunate to participate in such a collaboration with colleagues from the Robopsychology Lab at the Johannes Kepler University, Linz. Together, we studied how visual explanations affect the user performance in an AI-assisted, high-risk decision making task. In several experiments, we asked participants to go on a virtual mushroom hunt with the help of a fictitious mushroom identification app. Participants were presented with images of mushrooms together with AI predictions of the mushrooms' species. They then had to decide for each mushroom whether it was edible or not and whether to pick it or leave it (i.e., participants could trust the AI or overrule it). While one group only received the AI's predictions, a second group was given additional visual explanations. The group that received the explanations outperformed the group that did not in the task of correctly identifying whether a mushroom was edible or not (Leichtmann et al., 2023). However, we also found that some effects were difficult to measure and interpret due to the many possible moderating variables, such as the AI's

Figure 8.2: Impression from the *AI Forest* installation in which visitors went on a virtual, AI-assisted mushroom hunt (Leichtmann et al., 2022). Image by vog.photo/Ars Electronica, licensed under CC BY-NC-ND 2.0.

accuracy and the complexity of visual explanations. Therefore, future studies are vital to better understand how visualization and machine learning can be combined in applications targeted at laypersons.

In addition to scientific studies, it will be important to raise awareness about potential issues related to the interpretability and application of machine learning models. We conducted a replication study of the experiment mentioned above in an environment that let us engage with participants more directly. We created an artificial forest environment at an art festival and let visitors go on a virtual mushroom hunt in this immersive environment (Leichtmann et al., 2022). Figure 8.2 gives an impression of our installation. It was highly interesting to see how people from diverse demographics interacted with the AI and the visualizations and to talk to them about the implications of our work. It also reminded me how far a lot of the research work is detached from the actual technological impacts on society. I believe it is in the responsibility of us researchers, who study and develop visualizations and machine learning algorithms, to not forget about the societal contributions of our work. Only then will the combination of visualization and machine learning reach its full potential.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv: 1603.04467 [cs]*. Retrieved June 1, 2018, from http://arxiv.org/abs/1603.04467 (cit. on pp. 18–20, 100, 105)

Abdi, H., *&* Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(4), 433–459. https://doi.org/10.1002/wics.101 (cit. on pp. 10, 85)

Agrawal, A., Ali, A., *&* Boyd, S. (2021). Minimum-distortion embedding. *Foundations and Trends in Machine Learning*, *14*(3), 211–378. https://doi.org/10.1561/2200000090 (cit. on pp. 11, 86)

Aigner, W., Miksch, S., Schumann, H., *&* Tominski, C. (2011). *Visualization of time-oriented data*. Springer. (Cit. on pp. 9, 16).

Alsallakh, B., Hanbury, A., Hauser, H., Miksch, S., *&* Rauber, A. (2014). Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 1703–1712. https://doi.org/10.1109/TVCG.2014.2346660 (cit. on pp. 20, 21)

Alsallakh, B., Jourabloo, A., Ye, M., Liu, X., *&* Ren, L. (2018). Do convolutional neural networks learn class hierarchy? *IEEE Transactions on Visualization and Computer Graphics*, *24*(1), 152–162. https://doi.org/10.1109/TVCG.2017.2744683 (cit. on pp. 19, 20, 35, 38)

Andrienko, N., *&* Andrienko, G. (2006). *Exploratory analysis of spatial and temporal data*. Springer. (Cit. on p. 9).

Andrienko, N., *&* Andrienko, G. (2013). Visual analytics of movement: An overview of methods, tools and procedures. *Information Visualization*, *12*(1), 3–24. https://doi.org/10.1177/1473871612457601 (cit. on p. 9)

Aubry, M., *&* Russell, B. C. (2015). Understanding deep features with computer-generated imagery. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2875–2883. https://doi.org/10.1109/ICCV.2015.329 (cit. on p. 65)

Bach, B., Shi, C., Heulot, N., Madhyastha, T., Grabowski, T., *&* Dragicevic, P. (2016). Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '15)*, *22*(1), 559–568 (cit. on pp. 5, 10, 47, 49–51, 56).

Balntas, V., Riba, E., Ponsa, D., *&* Mikolajczyk, K. (2016). Learning local feature descriptors with triplets and shallow convolutional neural networks. *Procedings of the British Machine Vision Conference 2016*, 119.1–119.11. https://doi.org/10.5244/C.30.119 (cit. on p. 97)

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012 (cit. on p. 37)

Baumgartner, C. F., Kamnitsas, K., Matthew, J., Fletcher, T. P., Smith, S., Koch, L. M., Kainz, B., & Rueckert, D. (2017). SonoNet: Real-time detection and localisation of fetal standard scan planes in freehand ultrasound. *IEEE Transactions on Medical Imaging*, *36*(11), 2204–2215. https://doi.org/10.1109/TMI.2017.2712367 (cit. on pp. 80, 81)

Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I. W. H., Ng, L. G., Ginhoux, F., & Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology*, *37*(1), 38–44. https://doi.org/10.1038/nbt.4314 (cit. on p. 11)

Bellet, A., Habrard, A., & Sebban, M. (2013). A survey on metric learning for feature vectors and structured data. *arXiv: 1306.6709 [cs.LG]*. https://doi.org/10.48550/ARXIV.1306.6709 (cit. on pp. 11, 78)

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828. https://doi.org/10.1109/TPAMI.2013.50 (cit. on p. 11)

Bengio, Y., Delalleau, O., Roux, N. L., Paiement, J.-F., Vincent, P., & Ouimet, M. (2004). Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, *16*(10), 2197–2219. https://doi.org/10.1162/0899766041732396 (cit. on p. 11)

Bernard, J., Hutter, M., Zeppelzauer, M., Fellner, D., & Sedlmair, M. (2018). Comparing visual-interactive labeling with active learning: An experimental study. *IEEE Transactions on Visualization and Computer Graphics*, *24*(1), 298–308. https://doi.org/10.1109/TVCG.2017.2744818 (cit. on p. 27)

Bernard, J. (2015). *Exploratory search in time-oriented primary data* (Doctoral dissertation). Technische Universität Darmstadt. https://tuprints.ulb.tu-darmstadt.de/5173/. (Cit. on p. 9)

Bernard, J., Hutter, M., Reinemuth, H., Pfeifer, H., Bors, C., & Kohlhammer, J. (2019). Visual-interactive preprocessing of multivariate time series data. *Computer Graphics Forum*, *38*(3), 401–412. https://doi.org/10.1111/cgf.13698 (cit. on p. 22)

Bernard, J., Wilhelm, N., & Scherer, M. (2012). TimeSeriesPaths: Projection-based explorative analysis of multivariate time series data. *Journal of WSCG*, *20*(2), 97–106 (cit. on p. 49).

Bernard, J., Zeppelzauer, M., Lehmann, M., Müller, M., & Sedlmair, M. (2018). Towards user-centered active learning algorithms. *Computer Graphics Forum*, *37*(3), 121–132. https://doi.org/10.1111/cgf.13406 (cit. on p. 27)

Bernard, J., Zeppelzauer, M., Sedlmair, M., & Aigner, W. (2018). VIAL: A unified process for visual interactive labeling. *The Visual Computer*,

34(9), 1189–1207. https://doi.org/10.1007/s00371-018-1500-3 (cit. on p. 27)

Blackard, J. A., *&* Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, *24*(3), 131–151. https://doi.org/10.1016/S0168-1699(99)00046-0 (cit. on p. 96)

Blázquez-García, A., Conde, A., Mori, U., *&* Lozano, J. A. (2022). A review on outlier/anomaly detection in time series data. *ACM Computing Surveys*, *54*(3), 1–33. https://doi.org/10.1145/3444690 (cit. on p. 9)

Böhm, J. N., Berens, P., *&* Kobak, D. (2022). Attraction-repulsion spectrum in neighbor embeddings. *Journal of Machine Learning Research*, *23*, 1–32. https://jmlr.org/papers/v23/21-0055.html (cit. on pp. 11, 85, 86)

Bostock, M., Ogievetsky, V., *&* Heer, J. (2011). D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '11)*, *17*(12), 2301–2309. https://doi.org/10.1109/TVCG.2011.185 (cit. on p. 26)

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier *&* G. Saporta (Eds.), *Proceedings of COMPSTAT'2010* (pp. 177–186). Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-2604-3_16. (Cit. on p. 90)

Boz, H. A. (2019). *A visual multivariate dynamic EgoCentric network exploration tool* (Master's Thesis). Sabanci University. Istanbul, Turkey. (Cit. on p. 49).

Brent, R. P. (2002). *Algorithms for minimization without derivatives*. Dover Publications. (Cit. on p. 79).

Brown, E. T., Ottley, A., Zhao, H., Lin, Q., Souvenir, R., Endert, A., *&* Chang, R. (2014). Finding waldo: Learning about users from their interactions. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 1663–1672. https://doi.org/10.1109/TVCG.2014.2346575 (cit. on p. 69)

Brown, E. T., Yarlagadda, S., Cook, K. A., Endert, A., *&* Chang, R. (2018). ModelSpace: Visualizing the trails of data models in visual analytics systems. *MLUI 2018: Machine Learning from User Interactions for Visualization and Analytics*, 11 (cit. on p. 49).

