OVERCOMING CHALLENGING URBAN IMAGES

Deep Learning and Data Integration Methods for Detecting Trees Entangled with Power Lines

Artur André Almeida de Macedo Oliveira

IME USP

Overcoming Challenging Urban Images

Deep Learning and Data Integration Methods for Detecting Trees Entangled with Power Lines

Artur André Almeida de Macedo Oliveira

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Doctor of Science

Program: Computer Science
Advisor: Prof. Dr. Roberto Hirata Jr.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, São Paulo Research Foundation (FAPESP) 18/10767-0 and 15/22308-2 and MCTI (law 8.248, PPI-Softex - TIC 13 - 01245.010222/2022-44).

São Paulo
August, 2023

Overcoming Challenging Urban Images

Deep Learning and Data Integration Methods for Detecting Trees Entangled with Power Lines

Artur André Almeida de Macedo Oliveira

This version of the thesis includes the corrections and modifications suggested by the Examining Committee during the defense of the original version of the work, which took place on August 11, 2023.

A copy of the original version is available at the Institute of Mathematics and Statistics of the University of São Paulo.

Examining Committee:

Prof. Dr. Roberto Hirata Jr. (advisor) – IME-USP
Prof. Dr. Marcos S. Buckeridge – IB-USP
Prof. Dr. Roberto de Alencar Lotufo – UNICAMP
Prof. Dr. David Menotti Gomes – UFPR
Prof. Dr. Zhangyang Atlas Wang – UT AUSTIN

I dedicate this work to everyone passionate about
images and what can be learned with them.

Acknowledgments

It's not that I'm so smart, it's just that I stay with problems longer.
— Albert Einstein

I would like to thank my advisor Roberto Hirata Jr. for his advice, conversations and support over these years, which made this doctorate possible and super fun.

To friends Mateus Espadoto, Fabrício Machado, Sinais Robins, Nina Hirata, Roberto Cesar, Fabio Kon, Alfredo Goldman, Daniel Batista, Kelly Braghetto, Thilo Koch, Lucas Stankus, Diego Soler, Luan Casagrande and Vinícius Quintas Massimino, for the fun discussions and collaborations. To Mary Sirois and Francisco Pinheiro from Atos for the mentorship during the Atos IT Challenge.

I would also like to thank Nelson Lago, Prof. Fabio Kon, to Prof. Alfredo Goldman and colleagues from the systems laboratory for the interscity workshops and for discussions and collaborations.

To Prof. Zhangyang "Atlas" Wang for receiving me at the University of Texas at Austin, and all the discussions we've had which were very fruitful and allowed me to enhance both my writing and researcher skills. To Karen Little for all the assistance with documentation and bureaucracy. To friends at VITA team, Scott, Zhiwen, Dejia, Xiaohan, Peihao, Haotao, Xinyu, and Tianlong. To friends at the 21st Street for making my stay in Texas absolutely magical, specially to Trevor, Dylan and Logan.

To my family for their love, support, patience and understanding during these years, and to my wife Arlete for all the sweetness and kindness.

To the FAPESP, CAPES, Softex, Microsoft, and Google for the financial support received during the elaboration of this work.

To the comprehension of everyone I forgot to mention here xD

To all of you my sincere gratitude.

Resumo

Artur André Almeida de Macedo Oliveira. **Superando imagens urbanas desafiantes: *métodos de aprendizagem profunda e integração de dados para deteção de emaranhamentos entre árvores e fios elétricos*.** Tese (Doutorado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A classificação de imagens urbanas em nível de rua apresenta desafios devido à presença de diversos elementos, aparências variadas e poses complexas. Fatores como oclusão, confusão de fundo, condições climáticas e pontos de vista da câmera complicam ainda mais o processo de classificação. Neste estudo, aproveitamos as capacidades de redes de aprendizado profundo recentes, incluindo MobileNets, ResNets, DenseNets e EfficientNets, para enfrentar esses desafios. Nosso objetivo é avaliar o desempenho dessas redes, identificar limitações e propor novas técnicas para superá-las.

Nossa pesquisa se concentra na tarefa específica de classificar imagens urbanas com ou sem árvores próximas à rede elétrica. Através de uma exploração extensiva, fornecemos métodos e insights úteis não só para esse problema de classificação, mas também aplicáveis a tarefas de classificação em outros domínios.

Duas contribuições principais são introduzidas em nosso trabalho. Em primeiro lugar, ampliamos a plataforma INvestigate and Analyze a CITY (INACITY) integrando um banco de dados orientado a grafos, melhorando o desempenho e a cobertura da coleta de imagens urbanas com o Google Street View. Em segundo lugar, desenvolvemos a ferramenta Street-Level Image Labeler (SLIL), que reduz eficientemente o ônus de rotular imagens manualmente, facilitando a criação de conjuntos de dados. Com a ajuda do INACITY e do SLIL, criamos um conjunto de dados abrangente com 8.800 imagens urbanas em nível de rua rotuladas bináriamente como contendo árvores próximas à rede elétrica (i.e. classe positiva) ou não (i.e. classe negativa).

A avaliação humana do conjunto de dados revela a presença de imagens desafiadoras que confundem até mesmo classificadores experientes. Por exemplo, distinguir se fios de poste cruzam ou passam por trás das copas das árvores pode ser difícil, dependendo do ponto de vista da câmera.

A comparação de redes neurais profundas recentes nesse conjunto de dados revela que a maior precisão alcançada por redes comuns é de 74,6%. No entanto, ao introduzir uma nova classe distinta da positiva ou negativa, a classe *desafiadora*, e empregar o protocolo de treinamento *Noisy Student* e a função de custo *Focal Loss*, melhoramos efetivamente as taxas de revocação para as classe positiva de 66,5% para 83,7% e para a classe negativa de 63,7% para 78,8%. Essa abordagem nos permite identificar e classificar melhor imagens que anteriormente eram propensas a classificações incorretas.

Palavras-chave: Imagens urbanas. Visão Computacional. Aprendizagem Profunda. Dificuldade de Instância.

Abstract

Artur André Almeida de Macedo Oliveira. **Overcoming Challenging Urban Images: *Deep Learning and Data Integration Methods for Detecting Trees Entangled with Power Lines***. Thesis (Doctorate). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Urban image classification at the street-level poses significant challenges due to the presence of diverse elements, varying appearances, and complex poses. Factors such as occlusion, background clutter, environmental conditions, and camera viewpoints further complicate the classification process. In this study, we leverage the capabilities of state-of-the-art Deep Learning Networks (DLNs), including MobileNets, ResNets, DenseNets, and EfficientNets, to tackle these challenges head-on. We aim to evaluate the performance of these DLNs, identify limitations, and propose innovative techniques for overcoming them.

Our research focuses on the specific task of classifying urban images with or without trees near overhead powerlines. Through an extensive exploration, we provide methods and insights that not only address this classification problem but also offer generalizable solutions applicable to a range of classification tasks.

Two major contributions are introduced in our work. Firstly, we extend the INvestigate and Analyze a CITY (INACITY) platform by integrating a graph-oriented database, improving the performance and coverage of urban image collection from Google Street View. Secondly, we develop the Street-Level Image Labeler (SLIL) tool, which efficiently mitigates the manual labeling burden, facilitating dataset creation. With the help of INACITY and SLIL, we curate a comprehensive labeled dataset comprising 8,800 street-level urban images.

Human evaluation of the dataset reveals the presence of challenging images that perplex even experienced classifiers. For example, distinguishing whether powerlines intersect or pass behind tree canopies can be difficult depending on the perspective. The comparison of state-of-the-art DLNs on this dataset reveals that the highest accuracy achieved by plain DLNs is 74.6%. However, by introducing a new class *challenging* distinct from positive or negative, and employing the noisy student training protocol and focal loss, we effectively enhance the recall rates for positive and negative classes respectively from 66.5% and 63.7% to 83.7% and 78.8%. This approach enables us to better identify and classify images that were previously prone to misclassification.

Keywords: Urban images. Computer Vision. Deep Learning. Instance Hardness.

List of Figures

List of Tables

Contents

Appendixes

Chapter 1

Introduction

Nowadays, we have access to an extensive collection of urban images from all around the world, thanks to advancements in technology [8, 3]. We can extract from these images valuable insights about our urban landscapes [68]. However, the complexity and diversity of urban images pose significant challenges to their automatic processing and analysis. Despite the remarkable progress of Deep Learning Networks (DLNs) [169, 66], they can still struggle with simple images that humans can easily recognize [66, 69]. For example, Figs. 1.1 and 1.2 show an urban image of a busy street and the output of a trained DLN YoloV5[1] [84] detector, respectively. Figure 1.3a shows a single detection for two distinct individuals, and Fig. 1.3b a food truck undetected because the network training data did not include such vehicles. Surprisingly, DLNs overlook objects in urban scenes that humans can easily spot. Understanding the limits and weaknesses of DLNs is essential for deploying them safely and effectively in real-world applications [127].

We can harness Computer Vision (CV) with DLNs to detect and classify Urban Micro Events [186] related to urban trees, which play a vital role in shaping our cityscapes. The distribution and abundance of trees in cities can vary significantly due to differences in urban planning and green space allocation [113]. Monitoring these variations becomes crucial to address any negative impacts that trees may have on city structures. An example of a significant concern is the interaction between trees and the electrical grid. Such interactions can result in severe incidents, including fires [171, 207], disruptions in power distribution [24, 82], and even fatalities due to electrical shocks [144, 82]. In early August 2023, on the island of Maui, Hawaii, an incident occurred where the proximity of a tree to a section of the electrical distribution network initiated a fire that quickly spread across the entire island [171]. This devastating fire endured for four days [132] and tragically claimed the lives of 115 individuals. But such incidents are not isolated to distant locations. Also in early August, on Afrânio Peixoto Avenue in São Paulo (just outside the University of São Paulo), an alarming incident unfolded as a palm tree came into contact with the electrical wires, resulting in a brief fire outbreak[2] (illustrated in Fig. 1.4).

Effectively analyzing tree images using CV methods presents several challenges. The

[1] Available at https://pytorch.org/hub/ultralytics_yolov5/

[2] A short video recorded by Roberto Hirata can be seen at https://arturao.org/tree_igniting.mp4

Figure 1.1: *Typical urban picture from Google Street View at the Rua Bresser in São Paulo, SP, Brazil.*

Figure 1.2: *The output of a YoloV5 detection DLN. Notice that YoloV5 ignores trees, electric wires, building façades, poles, store signs, windows, and other elements despite being extremely common in urban scenes.*

(a) (b)

Figure 1.3: *Details of the output of the YoloV5 for the image from the Rua Bresser. In (a) we see two persons detected as a single one, possibly due to occlusion. In (b) the YoloV5 failed to detect the food truck despite being in plain sight because there were no food trucks in its training dataset.*

Figure 1.4: *Electric arc formed by the contact of electrical wires with a tree.*

appearance of trees can undergo drastic changes based on factors such as species, seasonal variations, pruning practices, occlusion, and even the camera perspective. Consequently, performing CV classification on tree images becomes inherently challenging. In this context, we use the term *challenging* to refer to specific classification tasks or images that pose difficulties. In the subsequent sections of this Introduction, we will present the advances we have made to understand the limitations of DLNs and enhance their performance on datasets containing such *challenging* images.

(a) **(b)**

(c) **(d)**

Figure 1.5: *The appearance of the same tree can change drastically between pictures, and sometimes grown bushes look like small trees (d). (a) a tree pruned due to its closeness to overhead powerlines. (b) the same tree before pruning does not fit the image. (c) by getting a picture from a further location the top and bottom of the tree can be seen, but with fewer details. (d) a bush that looks like a tree.*

1.1 Research questions

Detecting and classifying urban micro-events with trees pose significant challenges. The ambiguity in tree appearance and environmental factors like sun glare, shadows, and camera perspective, can make automatic detection difficult. Figures. 1.5a, 1.5b, 1.5c exemplify how drastically the appearance of a tree can change, and Fig. 1.5d shows a grown bush that looks like small trees such variability and ambiguity can make the automatic tree detection and classification difficult. Furthermore, a problem like spotting trees near overhead powerlines can be even more challenging due to sun glare (see Fig. 1.6a), shadows (see Fig. 1.6c), point-of-view of the camera (see Fig. 1.6e) and even the distance between the camera and the trees (see Fig. 1.6g).

In summary, we seek answers to the following research questions:

1. How good are state-of-the-art deep learning networks for detecting urban micro-events such as trees and their interactions with overhead powerlines in street-level urban images?

2. Can challenging images (e.g. with degradation and or a bad point-of-view) be automatically detected?

3. How do challenging images impact the generalization performance?

4. How can we deal with challenging images?

To answer the first question, we propose a dataset of street-level urban images considered easy or challenging by humans that classified them as having or not trees entangled with powerlines. We train several state-of-the-art deep learning networks with this dataset, multiple strategies, and hyperparameters. We found that while current models are good baselines, challenging images have a negative impact on their generalization performance.

To answer the second question, we marked challenging images and assessed the confidence of the output of DLNs for them. We found that training networks with the Focal-Loss cost function make the gap between the confidence (i.e. maximum value in the output vector of the model) of easy and challenging images larger than training them with the vanilla Cross-Entropy cost function.

To answer the third question, we compared the test accuracy of networks trained with and without challenging images. We found that removing them from the training dataset improves the test accuracy.

Finally, we consider the fourth question under the observations obtained while searching for answers to the previous questions. Challenging images may behave as noisy samples when used for training; removing them can improve generalization performance. Alternatively, one could collect other images for the same urban feature (e.g. tree) from a distinct point-of-view depending on their availability.

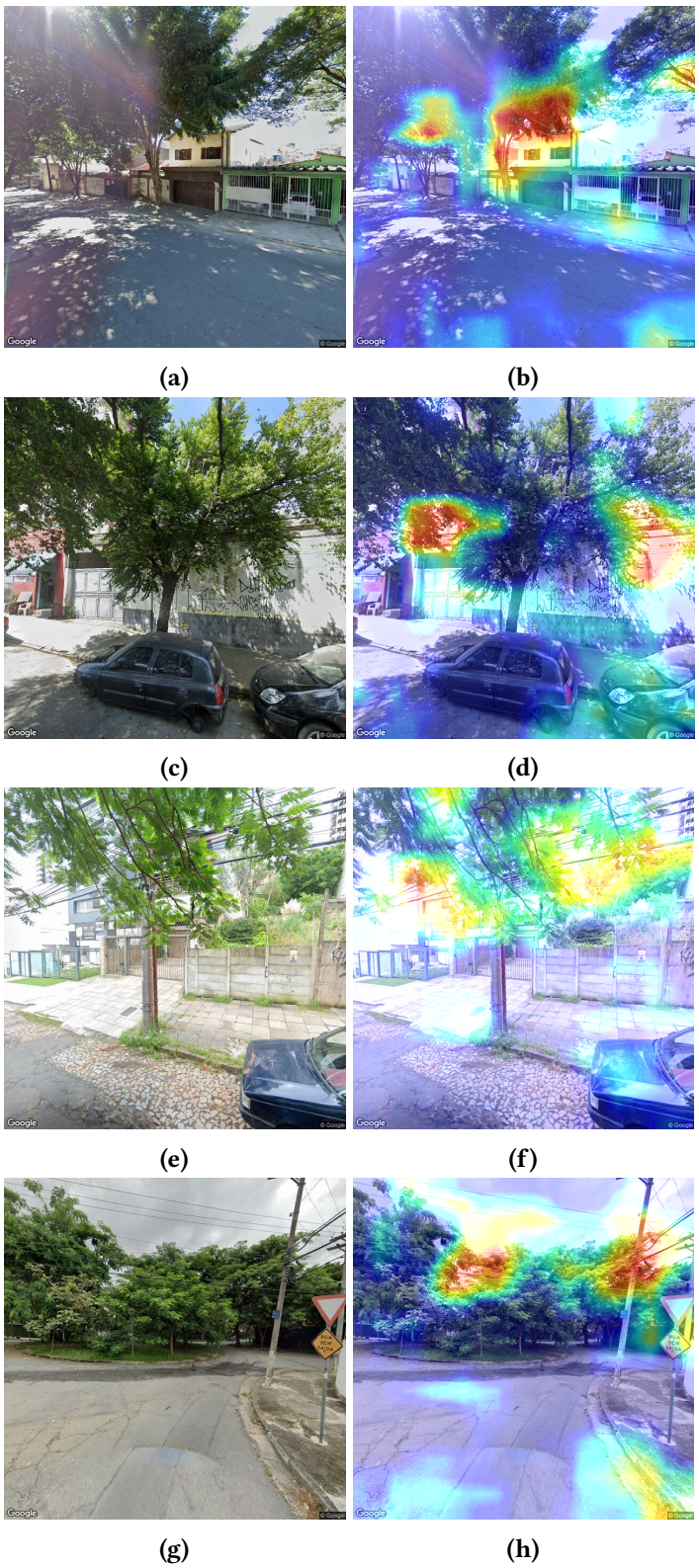


Figure 1.6: *Different sources of hardness, the left column shows the images, and the right column the corresponding GradCAM++ activation maps. Regions in red are the most relevant (according to GradCAM++) for classifying each image. (a) sun glare obfuscates a portion of the wires; (c) shadows degrade the visibility of the wires crossing tree branches; (e) occlusion of the wires by tree branches; (g) trees are far away from the wires, but in the image plane, they intersect. (b,d,f,h) The activation maps show that the most relevant regions are those where the wires can be more easily observed and are close to the tree branches.*

1.2 Structure of the thesis

Chapter 2 presents preliminaries, concepts, and definitions used throughout this text. Chapter 3 presents related works to give context and show current research on relevant topics to this text. Chapters 4, 5 and 6 present a detailed description of our contributions:

- **Chapter 4**, partially based on the paper: INACITY - INvestigate and Analyze a CITY - Artur André Almeida de Macedo Oliveira and Roberto Hirata. In: SoftwareX 15 (2021), p. 100777. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2021.100777
  Short abstract: INACITY is a platform to integrate geographical data, geolocalized imagery, and computer vision methods. During the elaboration of this thesis, we extended the INACITY platform by incorporating it with a graph-oriented database, performed analyses on its enhanced performance, and created a detailed software description of INACITY itself. We used the extended INACITY platform to compose all the datasets we created during this Ph.D.

- **Chapter 5**, partially based on the paper: Detecting tree and wire entanglements with deep learning - Artur André Oliveira, Marcos S Buckeridge, and Roberto Hirata Jr. In: Trees (2022), pp. 1-13
  Short abstract: We collected 11 thousand street-level urban images from the cities of São Paulo (SP) and Porto Alegre (POA) in Brazil to develop a method to automatically detect trees close to overhead powerlines. We labeled the images using the Street-Level-Image-Labeler (SLIL) software, which we developed and describe in Appendix A. We trained Deep Learning Networks (DLN) to develop our method and we provide a comparison of the performance of multiple DLN architectures trained on a subset of the collected images and tested on a disjoint subset of the images. Our results show that detecting trees close to overhead powerlines is feasible with DLN, but there is still room for improvement as urban scenes are very complex and some images are ambiguous even for humans labeling them. Figure 1.6 shows several examples of ambiguous images and their GradCAM++ [177] maps.

- **Chapter 6**, partially based on the paper: Locating Urban Trees near Electric Wires using Google Street View Photos: A New Dataset and A Semi-Supervised Learning Approach in the Wild - Artur André AM Oliveira, Zhangyang Wang, and Roberto Hirata. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022, pp. 4286-4294
  Short abstract: Locating trees close to electrical wires can be very challenging in street-level urban images for several reasons such as image degradation (e.g. sun-glare), scene complexity (e.g. background/foreground occlusion), or even due to the lack of depth information. We train teachers and students EfficientNetB0 networks with a combination of 11 thousand labeled images, and 40 thousand unlabeled images using a combination of semi-supervised learning through the Noisy Student learning protocol [211], the cost function Focal Loss [106] and a new label to characterize challenging images distinct from positive or negative. We show that with our approach we could improve previous results based only on labeled images and vanilla Cross-Entropy cost function.

In Chapter 7 we discuss our proposed methods and results in comparison with other related works. We show advantages, limitations, possibilities for improvements, conclusions on the classification of street-level urban images, and challenging images, and provide interesting research directions.

In the appendix, we present the following works we developed during this Ph.D. indirectly related to the main body of this thesis.

