Institute for Visualization and Interactive Systems
Abteilung Virtual Reality und Augmented Reality

University of Stuttgart
Allmandring 19
D–70569 Stuttgart

Bachelorarbeit

An Exploratory Approach on Information Visualization using Unsupervised Machine Learning

Mohamad Altaweel

Course of Study:	Softwaretechnik
Examiner:	Jun.-Prof. Dr. Michael Sedlmair
Supervisor:	M.Sc. Cristina Morariu
Commenced:	April 1, 2020
Completed:	November 26, 2020

Abstract

Nowadays, we use various types of charts in academic papers, journals, and newspapers for several purposes. Information Visualizations are powerful tools that researchers use to describe the data. It makes it easier to detect hidden information and patterns from the data, such as trends and relationships.

However, the huge increase in data amount and complexity leads to escalating the number of visualizations as well and complicating its design process. Designers should analyze all those factors to choose the best design decisions for readable and straightforward visualization. For this reason, researchers have been using machine learning approaches in building automated systems that extract information and attributes from visualizations and infographics. The ML-based automated system would simplify the evaluation of visualizations which makes it easy to automate generating charts from data, based on the visual encodings it has learned.

This project explores the research problem that states the ability of the machine to form a good representation of heterogeneous charts. These representations let the users compare and classify them from a meaningful human perspective. We aim to apply unsupervised machine learning methods on charts image to get a simplified representation from heterogeneous charts.

Our method applies transfer knowledge methods in computer vision. It uses a pre-trained CNN on the ImageNet dataset to get a chart representation vector and uses dimension reduction methods on the network output to project all charts representation on a 2D plane.

We evaluate this approach by applying it on different use-case scenarios of different charts' datasets that explain the projection results and determines the context of the distance measure on the output space.

Key words: Information Visualization, Automatic Visualization, Unsupervised Machine Learning, Dimension Reduction

Kurzfassug Heutzutage verwendet man verschiedene Arten von Diagrammen in akademischen Arbeiten, Zeitschriften und Zeitungen für verschiedene Zwecke. Diagramme und Informationsvisualisierungen sind gute Werkzeuge, die Forscher zur Beschreibung der Daten nutzen. Sie erleichtern es, verborgene Informationen und Muster aus den Daten zu erkennen, wie zum Beispiel Tendenz und Beziehungen.

Die enorme Zunahme der Datenmenge und -komplexität führt jedoch dazu, dass auch die Anzahl der Visualisierungen ansteigt und der Designprozess erschwert wird. Designer sollten all diese Faktoren analysieren, um die besten Designentscheidungen für eine lesbare und unkomplizierte Visualisierung zu treffen. Aus diesem Grund haben Forscher beim Aufbau automatisierter Systeme, die Informationen und Attribute aus Visualisierungen und Infografiken extrahieren, Ansätze des maschinellen Lernens verwendet. Das ML-basierte automatisierte System würde die Auswertung von Visualisierungen vereinfachen, so dass die Generierung von Diagrammen aus Daten auf der Grundlage der gelernten visuellen Kodierungen leicht automatisiert werden kann.

In diesem Projekt wird das Forschungsproblem untersucht, das die Fähigkeit der Maschine angibt, eine gute Darstellung heterogener Diagramme zu bilden. Diese Darstellungen ermöglichen es den Benutzern, sie aus einer sinnvollen menschlichen Perspektive zu vergleichen und zu klassifizieren. Der Zweck dieser Arbeit ist die anwendung von unsupervised Methoden des maschinellen Lernens auf das Bild von Diagrammen, um eine vereinfachte Darstellung von heterogenen Diagrammen zu erhalten.

Unsere Methode wendet (Trasnfer knowledge) Methoden im Bereich des Bildverarbeitung an. Sie verwendet einen pre-trained CNN auf dem ImageNet-Datensatz, um einen Diagrammdarstellungsvektor zu erhalten, und verwendet Dimensionsreduktionsmethoden auf der Netzwerkausgabe, um alle Diagrammdarstellungen auf eine 2D-Ebene zu projizieren.

Wir evaluieren diesen Ansatz, indem wir ihn auf verschiedene Use-Case-Szenarien verschiedener Datensätze von Diagrammen anwenden, die die Projektionsergebnisse erklären und den Kontext der Abstandsmessung auf dem Ausgaberaum bestimmen.

Contents

List of Figures

List of Tables

List of Abbreviations

AE Autoencoder. 23

CNN Convolutional Neural Network. 3, 4, 5, 6, 9, 13, 15, 23, 24, 26, 27, 28, 29, 30, 36, 44, 45, 46, 47, 48, 49, 50, 53, 54, 55, 56, 57, 58, 59, 60, 62, 66, 68, 93, 94, 95, 96, 97

DR dimensionality reduction. 6, 21, 22, 29, 36, 37, 38, 39, 41, 45, 48, 49, 50, 52, 53, 54, 55, 56, 57, 58, 59, 60, 68, 93, 94, 95, 96, 97

ML Machine Learning. 5, 12, 13, 17, 18, 19, 26, 28, 36, 63

MLP multi-layer perception. 29

NN Neural Network. 27

PCA Principal Component Analysis. 6, 7, 8, 21, 39, 40, 41, 42, 45, 46, 47, 48, 49, 52, 53, 56, 60, 76, 98

QM Quality Metric. 12, 30

SGD Stochastic gradient descent. 50

SVM Support Vector Machine. 9, 63, 64, 65

t-SNE t-distributed Stochastic Neighbor Embedding. 42

UMAP Uniform Manifold Approximation and Projection. 6, 7, 8, 21, 22, 39, 42, 43, 44, 45, 46, 47, 48, 53, 54, 55, 57, 58, 60, 63, 77, 99

1 Introduction

In recent years, data has become the primary key to drive technologies and businesses in the world. It can describe everything in this world with different types and resources, such as books, temperatures, and even humans. This massive expansion in data quantity is due to the incredible evolution in internet technologies that connect everything in this world, sharing information from heterogeneous resources. The Bigdata motivates the researchers to find out approaches to organize this data for efficient selection, access, and analysis purposes.

Charts are excellent tools researchers use to describe the data. People prefer to use charts over tabular representations because it interacts with the sense of vision. This makes it easier to infer hidden information and patterns from the data, such as trends and relationships. We can use various types of charts in academic papers, journals, and newspapers for several purposes to analyze the data presented in them.

The massive increase in data amount and complexity leads to escalating the number of visualizations as well and complicating its design process. Visualizing the current data has big challenges on designers because of high-dimensional samples, complicated relationships, or a vast amount of data. Designers have to carefully observe all those factors to choose the best design decisions for readable and straightforward visualization. for example, if we have many categories with same amount of samples, then it would be incomprehensible when we describe it using a pie chart, as shown in figure 1.1[1]. We can barely know the topic or compare the ratios between the countries.

Those main challenges of modelling high-dimensional and complex data visualization are the root of the automatic visualization research field. As defined in DeepEye [LQTL18], automatic data visualization focuses on transforming the raw data from a dataset and deciding the right type of visualization. This process includes tasks such as selecting attributes, grouping, binning values, and selecting the best visual encodings.

Evaluating a chart needs to have metrics to compute the quality of it. For this purpose, the visualization community has developed Quality Metrics (QM) to criticize and analyze

[1] https://www.stevefenton.co.uk/2017/11/pie-charts-dont-belong-board-room/

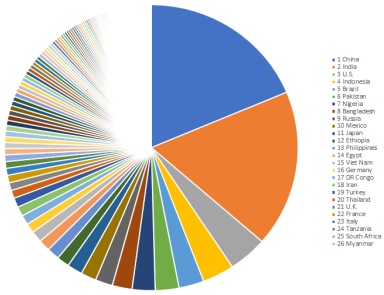


Figure 1.1: too many categories with same ratios makes the chart unreadable

data visualizations. The computation of a visualization's quality varies from measuring overlap in presented data to detecting specific visual patterns such as clusters, trends, and outliers. QMs allow us to select good visualizations without analyzing them manually.

The main problem of QM is dealing with different types of charts because researchers set those metrics for a particular chart type. Those metrics are manually calculated features by people for each chart, and it's complicated to extract them on a chart by chart basis. The two shown figures 1.2[2], [3] present a simple case. It is hard to compare those two charts in terms of quality because different design encodings were used to describe the data.
This problem limits the users from organizing the charts in a meaningful way. Consequently, it formalizes the following question: Is there a tool that can organize or explore these charts in a way that makes sense from a human perception perspective?
The question investigates the machine ability to extract and detect useful information to deploy them in high level tasks such as charts classification or visual pattern recognition. In order to solve this problem, we need to define a reliable method that read and understand the charts simulating human perception to automate the further applications.

Although there are still significant challenges regarding understanding humans' graphical comprehension capabilities and getting an accurate emulation of human perception, ML-based systems have many advantages. These automated system could simplify the evaluation of visualizations.

Using ML to evaluate visualizations, allowed us to think of even more exciting applications for the auto-users such as the organizing of data visualizations. The machine orders a set of visualizations regarding the quality score for a particular dataset, task, and visual encoding. In this context, we can define an optimization problem to find the

[2] https://community.cireson.com/discussion/3377/creating-more-complex-charts-in-the-cireson-portal
[3] https://www.ieltspodcast.com/academic-task-1-sample-essays/calorie-source-for-uk/

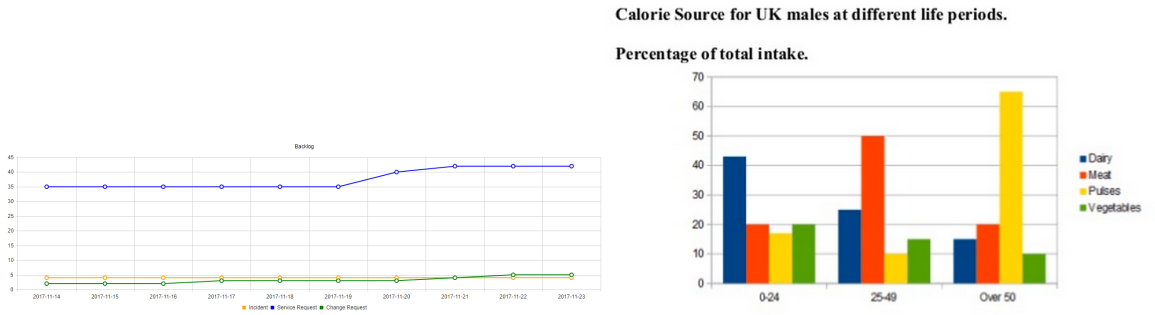


Figure 1.2: Two heterogeneous charts describes the data with different visual elements

visual encoding that has the best quality describing the specified dataset. This motivates us to discover good and reduced representations of data visualizations to simplify the process of searching and evaluation.

1.1 Problem Definition

So far, researchers in automatic visualization field were applying native analysis method on chart images to extract useful information and visual features. However, doing this task manually face the following problem:

- The analysis process is purpose-dependent
- the method is restricted to a specific type of chart that can not be used on other types
- Getting a stable method which considers all cases is time-consuming

In the image analysis field, researchers used deep learning and Convolutional Neural Network (CNN) techniques that aim to extract features of images. It forms a vector representation of them to use it for high-level tasks such as object detection, pattern recognition or classifications.
This project explores the following research problem: Can the machine learn to form a good representation of heterogeneous charts. These representations let the users compare and classify them from a meaningful human perspective. We aim to prove that using ML-based methods can extract features from images of charts, and replace quality metrics or other chart specifications extracted analytically from visualization. In figure 1.3 we introduce a simple prototype of the main process of the solution. The model can primarily process different type of charts and apply features extraction. At the same type, the model's output should have only one simple form for all charts, including the main

visual attributes. In this case, the model manage to overcome the hurdle of handling the various types as if it would transform these visualizations back again to a simple table.

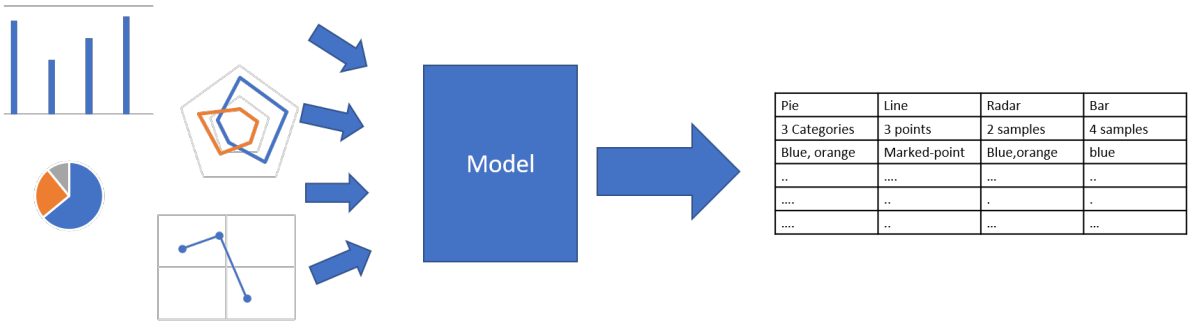


Figure 1.3: Learning a simple representation of various type of charts

1.2 Research Objectives

This project aims to apply unsupervised machine learning methods on image of charts to get a simplified representation from it. The reason for applying unsupervised methods to solve this problem is that the method is chart agnostic.
Our method follows an empirical approach, enabling to try several methods until we get two main methods as work contribution. In each step, we present the weak points of the results of every method to express the need for a more advanced method. Separating different types of charts evaluates our approach because it represents the ability to recognize essential visual elements. Starting with applying clustering algorithm, visualize, and discuss its results demonstrate the need of better methods. This process is repeated as shown in figure 1.4. The quality's results of each introduced method determines if we should start another sprint of exploring a new method.

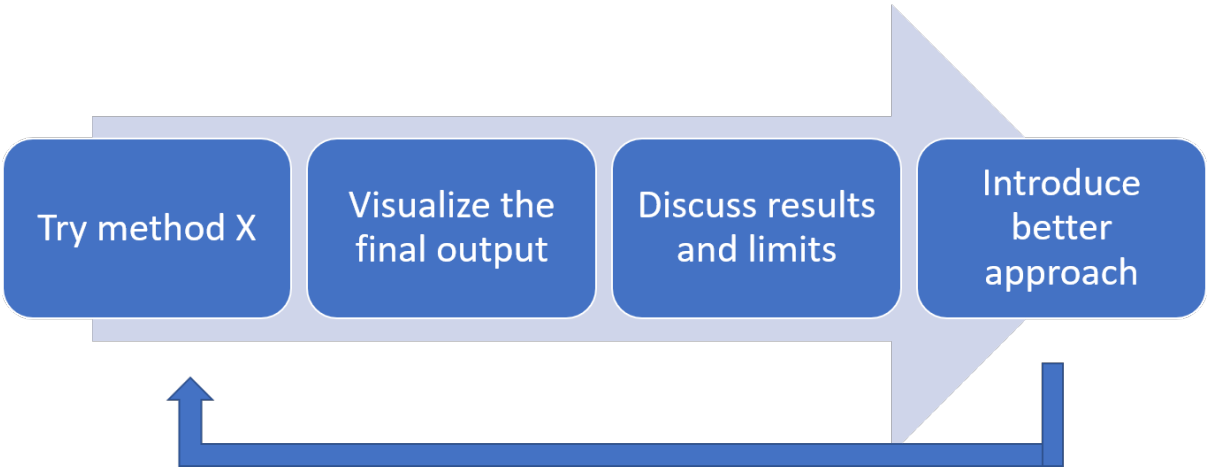


Figure 1.4: Discussing the method' results decides the end of the cyclic process

The primary method applies transfer knowledge methods in computer vision. It uses a pre-trained CNN on the ImageNet dataset to get a chart representation vector. Then we apply dimension reduction methods on the network output to project all charts representation on a 2D plane.
A second method uses the encoding part of an autoencoder to encode the charts in simplified representation. The autoencoder is trained on a dataset of charts collected from plotly platform for data visualization.
To evaluate both methods, we would analyze their outputs by using a classifier to perform a high-level task related to QMs, such as detecting the chart type or a specific used visual feature.

To sum up, our key contributions consist of the following :

- review and analyze the related works in the automatic visualization field, investigating their approaches and their goals.
- propose an independent-task approach to learn a representation of charts using an unsupervised machine learning method
- evaluate this method by using its output to train a classifier for a specific high-level task such as trend detection in line plots.

Structure

This project is divided into the following chapters:

Chapter 2 – Background: we introduce foundations of used techniques in this project.

Chapter 3 – Related work We introduce the latest research and developed technology on this topic.

Chapter 4 – Charts Collecting and Web Scraping we explain how we collect the data for training and testing.

Chapter 5 – Methods In this section we investigate the possible approaches that use unsupervised machine Learning.

Chapter 6 – Evaluation To evaluate the approach we test it through a set of experiments related to high level tasks.

Chapter 7 – Conclusion we give a summary of the whole work and results.

2 Background

In this chapter, we briefly explain all techniques of Machine Learning relevant to this project. All information in this section is cited from *Hands-On Machine Learning with Scikit-Learn & Tensorflow[Gro17], Advances in Deep Learning[Kha20], and Deep Learning with Python[Cho17b]*.
First, we introduce machine learning definition briefly. The second section explains all types of ML. Then, we describe the two relevant techniques in this project: Deep Learning & Dimension reduction in the last two sections.

2.1 Machine Learning (ML) History

ML can be considered as a young science. It became the most popular and most successful sub-field of AI, especially with the rapid evolution of hardware and collection of large datasets.

Computers' ability to learn to do some tasks without any further supervision from humans or even surpass our performance is the origin of ML. What makes it different from classical programming is that the machine would try to find out the rules by itself instead of explicitly given by humans through some logical conditions. The core of ML is calculating the rules from given inputs and outputs, as shown in figure 2.1. In simple words, machine learning is defined in [Gro17] as the science and art of programming computers so they can learn from data. Tom Mitchell, 1997 gave a further engineering perspective definition:

> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P,improves with experience E.

Machine-learning software explores the inner attributes of the given data and the statistical structure. The exploration process would help determine the relationship between the inputs and outputs so it can learn the pattern that determines how we could compute the right output from any input we might have. Unlike classical statistics, machine learning deals with large and complex databases that exceed statistical analysis

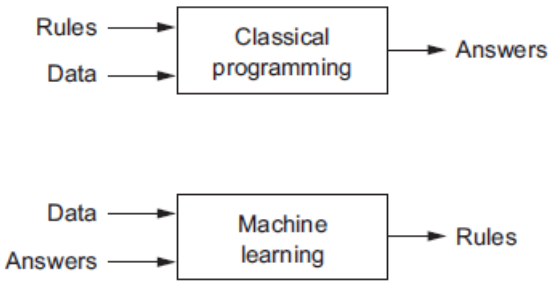


Figure 2.1: Machine Learning vs Classical Programming [Cho17b]

capability. Hence; we could consider this field as a combination of pure theoretical mathematical and engineering approaches.

Mathematically speaking, ML tries to optimize the system's outputs to converge the given results in the examples. If we assume that the Machine learning is a function $f : x \longrightarrow y$ our objective would be to choose the appropriate function f which returns approximately the same outputs of the Dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ so we could use it on some other examples later. Rather than explicitly programming all cases, it can be iteratively done by calculating the error difference in outputs between the generated one from our proposed function model ŷ and the actual output y. The training concept includes updating our function parameters iteratively until it can generate the optimal results approximately. The optimization process is defined in *Introduction to Machine Learning,Marc Toussiant* as :

$$f^* \arg\min_f \sum_{i=1}^{n} \ell^1(f(x_i), y_i)$$

2.2 Paradigms of ML

Hands-On ML[Gro17] divides ML into four types, depending on the system's human supervision level. We define them briefly and show which varieties we are using in the project:

Supervised Learning : the training samples come here in the form of (input, output) pairs. Usually, the output is called labels, denoting that the samples are annotated

[1]The Mean Square Error function is a popular example from statistics that is used as ℓ function.

with the desired results. A typical application of supervised learning is classification. The system should assign to which category is the object belongs computed from its given features. We list the most popular supervised learning algorithms:

- k-Nearest Neighbors (KNN)
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks

Unsupervised Learning : In contrast to the previous type, the data samples are not labelled. The system should analyze the features and attributes of samples. The analysis serves to detect hidden patterns in data and find out the relationship between samples. The primary applications of unsupervised machine learning listed as follow:

- Clustering: Splitting a set of samples into groups based on common attributes.
- Dimension reduction: finding a minimal representation of samples without too much loss of information. For example, we can reduce an object with 1000 features to 2D points.
- Association rule learning: discovering relationships between variables in large databases.

Other types are **Semi-supervised Learning** and **Reinforcement Learning**. Some other references consider Reinforcement Learning as a separate paradigm from Machine Learning. We will not explain both types because we do not use them in our approach.

Previous research used supervised methods, which are explained in the next chapter, because they labelled the data to solve their specified task. But here we evaluate the ability of supervised systems trained on natural images in finding representation of charts without labelling it as in the unsupervised case.

2.3 Dimensionality Reduction

Having too many features makes observations harder to cluster. Clustering uses a distance measure such as Euclidean distance to quantify the similarity between observations and this is a big problem in high dimensional spaces. For example, if the distances are all approximately equal, then all the samples appear equally similar as well as equally different, so the model can not form any representative clusters. This problem is referred to as the ***curse of dimensionality***[Gro17].

Dimension reduction aims to find a reduced representation of the raw data without too much loss of information. The system here analyzes the common attributes which have a small effect on sample identification. In our case, the samples are chart images. The background pixels with white colour values are not relevant, while the position and colour of lines in the line diagram decide the line's trend of the data.

Eliminating the unrelated features or dimensions, as in the projection case, also has challenges. Figure 2.2 shows a famous example of a swiss roll dataset. Projecting the dataset on 2D dimensions would squash different layers of the roll together. But what we need is to roll out this dataset on a 2D space[Gro17].
In order to achieve a good projection, there are primarily two types of reduction techniques: linear dimensionality reduction that transforms the data as a linear combination of the original variables and nonlinear reduction that is applied in case of nonlinear relationships between the data. Linear dimensionality reduction is applicable when the data lies in a linear subspace, while it shows a poor performance when it is applied on nonlinear correlated data[SS15].

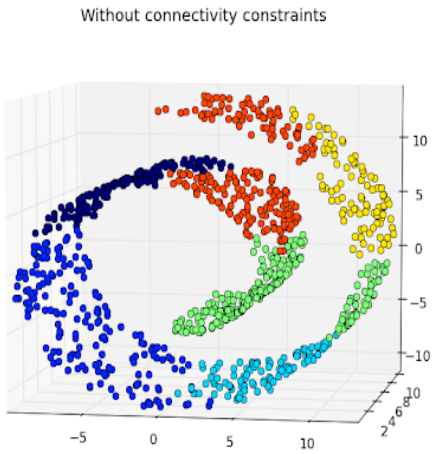


Figure 2.2: Swiss role dataset [20b]

We applied DR techniques to the neural network outputs to visualize charts' representations by reducing it to 2D points for each chart. In the next two sections, we explain two of DR techniques we used in our method which solve the previous problem.

2.3.1 Principal Component Anaylsis (PCA)[Jol11]

PCA is the most popular linear DR algorithm. It recognizes the hyperplane that lies closest to the data to project the data onto it. The selection of hyperplanes is based on preserving the maximum amount of variance between the samples, so that it will lose less information as possible than other projections. It is usually preferable to choose the number of dimensions that keep a high proportion of the variance (e.g., 95%).
In higher-dimensional cases, PCA would find a further axis, orthogonal to the previous axis, and so on till the number of dimensions in the dataset[Gro17]. Every unit vector that defines the i-th axis denotes the i-th principal component (PC). The following figure 2.3 shows a simple projection of 2D space into 1D. The first axis is the best choice as it preserves the maximum amount of data variance.