Bruckner, D. (2014). *ML-o-scope: A diagnostic visualization system for deep machine learning pipelines*. Defense Technical Information Center. Fort Belvoir, VA. https://doi.org/10.21236/ADA605112. (Cit. on pp. 19, 20, 38)

Cakmak, E., Seebacher, D., Buchmuller, J., *&* Keim, D. A. (2018). Time series projection to highlight trends and outliers. *2018 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 104–105. https://doi.org/10.1109/VAST.2018.8802502 (cit. on p. 49)

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*(1), 41–75. https://doi.org/10.1023/A:1007379606734 (cit. on p. 95)

Ceneda, D., Gschwandtner, T., *&* Miksch, S. (2019). A review of guidance approaches in visual data analysis: A multifocal perspective. *Computer*

Graphics Forum, *38*(3), 861–879. https://doi.org/10.1111/cgf.13730 (cit. on p. 104)

Chae, J., Gao, S., Ramanthan, A., Steed, C., & Tourassi, G. D. (2017). Visualization for classi cation in deep neural networks. *Workshop on Visual Analytics for Deep Learning at IEEE VIS*, 6 (cit. on pp. 19, 20, 38).

Chatzimparmpas, A., Martins, R. M., Jusufi, I., & Kerren, A. (2020). A survey of surveys on the use of visualization for interpreting machine learning models. *Information Visualization*, *19*(3), 207–233. https://doi.org/10.1177/1473871620904671 (cit. on pp. 2, 17, 37)

Chechik, G., Sharma, V., Shalit, U., & Bengio, S. (2010). Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, *11*(36), 1109–1135. https://www.jmlr.org/papers/v11/chechik10a.html (cit. on pp. 90, 96–98)

Chen, X., Weng, J., Lu, W., Xu, J., & Weng, J. (2018). Deep manifold learning combined with convolutional neural networks for action recognition. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(9), 3938–3952. https://doi.org/10.1109/TNNLS.2017.2740318 (cit. on p. 78)

Cherukuri, C., Dafna, I., Grout, J., Mabille, J., Young, K., Breddels, M., Renou, M., & Corlay, S. (2022). bqplot – plotting library for ipython/jupyter notebooks. Retrieved September 26, 2022, from https://github.com/bqplot/bqplot. (Cit. on p. 105)

Chinchalkar, S. (1996). An upper bound for the number of reachable positions. *ICGA Journal*, *19*(3), 181–183. https://doi.org/10.3233/ICG-1996-19305 (cit. on p. 63)

Chung, S., Suh, S., & Park, C. (2016). ReVACNN: Real-time visual analytics for convolutional neural network. *ACM SIGKDD Workshop on Interactive Data Exploration and Analytics (IDEA)*, 7 (cit. on pp. 19, 20, 38).

Cox, M. A. A., & Cox, T. F. (2008). Multidimensional scaling. In *Handbook of data visualization* (pp. 315–347). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-33037-0_14. (Cit. on p. 10)

Cunningham, J. P., & Ghahramani, Z. (2015). Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, *16*(1), 2859–2900 (cit. on pp. 10, 91).

Cutler, Z. T., Gadhave, K., & Lex, A. (2020). Trrack: A library for provenance tracking in web-based visualizations. *IEEE Visualization Conference (VIS)*, 116–120. https://doi.org/10.1109/VIS47514.2020.00030 (cit. on p. 70)

Damrich, S., Böhm, J. N., Hamprecht, F. A., & Kobak, D. (2022). Contrastive learning unifies t-SNE and UMAP. https://doi.org/10.48550/ARXIV.2206.01816 (cit. on p. 11)

Damrich, S., & Hamprecht, F. A. (2021). On UMAP's true loss function. *Advances in Neural Information Processing Systems*, *34*, 5798–5809 (cit. on p. 11).

de Bodt, C., Mulders, D., Verleysen, M., & Lee, J. A. (2019). Nonlinear dimensionality reduction with missing data using parametric multiple imputations. *IEEE Transactions on Neural Networks and Learning Sys-*

tems, *30*(4), 1166–1179. https://doi.org/10.1109/TNNLS.2018.2861891 (cit. on p. 11)

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848 (cit. on p. 29)

Deniz. (2018). Pgn2gif. Retrieved July 11, 2019, from https://github.com/dn1z/pgn2gif. (Cit. on p. 63)

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2014). DeCAF: A deep convolutional activation feature for generic visual recognition. *Proceedings of Machine Learning Research*, *32*, 647–655. Retrieved November 12, 2019, from http://arxiv.org/abs/1310.1531 (cit. on p. 65)

Dong, W., Moses, C., & Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th international conference on World wide web - WWW '11*, 577. https://doi.org/10.1145/1963405.1963487 (cit. on p. 79)

döt Net, I., Müller, T., Pantelis, A., Aro, E., Smith, T., Ben-Kiki, O., & Evans, C. C. (2021). *YAML ain't markup language (YAML™) version 1.2* (Revision 1.2.2). YAML Language Development Team. https://yaml.org/spec/1.2.2. (Cit. on pp. 87, 100)

Dou, W., Jeong, D. H., Stukes, F., Ribarsky, W., Lipford, H. R., & Chang, R. (2009). Recovering reasoning process from user interactions. *IEEE Computer Graphics & Applications*, *29*(3), 52–61 (cit. on p. 69).

Du, F., Cao, N., Zhao, J., & Lin, Y.-R. (2015). Trajectory bundling for animated transitions. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 289–298 (cit. on p. 9).

Eckelt, K., Hinterreiter, A., Adelberger, P., Walchshofer, C., Dhanoa, V., Humer, C., Heckmann, M., Steinparz, C. A., & Streit, M. (2022). Visual exploration of relationships and structure in low-dimensional embeddings. *IEEE Transactions on Visualization and Computer Graphics (Early Access)*. https://doi.org/10.1109/TVCG.2022.3156760 (cit. on pp. 7, 8, 74, 75, 103)

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, *32*(2). https://doi.org/10.1214/009053604000000067 (cit. on p. 91)

Endert, A., Ribarsky, W., Turkay, C., Wong, B. W., Nabney, I., Blanco, I. D., & Rossi, F. (2017). The state of the art in integrating machine learning into visual analytics: Integrating machine learning into visual analytics. *Computer Graphics Forum*. https://doi.org/10.1111/cgf.13092 (cit. on pp. 2, 3)

Engel, D., Hüttenberger, L., & Hamann, B. (2012). A survey of dimension reduction methods for high-dimensional data analysis and visualization. *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering – Proceedings of IRTG 1131 Workshop 2011*, *27*, 135–149. https://doi.org/10.4230/oasics.vluds.2011.135 (cit. on p. 10)

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, *11*, 625–660 (cit. on pp. 66, 69, 78).

Ersoy, O., Hurter, C., Paulovich, F., Cantareiro, G., & Telea, A. (2011). Skeleton-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2364–2373. https://doi.org/10.1109/TVCG.2011.233 (cit. on p. 9)

Espadoto, M., Martins, R. M., Kerren, A., Hirata, N. S. T., & Telea, A. C. (2019). Towards a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics*, 1–1. https://doi.org/10.1109/TVCG.2019.2944182 (cit. on pp. 10, 53, 91, 95, 103)

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, *17*(3), 37. https://doi.org/10.1609/aimag.v17i3.1230 (cit. on p. 9)

Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, *30*(1), 27–38. https://doi.org/10.1016/j.patrec.2008.08.010 (cit. on p. 37)

Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv: 1803.03635 [cs]*. Retrieved March 18, 2019, from http://arxiv.org/abs/1803.03635 (cit. on p. 32)

Fridrich, J. (1997). My system for solving Rubik's cube. Retrieved February 25, 2019, from http://www.ws.binghamton.edu/fridrich/system.html. (Cit. on p. 58)

Friedel, F. (2017). Fritz 16 – freund und trainer. Retrieved November 4, 2019, from https://de.chessbase.com/post/fritz-16-freund-und-trainer. (Cit. on p. 62)

Friendly, M. (2008). A brief history of data visualization. In C.-h. Chen, W. Härdle, & A. Unwin (Eds.), *Handbook of data visualization* (pp. 15–56). Springer. (Cit. on p. 1).

Frostig, R., Johnson, M. J., & Leary, C. (2018). Compiling machine learning programs via high-level tracing. *SysML Conference*, 3. https://mlsys.org/Conferences/2019/doc/2018/146.pdf (cit. on pp. 100, 105)

Fu, T.-c. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence*, *24*(1), 164–181. https://doi.org/10.1016/j.engappai.2010.09.007 (cit. on p. 9)

Furmanova, K., Gratzl, S., Stitz, H., Zichner, T., Jaresova, M., Lex, A., & Streit, M. (2020). Taggle: Combining overview and details in tabular data visualizations. *Information Visualization*, *19*(2), 114–136. https://doi.org/10.1177/1473871619878085 (cit. on p. 39)

Gadhave, K., Görtler, J., Cutler, Z., Nobre, C., Deussen, O., Meyer, M., Phillips, J. M., & Lex, A. (2021). Predicting intent behind selections in scatterplot visualizations. *Information Visualization*, *20*(4), 207–228. https://doi.org/10.1177/14738716211038604 (cit. on pp. 70, 71)

Gilday, D. (2019). How to build MindCub3r for Lego Mindstroms EV3. Retrieved November 28, 2019, from https://www.mindcuber.com/mindcub3r/mindcub3r.html. (Cit. on p. 133)

Gisbrecht, A., Mokbel, B., & Hammer, B. (2012). Linear basis-function t-SNE for fast nonlinear dimensionality reduction. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN.2012.6252809 (cit. on p. 11)

Gisbrecht, A., Schulz, A., & Hammer, B. (2015). Parametric nonlinear dimensionality reduction using kernel t-SNE. *Neurocomputing*, *147*, 71–82. https://doi.org/10.1016/j.neucom.2013.11.045 (cit. on pp. 11, 133)

Gleicher, M., Albers, D., Walker, R., Jusufi, I., Hansen, C. D., & Roberts, J. C. (2011). Visual comparison for information visualization. *Information Visualization*, *10*(4), 289–309. https://doi.org/10.1177/1473871611416549 (cit. on pp. 18, 23)

Glorot, X., Bordes, A., & Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. *Proceedings of the 28th International Conference on Machine Learning*, 8 (cit. on p. 13).