- – **Appendix A**, SLIL: Street-Level Image Labeler - We developed the SLIL tool to mitigate the burden of annotating images for classification. It is easier than other similar tools and specialized for images collected via the INACITY platform. It allows one to directly visit the location where that image was taken in the Google Street View platform. SLIL can be used for annotating images from other domains as well.

  – **Appendix B**, Assessing the effects of ambiguous images on training -
  Our study demonstrates that Deep Learning Networks trained on challenging urban images may exhibit poorer generalization performance compared to networks trained without such challenging images. To investigate this phenomenon, we conducted experiments utilizing the network architectures VGG16, ResNet50, ResNet152, MobileNetV3 Large, EfficientNetB0, DenseNet161, and DenseNet121. Interestingly, with the exception of VGG16, all networks demonstrated improved test accuracy when trained without challenging images.

  Furthermore, we explored the labeling of challenging images as positive or negative cases. We observed that, in general, assigning them to the negative class yielded superior results compared to assigning them to the positive class. However, the most favorable outcomes are achieved by removing challenging images from the training dataset.

  – **Appendix C**, Improving Self-supervised Dimensionality Reduction: Exploring Hyperparameters and Pseudo-Labeling Strategies - Artur André AM Oliveira, Mateus Espadoto, Roberto Hirata Jr, Nina ST Hirata, and Alexandru C Telea. In: Computer Vision, Imaging and Computer Graphics Theory and Applications: 16th International Joint Conference, VISIGRAPP 2021, Virtual Event, February 8–10, 2021, Revised Selected Papers. Springer. 2023, pp. 135-161
  Short abstract: Dimensionality reduction (DR) is an essential tool for the visualization of high-dimensional data. The recently proposed Self-Supervised Network Projection (SSNP) method addresses DR with a number of attractive features, such as high computational scalability, genericity, stability and out-of-sample support, computation of an inverse mapping, and the ability of data clustering. Yet, SSNP has an involved computational pipeline using self-supervision based on labels produced by clustering methods and two separate deep learning networks with multiple hyperparameters. In this paper, we explore the SSNP method in detail by studying its hyperparameter space and pseudo-labeling strategies. We show how these affect SSNP's quality and how to set them to optimal values based on extensive evaluations involving multiple datasets, DR methods, and clustering algorithms.

– **Appendix D**, Stability Analysis of Supervised Decision Boundary Maps - Artur AAM Oliveira, Mateus Espadoto, Roberto Hirata Jr, and Alexandru C Telea. In: SN Computer Science 4.3 (2023), p. 226
Short abstract: Understanding how a machine learning classifier works is an important task in machine learning engineering. However, doing this is for any classifier in general difficult. We propose to leverage visualization methods for this task. For this, we extend a recent technique called Decision Boundary Map (DBM) which graphically depicts how a classifier partitions its input data space into decision zones separated by decision boundaries. We use a supervised, GPU-accelerated technique that computes bidirectional mappings between the data and projection spaces to solve several shortcomings of DBM, such as accuracy and speed. We present several experiments that show that SDBM generates results that are easier to interpret, far less prone to noise, and compute significantly faster than DBM while maintaining the genericity and ease of use of DBM for any type of single-output classifier. We also show, in addition to earlier work, that SDBM is stable with respect to various types and amounts of changes in the training set used to construct the visualized classifiers. This property was, to our knowledge, not investigated for any comparable method for visualizing classifier decision maps, and is essential for the deployment of such visualization methods in analyzing real-world classification models.

– **Appendix E**, SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers - Artur AM Oliveira, Mateus Espadoto, Roberto Hirata Jr, and Alexandru C Telea. In: VISIGRAPP (3: IVAPP). 2022, pp. 77-87
Short abstract: Understanding the decision boundaries of a machine learning classifier is key to gaining insight into how classifiers work. Recently, a technique called Decision Boundary Map (DBM) was developed to enable the visualization of such boundaries by leveraging direct and inverse projections. However, DBM has scalability issues for creating fine-grained maps and can generate results that are hard to interpret when the classification problem has many classes. In this paper, we propose a new technique called Supervised Decision Boundary Maps (SDBM), which uses a supervised, GPU-accelerated projection technique that solves the original DBM shortcomings. We show through several experiments that SDBM generates results that are much easier to interpret when compared to DBM, and is faster and easier to use, while still being generic enough to be used with any type of single-output classifier.

Chapter 2

Preliminaries

In this chapter, we will describe relevant topics for the work, namely, Geographical Image Databases (GID), fundamentals on Deep Learning Networks (DLN), specific DLN architectures, the semi-supervised learning paradigm with a focus on the Noisy Student training protocol [211].

Geographical Image Databases A Geographical Image Database is a system that provides images with geolocated positions. Google Street View (GSV) is an example of a GID that provides ground-level images from most countries on every continent. In the GSV system, a Panorama refers to the metadata of an image and any geographic location containing images. The metadata in a Panorama contains the location of images, a timestamp, and references for other Panoramas that share its location and different timestamps or are nearby. Figure 2.1 shows a Neo4j graph with Panoramas spatiotemporally related, that is, nodes in the graph connected by 'link' labeled edges relate spatially nearby Panoramas, while 'time' edges relate Panoramas from the same location with pictures taken in different moments, 'view' edges connect Panoramas with 'View' nodes representing collected images.

Code 2.1 shows the data of a Panorama, which corresponds to metadata for images taken from a particular location containing the properties:

- 'description' with the address of the Panorama,
- a global identifier property named 'pano',
- the horizontal angle corresponding to the frontal direction of the vehicle at property 'centerHeading' under the 'tiles' property, and
- other spatially nearby Panoramas in the array under the property 'links'.

Figure 2.2 shows one of the images taken from the Panorama exemplified in Code 2.1. We refer the interested reader to the official GSV documentation for more on how to collect Panoramas[1].

[1] Google Street View official documentation: https://developers.google.com/maps/documentation/javascript/streetview#StreetViewService

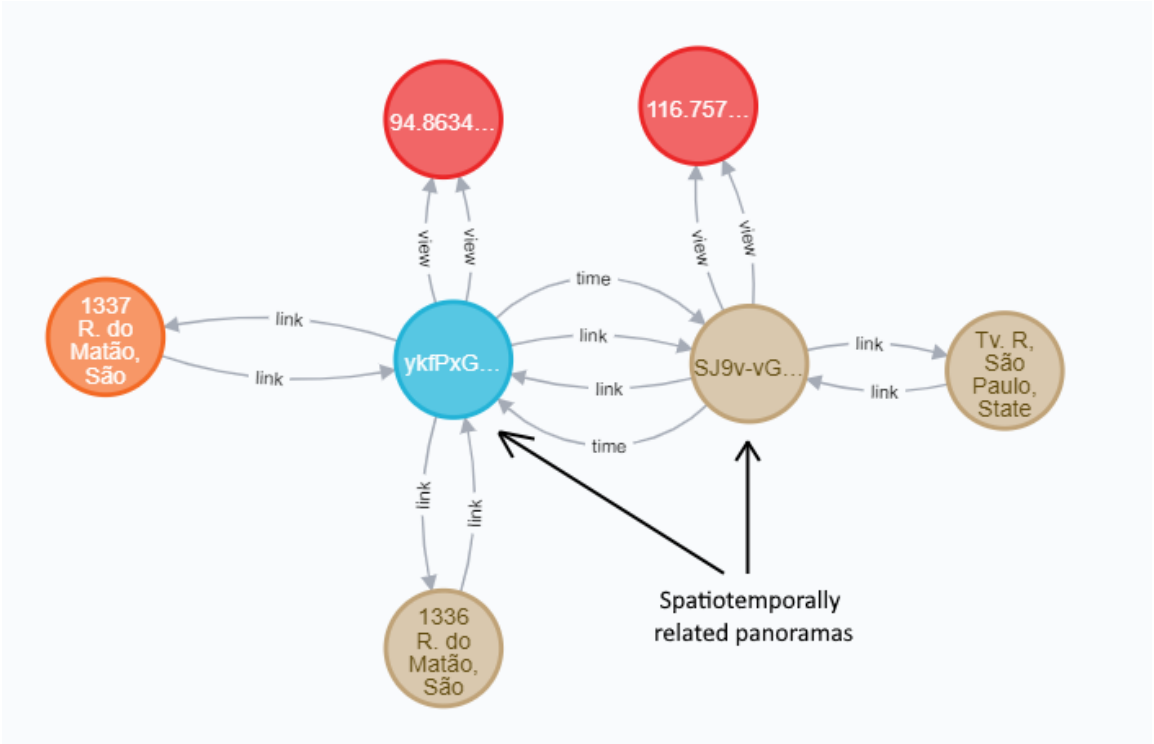


Figure 2.1: *Example of Panoramas (bottom five nodes) related spatially (by edges labeled as 'link') and temporally (by edges labeled 'time'). Each Panorama represents a location where images were collected. The two top nodes are Views representing images collected and are connected to a Panorama indicating the location of this image.*

```
{
  "location": {
    "lon": -46.733551036950125,
    "lat": -23.55726557534755,
    "shortDescription": "1400 Av. Prof. Luciano Gualberto",
    "description": "1400 Av. Prof. Luciano Gualberto, São Paulo, State of São
        Paulo", // Associated real address
    "pano": "N3vCnOJILasEoIco3awrBA" // Panorama identifier
  },
  "copyright": "2023 Google",
  "links": [ // Each "link" item in "links" denotes a nearby Panorama
  {
    "description": "Av. Prof. Luciano Gualberto",
    "heading": 297.59244,
    "pano": "hOKEvXLBn1K4sCdYCcbTbQ"
  }, ....
  ],
  "tiles": {
    "centerHeading": 297.91968, // Angular horizontal direction pointing
        towards the front of the vehicle (with respect to the true North).
    "originHeading": 297.91968,
    "originPitch": 0.017264999999994757, //Angular vertical direction,
        ranging from -90 (downright) to +90 (upright).
    "tileSize": {
      "b": "px",
```

```
23       "f": "px",
24       "height": 512,
25       "width": 512
26     },
27     "worldSize": {
28       "b": "px",
29       "f": "px",
30       "height": 6656,
31       "width": 13312
32     }
33   },
34   "time": [ // Each item in time denotes a Panorama at the same location, but
           with a different timestamp, which is denoted by the property '
          imageDate'.
35   {
36     "pano": "D83V1MpQNV3dHBawKAm8gQ"
37   }, ...
38   ],
39   "imageDate": "2017-07" //The timestamp of this Panorama.
40 }
```

Program 2.1: *Example Panorama in JSON notation. Ellipsis denote more items with the same structure as the one preceding them.*

Deep Learning Networks (DLN) The Multi-Layer Perceptron (MLP) is an example of a DLN, a feedforward network composed of stacked layers of units [166] also called neurons [62]. Each layer in an MLP performs a linear transformation of its input [166], followed by a (non-)linear transformation, also called activation function [62], such as sigmoid, Rectified Linear Unit (ReLU), or other [4]. An MLP is a function that maps a vector from an input d-dimensional space, in which each dimension may correspond to a feature from some observed data such as height, width, weight, and other, to an output n-dimensional space in which each dimension has a different interpretation depending on the context like the probability of the input belonging to a class in a classification problem, or a quantity in a regression problem. Formally, a MLP with k hidden layers h_i activation functions Θ_i for $i \in \{1, \ldots, k\}$, and output $f(x)$ for a given input $x \in \mathbb{R}^d$ is a function $f : \mathbb{R}^d \to \mathbb{R}^n$ defined as:

$$f(x) = \Theta_k(h_k(\Theta_{k-1}(\ldots \Theta_1(h_1(x)))))$$

Another family of DLNs known as Convolutional Neural Networks (CNN) uses convolutional layers, which perform a 'convolution' rather than a linear transformation [62]. A CNN can deal with grid-like topology data, such as images, text, temporal series, or others, more naturally than MLPs [62]. Formally, a convolutional layer s with a two-dimensional kernel K, which can be seen as a $k \times l$ real matrix, cross-correlated with a two-dimensional image I performs the operation:

$$s(i, j) = (K * I)(i, j) = \sum_{m}^{k} \sum_{n}^{l} I(i + m, j + n)K(m, n)$$

where $I(i, j)$ denote the i-th row and j-th column from image I and $K(m, n)$ the $m - th$

Figure 2.2: *Image pointing forward (with respect to the GSV car) from the Panorama exemplified in Program 2.1*

row and n-th column from kernel K [62]. CNNs can be a stacked mixture of linear layers, convolutional layers, and non-linear activations.

A large number of different CNNs have been proposed in the last decade. Here we briefly describe the ones most relevant for this work:

- VGGs [182], a simple baseline with several convolutional layers without residual connections;
- ResNet [65]is a family of architectures with a varying (and large) number of stacked convolutional layers with residual connections between them to mitigate exploding and vanishing gradient issues;
- DenseNets [77] explore the residual connections not only between subsequent layers but also between layers inside the same resolution block of convolutional layers;
- in MobileNetV2 [174] residual connections are used between bottleneck layers to decrease the memory footprint of models;
- MobileNetV3 [73] is a model found with a Network Architectural Search (NAS) approach optimized to be efficient on mobile devices; and
- EfficientNet [187] is a family of networks found with NAS and re-scaled depending on layer width, depth, and input resolution.

Each family of architectures differs in the number and type of layers, the layout of the connections between layers, and the activation functions they use.

Frameworks consolidated by the community like PyTorch [146] and TensorFlow [121] provide implementations of these models, which facilitate the comparison of results reported in different works using the same architectures. We experimented with these DLN architectures with different pretrained weights, classification heads, activation functions, training, cost functions, and learning rates. We also experimented with the training protocol Noisy Student [211], and the technique called temperature scaling to deal with overconfident trained models (c.f. [71] for details on temperature scaling).

Learning a Classification Model We based part of this paragraph on the book Deep Learning [62].

There are many hyperparameters and training strategies to choose from when training DLNs. Here we present the relevant cost functions, optimizers, the early stop strategy, and the Noisy Student learning protocol.

The ultimate goal of a classifier is to maximize the accuracy, or some other metric based on the proportion of correct/incorrect classifications [62]. However, optimizing a DLN over these performance metrics is difficult because they are not differentiable [62]. Thus a surrogate cost function is optimized, which usually has the advantage of being differentiable and can better exploit the training data [62].

Based on the Shannon entropy [179] one can derive the Cross-Entropy cost function, which is a way to measure the difference between probabilities distributions [62]. Consider

the binary classification problem with two probability distributions P and Q over a random variable X. The Cross-Entropy $H(P, Q)$ between P and Q is [62]:

$$-\mathbb{E}_{X\sim P} \log Q(x). \tag{2.1}$$

We are interested in assessing the performance of a DLN model f parameterized by θ, denoted f_θ. Given a dataset D, let $(x, y) \sim D$ be a sample $x \in \mathbb{R}^d$ with correct label $y \in \{0, 1\}$ draw from D. We can assess the cost $J(\theta)$ of a model f_θ as:

$$J(\theta) = -\sum_{(x,y)\sim D} y \log f_\theta(x), \tag{2.2}$$

which is a direct application of Equation 2.1 where y, and $f(x)$ are the target and model distributions, respectively [62]. Notice that a model f_θ that minimizes $J(\theta)$ maximizes its accuracy.

The Focal loss [106] is a modified version of the Cross-Entropy cost function designed for datasets with class imbalance issues [106]. Focal loss deals with the difficulty of learning the minority classes by assigning an exponential weight for misclassified samples, which ultimately implies a dynamic weighting for each sample [106]. Formally, the Focal loss cost function is defined as shown below:

$$FL(\theta) = -\sum_{(x,y)\sim D} y^\gamma \log(f_\theta(x)).$$

where γ is the focusing parameter, y, and $f_\theta(x)$ are the true and predicted label, respectively. Larger γ values make (in)correctly classified samples have (higher)lower impacts on the cost function.

Another relevant aspect while learning a classifier is the optimizer. Models optimized through gradient-based methods require computing the cost of all samples and the gradient of the aggregated cost with respect to the model parameters. However, for large datasets, this procedure can be computationally intensive [62]. Stochastic Gradient Descent (SGD) aims to mitigate this by estimating the gradient using a small random batch, also known as minibatch, with m' samples [62]. Formally, the estimate of the gradient with SGD is:

$$g = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

where L is some cost function for a model parameterized by θ, ∇_θ denotes the gradient of L with respect to θ, $x^{(i)}$ and $y^{(i)}$ denote the i-th minibatch sample and label, respectively.

The Early Stop regularization technique is a stopping criterion for training a DLN [62]. It involves evaluating the DLN with a validation dataset at each training iteration, stopping the training when the validation error does not decrease by a given margin for a fixed number of iterations, and choosing the DLN model with the best validation error as the final model [62].

Semi-supervised learning and Noisy Student Semi-supervised learning involves learning a model from labeled and unlabeled data to improve performance beyond supervised models trained on labeled data only [213]. The Noisy Student training protocol [211] is a semi-supervised approach created to leverage large amounts of unlabeled data. A *teacher* neural network model is trained with labeled data and generates *pseudo-labels* for unlabeled data. The labeled and pseudo-labeled data compose the training dataset of a *student network* model. The student network learns from images augmented with RandAugment [37], and the student model itself is augmented with Stochastic Depth [75] and Dropout [185]. Figure 2.3 illustrates the Noisy Student protocol. The (re)training cycle

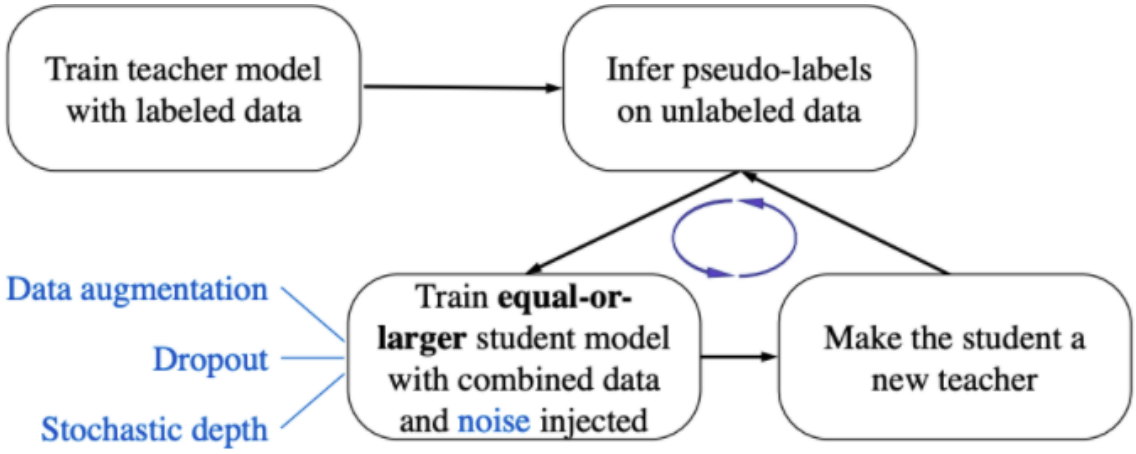


Figure 2.3: *The Noise Student training protocol. Image from [211].*

continues until convergence.

Chapter 3

Related Works

We review in this chapter the relevant literature for collecting and analyzing urban images, emphasizing greenery, trees, and powerlines. As previously mentioned, urban environments pose unique challenges, and we examine them from the perspective of instance hardness. Additionally, we discuss related research on ambiguity in natural images, which can also be viewed as a difficult visual search task.

Collecting and Analyzing Urban Images Geo-located Imagery Databases (GIDs) like Google Street View [8] are reliable sources for studying cities [168, 208]. Platforms like KartaView [3] and Mappilary Vistas [128] mitigate the data collecting burden by making it a distributed crowd-sourced task done by citizens, which is cheap, accessible, and aligned with citizen science efforts [41, 16]. Motivated by the abundance of public geolocalized images, [205] proposes a Convolutional Neural Network (CNN) [102] architecture to detect and classify urban objects using aerial and street-level imagery from Google Street View (GSV), showing that combining data from both modalities is better than just using only one. Greenery [18], tree cover [22], shade provision [103], and even abstract concepts as graffiti [193] can be automatically assessed through images using heuristics [18] and DLNs [22, 103, 193].