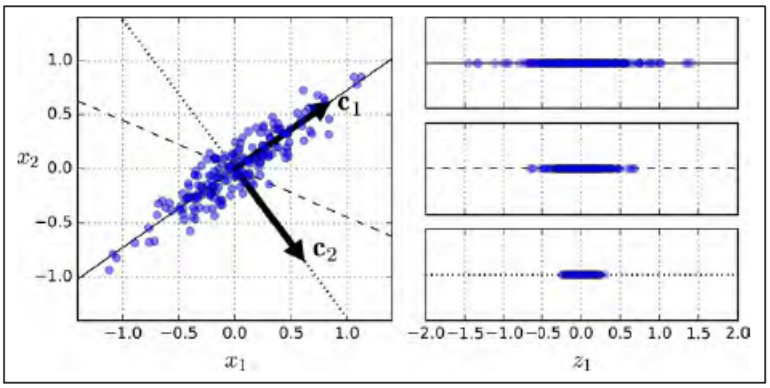


Figure 2.3: PCA determines the orthogonal axises [Gro17]

2.3.2 Uniform Manifold Approximation and Projection (UMAP) [MHSG18]

UMAP is an algorithm for dimension reduction based on manifold learning techniques and ideas from topological data analysis. Simplicial complexes are the basic modules that UMAP built on it. Geometrically a simplex is a very simple way to build a k-dimensional object. Simplices can provide building blocks, but to construct interesting topological spaces, we need to be able to glue together such building blocks. The tool we will

consider is the construction of a Čech complex given an open cover of a topological space. An open cover is essentially just a family of sets whose union is the whole space, and a Čech complex is a combinatorial way to convert that into a simplicial complex. We take this approach to get a topological representation then we can build a dimension reduction algorithm by finding a low dimensional representation of the data that has a similar topological representation. The following figure 2.4 describes a simple case of simplicial complex construction from a set of points.

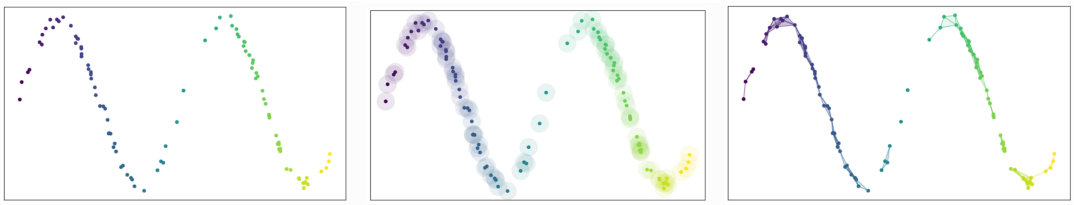


Figure 2.4: construction of simplicial complex from point dataset[20c]

The UMAP is faster than other DR techniques such as t-SNE and provides better scaling. It allows for generating high-quality embeddings of more massive data sets than had previously been achievable. The use and effectiveness of UMAP in different scientific fields expresses the strength of the algorithm.[MHSG18]

2.4 Deep Learning

The Neural Network is a new subfield of machine learning inspired by the biological neural network. This system solves the machine learning problems by processing the raw input data through neurons that can pass information with each other to get the desired output. A network consists of multiple neurons that can be activated to compute a piece of feature extraction task. Those neurons are arranged in stacked layers based on the abstraction level. The neural network is the core of the deep learning field.

Deep learning[Cho17b] points to the deep architectures containing multiple hidden layers to learn different features with multiple abstraction levels. Deep learning explores the unknown structure in the input to discover desired representations with higher-level learned features illustrated from lower-level features. Deep learning involves learning these features automatically during the training rather than defining the set of feature extraction rules. Figure 2.5 shows how a set of neurons stacked together that pass their outputs to the next layer until we get the final solution of the given instance.

The input is transformed into multiple representations through a series of layers. Every layer applies the transformation to its input through calculation with a set of numbers

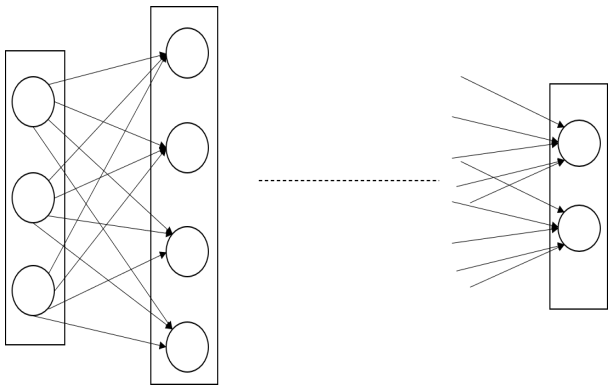


Figure 2.5: An example of deep learning architecture shows how neurons (circles) are connected to each other ordered in a set of stacked layers

called layer weights, so the training process concerns finding the values of the weights of all layers in the network so that input samples are correctly mapped to their labels. The loss function optimizes this training process by calculating the difference between generated output and the actual target. Each difference makes an adjustment signal for the weights to low the score of the loss function. This improvement is made by the optimizer—back propagation algorithm, the central algorithm in deep learning.

2.4.1 Convolutional Neural Network (CNN) [KSZQ20]

CNN is a deep learning technique that is widely used in computer vision applications. Its main characteristic is to get local features from the input at lower layers and combine them to detect complex structures at the higher layers. When the neurons are not fully connected but slides over the input, they can extract local features in every part of the input. Figure 2.6 illustrates a CNN architecture that classifies an input image on its type. Popular architectures are AlexNet, VGG, and ResNet, which we explain further in the approach section.

Our first method is evaluating the CNN architecture trained on natural images from ImageNet on extracting features from charts.

2.4.2 Autoencoders [Gro17]

Autoencoder (AE)s are one of the unsupervised learning techniques. It can learn efficient representations of the input data without any supervision from humans. The autoencoder transforms the input into a much lower dimension encoding. That is why autoencoders are considered useful for dimensionality reduction. AE aims to transform the input

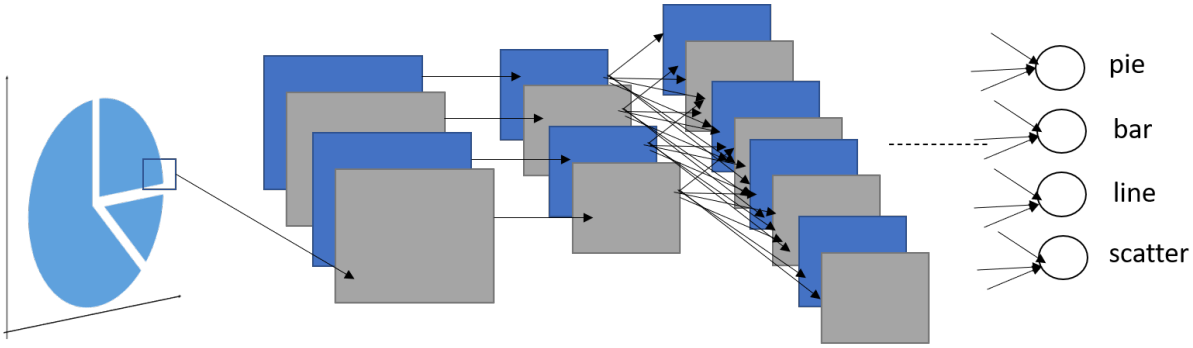


Figure 2.6: CNN takes the pixel values of the images and processes it through a set of deep layers to extract important features

vector into a low-dimensional encoded vector and then reconstruct the input data from the corresponding code with minimum reconstruction error. The encoding involves learning features by extracting useful features and filters undesired information. More important, autoencoders can have CNN architecture when the inputs are images.
We used also autoencoders to find a minimal representations of charts in our approach

2.4.3 Transfer Learning [TS10]

Neural networks and deep learning aim to process the data inputs through multiple layers before performing the main task. In recent years, most researches used pre-trained CNN models from prior works to handle other tasks. These models are trained on large datasets such as ImageNet. This concept is called transfer learning that intends to transfer knowledge gained while solving one problem and applying it to a different related problem.

The principle of transfer learning is to remove the top layer, that is related to the specific classification task, and adding a new layer to do the new task, as shown in figure 2.7. In this case, we need only to train this top layer and take the weights of other layer gained from the training on the previous task. This technique could highlight the relationship between different tasks or to solve problem where images are not labeled.

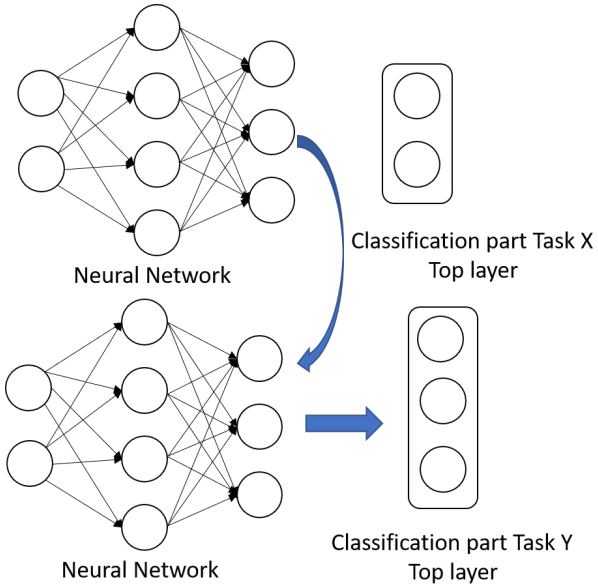


Figure 2.7: transferring the gained knowledge from Task X to Task Y

3 Related work

Our work serves to find a proper representation of charts for different purposes in the automatic visualizations field. We structured our research by exploring at first automatic visualization applications that use ML-methods. Then we look at evaluation studies of CNN on performing on chart images.

3.1 Automatic Visualization Applications

This section lists the related works in the automatic visualization field, outlining their goals and approaches. This would enable a deeper understanding of the motivation behind this work and why this contribution is so important. We summarily describe the related works grouped by their goals and their relatives to our work.

3.1.1 Visualization Redesign & Recommendation

One of the main contribution in this field is the redesign or recommendation of a given chart with better visual features for a better interpretation and chart's perception. We list the main relevant works in this area. We demonstrate the limitations of the methods used in the first two works to extract visual features and explain possible use cases with other two projects.
Revision[SKC+11] proposes a system that automatically redesigns visualizations to enhance human perception. It applies computer vision and machine learning techniques on the chart to identify its type (e.g., pie chart, bar chart, scatterplot) and then extracts the graphical elements and infers the described data. The images are preprocessed by randomly sampling a set of patches and vectorize them as inputs to ML model. In this case, the system does not learn a full representation of all features in a chart, but it reaches an 85% accuracy in chart type classification for this specific task. The main weakness in this study is that they take random patches of images that does not support full feature extraction of its features.
Converting Basic D3 Charts into Reusable Style Templates[HA18] aims to construct a style template from extracted underlying structures as the data, the marks, and the

visual encodings. It obtains important ranks for each new data field then adjusts the template mappings to represent the source data by matching the most critical data fields to the most perceptually useful mappings. The approach were undermined by implementing it only for D3-based charts. It means that we can not use it for other format chart images.

VizML[HBL+18] introduces a machine learning-based approach to visualization recommendation that learns visualization design choices from a vast corpus of datasets and their visualizations. The NN model predicts these design choices trained on one million dataset-visualization pairs collected from a popular online visualization platform. However, this model is trained on extracted visual encodings in JSON format, not from chart images. VizML approach seems to be well-grounded to integrate with other projects to build a complete end-to-end model trained for recommendation purposes.

Voyager 2[WQM+17] is a new mixed-initiative tool that blends manual and automatic chart specification in a unified system. Analysts use the wildcards concept to create multiple charts in parallel or related views that automatically recommends charts based on the current user-specified focus view. Finding a reduced representation of charts would make the work process of this work much more effective.

3.1.2 Data and encoding extraction

The following projects aim to develop a method for data and visual encoding from charts images. They classify the chart based on their type to expect which visual elements they need to handle.

Reverse-Engineering Visualizations[PH17] contributes an end-to-end pipeline to infer visual encodings from bitmap images. Basically, it detects text elements in a chart, classifies their role (e.g., chart title, x-axis label,y-axis title, etc.), and recovers the text content. Besides, it trains a CNN for mark type classification. The system uses the identified text elements and graphical mark type to infer a chart image's encoding specifications. The specification extraction from charts in this project is very plausible. However, the main disadvantage of REV that it describes information only related to the found text in the chart such as x-axis values or title.

Chartsense[JKS+17] is an interactive chart data extraction system. Similarly to REV[PH17], it detects the chart type using deep learning and extracts underlying data from the chart image using semiautomatic, interactive extraction algorithms proper for each chart type. Chart data extraction results have been used in several applications, such as chart redesigning, generating helpful overlay for a chart, mapping a text to a

mark based on crowd-sourcing, and aiding search and retrieval of chart images. Chartsense uses analytical methods to extract all data and features from the chart. This raises the question if ML-based methods would be more efficient.
Figureseer[SHL+16] localizes figures from research papers, classifies them, and analyses the content of the result-figures. This work's primary focus is to extract the plotted data and its association with the legend entries. This challenge is solved by formulating a novel graph-based reasoning approach using a CNN-based similarity metric. Figureseer seeks to parse a vast amount of charts that can be saved in a dataset. Finding minimized representations of those charts seems to be a innovative approach to improve the performance of searching and selection process.

Beagle[BDM+18] mines the web for SVG-based visualizations and classifies them by type (i.e., bar, pie, etc.). Beagle reaches an 85% accuracy across 24 visualization types. The main downside in this work, as mentioned before in D3 re-styling[HA18], that it deals with only a specified type of images that makes it harder to use on other images.

3.1.3 Visual analysis and charts retrieval

Due to the vast increasing in chart's amount, researchers try to organize those charts in a way they can obtain or split the charts based on a given features such as colour or trend in line.

Searching the Visual Style and Structure of D3 Visualizations[HA20] presents a similar approach to the previous work for D3 visualizations that allows visual style and underlying structure queries. Unfortunately, this search engine indexes this style and structure information and metadata about the *webpage* containing the D3-based chart.

DeepEye[LQTL18] presents a system for automatic data visualization that can rank and compare visualizations. The system works using a supervised learning method to rank visualizations and expert rules that specify partial orders for the top-k visualizations. An interesting case if the system can rank the output representations of the charts from our method results. The new input form enhance efficiency of the DeepEye system.

Computerized Detection of Trends in Line Charts[NL] attempts to detect the trend from a line graph splitting into three simple labels(increasing, decreasing, or neutral). It compares traditional analysis vision techniques and deep learning methods to classify graphs showing one of the mentioned labels trends. We intend to analyze if representations with unsupervised methods describe such related features.

ScatterNet[MTW+20] proposes an approach for modeling subjective similarity by employing human visual comprehension information. The principle is to use human

labels based on similarity measures on scatterplot as training data. It utilizes state-of-the-art deep neural networks to get features from the plot images. Those features are used for computing similarities by computing the euclidean distances between the hyperplane's final representations. Consequently, ScatterNet can effectively measure similarities between scatterplots by considering human supervision. Our steps proceed very much in the same way as what is indicated in ScatterNet. We seek also to project the chart representations on a 2D space to compare and analyze them but for several purposes.

SepEx[BHZ+20] addresses the visual analysis of class separation measures for high-dimensional data (e.g., 5D). SepEx is an interactive visualization approach for the assessment and comparison of class separation measures for multiple datasets. This work intends to compare multiple separation measures over many high-dimensional datasets, DR on measure outputs by transforming nD to 2D, and comparing the effect of different DR methods on measure outputs.

[LWL+20] uses the concept of Visual Information Flow(VIF), that is the underlying semantic structure associating graphical elements to obtain the information for the user. It applies deep neural network to identify visual elements related to those information that is not related to their various artistic appearances. Using this analysis, this paper characterizes the VIF design space by a taxonomy of 12 different design patterns. The main limitation point in this project is that visual information flow does not form a familiar pattern with an identified flow signature. Also, the VIF design space is analyzed with some user assistance. An interesting study is to show if the reduced representation resulted from our method can help detecting the defined pattern in this project.

3.2 Evaluation of CNN architecture used on charts

Several works aim to evaluate the performance of CNN-based methods on several tasks. We mention in this section two studies analyze different CNN architectures on classifying the chart type. This demonstrates the efficiency of the CNN architectures for our purpose. **Evaluating Graphical Perception with CNNs**[HTP19] tests three different CNN architectures shown in the table3.1. Basically, it uses a multi-layer perception (MLP) applied on charts directly or on outputs of CNN architectures. Those models were used in specific experiments to prove hypotheses related to perceptual tasks such as element recognition, positions angels, and length. These low-level tasks could be the baseline for chart classification, as explained in Hypothesis 1.1, that says that the CNNs can regress quantitative variables from graphical elements.

Network	Trainable Parameters
MLP	2,560,513
LeNet + MLP	8,026,083
VGG19 + MLP	21,204,545
Xception + MLP	25,580,585

Table 3.1: Models introduced in [HTP19]

Evaluation of Convolutional Neural Network Architectures for Chart Image Classification[CAM+18] aims to evaluate and compare the CNN architecture with other classifiers. With a generated dataset of 10 classes of chart images, three different CNN architectures (VGG-19, Resnet-50, and Inception-V3) are compared with conventional classifiers (HOG features combined with KNN, Naive Bayes, Random Forest, and Support Vector Machine). Interesting is that parameter weights come from ImageNet pre-training, assuming that training on the natural images is even better for general problems and can avoid overfitting.

3.3 Quality metrics

The project aims to cluster a set of charts based on high-level tasks. One of the interesting potential use-cases is to rank those visualization to check the clustering results based on the ranking. In this context, we are able to study the effect of changing visual elements on the chart's quality score. A formal definition of Quality Metrics is introduced in this survey[BBK+18]. It reports and presents a commonly applicable quality metric formalization that explains all constituting parts of a Quality Metrics.

4 Charts Collecting and Web Scraping

In this section, we introduce the three most popular data visualization communities to collect charts from them. Then we explain why we chose plotly as our primary data resource. Another dataset we have is a collection of figures and tables from IEEE visualization conference publications. We made up a quick statistical overview of the data we collected. In order to crawl plots from plotly community, we implement a script that grabs charts through REST API and extracts useful information from the associated encodings in the JSON file.

The three primary platforms, in which people can share and save plots, are listed as follows:

- **Plotly**[20d] library is an interactive, open-source plotting library that lets users create over 40 different chart types offering a wide range of design options. Those interactive visualizations can be displayed in Jupyter notebooks, saved to standalone HTML files, or used in python web applications using Dash.
- **Tableau**[20e] is a visual-analytics platform that transforms data into useful interactive visualizations. It motivates people and enterprises to make the most of their data as HTML files or serve as part of Python-built web applications.
- **D3**[20a] is a JavaScript library for manipulating documents based on data. D3 uses HTML, SVG, and CSS to generate interactive charts. Utilizing web standards allows using modern browsers' full capabilities without tying to a restrictive framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Each of these tools has its online community so that anyone can create visualizations using it. With millions of data visualizations, people can see and download them about any topic or from any publisher. This free access let us grab thousands of plots for our purpose.

We chose Plotly as our main resource for two main reasons:

- plotly community offers simple communication through REST API endpoints. These endpoints let the users view, download any published data visualization, or edit the charts to automate charts collecting.
- For each chart in Plotly, we can obtain different files: 1) the chart image, which is our model's input. 2) a JSON file containing all the chart's visual encodings that we use it to obtain required labels in evaluation. 3) described data in the CSV file.

Another dataset we have is all visualizations in IEEE conference publications[1]. It contains a wide range published between 1990 and 2019, varying from simple scatterplot to complex medical analysis images that consist of more complex shapes and objects than ones collected from plotly. We would analyze the common features of those visualizations through our model.

PlotlyCrawler : An automated approach to download plots from plotly community

The primary endpoint is GET /v2/plots that returns a list of 10 plot details. The JSON responses are linked as a series. Every response points to the next endpoint in the key "next", as shown in figure 4.1. From the JSON response, we get the file id to obtain the three files (png,JSON, and CSV) from the GET file endpoint.

Figure 4.1: A serie of JSON responses of REST endpoints

PlotlyCrawler sends a get request for the first list of plots. Then, for every plot, it starts a thread to download and save the three required files. The program splits the saved charts into 20 different generated folders for an efficient access later. After finishing all files, it checks the next list and repeats this process until we stop the program, or the linked series comes to an end. Figure 4.2 explains the pipeline of this program. The program's main hurdle is the time taken to write the files in the proper type, such as png or JSON...

[1] https://ieee-dataport.org/open-access/vis30k-collection-figures-and-tables-ieee-visualization-conference-publications

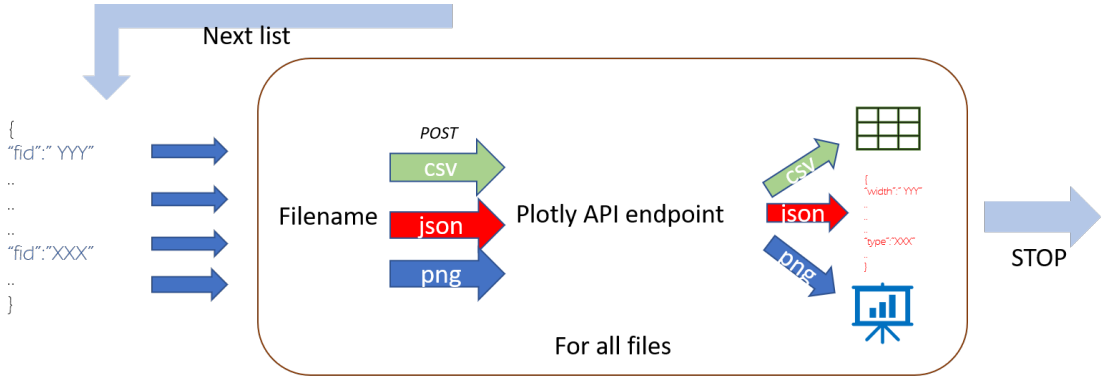


Figure 4.2: The workflow of PlotlyCrawler

Collected data

We collected 32,912 charts from plotly over different type. Figure 4.3 shows the distribution of chart types of the collected chart. While the number of scatter plots reaches to 22,380 samples in the dataset, some other charts such as radar has only 115 samples or heatmap that includes 1015 collected samples.

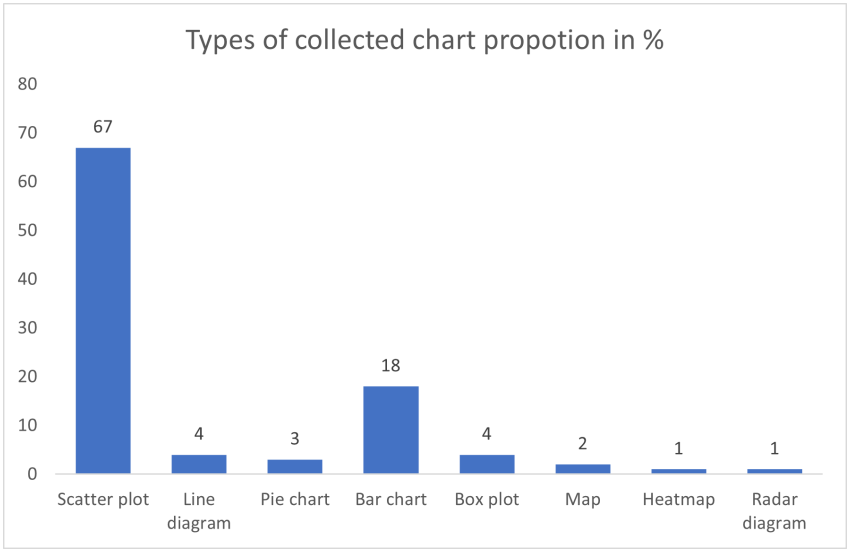


Figure 4.3: Scatter plot is the most frequent type found in the collected dataset

The collection from conference publications contains 29,549 visualizations. Each year folder involves 985 data on average, from 463 data in 1990 to 1456 file in 2019. Compared to the collected charts from plotly, this dataset includes various types of visualizations with some random images such as text snapshots. We made a random sampling from each year to represent this dataset, collecting approximately 1400 samples.

In figure 4.4 we describe the main found types in this sample. Compared to plotly dataset, we notice that all types here share a similar proportion.