Gogolou, A., Tsandilas, T., Palpanas, T., & Bezerianos, A. (2019). Comparing similarity perception in time series visualizations. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '18)*, *25*(1), 523–533. https://doi.org/10.1109/TVCG.2018.2865077 (cit. on pp. 10, 22)

Görtler, J., Hohman, F., Moritz, D., Wongsuphasawat, K., Ren, D., Nair, R., Kirchner, M., & Patel, K. (2022). Neo: Generalizing confusion matrix visualization to hierarchical and multi-output labels. *CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3491102.3501823 (cit. on p. 87)

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, *27*(4), 857. https://doi.org/10.2307/2528823 (cit. on p. 70)

Gratzl, S., Lex, A., Gehlenborg, N., Pfister, H., & Streit, M. (2013). LineUp: Visual analysis of multi-attribute rankings. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '13)*, *19*(12), 2277–2286. https://doi.org/10.1109/TVCG.2013.173 (cit. on pp. 39, 41)

Griffin, G., Holub, A., & Perona, P. (2007). *Caltech-256 object category dataset* (No. 7694). Caltech. http://authors.library.caltech.edu/7694. (Cit. on p. 29)

Hamel, P., & Eck, D. (2010). Learning features from music audio with deep belief networks. *Proceedings of the 11th International Society for Music Information Retrieval Conference*, 339–344 (cit. on p. 65).

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, *28*. https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html (cit. on p. 32)

Havard, P. (2019). Kingbase – a free chess games database, updated monthly. Retrieved July 11, 2019, from https://www.kingbase-chess.net/. (Cit. on p. 63)

He, J., & Chen, C. (2017). Visualizing temporal patterns in representation data. *VADL 2017: Workshop on Visual Analytics for Deep Learning*, 4 (cit. on p. 49).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90 (cit. on p. 13)

Henry. (2017). How is the distance of two random points in a unit hypercube distributed? [https://math.stackexchange.com/q/1985698]. (Cit. on p. 54).

Hinterreiter, A., Humer, C., Kainz, B., & Streit, M. (2022). ParaDime: A framework for parametric dimensionality reduction. *arXiv: 2210.04582*. https://doi.org/10.48550/arXiv.2210.04582. To be submitted to *EuroVis '23* (cit. on pp. 6, 8, 103)

Hinterreiter, A., Ruch, P., Stitz, P., Ennemoser, M., Bernard, J., Strobelt, H., & Streit, M. (2022). ConfusionFlow: A model-agnostic visualization for temporal analysis of classifier confusion. *IEEE Transactions on Visualization and Computer Graphics*, *28*(2), 1222–1236. https://doi.org/10.1109/TVCG.2020.3012063 (cit. on pp. 3, 4, 8, 37, 38, 45, 67, 103, 104)

Hinterreiter, A., Steinparz, C. A., Schöfl, M., Stitz, H., & Streit, M. (2021). Projection Path Explorer: Exploring visual patterns in projected decision-making paths. *ACM Transactions on Interactive Intelligent Systems*, *11*(3–4), Article 22. https://doi.org/10.1145/3387165 (cit. on pp. 5, 8, 103)

Hinterreiter, A., Streit, M., & Kainz, B. (2020). Projective Latent Interventions for understanding and fine-tuning classifiers. In J. Cardoso et al. (Eds.), *Interpretable and annotation-efficient learning for medical image computing. Proceedings of the 3rd workshop on interpretability of machine intelligence in medical image computing (iMIMIC 2020)* (pp. 13–22). Springer. https://doi.org/10.1007/978-3-030-61166-8_2. Best Paper Award at iMIMIC 2020. (Cit. on pp. 5, 99, 103)

Hinton, G. E. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647 (cit. on p. 11)

Hinton, G. E., & Roweis, S. T. (2002). Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*, *15*, 8 (cit. on pp. 10, 11).

Hinton, G. E., & Zemel, R. S. (1993). Autoencoders, minimum description length and helmholtz free energy. *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS '93)*, 3–10. https://doi.org/10.5555/2987189.2987190 (cit. on p. 11)

Hohman, F., Hodas, N., & Chau, D. H. (2017). ShapeShop: Towards understanding deep learning representations via interactive experimentation. *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, 1694–1699. https://doi.org/10.1145/3027063.3053103 (cit. on p. 18)

Hohman, F., Kahng, M., Pienta, R., & Chau, D. H. (2018). Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics*, *25*(8), 2674–2693. https://doi.org/10.1109/TVCG.2018.2843369 (cit. on pp. 1, 2, 37, 65, 103, 105)

Holten, D., & Wijk, J. J. v. (2009). Force-directed edge bundling for graph visualization. *Computer Graphics Forum (EuroVis '09)*, *28*(3), 983–990. https://doi.org/10.1111/j.1467-8659.2009.01450.x (cit. on p. 9)

Humer, C., Elharty, M., Hinterreiter, A., & Streit, M. (2022). Interactive attribution-based explanations for image segmentation. In M. Krone, S. Lenti, & J. Schmidt (Eds.), *EuroVis 2022 – Posters* (3 pages). The Eurographics Association. https://doi.org/10.2312/EVP.20221130. (Cit. on p. 105)

Humer, C., Heberle, H., Montanari, F., Wolf, T., Huber, F., Henderson, R., Heinrich, J., & Streit, M. (2022). ChemInformatics model explorer (CIME): Exploratory analysis of chemical model explanations. *Journal of Cheminformatics*, *14*(1), 21. https://doi.org/10.1186/s13321-022-00600-z (cit. on p. 75)

Humer, C., Hinterreiter, A., Leichtmann, B., Mara, M., & Streit, M. (2022). Comparing effects of attribution-based, example-based, and feature-based explanation methods on AI-assisted decision-making. *OSF Preprint*. https://doi.org/10.31219/osf.io/h6dwz. Submitted to the *27th Annual Conference on Intelligent User Interfaces (IUI '23)* (cit. on p. 7)

Hurter, C., Ersoy, O., Fabrikant, S. I., Klein, T. R., & Telea, A. C. (2013). Bundled visualization of dynamic graph and trail data. *IEEE Transactions on Visualization and Computer Graphics*, *20*(8), 1141–1157. https://doi.org/10.1109/TVCG.2013.246 (cit. on p. 9)

Kahng, M., Andrews, P. Y., Kalro, A., & Chau, D. H. (2017). ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, *24*, 88–97. https://doi.org/10.1109/TVCG.2017.2744718 (cit. on p. 38)

Kahng, M., Thorat, N., Chau, D. H. P., Viegas, F. B., & Wattenberg, M. (2019). GAN lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 1–11. https://doi.org/10.1109/TVCG.2018.2864500 (cit. on p. 19)

Kapoor, A., Lee, B., Tan, D., & Horvitz, E. (2010). Interactive optimization for steering machine classification. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, 1343. https://doi.org/10.1145/1753326.1753529 (cit. on p. 20)

Keim, D. A., Bak, P., Bertini, E., Oelke, D., Spretke, D., & Ziegler, H. (2010). Advanced visual analytics interfaces. *Proceedings of the International Conference on Advanced Visual Interfaces - AVI '10*, 3. https://doi.org/10.1145/1842993.1842995 (cit. on p. 3)

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization. *arXiv: 1412.6980 [cs]*. Retrieved March 30, 2022, from http://arxiv.org/abs/1412.6980 (cit. on pp. 90, 91)

Kobak, D., & Linderman, G. C. (2021). Initialization is critical for preserving global data structure in both t-SNE and UMAP. *Nature Biotechnology*, *39*(2), 156–157. https://doi.org/10.1038/s41587-020-00809-z (cit. on p. 11)

Kondo, B., & Collins, C. (2014). DimpVis: Exploring time-varying information visualizations by direct manipulation. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '14)*, *20*(12), 2003–2012. https://doi.org/10.1109/TVCG.2014.2346250 (cit. on p. 9)

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (Vol. 1, No. 4). University of Toronto. (Cit. on pp. 21, 22, 26, 29, 43, 80).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386 (cit. on p. 13)

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, *29*(1), 1–27. https://doi.org/10.1007/BF02289565 (cit. on p. 91)

Kulis, B. (2012). Metric learning: A survey. *Foundations and Trends in Machine Learning*, *5*(4), 287–364 (cit. on pp. 11, 78).

