UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Enhancing Dimensionality Reduction Techniques for Deep Neural Network Visualization

Gabriel Dias Cantareira

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Gabriel Dias Cantareira

Enhancing Dimensionality Reduction Techniques for Deep Neural Network Visualization

Thesis submitted to Instituto de Ciências Matemáticas
e de Computação – ICMC-USP – in accordance with
the requirements of the Computer and Mathematical
Sciences Graduate Program, for the degree of
Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Fernando Vieira Paulovich

USP – São Carlos
February 2021

Gabriel Dias Cantareira

**Aprimorando Técnicas de Redução de Dimensionalidade
para Visualização de Redes Neurais Profundas**

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Fernando Vieira Paulovich

USP – São Carlos
Fevereiro de 2021

ACKNOWLEDGEMENTS

I am not good at writing these parts. I have met many people and experienced a lot of things in the past few years that I am sure contributed to my work in one way or another, but I will try to keep it simple and express my thanks to the following:

My family, for always being there.

My friends, for being patient.

My supervisor, for offering more support than I deserved.

The doctors that put me back together, so I could finish this (and also not die).

Research colleagues, for discussions and inspiration.

The University, for making it possible.

This research was conducted with financial support from CAPES and FAPESP(2015/08118-6, 2017/08817-7).

*“Carry on my wayward son,
There will be peace when you are done.”
(Kansas)*

RESUMO

CANTAREIRA, G. D. **Aprimorando Técnicas de Redução de Dimensionalidade para Visualização de Redes Neurais Profundas**. 2021. 101 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Redes neurais profundas tem demonstrado resultados impressionantes em uma grande variedade de aplicações computacionais nos últimos anos, sendo responsáveis por diversos avanços em tecnologia. No entanto, testar e entender os mecanismos internos destes modelos é uma tarefa complexa, uma vez que o número de variáveis envolvidas em cada decisão pode chegar aos milhões. Visando resolver este problema, pesquisadores dos campos de Visual Analytics e Explainable Artificial Intelligence tem proposto várias abordagens para visualizar e explicar diferentes aspectos de modelos de redes neurais. Uma destas abordagens é o uso de técnicas de redução de dimensionalidade para a visualização do comportamento de camadas ocultas, empregado com relativo sucesso na literatura. Porém, aplicar tais técnicas neste contexto implica em certas limitações que precisam ser tratadas, principalmente no que diz respeito à comparação visual entre múltiplas projeções multidimensionais. Adicionalmente, certas características particulares deste domínio podem ser utilizadas para gerar visualizações especializadas mais informativas. Esta tese de doutorado mostra o processo de investigação de problemas e oportunidades em visualização de redes neurais utilizando redução de dimensionalidade e o desenvolvimento de métodos de visualização aprimorados para este domínio.

Palavras-chave: Deep Learning, Visualização de Redes Neurais, Explainable Artificial Intelligence, Redução de Dimensionalidade, Projeções Multidimensionais.

ABSTRACT

CANTAREIRA, G. D. **Enhancing Dimensionality Reduction Techniques for Deep Neural Network Visualization**. 2021. 101 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Deep Neural Networks have achieved impressive results in a wide range of applications over the past few years, being responsible for many advances in computational technology. However, debugging and understanding the inner workings from these models is a complex task, as there are often millions of variables involved in every decision. Aiming to solve this problem, researchers from the fields of Visual Analytics and Explainable Artificial Intelligence have proposed several approaches to visualize and explain different aspects of DNN models. One of such approaches is the use of Dimensionality Reduction techniques for hidden layer output visualization, which has been employed in literature with relative success. However, there are certain limitations to applying these techniques in this context that need to be addressed, such as the visual comparison between multiple multidimensional projections. Furthermore, the particular characteristics of this domain can be taken into account to generate specialized visual representations that are more informative. This doctorate thesis shows the process of investigating problems and opportunities in DNN visualization using dimensionality reduction and the development of improved visualization methods for this domain.

Keywords: Deep Learning, Neural Network Visualization, Explainable Artificial Intelligence, Dimensionality Reduction, Multidimensional Projections.

LIST OF FIGURES

Figure 1 – Projections p_0 and p_1 of two feature sets using standard UMAP (no alignment) and our model (MF-UMAP). The feature sets correspond to activation data from two hidden layers of a CNN. These layers provide similar outputs, so the variation in the projections should be small. However, the unaligned projections cause groups to change places (e.g., the orange group) and may result in misinterpretation.	30
Figure 2 – Projections of activations of 4 sequential layers of a CNN. The shapes identified by the first two convolutional layers are similar for specific classes, as they are not fully separated. The last two organize this information and divide the space among classes.	34
Figure 3 – Comparison between activations of the last hidden dense layers of MLP and a CNN using MF-UMAP and Dt-SNE. The CNN network separates data more explicitly, to the point of overfitting, but only MF-UMAP is capable of showing areas where separation improved (A and B).	35
Figure 4 – Last hidden layer activations from two ANNs using CIFAR-10 data. a) CNN and b)VGG16. While their accuracy results are somewhat similar (0.70 and 0.74, respectively), there are clear differences in how each network perceives data, and how the extra complexity present in the VGG16 model can generate more refined, concentrated outputs.	36
Figure 5 – Overview of the learning space vector field generation. From a DNN model, we obtain sequential data in the form of hidden layer activations. We compute projections from this highly multivariate data, aligning them to ensure that obtained projections are synchronized. Differences between each pair of projections in the sequence (transition steps) are turned into trajectories and then processed to obtain vector fields representing changes in data. Finally, everything is combined to compose distinct visual representations of the neural network.	42

Figure 6 – Example of visual representations for projection, trajectory, and vector field components stacked together. The vector field representation overviews the whole flow of the data inside a network, supporting the analysis of instance grouping over time. The scatterplot metaphor, used to represent a projection, enables the analysis of specific instances at a particular moment in time, and trajectory bundling can be used to derive insights about instances' behavior over time. Together, these representations define a powerful visual metaphor, providing abstract and detailed views of the dynamics of neural networks. . 46

Figure 7 – Outputs from the last hidden layer of a 4-layer MLP were gathered during training, after 0, 1, 3, and 20 epochs. Projected data refer to a sample of MNIST test set, with 2000 instances. Instance projected position at each epoch define trajectories that were used to approximate vector fields, as shown on the right side of the figure. A clear expansion pattern can be observed, as instances move outwards to form groups as the network is trained. 47

Figure 8 – Outputs from the last hidden layer of a 4-layer MLP after 20 training epochs, using MNIST test data input, projected over streamlines. Streamlines are calculated based on three other outputs from the same layer during training, at epochs 0, 1, and 3. Activity between projections is turned into a vector field, which is then used to generate streamlines that offer additional insight into data flow during training without displaying points from other projections. For streamlines, color intensity represents flow speed (L2 norm of vectors) and arrows indicate direction. 48

Figure 9 – Trajectory data from the same outputs as the Figure 8 (last hidden layer of MLP after 20 epochs using MNIST test data as input). Lines start from instance positions at epoch 0 and end in instance positions at epoch 20. Control points were set along the streamlines defined previously, attracting lines near them into bundles. 49

Figure 10 – When visualizing trajectories, it is possible to highlight the ones that attained high error values when approximating the vector field, in search of outliers. (a) shows trajectories with the error above a certain threshold (0.1) highlighted in red, while all others are shown in black; (b) filters the projection to only instances corresponding to label "7", which seem to contain a few outliers. Data points were plotted again and colored according to which feature set they were taken from, so trajectory direction can be better understood. In this case, the highlighted trajectories move through long distances, at times in the exact opposite way of the vector field. 50

Figure 11 – A single outlier trajectory is selected for examination. This data point was at the middle area at epoch 0, moved to an area containing many examples labeled as "2" at epoch 1, moved again to an area containing examples labeled "1" at epoch 3, and finally moved to another area, which contains a group of examples labeled "7", at epoch 20. We can then examine the instance to search for reasons behind this atypical behavior.	51
Figure 12 – Projected sequential output data from six hidden layers of pre-trained MobileNetV2 using 1000 ImageNet inputs. Projections are shown in left-to-right, upper-to-lower order. It is possible to notice that, while still concentrated in the middle area in the first upper projections, points are slowly separated into classes before finally expanding in the last two projections. On the right side is the generated vector field. A high concentration of movement in all directions in the middle area makes the resulting field rather muted, but its intensity increases in outer areas of the image.	52
Figure 13 – (a) projection from the last hidden layer of MobileNetV2 (global average pooling layer) over streamlines from movement trajectories of previous layers. It is possible to see paths that led from a central position to the positions occupied at the current projection, for all points; (b) trajectories of points over six layer outputs in the same network model. There is a high amount of clutter in the central section due to the resulting projections of the first layers, but certain movement patterns can be identified. Projected data are obtained using 1000 ImageNet data samples.	53
Figure 14 – Proposed framework overview. Original and adversarial examples are selected as input for a DNN model. Users then use the Architecture View (A) to select sequential layers to visualize. Input examples are then mapped in the Flow View (B) to (black) lines representing the selected layers' responses (outputs). At a certain point, the distance between these lines diverges drastically, indicating the turning point in the model for the adversarial example misclassification. Users can then select the layer in which this change happens for details using the Layer View (C), which shows activation heatmaps and detailed information for each unit in the layer so that the precise units and input features being exploited by an adversarial attack can be pinpointed. It is also possible to return to the previous steps to refine selections and observe different model parts.	56
Figure 15 – <i>Flow View</i> overview. Input data is processed by the model and hidden outputs of selected layers are transformed into 1-D arrays using a dimensionality reduction approach, resulting in <i>Activation Profiles</i> . <i>Activation Profiles</i> are then used to build visualizations, enabling the comparisons of regular and adversarial examples considering the model's perception.	63

Figure 16 – <i>Layer View</i> overview. Outputs from a selected layer are subjected to reduction to obtain a single output equivalent, representing activation data. This data is displayed using heatmap visualizations to highlight units with interesting behavior. Units can be selected and displayed using convolutional filter visualization and highlighting weights in their connections. Activation data can also be exported for further statistical analysis.	65
Figure 17 – Background data visualization for the CNN-MNIST case study. The upper diagram shows the Architecture View for the model, indicating which layers are selected for view. The line plot shows lines for all background data objects, colored by their predicted label. Class separation may occur at different layers. Many objects in classes “9” and “4” (brown and orange, respectively) only get properly separated at the fourth convolutional layer, while the second layer already separates many objects in class “1” (cyan). Class fingerprints, shown on the right, display the accumulation of points at each segment of the visual space, highlighting condensed areas with brighter colors.	68
Figure 18 – Lines representing original image x_0 and adversarial image x_0^{adv} being interpreted by the CNN-MNIST model. Background data representing classes “0” and “9” is also shown. The two images are perceived as relatively similar until layer $conv2d_2$, when separation occurs and each example merges into a different class. This <i>diverging point</i> highlights layer $conv2d_2$ as a point of interest.	69
Figure 19 – Visualization for layers $conv2d_1$ and $conv2d_2$ of the CNN-MNIST model. The first column identifies each unit in the layer. Second and third columns show activations for x_0 and x_0^{adv} from the two observed layers, and the fourth column shows the difference in activations between the two. Finally, the fifth column visualizes all filters in each layer using activation maximization. Activations from x_0 and x_0^{adv} at the first layer are still relatively similar but show much more variation in the second. Many activations appear to be stronger on the center-right side for x_0^{adv} , and stronger on the lower-left side for x_0	70
Figure 20 – This graph shows the difference between outputs at each unit, for the two examples as well as the two relevant classes. The adversarial example accentuates certain differences in activation (highlighted in red), which can be translated into visual features in Figure 19, but there are many large differences in activation that do not match class patterns.	71

- Figure 21 – *Flow View* displaying data flow from two images (control example x_0 and adversarial example x_0^{adv}) plus background data. The upper image contains layers from several parts of the model. A diverging point can be observed in the area highlighted in red, as the two trajectories suddenly branch in different directions between layers *block_14_add* and *block_15_add*. All layers contained between these two can then be selected for display, resulting in the lower image. The flow visualization allows investigation and discovery of the exact moment when the two images’ model perception splits apart. . 72
- Figure 22 – Analysis of outputs for x_0 and x_0^{adv} at the layer *block_15_project* of *MobileNet V2*. a) shows the 20 units with the most expressive changes in perception between the two examples (highest absolute difference). Blue and red represent positive and negative outputs, respectively. b) displays the absolute output sum for each of the 20 units on each data example. Units 34, 143, 89, and 152 (circled in black in both images) show high overall outputs and high variations, being more likely to be relevant for classification for objects x_0 and x_0^{adv} 73
- Figure 23 – Image patterns detected by the four most relevant units in layer *block_15_project*, as identified in Figure 22. Correlating filters and visual features can help explain how an adversarial example is able to exploit a CNN model, especially concerning the distinction between robust and non-robust features. 73

CONTENTS

1	INTRODUCTION	21
1.1	Problem Description	21
1.2	Goals and Hypothesis	23
1.3	Document Structure	23
2	ALIGNING MULTIDIMENSIONAL PROJECTIONS	27
2.1	Introduction	27
2.2	Related Work	29
2.3	Aligning Projections	30
2.3.1	<i>Dynamic t-SNE</i>	31
2.3.2	<i>Visual Feature Fusion</i>	31
2.4	Generic Projection Alignment Model	32
2.4.1	<i>Multiple Feature UMAP (MF-UMAP)</i>	33
2.5	Results	33
2.6	Conclusions	35
3	VISUALIZING ACTIVITY SPACE IN NEURAL NETWORKS	37
3.1	Introduction	37
3.2	Related Work	39
3.3	Visualizing the Activity Space	41
3.3.1	<i>Projecting Data</i>	42
3.3.1.1	<i>Aligning Projections</i>	43
3.3.2	<i>Vector Field Generation</i>	44
3.3.3	<i>Vector Field and Trajectory Visualization</i>	45
3.4	Results	45
3.4.1	<i>MNIST on the MLP model</i>	46
3.4.2	<i>ImageNet on MobileNetV2</i>	48
3.5	Limitations	50
3.6	Conclusions	51
4	ACTIVATION PROFILES AND ADVERSARIAL ATTACKS	55
4.1	Introduction	55
4.2	Related Work	57
4.3	Design Goals	60

4.4	Framework Structure	62
4.4.1	Overview	62
4.4.2	Architecture View	62
4.4.3	Flow View	63
4.4.4	Layer View	65
4.5	Study Cases	66
4.5.1	Adversarial Data Generation	66
4.5.2	Explaining Adversarial Examples	67
4.5.3	Vulnerable Features on Deep Networks	70
4.6	Discussion and Limitations	74
4.7	Conclusion	75
5	CONCLUSION	77
5.1	Work Overview	77
5.2	Hypothesis and Goals	78
5.3	Challenges	79
5.4	Future Work	80
	BIBLIOGRAPHY	81
	GLOSSARY	93
APPENDIX A	APPENDIX: CONCEPTS AND TOOLS	95
A.1	Introduction	95
A.2	Gradient Descent-based Dimensionality Reduction	95
A.2.1	VFF	95
A.2.2	<i>t</i> -SNE	96
A.2.2.1	<i>dt</i> -SNE	98
A.2.3	UMAP	98
A.3	Discrete Vector Field Generation	99
A.4	Generation of Adversarial Examples	101
A.4.1	FGSM	101
A.4.2	PGD	101

INTRODUCTION

1.1 Problem Description

With the evolution of technology, the amount and complexity of data generated and acquired by computer systems is rapidly increasing, as well as the presence of these systems in our daily lives. Decision-making tasks based on the analysis and interpretation of such data are becoming ubiquitous and increasingly difficult, as exploration and interpretation of the information contained in this data are ever more costly and complex to be executed.

This is known as the information overload problem: the danger of navigating through large amounts of data that may be irrelevant to the task at hand, processed in an inappropriate way, or presented ineffectively. This problem can result in losses of time, money and opportunities for the people or organizations, as it means that necessary information may not be obtained in time (KEIM *et al.*, 2010).

In order to overcome problems inherent to the interpretation of large and/or complex databases, new computational tools are being constantly developed. Among the existing solutions, two areas are of particular notice: Machine Learning (ML) (MOHRI; ROSTAMIZADEH; TALWALKAR, 2018) and Information Visualization (IV) (TELEA, 2014). In the first case, computational models are created with the intent of inferring results or finding patterns in a semi automatic fashion. In the second case, graphic representations are generated allowing users to employ their visual skills in the interpretation process. When used together, visualization can assist machine learning processes, with different objectives depending on the moment it is applied in the analytical cycle: When applied as preprocessing, visualization and user interaction can offer assistance by identifying groups, selecting outliers and in general get a preliminary view of the dataset before running computational analysis. When applied as postprocessing, visualization can be used to interpret the results of the analysis, by allowing the observation of groups or classes and quickly grasping important information about the results that may indicate

the need to analyze further.

When visualization methods are used simultaneously with data mining or ML approaches with the intent of examining and improving the analytical process, the process is called Visual Analytics (VA) (KEIM *et al.*, 2010). VA techniques allow users to actively participate in the generation and refinement of analytical results, enabling new approaches to the effective insertion of user knowledge in the computational model, replacing underperforming automated heuristics. Thus, the VA model can be seen as a cycle composed of two main components, visualization and model: alterations in any of the two must affect the other, with modifications in the visualization resulting in changes in the computational model and alterations in this model leading to new visual representations.

ML models have been widely successful in many different applications. Deep Neural Networks (DNNs), in particular, are currently among the state-of-the-art techniques for the analysis of large-scale, complex datasets, due to their ability to abstract high-level patterns and model data beyond most heuristics (LECUN; BENGIO; HINTON, 2015). Despite the prevalence of DNNs in different domains, such as natural language processing (OTTER; MEDINA; KALITA, 2020), face and speech recognition (LIU *et al.*, 2017c), and generation of artificial data (PAN *et al.*, 2019), its success heavily depends on the right choice of hyperparameters and architecture. However, in many applications these models are still seen as black boxes, making them difficult to interpret and hindering processes such as implementation, improvement, replacement, or adaptation. Several efforts have been made by the machine learning and visual analytics research communities directed at improving model explainability, comprehending a field known as Explainable Artificial Intelligence (XAI) (ADADI; BERRADA, 2018).

In the past few years, the use of visualization tools and techniques to support the understanding of neural network models has become more prolific, with many different approaches focusing on exploring and explaining different aspects of DNN training, topology, and parametrization (HOHMAN *et al.*, 2018). As deep models grow more complex and sophisticated, understanding what happens to data inside these systems is quickly becoming key to improving their efficiency and designing new solutions. Recently, the use of Dimensionality Reduction (DR) (NONATO; AUPETIT, 2018) techniques as a tool to aid the understanding of NN models has attained relative success (RAUBER *et al.*, 2017; RAUBER; FALCAO; TELEA, 2017; ESPADOTO; RODRIGUES; TELEA, 2019), being adopted in several state-of-the-art VA approaches (PEZZOTTI *et al.*, 2018; KAHNG *et al.*, 2017) to analyzing hidden neural network information.

DR techniques, also called *Multidimensional Projections* in the context of visualization, are mechanisms that embed complex data into low-dimensional spaces while still retaining meaningful relationships between objects, allowing for both computational and human interface operations to be conducted in a faster and more efficient manner. In a data visualization and visual analytics context, they are generally employed to map multivariate data into 2D or 3D

spaces, with the goal of exploration, observation, and comprehension of patterns and features. However, many XAI applications employ DR techniques in an "as-is" manner, without much attention being devoted to delving into how they work and how they can be adapted to better serve this field.

Through studying literature on Visual Analytics, Deep Neural Networks and Explainable Machine Learning, the researcher¹ uncovered limitations and problems in existing XAI approaches that use DR as a visualization tool, as well as applications that would benefit from a more focused application of such techniques. Therefore, this doctorate thesis consists of methods to overcome problems identified when using DR techniques to visualize neural network information, as well as approaches to adapt and build upon them to generate visual representations more suited to the needs of this domain.

1.2 Goals and Hypothesis

Within the context established in the previous section, the engineering goal of this doctorate research project is defined as follows:

Improving dimensionality reduction techniques for visualizing deep neural network information, developing new visualization methods and adjustments to existing ones to better suit the needs of this domain, as well as solving problems associated with it.

Therefore, the null hypothesis h_0 to be rejected in this thesis is written as:

There are no advantages to enhancing dimensionality reduction techniques and developing specific methods for exploratory visual analysis to improve explainability of neural network models.

The next section will detail how each chapter in this document tackles challenges in achieving the project's goals.

1.3 Document Structure

This thesis is structured as a *collection of articles*. Chapters 2, 3, and 4 are papers either published or in the process of submission to journals and conferences, followed by a conclusion.