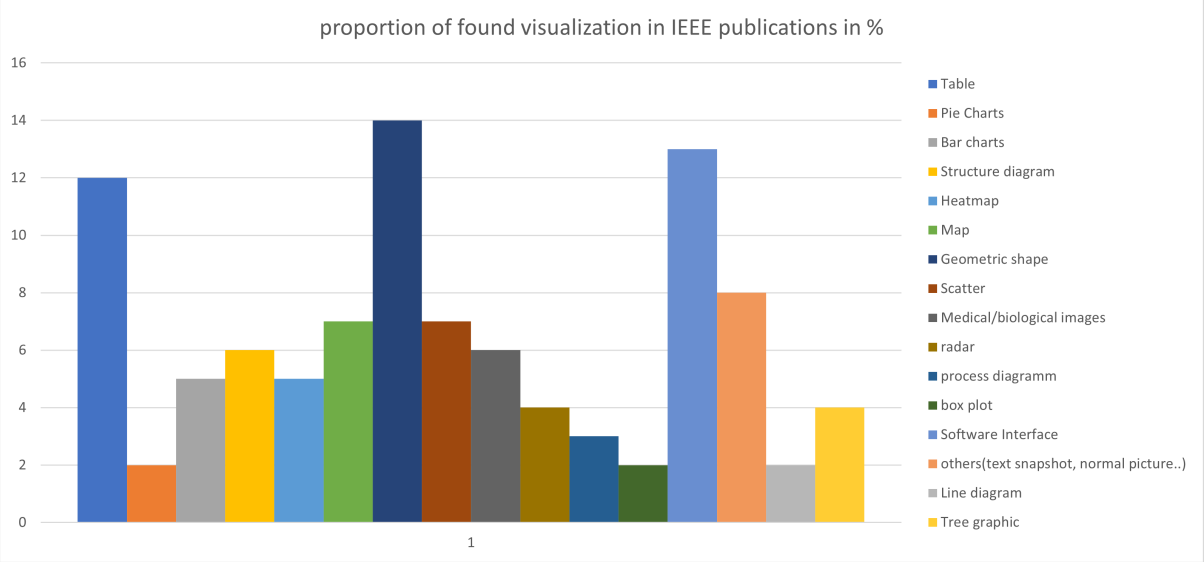


Figure 4.4: publications include many visualization types or random pictures

JSON Squirrel: Extraction of information from JSON files

The information in the JSON file describes the visual features and encodings used to design the chart. It consists of two main objects: data and layout. Data object describe all the visual features used to describe the data such as the type, or chart title, while layout include all attributes of the visual elements like the colour or size of the lines or the bars. We can use this information in the evaluation process to label a set of charts regarding a specific design perspective, such as chart type. To obtain the visual encodings, we intend to implement a tool that can search and get those values from the JSON file.
The main challenge is the diverse structure of the JSON files. Charts have their unique encoding way defined by different users. As an example, we can not define a fixed position for the attribute "type" in the JSON file. It depends on the chart's design complexity. A complex type ,containing much more element than a simple line diagram, would have more nested objects in its JSON. Another problem is the duplicated appearance of an attribute in different contexts. The attribute "title" for example, can describe the x-axis and the y-axis label. In this case, We also need to know what those values are describing and which one to select. That is why we should implement a generic tool that can extract a given key's value without any prior knowledge of the JSON structure.

We propose JSON Squirrel, a tool that looks up for a list of keys in JSON files and returns their values using jsonpath-rw[2] library. The workflow of the program is based on the Algorithm 4.1. The program takes the folder path for all JSON files, the list of keys we want to obtain their values, and the output file's name. For every JSON file in the given folder, it checks for each key all possible path from JSON root object till the key and obtain all those values, saved in a list and then write it in the final report.

Algorithmus 4.1 Key values extraction algorithm

```
Result: Output file contains all values found in JSON files of given keys
Input: destination path,list of keys,output file's name
for all files in the given path do
    for all given keys do
        list_of_all_paths_from_root = extract_all_paths_for_given_key(JSON file)
        for all paths in the list do
            value = extract_value(JSON file,key)
            output.add(key,value)
        end
    end
end
return output
```

[2] https://pypi.org/project/jsonpath-rw/

5 Methods

In order to obtain a useful encoded representation of charts, we apply unsupervised machine learning on charts' images. For this purpose, we implement three different approaches. In the first section, we use K-means and DR techniques directly on chart images, while the other two methods demonstrate the role of deep learning on charts. The second approach applies transfer knowledge on CNN architectures that are pre-trained on ImageNet dataset and we use the autoencoder in the third approach to encode the chart image. In the last two sections, we discuss the limitation and the advantages of every method demonstrating its performance when we apply different conditions on the input dataset. Figure 5.1 summarizes all methods and algorithms we used in this project.

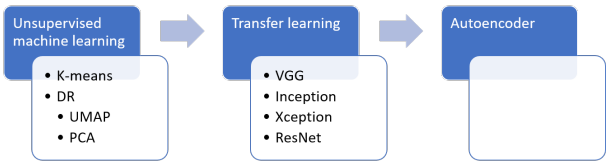


Figure 5.1: Summary of all used methods and techniques in this project

All code implementation used Tensorflow[MAP+15] and Scikit-learn[PVG+11] as dependences to apply ML methods. For results' visualization, we used primarly Matplotlib to generate the scatter plots.
All of introduced methods were applied first on 400 set of charts of four main types: (bar chart,pie chart,line diagram and scatter plot) for the following reasons:

- Choosing massive datasets with thousands of charts denotes more features and more charts that represent further nested cases that makes it hard to analyze and demonstrates the role of multidimensional visual features

- Applying the method on thousands of charts makes it harder to visualize the results and unreadable. We can not also visualize a subset of the projected result because it would hide important factors, as will be discussed later in this chapter.
- Selecting low amount of charts does not clarify the effect of visual attributes on the cluster embedding on the given charts. However, in some cases we could use only tens of charts based on the purpose of the projection.
- Those four types are the most frequent types found in the dataset, as explained in figure 4.3

Usage of unsupervised machine learning

Prior research in automatic visualization used supervised methods and neural networks. Supervised methods intend to optimize the model's parameters to solve a specific task. In contrast, unsupervised aims to discover the form of samples and generate their labels. We used unsupervised methods primarily for two reasons:

- **Data labeling**: papers such Revision[SKC+11],REV[PH17], and trends detector[NL] labeled their data samples according on high level task as type or trend classification. Labeling affects the feature extraction process to increase the trained model's accuracy but only related to the defined task. For example, suppose we label the data according to the chart type. In that case, the final model focus only on detecting the geometric shapes such as circles for pie or rectangles for bar and ignore other attributes like the number of partitions or the number of bars.
- **automating the labeling process with multidimensional classes**: Assuming we obtain a complex form of labels that describe multiple perspectives of the chart. In this case, the labeling process would be time-consuming and hard to automate. It needs to analyze each chart's features that is the same motivation for our project.

5.1 Unsupervised Machine Learning on images

In this section we present two of the main unsupervised methods. The first part examines the performance of applying K-means clustering algorithm when it is applied on the pixel values of the charts dataset. The results of applying K-means highlight the important rule of dimensionality reduction in image processing that we use in the second part of this chapter.

5.1.1 Applying K-means algorithm

K-means clustering algorithm groups the data by separating samples in a specified number of clusters of equal variance. It aims to minimize a measure known as the inertia or within-cluster sum-of-squares. Scaling well to a large amount of samples, K-means has been used across a broad range of applications in various fields.
The results shown in figure 5.2 demonstrate that the algorithm is hugely affected by the frequency of the pixel values, whatever they might represent. It means that K-means split the charts based on the common number of pixels that share the same value. For example, the second cluster includes all charts with the black background, while the third recognizes the spike lines. However, we can still notice some random charts that do not denote any related features to other charts in the same group as putting pie charts and the wave-broken line diagrams together in one cluster.

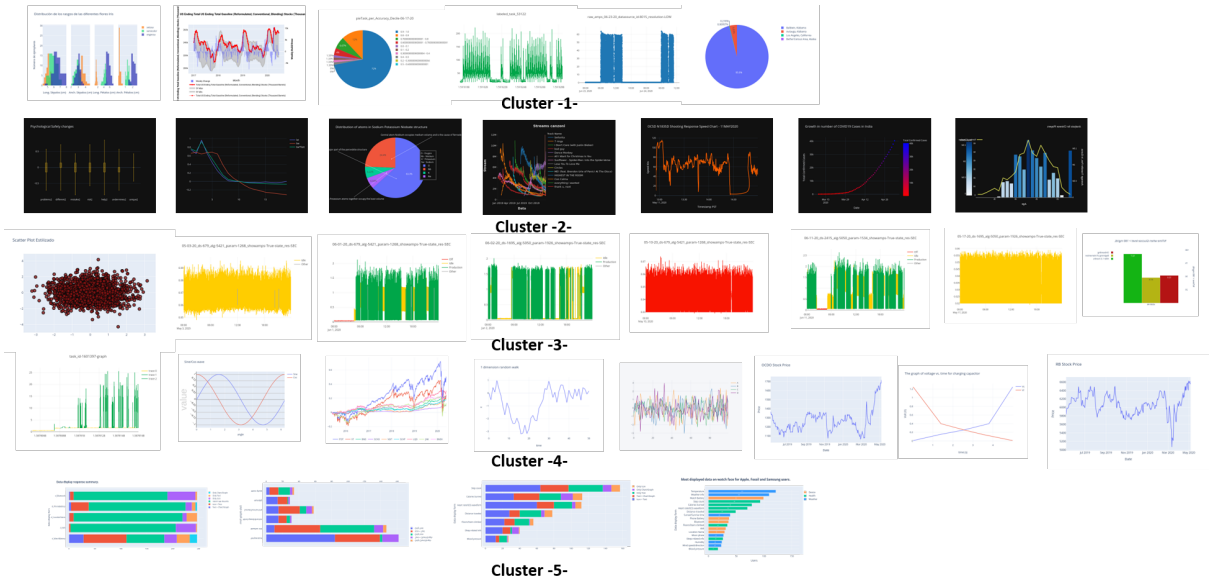


Figure 5.2: Applying k-means on a random selected group of charts

5.1.2 Applying Dimension reduction

DR techniuqes aim to eliminate irrelevant dimensions in data samples as in images case. Using this method in our project seems to be effective because charts are much more simple than natural images since it does not include too much details. In this section we apply two main DR methods: PCA and UMAP discussing the different results of both techniques.

Although PCA is still the most used technique over all projects as mentioned in this DR techniques survey[JT20], we show in this project that UMAP gives better results as a novel manifold learning technique.

Advantages of using dimensionality reduction

The main limitation in clustering algorithm is the nonrecognition of charts' features. One of the main reasons to the inefficiency of Clustering algorithm is the cure of dimensionality. One image is represented as a numerical matrix of $(200 \times 200 \times 3)$, which means 120000 elements as the pixel values. Most of those pixel values do not have a critical role in representing such a relative visual feature such as the backgrounds of the chart. Hence, it is really helpful to apply this technique on charts image to extract the most important dimensions.

Principal Component Analysis (PCA)

Although PCA is one of the most techniques, it does not deliver good results in some non-linear-correlated cases. For example, in our case PCA does not split the different types of charts when applied on four different types. In figure 5.3, most of the charts are located in the lower left of space that can not represent the clustering depending on the type, while charts with non-white background are isolated from other charts locating in the upper-right part of the subspace. The original size image of projection is in figure A.1

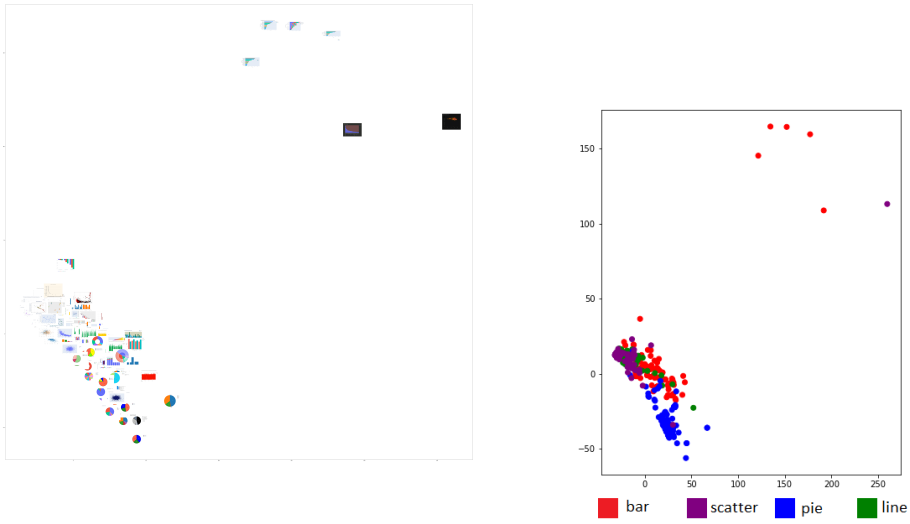
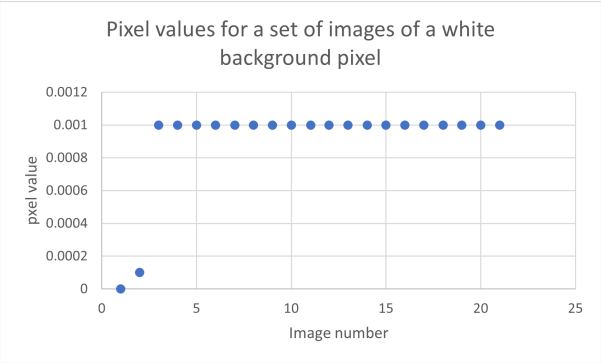


Figure 5.3: applying PCA directly on images pools all charts in one spot together

This bad separation is explained through PCA concept. We present two simple cases of the workflow of PCA on charts in figure 5.4. In figure 5.4, we observe the pixel values of different images on the same location. The values seem randomly distributed because images does not have any correlation in colouring the same pixel with related values. Since PCA focuses on finding orthogonal projections of the dataset that contains hidden **linear** correlations between variables of the dataset, it means that PCA can find directions that represents the linearly correlated data. This denotes that PCA is not efficient on this dimension because there is no liner correlation between data. Figure **??** presents a simple case of white pixels such as background pixels. When projecting the horizontal line on other sub-spaces, it appears as a one point, so that PCA can eliminate this dimension easily.

((a)) PCA finds no linear correlation between data values



((b)) PCA eliminates the dimension of white pixels

Figure 5.4: PCA eliminates two different dimensions

A useful measure to evaluate the projection is the explained variance ratio of each principal component. It intimates the proportion of the dataset's variance that lies along the axis of each component. Usually, It is preferable to choose the number of dimensions that reach a sufficiently large portion of the variance, such as 90% or even 95%. We find out that the number of required dimensions to get 95% explained variance is 159. This result highlights the critical role of DR which reduces the dimensions to 159 from 120000 in PCA case.

We observe the sum cumulative percentage of data's variance, starting from reducing to 2D dimensions until we get 95%. The elbows in the curves in figure 5.5, where the explained variance stops growing fast, can be considered as the intrinsic dimensionality of the dataset. In this case, we can assume that reducing the dimensionality from the curve's start would not lose too much explained variance.

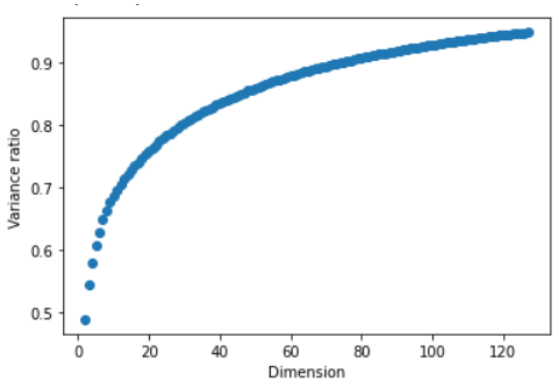


Figure 5.5: summing up the explained variance ratio of applying PCA on n-dimension

In conclusion, we proposed using PCA as a reliable approach to reduce the chart representation and remove unrelated dimensions. However, PCA have limitations in case of non-linear correlations. For this reason, we apply UMAP, that outperforms t-SNE algorithm, on charts and compare the results.

Uniform Manifold Approximation and Projection (UMAP)

Compared to PCA, UMAP splits better between the four main types of the given charts, as shown in figure A.2. This technique also preserves reasonable distances between the types that comprises a decisive 2D space to compare and analyze the similarity between charts. However, UMAP has still some limitations that affect the clustering between the types. Figure 5.6 clarifies this problem when bar,scatter, and line charts are overlapped in the middle cluster.

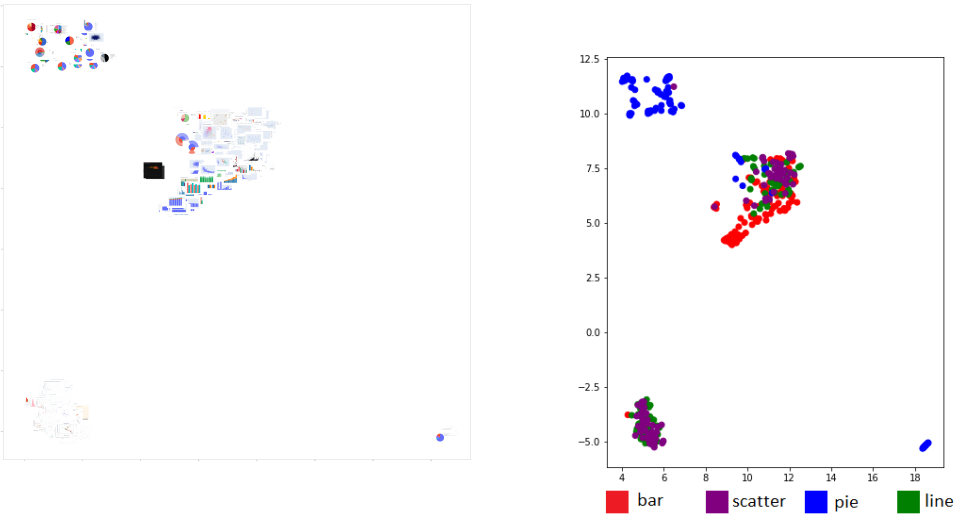
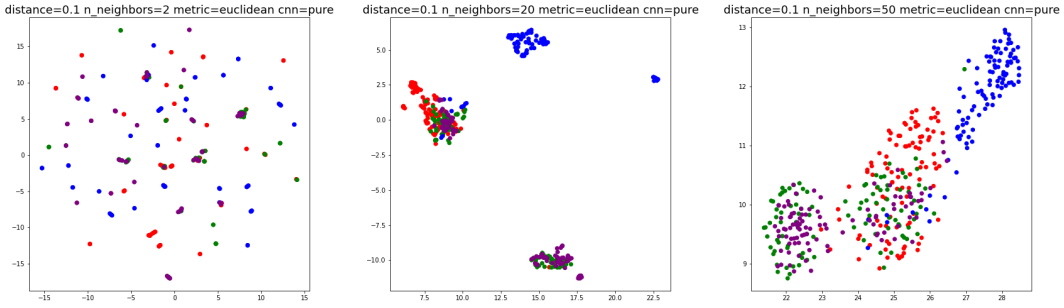


Figure 5.6: UMAP splits between pie,bar but mixes line and scatter plots together and with bar charts.

UMAP has various hyper-parameters that have a significant impact on the resulting embedding. Each of these parameters has a distinct effect, and we will analyze it each in turn.

- neighbors
- minimum distance
- metric

neighbors The neighbors parameter controls the balance between the local and global structure in the data. This parameter's value determines the local neighborhood's lookup size when the algorithm detects the manifold structure of the data. Low values force UMAP to focus on very local structure, while large values drive UMAP to look at larger neighborhoods of each point, losing fine-detailed structure to get the larger of the data. For example, figure 5.7 shows the effect of different values of neighbors parameter. Starting from the value of 2 (left in the figure), it seems that the algorithm finds every chart's twin regardless of the others. Giving a medium value such as 20 (middle in the figure) has a good performance on detecting common shapes that classify the chart's type. Higher values as 50 (right in the figure) would try to determine the association between the different types of charts because it focuses on detecting global patterns over a huge set of charts.

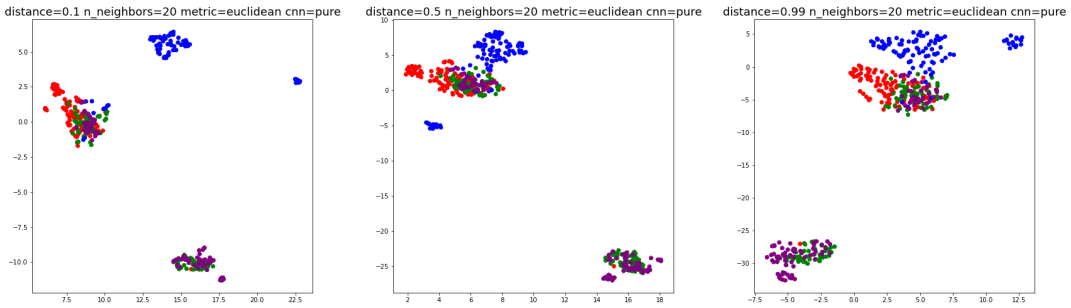


((a)) 2 **((b))** 20 **((c))** 50

Figure 5.7: increasing the neighbors parameter values of UMAP affects the clustering embedding.

• bar • line • pie • scatter

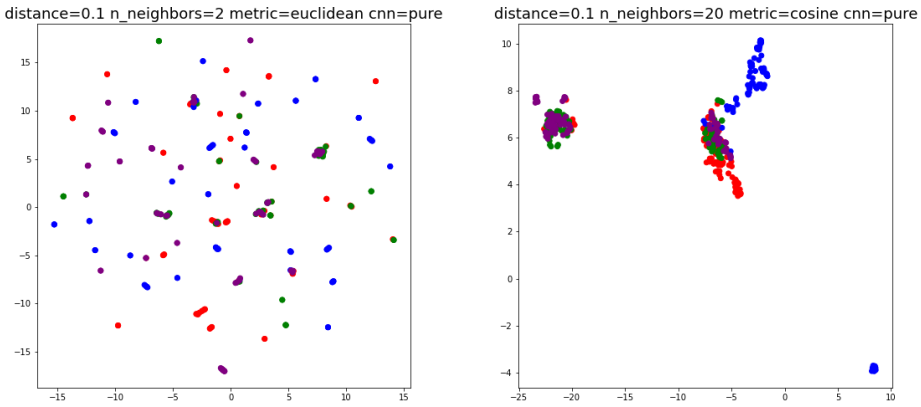
minimum distance This parameter controls the level of packing points together. It determines the minimum distance apart that points are allowed to be in the lower dimensional representation. Low values denote clumpier groups, while larger values prevent packing points and focus on saving the overall topological structure.
In the figure 5.8, we see that when min_dist=0.1 (in the left),UMAP manages to find small connected components in the data and preserve large distances between the different types of charts. Increasing min_dist pushes the structures apart into softer wide-spread general features(middle in the figure), providing a better overarching view of the data at the loss of the more detailed topological structure. Per contra, having an extremely high value as 0.99 (right in the figure) focuses only on isolating pie charts.



((a)) 0.1 **((b))** 0.5 **((c))** 0.99

Figure 5.8: increasing the min_dist parameter values of UMAP spreads the data samples more on the projection space with preserve of global structure
• bar • line • pie • scatter

metric Metric resembles the rational scale, which measures the distance between points in the projection space. Usually, on 2D space, euclidean is an excellent method to measure the distance between points, but we can also calculate the cosine similarity between the CNN output vectors. Figure 5.9 demonstrates this effect on two examples that concludes no influence of separation between types.



((a)) euclidean **((b))** cosine

Figure 5.9: different measures of UMAP
• bar • line • pie • scatter

UMAP surpasses PCA performance in clustering the various types of charts for two main reasons:

- PCA handle linear-correlated data samples. Dealing with non-linear cases shows poor performance.
- we can configure UMAP parameters depending on the use case and the data samples we have.

In conclusion, DR techniques show more advanced results than using k-means. Nevertheless, these techniques still analyze only the position and value of pixels that do not represent other information related to the chart attributes. Also, we find some extreme points and various distributions that do not denote any scalable measure. Consequently, we deduce that charts need to be processed before applying dimension reduction.

5.2 Usage of Transfer Learning

Transfer learning has been widely used in recent years because it conserves time for training and data pre-processing, especially when the model trained on huge dataset such as ImageNet. We use pre-trained models for detecting complex objects in natural images to extract features from simple geometric ones such as charts.
The pipeline consists of primarily using CNN to extract features and DR to project on 2D space to visualize the results, as introduced in figure 5.10. In this section, we introduce first various CNN architectures and compare their results using the same DR technique.

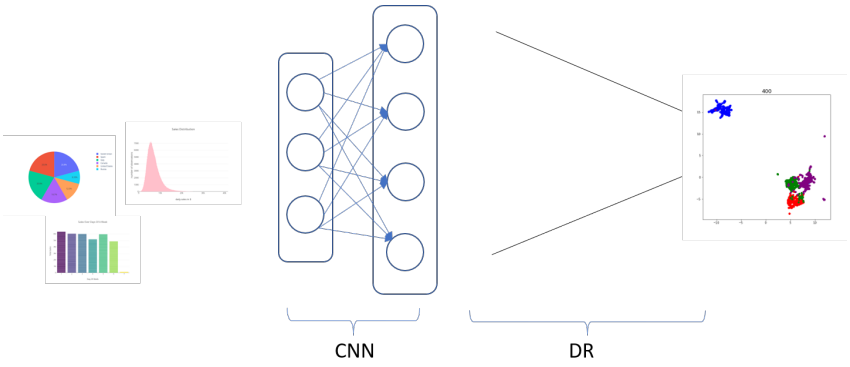


Figure 5.10: The method uses extracted feature vector from CNN and apply DR to visualize the subspace of the given samples

We use mainly four types of CNN architectures with different variants. Ordered by release date, we list them as follows:

- **VGG**[SZ15] is the first improved version after AlexNet[KSH12]. This architecture with two variants (16 & 19) layers is suitable to extract features from simple images
- **Inception**[SIVA17] uses kernels of different sizes for an effective recognition of such a variable-sized feature. For this reason, we test the effect of using different kernels size on charts processing that extracts large shapes such as pies and bars as well as small features such as points clusters in scatterplot.
- **Xception**[Cho17a] is an extension of the Inception architecture. It differs by replacing the standard modules with more deep separable convolutions. We are expecting that Inception and Xception deliver similar results.
- **ResNet**[HZRS16] is the latest architecture introduced in deep learning field. Using deep network with tens or hundreds of layers decreases the accuracy of the network. ResNet solves the vanishing gradient problem. We apply ResNet on charts to compare its performances since it outperforms the other models on ImageNet dataset.