Assessing Urban Trees Trees entangled with power lines conduct energy to the ground, causing electrical shock accidents and energy leakages, which can heat the wood, causing wildfires [114, 31], blackouts [5, 156] and partial discharges [144]. Suddenly pruning the trees can have negative consequences, such as spreading diseases and decreasing their shade provision, which is more likely to be criticized by citizens [53]. Managing and monitoring urban forests and their interaction with power lines require multiple indicators, as suggested by various studies [91, 143, 29, 87]. However, this process is costly, time-consuming, and potentially dangerous [87].

Ma et al. [114] use a machine learning approach for time series from electrical currents measured on branches of different species of trees connecting simulated powerlines. By identifying a high risk of ignition, they can prevent bushfires at their initial stages. On the other hand, Kobayashi et al. [93] used satellite images to detect tall vegetation close to power lines. The benefits of using satellite images include broad geographical extents and

the easy inspection of otherwise difficult-to-access areas. By using multi-spectral images, they can spot healthy vegetation, and by using stereoscopic techniques, they can compute the height of the vegetation close to power lines and transmission towers, allowing one to distinguish between harmless understory vegetation in the ground as bushes and grass from tall trees that can come into contact with the power lines.

Jwa et al. [87] propose to use airborne LiDAR data to detect and perform 3D reconstruction of powerlines and transmission towers without prior knowledge of tower position or the number of lines. This method has a better resolution compared with satellite data due to the low altitude in airborne imagery data and the LiDAR structured laser pattern, making it easier to spot issues arising from faulty tall vegetation. Wanik et al. [203] propose a LIDAR-based approach to directly assess the risk of an outage during a storm in locations where tall vegetation and overhead power lines are close. Other authors have proposed using ground Mobile Laser Scanner [214] or even airborne laser scanners and LiDAR data [32] to map vegetation and other urban objects along roads, providing both a high coverage area and accuracy. However, these data modalities are less available than simple RGB street-level images.

Street-level images allow for the visualization of tree entanglements with power lines from multiple points of view, leading to better results than aerial and satellite images in urban areas, especially in places where trees can be taller than pylons and cover the wires. Furthermore, ground-level imagery allows the analysis of various aspects from the urban environment [8, 16, 68], for instance, urban vegetation distribution [22], or the relationship between the environment (e.g., greenery, littering) and health or safety outcomes [129, 95]. Berland et al. [15] study the applicability of the GSV as a tool for auditing trees through a virtual tour, showing that GSV is more cost-effective and less labor-intensive than field audits. Cai et al. [22] propose a method to directly estimate the Green View Index (i.e. the amount of greenery perceived by pedestrians) from GSV images using a CNN. They compare three CNN-based methods: supervised/unsupervised semantic segmentation, and regression, showing that regression outperformed the other two. To assess the main input image features responsible for the Green View Index estimation qualitatively, they explore the gradient-weighted Class Activation Map (Grad-CAM++) [26] visualization method qualitatively.

A recent study by Liu et al. [107] assessed the urban tree inventory across five districts spanning 258.27 km^2 in Jinan, China. In their pipeline, they collect street-level images from Baidu [12] and Google Street View [81]. They annotated the images with bounding boxes encapsulating each tree, with the added task of assigning a tree species to each bounding box. The species classification encompassed eight distinct tree species and a category for 'other'. The bounding boxes encompassed completely the annotated trees. In addition to species classification, they also estimate the heights of the trees with a Monodepth2 [61] network trained on the KITTI stereo dataset [59] and validated using depth maps collected from the Google Street View API. Their findings demonstrate that detecting and classifying the species of the trees are challenging tasks because of the high variability in the appearance of the trees.

Ambiguity in the wild Challenging urban images have several natural sources of difficulty, including weather degradation such as haze, low light, and rain [212]; scene

complexity with cluttered backgrounds [108], occlusion [108]; and even the camera distance to objects of interest may be a source of difficulty [52]. A common characteristic of challenging images is that the visual search task for humans seems harder on them as evidenced by [52, 16, 124].

Detecting hard instances There are detection and quantification approaches for estimating Instance Hardness (IH). The former deals with out-of-distribution (OOD) detection during test time, while the latter assigns a score to each instance, allowing a classifier to learn sequentially from easier to harder instances [221], or data-pruning [147], for instance. In OOD detection methods and should be avoided during test-time [127]. Uncertainty estimation methods characterize hard instances as inputs with a label set disjoint from the training label set, with predictions for such instances having low confidences and/or high entropies [100]. Hendrycks et al. [69] propose two datasets of natural adversarial examples: ImageNetA and ImageNetO, with images similar, and from the ImageNet dataset [169], respectively. They use adversarial filtration to compose them, a technique where a trained model is used to sample instances misclassified with high confidence. ImageNetA contains new images with the same labels used to train the filtering model, but are not part of the training dataset. ImageNetO is constructed by removing ImageNet-1K images from the ImageNet-22K dataset. The remaining samples have a label set disjoint from the ImageNet-1K one, i.e. they are OOD samples with respect to the ImageNet-1K dataset; next, the filtering model trained only with ImageNet-1K predicts labels for the remaining images. Only those with a high confidence prediction are kept. Russakovsky et al. [169] show that natural adversarial samples are hard to other networks, that is, their 'hardness' generalizes, thus other models are also more likely to misclassify them.

Instance Hardness and Viewing Times Instance Hardness (IH) and Hardness Measures (HM) are measures to quantify how difficult it is to correctly classify an instance [183]. However, IH is relative to the choice of learning algorithm used [183], while HM assesses individual components contributing to the IH of an instance, providing insight into why some instances are harder than others, regardless of the learning algorithm used. Understanding why instances are difficult is important because models trained only on easy images often fail to recognize objects in 'hard' images [212]. Moreover, such an understanding helps to design specialized approaches. For example, dataset imbalance and noisy labels are both issues that can decrease the performance of a classifier. While the former can be mitigated by instance re-weighting strategies [96, 106], other methods are better suited for noisy labels [78].

Mayo al. [124] explore the time one takes to label an instance, known as viewing time, as a proxy for HM, showing that images with longer viewing times are more likely to be misclassified by humans and are correlated with C-scores, Prediction Depths, and Adversarial Robustness [63]. Beidi et al. [27] propose the Angular Visual Hardness (AVH), which is the angle formed between a vector embedding of an instance and the weights associated with a class in the final layer of a classifier. They propose the Human Selection Frequency (HSF) as a measure of human visual hardness. HSF is the fraction of human annotators that assign the correct label to a given instance. They show that AVH and HSF are strongly correlated. Thus AVH can be used as a proxy to assess human visual hardness.

Several works use statistics from trained Deep Learning Neural Networks (DLNN) as proxies to quantify instance hardness [63, 83, 13, 147]. Paul et al. [147] propose the GraNd and EL2N scores, which depend on the cost and on the error vector of an instance. GraNd depends on the gradient norm of the cost function, while EL2N depends on the expected L2 norm of the error vector. They show that instances with the lowest scores are redundant and can be removed from the training dataset, while instances with the highest scores are usually outliers or have noisy labels. Jiang et al. [83] propose the consistency score of instances (c-score for short). Given a trained DLN f that correctly classifies an unseen instance x, the c-score of this instance $C(x)$ is proportional to the size of the training dataset required to train f. C-score is a proxy for sample complexity as models require smaller/larger training datasets to correctly classify easier/harder samples with low/high c-scores. Baldock et al. [13] explore the relationship between intermediate features of a deep learning network and instance hardness. They compare the outputs of k-NN classifiers on intermediate features of a trained deep learning network (DLN) for a single instance. They propose the measure Prediction Depth defined as the number of subsequent k-NN classifiers (starting from the last one) that agree on a label for an input instance. They show that the Prediction Depth (PD) is correlated with how likely an input instance is correctly classified by the DLN, which implies that PD is inversely proportional to IH.

Chapter 4

INACITY
INvestigate and Analyze a CITY

This chapter is partially based on the paper of the same name [6].

In this chapter, we present several improvements to the INACITY platform. One of the main problems of the first implementation of the platform was that queries usually had a large number of inner joins, and we wanted to implement a more natural way to traverse a sequence as if we were capturing a video from inside a car.

We integrated INACITY with Neo4j graph-oriented database to improve its performance and to retrieve sequences of images in the same order as they are collected. We present how to collect and store geographical data from/into multiple databases, including the graph-oriented Database Management System Neo4j [2].

Our main contributions here are twofold:

- A deep analysis and thorough description of INACITY components and behavior;
- An extension for graph-oriented databases, with time analyses for some queries.

4.1 Software description

INACITY, like any other software, is an evolving piece of software and was nearly completely rewritten during this Ph.D. project, and this section presents the software architecture, components, and classes.

INACITY is modular, and components responsible for collecting data in the back-end have an interface in common, whose implementation is the only requirement when implementing new ones.

4.1.1 Software Architecture and Functionalities

We based INACITY in the client-server model [162] and implemented a *back-end* (data access layer) and a *front-end* (presentation layer) component. We made INACITY available

as a web application rather than having a client application for each major OS (e.g., Linux, macOS, and Windows) to maximize its accessibility, as most web browsers (e.g., Chrome, Firefox, Microsoft Edge) can access it. We made available Docker containers [1] to ease the deployment and make the platform even more accessible.

We developed the back-end with Python 3 because it is friendly for newcomers, has a rich set of libraries and packages publicly available, and a considerably large body of scientific work produced with it (e.g., the Scikit libraries family [151]) and web-development frameworks (e.g., Django and Flask), which are tuned to deal with database modeling even with geographical data.

We have chosen Django framework [55] as the core technology for the front-end and back-end because it provides all the machinery for handling web-based requests (i.e., REST-based requests), database access and modeling, user authentication and authorization, real-time communication (with the extension Django-channels), and has a stable and large community. We designed the front-end as a visualization tool that consumes data from the back-end, using Django template language (based on HTML) and Javascript to develop the front-end, and a user can create an account, log in, and manage work sessions in the front-end.

In the back-end, we define three main tasks, Geographical Information Systems (GIS) and Geographical Image Databased (GID) integration and image processing with Computer Vision (CV). We implement a Manager class for each main task using the design pattern known as Strategy [200]. Each Manager delegates requests to specialized classes that share an interface in common. Note that our Manager classes are unrelated to the Django Manager class [56]. The flow of a request to the back-end is as follows:

1. Django infrastructure delegates this request to the Manager Component (MC).
2. The MC delegates it to the class responsible for collecting/generating data for the request.
3. The data collecting/generating class returns data to the MC.
4. The MC formats the data received (if needed) and returns it to the infrastructure, which returns it to the caller that originated the request.

It is worth noting that the communication with the back-end has a REST API. Any client application can request to the back-end, not only the front-end developed in the INACITY platform. Figure 4.1 shows a diagram describing the back-end components with some comments about their functions and the relationship between Managers, abstract classes, and derived counterparts. The abstract classes `MapMinerManager`, `ImageProviderManager`, and `ImageFilterManager` define an interface for Manager classes, which delegate requests from a front-end client to specialized associated classes. The **URLs** component defines end-points that external clients can call; those end-points define the REST API functions of the back-end.

Figure 4.2 shows a class diagram describing the Manager classes, arranged according to the Strategy design pattern. For example, the class `OSMMiner` is a subclass derived from `MapMinerManager`, and it provides a unified way to collect data from the `OpenStreetMap` GIS [142] implementing functions with interfaces defined in its base class, which allow

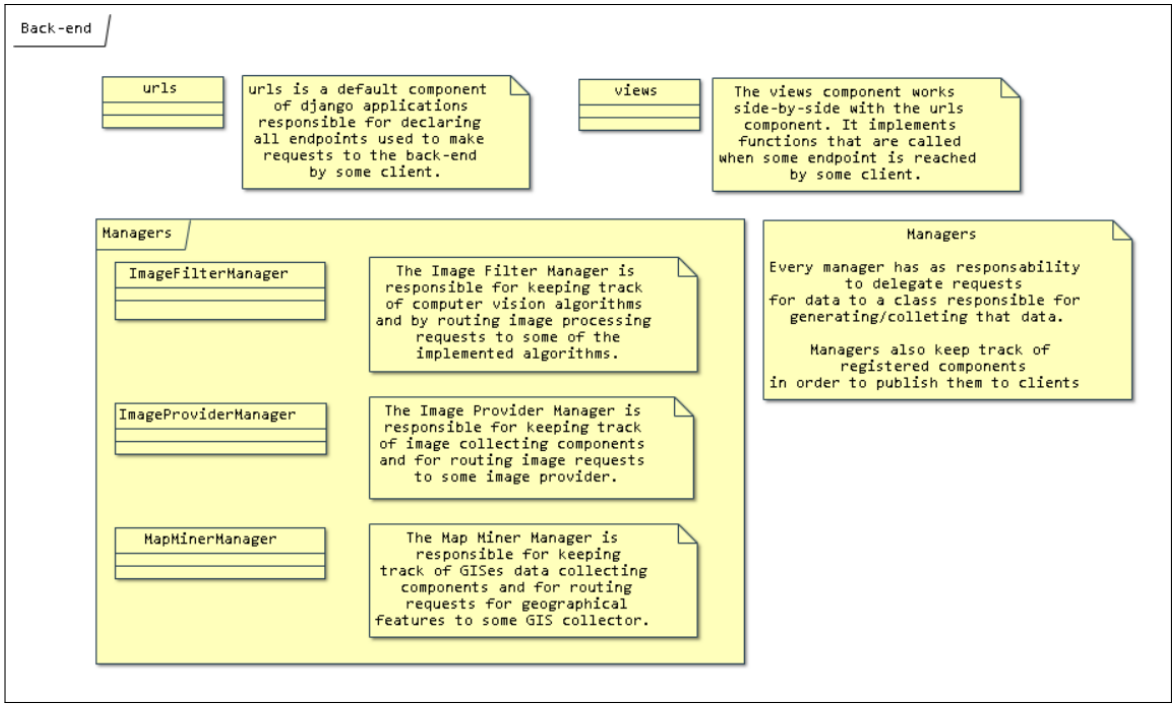


Figure 4.1: *back-end overview*

the `MapMinerManager` class to collect data from the `OpenStreetMap` GIS independently from the data model or the connection details. The `OSMMiner` class translates requests from the `MapMinerManager` to queries for the `OpenStreetMap`, and the response from the latter to a format in common (i.e., GeoJSON [21]) that can be transmitted back to the front-end client. Concerning the detection of urban features, one possibility would be to implement deep learning neural networks to detect traffic signs [10] in images collected from a crowdsourced imagery platform such as KartaView (previously known as OpenStreetCam) [3]. By subclassing the abstract base classes `ImageProvider` and `ImageFilter`, one can integrate KartaView (to collect the images) to INACITY to process the collected images and detect the traffic signs or other urban elements.

Figure 4.3 shows the front-end, a website for an end-user. Its main components are its pages and communication classes the former render updated information and allow the end-user to interact with the website, while the latter communicates with the back-end. The communication components are responsible for encapsulating requests for the back-end and, at the current version, for Google Street View (GSV) servers. The GSVService component, responsible for communication with GSV servers, is implemented at the front-end due to restrictions on GSV, but we store the signing key and implement the GSV request algorithms at the back-end. We use the Google Street View [81] as the standard GID because of its worldwide coverage and because it is a platform commonly used in several works analyzing the urban environment with street-level urban imagery [17].

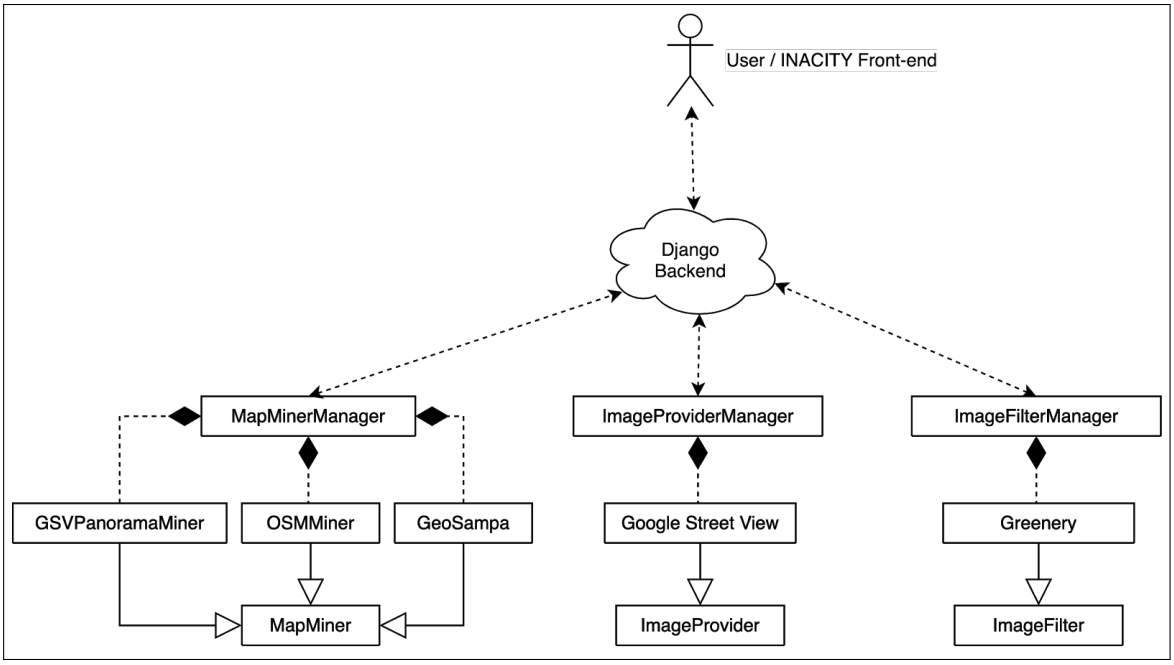


Figure 4.2: *INACITY back-end class diagram.*

4.2 Extending the platform

Integrating a new component to extend the platform requires the user to implement them directly in the source code. There are three main ways to extend the platform, new geographical databases, new imagery platforms, and new Computer Vision algorithms.

We extend the platform by implementing subclasses of specific base classes, which define interfaces for corresponding Manager Components. We describe how to extend the platform in the following sections.

4.2.1 GeoImage

To facilitate the integration between imagery data and geographical data, one can use the GeoImage object. Figure 4.5 shows a class diagram describing the GeoImage component based on GeoJSON. The implementation of this object is similar to the GeoJSON object to achieve better interoperability. As specified by RFC 7946 [21], the GeoJSON object has its fields well defined with a proper semantic, except for the `properties` field of the `Feature` object. This field can contain any JSON (JavaScript Object Notation) object. Therefore, we keep the imagery data related to the coordinates of a `Feature` object inside the `properties` field under the key **geoimages**. Every GeoJSON object is either a `FeatureCollection`, a `Feature`, or one of seven kinds of geometries [21]. We consider each geographical entity as a `FeatureCollection`, usually containing only a single `Feature`.

We treat every geographical entity as a `FeatureCollection`, possibly containing just a single `Feature`. Requests to `ImageProvider` contain a `FeatureCollection`, an array of Features, with the coordinates of the images retrieved by the `ImageProvider`. Figure 4.4 shows a diagram of the `GeoJSON` as an abstract class with nine possible subclasses

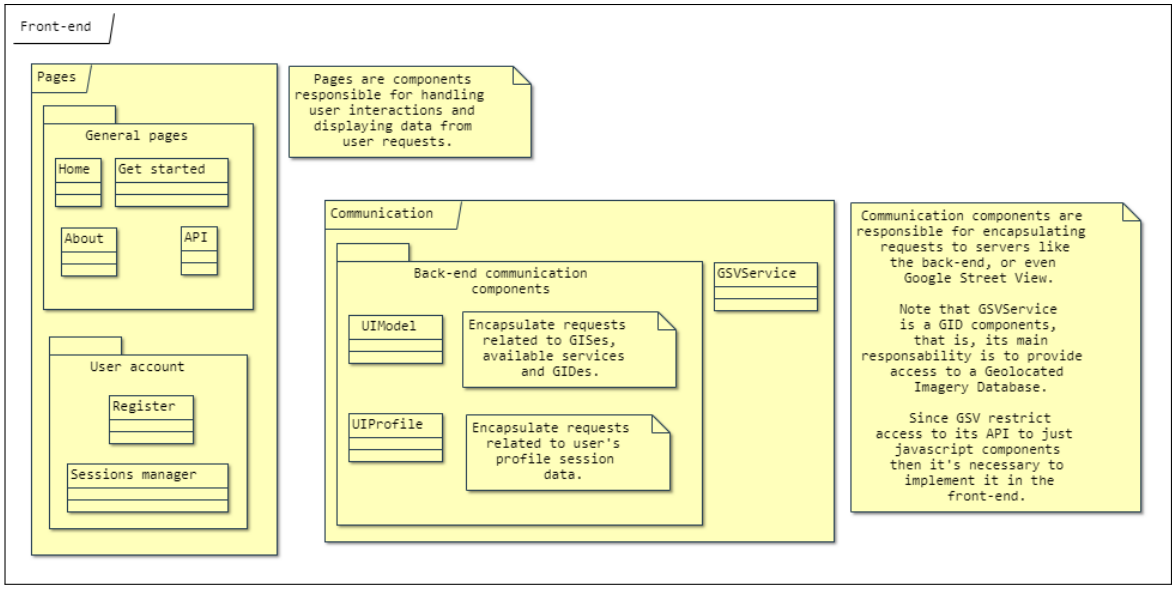


Figure 4.3: *front-end diagram*

(considering that the `Geometry` class could be one of six distinct types).

