1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Miguel Eduardo Pinto Galvão

MACHINE LEARNING FOR THE EVALUATION OF VISUALIZATION MODELS

Dissertation in the context of the Master in Data Science and Engineering, advised by Prof. Evgheni Polisciuc and Prof. João Nuno Gonçalves Costa Cavaleiro Correia and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

1 2 9 0

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Miguel Eduardo Pinto Galvão

Machine Learning for the Evaluation of Visualization Models

Dissertation in the context of the Master in Data Science and Engineering, advised by Prof. Evgheni Polisciuc and Prof. João Nuno Gonçalves Costa Cavaleiro Correia and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

Acknowledgements

I would like to thank my family, my girlfriend and my friends for their support throughout this journey. Without them, I would not have been able to finish this dissertation.

Additionally, I want to thank Professors Evgheni Polisciuc and João Correia for their continuous support and guidance throughout the year.

Abstract

With the increased use of Data Visualization, allied with the growing trend of the amount of data generated and processed, it is important that its reading is done in the most efficient way possible, both in terms of speed and accuracy. For this, we propose a method of evaluation based on graphical perception: "the visual decoding of information encoded on graphs". Applying a method with Machine Learning (ML) models where we perform a regression of the values represented in a chart, we get an estimation error associated with the reading. From this error, we can compare the performance of different visualization techniques, contributing to figuring out which are the best according to our criteria. For that, we tested different ML models, choosing the most suitable one for this task, and studied its capabilities and limitations by analyzing the results from various experiments, concluding that this method works but the proposed model might not be the most suitable to apply it.

Keywords

ML4VIS, Machine Learning, Data Visualization, Visualization Models, Visualization Evaluation, Deep Learning, Convolutional Neural Networks

Resumo

Com o aumento do uso de Visualização de Dados, aliado ao aumento da quantidade de dados gerados e processados, é importante que a sua leitura seja o mais eficiente possível, quer a nível de velocidade como de precisão. Para isto, propomos um método de avaliação baseado em perceção gráfica: "a descodificação visual de informação codificada em gráficos". Aplicando um método com modelos de Aprendizagem Computacional (AC) em que fazemos a regressão dos valores representados num gráfico, obtemos um erro associado com a sua leitura, que mede a capacidade de perceção visual proveniente de um tipo de visualização. Com base neste erro, conseguimos comparar o desempenho de diferentes técnicas de visualização, ajudando a perceber quais as melhores segundo o nosso critério. Para isso, testamos diferentes modelos, elegendo o mais apto para esta tarefa e estudamos as suas capacidades e limitações analisando os seus resultados ao longo de diferentes experiências, concluindo que este método funciona mas que o modelo proposto pode não ser o mais adequado para o aplicar.

Palavras-Chave

ML4VIS, Aprendizagem Computacional, Visualização de Dados, Modelos de Visualização, Avaliação de Visualização, Aprendizagem Profunda, Redes Neuronais Convolucionais

Contents

Acronyms

AE Autoencoder.

AI Artificial Intelligence.

API Application Programming Interface.

CNN Convolutional Neural Network.

DL Deep Learning.

MAE Mean Absolute Error.

ML Machine Learning.

ML4VIS ML for Visualization.

MLAE Mean Logarithmic Absolute Error.

MLP Multi-Layer Perceptron.

MSE Mean Squared Error.

NN Neural Network.

RL Reinforcement Learning.

SVM Support Vector Machine.

SVR Support Vector Regressor.

VAE Variational Autoencoder.

List of Figures

List of Tables

Chapter 1

Introduction

Developments in computing power have greatly contributed to the advancement of graphics for visualizing data [Friendly, 2008]. This fact, allied with the growing trend of the amount of data generated and processed [Holst, 2021], has led to the increased demand for the use of Data Visualizations, to help analyze and make sense of this large amount of information. Similarly, Artificial Intelligence (AI) and ML techniques have also been a target of immense investigation and great evolution in the past few years [Aggarwal et al., 2022] and have proved to be extremely useful in many areas. Due to that, there seems to be no reason why we shouldn't combine these two areas.

This dissertation presents a topic of research which aggregates two big fields, Data Visualization and Machine Learning. The main goal is to apply the latter to the former to contribute to its improvement. ML for Visualization (ML4VIS) is a relatively recent topic, but under great focus from the scientific community in the past few years [Wang et al., 2022; Wu et al., 2021]. In order to understand this emerging topic, a deeper understanding of each field is needed.

Data Visualization has been around for centuries now, mostly revolving around basic maps and diagrams up until the 17th century. From there, measurements of time, distance and space started drawing more attention, as progress in geometry and coordinate systems rose [Friendly, 2008]. New domains and graphic forms were adopted, and new data representations were adopted as more data began being collected. In order to better analyze, extract and present information hidden in the data, visualization can help by making use of the human's visual capabilities of detecting patterns, trends and outliers [Heer et al., 2010]. As more data starts being collected, the more important it is to properly process it visually. A valid approach would be to search for new and innovative techniques that allow for efficient analysis. However, developing novel visualization techniques is not the only approach. Existing ones might just be accurate enough to achieve the desired results. The problem is figuring out the most appropriate techniques depending on the data, the task being carried out and the user, which is a big challenge discussed deeper in chapter 2.

Machine Learning (ML), on the other hand, is a subfield of Artificial Intelligence (AI) in which algorithms are used for computers to learn and solve tasks on their own, based on a set of data called training data. These algorithms are based on statistical methods to enable machines to improve with experience [Aggarwal et al., 2022]. Several application domains and subdomains can be identified within ML applications, including tasks such as Prediction, Classification, Natural Language Processing and Computer Vision, among others [Shinde and Shah, 2018]. These techniques can be (and have been) applied to many areas. In medicine, to diagnose diseases or assess future risks. In finance, to detect fraudulent transactions or to predict the behavior of stock prices. Speech recognition, language translation, fault prevention, recommendation systems, self-driving vehicles, object detection in images are only a few examples of what can be done. As we mentioned before, ML is still a growing field, as many researchers tackle on new approaches and new areas in which to apply these techniques. Giant companies like Google, Facebook and Twitter invest huge amounts of money in Deep Learning (DL) research, a subset of ML that apply complex Neural Network (NN) [Aggarwal et al., 2022]. These attempt to mimic the neural network of the human brain, thus resulting in very complex structures but showing very promising results in many fields. One such example is the area of Data Visualization, which this dissertation aims to contribute to.

In order to understand the purpose of ML4VIS, it is important to mention the factors that motivate this dissertation. This project started in a research grant where the goal is to integrate an evaluating node in an adaptive visualization framework, that aims to automatically present the most suitable visualization model according to the data in question. It is based on two main cycles, which combined compose the execution of the framework.

The first cycle starts with the visualization pipeline [Card et al., 1999b], which transforms raw data into views, followed by the user observation and then by their interaction with the visualization, to carry out a specific task. This interaction leads to an update on the view, restarting the cycle with the visualization pipeline. The second cycle, performed in parallel, is where visual evaluation is conducted automatically. Using ML models, that take the presented visualization and contextual data as input, the resulting information is provided to the adaptation component that adapts the corresponding modules in the visualization pipeline, according to a set of predefined rules. By combining these cycles, the framework generates graphics that adapt themselves automatically based on visual evaluation by ML models. If the suitable model for a certain visualization technique is not found, it is submitted to offline learning where a new model is trained to evaluate that visualization type. This final component is the core of the research presented in this dissertation.

The evaluation of visualization models is intrinsic to their usage. Generally, the model that better allows the correct interpretation of the data, or the one that does it in the easiest way, is the most desirable. However, this assessment can be a difficult challenge as it heavily depends on the human-centered nature of visualization, which requires large-scale empirical studies involving users [Lam et al., 2011]. Several approaches, described in chapter 2, have managed to automate this process by developing rule-based systems that enable automatic evaluation. Even so, they are mostly based on domain knowledge and still require considerable effort to design [Wu et al., 2021].

We intend to go one step further with the goal to fully automate this process by applying ML techniques to assess the quality of different models and determining which works better when it comes to the visual perception of the values. This can lead to a better use of Data Visualization in the present and in the future, especially since data analysis has an increasing importance, due to the growth of areas like Artificial Intelligence and Data Science. Not only better use but also made easier: our ultimate goal is to facilitate the creation and selection of visualizations based on automatic algorithms designed for assessment.

With this dissertation, we intend to search for and test a method to evaluate Data Visualization Models using automated ML approaches. This evaluation consists of the visual perception of graphics: how easily or accurately can the values represented in a data chart be perceived. By training a machine to "read" a type of visualization, and therefore extracting a set of values, we can calculate its error compared to the true values used to create the chart, obtaining a performance score. That score can be used to compare different visualization types in a way to assess which ones are more efficient (the less error, the better). By doing this, we will be using a machine to automatically conduct this assessment, optimizing the process. The term efficient can be related to both speed and accuracy: if the presented values are easy to perceive, it leads to a faster "reading", as well as a more precise one. In order to better understand this, we provide a deeper explanation along with an illustrative example.

It all starts with a comparison between two (or more) visualization types to show the same data. Let us take the example in Fig. 1.1, which shows the results of the 2016 US Presidential Elections in two different ways: through a pie chart and through a bar chart. Which one is the better representation?

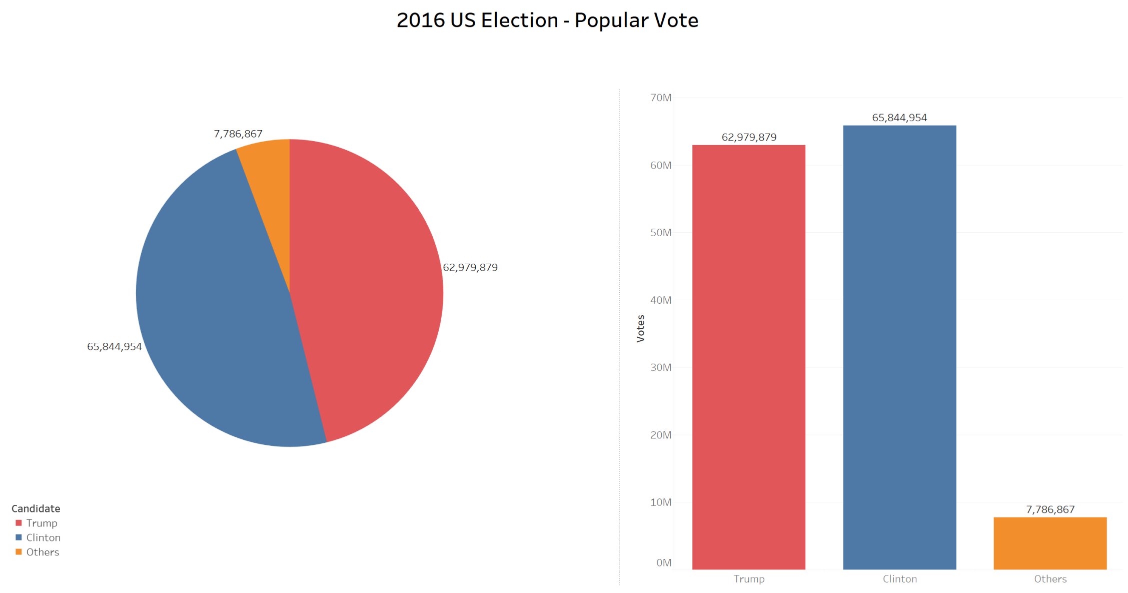


Figure 1.1: 2016 US Presidential Election results presented in two different visualization types [Baddock, 2021].

Many points can be made as to which is the better one. Behrisch et al. [2018] presented a survey where they gather and attempt to aggregate various articles on quality metrics for data visualization, aiming to establish a common vocabulary among different understandings. One of the elements presented is the categorization of numerous metrics according to its cognitive complexity. These can be grouped into low-level, mid-level and high-level metrics, where, for example, low-level metrics relate to low-level perceptual tasks, such as visual search, change detection and magnitude estimation while high-level refer to measures like memorability, aesthetics, and engagement. These last ones are often considered subjective, according to the authors.

Therefore, we aim for a more objective analysis by targeting low-level metrics, namely through **graphical (or visual) perception**. It is important to explicitly define its meaning since it will be used many times throughout the document: this phrase, defined by Cleveland and McGill [1984], relates to "the visual decoding of information encoded on graphs". In other words, the ability to retrieve the original data used to create the visualization. We can consider this criterion as objective since it is based on the human capabilities of processing visual information. By maximizing this aspect as much as possible, we can improve the accuracy and/or speed of a user reading a chart. But how can we use this concept to compare different representations?

In their paper, Cleveland and McGill [1984] presented several experiments that attempt to measure human visual perception. Such an example is the "position-angle" experiment, where the authors had users estimate the percentage of the values relative to the maximum, within different charts. In order to better understand this, we draw the reader's attention to Fig. 1.2. Let us say the two charts are presented and we ask a human user to estimate the values for A through E, in relation to the maximum value, D, in both cases. Assuming the value for D is 100, the goal is for the user to try to provide the remaining values. The reader may notice the number 40 on the vertical axis of the bar chart and be confused as to how D can be 100, but the actual values have no influence on this task. We simply assume D as 100 since it is the largest. This way, the exercise is to estimate the percentage of the values in relation to the maximum. For example, if one of the values appears to be half the size of the maximum one, we should assume its relative value is 50. We invite the reader to also attempt this exercise to see whether the result follows the previously tested theory on graphical perception presented by Cleveland and McGill [1984].

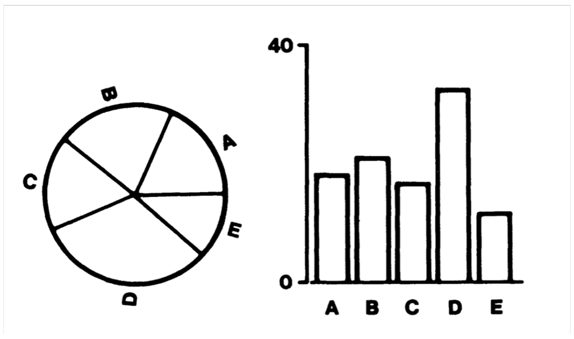


Figure 1.2: Labeled examples of a pie chart and a bar chart [Cleveland and McGill, 1984].

The user starts with the pie chart, on the left of Fig. 1.2, estimating the values for A through E as 40, 60, 50, 100 and 25, respectively. Then, the user does the same for the bar chart on the right side of the figure, concluding that the values represented for A through E correspond to 53, 60, 50, 100 and 35. Now, knowing the original values and the corresponding real (approximate) percentages, we can calculate the estimation error for the user. So taking the true values (62, 68, 55, 100 and 39) and the values estimated for the pie chart (40, 60, 50, 100 and 25) we can calculate the difference between each one, thus calculating the error associated with each variable, and then summing all those to compute the overall estimation error for the reading of the chart, as it is shown in the following expression, where A through E represent the true values and Â through Ê represent the estimated values:

$$\begin{gathered}|A-\hat{A}|+|B-\hat{B}|+|C-\hat{C}|+|D-\hat{D}|+|E-\hat{E}|= \\ |62-40|+|68-40|+|55-50|+|100-100|+|39-25| \\ =22+28+5+0+14 \\ =69\end{gathered}$$

As we can see, an error of 69 was obtained by summing the error associated with each variable. Doing the same for the values estimated for the bar chart (53, 60, 50, 100 and 35), which was generated from the same original data, we get the following expression:

$$\begin{gathered}|A-\hat{A}|+|B-\hat{B}|+|C-\hat{C}|+|D-\hat{D}|+|E-\hat{E}|= \\ =|62-53|+|68-60|+|55-50|+|100-100|+|39-35|= \\ =9+8+5+0+4= \\ =26\end{gathered}$$

In this case, from the bar chart, the estimation error was smaller, only 26. What can we conclude from this? Obviously, from this small sample of examples, nothing can be concluded, but one could argue that, possibly, bar charts can be better than pie charts in terms of the ability to decode the information presented. The point of this example is to illustrate how one could develop an experiment to try and compare the performance of humans in terms of graphical perception, with a proper number of users and examples. In their version of this experiment, with 54 human users and 20 different charts, Cleveland and McGill [1984] concluded that bar charts were superior representations in terms of human graphical perception when compared to pie charts, which is in accordance with the example previously shown. We intend to expand this reasoning used for the comparison of data visualization models to machines. The logic is the same, but instead of having a human look at papers with charts and asking them to provide estimations, we set up a ML model which takes as an input a digital version of those same charts and outputs the estimation of the values presented. This way, we can use a fully automatized process to perform a comparison between different visualization types.

In order to achieve this goal, we proceed with the following objectives:

- Define and develop a setup for the experiments to be conducted, including which models and visualizations to test.
- Run baseline experiments to compare the performance of different models.
- Assert the most suitable model for the task and test its capabilities and limitations with several variations.
- Analyze the obtained results in order to draw conclusions.

By the end of the work, we expect to have a model capable of being trained to "read" different types of data visualizations, allowing for their evaluation based on the performance. This evaluation can then be used for their comparison, thus supporting the decision-making of the adaptive visualization framework previously mentioned. More importantly than that though, we hope this work will serve as a starting point for further research on the field of ML4VIS because, despite our efforts, we are unable to develop a perfect model for this method of visualization assessment. We hope to contribute by sharing our results and conclusions regarding capabilities and limitations.

This document is divided into six chapters, the first being the current one, presenting the introduction of the dissertation. This includes the motivation, context, research statement and a brief overview of our goals.

Chapter 2 is where the state-of-art on the topic is explored, by analyzing and reflecting on past work from other authors.

Chapter 3 presents a comparison of the original proposed work plan with the actual sequence of tasks carried out for the development and its respective schedule.

Chapter 4 provides a description of our approach to the task and outlines the development conducted.

Chapter 5 describes all the analysis conducted on the results obtained from our experiments.

Chapter 6 wraps up this dissertation by reflecting on our results, discussing key takeaways and limitations of our work.

Chapter 2

State of the Art

Throughout this chapter, the state of the art in Machine Learning (ML) for visualization evaluation is presented. By analyzing research that has previously been conducted one can better understand which paths to follow and which to avoid to further explore the topic in question.

This chapter is divided into three sections. The first presents data visualization fundamentals, from the visual encodings that form a visualization, their efficiency and up to their evaluation. Both classical and automated approaches for this task are presented. The second section focuses on the use of ML for visualization. First, some basic concepts are explained, followed by general applications of ML for Visualization (ML4VIS), ending with uses of ML specifically for visualization evaluation. The third section summarizes the contents and reflections presented.

2.1 Data Visualization Fundamentals

Meirelles [2013] explores and explains the concept of Information Design. The challenge of representing information with multiple dimensions in the usual two-dimensional visual display is not as obvious as it seems. The process of designing such visualizations is dependent on visual perception and cognitive processes. It is essential to understand the constraints and capabilities of cognition in order to develop efficient visual designs: if the process of decoding the information present fails, so does the visualization altogether.

The term Information Design is used to describe communication design practices, used mainly to provide information, as opposed to more persuasive methods used, for example, for advertising. Many outputs can be produced, such as infographics, that combine graphic elements with textual information, wayfinding systems, used to assist in navigation, or visualization of statistical data. What all of these have in common is the objective of revealing patterns and relationships that in other ways would be unknown or hard to identify.

Related to Information Design is the term Data Visualization (or information visualization). These correspond to the use of computer-supported, interactive, visual representations of abstract data to amplify cognition, that is, acquiring knowledge in an effective and efficient way [Card et al., 1999a].

Another important aspect mentioned by Meirelles [2013] is the purpose of visualizations. They can serve as a communication tool, a way to present stories or research findings, but can also be used for data manipulation and exploration, especially through user interaction, another very relevant component of Data Visualization. We can look at it as a complement or strengthening element of our mental abilities.

In this section, we present a few basic principles about Data Visualization, in a way to better understand the purpose of research on this topic and how it can contribute to it. Additionally, we also dig into studies and approaches conducted towards the goal of improving this field.

One of the main components of these representations is the set of visual encodings used to translate the information into graphical features [Heer et al., 2010]. This element is explored in the following subsection.

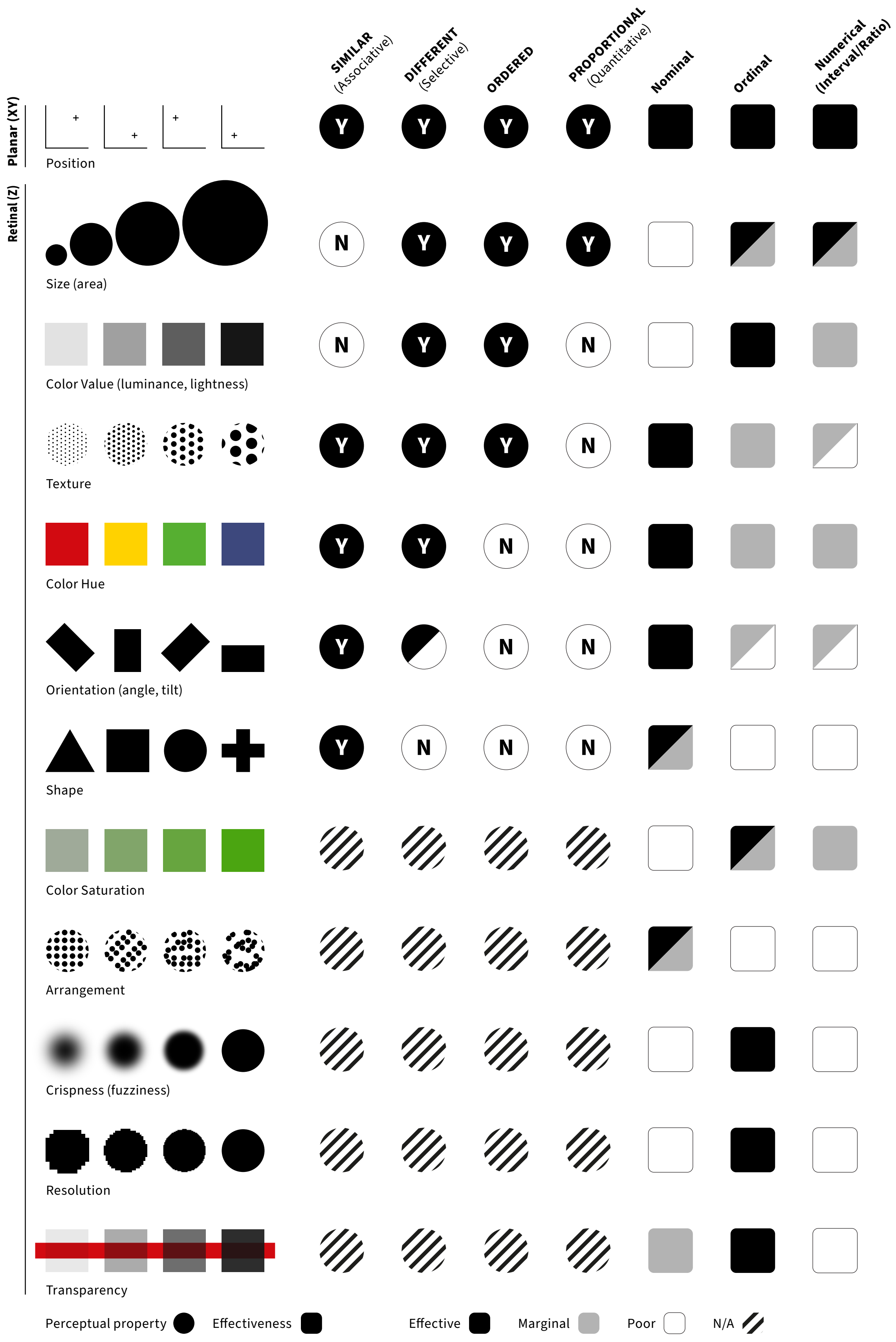
2.1.1 Visual Encodings and their Efficiency

Visual Encodings, or visual variables, describe what graphical dimensions are used in a visualization to transmit information [Roth, 2017]. As we mentioned, the decoding of these variables is done through graphical perception. Cleveland and McGill [1984] studied and tested human graphical perception in their article. They identified elementary perceptual tasks that are used to extract information from charts, providing examples of basic models that apply them. They also ordered them according to how accurately people perform them, by testing with subjects. In other words, they compared the performance of different tasks and drew conclusions on which ones should be prioritized in order to maximize accuracy when creating graphs.

This perception analysis is the base of the use of visual variables. Many authors over the years have contributed to list the different types of variables available [Bertacco, 1975; Bertin, 1967/1983; MacEachren, 1995]. Roth [2017] has condensed them in this article, as well as their perception, syntactics, and conjunctive uses. Some examples of such visual variables include location/position, size, color and orientation, and those examples, among others, are shown in Fig. 2.1.

By combining visual encodings, many visualizations can be created. Common data graphs, like bar charts, pie charts or line charts are basic examples. However, more complex examples can be created. Heer et al. [2010] showcases more sophisticated techniques, compared to basic graphs, prepared to display complex and diverse datasets. By using several visual channels simultaneously, more information can be encoded at the same time, allowing for a better visualization and better data analysis.