The following table 5.1 shows a list of all CNNs we used, comparing their output shape and the depth of each network.

Network	number of layers	output shape (n-dimensional vector)
VGG16	16	512×1
VGG19	19	512×1
ResNet50	50	2048×1
ResNet101	101	2048×1
ResNet152	152	2048×1
Xception	71	2048×1
InceptionV3	48	2048×1

Table 5.1: Models used in our project

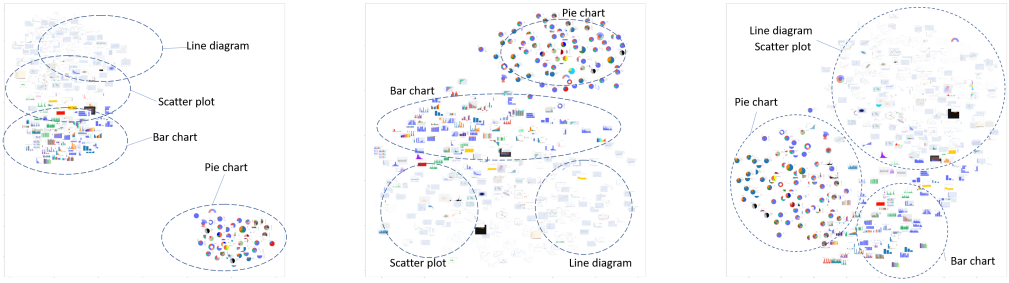
We discuss each network results when applying PCA and UMAP on its outputs with max-pooling:

- **VGG** On the one hand, applying PCA on VGG output (both variants) define three main clusters, as shown in figure A.9. One cluster combines the line diagrams and scatter plots. The deep processing of chart images dissolve the difference between them, especially that connecting points in scatter plots forms a line diagram in some cases. On the other hand, applying UMAP in figures A.3 and A.4 improves

the separation between types. It splits the four types of charts, although the close distance between bar charts, line diagrams, and scatter plots. Pie charts consists of circles that isolated from other types. The main difference between VGG16 and VGG19 is the flatten direction of the clusters.

- **Inception & Xception** works similarly as VGG network, as seen in figures A.5,A.6, and A.8. However, both networks use different kernel sizes to detect different levels of details. Hence, it defines wider distances between the charts which represents the similarity between charts. For example, we notice the right lower part of the pie cluster in figure A.6 charts that use dark blue instead of light blue together.
- **ResNet** variants gives the comparative poor results. Regardless of the network's depth, ResNet variants spread the charts over all the space forming three basic clusters. The key limitation of using ResNet here is the intersection between pie charts' cluster and Line diagrams, as shown in figure A.7. In this context, the projection on the 2D space does not demonstrate any similarity measure because a scatter plot can not be much similar to a pie chart than another scatter plot that locates at the other side of the embedding circle. We assume that using such a deep layer with more than 50 layers could vanish the features extracted from such simple images. Hence, UMAP spread them in a wide cloud-form cluster, while PCA put them in a small one spot where all charts are overlapped, as seen in figure A.10. However, bar and pie charts roll away from the spot, composing two arbitrary of their types.

To sum up, we compare the three main type of networks using UMAP in figure 5.11. We notice that although ResNet architecture is the best model when applied on normal images, here it shows an arbitrary result when is project on a 2D space. At the same time, VGG shows the best performance which denotes that using a super CNN network such as ResNet does not mean that it fits good to simple cases.



((a)) VGG19 **((b))** Xception **((c))** ResNet50

Figure 5.11: VGG network is the best option to separate the different chart types
Model config:DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

Using average instead of max pooling weakens the clustering of the used model. For example, using Inception CNN network with average pooling in figure A.11 can only define three types of charts similar to ResNet. The intersections between the clusters raise many questions about the distance defined between the charts in the subspace.

However, when applying PCA on a CNN network with average pooling require less dimensions than using UMAP to preserve higher explained variance of the dataset. Table 5.5 and figure 5.12 explain the required number of dimensions to get a high percentage of variance when applying PCA on the various combinations of used CNN architectures. This highlight the weakness of the model using average pooling because PCA preserves more variance of its given representations.

Network	dimensions required to preserve 95% variance	percentage of cumulative preserved variance on 2D space
VGG19 + max-pooling	101	46.23%
VGG19 + avg-pooling	54	49.02%
ResNet50 + max-pooling	62	44.47%
ResNet50 + avg-pooling	24	63.85%
Xception + max-pooling	148	28.07%
Xception + avg-pooling	135	27.75%
apply DR directly	159	42.64%

Table 5.2: number of required dimensions to preserve 95% of the explained variance for each network

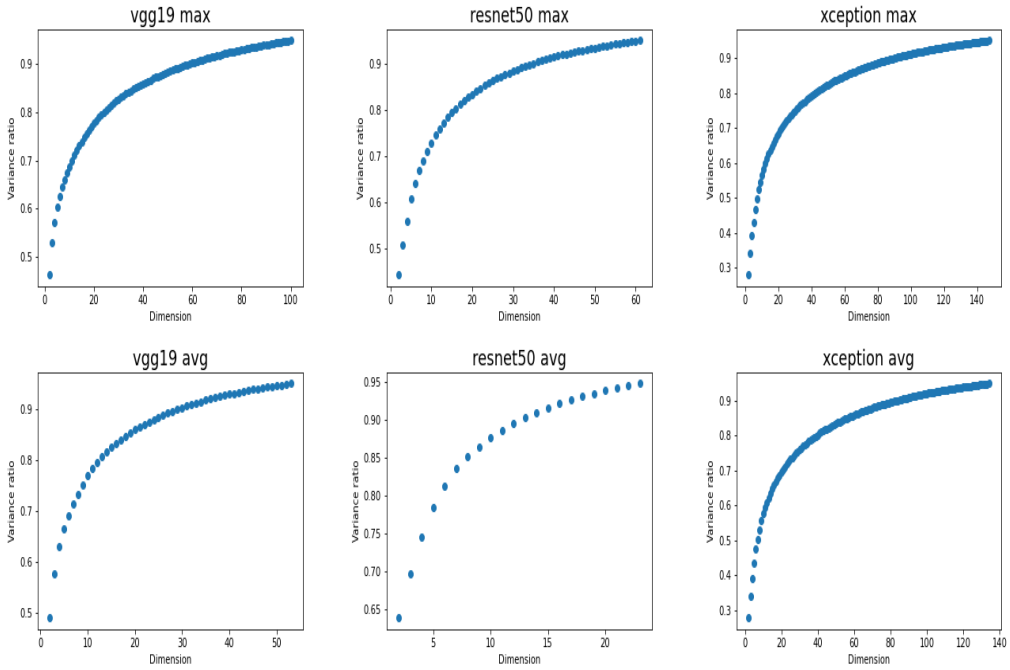


Figure 5.12: cumulative percentage of the explained variance from 2D projection to 95% case

To conclude, feature extraction using various CNN architectures affects the DR projection on the 2D space. Increasing the number of layers does not improve the separating results as in ResNet case. However, max-pooling has shown a better distribution of data samples contingently as distinct clusters regarding the chart type.

In summary, feature extraction using CNN achieves better results compared to applying unsupervised methods directly on images. The reason of that is the deep processing on charts using network weights trained on several tasks such as detection of harder shapes as people or cars. However, this method has still some disadvantages:

- **CNN output**: The output vector of the used CNNs might include overwhelming information because it is designed to handle more complex images. An instance of this problem is the critical loss of explained variance when applying PCA.
- **CNN model**: CNN layers used to be stacked with a top layer that accomplishes the final task. The accuracy of performing this task controls the optimization process of the model. In our project, feature extraction and evaluation are two separate stages. We assumed that trained models on a huge set of natural images would be suitable to handle simple charts. Nevertheless, this remains an open issue that needs to be critically discussed and evaluated for this approach.

5.3 Autoencoder

An autoencoder applies backpropagation, setting the target values to be equal to the inputs. In fact, autoencoder often ends up learning a low-dimensional representation very similar to PCAs. Through this compressed representation, we can discover the data's interesting structures by imposing other constraints on the network.

In this section, we propose an autoencoder approach to apply it on chart images. We discuss the main reasons that motivate to use autoencoder instead of using famous pre-trained CNN architectures. The critical advantage in autoencoder that we can propose a new architecture. We explain it in details and show the accuracy of this autoencoder trying to reconstruct images from three different datasets. At the end of this section, we apply the encoding output to DR techniques and visualize the results.

We propose the autoencoder approach for the following advantages over using popular CNN architectures:

- **Model's output**: We design our own autoencoder model so that we can have output shape different from VGG or ResNet output form.
- **Encoding's evaluation**: Autoencoders aim to reconstruct the original input from the final encoding's result. In this case, we ensure that the reduced output describes valid chart features and makes a reliable representation of the chart on a lower-dimensional space.

We designed an autoencoder consisting of three convolutional layers with max-pooling as the encoding part and three convolutional layers in the decoding part. from $25 \times 25 \times 16$ it reconstructs the input image of (200×200) image size.The table 5.3 describe the full details of our autoencoder.
We train the autoencoder on 17.376 chart images, splitting the dataset into 12.898 train samples and 4.478 validation samples(approx. 75% by 25%). Adam optimizing algorithm outperforms classic Stochastic gradient descent (SGD) by computing individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Hence, we use it to compile the model as an optimizer and the binary cross-entropy as a loss function because we normalize the input data to the range of 0 to 1 values. After 15 epochs, the model reaches a 15% loss and cannot be improved anymore.

Layer	output shape	number of parameters
Input Layer	(200,200,3)	0
Conv2D	(200,200,64)	1792
Batch Normalization	(200,200,64)	256
Activation	(200,200,64)	0
MaxPooling2D	(100,100,64)	0
Conv2D	(100,100,32)	18464
Batch Normalization	(100,100,32)	128
Activation	(100,100,32)	0
MaxPooling2D	(50,50,64)	0
Conv2D	(50,50,16)	4624
Batch Normalization	(50,50,16)	64
Activation	(50,50,16)	0
MaxPooling2D	(25,25,16)	0
Conv2D	(25,25,16)	2320
Batch Normalization	(25,25,16)	64
Activation	(25,25,16)	0
UpSampling2D	(50,50,16)	0
Conv2D	(50,50,32)	4640
Batch Normalization	(50,50,32)	128
Activation	(50,50,32)	0
UpSampling2D	(100,100,32)	0
Conv2D	(100,100,32)	18496
Batch Normalization	(100,100,32)	256
Activation	(100,100,64)	0
UpSampling2D	(200,200,64)	0
Conv2D	(200,200,3)	1731
Batch Normalization	(200,200,3))	12
Activation	(200,200,3)	0

Table 5.3: Autoencoder architecture
Total parameters: 52,975
Trainable parameters: 52,521

To evaluate the autoencoder, we test the restructured output image quality using three different datasets:

- Charts collected from plotly

- visualizations from IEEE conferences publications
- household objects: this dataset contains images of small household objects positioned on a white background[1]

Table 5.4 shows the accuracy percentage of the autoencoder when it is evaluated on different datasets.

Dataset	accuracy percentage
Charts collected from plotly	86,86%
Visualization from IEEE conferences papers	74,71%
household objects	38,62%

Table 5.4: Autoencoder accuracy percentage on different dataset

This loss value ,that ranges between 15% to 25% and reaches to 60% on normal images, causes a strong blur effect similar to a gaussian blur with intermediate values and increases on natural images. The figure 5.13 shows three images examples of autoencoder results from the three datasets.

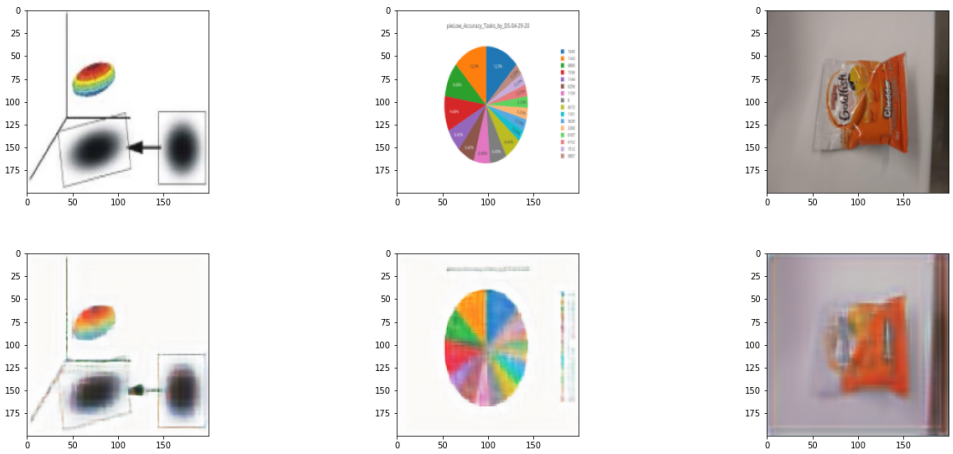
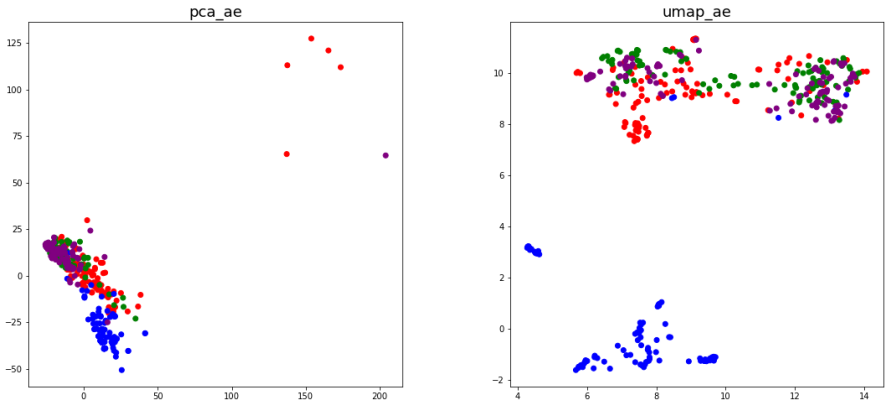


Figure 5.13: Three autoencoder output examples from three different datasets

Applying PCA on the encoded vector does not surpass the transfer learning and DR methods, as shown in figures A.17 & A.18. All charts with a white background are

[1] https://ieee-dataport.org/open-access/annotated-image-dataset-household-objects-robofeihome-team

clustered together while charts with black or blue backgrounds are separated. However, this method can still split the pie charts from other types. We were surprised to find that the number of required dimensions to get 95% explained variance is 159, **the same as applying DR on images directly**.
Although UMAP has shown previously better clustering results, it differs slightly from PCA case by spreading the points more over the space and isolating the pie charts, as shown in figure 5.14.



((a)) PCA **((b))** UMAP

Figure 5.14: UMAP spreads types samples while PCA do an All-in-one clustering
UMAP parameters: n_neighbors=15, min_dist=0.1, metric=euclidean

The findings of this approach indicate that applying DR on encoding representation of autoencoders does not outperform from applying it on raw image directly. This is a piece of empirical evidence to prove that autoencoders and DR techniques work similarly.
However, this small contribution does not vouch that autoencoder is not an acceptable approach for obtaining a reduced representation of chart images. Specifically, chart images do not contain many complicated features, as in other images.

Summary

We have proposed three main methods that aim to obtain a useful representation of charts to compute and compare similarity between them. To overcome the limitations of applying DR on charts directly, we supported it by pre-processing the images using a pre-trained CNN model. Anyway, we implemented an autoencoder that works similarly as DR to compare the outputs between the two methods' results. From all experiments,

we show that using CNN + DR method provides the most reasonable results regarding our use case. In the next section we aim to test this method with different data samples or different conditions.

5.4 Data Sampling

Previously we studied the effect of internal various configuration of the model on the output results such as the CNN architecture or the DR technique. In this section, we analyze the effect of input data on the results and how it can improve the final projection. We divide this section to study three factors:

- Sampling analyzes the effect of stochastic distribution of the types in the dataset
- Quantity tests the performance of the model when increasing the amount of given charts
- Data type quality checks the results when applying different data types

Sampling

There are primarily two types of sampling:

- Random sampling means selecting the data with a random function without any restrictions
- Systematic sampling denotes selecting the data based on a determined schema. For example, we always select an equal amount of charts of each type.

Systematic sampling plays an essential role that affects the clustering process. We can clearly remark this effect in figure 5.15 that applies the K-means but on a dataset composed of 100 samples for each of four main types (bar, pie, line, scatter). The algorithm splits nearly between the chart types but still preserves the black-background charts together. We can not consider those results as a reliable and accurate clustering yet. For that reason, we hold applying K-means approach not accepted for our case.
To simplify this effect, we applied transfer learning using VGG19 and UMAP on three different datasets, compared in 5.16. The first one consists of completely randomly selected charts in figure A.12. The other one, shown in figure A.13, is also randomly selected from 1600 charts but equally distributed over the four main types, so the probability of selecting a specified type is $1/4$. Thirdly, we compare both results with the full systematic sampling used previously in figure A.4.

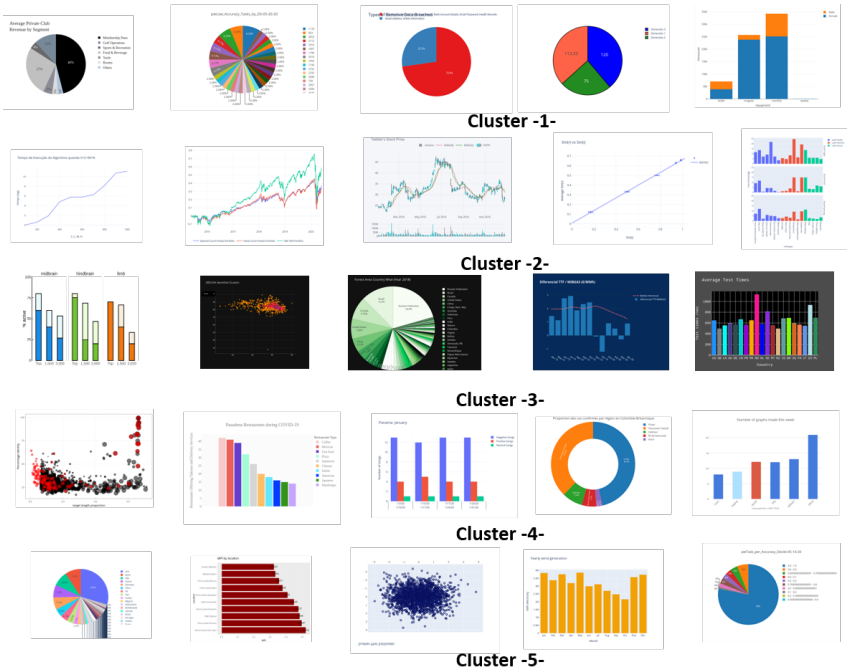
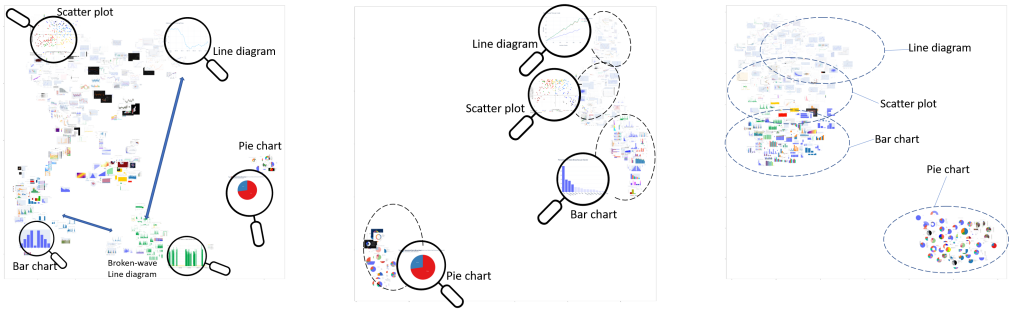


Figure 5.15: Applying k-means on a systematic selected group of charts



((a)) Complete random

((b)) Random selection from an equally distributed dataset

((c)) Systematic sampling

Figure 5.16: UMAP can split the charts' types regardless of data distribution
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

It is crucial to note that the coordinates of the points are not fixed when the distribution changes. It means that these results need to be interpreted with caution in the potential applications of this project. However, the type of sampling does not affect the performance of charts' types clustering seen in the three cases A.4, A.13, and A.12. The random sampling, shown in figure A.12, reveals an interesting issue that shows the

similarity between dense wave-shape line diagrams and bar charts. The density of the lines forms a similar shape to bars. Thus, bar charts seem much more similar to those type of diagrams than line diagrams.

Data quantity

Increasing the input data amount makes it harder for DR technique to analyze the given dataset because it should analyze more different values on the same dimension and much more samples.
First, we applied the PCA on VGG19, resnet50, and xception outputs on 800 charts and 1600. The table 5.5 compares the number of required dimensions to preserve 95% of the explained variance for each data amount. In figure 5.17, we see that this increment is linearly related to the quantity of input data.

Network	400 Charts	800 Charts	1600 Charts
VGG19	101	138	171
ResNet50	62	86	118
Xception	148	222	323

Table 5.5: number of required dimensions to preserve 95% of the explained variance for dataset applying various CNNs with max-pooling

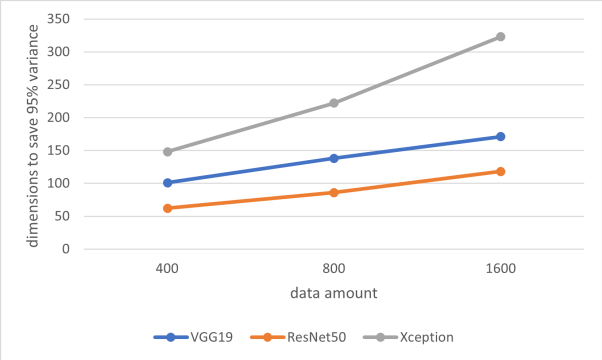


Figure 5.17: Increment of required dimensions to preserve 95% of the explained variance when the data amount increases

We apply various amounts ranging from 100 to 1600 charts on VGG19+umap output in figure 5.18. Changing the amount does not affect the separation performance, but it is crucial to notice the four types' non-fixed coordinates. Starting with 100 charts (Figure

5.18 upper left) defines three spots that have an equal distance between each other: Pie charts, bar charts and a combined spot of line diagrams and scatter plots. Increasing the amount of charts reduces the distance between bar charts and line diagrams with scatter plots, while pie charts remain on a constant distance to the other spots.

However, The values' change is more restricted than the random sampling case as it moves only in a limited area. For example, pie charts x-axis values range between -4 and -15. In this case, the types are preserving the positional relationship between each other .i.e pie charts' values are always lower than line and scatter plots. At the same time, it obtains higher values on the x-axis on random sampling shown in figure A.12 compared to line plots, and then they switch when the sampling is customized.