Chapter 2 discusses the first problem uncovered when researching DR methods for visualization of NN models. The most common application of dimensionality reduction in NN visualization is to display multivariate outputs from a hidden intermediate layer in an NN model, in

¹ Author of this thesis.

order to observe patterns in how that layer perceives data and performs discrimination (RAUBER *et al.*, 2017; KAHNG *et al.*, 2017; PEZZOTTI *et al.*, 2018). These outputs may contain thousands of attributes per data instance, and therefore sophisticated projection techniques are required.

Often, these techniques are non-linear, allowing for distortions between original and projected space, while following guidelines such as local distance preservation. One of the most employed techniques in this context is t-SNE (MAATEN; HINTON, 2008), which is capable of generating representations that are highly efficient in highlighting clusters and patterns within datasets. However, visualizations obtained in this manner cannot be compared safely. Geometric positioning in the projected spaces of non-linear methods has no inherent meaning, especially in the case of t-SNE where global distance preservation is not prioritized (WATTENBERG; VIÉ-GAS; JOHNSON, 2016). This is an issue for NN visualization, as performing visual comparison between outputs from different layers subjected to the same data is useful for understanding model behavior.

The paper describes the researcher's approach to solving this problem: a novel projection alignment constraint that can be added to gradient-descent dimensionality reduction algorithms. This method allows different feature sets (multiple representations of the same data points) to be synchronized so that every point remains in a similar position in all projections, only moving if its similarity relations towards other objects change between feature sets. Applications for this approach are presented and the effectiveness of this method over using previous DR techniques is discussed. This work was published and presented at the 2020 Eurovis Workshop in Visual Analytics (EuroVA).

Chapter 3 shows an approach to adapting DR techniques to better suit the NN visualization domain. Once a method to safely generate multiple comparable projections was reached in Chapter 2, the researcher used it to generate sequential projections from different parts of a NN model, in order to observe how abstract representations of data evolve and how each subsequent connected layer is able to further separate and organize objects.

However, visually comparing multiple 2D projections to find points of interest can be a troublesome task, especially as the number of projected points increases. The solution found was to compute trajectories from all movement performed by data points between projection and apply a technique that approximates a discrete 2D vector field (a grid with arbitrary resolution in which every point contains a 2D vector) that best encompasses all trajectories contained in the dataset. This vector field is then used to generate additional visual representations that provide movement information in a single image.

Study cases are shown to explain how this visualization can aid in finding outliers and understanding data flow throughout different layers of a model. This work was presented at the 2020 International Conference on Information Visualization Theory and Applications (IVAPP), winning a Best Paper award. It was then expanded into an article for the MDPI Information Journal and published in the same year.

Chapter 4 shows the application of concepts developed throughout the project to a specific problem: the explanation of evasion adversarial attacks in DNN models. Adversarial attacks consist of input data generated with the intent of fooling a model's analysis, e.g., creating an image that, when subjected to a classifier, results in an incorrect result compared to its human interpretation. DNN vulnerability to adversarial examples is an important subject in research - especially considering how the adoption of deep learning for all kinds of tasks has increased in the past decade - and one that is not completely understood (SZEGEDY *et al.*, 2013).

While not directly using the methods developed in Chapter 3, Chapter 4's approach makes heavy use of the concept of visualizing a sequence of layer outputs: by projecting each hidden layer as an 1D line, it is possible to stack them and observe how model perception of data changes as it passes through. This visualization is particularly useful to pinpoint moments where perception of adversarial objects and regular objects change, and isolate layers and units that are under heavy attack.

Additionally, the proposed approach encapsulates these novel visualization metaphors in a framework, which also provides visual details of model architecture and inner workings of layers, so that a complete exploration process can be conducted to understand a model's weaknesses and input space features exploited by adversarial examples. This work is under preparation for submission.

Finally, Chapter 5 contains conclusions about the research described in this thesis, and a discussion about the achievement of goals and verification of hypothesis. This document also includes an Appendix, A, with detailed descriptions of techniques and tools involved in the project, as they are simplified in the respective articles due to space constraints in publishing.

ALIGNING MULTIDIMENSIONAL PROJECTIONS

Abstract: Dimensionality reduction techniques are popular tools for the visualization of neural network models due to their ability to display hidden layer activations and aiding the understanding of how abstract representations are being formed. However, many techniques render poor results when used to compare multiple projections resulted from different feature sets, such as the outputs of different hidden layers or the outputs from different models processing the same data. This problem occurs due to the lack of an alignment factor to ensure that visual differences represent actual differences between the feature sets and not artifacts generated by the technique. In this paper, we propose a generic model to align multiple projections when visualizing different feature sets that can be applied to any gradient descent-based dimensionality reduction technique. We employ this model to generate a variant of the UMAP method and show the results of its application.¹

2.1 Introduction

Dimensionality reduction (DR) techniques are mechanisms that embed complex data into low-dimensional spaces while still retaining meaningful relationships between objects, allowing for both computational and human interface operations to be conducted in a faster and more efficient manner. In data visualization and visual analytics, they are generally used to map multivariate data into 2D or 3D spaces, with the goal of exploration, observation, and comprehension of patterns and features.

¹ CANTAREIRA, G. D.; PAULOVICH, F. V. A Generic Model for Projection Alignment Applied to Neural Network Visualization. In: TURKAY, C.; VROTSOU, K. (Ed.).EuroVis Workshop on Visual Analytics (EuroVA). [S.l.]: The Eurographics Association, 2020. ISBN 978-3-03868-116-8. ISSN 2664-4487.

Commonly called Multidimensional Projections in this context (NONATO; AUPETIT, 2018), these techniques have been used in many different domains and applications, such as document collection exploration, vector field analysis, and multimedia organization. Recently, the use of projections as a tool to aid the understanding of artificial neural network (ANN) models has become an active subject in the research community (RAUBER *et al.*, 2017; RAUBER; FALCAO; TELEA, 2017; ESPADOTO; RODRIGUES; TELEA, 2019), with their use being adopted in several state-of-the-art VA approaches (PEZZOTTI *et al.*, 2018; KAHNG *et al.*, 2017) to analyzing hidden neural network information.

However, comparing projections generated by different feature sets representing the same data, such as outputs from multiple hidden layers of a neural network, present a few problems: attribute values and ranges may be widely different, and the number of attributes itself may not be the same. Furthermore, many sophisticated projection techniques are non-linear, meaning that there are no guarantees that small variations in the same feature set (e.g., small adjustments in neural network outputs from one training epoch to the next) will not result in widely different projections (WATTENBERG; VIÉGAS; JOHNSON, 2016; RAUBER; FALCÃO; TELEA, 2016).

While there are a few works dedicated to solving these issues, there are certain problems when generalizing their approaches, and there is no discussion over how to compare how projection techniques perform in the task of generating aligned embeddings of different feature sets so that their properties can be observed. Many projection techniques offer properties that are desirable for specific applications, but cannot be used in this context due to their inability to holding stable projections between observations.

In this short paper, we present a generic model that can be applied to any gradient descent-based projection technique to include an alignment factor that minimizes non-relevant variation between projections. Our model addresses issues that arise with this generalization, such as dimensionality differences and cost function variations. We apply this model to generate an extension of the well-known UMAP technique (MCINNES; HEALY; MELVILLE, 2018) that can keep alignment over multiple feature sets, and show its functionality in neural network applications. This extension is shown to produce more reliable aligned results while maintaining desirable features of UMAP.

In short, the main contributions of this paper are:

- A generic flexible framework for supporting projection alignment with any gradient descent-based DR technique;
- A variant of the UMAP projection technique that performs alignment.

2.2 Related Work

Multidimensional Projections can be described as operations that receive an input $X = (x_1, x_2, \dots, x_n) \in \mathbf{R}^m$ and produce a representation $X' \in \mathbf{R}^p$, $p < m$ as output. This representation is designed as to preserve dissimilarity relations δ_{ij} between points x_i and x_j as much as possible. Most visualization-related applications aim to embed data in the visual space, with $p = 2$ or $p = 3$.

There are many techniques to perform projection. Recent surveys (NONATO; AUPETIT, 2018; ESPADOTO *et al.*, 2019) discuss each technique and classify them according to many parameters, such as linearity, scope of optimization (local or global), data type, or capability of supervision. Among the most known techniques, we can cite the classical MDS (TORGERSON, 1952), PCA (HOTELLING, 1933), LDA (FISHER, 1936), LSP (PAULOVICH *et al.*, 2008), LAMP (JOIA *et al.*, 2011), PLMP (PAULOVICH; SILVA; NONATO, 2010), t-SNE (MAATEN; HINTON, 2008).

Recent techniques, such as H-SNE (PEZZOTTI *et al.*, 2016) and UMAP (MCINNES; HEALY; MELVILLE, 2018), aim to provide non-linear dimensionality reduction that preserves both local neighborhoods and global data structure. These techniques are shown to be of great use in data analysis applications, especially in machine learning. (PEZZOTTI *et al.*, 2018; CARTER *et al.*, 2019)

One type of application for projections that have become very popular is in the analysis of (deep) neural networks (RAUBER *et al.*, 2017; RAUBER; FALCAO; TELEA, 2017; ESPADOTO; RODRIGUES; TELEA, 2019; PEZZOTTI *et al.*, 2018; KAHNG *et al.*, 2017; CANTAREIRA; PAULOVICH.; ETEMAD., 2020). One typical use of projection is to compare outputs from multiple hidden layers of a neural network, producing a different image for each layer. In this scenario, projections need to be aligned so they can be properly compared. The idea of *Projection Alignment* is to obtain multiple projections from the same number of feature representations of the same data in such a way that projected distances are as similar as possible to distances in the original feature sets while also keeping projections as similar as possible with one another.

Currently, only a few projection techniques support alignment natively, mostly being restricted to using the same initialization parameters (MAATEN; HINTON, 2008) or a fixed set of control points (JOIA *et al.*, 2011). To the best of our knowledge, the only techniques capable of actively aligning outputs during projection generation are the Dynamic t-SNE (Dt-SNE (RAUBER; FALCÃO; TELEA, 2016)) and the Visual Feature Fusion (VFF (HILASACA; PAULOVICH, 2019)). These two are better detailed in the next section.

Figure 1 – Projections p_0 and p_1 of two feature sets using standard UMAP (no alignment) and our model (MF-UMAP). The feature sets correspond to activation data from two hidden layers of a CNN. These layers provide similar outputs, so the variation in the projections should be small. However, the unaligned projections cause groups to change places (e.g., the orange group) and may result in misinterpretation.

2.3 Aligning Projections

As previously discussed, alignment is essential when comparing projections in an analytical task. Figure 1 illustrates that, presenting an example of projections with and without alignment. We show projections from two feature sets, f_0 and f_1 , that are produced from the activations of the last two hidden layers of a Convolutional Neural Network (CNN) from 2,000 samples from the MNIST (LECUN *et al.*, 1998) dataset. The upper row contains projections p_0 and p_1 obtained from the standard UMAP technique, and the lower row contains projections from our modified version of UMAP to preserve alignment (*Multi-feature UMAP*). We know beforehand that the two activation sets are quite similar. However, even if initialized with the same parameters, standard UMAP generates projections whose clusters are in different positions, and linear transformations, such as Procrustes analysis, cannot correct this mismatch. This is even more common in projection techniques that are less strict regarding inter-cluster distances, such as t-SNE. Techniques that solve this problem, and therefore form the basis of our generic alignment framework, are presented next.

2.3.1 Dynamic t-SNE

When projecting layer activation data from neural networks, Rauber et al. (RAUBER *et al.*, 2017) discussed methods of reducing non-relevant variability in data projected with t-SNE, such as using point-cloud registration and using previous projections as initializations. The authors later proposed the Dt-SNE (RAUBER; FALCÃO; TELEA, 2016), a variation of t-SNE designed to produce successive projections of a time-varying dataset while minimizing irrelevant projection variability and ensuring *temporal coherence*. This is achieved by adding a term to t-SNE’s cost equation, as follows:

$$C_{dtsne}[t] = C_{tsne}[t] + \frac{\lambda}{2N} \sum_{i=1}^N \|p_i[t] - p_i[t+1]\|^2 \quad (2.1)$$

In this equation, N is the number of data objects in each time step $t \in T$ to be projected, $C_{tsne}[t]$ is the standard t-SNE cost for projection t , $p_i[t]$ is the position of data instance i in projection t , and λ is a parameter that controls the trade-off between temporal alignment and fidelity to the standard t-SNE optimization.

The authors present a discussion on the different approaches to alignment, and their reasoning for using an additional cost function term is applicable to our method as well. While Dt-SNE was designed to align different versions of the same feature set, the algorithm itself offers no such restriction, but distances in each set need to be normalized and comparable. Its main limitation is its temporal focus: projections are only aligned concerning the ones immediately before or after them.

2.3.2 Visual Feature Fusion

Proposed by Hilaraca et al. (HILASACA; PAULOVICH, 2019), VFF itself is a system designed for adapting and fusing different feature representations of the same data. During its user interaction phase, the system displays projections of a sample of the data according to each feature set, which needs to be aligned as to highlight differences in how they describe the data instances. To achieve this goal, the technique first generates a projection $\bar{p}$ using the mean pairwise distances for data points in all feature sets. Each projection t is then obtained by minimizing

$$C_{ff}[t] = (1 - \lambda) \cdot C_{stress}[t] + \frac{\lambda}{N^2} \sum_i^N \sum_j^N (d(\bar{p}_i, \bar{p}_j) - \| \bar{p}_i - p_j[t] \|^2) \quad (2.2)$$

where $C_{stress}[t]$ is the normalized stress (ESPADOTO *et al.*, 2019) for projection t , $p_j[t]$ is the position of object j in projection t , $\bar{p}_i$ and $\bar{p}_j$ are the positions of objects j and i in $\bar{p}$, $d(\bar{p}_i, \bar{p}_j)$ is the distance between points i and j in $\bar{p}$, and λ is the trade-off controlling parameter.

While VFF is fully designed with the intent of merging different feature sets, its global stress-based optimization makes it not ideal for tasks that require neighborhood preservation and highlighting clusters. The alignment component is also more limiting as it attracts all points towards a previously set projection.

2.4 Generic Projection Alignment Model

Our generic model for projection alignment uses a cost function modification that can be included in any projection method that employs gradient descent-based optimization, such as the traditional MDS (TORGERSON, 1952), force schemes (EADES, 1984; CHALMERS, 1996), SNE-based methods (MAATEN; HINTON, 2008; INGRAM; MUNZNER, 2015; PEZZOTTI *et al.*, 2016), among others. In our model, to make distances comparable between feature sets $F[t]$, we first apply the following normalization for data instance $F[t]_i$:

$$F'[t]_i = N \frac{F[t]_i}{\sum_{j=1}^N ||F[t]_j||_2} \quad (2.3)$$

ensuring that distance between vectors is bounded, despite the scale of each feature set or the number of attributes they contain. We opt for the mean of $F[t]$ norms instead of maximum to avoid distortion by outliers that may appear during feature generation. After that, the projection optimization algorithm is executed, with the total cost function for each projection t given by

$$C[t] = (1 - \lambda) \cdot C_{projection}[t] + \lambda \cdot \gamma \cdot C_{alignment}[t] \quad (2.4)$$

where $C_{projection}[t]$ is the standard cost function for the technique being used (e.g., t-SNE, UMAP, and so on), λ is the parameter that controls how much the alignment interferes in the projection, and γ is a scaling parameter. $C_{alignment}[t]$ is the penalty for moving away from other projections, given by

$$C_{alignment}[t] = \frac{1}{2N} \sum_{i=1}^N d(p_{i[t]}, \bar{p}_i) \quad (2.5)$$

where $d(p_{i[t]}, \bar{p}_i)$ is the distance between the point p_i in the projection t and the mean position of the same point between all projections $\bar{p}_i$. When Euclidean distance is employed, the differentiation of $C_{alignment}$ is simple and has a low computing cost, resulting in the following gradient matrix to update the projection $p[t]$

$$\nabla C[t] = (1 - \lambda) \nabla C_{projection}[t] + \lambda \gamma [(p[t] - \bar{p})] \quad (2.6)$$

The parameter γ was added as a way to compensate for the difference in magnitude between cost function values in different projection techniques and bring the norms of projection and alignment gradients to a similar scale, increasing comparability between techniques and avoiding a trial-and-error process in finding an appropriate λ value. We set it to $||\nabla C_{projection}[t]|| / ||\nabla C_{alignment}[t]||$ so the two gradients always have the same norm and λ can explicitly control the ratio between forces in each point. It can, however, be fixed to $\gamma = 1$, just

requiring the user to bear in mind that the projection’s behavior according to λ will likely be unique for each case.

2.4.1 Multiple Feature UMAP (MF-UMAP)

UMAP has been increasingly adopted for visualization in data analysis tasks in recent works due to its ability to better represent global distance relationships (smaller *stress* values) when compared to t-SNE, while retaining similar properties (MCINNES; HEALY; MELVILLE, 2018). Its optimization is based on minimizing the cross-entropy between neighborhood probability distributions in projected and original spaces.

These properties make UMAP a desirable approach to projecting multiple feature set data, especially relating to neural network visualization: for instance, global distances contain information such as similarity between classes, or groups that get closer or further apart in different observations. Therefore, we chose this technique to apply the general alignment model and evaluate its capabilities. The implementation was done in Python, using scikit-learn, scipy and numpy libraries, following the UMAP algorithm proposed in (MCINNES; HEALY; MELVILLE, 2018). As the extra terms have low computational cost compared to the projection optimization itself, UMAP’s performance does not suffer a significant impact.

2.5 Results

To verify how the aforementioned concepts can be employed in machine learning visualization, we performed experiments projecting activations from hidden layers of ANNs. We used CIFAR-10 (KRIZHEVSKY; HINTON *et al.*, 2009) and MNIST (LECUN *et al.*, 1998) datasets, projecting samples of 2,000 instances from test data. We used three architectures: a 4-layer MLP with 20 processing units per layer; a CNN with 2 convolutional layers, 2 dense layers (20, 20, 40, and 40 units respectively), Batch Normalization and Max Pooling 2×2 between the convolutional layers; the VGG16 (SIMONYAN; ZISSERMAN, 2014) model pre-trained on Imagenet dataset (DENG *et al.*, 2009) without top layers, which were substituted by two layers of 1,024 dense units. All networks were trained using Adam (KINGMA; BA, 2014) optimizers, with starting learning rate = 0.001.

First, we observe how knowledge is acquired between hidden layers. Figure 2 shows projections of activations from the 4 layers of the CNN over samples from the MNIST data set (convolutional layer activations were captured after the max-pooling layers). These layers have widely different output sizes (3,920 dimensions in the first layer and 40 in the fourth), but the resulting projections are comparable. The separation between certain classes is unclear in the first layer and gets more explicit in later ones. Observing label information, we notice classes that the network considers to be more similar, such as the numbers 4, 7, and 9 in the top region and 3, 5, and 8 at the bottom area. This information remains in the last projections, although with clearer separation.

Activations of 2000 MNIST test samples over CNN 4-layer network

Figure 2 – Projections of activations of 4 sequential layers of a CNN. The shapes identified by the first two convolutional layers are similar for specific classes, as they are not fully separated. The last two organize this information and divide the space among classes.

Our second experiment is to compare different ANNs. Figure 3 shows a comparison between the last layers of CNN and MLP models, using MNIST data activations. We included additional images of projections generated using Dt-SNE, to highlight a relevant aspect of UMAP-based alignment. Both networks are well-trained and capable of separating data at the last layer, but the MLP shows areas (A and B) where a few threads of connection remain between classes. Dt-SNE projection misses this information.

We calculated the centroid for each class and a distance matrix between all centroids in all projections, and then obtained the difference in centroid distances between the projections and the original feature data. In MF-UMAP projections, the resulting mean difference was $= 0.3688$, while the mean difference in the Dt-SNE projections was $= 0.5015$, showing that the first better preserves inter-group distances.

Finally, we compare VGG16 and CNN models. These models were both trained using the CIFAR-10 dataset and obtained similar results (accuracy scores of 74% and 70%, respectively). The VGG16 model was frozen and only had the new top layers trained. Figure 4 shows the

Figure 3 – Comparison between activations of the last hidden dense layers of MLP and a CNN using MF-UMAP and Dt-SNE. The CNN network separates data more explicitly, to the point of overfitting, but only MF-UMAP is capable of showing areas where separation improved (A and B).

resulting projections from the last hidden layers from both models. Both projections show a central area with mixed classes, representing instances with uncertain outputs. However, VGG16 shows more concentrated areas, and is able to isolate a higher amount of green, red, and brown points, indicating a better grasp of what composes these classes. This more specialized output can, however, lead to overfitting.

2.6 Conclusions

In this paper, we proposed a generic method for producing aligned projections of multiple feature sets, allowing the visualization to focus on the actual differences between each set. Our method improves on existing alignment solutions by supporting the flexibility needed by different projection techniques as they may offer advantages for specific tasks. We used this method to create a variation of UMAP that enabled alignment and showed its usefulness in neural network visualization applications. Future research directions may include developing a quantitative evaluation method to determine how well different techniques fit in the trade-off between projection fidelity and alignment, and how alignment interferes with other quality measures.