As we can see, different visual encodings can be used for different types of data (nominal, ordinal, numerical), with different levels of efficiency. For that reason,



		SIMILAR (Associative)	DIFFERENT (Selective)	ORDERED	PROPORTIONAL (Quantitative)	Nominal	Ordinal	Numerical (Interval/Ratio)
Planar (XY)	Position	Y	Y	Y	Y	■	■	■
Retinal (Z)	Size (area)	N	Y	Y	Y	□	◪	◪
	Color Value (luminance, lightness)	N	Y	Y	N	□	■	▒
	Texture	Y	Y	Y	N	■	▒	◩
	Color Hue	Y	Y	N	N	■	▒	▒
	Orientation (angle, tilt)	Y	◐	N	N	■	◩	◩
	Shape	Y	N	N	N	◪	□	□
	Color Saturation	▨	▨	▨	▨	□	◪	▒
	Arrangement	▨	▨	▨	▨	◪	□	□
	Crispness (fuzziness)	▨	▨	▨	▨	□	■	□
	Resolution	▨	▨	▨	▨	□	■	□
	Transparency	▨	▨	▨	▨	▒	■	□

Perceptual property ● Effectiveness ■ Effective ■ Marginal ▒ Poor □ N/A ▨

Figure 2.1: Visual Variables and their effectiveness in encoding different types of data [Polisciuc, 2021].

the choice of visual variables to use is a challenge to maximize the efficiency of a visualization. In fact, Munzner [2009] proposes a visualization model with four levels, with the third one being the choice of visual encodings and interaction techniques. They also discuss the problems that can appear by the wrong use of variables, as well as some methods to validate possible options. However, the validation (or evaluation) of visualization aspects is a much bigger challenge that has been at the center of many studies and articles. That discussion is done in the next subsection.

2.1.2 Visualization Models Evaluation - Classic Approaches

Why is visualization evaluation important? This might seem like an obvious question, easily answered with "to get the best visualizations possible". While it is true, the reasons behind this evaluation go a bit deeper. Good evaluation can help people choose the most suitable techniques or parameters to use, which can help convince users to integrate new methods or tools [Wijk, 2013]. This leads to innovation, which can then lead to better progress in all fields where visualization is applied.

But even though the concept seems simple enough, many challenges appear when performing this assessment. One of the main ones, pointed out by Wijk [2013] is the complexity of the visualization process. The four components of this process (users, tasks, data and artefacts) can have multiple variants, which makes the evaluation method very complex. Users can have different backgrounds or degrees of knowledge, tasks can be low or high-level, and data can be of many types and domains. Kim and Heer [2018] and Saket et al. [2019] tackle this issue by analyzing the effects of tasks and data distribution of the effectiveness of visual encodings, going beyond the initial work done by Cleveland and McGill [1984]. Plaisant [2004] mentions a similar issue of matching visualization tools with users, tasks and problems. Another challenge mentioned by Wijk [2013] is the generalization of findings. That issue is naturally related to the high complexity of the previous challenge since an assessment for a specific combination of users, tasks and data might not be the same for a different one. In fact, generalization is one of the three main factors when considering a visualization study, along with precision and realism, as first mentioned by McGrath [1995] and then referred by Carpendale [2008] and Wijk [2013] when considering different approaches that attempt to balance them.

This opens up many possibilities in terms of how to conduct this evaluation, or even when. Lam et al. [2011] mention evaluation can be used in different stages of a visualization development: in pre-design, design, prototype, deployment and re-design. In their article, they aggregate known approaches into seven different scenarios, mentioning what questions are asked in those scenarios and providing related literature.

1. Environments and Work Practices (EWP) is where the focus is on the environment or the processes in which the visualization and its tools are used. [Isenberg et al., 2008] is an example where this is applied. They explore

ground evaluation, in which the visualization tool is analyzed within the context where it is used, so as to adapt it to better perform.

2. Visual Data Analysis and Reasoning (VDAR) analyzes how well the visualization tool supports analysis and reasoning of the data, that is, generating knowledge/information.

3. Communication through Visualization (CTV) is the assessment of communication tasks using visualizations, such as learning/teaching, presenting or simple consumption of information. The goal is to see how well these tools perform conveying a message.

4. Collaborative Data Analysis (CDA) - how well a tool allows for collaboration, to have a group of people conducting analysis together, helping in joint decision-making, conclusions or discovery.

5. User Performance (UP) studies if and how certain features can affect UP. Using task completion time or task accuracy, it is possible to conduct that evaluation. [Heer and Bostock, 2010; Hu et al., 2019; Kim and Heer, 2018] are some examples of such evaluation being used. Controlled experiments (user tests) are the most used technique, but system logs can also be analyzed to see how a user interacts with the tool or what settings they use to figure out their performance.

6. User Experience (UE) relates to a user's subjective feedback, opinions on how well the visualizations work for the designated tasks, to discover new requirements or needs.

7. Automated Evaluation of Visualizations (AEV) - automatic/computer-based evaluation approaches. Here, the goal is to output quality or efficiency scores.

An interesting note is that UP, UE and AEV represented 85% of the papers analyzed in this summary, meaning those were the main areas being focused on at the time of publication. A combination of UP and AEV is what we look for with this research.

As we can see, apart from AEV, all the scenarios mentioned imply the use of human subjects to participate in the evaluation project. Most approaches mentioned in other papers that also attempt to summarize evaluation methods face this issue [Carpendale, 2008; Munzner, 2009; Wijk, 2013]. The need for subjects had Heer and Bostock [2010] analyzing the viability of crowdsourcing for graphical perception, replicating, once more, the work of Cleveland and McGill [1984] using a tool from Amazon, Mechanical Turk (also used by Hu et al. [2019]). One way to ease this process could be done through the use of heuristics. Zuk et al. [2006] developed a decision support systems is based on general rules and guidelines for visualization design, which can be easy and quick to apply. However, we believe the answer lies in the automatization of evaluation processes, which is the topic discussed in the following subsection.

2.1.3 Visualization Models Evaluation - Automatic Approaches

Based on the past work analyzed, we conclude that the main goal of automating the evaluation process is to support the tasks of visualization generation [Mackinlay, 1986; Moritz et al., 2019], recommendation [Ehsan et al., 2018; Lee et al., 2019; Wongsuphasawat et al., 2016a,b] and improvement [Bryan et al., 2017]. These tasks are obviously related, since the goal is the same: provide the most effective versions of data visualization.

One of the first examples of automated model evaluation comes from Mackinlay [1986]. He developed a tool that uses graphical design criteria, expressiveness and efficiency, and a composition algebra to create visualizations automatically, just like generating sentences from a certain language. The composition algebra is composed by the graphical marks/techniques identified by Bertin [1967/1983]. This tool concentrates on generating designs that can be accurately interpreted. The two sets of criteria mentioned are the following:

- Expressiveness criteria - These identify graphical encodings that express the desired information, all of it and nothing more (to avoid possible incorrect extra information).
- Effectiveness criteria - These identify which of these graphical encodings, in a given situation, is the most effective at exploiting the human visual system. It can be based on the comparison of the perceptual tasks required by alternative graphical variables.

Mackinlay [1986] extended Cleveland and McGill [1984]'s ranking of graphical perception and uses that knowledge to compare approaches in order to figure out the most effective options.

Although the tool does not directly evaluate visualizations, that process is implicit since it compares different options based on theoretical analysis and previously acquired knowledge, outputting the one that maximizes effectiveness and expressiveness. Therefore, we can identify this approach as knowledge-based method of automated evaluation.

This tool, named "A Presentation Tool" (APT) was a pioneer in this field, making way for other similar, yet improved, tools. One such example is Draco [Moritz et al., 2019], where they attempted to re-implement APT. The goal was to develop a system that represents design guidelines as a collection of constraints. Some of these constraints are hard, so as not to violate basic design principles, but others are soft and are used to rank visualizations. The soft constraints are represented through weights that can be updated according to user preference or professional experience. That way, the system can keep up with research. The weights were initialized according to graphical perception studies [Cleveland and McGill, 1984; Kim and Heer, 2018; Saket et al., 2019], just as with APT. These constraints are used to calculate the cost (or score) of a visualization, thus allowing for a scoring/ranking system.

However, a different set of methods of automated evaluation can be identified. Statistical-based methods analyze properties of the data to assert the most effective ways of visualizing it to gather insights. Wongsuphasawat et al. [2016b] presents a tool, named Voyager, that allows a mixture of manual and automatic visualization generation by having a user input of variables and encodings, but also recommending some options. Voyager privileges data variation (different variable selections and transformations) over design variation (different visual encodings of the same data), which is a different strategy of improving data analysis: focusing more on which information is presented and how it is processed before being encoded. Voyager is based on a recommendation engine called Compass. This engine, presented in an article of its own [Wongsuphasawat et al., 2016a], is capable of enumerating, clustering and ranking visualizations according to data properties and graphical perception principles. Some examples of the use of statistics in Compass include the analysis of correlation between data attributes and presence of outliers. Those criteria can help better rank available visualization options, allowing the recommendations suggested to the user in Voyager.

Another way of using statistics for evaluation, suggested by Ehsan et al. [2018], is by analyzing numerical attributes of data, possible aggregations and binning of values in order to present better insights. Even though it seems simple enough, the combinations of these possibilities are nearly endless. To reduce this number, they apply a multi-objective utility function that captures the impact of numerical dimension attributes. The goal of the function is to measure interestingness, usability and accuracy, balancing each other to achieve the highest performing visualization. Once again, like in the first example of this subsection, although the evaluation is not directly applied to a visualization, it still occurs, but this approach manages to avoid the unnecessary generation of poor examples.

A similar strategy is presented by Lee et al. [2019]. The authors attempt to aid analysts in presenting visualizations that maximize the information output of a dataset. The focus is also in analyzing combinations of attributes, but in addition to selecting combinations that provide more insight, the aim is also in avoiding cases where the data presented is deceiving. For that, they apply a utility function that calculates how much a different visualization contributes in terms of informativeness.

Statistics can also be used to determine certain points in a visualization that should have more focus. Bryan et al. [2017] present an approach that helps to explore Time Series in order to find and annotate points of interest (important timestamps), to improve the quality of the presentation.

In summary, we can condense automated evaluation approaches into knowledge-based and statistical-based ones. In spite of being based on carefully studied design and cognitive principles, knowledge-based methods can be hard to apply since they require the translation of theoretical concepts into algorithms. Not only that, but as we have covered in this section, these concepts are heavily affected by the context in which they are used, making the challenge even tougher to generalize and automate. Statistical-based approaches, on the other end, seem to be more focused on the information output than on the visualization technique itself. Therefore, despite seemingly promising, we believe we can surpass this set of automated approaches by applying Machine Learning methods to evaluate visualizations. That discussion is done in the following section.

2.2 Machine Learning for Data Visualization

As was mentioned before in the introduction, ML can be very helpful in assisting in various fields and contexts, and Data Visualization is no exception. In fact, the number of publications regarding ML4VIS have been steadily increasing during the second half of the past decade [Wang et al., 2022; Wu et al., 2021] which shows an increasing interest in this field. As Wang et al. [2022] put it, "the application of ML techniques can benefit a variety of visualization related problems". In the survey they explore processes related to visualization that can benefit from the use of such techniques. These seven processes are the following:

- Data Processing: improving the efficiency of data processing and enhancing the perception of visualization.
- Data-Visualization Mapping: mapping of data into visual channels and improving the efficiency of visualization creation.
- Insight Communication: interpreting insights and improving efficiency of communication through visualization.
- Style Imitation: imitating color and layout selection from designers' practices.
- Visualization Interaction: understanding natural user interaction and refining results of said interactions.
- User Profiling: predicting user behavior and characteristics.
- Visualization Reading: extracting and interpreting content and estimating human perception.

We can see that most of these processes, while closely related to the visualization pipeline, are less focused on the final product of the graphical representation, except for the final one, which closely relates to our research goal. In this section we explore approaches for applying ML methods in Data Visualization, going over several contexts of this field and focusing on the analysis and evaluation part.

2.2.1 ML Base Concepts

Since in this section we will be presenting ML approaches, a context of basic terms, concepts and definitions is necessary for the reader to fully grasp the topic in hands. This subsection fulfills that requirement.

Starting from the beginning, as previously explained, **Machine Learning** is a subfield of Artificial Intelligence (AI) that allows computers to automatically learn how to solve certain tasks or improve its performance on them, without the need of being explicitly programmed for that [Aggarwal et al., 2022]. This process is based on ML algorithms that use data to train and adapt models to provide a useful output based on new input data. This learning factor is what separates ML from other subfields of AI [Aggarwal et al., 2022].

The data used to train the model, called training data, can be used in different ways for the model to learn. One example is **supervised learning**, where the training data is labeled with the corresponding output so that the model can adjust itself in order to deliver the expected response. **Unsupervised learning**, on the other hand, does not have those labels, leaving it to model to find patterns and relationships in the data on its own. A middle term between these two approaches is **semi-supervised learning**, where part of the data is labeled while the other is not. These two components are used during the training process in different ways to improve the model. Another existing method is **Reinforcement Learning (RL)**, in which a model learns in a trial and error fashion. This is done by assigning a reward (or punishment) to an action/decision from the model, allowing it to adapt to its environment/task [Goodfellow et al., 2016].

For the training process of the models, it is common to split the available data into two sets, a training set and a test set (**train/test split**). As the names indicate, the training set is used to train the model, while the test set is used to evaluate the model's performance. This is done to get a more accurate estimate of how well the model generalizes. Therefore, we test the model on new, unseen data to verify if it didn't simply adapt itself to the training data. If this happens, and the model performs poorly on new data (after performing well on the training data), it is a case of overfitting, where the model is not able to generalize. When the performance is bad in both sets, it is called underfitting [Bonaccorso, 2017]. In some cases it is common to split the data into three sets, with the additional one being a validation set (**train/validate/test split**), which can be used for parameter tuning. Different combinations of parameters can be tested with the validation set to assert which are the highest performing ones. Additionally, it can also be used during the training process evaluate the model's performance and assert if it is possibly underfitting or overfitting. The third set is needed because the final evaluation must always be done on unseen data.

The data used in ML is typically composed of different attributes. For example, in a dataset of houses, each house can be described by its size, location and price. These attributes can also be referred to as **dimensions** or **features**. This information is what the model uses to find patterns and relationships needed to solve the designated tasks. The bigger the feature space (or dimensionality), the more complex the data is, which can make it more difficult for the models to learn.

ML algorithms can solve tasks in many ways. Two of the most common are classification and regression. In **classification** problems, the goal is to predict a class label for a given input. Classification can be binary, with only two classes (e.g. true or false) or multi-class (e.g. fruit type). In **regression**, on the other hand, the goal is to estimate a continuous output value (e.g. the price of a house, given its characteristics).

Many ML algorithms have been developed and presented over the years [Das and Behera, 2017]. Since this subsection is not meant to be a comprehensive list of existing approaches, we will only present those that are mentioned later in the section.

Support Vector Machine (SVM) are a type of supervised learning algorithm that can be used for both classification and regression tasks. The core idea behind SVMs is to find the hyperplane in the feature space that best fits the training data. For classification tasks, SVMs find the hyperplane that maximizes the separation between the classes by maximizing the margin, or the distance between the hyperplane and the nearest support vectors (for multi-class problems, multiple support vectors are used). This is known as the maximum margin classifier [Jakkula, 2006]. For regression tasks, SVMs (known as **Support Vector Regressors (SVR's)**) find the hyperplane that best fits the data by minimizing the error between the predictions and the true output values.

A **Neural Network (NN)** is a machine learning model inspired by the structure and function of the brain, consisting of layers of interconnected nodes (neurons) [Wang, 2003]. NN's are able to learn by adapting the weight of each connection between nodes to improve its predictions. Therefore, they are able to learn complex relationships between input and output data, and can be used for a wide range of tasks. There are several types of neural networks, including feedforward neural networks, convolutional neural networks, relational networks, deep belief networks and autoencoders. NN's with multiple hidden layers (layers between the input and output layers) are considered Deep Neural Networks, and its corresponding subfield in ML is called **Deep Learning** [Shinde and Shah, 2018].

In a feedforward neural network, the input data is fed through the network, layer by layer, without looping back, unlike others where the data can also flow backwards. An example of such a NN is a **Multi-Layer Perceptron (MLP)**, where each node receives an input from the previous layer and combines these inputs using an activation function [Manning et al., 2013]. The output of the last layer is the overall output of the MLP. A simple example of a structure for this type of feedforward NN is presented in Fig. 2.2.

Deep Belief Networks (DBN) are a different type of neural network composed of multiple layers of other neural networks (restricted Boltzmann machines) that apply probabilistic models to extract relevant features from the input in a hierarchical way [Hinton, 2009]. The final layer is used to make predictions based on the input data.

Relational Networks are yet another type of neural networks that are used for data with complex relationships [Santoro et al., 2017]. They are composed of multiple modules, each responsible for part of the input data. Attention mechanisms are used to connect these modules, allowing the network to learn relationships between different subsets of the input. Based on these relationships, relational networks can, for example, join inputs of different types to make more informed predictions.

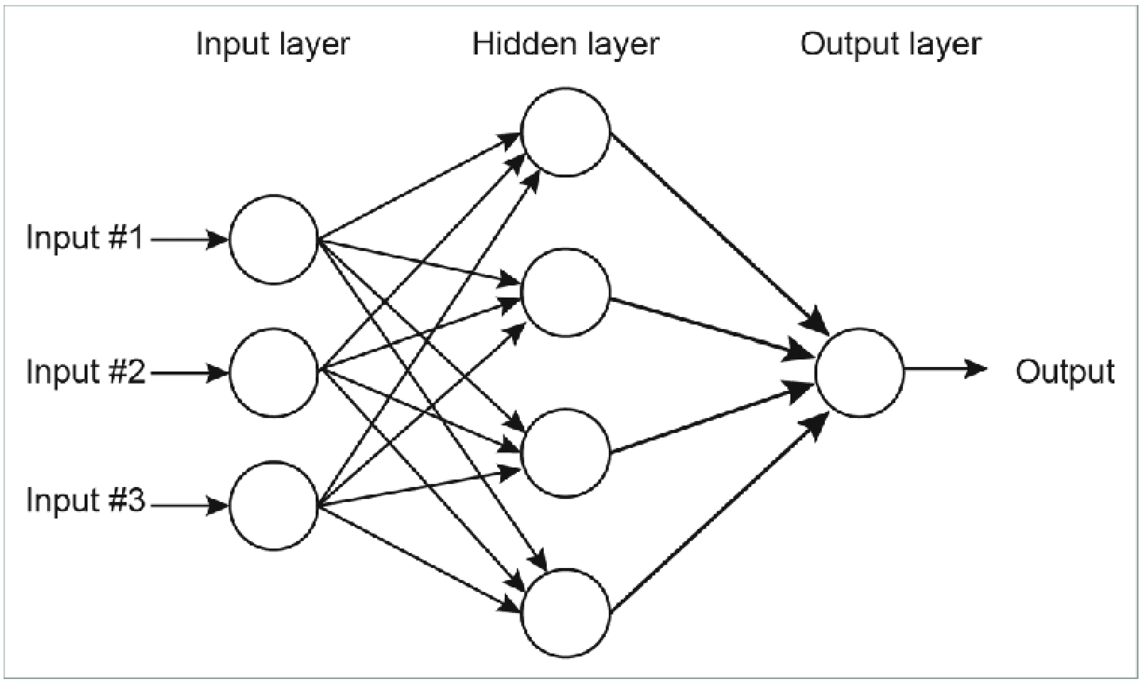


Figure 2.2: Example of structure of a typical 3 layer feed forward multilayer perceptron [Manning et al., 2013].

Convolutional Neural Network (CNN) is a type of neural network designed specifically for processing input data with a grid-like topology, such as an image (2 dimensional grid of pixels) [Goodfellow et al., 2016]. CNN's are composed of multiple convolutional and pooling layers, which extract features from the input data. Convolutional layers use filters to produce feature maps, that highlight certain locations or patterns of the input, such as edges, corners or lines, for example. These filters are adjusted during the training, thus allowing the CNN to learn how to extract features. Fig. 2.3 illustrates how a convolutional layer can be used to obtain the edges of something in an image.

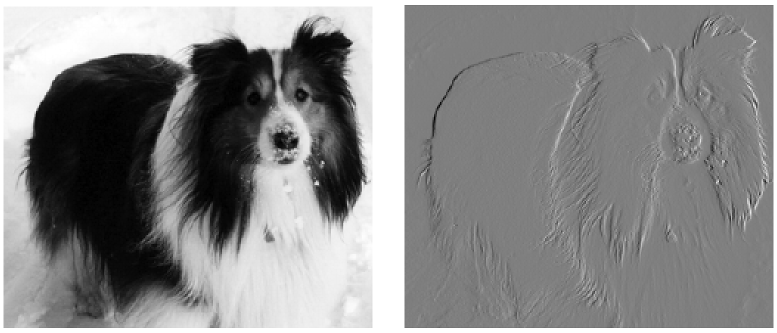


Figure 2.3: Convolution operation for edge detection in an image [Goodfellow et al., 2016].

Convolutional layers are usually followed by Pooling layers that down-sample the feature maps previously generated by aggregating values through a certain criterion (maximum, average or sum, for example). This pooling operation helps reduce the complexity of the feature maps, allowing for an easier processing and better generalization of the model for future data. A simple example is presented in Fig. 2.4, where an average pooling is applied.

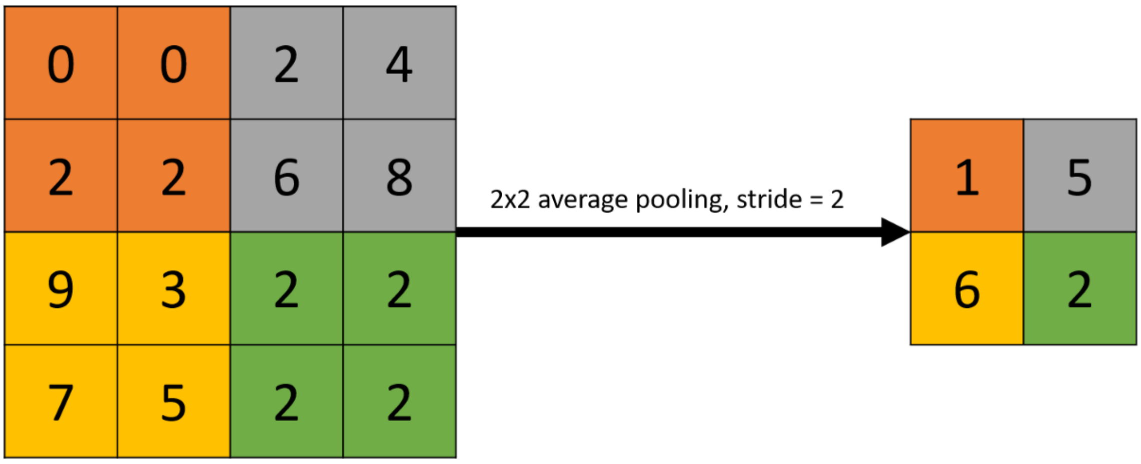


Figure 2.4: Average pooling operation example [Sanagapati, 2017].

These extracted features are then fed to a fully connected neural network that processes the features into the final output of the CNN. Fig. 2.5 shows an example of a CNN structure for number recognition, containing convolutional layers, pooling layers and a fully connected neural network used to make a prediction.

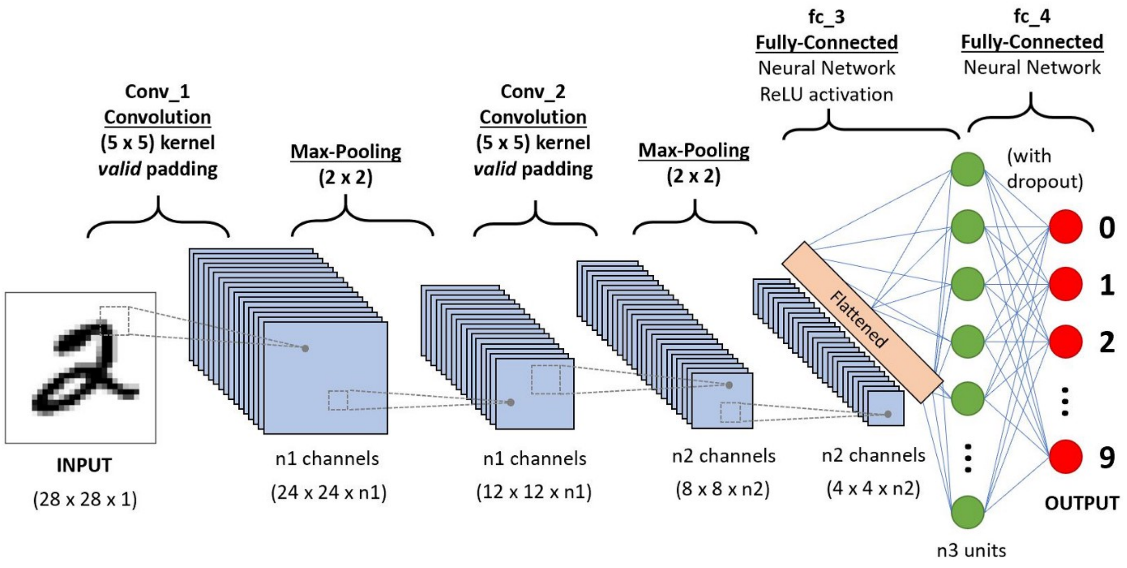


Figure 2.5: CNN example for number recognition [Saha, 2022].

