TU WIEN Informatics

Prototypische Netzwerke für die Visualisierung von großen unstrukturierten Daten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

Mario Stoff, BSc
Matrikelnummer 11777706

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.in techn. MSc Manuela Waldner

Wien, 27. November 2024

Mario Stoff	Manuela Waldner

Informatics

Prototypical Visualization

Using Prototypical Networks for Visualizing Large Unstructured Data

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media and Human-Centered Computing

by

Mario Stoff, BSc
Registration Number 11777706

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.in techn. MSc Manuela Waldner

Vienna, November 27, 2024

Mario Stoff Manuela Waldner

Erklärung zur Verfassung der Arbeit

Mario Stoff, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. November 2024

Mario Stoff

Danksagung

Zuallererst schulde ich meiner Betreuerin Manuela Waldner eine große Menge Dank. Ich möchte ihr für all die Stunden danken, in denen sie mit mir meine Ergebnisse angeschaut und Lösungen für Probleme vorgeschlagen hat. Ihr aufrichtiges Interesse and jeglichem noch so kleinen Fortschritt hat mich immer sehr zum Weitermachen motiviert.

Außerdem möchte ich all den Freunden danken, die ich im Zuge dieses Studium kennengelernt habe. Ich kann mir keine Gruppe vorstellen, mit der ich lieber durch all die Jahre an Vorlesungen, Projekten und Prüfungen gegangen wäre.

Aber der größte Dank gebührt meiner Familie, die mich all die Jahre lang sowohl finanziell, als auch emotional, unterstützt und ermutigt haben. Dafür bin ich unendlich dankbar!

Acknowledgements

First and foremost, I owe a great deal of thanks to my supervisor Manuela Waldner. She is a constant stream of good feedback and new ideas. I would like to thank her for all the hours she spent listening to my reports, working out solutions for my problems, and for her genuine interest in every little new progress that I made.

I would also like to thank all the friends I have made thanks to my studies. I could not ask for better company in all the years of lectures, projects, and exam periods.

But the biggest thanks goes to my family who have encouraged and supported my studies both financially and emotionally throughout the years. For that, I am deeply grateful!

Kurzfassung

Das Untersuchen und Verstehen von Daten sind alltäglicher Bestandteil der Arbeit vieler Fachläute. Das kann ein schwieriges Unterfangen sein, wenn die Menge an Daten so groß ist, dass sie schwer zu überschauen sind. Eine sehr effektive Methode, um einen schnellen Überblick über die Strukturen und Zusammenhänge in einem Datensatz zu bekommen, is diesen visuell darzustellen. Oft ist dies aber aufgrund der Größe und hoch-dimensionalen Charakteristik des Datensatzes nicht so leicht möglich. Machine Learning Modelle können zwar mit der Organisation und Klassifizierung der Daten helfen, jedoch benötigen diese oft Unmengen an beschrifteten Trainingsdaten, welche oft kostspielig und umständlich zu beschaffen sind. Deshalb sind Modelle besonders interessant, welche Daten zuverlässig auf Basis von nur wenigen Beispieldaten klassifizieren können. Eine solche Art von Modellen sind sogenannte prototypische Netzwerke. Diese sind in der Lage, Daten so in einen niedrigdimensionalen Raum einzubetten, dass sich ähnliche Daten um einen klassenspezifischen Prototypen scharen.

Im Rahmen dieser Arbeit untersuchen wir, ob der Einbettungsraum eines prototypischen Netzwerks eine brauchbare Methode darstellt, um hoch-dimensionale Daten in einem zwei-dimensionalen Scatter Plot zu visualisieren. Das Ziel ist es, die Dimensionalität von Daten mit Hilfe eines prototyischen Netzwerks so zu reduzieren, dass die hoch-dimensionalen Strukturen und Relationen erhalten bleiben. Daraus soll eine zwei-dimensionale Repräsentation der Daten entstehen, in welcher ähnliche Datenpunkte zusammenhängende Gruppierungen bilden. Diesen Ansatz vergleichen wir mit anderen überwachten und unüberwachten Methoden zur Reduzierung der Datendimensionalität. Mit Hilfe quantitativer Experimente, in denen wir die Separierung verschiedener Klassen von Daten anhand statistischer Größen bemessen, sowie qualitativer Untersuchung unserer Ergebnisse, kommen wir zu dem Schluss, dass unser ProtoNet Ansatz eine genau so gute, wenn nicht bessere, Separierung der Daten erzeugen kann wie die anderen untersuchten Methoden.

Abstract

Making sense of data is something that many professionals are required to do on a daily basis. This can be a difficult task if the amount of data is so large that it can not be easily examined. One effective method of quickly getting an overview of data structure is visualization, but this is not always a feasible solution with large data due to the sheer amount of data and also the potentially high dimensionality. Machine learning models can help with with the organization and classification of data, but they often require large quantities of labeled training data, which is frequently not readily available. This is why models that can reliably classify data based on only few examples for each class are an interesting topic of research. One such kind of model are prototypical networks. They utilizes few samples to create an embedding space in fewer dimensions, in which similar data points cluster around a single class prototype.

In this thesis, we investigate if the embedding space of a prototypical network makes for a good approach for the purpose of visualizing high-dimensional, unstructured data. The goal is to reduce the dimensionality of the data in such a way that the high-dimensional relations and structures between data points are preserved, resulting in 2D representations of the data that form coherent class clusters in a scatter plot visualization. This approach is compared with, and evaluated against, other well known supervised and unsupervised dimensionality reduction techniques. Through quantitative experiments relying on statistical measures, as well as a qualitative evaluation of our results, we find that our ProtoNet is capable of producing point embeddings in which the spatial separation of classes is as good or better than the other methods.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

In many fields of science, data analysts have to make sense of huge amounts of unstructured data. This can be an incredibly difficult task when they do not even know what exactly they are looking for. Oftentimes, the most interesting parts of the data are patterns that were not previously expected. Humans can best detect patterns through visual representations, but visualizing enormous sets of high-dimensional data can quickly become unclear and confusing, both because high-dimensional data cannot be directly visualized and the more data we try to visualize at once, the harder it gets to read. That is why the data needs to be selectively sampled and visualized in a fashion that makes it clear for the user without jeopardizing the original structure of the data.

One very common way to visualize high-dimensional data is to embed the data in a low-dimensional space. To achieve this, the data is projected on a lower-dimensional manifold where the goal is to preserve the high-dimensional structures and relations after the transformation of the data. Most commonly, a distance function is used to determine similarity between data points. This, in turn, determines the positioning of the data points in the new data space. There are both linear and non-linear variations of dimensionality reduction (DR) algorithms. These categories refer to the underlying topologies that the algorithms can learn. A linear dimension reduction (e.g. Principle Component Analysis) can only learn a linear representation of the data, meaning that the output axes of the reduced data space will be linear combinations of the original features, while non-linear algorithms are capable of learning more complex structures. While linear reduction is oftentimes faster, the limitations it poses on the data transformation can lead to a lack in outcome quality. This is particularly true when the data lies on a lower-dimensional non-linear manifold ("Swiss Roll" [RS00]). DR exceeds other visualizations of high-dimensional data, such as parallel coordinate plots, in terms of scalability, but suffers from a certain degree of information loss due to the data transformation.

One key aspect of the data that should be preserved even through data transformation is class structure, i.e., clustering of similar data points. By plotting data points on a scatter plot, where between-point similarities are preserved, users can gain a quick understanding of the observed data. This allows to make visible both relationships that the user would expect in the data and also such structures that are maybe entirely new to the user's understanding and therefore unexpected and beneficial for deepening the user's understanding. But problems arise when the data is completely unknown. Traditional machine learning classifiers only provide limited generalization ability, meaning they need large amounts of labeled ground truth data for training and they falter when it comes to identifying classes where no, or only few, examples were present in the training data.

Now what if the entire dataset is completely unlabeled? The concept of few-shot learning describes a number of machine learning algorithms that aim to confidently classify items even when there are only a few selected examples available for training, or even when there are classes without any training samples to begin with (zero-shot learning). One example that has been shown to be reliable for these kinds of tasks are prototypical networks (PN). A PN will learn an embedding space with the goal to minimize the distance between the few available samples with the same label and an artificial class prototype. Unknown data items should then cluster around the prototype they are most similar to. In the scope of this thesis, we are dealing with image datasets containing large amounts of unlabeled data that are challenging to reasonably visualize with a traditional scatter plot projection. We want to investigate to what extent we can use the class prototypes produced by PN to improve the layout of scatter plots. We want to see if they can be used as meaningful samples of the original data and whether the layout produced by PN displays distinguishable clusters that represent the actual class structure within the data.

1.2 Aim of the Work

The problem we are dealing with is large amounts of unlabeled image data that we want to visualize so that a human user can easily detect cluster structures and determine whether the data structure resembles what they would have expected from the data and whether there are other unexpected classes. Since this problem is very akin to few shot learning problems, we propose to use a prototypical network as our data transformation method, as these networks have been shown to produce state-of-the-art results for such problems. The hypothesis is that prototypical networks are a useful tool for generating scatter plot visualizations from large datasets where the goal is to show class structures according to the user's preconceived mental model of the data, i.e., what classes they expect to find in the data. This is implied by the labeled samples that are selected for the PN. The goal of this thesis is to explore how well PNs are suited to tackle this problem compared to other, more traditional DR methods like Uniform Manifold Approximation and Projection (UMAP), semi-supervised UMAP, or Linear Discriminant Analysis (LDA). To work towards this goal, this thesis aims to answer the following four research questions:

R1) How does the interpretation of unstructured data differ between projections based on prototypical networks and traditional unsupervised or supervised dimensionality reduction methods?

R2) How does the computational performance differ between these methods?

R3) How can we effectively visualize prototypes and other data items, especially for very large datasets?

R4) How well can prototypical network-based data visualization be integrated into interactive visual analysis of large, unstructured data?

1.3 Methods Overview

Answering these questions is done using two separate methodological approaches. **R1)** and **R2)** are explored through exhaustive quantitative experiments on a variety of datasets. Applying the different dimensionality reduction methods and visualizing the resulting output allows us to directly measure the computational performance. At the same time we can apply state-of-the-art metrics to evaluate the visual separation quality of the scatter plots. We use metrics that have been proven to score scatter plot layouts in a way that coincides with human's interpretation of what a well separated cluster layout is. Using these metrics, we can find which of our methods produce plots that are easiest to interpret thanks to the clearest separation of ground truth classes.

For the investigation of **R3)** and **R4)** we create an interactive visualization interface, where users can explore different datasets. Using this visualization, we can learn how to best represent the available data in a way that is comprehensive to the user. The suitability for interactive visual analysis can be assessed by investigating how well the visualization can deal with edge cases, where, for instance, classes in the dataset had no labeled samples at all and were therefore completely disregarded in the training of the prototypical network. A qualitative inspection based on selected use cases will demonstrate the strengths and limitations of our chosen methods. We can call our method a success if these disregarded classes are nonetheless discernible in the resulting scatter plot.

1.4 Contribution

In the scope of this thesis, we introduce a novel use for prototypical networks. To the best of our knowledge, prototypical networks have never been investigated in the context of high-dimensional data visualization or comprehensive sampling. Through rigorous experiments, we can show that this technique can add valuable benefits to the similarity-preserving quality of the dimensionality reduction step in comparison to other semi- and unsupervised techniques. We furthermore present the prototype of a visualization interface, allowing users to explore the class structure of datasets and investigate representative samples and outliers.

1.5 Outline

The structure of this thesis will be as follows: In Chapter 2, we introduce the underlying basic concepts and technologies that are the building blocks of this thesis. This includes the visualization of high-dimensional data and dimensionality reduction, prototypical networks and their intended use, and traditional sampling techniques. Chapter 3 introduces the basic steps of an interactive machine learning visualization and how we execute these steps. We explain our design choices and showcase the functionality of our prototype implementation. In Chapter 4, we then go into more detail on the technical aspects of our implementation. We highlight the necessary pre-processing steps, our version of the prototypical network, as well as the algorithms used to pick representative samples. The outcome of our experiments will be reported in detail in Chapter 5, as well as a qualitative walkthrough of our application. The evaluation of the reported results will be discussed in Chapter 6. Finally, Chapter 7 brings the thesis to a close with a final summary of the contents of this thesis, the limitations of this work an outlook on future work.

CHAPTER 2

Background & Related Work

In this chapter, we introduce the concepts and technologies that make up the underlying principles and basis for our work. We take a look at what these concepts are, how they have been applied by others, and how they tie into our work.

2.1 High-Dimensional Data

Visualizing high-dimensional data in a comprehensive manner has been a well studied topic for a long time. Many different techniques have been developed for a myriad of use cases. Liu et al. [LMW$^+$16] have summarized many of these techniques and provided a comprehensive survey of advances in high-dimensional data visualization. They group visualization techniques into three different categories based on where in the visualization pipeline these techniques are applied. These three categories are data transformation, visual mapping, and view transformation.

The most relevant category for this thesis are techniques that are used during data transformation. This includes dimensionality reduction techniques which we will investigate, such as PCA [JC16] or LDA [BG98]. But there are also other approaches like subspace clustering [TFH11][CFZ99][Vid11], where multiple embeddings of the data are identified to capture different aspects of the data, or topological data analysis [HLH$^+$16], where mathematical disciplines are combined with computer science to describe the shape of the data.

For techniques that are used during the visual mapping, Liu et al. [LMW$^+$16] give examples such as axis-based methods. This includes well known visualizations like scatter plot matrices and parallel coordinate plots [HW13], where different dimensions of the data are directly mapped to axes of the visualization plot. These techniques are less relevant for unstructured data, as the axes carry little meaning for that kind of data. It also includes other approaches like encoding dimensions in glyphs [Che73][CHUW08][CGSQ11], using separate displays for dimensions where information is encoded in each pixel [KK94][YPH$^+$04], or animations [HR07].

The third category of techniques that apply during view transformation includes examples such as illustrative rendering [SW09][JLJC05], which is the use of artistic rendering techniques to highlight certain parts of a visualization, using color blending, or utilising image space metrics [DK10][WAG05], all to accentuate different aspects of the data.

As stated, the most relevant family of methods for working with high-dimensional data for this thesis is dimensionality reduction (DR). The goal of DR is to find an optimal representation of the data within a space of fewer dimensions (ideally two or three for visualization), where local and global structures within the data are preserved. Methods for DR can be roughly grouped into two categories: linear and nonlinear. Linear methods mostly use statistical analysis methods to find linear combinations of the original features that make up the axes of a lower dimensional space. This makes them often easier to calculate and require less computational power, but also restricts their flexibility in terms of data mapping. Some prominent examples of linear DR are Principle Component Analysis (PCA) [JC16], Linear Discriminant Analysis [BG98], or Independent Component Analysis (ICA) [WC06]. Nonlinear methods, on the other hand, are computationally more expensive, as they have to learn embedding functions that transform the data to a new space of fewer dimensions. Nonlinear methods include, e.g., Self Organizing Maps (SOM) [Koh90], t-Distributed Stochastic Neighbor Embedding (t-SNE) [VdMH08], or Uniform Manifold Approximation and Projection (UMAP) [MHM18].

One obvious drawback of reducing the dimensions of the data is the inevitable loss of information. In order to map the data into fewer dimensions, much information needs to be condensed, converted, or removed. This loss tends to grow the higher the dimensionality of the original data is. Since we start out with image data, and our visualization space is two-dimensional, we can potentially lose a lot of information when applying DR methods. To minimize this information loss, Choo et al. [CBP09] proposed a two-stage framework for the visualization of clustered high-dimensional data. They argue that a single DR method can only achieve a certain optimal reduced dimension, which is often larger than two. Therefore, they apply one DR method to reduce the dimensionality of data to a certain optimal target dimension based on some optimization criteria, and then further reduce the data to a two-dimensional space with another DR method. In their experiments, the best results could be achieved when using LDA followed by PCA. They were, however, only investigating linear DR methods. For this thesis, we also adopt a multi-stage framework to find two-dimensional embeddings for images, so that their cluster structure can be visualized. The first stage can rather be viewed as a data preparation step, as it is preemptively performed on all the datasets. It is comprised of a feature extraction method called DINO (more detail in 3.1), while the later stage is made up of the different DR methods that we want to compare.

2.2 Interactive Machine Learning

Machine Learning (ML), in it's basic sense, is an automatic process, where data and algorithms are used to imitate the way humans learn. The actual inner workings of such ML processes are often obscure and hidden like in a "black box". We know what data we put into the model, and we know what outcome is produced. In between there is no interaction by a user. Fully automatic ML technologies have been very successful, e.g, voice recognition or self-driving cars, but for highly complex domains, like biomedicine, to forgo human domain knowledge can be a disadvantage.

Interactive Machine Learning (IML), on the other hand, is a combination of human expertise and computational power. There is still an algorithm that learns to produce a certain output from input data, but now human reasoning is introduced into the mix. The idea of IML is based on three approaches. First, Reinforcement Learning (RL) is a method where an artificial intelligence is trained to act in a way that maximizes some kind of reward. This training involves not correct input-output pairs, but rather a trial-and-error approach to find the best outcome [NiA24]. Second, Preference Learning is a specialization of RL and describes systems, where the ranking of the predicted output is important [FH10]. And third, Active Learning describes systems, where the algorithm is allowed to ask for a human annotator to label select data instances during training [Set09]. The key aspect of all these approaches is that the learning part is not based solely on the computer, but done in cooperation with a human expert.