The GeoImage object keeps metadata and data extracted, using some CV algorithm, from images collected from a GID. We keep data extracted in a separate object called `ProcessedImageData`. Notice that the same GeoImage can hold a reference to multiple `ProcessedImageData` because each image can be processed by different Computer Vision algorithms, yielding multiple distinct extracted data.

We add a new entry with the key `geoimages` into the JSON field `properties` of the `Feature` object to keep the same indexes between the coordinates of the geometry property. That is an easy way to access the `GeoImage` related to a particular coordinate. The `geoimages` entry may have the same structure (i.e., nesting indexes) of the coordinates in the geometry property of the `Feature`. When an image is unavailable for a particular coordinate, an error string will fulfill that particular index position in the `geoimages` entry.

4.2.2 Image filter module

Figure 4.6 illustrates how to include a new Image Filter in the platform. `Image Filter` subclasses extract data from images. We use rectangles highlighted in yellow to denote components under development. The `Trees & Powerlines` component (based on Chapters 5 and 6) represents a detector of overhead powerlines that intersect trees. The `Scene Captioning` class provides a textual description of the scene. Similarly, the `Pavement quality` class could provide a score for road damage. All these components inherit from the abstract class `ImageFilter`, which provides an interface in common to process images.

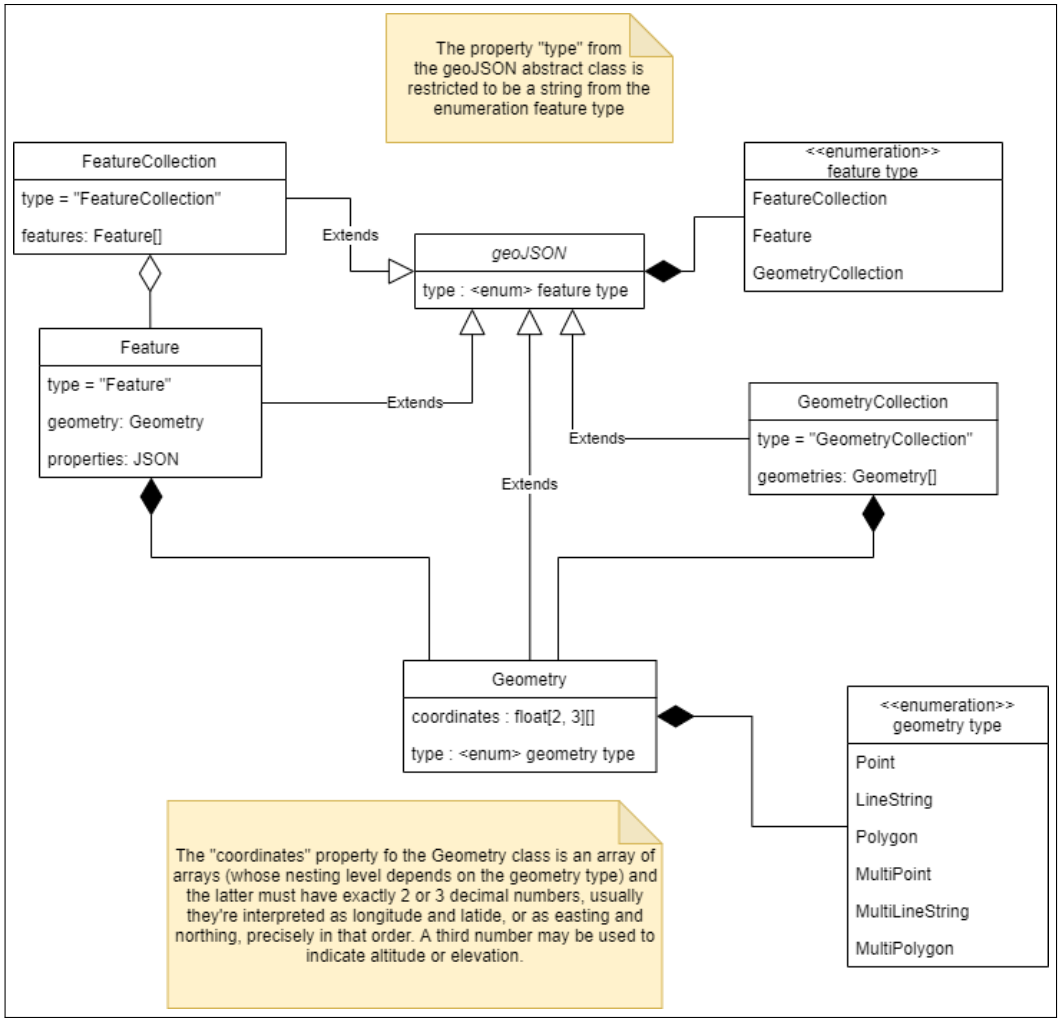


Figure 4.4: *GeoJSON class diagram.*

4.2.3 *Image Provider* modules

Accessing and retrieving geolocated images from a GID is a task for an Image Provider module. They are used to integrate new image sources into INACITY. Since each one of these sources supports different image modalities, like frontal views, panoramic images, RGB-D, and sometimes a combination of them, each Image Provide module performs requests with parameters expected by the external system integrated with INACITY. For example, the GSV API allows one to query for the Panorama closest to some given coordinate and to select the vertical and horizontal angles, the field-of-view, image resolution, and other image parameters. Figure 4.7 presents an overview of the Image Provider modules.

A subclass, derived from the `Image Provider` abstract class, must be created to extend the *Image Provide* module. In Fig. 4.7, the subclasses `Mappilary`[120], `Baidu Total View` [12] and `Crowdsource` are highlighted in yellow because they are under development. The `Crowdsource` subclass allows users to send and retrieve images. An Image Provider module returns a `GeoImage` object to the `ImageProviderManager` class.

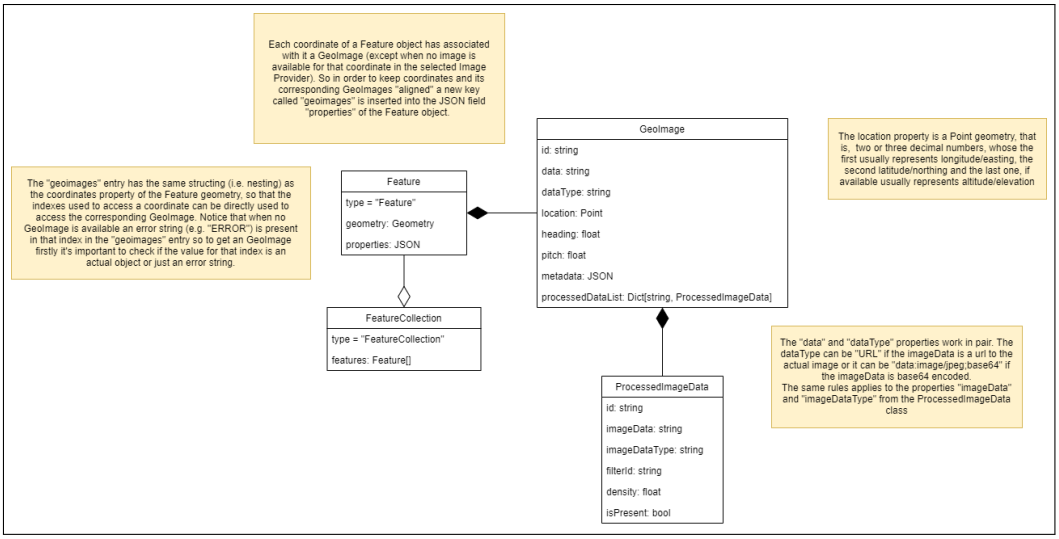


Figure 4.5: *GeoImage class diagram. We use the base64 serialization of an image for simplicity.*

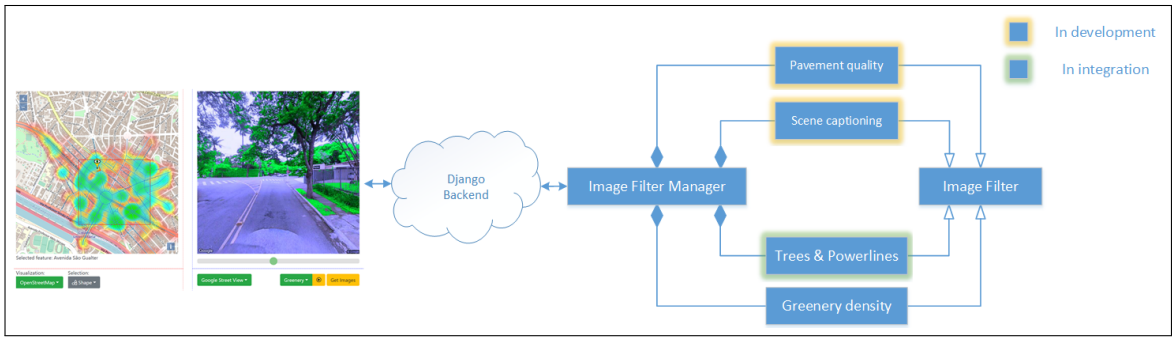


Figure 4.6: *Image Filtering module. All the subclasses of the Image Filter class have an interface in common used by the ImageFilterManager class during a request. The subclasses highlighted in yellow are under development in the current version of the INACITY platform, the one highlighted in green is under integration with the platform, and the rest is already available.*

4.2.4 *Map Miner* module

The *Map Miner* module integrates GISes to INACITY. Figure 4.8 presents the *Map Miner* module and its connections with some GIS databases. The GeoSampa component collects data from the internal PostgreSQL database of bus stops from São Paulo. We inserted an extract from the GeoSampa [175] into the INACITY database to mitigate the number of external queries.

The `OSMMiner` class defines how to collect street geometry from the OpenStreetMap[142] platform, which uses the GeoJSON specification [21] and represent streets as a collection of interconnected `LineString` objects. We use a sequence of collected `LineString` objects (each with geographical coordinates) to get geolocalized images from a GID. The `PanoramaMiner` class performs queries over a Neo4j database instance, a graph-oriented Database Management System (DBMS) [2]. The Neo4j instance is hosted together with the Django server in the same Docker container in the current version and is responsible for relating physical entities (e.g., objects from some GIS), their

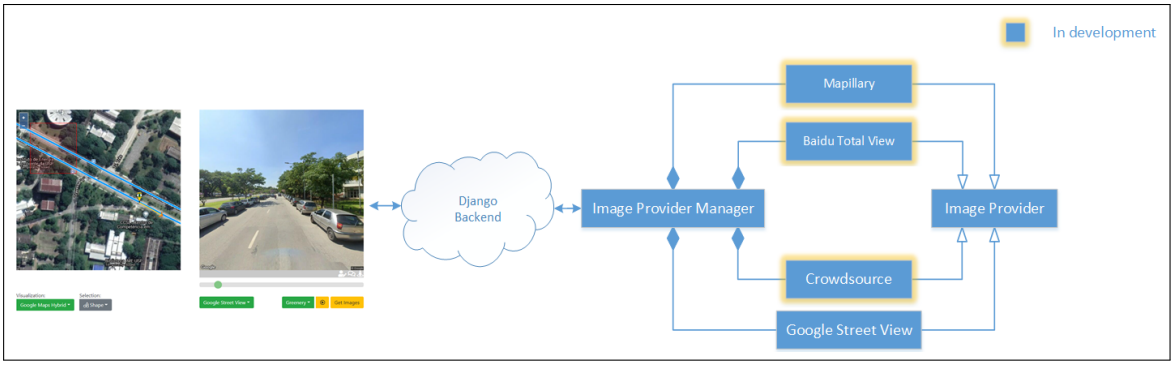


Figure 4.7: *Image Provider module. A Manager component (e.g., ImageProviderManager) receives each user request and routes it to the ImageProvider subclass specified in the request. These subclasses have an interface in common to collect images from a given external image provider (e.g., GSV), thus abstracting the particularities of the API and rules (e.g., the minimum time between requests) of each external image provider.*

images (sampled from some Image Provider system), and even data extracted from those images (using some Image Filter component).

Figure 4.9 shows an example of the graph representation we use to store metadata from Google Street View images and data extracted from them. **Panorama** nodes (i.e., orange nodes) store the address, heading, and pitch angles of the front of the vehicle, timestamp, and other information shared by several images. The **View** nodes (i.e., blue nodes) store image heading and pitch angles. A **FilterResult** node (i.e., gray nodes) stores image-extracted data associated with its connected View node.

4.3 Illustrative Examples

In this section, we present two use-case examples of the platform.

4.3.1 Neighborhood visual inspection

The most simple use case of the platform involves selecting a region of interest and fetching images from that region for auditing neighborhoods [168]. In the INACITY platform, the pictures from each street in the selected region sequentially as taken by the vehicle traversing it. We provide a short video showing this use case [138].

A person selects a region, OpenStreetMap as GIS, and Google Street View as GID in the front-end and presses the 'Get Images' button, which triggers a request from the front-end class `UIModel` to the back-end containing the selections, collecting streets inside the selected region from the selected GIS, and images for them from the selected GID. Figure 4.10 shows a diagram of the process that follows the `UIModel` request. Since this is a request for images from a geographical entity, the `MapMinerManager` component receives the request and delegates it to the `MapMiner` subclass `OSMMiner`, which parses the request with the Overpass Query Language [141] and sends it to the OpenStreetMap platform.

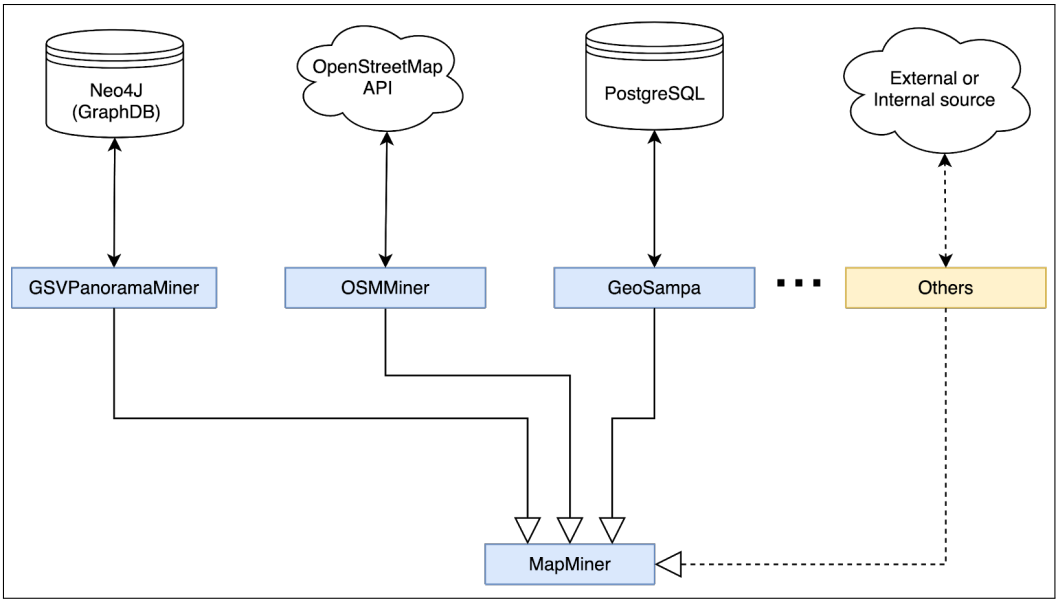


Figure 4.8: *Map Miner module. We deploy the Neo4j and the PostgreSQL databases (cylinder icons) locally, while the components depicted with a cloud icon denote external systems accessed through the subclasses of the Map Miner class. As with the other modules, user requests are routed by a Manager component (omitted in this figure; see Figs. 4.7 and 4.6 for examples) to the Map Miner subclass specified in the request. This subclass must translate the user request into a request to the target GIS (e.g., OpenStreetMap) or local database (e.g., Neo4j), which contains geographical data (e.g., the location of GSV panoramas or even bus stops imported from GeoSampa into the local PostgreSQL database). We use the Django ORM to interact with the PostgreSQL and packages provided by Neo4j to interact with the graph database via the bolt protocol.*

After the OpenStreetMap returns the query results, the `OSMMiner` subclass will format the response, streets geometry, using the GeoJSON [21] specification and return it to the MapMinerManager, which will return it to the front-end as blue lines on the digital map.

Next, the `UIModel` class sends a second request containing the geographical entities collected in the first step and the selected GID. This request follows a similar flow as the one used to collect street geographical data, except that this time `ImageProviderManager` receives the request. As the user selected Google Street View as GID, the `ImageProvider-Manager` will delegate the request to the `GoogleStreetViewImageProvider` subclass of the `ImageProvider` component.

4.3.2 Urban feature visualization (greenery)

Visualizing the distribution of urban features, like greenery, is another use case of the INACITY platform in a given geographical region. The pipeline starts as before (i.e., a user selects a region of interest) and then triggers a request for processing the images collected during a neighborhood audition. In the video accompanying this paper, we show a filter called `Greenery filter` to estimate the `Green View Index` by segmenting which image parts correspond to green vegetation. The **density** property of a `ProcessedImage-Data` object associated with the `GeoImage` stores the image proportion marked as green

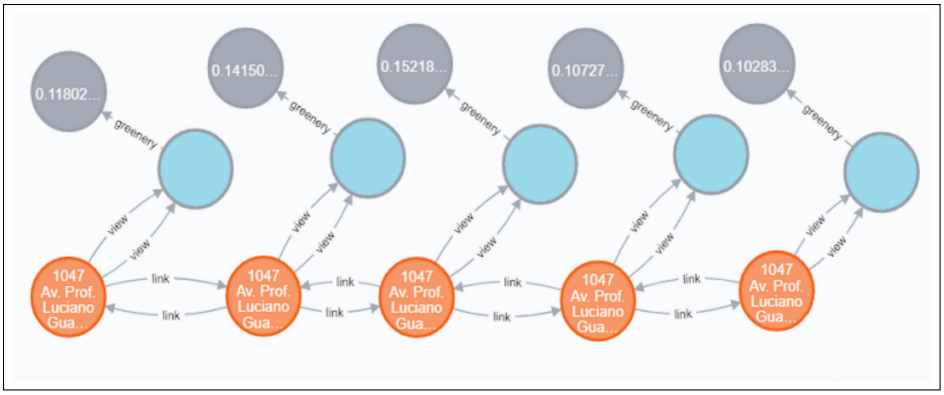


Figure 4.9: *Nodes stored in the local Neo4j instance. Panorama nodes in orange represent locations (i.e., latitude and longitude) of images. Views (in blue) represent images and encapsulate the horizontal (heading) and vertical (pitch) angles of the camera from the front of the vehicle to the magnetic North and flat ground, respectively, at the moment of the shot. Filter results (nodes in gray) maintain data extracted from a view by some ImageFilter subclass (Fig. 4.6).*

vegetation. When the `FeatureCollection` is returned to the front-end, its features and corresponding `GeoImages` (if available) will have an associated `ProcessedImageData`, which can have extracted data as the density of Green View Index displayed as in Fig. 4.11, for example.

Besides the heat map, INACITY can present some urban features overlayed on the original images. Figure 4.12b shows an example of a processed image. Greenery and non-greenery regions, detected by the back-end, are highlighted in green and in blue, respectively, and overlayed with the processed image, allowing the inspection of the greenery filter module, which we implemented with the Python packages numpy [133] and scikit-image [202].