By combining convolutional and pooling layers (and its corresponding parameters), different **CNN architectures** can be defined. These can vary according to the input data and the designated task to perform. Over the years, many architectures have been proposed for different purposes, which regularly serve as starting points for projects involving CNN's, since they have proven to be successful in the past. Some popular architectures include LeNet, Inception and

VGG, which have also different variations (e.g. VGG has VGG16 and VGG19). However, many more exist and are mentioned throughout this document.

The training process of a CNN usually takes significant amounts of time and computational resources, as well as the need for very large datasets, when done from scratch. To avoid this, one can use **Transfer Learning**, which basically consists of adapting a previously trained CNN to a new task or domain. Since the model was already trained, all that is needed is to fine-tune it, which reduces training time and resources, and can also help provide better results [Shin et al., 2016]. A common example is ImageNet trained models. Many CNN's of different architectures have been trained on ImageNet [Deng et al., 2009], a benchmark dataset for image classification and are used as the base for transfer learning.

The final model presented is the **Autoencoder (AE)**, a type of neural network designed to learn a reduced, compact, representation of the input data [An and Cho, 2015] (e.g. an image). This low-dimensional version is called the latent representation (or code). The AE is composed of two main elements: an encoder and a decoder. The encoder takes the input and maps it to its latent representation through a series of hidden layers. The decoder, on the other hand, reverses that process and reconstructs the input, also through several hidden layers. The main goal of an AE is to minimize the difference between the original and the reconstructed data, after reducing it to its latent representation. A simple schema of how an AE works is presented in Fig. 2.6.

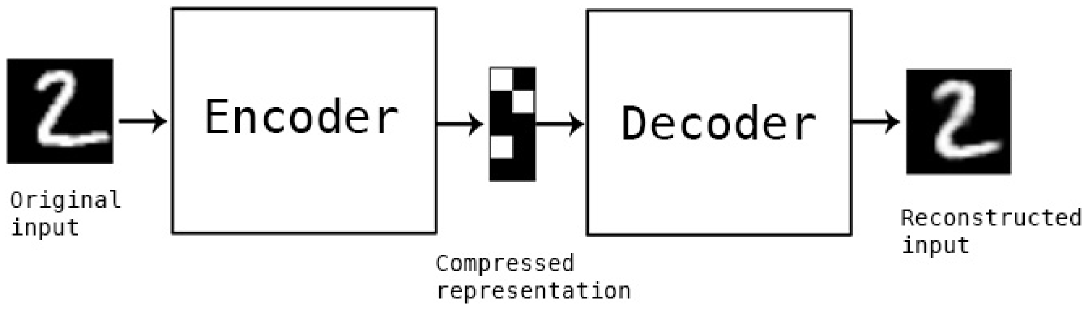


Figure 2.6: Autoencoder schema example [Chollet, 2016].

A more complex version of this algorithm is the **Variational Autoencoder (VAE)**, that differs from the AE by how it learns to represent the data compactly. While AE learns to compress the input, the VAE learns a continuous latent space that is able to capture the underlying structure of the data. By applying a probabilistic model, all that is needed are the estimated parameters for the distribution of the input. These parameters allow the reconstruction by sampling values from the corresponding distributions [An and Cho, 2015].

In this subsection, we presented some basic ML concepts as well as popular approaches used in the field, which are mentioned later in this section. As it may have been noticed, deep learning methods, especially CNN's, were the most emphasized, since those are the main approaches applied in the gathered literature. Nevertheless, only a brief overview was provided. For that reason, should the reader be interested to deeply explore these concepts, we suggest the following book [Goodfellow et al., 2016].

Lastly, other ML algorithms, such as K-Nearest Neighbors, Naive Bayes, Decision Trees, Random Forest or Logistic Regression, more classical approaches, are also mentioned throughout the rest of the section. However, since they are more used as baseline models to compare to novel ones and are not the focus of our research, they were not be presented in this subsection. However, similarly to deep learning methods, if the reader wishes to find out more about these techniques, or even other topics less discussed, such as Reinforcement Learning, we encourage to explore the following reference, since the a deep analysis of algorithms is out of this dissertation's scope [Mitchell, 1997].

2.2.2 Applications of ML in Data Visualization

ML can be used in visualization for many reasons and in many ways. As far as reasons go, Wu et al. [2021] identify three main goals in their survey: visualization generation, enhancement and analysis. These can be achieved in many ways. The survey identifies seven primary tasks with that apply ML for visualization:

Transformation
This operation converts the formats of visualizations. It can be used to transform visualization programs, another known visualization format other than the classic graphical image, into the other version. Programs consist of the description of the visualization, rather than simply an image, which can be easier to adapt.

Such an adaptation example is VisCode [Zhang et al., 2020], an approach aiming to embed information in visualization images, without distorting the original chart. This can help enter more information in the visualization without being seen, avoiding a loss in quality or appearance. It can be used to insert metadata, source code or for visualization retargeting. This tool has three main components: A Visual Importance Network (a CNN with a BASNet architecture) that analyzes the most important areas of the chart in order to avoid them when inserting information. An Encoder Network that inserts a message within the image, encoding it. This is a CNN acting as an Autoencoder. The Decoder Network, that reverts that process and recovers the encoded message. This is also a CNN (smaller) acting as an Autoencoder.

Another example is MobileVisFixer [Wu et al., 2020]. They use RL in order to train a model to adapt visualizations in their "regular" desktop/web dimensions to mobile friendly views, so as to be properly used in that context. They use RL since the goal is to mimic human behavior into automating the adapting process. To train this model, they apply heuristics for the learning policy.

Part of the transformation process of turning a graphical visualization into a program is the classification of the visualization received as input. That is a challenge explored by Deepchart [Tang et al., 2016]. They combine CNN's and Deep Belief Networks (DBN) in order to classify data charts of five types: pie charts, scatter charts, line charts, bar charts, and flow charts. The CNN are used to extract deep hidden features of the images and the DBN makes use of those to make the prediction of the type of chart.

Also regarding this problem is the contribution from Chagas et al. [2018], where they compare different CNN architectures (VGG19, Resnet-50, and Inception-V3) and compare these approaches against conventional classifiers (K-Nearest Neighbors, Naive Bayes, Random Forest, Support Vector Machine). The dataset used comprised of 10 classes of chart images. The results showed Resnet-50 and Inception-V3 performed best, achieving an accuracy of 77.76% and 76.77% respectively, much better than the other approaches, barely surpassing 45%.

Transformation might also need pre-processing, such as decomposition of multipart figures. Tsutsui and Crandall [2017] presents an automated approach for this task, so that each chart can be analyzed individually. They try to improve the existing methods, based on manually defined rules, by using YOLO V2 CNN architecture to identify bounding boxes needed to separate the sub-figures. They also apply Transfer Learning by using ImageNet's set of weights for the neural network.

Comparison
Comparing two visualizations through a certain criterion, such as, for example, similarity. This can be also be considered part of visualization analysis and can be useful when looking into a collection of examples.

One example of this task is ScatterNet [Ma et al., 2020]. ScatterNet attempts to represent human perception in analyzing scatter plots. To do this, they use CNN's with a triplet network structure (and Krizhevsky et al. [2017]'s architecture) to assess the similarity (or dissimilarity) of scatter plots. For human evaluation, they conducted user studies, and concluded that ScatterNet's learned features capture the human perception of scatter plots effectively.

Querying
Retrieving relevant visualizations from a collection according to the user's needs. This can be very useful for search engines, for example, when searching for visualizations regarding a specific topic.

An example of this task is present in VizWiki [Lin et al., 2018], a system that scans the internet, namely the Wikipedia database (Wikimedia Commons) for visualizations to enrich news articles. First, they separate visualization images from non-visualization ones by using a CNN (InceptionV3 with ImageNet weights) and a Support Vector Machine (SVM). After that, the visualizations are analyzed and ranked in terms of usefulness. This analysis combines textual and graphical features (that ended up not being so useful) which are used by a Ranking SVM (RankSVM) to rank.

Reasoning
Interpreting visualizations in order to extract high-level information, as in more than just assessing the type of chart of visual encoding used. The goal of this task is for the machine to read visualizations like a human would.

A good example to illustrate this task is the work of Poco and Heer [2017]. They present a pipeline that allows the automatic recovery of visual encodings from a chart image. First, it identifies textual elements in the chart, using Darknet (an open source CNN framework). Then, it recovers the text using optical character recognition. After that, they classify the role of each element. Next, they identify the graphical mark used, using a CNN with AlexNet's architecture. Finally, using this information, it is possible to recover what variables are encoded and how they are encoded in the chart.

With Chart decoder [Dai et al., 2018] the authors go even deeper in the reasoning task. They present a system that decodes visual features in order to generate textual and numeric information from a chart. Their approach consists of four pipelines: chart classification with CNN's (Alexnet, VGG16, GoogLeNet and ResNet architectures were tested, with VGG16 being a close best), textual component extraction (through identification, text recovery and role classification, like in the previous example), graphical component extraction by detecting bars in bar charts and assessing their values, and chart data recovery by combining the information gathered until this point. The final two steps were only implemented for bar charts, though. This example could also be included in the Transformation task, since it allows for a change in the visualization format, in a way.

Other examples [Kafle et al., 2018; Kahou et al., 2017] explore this task by presenting datasets of visualizations and question-answer pairs for each chart in order to study aspects of machine understanding of data visualization, along with model performance evaluation on the dataset. Kafle et al. [2018] present two novel approaches using Neural Networks that generate strings to answer the questions, while Kahou et al. [2017] apply a text-only Long Short-Term Memory (LSTM) model, that same model combined with CNN's of different architectures and a Relational Network (a network model that analyzes the relation between different elements) to provide a binary response to the questions: yes or no.

Recommendation
Even though the name is pretty straightforward, this task helps the user by suggesting or advising them in terms of visualization choice.

An illustrative example on how it can be used is VizML [Hu et al., 2018]. They present an ML-based approach for recommendation that learns visualization design choices (such as selecting visualization type or choosing which axis do encode) from a large dataset. The model used is a fully connected, feed-forward neural network that receives manually chosen features. They compared the performance of this model against baseline models (Naive Bayes, Logistic Regression, K-Nearest Neighbors, Random Forest) and against other known ML-based approaches. They also showed the performance of their model is comparable to human performance, through a crowdsourced test set.

Mining
Gathering insights from a visualization database. This task differs from querying since it does not return a set of visualizations from a collection, but rather patterns of data or of visualizations, that can result in cluster or statistical analysis, for example.

VizByWiki [Lin et al., 2018] can also be included in this task since the search process also returns statistics regarding percentages of analyzed visualizations and their level of relevance (useful, somewhat useful of not useful).

As we can see, many applications can be identified for ML. It is worth reminding that all these tasks are all part of the same domain, data visualization. This just goes the show how deep ML techniques can be explored within a single field, for a great profit.

However, one particular task was not mentioned in the listing throughout this subsection. That task is assessment, which is the main focus of this dissertation and discussed in the following subsection.

2.2.3 ML for Evaluation

The assessment task, or evaluation, consists of assessing a visualization according to a certain criterion, either in an absolute way, by calculating a score, or in a relative way, by ranking multiple visualizations. However, an absolute score can be more valuable since it can also be used for ranking but can, in addition, be used for optimization of algorithms, for example, for visualization generation, recommendation or enhancement. Even though a wide range of criteria can be identified, such as similarity between visualizations as presented in ScatterNet [Ma et al., 2020] in the previous subsection, we believe the most useful one might be a quality assessment.

However, the definition of the quality criterion can be highly subjective and can also be measured in many ways. For example, VizByWiki [Lin et al., 2018], also previously discussed, makes a ranking of visualizations according to their usefulness to the article being written, which, in a way, represents its quality.

A different criterion that may make more sense to mention related to quality is aesthetics. In other words, it measures how good the figure looks, which can certainly add value to the visualization. Fu et al. [2019] make use of this criterion in their research. They automate the process of visualization assessment through ML techniques. Using a semi-supervised learning approach, they apply a Variational Autoencoder (unsupervised learning) to extract effective features from the visualizations, followed by the model that calculates a score based on the features and on the corresponding assessment criteria. More specifically, the criteria used are aesthetics and memorability. For each of these, a Support Vector Regressor (SVR) is trained (supervised learning) to estimate a score for a visualization fed to the model, thus allowing for an assessment.

Aesthetics is also analyzed by Haleem et al. [2019], although with a different, more objective goal than the previous approach. They attempt to optimize the process of evaluating dense graphs in terms of readability, by assessing the image layout of the graph. For this, they present a new CNN architecture that takes into account multiple metrics like edge crossing or node spreading, for example. Inspired by AlexNet and VGG, their architecture consists of 6 convolutional layers that feed a two-layered, fully connected Neural Network that makes a regression estimation of each metric designed.

Readability is certainly an important criterion of visualizations, that can reflect on other aspects such as accuracy or completion time of visualization related tasks.

Giovannangeli et al. [2020] attempt to automate the evaluation process according to this aspect. The goal is to use it as supporting evidence for standard user assessment, since existing approaches cannot yet reflect human capabilities but can still be useful for comparing different visualization techniques. Therefore, they assess the correlation between the performance of users and CNN's through experiments that compare certain tasks carried out with graphs of two types: node-link diagrams and adjacency matrix diagrams. For each task and type of graph, they train two CNN models, with LeNet and VGG16 architectures, and compare the results through several metrics. They conclude that user's and machine's performances could be correlated for some tasks, but more work is needed to generalize this assumption.

Just as readability is related to accuracy, accuracy itself is related to graphical perception, as was explained in subsection 2.1.1. DeepEye [Luo et al., 2018] explores this as a metric for evaluation. DeepEye is a system that supports automatic data visualization by tackling three problems: i) deciding if a visualization is good or bad, ii) choosing which visualization is better between two examples and iii) retrieving the *n* best visualizations in a dataset. This evaluation is done in terms of human graphical perception, which can be hard to capture and compare since there is no ground truth. They compare knowledge-based approaches with ML-based ones, using pre-defined features such as number of distinct values in a column, data types or correlation of different columns. Through these features they train a binary classifier for problem i), using Decision Trees as the highest performing one. For problems ii) and iii) they apply the LambdaMART algorithm to find a function that outputs a score for each example, allowing for a ranking of visualizations. For problem iii) they also proposed a solution of partially ordering visualizations to better rank them, which is relevant and presents better results since different visualization types might not be directly comparable.

Another approach for this metric is explored by Haehn et al. [2018], in which they measure the graphical perception capabilities of CNN's by reproducing Cleveland and McGill's experiments [Cleveland and McGill, 1984]. Only five of those are conducted, aiming to predict values in the visualizations from the visual marks presented in each case, through logistic regression. Four models are tested and compared for these tasks. As a baseline model, a simple MLP is used, with no feature extraction. Then, three CNN architectures are used, LeNet, VGG19 and Xception, with and without transfer learning by importing ImageNet's weights. They also compared these models against human performance, concluding the CNN's are not, as of yet, the most effective way of capturing human graphical perception, since they show better performance in certain tasks but worse in others.

2.3 Summary

Throughout this chapter we carried out an extensive research regarding data visualization. We started by presenting the principles in the basis of information design in section 2.1, namely regarding visual encodings and the way they are used to transmit information to the viewer. This is an vital aspect to look into since the visual variables are the main element of a visualization, thus having a great impact.

Next, we explored the methods used to assess their efficiency. In subsection 2.1.2 more classical methods are discussed, such as user studies of different types conducted to draw conclusions, while in subsection 2.1.3 more innovative ones, that automate the process by using computers for that analysis, have been presented and discussed. These can be done in different ways, summing up to knowledge-based approaches and statistical-based ones. These methods, although successful to an extent, have their weaknesses: user studies require human subjects to engage in tests, knowledge-based tools require the mapping of theoretical findings into computer algorithms and statistical-based methods, while useful, do not seem to target the same problem as us.

Then, we introduced ML techniques in subsection 2.2.1 and reviewed cases of how they could be applied for visualization-related tasks. Subsection 2.2.2 covers a range of scenarios, such as visualization transformation, comparison, querying, reasoning, recommendation and mining, providing us with a valuable list of ML approaches for visualization-related tasks, which can be useful when we need to select which ones to use.

The evaluation task is more emphasized in subsection 2.2.3, given that it is the focus of our research. Several strategies were presented, including the ones we believe better align with our goals, such as readability, accuracy and visual perception. Being able to capture human graphical perception with machines would be perfect for evaluation, since we could replicate those capabilities in order to properly evaluate visualizations, as if they were humans. However, it seems we are not at that stage yet. Moreover, we can see that few ML approaches have been presented that take into account the mentioned metrics, and even fewer (or none) that generalize that type of evaluation. We have seen examples of assessment for graphs only [Giovannangeli et al., 2020; Haleem et al., 2019], an approach that simply classifies visualizations as good or bad [Luo et al., 2018] and an approach that merely replicates simple experiences [Haehn et al., 2018]. Nevertheless, these findings are a very important basis for our work: we are aware of the progresses made so far, their approaches and their limitations, which serve both as inspiration and support. What we look for though, is a more generalized approach that can help better compare different types of visualization. A more thorough explanation, as well as the goals to achieve such an objective, are presented in the following chapter.

Chapter 3

Work Plan

In this chapter, we discuss how the work plan defined at the beginning was actually carried out throughout the past months. We start by summarizing the tasks previously suggested and how they were later adapted to better fit the requirements of our goals. However, we will not get into the full details of the tasks in this chapter since they are deeply explored later in the document. Additionally, we compare the time window of the developed tasks against the original plan.

3.1 Tasks

As it was mentioned in the introduction, the main goal of this research was to achieve a method of evaluating data visualizations in terms of visual perception. In other words, we want to be able to assess a certain model on how easily or accurately the encoded information can be perceived. Inspired by the experiments conducted by Haehn et al. [2018], our strategy consists of applying Machine Learning (ML) techniques that attempt to extract the values embedded in the visualization and compare them to the original values used to create the visualization. These values, called ground truth, can be used to calculate the estimation error of the predictive model. Based on that error, essentially a scoring metric, it is possible to evaluate the performance of the respective model. On top of that, it allows for a comparison between different visualization models, which is one of the reasons for this work. However, in order to apply that strategy and achieve our goals, multiple steps are needed, which were discussed at the start of this work. In Fig. 3.1, we show the Gantt diagram corresponding to the work plan originally defined, featuring all the tasks and their associated time frame.

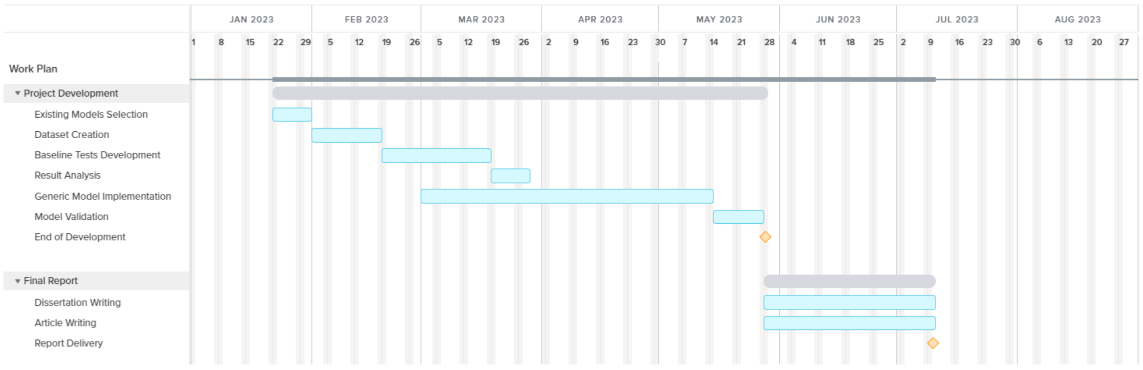


Figure 3.1: Gantt diagram of the scheduled project development.

The first task defined (Select Existing Models) essentially meant looking into the gathered literature on the topic of ML for Visualization (ML4VIS) in order to figure out which ML models would be the most suitable to test for our method of evaluation. The second one (Create Dataset) was mostly related to increasing the variety of data visualizations available to test our models, which meant deciding which chart types to use. These two tasks were aggregated into a single one (**Experiments' Setup Definition**) where, in addition to these two tasks, it was decided how to conduct the experiments and what information to save for each one.

The third task stipulated (Baseline Tests Development) consisted of developing the necessary code to conduct the base experiments and running them. This task was split into two to separate the development and the experiments. The development ended up being renamed (**Experiments' Setup Development**) since the code is not exclusively meant for the base experiments and is later used for other tests, while the experiments essentially kept the name (**Base Experiments**).

We added a task specifically for the development of the API for integration (**API Development** since it is unrelated to the experiments but necessary work for the project.

Regarding the fourth task (**Results Analysis**), it remained unchanged, apart from its duration, since it is fundamental in a scenario of research and investigation such as this one. Despite not being the last task presented in this section, it is the final task of the development stage, due to it being initiated after the base experiments but carrying on after the variations experiments, since those results should obviously be analyzed too.

The fifth task (Generic Model Implementation) was somewhat discarded from the original plan. Part of that task had already been developed at the beginning of the semester, while another part was integrated into the experiment setup, where the automation of training models was introduced. Additionally, the name of the task was considered misleading as to what it actually meant, contributing to its removal.

The final task of the development (Model Validation) had the goal of testing our best-suiting model further in order to validate it for deployment. While the principle remained, its name was changed, as well as its purpose, in a way. Now related to visualization variations used to test the capabilities and limitations of our most suitable model, this task was also split into two, the first one being the setup necessary for these new experiments (**Variations Setup**) and the second one running the actual experiments (**Variations Experiments**).

Regarding the writing of the dissertation and article, a paper was written but has not yet been published. This is planned to happen as soon as the target conferences start in 2023, which correspond to IEEE VIS, EuroVis, and PacificVis, regarding visualization conferences, and IJCNN, ECML and ICML, regarding ML ones.

3.2 Schedule

Regarding the scheduling of the work, we present an updated Gantt diagram in Fig. 3.2 which takes into account not only the changes to the designated tasks but also their time frame, indicating their true start and end. Despite these changes, the waterfall methodology [Royce, 1987] was used anyway since the sequential nature of the tasks remained, justifying this choice.

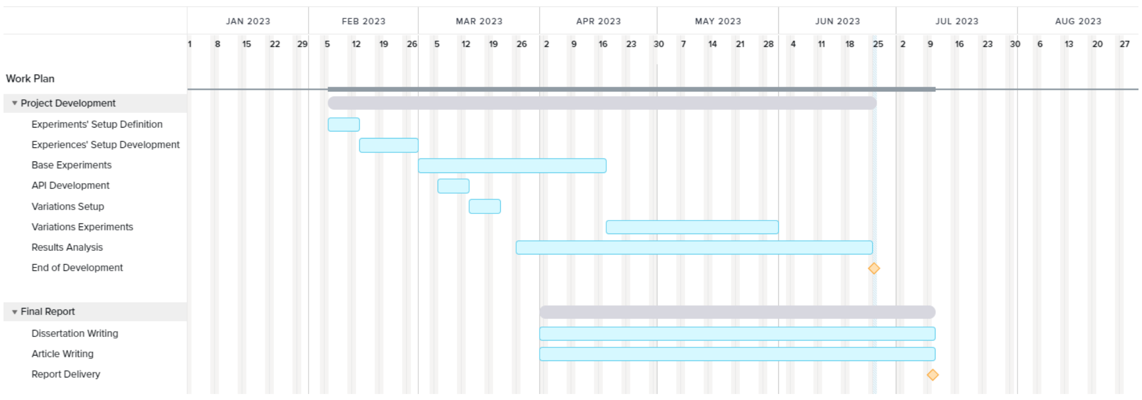


Figure 3.2: Gantt diagram of the actual project development.

As we can see, several tasks had a noticeable difference in terms of duration. The tasks regarding experiments (Base Experiments and Variations Experiments) took longer than expected due to three main reasons: hardware limitations, which forced us to adapt our code so as to be more optimized in terms of memory usage, unpredictable duration for the training of the models and small mistakes in the configuration of the experiments which on occasion led to their repetition.

The analysis of the obtained results also appeared to be longer than initially planned. Although marked as a single continuous task, it consisted of several moments as new results were being produced by experiments, thus appearing longer than it actually was. In spite of that, the analysis took in fact longer than expected since each time new information was discovered from the results, new ideas of how to analyze that data appeared, leading to a chain of sub-tasks which took a significant (yet justified) amount of time.

Similarly, the writing tasks (dissertation and article) also have a longer duration since they consisted of multiple moments throughout the course of the semester, especially while the experiments were running, to avoid dead times.