IML can be used in all sorts of domains and applications. Berg et al. [BKK+19] describe an IML system for (bio)image analysis, where users can influence the decision making of the ML classifier by annotating images. Choo et al. introduce UTOPIAN [CLRP13], a visual analytics system for topic modeling used in the analysis of text document collections, where users are able to steer the results. Morpheus by Müller et al. [MAK+08] is a visualization tool that enables users to explore subspace clusters of high-dimensional data where they can see the effects of different parameter settings and can provide feedback to the system to improve the subspace clustering.

This is already very akin to the approach we investigate in this thesis. We want a user to be able to explore the class structure of image data in a low-dimensional subspace where the user can iteratively provide samples to train and refine the embedding methods.

2.3 Prototypical Networks

Prototypical networks come from the classification domain and have first been introduced by Snell et al. in 2017 [SSZ17b]. They are a type of machine learning classifier that aims to address few-shot classification problems. In their survey on few-shot learning, Wang et al. [WYKN20] define few-shot learning as a type of machine learning problem where the experience of the program, i.e., the training data consists of only a limited number of examples for which supervised information is available. The core idea of prototypical networks is that an embedding space exists, where all points of a certain class cluster around a class prototype. A neural network is used to create such an embedding, where these class prototypes are generated by averaging the positions of the respective (few) training samples in the embedding space. Unknown data is then classified by computing the distance in the metric space of the items to the prototype representations of each class. The nearest prototype decides the classification result.

The creators of prototypical networks have used them for image classification benchmark tests, but prototypical networks like this have also been used and adapted for various tasks in different domains. They have, e.g., been applied for Named Entity Recognition [FLK19], Event Detection [DZK+20], or Relation Classification [FBSL19] [GHLS19] in the field of natural language processing. Extensions are, e.g., Gaussian Prototypical Networks [For17], where a confidence region and uncertainties are introduced, or Improved Prototypical Networks [JCY+20], where weighted prototypes are generated by weighting samples based on their class representativeness.

All these applications and variations of prototypical networks aim to solve classification problems, but for this thesis, we are only marginally interested in the classification of the data. We want to find low-dimensional representations of image data that can be visualized while maintaining the high-dimensional structure inherent to the data. We believe that the metric space learned by prototypical networks can be a useful step to achieve this goal, which is why we try to re-imagine prototypical networks as a form of dimensionality reduction for the visualization of large, unstructured image data.

Besides prototypical networks, other approaches have been developed to address few-shot learning in various domains. A prominent use for few-shot learning are classification tasks. Similar to prototypical networks, Vinyals et al. [VBL+16] describe matching networks for image classification. Yu et al. [YGY+18] classify text based on sentiment from short examples, and Fei-Fei et al. [FFFP06] use few-shot learning to perform object recognition. Few-shot learning can also be utilized for regression problems. For example Finn et al. [FAL17] and Yoon et al. [YKD+18] both use few samples of input and output pairs of a function to estimate the regression function itself.

2.4 Scatter Plot Visualization

Scatter plots are one of the most versatile and commonly used methods to visualize data in a two-dimensional space. They are defined as a plot of two typically orthogonal axes on which an item is displayed as a single point or other mark positioned according to two variables along the two axes [FD05]. The aim is to display the relation of the two variables. Today, scatter plots have long outgrown the simple display of two variables against each other. There are many techniques that enhance the capabilities of the scatter plot by, e.g., including more than two dimensions through shape, size, or color of the marks used to display individual items, or utilizing multiple plots in a scatter plot matrix. With the growing size of data, traditional scatter plots can struggle to effectively display all the relevant information due to overlapping marks. To combat this, numerous adaptations like collecting items in bins [CLNL87] or abstracting the information by blending densely packed areas of the plot into contiguous blobs [MG13].

In this thesis, we choose the scatter plot as our method of visualization, because we are interested in the spacial proximity of individual data items in a two-dimensional space and their class membership. We can use the two-dimensional coordinates we calculate through dimensionality reduction to position these data items on the plot and the color to portray class membership. By aggregating points into density based contour lines, we address the issue of overplotting.

2.5 Representative Sampling

When dealing with very large amounts of data, it is often not easily possible to reasonably present it all. Big data can lead to crowded, uninformative displays where the sheer number of things to display can outnumber the resolution of the screen [LJH13]. This is especially true for our scatter plot visualizations, where the screen space is limited and potentially thousands of marks need to be placed. A straight forward solution to this problem would be to simply not display all of the data at once. But then we need to make the decision of what data to keep and what data to omit. When analyzing a subset of the data, we want it to be representative of the whole data, but random sampling, where every point of data has the same probability of being chosen, may fail to capture key structures and outliers in the data. More informed probabilistic sampling methods like sorting the data and selecting points at regular intervals or sampling from disjoint subgroups of the data can mitigate this effect, but they need prior knowledge of the data. A different strategy is the aggregation of data into predefined bins. Here, data is represented on a plot based on how many data points fall into a certain area. Such binning strategies have the advantage that they account for all the data and can be applied in different resolutions depending on the available visualization space [EF09]. A kind of aggregation strategy that we use in this thesis is when we display the points in our scatter plot as density based contour lines.

One interesting sampling strategy for our thesis is introduced in the paper "Examples are not Enough, Learn to Criticize¡' by Kim et al. [KKK16]. In this paper, they describe a framework they call "MMD-critic", where they (1) calculate a number of prototype samples based on the distribution of all data points and (2) also find additional points in the data space that are not well represented by the chosen samples. These underrepresented samples they call criticism points. The MMD in MMD-critic stands for the statistical measure of maximum mean discrepancy, which measures the similarity between points and potential new samples which are selected to maximize the statistic. To find criticism points they employ a witness function that compares the distribution of the data itself and the chosen samples. Where these distributions least align, a criticism point is identified. We make use of the MMD-critic framework both for finding prototypes in our unsupervised DR method and also for data exploration purposes when identifying areas in our visualizations that are underrepresented by the given prototypes.

CHAPTER 3

Methodology

3.1 Interactive Machine Learning Visualization

Based on typical components from ML and VA pipelines, Sacha et al. [SSZ+17a] have conceived a framework that functions as a basis to describe the steps that are typically taken during the design of a visual IML interface (Figure 3.1). We will follow this framework to explain our approach, and describe what actions we have taken at all of these steps to create an effective IML process.

Edits & Enrichment (A). This first part of the framework is mostly dominated by actions that are directly controlled by the human user. The user is choosing what data to insert into the pipeline and what aspects of the data to consider. Depending on the task, a domain expert may need to sort through the data, group it into classes, assign labels,

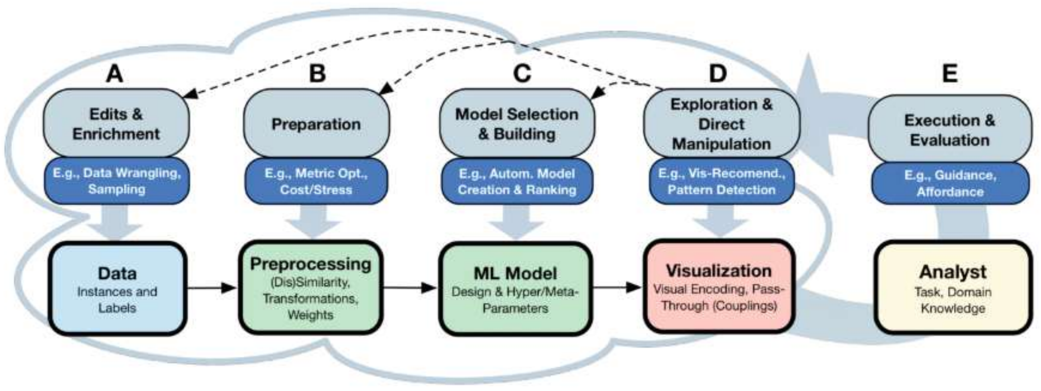

A
Edits & Enrichment
E.g., Data Wrangling, Sampling
B
Preparation
E.g., Metric Opt., Cost/Stress
C
Model Selection & Building
E.g., Autom. Model Creation & Ranking
D
Exploration & Direct Manipulation
E.g., Vis-Recomend., Pattern Detection
E
Execution & Evaluation
E.g., Guidance, Affordance
Data
Instances and Labels
Preprocessing
(Dis)Similarity, Transformations, Weights
ML Model
Design & Hyper/Meta-Parameters
Visualization
Visual Encoding, Pass-Through (Couplings)
Analyst
Task, Domain Knowledge

Figure 3.1: A framework describing an interactive VA/ML pipeline [SSZ+17a].

and create test and training splits for the system to be trained on. This can also include various other actions that directly manipulate the contents of the dataset, like filtering the available data for certain relevant aspects, detecting missing or invalid values and substituting them with valid replacements. As part of an interactive system, the data analyst might want to perform these actions repeatedly over the course of their analysis task. This can be useful when trying to get a new perspective on the data or exploring what-if scenarios.

We have performed similar actions at the start of our visualization process. We have chosen and prepared various datasets of images on the basis that they contain a decently large number of images that are fully labeled with each one belonging to a single ground truth classes. For the purpose of our experiments, labeled training data and unlabeled test data is automatically selected from these datasets. In an actual IML-scenario, the test data would most likely span the entire dataset and the few labeled training samples would be handpicked by the data analyst.

Preparation (B) . The preparation part, according to Sacha et al., includes operations that are universally applied to all items across the dataset. This is in contrast to edits and enrichment activities, where changes can be selectively made only to certain parts of the data. Typical preparation operations that are performed on data before introducing it to a ML model include standardization, scaling, Fourier or wavelet transformation, or weighting of data features.

For our pipeline, we perform only one major transformation of our image data. After initially working with raw image data, i.e. pixel values, we decided to apply a feature extraction algorithm to our data to gain uniform feature vectors. This decision was to the benefit of both quality of output and computational performance. The method we chose to extract representative feature vectors from our images is DINO, a self-supervised approach introduced by Caron et al.[CTM+21] at Facebook AI research. DINO is a self-supervised approach which stands for knowledge **di**stillation with **no** labels. DINO works by feeding input to two identical networks with different parameters, one serving as a teacher and the other as a student network. Both these networks receive different transformations of the same input image and the goal is for the student to predict the output of the teacher. The similarity of the outputs is measured with cross-entropy loss and the teacher parameters are updated with an exponential moving average of the student parameters. The DINO architecture works for both Vision Transformer and Convolutional Nets. Since the student network is trained to predict the teacher's output based on different views of the same images, and the teacher is updated based on the student parameters, both networks learn to make representations that focus on the parts of the images that make up the prevalent objects in the image. This produces vectors of uniform length that represent attention maps that depict the most prominent features in an image. Since the images in our datasets are fixed and will not be modified throughout all our experiments, we performed this transformation of the images only once in advance for all the datasets and saved the resulting representations to be used as the input for our system.

Model Selection & Building (C). In a typical IML setup, domain experts would at this point be able to interact with the core aspects of the model that they use. This ranges from choosing different algorithms to process the data, to directly manipulating model (hyper-)parameters. For the process followed in this thesis, there is no direct interaction with the ML algorithms. The available models and algorithms are, however, thoroughly tested with a variety of parameterizations throughout all our experiments in order to find configurations that produce the best results on our datasets. We explore prototypical networks as the main target for our research and compare them to both unsupervised and semi-supervised UMAP [MHM18] and LDA [BG98]. We also briefly experiment with PCA [JC16]. These are all methods for computing a 2D representation of high-dimensional data with the aim of preserving neighborhood structures. We chose UMAP because it is de facto the state-of-the-art when it comes to DR. It uses similar techniques to previous well-performing algorithms like t-distributed stochastic neighbor embedding (t-SNE) [VdMH08]. They both compute a graph in high-dimensional space and optimize a low-dimensional layout that is as structurally similar as possible. The main advantage that UMAP presents over t-SNE is computation time, as well as a stronger emphasis on separating global structures. The unsupervised version of this algorithm operates on completely unlabeled data, whereas the semi-supervised version allows for some part of the data to be labeled while the rest remains unlabeled. This also resembles few-shot learning, the task that we use prototypical networks for, where we operate on few labeled examples to obtain an embedding that groups similar data. The other algorithm that has been more closely investigated is LDA, a technique that is designed to maximize the ratio of between-class variance to the within-class variance, resulting in maximal separability [BG98]. Since this is not a graph based technique, but one that operates on matrices, it is expected to run much faster, but our experiments will show that the outcome will show significantly worse separability.

Exploration & Direct Manipulation (D). At this point in the pipeline, the ML systems have performed their tasks and the results need to be displayed to the user. For this, there are numerous known visualization techniques for any sort of data format. The aim is to allow the user to explore the data with various interaction techniques, such as selecting, zooming, panning, or changing views. But the transformed data is far from the only interesting thing that can be visualized. Depending on their expertise, the user might be interested to see the inner workings of the ML pipeline. In an interactive system, where model characteristics can be iteratively changed, visualizing model parameters and structures can help identify points where promising changes could be made.

In our visualization, we focus primarily on the aspects that convey information about the data. As we are mostly interested in the separation of classes, the main body of our interface is the area for displaying the color-coded data points. Users have the choice between two types of visualization: Either a scatter plot, where all data points are rendered in their class-specific color (Figure 3.2a), or a representation of the data using colored contour lines that show the distribution and density of the data (Figure 3.2b). In both cases, the class prototypes that were calculated are highlighted. Users can hover prototypes to view the images that lie closest to these prototypes. In addition, users are able to request any number of so-called criticism points. These are data points that are poorly represented by the distribution of class prototypes and therefore most likely candidates for misclassified outliers or probable locations for additional classes that were not initially considered. Changes the user can make to influence the next iteration of the system are as follows: Choose the dataset to analyse, select the method with which to calculate the data distribution (options are ProtoNet or unsupervised UMAP), pick the number of classes that the dataset should be separated into, and select the number of labeled samples that are available during training.

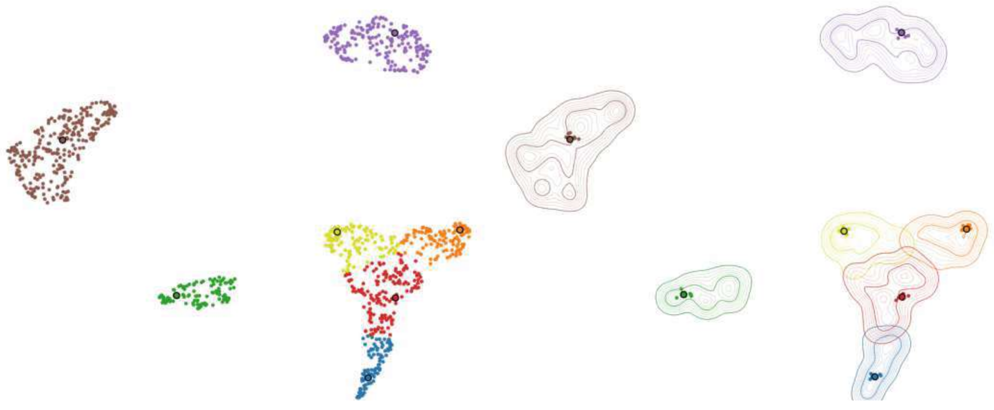


Figure 3.2: Visualization of PACS Photo dataset [LYSH17], using (a) all data points and (b) contour lines.

Execution & Evaluation (E). This stage of the pipeline is somewhat separated from the rest. It describes the overarching engagement of the user with the rest of the pipeline. The user is now able to observe and interpret the results, evaluate their findings, and execute interactions. What happens at this stage is very dependent on the task the user intends to achieve. In the case of our system, this is the point where we interpret the quality of the created visualizations by comparing the metrics we calculate for the different dimensionality reduction techniques. Over the course of our experiments, we repeatedly execute and adjust the steps in our pipeline, measure results, and interpret the findings in order to answer our research questions.

3.2 Measurements

To be able to evaluate the quality of the scatter plot layouts produced by the different methods we applied, we need to calculate some metrics to describe the differences and argue why one method might produce better results than the others. The main quality that we are looking for is the clean spatial separability of clusters containing items belonging to the same ground truth class. We also want visualizations that intuitively highlight links and correlations between classes, as well as class outliers. The less the clusters overlap and the less items from different classes are mixed, the more the visualization represents the actual differences that can be observed in the data. All these factors coalesce in our first research question (R1), which aims to find out how the interpretation of unstructured data differs between projections based on different dimensionality reduction techniques.