Figure 4 – Last hidden layer activations from two ANNs using CIFAR-10 data. a) CNN and b) VGG16. While their accuracy results are somewhat similar (0.70 and 0.74, respectively), there are clear differences in how each network perceives data, and how the extra complexity present in the VGG16 model can generate more refined, concentrated outputs.

VISUALIZING ACTIVITY SPACE IN NEURAL NETWORKS

Abstract: Deep Neural Networks are known for impressive results in a wide range of applications, being responsible for many advances in technology over the past few years. However, debugging and understanding neural networks models' inner workings is a complex task, as there are several parameters and variables involved in every decision. Multidimensional projection techniques have been successfully adopted to display neural network hidden layer outputs in an explainable manner, but comparing different outputs often means overlapping projections or observing them side-by-side, presenting hurdles for users in properly conveying data flow. In this paper, we introduce a novel approach for comparing projections obtained from multiple stages in a neural network model and visualizing differences in data perception. Changes among projections are transformed into trajectories that, in turn, generate vector fields used to represent the general flow of information. This representation can then be used to create layouts that highlight new information about abstract structures identified by neural networks.¹

3.1 Introduction

Given their ability to abstract high-level patterns and model data beyond most heuristics (LECUN; BENGIO; HINTON, 2015), Deep Neural Networks (DNNs) are currently among the state-of-the-art techniques for the analysis of large-scale, complex datasets. Despite the prevalence of DNNs in different domains, such as natural language processing, face and speech recognition, and generation of artificial data, its success heavily depends on the right choice of hyperparameters and architecture.

¹ CANTAREIRA, G. D.; ETEMAD, E.; PAULOVICH, F. V. Exploring neural network hidden layer activity using vector fields. Information, Multidisciplinary Digital Publishing Institute, v. 11, n. 9, p. 426, 2020.

In recent years, visualization strategies have become increasingly popular in the research community to help analysts interpret the results and support the improvement of DNNs. One of the most popular visualization strategies, multidimensional projection techniques (NONATO; AUPETIT, 2018), has reportedly attained relative success in helping users explore and explain what happens inside DNNs (RAUBER *et al.*, 2017; MAHENDRAN; VEDALDI, 2015; SRIVASTAVA *et al.*, 2014). These techniques aim to generate lower-dimensional visual representations capable of preserving multidimensional data structure, such as relationships between data instances or clusters' presence. However, the currently available techniques are somewhat limited when exploring sequential processes inside neural networks, such as the state of hidden layers during training or the shaping of high-level representations as data flows through different layers of a network. This information can provide insights to important issues, such as determining different layers' contributions in discriminating certain objects or tracking noteworthy behavior during training, to better understand unusual data.

In this paper, we present a novel approach to visualize the hidden structure of DNNs, designed to aid in understanding abstract high-level representations inside a model and how they are formed. In our approach, we project data extracted from various states of a neural network and estimate a common space between projections by computing a vector field that can show how these projections relate to one another. We show how this space can be used in visualization, as well as possibilities in identifying progress and tracking down meaningful changes in the neural network as training epochs go by or as data flows through the model. This paper is an extension of work presented at the IVAPP 2020 conference (CANTAREIRA; PAULOVICH.; ETEMAD., 2020), refining dimensionality reduction methodology by employing a general approach to projection alignment (CANTAREIRA; PAULOVICH, 2020), developing new visual representations for vector field and trajectory data incorporated to sequential projections, and presenting new experiments that highlight these properties.

Our main contributions are:

- A projection-based visual representation suited to represent sequential aspects of DNNs, reducing movement clutter while keeping distances meaningful;
- A visual transition space and vector field between two or more sequential projections that can be used to represent flow or evolution.

The remainder of the paper is organized as follows. In Section 3.2, we discuss related work regarding visualization techniques for neural networks. In Section 3.3, we formalize the problem and present a complete overview of our solution. In Section 3.4, we present an extensive set of experiments to show our solution in practice. In Section 3.5, we discuss our limitations, and the paper is concluded in Section 3.6.

3.2 Related Work

Artificial Neural Networks (ANNs) are structures composed of groups of simple and complex cells. Simple cells are responsible for extracting basic features, while complex ones combine local features producing abstract representations (SCHERER; MÜLLER; BEHNKE, 2010). This structure hierarchically manipulates data through layers (complex cells), each using a set of processing units or neurons (simple cells) to extract local features. In classification tasks, each neuron divides input data space using a linear function (i.e., hyperplane), which is positioned to obtain the best separation as possible between labels of different classes. Thus, the connections among processing units are responsible for combining the half-spaces built up by those linear functions to produce nonlinear separability of data spaces (LECUN; BENGIO; HINTON, 2015). Deep Neural Networks (DNNs) are artificial neural network models that contain a large number of layers between input and output, generating more complex representations. Such networks are called convolutional neural networks (CNNs) when convolutional filters are employed.

In the past few years, the use of visualization tools and techniques to support the understanding of neural network models has become more prolific, with many different approaches focusing on exploring and explaining different aspects of DNN training, topology, and parametrization (HOHMAN *et al.*, 2018). As deep models grow more complex and sophisticated, understanding what happens to data inside these systems is quickly becoming key to improving their efficiency and designing new solutions.

When exploring layers of a DNN, a common source of data are the hidden layer *activations*: the output value of each neuron of a given layer when subjected to a data instance (input). Many DNN visualization approaches are focused on understanding the high-level abstract representations that are formed in hidden layers. This is often attained by transferring the activations of hidden layer neurons back to the feature space, as defined by the Deconvnet (ZEILER; FERGUS, 2014) and exemplified by applications such as the Deep Dream (SZEGEDY *et al.*, 2015). Commonly associated with CNNs, techniques based on this approach often try to explain and represent which feature values in a data object generate activations in certain parts of hidden layers. The Deconvnet is capable of reconstructing input data (images) at each CNN layer to show the features extracted by filters, supporting the detection of incidental problems based on user inspections.

Other techniques focus on identifying content that activates filters and hidden layers. Simonyan *et al.* (SIMONYAN; VEDALDI; ZISSERMAN, 2013) developed two visualization tools based on Deconvnet to support image segmentation, allowing feature inspection and summarization of produced features. Zintgraf *et al.* (ZINTGRAF *et al.*, 2017) introduced a feature-based visualization tool to assist in determining the impact of filter size on classification tasks and identifying how the decision process is conducted. Erhan *et al.* (ERHAN *et al.*, 2009) proposed a strategy to identify features detected by filters after their activation functions, allowing the visual inspection of the impact of network initialization as well as if features are humanly

understandable. Similarly, Mahendran et al. (MAHENDRAN; VEDALDI, 2016b) presented a triple visualization analysis method to inspect images. Babiker et al. (BABIKER; GOEBEL, 2017) also proposed a visual tool to support the identification of unnecessary features filtered in the layers. Liu et al. (LIU *et al.*, 2017b) present a system capable of showing a CNN as an acyclic graph with images describing each filter.

Other methods aim to explore the effects of different parameter configurations in training, such as regularization terms or optimization constraints (SRIVASTAVA *et al.*, 2014). These can also be connected to different results in layer activations or classification outcomes. Some techniques are designed to help evaluate the effectiveness of specific network architectures, estimating what kind of abstraction can be learned in each section, such as the approach described by Yosinski et al. (YOSINSKI *et al.*, 2015).

The research previously described is focused on identifying and explaining *what* representations are generated. However, it is also important to understand *how* those representations are formed, regarding both the training process and the flow of information inside a network. Comprehending these aspects can lead to improvements in network architecture and the training process itself. The DeepEyes framework, developed by Pezzotti et al. (PEZZOTTI *et al.*, 2018), provides an overview of DNNs identifying when a network architecture requires more or fewer filters or layers. It employs scatterplots and heatmaps to show filter activations and allows the visual analysis of the feature space. Kahng et al. (KAHNG *et al.*, 2018a) introduce a method to explore the features produced by CNN's projecting activation distances and presenting a neuron activation heatmap for specific data instances. However, these techniques are not designed for projecting multiple transition states, and their projection methods require complex parametrization to show the desired information.

Multidimensional projections (or dimensionality reduction techniques) (NONATO; AU-PETIT, 2018) are popular tools to aid the study of how abstract representations are generated inside ANNs. Specific projection techniques, such as the UMAP (MCINNES; HEALY; MELVILLE, 2018), were developed particularly with machine learning applications in mind. While dimensionality reduction techniques are generally used in ANN studies to illustrate model efficacy (DONAHUE *et al.*, 2014; SRIVASTAVA *et al.*, 2014; HAMEL; ECK, 2010; MOHAMED; HINTON; PENN, 2012; MAHENDRAN; VEDALDI, 2015), Rauber et al. (RAUBER *et al.*, 2017) showed their potential on providing valuable visual information on DNNs to improve models and observe the evolution of learned representations. Projections were used to reveal hidden layer activations for test data before and after training, highlighting the effects of training, the formation of clusters, confusion zones, and the neurons themselves, using individual activations as attributes. Despite offering insights on how the network behaves before and after training, the visual representation presented by the authors for the evolution of representations inside the network or the effects of training between epochs displays a great deal of clutter; when analyzing a large number of transition states, information such as the relationships between

classes or variations that occur only during intermediate states may become difficult to infer. Additionally, the method used to ensure that all projections share a similar 2D space is prone to problems in alignment. In this paper, we propose a visualization scheme that employs a flow-based approach to offer a representation better suited to show transition stated and evolving data in DNNs. We also briefly address certain pitfalls encountered when visualizing neuron activation data using standard projection techniques, such as t-SNE (MAATEN; HINTON, 2008), and discuss why these pitfalls are relevant to our application.

3.3 Visualizing the Activity Space

Our visual representation is based on gathering hidden layer activation data from a sequence of ANN outputs (steps), then projecting them onto a 2D space while sharing information to ensure semantically similar data remain in similar positions between projections. The movement of the same data instance throughout each projected step generate trajectories, which are then condensed into vector fields that reflect how data flows throughout the sequence. There are two main sequential aspects to be visualized with ANN outputs: how abstract data representations inside the network change and evolve with training and how representations are built and connected as one layer forwards information to the next. For each aspect, a different sequence of hidden layer outputs is needed: outputs from the same given layer during different epochs of training are used to visualize how training changes data representations in that layer, while outputs from different layers from the same network are used to visualize how data representations evolve as layers propagate information.

To build this representation, we first extract activation sets $A[1], A[2], \dots, A[T]$ representing network outputs from T sequential steps of the process we want to explore. In this paper, we either a) save a network model at different epochs of training, choose a slicing layer, feed the same set of input data to the saved models, and then save the outputs from the slicing layer, or b) pick a given network model, slice it at different layers, feed the input data set, and save the outputs from these layers as activation sets.

Once the activation data are extracted, it is projected onto a 2D space using a multidimensional projection technique, obtaining $P[1], P[2], \dots, P[T]$. To produce coherent projections, an alignment method is applied, ensuring that projections stay similar to each other while preserving original multidimensional distances. Then, positions of the same points in two subsequent projections form movement vectors $\phi[1], \dots, \phi[T - 1]$ that describe how data instances in one output changed to the next. This movement data are joined for all output transitions, generating trajectories (or *trails*) for each data instance across all T steps, which are then used to compute vector fields. Finally, projections, trajectories, and vector fields are combined to create visualizations. The 2D space shared by all projections is our *visual activity space*, and trajectories and vector fields describe how network outputs flow through it. Figure 5 summarizes this process.

Figure 5 – Overview of the learning space vector field generation. From a DNN model, we obtain sequential data in the form of hidden layer activations. We compute projections from this highly multivariate data, aligning them to ensure that obtained projections are synchronized. Differences between each pair of projections in the sequence (transition steps) are turned into trajectories and then processed to obtain vector fields representing changes in data. Finally, everything is combined to compose distinct visual representations of the neural network.

The following sections detail how data are projected, and how vector fields are generated and visualized in our model.

3.3.1 Projecting Data

Although we can compare any set of projections to produce vector fields, we need to eliminate changes between projections that do not reflect variations in the high-dimensional data as much as possible. Therefore, the projection technique itself must share information between all data sets to ensure the generation of a synchronized view. Although precise and popular, nonlinear projection techniques generally do not guarantee consistent visual spaces when comparing distinct projections since axes (dimensions) of the original space are not projected into straight lines on visual spaces, and thus cannot be considered equivalent in projected spaces. Some techniques offer a certain degree of control over projection layout, such as fixing control points (JOIA *et al.*, 2011) or selecting the same initialization parameters (MAATEN; HINTON, 2008), but often this means a trade-off between local and global distance preservation as they are not designed with this task in mind.

In this context, projections need to be *aligned* as best as possible, i.e., distances between points in projections must be as similar as possible to those in the original data, while also keeping projections as similar as possible with each other. Currently, there are three different approaches to generating aligned projections of multiple feature sets in literature: Dynamic t-SNE (Dt-SNE) (RAUBER; FALCÃO; TELEA, 2016), Visual Feature Fusion (VFF) (HILASACA; PAULOVICH, 2019), and the generic alignment model (CANTAREIRA; PAULOVICH, 2020).

Dt-SNE is a variation of t-SNE capable of minimizing data variability in successive projections to a certain degree to observe changes in multiple DNN states. This is achieved by adding a term to t-SNE’s cost function to approach points representing the same instance in successive projections close to each other. Although attaining better results if compared to the original t-SNE in terms of aligning subsequent projections, it inherits a major problem: the misleading effect of cluster distance and shape in t-SNE projections resulted from the local nature of optimization and how hyperparameters are set up (WATTENBERG; VIÉGAS; JOHNSON, 2016)—the observed distance between clusters is unreliable and sensitive to parametrization (perplexity parameter).

The VFF technique was originally developed to fuse different feature representations of data to build a new, user-driven representation. VFF calculates pairwise distances between data points in all feature sets and generates a new pairwise distance table in which the distance between points x_i, x_j is the mean distance $d_T(x_i, x_j)$ across all feature sets T . These distances are used to generate a guide projection p using a force-based model that minimizes metric stress. The technique then proceeds to generate a projection for each feature set, attracting every point towards its counterpart in p at each iteration.

The generic alignment model is the one adopted in this paper. While t-SNE’s global distance preservation is unreliable and VFF’s stress minimization makes it perform poorly for visualizing local neighborhoods, the generic model shows an implementation of UMAP (MCINNES; HEALY; MELVILLE, 2018), which we view as a reasonable compromise. Additionally, being designed for any gradient descent-based projection technique, this model can be useful for further experiments with other techniques.

3.3.1.1 Aligning Projections

According to the generic alignment model, the alignment process consists of obtaining samples $F[1], \dots, F[T]$ from activation sets $A[1] \dots A[T]$ with the same indexes (i.e., all samples contain data related to the same points) and then calculating projections $P[1], \dots, P[T] \in \mathbb{R}^2$ that preserve distance relationships in each set $F[k]$ as best as possible while aligning themselves to one another. This process implies performing an optimization task considering two restrictions. For each feature set k , the cost function to minimize is given by

$$C[k] = (1 - \lambda) \cdot C_{umap}[k] + \lambda \cdot \gamma \cdot C_{alignment}[k] \quad (3.1)$$

where $C_{umap}[k]$ is the standard cost function for UMAP (binary cross-entropy), λ is the parameter that controls alignment interference (0.1 for all experiments in this paper), and γ is an internal scaling parameter defined automatically at each iteration. $C_{alignment}[k]$ is the penalty for inter-

projection distance, given by

$$C_{alignment}[k] = \frac{1}{2N} \sum_{i=1}^N d(P_i[k], \bar{P}_i) \quad (3.2)$$

where $d(P_{i[t]}, \bar{P}_i)$ is the distance between point P_i in the projection k and the current mean position of the same point between all projections $\bar{P}_i$.

This optimization is done by gradient descent. To generate aligned projections from hidden layer outputs with widely different number of dimensions, pairwise distances need to be normalized. We performed normalization by dividing attributes by the mean instance norm from every feature set. Once aligned projections are generated, we can proceed to the generation of vector fields.

3.3.2 Vector Field Generation

The vector fields contained in our visualization are generated using an adapted version of the Vector Field K-means technique, proposed by Ferreira et al. (FERREIRA *et al.*, 2013a). This technique defines a distance measure $d(X, a)$ between a discrete grid representation of a vector field X and a trajectory a , composed by pairs of time stamp and spatial position $(t, a(t))$. Once this distance function is defined, it is minimized to find discrete vector fields that provide the best approximation to a group of trajectories and match each trajectory to a field, similar to the conventional k-means algorithm.

In this approach, a trajectory is represented by a path written as $\alpha : [t_0, t_1] \rightarrow \mathbb{R}^2$ and a vector field is defined in a domain $\Omega \subset \mathbb{R}^2$, i.e., a function $X : \Omega \rightarrow \mathbb{R}^2$. For all data points, each transition ϕ is considered a path, and time intervals between all steps are presumed equal to $1/T$. Finding the vector field that best matches a set of trajectories Φ can then be described as an optimization problem of minimizing

$$E = \min_X \lambda_L \|\Delta X\|^2 + \frac{(1 - \lambda_L)}{N} \sum_{\alpha_i \in \Phi} \int_{t_0^i}^{t_1^i} \|X(\alpha_i(t)) - (\alpha'_i(t))\|^2 dt \quad (3.3)$$

where $\alpha'_i(t)$ is the velocity vector of trajectory α on instant t . The first term is a regularization restriction, where Δ is the Laplace operator. This restriction ensures smoothness of the resulting vector field, with parameter λ_L controlling the weight of each term in the equation and therefore determining if the optimization should prioritize smoothness or matching vector field to trajectories.

In the Vector Field k-Means technique, vector fields are generated from trajectory groups and trajectories are then reassigned to the best matching fields at every iteration. Since our focus is to generate only one vector field that approximates a set of trajectories, we employ a simplified

version of this technique that fits a single vector field to the trajectory set, with no need for reassignment.

3.3.3 Vector Field and Trajectory Visualization

The final step in our approach is to combine all information obtained from (1) projections, (2) trajectories, and (3) vector fields components to generate visual representations. For each of these components, we generate different visual representations that are stacked to convey distinct properties of the NN model under analysis. For data projection, we use the conventional scatterplot metaphor with points representing data instances. Since multiple projections $P[1], P[2], \dots, P[T]$ are available, we allow users to select any projection to compose the visualization, allowing the inspection of the state of given data instances at any moment in time.

Visualization and analysis of vector fields and spatiotemporal data (either abstract or geographic) are research fields with their own bodies of literature, with several proposed methods to solve different tasks (WANG; WANG; LI, 2016; SANNA; MONTUSCHI; MONTUSCHI, 2000; PENG; LARAMEE, 2009; BIAN *et al.*, 2018; Kong *et al.*, 2018). In this paper, we adopt two strategies to visualize vector fields: a simple glyph-based approach, showing flow directions, and streamlines, revealing motion across the whole field. With vector field visualization, users can overview data flow inside a NN model through the analysis of instance group formation and separation.

Finally, for trajectories, we employ an approach similar to the one presented by Rauber *et al.* (RAUBER *et al.*, 2017), using bundling to group similar trajectories reducing visual clutter. However, in our approach, we define control points for bundling along the streamlines extracted from the vector field so that bundling follows the flow of data inside the network and allows analysis of the behavior of individual data instances over time. Figure 6 shows an example of these three layers of visualizations used together to represent evolution of representations in the last layer of a NN model during training. In the next section, we detail how this can be used in practice.

3.4 Results

One typical application of DNNs is data classification, which consists of inferring some model $f : \mathcal{X} \rightarrow \mathcal{Y}$ to correctly label unknown data based on a set of known labelled examples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ (LECUN; BENGIO; HINTON, 2015). This learning process is known as supervised learning, in which the model is iteratively adapted according to training examples.

Although it is well established in the literature that DNNs yield outstanding results in classification tasks for different domains, training the network and choosing appropriate parameters can still be a complex, time-consuming process. In this section, we visualize the projected activity space on a few NN models performing classification tasks with the objective

Figure 6 – Example of visual representations for projection, trajectory, and vector field components stacked together. The vector field representation overviews the whole flow of the data inside a network, supporting the analysis of instance grouping over time. The scatterplot metaphor, used to represent a projection, enables the analysis of specific instances at a particular moment in time, and trajectory bundling can be used to derive insights about instances' behavior over time. Together, these representations define a powerful visual metaphor, providing abstract and detailed views of the dynamics of neural networks.

of gaining insights on the training process and on how conclusions are drawn when performing classification.

It is important to clarify that these visualization methods are not restricted to a specific architecture or application. These models were chosen as to provide informative examples, and because the classification problem offers an easy way to keep track of instances by coloring points representing them using label data. Experiments were conducted using tensorflow(<www.tensorflow.org>) python libraries, and visualization systems were built using d3.js(<d3js.org>).