Despite the changes to the work plan, the updated tasks maintain the same overall strategy and the same end goal: assessing the method of evaluation data visualizations in terms of graphical perception through the use of ML models. By conducting baseline experiments to assert the most suitable model, further testing it to understand its capabilities and limitations and analyze the obtained results, we are able to reach that goal. The definition of the work plan at the start of the project was fundamental in order to follow a logical sequence of steps and optimize time usage. The tasks mentioned throughout this chapter are described in full detail in the next chapters.

Chapter 4

Development

In this chapter, we describe all the work developed for the research on the topic of this dissertation. The development had several stages, each with different contributions towards the end goal. The necessary code for each task was written in Python, the main programming language used throughout the development.

The chapter is composed of five sections, organized as follows: firstly, we explain the approach used for our method for evaluating Visualization Models. Then, we present the pipeline necessary for training the ML models. Next, we go over the experiments we conducted to test different Convolutional Neural Network (CNN) architectures, along with different types of visualization models. After those experiments, we also test some variations in order to understand the limitations of our proposed model. Finally, we describe the integration of our solution through an Application Programming Interface (API) and finish with a small summary of what was presented.

4.1 Approach

In this section, we describe our approach towards the development of a solution for a method of evaluating data visualization models. This approach follows the logic presented in the introduction, where the concept of graphical perception is used as the criterion of evaluation. However, instead of conducting studies with human users, we make use of ML models to serve as the eyes and brain in order to provide answers. More specifically, we use CNN's since the inputs are images of data charts and, as it was presented in subsection 2.2.1, these are suitable for image processing.

Our models are composed of two main elements. The first one, **feature generator**, is what we named the CNN architecture that takes an image as an input and extracts its features, as it was explained in subsection 2.2.1. The feature generators used, which are discussed later in subsection 4.3.1, have different characteristics but the goal is the same: generate the necessary features to properly analyze the image, which serves as input for the second element. This one, with the exact same structure for every instance of our model, is a Multi-Layer Perceptron

(MLP) consisting of an input layer which receives the features from the feature generator, a hidden layer with 256 neurons and an output layer with the number of neurons according to the number of values to regress (for example, a model trained to process a visualization with five values must end with five neurons in its final layer). To optimize this network we use an algorithm called Stochastic Gradient Descent (SGD) with a learning rate of 0.0001, a Nesterov momentum with a value of 0.9 and a batch size of 32 (32 samples processed at each time). This optimization occurs throughout 1000 epochs, which means the training set is processed a thousand times, as long as the network keeps improving.

The following diagram in Fig. 4.1 illustrates the structure of the model including the two elements previously mentioned, as well as an example of how the input image is processed into a set of labels.

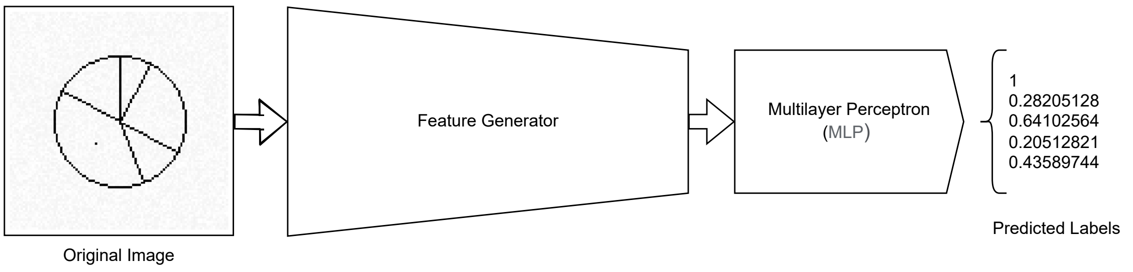


Figure 4.1: Model structure with an example - An image representing a chart is inputted into the feature generator, which extracts the relevant features. Then, those features serve as input for the MLP, which processes them and outputs the estimated values represented in the original image.

The input consists of an image of 100 by 100 pixels, with each one having a value between 0 and 1, where 0 is white and 1 black, therefore using only tones of gray. Essentially, the image simply consists of a matrix of numbers, with 100 rows and 100 columns. This is further explained later in subsection 4.2.2. Regarding the output, the number of values to regress corresponds to the number of variables represented in the chart. However, how are the labels shown in the figure obtained from the values used to generate the image? Well, given the original values, (17, 39, 11, 25, 8), we take the largest one, 39, and divide all the numbers by this maximum value, obtaining the following set of labels:

$$\begin{matrix} 0.43589744 \\ 1 \\ 0.28205128 \\ 0.64102564 \\ 0.20512821 \end{matrix}$$

This means that the labels simply reflect the proportion of the values in relation to the maximum, just like the example provided in the introduction. The only difference is that previously, for ease of demonstration, we multiplied all values by 100 to simulate percentages and rounded the values to integers. One more step is needed though. We need to roll the labels so that the first one corresponds to the largest value, maintaining the order. By doing that, we obtain the labels according to the example, which is the logic used in the generation of the datasets, further explained in subsection 4.2.2.

By using CNN's to make a regression of the values presented in visualizations, we can calculate how the error associated with the "reading" of the chart, as it was explained in the introduction. This error allows the comparison of different visualization types (or variations) by providing us with an absolute value for the performance.

In order to make use of CNN's for this task, we must first train them for that purpose, just like a person first needs to learn how to perform a new skill, as simple as it may be. But even though the logic is similar, the learning process is obviously much different between machines and humans. Not only that, while humans easily learn and retain multiple skills simultaneously, the ML models used here are fairly simple and can only be trained for a single task. This means that in order to test different visualization techniques and their respective variations, multiple models must be trained. For that reason, it is important to establish all the steps necessary to train and test a model and implement them in such a way that is easily repeatable, which is discussed in the following section.

4.2 Pipeline for Training

Due to the need to train multiple models in a repetitive way, many of them with different configurations, we designed and implemented a pipeline for training, which is described in this section. This consists of a series of tasks necessary for the full process of training and testing a ML model, as well as making it available for future use. These steps are presented in Fig. 4.2 and then described in the following subsections.

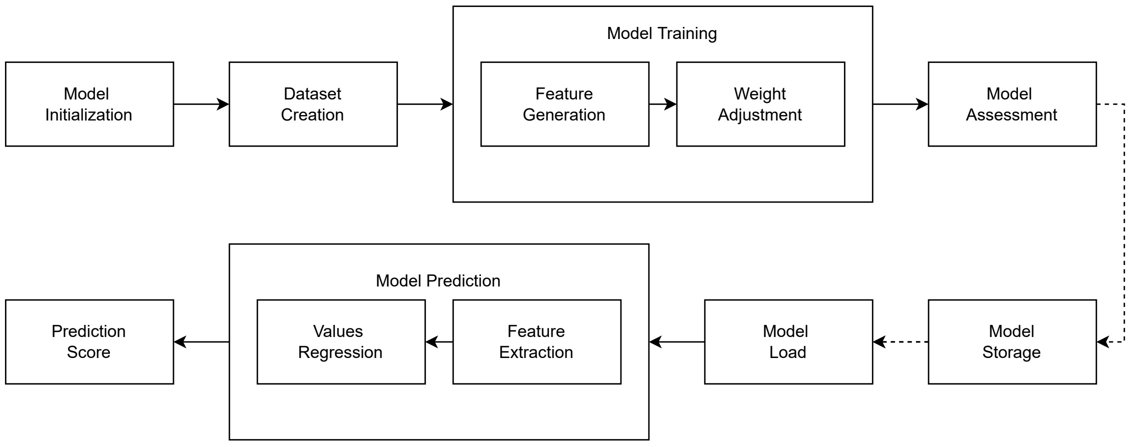


Figure 4.2: Sequence diagram of the developed pipeline process. Dashed lines indicate optional tasks.

4.2.1 Model Initialization

Firstly, the model is initiated with all necessary parameters defined: a name for the model, the CNN architecture to use, if the training should be done from scratch or from ImageNet weights (see subsection 2.2.1 for further explanation) and the number of variables to consider in the visualizations. This last parameter is required for the final layer of the neural network previously mentioned: we must specify the number of values to output in order to initialize it. Regarding the remaining parameters, more details are provided in the following sections.

4.2.2 Dataset Creation

In order for the CNN's to be trained, a set of examples is necessary. In this case, these correspond to charts with their represented values, just like the example described in the introduction. Despite being a seemingly simple task, it is comprised of many steps. It is important to mention that the process described in this subsection is heavily inspired by the work developed by Haehn et al. [2018] since some of their code is used here.

First off, *n* values are randomly generated. This number varies according to the model configuration. To facilitate the explanation, let us assume *n* as 5. This means that 5 values are generated, which will be used to construct the visualization. However, this generation obeys a set of rules to ensure a fair training:

- A minimum value;
- A maximum value;
- A minimum difference between values (e.g. with a minimum difference of 3, values such as 10 and 12 could not appear in the same set);
- A number for the total sum of the values (e.g. with a total sum of 100, the sum of all values should match that number).

For 5 variables, the generation takes into account a minimum of 3, a maximum of 39, a minimum difference of 3, and a total sum of 100. As previously mentioned, these properties vary slightly according to the number of variables desired for generation, but the principle is the same.

Once we have the values, the next step is the generation of the image. This is done by initializing a matrix of 100 by 100 values with zeros, which corresponds to a plain white canvas, where the charts are drawn. For this, we use a Python package called sickit-image which allows the drawing of lines, polygons, circumferences and other elements, by providing coordinates and dimensions. These are calculated through the use of the variables previously generated, therefore representing them on the image.

Next, we mark the largest element. Why is this necessary? If we were to show these images to a human being and ask them to perform the regression task, we

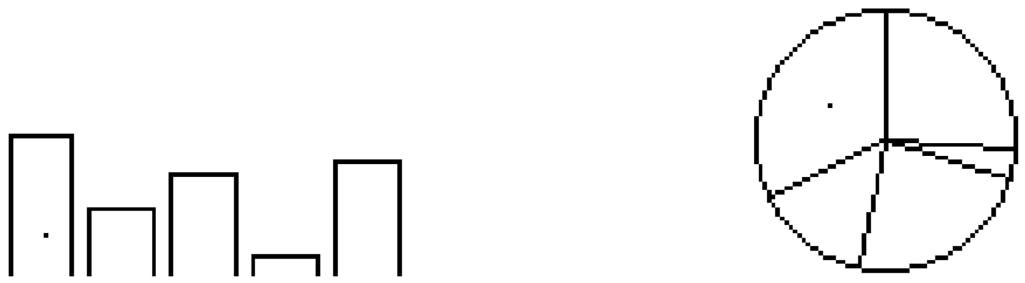

(a) Bar chart (b) Pie chart

Figure 4.3: Examples of images generated for the datasets. The largest element in each chart is marked with a single dot.

would simply point to the largest value in order to guide them. Since we cannot do that with a machine, we overcome that issue it by marking the largest variable visually. This is shown in Fig. 4.3, where each chart has a single dot identifying the largest value. If this was not done, it would mean that in addition to estimating the ratio of the represented values, the models would also need to identify the largest value, which is an extra task that can influence the regression results and is not what we intend to test. This hypothesis was tested and its results are later presented in chapter 5.

Then, we add random noise to the background. This just means that instead of having a pure white background, there are slight, random variations in the pixels, within different tones of gray. Fig. 4.4 shows such an example, where we notice the random noise in the background of the chart. Digitally speaking, instead of having a matrix simply consisting of 1's and 0's, each pixel has a random variation within the range of -0.025 to 0.025. This is done for the model to focus on the relevant features of the images, rather than on very specific, less important patterns, which helps prevent over-fitting. Finally, after all the images have been generated, their pixels' values are normalized and scaled to the range of -0.5 to 0.5, before being ready to be fed into the models for training and testing.

Regarding the labels for the images, further processing is needed for them to be used in training and testing. Going back, we mentioned that n values, let us say 5, are generated according to a set of rules. However, the task at hand is not to make a regression of the values represented in the chart, but of their proportion. In other words, we do not care about the specific values but only their ratio. As we previously mentioned, the ratio should be calculated in relation to the largest value. This means that in order to obtain these ratios based on the original values, we need to divide them all by the largest one, as it was explained in section 4.1. After that, the labels are rolled so that the first value corresponds to the maximum, which will be 1 (since the largest value is divided by itself),

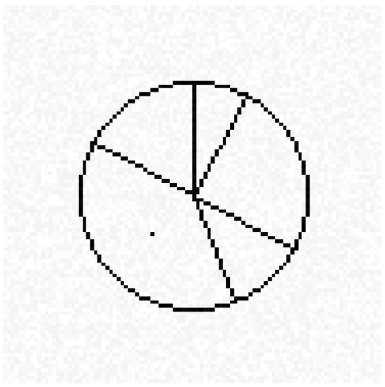


Figure 4.4: Example of a chart generated with noise in the background.

maintaining the relative order for the remaining labels. Let us look at an example. The values 8, 17, 39, 11, and 25 are randomly generated. Then, since 39 is the largest value present, we divide them all by 39, obtaining 0.21, 0.44, 1, 0.28, and 0.64 (approximately). By successively rolling the numbers, we are able to move the number 1 to the first position while maintaining the order, as it is shown below:

$$\begin{array}{ccccc} 0.21 & 0.44 & 1 & 0.28 & 0.64 \\ 0.64 & 0.21 & 0.44 & 1 & 0.28 \\ 0.28 & 0.64 & 0.21 & 0.44 & 1 \\ 1 & 0.28 & 0.64 & 0.21 & 0.44 \end{array}$$

This process is done for every single set of labels in order to be used for training. It is necessary to do so because this way the order in which the model outputs the variables is clearly defined. The last step for the labels is, once again, a normalization in the range of 0 and 1.

Fig. 4.5 summarizes these steps into a diagram. First, we generate a set of random values, which are used for two purposes. One of them is to produce a chart representing the values, in which the largest element is marked and random noise is added to introduce variation. The other purpose is to create the labels. For this, we divide all the values by their largest and then roll the labels so that the largest one is at the first position.

4.2.3 Model Training

The training of the model is slightly different according to a configuration defined in the initialization stage. If the model is set to be trained from scratch, both the feature generator and the MLP are trained. The former adjusts the way it extracts features, while the latter learns how to process these features in order to calculate the values, by adjusting the weights in the neural network, as it was explained in subsection 2.2.1.

On the other hand, if the model should make use of transfer learning, by applying ImageNet weights to the feature generator, only the MLP is trained, fine-tuning

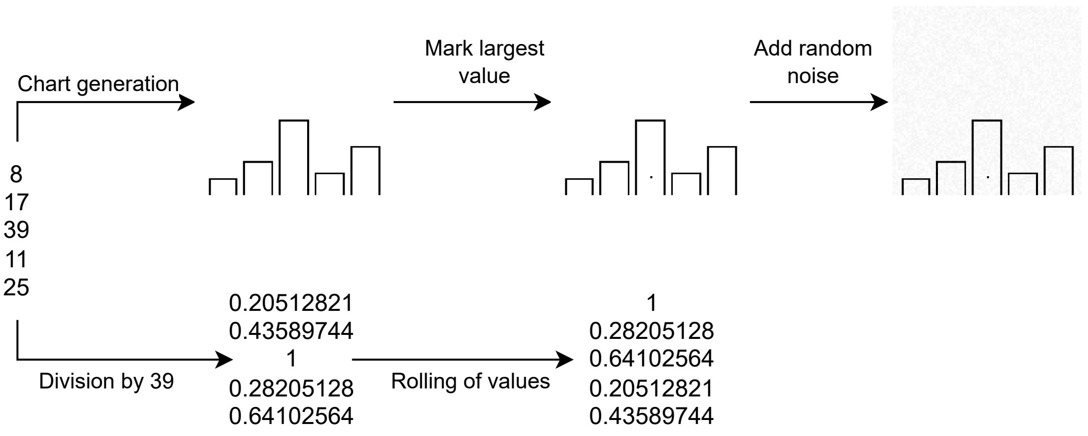


Figure 4.5: Diagram for the process of data generation.

the pre-trained convolutional network. This means that there are fewer parameters to adjust, compared to training from scratch, resulting in a faster training time. However, it leads to worse results since we are only adapting a previously trained network to a new task instead of solely training it for this challenge.

4.2.4 Model Assessment

The next step is the model assessment, in which the test set is used to assess the model's performance. This step is important to understand how well the model learned how to perform the task, which in this case corresponds to extracting the values represented in visualizations.

The way this is done is simply by having the trained model predict the labels presented in the test set of the dataset. Then, using the predicted labels and the true ones, it is possible to calculate the performance metrics based on the estimation error. These metrics are what allow the comparison between different visualization types, according to the model's performance on them.

4.2.5 Model Storage and Load

At this step, since the model is trained, it is ready to be used and thus, can be saved (**model storage**) to be utilized later, in a different setting, if needed. If that is the case, it is then loaded (**model load**) and ready to make predictions based on more data. It is especially important once the pipeline is integrated since a trained model will have to be used multiple times to assess new visualization examples, but it is also useful for testing/experimenting purposes, to repeat or conduct more tests on a particular model.

4.2.6 Model Prediction

This stage of the pipeline can be considered the main one. This is because the end goal is to have a method to assess a new example of a visualization and this is achieved through the prediction (and its error) for a new input.

For this task, two steps are also needed. Facing a new visualization image, its features are first extracted (*feature extraction*) using the feature generator previously mentioned, and then based on those, the MLP makes the regression (*values regression*), outputting the estimated values.

4.2.7 Prediction Score

The final part of the pipeline consists of, once again, calculating scoring metrics associated to the predictions made by the model. This is the way to transform the estimated values into a metric that allows for a comparison between different options.

4.3 Base Experiments

Throughout the previous section, we described the several steps included in the pipeline for training, some of which require some parameter definition, namely the CNN architecture to use, or whether or not to use transfer learning for the training of models. Therefore, we must run experiments to try and assess the best options for our evaluation method, as well as to attempt to figure out if it is viable to use. This section describes that process, which, in summary, compares the different options for the ML model to use in our evaluation, across a range of selected visualizations. We named these "base experiments" due to being the first ones and serving as a foundation for the research.

4.3.1 CNN Selection

The first, and most obvious question, is simply "which is the most suitable model for this task?". Well, for starters we know from domain knowledge mentioned in subsection 2.2.1 that CNN's have proven themselves to be highly effective with image processing, which is the present case. On top of that, from our analysis of the state of the art on ML4VIS, we also have some suggestions in terms of what architectures to use for our domain specifically, along with some baseline results for comparison [Haehn et al., 2018].

Therefore, we decided the most suitable CNN architectures to test for this research were VGG19 [Simonyan and Zisserman, 2015], XCEPTION [Chollet, 2017] and ResNet50 [He et al., 2015], since they have previously been used in tasks related to data visualization and proven successful. Obviously, more options could also have been tested, but given time limitations, these were the prioritized ones.

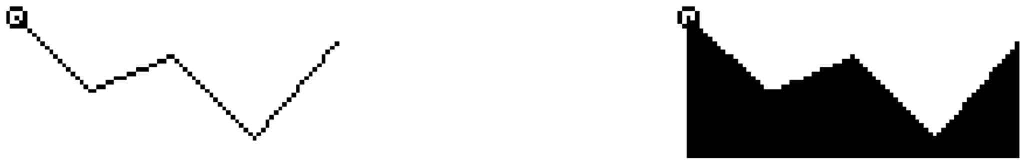
Another question important to test is the use of transfer learning. The available implementations of these architectures allow the use of a previously trained version on ImageNet, a widely known dataset of images. This adds a binary parameter to each of the mentioned architectures, turning the three options into six: each one of VGG19, XCEPTION and ResNet50 with and without the use of ImageNet weights. Unfortunately though, due to hardware limitations, ResNet50 with the use of transfer learning could not be properly tested, finalizing the number of possible CNN's to use to five. In order to integrate these architectures into our pipeline, we made use of their implementation provided by the ML package Tensorflow for Python. Should the reader wish to find out more about the structure of these CNN's, we invite them to check the references cited in the previous paragraph.

4.3.2 Visualization Selection

After defining which CNN's to test, the next question relies on which visualization techniques we test them on. This way, we also have the opportunity to simultaneously test the performance for different visualization models and possibly draw some initial conclusions. It is important to mention that for this set of base experiments, all visualizations used are generated from five values, which means they only represent five variables, as we will see in the examples shown in this subsection.

The first two available options are the **bar chart** and the **pie chart**, which need no introduction as they are two of the most commonly used visualization techniques. Examples of these are presented in Fig. 4.3. They are particularly important because since they are already used by Haehn et al. [2018] in their experiments, we have some baseline results to compare our own for confirmation. It is worth pointing out that for pie charts, the layout of the values starts at the top (or, to be clear, at 12 o'clock) and progresses counter-clockwise. The question of establishing that initial position as opposed to randomly choosing one was tested and its results are later presented in chapter 5.

However, we want to expand further to other visualizations, especially those suitable for different data types. For instance, bar and pie charts are mostly used for categorical data, where the values are assigned to categories and have no order or numerical meaning. On the other hand, time series, a different type of data, represent a sequence of samples observed over a period of time, so their representation should highlight the evolution between sequential samples. For that reason, we added the **line chart** to our set of visualizations, visible on the left of Fig. 4.6. This type of visualization also makes use of the position to represent the values, but unlike the bar chart, it connects adjacent points with a straight line. Additionally, we wanted to evaluate the effect of the area under the line being filled with color, so we also added that version (**line chart filled**) to the set, also visible in Fig. 4.6 on the right side. In this case, to identify the largest value, we mark it with a small circumference.



(a) Line chart (b) Line chart filled

Figure 4.6: Examples of a line chart and line chart filled. The largest element in each chart is marked with a small circumference.

Finally, we also propose a model that represents the values through the size of rectangles, whose area varies horizontally. We call this one **area chart**, and an example is shown in Fig. 4.7. This technique was mainly chosen for being a poor representation, since the use of area is the least accurate perceptual task according to Cleveland and McGill [1984], among those presented here. Just like the first two presented visualizations, here the largest element is also marked with a dot.

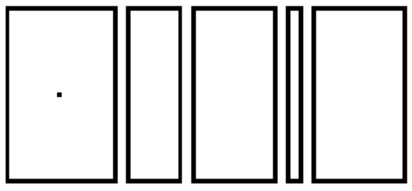


Figure 4.7: Example of area chart. The largest value is marked with a single dot.

4.3.3 Experiment Structure

With five different CNN's to test (VGG19 with and without transfer learning, XCEPTION with and without transfer learning and ResNet50) and five different visualization models (bar chart, pie chart, line chart, line chart filled and area chart), that give us a total of 25 experiments, since we want to test every visualization with every model.

Additionally, due to a random factor associated with the testing, both for the generation of the datasets and for the weight initialization of the MLP (and feature generator, if applicable), we decided to run every experiment 5 times with each set of parameters, totalling the number of base experiments to 125 (25 x 5 = 125).

In these experiments we follow most of the steps presented in the pipeline presented in Fig. 4.2, initializing the model, generating the appropriate dataset (according to the visualization type), training the model, storing it, and testing. The datasets used for these experiments consist of 100.000 samples, with a Train-Validate-Test split of 60/20/20, which means 60.000 samples are used for training, 20.000 for validation and 20.000 for testing. We calculate several metrics, which are covered in section 5.1, both for the validation and test sets, storing them along with the trained model itself (and its configuration).

After testing the model we also store all the true and predicted labels by the model, as well as the original values used to produce the charts in the test. This allows us to calculate any metric we might need in the future, as well as analyzing individual examples that might stand out.

Additionally, we also store other information such as the total time for the experiment, the random seed used, the size of the dataset and the date and time of when the experiment was conducted.

4.4 Visualization Variations

After asserting the most suitable CNN to use by analyzing the results of the base experiments, we need to test it further in order to better understand its capabilities and discover its limitations. For that, we defined variations for the proposed visualizations. For each variation, the same logic from the base experiments is followed: five experiments are conducted, training and testing a new model in order to properly analyze the effects of the changes to the charts. All further configurations remain the same, as well as the saved information regarding the experiment.

Rotation
The first variation introduced is a simple rotation. The point is to analyze whether the orientation of the chart has any influence on the performance of the CNN. For that, we simply apply a 90º clockwise rotation for the visualizations before using them for training and testing. Examples of this variation are visible in Fig. 4.8. Regarding the pie chart, this variation was not tested because of its nature: since it is an angle-based visualization, a rotation will have little to no significance, which is why it does not appear in the examples.