3.2.1 Distance Consistency

One very popular measure to quantify separability in a 2D scatter plot is distance consistency (DSC) introduced by Sips et al. [SNLH09]. They initially conceived this measure to rate and compare different 2D views of high-dimensional data where they would calculate this measure for visualizations of pairs of attributes. They describe such a view as consistent with the structure of the data if the m classes in the data are mapped to regions in the scatter plot that are visually separable. To measure this separability, they calculate Centroid Distance (CD). This denotes, whether the distance of each data point to the centroid of all data points belonging to the same class is minimal compared to all other class centroids. DSC is then defined as the classification error across all data points, meaning the percentage of all data points for which CD is not true. Sips et al. normalize their scores across all combinations of attributes to a scale of 0 to 100 to reach better interpretability. In a similar fashion, we calculate DSC for our single 2D representation of the data but use the inverse score, i.e. the percentage of data points that are closest to their own cluster center, to reach a score where 100 means perfect separability and 0 is no separability at all. Through experiments of their own, Sips et al. were able to show that scatter plots that were rated high by DSC were the same plots that human observers picked to be well separated. Since DSC can achieve this good correlation with human interpretation, we use it to identify the differences in human interpretability between the different techniques we are investigating.

3.2.2 Gamma Observable Neighborhood

With the explicit goal to build better visual separation measures that better resemble the human perception of color-coded scatter plots, Aupetit and Sedlmair [AS16] have systematically generated 2002 new separation measures by varying and parameterizing (1) neighborhood definitions and (2) class-purity functions. They reach the large number of 2002 new measures by combining a total of 17 different types of neighborhood graphs (increased to 143 through parameterization) and 14 class-purity functions. All these measures were quantitatively evaluated using a previously proposed Machine Learning framework [SA15] that can quantify how well a measure predicts human judgement of class separability. Out of all their new proposed measures, they found that 1170 (58.4%) outperformed DSC in their evaluation framework. The one that performed the best overall is *GONG 0.35 DIR CPT* being rated 11.7% better than DSC. This code name breaks down to a directed (DIR) Gamma-Observable Neighborhood Graph (GONG) where γ is 0.35 as the neighborhood definition and the average Class-Proportion in the

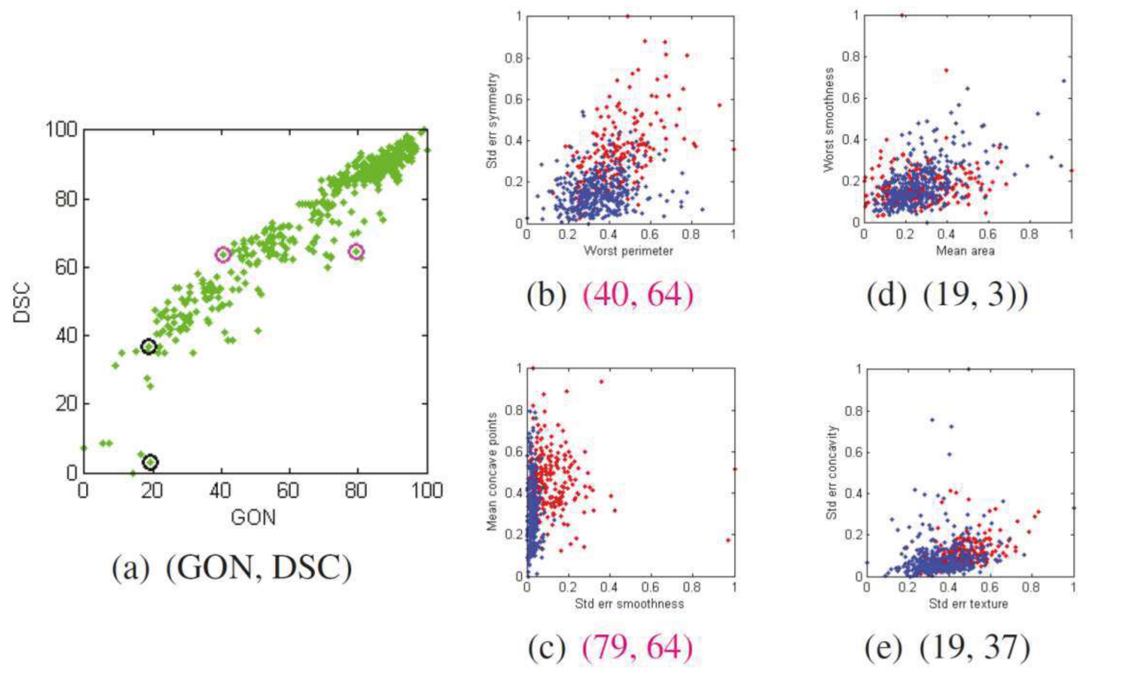


Figure 3.3: (a) GON and DSC scores for all pairs of Wisconsin Cancer data scatter plots. Scatter plots with DSC fixed and low (b) and high (c) GON values (magenta circles in (a)). Scatter plots with GON fixed and low (d) and high (e) DSC values (black circles in (a)) [AS16].

Target class (CPT) as the class-purity function. That is, what proportion of neighbors to any one data point belong to the same class as the point itself. Aupetit and Sedlmair showed in their evaluation that *GONG 0.35 DIR CPT* (short GON) scores mostly correlate with DSC. But they can also show cases where the scores deviate significantly (Figure 3.3). These cases show that GON tends to better represent a good separability to the human eye. We decided to calculate both DSC and GON scores in our experiments to gain a secondary judgement on all our results. When both scores closely resemble each other, then the separability measure should be quite accurate. But when they noticeably deviate, then one may be a misguided outlier.

3.2.3 Computation Time

The final performance indicator that we use to evaluate the different methods of producing scatter plots is the time expended to compute the 2D embeddings of the high-dimensional data (R2). While this does not say anything about the quality of the resulting scatter plots, it is arguably of greater importance in regard to the practical usability of these methods. Since the eventual goal of this research is to evaluate whether these methods are suitable for an interactive data exploration process (R4), a minimal computation cost is integral to their success. It has long been established that any application should take at most one second for user input to produce visible change [Nie93] in order to feel interactive and not disrupt the user's flow of thought. This means we want to achieve a minimal delay between the user's request to re-train the model and the change of the visual output.

3.3 Datasets

Over the course of our experiments, we use several different datasets to test our methods on. We perform the same experiments on different datasets, because we need to show that any particularly good or bad results can actually be attributed to the methods used, and are not just products of surprisingly well fitting input data. The datasets used in our experiments are as follows:

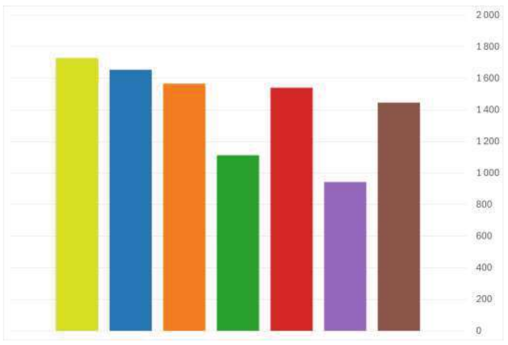
PACS. This dataset was originally designed for domain generalization problems [LYSH17]. It contains images of seven different classes, each spread across four domains, i.e. different styles of images. These domains are real photos, art paintings, cartoons, and sketches. The seven classes in this dataset are dogs, elephants, giraffes, guitars, horses, houses, and persons. In total, the PACS dataset contains 9,991 images and the class and style distributions can be seen in Figure 3.4. Due to the variety of styles in this dataset, we can use it in several different ways throughout our experiments to create challenges of varied difficulty for the different embedding methods. For some experiments we use only certain subsets of the data, like the real photos or only the cartoons. We also perform experiments on the whole dataset where we either try to separate the images by domain, leading to a four class problem, or as the most challenging test separating the whole dataset into its seven classes while all four domains are included.

Imagenette. This is a rather small subset of the usually enormous ImageNet dataset [DDS$^+$09]. Imagenette was constructed by Jeremy Howard [H$^+$20] for the purpose of quickly testing algorithm ideas without the usually long computation time when running them on the whole ImageNet. It consists of 10 easily classified classes from ImageNet and contains a total of 9,469 images. The classes include tench, English springer, cassette player, chain saw, church, french horn, garbage truck, gas pump, golf ball, and parachute (Figure 3.5). Among all our datasets, this is the one with the most even class balance.

CelebHair. CelebHair is a subset of the much larger CelebA dataset [LLWT15]. This is a large-scale dataset with over 200 thousand images of celebrity faces, each with 40 attribute annotations. For the CelebHair subset, we only picked images that had the attributes for either black hair, brown hair, blond hair, or gray hair. For the sake of computation time when loading the dataset during our experiments, and because we always picked a uniform set of test data (size 1,000) for our experiments, we decided to further reduce the dataset size by using only 10% of the available data. This results in a dataset of 9848 unique images, each belonging to one of four classes. We decided to keep the original class proportions, leading to the gray hair class having notably less members

Color	Name	#
🟨	Dog	1729
🟦	Elephant	1654
🟧	Giraffe	1566
🟩	Horse	1113
🟥	Guitar	1540
🟪	House	943
🟫	Person	1446

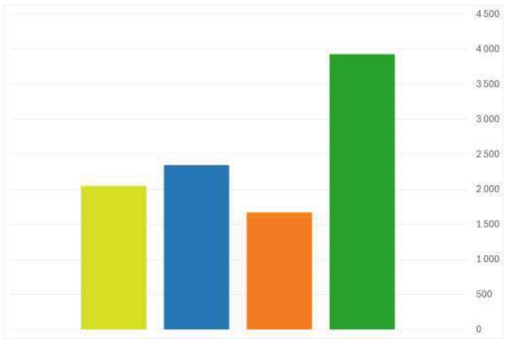
(a)



(b)

Color	Name	#
🟨	Art	2048
🟦	Cartoon	2344
🟧	Photo	1670
🟩	Sketch	3929

(c)



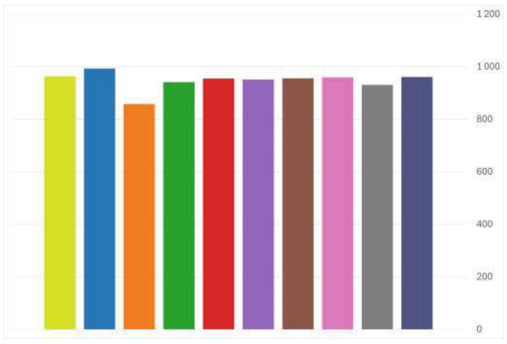
(d)

Figure 3.4: Ground truth class distribution of the PACS dataset by classes (top) and styles (bottom).

compared to the other three (Figure 3.6). Due to the high similarity of all images in this dataset, even between classes (all show human faces), it poses a much more challenging problem for all the embedding methods.

Color	Name	#
🟨	Tench	963
🟦	English Springer	993
🟧	Cassette Player	858
🟩	Chainsaw	941
🟥	Church	955
🟪	French Horn	951
🟫	Garbage Truck	956
🟪	Gas Pump	960
⬜	Golf Ball	931
🟦	Parachute	961

(a)

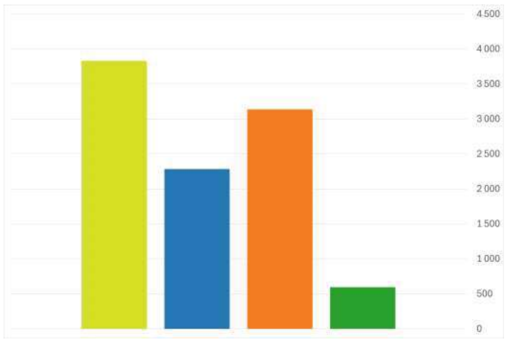


(b)

Figure 3.5: Ground truth class distribution of the Imagenette dataset.

Color	Name	#
🟨	Black	3829
🟦	Blond	2285
🟧	Brown	3138
🟩	Grey	596

(a)



(b)

Figure 3.6: Ground truth class distribution of the CelebHair dataset.

CHAPTER 4

Implementation

The prototype application we designed to present and explore the findings of this thesis is a simple Flask application written in Python 3.10. It exposes a REST API where requests from the frontend user interface are received and processed before returning the relevant data for the visualization. In this chapter, we describe the technical details on how we process our data, how the different embedding methods work, and how the results are presented in the user interface.

4.1 Dataset Preparation

As our experiments will show, using actual image data is not a feasible way to implement any of our methods. Before everything else, we need to apply one step of pre-processing to all the images in all our datasets. We convert them from image files to a much smaller representation of themselves, a vector of values that describes the scene layout and objects contained in the image. To do this, we use the DINO method described in Section 3.1. We make use of a pre-trained model available on the Pytorch Hub model repository. This model applies DINO on a Resnet-50 architecture and we let it process all the images in a dataset. The resulting vectors are converted to numpy arrays and directly saved into a npz file along with the target label and the filepath to the respective images.

4.2 Loading Datasets

Upon receiving a load request for one of the available datasets, our backend application will use the numpy load method to load the npz files containing the arrays for feature vectors, labels, and image paths from the data folder. Afterwards, a sample of 1,000 data points is picked from the dataset to serve as the unlabeled test data. This test data is picked at random, ideally containing data from all ground truth classes and representing the class balance of the dataset. Since this application imitates IML for controlled experiments, labeled training data is automatically selected for a given number of test classes. The number of test classes and the number of samples per class that will be picked is included in the dataset request received from the user interface. In a real data exploration scenario, labeled training data would be manually selected (Figure 4.1). For each class, up the the requested number, the right amount of data points is picked from the dataset at random. If there are fewer test classes than there are ground truth classes, we pick the first m classes as test classes from the set of n ground truth classes, resulting in classes in the test data that are not represented in training. If there are more test classes requested than there are ground truth classes ($m > n$), classes that already have training samples picked will be picked again to create new training classes. This results in multiple artificial classes during training that are actually a single class in the test data. The influence of both these scenarios will be investigated in our experiments. In the end, the training and test data is stored in an internal data storage, from where it can be used to calculate both supervised and unsupervised embeddings without the need to reload the data with every embedding request.

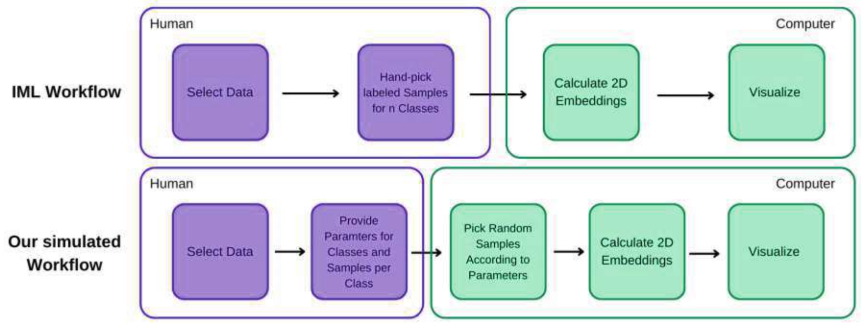


Figure 4.1: Actual IML workflow compared to our simulated one with regard to sample picking.

4.3 ProtoNet

The core technology applied in this thesis is the ProtoNet neural network used to calculate low-dimensional embeddings of high-dimensional feature vectors representing images. Most parts of the ProtoNet architecture, such as the training process and especially the loss function, are directly inspired by the original prototypical network implementation by Snell et al. [SSZ17b]. The model itself, however, needed to be adjusted to handle the new type of input. Where prototypical networks use a number of convolutional layers to reduce square images to a single vector of feature values, we feed ProtoNet with already one dimensional vectors of size 2048. Instead of using four convolution blocks, each consisting of 2D convolution, batch normalization, an activation function, and max pooling, our ProtoNet architecture consists of three blocks of linear transformation (Figure 4.2). These three blocks gradually reduce the size of our input vectors from 2048 to 512, 256, and finally 64, which is the same size of output that Snell et al. produced in their paper. We use ReLU as our activation function after each linear transformation. All ProtoNet operations are performed on a graphics card using CUDA.

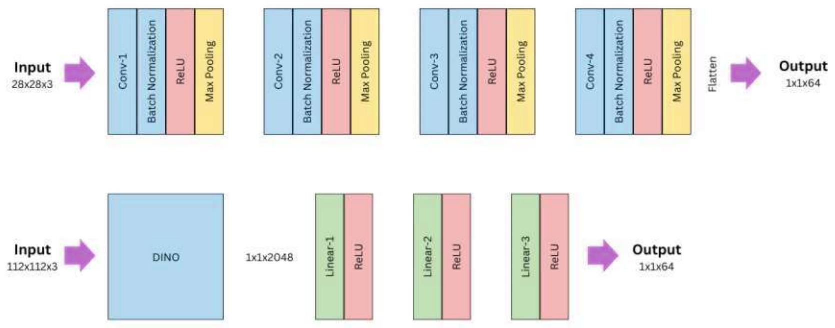


Figure 4.2: Architecture of a prototypical network by Snell et al. [SSZ17b] (top) and our ProtoNet method (bottom).

Training the Model

For the training progression of our model, we use the Adam optimizer with an initial learning rate of 0.001. We run five epochs of training, decreasing the learning rate after each one. Each training epoch consists of ten episodes, where we first shuffle the training data and then perform a loss calculation. At the end of each episode, the loss is propagated backwards and the optimizer takes a step forward. For each new loss calculation, the available training samples per class are shuffled and then split into two sets. 40% of samples serve as the support set for this training iteration and the remaining 60% make up what is called the query set. After all training samples, support and query sets, are embedded in the 64-dimensional space by the model, the samples in the support set are used to generate a prototype for each class by calculating the mean over all support samples belonging to that class. We then calculate the Euclidean distance between the samples in the query set and the prototypes. The loss for this training iteration is calculated by applying a logarithmic softmax over these distances.