4.4 Demo Site, Impact, and Limitations

A non-specialist user can use the public instance to select a region, query images, and extract features from them. Available geographical features include street network locations from OpenStreetMap [142] and Bus Stops from GeoSampa [175], and Google Street View [81] is the only implemented imagery source. Additional capabilities can be implemented by developers and researchers in the future, possibly in locally deployed instances of the INACITY platform.

INACITY can be part of a more extensive pipeline of research. For instance, in [137], we collect images from some locations in the cities of Porto Alegre (BR) and São Paulo to build a machine-learning model to detect entanglements between electric wires and tree branches, which can be coupled into INACITY by subclassing the ImageFilter class, thus enabling the platform to help city managers to detect tree and wire entanglements and prevent accidents.

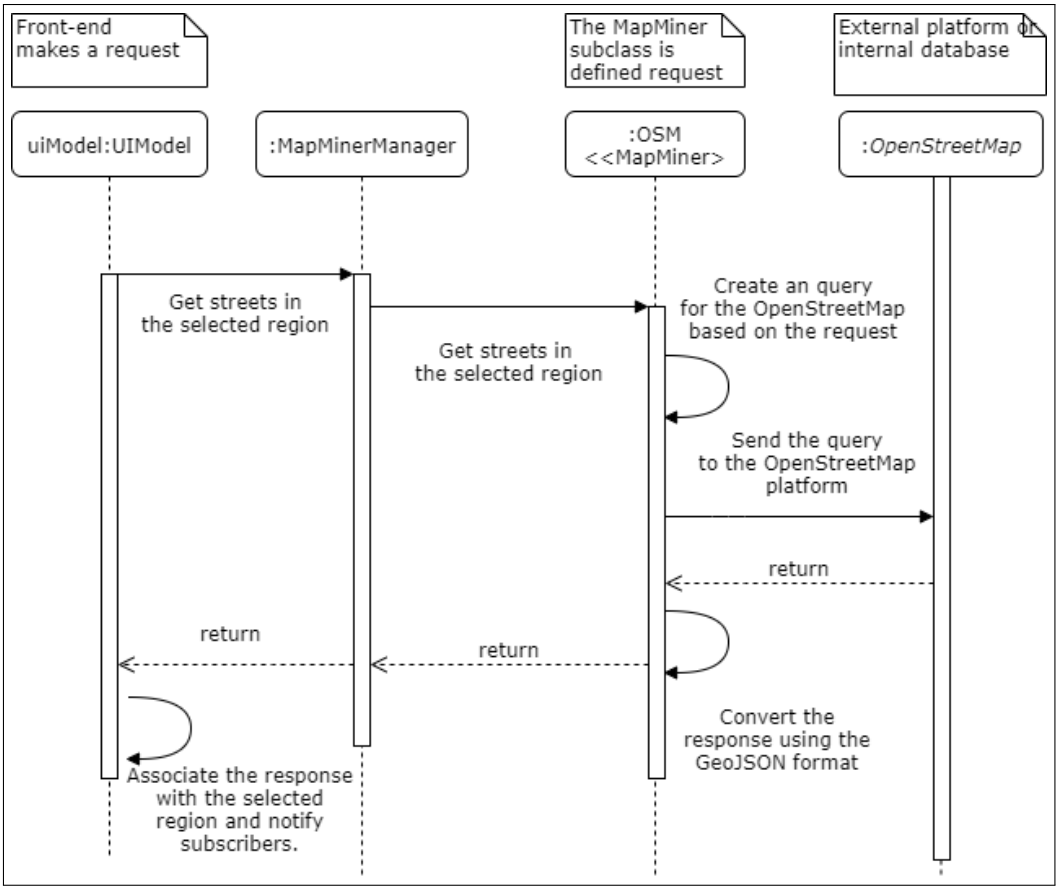


Figure 4.10: *Flow diagram for street collection.*

4.4.1 Quota module

The quota module keeps track of how many calls a user performs. We host a public instance of INACITY is available at http://inacity.org, and due to GSV costs, a user-level quota system is necessary to allow more users to try the platform, but users can provide their own GSV credentials and avoid the quota limits. The class `QuotaManager` is responsible for registering a new entry in the database and keeping track of the available quota of a user. Figure 4.13 show the main components of the quota subsystem. We use the decorator factory **quota_request_decorator_factory** to track the usage of a function. The parameters of the decorator factory are **default_user_quota**, **default_anonymous_quota** and **skip_condition**. The first two specify how many calls will be available for a registered and an anonymous user, respectively, and the last parameter is a Boolean function used to disable the quota manager when the user provides GSV credentials.

4.4.2 Performance tests

We created a benchmark test to assess the effect of multiple simultaneous requests to the back-end. The tests consist of collecting streets and images for two disjoint urban regions. Each region has a different size and number of streets and images. Table 4.1 presents the results and information details of both areas used in the benchmark. Rows in the table represent the total area (in squared meters), number of streets, and number of

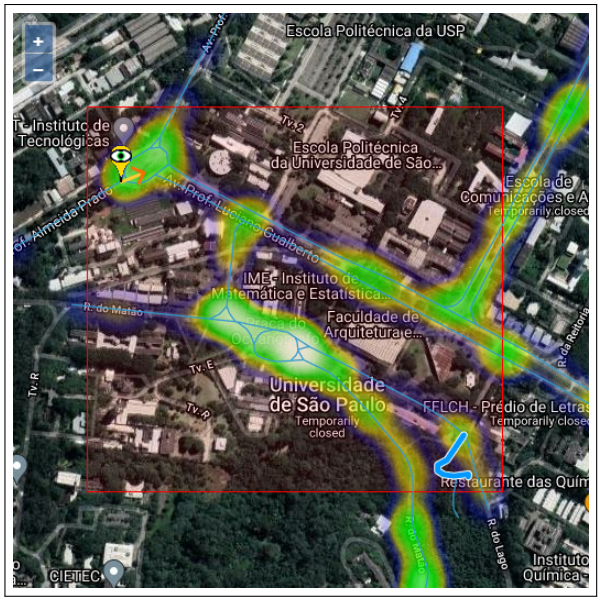


Figure 4.11: *Greenery heat-map (lighter colors for higher values) for a region selected (i.e., the red square).*

(a) *Unfiltered image* **(b)** *Filtered (greenery) image*

Figure 4.12: *(a) An example image from Google Street View. (b) Image shown in (a) after being filtered by the 'Greenery density' subclass. We highlight greenery regions in green and other regions in blue.*

images collected, and several statistics for time elapsed to collect streets geometry (i.e., points) and imagery and to process images. We split the minimum, maximum, average, and standard deviation times between collecting streets in each region and collecting the images for the corresponding streets collected. Finally, the time spent processing all the collected images using the currently implemented `Greenery` image filter. We performed 100 requests to collect the response times, 50 for **area 1** and 50 for **area 2**. We performed ten concurrent requests at all times, interspersing requests for **area 1** with requests for **area 2**. We used an Intel Xeon E5420 2.5 GHz with eight cores to perform the benchmark tests.

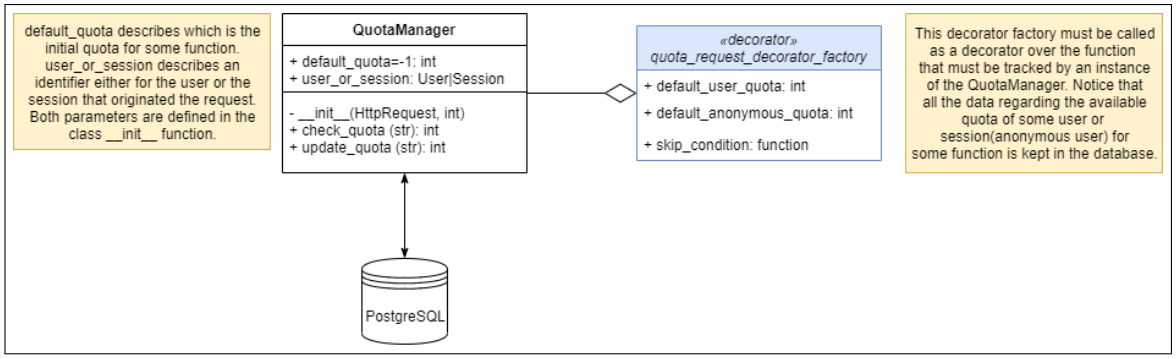


Figure 4.13: *Quota system overview.*

	Both areas	Area 1	Area 2
Area	348906 m^2	20931 m^2	327975 m^2
Num. Streets	27	2	25
Num. Images	429	71	358
Min Time (Streets)	7.78 s	7.78 s	7.89 s
Avg. Time (Streets)	9.14 s	9.36 s	8.91 s
Std. Time (Streets)	2.2 s	2.34 s	2.05 s
Max Time (Streets)	16.76 s	16.68 s	16.76 s
Min Time (Images)	8.57 s	8.57 s	108.36 s
Avg. Time (Images)	84.55 s	18.79 s	152.04 s
Std. Time (Images)	68.79 s	9.52 s	19.81 s
Max Time (Images)	194.92 s	50.13 s	194.92 s
Min Time (Greenery filter)	153.70 s	153.70 s	812.96 s
Avg. Time (Greenery filter)	494.61 s	170.53 s	843.63 s
Std. Time (Greenery filter)	342.97 s	10.97 s	15.32 s
Max Time (Greenery filter)	867.48 s	187.24 s	867.47 s

Table 4.1: *Multiple statistics taken upon the execution times and request sizes (in terms of streets and images collected) for two distinct regions.*

4.5 Discussions and Conclusion

We created the INACITY platform to facilitate collecting and integrating geographical data, images, and computer vision algorithms. Each module base class allows the independent implementation of new Geographical Imagery Databases, Geographical Information Systems, and Computer Vision algorithms. The front-end is simple and allows end-users (e.g., citizens, developers, researchers, or government administration agents) to use the platform to gather data for future use and, because it is open-source, further improve the platform by modifying it to their needs.

We can present images as a video because they appear in the graph-oriented database as a sequence of connected nodes. Retrieving geographical data and data extracted from images is easier and faster since Panorama and View nodes in the graph represent them, respectively, so traversing from a View node to its connected Panorama node allows one to collect data stored at both nodes. A future application for INACITY and our proposed extension is integrating multi-modal geographical data, similar to [186], to detect irregular waste disposal or even to track localized complaints reported by citizens.

Chapter 5

Detecting tree and wire entanglements with deep learning

This chapter is partially based on the paper of the same name [137].

In this chapter, we propose a method to classify images with trees entangled or not with power lines[1], and a new dataset with images labeled using an in-house developed application crafted to ease the annotation process.

We fine-tune a pretrained Convolutional Neural Network (CNN), named MobileNetV2 [174], with the annotated images to detect entanglements between power lines and trees. The network achieves over 74% of test accuracy on a test set of 1001 images from two Brazilian cities.

The main contributions of this research are:

1. An approach to detect tree and wire entanglements in street-level imagery based on deep learning;
2. An annotated dataset of tree entanglements with power lines;
3. An open-source software for annotating street-level images.

5.1 Method

This section presents our solution to locate tree entanglements with power lines. We first present a dataset of more than six thousand images collected from São Paulo and Porto Alegre, two cities in the Southeast and South of Brazil, respectively, using Google Street View. We then present the strategy to annotate the images, and we finish the section presenting the strategy to tune a Convolutional Neural Network using transfer learning.

[1] Our code is available at https://github.com/arturandre/tree-wires.

5.1.1 Image dataset

We start collecting geographical position data of 2981 trees from an open government public database called GeoSampa [175], with data for SP and 3111 positions of power poles on POA from the energy company CEEE's website [23].

To collect an image using INACITY, we first sample the nearest GSV panorama from a given location L of the desired target object (i.e., trees from GeoSampa or power poles from CEEE's dataset). After finding the nearest Panorama P from L, we extract an image from P such that the camera will point from P to L, making the target object appear approximately at the (horizontal) center of the image. Notice that there might be inaccuracies in the positions of P and L, making the target object further from the image center, or eventually, the time when the image was taken and the position L was recorded are too far apart, thus the target object could have been removed. Figure 5.1 shows four examples of images from São Paulo city and four examples of images from Porto Alegre city. Notice that sometimes the target objects may not be visible due to occlusion, or they may not be there when the picture was taken.

Figure 5.1: *Sample images from (top row) São Paulo and from (bottom row) Porto Alegre.*

5.1.2 Dataset labeling

We label each image as a vector of three binary values, trees, power lines, and entanglements. For example, if an image contains a tree, overhead power lines, and no entanglements, the binary vector associated with this image will be $(1, 1, 0)$. We used the SLIL application (presented in Appendix A) to label the GSV images. Figure 5.2 shows a screenshot from SLIL. The image being labeled has trees, overhead power lines, and an entanglement between them.

Some images can be challenging to label for reasons like the camera perspective. In Fig. 5.3, it is hard to decide whether the tree is touching or not the power lines because the wires and the trees seem entangled at the top. However, they seem far from each other at

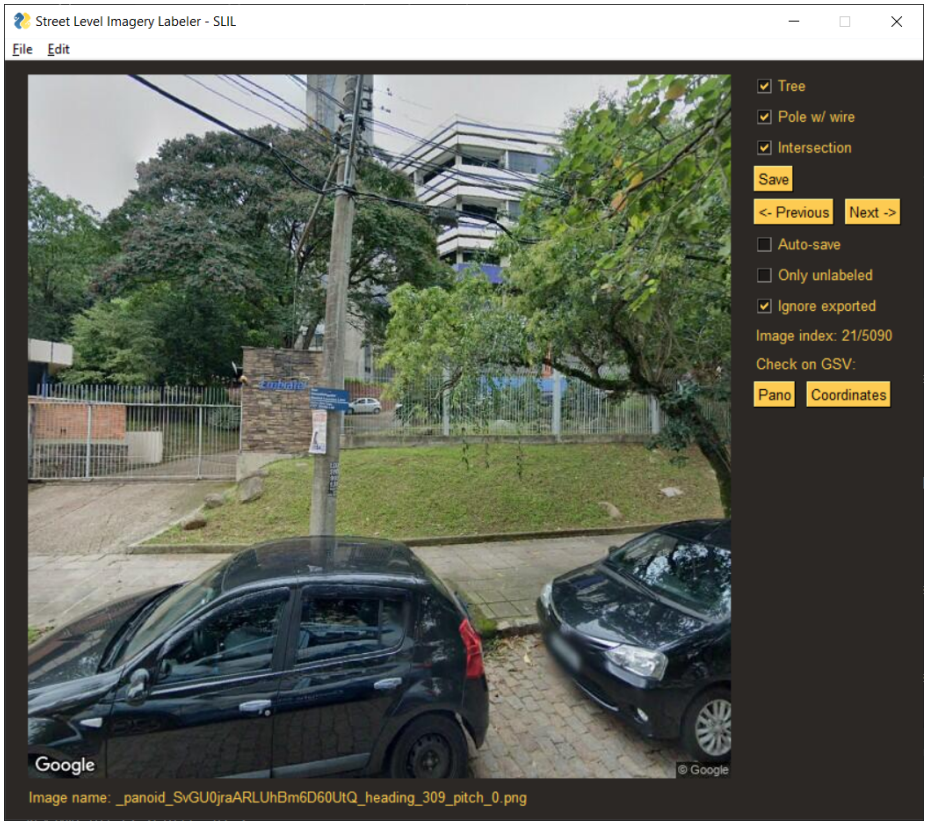


Figure 5.2: *Screenshot of the 'Street Level Imagery Labeler (SLIL)' application, used to annotate the GSV images in the dataset.*

the bottom, leading to the opposite conclusion. Figure 5.4 shows another example where the camera is pointing in a direction parallel to the power lines, and a tree below the power lines is very far from the camera. Notice that the power lines are no longer visible when they intersect distant trees. Figure 6.1b shows a challenging image due to Sun glare.

Figure 5.5 shows the distribution of labels for São Paulo city (left) and Porto Alegre city (right) in the test dataset. The labels 1 and 0 indicate the presence or absence of the objects: 'Tree,' 'Electric wire' and 'Entanglement'. To compute the histograms (Fig. 5.5), we consider how many images are labeled with each of the classes present in the test dataset (i.e., Tree, Electric wire, and Entanglement) from each city separately. If an image is labeled with a given class, it counts as one positive case of that class in the histogram. There is an unbalance between the number of labels with 'Tree' in São Paulo city and 'Electric wire' in Porto Alegre city because we used the positions of trees and poles for collecting images in SP and POA, respectively.

5.1.3 Training the model

We perform transfer learning using a model pretrained on the ImageNet dataset [169]. Pan and Yang [145] show that transfer learning is an efficient way to take advantage of a much larger dataset, reducing the required size of the dataset for fine-tuning and the number of training iterations. We used the MobileNetV2 architecture [174, 189], a Convolution Neural Network model for object classification, detection, and semantic

Figure 5.3: *Power lines in front of tree tops*

segmentation. MobileNetV2, based on MobileNet [74], is a simplified model adapted to mobile devices with several hardware constraints as smartphones. We have chosen this architecture to employ our method in embedded applications. Figure 5.6 shows a diagram of the MobileNetV2 architecture used in this work. The final blocks MobileNetV2, which we call collectively as *classification head*, are a Global Average Pooling (GAP) [105], and a dense layer with 1000 neurons.

Although our dataset has labels for trees and wires, we only use the entanglement labels for simplicity. Therefore, we replace the classification head with a new GAP and three dense layers, each with 1080, 540, and one neuron, respectively, with sigmoid activations. We fine-tune only the new classification head for 1000 epochs with our proposed dataset. To avoid overfitting, we use the model with the highest validation accuracy as the final one. Figure 5.7 shows our classification head, and Fig. 5.8 shows an overview of our modifications to MobileNetV2.