LeCun, Y. (2005). The MNIST database of handwritten digits. Retrieved September 20, 2022, from http://yann.lecun.com/exdb/mnist/. (Cit. on pp. 67, 80, 85, 92, 94, 95)

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539 (cit. on p. 65)

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791 (cit. on p. 28)

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, *38*, 562–570. http://proceedings.mlr.press/v38/lee15a.html (cit. on p. 78)

Leichtmann, B., Hinterreiter, A., Humer, C., Streit, M., & Mara, M. (2022). Explainable artificial intelligence improves human decision-making: Results from a mushroom picking experiment at a public art festival. *OSF Preprint*. https://doi.org/10.31219/osf.io/68emr. To be submitted to the *International Journal of Human–Computer Interaction* (cit. on pp. 7, 8, 106)

Leichtmann, B., Humer, C., Hinterreiter, A., Streit, M., & Mara, M. (2023). Effects of explainable artificial intelligence on trust and human behavior in a high-risk decision task. *Computers in Human Behavior*, *139*, 107539. https://doi.org/10.1016/j.chb.2022.107539 (cit. on pp. 7, 8, 105)

Lekschas, F., Behrisch, M., Bach, B., Kerpedjiev, P., Gehlenborg, N., & Pfister, H. (2020). Pattern-driven navigation in 2d multiscale visualizations

with scalable insets. *IEEE Transactions on Visualization and Computer Graphics*, *26*(1), 611–621. https://doi.org/10.1109/TVCG.2019.2934555 (cit. on p. 104)

Lex, A., Streit, M., Schulz, H.-J., Partl, C., Schmalstieg, D., Park, P., & Gehlenborg, N. (2012). StratomeX: Visual analysis of large-scale heterogeneous genomics data for cancer subtype characterization. *Computer Graphics Forum*, *31*(3), 1175–1184. https://doi.org/10.1111/j.1467-8659.2012.03110.x (cit. on p. 39)

Lex, A., Schulz, H.-J., Streit, M., Partl, C., & Schmalstieg, D. (2011). VisBricks: Multiform visualization of large, inhomogeneous data. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '11)*, *17*(12), 2291–2300. https://doi.org/10.1109/TVCG.2011.250 (cit. on p. 39)

Li Fei-Fei, Fergus, R., & Perona, P. (2004). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *2004 Conference on Computer Vision and Pattern Recognition Workshop*, 178–178. https://doi.org/10.1109/CVPR.2004.383 (cit. on p. 29)

Liberacki, L., & Brannan, T. (2015). Rubik's cube solver coded in python. Retrieved February 28, 2019, from https://github.com/CubeLuke/Rubiks-Cube-Solver. (Cit. on p. 58)

Liu, D., Cui, W., Jin, K., Guo, Y., & Qu, H. (2019). DeepTracker: Visualizing the training process of convolutional neural networks. *ACM Transactions on Intelligent Systems and Technology*, *10*, Article 6. https://doi.org/10.1145/3200489 (cit. on pp. 19, 20, 38)

Liu, S., Maljovec, D., Wang, B., Bremer, P. T., & Pascucci, V. (2017). Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, *23*(3), 1249–1268. https://doi.org/10.1109/TVCG.2016.2640960 (cit. on p. 3)

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *arXiv: 1810.05270 [cs, stat]*. Retrieved March 18, 2019, from http://arxiv.org/abs/1810.05270 (cit. on p. 32)

Long, D., & Magerko, B. (2020). What is AI literacy? competencies and design considerations. In R. Bernhaupt (Ed.), *Proceedings of the 2020 CHI conference on human factors in computing systems* (pp. 1–16). Association for Computing Machinery. https://doi.org/10.1145/3313831.3376727. (Cit. on p. 105)

Lu, W.-L., Wang, Y.-S., & Lin, W.-C. (2014). Chess evolution visualization. *IEEE Transactions on Visualization and Computer Graphics*, *20*(5), 702–713. https://doi.org/10.1109/TVCG.2014.2299803 (cit. on p. 62)

L'Yi, S., Wang, Q., Lekschas, F., & Gehlenborg, N. (2022). Gosling: A grammar-based toolkit for scalable and interactive genomics data visualization. *IEEE Transactions on Visualization and Computer Graphics*, *28*(1), 140–150. https://doi.org/10.1109/TVCG.2021.3114876 (cit. on p. 87)

Mao, Y., Dillon, J., & Lebanon, G. (2007). Sequential document visualization. *IEEE Transactions on Visualization and Computer Graphics*, *13*(6), 1208–1215. https://doi.org/10.1109/TVCG.2007.70592 (cit. on p. 49)

Mayr, A., Klambauer, G., Unterthiner, T., Steijaert, M., K. Wegner, J., Ceulemans, H., Clevert, D.-A., & Hochreiter, S. (2018). Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chemical Science*, *9*(24), 5441–5451. https://doi.org/10.1039/C8SC00148K (cit. on p. 13)

McInnes, L., Healy, J., & Melville, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv: 1802.03426 [cs, stat]* (cit. on pp. 11, 57, 77, 85, 90, 96).

McLachlan, P., Munzner, T., Koutsofios, E., & North, S. (2008). LiveRAC: Interactive visual exploration of system management time-series data. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, 1483–1492 (cit. on p. 34).

Mead, A. (1992). Review of the development of multidimensional scaling methods. *The Statistician*, *41*(1), 27. https://doi.org/10.2307/2348634 (cit. on p. 77)

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, *26*. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html (cit. on p. 90)

Miksch, S., & Aigner, W. (2014). A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data. *Computers & Graphics*, *38*, 286–290. https://doi.org/10.1016/j.cag.2013.11.002 (cit. on p. 9)

Min, M. R., van der Maaten, L., Yuan, Z., Bonner, A. J., & Zhang, Z. (2010). Deep supervised t-distributed embedding. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. https://icml.cc/Conferences/2010/papers/149.pdf (cit. on pp. 11, 77, 78)

Ming, Y., Cao, S., Zhang, R., Li, Z., Chen, Y., Song, Y., & Qu, H. (2017). Understanding hidden memories of recurrent neural networks. *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 13–24. https://doi.org/10.1109/VAST.2017.8585721 (cit. on pp. 18, 20)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236 (cit. on p. 65)

Mohamed, A.-r., Hinton, G., & Penn, G. (2012). Understanding how deep belief networks perform acoustic modelling. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4273–4276. https://doi.org/10.1109/ICASSP.2012.6288863 (cit. on p. 65)

Molnar, C. (2022). *Interpretable machine learning: A guide for making black box models explainable* (2nd ed.). christophm.github.io/interpretable-ml-book/. (Cit. on p. 99)

Munzner, T. (2014). *Visualization analysis and design*. CRC Press, Taylor & Francis Group. (Cit. on p. 1).

Narayan, A., Berger, B., & Cho, H. (2021). Assessing single-cell transcriptomic variability through density-preserving data visualization. *Nature Biotechnology*, *39*(6), 765–774. https://doi.org/10.1038/s41587-020-00801-7 (cit. on p. 101)

Ngo, Q. Q., Dennig, F. L., Keim, D. A., & Sedlmair, M. (2022). Machine learning meets visualization – experiences and lessons learned. *it - Information Technology*, *64*(4), 169–180. https://doi.org/10.1515/itit-2022-0034 (cit. on pp. 1, 2)

Nguyen, P. H., Xu, K., Wheat, A., Wong, B. L. W., Attfield, S., & Fields, B. (2016). SensePath: Understanding the sensemaking process through analytic provenance. *IEEE Transactions on Visualization and Computer Graphics*, *22*(1), 41–50. https://doi.org/10.1109/TVCG.2015.2467611 (cit. on p. 69)

NHS. (2015). *Fetal anomaly screening programme: Programme handbook june 2015*. Public Health England. Retrieved September 25, 2022, from https://www.gov.uk/government/publications/fetal-anomaly-screening-programme-handbook. (Cit. on p. 80)

Nogueira dos Santos, C., & Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. *Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014)*, 69–78 (cit. on p. 13).

Nonato, L. G., & Aupetit, M. (2019). Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE Transactions on Visualization and Computer Graphics*, *25*(8), 2650–2673. https://doi.org/10.1109/TVCG.2018.2846735 (cit. on pp. 74, 103)

O'Connor, R. (2021). PyTorch vs TensorFlow in 2022. Retrieved September 29, 2022, from https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/. (Cit. on pp. 100, 105)

Oskolkov, N. (2019). How exactly UMAP works. Retrieved April 20, 2020, from https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668. (Cit. on p. 72)

Park, D., Drucker, S. M., Fernandez, R., & Elmqvist, N. (2018). Atom: A grammar for unit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, *24*(12), 3032–3043. https://doi.org/10.1109/TVCG.2017.2785807 (cit. on p. 87)

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch. *NIPS 2017 Autodiff Workshop*, 4 (cit. on p. 86).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*, 12 (cit. on pp. 86, 90, 100, 105).

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine*

and Journal of Science, *2*(11), 559–572. https://doi.org/10.1080/14786440109462720 (cit. on p. 10)

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830 (cit. on pp. 57, 91).