Embedding the data

After the training of the model is complete, we use it to generate vectors of length 64 for all the available data, both labeled and unlabeled. The means of those data points that were labeled training samples represent the “prototype” of each class. Labels for the large amount of previously unlabeled data are predicted by assigning them the label of the closest prototype. For the final step of the ProtoNet pipeline, we have to migrate the data from the GPU back to the CPU to use unsupervised UMAP to reduce the dimension of all the prototypes and data points from 64 to 2 so that they can be visualized in a 2D scatter plot. While there are GPU based versions of UMAP, they are only available for Linux systems which clashes with the rest of our Windows-based application.

4.4 Unsupervised Embedding

When using our unsupervised method to create 2D embeddings for the chosen dataset, we use unsupervised UMAP to reduce each DINO feature representation of the images from 2,048 dimensions to only 2. Since UMAP only embeds the data points in two dimensions without classifying them, we use a k nearest neighbor classifier, trained on the training samples that were automatically chosen (See Section 4.2), to predict labels for all data points so that we can observe a predicted class structure in the visualization.

To be able to determine class prototypes in an unsupervised fashion, we take inspiration from the MMD-critic framework [KKK16], which allows us to pick representative samples from each class based on the point distribution. They use squared maximum mean discrepancy to determine which points in the data best represent the data distribution. We use this to determine a single prototype for each class by comparing the average proximity of all data points in that class to the average proximity of each individual point to all the others. The point where these two distributions are closest together is chosen as the class prototype.

4.5 Other Semi-Supervised Embeddings

Semi-supervised embedding methods that are examined over the course of our experiments are semi-supervised UMAP and LDA. Semi-supervised UMAP, much like the unsupervised variant, tries to learn a low-dimensional manifold of the data based on topological structure. But this time, we provide the algorithm with target labels for the chosen training data. Combined with the unlabeled rest of the test data, UMAP then performs its embedding task with regard to the known class information of the provided samples. This expectedly takes more time than using UMAP with no supervision, but as our experiments will show, we find little to no improvement in terms of class separability of the output with the few samples we provide.

When creating 2D embeddings of the data using LDA, we use the scikit-learn implementation of the algorithm. We fit the model on the labeled training samples before using it to transform the unlabeled test data. Since we operate on vectors of size 2,048, we use the "svd" (single value decomposition) solver of the algorithm, which is recommended for data with a large number of features. Both LDA and semi-supervised UMAP are only used during the experiments and are not available in the simulated IML application because of poor performance shown in the experiments.

4.6 Visualization

We visualize the calculated 2D embeddings via the frontend part of our Flask application. This is a simple HTML page where parameters and datasets can be selected, requests are sent to the backend, and the received data is presented using the JavaScript library d3. Users can select one of the available datasets from a dropdown menu, which sends a request to the backend to load the respective dataset into memory. Since this application is a an experimental platform and not a fully functional IML framework, the labeled training samples are not manually picked but rather automatically picked at random. The number of test classes to pick from and the number of training samples per class can be specified in input fields. The number of test classes can match the number of ground truth classes in the selected dataset, but training can also be run with fewer or more test classes than ground truth classes actually present in the dataset. Choosing more test classes than the dataset contains leads to two or more sets of samples from the same class being presented to the ProtoNet as belonging to different ones, resulting in split classes.

Two buttons allow the user to choose the DR method to apply, with the choice being between ProtoNet or unsupervised. Other semi-supervised methods are not included in the final visualization because of their inferior performance in the experiments (Chapter 5). Regardless of which, clicking these buttons triggers the backend request for the embedded data points. Upon completion of this request, there are three arrays received in JSON notation: 1) the 2D positions and lables of the prototypes, 2) the 2D positions and labels of all other data points, and 3) filepaths to the images of those points closest to each prototype. We choose to use these images of closest points, because the prototypes generated through ProtoNet are artificial points in the embedding space that are the result of averaging the training data. There are no actual images in the dataset that exactly match with the them, so showing a number of closest images is a good approximation.

The actual visualization of the dataset is then performed in the form of a scatter plot, where each data point and prototype is positioned using its 2D coordinates. We use the d3.js framework to manipulate SVG elements into the required shape. There are several options available for this visualization: Two regarding the form in which the data is presented and two regarding the labels of the data points. The visualization can either be viewed as a scatter plot of points, where each data point is printed in the color of its label, or the clusters can be shown through contour lines that indicate cluster densities. The color of the data in the plot is controlled by the label that is assigned to each point. We can either show the labels of the test classes predicted by the algorithms, or – for validation purposes – the ground truth labels. Regardless of the type of visualization, the class prototypes are always rendered as larger points with black borders. Hovering the cursor over one such prototype will load the images from the filepaths that were received from the backend request and display those images representing the data points closest to the respective prototype next to the plot.

4.7 Sampling for Data Exploration

For the purpose of data exploration, we investigate three different methods to identify criticism points. By calculating any number of criticism points (freely selectable via numeric input) we aim to find areas in the data, where possibly interesting new discoveries might be found. The first and simplest method is plain random sampling. This marks criticism points with no reasonable decision making involved. The points can fall anywhere on the scatter plot. We can also calculate criticisms based on the largest absolute distance in the 2D embedding space of the data points to all the available class prototypes. This makes it so that criticism points are chosen to be the furthest away from the prototypes, which could mean an area between prototypes where another class could be hidden, but also might hit far outliers. And finally, we make use of the second part of the MMD-critic framework. This is done by comparing the distribution of prototypes to the distribution of the rest of the data. More precisely, the average proximity of each point to all other points in the data is compared with its average proximity to all of the prototypes. The point with the largest absolute difference between these two proximity measures is chosen as a criticism point, meaning that it lies in an area that is not well covered by the distribution of prototypes. Compared the the distance based method, this is more likely to place criticism points in more densely populated areas of the scatter plot. When generating more than one criticism point, we apply a regularizer term to the calculation that takes previously selected criticism points into consideration in order to avoid all the criticism point falling into the same underrepresented area. Criticism points are shown as black dots in the visualization. These can be hovered with the cursor to show the corresponding image besides the scatter plot. By choosing a large number of criticism points, the whole data space gets evenly sampled, making a well represented overview of the data possible.

4.8 Evaluation Metrics

In order to evaluate the separation quality of the plots produced by the different methods, we make use of two separation metrics: DSC and GON. As described previously in Chapter 3.2, DSC is calculated as the percentage of data points where the closest prototype is of the same class as the data point itself. All that needs to be computed is the distance between each data point and all projected prototypes. We then evaluate whether the prototype with the minimal distance is of the data point's class.

GON on the other hand does not compare data points and prototypes, but data points among each other in much more localized neighborhoods. We first need to generate a graph that determines which points in the data are neighboring each other. The Gamma Observable Neighborhood graph is constructed by connecting each point x to any other point p if an intermediary point p_i positioned at the γ (0.35) mark between them has p as its closest neighbor. We then look at all the neighbors of each point and determine the percentage of neighbors that belong to the same class as the observed point. Averaging over all points in the dataset results in the GON score for the dataset.

CHAPTER 5

Experiments and Results

In this chapter, we describe the various experiments that we conducted to determine how well ProtoNet works in comparison to the other methods. We show the advantage of using feature representations instead of actual images, determine the best parameters for our pipeline, and assess the quality of the outputs that the different methods produce. All these results come from experiments performed in a Jupyter notebook.

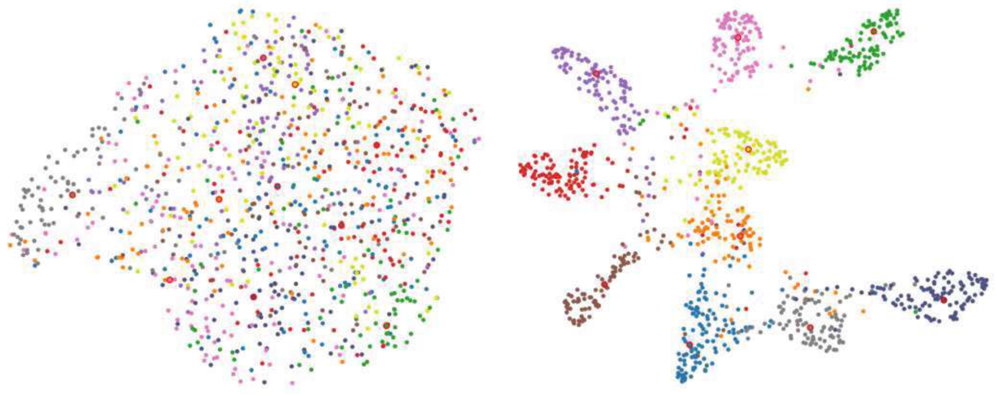
5.1 Images vs. Feature Representation

The original implementation of prototypical networks by Snell et al. [SSZ17b] makes use of a convolutional neural network (CNN) to embed images in a lower dimensional embedding space. They managed to achieve great results for tests on the Omniglot dataset [LSGT11], where similar images would cluster around the class prototype that was calculated during training. This dataset, however, consists of only small grayscale images of hand-drawn characters measuring 28×28 pixel. Where the prototypical network classifier achieved a classification accuracy of 98.8% on a 5-class problem on the Omniglot dataset, a similar experiment on the *mini*ImageNet dataset (84×84 color images) [VBL+16] yielded only 68.2% accuracy.

In the beginning, we performed our experiments with the same kind of simple images, using both the Omniglot dataset and the MNIST dataset of handwritten digits [LBBH98]. Our goal, however, was to use prototypical networks to create embedding spaces for real-world image data; this means embedding larger, more complex, colored images. Since we could observe similar drops in performance to the ones observed by Snell et al. when training our ProtoNet on the CelebHair dataset, we decided to try a different approach of presenting our images to the network.

Preprocessing all the images in our datasets using the self-supervised DINO method [CTM+21], we gain a feature vector of length 2048 for each image that encodes the scene layout and object boundaries. Using these feature vectors as input for the prototypical network (now using linear transformation instead of convolution), results vary from only slightly improving our measuring scores, to making a difference of almost 60 points in DSC depending on the dataset.

The least improvement could be measured on the CelebHair dataset. There, the convolution based ProtoNet and the variant using DINO features scored almost equally well. Much bigger improvements were measured on the PACS photo dataset. There, DSC and GON both reached average scores above 90 with DINO features, while image convolution only measured results around 50. The biggest overall improvement could be observed on the Imagenette dataset. Image convolution was not able to produce any sensible separation, while using DINO features led to decent results scoring around 80 points in both DSC and GON metrics. This stark difference can be seen in Figure 5.1. All average measurements are depicted in Figure 5.2.



(a) (b)

Figure 5.1: Comparison of ProtoNet + UMAP embedding of Imagenette dataset using (a) image convolution and (b) DINO feature vectors.

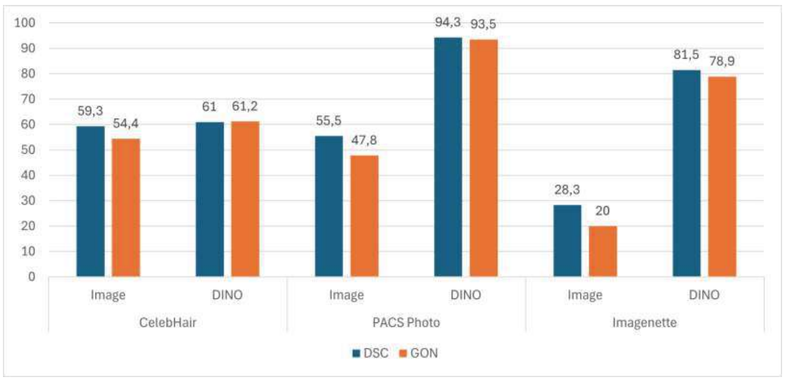


Figure 5.2: Average DSC and GON scores on different datasets when ProtoNet input is Images vs. DINO feature vectors.

One way we test to improve the performance of the image convolution method is artificially increasing the variation in labeled training samples by randomly applying some new image augmentation to the training samples each iteration of the training. These augmentations include horizontally flipping the images, slightly changing brightness, contrast, or saturation, and converting the image to grayscale. This, however, does not noticeably alter the outcome. Separation metrics, as well as the visual output are similar to runs of the image convolution ProtoNet without data augmentation. All that can be measured is that it takes almost triple the time to train the model.

5.2 UMAP Parameter Variation

Uniform Manifold Approximation and Projection is used both as a control method and as a key element of our ProtoNet pipeline. This dimension reduction technique presents a handful of hyperparameters that can slightly influence the outcome of our tests. In order to achieve the best possible results and to create an environment of fair comparison conditions, we experiment with variations of these parameters, namely the number of neighbors and the minimum distance. The number of neighbors has influence over how much emphasis UMAP places on local versus global structures in the data. Using very small values narrows the view UMAP has on the data which can result in smaller chains of points and disconnected components, while using very large values leads to global structures being well captured. There is a trade-off between accurate local proximity of similar data points and the display of overarching structure contained in the data. The minimum distance parameter imposes a limitation on how tightly UMAP is allowed to pack data points. We do not touch the number of components, as this is the number of dimensions the final embedding should have, which in our case is always two. We also always use the 'Euclidean' metric, as Euclidean distance is also the metric that ProtoNet uses in its training. For this set of experiments, we use the PACS Photo and Imagenette datasets. We first test the variation in number of neighbors while keeping the minimum distance at UMAP's devault value of 0.1. We embed three different sets of 1,000 test data points with the number of neighbors parameter taking the values of 5, 15, 50, and 200. Because UMAP is nondeterministic, we run each set of test data three times, resulting in a total of nine measurements for each configuration of which the resulting averages can be seen in Table 5.1. For any further experiments we use a value of 25 for this parameter. This is higher than UMAP's default of 15, but not so large as to include too big a portion of our dataset.

Similarly, we test the influence of the minimum distance parameter. For this set of tests, we fix the number of neighbors to 25, since the previous tests suggest that the optimal number for these two datasets lies somewhere between 15 and 50. The minimum distance is then varied between the values 0.1, 0.3, and 0.5. We once again run three different sets of 1,000 data points three times each to take into account the nondeterministic nature of UMAP. The average scores of the nine test runs can be seen in table 5.2. These tests show that the minimum distance parameter has little to no influence on the cluster separation in terms of metrics. The DSC scores for both datasets lie within a range of

Testing no. neighbors				
PACS Photo	5	15	50	200
DSC	91.15	92.35	**93.56**	93.13
GON	92.62	93.17	**93.51**	92.50
Imagenette	5	15	50	200
DSC	73.60	80.14	**83.42**	79.76
GON	84.52	**84.02**	83.75	80.33

Table 5.1: Influence of number of neighbors in UMAP.

variation of only 0.65 which can easily be due to natural fluctuation. The reason why the GON score might be higher with a lower minimum distance, like with the Imagenette dataset in this test, is that more points are included as immediate neighbors of points in the neighborhood graph when they are closer together. This can lead to higher scores when there are only few impurities in the class cluster. Since the metrics show little difference, we use visual comparison to determine a good value for further experiments. When comparing the scatter plots, we can see that 0.1 places points too close together, resulting in over-plotting, while 0.5 spreads them farther apart than they need to be. We choose to set the minimum distance to 0.3, as this value lies in between and shows the best results.

Testing minimum distance			
PACS Photo	0.1	0.3	0.5
DSC	93.93	93.46	**93.97**
GON	93.24	**93.48**	93.03
Imagenette	0.1	0.3	0.5
DSC	78.98	**79.01**	78.36
GON	**84.56**	82.55	80.81

Table 5.2: Influence of minimum distance in UMAP.

5.3 Number of Training Samples

One of the main aspirations of this thesis is to find a system that is able to accurately cluster large amounts of data without the need to provide exorbitant amounts of labeled training samples. Therefore, it is crucial to explore how much labeled data is actually required to produce results with decently high separability scores. We have created ProtoNet embeddings for each of our three main datasets while using either around five, fifteen, or thirty labeled training samples per class. We introduce slight variation by fluctuating the actual number of samples between -1 and +2 of the given base amount for each class. For each of these base values, the ProtoNet model is trained ten different times in order to gain average scores that show the changes in separation quality that come with the increase of training samples. Table 5.3 shows the results of these test, where we can see that, for all datasets, increasing the number of labeled training data leads to better separation metrics. This comes at no surprise, as more training data should always result in a more accurate model.

Testing number of labeled samples			
CelebHair	5	15	30
DSC	46.59	57.86	**<u>64.90</u>**
GON	47.03	57.04	**<u>64.07</u>**
PACS Photo	5	15	30
DSC	84.78	92.68	**<u>95.39</u>**
GON	86.67	92.47	**<u>94.38</u>**
Imagenette	5	15	30
DSC	67.16	79.68	**<u>85.73</u>**
GON	69.81	78.97	**<u>83.19</u>**

Table 5.3: Influence of the number of training samples on the separation quality metrics for ProtoNet.