3.4.1 MNIST on the MLP model

This example shows projected data obtained from a Multilayer Perceptron (MLP) NN model, consisting of four dense layers with 256 processing units each, using ReLU activation. The output is a 10-unit layer with softmax activation to perform classification on the MNIST dataset (LECUN *et al.*, 1998). Training is done via stochastic gradient descent (SGD), using learning rate = 0.0001 and momentum = 0.9. Output data were gathered from the last hidden layer after 0, 1, 3, and 20 training epochs. Accuracy results increased rather quickly, reaching 0.9191, 0.9533, and 0.9800 on the test set after epochs 1, 3, and 20, respectively. Figure 7 shows

projected data and the obtained vector field. The projections show an expansion as the model adapts to classification, and data points move outwards to form groups, in the same way as observed by Rauber et al. (RAUBER *et al.*, 2017). This movement is reflected in the vector field.

Figure 7 – Outputs from the last hidden layer of a 4-layer MLP were gathered during training, after 0, 1, 3, and 20 epochs. Projected data refer to a sample of MNIST test set, with 2000 instances. Instance projected position at each epoch define trajectories that were used to approximate vector fields, as shown on the right side of the figure. A clear expansion pattern can be observed, as instances move outwards to form groups as the network is trained.

An organized flow of instances produces a more intense vector field, while a more disorganized movement implies frequent opposing directions and a muted vector field. This data can be used to display flow between projections: by displaying streamlines, previous movement information can be infused in the visualization without multiple plots or tracking points between projections. Figure 8 shows a projection of the last epoch of training, with streamlines indicating how groups moved to their positions.

While projecting data points over streamlines can offer important insight, additional information can be revealed by directly plotting trajectories. Previous work (RAUBER *et al.*, 2017) used trail bundling (HURTER *et al.*, 2013) to reduce visual clutter from multiple trajectories in sequential projections. We defined control points along the generated streamlines that are used to attract nearby trajectories, as can be seen in Figure 9.

The underlying vector field also produces other exploration opportunities: after optimization, there is a resulting error value for each trajectory, which indicates how well it fits to the vector field. This value can be used to explore instances that are not well represented, and identify outliers in the visualization: instances whose movement differs highly to the field surrounding it. Figure 10 shows a highlight of a few trajectories with the highest error, moving against the

general flow. Further exploration can be conducted, filtering trajectories and coloring time steps as to understand movement direction and reasoning behind the highlighted behavior.

Figure 8 – Outputs from the last hidden layer of a 4-layer MLP after 20 training epochs, using MNIST test data input, projected over streamlines. Streamlines are calculated based on three other outputs from the same layer during training, at epochs 0, 1, and 3. Activity between projections is turned into a vector field, which is then used to generate streamlines that offer additional insight into data flow during training without displaying points from other projections. For streamlines, color intensity represents flow speed (L2 norm of vectors) and arrows indicate direction.

Figure 11 shows further analysis of a highlighted trajectory. A single instance was selected and its path observed. We can quickly notice what resulted in a high error value from vector field optimization: the instance moved rather quickly and often against the overall flow, crossing large distances and changing groups. Using previous projections as reference, we can identify large instance groups in each area this instance moved to after the first time step. This means the instance was deemed similar to these groups at one point in time, and shifted rather quickly towards a different group afterwards. This behavior is particularly interesting since misclassified examples are often projected on fringe regions of groups, indicating output uncertainty, or projected repeatedly within the same group, being resistant to changes. We can then inspect the data instance to search for clues to this behavior.

3.4.2 ImageNet on MobileNetV2

This experiment shows our approach applied to a more complex combination of model and data. We used the MobileNetV2 network model ([SANDLER *et al.*, 2018](#)), pre-trained on

Figure 9 – Trajectory data from the same outputs as the Figure 8 (last hidden layer of MLP after 20 epochs using MNIST test data as input). Lines start from instance positions at epoch 0 and end in instance positions at epoch 20. Control points were set along the streamlines defined previously, attracting lines near them into bundles.

Imagenet data (DENG *et al.*, 2009). We randomly selected 10 classes ('clasp', 'delphinium', 'giant panda', 'honker', 'ice cream', 'lunch', 'newfoundland', 'outbuilding', 'puffin', 'rose') from the data set and fine-tuned a new top layer to classify them. Input images were resized to 224×224 pixels, resulting in a higher number of attributes when gathering intermediate layer outputs when compared to the previous experiment. MobileNetV2 has a depth of over 150 layers, and while it is possible to generate projections for all of them and visualize them as needed, we opted to select the last merging layer from every two network blocks ('add' layers), as well as the last global pooling layer, to simplify description. A sample of 1000 data instances from the 10 selected classes was used as input for intermediate layer output projections. All convolutional output arrays were flattened and data were projected using the aligned UMAP model described in Section 3.3.1.1. Results can be seen in Figure 12.

Our goal with this experiment was to show the forming of abstract representations and knowledge as data flows through the network. While useful information can be gathered by examining projections side by side, this visualization can quickly get out of hand, as larger numbers of projections may need to be generated to understand deeper or more complex models. Figure 13a shows a single projection of the last hidden layer, with streamlines representing previous flow in the background. Certain flow details come to light, such as movement being

Figure 10 – When visualizing trajectories, it is possible to highlight the ones that attained high error values when approximating the vector field, in search of outliers. (a) shows trajectories with the error above a certain threshold (0.1) highlighted in red, while all others are shown in black; (b) filters the projection to only instances corresponding to label "7", which seem to contain a few outliers. Data points were plotted again and colored according to which feature set they were taken from, so trajectory direction can be better understood. In this case, the highlighted trajectories move through long distances, at times in the exact opposite way of the vector field.

stronger towards the lower-left side of the image, as the right side contains classes still not fully separated (9 and 2, 5 and 6).

Finally, Figure 13b shows trajectories for this experiment. As the number of projected steps gets higher, the more cluttered the trajectory visualization becomes. This can be solved by dividing trajectories in multiple steps, such as observing how separation in the central section was achieved for the first layers and how expansion was conducted afterwards as two separate images.

3.5 Limitations

The work described in this paper has certain limitations: one of them is the fact that our projections are forced to be aligned; forcing the layout to keep an aligned state can generate distortions in projected data, especially if the analyzed feature sets offer highly conflicting distance relations or form different structures. Therefore, care must be taken as to not interpret alignment restrictions as patterns in data. The trade-off between alignment and projection fidelity also has a large impact in the generation of vector fields, as an alignment parameter configuration that is too restrictive may result in vector fields that miss too much information, while not being restrictive enough will result in vector fields that show movement that is not relevant.

Figure 11 – A single outlier trajectory is selected for examination. This data point was at the middle area at epoch 0, moved to an area containing many examples labeled as "2" at epoch 1, moved again to an area containing examples labeled "1" at epoch 3, and finally moved to another area, which contains a group of examples labeled "7", at epoch 20. We can then examine the instance to search for reasons behind this atypical behavior.

Additionally, outputs from the employed vector field approximation method are not always visually intuitive: for instance, the most intense areas (largest vector norms) in a field will not necessarily be ones with high trajectory activity. Additionally, a single vector field may not be enough to display subtleties of some networks. There is the possibility of generating a layered visualization from multiple vector fields, in order to estimate and explore more complex activity spaces.

3.6 Conclusions

In this paper, we presented a new approach for projection-based ANN hidden layer visualization that uses trajectories and vector fields to provide insights on how knowledge is generated in a DNN through training and how abstract representations are formed between layers. The presented work offers a flow-based model to represent a transition space between a sequence of projections in order to remove point-based clutter, and incorporate this model into analysis to discover useful information about model and data. Observing trajectories between layers can help understand the importance of each layer in separating certain instances in the data set, and object movement can highlight the effects of a certain training period. Exploring movement throughout the sequence can highlight objects that the model finds difficult to analyze, or objects that are not being perceived in an expected manner.

Figure 12 – Projected sequential output data from six hidden layers of pre-trained MobileNetV2 using 1000 ImageNet inputs. Projections are shown in left-to-right, upper-to-lower order. It is possible to notice that, while still concentrated in the middle area in the first upper projections, points are slowly separated into classes before finally expanding in the last two projections. On the right side is the generated vector field. A high concentration of movement in all directions in the middle area makes the resulting field rather muted, but its intensity increases in outer areas of the image.

This approach is not limited to ANN visualization, since any sequence of multidimensional projections can be explored in this manner, such as projecting any pipeline that processes data in multiple steps. The creation of a visual space that ties projections together may provide support to other visualization methods in the future.

We performed experiments that aim to show the usefulness of our method, and how it may lead to knowledge and model improvement. Our analysis was able to go further in certain aspects of the training process of neural networks, attempting to explain subtle aspects of how knowledge is generated in a DNN system.

Future research directions include investigating how projected vector fields relate to other aspects of neural network interpretation methods, such as filter visualization in CNNs, or recurrent models: almost any layer output from ANNs can be projected, and all differences between a sequence of outputs should have an effect on the resulting vector field. Work can also be conducted in finding correlations between movement observed in vector fields and features built by the network, such as identifying neurons (or convolutional filters) that are the most responsible for generating certain movement patterns in the vector field visualization.

Figure 13 – (a) projection from the last hidden layer of MobileNetV2 (global average pooling layer) over streamlines from movement trajectories of previous layers. It is possible to see paths that led from a central position to the positions occupied at the current projection, for all points; (b) trajectories of points over six layer outputs in the same network model. There is a high amount of clutter in the central section due to the resulting projections of the first layers, but certain movement patterns can be identified. Projected data are obtained using 1000 ImageNet data samples.

ACTIVATION PROFILES AND ADVERSARIAL ATTACKS

Abstract: As neural networks become the tool of choice to solve an increasing variety of problems in our society, adversarial attacks become critical. The possibility of generating data instances deliberately designed to fool a network’s analysis can have disastrous consequences. Recent work has shown that commonly used methods for model training often result in fragile abstract representations that are particularly vulnerable to such attacks. This paper presents a visual framework to investigate neural network models subjected to adversarial examples, revealing how models’ perception of the adversarial data differs from regular data instances and their relationships with class perception. Through different use cases, we show how observing these elements can quickly pinpoint exploited areas in a model, allowing further study of vulnerable features in input data and serving as a guide to improving model training and architecture.¹

4.1 Introduction

Artificial Intelligence and Machine Learning are currently among the state-of-the-art technologies to model and analyze complex phenomena, with applications in malware detection (YE *et al.*, 2017), robotics and self-driving systems (GRIGORESCU *et al.*, 2020), speech recognition (PADMANABHAN; PREMKUMAR, 2015), biometric authentication (SUNDARARAJAN; WOODARD, 2018), among others. A subset of modern machine learning technology, the deep learning models are neural network (NN) that perceive inputs as a deep hierarchy of abstract representations, modeling data beyond most heuristics and finding high-level abstract patterns (LECUN; BENGIO; HINTON, 2015).

¹ CANTAREIRA, G. D.; MELLO, R. F.; PAULOVICH, F. V. Explainable Adversarial Attacks In Artificial Neural Networks Using Activation Profiles. To be published, 2020.

Figure 14 – Proposed framework overview. Original and adversarial examples are selected as input for a DNN model. Users then use the Architecture View (A) to select sequential layers to visualize. Input examples are then mapped in the Flow View (B) to (black) lines representing the selected layers' responses (outputs). At a certain point, the distance between these lines diverges drastically, indicating the turning point in the model for the adversarial example misclassification. Users can then select the layer in which this change happens for details using the Layer View (C), which shows activation heatmaps and detailed information for each unit in the layer so that the precise units and input features being exploited by an adversarial attack can be pinpointed. It is also possible to return to the previous steps to refine selections and observe different model parts.

With the increasing adoption of deep learning models in several systems that surround human life, model security and integrity have become concerning issues (MA *et al.*, 2019; HAN *et al.*, 2020). An example of a security breach is the *Adversarial Attack* (CHAKRABORTY *et al.*, 2018), which consists of an attempt to generate input data (or modify existing data) to fool model interpretation. Adversarial attacks have been employed in almost every scenario that uses machine learning, from writing spam e-mails undetected by spam filters (NELSON *et al.*, 2008) to forging biometric data that can pass authentication (BIGGIO *et al.*, 2015). Recent research has shown that deep learning models offer several vulnerabilities to adversarial attacks (SZEGEDY *et al.*, 2013). Hence, defending against such attacks has become an important research topic, and many approaches to improve model security and robustness have been proposed, including improvements to model design, training data augmentation, input preprocessing, defensive validation, among others. Identifying vulnerabilities and addressing them also plays a vital role in obtaining a more robust model (MADRY *et al.*, 2017; MA *et al.*, 2019).

In this context, Visual Analytics (VA) can aid in developing improved models and better security. Machine learning models are often used as black boxes and benefit from explanation methods, to which VA has offered a sizable amount of solutions (HOHMAN *et al.*, 2018). Visual descriptions of how adversarial examples lead a model to incorrect conclusions may provide essential information on improving security or awareness of existing vulnerabilities. While there are many VA approaches to understanding models and visualizing inner structures, they

do not offer a comprehensive view of a model’s perception of data, which is essential when comparing adversarial examples against known patterns in the training data distribution. There are approaches explicitly designed for exploring adversarial attacks and vulnerabilities. However, they are either focused on analyzing vulnerabilities in training data distribution (MA *et al.*, 2019) or lack visual descriptors on how model interpretation evolves in a global context and how hidden layer interpretations of data relate to each other (LIU *et al.*, 2018b).

This paper proposes a VA framework to explain adversarial attacks, focusing on how adversarial data is perceived by a trained model and how model vulnerabilities are exploited on a unit activation level. We explore layers and weights inside the model to determine which areas are triggered by adversarial examples and identify vulnerabilities in input patterns. Our framework allows comparison between adversarial examples and training data, either in single control examples or groups of examples with similar characteristics. This comparison can be made at different levels so that users can pinpoint diverging paths and obtain insights over which areas to explore in detail, identifying patterns to be reinforced in further training or individual units with undesired behavior.

In short, our main contributions are:

- A VA framework that explores how adversarial attacks interact with inner elements of a neural network;
- Activation Profiles, a similarity-based representation of models perception of data at different hidden layers, allowing for the visual comparison of different data instances; and
- Different visual metaphors for condensing groups of activation data, enabling visual understanding of how classes are defined and separated inside a model.

4.2 Related Work

Visual Analysis of Neural Networks. Machine learning (ML) models have been widely successful in many different applications over the past years. However, with their continually increasing adoption, a problem has become more pronounced: the black-box nature of such models makes them difficult to interpret, hindering processes such as implementation, improvement, replacement, or adaptation (HOHMAN *et al.*, 2018). Several efforts have been made by the machine learning and visual analytics research communities directed at improving model explainability, comprehending a field known as Explainable Artificial Intelligence (XAI) (ADADI; BERRADA, 2018).

Visual explanations for ML models, in particular deep neural networks (DNNs), have different approaches and goals, offering visual representations of various aspects of a model. In a 2018 survey, Hohman et al. (HOHMAN *et al.*, 2018) classify existing techniques according

to the end goal, visualized data, moment of visualization, intended user, base visual metaphor, and application. Conversely, Liu et al (LIU *et al.*, 2017b) and Choo et al. (CHOO; LIU, 2018) categorize techniques according to their perceived main goal: model understanding, performance diagnosis, and model refinement. In this context, many techniques fit multiple categories, as the primary goal of XAI approaches is to provide understanding leading to improvements.

Model understanding techniques allow users to investigate a model's analytic process and why it performs in a certain way. They can be model-independent, focusing on broader aspects of neural network functions (WONGSUPHASAWAT *et al.*, 2017; KAHNG *et al.*, 2017), or focused on understanding a specific network architecture, such as Recurrent Neural Networks (RNNs) (KWON *et al.*, 2018a; MING *et al.*, 2017; KARPATHY; JOHNSON; FEI-FEI, 2015; LI *et al.*, 2015; STROBELT *et al.*, 2017), Convolutional Neural Networks (CNNs) (LIU *et al.*, 2017a; PEZZOTTI *et al.*, 2018; LIU *et al.*, 2018a; Wang *et al.*, 2020), Q-networks (QNNs) (Wang *et al.*, 2019), or Generative Adversarial Networks (GANs) (KAHNG *et al.*, 2018b; Wang *et al.*, 2018). In common, they present structural views of how the model processes data and how it behaves when subjected to a specific data instance or set of instances. Certain techniques are also directed towards specific data types, as they are often related to the network layout (NIE *et al.*, 2018). Most visual representations are based on heatmaps, graphs, and multidimensional projections representing weight values, activation outputs, and network layout. Information can be displayed in a weight-centric, data-centric, or neuron-centric fashion, depending on the analysis at hand. Many techniques are mainly defined by how these representations are joined to provide relevant information for a particular task.

Performance diagnosis techniques are designed to help comprehend model training and efficacy at a given task, allowing users to identify problematic data or pinpoint causes of a failed training process that are not easily summarized by metrics such as accuracy or recall (AMERSHI *et al.*, 2015). Model refinement is often achieved by incorporating additional user knowledge to an existing model, and certain techniques are designed to help users identify improvements or fill gaps in areas of uncertainty in input space. Previously mentioned techniques offer debugging tools for interactive refinement and observation of model behavior (WONGSUPHASAWAT *et al.*, 2017; STROBELT *et al.*, 2017), while others focus on iteratively providing performance explanations through instance or attribute-based representations (KRAUSE *et al.*, 2017; KRAUSE; PERER; BERTINI, 2014; REN *et al.*, 2016; ZHANG *et al.*, 2018). These methods may offer insights when used in the context of adversarial attacks but lack a straightforward way of relating this information to changes observed in final output behavior when analyzing adversarial examples.

A common approach to explaining internal abstract structures in neural networks is to reflect intermediate model outputs into input space. This approach is often associated with image data and CNNs. Several techniques are dedicated to analyzing the distribution of processing unit activations for an input image and generating heatmaps of each pixel's influence in the

model’s final decision, also known as *saliency maps* (KINDERMANS *et al.*, 2017; SIMONYAN; VEDALDI; ZISSERMAN, 2013; SAMEK *et al.*, 2016; MAHENDRAN; VEDALDI, 2016a; MONTAVON *et al.*, 2017; BABIKER; GOEBEL, 2017; ANCONA *et al.*, 2017; SELVARAJU *et al.*, 2017). Other techniques are focused on optimizing an image to display patterns recognized by the network, such as maximizing layer activations (YOSINSKI *et al.*, 2015; NGUYEN; YOSINSKI; CLUNE, 2016), using another trained model to generate images from intermediate layer outputs (ZEILER; FERGUS, 2014; ZEILER; TAYLOR; FERGUS, 2011), or applying activations from different images to perform style transfer (MORDVINTSEV; OLAH; TYKA, 2015). Feature explanation frameworks such as SUMMIT (HOHMAN *et al.*, 2019), delve directly into visualizing layer activations, aiming to correlate classification, descriptive features, and processing unit activations inside hidden layers to understand the meaning of specific activation patterns. In the context of our work, the information provided by these techniques is useful in translating an exploited adversarial vulnerability into patterns in input data. However, as modern models become larger and more complex, identifying which layers or processing units to visualize is not a trivial task. Additionally, showing patterns over input data is not enough when studying adversarial attacks as the origin of such patterns is also a subject to be investigated.

Our framework’s primary goal is to generate high-level representations of a model connected to low-level feature explanations to quickly pinpoint features and processing units being exploited by an adversarial example. The concept of visualizing the overall flow of data inside models has been approached by some authors and model understanding techniques (CANTAREIRA; PAULOVIK.; ETEMAD., 2020; HALNAUT *et al.*, 2020; LIU *et al.*, 2017a), but comparisons between data examples with the provided level of information are limited. In particular, Halnaut et al. (HALNAUT *et al.*, 2020) present a method that clusters data instances at each layer output and then displays layers as a sequence of vertical axes, with object exchanges between clusters at each layer shown as a Sankey diagram. However, each layer’s visual space is not consistent, and cluster positioning is arbitrary, which would lead to inconsistent representations for classes and make comparisons between instances difficult in our scenario.

Adversarial Attacks and Visual Analytics. An *adversarial attack* is a manipulation of input data to force a trained ML model to produce incorrect outputs (CHAKRABORTY *et al.*, 2018). In the context of neural networks, Szegedy et al. (SZEGEDY *et al.*, 2013) discovered that several state-of-the-art NN models were vulnerable to *adversarial examples*, i.e., manipulated input data that would be misclassified by the model, despite being largely similar to correctly classified examples drawn from the data distribution. With the adoption of deep learning in an increasing variety of applications, the subject of adversarial attacks has drawn the research community’s attention, with many recent developments in both building models with better security and producing more sophisticated attacks.