Petrus, L. (1997). Solving rubik's cube for speed. Retrieved July 21, 2020, from https://lar5.com/cube/:%202019-11-20. (Cit. on p. 58)

Peysakhovich, V., Hurter, C., & Telea, A. (2015). Attribute-driven edge bundling for general graphs with applications in trail analysis. *2015 IEEE Pacific Visualization Symposium (PacificVis)*, 39–46. https://doi.org/10.1109/PACIFICVIS.2015.7156354 (cit. on p. 9)

Pezzotti, N., Hollt, T., Van Gemert, J., Lelieveldt, B. P., Eisemann, E., & Vilanova, A. (2018). DeepEyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, *24*(1), 98–108. https://doi.org/10.1109/TVCG.2017.2744358 (cit. on pp. 19, 20, 38)

Pike, W. A., Stasko, J., Chang, R., & O'Connell, T. A. (2009). The science of interaction. *Information Visualization*, *8*(4), 263–274. https://doi.org/10.1057/ivs.2009.22 (cit. on p. 3)

Pirolli, P., & Card, S. (2005). The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. *Proceedings of Conference on Intelligence Analysis*. https://analysis.mitre.org/proceedings/Final_Papers_Files/206_Camera_Ready_Paper.pdf (cit. on p. 3)

Pohl, M., Smuc, M., & Mayr, E. (2012). The user puzzle - explaining the interaction with visual analytics systems. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2908–2916. https://doi.org/10.1109/TVCG.2012.273 (cit. on p. 69)

Poličar, P. G., Stražar, M., & Zupan, B. (2019). openTSNE: A modular python library for t-SNE dimensionality reduction and embedding. *bioRxiv*. https://doi.org/10.1101/731877 (cit. on pp. 57, 72, 79, 92)

Pühringer, M., Hinterreiter, A., & Streit, M. (2020). InstanceFlow: Visualizing the evolution of classifier confusion at the instance level. *2020 IEEE Visualization Conference –- Short Papers*. https://doi.org/10.1109/VIS47514.2020.00065 (cit. on pp. 4, 8, 35, 103, 104)

Ragan, E., Endert, A., Sanyal, J., & Chen, J. (2016). Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purposes. *IEEE Transactions on Visualization and Computer Graphics (VAST '15)*, *22*(1), 31–40. https://doi.org/10.1109/TVCG.2015.2467551 (cit. on p. 69)

Rao, R., & Card, S. K. (1994). The table lens: Merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 318–322. https://doi.org/10.1145/191666.191776 (cit. on pp. 39, 42)

Rauber, P. E., Fadel, S. G., Falcão, A. X., & Telea, A. C. (2017). Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 101–110. https://doi.org/10.1109/TVCG.2016.2598838 (cit. on pp. 49, 66, 78)

Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2018). Do CIFAR-10 classifiers generalize to CIFAR-10? *arXiv: 1806.00451 [cs, stat]*. Retrieved June 13, 2018, from http://arxiv.org/abs/1806.00451 (cit. on pp. 21, 22, 26)

Ren, D., Amershi, S., Lee, B., Suh, J., & Williams, J. D. (2017). Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 61–70. https://doi.org/10.1109/TVCG.2016.2598828 (cit. on pp. 20, 21, 38)

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144. https://doi.org/10.1145/2939672.2939778 (cit. on p. 35)

Rokicki, T., Kociemba, H., Davidson, M., & Dethridge, J. (2010). God's number is 20. Retrieved November 2, 2019, from http://www.cube20.org/. (Cit. on p. 73)

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519 (cit. on p. 1)

Rosling, H., & Zhang, Z. (2011). Health advocacy with gapminder animated statistics. *Journal of Epidemiology and Global Health*, *1*(1), 11. https://doi.org/10.1016/j.jegh.2011.07.001 (cit. on p. 69)

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., & Hadsell, R. (2018). Meta-learning with latent embedding optimization. *arXiv: 1807.05960*. https://doi.org/10.48550/ARXIV.1807.05960 (cit. on p. 78)

Sacha, D., Stoffel, A., Stoffel, F., Kwon, B. C., Ellis, G., & Keim, D. A. (2014). Knowledge generation model for visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 1604–1613. https://doi.org/10.1109/TVCG.2014.2346481 (cit. on p. 3)

Sacha, D., Zhang, L., Sedlmair, M., Lee, J. A., Peltonen, J., Weiskopf, D., North, S. C., & Keim, D. A. (2017). Visual interaction with dimensionality reduction: A structured literature analysis. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '16)*, *23*(1), 241–250. https://doi.org/10.1109/TVCG.2016.2598495 (cit. on p. 3)

Sainburg, T., McInnes, L., & Gentner, T. Q. (2021). Parametric UMAP embeddings for representation and semisupervised learning. *Neural Computation*, *33*(11), 2881–2907. https://doi.org/10.1162/neco_a_01434 (cit. on pp. 11, 86, 93)

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., & Heer, J. (2017). Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 341–350. https://doi.org/10.1109/TVCG.2016.2599030 (cit. on pp. 87, 105)

Satyanarayan, A., Russell, R., Hoffswell, J., & Heer, J. (2016). Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, *22*(1), 659–668. https://doi.org/10.1109/TVCG.2015.2467091 (cit. on p. 87)

Schreck, T., Tekušová, T., Kohlhammer, J., & Fellner, D. (2007). Trajectory-based visual analysis of large financial time series data. *ACM SIGKDD Explorations Newsletter*, *9*(2), 30–37 (cit. on p. 49).

Setlur, V., Battersby, S. E., Tory, M., Gossweiler, R., & Chang, A. X. (2016). Eviza: A natural language interface for visual analysis. *Proceedings of the Symposium on User Interface Software and Technology*, 365–377. https://doi.org/10.1145/2984511.2984588 (cit. on p. 69)

Settles, B. (2012). *Active learning*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00429ED1V01Y201207AIM018. (Cit. on p. 27)

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144. https://doi.org/10.1126/science.aar6404 (cit. on p. 75)

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In Y. Bengio & Y. LeCun (Eds.), *International conference on learning representations (ICLR) workshop track proceedings*. Retrieved January 30, 2018, from http://arxiv.org/abs/1312.6034. (Cit. on p. 15)

Smilkov, D., Carter, S., Sculley, D., Viégas, F. B., & Wattenberg, M. (2017). Direct-manipulation visualization of deep networks. *arXiv: 1708.03788 [cs, stat]*. Retrieved January 30, 2018, from http://arxiv.org/abs/1708.03788 (cit. on p. 19)

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Conference on Empirical Methods in Natural Language Processing*, 1631–1642 (cit. on p. 13).

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002 (cit. on p. 13)

Stahnke, J., Dörk, M., Müller, B., & Thom, A. (2016). Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '15)*, *22*(1), 629–638. https://doi.org/10.1109/TVCG.2015.2467717 (cit. on p. 75)

Steinparz, C. A., Hinterreiter, A., Stitz, H., & Streit, M. (2019). Visualization of Rubik's cube solution algorithms. In T. v. Landesberger & C. Turkay (Eds.), *EuroVis workshop on visual analytics (EuroVA '19)*. The Eurographics Association. https://doi.org/10.2312/eurova.20191119. (Cit. on pp. 6, 8)

Stitz, H., Gratzl, S., Piringer, H., Zichner, T., & Streit, M. (2019). KnowledgePearls: Provenance-based visualization retrieval. *IEEE Transactions on Visualization and Computer Graphics (VAST '18)*, *25*(1), 120–130. https://doi.org/10.1109/TVCG.2018.2865024 (cit. on p. 69)

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *Proceedings of the 30th International Conference on Machine Learning*, *28*, 1139–1147. https://proceedings.mlr.press/v28/sutskever13.html (cit. on p. 90)

Swihart, B. J., Caffo, B., James, B. D., Strand, M., Schwartz, B. S., & Punjabi, N. M. (2010). Lasagna plots: A saucy alternative to spaghetti plots. *Epidemiology*, *21*(5), 621–625. https://doi.org/10.1097/EDE.0b013e3181e5b06a (cit. on pp. 22, 23)

Talbot, J., Lee, B., Kapoor, A., & Tan, D. S. (2009). EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1283–1292. https://doi.org/10.1145/1518701.1518895 (cit. on pp. 19, 20)

Tang, J., Liu, J., Zhang, M., & Mei, Q. (2016). Visualizing large-scale and high-dimensional data. *Proceedings of the 25th International Conference on World Wide Web*, 287–297. https://doi.org/10.1145/2872427.2883041 (cit. on pp. 90, 93, 94)

Tenenbaum, J. B. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*(5500), 2319–2323. https://doi.org/10.1126/science.290.5500.2319 (cit. on pp. 10, 11, 77, 94)

Thomas, J. J., & Cook, K. A. (2005). *Illuminating the path: The research and development agenda for visual analytics*. IEEE Computer Society Press. (Cit. on p. 3).

Tomar, V. S., & Rose, R. C. (2014). Manifold regularized deep neural networks. *Fifteenth Annual Conference of the International Speech Communication Association*, 348–352. https://wiki.inf.ed.ac.uk/twiki/pub/CSTR/ListenTerm1201415/rose.pdf (cit. on p. 78)

Torgerson, W. S. (1952). Multidimensional scaling: I. theory and method. *Psychometrika*, *17*(4), 401–419. https://doi.org/10.1007/BF02288916 (cit. on p. 10)

Tukey, J. W. (1962). The future of data analysis. *The Annals of Mathematical Statistics*, *33*(1), 1–67. https://doi.org/10.1214/aoms/1177704711 (cit. on p. 1)

van den Elzen, S., Holten, D., Blaas, J., & van Wijk, J. (2016). Reducing snapshots to points: A visual analytics approach to dynamic network exploration. *IEEE Transactions on Visualization and Computer Graphics*, *22*(1), 1–10. https://doi.org/10.1109/TVCG.2015.2468078 (cit. on p. 49)

van den Elzen, S., & van Wijk, J. (2011). BaobabView: Interactive construction and analysis of decision trees. *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST '11)*, 151–160 (cit. on pp. 19, 20).

van der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, *5*, 384–391. http://proceedings.mlr.press/v5/maaten09a.html (cit. on pp. 11, 77–79, 86)

van der Maaten, L. (2020). T-SNE – FAQ. Retrieved April 20, 2020, from https://lvdmaaten.github.io/tsne/%5C#faq. (Cit. on p. 72)

van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579–2605 (cit. on pp. 10, 11, 67, 77, 85, 88, 92).

van der Maaten, L., Postma, E., & van den Herik, J. (2009). *Dimensionality reduction: A comparative review* (Technical Report TR 2009-005). Tilburg University. Tilburg, Netherlands. (Cit. on p. 10).