In order to assess whether the number of training samples can lead to a noticeable performance vs. quality trade-off we also look at how much an increase in training samples affects the computation time of methods that involve any degree of training. Here, we investigate ProtoNet training, semi-supervised UMAP, and LDA. Since the visual separation quality of the result is of no consequence in this case, we can perform all tests using a single dataset. We describe our observations based on a test set of 1,000 images from the Imagenette dataset with ten classes and varying sizes of training data. The resulting computation times are tabulated in Table 5.4 and also illustrated in Figure 5.3. The values depicted are averages measured across ten different runs on the same dataset. Tests across 5, 15, 30, 50, 75 and 100 labeled samples per class show that ProtoNet training has a constant computation time of around 1,000 ms regardless of the number of training samples. In comparison, the time needed to calculate a semi-supervised UMAP embedding grows considerably. We can see the actual influence the labeled samples have on the UMAP algorithm by comparing these measurements to unsupervised UMAP, which takes around 4,450 ms to embed this set of test data. With LDA, we can observe a linear scaling relating number of training samples to an increasing computation time. From five to one hundred samples per class, the time LDA needs to calculate the embedding increases from around 15 ms to almost 400 ms.

Since ProtoNet shows to have no real cost relating to the number of labeled samples and the quality of the output will inevitably increase with regards to it, we choose to use 25 labeled samples per class for all further tests, as this seems to be a reasonably large number that could still be selected in an IML workflow.

Computation Times in ms						
Samples per class	5	15	30	50	75	100
ProtoNet training	942	1,059	1,131	1,098	1,052	1,137
unsup. UMAP	4,416	4,457	4,562	4,413	4,436	4,451
semi-sup. UMAP	5,026	5,271	6,294	6,846	7,806	8,785
LDA	15.2	48.5	115.5	197.6	306.4	389.8

Table 5.4: Influence of number of training samples for ten classes on computation time.

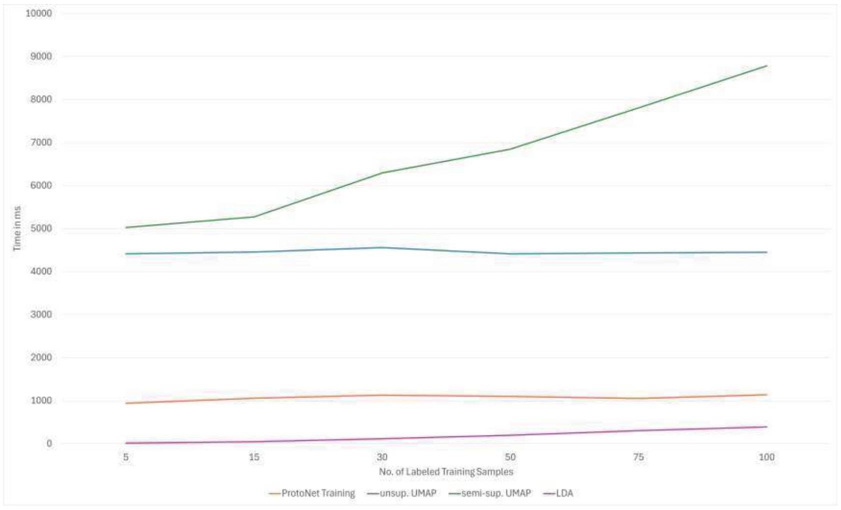


Figure 5.3: Average computation time with varying numbers of labeled training samples based on 1,000 data points from the Imagenette dataset.

5.4 Embedding Methods

After disregarding the method of using image convolution in the ProtoNet approach, we settled on mainly comparing ProtoNet to two other standard methods of dimension reduction. These methods are UMAP and LDA. With UMAP, we tested the fully unsupervised method as a base line and investigated how the results were influenced when a number of samples were provided with their respective labels in a semi-supervised fashion. We found that introducing a small number of labeled samples to UMAP (always the same samples that were given during ProtoNet training) makes little to no difference compared to the unsupervised method. Any variations in DSC and GON scores between the two methods can be attributed to the natural variance that comes with the nondeterministic nature of UMAP. LDA, on the other hand, is an interesting case. For the most part, it behaved as would be expected, with its main advantage being much faster computation time with a trade-off in terms of separation quality. LDA was only able to produce results that came somewhat near the metrics that were produced by ProtoNet on one of the evaluated datasets. That dataset is the CelebHair dataset, which proved to be one with which UMAP especially struggled. This fact also impacted the performance of ProtoNet on this dataset, as the last step in the Protonet pipeline utilizes UMAP. The final dimension reduction using UMAP is also the main bottleneck for ProtoNet when it comes to computational performance. For this reason, we tried to substitute this UMAP reduction with PCA in the ProtoNet pipeline. What we found is that PCA is significantly faster than UMAP. However, the resulting class separation ranged from being quite similar in terms of separation metrics to being noticeably worse compared to a ProtoNet that uses UMAP. Figures 5.4 and 5.5 illustrate this difference.

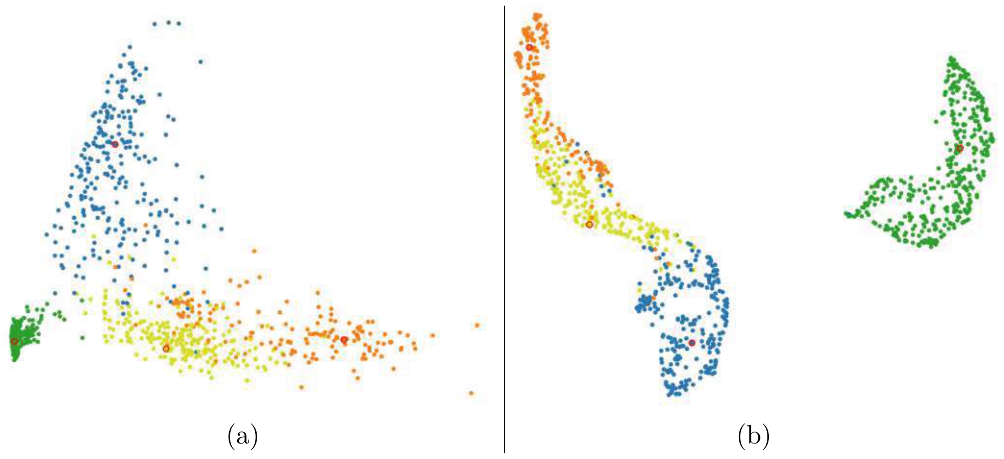


Figure 5.4: PACS Styles: Comparison between ProtoNet with (a) PCA and (b) UMAP as final reduction method. They have similar separation metrics.

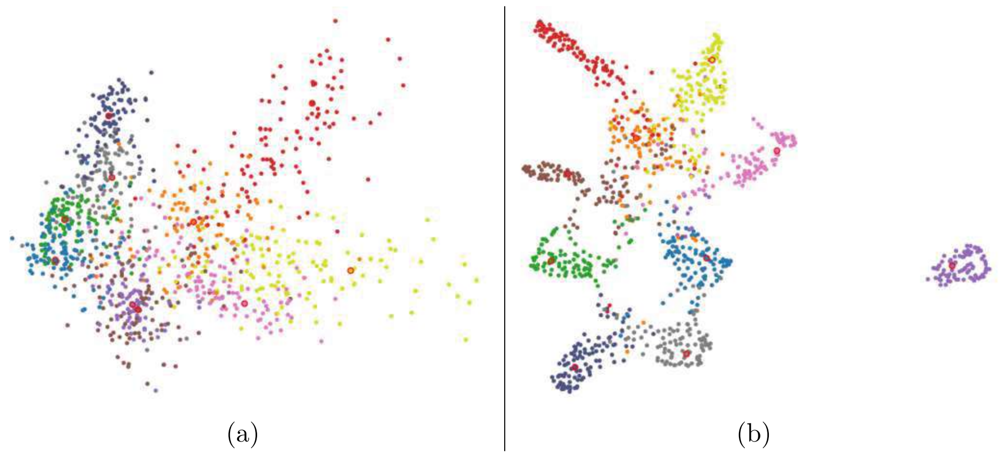
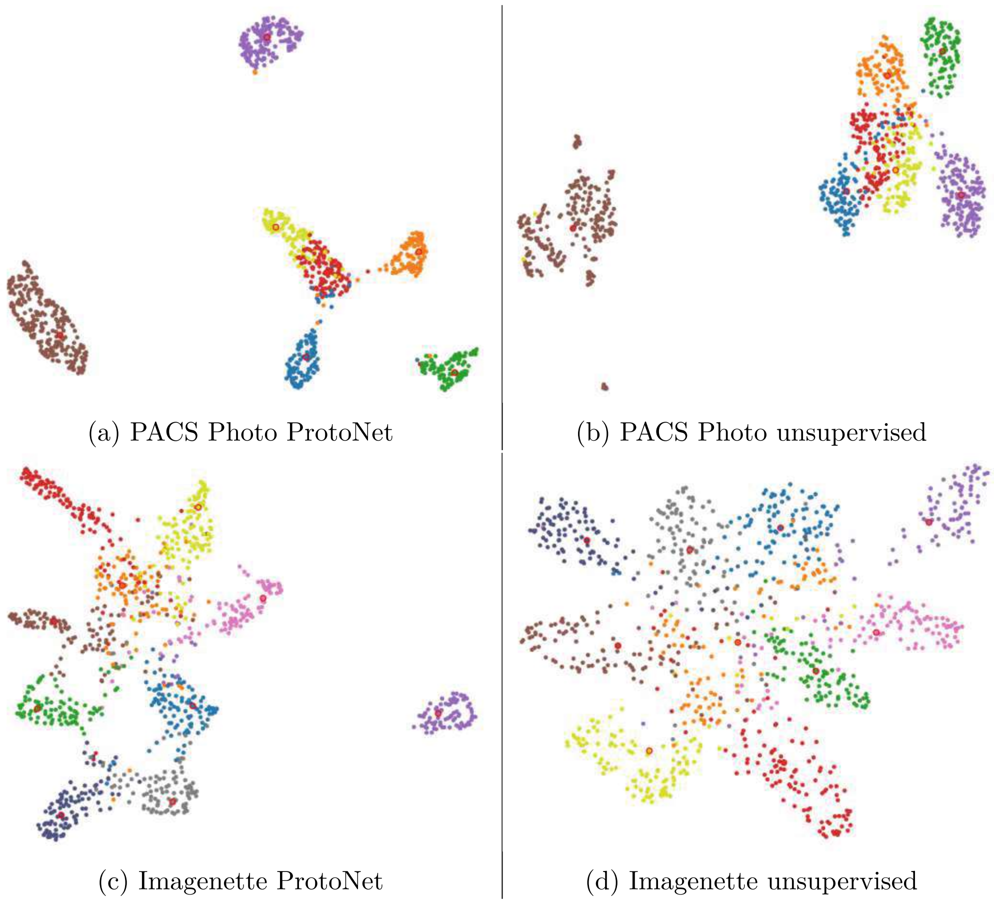


Figure 5.5: Imagenette comparison between ProtoNet with (a) PCA and (b) UMAP as final reduction method. The score for PCA is significantly worse.

The overall best results were recorded on the PACS Photo and the PACS Styles datasets with scores above 90 for both ProtoNet and UMAP. On the Imagenette dataset, while they showed similar layouts, training the ProtoNet lead to slightly more compact clusters compared to unsupervised UMAP, leading to slightly better DSC scores. Examples for this are shown in Figure 5.6.


(a) PACS Photo ProtoNet (b) PACS Photo unsupervised

(c) Imagenette ProtoNet (d) Imagenette unsupervised

Figure 5.6: Comparison of ProtoNet and unsupervised UMAP output on the PACS Photo and Imagenette datasets. ProtoNet and unsupervised UMAP produce similar results.

CelebHair and PACS Cartoon are datasets where we could only achieve middling success across all methods, never exceeding a score of 70. On both these datasets, using methods involving training lead to noticeably better results over the unsupervised method. ProtoNet outperformed unsupervised UMAP by over 20 points in terms of DSC in both cases. Looking at the PACS All dataset, this is arguably the most challenging dataset in our selection. Here, images belonging to seven classes are represented in different styles. We can see that the pure UMAP methods perform noticeably worse than their trained counterparts. The unsupervised embedding of all PACS data results in a clustering that is influenced more by the style of the images, rather than the actual content depicted in them. In contrast to that, a trained method like ProtoNet or LDA tries to create clusters according to the seven training classes, leading to better results. The difference can be seen in Figure 5.7.

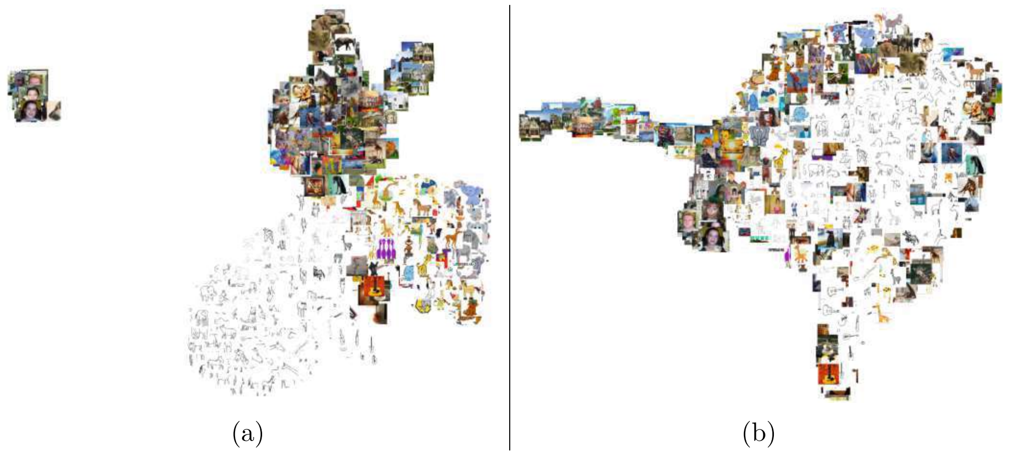


Figure 5.7: PACS All: Comparison between unsupervised UMAP (a) and ProtoNet + UMAP (b). ProtoNet separates clusters by semantic class, while unsupervised UMAP groups by style.

A comparison of the DSC and GON scores that all the methods achieved on the different datasets can be seen in Table 5.5. The scores reported in this table are the averages across twenty-five runs with the respective method and dataset. The full range of all these runs can be seen in Figure 5.8.

Computation Times

Computation time required for embedding the data is independent of the images that are in the dataset, but it does scale with the size of the dataset. For the experiments evaluated here, we chose to embed each type of dataset with test data containing 1000 data points, as well as training data of around 25 labeled samples per class, again varying from -1 to +2 labeled samples. Here, the number of classes in the dataset does slightly influence computation time, as more training data needs to be processed with more classes in the dataset, but overall, comparisons between the methods stay similar. An example of computation times for each method on the Imagenette dataset can be seen in Table 5.6.