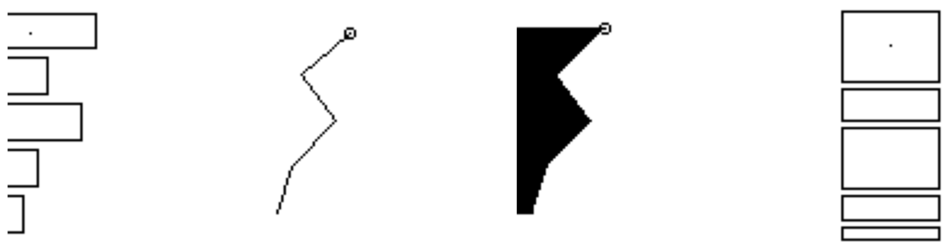
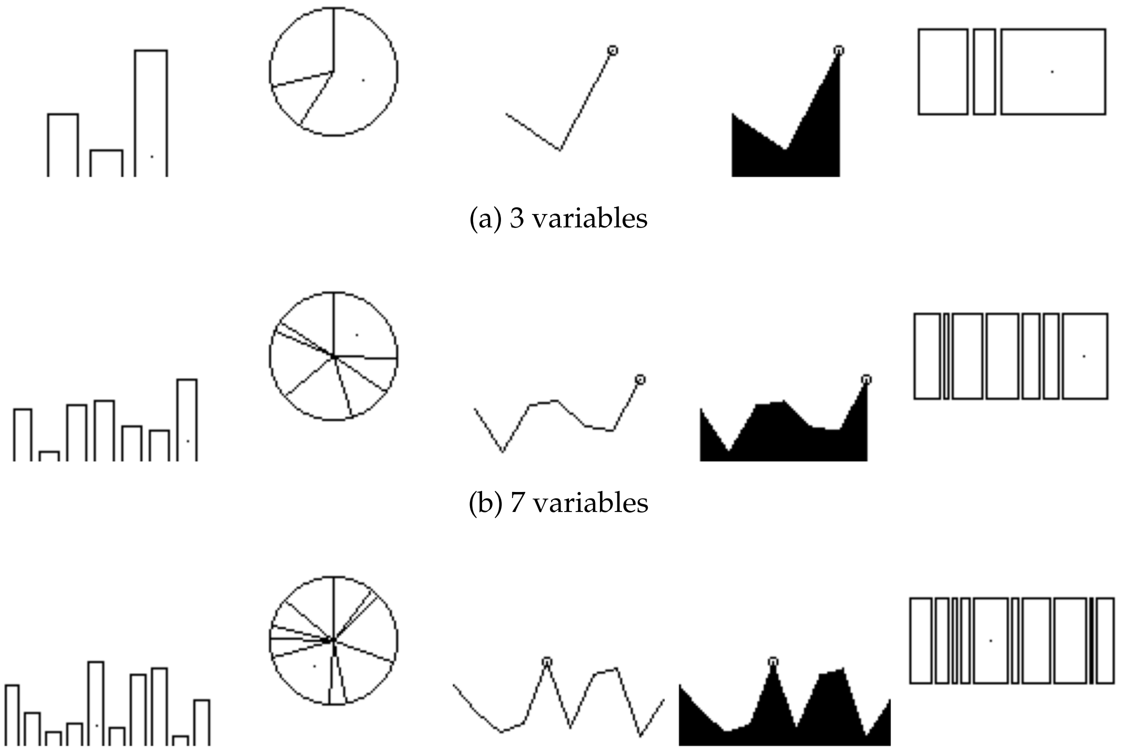


Figure 4.8: Examples of charts rotated 90º clockwise.

Number of variables
In the base experiments, and in all the examples shown so far in this chapter, only five values are represented in each visualization. The reason for that is that the initial tests conducted by Cleveland and McGill [1984] and Haehn et al. [2018] only made use of such examples, so it was our starting point. However, it is unrealistic to assume that all charts will only have 5 variables, as is assuming that a different number of values will not influence the performance of the CNN's. Therefore, in order to expand on previous research, we must test that change in our experiments. We added the feature to control the number of variables present to our dataset generator, and conducted tests with 3, 7 and 10 values in each visualization, hoping to understand how the variation of performance behaves (i.e. grows, decreases) in each of those scenarios, in comparison to the baseline of 5 values. Examples of this variation are shown in Figs. 4.9a through 4.9c.



(a) 3 variables

(b) 7 variables

(c) 10 variables

Figure 4.9: Examples of charts with different number of variables represented.

Color
Another property of the charts presented so far that is constant is the color display: black lines drawn on top of a white background. The line chart filled is already a variation of the line chart, but it is the only example, and a fairly simple one. Still, that same principle was applied to the remaining visualizations, creating a variation worth testing. Examples of such a change are shown in Fig. 4.10.

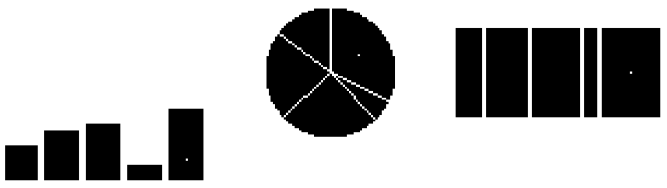


Figure 4.10: Examples of charts with the filled variation.

Additionally, another color-related variation used is reversing the colors used for the background and for drawing the chart. In other words, this variation consists of drawing the charts in white, over a black background, as opposed to the reverse in previous tests. Examples of this variation are shown in Fig. 4.11.

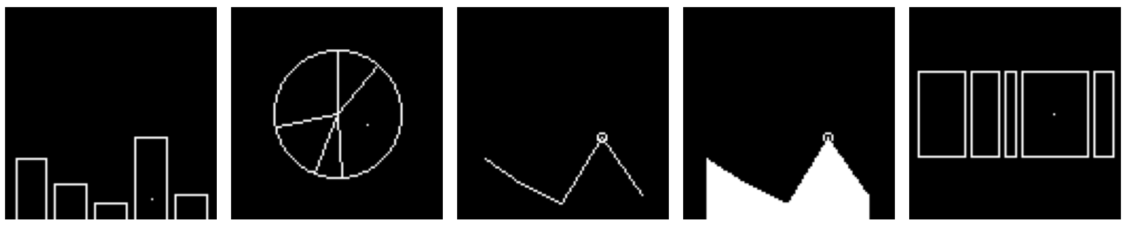


Figure 4.11: Examples of charts with the reversed variation.

Finally, we also propose a combination of these two color variations: reversing the colors and filling the charts. Examples are shown in Fig. 4.12.

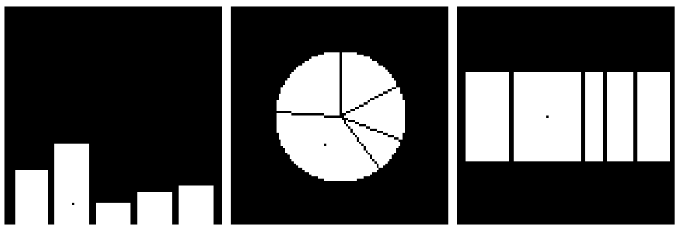


Figure 4.12: Examples of charts with the reversed and filled variations combined.

In order to test these color variations, the approach was different from the previous ones, which was similar to the one used in the base experiments. Here, instead of testing each color variation in its own experiment, we conduct a single experiment where the dataset is comprised of all the variations in a balanced distribution. This means that, for example, a dataset of bar charts, which has four possible versions, will have 25% of each one, in order to properly test the effect of the variation.

Even though we refer to these variations as “color-related“, we understand that these still only generate charts in black and white. However, if we were to test with the color spectrum outside the tones of gray, endless possibilities and combinations could be found, which would consume too much time to be beneficial to the experiments. Not only that but from our domain knowledge of how CNN’s work, we believe that changes of color would not deeply impact the performance, as long as there is enough contrast to notice the edges of the chart from the background.

4.5 API for Integration

Even though the main goal of this dissertation is research on the topic of ML4VIS and, more specifically, on methods using ML to evaluate visualization techniques, its integration into a visualization framework was the initial purpose for this work, as it was mentioned in the introduction (chapter 1). So integration plays a very important role in the development, being necessary to bridge the gap between these two components: the visualization framework as a whole and its evaluating node.

However, due to a conflict of programming languages used in both elements, this integration is not trivial. The proposed solution consists of developing a simple, yet effective REST (Representational state transfer) API. An API act as a standardized interface that facilitates the interaction and data exchange, necessary for the proper functioning of the framework.

As we mentioned, at this point the API if fairly simple with only a few endpoints necessary to connect the two elements, presented below.

List models
This endpoint allows the requester to retrieve the list of models available to use, i.e., models previously trained in the system. This way, the framework has access to this information and is able to make use of the appropriate models if they are available.

Model information
Given the full list of available models in the system, the framework also has the option to fetch more information about a particular model through this endpoint. The response in this case includes the configuration of the model along with the stored performance metrics calculated from its testing.

Evaluate visualization
This endpoint is what allows the main operation to be conducted: evaluating data visualizations. The way this endpoint works is by making a request to the API with an image of a chart, along with the values represented in it. The two elements correspond, respectively, to the input and expected output of a trained model, as has been previously explained in this chapter. With this pair, it is possible to conduct the evaluation by having the model predict the values represented in the figure and calculate the estimation error. This error is then returned as the

response to the request, providing the framework with an absolute value that can be used to compare different variations of charts to be presented.

4.6 Summary

Throughout this chapter, we described in full detail the work developed throughout the past months. More than just reporting the software developed and what experiments were conducted from that software, we explained our approach from the start and the reason for our sequence of tasks.

Starting from our logic behind the proposed method for visualizations evaluation, we explained how we can use the extraction of values from charts as a performance metric associated with a certain visualization technique. However, in order to affirm this, we need several tests to figure out the most suitable model for the task, as well as to assert whether or not this approach is even possible. For that, we propose a set of base experiments, to draw these basic conclusions, which require all the software necessary to generate an appropriate dataset, training a model and testing it to assert its performance, among other steps.

After that, more tests are needed so as to understand the viability of this method and of our models. These include slight variations in the data charts fed to the models to test their capabilities and limitations since in practice data visualizations come in many different ways. We tested variations regarding the orientation of charts, the number of variables presented and color settings used.

Additionally, in order to integrate our method with other tools, namely the adaptive visualization framework described at the start of the document, we implemented a simple API to allow the use of our software from external sources.

Apart from this last element, all the work developed provided us with a large amount of data resulting from the experiments conducted. In order to draw any conclusions from them, a deep analysis is needed, involving processing and visualizing our data in different ways. The next chapter describes in detail all those tasks and also provides a critical analysis of what we can extract from them.

Chapter 5

Results and Analysis

In this chapter, we describe our analysis of the results obtained from the experiments conducted. These results stem from the predictions made by the models on the test set from the dataset used for training. Here, our goal is to go beyond simply showing the basic overall metrics but to dig deeper and analyze the performance of the models in a more detailed level. We also attempt to discover why some of the results were obtained, in order to better understand the limitations of our models.

We start by providing a description of the metrics used for the analysis, as well as the reasons for their choice. Then, we discuss the results obtained from the base experiments, where we attempt to discover the most suitable model and how it behaved during those experiments with more detail. After that, we take a look at the results obtained from the variations experiments, used to further test the most suitable model, chosen from the analysis of the base experiments.

5.1 Metrics

In this section, we present the metrics used throughout this chapter to expose the results obtained from our experiments. Several metrics are calculated and saved from the predictions of the model, both on the test and validation sets. This duality allows us to compare both performances if we want, which can be useful to figure out whether or not over-fitting occurred during the training.

The first metrics considered are the Mean Absolute Error (MAE) and Mean Squared Error (MSE), two popular metrics used for regression problems. These calculate the overall error of the model's predictions by calculating the mean of each individual error, as the name indicates. This means that for each estimated label in the set, its error is calculated, summed to the total and then divided by the number of predicted labels, aggregating the error. It is important to mention that for these calculations, the labels are multiplied by 100 to turn them into percentages (remember that they represent the proportions of the variables, as it was explained in chapter 1 and section 4.1). The only difference between MAE and MSE is that for each individual error, MAE computes its absolute value while MSE computes its squared value, as it is visible in the following expressions (where y corresponds to the true label, ŷ to the predicted one and *n* to the size of the set):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i * 100 - \hat{y}_i * 100|$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i * 100 - \hat{y}_i * 100)^2$$

However, the main metric used for the overall performance of the models is the Mean Logarithmic Absolute Error (MLAE). This metric simply takes the MAE previously mentioned, sums 0.125 to it and computes its base 2 logarithm, as it is shown below.

$$\text{MLAE} = \log_2(MAE + 0.125)$$

The main reason for the use of this metric is the fact that part of our base experiments corresponds to a replication of similar experiments conducted by Haehn et al. [2018]. Therefore, in order to confirm that our implementation appeared correct, we made use of the same metric to compare our results with theirs. After that, we stuck with it as the main overall metric.

In addition to these overall metrics, which aggregate the errors associated with each individual prediction, we also analyze the error associated with each sample in the dataset, to perform a more detailed analysis on the performance of the model. In this case, the metric used is the absolute error, where, for each sample, the absolute error of each estimated label (remember that each sample, i.e. each chart, has multiple values represented) is calculated and summed as the total prediction error for a chart, much like the examples provided in chapter 1. This means that, for each sample, the absolute error is calculated in the following way, where *n* corresponds to the number of variables represented in the chart:

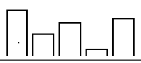
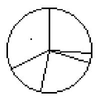
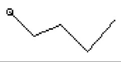
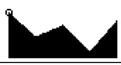
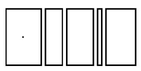
$$\text{Absolute Error} = \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Now that we have established the metrics used for our analysis, we are ready to move on and present the obtained results from the experiments, which starts in the following section.

5.2 Base experiments

As we mentioned before, we separated the experiments conducted into two sets. In this section, we present the results obtained from the base experiments, where we test different models with different visualization techniques in order to determine which model is the most suitable for the task of extracting the numeric values represented in a chart. Having said that, we present in Table 5.1 the results obtained from the experiments, from the main metric mentioned, MLAE. This metric represents an error, which means that lower values represent less error, and therefore a better performance. It is important to mention that since we ran five experiments for each combination of model and visualization, we present the mean and standard deviation obtained from the values of all runs. Additionally, we highlight the lowest values obtained for each visualization type, showing the best performing model.

Table 5.1: Results from base experiments. Values expressed in MLAE with standard deviation.

Chart Type	VGG19 w/ TL	VGG19	XCEPTION w/ TL	XCEPTION	ResNet50
	2.62 ± 0.10	1.30 ± 0.07	2.24 ± 0.04	1.29 ± 0.11	2.64 ± 0.16
	2.97 ± 0.04	1.71 ± 0.17	2.57 ± 0.04	2.12 ± 0.05	3.20 ± 0.15
	2.21 ± 0.05	1.18 ± 0.22	2.36 ± 0.01	1.09 ± 0.05	2.58 ± 0.07
	2.12 ± 0.08	1.12 ± 0.14	2.24 ± 0.04	1.06 ± 0.09	2.53 ± 0.06
	2.98 ± 0.07	1.44 ± 0.06	2.86 ± 0.07	1.72 ± 0.07	3.52 ± 0.04

As we clearly see, the VGG19 and XCEPTION models trained from scratch (i.e., without transfer learning) showed the best results, as they have the lowest errors. Unsurprisingly, these outperform the models employing transfer learning, since their Convolutional Neural Networks (CNN's) were previously trained and now adapted to this task, therefore not achieving the same results, as it was explained in subsection 2.2.1. On the other hand, the poor performance from the ResNet50 architecture was unexpected, as it presents the worst results from all the models tested, even those with transfer learning. It might be the case that ResNet50 is not as capable to capture the necessary features to perform the visual regression task.

While conducting these experiments, we came across a small discovery worth sharing. Regarding the pie chart visualization, initially, the chart generation was slightly different. We mentioned in subsection 4.3.2 that the layout of the values always starts at the top (12 o'clock) and progresses counter-clockwise. However, initially, that starting position was not fixed but actually randomly generated. Since we suspected this could influence the results, we decide to repeat the experiments and compare the performances in order to analyze this aspect. Those results are presented in Table 5.2.

Table 5.2: Pie chart variation comparison - Random starting position vs Fixed starting position, expressed in MLAE with standard deviation, along with the difference the results.

	VGG19 w/ TL	VGG19	XCEPTION w/ TL	XCEPTION	ResNet50
Random Position	3.52 ± 0.05	3.50 ± 0.98	3.03 ± 0.02	2.76 ± 0.11	3.87 ± 0.08
Fixed Position	2.97 ± 0.04	1.71 ± 0.17	2.57 ± 0.04	2.11 ± 0.05	3.20 ± 0.13
Difference	0.55	1.79	0.46	0.65	0.67

As we can see, our assumptions were correct and the random factor of the chart's starting point has an influence on the performance of models. Therefore, we decided to stick with the option of fixating the starting position, since it reduces the random factor associated with the training of the model. By reducing it, the model focuses only of the regression of the models which is the main task we want to assess.

Getting back to the overall results, in order to better visualize the performance of the models, we also present a set of histograms that show the distribution of the errors associated with each individual sample in the test set. In other words, with a test set of 20.000 charts and labels, the model will make 20.000 predictions, generating 20.000 sets of estimated labels and from their comparison to the true values we obtain 20.000 different errors, one for each sample. By plotting these values in a histogram, we get the distribution of the error, allowing us to visualize properties such as the most common error values. However, since we run each experiment five times, five histograms will be generated, each with its results. Therefore, we calculate the average of those five histograms to visualize the average performance of the model. Fig. 5.1 shows an example to illustrate this strategy, presenting the results from the XCEPTION model trained from scratch on a dataset of bar charts (Barchart XCEPTION experiment).

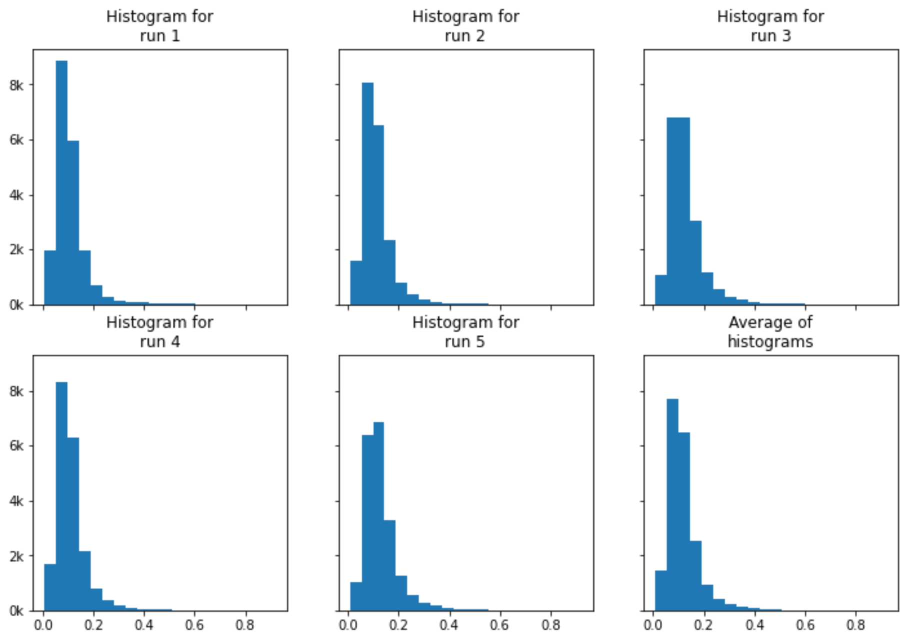


Figure 5.1: Example for average of histograms strategy from Barchart XCEPTION experiment results. The first 5 plots show the distribution of absolute errors from the test set for each run of the experiment, while the last plot shows the average number of samples in each bin.

Here, after calculating the absolute error for each prediction, as it was explained in the previous section, we obtain 20.000 values for each run. After that, we take the overall minimum and maximum error from the five runs, create 20 evenly spaced bins within those limits and for each bin, count the number of errors in each one. This is needed because if we want to calculate the average count in each bin, they must consist of the same interval. Then, we simply plot the histogram with the average count for each bin (bottom right corner of the figure), providing us with an average error distribution for this model. Applying the same logic for all trained models, we get the set of histograms presented in Fig. 5.2.

First of all, we can see that the position of the histogram reflects the results presented in Table 5.1: the closer to the left (closer to zero), the lower the error, which, once again, show VGG19 and XCEPTION as the best performing models. Another aspect to take into account is the shape of the histogram. If the shape resembles a bell-shaped curve (such as Barchart ResNet50), similar to a Gaussian distribution, it means that the most common result is considered "average", or in other words, not the worse but also not the best. On the other hand, histograms whose most common results lean closer to one of the sides, i.e., with skewness, show that most times the model performs to its best (or worse, according to the side in which the histogram leans), while the other observed values are considered "exceptions". This is a relevant detail since it allows us to measure the variance of the model's performance, which preferably should be low.

According to both criteria, it is clear that the two most suitable models are VGG19 and XCEPTION. Looking once again into the MLAE values (Table 5.1), we can see that XCEPTION performs slightly better (though very close) than VGG19 on

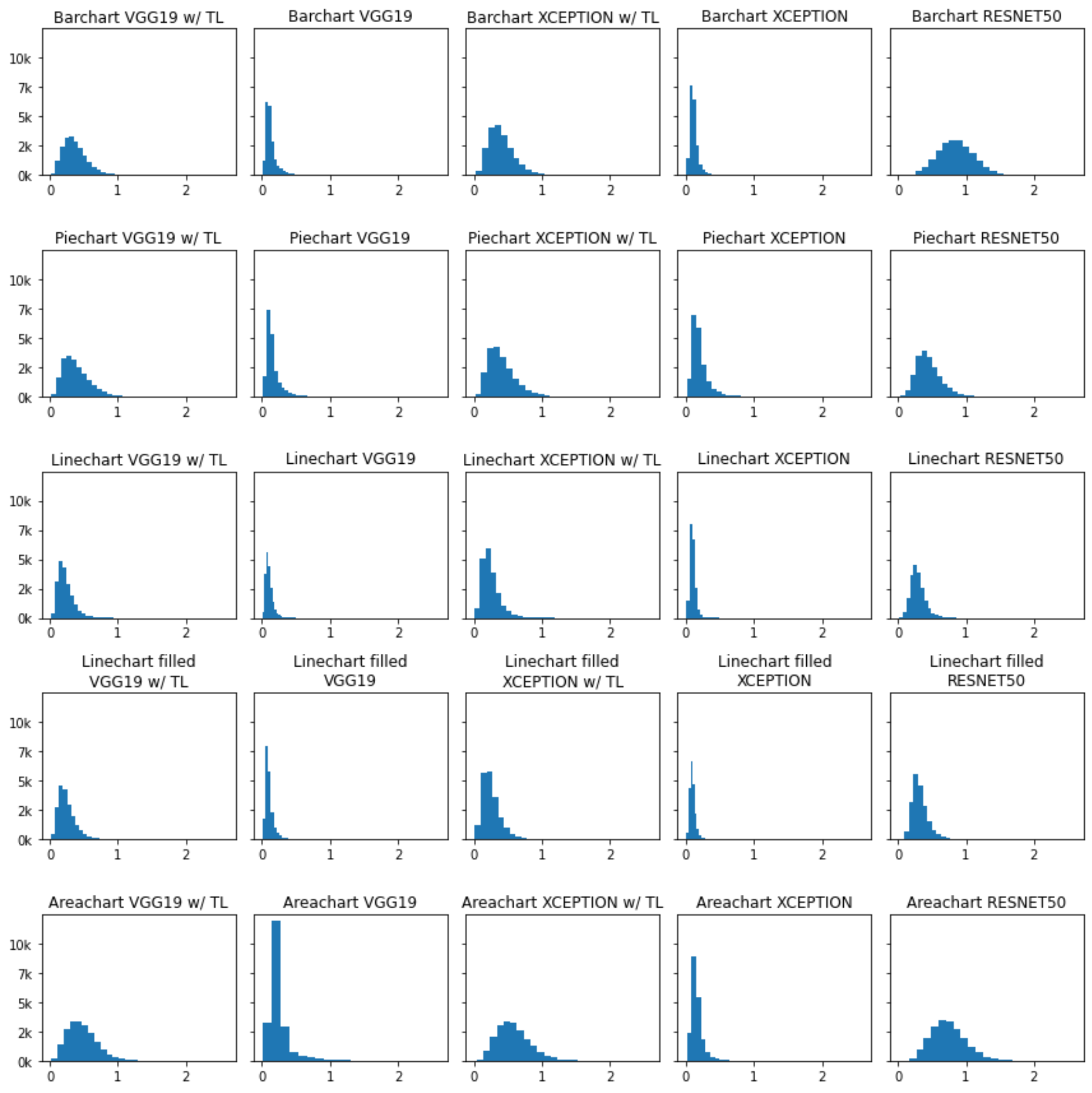


Figure 5.2: Average histograms for absolute error distribution for all experiments.

bar charts, line charts and line charts filled, whereas VGG19 more clearly outperforms XCEPTION on pie and area charts. Which one should we choose as the most suitable?

We believe XCEPTION is the better option to perform this task, for two reasons that are related. First of all, had we conducted similar experiments with users, we would choose a model that could better resemble the performance of a human. For these results, we have some useful data from one of the papers which we based on, by Haehn et al. [2018]. Fig. 5.3 shows the results from similar experiments as ours, for bar charts and pie charts and from several Machine Learning (ML) models, as well as from human users.