Adversarial machine learning literature describes four main features of an adversary:

goal, knowledge, capability, and strategy (BIGGIO; ROLI, 2018). An attack is often described by its operation regarding one of these features. *Black-box* vs. *white-box* attacks, for instance, refer to adversary knowledge of model details. Attacks can be conducted before or after a model is trained, depending on the adversary’s goal and access capability: *Poisoning* attacks manipulate training data to induce a particular behavior in the optimized model, such as misclassifying examples (MUÑOZ-GONZÁLEZ *et al.*, 2017) or providing a backdoor when a specific pattern is encountered (GAO *et al.*, 2019). Conversely, *evasion* attacks target an already trained model during run time, aiming to force an incorrect output (KWON *et al.*, 2018b).

Recent research shows sophisticated approaches to producing adversarial examples, from methods to add undetectable triggers in training images (SAHA; SUBRAMANYA; PIR-SIAVASH, 2019; LIAO *et al.*, 2018) to generating misclassified examples by altering a single pixel (SU; VARGAS; SAKURAI, 2019). Efforts in explaining the causes behind a model’s vulnerability have also been made, such as analyzing robustness concerning output functions (CARLINI; WAGNER, 2017b; OZBULAK; NEVE; MESSEM, 2018; ROZSA; GÜNTHER; BOULT, 2017), or showing how adversarial vulnerabilities can reveal insights on training data (ILYAS *et al.*, 2019).

Currently, literature comprehending visual methods to interpret and understand the causes and effects of adversarial attacks in neural network models is quite limited. Ma *et al.* (MA *et al.*, 2019) proposed a framework to explain classifier vulnerabilities through high-dimensional relationships between training data examples. However, this framework does not provide information regarding how such vulnerabilities relate to abstract representations formed inside a model, which may be vital in determining model trustworthiness (RIBEIRO; SINGH; GUESTRIN, 2016). Liu *et al.* (LIU *et al.*, 2018b; Cao *et al.*, 2020) proposed AEVis, a VA tool that explores models perception of adversarial examples by extracting paths taken by data inside a network and allowing comparison in multiple levels (layer, feature map, neuron). While the multi-level aspect of this approach is informative and in ways similar to what we present in our framework, it has some limitations. Although datapaths can identify the most critical units towards a particular classification at each layer, guidance as to which layers may be of interest (namely, *diverging points* where the model perception of data instances changes drastically) is limited, as it only shows cosine dissimilarity between outputs at each layer using dots. Additionally, its explanation process is focused on the observation of robust features, and understanding non-robust features may be crucial when investigating adversarial examples (ILYAS *et al.*, 2019).

4.3 Design Goals

With the problem at hand and limitations of existing approaches in mind, we devised a framework to explain adversarial attacks in an internal representation context through VA. While

the concepts presented here can be employed in other applications, our framework is based on CNN models for image classification, which is a common subject in literature (KRIZHEVSKY; SUTSKEVER; HINTON, 2012; KRIZHEVSKY; HINTON, 2010). We also focus our discussion on *evasion attacks*, i.e., when the attacker’s goal is assumed to be manipulating an image to provoke incorrect classification. In this paper, adversarial examples are generated with full knowledge of the model (white-box perspective).

After reviewing the literature on evasion attacks (Yuan *et al.*, 2019; Xu *et al.*, 2019; AKHTAR; MIAN, 2018; Jiang *et al.*, 2020; KWON *et al.*, 2018b) and adversarial image generation (GOODFELLOW; SHLENS; SZEGEDY, 2014; MADRY *et al.*, 2017; ATHALYE *et al.*, 2018), how they relate with XAI approaches in VA, as well as discussing with an ML expert, we defined the following tasks to be performed by our framework:

- T1. **Summarize Model Perception of Adversarial Examples.** Our goal is to provide an understanding of how an adversarial example exploits the inner properties of a neural network. To achieve this, our first step is to provide a global view of hidden outputs so that the user can quickly grasp model behavior when exposed to a given input, as well as perform comparisons between adversarial and regular data examples so that points of abnormal behavior can be quickly identified and further examined.
- T2. **Identify Vulnerable Features.** Once an abnormal behavior point is chosen, we must be able to delve into the model’s processing units to identify which features or filters were the most responsible for the observed difference, and therefore the most exploited by the attack. The user can then visualize these features to correlate them with their expertise or identify robust vs. non-robust features.

From the analytical tasks, we devise a set of VA requirements to be fulfilled:

- R1. **Highlight areas of interest.** There are visualization techniques focused on providing an understanding of layers or features, but visiting each layer and feature is impractical (visually and computationally) when dealing with models with a large number of layers or processing units per layer. Therefore, a way to profile global data flow for a given input to identify and explore where pivotal changes happen is needed.
- R2. **Visual comparison between data flows.** Besides being global, data flow through hidden outputs must also be comparable in a simple manner. It is necessary to understand how the model perceives regular data and adversarial examples and pinpoint locations where divergences are present.
- R3. **Visual profile for data groups.** When observing data flow in a classification model, it also becomes necessary to represent output patterns from multiple input data, such as

classes. To achieve this, we need a visualization for the accumulation of outputs from several data instances, displaying taken paths and density.

- R4. **Feature exploration.** Once we can pinpoint a layer where substantial differences occur, we then need to understand how they came to be. A detailed examination of the actual meaning of features and their impact on data separation is needed.

4.4 Framework Structure

4.4.1 Overview

Our framework is divided into three main components, presented in Figure 14. The first is the *Architecture View* (A), which displays the model under analysis topology as a compact graph representation acting as the starting point where users select a sequence of layers to display. The second is the *Flow View* (B), which shows data objects progressing through the model's selected layers. These data objects can be displayed individually or in groups. The selection of layers displayed at the *Flow View* can be changed at any moment, with the possibility of selecting a set of representative layers from the full extension of the model, visualizing them to find an area of interest, and then making a new selection with layers from that area to explore in more detail (tasks **T1**, **T2**). The last component is the *Layer View* (C), in which users can choose a single layer to explore on a per-unit basis. Here, users can observe outputs from every unit in the layer, identifying patterns for groups, or comparing outputs between individual data objects (task **T2**).

4.4.2 Architecture View

To help users select which layers to visualize, we first present a compact graph representation of the entire model, similar to the computation graph proposed in (KAHNG *et al.*, 2017) or TensorBoard's graph view (<<https://www.tensorflow.org/tensorboard>>). This graph is organized horizontally from input to output layers. Layers with external inputs or outputs are represented as circles, while other layers are represented as squares linked to their inputs on the left side and outputs on the right side. They are color-coded according to type: layers that do not have any trainable parameters are colored in gray. Connected layers with trainable weights (e.g., dense or convolutional layers) are colored in blue, and other trainable layers (e.g., batch normalization) are colored in red. Multiple layers can be grouped in blocks, which are shown as white rectangles. Branching connections (layers with multiple input or output tensors) between blocks are kept visible since a selection of layers that is non-sequential should not be displayed at the *Flow View*. This representation is designed to occupy as little space as possible while providing models overview.

4.4.3 Flow View

An essential task in our framework is to provide a representation of models perception of adversarial data to find points of interest to zoom in further and explore in detail (task **T1**). Highlighting layers that significantly improve an example's similarity towards a target group or further separate it from its original class accelerates visual analysis and avoids users having to explore a model on a layer-by-layer basis (requirement **R1**). This task can be performed using the *Flow View* component, which offers visual representations to support layers' interpretation given an input data set.

Figure 15 – *Flow View* overview. Input data is processed by the model and hidden outputs of selected layers are transformed into 1-D arrays using a dimensionality reduction approach, resulting in *Activation Profiles*. *Activation Profiles* are then used to build visualizations, enabling the comparisons of regular and adversarial examples considering the model's perception.

Once a set of layers $L = (L_1, L_2, \dots, L_t)$ is selected at the *Architecture View*, input data is processed through the model, and outputs at each layer are displayed. In our framework, we analyze a set of input data X considering two different perspectives: *background* and *foreground* data. Background data provides context to the visualization, being displayed as groups (requirement **R3**). Foreground data are objects to be displayed individually: adversarial and control examples (requirement **R2**).

To show how different layers perceive data objects and how this perception evolves as data travels further in the model, we build *Activation Profiles*. For every layer $L_i \in L$ and data object $x_j \in X$, hidden outputs $L_i(x_j)$ are transformed into scalar values $p_i(x_j)$ whose distances in $\mathbb{R}^1$ space represent dissimilarity relationships in $L_i(X)$ as best as possible. For each x_j , then, there is a 1-D array $p(x_j) = [p_1(x_j), p_2(x_j), \dots, p_n(x_j)]$ containing its interpretation according to each layer in L . As stated in Sec. 4.4.2, layer selection should follow a sequential pattern, so that outputs from L_i are in some way part of the input in L_{i+1} even if they are not immediately connected. Non-sequential models can be explored using the framework, as long as the selected layers are sequential.

To transform multivariate layer outputs into scalar values, we employ UMAP (MCINNES; HEALY; MELVILLE, 2018). Although the information contained in hidden layer outputs cannot be fully represented in one dimension, full distance preservation is not our goal. We guide the projection process to focus on displaying distance relations that change between layer outputs,

preserving relative cluster positions, and providing relevant information for comparison. This is achieved by applying an alignment constraint (CANTAREIRA; PAULOVICH, 2020) to all projections. This constraint ensures that overall cluster positioning is similar in all projected layers, providing context for changes between projections (requirement **R1**). Additionally, alignment indirectly favors accumulating error in the same pairwise distances across all projections, meaning that if a distance between two points cannot be properly replicated but does not display expressive change across projections, its representation will remain relatively stable.

After *Activation Profiles* $p(X)$ values are obtained, we build the *Flow View*. Foreground data are visualized directly: selected layers are shown as a sequence of vertical axes, and each data object is represented as a line connecting its positions at each axis, similar to a Parallel Coordinates (HEINRICH; WEISKOPF, 2013) visualization. Background data are shown in the same way, but lines are colored according to label (either true or predicted) and curved and bundled together (ZHOU *et al.*, 2013) using label and initial and final positions between each pair of layers as clustering inputs in the bundling process. This representation, which we call *Background Flow*, allows for direct comparison between flow patterns from different groups (requirement **R3**) and offers a non-intrusive background to provide context when comparing two individual instances (requirement **R2**).

Background context is essential to providing meaning to object positioning. When comparing only two data objects, the default approach measures and shows distances between them at each layer output (LIU *et al.*, 2018b). However, layer outputs may vary, both in the number of dimensions and magnitude (especially when considering factors such as vanishing or exploding gradients), and correctly normalizing values to enable proper comparison may become a problem. Additionally, only stating that objects became closer or further apart is not enough – by examining them in contrast to background groups, it is possible to notice if such distancing is atypical, or if an object became closer to a different group (requirements **R2** and **R3**). In our framework, background and foreground data are projected as a single dataset. UMAP is designed so that a projection can be fit to background data and then used to transform future data observations, allowing exploration and comparison of multiple examples while maintaining the same context (requirement **R3**).

In addition to *background flow*, we devise two more visualizations for background data to complement the representation of classes and other groupings. The first, *profile histogram*, is composed of histograms over each layer axis, providing a clear notion of density at different parts of a line. The second consists of joining all histograms for each class in a heatmap providing a compact visual descriptor of each class that can be compared side-by-side, which we call *Class Fingerprint* (requirements **R2** and **R3**).

Figure 15 shows the process for visualizing data in the *Flow View*. Using this view, users can iteratively select different layer combinations to observe how a model’s perception of objects evolve and pinpoint changes in behavior. Once a particular layer of interest is found, it can be

displayed using the *Layer View*.

4.4.4 Layer View

The *Layer View* is focused on displaying output information $L_s(x)$ from a single layer L_s , selected in the *Flow View*, to allow users to investigate the causes behind changes in behavior, down to individual unit (filter) activations. Visual metaphors employed in the *Layer View* merge and extend ideas presented in (KAHNG *et al.*, 2017; HOHMAN *et al.*, 2019). Analog to the *Flow View*, outputs are obtained from a set of input examples X ; either foreground or background data can be selected for visualization. Units can be selected and inspected individually, through weight display and filter visualization (YOSINSKI *et al.*, 2015). Figure 16 illustrates this process.

Figure 16 – *Layer View* overview. Outputs from a selected layer are subjected to reduction to obtain a single output equivalent, representing activation data. This data is displayed using heatmap visualizations to highlight units with interesting behavior. Units can be selected and displayed using convolutional filter visualization and highlighting weights in their connections. Activation data can also be exported for further statistical analysis.

The *Layer View* produces visual representations from outputs $L_s(x_i)$ from a single object x_i at a time. We formulate it so that the output set $L_s(X)$ is subjected to one of four reduction operations, producing the activation data $L_s(r)$, which has the same dimensions as an output for a single data instance. The choice of operation depends on the exploratory task to be conducted:

- **mean()**: corresponds to the mean output values at each unit for the outputs $L_s(X)$, which we use to determine activation patterns for classes and large groups.
- **max()**: maximum values in $L_s(X)$ at each processing unit in L_s . This output is used for examining layer-class combinations where small subsets of X produce high outputs in different units spread throughout the layer.
- **difference(i, j)**: output with unit-wise difference ($L_s(x_i) - L_s(x_j)$) between two outputs. This is the operation we use to compare outputs between an adversarial image and its original counterpart, as the resulting output contains the units most affected by the adversarial manipulation.
- **direct_output(i)**: allows the observation of a single given output $L_s(x_i)$.

The *Layer View* is then composed as a heatmap. All units are arranged in a square grid and colored according to their output value (in the case of 2D convolutional units, we pick the maximum activation from each matrix). Besides the heatmap, activation maps for convolutional units are also shown, consisting of a grid of output images for units in L_s that show the highest activation values, indicating areas that generated the strongest activations in the layer.

Filters can then be selected and visualized through activation maximization (ERHAN *et al.*, 2009). An input image with visual patterns that stimulate filter $L_{s,k}$ is generated from random noise and displayed. Similar images are also generated for filters in L_{s-1} , L_{s+1} whose connections to $L_{s,k}$ have the strongest weights, represented as lines connecting them. In this way, users can grasp which abstract representations are the most influential in activating $L_{s,k}$, and which representations are the most influenced by $L_{s,k}$, respectively. Additionally, we can export the generated activation data to perform further analysis, such as building graphs or obtaining more statistical information (ranking units by output, identifying maximum, minimum, and standard deviation in outputs, inspecting covariance between units for multiple outputs from background data, among others).

4.5 Study Cases

In this section, we demonstrate the application of our framework in two study cases. The first case presents an evasion adversarial attack study over a small CNN architecture for classifying MNIST handwritten digits (LECUN *et al.*, 1998). The second shows vulnerable features and units in a deep model, the MobileNet V2 (SANDLER *et al.*, 2018) trained on ImageNet (DENG *et al.*, 2009) data. The framework front-end was implemented using d3.js², with tensorflow³ and keras⁴ as back-end.

4.5.1 Adversarial Data Generation

Adversarial machine learning literature defines an adversarial example x_i^{adv} as the result of applying a perturbation s (based on data distribution) over an example x_i to obtain an incorrect outcome from a model. For image classification purposes, perturbations come from a set of allowed perturbations S that is commonly chosen to ensure that modified examples are still perceptually similar to the originals, such as being bound to an l_2 or l_∞ ball around original examples.

In this paper, we use Projected Gradient Descent (PGD) (MADRY *et al.*, 2017) to generate adversarial data. A gradient-based method originating from *Fast Gradient Sign* (FGSM) (GOOD-FELLOW; SHLENS; SZEGEDY, 2014), PGD is considered a universal first-order adversary,

² <<https://d3js.org/>>

³ <<https://tensorflow.org/>>

⁴ <<https://keras.io/>>

i.e., its output tends towards the strongest possible attack with local first-order information from a network model. Several studies propose adversarial PGD in training to improve model robustness (MADRY *et al.*, 2017; ARAUJO *et al.*, 2019; SHAFABI *et al.*, 2019; WONG; RICE; KOLTER, 2020), with remarkable results. We note that other methods for adversarial data generation (CARLINI; WAGNER, 2017a; XIAO *et al.*, 2018; MOOSAVI-DEZFOOLI; FAWZI; FROSSARD, 2016) could provide their own set of properties to be visualized and explained as well.

4.5.2 Explaining Adversarial Examples

Our first study case consists of the interpretation and exploration of evasion attacks on a small convolutional model. The model is a 4-layer CNN, trained for image classification on the MNIST handwritten digit dataset. Each layer has a 3x3 filter, ReLU activation, followed by a max-pooling layer of stride size 2, effectively halving output dimensions. The number of units at each layer is 4, 8, 16, and 32. Following the last convolutional layer, a global max-pooling layer is linked directly to a 10-unit softmax output.

The training was conducted using Adam optimization (initial learning rate = 0.001), and accuracy results reached 99% for training and test sets after 10 epochs.

A 2,000-object sample X from the training data set was drawn at random as background data to establish a baseline for projections. *Activation Profiles* from this sample in each of the weighted layers (four convolutional, one dense for output - captured before softmax activation) are shown in Figure 17. The *Architecture View* is shown on the upper area, allowing selection of layers to be observed, while the *Flow View* provides information on how each layer affects relationships and separation between different classes. Each class's paths are condensed using *Class Fingerprints*, shown on the right side. In this Figure, it is possible to notice that separation between class groups may occur at different points of the model, depending on the class. For instance, classes “9” and “4” only get properly separated at layer *conv2d_3*, while class “1” has many of its instances already grouped at layer *conv2d_1*. Class fingerprints get brighter as instances are condensed in a region of the vertical axis, indicating higher separation. An adversarial set X^{adv} was created by producing an adversarial image from each object in X . Adversarial data were generated using PGD, with parameters $\epsilon = 0.3$, $\alpha = 0.075$ and 100 iterations. When predicted by the model, accuracy for the adversarial sample was 1.6%.

Next, we pick an image x_0 from X to explore the model's reaction. We run it through the model and project its outputs on the flow view along with its adversarial counterpart, x_0^{adv} , and observe how their behaviors change at each layer. Figure 18 shows this comparison. x_0 had its label (“0”) correctly predicted by the model, but x_0^{adv} was predicted as belonging to the class “9”. Both examples are projected alongside background data from their perceived and correct classes. It is possible to observe that, while similar up to the second convolutional layer (*conv2d_1*), x_0 and x_0^{adv} becomes vastly different afterward.

Figure 17 – Background data visualization for the CNN-MNIST case study. The upper diagram shows the Architecture View for the model, indicating which layers are selected for view. The line plot shows lines for all background data objects, colored by their predicted label. Class separation may occur at different layers. Many objects in classes “9” and “4” (brown and orange, respectively) only get properly separated at the fourth convolutional layer, while the second layer already separates many objects in class “1” (cyan). Class fingerprints, shown on the right, display the accumulation of points at each segment of the visual space, highlighting condensed areas with brighter colors.

At the fourth convolutional layer ($conv2d_3$), x_0^{adv} is already grouped with other objects in class “9”. As highlighted in Figure 18, There is a *diverging point* between $conv2d_1$ and $conv2d_2$ where the relationship between both outputs changes drastically, and it is likely to be where units most affected by the adversarial manipulation in x_0^{adv} are located. This behavior was observed in all experiments we performed; while output similarity between a given x_i and its adversarial deviation x_i^{adv} may vary at the first layers of a model due to perturbations being detected by low-level units, activations at the middle sections see the objects as somewhat similar. When the diverging point is reached, there is a sudden shift, where the two objects become completely separated, and the adversarial example assumes behavior more similar to the perceived (incorrect) class. Having observed where this shift is located, we can proceed to

Figure 18 – Lines representing original image x_0 and adversarial image x_0^{adv} being interpreted by the CNN-MNIST model. Background data representing classes “0” and “9” is also shown. The two images are perceived as relatively similar until layer $conv2d_2$, when separation occurs and each example merges into a different class. This *diverging point* highlights layer $conv2d_2$ as a point of interest.

examine layers at both ends of the diverging point in detail.

As this model is narrow and shallow, it is possible to observe all unit outputs for comparison. Figure 19 shows outputs at layers $conv2d_1$ and $conv2d_2$ from both images. For each layer, there are two rows. The top row contains complete convolutional layer outputs, with one resulting image for each unit, showing how strong units reacted at each pixel. The bottom row shows *unit maximum* heatmaps colored with the strongest activation among all pixels in that unit. The third column is the difference between activations from x_0 and x_0^{adv} . Activations that are stronger in x_0 are colored in red, while activations stronger in x_0^{adv} are colored in blue. The fourth column shows a visualization of input patterns that activate each unit in that layer.

The layer that precedes the large shift in x_0^{adv} 's positioning, $conv2d_1$ (top row), shows largely similar maximum outputs among all units, with only one of them (top right) showing expressive differences. The unit maximum heatmaps from $conv2d_2$, however, show many differences between outputs from the two examples. Looking at the activation maps for $L_i(x_0) - L_i(x_0^{adv})$, it is possible to notice that, in $conv2d_2$, units that show stronger activations for x_0 seem to be concentrated at the lower left side of the image, while stronger activations for x_0^{adv} are generally located on the center-right side. Analyzing the filter visualizations (fourth column), x_0^{adv} appears to be increasing the response of vertical ramification patterns on the right side of the image, which favors label “9” while decreasing response on curved patterns on the lower left side of the image ($conv2d_2$, units 0 and 13), which reduces the odds of a label “0”.