5.5 A Qualitative Walkthrough

To evaluate the functionality of the implemented interface, and to look at some interesting cases a data analyst might encounter when using our visualization methods, we will now perform a qualitative walkthrough, where we showcase the different versions of the scatter plot visualization. We also explore different methods to suggest possible points of interest in the data visualization. The interesting cases for the data analyst's work are such where we purposefully introduce mistakes in the data we use for training the ProtoNet model. This means we cannot use the metrics DSC and GON we used before

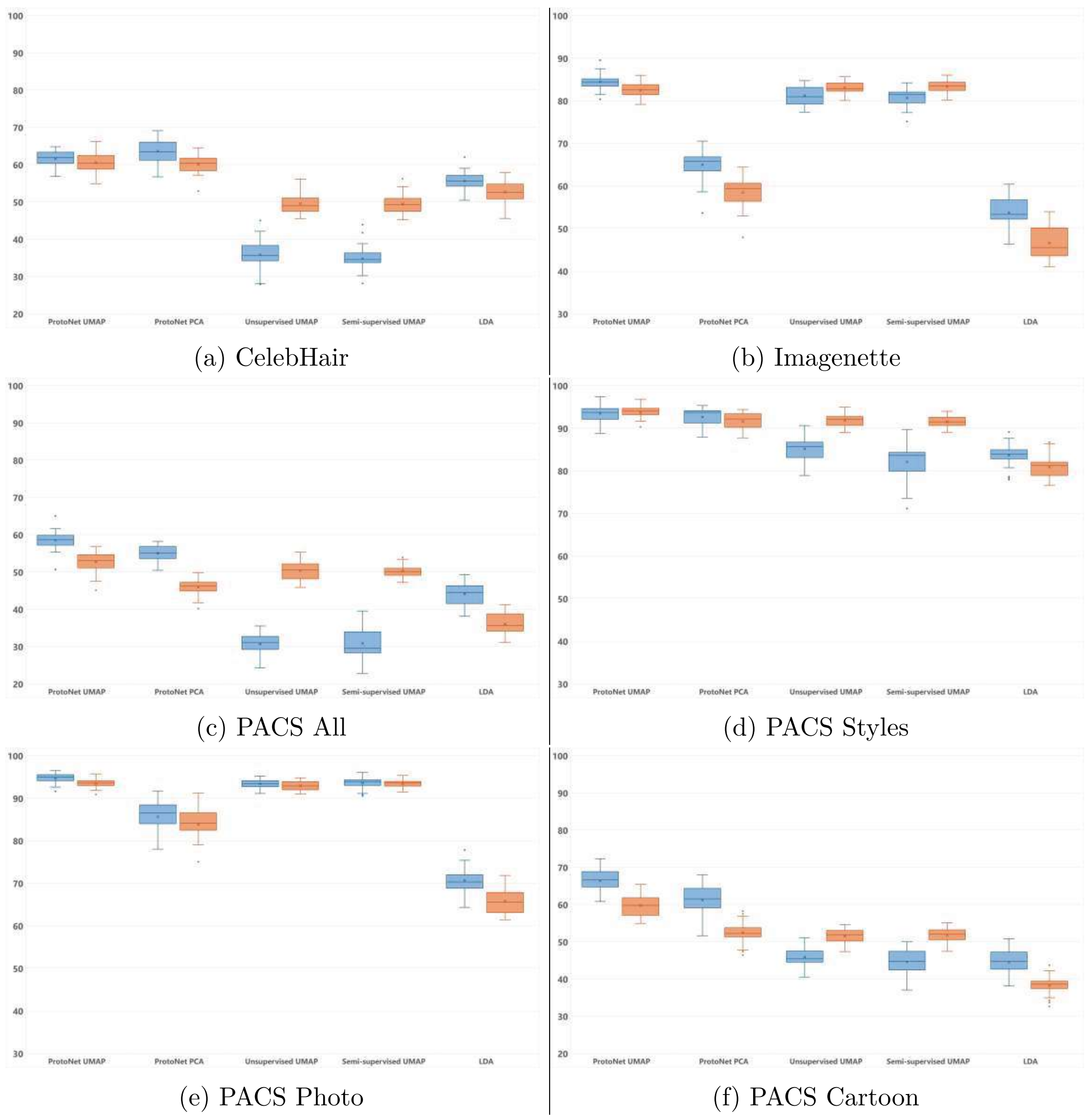
Different methods of data embedding					
CelebHair	ProtoNet UMAP	ProtoNet PCA	unsup. UMAP	semi-sup UMAP	LDA
DSC	61.51	**63.60**	35.85	34.82	55.56
GON	**60.54**	60.06	49.51	49.41	52.63
PACS Photo	ProtoNet UMAP	ProtoNet PCA	unsup. UMAP	semi-sup UMAP	LDA
DSC	**94.68**	85.63	93.37	93.51	70.65
GON	**93.46**	83.81	92.91	93.36	65.75
PACS Cartoon	ProtoNet UMAP	ProtoNet PCA	unsup. UMAP	semi-sup UMAP	LDA
DSC	**66.40**	61.16	45.86	44.56	44.50
GON	**59.72**	52.47	51.48	51.74	38.14
PACS All	ProtoNet UMAP	ProtoNet PCA	unsup. UMAP	semi-sup UMAP	LDA
DSC	**58.42**	55.00	30.62	30.83	44.08
GON	**52.69**	45.87	50.32	50.28	36.04
PACS Styles	ProtoNet UMAP	ProtoNet PCA	unsup. UMAP	semi-sup UMAP	LDA
DSC	**93.44**	92.64	85.15	82.09	83.64
GON	**93.65**	91.60	91.79	91.48	80.86
Imagenette	ProtoNet UMAP	ProtoNet PCA	unsup. UMAP	semi-sup UMAP	LDA
DSC	**84.44**	64.97	81.21	80.67	53.69
GON	**82.37**	58.49	83.02	83.32	46.58

Table 5.5: Comparison of the DSC and GON scores of the different embedding methods.

Computation time for x points	500	1000	1500
ProtoNet training & embedding	1,092	1,102	1,276 ms
UMAP	3,200	4,450	5,540 ms
PCA	3.38	5.48	5.89 ms
unsup. UMAP	3,440	4,480	7,100 ms
semi-sup UMAP	4,550	6,260	8,540 ms
LDA	71.8	78.9	87.9 ms

Table 5.6: Computation times on Imagenette with 500, 1000, and 1500 points.

to make reliable judgments, since these are built to score separability bases on ground truth labels and class prototypes, which may or may not be present in these scenarios.

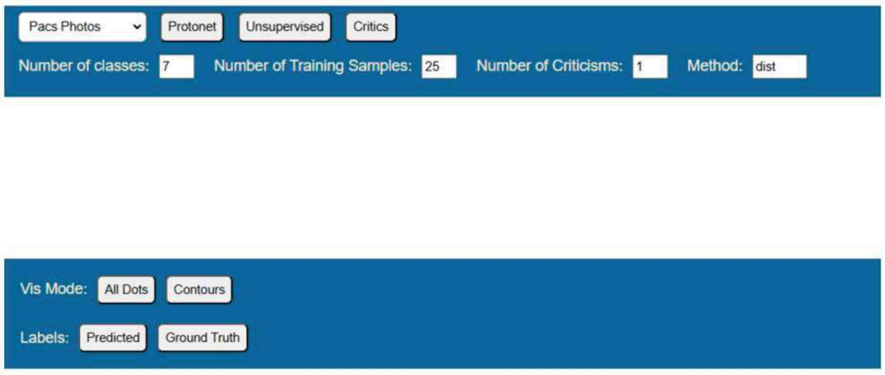


(a) CelebHair (b) Imagenette (c) PACS All (d) PACS Styles (e) PACS Photo (f) PACS Cartoon

Figure 5.8: Distribution of DSC (blue) and GON (orange) scores for each dataset and method across 25 different runs.

5.5.1 The Interface

When using the application, the user is first presented with the choice of dataset. They can select this from a drop-down menu of all the available options. The second step is to select the data that should serve as training data for the ProtoNet model. If this were an actual application, the user would pick samples from a display of some sorts, but since this is only an imitation for validation purposes, we only require the user to select the number of classes the dataset should be split into and the number of training samples per class. The test and training data is then automatically extracted from the dataset as described in Section 4.2.



Number of classes: 7 Number of Training Samples: 25 Number of Criticisms: 1 Method: dist

Figure 5.9: Interface of the Visualization Application.

Clicking either the button labeled ”ProtoNet“ or ”Unsupervised“, the user can choose which embedding method to use. As stated before, the only options here are ProtoNet and unsupervised UMAP. This is (1) because ProtoNet has shown throughout the other experiments to be the best semi-supervised method, and (2) to compare it to an unsupervised method to see if it would work just as fine without any training. Either of these buttons will result in a data embedding being calculated and the resulting scatter plot being displayed in the empty space beneath the parameter choice. The full interface can be seen in Figure 5.9.

Once a visualization is displayed, the user may switch between using individual dots for each data point or density based contour blobs. The color assigned to each point in the plot is defined by the class label given to it. Normally, this label is predicted by the embedding algorithm (as described in Chapter 4), but we also have the option to display the ground truth labels for validation purposes. We can see in Figure 5.10 that using contour lines makes cleaner clusters and the class overlap is easier to discern compared to the point visualization in cases where the classes are coherent. However, the Figure also shows that once there is no clean separation between classes, the visualization with contour lines becomes very hard to interpret.

Now that we have a color-coded scatter plot, we need to start making sense of what it is displaying. Hovering the mouse cursor over a class prototype will display the six images that lie closest to the respective class prototype to the right beside the plot (Figure 5.11). That way, we can identify what kind of image this class is meant to contain. To explore the space further, we can automatically detect areas in the visualization that are under-represented by the prototypes. For this, there are the final remaining input fields and button on the interface. The user is able to declare any number of criticism points to be calculated using one of three available methods. We can use ’dist’, a method returning the point with the maximum distance to all prototypes, ’mmd’, which uses the squared maximum mean discrepancy to find the most under-represented points based on point

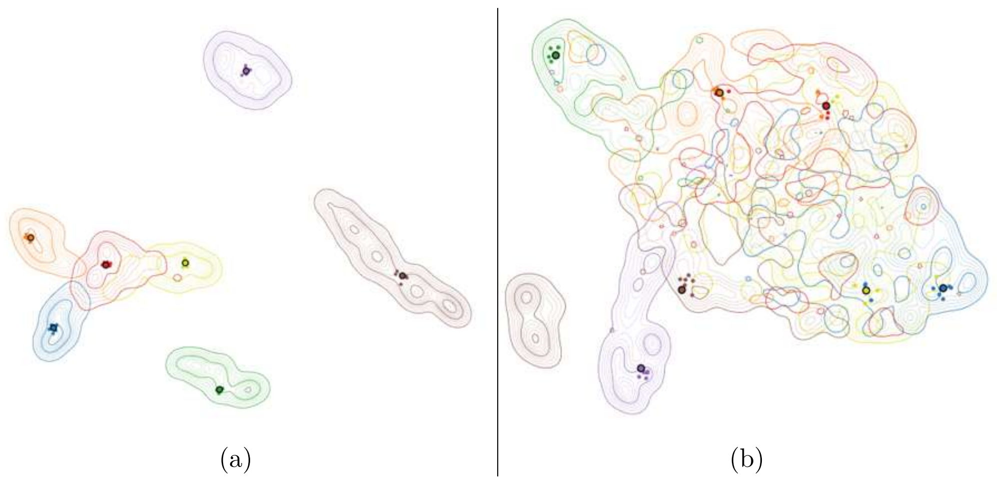


Figure 5.10: Contour lines with good (a) and bad separability (b).

density, and finally we can also use plain random sampling. Hovering such a criticism point will also display the corresponding image to the right of the plot. The usefulness and accuracy of these criticism points will play a bigger role in the following section.

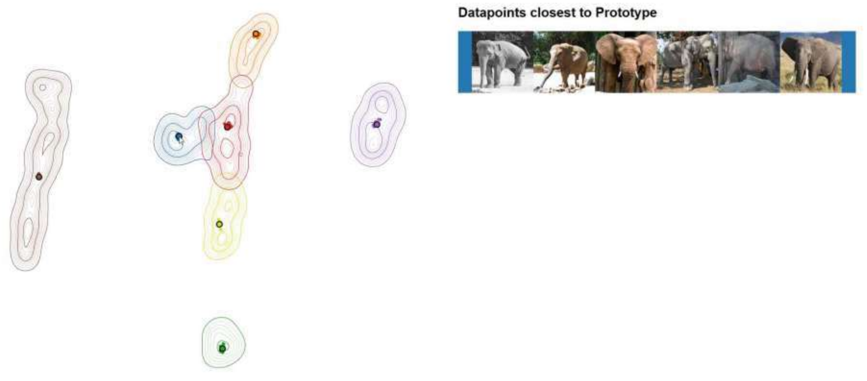


Figure 5.11: Actual images located closest to the prototype are displayed besides the plot on hovering a prototype.

5.5.2 Missing and Duplicate Classes

In an interactive data exploration scenario, where a user can select a few samples per class with which to train the embedding methods, they may not account for every class that is actually in the data. This is moreover a plausible case, because the data for which we want to apply these methods is largely unknown. There are three possible outcomes when it comes to the selection of training samples by the user. They could miss a class of data, meaning that it is not represented in the training (missing classes). They could also mistakenly match samples to two different classes that would in actuality be the same class (duplicate classes). And in the ideal case, they would provide samples for every class that is contained in the ground truth. When there are classes of images in the dataset which were not available during training, we would expect a good embedding method to place the data points of such classes in otherwise unoccupied parts of the embedding space. Similarly, we would expect a good embedding method to recognize the similarity in the two training classes that were split from one ground truth class and create an embedding where these two classes lie close together.

The case where there is exact training data for every class has been thoroughly evaluated throughout the previous experiments. We can now explore the suitability of ProtoNet as a method for interactive data exploration based on how well it manages to separate the class structure when there are missing and duplicate classes. The first example in Figure 5.12 shows a layout of the PACS Photo dataset where the class containing images of people was not included in the training process. This class is usually very well separated with full training. Figure 5.12a shows the layout with predicted labels based on the closest prototype. We can see that there is a distinctive area separate from other clusters, where the colors of different classes overlap. When looking at the same layout coded with ground truth labels in Figure 5.12b, we can see that this area consists of almost exclusively items that belong to the missing brown class.

What is also shown in this Figure are examples of the different criticism methods. The criticism point (single black point) in Figure 5.12a is calculated with the distribution based method, while the point in Figure 5.12b is based on maximum distance.

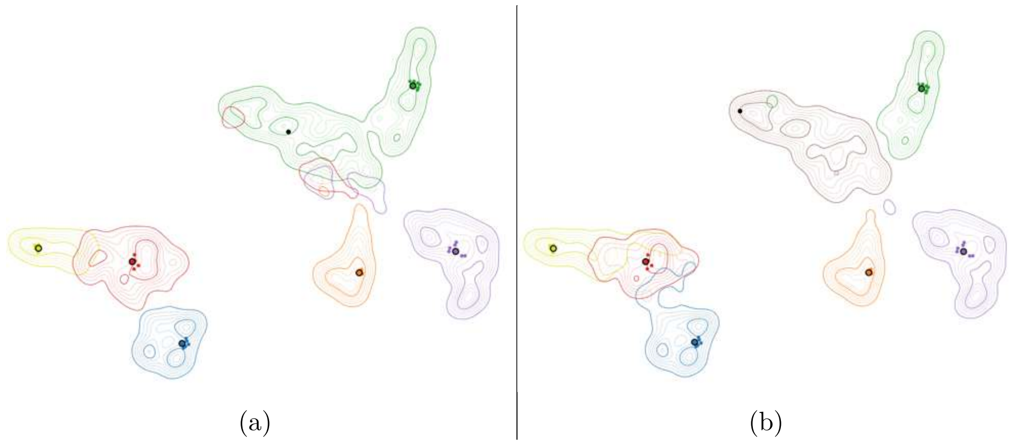

(a) (b)

Figure 5.12: PACS Photo: One class (brown) is missing during training. Predicted labels (a) and ground truth (b) show that the missing class is well separated.

We also use the PACS Photo dataset to showcase a good example for how duplicate classes should be handled. For the layout shown in Figure 5.13, samples from the images of horses (colored red) were introduced as two separate classes for the training of ProtoNet. We imagined this to be the most challenging in this dataset, as this is usually the least well separated class. Figure Table 5.13a is again coded with the predicted labels, where the two classes (red and pink) occupy the same space. Figure 5.13b also shows that the combination of these two clusters exactly makes up the area where the actual red cluster should be.

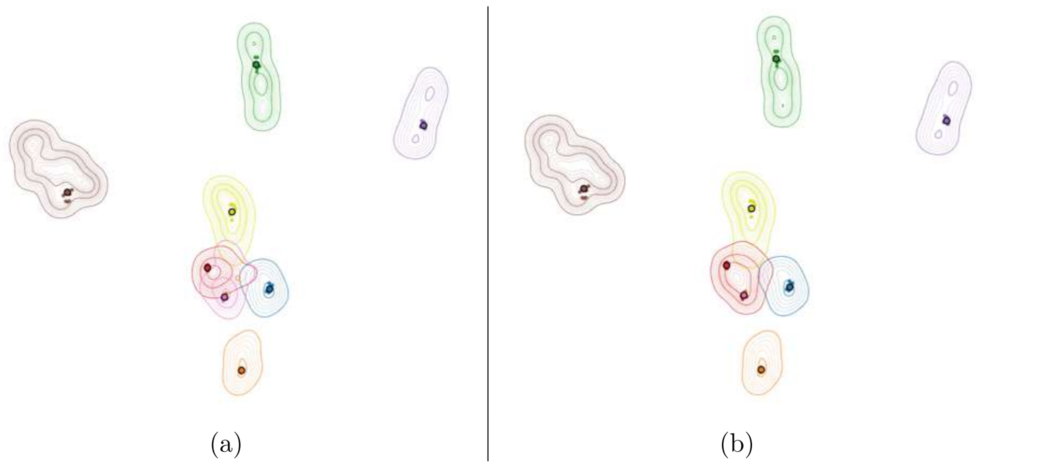


Figure 5.13: PACS Photo: The red class is split in two during training. Predicted labels (a) and ground truth (b) show that these classes are placed close together.

What we can see from these examples is that ProtoNet is able to handle cases of missing and duplicate training classes fairly well when the underlying data is very well separable. But we will also look at another example where this is not so much the case. Figures 5.14 and 5.15 show the same examples but with the PACS Cartoon dataset. Once again, no images of people were available when training. Unlike the previous example, there is now no discernible "extra cluster" in Figure 5.14a and Figure 5.14b, showing the true labels, makes it clear that the brown points are scattered throughout the plot. We choose to display the images as points over contour lines this time, because the busy nature of this layout makes it even harder to interpret the position of the missing class.

When duplicating the horse class with the PACS Cartoon dataset, we can see that the results are still fairly good. We can see in Figure 5.15a that the main body of the predicted pink and red classes are rather close together and Figure 5.15b confirms that they in fact share an area where actual red data items lie.

Although we have already ruled out LDA as a suitable DR method, we still performed some experiments to see how it would handle missing and duplicate classes. We can now say that LDA, across all datasets, proved to be very unreliable when it comes to missing training classes. Points of the missing classes are mostly scattered across other classes with rarely a coherent cluster to be found. When introducing samples of the same class as separate classes, LDA most of the times forces the two resulting prototypes far apart, resulting in a wide band of data points with a prototype at either end. Examples of this are shown in Figure 5.16.