We used 5091 labeled images to train the model, 2484 and 2607 images from the cities of São Paulo and Porto Alegre (POA), respectively, as explained in Section 5.1. The training

Figure 5.4: *Power lines possibly intersecting with trees in the distance.*

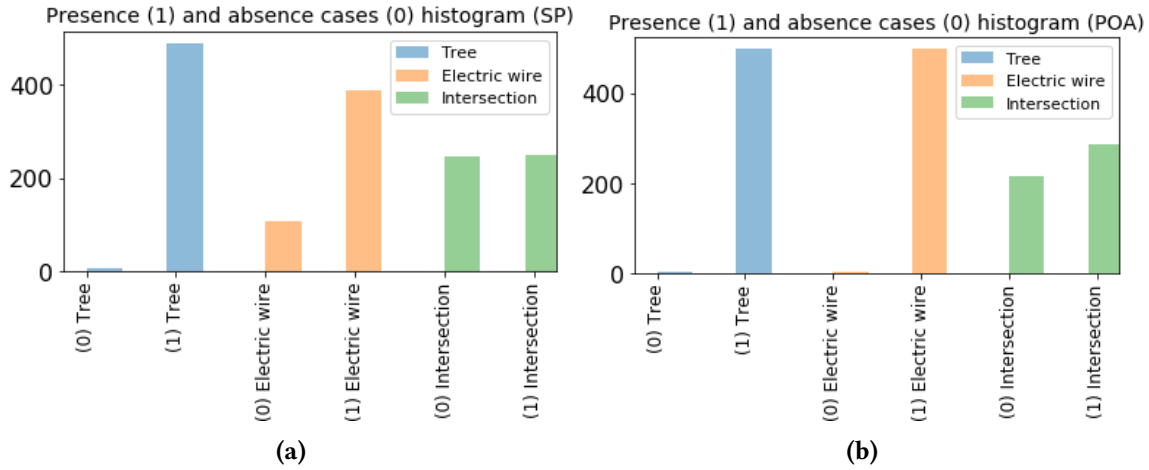


Figure 5.5: *Histogram of with the distribution of different classes for each test dataset in the images (a) from SP and (b) from POA.*

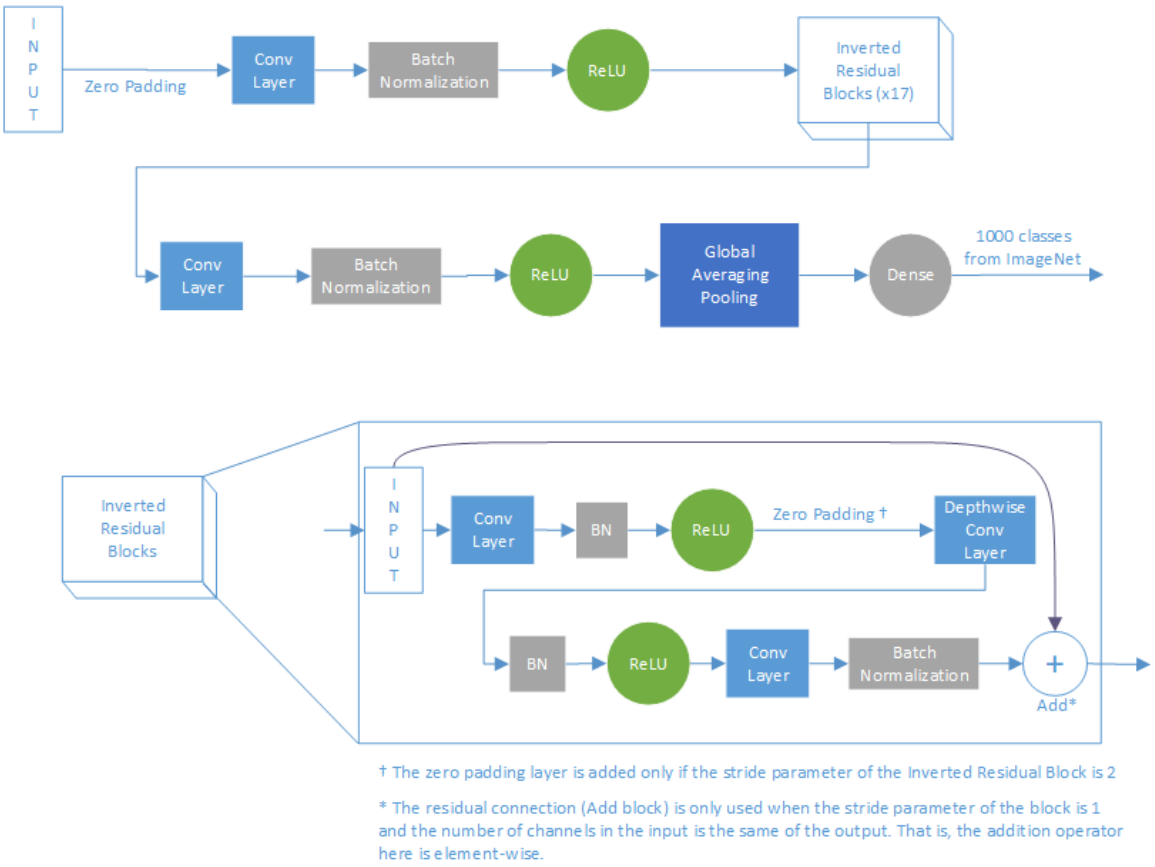


Figure 5.6: *MobileNetV2 architecture. The network has as output a vector representing the probability distribution over the 1000 classes from the ImageNet used to pretrain the network.*

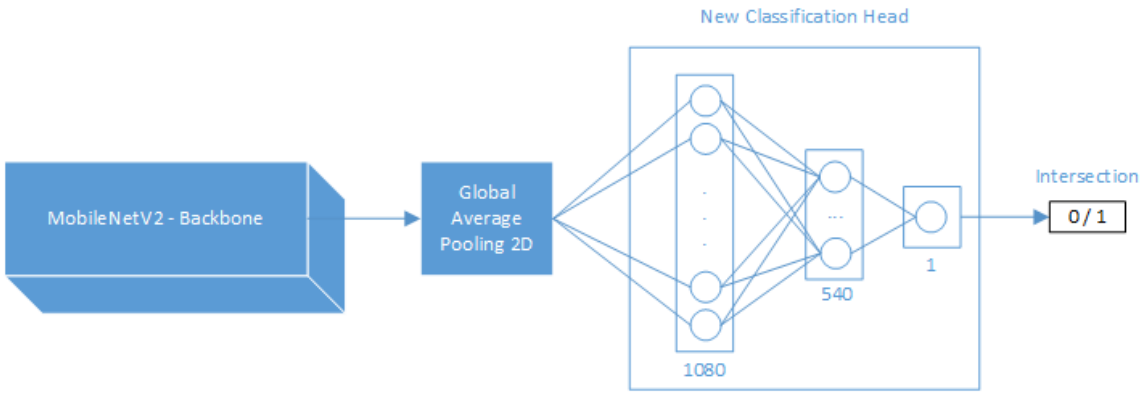


Figure 5.7: *New proposed classification head used on top of the MobileNetV2 backbone. We use two fully connected layers whose neurons have the sigmoid function as activation and a final layer with a single neuron and sigmoid activation function.*

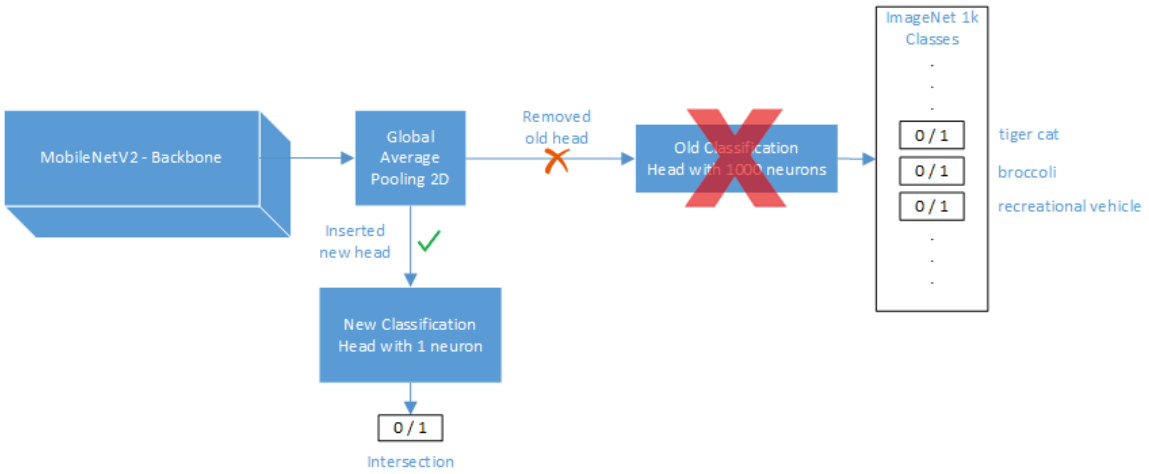


Figure 5.8: *MobileNetV2 architecture adaption. The network is adapted to output a single value instead of a probability distribution over the 1000 classes from the ImageNet used to pretrain the network.*

process uses 80% of the training set effectively for training and 20% to validate and control the whole process, which took 340 epochs to converge.

5.1.4 Interpreting Network Results

Interpreting the results of a network is an essential topic of research, and there are several methods proposed in the literature [218] to help with that. We use a generalization of the Gradient-weighted Class Activation Map (Grad-CAM++) [26], an improved modification of Grad-CAM [178], to interpret and qualitatively assess the results. The Grad-CAM++ intuition is to build an image with a weighted sum of the positive partial derivatives of the output (just before the last sigmoid activation) with respect to the last convolutional layer feature maps (activation map) to show which pixels are relevant to classify the image according to a specific target label defined by the user or by the assessment. The images generated by this method, or saliency maps, are color mapped from blue to red, indicating regions that are less to more relevant to the classification output. Fig. 5.9a shows an entanglement, and Fig. 5.9b its corresponding Grad-CAM++.

The network correctly classified the image as positive, and the Grad-CAM++ shows the entanglement region as relevant for this prediction. Figures 5.9c and 5.9d show another example with no tree entanglement and its corresponding Grad-CAM++ output, respectively. The model has misclassified it as a negative case, but its Grad-CAM++ correctly highlighted relevant regions for the classification, namely the image regions with wires and tree canopies. To assess the results qualitatively, one can visually inspect the Grad-CAM++ output for a test image and check whether the highlighted visual clues are the ones expected.

5.2 Quantitative results

The final validation accuracy was 81.94% (on the 20% of the training set), and the training accuracy was 96.64% (on the 80% of the training set). Table 5.1 presents a confusion

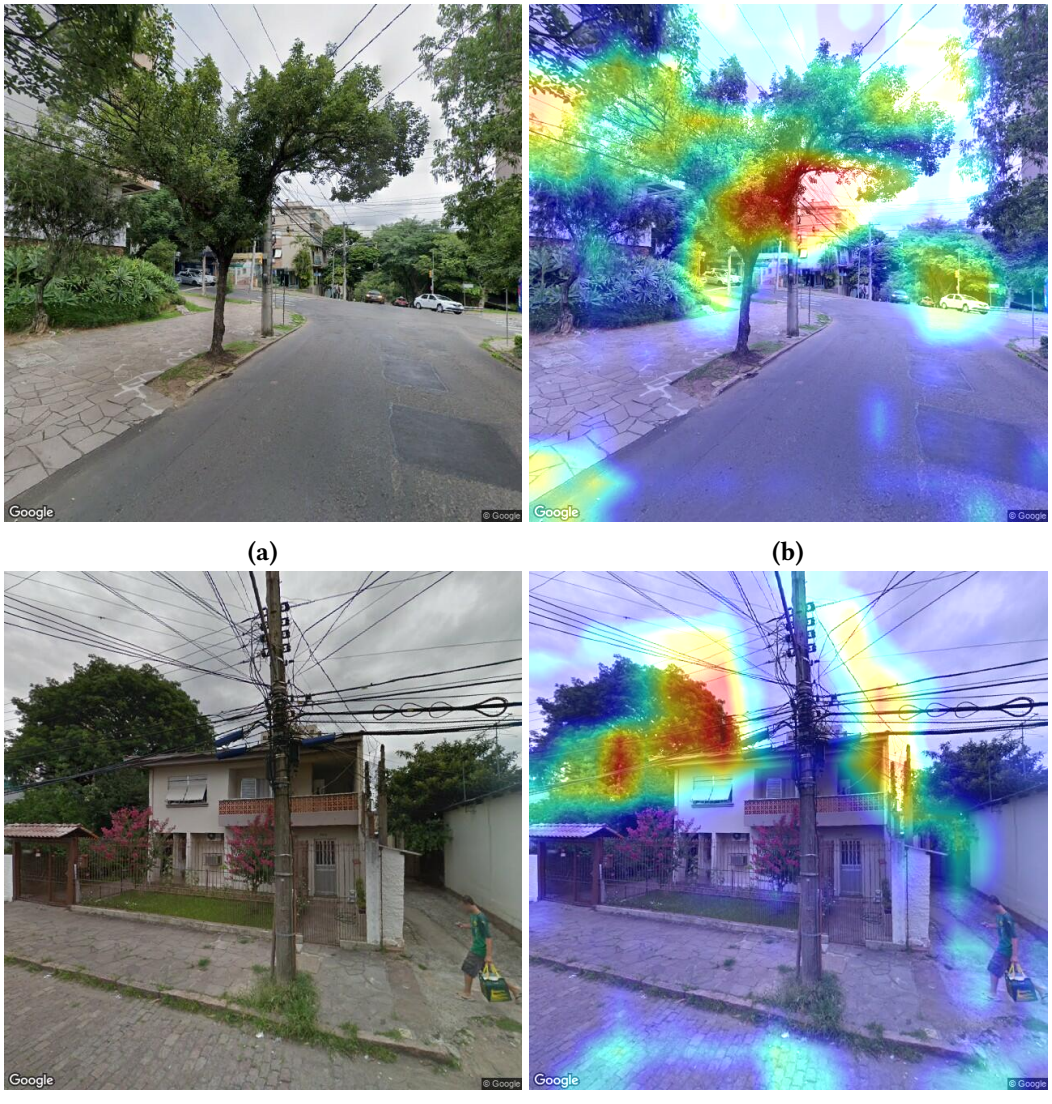


Figure 5.9: *(a): An entanglement between a tree and some overhead wires. (b): The Grad-CAM++ image with the dark reddish regions indicates the places relevant for the classification (positive) in this case. (c): An image without entanglements. (d): The corresponding Grad-CAM++ shows that the regions with trees and wires are relevant, even though they do not intersect with each other.*

(a) **(b)**

Figure 5.10: *(a): Image classified as a true positive. (b): The corresponding Grad-CAM++ image, showing that the wires passing through the canopy of the tree was identified and relevant for the classification.*

matrix with the number of correctly and incorrectly classified images. When the class assigned by the classifier (rows) agrees with the ground truth (columns), we say the classification is correct (top-left position of the table, or true positive; or bottom right corner of the table, or true negative). When the class assigned by the classifier disagrees with the ground truth, we say the classification is incorrect (top-right position of the table, or false positive; or bottom-left corner of the table, or false negative). A row (column) with 1 represents images classified (manually annotated) as positive cases. A row (column) with 0 represents images classified (manually annotated) as negative cases.

		GT (Train)	
		1	0
Ours	1	1127	137
	0	0	2808

		GT (Validation)	
		1	0
Ours	1	238	122
	0	62	597

Table 5.1: *The confusion matrix shows the distribution of the predictions for the training (with 4072 images) and validation datasets (with 1019 images). In the top left corner are the number of true positives; in the top right the number of false positives; in the bottom left the number of false negatives; and at the bottom right the number of true negatives.*

From the 6092 images collected (see Sec. 5.1.1), 1001 images compose the test dataset. There are 504 images from Porto Alegre city and 497 from São Paulo city. The test dataset contains no images used in the training process, and we use it to assess the generalization capability of the network. Table 5.2 shows three confusion matrices for the overall test set, the São Paulo test set, and the Porto Alegre test set, respectively, from left to right. We obtained an overall test accuracy of 74.63%. For São Paulo the test accuracy was 73.64% and for Porto Alegre city, it was 75.60%.

		GT (Test all)	
		1	0
Ours	1	373	86
	0	166	376

		GT (Test SP)	
		1	0
Ours	1	143	20
	0	109	225

		GT (Test POA)	
		1	0
Ours	1	230	66
	0	57	151

Table 5.2: *The confusion matrices showing the distribution of the predictions for the test dataset (1001) composed of 497 images from São Paulo (SP) and 504 from Porto Alegre (POA).*

5.2.1 Qualitative analyses of the errors

This section presents a qualitative analysis based on misclassified images using Grad-CAM++ (Sec. 5.1.4). Through these analyses, we empirically found directions to improve the classification results. Figures 5.10a and 5.10b show a salience map for a positive case. Figure 5.10a is a true positive. The region containing wires crossing through the canopy of a tree is the most relevant for the model prediction according to Grad-CAM++. This is

(a) **(b)**

Figure 5.11: *(a): An image misclassified as having an entanglement between a tree and electric wires. The camera is just below the foliage, and it seems there is an entanglement, but from another point of view, it is clear that is not the case. (b): The salience map generated with the Grad-CAM++ method reveals that the relevant regions observed by the network are the branches and the wires in the image.*

empirical evidence that the network can correctly detect entanglements because it is tuned to detect the relevant image features that characterize an entanglement. Furthermore, the visual inspection of several misclassified images and their corresponding salience maps shows that some errors are caused by:

- the viewpoint of the camera (see Figs. 5.11a and 5.11b);
- wires that seem to meet with trees far away from the camera (see Figs. 5.12a and 5.12b); and
- bad lighting conditions (e.g. poor contrast) as exemplified in Figs. 5.13a, 5.13b, 5.14a

and 5.14b)

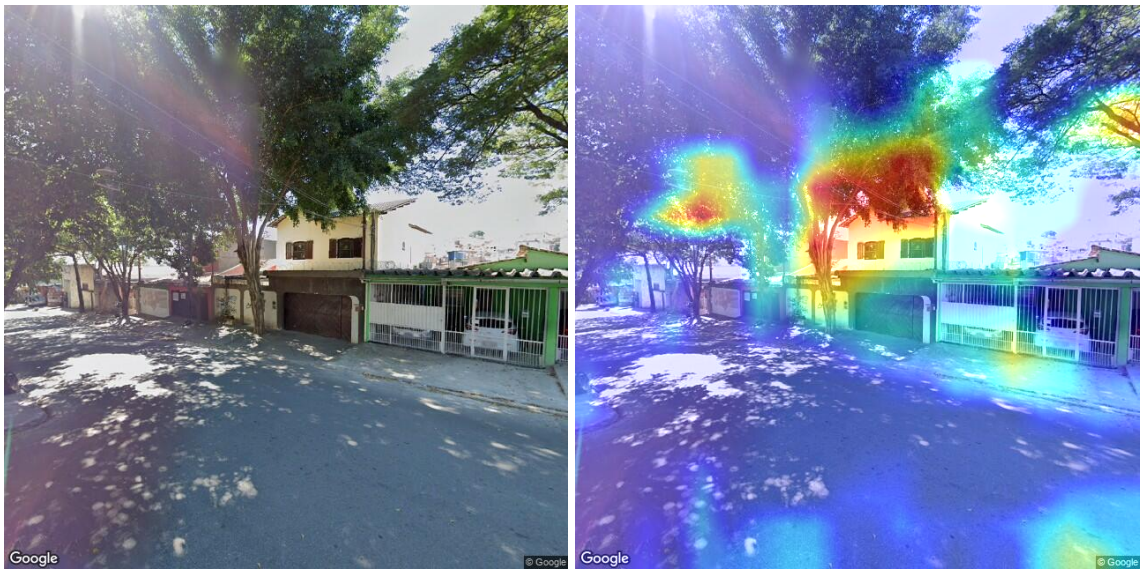
(a) **(b)**

Figure 5.12: *(a): input image misclassified as a false positive. (b): Saliency map showing relevant regions with the foliage of trees and overhead wires (in red). The wires do cross the branches of the trees on the image plane, but one can see that they are indeed apart from each other. The depth information is not available for the network during training so this kind of image is 'hard' to classify correctly.*

5.3 Discussion and Conclusions

We discuss in this section our approach to detecting trees entangled with power lines and our proposed dataset. We observed that several images are challenging to classify due to the perspective and lack of depth information and are confusing for human annotators, which usually take longer and may assign incorrect labels for them, harming the accuracy of the models. To mitigate issues related to challenging images, a user can use SLIL to navigate directly to GSV to look at the same tree from alternative perspectives.

Furthermore, we employed state-of-the-art deep learning networks, and our results show that although they achieve good results, there is plenty of room for improvement. While other urban imagery datasets, such as Cityscapes [34], contain rich semantic annotations for vegetation and poles, they lack annotations for powerlines, and it is not clear how to assess the interaction between two distinct entities based on a 2D semantic map (e.g., how close they are). We use the location of trees and pylons to collect our images, which almost always contain at least one of them. However, we balanced it for entanglements to make it suitable for training a convolutional neural network on the entanglement detection task. We aim to balance it for trees and powerlines by including more images, as we believe that more semantic diversity can help the network to generalize better, and training with multiple classes simultaneously could benefit interpretability with Grad-CAM++ as one can inspect each one separately. Furthermore, we conjecture that masking the bottom of the images could improve the results, as entanglements only occur in the top half of them, and we will try this idea in future experiments.



(a) **(b)**

Figure 5.13: *(a): Input image with Sun glare misclassified as a false negative. (b): The salience map shows that entanglement regions are relevant, but the final classification was negative, possibly because of the low contrast between the wires and the trees just behind them due to sun glare.*

(a) **(b)**

Figure 5.14: *(a): Input image misclassified with an entanglement of wires passing just under the shadow provided by the canopy. (b): The saliency map shows that the entanglement regions were relevant, but similarly to the effect of the Sun glare, one can notice the low contrast caused by the canopy shadow over the wires.*

Finally, one can extend our method with methods for detecting entanglements in above-ground images (i.e., satellite and airborne [93, 87]). The above-ground point-of-view could help in ambiguous cases, and ground-level images usually have a better resolution, which can provide information about entanglements occluded below canopies in above-ground imagery.

Our method reached 74, 6% accuracy in detecting entanglements between trees and wiring, and together with the platform INACITY, can be a valuable tool for better management of vegetation, preventing accidents, and reducing the diseases risk and insects in trees by prompting early pruning when necessary.

The results show that the adapted MobileNetV2 generalizes to new images in cities which similar visual characteristics as those composing our test dataset. Nevertheless, our approach is general enough, and one can fine-tune a classifier in datasets containing vegetation with different visual characteristics.

Our qualitative analyses based on Grad-CAM++ reveal that the most relevant regions of the images for the network are regions predominated by vegetation and power lines. It is also worth noting that the relevant parts contain trees and power lines, according to the Grad-CAM++ heat map, even in images misclassified due to the lack of depth information or poor weather.