In additional exploration, in Figure 20, we export activation data and plot the difference between outputs from the two examples at each unit and the differences between mean activations

for background data in classes “0” and “9”. Our goal is to identify units in which differences in activation between classes “0” and “9” were most reflected in our adversarial example. We can identify four units (0, 11, 13, and 14) where changes in x_{adv} follow the same pattern as in class activations. These are not the units with the highest mean activations for each class, representing secondary patterns. Interestingly, the adversarial example also generates highly different activations in certain units (such as 5,6, and 7) that seem to have no impact in changing predictions between the two classes.

Figure 19 – Visualization for layers *conv2d_1* and *conv2d_2* of the CNN-MNIST model. The first column identifies each unit in the layer. Second and third columns show activations for x_0 and x_0^{adv} from the two observed layers, and the fourth column shows the difference in activations between the two. Finally, the fifth column visualizes all filters in each layer using activation maximization. Activations from x_0 and x_0^{adv} at the first layer are still relatively similar but show much more variation in the second. Many activations appear to be stronger on the center-right side for x_0^{adv} , and stronger on the lower-left side for x_0 .

4.5.3 Vulnerable Features on Deep Networks

In this case study, we explore hidden layer outputs of a deeper model, locate its diverging point for an adversarial example, and observe how visual features relate to its layer’s units. The studied model is *MobileNet V2*, an architecture designed for image data analysis on mobile devices. Keras *MobileNet V2* implementation has 3,538,984 parameters and 157 layers. Our model was trained on *ImageNet* images for data classification into 1,000 different classes, with the output layer positioned immediately after a global average pool of the last convolutional layer.

Figure 20 – This graph shows the difference between outputs at each unit, for the two examples as well as the two relevant classes. The adversarial example accentuates certain differences in activation (highlighted in red), which can be translated into visual features in Figure 19, but there are many large differences in activation that do not match class patterns.

Base image x_0 for this study consists of a giant panda photo from *ImageNet*, as seen in Figure 21. The model is capable of predicting its label correctly, with 93% confidence. From this image, we generate an adversarial example x_0^{adv} using PGD, with $\epsilon = 0.3$, $\alpha = 0.075$ and 200 iterations. When analyzed by the model, x_0^{adv} was misclassified as 'kelpie,' with 99% confidence. We use our framework to compare how both objects are perceived and identify where pivotal differences begin to occur.

Since the model is too deep for all layers to be displayed at once in the *Flow View*, we use the *Architecture View* to make a selection of layers of interest. *MobileNet V2* is divided into blocks and has residual components that merge at each block's end. We choose to visualize all *add* layers in the model (10 in total), as they are sequential and evenly spaced enough to provide an approximation for the whole architecture. We also visualize the last hidden output in the network (global average pooling before the dense output).

Background data is also essential for the *Flow View* visualization. Since it will provide context for the two examples being compared, the choice of background data must contain information relevant to the analysis at hand. While it is possible to draw a sample from all 1,000 classes in *ImageNet*, a large amount of space would be occupied by objects and features that are not relevant for this comparison. On the other hand, using only data from the two involved classes would not provide enough context of what separation between objects might mean. Selecting data with known shared characteristics may lead to important details being highlighted. Therefore, we opt to use 2,000 examples from five different animal-related classes as background data: 'king penguin', 'eagle', 'golden retriever', 'giant panda', and 'kelpie'.

Figure 21 shows the projected output values for both examples and background data

Figure 21 – *Flow View* displaying data flow from two images (control example x_0 and adversarial example x_0^{adv}) plus background data. The upper image contains layers from several parts of the model. A diverging point can be observed in the area highlighted in red, as the two trajectories suddenly branch in different directions between layers *block_14_add* and *block_15_add*. All layers contained between these two can then be selected for display, resulting in the lower image. The flow visualization allows investigation and discovery of the exact moment when the two images' model perception splits apart.

at each of the 11 selected layers. Here, we opt to show projected background data only as *profile histograms* colored by predicted class for a cleaner representation. Outputs from the two images are considerably similar in the first half of the model. This image's diverging point is easily identifiable: at a certain point between layers *block_14_add* and *block_15_add*, model perception of x_0 and x_0^{adv} becomes drastically different (area highlighted in red).

In *MobileNet V2*, block 15 is a sequence of layers that branch off the output from block 14. At the end of the block, the layer *block_15_add* merges its output back with the unmodified output from block 14. Since x_0 and x_0^{adv} are still viewed as similar at *block_14_add*, we can assume that the change resides in this sequence of layers. We select all layers in block 15 and visualize them, generating the image on the bottom half of Figure 21. With this visualization, we can observe the effects of connected convolutional or dense layers and supporting layers, such as batch normalization or additional activation layers.

From Figure 21, it is possible to notice a clear separation at *block_15_project*, so we

Figure 22 – Analysis of outputs for x_0 and x_0^{adv} at the layer *block_15_project* of *MobileNet V2*. a) shows the 20 units with the most expressive changes in perception between the two examples (highest absolute difference). Blue and red represent positive and negative outputs, respectively. b) displays the absolute output sum for each of the 20 units on each data example. Units 34, 143, 89, and 152 (circled in black in both images) show high overall outputs and high variations, being more likely to be relevant for classification for objects x_0 and x_0^{adv} .

Figure 23 – Image patterns detected by the four most relevant units in layer *block_15_project*, as identified in Figure 22. Correlating filters and visual features can help explain how an adversarial example is able to exploit a CNN model, especially concerning the distinction between robust and non-robust features.

select it for inspection using the *Layer View* (Figure 22). Outputs on this layer are slightly more complex to visualize due to negative values that can be carried over to the next weighted layer without passing through a rectified linear function. We build activation heatmaps in the top row by calculating the sum of all absolute output values for each unit. In these heatmaps, it is noticeable that, while changes happen between the two objects, they are in the middle of the range, while the highest outputs remain the same. We calculate the total difference between all unit outputs for x_0 and x_0^{adv} , and select the units with the 20 most diverging values. Their

activation maps are displayed in the second row, with positive outputs in blue and negative outputs in red. It is possible to notice that many of the units perceive differences between x_0 and x_0^{adv} at the center of the picture.

We then export activation data and use a graph to correlate the output sum at each unit with units that show the most diverging outputs for the two images to find units where intense changes occur and, at the same time, produce strong outputs since large output values are most likely to be used for classification. Four of them were highlighted due to visually distinct behavior. They are circled in black, and activation maps for these units were also highlighted in black in the second upper row. Two of them (34, 143) show stronger outputs for x_0 , while the other two (89, 152) show stronger outputs for x_{adv} .

Using filter visualization, these units are then visualized in input space, as can be seen in the last row of Figure 22. These are the visual features captured by the most divergent filters in this layer, which can then be correlated to patterns observed in the images themselves. An analysis can be performed as to their relevance in visually identifying the original images to determine the presence of robust vs. non-robust features. Model improvement regarding robust features can be achieved by performing further training using images that clearly show these patterns, while non-robust features may reveal insights on dataset bias and overfitting.

4.6 Discussion and Limitations

Our approach's main limitation is that the effectiveness of the *Flow View* representations depend on how representative the background data is. We conducted experiments with randomly sampled data from smaller training datasets such as MNIST or CIFAR10 (KRIZHEVSKY; HINTON *et al.*, 2009) and empirically observed that sample sizes as small as 1,000 instances generated *Activation Profiles* similar to the complete dataset. However, for large datasets such as the *ImageNet*, it is not viable to draw random samples from the whole data distribution, as discussed in Section 4.5.3. Therefore, it is crucial to draw background data from samples that are somewhat representative of the training dataset's subdomain, whose features are being observed.

Another limitation in the *Activation Profiles* is that the vertical axis carries no inherent meaning in our visual representations. Visual analysis is based on distances between points and how observed data relate to background groups. However, the fact that a specific group is displayed above or below another is simply a byproduct of the distance minimization process. It is possible to add information to this axis by controlling alignment and initialization, but it is out of this paper's scope.

Finally, our analysis over which units are exploited by an adversarial example can display locations where the greatest differences are detected, but it does not guarantee that these units are responsible for classification. It is known that the presence of high activation values does not necessarily mean that a given unit has a strong influence in model output for that object (LIU

et al., 2018b). However, observing why specific units display different behavior towards the adversarial example despite not being relevant to that classification might provide insights into this analysis. Combining our framework with other approaches, such as AEVis (LIU *et al.*, 2018b), could provide informative results.

4.7 Conclusion

This paper presented a novel visual analytics framework capable of summarizing model perception of input data and highlighting differences between adversarial and control examples. Users can acquire insight over areas of interest that can be observed in detail by examining hidden layer outputs and components of its inner structure, such as layer weights or convolutional filters. We show its application in two study cases, a small benchmark data/model combination and a large dataset/deep network model scenario more akin to real-world applications.

Our framework comprehends a novel visualization approach (the *Activation Profiles*), as well as building upon state-of-the-art techniques to provide detailed visual information (*Architecture View* and *Layer View*). This framework is intended as a guide to provide insight into a model’s behavior, as opposed to direct suggestions for model improvement/debugging. Its utilization can be combined with other tools for a detailed mental picture of how representations are formed within hidden layers and how adversarial examples exploit these structures to provoke unexpected outputs.

Future work may follow different paths. The framework can be extended to provide full support for non-sequential network architectures (such as RNNs) and comparison between different models. The *background flow* and *class fingerprint* metaphors offer insight in understanding group data behavior, and their utility can be investigated in different contexts, such as supporting training and model topology design.

CONCLUSION

5.1 Work Overview

In this thesis, the researcher presented work done over the course of a doctorate project to develop new visual analytics approaches and elevate the state-of-the-art in dimensionality reduction applied to neural network visualization. Problems with existing techniques, especially concerning multiple projection alignment for hidden layer comparison, have been identified and overcome (to a certain extent) and new applications were investigated, bringing forth new visualization metaphors capable of shedding some light to obscured information contained in these models.

The generic method for projection alignment, described in Chapter 2, was a solution for a problem encountered by the researcher when dealing with projections from multiple feature sets referring to the same data points: when a non-linear dimensionality reduction technique is used, especially in the case of gradient-descent based techniques, outputs for each feature set cannot be directly compared to each other, as distance relationships may not carry the same meaning across all projections. This problem occurs not only when visualizing information from NN models, but with any comparison between different feature sets: visualizing the differences in results between different data preprocessing methods, or changes between a sequence of steps in a processing pipeline, for instance, can suffer from the same issues.

There are DR techniques that offer a certain degree of control over the final layout (e.g., requesting control points as input). For other techniques, such as t-SNE, many research papers address this problem by sharing the same initialization or using the same parameters (RAUBER *et al.*, 2017), which may not work as intended. Although the proposed method suffers from a trade-off between projection fidelity and alignment, it provides a step in the direction of ensuring that visual information reflects into actual data properties in the context of comparing multiple feature sets.

With the approach presented in Chapter 3 for visualizing a sequence of projections, the researcher faced the fact that displaying movement between multiple data states in a single image is quite a complex issue to tackle, and while the article presents informative results, future work would have to be guided in a different direction in order to achieve the desired goals. Vector fields and trajectories provide useful information, but may as well overload visual space and generating confusing results. This is the main reason that resulted in efforts being moved from generating a single 2D projection with all data states to several interconnected 1D projections.

Chapter 4 shows the next step in that path, as the researcher worked on a new visualization approach with a specific application in mind. While there are other techniques in literature that may offer the ability to globally compare a network's perception of data instances, they do not contextualize it very efficiently, which is essential since output behavior varies heavily with each hidden layer and the notion of what defines "small" or "large" differences between data objects may quickly become elusive.

The concept of profiling network hidden outputs for groups of data can be employed in other applications aside adversarial data comparison, and while this possibility was hinted at when describing class fingerprints, no more work has been conducted in this direction. There is a high potential for research in NN visualization, but focusing in certain domains is required when designing a framework in order to avoid too much generalization and reduced usability.

5.2 Hypothesis and Goals

In Chapter 1, this project's goals were described as:

Improving dimensionality reduction techniques for visualizing deep neural network information, by developing new visualization methods and adjustments to existing ones to better suit the needs of this domain, as well as solving problems associated with it.

All work reported in this document was directed towards achieving these goals, and while some of the problems and limitations in existing techniques have been addressed, it is important to note that the domain of neural network visualization is too expansive to be fully covered by what is presented here. There are models and datasets that may not be suited to the approaches described in this document. However, the proposed methods do provide insights and efficiency in certain scenarios, and therefore the researcher considers the goals of this project achieved.

The null hypothesis h_0 , also described in Chapter 1:

There are no advantages to enhancing dimensionality reduction techniques and

developing specific methods for exploratory visual analysis to improve explainability of neural network models.

The case studies and experiments performed in this project are somewhat limited with respect to real-world usability, as they refer to benchmark models and data instead of actual running systems or businesses and formal user tests could not be performed. However, the results obtained show noticeable improvements when compared to previous approaches, such as highlighting information that would not appear otherwise and performing tasks in ways that previous techniques cannot. Thus, while formal proof to refute h_0 was not attained, this thesis raises many evidences against it, as there are not only visible advances in the presented methods as it is shown that there is room for further development in this field as well.

5.3 Challenges

Throughout the development of the project, challenges and obstacles were encountered, a few of which will now be listed in this section. The main challenge resided in that the target domain for study was, at times, too broad. Initially, the researcher planned on improving interaction between users and analytical systems in a general manner. The project was then redefined as to focus on visualization of deep neural networks, as it was noticed that deeper knowledge of a domain was needed in order to provide the desired level of improvement in visualization methods. However, the amount of possible applications and tasks present in DNN research and the VA tools developed towards assisting them was also considerable. Other directions in DNN visualization were explored, such as visualizing the interaction between training hyperparameters.

As work was done to further focus research direction, the researcher faced one more obstacle: understanding neural network models to a level that would allow proper experimentation and interpretation of results. As a visual analytics researcher, a certain amount of time was dedicated to studying machine learning literature so that needs and limitations described by experts in this domain could be properly understood. At a personal level, the researcher considers the knowledge obtained from this process to be highly beneficial not only for the ability to use DNN models as a tool but also to the comprehension of computer science as a whole.

The third obstacle encountered was that data for experiments was not readily available. Benchmark datasets can easily be found online, but real world data that would allow for the building of a complete use case was not obtained. While convincing results were presented using research community standard datasets (MNIST, CIFAR10, ImageNet), the researcher considers that working towards solving a specific problem in industry would also provide beneficial results.

5.4 Future Work

The articles presented at each Chapter in this thesis offer interesting research opportunities for future work. As discussed in previous sections, the domain of DNN visualization itself still offers many challenges and unsolved problems to be addressed. As research in neural network models develops new and more sophisticated technologies, the need for tools to understand them will follow.

There is currently no agreed upon solution to the problem of visualizing different feature sets from the same data, and the available methods (including the ones presented in this thesis) still face certain limitations. Additionally, identifying how human-recognizable features in data relate to internal abstract representations in a model remains a difficult task, despite many recent developments.

BIBLIOGRAPHY

ADADI, A.; BERRADA, M. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). **IEEE Access**, IEEE, v. 6, p. 52138–52160, 2018. Citations on pages [22](#) and [57](#).

AKHTAR, N.; MIAN, A. Threat of adversarial attacks on deep learning in computer vision: A survey. **IEEE Access**, IEEE, v. 6, p. 14410–14430, 2018. Citation on page [61](#).

AMERSHI, S.; CHICKERING, M.; DRUCKER, S. M.; LEE, B.; SIMARD, P.; SUH, J. Model-tracker: Redesigning performance analysis tools for machine learning. In: **Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems**. [S.l.: s.n.], 2015. p. 337–346. Citation on page [58](#).

ANCONA, M.; CEOLINI, E.; ÖZTIRELI, C.; GROSS, M. A unified view of gradient-based attribution methods for deep neural networks. In: ETH ZURICH. **NIPS Workshop on Interpreting, Explaining and Visualizing Deep Learning-Now What?(NIPS 2017)**. [S.l.], 2017. Citation on page [59](#).

ARAUJO, A.; MEUNIER, L.; PINOT, R.; NEGREVERGNE, B. Robust neural networks using randomized adversarial training. **arXiv preprint arXiv:1903.10219**, 2019. Citation on page [67](#).

ATHALYE, A.; ENGSTROM, L.; ILYAS, A.; KWOK, K. Synthesizing robust adversarial examples. In: PMLR. **International conference on machine learning**. [S.l.], 2018. p. 284–293. Citation on page [61](#).

BABIKER, H. K. B.; GOEBEL, R. An introduction to deep visual explanation. **arXiv preprint arXiv:1711.09482**, 2017. Citations on pages [40](#) and [59](#).

BIAN, J.; TIAN, D.; TANG, Y.; TAO, D. **A survey on trajectory clustering analysis**. 2018. Citation on page [45](#).

BIGGIO, B.; ROLI, F. Wild patterns: Ten years after the rise of adversarial machine learning. **Pattern Recognition**, Elsevier, v. 84, p. 317–331, 2018. Citation on page [60](#).

BIGGIO, B.; RUSSU, P.; DIDACI, L.; ROLI, F. *et al.* Adversarial biometric recognition: A review on biometric system security from the adversarial machine-learning perspective. **IEEE Signal Processing Magazine**, IEEE, v. 32, n. 5, p. 31–41, 2015. Citation on page [56](#).

CANTAREIRA, G. D.; PAULOVOICH, F. V. A Generic Model for Projection Alignment Applied to Neural Network Visualization. In: TURKAY, C.; VROTSOU, K. (Ed.). **EuroVis Workshop on Visual Analytics (EuroVA)**. [S.l.]: The Eurographics Association, 2020. ISBN 978-3-03868-116-8. ISSN 2664-4487. Citations on pages [38](#), [42](#), and [64](#).

CANTAREIRA, G. D.; PAULOVOICH, F. V.; ETEMAD, E. Visualizing learning space in neural network hidden layers. In: INSTICC. **Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume**

3: IVAPP, [S.l.]: SciTePress, 2020. p. 110–121. ISBN 978-989-758-402-2. Citations on pages 29, 38, and 59.

Cao, K.; Liu, M.; Su, H.; Wu, J.; Zhu, J.; Liu, S. Analyzing the noise robustness of deep neural networks. **IEEE Transactions on Visualization and Computer Graphics**, p. 1–1, 2020. Citation on page 60.

CARLINI, N.; WAGNER, D. Adversarial examples are not easily detected: Bypassing ten detection methods. In: **Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security**. [S.l.: s.n.], 2017. p. 3–14. Citation on page 67.

_____. Towards evaluating the robustness of neural networks. In: IEEE. **2017 IEEE Symposium on Security and Privacy (SP)**. [S.l.], 2017. p. 39–57. Citation on page 60.

CARTER, S.; ARMSTRONG, Z.; SCHUBERT, L.; JOHNSON, I.; OLAH, C. Activation atlas. **Distill**, 2019. <https://distill.pub/2019/activation-atlas>. Citation on page 29.

CHAKRABORTY, A.; ALAM, M.; DEY, V.; CHATTOPADHYAY, A.; MUKHOPADHYAY, D. Adversarial attacks and defences: A survey. **arXiv preprint arXiv:1810.00069**, 2018. Citations on pages 56 and 59.

CHALMERS, M. A linear iteration time layout algorithm for visualising high-dimensional data. In: IEEE. **Proceedings of Seventh Annual IEEE Visualization'96**. [S.l.], 1996. p. 127–131. Citation on page 32.

CHOO, J.; LIU, S. Visual analytics for explainable deep learning. **IEEE computer graphics and applications**, IEEE, v. 38, n. 4, p. 84–92, 2018. Citation on page 58.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. ImageNet: A Large-Scale Hierarchical Image Database. In: **CVPR09**. [S.l.: s.n.], 2009. Citations on pages 33, 49, and 66.

DONAHUE, J.; JIA, Y.; VINYALS, O.; HOFFMAN, J.; ZHANG, N.; TZENG, E.; DARRELL, T. Decaf: A deep convolutional activation feature for generic visual recognition. In: **International conference on machine learning**. [S.l.: s.n.], 2014. p. 647–655. Citation on page 40.

EADES, P. A heuristic for graph drawing. **Congressus numerantium**, v. 42, p. 149–160, 1984. Citation on page 32.

ERHAN, D.; BENGIO, Y.; COURVILLE, A.; VINCENT, P. Visualizing higher-layer features of a deep network. **University of Montreal**, v. 1341, n. 3, p. 1, 2009. Citations on pages 39 and 66.