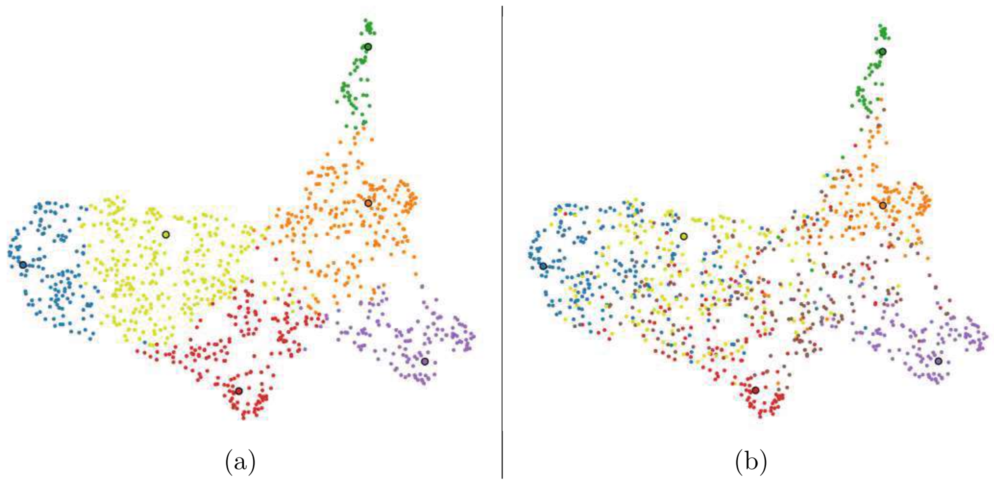


Figure 5.14: PACS Cartoon: The brown class is missing during training. Predicted labels (a) and ground truth (b) show that test data of that class is not grouped together.

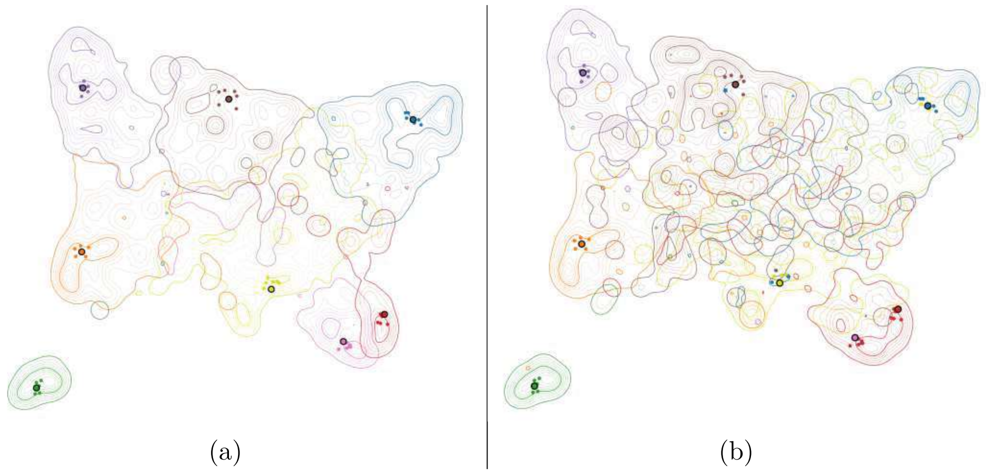
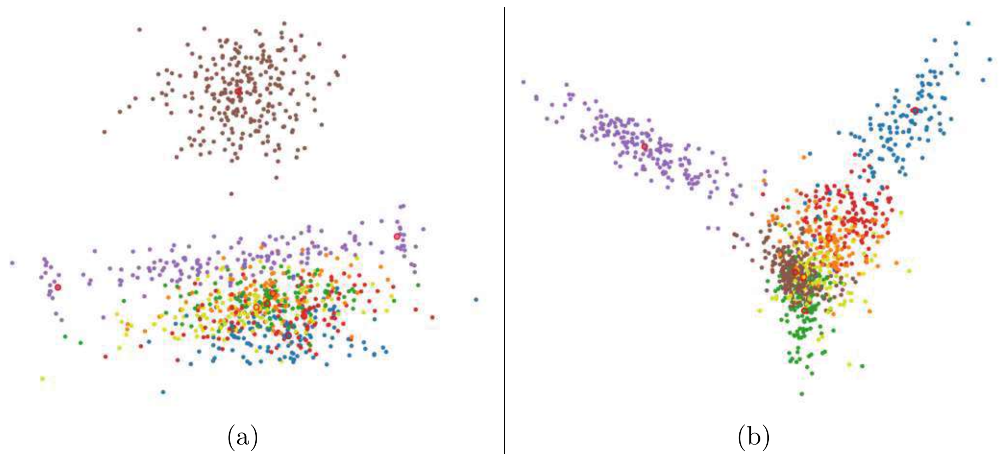


Figure 5.15: PACS Cartoon: The red class is split in two during training. Predicted labels (a) and ground truth (b) show that these classes are still placed in the same area.



(a) (b)

Figure 5.16: Examples of how LDA deals with missing and duplicate classes. (a) ground truth labels with the purple class being split in training. The purple and pink prototype are not close together. (b) ground truth labels with the brown class missing during training. The usually well separated class is overlapping other classes.

CHAPTER 6

Discussion

Our first research questions asks how the interpretation of unstructured data differs between projections based on prototypical networks and other dimensionality reduction methods. We argue that this interpretation of the data is based on how well the data is clustered according to the class membership of its items. First, depending on the dataset, results can vary by large margins. Especially the results of the unsupervised UMAP method vary greatly when compared across datasets. Since there is no supervision involved in this method, this reflects the inherent separability of these datasets. This is even more apparent when looking at the datasets themselves. It is easy to understand that differentiating between pictures of faces with black, blond, or brown hair like in the CelebHair dataset (one where unsupervised UMAP performed fairly poorly) is a lot harder than to distinguish a photo of a person from a house or an elephant (PACS Photo).

We found that whenever an unsupervised approach managed to produce good results, ProtoNet still managed to slightly improve these good results (See Figure 5.6 and Table 5.5). We have not encountered any cases where the unsupervised method exceeded the ProtoNet embeddings in terms of separation metrics. But we can also say that in these cases of datasets with high inherent separability (i.e., PACS Photo and Imagenette), ProtoNet does not provide any noticeable advantage over the purely unsupervised method. An interesting exception to this are the observations made on the PACS Styles dataset. This dataset, too, has high inherent separability, being divided in four groups of different art styles, but embedding them with no supervision leads to photos of persons being separated far from all other images (Figure 6.1). This results in a photo class that is separated from itself, distorting the position of the class center, and decreasing the DSC score, all while keeping a GON score that is on par with ProtoNet. This tells us that the class may be disconnected (lower DSC), but similar items are still grouped together into pure clusters (high GON).

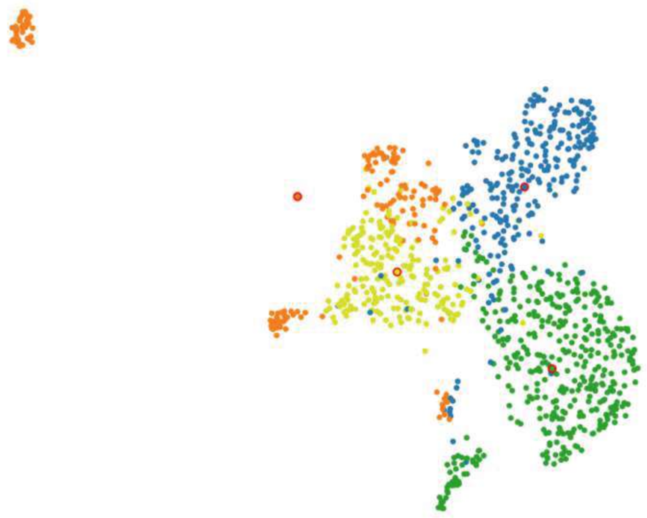


Figure 6.1: PACS Styles: No supervision leads to a disconnected class.

Where the advantages of ProtoNet become apparent are the datasets where the inherent separability is a lot lower. On these datasets, unsupervised embeddings with UMAP largely fail to produce well separated clusters with only about one third (CelebHair) or 45% (PACS Cartoon) of points being closest to their own cluster center. Here, the little training that is introduced with ProtoNet manages to improve the clustering of points by a fair margin, although the results are still far from perfect with scores of around 60 (DSC and GON CelebHair, GON PACS Cartoon) and 67 (DSC PACS Cartoon). The dataset that best shows the usefulness of training ProtoNet over using unsupervised embedding is PACS All. Like PACS Styles, this includes all the images of the PACS dataset, but they are not labeled by their art style, but by the objects they depict. The unsupervised method, as expected, embedded all points in the same way as it did with the PACS Styles dataset (Figure 5.7a), since without supervision, they are both the same. Naturally, this does not reflect the intended target classes at all, leading to a low DSC score of around 30. Using ProtoNet on the same data (Figure 5.7b), with some samples per class, we manage to create embeddings that reach scores that are much closer to what we achieved on other datasets, separating some classes rather well, while others remain fairly mixed.

Creating two-dimensional embeddings of our DINO feature vectors using only LDA always resulted in projections that were worse than what ProtoNet could produce. This is reflected in both the scores we measured and also qualitative observation of the resulting plots. LDA barely exceeded scores of 50 on four of the six tested datasets. And even on the two datasets where it separated the classes a bit better (PACS Photo and PACS Styles) it still trailed behind ProtoNet by large margins of more than 20 (Photo) and 10 (Styles) DSC points, respectively. Overall, LDA never managed to create compact clusters, always leaving low-density blotches with frizzy edges (Figure 6.2).

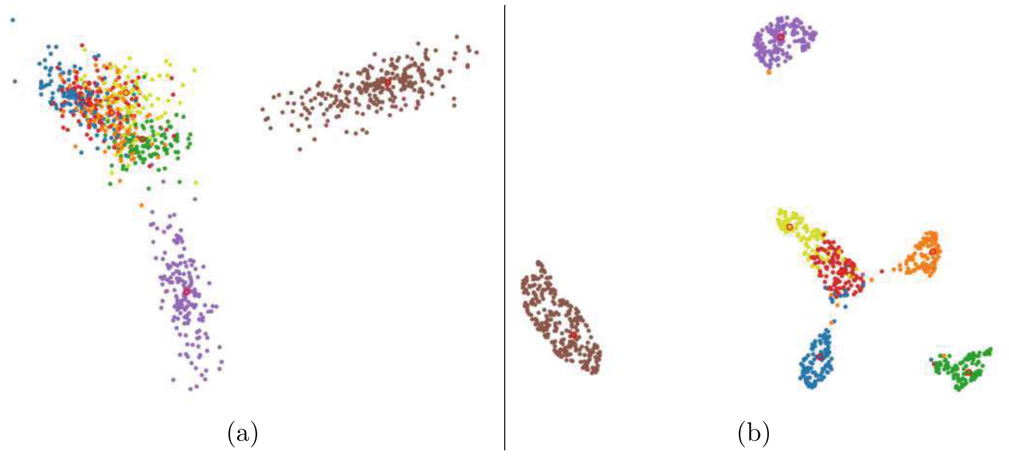


Figure 6.2: LDA produces less dense clusters with frizzy edges (a) compared to ProtoNet (b) on the PACS Photo dataset.

In a similar fashion, using PCA as the final step in the ProtoNet pipeline instead of UMAP almost always resulted in clusters that were less compact. Although the separation metrics score this method almost always only closely behind ProtoNet with UMAP (with the exception of the Imagenette dataset), looking at the results clearly shows more compact and better separated clusters with the UMAP variant (Figure 6.3).

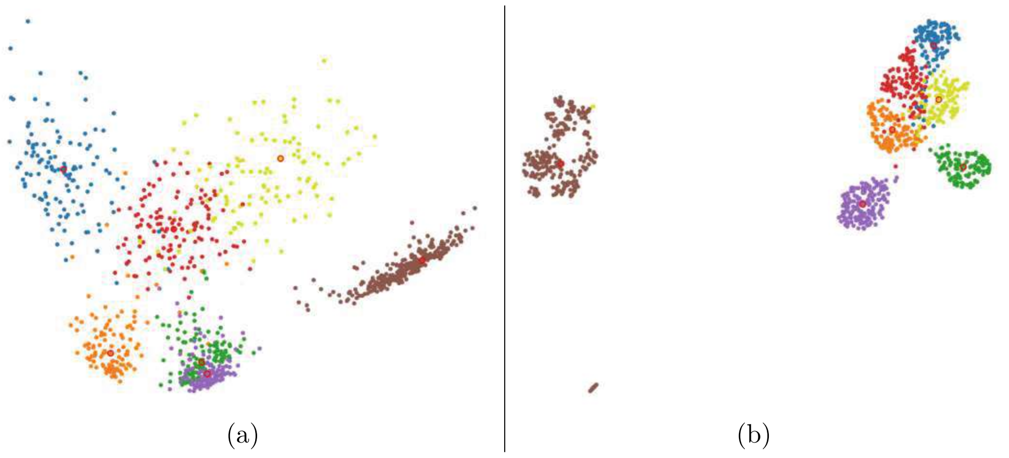


Figure 6.3: ProtoNet + PCA (a) produces only slightly worse scores compared to ProtoNet + UMAP (b), but visually they are quite different.

The second research question asks how the computational performance differs between all these methods. Looking at the table of computation times (Table 5.6), we can immediately see that our methods fall into two categories. The faster ones, using linear DR techniques like PCA and LDA, and the much slower ones, utilizing UMAP. This almost directly correlates with the output quality of the embedding spaces. While PCA and LDA are very fast, they are not capable of producing such compact clusters as the much slower methods using UMAP. As Table 5.6 shows, computation time increases with the number of data points in the dataset for all methods. This is almost negligible for the faster methods, as they operate in a matter of milliseconds, but for methods using UMAP, computation time is an issue we need to concern ourselves with. Taking multiple seconds to calculate a new embedding can already interrupt the workflow of a data analyst. UMAP is a method whose computation time not only grows with an increasing numbers of data points, but also with the number of features that need to be reduced. While ProtoNet training initially adds an overhead of about one second to the UMAP embedding, using ProtoNet also decreases the dimensions UMAP needs to reduce from the initial 2,048 of the DINO feature vectors to only 64. Additionally, as Table 5.4 has shown, training the ProtoNet and embedding the data with it operates on constant time. With an increasing dataset size, this overhead is quickly compensated by the reduced workload UMAP has to perform. Figure 6.4 shows that unsupervised UMAP already exceeds ProtoNet + UMAP in computation time at a dataset size below 1,500.

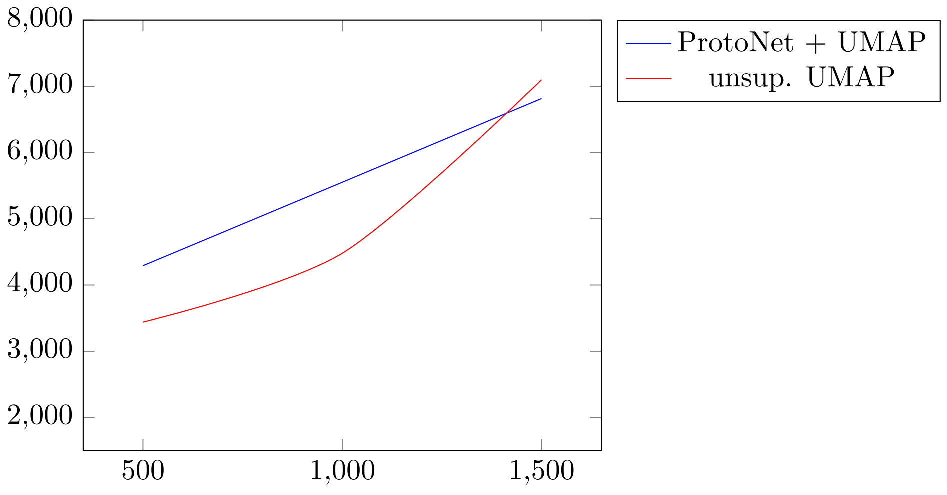
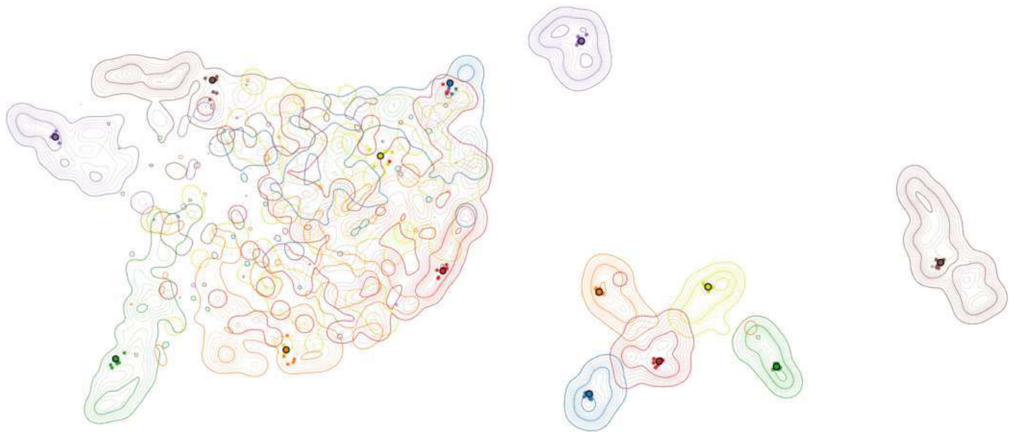


Figure 6.4: Computation time scaling with increased dataset size in milliseconds.