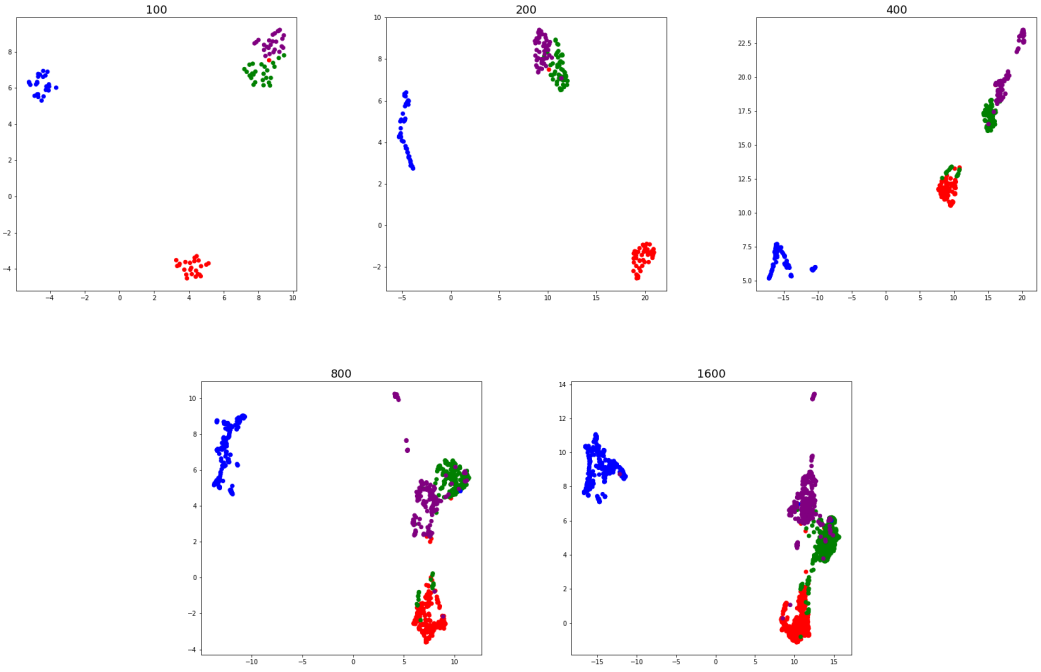


Figure 5.18: Increasing the data amount does not affect the cluster embedding
Model used: CNN:VGG19 , DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean
• bar • line • pie • scatter

Data type quality

In this section we apply three kinds of experiments. The first one demonstrates the model's performance when applying more than four types of samples. Detecting the chart

types is not the only purpose of this project. Thus, we use the approach to reveal more visual features between the same type of the chart. The last experiment test the model by clustering various types of visualizations collected from IEEE publications conferences.

So far, we have applied our approaches primarily on four types that it seems to be an easy task to split between them. We added to further chart types to increase this task's complexity (box plots and maps). Figure 5.19(original found in appendix A.14) demonstrates the performance of applying VGG19 with UMAP on 6 types of charts. The method outlines two basic spots, isolating pie charts from the other types that form four type of clusters closed together. A limitation here is the deficiency of separating scatter plots. A one possible reason is that the distribution of the points on the scatter can compose various stroke shapes that could be bars, lines or maps.
However, all configurations we used previously can isolate pie charts from other types.

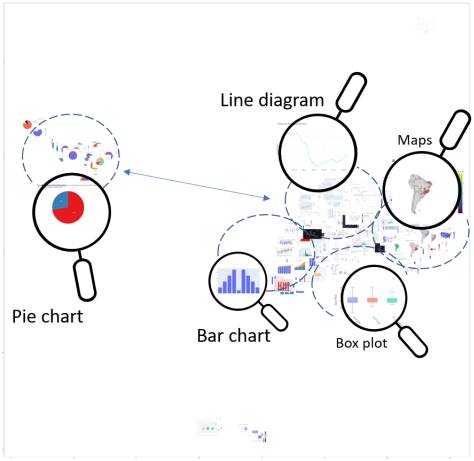


Figure 5.19: Pie charts spot subsides from other type clusters
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

The reason for it that it can easily detect the circles that only exist in pie charts, while rectangles are the main components in both of box plots and bar charts. It makes it harder to observe the interior different visual features between charts of the same type. For this reason, we apply VGG19+UMAP on bar charts only in figure 5.20(original found in appendix A.16). The used approach defines a horizontal determinant regarding the number of bars in the chart starting with less than 5 bars pro chart at the top and increases the number gradually. We notice also a little deviation because the width and density of the described bars. Two main groups are also separated from other charts

which are the horizontal bar charts and charts describing more than one category of each sample.

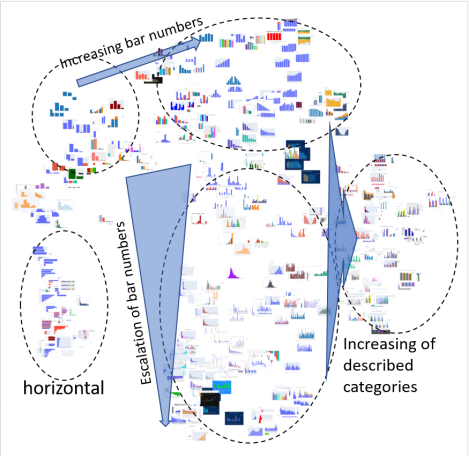


Figure 5.20: numbers of bars increases with lower values on the 2D space
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

The approach provides good results when applying it on simple chart images. However, we test this model on 1166 visualizations collected from IEEE publications. As shown in figure 5.21(original found in appendix A.15), it can detects many categories of the given data. The drawback in this case is the interpretation of the distances between the clusters. The closed location of the three clusters of pie charts, cycle diagrams, and 3D spheres can be explained through using the circle as the main visual elements in all this samples. At the same time, the closed distance between bar charts and software interfaces raises an open question since there is no matching of visual elements between both types.

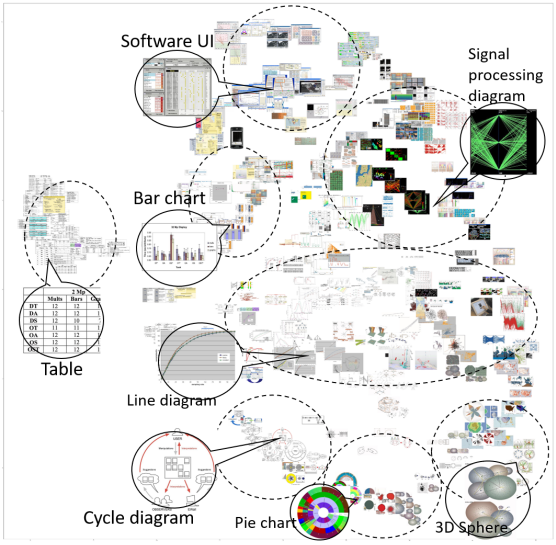


Figure 5.21: Model identifies various visualizations and clusters similar ones together
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

Summary

This chapter has led us to conclude that the selection of the method's configuration depends on the use-case itself. For example, if we are looking to separate the different chart types, it is better to use the UMAP with lower neighbors' values. When looking to study and analyze a specific type's visual encodings, we should only feed the model with this specific dataset with the attention of data distribution.

Broadly speaking about model components, using ResNet network that works better than VGG19 on natural images does not infer that it would perform better in our use-case. In general, UMAP has shown better performance than using PCA over all experiments we had, especially that we can configure its parameters.

Changing the data quantity or type distribution affects the DR's workflow because it enforces the technique to handle the less given amount of a specific type' samples as extreme values and roll it away from other charts, as shown in pie charts cluster separating case.

The weights obtained from the pre-training on ImageNet let us extend the model use case and apply it on advanced visualizations that includes complex shapes. While the

method proves itself useful in many scenarios, it's not without its shortcomings. The visual semantics of the representations seems to be disregarded, for example as it can be seen with tables and bar charts in figure A.15; while both samples representations can represent the same data, they do not share almost any common visual aspects, yet they are grouped closely together.

6 Evaluation

The last step of this work is to evaluate and test the model if it is getting reliable and sensible outputs that can be used in probable applications. The critical point is analyzing the final representation from the CNN output. The figures 6.1 and 6.2 clarify the difference between the related work models and the proposed model in this project. In the first section, We perform quantitative experiments on the outputs by comparing the classifier accuracy on the model outputs and the related works' trained model. We introduce potential use cases in the second section, which depends on the result of this work and demonstrates the trained model's effectiveness and efficiency.

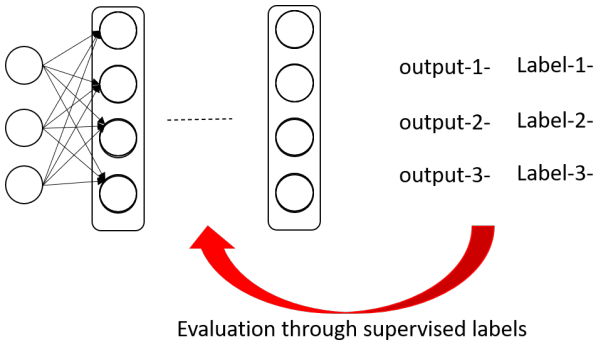


Figure 6.1: Model pipeline used in the related work

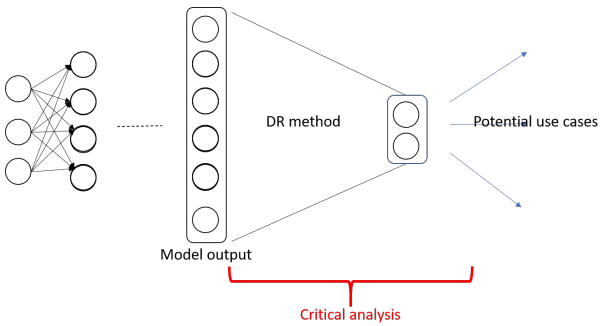


Figure 6.2: Model pipeline of this project showing the critical part

6.1 Experiments

This section aim to compare the performance of the classifier using the model outputs as the data sample inputs for it. We evaluate the model first on the charts collected from plotly through feeding the model outputs to an SVM classifier and then compare their accuracy with related works model used for the same purpose. We label the subset input data manually to a set of categories regarding the classification task such as chart type identification. This experiment demonstrates the generic representation's quality of charts for different purposes, comparing it with the state-of-art ML contribution in the automatic visualization field.

6.1.1 Experiments on charts

Charts classification based on type

We trained the SVM classifier on the 1600 2D output points to detect six main types: (bar, pie, line, scatter). The 1600 charts dataset is divided into 1200 training samples and 400 for testing. The following table 6.1 explain the accuracy score when applying VGG19+UMAP that reaches the highest accuracy between all other configurations. The perfect classification done on those types is degraded when we add two more types as explained in table 6.2. However, our classification outperforms better accuracy than Chartsense[JKS+17] and Revision[SKC+11] and nearly reaches the state-of-the-art done to this task in REV[PH17]. In table 6.3, we see that our method performs better than REV[PH17] classifier on pie charts, line diagrams, and maps, while it shows a poor achievement on detecting bar charts.

Chart type	precision	recall	f1-score	support
Bar charts	99%	99%	99 %	100
Line diagrams	98%	96%	97%	100
Pie chart	99%	100%	100%	100
Scatter plot	96%	97%	97%	100
accuracy			98%	400
macro average plot	98%	98%	98%	400
weighted average	98%	98%	98%	400

Table 6.1: SVM reaches 98% when classifying the projected samples on the 2D space to its type
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

Chart type	precision	recall	f1-score	support
Bar charts	85%	92%	88%	25
Line diagrams	74%	100%	85%	25
Pie chart	100%	100%	100%	25
Scatter plot	89%	100%	94%	25
Box plot	93%	52%	67%	25
Maps	100%	88%	94%	25
accuracy			89%	150
macro average plot	90%	89%	88%	150
weighted average	90%	89%	88%	150

Table 6.2: Adding two further chart types downgrades the accuracy of SVM on classifying the projected samples on the 2D space to its type
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

Chart type	Ours(VGG19+UMAP)	REV[PH17]	Revision[SKC+11]	Chartsense [JKS+17]
Bar charts	85%	97%	78 %	93%
Line diagrams	100%	94%	73%	78%
Pie chart	100%	100%	100%	99%
Scatter plot	92%	97%	97%	99%
Maps	92%	96%	84%	88%
Average accuracy	93.8%	94.4%	86.4%	93.6%
Overall average accuracy	92%	94%	80%	90%

Table 6.3: Comparing this approach with related work on classification of chart type
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

Line diagrams classification based on line trend

Line-trend classifier in [NL] splits the line diagrams into three main categories:

- Increasing: A significant increase in the plotted value.
- Decreasing: A significant reduction in the plotted value.

- Neutral: Either there is no significant difference in the plotted value, or the graph is too noisy to determine a significant trend such as flat lines, one period of a sinusoidal function, or random noise.

We compare in this experience the accuracy of the this classifier and a trained SVM on our model outputs to classify them into the previous mentioned three classes. The model is restricted to achieve only 62% accuracy, less than the introduced classifier in [NL] as clarified in table 6.4.

Ours(VGG19+UMAP)	Trend classifier[NL]
62%	84.061%

Table 6.4: Our approach can not achieve better accuracy in trend detection than [NL] paper
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

In summary, transforming the problem of a classifier from assigning the label based on a set of extracted features to the coordinates of every class achieves good results. However, detecting fine-detail such as line trend seems to still need improvements on the used models to achieve better accuracy.

6.2 Potential use cases

Search Engine and database Building a search engine only for data visualization has been an interest for researchers and designers. E.Hoque and M. Argawala propose a query-based search engine for visualization[HA20]. However, This search engine is specified only to D3 visualization. It obtains the visual features through the deconstruction of collected 7860 D3 visualizations and indexes their style and structure information and metadata about the webpage, including the chart.
Finding a one-to-one correspondence between the query scheme and the representation output would make it possible to inquire about any chart found on the web. The system projects the given query on the representations' hyperplane and then returns the k-nearest charts.

Redesign visualization Most of prior works in automatic visualization concern redesigning the current chart for a better understanding and improved data visualization. As mentioned early in the related work section 3, projects such as Revision[SKC+11] or Voyager2 [WQM+17] try to extract visual features from the chart

image or VizML[HBL+18] restricts to charts associated with its visual encoding as JSON file to train the recommendation model later. The problem of data extraction has been widely investigated, as seen in Reverse-Engineering Visualizations[PH17], Deep-eye[LQTL18], and Chartsense[JKS+17] and still have some limitations. An interesting application would be using the model output of this project with the raw data for the redesign. In this case, it will overcome the problem of data extraction.

Studying the effect of encodings on similarity ScatterNet [MTW+20] achieved to define a subspace which allows measuring the similarity between scatterplots by euclidean distance. A similar approach we can do it since the outputs of CNN are reduced to 2D dimension. Retaining all charts represented as 2D points would make it possible to measure the similarity between two points. In this case, the designers and researchers can study the effect of changing visual features on the similarity between charts. The visualization shows when two neighbor charts might close even more or roll away from each other.
However, we need to evaluate the clustering and positioning of the charts' points. Performing a user study in which researchers participate to rate the points' clustering. For example, in figure 6.3 we generated a bar chart with two different variants that is changing one of the related visual attributes. The first one order the values in ascending order, while the other one uses horizontal bars. The projection in figure 6.4 shows that chart generated from sorting the bars on values is visually similar than using horizontal bars. Also it is located in the wider similarity area from the specified chart. We ask the participants to rate the cluster embedding in terms of similarity regarding the given visual features and address some switches such removing the bar chart located in the blue closed area. The study demonstrates how well the euclidean distance reflects the human perception of similarity measuring between the charts. The switches a user might suggest gives a rational scale to evaluate projection space.

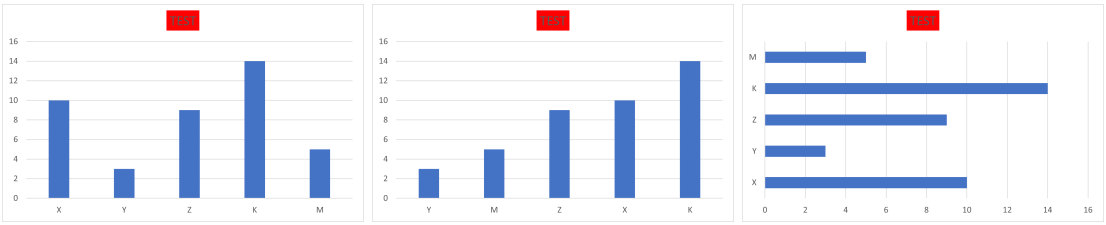


Figure 6.3: Bar chart with two different variants changing a specific visual attribute

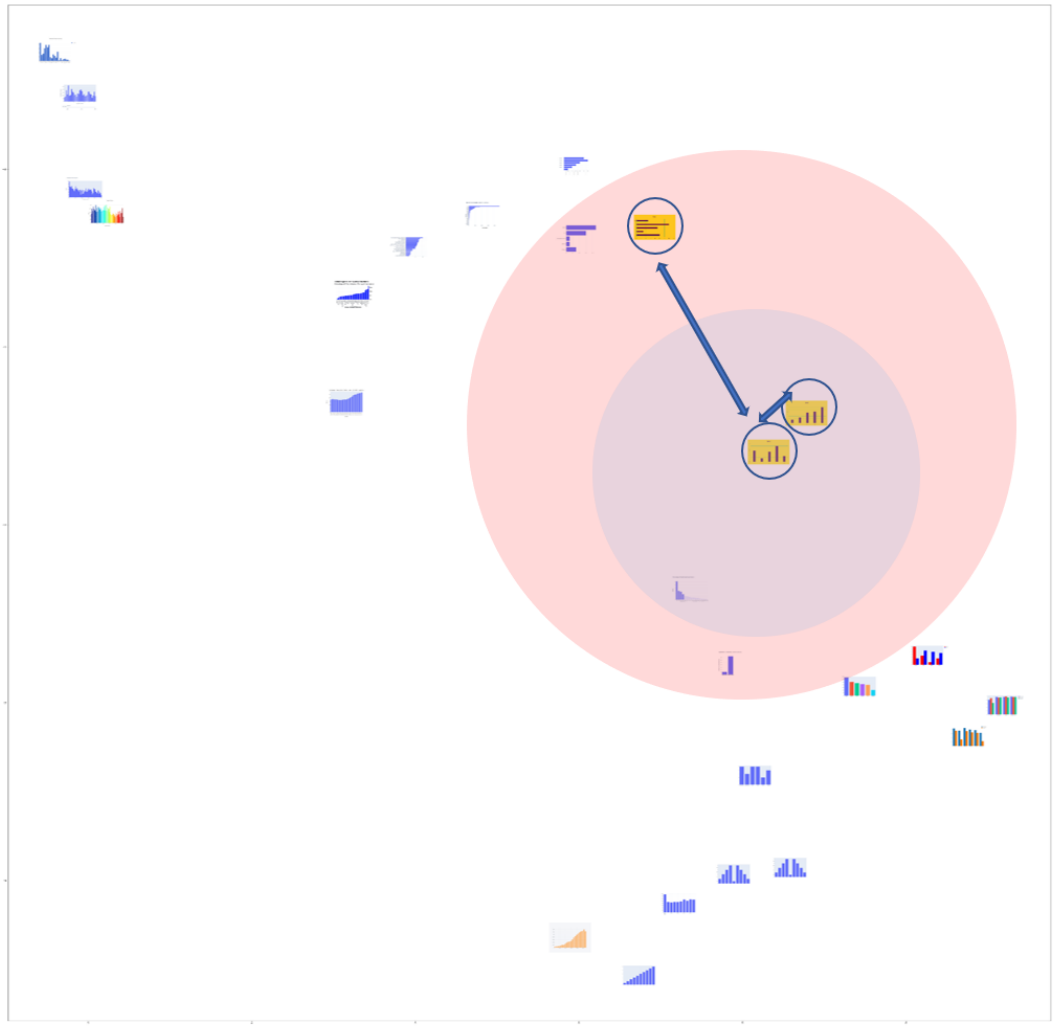


Figure 6.4: Sort the bars is visually similar than switching the axises
UMAP configuration: neighbors=3, min_distance=0.1, metric=euclidean

7 Conclusion

In the last section, we summarize this project's findings with its purpose to gett an overview of the whole work. We discuss also the drawbacks of this project and potential further developments.

Summary

We explored different approaches in unsupervised machine learning to obtain a reduced useful representation of charts for comparison and analysis purposes. Comparing clustering, dimension reduction, transfer learning, and autoencoder, we found out that applying DR on the processed output of CNN shows the best performance.
The results of the done experiments in both section 5.4 and 6.1 support the idea of using the final projection space to measure similarity between given charts. It provides the basis of implementing a system that can compute the correlation between charts regarding some visual features or rank them based on quality metrics.

7.1 Limitations and future work

Finally, a number of potential need to be considered, mentioned in the following list:

- We applied all methods on only four types of charts. This seems to be an easy task, as done by isolating pie charts. The usage of six types instead of four support the idea that there are still further types of charts that could make it hard to split between all the types.
- The autoencoder approach introduced a three layer deep for encoding as well as for decoding. It remains an open question that if we used more deep autoencoder denotes getting better clustering embedding from the encoded representations.

- The detection of chart type was the main measure we followed to evaluate the different results of the methods. However, if we are looking to build a complete analysis system based on quality metrics, then we should define more evaluation measures that concern fine-detail attributes.

7.2 Future work

This project set the first step of obtaining a reduced representation of chart images, that can be deployed in different Automated Visualization fields instead of training a task-related model. We propose that further research should be undertaken in the following areas:

- Tableau and D3 visualizations community includes interesting charts with more complicated design encoding. An important issue is to check the success of our method when applying it on other style of charts.
- Future studies should target implementing an autoencoder that reaches high accuracy to observe the relationship between the success of the image's reconstruction and extracting useful features from its encoding version.
- An important evaluation of our method is to investigate the performance of automated visualizations projects in case of integration with it. In this case, we prove that the model's output represents valid visual features of the chart.

Bibliography

[20a] *D3 visualizations*. https://d3js.org/. 2020 (cit. on p. 31).

[20b] *Hierarchical clustering: structured vs unstructured ward*. http://scikit-learn.sourceforge.net/0.8/auto_examples/cluster/plot_ward_structured_vs_unstructured.html. 2020 (cit. on p. 20).

[20c] *How UMAP Works*. https://umap-learn.readthedocs.io/en/latest/how_umap_works.html. 2020 (cit. on p. 22).

[20d] *plotly platform*. https://plotly.com/. 2020 (cit. on p. 31).

[20e] *Tableau community*. https://www.tableau.com/why-tableau/what-is-tableau. 2020 (cit. on p. 31).

[BBK+18] M. Behrisch, M. Blumenschein, N. W. Kim, L. Shao, M. El-Assady, J. Fuchs, D. Seebacher, A. Diehl, U. Brandes, H. Pfister, T. Schreck, D. Weiskopf, D. A. Keim. "Quality Metrics for Information Visualization." In: *Computer Graphics Forum* 37.3 (2018), pp. 625–662. DOI: 10.1111/cgf.13446. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13446. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13446 (cit. on p. 30).

[BDM+18] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, M. Stonebraker. "Beagle: Automated Extraction and Interpretation of Visualizations from the Web." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: Association for Computing Machinery, 2018. ISBN: 9781450356206. DOI: 10.1145/3173574.3174168. URL: https://doi.org/10.1145/3173574.3174168 (cit. on p. 28).

[BHZ+20] J. Bernard, M. Hutter, M. Zeppelzauer, M. Sedlmair, T. Munzner. "SepEx: Visual Analysis of Class Separation Measures." In: *EuroVA@Eurographics/EuroVis*. 2020 (cit. on p. 29).

[CAM+18] P. Chagas, R. Akiyama, A. Meiguins, C. Santos, F. Saraiva, B. Meiguins, J. Morais. "Evaluation of Convolutional Neural Network Architectures for Chart Image Classification." In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8 (cit. on p. 30).

[Cho17a] F. Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions.” In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1800–1807. DOI: 10.1109/CVPR.2017.195 (cit. on p. 46).

[Cho17b] F. Chollet. *Deep Learning with Python*. 1st. USA: Manning Publications Co., 2017. ISBN: 1617294438 (cit. on pp. 17, 18, 22).

[Gro17] A. Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O’Reilly Media, Inc., 2017. ISBN: 1491962291 (cit. on pp. 17, 18, 20, 21, 23).

[HA18] J. Harper, M. Agrawala. “Converting Basic D3 Charts into Reusable Style Templates.” In: *IEEE Transactions on Visualization and Computer Graphics* 24.3 (Mar. 2018). DOI: 10.1109/TVCG.2017.2659744. URL: http://par.nsf.gov/biblio/10059675 (cit. on pp. 26, 28).

[HA20] E. Hoque, M. Agrawala. “Searching the Visual Style and Structure of D3 Visualizations.” In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 1236–1245 (cit. on pp. 28, 65).

[HBL+18] K. Hu, M. Bakker, S. Li, T. Kraska, C. Hidalgo. *VizML: A Machine Learning Approach to Visualization Recommendation*. Aug. 2018 (cit. on pp. 27, 66).

[HTP19] D. Haehn, J. Tompkin, H. Pfister. “Evaluating ‘Graphical Perception’ with CNNs.” In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2019), pp. 641–650 (cit. on pp. 29, 30).

[HZRS16] K. He, X. Zhang, S. Ren, J. Sun. “Deep Residual Learning for Image Recognition.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on p. 46).