ESPADOTO, M.; MARTINS, R. M.; KERREN, A.; HIRATA, N. S.; TELEA, A. C. Towards a quantitative survey of dimension reduction techniques. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, 2019. Citations on pages 29 and 31.

ESPADOTO, M.; RODRIGUES, F. C. M.; TELEA, A. C. Visual analytics of multidimensional projections for constructing classifier decision boundary maps. In: **International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)**. [S.l.: s.n.], 2019. v. 3, p. 28–38. Citations on pages 22, 28, and 29.

FERREIRA, N.; KLOSOWSKI, J. T.; SCHEIDEGGER, C. E.; SILVA, C. T. Vector field k-means: Clustering trajectories by fitting multiple vector fields. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2013. v. 32, p. 201–210. Citation on page 44.

_____. Vector field k-means: Clustering trajectories by fitting multiple vector fields. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2013. v. 32, n. 3pt2, p. 201–210. Citation on page [99](#).

FISHER, R. A. The use of multiple measurements in taxonomic problems. **Annals of eugenics**, Wiley Online Library, v. 7, n. 2, p. 179–188, 1936. Citation on page [29](#).

GAO, Y.; XU, C.; WANG, D.; CHEN, S.; RANASINGHE, D. C.; NEPAL, S. Strip: A defence against trojan attacks on deep neural networks. In: **Proceedings of the 35th Annual Computer Security Applications Conference**. [S.l.: s.n.], 2019. p. 113–125. Citation on page [60](#).

GOODFELLOW, I. J.; SHLENS, J.; SZEGEDY, C. Explaining and harnessing adversarial examples. **arXiv preprint arXiv:1412.6572**, 2014. Citations on pages [61](#), [66](#), and [101](#).

GRIGORESCU, S.; TRASNEA, B.; COCIAS, T.; MACESANU, G. A survey of deep learning techniques for autonomous driving. **Journal of Field Robotics**, Wiley Online Library, v. 37, n. 3, p. 362–386, 2020. Citation on page [55](#).

HALNAUT, A.; GIOT, R.; BOURQUI, R.; AUBER, D. Deep dive into deep neural networks with flows. In: **Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2020): IVAPP**. [S.l.: s.n.], 2020. v. 3, p. 231–239. Citation on page [59](#).

HAMEL, P.; ECK, D. Learning features from music audio with deep belief networks. In: UTRECHT, THE NETHERLANDS. **ISMIR**. [S.l.], 2010. v. 10, p. 339–344. Citation on page [40](#).

HAN, X.; HU, Y.; FOSCHINI, L.; CHINITZ, L.; JANKELSON, L.; RANGANATH, R. Deep learning models for electrocardiograms are susceptible to adversarial attack. **Nature Medicine**, Nature Publishing Group, p. 1–4, 2020. Citation on page [56](#).

HEINRICH, J.; WEISKOPF, D. State of the Art of Parallel Coordinates. In: **Eurographics 2013 - State of the Art Reports**. [S.l.]: The Eurographics Association, 2013. ISSN 1017-4656. Citation on page [64](#).

HILASACA, G.; PAULOVICH, F. Visual feature fusion and its application to support unsupervised clustering tasks. **arXiv preprint arXiv:1901.05556**, 2019. Citations on pages [29](#), [31](#), [42](#), and [95](#).

HOHMAN, F.; PARK, H.; ROBINSON, C.; CHAU, D. H. P. Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. **IEEE transactions on visualization and computer graphics**, IEEE, v. 26, n. 1, p. 1096–1106, 2019. Citations on pages [59](#), [65](#), and [98](#).

HOHMAN, F. M.; KAHNG, M.; PIENTA, R.; CHAU, D. H. Visual analytics in deep learning: An interrogative survey for the next frontiers. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, 2018. Citations on pages [22](#), [39](#), [56](#), and [57](#).

HOTELLING, H. Analysis of a complex of statistical variables into principal components. **Journal of educational psychology**, Warwick & York, v. 24, n. 6, p. 417, 1933. Citation on page [29](#).

HURTER, C.; ERSOY, O.; FABRIKANT, S. I.; KLEIN, T. R.; TELEA, A. C. Bundled visualization of dynamicgraph and trail data. **IEEE transactions on visualization and computer graphics**, IEEE, v. 20, n. 8, p. 1141–1157, 2013. Citation on page 47.

ILYAS, A.; SANTURKAR, S.; TSIPRAS, D.; ENGSTROM, L.; TRAN, B.; MADRY, A. Adversarial examples are not bugs, they are features. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2019. p. 125–136. Citation on page 60.

INGRAM, S.; MUNZNER, T. Dimensionality reduction for documents with nearest neighbor queries. **Neurocomputing**, Elsevier, v. 150, p. 557–569, 2015. Citation on page 32.

Jiang, W.; Li, H.; Liu, S.; Luo, X.; Lu, R. Poisoning and evasion attacks against deep learning algorithms in autonomous vehicles. **IEEE Transactions on Vehicular Technology**, v. 69, n. 4, p. 4439–4449, 2020. Citation on page 61.

JOIA, P.; COIMBRA, D.; CUMINATO, J. A.; PAULOVICH, F. V.; NONATO, L. G. Local affine multidimensional projection. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, v. 17, n. 12, p. 2563–2571, 2011. Citations on pages 29 and 42.

KAHNG, M.; ANDREWS, P. Y.; KALRO, A.; CHAU, D. H. P. Activis: Visual exploration of industry-scale deep neural network models. **IEEE transactions on visualization and computer graphics**, IEEE, v. 24, n. 1, p. 88–97, 2017. Citations on pages 22, 24, 28, 29, 58, 62, and 65.

_____. A cti v is: Visual exploration of industry-scale deep neural network models. **IEEE transactions on visualization and computer graphics**, IEEE, v. 24, n. 1, p. 88–97, 2018. Citation on page 40.

KAHNG, M.; THORAT, N.; CHAU, D. H. P.; VIÉGAS, F. B.; WATTENBERG, M. Gan lab: Understanding complex deep generative models using interactive visual experimentation. **IEEE transactions on visualization and computer graphics**, IEEE, v. 25, n. 1, p. 1–11, 2018. Citation on page 58.

KARPATHY, A.; JOHNSON, J.; FEI-FEI, L. Visualizing and understanding recurrent networks. **arXiv preprint arXiv:1506.02078**, 2015. Citation on page 58.

KEIM, D. A.; KOHLHAMMER, J.; ELLIS, G.; MANSMANN, F. **Mastering the information age-solving problems with visual analytics**. [S.l.]: Florian Mansmann, 2010. Citations on pages 21 and 22.

KINDERMANS, P.-J.; SCHÜTT, K. T.; ALBER, M.; MÜLLER, K.-R.; ERHAN, D.; KIM, B.; DÄHNE, S. Learning how to explain neural networks: Patternnet and patternattribution. **arXiv preprint arXiv:1705.05598**, 2017. Citation on page 59.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014. Citation on page 33.

Kong, X.; Li, M.; Ma, K.; Tian, K.; Wang, M.; Ning, Z.; Xia, F. Big trajectory data: A survey of applications and services. **IEEE Access**, v. 6, p. 58295–58306, 2018. Citation on page 45.

KRAUSE, J.; DASGUPTA, A.; SWARTZ, J.; APHINYANAPHONGS, Y.; BERTINI, E. A workflow for visual diagnostics of binary classifiers using instance-level explanations. In: **IEEE. 2017 IEEE Conference on Visual Analytics Science and Technology (VAST)**. [S.l.], 2017. p. 162–172. Citation on page 58.

KRAUSE, J.; PERER, A.; BERTINI, E. Infuse: interactive feature selection for predictive modeling of high dimensional data. **IEEE transactions on visualization and computer graphics**, IEEE, v. 20, n. 12, p. 1614–1623, 2014. Citation on page 58.

KRIZHEVSKY, A.; HINTON, G. Convolutional deep belief networks on cifar-10. **Unpublished manuscript**, v. 40, 2010. Citation on page 61.

KRIZHEVSKY, A.; HINTON, G. *et al.* **Learning multiple layers of features from tiny images**. [S.l.], 2009. Citations on pages 33 and 74.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105. Citation on page 61.

KWON, B. C.; CHOI, M.-J.; KIM, J. T.; CHOI, E.; KIM, Y. B.; KWON, S.; SUN, J.; CHOO, J. Retainvis: Visual analytics with interpretable and interactive recurrent neural networks on electronic medical records. **IEEE transactions on visualization and computer graphics**, IEEE, v. 25, n. 1, p. 299–309, 2018. Citation on page 58.

KWON, H.; KIM, Y.; PARK, K.-W.; YOON, H.; CHOI, D. Multi-targeted adversarial example in evasion attack on deep neural network. **IEEE Access**, IEEE, v. 6, p. 46084–46096, 2018. Citations on pages 60 and 61.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015. Citations on pages 22, 37, 39, 45, and 55.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, November 1998. Citations on pages 30, 33, 46, and 66.

LI, J.; CHEN, X.; HOVY, E.; JURAFSKY, D. Visualizing and understanding neural models in nlp. **arXiv preprint arXiv:1506.01066**, 2015. Citation on page 58.

LI, Y.; CHENG, S.; XUE, Y.; TIAN, L. Displacement-agnostic coherent imaging through scatter with an interpretable deep neural network. **arXiv preprint arXiv:2005.07318**, 2020. Citation on page 98.

LIAO, C.; ZHONG, H.; SQUICCIARINI, A.; ZHU, S.; MILLER, D. Backdoor embedding in convolutional neural network models via invisible perturbation. **arXiv preprint arXiv:1808.10307**, 2018. Citation on page 60.

LIU, D.; CUI, W.; JIN, K.; GUO, Y.; QU, H. Deeptacker: Visualizing the training process of convolutional neural networks. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM New York, NY, USA, v. 10, n. 1, p. 1–25, 2018. Citation on page 58.

LIU, M.; LIU, S.; SU, H.; CAO, K.; ZHU, J. Analyzing the noise robustness of deep neural networks. In: IEEE. **2018 IEEE Conference on Visual Analytics Science and Technology (VAST)**. [S.l.], 2018. p. 60–71. Citations on pages 57, 60, 64, and 75.

LIU, M.; SHI, J.; LI, Z.; LI, C.; ZHU, J.; LIU, S. Towards better analysis of deep convolutional neural networks. **IEEE transactions on visualization and computer graphics**, IEEE, v. 23, n. 1, p. 91–100, 2017. Citations on pages 58 and 59.

LIU, S.; WANG, X.; LIU, M.; ZHU, J. Towards better analysis of machine learning models: A visual analytics perspective. **Visual Informatics**, Elsevier, v. 1, n. 1, p. 48–56, 2017. Citations on pages 40 and 58.

LIU, W.; WANG, Z.; LIU, X.; ZENG, N.; LIU, Y.; ALSAADI, F. E. A survey of deep neural network architectures and their applications. **Neurocomputing**, Elsevier, v. 234, p. 11–26, 2017. Citation on page 22.

MA, Y.; XIE, T.; LI, J.; MACIEJEWSKI, R. Explaining vulnerabilities to adversarial machine learning through visual analytics. **IEEE transactions on visualization and computer graphics**, IEEE, v. 26, n. 1, p. 1075–1085, 2019. Citations on pages 56, 57, and 60.

MAATEN, L. V. D.; HINTON, G. Visualizing high-dimensional data using t-sne. *journal of machine learning research*. **J Mach Learn Res**, v. 9, p. 26, 2008. Citations on pages 24, 29, 32, 41, 42, and 96.

MADRY, A.; MAKELOV, A.; SCHMIDT, L.; TSIPRAS, D.; VLADU, A. Towards deep learning models resistant to adversarial attacks. **arXiv preprint arXiv:1706.06083**, 2017. Citations on pages 56, 61, 66, 67, and 101.

MAHENDRAN, A.; VEDALDI, A. Understanding deep image representations by inverting them. In: IEEE. **2015 IEEE conference on computer vision and pattern recognition (CVPR)**. [S.l.], 2015. p. 5188–5196. Citations on pages 38 and 40.

_____. Salient deconvolutional networks. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2016. p. 120–135. Citation on page 59.

_____. Visualizing deep convolutional neural networks using natural pre-images. **International Journal of Computer Vision**, Springer, v. 120, n. 3, p. 233–255, 2016. Citation on page 40.

MCINNES, L.; HEALY, J.; MELVILLE, J. Umap: Uniform manifold approximation and projection for dimension reduction. **arXiv preprint arXiv:1802.03426**, 2018. Citations on pages 28, 29, 33, 40, 43, 63, and 98.

MING, Y.; CAO, S.; ZHANG, R.; LI, Z.; CHEN, Y.; SONG, Y.; QU, H. Understanding hidden memories of recurrent neural networks. In: IEEE. **2017 IEEE Conference on Visual Analytics Science and Technology (VAST)**. [S.l.], 2017. p. 13–24. Citation on page 58.

MOHAMED, A.-r.; HINTON, G.; PENN, G. Understanding how deep belief networks perform acoustic modelling. **neural networks**, p. 6–9, 2012. Citation on page 40.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of machine learning**. [S.l.]: MIT press, 2018. Citation on page 21.

MONTAVON, G.; LAPUSCHKIN, S.; BINDER, A.; SAMEK, W.; MÜLLER, K.-R. Explaining nonlinear classification decisions with deep taylor decomposition. **Pattern Recognition**, Elsevier, v. 65, p. 211–222, 2017. Citation on page 59.

MOOSAVI-DEZFOOLI, S.-M.; FAWZI, A.; FROSSARD, P. Deepfool: a simple and accurate method to fool deep neural networks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 2574–2582. Citation on page 67.

MORDVINTSEV, A.; OLAH, C.; TYKA, M. **Inceptionism: Going Deeper into Neural Networks**. [S.l.], 2015. Available: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>. Citation on page 59.

MUÑOZ-GONZÁLEZ, L.; BIGGIO, B.; DEMONTIS, A.; PAUDICE, A.; WONGRASSAMEE, V.; LUPU, E. C.; ROLI, F. Towards poisoning of deep learning algorithms with back-gradient optimization. In: **Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security**. [S.l.: s.n.], 2017. p. 27–38. Citation on page 60.

NELSON, B.; BARRENO, M.; CHI, F. J.; JOSEPH, A. D.; RUBINSTEIN, B. I.; SAINI, U.; SUTTON, C. A.; TYGAR, J. D.; XIA, K. Exploiting machine learning to subvert your spam filter. **LEET**, v. 8, p. 1–9, 2008. Citation on page 56.

NGUYEN, A.; YOSINSKI, J.; CLUNE, J. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. **arXiv preprint arXiv:1602.03616**, 2016. Citation on page 59.

NIE, S.; HEALEY, C.; PADIA, K.; LEEMAN-MUNK, S.; BENSON, J.; CAIRA, D.; SETHI, S.; DEVARAJAN, R. Visualizing deep neural networks for text analytics. In: IEEE. **2018 IEEE Pacific Visualization Symposium (PacificVis)**. [S.l.], 2018. p. 180–189. Citation on page 58.

NONATO, L. G.; AUPETIT, M. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. **IEEE Transactions on Visualization and Computer Graphics**, p. 1–1, 2018. ISSN 1077-2626. Citations on pages 22, 28, 29, 38, and 40.

OTTER, D. W.; MEDINA, J. R.; KALITA, J. K. A survey of the usages of deep learning for natural language processing. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, 2020. Citation on page 22.

OZBULAK, U.; NEVE, W. D.; MESSEM, A. V. How the softmax output is misleading for evaluating the strength of adversarial examples. **arXiv preprint arXiv:1811.08577**, 2018. Citation on page 60.

PADMANABHAN, J.; PREMKUMAR, M. J. J. Machine learning in automatic speech recognition: A survey. **IETE Technical Review**, Taylor & Francis, v. 32, n. 4, p. 240–251, 2015. Citation on page 55.

PAN, Z.; YU, W.; YI, X.; KHAN, A.; YUAN, F.; ZHENG, Y. Recent progress on generative adversarial networks (gans): A survey. **IEEE Access**, IEEE, v. 7, p. 36322–36333, 2019. Citation on page 22.

PAULOVICH, F. V.; NONATO, L. G.; MINGHIM, R.; LEVKOWITZ, H. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, v. 14, n. 3, p. 564–575, 2008. Citation on page 29.

PAULOVICH, F. V.; SILVA, C. T.; NONATO, L. G. Two-phase mapping for projecting massive data sets. **IEEE Transactions on Visualization and Computer Graphics**, IEEE, v. 16, n. 6, p. 1281–1290, 2010. Citation on page 29.

PENG, Z.; LARAMEE, R. S. Higher dimensional vector field visualization: A survey. In: **TPCG**. [S.l.: s.n.], 2009. p. 149–163. Citation on page 45.

PEZZOTTI, N.; HÖLLT, T.; GEMERT, J. V.; LELIEVELDT, B. P.; EISEMANN, E.; VILANOVA, A. Deepeyes: Progressive visual analytics for designing deep neural networks. **IEEE transactions on visualization and computer graphics**, IEEE, v. 24, n. 1, p. 98–108, 2018. Citations on pages [22](#), [24](#), [28](#), [29](#), [40](#), and [58](#).

PEZZOTTI, N.; HÖLLT, T.; LELIEVELDT, B.; EISEMANN, E.; VILANOVA, A. Hierarchical stochastic neighbor embedding. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2016. v. 35, n. 3, p. 21–30. Citations on pages [29](#) and [32](#).

PEZZOTTI, N.; MORDVINTSEV, A.; HOLLT, T.; LELIEVELDT, B. P.; EISEMANN, E.; VILANOVA, A. Linear tsne optimization for the web. **arXiv preprint arXiv:1805.10817**, 2018. Citation on page [98](#).

PEZZOTTI, N.; THIJSEN, J.; MORDVINTSEV, A.; HÖLLT, T.; LEW, B. V.; LELIEVELDT, B. P.; EISEMANN, E.; VILANOVA, A. Gpgpu linear complexity t-sne optimization. **IEEE transactions on visualization and computer graphics**, IEEE, v. 26, n. 1, p. 1172–1181, 2019. Citation on page [98](#).

RAUBER, P. E.; FADEL, S. G.; FALCAO, A. X.; TELEA, A. C. Visualizing the hidden activity of artificial neural networks. **IEEE transactions on visualization and computer graphics**, IEEE, v. 23, n. 1, p. 101–110, 2017. Citations on pages [22](#), [24](#), [28](#), [29](#), [31](#), [38](#), [40](#), [45](#), [47](#), [77](#), and [98](#).

RAUBER, P. E.; FALCÃO, A. X.; TELEA, A. C. Visualizing time-dependent data using dynamic t-sne. In: EUROGRAPHICS ASSOCIATION. **Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers**. [S.l.], 2016. p. 73–77. Citations on pages [28](#), [29](#), [31](#), [42](#), and [98](#).

RAUBER, P. E.; FALCAO, A. X.; TELEA, A. C. Projections as visual aids for classification system design. **Information Visualization**, 2017. Citations on pages [22](#), [28](#), and [29](#).

REN, D.; AMERSHI, S.; LEE, B.; SUH, J.; WILLIAMS, J. D. Squares: Supporting interactive performance analysis for multiclass classifiers. **IEEE transactions on visualization and computer graphics**, IEEE, v. 23, n. 1, p. 61–70, 2016. Citation on page [58](#).

RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. " why should i trust you?" explaining the predictions of any classifier. In: **Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining**. [S.l.: s.n.], 2016. p. 1135–1144. Citation on page [60](#).

ROZSA, A.; GÜNTHER, M.; BOULT, T. E. Adversarial robustness: Softmax versus openmax. **arXiv preprint arXiv:1708.01697**, 2017. Citation on page [60](#).

SAHA, A.; SUBRAMANYA, A.; PIRSIIVASH, H. Hidden trigger backdoor attacks. **arXiv preprint arXiv:1910.00033**, 2019. Citation on page [60](#).

SAMEK, W.; BINDER, A.; MONTAVON, G.; LAPUSCHKIN, S.; MÜLLER, K.-R. Evaluating the visualization of what a deep neural network has learned. **IEEE transactions on neural networks and learning systems**, IEEE, v. 28, n. 11, p. 2660–2673, 2016. Citation on page [59](#).

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 4510–4520. Citations on pages [48](#) and [66](#).

SANNA, A.; MONTUSCHI, P.; MONTUSCHI, P. A survey on visualization of vector fields by texture-based methods. **Devel. Pattern Rec**, v. 1, p. 2000, 2000. Citation on page 45.

SCHERER, D.; MÜLLER, A.; BEHNKE, S. Evaluation of pooling operations in convolutional architectures for object recognition. In: SPRINGER. **International conference on artificial neural networks**. [S.l.], 2010. p. 92–101. Citation on page 39.

SELVARAJU, R. R.; COGSWELL, M.; DAS, A.; VEDANTAM, R.; PARIKH, D.; BATRA, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In: **Proceedings of the IEEE international conference on computer vision**. [S.l.: s.n.], 2017. p. 618–626. Citation on page 59.