Our third research question asks how we can effectively visualize prototypes and other data items, especially for very large datasets. While implementing our prototype visualization interface, we found that marking all individual data points in a scatter plot only works well up to a certain amount of data before we run into the problems of over-plotting and visual clutter. We therefore made the decision to display the vast majority of data points as contour lines that represent the density of the depicted clusters. This not only reduced the number of elements on the screen, but also proved to better indicate cluster overlap compared to individual points that are intermixed. However, this only produces appealing results when the used embedding method created a point layout with good cluster separation. When a very poor separation result is created, using contour lines arguably introduces more confusion by creating a very busy plot (Figure 6.5).

By displaying the class prototypes as marked dots, as well as their closest neighbors, we



(a) (b)

Figure 6.5: Bad separability (a) of PACS All and good separability (b) of PACS Photo visualized with contour lines.

create a detectable area of interest within the contour lines, where a user can access the actual images behind those data points to identify the class identity. For possible new points of interest, we found that generating criticism points with the distribution based approach of the MMD-critic framework performed better at locating the center point of underrepresented areas in the plot compared to points that are found via the maximum distance between prototypes. These often lie somewhere on the edge of the potential area of interest. These new criticism points, which we paint in solid black, contrast with the otherwise colorful plot and allow the user to investigate new areas of the data that were previously underrepresented by the class prototypes.

The fourth research question asks how the ProtoNet based data visualization can be integrated into interactive visual analysis of large, unstructured data. We argue that our ProtoNet approach does a good job in creating similarity preserving scatter plots for large amounts of unstructured data. The main point of an interactive visual analysis task is to examine previously unknown data. ProtoNet allows the user to select few examples of any amount of classes that the user thinks there are in the data. Based on these few examples, it generates prototypes and creates an embedding space where all of the data is laid out according to the provided samples. We have shown that ProtoNet can match or exceed the embedding quality of other supervised or unsupervised methods and even performs well in the very plausible case that the user declared too many classes or missed out on others. Especially when images depicting the same object are put in different training classes, ProtoNet managed to mitigate these errors quite well and produced accurate layouts. Classes missing from training can still be somewhat compact, if not always as well separated as others. In the best cases, these unidentified areas of the plot are still visually discernible and most of the time the first target for criticism points (e.g. Figure 6.6), indicating to the user that these areas are underrepresented by the generated class prototypes and may hide other classes of data points that the user did not include in the training.

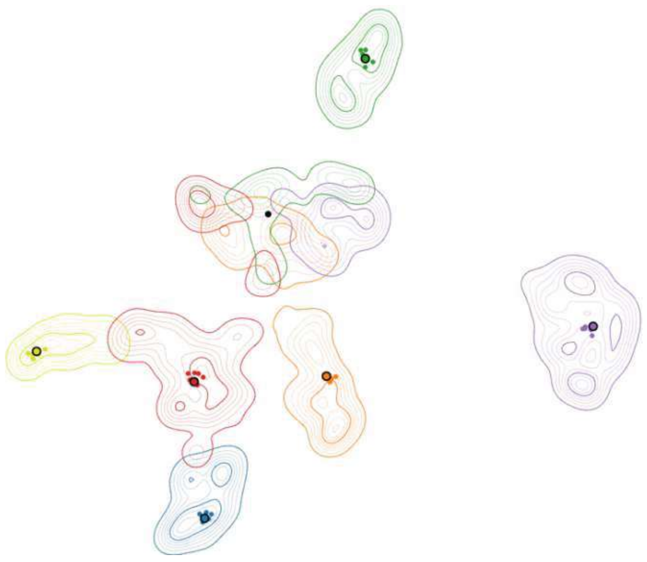


Figure 6.6: Class missing in training is clearly distinguishable and criticism point suggests area is of interest.

This combination of creating good layouts and being resistant to user inflicted training errors, as well as the ability to indicate new interesting areas of the data based on prototype and data density makes ProtoNet well suited for interactive visual analysis. Unfortunately, the use of UMAP in our ProtoNet DR pipeline requires quite a large amount of computation time, resulting in noticeable delays in response time and somewhat diminishing the usability in an interactive setting.

CHAPTER 7

Conclusion

In conclusion, this thesis proposed the novel use of a prototypical network based approach for the visualization of high-dimensional, unstructured data. We discovered that applying some sort of feature extraction on real image data is a necessary condition for the success of this approach, as both ProtoNet and all the other investigated methods performed very poorly on raw pixel information. Through systematic experiments, we first determined the best parameterization of our methods and then compared the outputs of those methods based on various different datasets. The outputs were assessed both qualitatively and quantitatively through the calculation of metrics that measure the visual separability of ground truth classes in the resulting two-dimensional visualizations. Regarding RQ1, we determined that the most successful method we investigated was a ProtoNet embedding in combination with a final 2D reduction using UMAP. While this method produced similar results to a fully unsupervised UMAP on some datasets, it clearly showed its advantages on “hard cases”, where unsupervised UMAP was not able to produce favourable results. The much faster methods LDA and ProtoNet in combination with PCA may have excelled in terms of computation time, but the quality of the output was decidedly worse.

This already ties into RQ2, the question of computational performance. PCA and LDA are extremely fast, but fail to produce satisfying results. On the other hand, UMAP is the major bottleneck that dampens the usability of our DR pipeline. The time needed to produce results is slightly too long to be considered for an interactive environment.

Concerning RQ3, we implemented and described a prototype visualization interface, where we present different methods of displaying the data on a 2D scatter plot, either as individual data points or density based contour lines. Class prototypes and candidates for undiscovered aspects of the data are also present. A qualitative walkthrough of this application showed that displaying the embedded data as density based contour lines leads to a more visually appealing and easier to interpret outcome, so long as the data could be separated into coherent class clusters.

Finally, we address RQ4 by evaluating the usefulness of our ProtoNet approach for interactive visual analysis. We found that ProtoNet is quite robust when it comes to dealing duplicated classes in the training data. When images that are actually very similar are labeled as different classes for the training, then ProtoNet will place these classes close together regardless. Regarding classes omitted from training, our observations have shown that ProtoNet can produce decent results when the omitted class is somewhat distinctive from the rest and the overall separability of classes is high. In ”hard cases“ the detection of untrained classes is difficult. Also, the distribution based detection of underrepresented areas in the data showed to be more beneficial compared to the distance based approach and random sampling. Criticism points calculated based on distribution tend to lie in the center of undiscovered clusters, while points based on distance can quite often fall on the outer edges of data clusters.

7.1 Limitations & Future Work

This work performed very basic experiments to evaluate the separating capabilities of different DR approaches. Our whole setup is simulating a data analysis task without the involvement of actual human agents. We also rely on quantitative measures to score the visual separation of our scatter plots. Although these measures have been shown to closely resemble human judgment, a more detailed study with human participants might uncover different preferences in terms of the visual appearance of the separated clusters that the different methods produce.

With our automatic process of selecting training samples, we assume that the samples that are randomly picked from each class are representative of that class. A real user most likely would not pick samples at random, but choose to pick samples that are very prototypical of the type of image they want to group. They might also deliberately choose such samples that depict edge cases that should be included in the grouping of that class. Future research could try to evaluate if samples hand-picked with human reasoning would further improve the quality of the output, maybe even with less samples required.

Throughout our research, we have shown that some form of feature extraction is necessary for a good result. Working with raw images led to no proper class separation at all. For this thesis, we relied on but one feature extraction method. Using the feature vectors we got from DINO, we were able to create 2D embeddings that resulted in both easily separable datasets and also datasets where separation did not work quite as well. Future research could try to extract feature representations from these same datasets by methods other than DINO to see if ProtoNet is able to produce better results if the input were different.

The major bottleneck that hampers the usefulness of our ProtoNet method is the use of UMAP as a final reduction method in the pipeline. This method takes too much time to calculate the data embeddings for it to be used in an actual interactive environment. One reason for this is that, while the ProtoNet embedding is performed on a GPU, UMAP is only able to operate on the much slower CPU. There are implementations of UMAP that run on a GPU, however, these are only available for Linux, which clashes with the Windows based implementation of the rest of our application. Very rudimentary tests with such a UMAP implementation for GPU have also shown that this scales poorly with the size of the data. Further experiments and evaluations of the real-time usability of our ProtoNet approach could be conducted if a GPU version of UMAP were available on Windows, or another method were to produce comparably good results to UMAP. We have, of course, only evaluated a handful of well known DR techniques that we deemed promising. There are a numerous other methods that could be built on top of a ProtoNet instead of UMAP that could potentially achieve better results while saving computation time.

List of Figures

List of Tables

Bibliography

[AS16] Michaël Aupetit and Michael Sedlmair. Sepme: 2002 new visual separation measures. In *2016 IEEE pacific visualization symposium (PacificVis)*, pages 1–8. IEEE, 2016.

[BG98] Suresh Balakrishnama and Aravind Ganapathiraju. Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing*, 18(1998):1–8, 1998.

[BKK$^+$19] Stuart Berg, Dominik Kutra, Thorben Kroeger, Christoph N Straehle, Bernhard X Kausler, Carsten Haubold, Martin Schiegg, Janez Ales, Thorsten Beier, Markus Rudy, et al. Ilastik: interactive machine learning for (bio) image analysis. *Nature methods*, 16(12):1226–1232, 2019.

[CBP09] Jaegul Choo, Shawn Bohn, and Haesun Park. Two-stage framework for visualization of clustered high dimensional data. In *2009 IEEE Symposium on Visual Analytics Science and Technology*, pages 67–74. IEEE, 2009.

[CFZ99] Chun-Hung Cheng, Ada Waichee Fu, and Yi Zhang. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 84–93, 1999.

[CGSQ11] Nan Cao, David Gotz, Jimeng Sun, and Huamin Qu. Dicon: Interactive visual analysis of multidimensional clusters. *IEEE transactions on visualization and computer graphics*, 17(12):2581–2590, 2011.

[Che73] Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American statistical Association*, 68(342):361–368, 1973.

[CHUW08] Chun-houh Chen, Wolfgang Härdle, Antony Unwin, and Matthew O Ward. Multivariate data glyphs: Principles and practice. *Handbook of data visualization*, pages 179–198, 2008.

[CLNL87] Daniel B Carr, Richard J Littlefield, WL Nicholson, and JS Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[CLRP13] Jaegul Choo, Changhyun Lee, Chandan K Reddy, and Haesun Park. Utopian: User-driven topic modeling based on interactive nonnegative matrix factorization. *IEEE transactions on visualization and computer graphics*, 19(12):1992–2001, 2013.

[CTM+21] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DK10] Aritra Dasgupta and Robert Kosara. Pargnostics: Screen-space metrics for parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1017–1026, 2010.

[DZK+20] Shumin Deng, Ningyu Zhang, Jiaojian Kang, Yichi Zhang, Wei Zhang, and Huajun Chen. Meta-learning with dynamic-memory-based prototypical network for few-shot event detection. In *Proceedings of the 13th international conference on web search and data mining*, pages 151–159, 2020.

[EF09] Niklas Elmqvist and Jean-Daniel Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE transactions on visualization and computer graphics*, 16(3):439–454, 2009.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[FBSL19] Miao Fan, Yeqi Bai, Mingming Sun, and Ping Li. Large margin prototypical network for few-shot relation classification with fine-grained features. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2353–2356, 2019.

[FD05] Michael Friendly and Daniel Denis. The early origins and development of the scatterplot. *Journal of the History of the Behavioral Sciences*, 41(2):103–130, 2005.

[FFFP06] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[FH10] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning and ranking by pairwise comparison. In *Preference learning*, pages 65–82. Springer, 2010.

[FLK19] Alexander Fritzler, Varvara Logacheva, and Maksim Kretov. Few-shot classification in named entity recognition task. In *Proceedings of the 34th ACM/SIGAPP symposium on applied computing*, pages 993–1000, 2019.

[For17] Stanislav Fort. Gaussian prototypical networks for few-shot learning on omniglot. *arXiv preprint arXiv:1708.02735*, 2017.

[GHLS19] Tianyu Gao, Xu Han, Zhiyuan Liu, and Maosong Sun. Hybrid attention-based prototypical networks for noisy few-shot relation classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6407–6414, 2019.

[H+20] Jeremy Howard et al. Imagenette. *URL https://github.com/fastai/imagenette*, 2, 2020.

[HLH+16] Christian Heine, Heike Leitte, Mario Hlawitschka, Federico Iuricich, Leila De Floriani, Gerik Scheuermann, Hans Hagen, and Christoph Garth. A survey of topology-based methods in visualization. In *Computer Graphics Forum*, volume 35, pages 643–667. Wiley Online Library, 2016.

[HR07] Jeffrey Heer and George Robertson. Animated transitions in statistical data graphics. *IEEE transactions on visualization and computer graphics*, 13(6):1240–1247, 2007.

[HW13] Julian Heinrich and Daniel Weiskopf. State of the art of parallel coordinates. *Eurographics (state of the art reports)*, pages 95–116, 2013.

[JC16] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[JCY+20] Zhong Ji, Xingliang Chai, Yunlong Yu, Yanwei Pang, and Zhongfei Zhang. Improved prototypical networks for few-shot learning. *Pattern Recognition Letters*, 140:81–87, 2020.

[JLJC05] Jimmy Johansson, Patric Ljung, Mikael Jern, and Matthew Cooper. Revealing structure within clustered parallel coordinates displays. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 125–132. IEEE, 2005.

[KK94] Daniel A Keim and H-P Kriegel. Visdb: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, 1994.

[KKK16] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems*, 29, 2016.

[Koh90] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LJH13] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. In *Computer graphics forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[LLWT15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[LMW$^+$16] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE transactions on visualization and computer graphics*, 23(3):1249–1268, 2016.

[LSGT11] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011.

[LYSH17] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[MAK$^+$08] Emmanuel Müller, Ira Assent, Ralph Krieger, Timm Jansen, and Thomas Seidl. Morpheus: interactive exploration of subspace clustering. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1089–1092, 2008.

[MG13] Adrian Mayorga and Michael Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE transactions on visualization and computer graphics*, 19(9):1526–1538, 2013.

[MHM18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[NiA24] Miquel Noguer i Alonso. Reinforcement learning: A historical and mathematical overview (1950-2024). *Available at SSRN 4963741*, 2024.

[Nie93] Jakob Nielsen. Response times: the three important limits. *Usability Engineering*, 1993.

[RS00] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[SA15] Michael Sedlmair and Michaël Aupetit. Data-driven evaluation of visual quality measures. In *Computer graphics forum*, volume 34, pages 201–210. Wiley Online Library, 2015.

[Set09] Burr Settles. Active learning literature survey. 2009.

[SNLH09] Mike Sips, Boris Neubert, John P Lewis, and Pat Hanrahan. Selecting good views of high-dimensional data using class consistency. In *Computer Graphics Forum*, volume 28, pages 831–838. Wiley Online Library, 2009.

[SSZ+17a] Dominik Sacha, Michael Sedlmair, Leishi Zhang, John A Lee, Jaakko Peltonen, Daniel Weiskopf, Stephen C North, and Daniel A Keim. What you see is what you can change: Human-centered machine learning by interactive visualization. *Neurocomputing*, 268:164–175, 2017.

[SSZ17b] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[SW09] Harald Sanftmann and Daniel Weiskopf. Illuminated 3d scatterplots. In *Computer Graphics Forum*, volume 28, pages 751–758. Wiley Online Library, 2009.

[TFH11] Cagatay Turkay, Peter Filzmoser, and Helwig Hauser. Brushing dimensions-a dual visual analysis model for high-dimensional data. *IEEE transactions on visualization and computer graphics*, 17(12):2591–2599, 2011.

[VBL+16] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[Vid11] René Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2011.

[WAG05] Leland Wilkinson, Anushka Anand, and Robert Grossman. Graph-theoretic scagnostics. In *Information Visualization, IEEE Symposium on*, pages 21–21. IEEE Computer Society, 2005.

[WC06] Jing Wang and Chein-I Chang. Independent component analysis-based dimensionality reduction with applications in hyperspectral image analysis. *IEEE transactions on geoscience and remote sensing*, 44(6):1586–1600, 2006.

[WYKN20] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[YGY+18] Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. Diverse few-shot text classification with multiple metrics. *arXiv preprint arXiv:1805.07513*, 2018.

[YKD+18] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *Advances in neural information processing systems*, 31, 2018.

[YPH+04] Jing Yang, Anilkumar Patro, Shiping Huang, Nishant Mehta, Matthew O Ward, and Elke A Rundensteiner. Value and relation display for interactive exploration of high dimensional datasets. In *IEEE Symposium on Information Visualization*, pages 73–80. IEEE, 2004.