[JKS+17] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, J. Seo. “ChartSense: Interactive Data Extraction from Chart Images.” In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 6706–6717. ISBN: 9781450346559. DOI: 10.1145/3025453.3025957. URL: https://doi.org/10.1145/3025453.3025957 (cit. on pp. 27, 63, 64, 66).

[Jol11] I. Jolliffe. “Principal Component Analysis.” In: *International Encyclopedia of Statistical Science*. Ed. by M. Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_455. URL: https://doi.org/10.1007/978-3-642-04898-2_455 (cit. on p. 21).

[JT20] L. Jing, Y. Tian. “Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (May 2020), pp. 1–1. DOI: 10.1109/TPAMI.2020.2992393 (cit. on p. 39).

[Kha20] M. A. W. F. A. B. S. A. A. I. Khan. *Advances in Deep Learning*. 1st. Springer Singapore, 2020. ISBN: 978-981-13-6794-6 (cit. on p. 17).

[KSH12] A. Krizhevsky, I. Sutskever, G. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386 (cit. on p. 46).

[KSZQ20] A. Khan, A. Sohail, U. Zahoora, A. S. Qureshi. “A survey of the recent architectures of deep convolutional neural networks.” In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516 (cit. on p. 23).

[LQTL18] Y. Luo, X. Qin, N. Tang, G. Li. “DeepEye: Towards Automatic Data Visualization.” In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 2018, pp. 101–112 (cit. on pp. 11, 28, 66).

[LWL+20] M. Lu, C. Wang, J. Lanir, N. Zhao, H. Pfister, D. Cohen-Or, H. Huang. “Exploring Visual Information Flows in Infographics.” In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20. Honolulu, HI, USA: Association for Computing Machinery, 2020, pp. 1–12. ISBN: 9781450367080. DOI: 10.1145/3313831.3376263. URL: https://doi.org/10.1145/3313831.3376263 (cit. on p. 29).

[MAP+15] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 36).

[MHSG18] L. McInnes, J. Healy, N. Saul, L. Grossberger. “UMAP: Uniform Manifold Approximation and Projection.” In: *The Journal of Open Source Software* 3.29 (2018), p. 861 (cit. on pp. 21, 22).

[MTW+20] Y. Ma, A. K. H. Tung, W. Wang, X. Gao, Z. Pan, W. Chen. "ScatterNet: A Deep Subjective Similarity Model for Visual Analysis of Scatterplots." In: *IEEE Transactions on Visualization and Computer Graphics* 26.3 (2020), pp. 1562–1576 (cit. on pp. 28, 66).

[NL] A. Newman, N. Landman. "Computerized Detection of Trends in Line Charts." In: () (cit. on pp. 28, 37, 64, 65).

[PH17] J. Poco, J. Heer. "Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images." In: *Computer Graphics Forum (Proc. EuroVis)* (2017). URL: http://idl.cs.washington.edu/papers/reverse-engineering-vis (cit. on pp. 27, 37, 63, 64, 66).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 36).

[SHL+16] N. Siegel, Z. Horvitz, R. Levin, S. Divvala, A. Farhadi. "FigureSeer: Parsing Result-Figures in Research Papers." In: *Computer Vision – ECCV 2016*. Ed. by B. Leibe, J. Matas, N. Sebe, M. Welling. Cham: Springer International Publishing, 2016, pp. 664–680. ISBN: 978-3-319-46478-7 (cit. on p. 28).

[SIVA17] C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 4278–4284 (cit. on p. 46).

[SKC+11] M. Savva, N. Kong, A. Chhajta, F.-F. Li, M. Agrawala, J. Heer. "ReVision: automated classification, analysis and redesign of chart images." In: *UIST '11*. 2011 (cit. on pp. 26, 37, 63–65).

[SS15] V. Sumithra, S. Surendran. "A review of various linear and non linear dimensionality reduction techniques." In: *Int. J. Comput. Sci. Inf. Technol* 6 (2015), pp. 2354–2360 (cit. on p. 20).

[SZ15] K. Simonyan, A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (2015) (cit. on p. 46).

[TS10] L. Torrey, J. Shavlik. "Transfer learning." In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264 (cit. on p. 24).

[WQM+17] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, J. Heer. “Voyager 2: Augmenting Visual Analysis with Partial View Specifications.” In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017) (cit. on pp. 27, 65).

All links were last followed on October 30, 2020.

A Projection results on 2D space of methods used in chapter-5

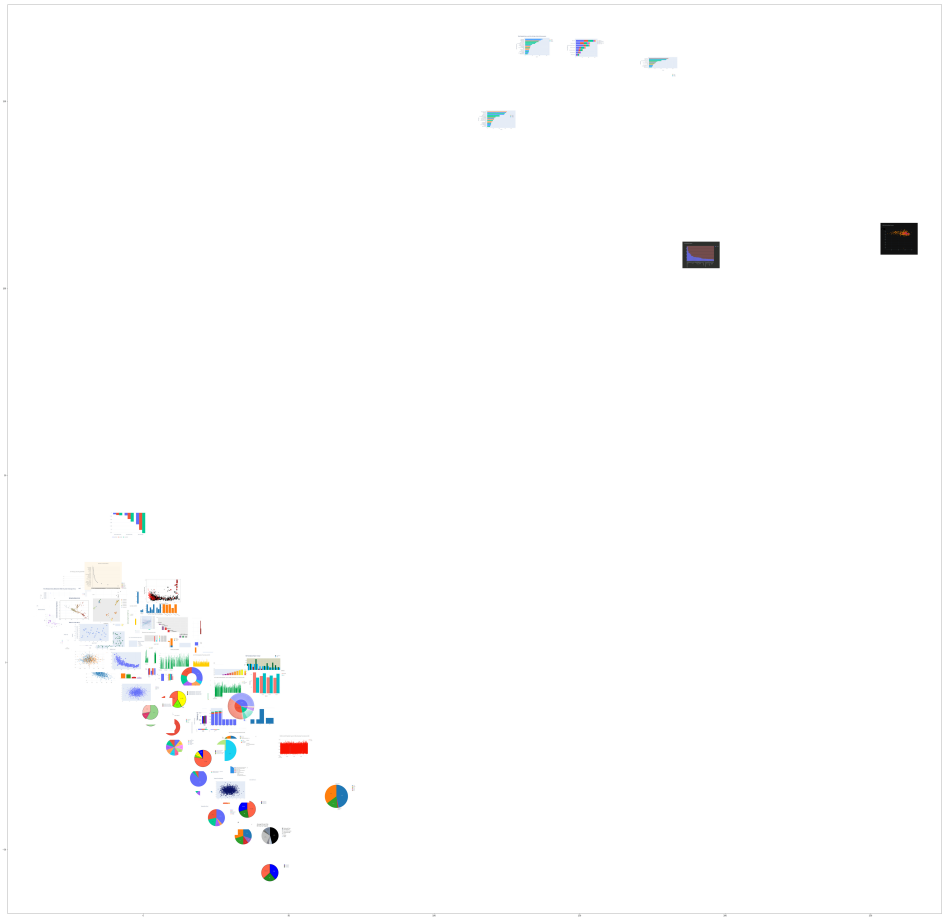


Figure A.1: applying PCA directly on images pools all charts in one spot together

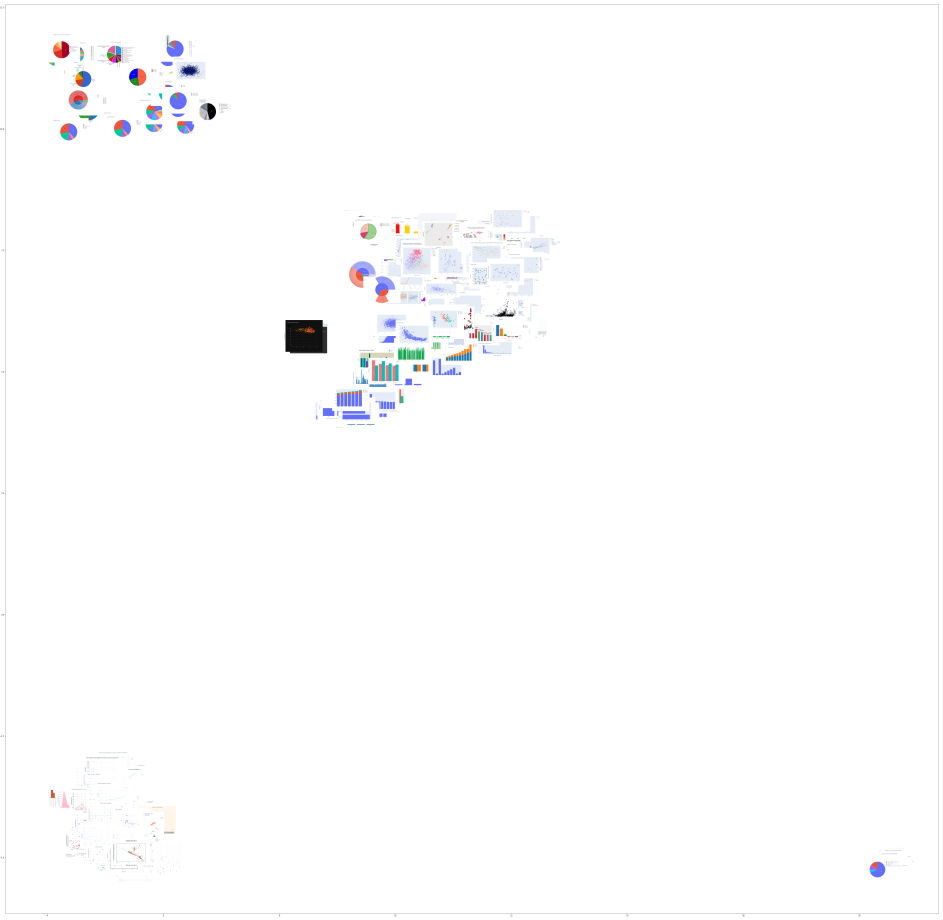


Figure A.2: applying UMAP on images can split most of pie charts in a seperated cluster

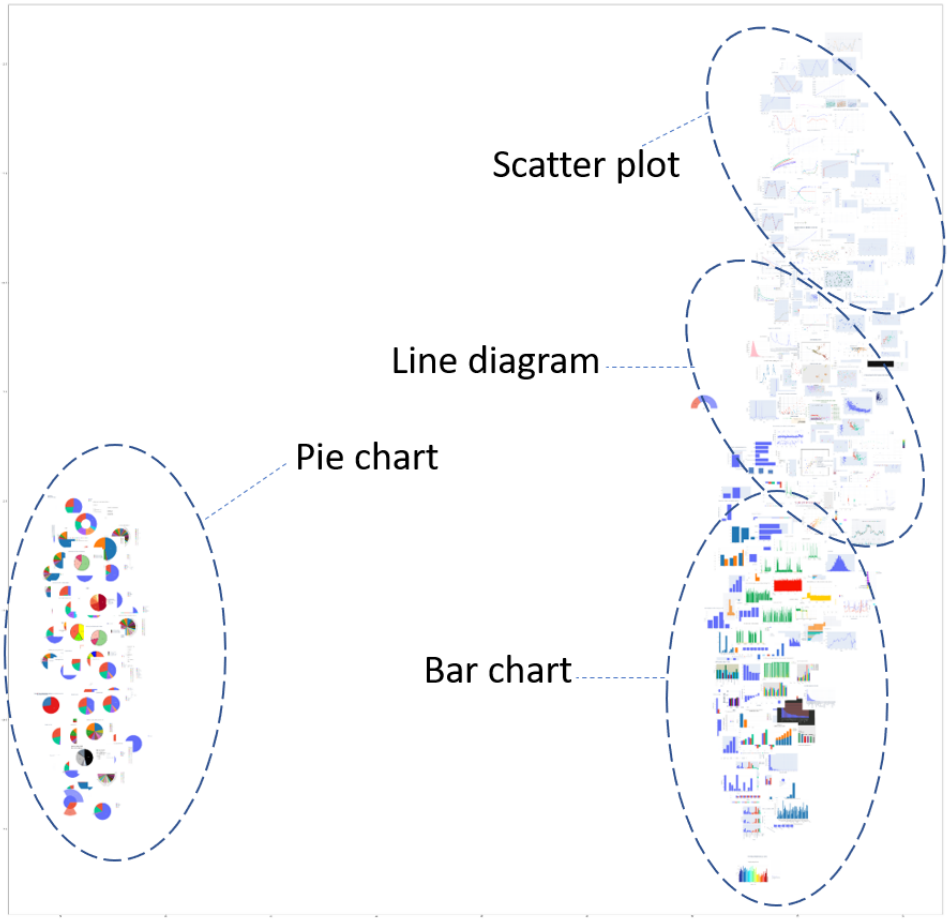


Figure A.3: VGG16 + UMAP splits between the different types of charts
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

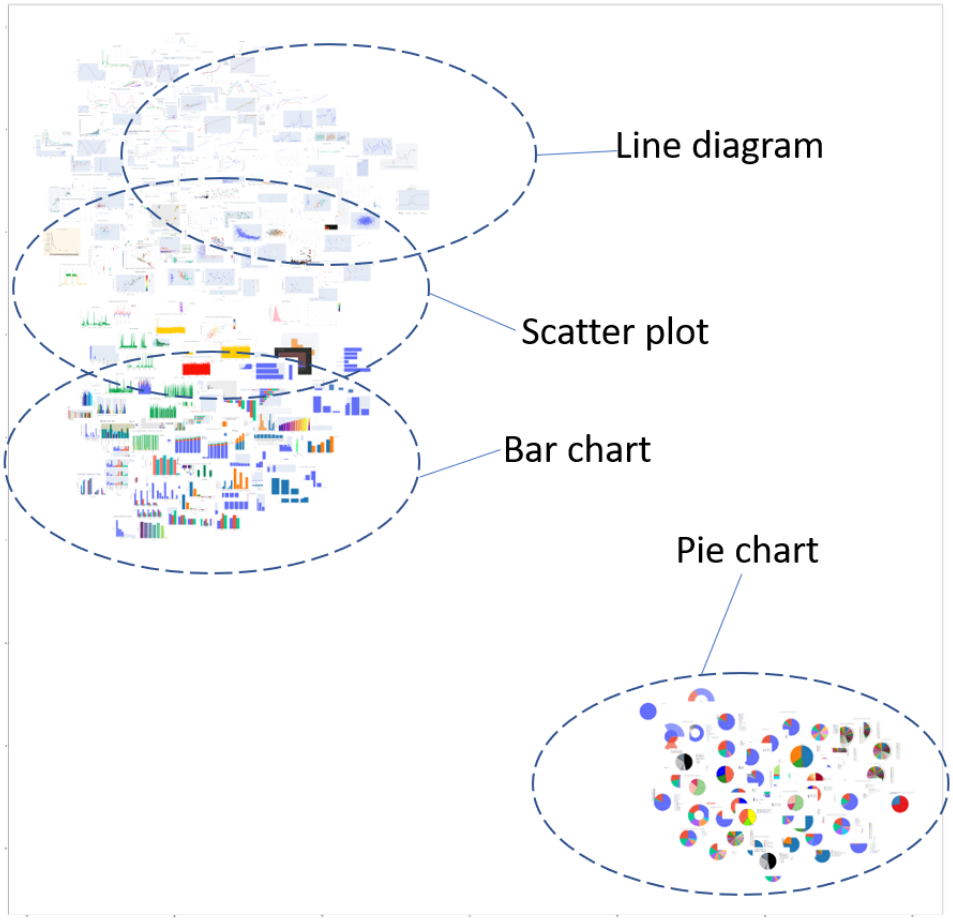


Figure A.4: VGG19 + UMAP splits between the different types of charts with horizontal clusters

UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

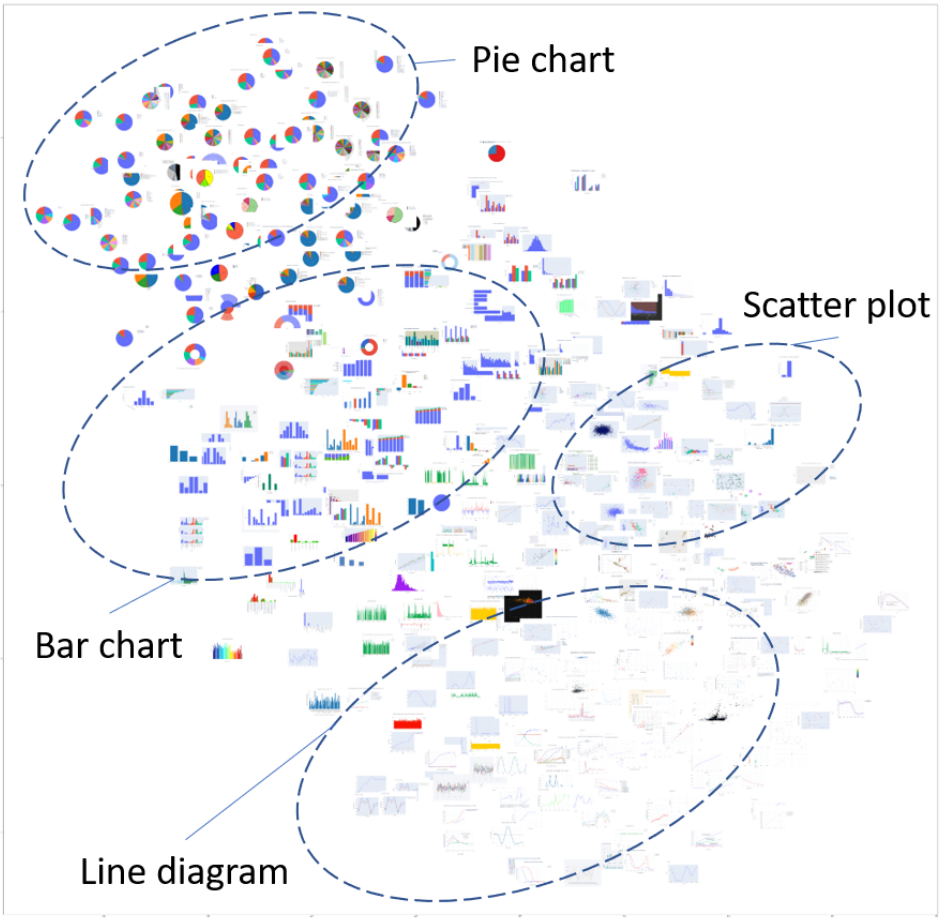


Figure A.5: Inception + UMAP: Bar charts forms a dividing line between pie and other types

UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean

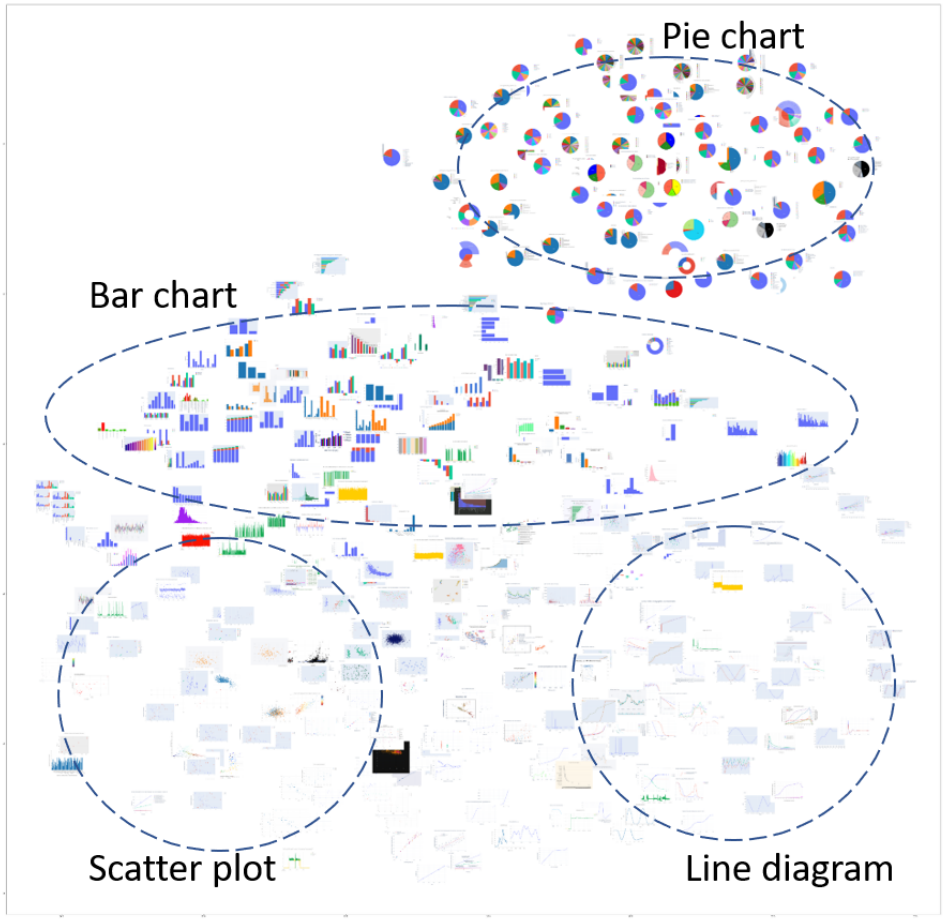
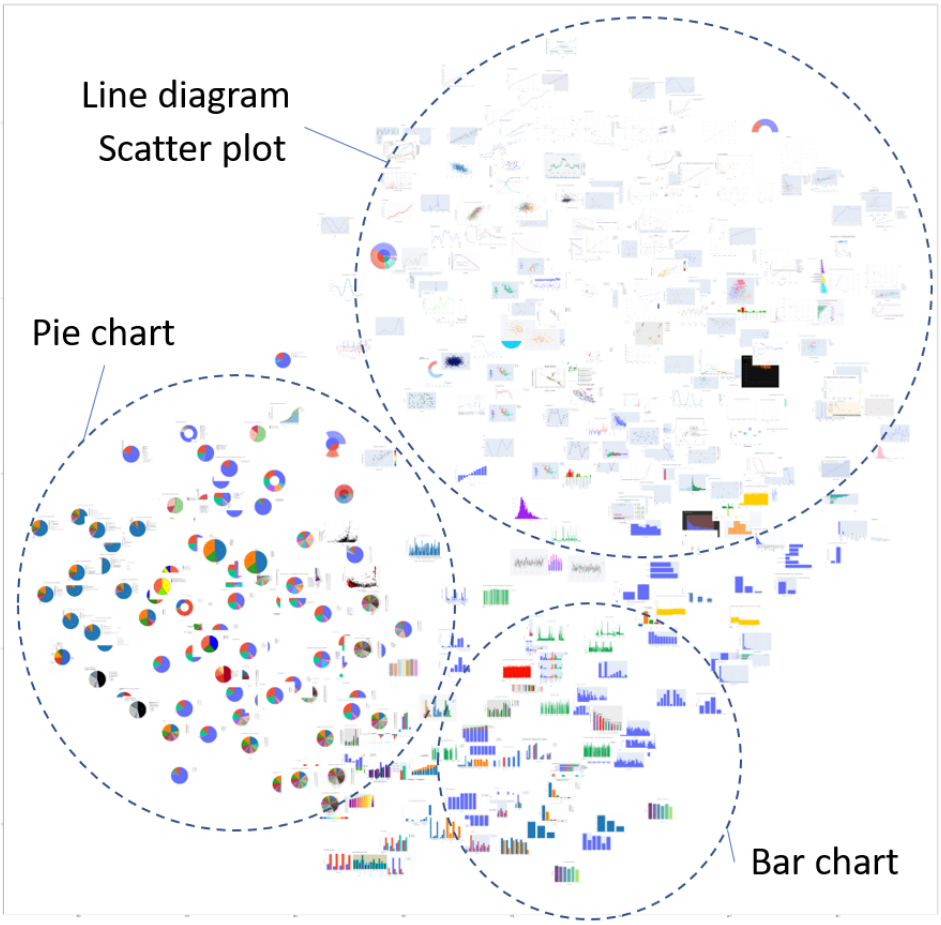
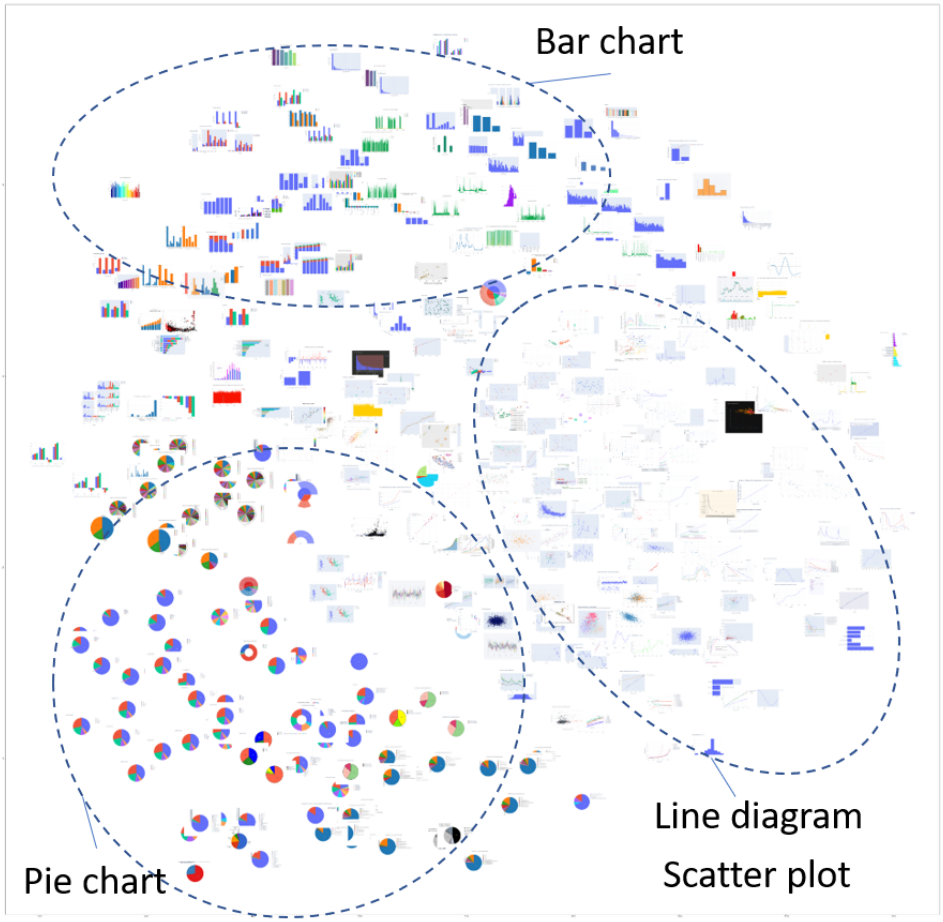