SHAFABI, A.; NAJIBI, M.; GHIASI, M. A.; XU, Z.; DICKERSON, J.; STUDER, C.; DAVIS, L. S.; TAYLOR, G.; GOLDSTEIN, T. Adversarial training for free! In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2019. p. 3358–3369. Citation on page 67.

SIMONYAN, K.; VEDALDI, A.; ZISSERMAN, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. **arXiv preprint arXiv:1312.6034**, 2013. Citations on pages 39 and 59.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014. Citation on page 33.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. Citations on pages 38 and 40.

STROBELT, H.; GEHRMANN, S.; PFISTER, H.; RUSH, A. M. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. **IEEE transactions on visualization and computer graphics**, IEEE, v. 24, n. 1, p. 667–676, 2017. Citation on page 58.

SU, J.; VARGAS, D. V.; SAKURAI, K. One pixel attack for fooling deep neural networks. **IEEE Transactions on Evolutionary Computation**, IEEE, v. 23, n. 5, p. 828–841, 2019. Citation on page 60.

SUNDARARAJAN, K.; WOODARD, D. L. Deep learning for biometrics: A survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 51, n. 3, p. 1–34, 2018. Citation on page 55.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCHE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9. Citation on page 39.

SZEGEDY, C.; ZAREMBA, W.; SUTSKEVER, I.; BRUNA, J.; ERHAN, D.; GOODFELLOW, I.; FERGUS, R. Intriguing properties of neural networks. **arXiv preprint arXiv:1312.6199**, 2013. Citations on pages 25, 56, and 59.

TELEA, A. C. **Data visualization: principles and practice**. [S.l.]: CRC Press, 2014. Citation on page 21.

TORGERSON, W. S. Multidimensional scaling: I. theory and method. **Psychometrika**, Springer, v. 17, n. 4, p. 401–419, 1952. Citations on pages 29 and 32.

Wang, J.; Gou, L.; Shen, H.; Yang, H. Dqnviz: A visual analytics approach to understand deep q-networks. **IEEE Transactions on Visualization and Computer Graphics**, v. 25, n. 1, p. 288–298, 2019. Citation on page 58.

Wang, J.; Gou, L.; Yang, H.; Shen, H. Ganviz: A visual analytics approach to understand the adversarial game. **IEEE Transactions on Visualization and Computer Graphics**, v. 24, n. 6, p. 1905–1917, 2018. Citation on page 58.

WANG, W.; WANG, W.; LI, S. From numerics to combinatorics: a survey of topological methods for vector field visualization. **Journal of Visualization**, Springer, v. 19, n. 4, p. 727–752, 2016. Citation on page 45.

Wang, Z. J.; Turko, R.; Shaikh, O.; Park, H.; Das, N.; Hohman, F.; Kahng, M.; Chau, D. H. Cnn explainer: Learning convolutional neural networks with interactive visualization. **IEEE Transactions on Visualization and Computer Graphics**, p. 1–1, 2020. Citation on page 58.

WATTENBERG, M.; VIÉGAS, F.; JOHNSON, I. How to use t-sne effectively. **Distill**, v. 1, n. 10, p. e2, 2016. Citations on pages 24, 28, 43, and 98.

WONG, E.; RICE, L.; KOLTER, J. Z. Fast is better than free: Revisiting adversarial training. **arXiv preprint arXiv:2001.03994**, 2020. Citation on page 67.

WONGSUPHASAWAT, K.; SMILKOV, D.; WEXLER, J.; WILSON, J.; MANE, D.; FRITZ, D.; KRISHNAN, D.; VIÉGAS, F. B.; WATTENBERG, M. Visualizing dataflow graphs of deep learning models in tensorflow. **IEEE transactions on visualization and computer graphics**, IEEE, v. 24, n. 1, p. 1–12, 2017. Citation on page 58.

XIAO, C.; LI, B.; ZHU, J.-Y.; HE, W.; LIU, M.; SONG, D. Generating adversarial examples with adversarial networks. **arXiv preprint arXiv:1801.02610**, 2018. Citation on page 67.

Xu, G.; Li, H.; Ren, H.; Yang, K.; Deng, R. H. Data security issues in deep learning: Attacks, countermeasures, and opportunities. **IEEE Communications Magazine**, v. 57, n. 11, p. 116–122, 2019. Citation on page 61.

YE, Y.; LI, T.; ADJEROH, D.; IYENGAR, S. S. A survey on malware detection using data mining techniques. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 50, n. 3, p. 1–40, 2017. Citation on page 55.

YOSINSKI, J.; CLUNE, J.; NGUYEN, A.; FUCHS, T.; LIPSON, H. Understanding neural networks through deep visualization. **arXiv preprint arXiv:1506.06579**, 2015. Citations on pages 40, 59, and 65.

Yuan, X.; He, P.; Zhu, Q.; Li, X. Adversarial examples: Attacks and defenses for deep learning. **IEEE Transactions on Neural Networks and Learning Systems**, v. 30, n. 9, p. 2805–2824, 2019. Citation on page 61.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2014. p. 818–833. Citations on pages 39 and 59.

ZEILER, M. D.; TAYLOR, G. W.; FERGUS, R. Adaptive deconvolutional networks for mid and high level feature learning. In: IEEE. **2011 International Conference on Computer Vision**. [S.l.], 2011. p. 2018–2025. Citation on page 59.

ZHANG, J.; WANG, Y.; MOLINO, P.; LI, L.; EBERT, D. S. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. **IEEE transactions on visualization and computer graphics**, IEEE, v. 25, n. 1, p. 364–373, 2018. Citation on page [58](#).

ZHOU, H.; XU, P.; YUAN, X.; QU, H. Edge bundling in information visualization. **Tsinghua Science and Technology**, TUP, v. 18, n. 2, p. 145–156, 2013. Citation on page [64](#).

ZINTGRAF, L. M.; COHEN, T. S.; ADEL, T.; WELLING, M. Visualizing deep neural network decisions: Prediction difference analysis. **arXiv preprint arXiv:1702.04595**, 2017. Citation on page [39](#).

GLOSSARY

ANN: Artificial Neural Network.

CNN: Convolutional Neural Network.

DNN: Deep Neural Network.

ML: Machine Learning.

VA: Visual Analytics.

XAI: Explainable Artificial Intelligence.

APPENDIX: CONCEPTS AND TOOLS

A.1 Introduction

This appendix contains descriptions for techniques and tools in literature that were researched and employed in this thesis. As each chapter has the contents of a scientific article that is either published or intended for publication, description of related techniques are kept to a basic level due to scope and space constraints. Therefore, a more detailed presentation of these techniques is included here.

Subjects are presented in the order as they appear in text: Section [A.2](#) describes gradient descent-based projection techniques involved in Chapter [2](#), Section [A.3](#) describes the *vector field k-means* technique used in Chapter [3](#).

A.2 Gradient Descent-based Dimensionality Reduction

This section briefly describes DR techniques involved in building the generic approach to projection alignment in Chapter [2](#). These techniques receive a dataset X as input, consisting of points in the original space, and generate a set of points Y in a projected space by iteratively calculating the gradients of a given error function between original and projected data to minimize said error.

While there are other techniques that perform such optimization, this section is focused on the ones involved in Chapter [2](#): VFF, t-SNE, and UMAP.

A.2.1 VFF

The Visual Feature Fusion technique ([HILASACA; PAULOVICH, 2019](#)) was originally developed to fuse different feature representations of data with the aim of building a new, user-driven representation. From a set of T different feature representations $X_1, \dots, X_T \in \mathbb{R}^m$

of the same dataset being used as original data, the technique aims to calculate projections $Y_1, \dots, Y_T \in \mathbb{R}^2$ that preserve distance relationships in each set X_t as best as possible while aligning projections. In this process, two restrictions are considered for optimization. The first is responsible for ensuring that the projection matches the original data and is given by

$$C_{stress}(X_t) = \frac{1}{|X_t|^2} \sum_i \sum_j \left(\frac{\delta(x_i^t, x_j^t)}{\delta_{max}^t} - \|y_i^t - y_j^t\| \right)^2 \quad (\text{A.1})$$

where x_i^t and x_j^t are instances in X_t , $\delta(x_i^t, x_j^t)$ is the distance between them on their original space, δ_{max}^k is the maximum pairwise distance between instances in X_k , and y_i^t and y_j^t are the representations of x_i^t and x_j^t on the m-dimensional space, respectively.

To define the second restriction, a mean projection $\bar{Y}$ is obtained from A.1 using the mean pairwise distances among all projections. This restriction refers to projection alignment and is given by

$$C_{alignment}(X_t) = \frac{1}{|X_t|^2} \sum_i \sum_j \left(d(\bar{y}_i, \bar{y}_j) - \|\bar{y}_i - y_j^t\| \right)^2 \quad (\text{A.2})$$

where $d(\bar{y}_i, \bar{y}_j)$ is the distance between i and j in the mean projection. Joining the two equations, the final optimization problem is described by the equation

$$C_{VFF}(X_k) = \lambda \cdot C_{stress}(X_t) + (1 - \lambda) \cdot C_{alignment}(X_t) \quad (\text{A.3})$$

where λ is a parameter used to control the importance of each aspect of the optimization.

A.2.2 t-SNE

The t-distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction technique that uses statistical distributions to map data to an embedded space, usually containing two or three dimensions (MAATEN; HINTON, 2008). This technique is widely used to project and visualize data in machine learning applications, especially due to its preservation of neighborhood information, making it particularly efficient at discriminating groups or classes.

t-SNE is based on Stochastic Neighbor Embedding (SNE), a DR technique that projects data by converting high-dimensional euclidean distances between datapoints into conditional probabilities that represent similarities (MAATEN; HINTON, 2008). The differences between probabilities in projected space and original space are minimized using *Kullback-Leibler divergences* (KL) as a cost function:

$$C_{SNE} = \sum_i KL(P_i Q_i) = \sum_{i=1}^N \sum_{j=1}^N p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \quad (\text{A.4})$$

where $p_{j|i}$ is the probability of a given point j being in the neighborhood of another point i in the original space, and $q_{j|i}$ is the probability of point j being in the neighborhood of point i in the projected space. SNE assumes that the distribution of probabilities P_i for each point i is a gaussian, being written as

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \quad (\text{A.5})$$

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)} \quad (\text{A.6})$$

While gaussian variance in projected space is set as $1/\sqrt{2}$, variances σ_i in the original space are unique for every i and need to be estimated according to each point's neighborhood. SNE does this estimation by performing a binary search for σ_i that best satisfies the following condition:

$$\text{perp}(P_i) = 2^{H(P_i)} \quad (\text{A.7})$$

where $\text{perp}(P_i)$ is a *perplexity* parameter set by the user and $H(P_i)$ is the binary Shannon entropy for P_i , given by

$$H(P_i) = - \sum_j P_{j|i} \log_2 p_{j|i} \quad (\text{A.8})$$

t-SNE modifies SNE by calculating the cost function as the KL divergence between two joint probability distributions P and Q :

$$C_{tSNE} = KL(P||Q) = \sum_{i=1}^N \sum_{j=1}^N p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (\text{A.9})$$

where P forces the symmetry $p_{ji} = p_{ij}$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \quad (\text{A.10})$$

and Q is defined using a Student t-distribution with one degree of freedom instead of a gaussian to determine q_{ji} , as to include heavier tails:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}} \quad (\text{A.11})$$

Although it is capable of providing impressive results for many applications, t-SNE presents a few inherent limitations that must be kept in mind: as its cost function prioritizes

local neighborhood preservation, the meaning of cluster distance and shape in t-SNE projections can be misleading. The technique is also highly sensitive to hyperparameters, namely perplexity and neighborhood size, producing visual outcomes that may not reflect dataset properties (WATTENBERG; VIÉGAS; JOHNSON, 2016). Additionally, t-SNE also has a relatively high computational cost that may lead to performance and scalability issues, although recent works (PEZZOTTI *et al.*, 2018; PEZZOTTI *et al.*, 2019) have achieved remarkable results in improving performance.

A.2.2.1 *dt-SNE*

When projecting multiple layer activation data from neural networks, Rauber *et al.* (RAUBER *et al.*, 2017) discuss methods of reducing non-relevant variability in data projected with t-SNE, such as using point-cloud registration and using previous projections as initializations for each step. The solution presented by these authors is to use the same projection A_p as initialization for all projections $A_p[t]$. However, this initialization does not ensure that the projections will stay aligned, especially in cases where transitions are less well-behaved.

The authors were aware of alignment issues and later proposed Dynamic t-SNE (Dt-SNE) (RAUBER; FALCÃO; TELEA, 2016), a variation of t-SNE capable of minimizing data variability in a temporal series of projections to a certain degree in order to observe change in multiple DNN states. This is achieved by adding a term to the t-SNE’s cost equation, as follows:

$$C = \sum_{t=1}^T C[t] + \frac{\lambda}{2N} \sum_{i=1}^N \sum_{t=1}^{T-1} \|p_i[t] - p_i[t+1]\|^2 \quad (\text{A.12})$$

where T is the number of projections, $C[t]$ is the original t-SNE cost for projection t , λ is a hyperparameter that controls the bias for temporal coherence, and $p_i[t]$ is the position of a given point p_i in the projection t . The generic alignment constraint presented in Chapter 2 uses the same principle as Dt-SNE, but removes the temporal limitation (allowing for comparison between any number of projections in any order) and addressing the issue of matching λ for techniques with gradients of different magnitudes.

A.2.3 *UMAP*

UMAP is a DR technique that aims to provide a level of local fidelity similar to t-SNE while offering improvements with respect to some of its pitfalls, such as better preserving global structure and providing superior run performance (MCINNES; HEALY; MELVILLE, 2018). Since its publication, UMAP has been increasingly adopted for multivariate data visualization (HOHMAN *et al.*, 2019; LI *et al.*, 2020).

UMAP performs optimization by minimizing a cost function in a similar fashion to t-SNE, but its cost is defined by *binary cross-entropy* (CE):

$$C_{UMAP} = CE(PQ) = \sum_i \sum_j \left[p_{ij} \log \left(\frac{p_{ij}}{q_{ij}} \right) + (1 - p_{ij}) \log \left(\frac{1 - p_{ij}}{1 - q_{ij}} \right) \right] \quad (\text{A.13})$$

This function ensures a better representation of global distances while providing a relatively small impact on neighborhood preservation. There are also differences in methods for calculating P and Q , as UMAP removes normalization from its variables. Probabilities $p_{i|j}$ are given by

$$p_{i|j} = \exp \left(- \frac{d(x_i, x_j) - \rho_i}{\sigma_i} \right) \quad (\text{A.14})$$

where $d(x_i, x_j)$ is a chosen distance metric between points x_i and x_j and ρ_i is the distance from point i to its nearest neighbor. The symmetry constraint for pairs i, j is also tweaked:

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j} p_{j|i} \quad (\text{A.15})$$

Finally, variances σ_i in P are obtained using a number of neighbors parameter k instead of perplexity:

$$k = 2^{\sum_i p_{ij}} \quad (\text{A.16})$$

For probability modeling in Q , UMAP employs a family of curves $1/(1 - ay^{2b})$, with a and b being obtained from a *minimum distance* parameter that indicates a minimum distance value under which all distances are considered equally close. Thus, q_{ij} are given by

$$q_{ij} = \left(1 + a(y_i - y_j)^{2b} \right)^{-1} \quad (\text{A.17})$$

A.3 Discrete Vector Field Generation

In the context of this thesis, the most important aspect of the *vector field k-means* technique (FERREIRA *et al.*, 2013b) employed in Chapter 3 is that it defines a distance measure $d(X, a)$ between a discrete grid representation of a vector field X and a trajectory a , composed by pairs of time and spatial position $(t, a(t))$. Once this distance function is defined, it can be used to optimize discrete vector fields that provide the best approximation to a group of trajectories through minimization and to match trajectories to their most representative fields through comparison and selection of lowest values, similarly to the standard k-means algorithm.

In this approach, a trajectory is represented by a path written as $\alpha : [t_0, t_1] \rightarrow \mathbb{R}^n$ and a vector field is defined in a domain $\Omega \subset \mathbb{R}^n$, i.e., a function $X : \Omega \rightarrow \mathbb{R}^n$. So that, finding the vector field that best matches a set of trajectories ϕ can be described as the optimization problem

$$E = \min_X \sum_{\alpha_i \in \phi} \int_{t_0^i}^{t_1^i} \|X(\alpha_i(t)) - (\alpha'_i(t))\|^2 dt \quad (\text{A.18})$$

where $\alpha'_i(t)$ is the velocity vector of the trajectory α on instant t . Also included in this problem is a regularization restriction

$$E = \min_X \lambda_L \|\Delta X\|^2 + (1 - \lambda_L) \sum_{\alpha_i \in \phi} \int_{t_0^i}^{t_1^i} \|(\alpha_i(t)) - (\alpha'_i(t))\|^2 dt \quad (\text{A.19})$$

where Δ is the Laplace operator. This restriction ensures the smoothness of the resulting vector field, with parameter λ_L controlling the weight of each term in the minimization equation and therefore determining if the optimization should prioritize smoothness or matching the vector field to the trajectories.

It is impractical to perform a continuous optimization of a vector field in such a manner. Therefore, a matrixial parametric representation is necessary. Since the desired output is a visualization, trajectories, and vector fields are constrained to the 2D visual space, with trajectories given as samples – a sequence of pairs composed by a point in the 2D space and a time stamp. It is presumed that the path between a given point and the previous point in the sequence is a straight line with constant velocity, thus forming a polygonal line (segment s_j). The discrete representation of the 2D vector field X is given by a triangular grid G with resolution R , with a vector attributed to each vertex. Vectors corresponding to middle points are linearly interpolated between vectors from the three vertices in each triangle.

Each trajectory is previously processed to generate intermediate points ensuring that every segment is contained in only one triangle in the grid. Thus, it is possible to compute the difference between a vector from the field and the velocity vector from a trajectory α in a position $\alpha(t)$: using

$$d(X(\alpha(t)), \alpha'(t)) = |C_{\alpha(t)}X - \alpha'(t)| \quad (\text{A.20})$$

where $C_{\alpha(t)}$ is the vector with the barycentric coordinates equals to $\alpha(t)$. In practice, X is a matrix $(R^2 \times 2)$ indexing 2D vectors for every vertex on the grid and $C_{\alpha(t)}$ is a vector with R^2 elements, with zeros in all indexes except for the ones corresponding to the vertices that compose the triangle containing $\alpha(t)$. To discreetly represent the Laplacian operator, the authors use the cotangent weighted Laplacian matrix, and the minimization problem is solved by least squares.

A.4 Generation of Adversarial Examples

Adversarial machine learning literature defines an adversarial example x_{adv} as the result of applying a perturbation s over an example x from data distribution, to obtain an incorrect outcome from a model. Perturbations come from a set of allowed perturbations S that define rules that generated adversarial data must adhere. For the context presented in Chapter 4 (image classification), S is commonly chosen as to ensure that modified examples are still perceptually similar, such as being bound to an l_2 or l_∞ ball around original examples. This section presents methods employed in Chapter 4 to generate adversarial examples from images.

A.4.1 FGSM

One of the most known methods to generate adversarial examples for neural networks in literature is by gradient ascent, as shown by Goodfellow et al. (GOODFELLOW; SHLENS; SZEGEDY, 2014). Given a loss function $L(\Theta, x, y)$ from a NN model, where Θ are model parameters (weights), x is the input data and y is the desired output, a gradient can be calculated as to alter input data towards the direction of maximum discrepancy between desired and actual outputs:

$$x^{adv} = x + \varepsilon \operatorname{sgn}(\nabla_x L(\Theta, x, y)) \quad (\text{A.21})$$

This method is known as *Fast Gradient Sign* (FGSM). The ε parameter controls the influence of adversarial perturbation in the resulting object, effectively bounding it to an l_∞ ball of radius ε .

A.4.2 PGD

The perturbation produced by FGSM can be repeatedly applied as iterations of a larger process, leading towards an output that maximizes loss. This process is known as Projected Gradient Descent (PGD), and the result x^{t+1} of every new iteration can be written as:

$$x^{t+1} = \Pi_{B_p(x, \varepsilon)} \left(x^t + \alpha \operatorname{sgn}(\nabla_x L(\Theta, x, y)) \right) \quad (\text{A.22})$$

where α dictates step size or learning rate and $\Pi_{B_p(x, \varepsilon)}$ is a projection ensuring that resulting data is contained within the l_p ball $B_p(x, \varepsilon) = \{x + r \text{ s.t. } \|r\|_p < \varepsilon\}$. PGD is considered a universal first-order adversary, i.e., its output tends toward the strongest possible attack with local first-order information from a network model (MADRY et al., 2017). Both FGSM and PGD can be set up as to alter an example towards a target label, changing y to the *desired* output and subtracting the gradient.