VanderPlas, J., Granger, B. E., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., Lees, E., Timofeev, I., Welsh, B., & Sievert, S. (2018). Altair: Interactive statistical visualizations for python. *Journal of open source software*, *3*(32), 1057 (cit. on p. 104).

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2014). OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), 49–60. https://doi.org/10.1145/2641190.2641198 (cit. on p. 29)

Venna, J., & Kaski, S. (2001). Neighborhood preservation in nonlinear projection methods: An experimental study. In G. Dorffner, H. Bischof, & K. Hornik (Eds.), *Artificial neural networks — ICANN 2001* (pp. 485–491). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-44668-0_68. (Cit. on p. 95)

Vu, V. M., Bibal, A., & Frenay, B. (2022). Integrating constraints into dimensionality reduction for visualization: A survey. *IEEE Transactions on Artificial Intelligence*, 1–19. https://doi.org/10.1109/TAI.2022.3204734 (cit. on pp. 86, 96)

Walchshofer, C., Hinterreiter, A., Xu, K., Stitz, H., & Streit, M. (2021). Provectories: Embedding-based analysis of interaction provenance data. *IEEE Transactions on Visualization and Computer Graphics (Early Access)*. https://doi.org/10.1109/TVCG.2021.3135697 (cit. on pp. 6, 8, 69–71, 103)

Wang, C., & Han, J. (2022). DL4scivis: A state-of-the-art survey on deep learning for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 1–1. https://doi.org/10.1109/TVCG.2022.3167896 (cit. on p. 2)

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., & Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1386–1393. https://doi.org/10.1109/CVPR.2014.180 (cit. on p. 97)

Wang, J., Gou, L., Shen, H.-W., & Yang, H. (2019). DQNViz: A visual analytics approach to understand deep q-networks. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 288–298. https://doi.org/10.1109/TVCG.2018.2864504 (cit. on pp. 19, 20, 38)

Wang, J., Gou, L., Yang, H., & Shen, H.-W. (2018). GANViz: A visual analytics approach to understand the adversarial game. *IEEE Transactions on Visualization and Computer Graphics*, *24*(6), 1905–1917. https://doi.org/10.1109/TVCG.2018.2816223 (cit. on pp. 18–20, 38)

Wang, Q., Chen, Z., Wang, Y., & Qu, H. (2021). A survey on ML4vis: Applying MachineLearning advances to data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 1–1. https://doi.org/10.1109/TVCG.2021.3106142 (cit. on p. 2)

Wang, Z., & Oates, T. (2015). Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. *Trajectory-Based Behavior Analytics: Papers from the 2015 AAAI Workshop*, 40–46 (cit. on p. 10).

Ward, M. O., & Guo, Z. (2011). Visual exploration of time-series data with shape space projections. *Computer Graphics Forum*, *30*(3), 701–710. https://doi.org/10.1111/j.1467-8659.2011.01919.x (cit. on p. 49)

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral science* (Doctoral dissertation). Harvard University. (Cit. on p. 1).

Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viegas, F., & Wilson, J. (2019). The what-if tool: Interactive probing of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 1–1. https://doi.org/10.1109/TVCG.2019.2934619 (cit. on p. 18)

Williams, C. K. (2002). On a connection between kernel PCA and metric multidimensional scaling. *Machine Learning*, *46*, 11–19. https://doi.org/10.1023/A:1012485807823 (cit. on p. 10)

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, *2*(1), 37–52. https://doi.org/10.1016/0169-7439(87)80084-9 (cit. on p. 77)

Wongsuphasawat, K., Smilkov, D., Wexler, J., Wilson, J., Mane, D., Fritz, D., Krishnan, D., Viegas, F. B., & Wattenberg, M. (2018). Visualizing dataflow graphs of deep learning models in TensorFlow. *IEEE Transactions on Visualization and Computer Graphics*, *24*(1), 1–12. https://doi.org/10.1109/TVCG.2017.2744878 (cit. on p. 18)

Wongsuphasawat, K. (2020). Encodable: Configurable grammar for visualization components. *IEEE Visualization Conference (VIS'20)*, 131–135. https://doi.org/10.1109/VIS47514.2020.00033 (cit. on p. 87)

Wu, Y., Kozintsev, I., Bouguet, J., & Dulong, C. (2006). Sampling strategies for active learning in personal photo retrieval. *2006 IEEE International Conference on Multimedia and Expo*, 529–532. https://doi.org/10.1109/ICME.2006.262442 (cit. on p. 27)

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv: 1708.07747 [cs, stat]*. Retrieved March 18, 2019, from http://arxiv.org/abs/1708.07747 (cit. on p. 32)

Zeng, H., Haleem, H., Plantaz, X., Cao, N., & Qu, H. (2017). CNNComparator: Comparative analytics of convolutional neural networks. *VADL 2017:*

Workshop on Visual Analytics for Deep Learning. https://vadl2017.github.io/paper/vadl_0108-paper.pdf (cit. on pp. 18–20, 38)

Zhang, J., Wang, Y., Molino, P., Li, L., & Ebert, D. S. (2019). Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 364–373. https://doi.org/10.1109/TVCG.2018.2864499 (cit. on pp. 18, 20)

Zheng, Y. (2015). Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology*, *6*(3), 1–41. https://doi.org/10.1145/2743025 (cit. on pp. 10, 55)

Zhong, W., Xie, C., Zhong, Y., Wang, Y., Xu, W., Cheng, S., & Mueller, K. (2017). Evolutionary visual analysis of deep neural networks. *ICML Workshop on Visualization for Deep Learning*, 9 (cit. on p. 38).

Zhu, B., & Chen, W. (2016). Performance histogram curve: Abstractive visualization and analysis of NBA games. *International Journal of Software & Informatics*, *10*(3), 11. https://doi.org/10.21655/ijsi.1673-7288.00231 (cit. on p. 49)

CONFUSIONFLOW: ADDITIONAL INFORMATION

A

A.1 PERFORMANCE MEASURES

Several performance measures can be derived from intermediate results during the training of classifiers. Consider a classification problem with a training dataset X consisting of instances $\{x_1, ..., x_N\}$, for which *ground truth* labels $g_i \in \Gamma$ are available. The set of different classes $\Gamma = \{\gamma_1, ..., \gamma_K\}$ is called the class alphabet. A classifier model $c(\theta)$ with parameters θ is trained on X such that the predicted class labels $c(\theta)(x_i)$ best match the ground truth labels g_i. The exact notion of "best match" is specified in the loss function. This function, together with an optimization method, determines how the parameters θ are updated at each training step t. In case of neural networks, one training step is called an *epoch*, and is comprised of passing the whole training data through the network once. Over the course of the training, the classifier progresses through different model states $c(\theta_t)$, each with different parameters and a different error behavior. The term *parameters* in this case is not to be confused with the term *hyperparameters*, which stands for model and optimization settings that have to be chosen before training. We chose the term *configuration* to refer to the set of model, optimization techniques, hyperparameters, and input data.

A typical measure for a classifier's quality is the accuracy A. It is given by the fraction of correctly classified instances:

$$A_t = \frac{|\{x_i \mid c(\theta_t)(x_i) = g_i\}|}{N}. \tag{A.1}$$

Other popular aggregate measures are precision and recall, which are built on the binary classification notions of true positives, false positives, and false negatives. For a given class γ_j, the number of true positives is given by $\mathrm{TP}_{j,t} = |\{x_i \mid c(\theta_t)(x_i) = \gamma_j \wedge g_i = \gamma_j\}|$, i. e. the number of correctly classified instances of that class. The number of false positives is defined by $\mathrm{FP}_{j,t} = |\{x_i \mid c(\theta_t)(x_i) = \gamma_j \wedge g_i \neq \gamma_j\}|$, i. e. the number of instances classified as γ_i that actually belong to another class. Finally, the number of false negatives is given by $\mathrm{FN}_{j,t} = |\{x_i \mid c(\theta_t)(x_i) \neq \gamma_j \wedge g_i = \gamma_j\}|$, i. e. the number of instances of class γ_j that the classifier wrongly assigned to another class. With these quantities, precision and recall can be defined for each class γ_j by:

$$\mathrm{Precision}_{j,t} = \frac{\mathrm{TP}_{j,t}}{\mathrm{TP}_{j,t} + \mathrm{FP}_{j,t}} \qquad \mathrm{Recall}_{j,t} = \frac{\mathrm{TP}_{j,t}}{\mathrm{TP}_{j,t} + \mathrm{FN}_{j,t}}. \tag{A.2}$$

The harmonic mean of precision and recall is called F_1-score.

An extension of individual class-wise performance measures is the confusion matrix M, which lists confusion counts for all pairs of classes:

$$M_{ij,t} = |\{x_l \mid g_l = \gamma_i \wedge c(\theta_t)(x_l) = \gamma_j\}|. \tag{A.3}$$

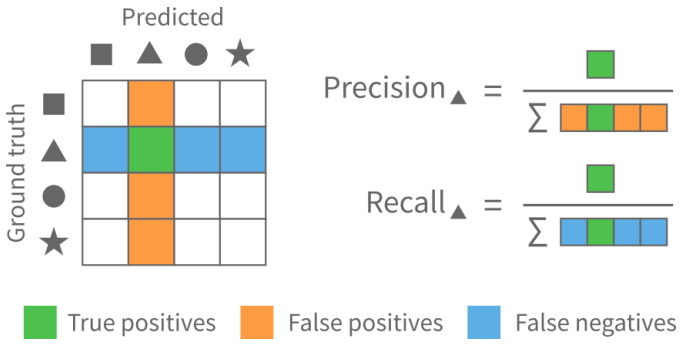


Figure A.1: Class-conditional precision and recall in the context of the confusion matrix.