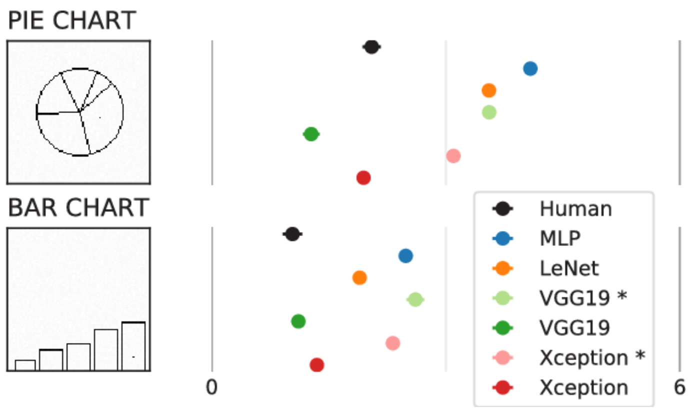


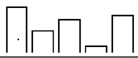
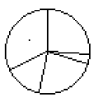
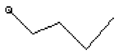
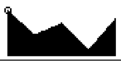
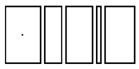
Figure 5.3: MLAE results from experiments with bar and pie charts, for different ML models and human performance [Haehn et al., 2018].

From this figure, we can draw one main conclusion: XCEPTION more closely resembles human performance than VGG19. Yes, VGG19's MLAE value is closer to human MLAE for bar charts, but XCEPTION's is far closer when it comes to pie charts, compensating the loss for bar charts, and is not that far behind either. In summary, having to choose one according to this criterion, we are going with XCEPTION.

The second reason for choosing XCEPTION over VGG19 is their performance regarding pie and area charts. VGG19 clearly outperforms in these examples, but let us keep in mind that we do not wish for the best possible performance - we wish for one that looks more like a human's. As it was mentioned before in subsection 4.3.2, Cleveland and McGill [1984] list angle and area (used by pie and area charts, respectively) as poor elements in terms of perceptual accuracy, unlike position, for example, used by the remaining chart types, which is considered the best. Therefore, we would expect a worse performance for pie charts and area charts. Although we observe that for both models, the difference between those two and the remaining ones for VGG19 is not a considerable one. We believe a larger difference in performance would be expected, such as the one visible for XCEPTION. For these reasons, we choose XCEPTION trained from scratch as the most suitable model for the task.

Before advancing into a deeper analysis of the results for this model, we decided to test another hypothesis regarding the generated charts and the performance of the models. We mentioned the importance of marking the largest value represented in the chart in subsection 4.5, so we wanted to put that theory to the test. In order to do so, we repeated the experiments for each visualization type without any sort of marking for the largest value, but only for XCEPTION from scratch, which we found to be our best suited model. Just like every other experiment, we ran it five times and calculated the mean and standard deviation from the results, presented in Table 5.3.

Table 5.3: Results from experiments comparing the performance between regression from charts with the largest element marked against charts without it. Values expressed in MLAE with standard deviation, as well as the difference between the results for each chart type.

Chart Type	With mark	Without mark	Difference
	1.29 ± 0.11	1.24 ± 0.07	0.05
	2.12 ± 0.05	2.15 ± 0.05	-0.03
	1.09 ± 0.05	1.47 ± 0.09	-0.38
	1.06 ± 0.09	1.28 ± 0.04	-0.22
	1.72 ± 0.07	1.89 ± 0.10	-0.17

These results are somewhat inconclusive, as far as we can tell. For bar charts and pie charts, it seems as though there is little to no difference in the performance, so much so that for bar charts the performance even improves slightly, but within the standard deviation intervals. However, the same cannot be said for the remaining visualizations. Despite not being a great loss in performance, we cannot deny it exists. Since we want to maximize the capabilities of the models in performing visual regression, and this is a factor that might influence that, ever so slightly, we would rather avoid that possibility. Especially considering that the experiments discussed in the next section have an expected increase in difficulty for the model.

Now that we have settled on the most suitable model, let us take a deeper look at the results obtained from those experiments. First, we isolate the histograms related to that particular model from the others, in other to allow for a close-up view, shown in Fig. 5.4.

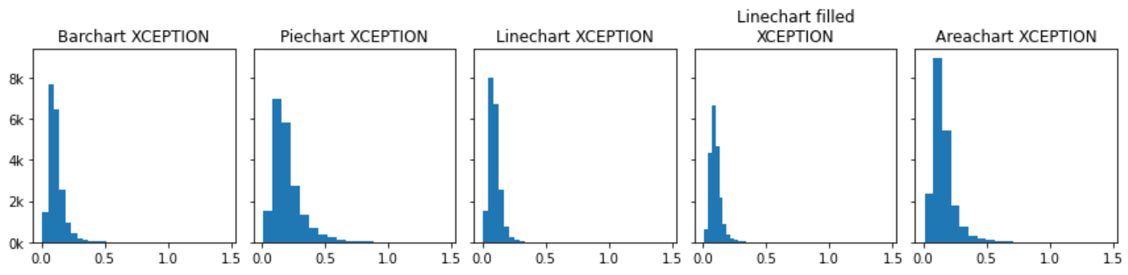


Figure 5.4: Results from the experiments with XCEPTION trained from scratch - average histogram of absolute error distribution for each chart type.

Although we can now see more clearly the distribution of errors, it still does not convey as much information as it possibly could, since the values shown represent the absolute error from the real and predicted labels, which are normalized, therefore yielding small errors (less than 1). For that reason, we looked into the process of reverting the normalization (described in section 4.1 and subsection 4.2.2), transforming the predicted labels back into values capable of generating a chart. That way, we calculate the error in a much more clear and visible way. For example, having the original values used to create the initial image and the ones predicted by the model we can compare the two corresponding charts, as it is shown in Fig. 5.5.

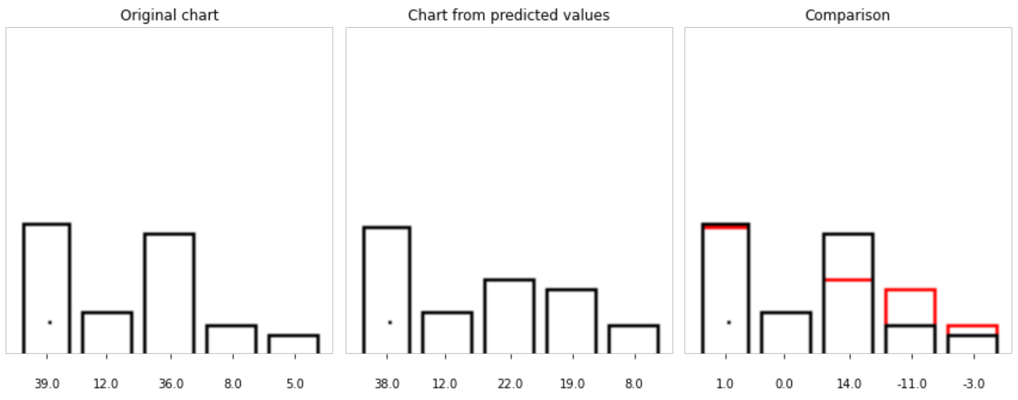


Figure 5.5: Comparison of original chart against the one generated from the perceived values by the model. The third image shows their comparison by plotting the chart obtained from the perceived values (red) over the original one (black).

In the first image (from left to right) we have the original chart, fed to the model during the prediction stage, with its values underneath. Then, in the second image, the chart generated from the values perceived by the model from the original chart is shown underneath as well. Finally, in the third image, we compare their visual differences by placing the second chart (red) on top of the first one (black), showing their numerical difference underneath.

Applying this de-normalization process for each set of labels obtained from the test set, we can now show the previously shown histograms again, but this time with discrete errors, based on the original/de-normalized values and rounded to the unit, as it is seen in Fig. 5.6.

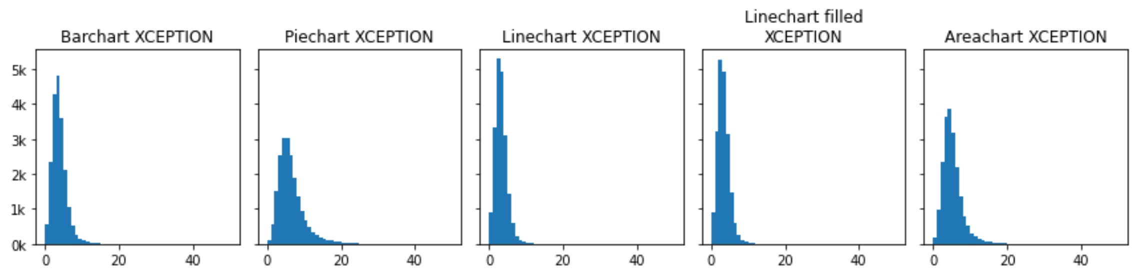
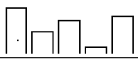
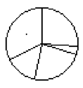
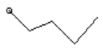
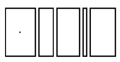


Figure 5.6: Results from the experiments with XCEPTION trained from scratch - histogram of absolute error distribution with de-normalized labels, for each chart type.

Through these plots we can better understand the magnitude of the errors from the model's predictions. Remember that these represent the average performance of the five runs for this model. For example, we can clearly understand the magnitude of an error of 3 (knowing the scale, obviously) - among values ranging from 3 to 39, an error of 3 is not that large, whereas one of 20 has more impact. Additionally, these plots allow us to extract more information from the results that we otherwise could not, such as the number of "perfect" estimations - predictions in which the corresponding error is zero (approximately, due to rounding), represented in the first bar - or the most common error, for example. Table 5.4 shows some of this data.

Table 5.4: Details from histograms of discrete error distribution. Perfect estimations (approximate error of 0), most common error and average error for each chart type.

Chart Type	Perfect Estimations	Most Common Error	Average Error
	549	3	3
	88	4	6
	891	2	3
	900	2	3
	174	4	5

The next step we took was to look at the best and worst performances of the model for each type of visualization. By looking at the charts and the represented values, we might be able to better understand where the model struggles and where it exceeds. In order to do that, we gathered the top 5 largest and lowest errors obtained for each visualization type and analyzed them to attempt to draw conclusions.

Regarding the best regressions from bar charts, we found that the model excels with relatively soft transitions in the values and a minor difference between the largest and smallest numbers represented. Such an example is shown in Fig. 5.7, representative of the top 5.

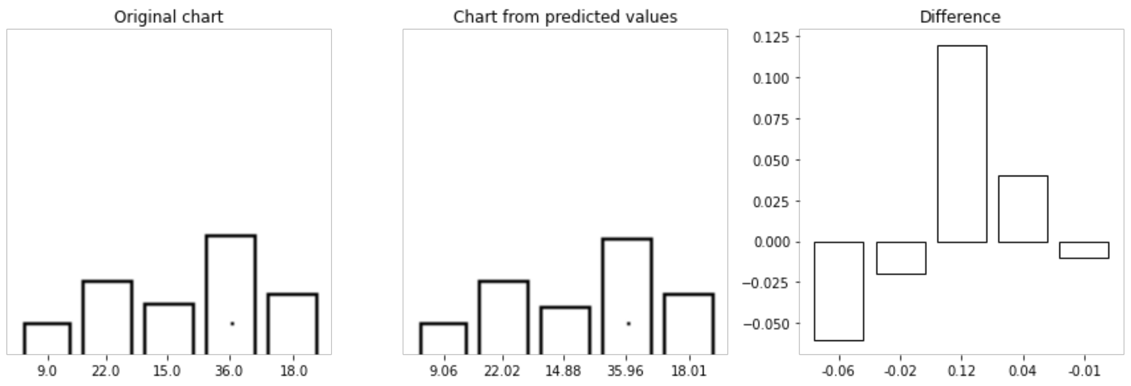


Figure 5.7: Best estimation for the bar chart. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each variable (third panel).

Similar to Fig. 5.5, we compare the original chart with the one created from the values perceived by the model, but instead of plotting one on top of the other (rightmost image in Fig. 5.5), we simply present a bar chart representing the difference between the original and restored data. Additionally, it is worth pointing out that in this analysis we do not fully discretize the predicted values. Instead, we round them to two decimals cases. Since we are showing the best estimations and the error is so reduced, if we were to round the predicted values all to the unit, we would consistently get an error of zero.

Getting back to the analysis, we can see that there are really no abrupt transitions (compared to the top 5 worst performances) and that the difference between the minimum and maximum values is relatively small too (9 to 36), which leads to a better distribution of the values and, therefore, softer transitions, as opposed to larger ones (4 to 39, for example). We can see the exact opposite by looking at the examples from the top 5 worst performances, presented in Fig. 5.8, where the individual errors are much larger.

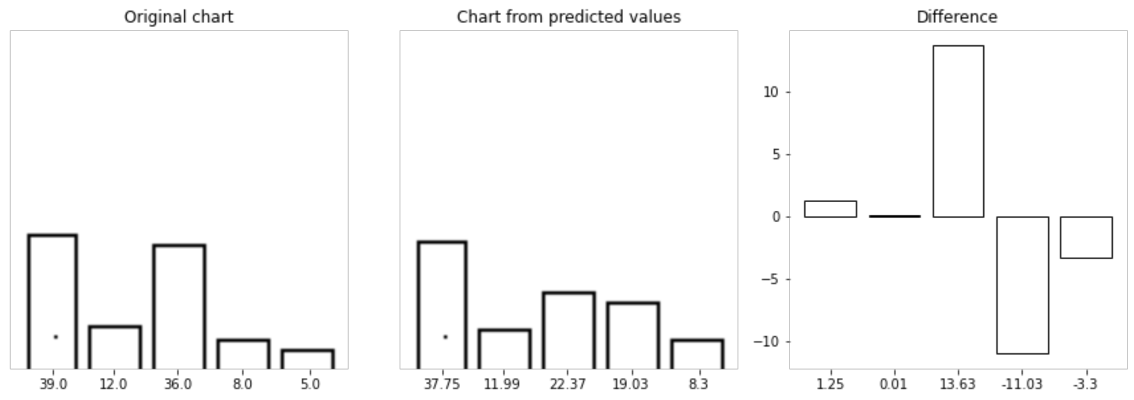
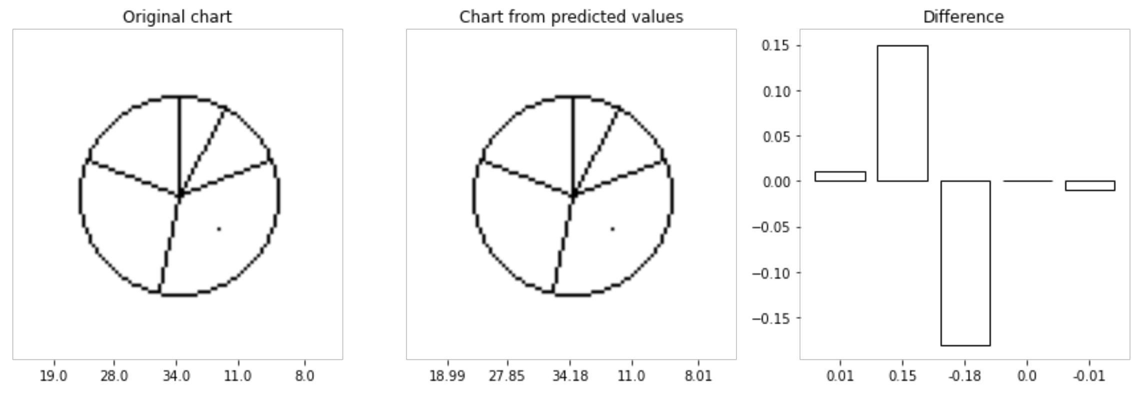


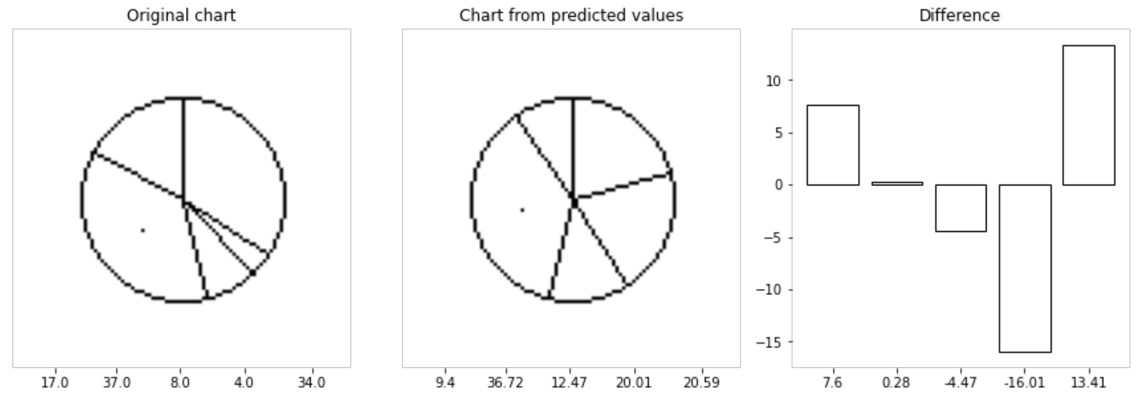
Figure 5.8: Worst estimation for the bar chart. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each original and perceived variable (third panel).

Here we can see that the model really struggled with two sudden transitions, leading to very large errors. Additionally, we can notice a substantial difference between the values: two of them over 35 and two under 10, which corresponds to a more unequal distribution, which contributes to the sudden transitions mentioned.

Looking at the best and worst examples for pie charts, we spot similar patterns, visible in Fig. 5.9.



(a) Best estimations

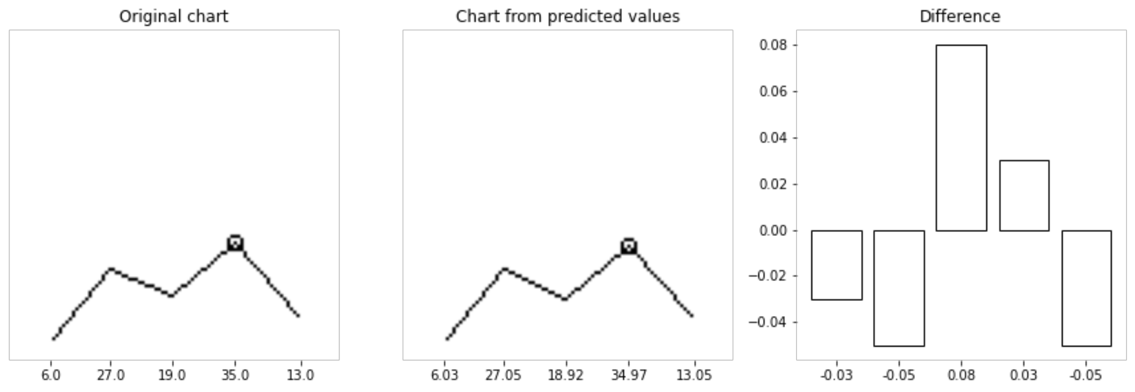


(b) Worst estimations

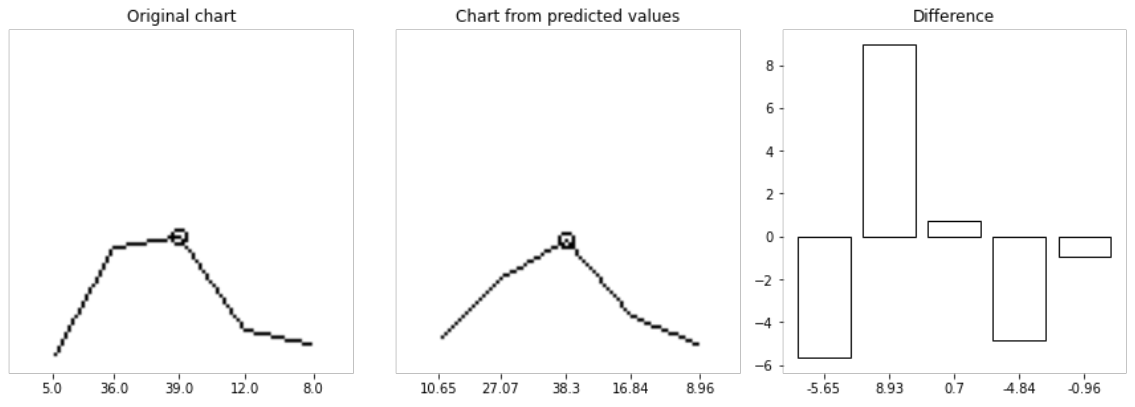
Figure 5.9: Best and worst estimations for pie charts. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each original and perceived variable (third panel).

Again, we can clearly see that the model performs well with more equally distributed values, as opposed to the largest differences, visible in the worst example. In addition to that, we also noticed a pattern for a poor performance of two small values in between two large ones (37, 8, 4, 34), for example.

Regarding line charts and line charts filled, we find similar patterns, visible in Figs. 5.10 and 5.11.

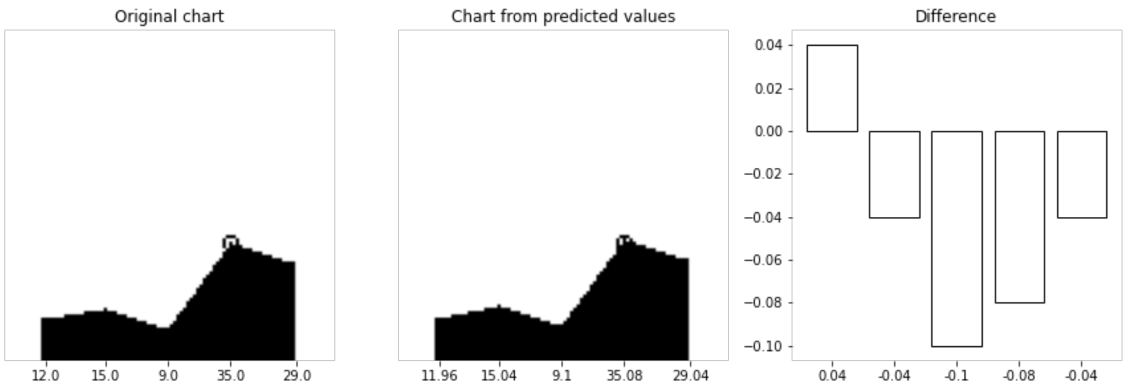


(a) Best estimations

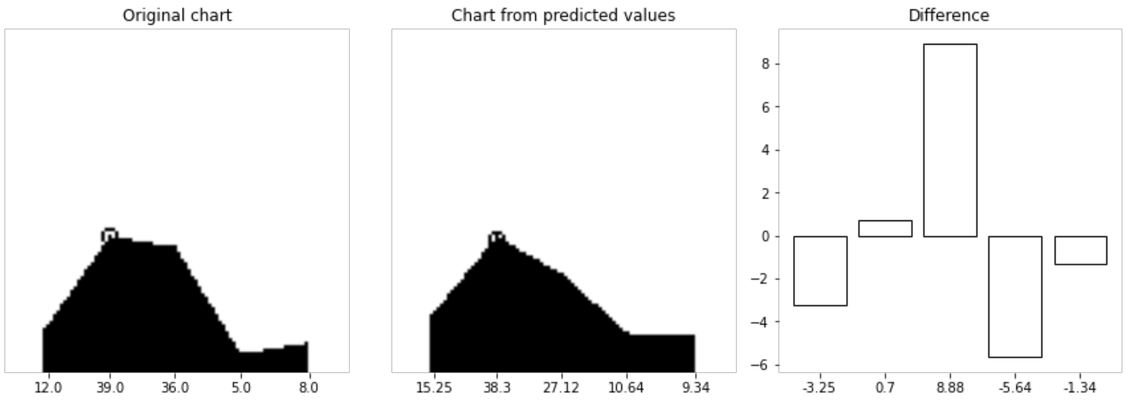


(b) Worst estimations

Figure 5.10: Best and worst estimations for line charts. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each original and perceived variable (third panel).