Figure A.6: Xception + UMAP: Bar charts forms also a dividing line similar as in figure A.5
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean



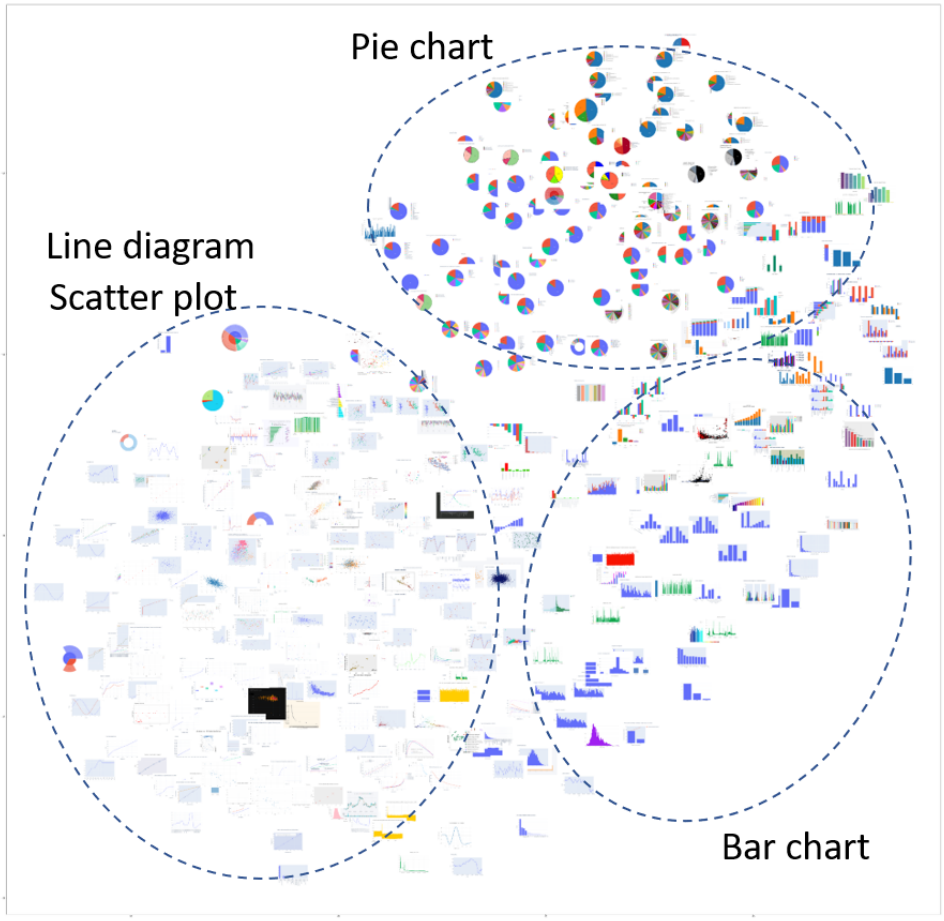
((a)) ResNet50

Figure A.7: ResNet + UMAP overlap between scatter plots and line diagrams
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean



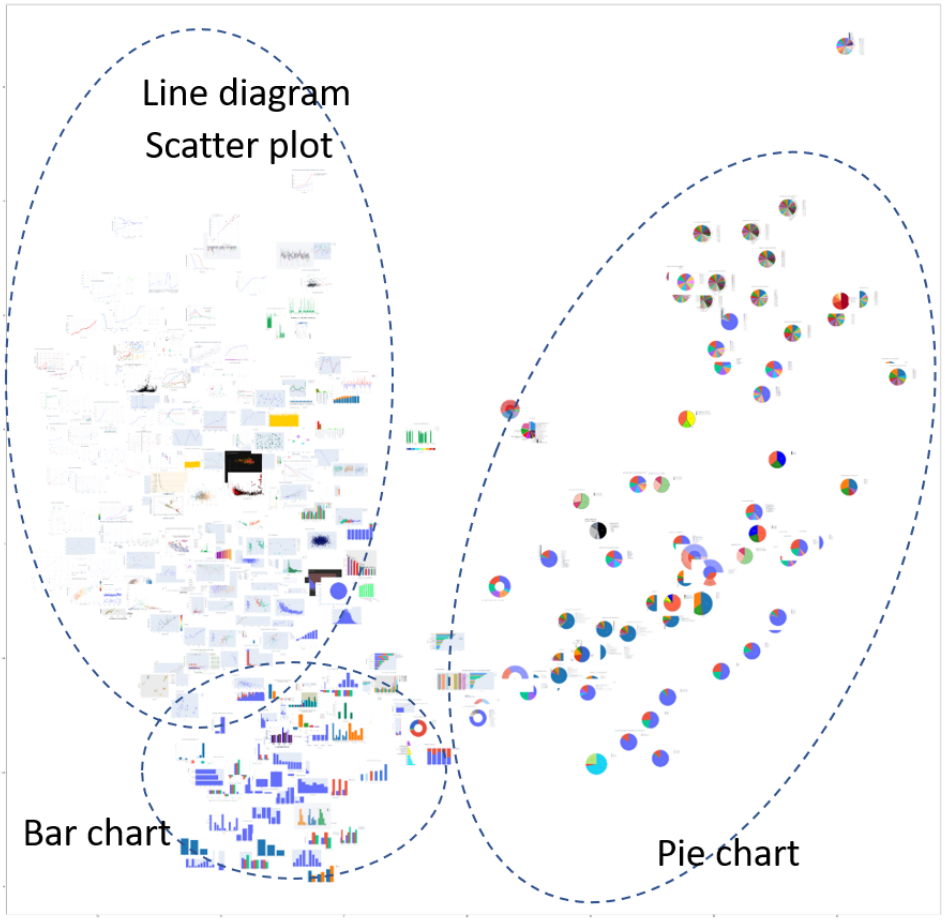
((a)) ResNet101

Figure A.7: ResNet + UMAP overlap between scatter plots and line diagrams
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean



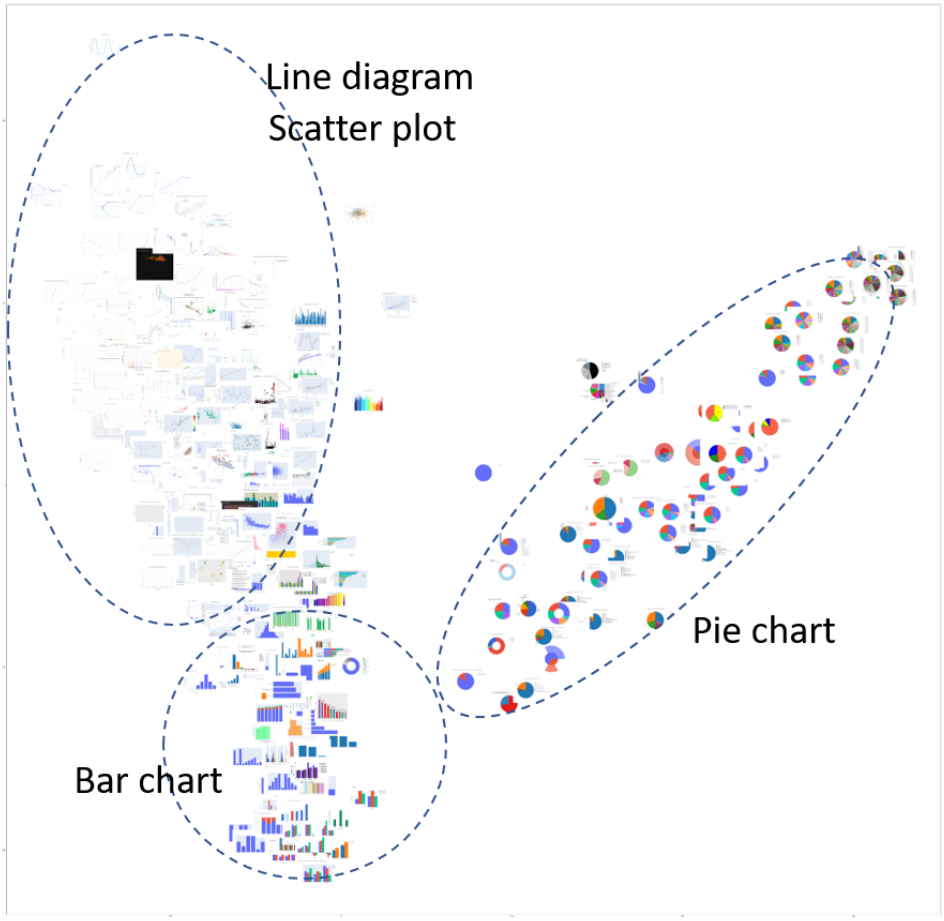
((a)) ResNet152

Figure A.7: ResNet + UMAP overlap between scatter plots and line diagrams
UMAP configuration: neighbors=20, min_distance=0.1, metric=euclidean



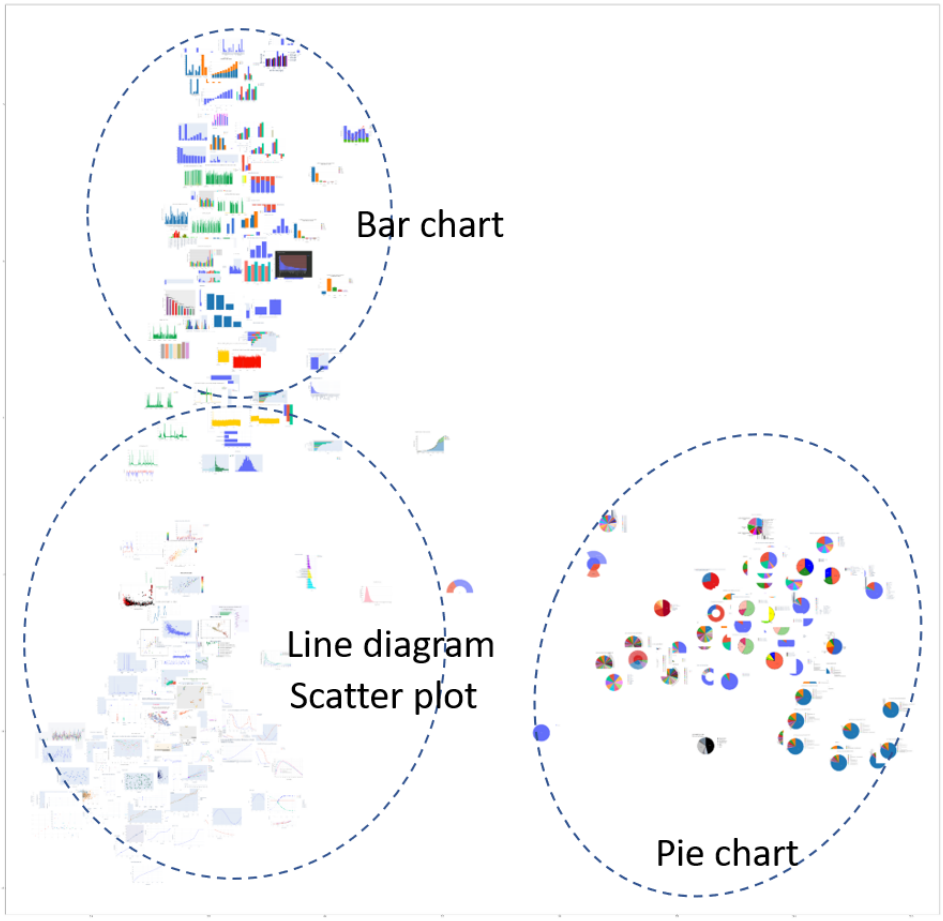
((a)) Inception

Figure A.8: Inception & Xception + PCA detects more small feautres that spreads the three detected type as in figure A.9



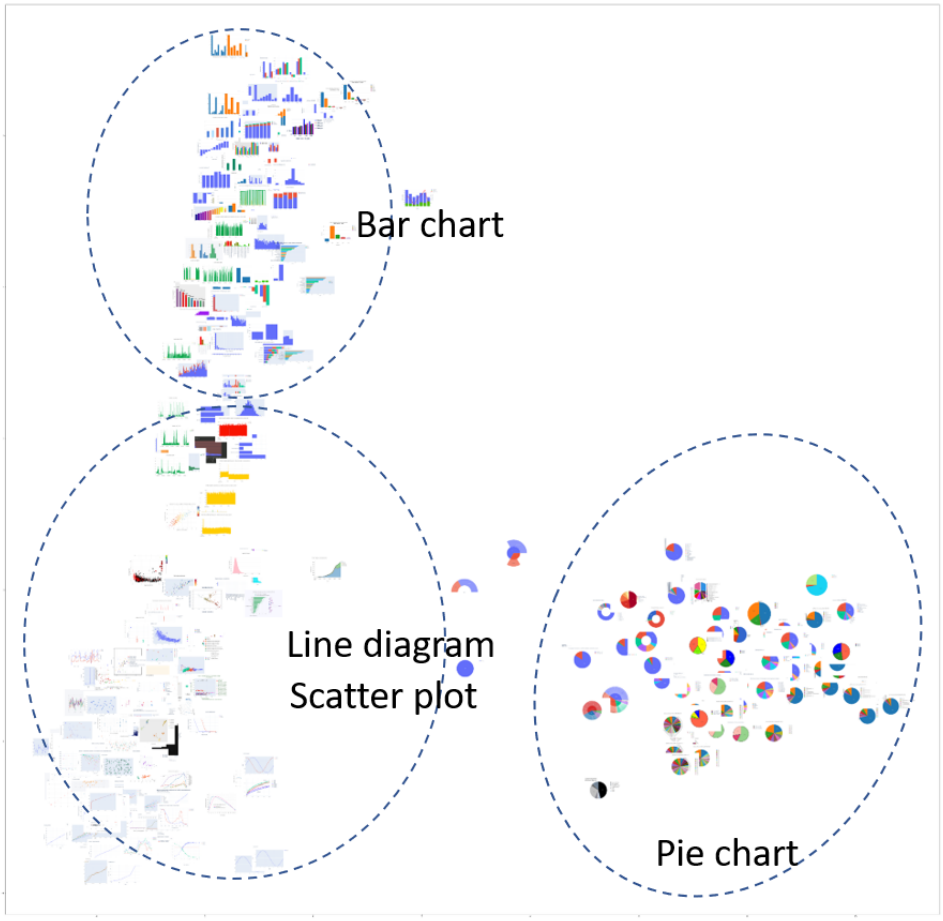
((a)) Inception

Figure A.8: Inception & Xception + PCA detects more small feautres that spreads the three detected type as in figure A.9



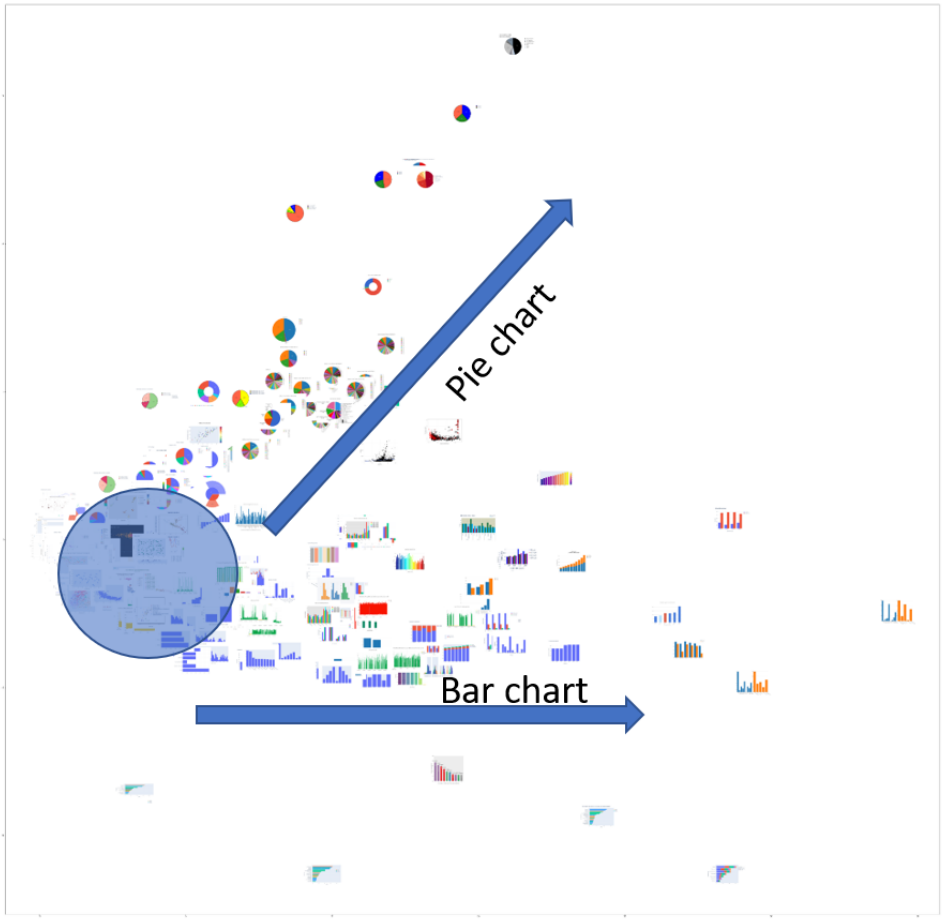
((a)) VGG16

Figure A.9: VGG + PCA defines three main types of four given



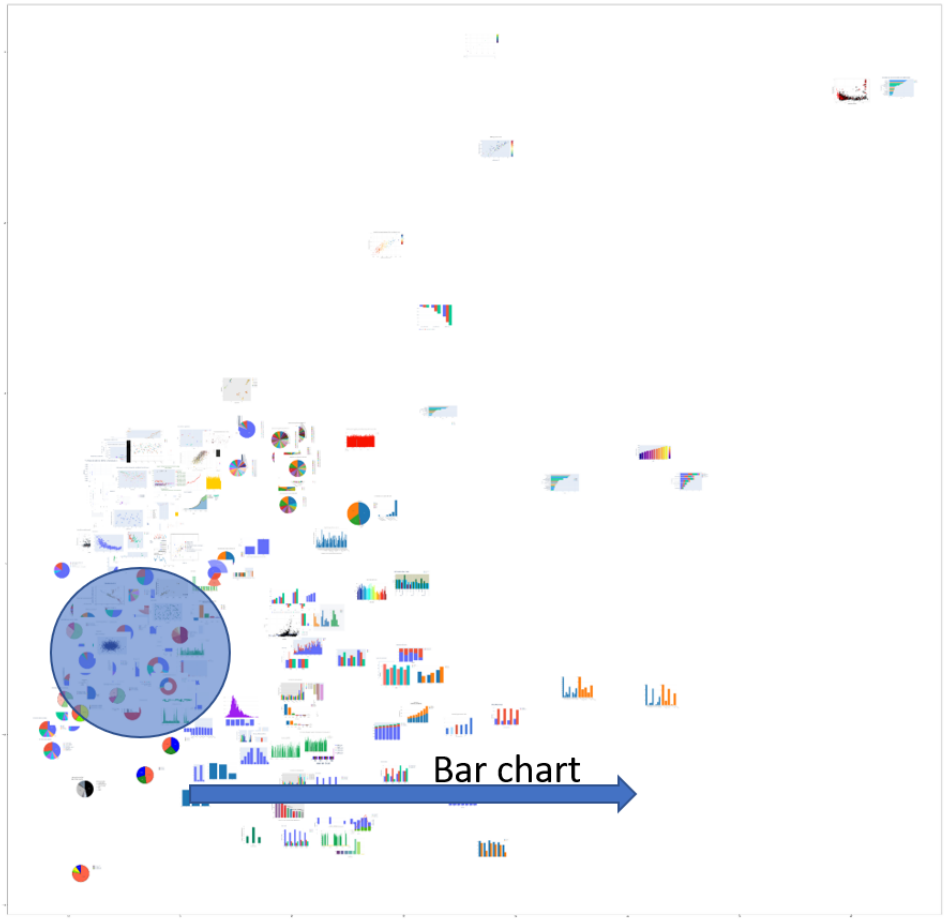
((a)) VGG19

Figure A.9: VGG + PCA defines three main types of four given



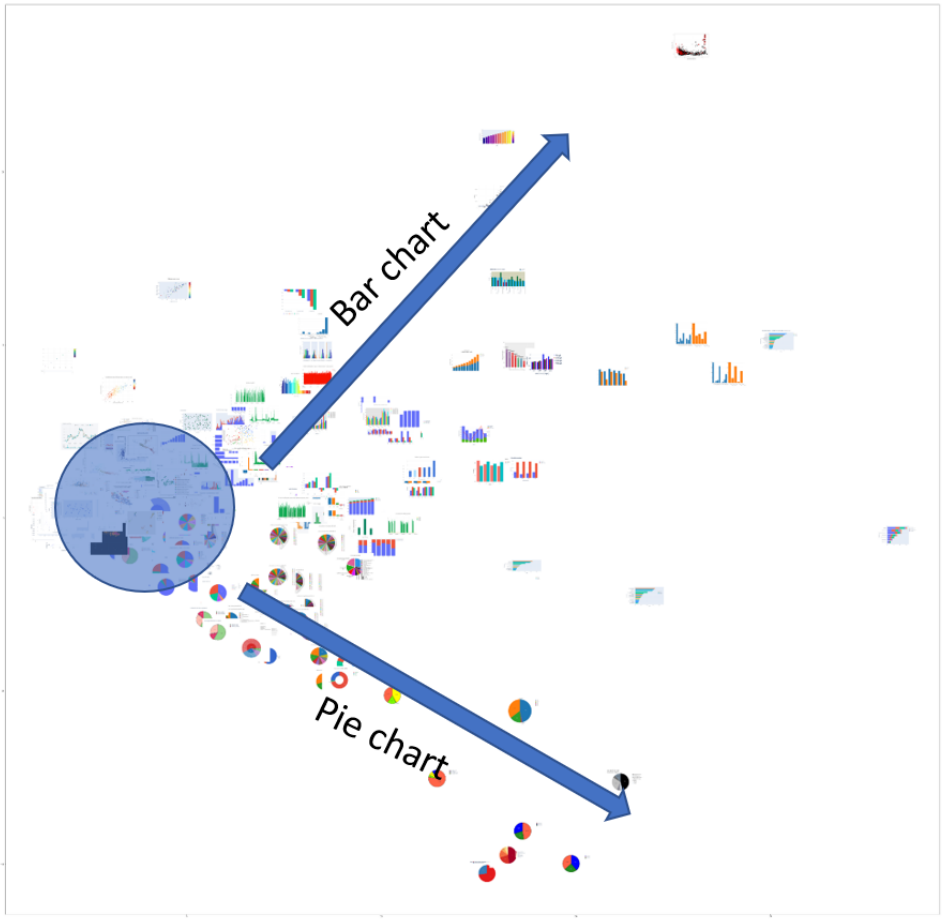
((a)) ResNet50

Figure A.10: ResNet + PCA forms a central cluster that overlaps all types



((a)) ResNet101

Figure A.10: ResNet + PCA forms a central cluster that overlaps all types



((a)) ResNet152

Figure A.10: ResNet + PCA forms a central cluster that overlaps all types

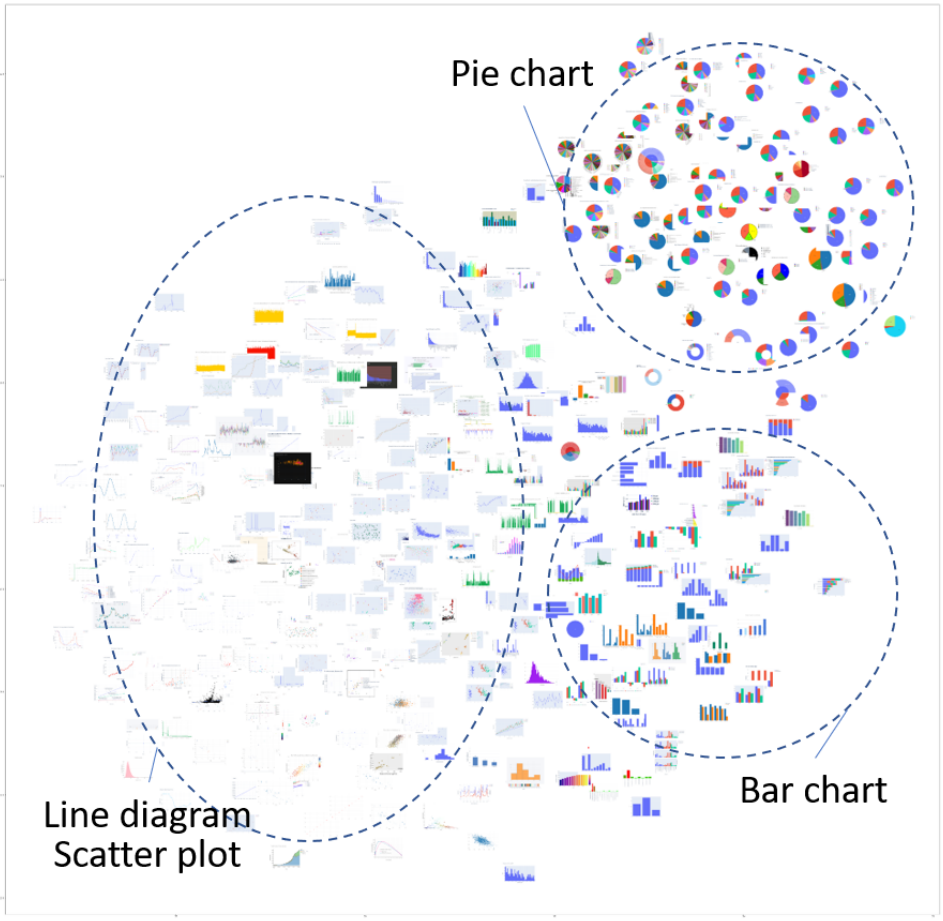


Figure A.11: Inception with average pooling + UMAP builds three intersected clusters