This means that entry (i, j) in M is the number of instances with ground truth class label γ_i that the classifier assigned to class γ_j.

Figure A.1 schematically shows the relationship between the confusion matrix and precision and recall.

A.2 SCALABILITY: ADDITIONAL FIGURES

Here we provide a number of additional screenshots from the multi-step scalability study presented in Section 3.4.2..

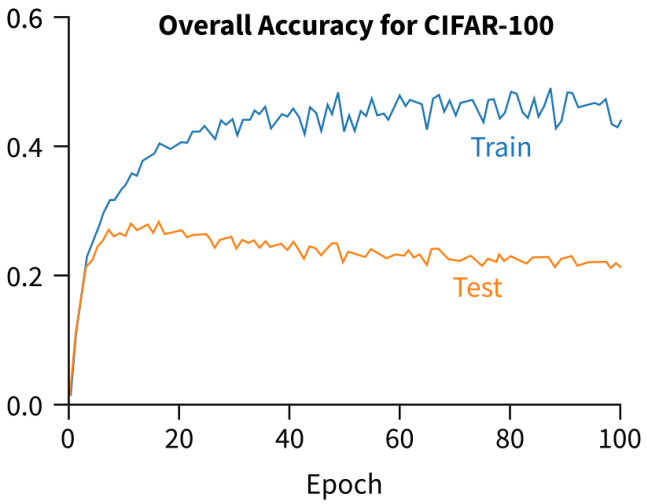
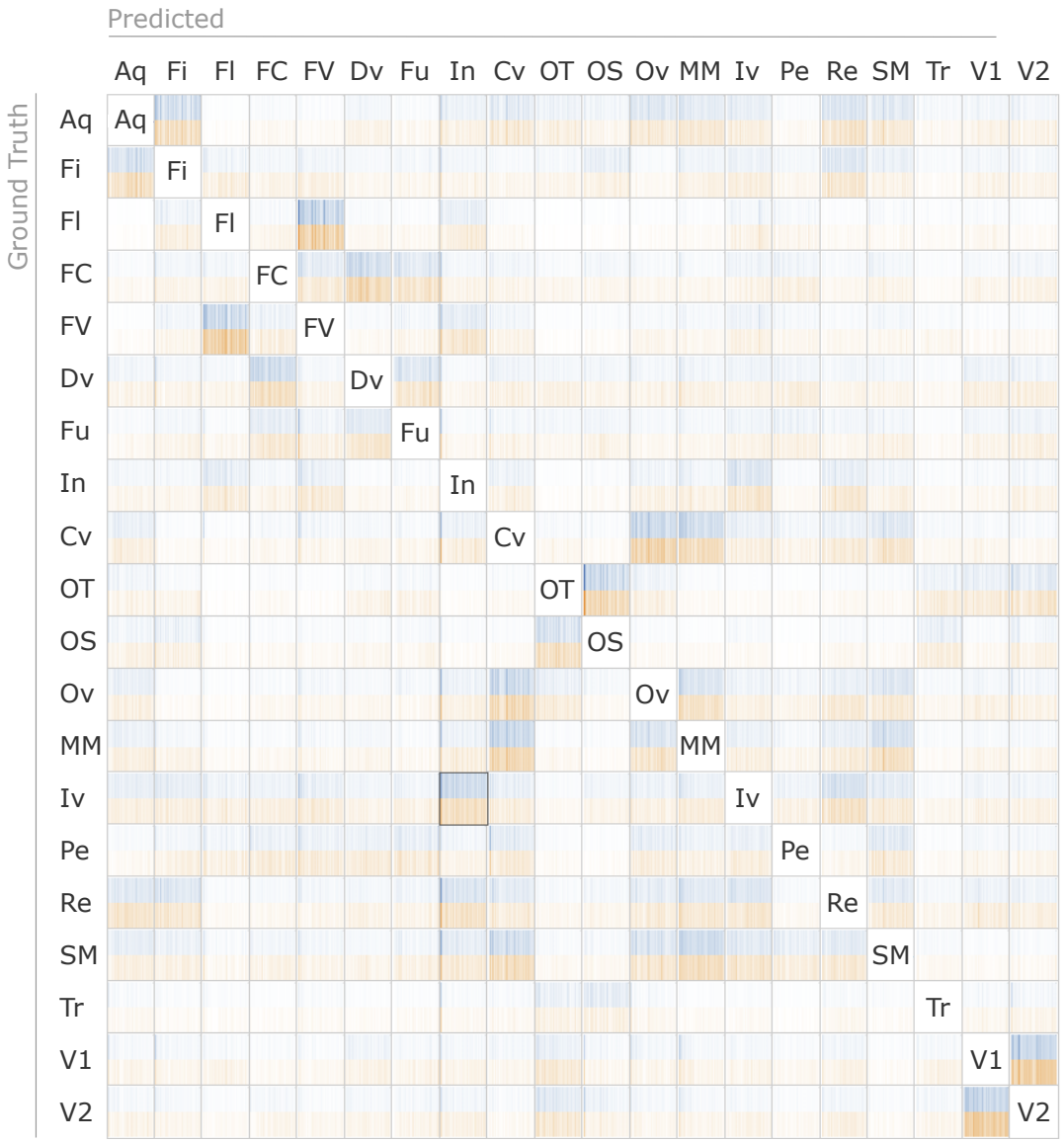


Figure A.2: Global accuracy during training of a neural network on the CIFAR-100 images dataset. Severe overfitting is apparent from the temporal trend of the global accuracy. The plot was slightly adapted from the ConfusionFlow detail view.



Aq	Aquatic mammals	In	Insects	Pe	People
Fi	Fish	Cv	Carnivores	Re	Reptiles
Fl	Flowers	OT	Outdoor things	SM	Small mammals
FC	Food containers	OS	Outdoor scenes	Tr	Trees
FV	Fruits/Vegetables	Ov	Omnivores/Herbivores	V1	Vehicles 1
Dv	Devices	MM	Medium-sized mammals	V2	Vehicles 2
Fu	Furniture	Iv	Non-insect invertebrates		

Figure A.3: Superclass ConfusionFlow matrix for a neural network classifier trained on the CIFAR-100 image dataset, and evaluated on the train (■) and test folds (■), respectively. The low-level class labels were used during training. The superclass confusion values were subsequently determined from the class predictions.

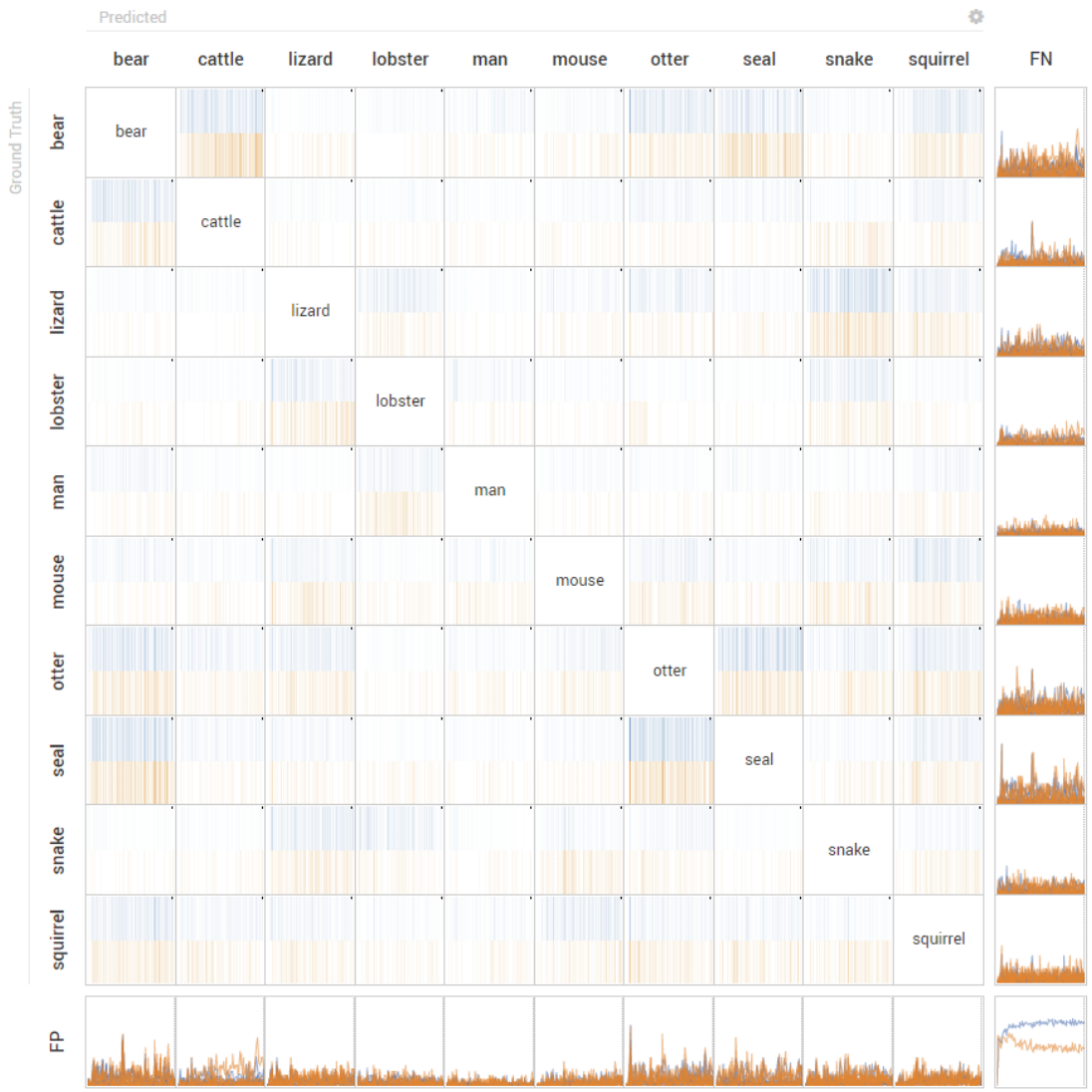


Figure A.4: ConfusionFlow matrix for a neural network classifier trained on the CIFAR-100 image dataset, and evaluated on the train (■) and test folds (■), respectively. The ten classes with the lowest F_1-scores were selected.