Chapter 6

Locating Urban Trees near Electric Wires using Google Street View Photos: A New Dataset and A Semi-Supervised Learning Approach in the Wild

This chapter is partially based on the paper of the same name [139].

In this chapter, we improve the robustness of our first solution for the classification of trees and wires entanglements by adding the Focal Loss (FL) [106] technique and comparing it with the vanilla Cross Entropy (CE). Our findings suggest that FL can naturally account for ambiguous images. Furthermore, we also study applying the semi-supervised Noisy Student training protocol [211] with thousands of unlabeled images, which improved the accuracy of the model up to 6%. We collected 48,800 urban ground-level images to build a dataset for training, testing, and validating our method, the first dataset of its kind. Our experiments show that our proposed method of training with FL and Noisy Student can improve the recall rates of the positive and negative classes respectively from 66.5% and 63.7% to 83.7% and 78.8%.

Our contributions are three-fold:

- A public dataset with 48k8 urban images, of which 8k8 were manually annotated for trees, electrical wires, and intersections.
- A classification approach to distinguish between ambiguous and easy cases based on Focal Loss and the Noisy Student training protocol.
- An experimental comparison between FL and the vanilla Cross-Entropy loss (CE) cost functions for this classification problem.

(a) (b)

Figure 6.1: *Images from Google Street View. (a) Trees are in contact with electrical wires. (b) Bad visibility due to sun glare.*

6.1 Method

For convenience here we briefly recap the Noisy Student method and Focal Loss, presented in Chapter 3, before presenting our approach to classify trees and wires intersection. The Noisy Student training protocol [211] is a semi-supervised approach created to leverage large amounts of not annotated data. A *teacher* neural network model is trained with labeled data. Then it generates *pseudo-labels* for unlabeled data, and the set of labeled and pseudo-labeled data is used to train a *student network* model.

The Focal Loss cost function is a generalization of the vanilla Cross-Entropy loss to deal with the class imbalance in object detection due to a significantly large number of easy negative cases in comparison with positive cases. Defined as:

$$FL(\theta) = -\sum_{(x,y)\sim D} y^{\gamma} \log(f_{\theta}(x)).$$

where θ are the parameters of a DLN f denoted as f_{θ}, and γ is the focusing parameter. By increasing the focusing parameter of FL, γ, samples correctly classified will have a smaller impact on the cost function while samples misclassified (e.g. harder instances) will have a larger impact.

We partially used the Noisy Student protocol, i.e., we trained a teacher model using only the labeled images, then we used the first-generation teacher to generate pseudo-labels for all the unlabeled images. Then we trained a noisy student model using both labeled and unlabeled images with pseudo-labels generated by the first-generation teacher. After that, we used the noisy student model to generate a new set of pseudo-labels, replacing the pseudo-labels generated by the previous generation teacher. Thus effectively, the noisy student model becomes a second-generation teacher model (Figure 2.3 depicts the training procedure with Noisy Student). We kept training new student models until accuracy in the validation dataset stopped increasing. Furthermore, we use Early Stopping as our convergence criteria while training each model. To avoid overfitting we use only the EfficientNetB0, the EfficientNet with the least number of parameters, across generations. Due to the size of our dataset, we did not use any form of model noise (e.g. stochastic depth [76]).

Algorithm 1 describes formally the procedure used to train a student. D_t and D_v

correspond to the training and validation datasets, respectively. X_t, X_v and X_u are sets of images for the training, validation, and unlabeled datasets. X_t and X_v have corresponding label sets Y_t and Y_v. Unlabeled images from X_u have pseudo-labels $\hat{Y}_u$ generated by a trained network ϕ_t, the teacher model. We use early stop with the patience of ten epochs (i.e. hyper-parameter *maxPatience* at Algorithm 2 and 1, lines four and five, respectively), that is, the model trains for other ten epochs after achieving the highest accuracy in the validation dataset. The first teacher network ϕ_t is trained using the same procedure as the student (see Algorithm 2), but using only the training and validation datasets D_t and D_v.

Algorithm 1: The student training procedure

Input: $D_t = \{X_t, Y_t\}, D_v = \{X_v, Y_v\}, X_u, \phi_t$
Output: ϕ_s
$\hat{Y}_u = \phi_t(X_u)$;
$D_u = \{X_u, \hat{Y}_u\}$;
$Patience = maxPatience$;
$D = D_t \bigcup D_u$;
Let $lastAcc$ be the accuracy of ϕ_s over D_v;
while $Patience > 0$ **do**
    Optimize ϕ_s using D;
    Let $newAcc$ be the accuracy of ϕ_s over D_v;
    if $lastAcc < newAcc$ **then**
        $lastAcc = newAcc$;
        $Patience = maxPatience$;
    else
        $Patience = Patience - 1$;
    end
end

6.2 Experiments

The proposed dataset has 48k8 images, each with a resolution of $640x640$ pixels, and we labeled 8k8 of them following a simple label protocol described in Chapter 2. We collected images (with INACITY) corresponding to four directions for each Panorama: the vehicle's forward, backward, left, and right. We split the 8k8 labeled images into training, validation, and test datasets, each with 5k, 3k, and 800 images, respectively.

Labeling strategy for challenging images Urban scenery images can be complex, and one of the difficulties is the depth of information lost in 2D images. In our particular case, some intersection instances may be ambiguous to determine if wires appear before or contact the branches of a tree.

Due to this complexity, different human annotators may judge the same picture as having distinct classification labels. To capture possible ambiguities, we propose a new label for challenging images, so an annotator may either consider an image as a *positive*

Figure 6.2: *Challenging samples may be confusing (to a human annotator) to determine if there are intersections between trees and wires or not. Images from GSV.*

Algorithm 2: The first teacher training procedure

Input: $D_t = \{X_t, Y_t\}, D_v = \{X_v, Y_v\}$
Output: ϕ_t
Initialize ϕ_t;
$Patience = maxPatience$;
$D = D_t$;
Let $lastAcc$ be the accuracy of ϕ_t over D_v;
while $Patience > 0$ **do**
    Optimize ϕ_t using D;
    Let $newAcc$ be the accuracy of ϕ_t over D_v;
    if $lastAcc < newAcc$ **then**
        $lastAcc = newAcc$;
        $Patience = maxPatience$;
    else
        $Patience = Patience - 1$;
    end
end

(an intersection is present), negative (no intersection), *challenging* (an intersection may be present, or not), or having no trees. Formally we define four possible labels for an image, these are:

- Trees with intersection (Trees w/ int.): Trees with an intersection- the images in this class have one or more trees, and the intersection between the branches and the wires is visible;
- Trees maybe with intersection (Trees maybe w/ int.): In this case, both the trees and the wires are visible, but it is challenging to tell if they are in contact or not;
- Trees without intersection (Trees w/o int.): Images in this class have trees but no visible wires;
- No trees: There are no visible trees in this class.

6.2.1 The neural network model

In our experiments, we used the network architecture MobileNetV3 [73] pretrained with the ImageNet dataset [97] both for teacher and student networks. We also experimented with all the family of EfficientNet networks (i.e. B0 to B7) as proposed in [211], both using the same architecture across all generations of teachers and student networks and also using for each new generation an architecture that has the same number or a bigger number of parameters. Surprisingly, the best results in terms of test accuracy were obtained by using the MobileNetV3 in all generations. Furthermore, we also experimented using random initialization for the network weights and observed that for every tried architecture using weights pretrained with the ImageNet dataset provided better results.

6.2.2 Overconfident uncalibrated teachers

We observed that each teacher model tends to be overconfident even on wrong predictions lowering the quality of the pseudo-labels generated. We estimate the confidence of the network over its predictions as the maximum value out of the vector obtained by computing the softmax of the output of the network. Unfortunately, this approach leads to uncalibrated networks that are overconfident in their predictions. We observed that a student network trained with uncalibrated low-quality pseudo-labels overfits the training data and the pseudo-labels. To mitigate this issue, we apply the temperature scaling strategy proposed in [64]. Using this strategy we want to find new confidence values for the predictions of the network such that samples correctly classified have a higher confidence value, and samples misclassified have a smaller confidence value.

For completeness, we briefly describe here the strategy to find the optimal temperature scale factor T to calibrate the predictions of a trained network. Let $x \in X_v$ be a sample from the validation dataset. In this section we consider the true label $y_x \in Y_v$ of sample x to be a one-hot encoded vector and the prediction $\hat{y}_x$ a probability vector, computed as the softmax $\sigma(.)$ of the output logits z_x of a trained network ϕ, that is

$$\hat{y}_x = \sigma(z_x) = \sigma(\phi(x))$$

Let the confidence of the network for this prediction be

$$q_x = \max_k \hat{y}_x^{(k)},$$

that is, the confidence is the maximum value in the predicted vector $\hat{y}_x$. Notice that (k), for $k \in K = [0, 1, 2, 3]$, in the exponent indicates a position in the prediction vector corresponding to the predicted class for sample x. Dividing z_x by a temperature scale factor T and then taking its softmax value produces a scaled prediction vector $\hat{y}'_x = \sigma(z_x/T)$. When $T \to \infty$ the values of the vector $\hat{y}'_x$ approach $\frac{1}{|K|}$, that is, every class will have nearly the same probability, thus the confidence of the network for this prediction is the nearly same for every class. In the opposite case, when $T \to 0$, $\hat{y}_x$ will approach a vector where every value is zero, except for $\hat{y}_x^{(k)}$ which will be one, that is, the confidence of the network for this prediction will be 1. Since the temperature scale factor is applied over the logits, before taking the softmax, the final prediction is kept unchanged, thus the temperature scaling doesn't change the accuracy of the network.

The optimal temperature scale T_{opt} to calibrate the predictions of the network as proposed by [64] is obtained by minimizing the Cross-Entropy loss between the scaled predictions vector y'_x and the corresponding true label y_x for sample x from the validation dataset, formally:

$$T_{opt} = \min_T \mathbb{E}[\sum_k^K y_x^{(k)} log(\hat{y}_i'^{(k)})]$$

$$= \min_T \mathbb{E}[\sum_k^K y_x^{(k)} log(\sigma(z_x/T)^{(k)})]$$

finally the new calibrated confidence q'_x is defined as:

$$q'_x = \max_k \sigma(z_x/T_{opt})^{(k)}$$

In our experiments, we compute T_{opt} only after the training of a network is done. Then we use this trained network as a teacher to compute pseudo-labels (using the unlabeled dataset) for a new student. These pseudo-labels are then calibrated with T_{opt} and only then they are used to train the next student network generation together with the labeled training dataset.

We performed experiments by training a sequence of teachers/student networks using either Cross-Entropy or Focal loss as cost functions. We experimented with different values for the hyperparameters γ and α of the FL (the weighting vector and the focusing parameter, respectively). We report the results for $\gamma = 2$ and $\alpha = [0.5, 0.1, 0.2, 0.2]$. Note that the weights vector α has an assigned weight for each of the classes described in Section 6.2.

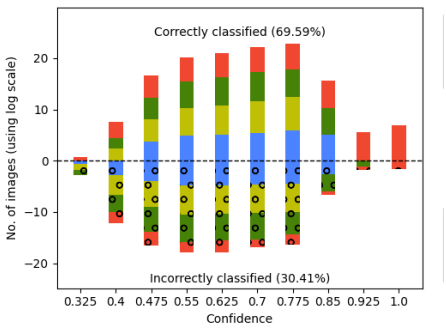
6.3 Results

Cost Function	Test Accuracy			
	1st gen.	2nd. gen.	3rd gen.	4th gen.
Focal Loss	49.37%	53.50%	55.37%	52.12%
Cross Entropy	60.87%	61.75%	60.12%	62.12%

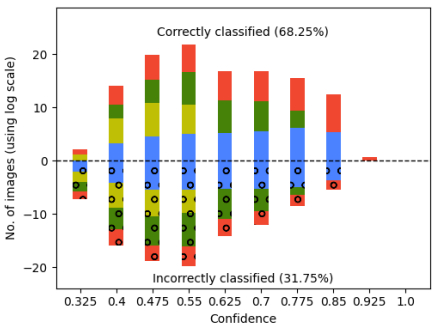
Table 6.1: *Comparison of the accuracy for each teacher-student generation and different cost functions.*

Here we analyze the trade-off between accuracy and confidence levels for FL and CE cost functions. The horizontal axis of our graphs displays bins with the confidence of predictions. Each bin represents samples predicted with confidence higher or equal to the value of the previous bin and strictly smaller than the current bin. In the vertical axis, we plot (in log scale) two mirrored stacked bars graph, one above and one below a dashed line. The stacked bars above the dashed line are correct predictions, and the ones below are incorrect predictions. Each one of the Figures 6.3a to 6.3g has two graphs, the one on the right has results obtained from networks trained with the Cross-Entropy cost function, and the graph on the left are results obtained by training the networks with the Focal Loss cost function.

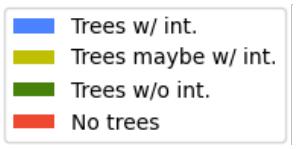
Figures 6.4a and 6.4b present the confusion matrices for the classification results of the networks trained with the Cross-Entropy and with the FL cost functions, respectively. These matrices contain the classifications for the test dataset, and the labels are (0) Trees with an Intersection, (1) Trees maybe with intersection, (2) Trees without intersection, and (3) No trees. Table 6.3 shows the recall rates, over the test dataset, for the network trained with the Cross-Entropy (CE) and Focal Loss (FL) cost functions. The last row in Table 6.3 (2+3) is the union of these classes, i.e. images without intersections independent of the presence of trees. We further compare the FL and CL trained networks in Figs. **??**, **??** and **??**, show their precision-recall curves for the Challenging, Positive and Negative classes, respectively.



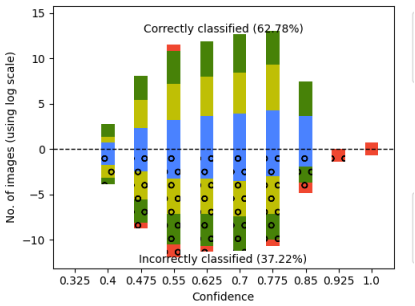
(a) *Results for the training partition with CE.*



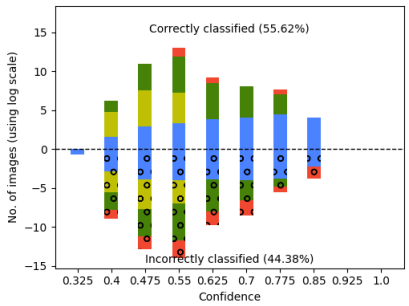
(b) *Results for the training partition with FL.*



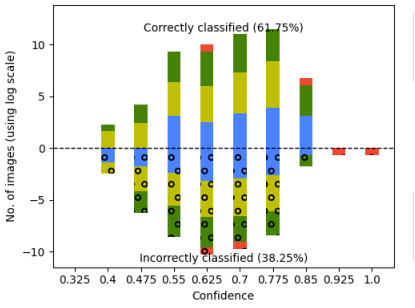
(c) *Legend for the colors in the graphs.*



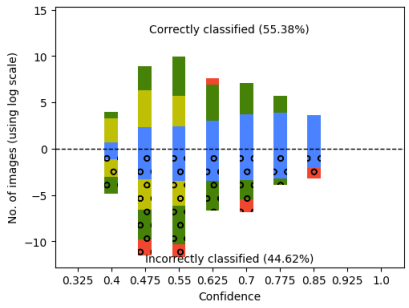
(d) *Results for the validation partition with CE.*



(e) *Results for the validation partition with FL.*



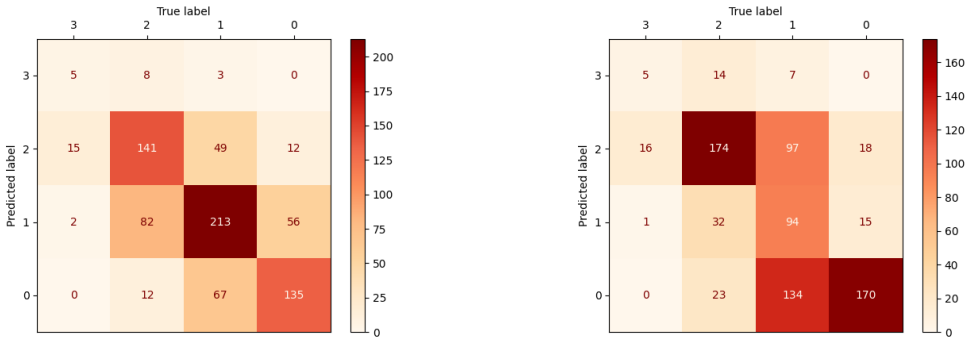
(f) *Results for the test partition with CE.*



(g) *Results for the test partition with FL.*

Figure 6.3: *Comparisons of networks trained with Cross-Entropy and Focal Loss at the training, validation, and test sets.*

-	Recall CE	Recall FL
(0)	66.5%	83.7%
(1)	64.1%	28.3%
(2)	58.0%	71.6%
(3)	22.7%	22.7%
(2+3)	63.7%	78.8%



(a) **(b)**

Figure 6.4: *(a) Confusion matrix for training with FL over the test dataset. (b) Confusion matrix for training with CE over the test dataset.*

6.4 Discussion

Figures 6.3a to 6.3g show that networks trained with CE classify images with all possible confidence levels, even challenging ones, labeled "Trees maybe with intersection", which are supposed to have lower classification confidence. On the other hand, training with FL provides an upper bound of 55% for challenging images. Furthermore, most misclassifications with FL also have a low confidence score, implying that these models are more calibrated. The precision-recall curve in Fig. **??** shows that the CL trained network (orange curve) has a performance equal or better than the FL trained network (blue curve), however, Figs. **??** and **??** show that the exact opposite is true for the Positive and Negative classes, which suggests that using the former to detect Challenging images, and the latter to classify non-Challenging images could potentially improve the overall accuracy.

For practical applications like detecting trees entangled with wires, the higher the recall rate for positive cases the better because a human agent can easily discard false positives. Networks trained with FL achieved a recall of 83.7% for positive cases, against 66.5% for those trained with CE. We conjecture that the high recall for the other classes obtained by vanilla CE was due to equal weights assigned to each class.

The problem of classifying intersections between trees-and-wires from 2D images is ill-posed due to the lack of depth information, thus in future works, we will explore Multi-View-Classification to combine the predictions for multiple viewpoints on challenging entanglements to improve the final prediction. Furthermore, another interesting research direction is the use of automatic active learning based on the detection of Challenging

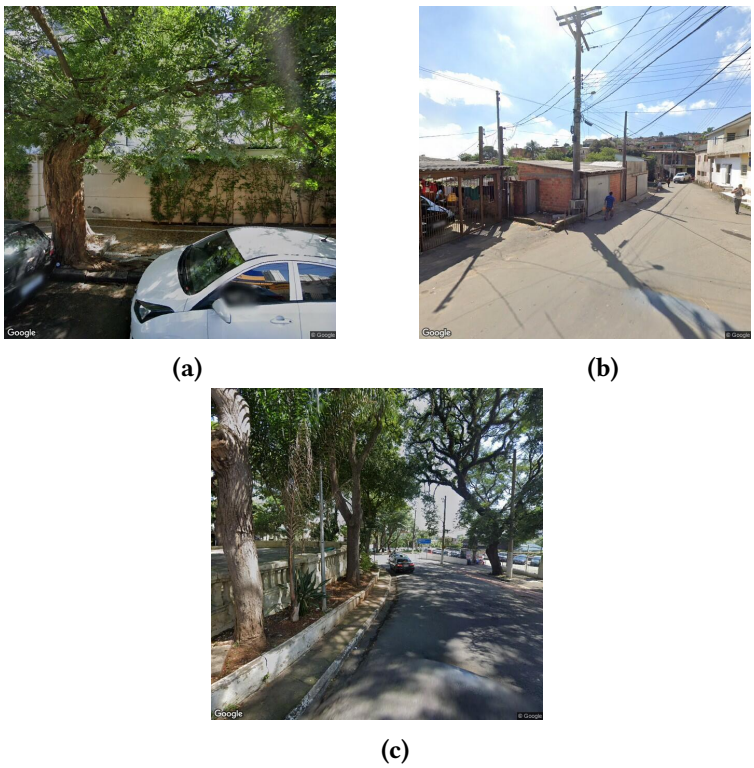


Figure 6.5: *Images from Google Street View. (a) Image misclassified by both networks as 'Trees w/o int'. The confidence for prediction dropped from 59% with CE to 49% with FL. (b) Image misclassified with CE as 'maybe with intersection' and correctly classified with FL as 'without intersection'. (c) Image misclassified with FL as 'without intersection' but with a low confidence of 40%.*

images, more specifically, when an image is classified as Challenging then other viewpoints are collected automatically and used together to in the final prediction.