(a) Best estimations

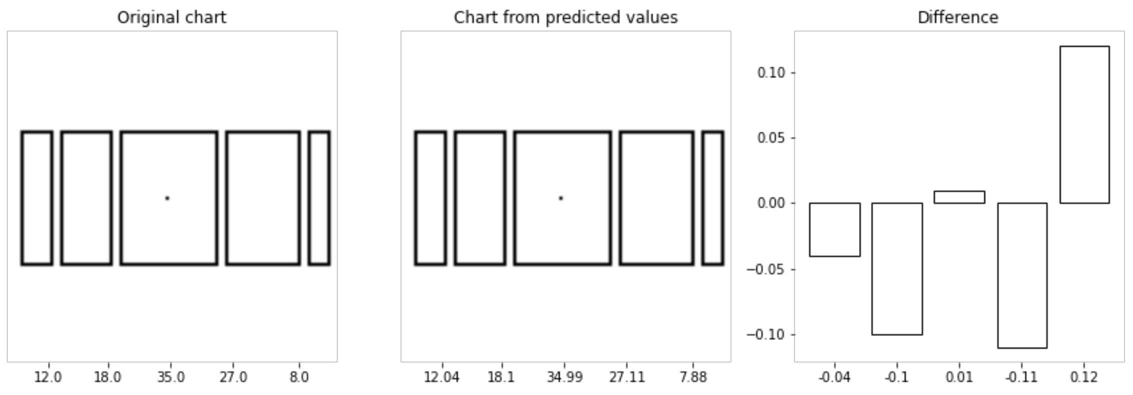


(b) Worst estimations

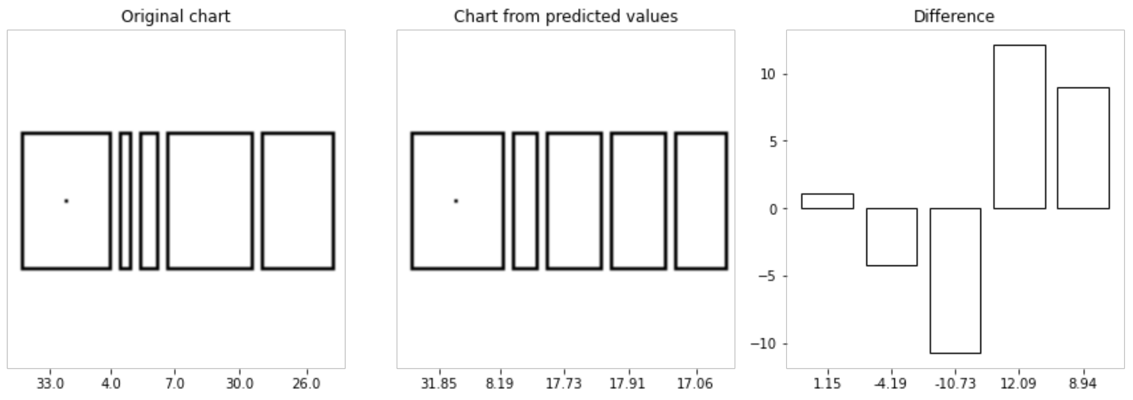
Figure 5.11: Best and worst estimations for line charts filled. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each original and perceived variable (third panel).

Once again, it appears that the model deals better with a more equal distribution of values. However, it seems as though for line charts the model is better at dealing with sudden changes, especially line charts filled, as is shown in the example in sub-figure 5.11a, since there are other similar examples with good performances. Perhaps it might be due to its main feature: a line connecting each point with the adjacent ones, which can help dealing with those changes.

Finally, regarding area charts, our observations are very similar to the ones for pie charts. Examples are shown in Fig. 5.12.



(a) Best estimations



(b) Worst estimations

Figure 5.12: Best and worst estimations for area charts. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each original and perceived variable (third panel).

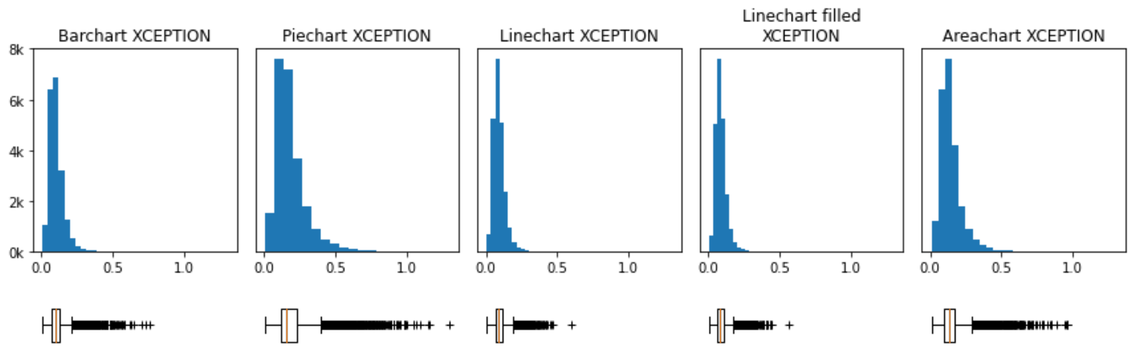
One more time, the model struggles with relatively unequally distributed values, namely in cases where there is a large difference between the smallest and largest values. Also, it appears that, just like the case with pie charts, having two small values between two large ones (33, 4, 7, 30 in sub-figure 5.12b, for example) also presents itself as a challenge.

In general, we can conclude that visualizations with high contrast in the values can be challenging to perform visual regression. Sudden changes in the values might also provoke this, but it seems like line charts and filled line charts allow the model to better deal with them. That is probably why they are the two best-performing visualizations, too.

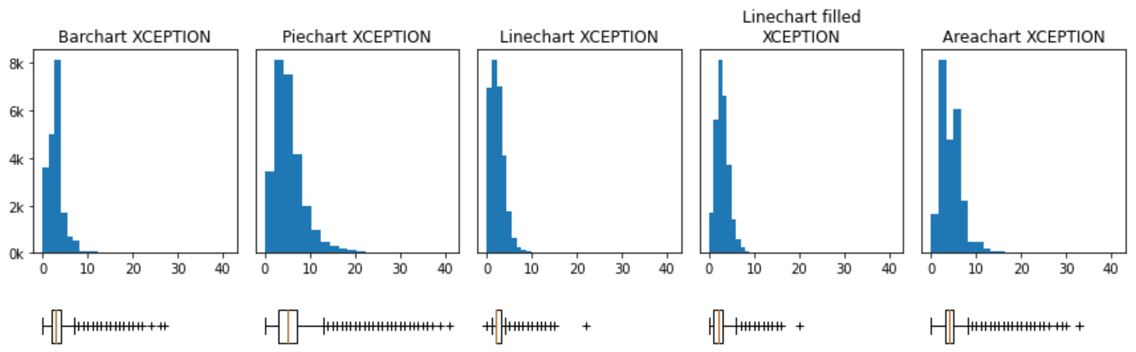
Furthermore, we can also conclude that there is no general tendency for the model to consistently overestimate or underestimate variables. In other words, we could potentially observe that when having a noticeable prediction error the tendency of the model would be to estimate values consistently lower, but that was not the case.

It is also very important to remind the following: even though we showed examples of very poor predictions by the model, those are very rare exceptions. The examples shown represent summed absolute errors of over 20, while as we presented in Table 5.4 we show that the average errors are nowhere close to that. Regardless, it is important to analyze them so as to understand the weaknesses of our models.

In fact, if we look into outliers from the errors obtained we can confirm this. In Fig. 5.13 we show the histograms from the errors obtained for the best run of each visualization type. This means that from the five runs we tested for each visualization, we took the best from each one (the one with the least overall error) and plotted the absolute errors from each sample, both normalized and denormalized. Additionally, to show the presence of outliers, we also added a box and whiskers plot associated with each histogram.



(a) Normal error



(b) Discrete error

Figure 5.13: Histograms of absolute error (normal and discrete) distributions from the best run of each visualization type. For each histogram there is a corresponding box and whiskers plot showing the data distribution, including outliers.

As we can see, there are many outliers marked with a plus sign (+) for all chart types. This goes to show that even though there are a few examples of poor results, they are not the norm. As a matter of fact, they are very far away from the median (marked by the yellow line), showing these are exceptions to the norm.

Finally, after analyzing the performance of the model, we can also look into the performance of the visualization types. Although the main goal of our analysis is to assess the viability of our method and of our proposed ML models, we should not forget that the whole point of even having this method is so that we can draw conclusions regarding which visualization techniques allow for a better perception of the represented values. With that being said, we should analyze our obtained results in that sense.

Based on the MLAE values shown in Table 5.1 and on the histograms for the errors shown in Fig. 5.2, we can easily compare the performances between the different visualization types.

First and foremost, we confirmed the results obtained from Cleveland and McGill [1984] and Haehn et al. [2018] proving the bar chart is a superior chart type than the pie chart in regards to visual perception.

Then, we also confirmed that using the area as a visual element is a poor approach following the same criterion. However, it was surprising to find that in spite of that poor performance from the area charts, the results were still better than the ones from the pie charts, placing the latter at the bottom of our ranking of visualizations.

Lastly, the performance from the line chart and line chart filled was also somewhat surprising. Since the main element used (position) is the same as for the bar chart, we were expecting similar results, which was not the case. Though there was not a huge difference, it is still noticeable. It might not be the most useful piece of information, however. Bar charts and line charts are suitable for different types of data (categorical data and time series, respectively), so their comparison should not happen often. Additionally, it seems as though the filling underneath the line plot also helps the model to better extract the values represented, which can be a useful detail when designing such a chart.

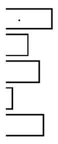
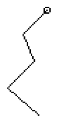
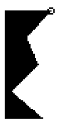
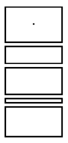
Throughout this section we conducted a deep analysis of the results obtained from our experiments, showing the performance of the models trained for the visual regression task, as well the performance of the different visualization techniques. This was done with the main goal of determining the best model for our method of evaluation. We conducted this exploration in many ways and many levels of detail, from analyzing a single overall metric for each experiment, to looking into the distribution of errors resulting from the predictions of the models, and even visualizing the best and worst individual predictions. This last example was done in a way to understand the strengths and weaknesses of our model. However, it is not enough. In order to fully grasp the capabilities of our models, as well as their limitations, we had to conduct not only more experiments but more varied ones. The results from those are presented and analyzed in the following section.

5.3 Variations

Now that we have established the XCEPTION CNN trained from scratch as the most suitable model for the task of extracting the values represented in a data chart through the use of a set of simple experiments, we intend to test it further. The base experiments previously mentioned, despite testing different visualization techniques, are still only comprised of simple examples: charts representing five variables, drawn in black lines over a white background. As we described in full detail in section 4.4, we tested different variations in the proposed visualizations in order to determine the full capabilities and limitations of our model. Throughout this section we present the results from these experiments, analyzing them during the process. Just like the base experiments, every test was conducted five times to take into account the random factor, with a dataset size of 100.000, and the main metric MLAE.

Rotation
The first proposed variation was related to rotation. More specifically, a 90º rotation clockwise. This is an important variation since not all charts follow the left-to-right display that we use for most of our visualizations. For example, bar charts with horizontal bars are common. So we should test this variation in order to understand whether or not the model deals well with different layouts. As we mentioned before, we did not run this experiment for pie charts as it does not provoke any meaningful difference in its layout, due to its nature. The results for the remaining visualizations are presented in Table 5.5.

Table 5.5: Comparison of results for normal and rotated charts. Values expressed in MLAE with standard deviation, as well as the difference between the results for each chart type.

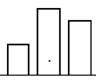
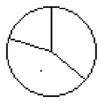
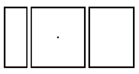
Chart Type	Normal	Rotated	Difference
	1.29 ± 0.11	1.20 ± 0.14	0.09
	1.09 ± 0.05	1.19 ± 0.08	-0.10
	1.06 ± 0.09	1.03 ± 0.11	0.03
	1.72 ± 0.07	1.79 ± 0.06	-0.07

As we can see the rotation had barely any impact in the results. Besides being very small differences, in two of the visualizations, the error was even actually less than that of the normal, non-rotated charts. Therefore, we conclude that rotation, or rather, orientation, has no impact when it comes to the performance of the model. This is not surprising since to the CNN the task difficulty is exactly the same. The only difference is that instead of reading the chart from left to right, it does so from top to bottom. But since they are trained for that purpose in exactly the same way, the overall performance is essentially the same. However, it is important to point out that the variation has no impact if the model was trained under that variation. If we were to feed a 90° rotated chart to a model trained only with non-rotated charts, its performance would obviously be very poor.

Number of variables
Moving on to the next variation, we decided to test the performance of the models in regards to the number of variables represented in the charts. As we mentioned before, this change in the number of variables also required a change in some parameters in the code used to generate the datasets, namely the maximum value, minimal difference and total sum of the values. The explanation of these parameters is provided in section 4.5.

We start by presenting the results from the experiments with 3 variables represented in each chart in Table 5.6. For this number, we adjusted the maximum value possible to 60, from 39. This was needed because having a total sum of 100 with that limit, there would be a relatively small number of possible combinations, leading not only to unrealistic training, but also to an unfair comparison against models trained with far more diverse datasets and even, while less relevant, much larger duration to generate the charts (since the values are randomly generated, a longer time would be needed to find the possible combinations).

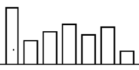
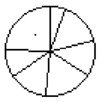
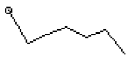
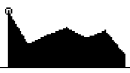
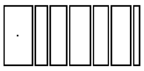
Table 5.6: Comparison of results from experiments with charts representing 5 and 3 variables. Values expressed in MLAE with standard deviation, as well as the difference between the results for each chart type.

Chart Type	5 variables	3 variables	Difference
	1.29 ± 0.11	1.91 ± 0.11	-0.62
	2.12 ± 0.05	2.48 ± 0.11	-0.36
	1.09 ± 0.05	1.96 ± 0.17	-0.87
	1.06 ± 0.09	1.54 ± 0.12	-0.48
	1.72 ± 0.07	2.46 ± 0.15	-0.74

In order to analyze these results, we decided it would be best to compare them with the ones from the experiments with 5 variables, therefore using those as baseline results. As we can see, there is a decrease in performance. Although noticeable, we do not consider these to be terrible results but rather a loss in precision. This decrease is surprisingly large for line charts, especially since the same was not observed from its own variant, the line chart filled. We would expect that the results would maybe improve due to the fact that the model has fewer values to regress, but it was not the case. On the one hand, we could argue that this could be a consequence of also having fewer variables to compare in the chart, making it harder to precise the correct proportions. But according to this theory, more variables should account for more precision, which we find highly unlikely.

The next results presented are the ones obtained from the experiments with charts containing 7 variables, shown in Table 5.7. For this experiment, we made the following adjustments in regard to the charts' generation: the total sum of values increased to 150 from 100 and the minimum difference decreased to 2 from 3. Since now there are more variables to be represented, we need a larger sum and more flexibility in which values can be used in order to generate the appropriate datasets.

Table 5.7: Comparison of results from experiments with charts representing 5 and 7 variables. Values expressed in MLAE with standard deviation, as well as the difference between the results for each chart type.

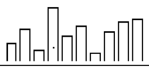
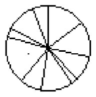
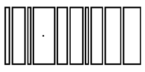
Chart Type	5 variables	7 variables	Difference
	1.29 ± 0.11	1.63 ± 0.07	-0.34
	2.12 ± 0.05	3.58 ± 0.05	-1.46
	1.09 ± 0.05	1.71 ± 0.11	-0.62
	1.06 ± 0.09	1.57 ± 0.09	-0.51
	1.72 ± 0.07	2.27 ± 0.47	-0.55

In general, these results are close to the ones obtained from 3 variables. In some cases (bar, line and area chart) the decrease in performance is actually smaller, although not by a wide margin. However, there was a very large and unexpected difference noticed for pie charts. A possible reason for this poor result may be related to having less "space" for each variable to be represented: regardless of the number of variables to show, the pie chart will only have a 360° amplitude to represent them. Having more values leads to less amplitude for each one, which causes them to look more similar than they would if there were fewer, therefore leading to a tougher regression for the model.

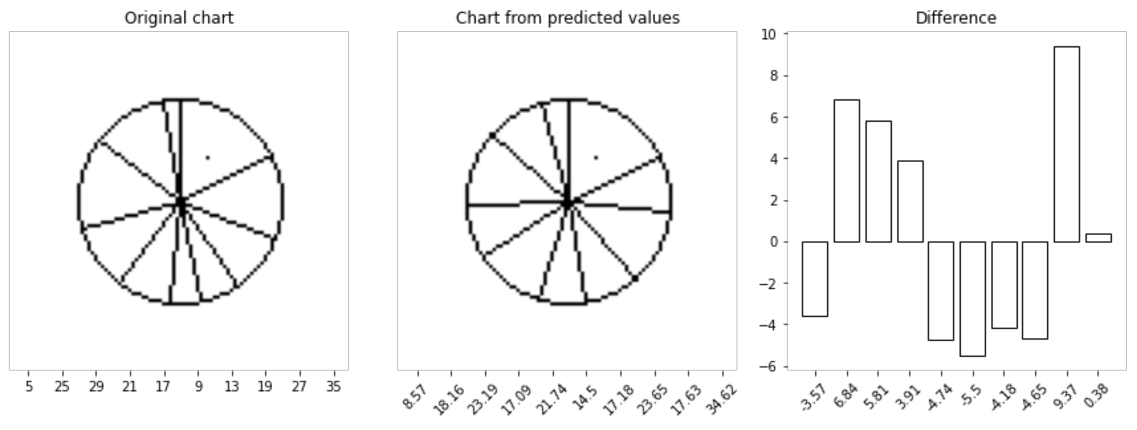
The final results presented for this variation correspond to the experiments using charts with 10 variables represented. In this case we made the following adjustments to the data generation process: a total sum of 200, as opposed to the original 100 and a minimum difference of only 2, compared two the initial one of

3. Just like the case of 7 variables, these changes were done in order to allow for more possible combinations of values to be generated. The results are presented in Table 5.8.

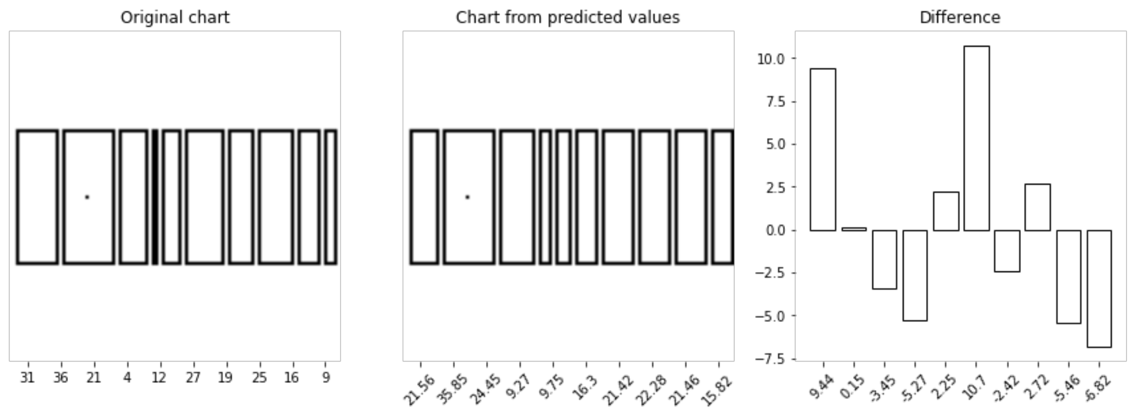
Table 5.8: Comparison of results from experiments with charts representing 5 and 10 variables. Values expressed in MLAE with standard deviation, as well as the difference between the results for each chart type.

Chart Type	5 variables	10 variables	Difference
	1.29 ± 0.11	2.15 ± 0.16	-0.86
	2.12 ± 0.05	4.35 ± 0.02	-2.23
	1.09 ± 0.05	2.13 ± 0.06	-1.04
	1.06 ± 0.09	2.02 ± 0.17	-0.96
	1.72 ± 0.07	4.33 ± 0.03	-2.61

As we can see, the decrease in performance is greater than previously seen, namely when comparing these experiments with the ones with 3 and 7 variables. Regarding bar and line charts with and without filling, this decrease simply reflects a loss in precision, whereas for pie and area charts it is much different. It is clear that the decrease is much larger, and the obtained MLAE is well over 4. We checked the metrics obtained from the validation set in order to assess whether or not it was a problem of over-fitting, but it was not. We tested the performance of a random model and obtained an MLAE value of 5.18, meaning that our results represent a performance only slightly better than that. We suspect this means these models did not actually properly learn the task. While the model for bar and line charts learned how to perform the regression but can only do so with limited precision, the models for pie and area charts are not even able to do that. The reason why they might perform better than a random model is that they simply learn to estimate the largest value as 1, therefore reducing the error. That task is relatively easy to learn since the model simply needs to output the value 1 in the first position, as it was explained in section 4.1. In order to verify this theory, we took a look at a few specific examples. But while in the previous section, we analyzed the top 5 best and worst cases, here we must look at the ones in the middle, so as to analyze the average performance of the model. Fig. 5.14 shows representative examples of such cases, both for pie charts and area charts.



(a) Pie chart



(b) Area chart

Figure 5.14: Examples of average predictions from models trained for pie and area charts with 10 variables. Comparison between the original chart (first panel) and the one generated from the perceived values (second panel) is done by presenting the difference between each original and perceived variable (third panel).

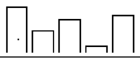
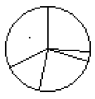
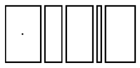
These examples show that one of our suppositions was correct but the other was not. The model in fact learns how to perform the regression, although with very low precision, leading to the large errors visible in the the right panel in each of the sub-figures. On the other hand, we were right about the second hypothesis. As we can see, the error associated with the largest variable is very reduced, especially compared to all the others. But while this reduces the error, it does not actually consist of a useful contribution to the model's performance because it simply corresponds to producing an output of 1 for the first value.

In summary, despite being able to perform the regression task, the models do so with so little precision that they can be mistaken for random predictions. This loss of precision is most likely due to the reason explained in the previous analysis (charts with 7 variables): for both pie charts and area charts the values have different representations according to the number of variables to be shown. For example, for bar charts, a value of 20 will always correspond to a bar of the same height, regardless of the total number of variables. On the other hand, a value of 20 for a pie chart will correspond to a different amplitude of an angle according to the number of variables, since it shows the values in a relative way, as parts of a whole. The same logic is applied for area charts, but instead of using the amplitude of angles, it uses the width of the drawn rectangles The total width of all the rectangles is fixed, as are the 360º degrees for the pie charts.

Therefore, we can consider this a limitation of our model. With 10 variables, a comparison between visualizations might be unfair since the model might not be capable to properly analyze the chart in order to assess it for visual perception. However, one could also argue that the fact that the model is not capable of learning to read such charts shows that they present a poor visualization approach. In order to settle this argument, we would have to conduct similar experiments with humans to conclude if it is a matter of bad performance from the visualization technique or in fact a limitation of the ML model from being overwhelmed by too many variables.

Color
Regarding the third and final variation explored, we present the results comparing the performance from our base experiments against similar ones but with diverse datasets in terms of color variations. These are shown in Table 5.9. Since the number of variables in the charts for this experiment is always 5, there were no changes needed in terms of data generation parameters. The only difference is that during the dataset generation we alternate between each color variation, so as to achieve a perfectly balanced set of examples.

Table 5.9: Comparison of results from experiments for charts with and without color variations. Values expressed in MLAE with standard deviation, as well as the difference between the results for each chart type.

Chart Type	Normal	With variations	Difference
	1.29 ± 0.11	1.55 ± 0.08	-0.26
	2.12 ± 0.05	2.81 ± 0.15	-0.69
	1.09 ± 0.05	1.24 ± 0.09	-0.15
	1.06 ± 0.09	1.28 ± 0.06	-0.22
	1.72 ± 0.07	2.13 ± 0.09	-0.41

As we can see, there is a slight decrease in performance from the models trained with more diverse datasets. However, this might not necessarily reflect a loss in the performance of the model overall. As we mentioned, the way we chose to test this variation was to run the same experiments but with a dataset of different color settings. By doing so, the model has to adapt to those changes in addition to learning the regression process, having an increase in the error. Additionally, as the dataset is diverse in terms of color settings, this means that it has a smaller number of samples of each type of chart, which can also affect the training. By looking at the results for line charts and line charts filled, we are able to tell a small difference that can potentially support this claim. Those two examples only have two variations, while the remaining ones have four. Nevertheless, this issue could possibly be overcome by training the models with a larger dataset. In general, there was only a very slight loss in performance. The reason for this is that the model can easily overcome the color changes by simply focusing on the contrast of the lines rather than on the actual colors.

In order to visualize the impact and difference in performance from the variations analyzed throughout this section, we present the following plots in Fig. 5.15. Here we graphically present the MLAE results for each experiment: the number of variables variation is represented through the blue line plot, where we can see the difference in the error as the number of variables increases, while the color and rotation variations are represented by the orange and green dots, respectively. The missing green dot in the pie chart sub-plot is due to the fact that we did not conduct that experience as it would be pointless.

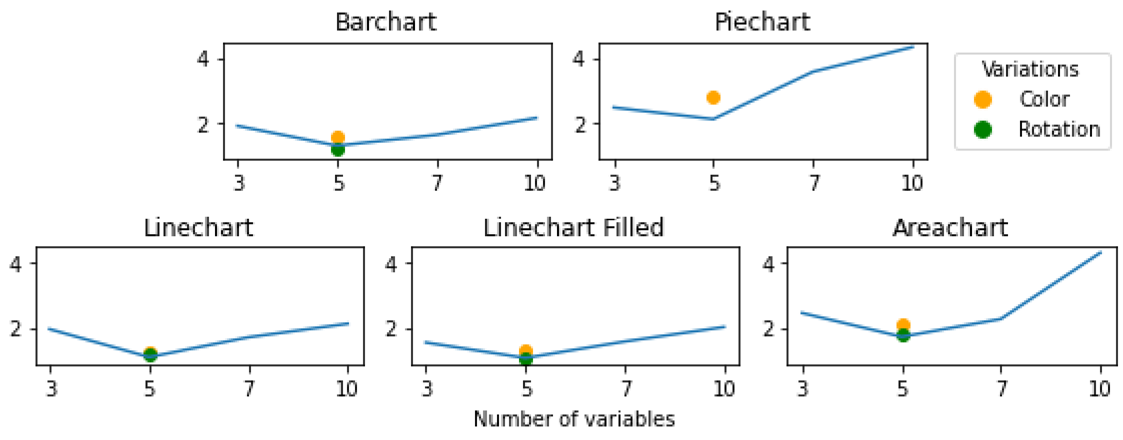


Figure 5.15: Comparison between the performance of the different variations, for each chart type. Blue line plot shows the effect of the number of variables along the horizontal axis, while green and orange dots represent the experiments with rotation and color variations, respectively. Values expressed in MLAE.