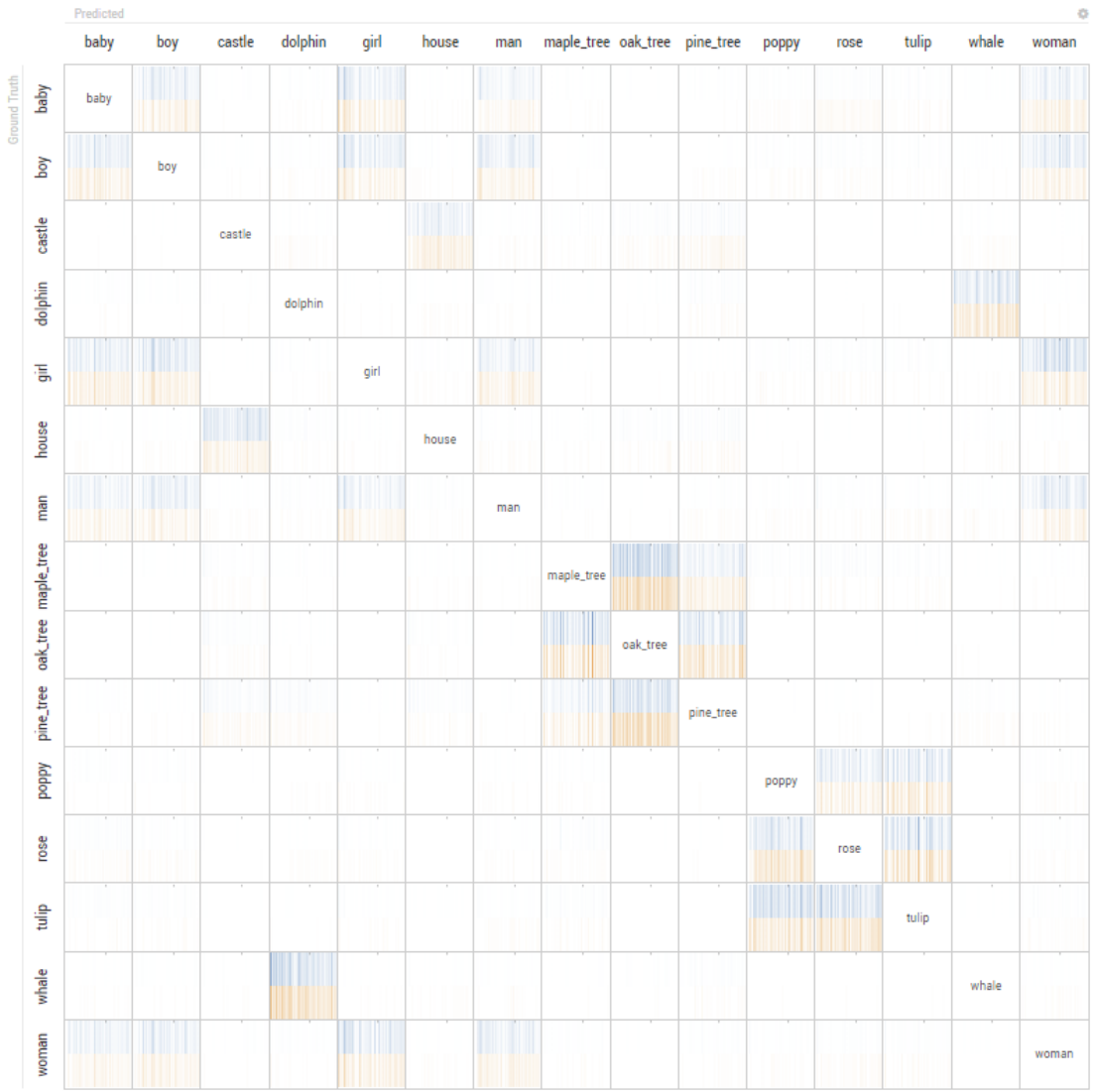


Figure A.5: ConfusionFlow matrix for a neural network classifier trained on the CIFAR-100 image dataset, and evaluated on the train (■) and test folds (■), respectively. All classes present in the twenty worst pairs with respect to class confusion were selected.

RUBIK'S CUBE DEMONSTRATOR

B

To tighten the connection between our visualization approach for Rubik's cube solution strategies (Section 5.4.1) and real-world actions and decisions made by users, we built a physical interactive demonstrator. The setup is shown schematically in Figure B.1. It consists of a special Rubik's cube with sensors and Bluetooth connectivity (Giiker Supercube, `http://www.giiker.cn`), a Lego Mindstorm robot (MindCub3r; Gilday, 2019), and a display showing an adapted version of the interactive Rubik's cube visualization.

While the user tries to solve the cube, rotations are immediately reflected in the visualization. The trail of previous cube states, as well as the current one, are shown in the embedded state space of several hundreds of reference solution trajectories of random initial cubes. Since the current cube state may not have been part of this initial set of calculated solutions, out-of-sample extension must be used. We implemented a parametric, real-time out-of-sample extension, based on the work by Gisbrecht et al. (2015). Along with the reference trajectories and the trajectory of movements performed so far, also the future path necessary to arrive at the solution is displayed (for a selected solution algorithm). Upon each cube movement, the future path is updated, if it has been affected by the user's decision. This serves as a highly salient visual feedback for the user, as "incorrect" movements may result in drastic changes of the computed future paths.

To aid the users, the correct movements to stay on the right track are shown with a rendering of Rubik's cube. If users decide to stop trying to solve the cube on their own, they can hand the cube over to a Lego Mindstorms robot that will continue the solution. Users can then simultaneously watch the real cube being solved, while the current cube state moves along the solution trajectory in the visualization. The demonstrator setup also includes a two-player mode, in which two players can race through the solution trajectories with two separate Bluetooth cubes. The setup is mainly used to introduce prospective computer science students to the functioning of algorithms, interactive visualization, and projection techniques.

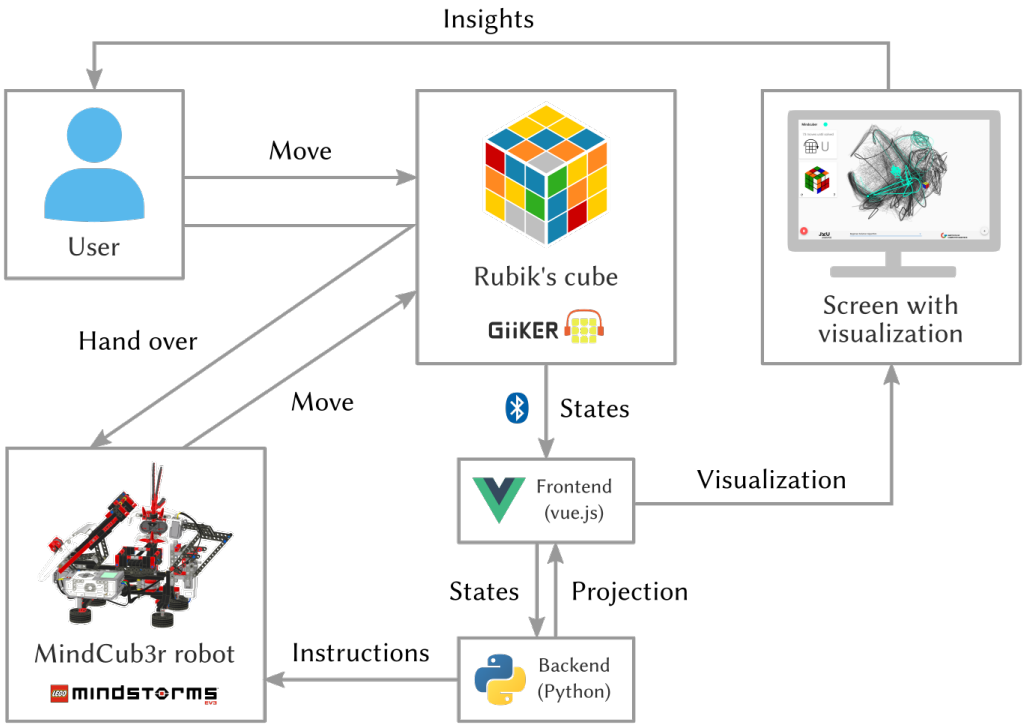


Figure B.1: Schematic of the Rubik's cube physical demonstrator setup. Users can solve a Rubik's cube, which transmits its states to a vue.js-powered web application via Bluetooth. The projected cube states are calculated in real time using out-of-sample extension implemented in Python. The projection of the current state and the previous trajectory are visualized on a screen. Users can hand off the unsolved cube to a Lego Mindstorms robot, which receives its instructions from the Python backend, and automatically solves the cube.