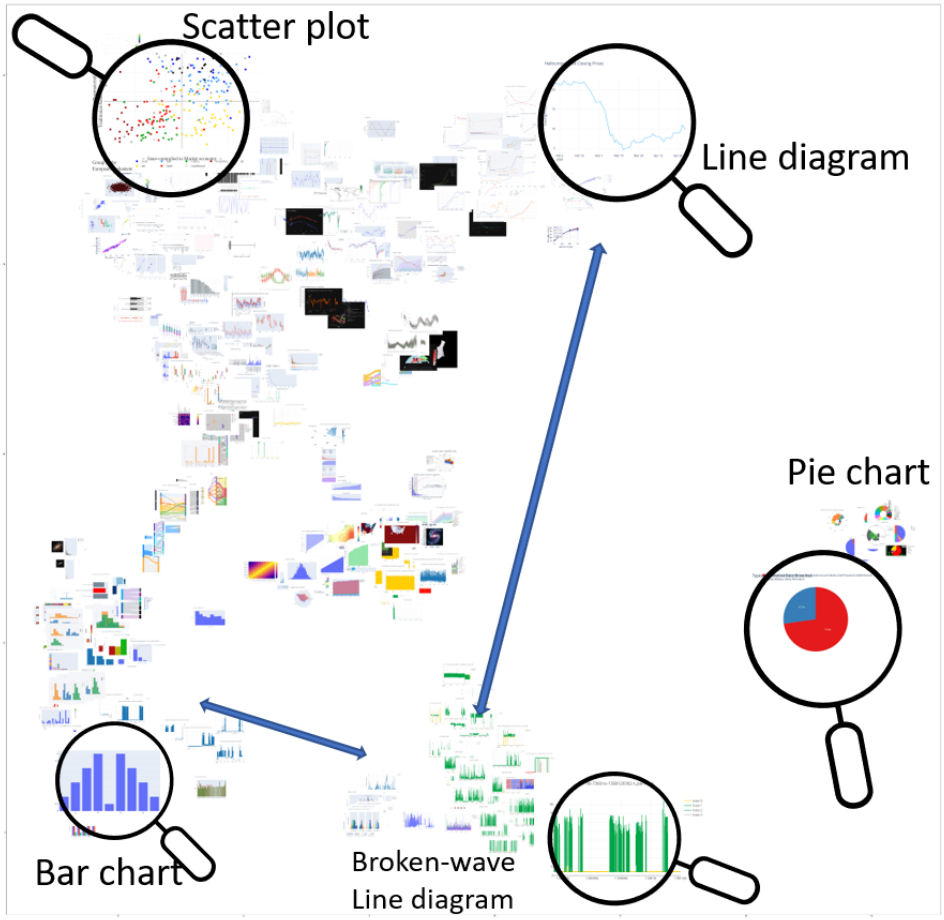


Figure A.12: Model can split the charts' types as well as wave lines when charts are randomly selected

Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

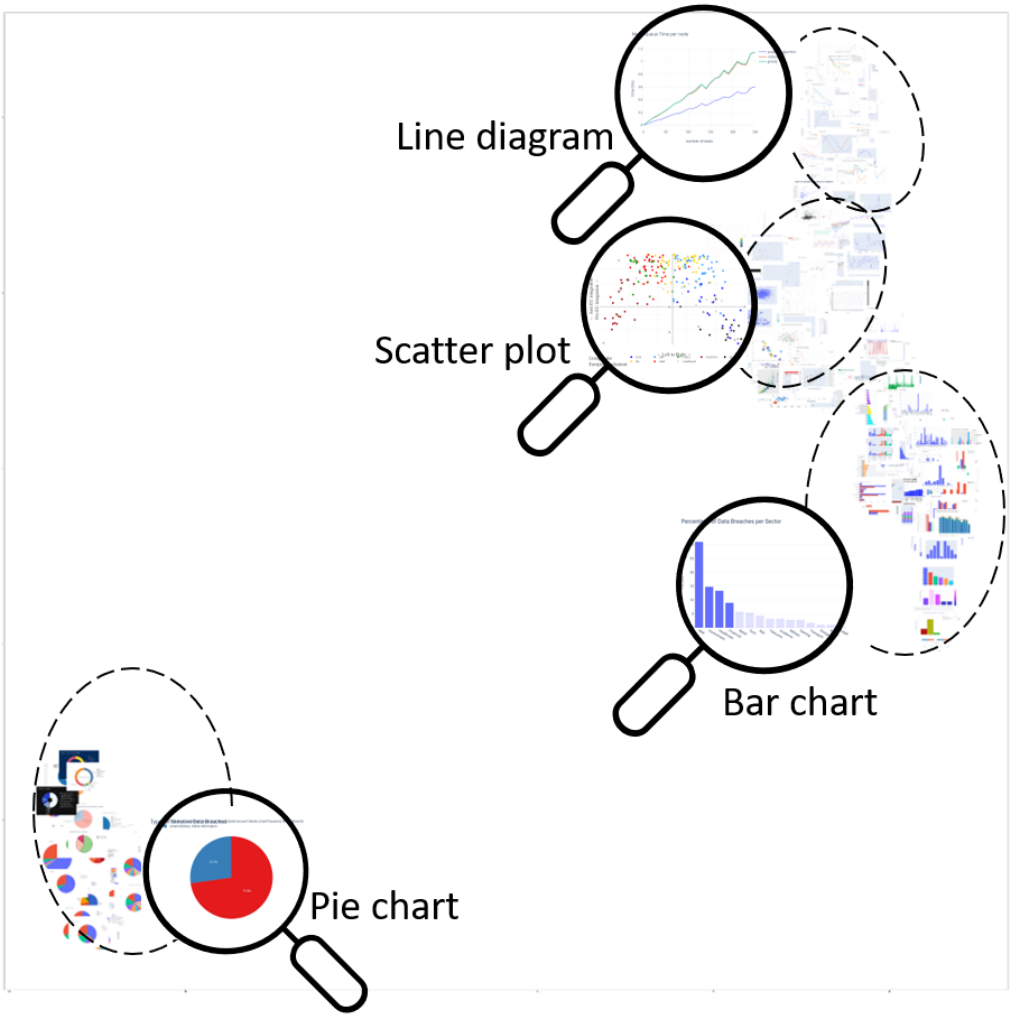


Figure A.13: Model can split the charts' types similar when sampling is completely systematized

Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

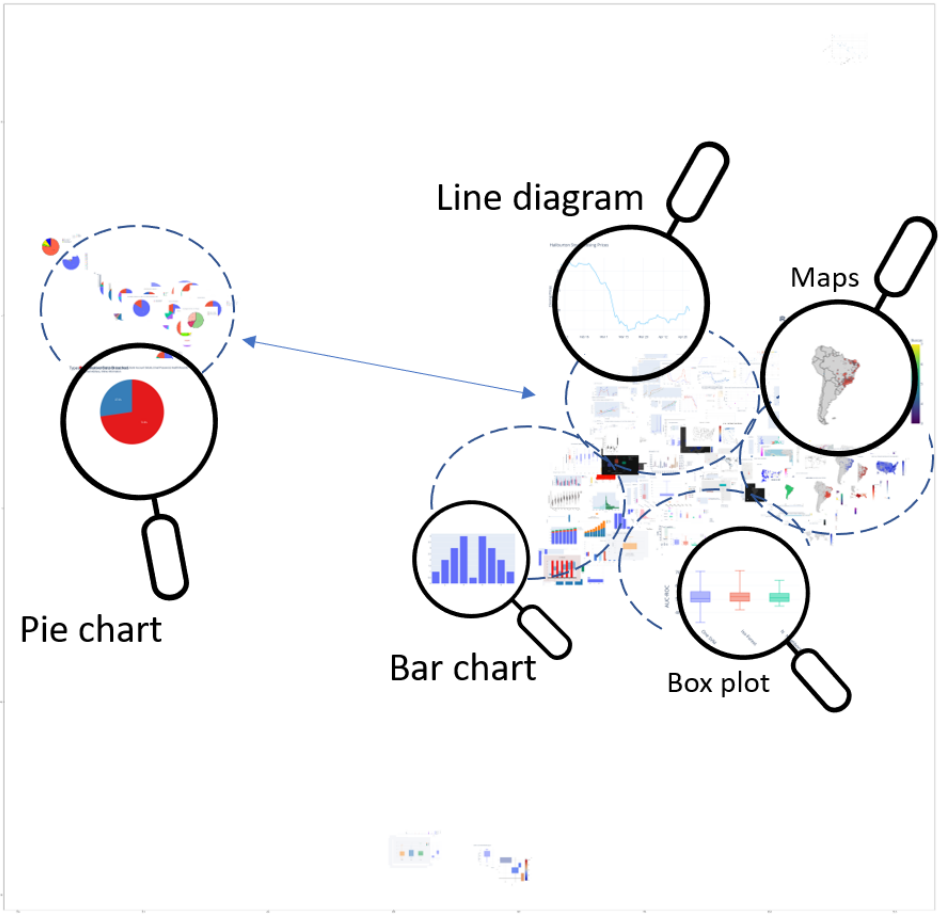


Figure A.14: Pie charts spot subsides from other type clusters
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

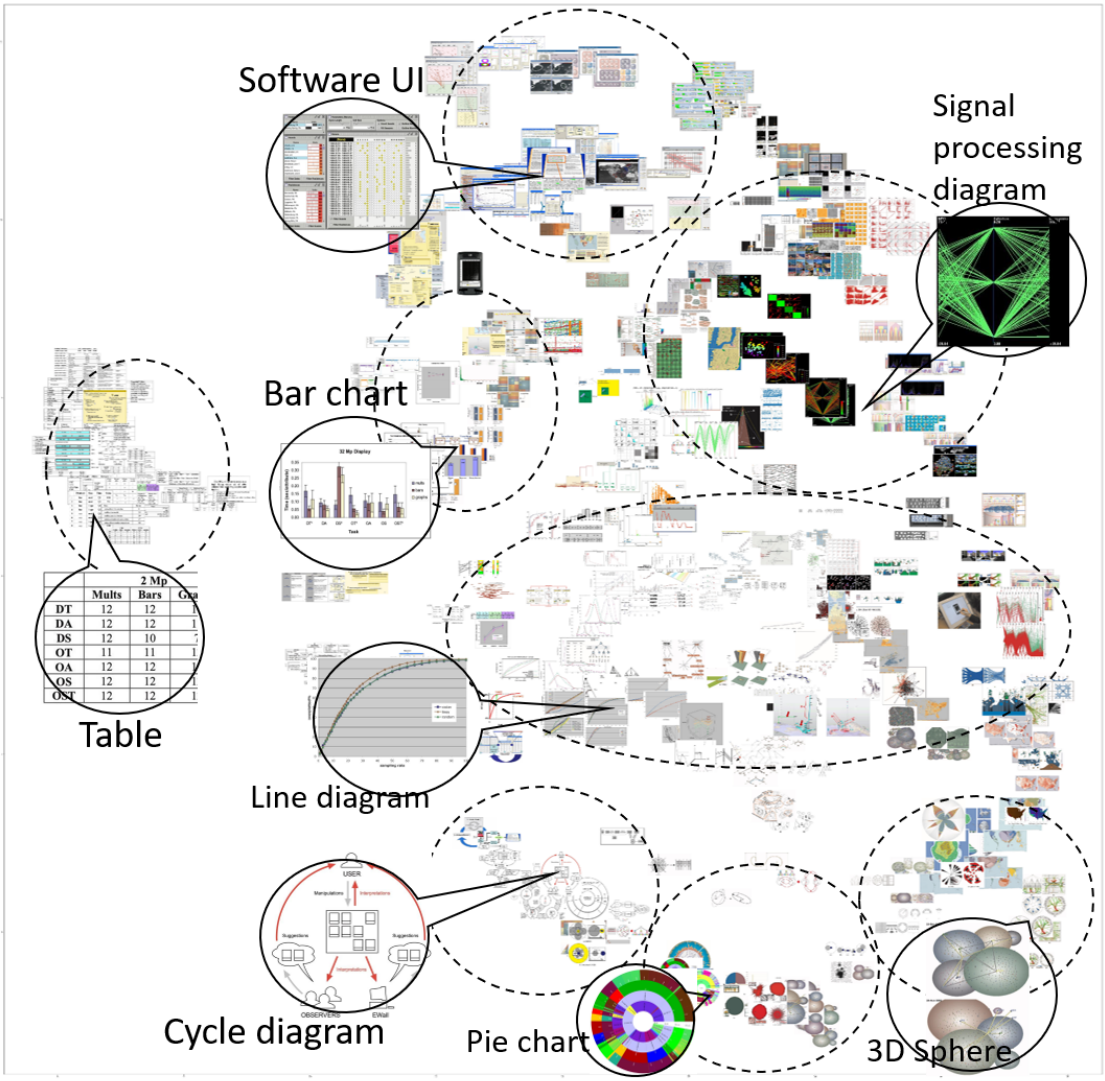


Figure A.15: Model identifies various visualizations and clusters similar ones together
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

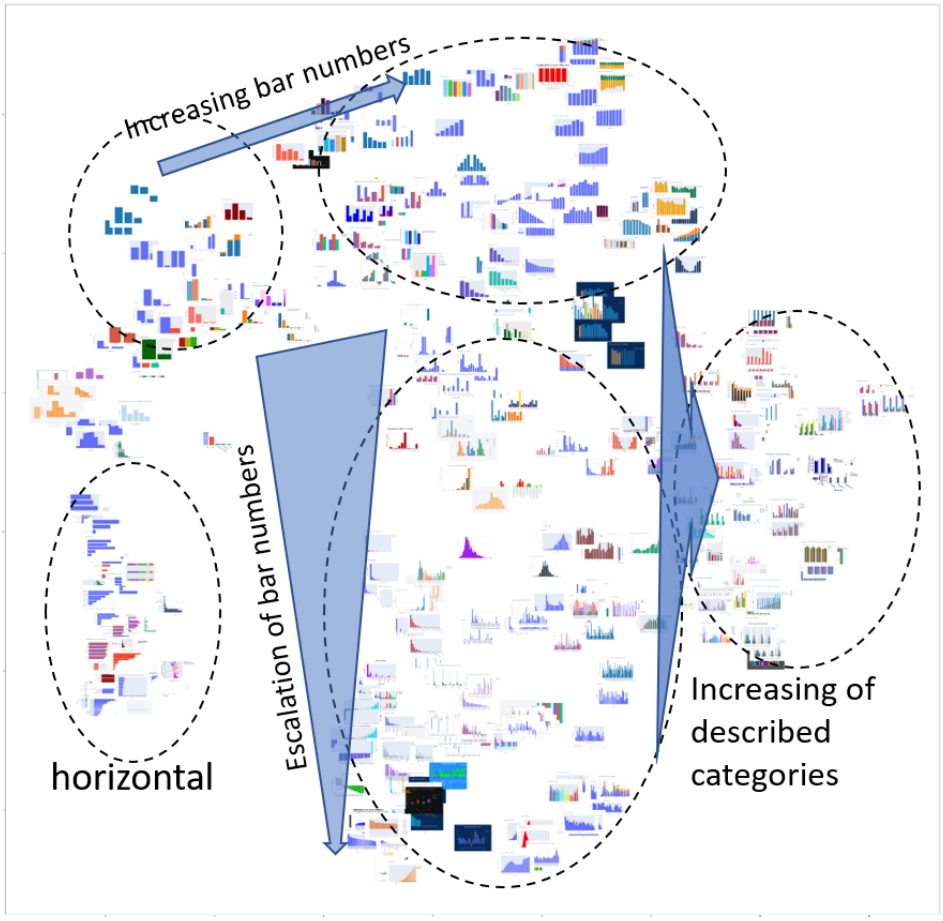


Figure A.16: numbers of bars increases with lower values on the 2D space
Model config: CNN: VGG19 - DR:UMAP n_neighbors=20 min_dist=0.1 metric=euclidean

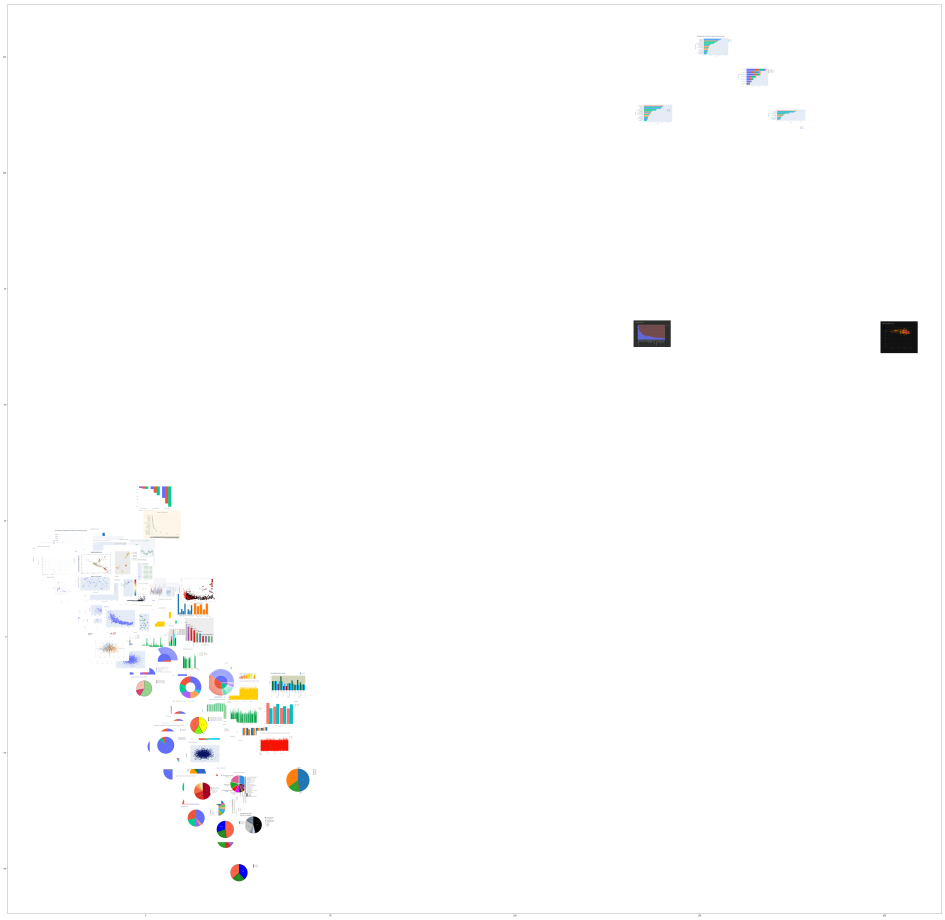


Figure A.17: Applying PCA on encoding output of autoencoder

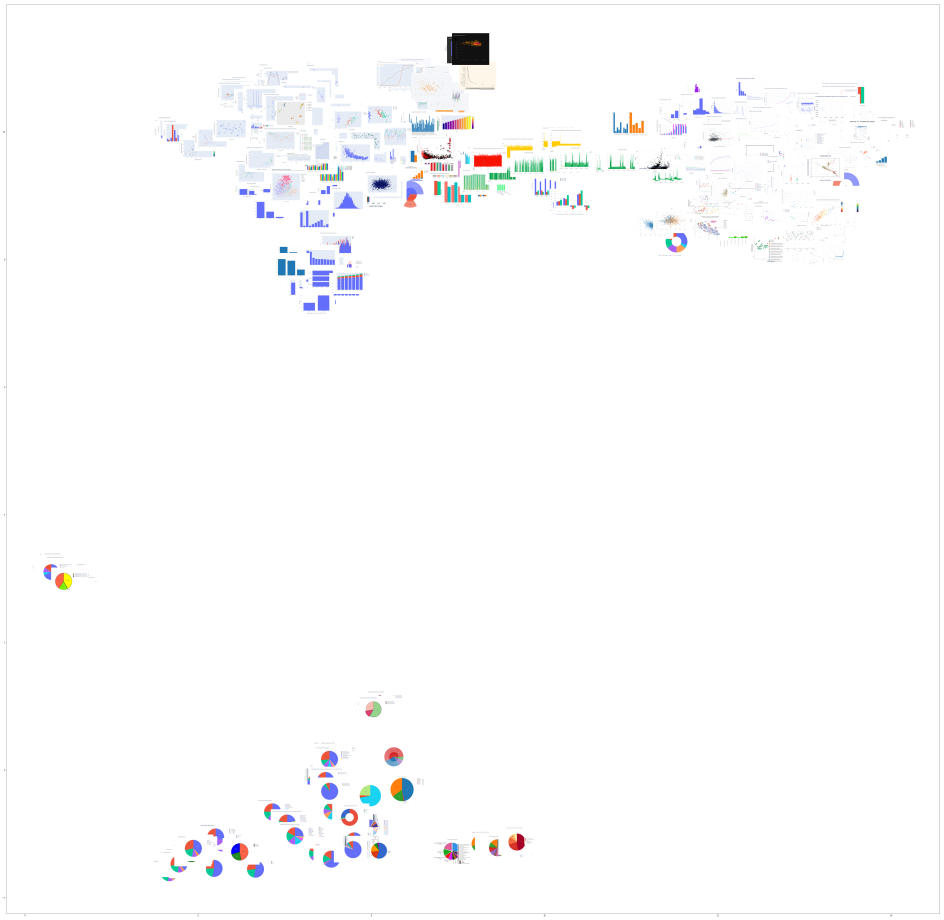
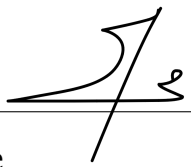


Figure A.18: Applying UMAP on encoding output of autoencoder
UMAP parameters: n_neighbors=15, min_dist=0.1, metric=euclidean

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 30.10.2020 

place, date, signature