Regarding the color and rotation variations, we can see that they had little impact on the performance of the model in general. For pie charts and area charts, it appears that the color variation affected more its performance, presenting a noticeable decrease. This difference might simply be related to those visualization techniques already being weaker ones, and the variation accentuating their flaws since the increased diversity of the dataset can lead to a loss in performance, as we previously explained. More tests would be needed to confirm this, though. However, in general, we can accept that the model has the capability of dealing well with rotated figures and with different color settings (not the same as different colors, as we still only tested black and white charts).

As far as the number of variables represented goes, we can see that no other option was able to outperform the experiments with five values per chart. We do not really have an explanation for this other than the possibility that this number can simply be the “sweet spot” for the model in order to learn the visual regression task: enough variables to allow for a proper comparison, but not too many as to overwhelm the model with such a complex task, leading to a loss of precision. If we were to run these experiments with the remaining number of variables in between the ones tested (4, 6, 8 and 9), we could draw a more complete line showing the performance and confirming these assumptions. Additionally, by defining a threshold for a maximum MLAE allowed, we can more precisely define the limitation for the number of variables the model can handle - as of this moment we can only exclude 10 variables for pie charts and area charts, and possibly 7 variables for pie charts as well, as valid capabilities for the model.

5.4 Summary

Throughout this chapter we analyzed and discussed the results obtained from our experiments. These ranged from base tests, used to compare the performance of different CNN architectures to attempting to discover the limits in performance of the model established as the most suitable. For the base experiments, we went beyond simply comparing the average MLAE values obtained from each experiment. We looked into the distribution of individual, absolute errors associated with each predicted sample, to get a better view of the performance of each model. We analyzed this error both in its raw, normalized form, but also in its more readable, non-normalized version as well. This de-normalization process also allowed us to visualize the impact of the estimation error obtained, by comparing the original charts with the ones generated from the perceived values. In order to attempt to understand the strengths and weaknesses of our models, we took a look into the best and worst estimations performed by them, allowing for a few simple, yet useful conclusions on that topic. We ended up choosing the XCEPTION trained from scratch as the most suitable model for this task and also compared the performance of different visualization techniques based on the results obtained from that CNN.

After that, we put this chosen model to the test even further, attempting to find its limits in terms of performance, and when we did not find any, we were able to assert another capability to the list. We conducted new experiments in order to test how well the model could adapt to three different types of variations in the charts: orientation, which the model was able to easily deal with, number of variables, which presented itself as a challenge, especially associated with some of the visualization techniques, and color, which even though it also presented itself as a slight challenge, we believe we can easily overcome it.

Now that we analyzed all these results, we believe to have the necessary information to draw some overall conclusions regarding not only our chosen model but also the evaluation method used. This reflection is done in the next chapter.

Chapter 6

Conclusion

This chapter concludes this dissertation, summarizing our steps taken from the beginning up to this point and presenting our final remarks regarding what was achieved. We started by presenting an overview of the problem and the motivation behind the research, proposing a method for evaluating data visualizations through graphical perception, the ability to decode the information represented in a chart. We then conducted a thorough literature review in order to understand the basics of data visualization, its evaluation and existing approaches applying Machine Learning (ML) techniques for the purpose of assessing the quality of data charts. From that, we were able to conclude that many studies have tested different strategies to perform this evaluation, but very few according to the criteria we intend to follow, hence leaving a large gap for us to fill with our own tests and conclusions while having a starting basis for us to stand on.

For this, we developed a software-based environment to conduct experiments regarding our proposed method, in order to test its viability, with various options for visualization techniques. We started by testing and analyzing the performance of different models, before going into a set of experiments designed to determine the capabilities of our proposed model for the challenge, presented in the previous chapter. Based on our obtained results, we are able to draw a set of conclusions, presented in the following sections. We start by presenting the key points to be taken from our work, followed by the limitations found throughout the analysis, which also include non-confirmed hypothesis, and finally by suggesting possible steps to take in the future to continue building on top of our developed work.

6.1 Key takeaways

In order to discuss the key takeaways from our work, it is important to understand our intention regarding what we have accomplished. We do not see the proposed model and its results as a perfect answer to our problem. Instead, we look at them as building blocks for a project with a larger scope than this dissertation. Just as we have built our work from previously presented research on this topic, we hope that others will do the same with ours. In that sense, we believe we succeeded with what we have done. We were able to expand on the little related work found, by testing more models and more chart types, as well as more experiments and much deeper analysis. This way, we have successfully made a contribution to the field of ML for Visualization (ML4VIS), which is also a success criterion for our work.

The first takeaway from our research is that our method works. The task of retrieving the values encoded in a chart produces different performance results according to the visualization technique used to represent them, providing a single absolute metric that allows their comparison. However, this fact, by itself, can be useless if the results do not follow those from human performance. How good would be an evaluation method that classifies something as good when in reality it is not? Our model could potentially indicate that a certain visualization was the best but then for the end user, a human, would be terrible. Fortunately, it does not seem to have gone that way. As we mentioned in chapter 5, the results obtained mostly follow the theoretical foundation stipulated and tested by Cleveland and McGill [1984] regarding graphical perception. By ordering the different visualization types according to the corresponding obtained Mean Logarithmic Absolute Error (MLAE) from the experiments, we can obtain a ranking by placing the line chart at the top, followed by the bar chart and then the area and pie chart, in this order. The fact that the area chart performed better than the pie chart was the only exception found, but still far behind the others, which is the expected ranking.

One could argue that this comparison of different visualization types can be unfair due to the characteristics of the chosen ML model, in this case, the XCEPTION Convolutional Neural Network (CNN). However, just as the characteristics and capabilities of the network are the same regardless of which visualization technique it was trained on, the same principle applies to the human eyes and brain. The way we perceive and process the visual elements of visualization leads to some chart types being more suitable than others. This means that as long as our chosen model follows the same performing patterns as that of humans, the comparison between different visualization techniques is fair and valid.

In addition to our visualization ranking, we can also conclude a few other aspects. We have concluded that orientation has no influence on the performance of the model and that the color setting has little or no influence as well. However, we are aware that in order to declare it to be invariant to these aspects, we would need to train it and test it with a properly varied dataset that includes samples representing these variations. If the model's performance does not change, we can consider it invariant. Moreover, despite having a difference in the resulting error, we can also conclude the model can adapt to different numbers of variables represented in the charts, within certain boundaries. Finally, it also seems that for line charts the filling of the area underneath the plot helps to better perceive the represented values. We can only conclude this for line charts since we did not specifically test this hypothesis for the remaining visualization techniques.

Despite these conclusions taken from our results, there are some issues identified and other uncertain ones that should also be discussed in order to better reflect on the work developed, which are presented in the following section.

6.2 Limitations

Even though our method works, we do not claim our proposed model to be perfect, or even the best possible option. There are aspects in which the model appears to have some limitations, besides others that were not even tested.

First of all, the biggest uncertainty is that despite the obtained results following some known patterns of human performance, there is no fully proven correlation between these two. For that, we would need to conduct experiments with human users in order to produce comparative metrics. For example, by analyzing the worst estimations from the model, we concluded that it generally struggles with sudden transitions and highly unequally distributed values, which can simply not be that much of a struggle for a person.

Additionally, we established that there is an upper limit on the number of variables that our model is capable of properly process in order to perceive its represented values (through regressing their proportions). We have shown that for area and pie charts the performance strongly declines with 10 variables, losing most of the precision, compared to the baseline of 5. While the same cannot be said for the remaining visualization types, their comparison might no longer be fair, defining this as a limitation of our model.

Furthermore, we must also note other aspects not tested, which could possibly result in other limitations. All the charts used for training and testing the model throughout the work are images with 100 by 100 dimensions. There are certainly upper and lower limits as to what the model can handle, but we were unable to establish them in this dissertation. However, that limitation can possibly be overcome by resizing the images containing charts to dimensions where the model performs well.

Finally, we have also previously mentioned the lack of tests regarding the different colors used. We have experimented with black and white (gray tones) images. While this may present itself as a limitation, we believe that as long as there is enough contrast between the drawn chart and the background, color should not be an issue for the model.

In conclusion, we have established certain limitations and other possible ones regarding our model. Due to the length of our work and the resources available, we believe to have achieved what was within our reach. Nevertheless, we present in the next section what we would suggest as the next steps for this topic.

6.3 Future Work

Since we do not have all the answers or solutions to our problems, we propose in this section the following steps to take in the future in a way to carry on the research on the topic of ML for evaluation of visualization models.

Firstly, the most important next step would be to conduct similar experiments as the ones described with human users, as it was mentioned many times already. This is the only way to assess the correlation between a proposed ML model and the standard of human performance, very important to understand the viability of our method for a real world scenario.

Another obvious suggestion is to test other ML models, namely other CNN. Other architectures such as CORNet [Kubilius et al., 2018], for example, or InceptionV3 [Szegedy et al., 2015] are definitely worth testing. Other more advanced models, such as YOLOR [Wang et al., 2021] might also be appropriate but as they are more complex, they might be harder to setup for our task. Different approaches, such as Autoencoders (AE) are also worth testing: by calculating the error associated with the reconstruction of a chart, one can measure how well the model perceives the represented values, therefore assessing the corresponding visualization technique in regards to graphical perception.

A very useful feature for a system employing our trained models would be a classifier capable of identifying the chart type shown in an image. That way, the system could automatically choose the appropriate model to assess that visualization, instead of having to manually specify which technique is used. The same logic could be used for other aspects, such as orientation or color setting, in order to automatically apply the correct models to their evaluation.

One thing to notice about the visualization models tested in this dissertation is that they all convey variables with only 2 dimensions. In other words, in each chart we represent variables A, B, C, D and E, for example, and a single respective value for each one. Many visualizations, more advanced, can represent more than one value for each variable simultaneously. Such visualization techniques are also worth experimenting with in order to better test the proposed ML models.

Finally, while not directly related to the task of evaluating visualization models, we also propose some improvements regarding the Application Programming Interface (API) for integration: an endpoint to start the training for a new model and upgrading the evaluation endpoint mentioned in section 4.5 for multiple images simultaneously. Regarding the latter, we also suggest the possible option of loading all the available/stored models as the API starts instead of loading them at run-time as they are needed, since this step might take a few seconds, causing a relatively large response time for the evaluation endpoint.

Following these steps, one can build upon what we have achieved and presented throughout this dissertation, improving our contribution towards a promising path.

References

Karan Aggarwal, Maad M. Mijwil, Sonia, Abdel-Hameed Al-Mistarehi, Safwan Alomari, Murat Gök, Anas M. Zein Alaabdin, and Safaa H. Abdulrhman. Has the future started? the current growth of artificial intelligence, machine learning, and deep learning. *Iraqi Journal For Computer Science and Mathematics*, 3(1):115–123, Jan. 2022. doi: 10.52866/ijcsm.2022.01.01.013. URL `https://journal.esj.edu.iq/index.php/IJCM/article/view/100`.

Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

Ian Baddock. Why you need to stop using pie charts to present data, Nov 2021. URL `https://syntagium.com.au/why-you-need-to-stop-using-pie-charts-when-presenting-visual-analytics/`.

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, and D. A. Keim. Quality metrics for information visualization. *Computer Graphics Forum*, 37(3):625–662, 2018. doi: https://doi.org/10.1111/cgf.13446. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13446`.

John P. Bertacco. International yearbook of cartography, 1974, volume xiv, edited by g.m. kirschbaum, and k-h. meine, kirschbaum verlag, bonn-bad godesberg, 1974. 175 x 250 mm, 182 pages, 4 tables, 55 figures, 4 plates (colour). *Cartography*, 9(2):117–117, 1975. doi: 10.1080/00690805.1975.10437878. URL `https://doi.org/10.1080/00690805.1975.10437878`.

J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. UMI Research Press, 1967/1983. ISBN 9780835735322. URL `https://books.google.pt/books?id=OJu8bwAACAAJ`.

G. Bonaccorso. *Machine Learning Algorithms*. Packt Publishing, 2017. ISBN 9781785884511. URL `https://books.google.pt/books?id=_-ZDDwAAQBAJ`.

Chris Bryan, Kwan-Liu Ma, and Jonathan Woodring. Temporal summary images: An approach to narrative visualization via interactive annotation generation and placement. *IEEE Transactions on Visualization and Computer Graphics*, 23(1): 511–520, 2017. doi: 10.1109/TVCG.2016.2598876.

Stuart Card, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision To Think*. 01 1999a. ISBN 978-1-55860-533-6.

Stuart K Card, Jock Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999b.

Sheelagh Carpendale. *Evaluating Information Visualizations*, pages 19–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-70956-5. doi: 10.1007/978-3-540-70956-5_2. URL `https://doi.org/10.1007/978-3-540-70956-5_2`.

Paulo Chagas, Rafael Akiyama, Aruanda Meiguins, Carlos Santos, Filipe Saraiva, Bianchi Meiguins, and Jefferson Morais. Evaluation of convolutional neural network architectures for chart image classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018. doi: 10.1109/IJCNN.2018.8489315.

Francois Chollet. The keras blog, May 2016. URL `https://blog.keras.io/building-autoencoders-in-keras.html`.

François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. ISSN 01621459. URL `http://www.jstor.org/stable/2288400`.

Wenjing Dai, Meng Wang, Zhibin Niu, and Jiawan Zhang. Chart decoder: Generating textual and numeric information from chart images automatically. *Journal of Visual Languages & Computing*, 48:101–109, 2018. ISSN 1045-926X. doi: https://doi.org/10.1016/j.jvlc.2018.08.005. URL `https://www.sciencedirect.com/science/article/pii/S1045926X18301162`.

Kajaree Das and Rabi Narayan Behera. A survey on machine learning: concept, algorithms and applications. *International Journal of Innovative Research in Computer and Communication Engineering*, 5(2):1301–1309, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis. Efficient recommendation of aggregate data visualizations. *IEEE Transactions on Knowledge and Data Engineering*, 30(2):263–277, 2018. doi: 10.1109/TKDE.2017.2765634.

Michael Friendly. *A Brief History of Data Visualization*, pages 15–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-33037-0. doi: 10.1007/978-3-540-33037-0_2. URL `https://doi.org/10.1007/978-3-540-33037-0_2`.

Xin Fu, Yun Wang, Haoyu Dong, Weiwei Cui, and Haidong Zhang. Visualization assessment: A machine learning approach. In *2019 IEEE Visualization Conference (VIS)*, pages 126–130, 2019. doi: 10.1109/VISUAL.2019.8933570.

L. Giovannangeli, R. Bourqui, R. Giot, and D. Auber. Toward automatic comparison of visualization techniques: Application to graph visualization. *Visual Informatics*, 4(2):86–98, 2020. ISSN 2468-502X. doi: https://doi.org/10.1016/j.visinf.2020.04.002. URL `https://www.sciencedirect.com/science/article/pii/S2468502X20300140`. PacificVis 2020 Workshop on Visualization Meets AI.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Daniel Haehn, James Tompkin, and Hanspeter Pfister. Evaluating 'graphical perception'with cnns. *IEEE Transactions on Visualization and Computer Graphics*, 25 (1):641–650, 2018.

Hammad Haleem, Yong Wang, Abishek Puri, Sahil Wadhwa, and Huamin Qu. Evaluating the readability of force directed graph layouts: A deep learning approach. *IEEE Computer Graphics and Applications*, 39(4):40–53, jul 2019. doi: 10.1109/mcg.2018.2881501. URL `https://doi.org/10.1109%2Fmcg.2018.2881501`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Jeffrey Heer and Michael Bostock. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 203–212, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589299. doi: 10.1145/1753326.1753357. URL `https://doi.org/10.1145/1753326.1753357`.

Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *Commun. ACM*, 53:59–67, 06 2010. doi: 10.1145/1743546.1743567.

Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

Arne Holst. Total data volume worldwide 2010-2025. *Statista. url: https://www.statista.com/statistics/871513/worldwide-data-created/(accessed on 09/27/2021)*, 2021.

Kevin Hu, Neil Gaikwad, Michiel Bakker, Madelon Hulsebos, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. Viznet: Towards a large-scale visualization learning and benchmarking repository. In *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI)*. ACM, 2019.

Kevin Z. Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo. Vizml: A machine learning approach to visualization recommendation, 2018. URL `https://arxiv.org/abs/1808.04819`.

Petra Isenberg, Torre Zuk, Christopher Collins, and Sheelagh Carpendale. Grounded evaluation of information visualizations. In *Proceedings of the 2008 Workshop on BEyond Time and Errors: Novel EvaLuation Methods for Information*

Visualization, BELIV '08, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580166. doi: 10.1145/1377966.1377974. URL `https://doi.org/10.1145/1377966.1377974`.

Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006.

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering, 2018. URL `https://arxiv.org/abs/1801.08163`.

Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Akos Kadar, Adam Trischler, and Yoshua Bengio. Figureqa: An annotated figure dataset for visual reasoning, 2017. URL `https://arxiv.org/abs/1710.07300`.

Younghoon Kim and Jeffrey Heer. Assessing effects of task and data distribution on the effectiveness of visual encodings. *Computer Graphics Forum*, 37:157–167, 06 2018. doi: 10.1111/cgf.13409.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL `https://doi.org/10.1145/3065386`.

Jonas Kubilius, Martin Schrimpf, Aran Nayebi, Daniel Bear, Daniel L. K. Yamins, and James J. DiCarlo. Cornet: Modeling the neural mechanisms of core object recognition. *bioRxiv*, 09/2018 2018. doi: https://doi.org/10.1101/408385. URL `https://www.biorxiv.org/content/10.1101/408385v1.full.pdf`.

Heidi Lam, Enrico Bertini, Petra Isenberg, Catherine Plaisant, and Sheelagh Carpendale. Seven Guiding Scenarios for Information Visualization Evaluation. Research Report 2011-992-04, 2011. URL `https://hal.inria.fr/hal-00723057`. Superseded by and improved in a follow-up journal article.

Doris Jung-Lin Lee, Himel Dev, Huizi Hu, Hazem Elmeleegy, and Aditya Parameswaran. Avoiding drill-down fallacies with vispilot: Assisted exploration of data subsets. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI '19, page 186–196, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362726. doi: 10.1145/3301275.3302307. URL `https://doi.org/10.1145/3301275.3302307`.

Allen Yilun Lin, Joshua Ford, Eytan Adar, and Brent Hecht. Vizbywiki: Mining data visualizations from the web to enrich news articles. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 873–882, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186135. URL `https://doi.org/10.1145/3178876.3186135`.

Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 101–112, 2018. doi: 10.1109/ICDE.2018.00019.

Yuxin Ma, Anthony K. H. Tung, Wei Wang, Xiang Gao, Zhigeng Pan, and Wei Chen. Scatternet: A deep subjective similarity model for visual analysis of scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 26(3): 1562–1576, 2020. doi: 10.1109/TVCG.2018.2875702.

A.M. MacEachren. *How Maps Work: Representation, Visualization, and Design*. Guilford Publications, 1995. ISBN 9780898625899. URL `https://books.google.pt/books?id=v4GyQgAACAAJ`.

Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, apr 1986. ISSN 0730-0301. doi: 10.1145/22949.22950. URL `https://doi.org/10.1145/22949.22950`.

Timmy Manning, Roy Sleator, and Paul Walsh. Biologically inspired intelligent decision making. *Bioengineered*, 5, 12 2013. doi: 10.4161/bioe.26997.

Joseph E. McGrath. Methodology matters: Doing research in the behavioral and social sciences. In RONALD M. BAECKER, JONATHAN GRUDIN, WILLIAM A.S. BUXTON, and SAUL GREENBERG, editors, *Readings in Human–Computer Interaction*, Interactive Technologies, pages 152–169. Morgan Kaufmann, 1995. ISBN 978-0-08-051574-8. doi: https://doi.org/10.1016/B978-0-08-051574-8.50019-4. URL `https://www.sciencedirect.com/science/article/pii/B9780080515748500194`.

Isabel Meirelles. *Design for information : an introduction to the histories, theories, and best practices behind effective information visualizations*. Rockport Publishers, Beverly, MA, 2013. ISBN 9781610589482.

T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN 9780071154673. URL `https://books.google.pt/books?id=EoYBngEACAAJ`.

Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):438–448, 2019. doi: 10.1109/TVCG.2018.2865240.

Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, 2009. doi: 10.1109/TVCG.2009.111.

Catherine Plaisant. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, page 109–116, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138679. doi: 10.1145/989863.989880. URL `https://doi.org/10.1145/989863.989880`.

Jorge Poco and Jeffrey Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. *Comput. Graph. Forum*, 36(3):353–363, jun 2017. ISSN 0167-7055. doi: 10.1111/cgf.13193. URL `https://doi.org/10.1111/cgf.13193`.

Evgheni Polisciuc. *Thematic Cartography for Adaptive Visualization Systems*. PhD thesis, 00500:: Universidade de Coimbra, 2021.

Robert Roth. *Visual Variables*, pages 1–11. 01 2017. doi: 10.1002/9781118786352.wbieg0761.

W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, page 328–338, Washington, DC, USA, 1987. IEEE Computer Society Press. ISBN 0897912160.

Sumit Saha. A comprehensive guide to convolutional neural networks - the eli5 way, Nov 2022. URL `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

Bahador Saket, Alex Endert, and Çağatay Demiralp. Task-based effectiveness of basic visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(7):2505–2512, 2019. doi: 10.1109/TVCG.2018.2829750.

Pavan Sanagapati. What is pooling in deep learning?: Data science and machine learning, 2017. URL `https://www.kaggle.com/questions-and-answers/59502`.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning, 2017. URL `https://arxiv.org/abs/1706.01427`.

Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M. Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5):1285–1298, 2016. doi: 10.1109/TMI.2016.2528162.

Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018. doi: 10.1109/ICCUBEA.2018.8697857.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

Binbin Tang, Xiao Liu, Jie Lei, Mingli Song, Dapeng Tao, Shuifa Sun, and Fangmin Dong. Deepchart: Combining deep convolutional networks and deep belief networks in chart classification. *Signal Processing*, 124:156–161, 2016. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2015.09.027. URL `https://www.sciencedirect.com/science/article/pii/S0165168415003291`. Big Data Meets Multimedia Analytics.

Satoshi Tsutsui and David J. Crandall. A data driven approach for compound figure separation using convolutional neural networks. *CoRR*, abs/1703.05105, 2017. URL `http://arxiv.org/abs/1703.05105`.

Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. You only learn one representation: Unified network for multiple tasks, 2021.

Qianwen Wang, Zhutian Chen, Yong Wang, and Huamin Qu. A survey on ML4vis: Applying machine learning advances to data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):5134–5153, dec 2022. doi: 10.1109/tvcg.2021.3106142. URL `https://doi.org/10.1109%2Ftvcg.2021.3106142`.

Sun-Chong Wang. *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA, 2003. ISBN 978-1-4615-0377-4. doi: 10.1007/978-1-4615-0377-4_5. URL `https://doi.org/10.1007/978-1-4615-0377-4_5`.

Jarke Wijk. Evaluation: A challenge for visual analytics. *Computer*, 46:56–60, 07 2013. doi: 10.1109/MC.2013.151.

Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Towards a general-purpose query language for visualization recommendation. In *ACM SIGMOD Human-in-the-Loop Data Analysis (HILDA)*, 2016a. URL `http://idl.cs.washington.edu/papers/compassql`.

Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016b. doi: 10.1109/TVCG.2015.2467191.

Aoyu Wu, Wai Tong, Tim Dwyer, Bongshin Lee, Petra Isenberg, and Huamin Qu. Mobilevisfixer: Tailoring web visualizations for mobile phones leveraging an explainable reinforcement learning framework, 2020. URL `https://arxiv.org/abs/2008.06678`.

Aoyu Wu, Yun Wang, Xinhuan Shu, Dominik Moritz, Weiwei Cui, Haidong Zhang, Dongmei Zhang, and Huamin Qu. Ai4vis: Survey on artificial intelligence approaches for data visualization, 2021. URL `https://arxiv.org/abs/2102.01330`.

Peiying Zhang, Chenhui Li, and Changbo Wang. Viscode: Embedding information in visualization images using encoder-decoder network. *CoRR*, abs/2009.03817, 2020. URL `https://arxiv.org/abs/2009.03817`.

Torre Zuk, Lothar Schlesier, Petra Neumann, Mark S. Hancock, and Sheelagh Carpendale. Heuristics for information visualization evaluation. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, page 1–6, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595935622. doi: 10.1145/1168149.1168162. URL `https://doi.org/10.1145/1168149.1168162`.