Chapter 7

Conclusion

In this chapter, we discuss the works presented previously in the light of alternative approaches and show the advantages of our proposed methods, their limitations, and some possibilities for improving them.

We extended INACITY by integrating it with the Neo4j graph-based database. We proposed a graph-based data model to merge geographical features from multiple Geographical Information Systems (GIS) with Google Street View (GSV) imagery. The graph representing the relationship between images and geographical features can facilitate the creation of datasets. One example of use is acquiring images from multiple points of view of the same tree; we use the location of trees from the GeoSampa [175] GIS to collect the nearest Panoramas and images of trees, then we can collect alternative images for the same tree using the adjacent Panoramas in the graph, but from distinct points-of-view.

There are cases where the GIS locations may be incorrect because locations for trees were recorded in GeoSampa in 2010, while some of their corresponding images in Google Street View were taken (several) years later, and sometimes the trees are no longer there.

A second benefit of extending INACITY is its performance improvement and coverage of Geographic Imagery Databases (GID). We store a sequence of connected Panoramas in the graph database, which matches the order of images taken sequentially on a road. This is possible because the metadata for images from a GID like KartaView and Google Street View induces a spatial relationship for them, which usually is based on a sequence of images taken from a moving car along a road. Sampling GIDs using INACITY is more efficient than grid-search approaches (e.g. [193]) because INACITY will avoid checking for images in unusual locations like water bodies and inside buildings. Furthermore, after collecting image metadata, we use its references to nearby images to retrieve more images available in GSV. Thus INACITY is guaranteed to obtain every available image in a region and has better coverage than sampling methods based on points along a road registered in some GIS (e.g. [22]). Another limitation of our method arises when the sampled GIS data does not include the address of interest (e.g. OpenStreetMap [142]). Our approach cannot directly sample GSV panoramas for those addresses. However, if the missing address is connected to addresses in the sampled GIS data, INACITY can still collect relevant imagery

using metadata from GSV, which provides connected panoramas for connected addresses. A limitation intrinsic to using external Application Programming Interfaces (APIs) is due to the integration with GSV and, in general, with any external GID. GSV has an API that facilitates its integration with other softwares, but since the development of the INACITY platform in 2016, and even before considering the project from our group [104], the GSV API changed multiple times. Some of these updates are not compatible with previous versions, and GSV only allows the usage of the most updated version of its API, thus the integration of INACITY with GSV had to be fixed multiple times. To our knowledge, INACITY is the only platform with such data acquisition capabilities.

Next, to mitigate the image labeling burden, we developed the SLIL (Appendix A), which has two advantages over the existing labeling tools: it runs independently from a self-contained web server as labelMe [170], imglab [80], and VoTT [201]; and we adapted it to work with INACITY collected imagery, such that one can directly visit the panorama (i.e. geographical location) where the image was taken in GSV by clicking at the buttons named *Pano* or *Coordinates* (see Fig. 5.2 at Chapter 5 for a screenshot of SLIL).

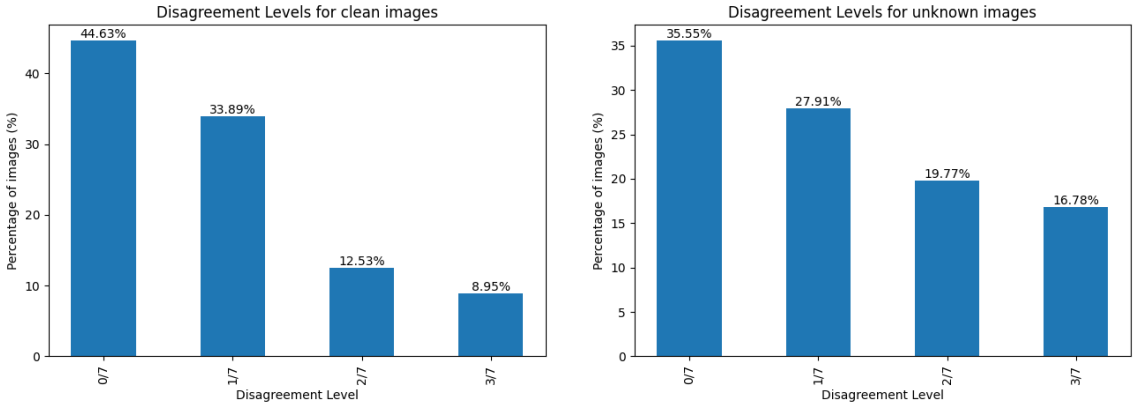
Regarding the practical problem of detecting trees near powerlines, which we call the entanglements detection problem, we show in Chapter 5 that it is a challenging problem due to the presence of training images that can worse the test performance of the models, and removing them improves the test performance. Furthermore, the appearance of trees varies a lot, and sometimes an image includes only a partial view of the tree due to its size. Different from other approaches based on aerial images and LiDAR data, we used only street-level imagery (created in Chapter 5 and extended in Chapter 6).

In Chapter 5, we trained a MobileNetV2 [174] for the entanglements detection problem, achieving an accuracy of 74.6%. We performed a qualitative analysis using Grad-CAM++ [177] to explain the results. The Grad-CAM++ shows that the relevant regions in some challenging incorrectly classified images had sun-glare, low contrast due to shadows, or even a point-of-view that made it hard to check if some distant powerlines were intersecting or ahead of the trees. Those are valuable insights because these images are difficult for humans, reinforcing our hypothesis that challenging images for humans are also challenging for Deep Learning Networks.

In Chapter 6, we try to automatically detect challenging images in the entanglements detection problem using the third label 'Unknown'. Instead of labeling a challenging image as positive or negative, we labeled it as 'Unknown'. We trained networks using the popular vanilla Cross-Entropy (CE) cost function and found that both test accuracy and recall rates for each class (i.e. positive, negative, and 'Unknown') are low. Based on the previous observations that most of the incorrect predictions occurred for challenging images we experimented with training the same networks with the Focal Loss (FL) cost function, designed to deal with unbalanced datasets with more easy samples and few harder ones [106] by exponentially weighting misclassified instances. The recall rates for the positive and negative classes of the networks trained with FL improved with a low decrease in the overall accuracy and recall rate for the 'Unknown' class. However, we observed that the confidence of the predictions of networks trained with FL for challenging images was at most 55%.

We have experimented recently different ways to detect challenging images and also to

assess their impact on the training dataset. We found that a set of binary classifiers disagree more on challenging images than on non-challenging ones, which is similar to previous evidence that internal classifiers, in a single deep learning model, tends to disagree on uncommon complex inputs [89]. Figures 7.1a and 7.1b show histograms of disagreements of eight DLN classifiers (with distinct architectures) respectively for challenging and non-challenging images from the Trees and Wires test set presented in Chapter 6.



(a) **(b)**

Figure 7.1: *Disagreement Level for images from the Trees and Wires test dataset. (a) Clean (positive and negative) images. (b) 'Unknown' images.*

We believe a heuristic based on the predicted probabilities and confidences can improve the performance. For instance, assigning the 'Unknown' label to samples with confidence below a certain threshold creates a trade-off between precision and recall. Alternatively, we can use the location of images predicted as challenging to collect more images for the same geographical feature from other nearby Panoramas and use a combination of the predictions for all the images as the final prediction. However, such an approach depends on the granularity of the GID used. We have observed cases where the challenging images and images from the nearest Panoramas were all challenging. It usually happens when the nearest Panoramas to an object of interest (e.g., a tree that seems entangled with overhead powerlines) are all too far from it and distant from each other.

We show in Appendix B that, in general, removing challenging images from our training dataset improves the generalization performance of the networks.

While we have made some progress in answering the initial research questions, more research is necessary. The dataset we have created is challenging, even for state-of-the-art deep learning networks and methods, and ambiguity in the images makes them less reliable. Our findings not only contribute to the field of computer vision but also have practical implications for urban planning and infrastructure management. With our proposed solutions, we can develop more accurate and efficient methods to detect and classify objects in urban environments.

We plan to combine the lessons learned with our experiments in an active learning paradigm. Once we have identified a challenging image, we will collect more images for the same feature from multiple nearby places, and use a combination of the predictions to make the final classification.

We will also work on other urban problems such as mapping removed or planted trees using images from the same feature taken at different moments, which are available in GIDs like Google Street View and KartaView; detecting people in vulnerable situations (e.g. laying down or sleeping in the sidewalk); detecting urban micro-events such as irregular waste disposal, street lamps lit during the daytime and so on. Using the graph database to relate all the detected urban features with their corresponding View and Panoramas nodes allows one to investigate aspects of the urban scene, such as how concentrated urban issues are in a given region and the relationships between different urban features.

Appendix A

Appendix: SLIL: Street-Level Image Labeler

(Semi-)Supervised deep learning methods for classification problems require both instances and their corresponding labels. Some simple ways to assign labels to instances include using part of the instance name as its label and grouping instances using distinct folders for each class. These methods can be cumbersome and tedious when the number of instances or classes is large, hard to maintain as new instances become available, and may be unfeasible when classes are non-exclusive or have non-binary labels like the "Unknown" label for the trees and wires problem discussed in Chapter 6. To make this process less time-consuming and error-prone, we propose the Street-Level Image Labeler (SLIL)[1], a tool to traverse, visualize and label images. Similar labeling tools: labelMe [170], imglab [80], and VoTT [201] run over a web server, which is more cumbersome and difficult to install and maintain for final users than a local application. We developed SLIL using pySimpleGUI [157], a Python framework for developing desktop Python applications with a Graphical User Interface (GUI). Users can define the available classes in SLIL to assign to an instance through a configuration file. Below we show an example of three user-defined classes (Tree, Pole w/ wire, and Intersection) in a configuration file:

```
    labels = 'Tree,Pole w/ wire,Intersection'
```

Figure A.1 shows the SLIL Graphical User Interface with the user-defined classes as a matrix of buttons in its top-right corner. Each row corresponds to a class, and the columns "-1", "0", and "1" correspond to a negative, neutral, or positive assignment for a class, respectively. We created the neutral assignment to account for the user uncertainty when labeling challenging instances.

The SLIL software is open-source and in constant development. Since its creation, motivated by the work presented in Chapter 5, we enhanced it with the option for neutral labels and quality-of-life features to allow users to quickly label an instance and move to the next one using only keyboard shortcuts without pressing GUI buttons with the mouse. Lucas and Hirata [215] used SLIL to create a dataset for Human Action Recognition in images with 15 classes denoting actions.

[1] available at https://github.com/arturandre/SLIL

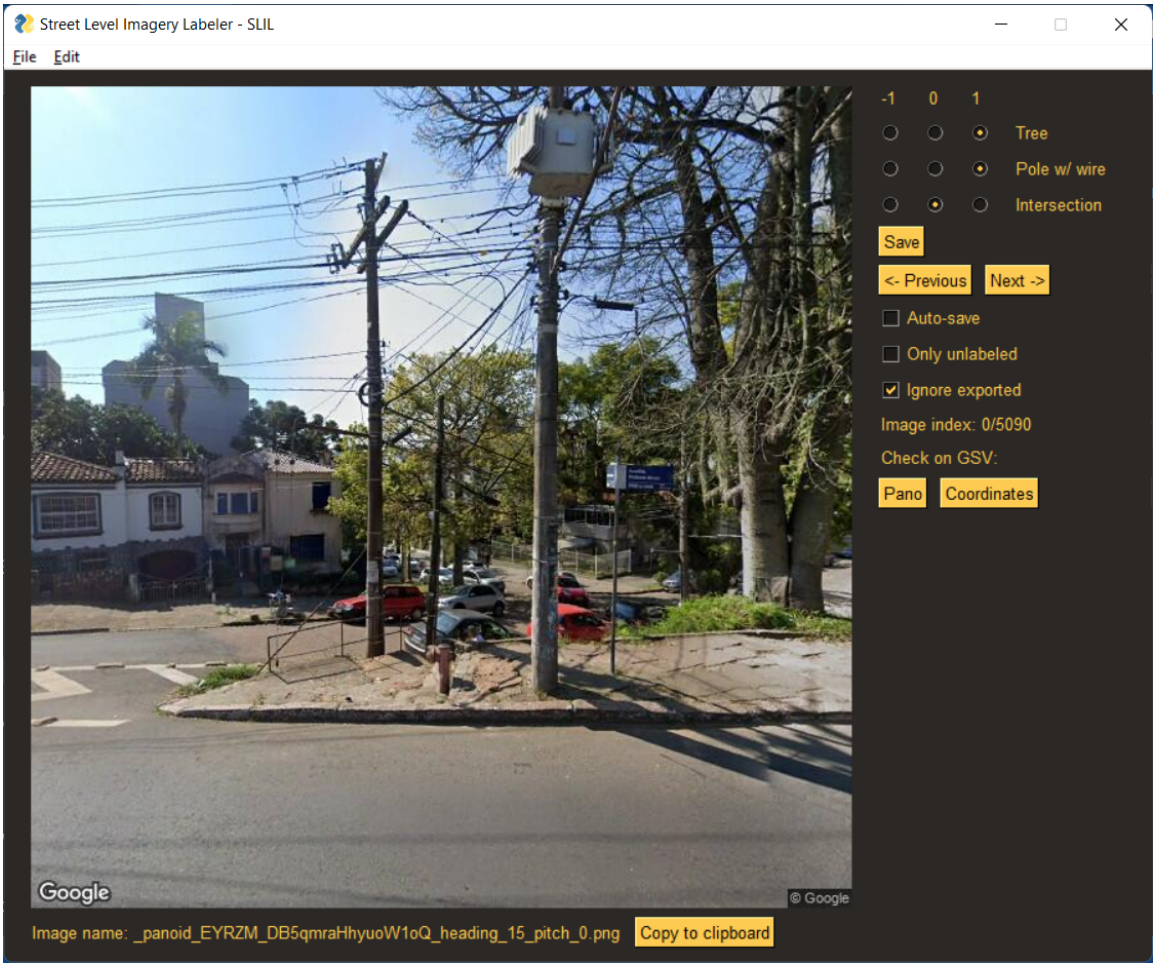


Figure A.1: *The current SLIL Graphical User Interface (compared with the previous version) includes a neutral assignment for defined classes (top-right) and allows the user to copy to the clipboard the filename of the current image.*

In the future, we will enhance SLIL with tools for drawing boxes around the objects of interest and features we believe to be useful for research of challenging urban images like how long it takes for a user to label each instance, zooming and a flag indicating if zooming was used, and considering sequential urban pictures, taken along a street, we will include a flag to represent when a user used two or more images of the same urban feature (e.g., tree) to make a decision.

Appendix B

Appendix: Assessing the effects of ambiguous images on training

This appendix is part of a paper to be submitted soon where we explore the hypothesis that challenging images behave as noise when used as training data, i.e., training a model with them reduces its performance on a test dataset. To test this hypothesis we first create a dataset with two parts, one containing ambiguous and another containing non-ambiguous images. A human labeler assigns a specific label for an ambiguous image when indecisive about its correct class. We train Deep Learning Networks (DLNs) using only non-ambiguous images or all available images, and then we compare their performances on a test dataset.

The Trees and Wires dataset contains a training set with 5760 images, 3361 of which are non-ambiguous and 2399 labeled as 'Unknown'. The test set includes 1440 images, but we only use non-ambiguous ones (i.e., 838) to report our results.

We used a variety of DLN architectures including VGG16 [182], ResNet50 and ResNet152 [65], DenseNet121 and DenseNet161 [77], and EfficientNetB0 [187]. All the networks were pretrained on the ImageNet [169] dataset. We removed their final classification head with 1000 nodes and inserted a new one with only one node to perform binary classification of images with trees entangled or not with powerlines. We fine-tuned the networks using the Adadelta optimizer [216] with an initial learning rate of 1.0, a step learning rate decay of 0.7, a batch size of 80, and 20 epochs. We selected the final model for each architecture based on the training epoch with the highest validation accuracy.

We explored three strategies for dealing with the 'Unknown' images removing, labeling them as positive instances, and labeling them as negative instances. The underlying rationale for these strategies stems from the notion that despite the uncertainty of the human labeler in an 'Unknown' image, one of two possibilities must hold. Thus, we simulate both scenarios by assigning positive and negative labels to these 'Unknown' images. Figure B.1 shows the test accuracy for three groups of models trained with each strategy.

Six of the eight tested architectures performed better with the negative label, and the other two had opposite results. Removing 'Unknown' images from the training dataset

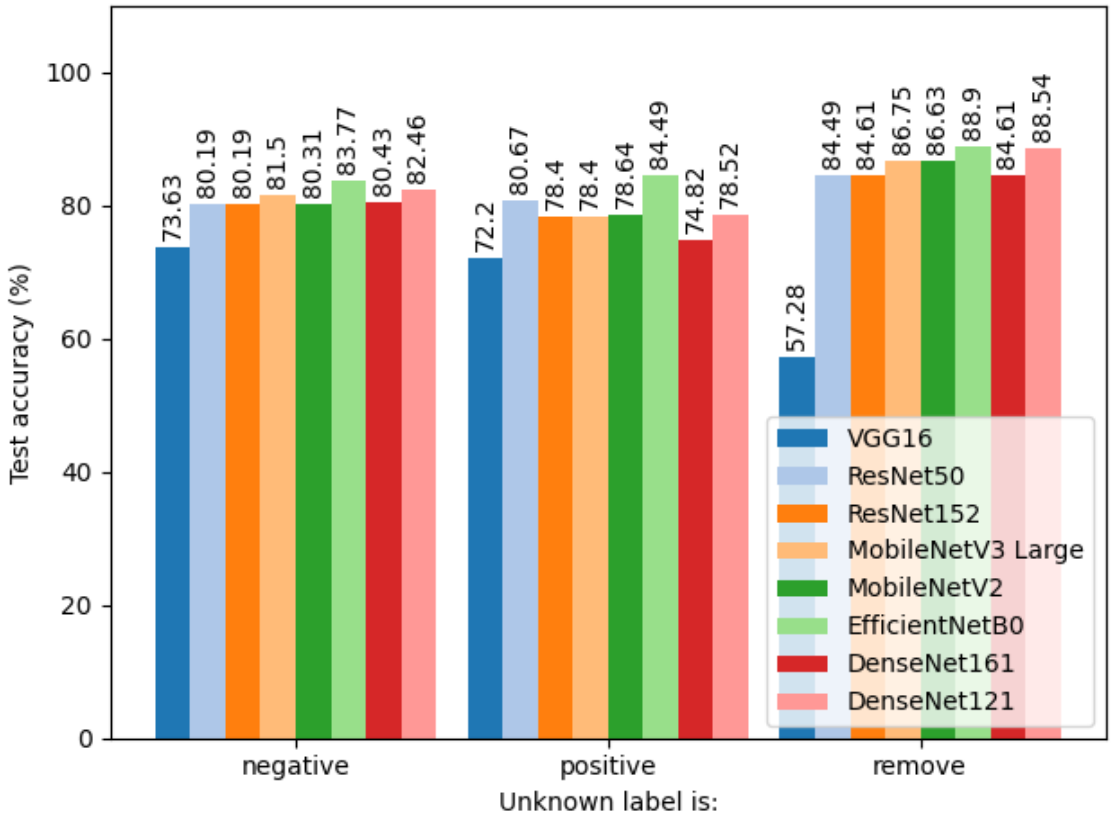


Figure B.1: *Test accuracy for networks trained without 'Unknown' images, or assigning positive/negative labels to them.*

yields a better test accuracy than labeling them as positive or negative, except for the VGG16, which we believe has worse results because its performance is significantly smaller than the other architectures.

In conclusion, our study investigates the impact of challenging images on the performance of Deep Learning Networks (DLNs) when used for training. We create a dataset comprising ambiguous and non-ambiguous images to test our hypothesis. The non-ambiguous images serve as the training data, while the ambiguous images are excluded or labeled as positive/negative. We compare the DLNs' performances on a separate test dataset.

Using various DLN architectures, including VGG16, ResNet50, ResNet152, DenseNet121, DenseNet161, and EfficientNetB0, pretrained on ImageNet, we fine-tune them for binary classification of images with or without trees entangled with powerlines. Among the strategies for dealing with ambiguous images, most architectures achieved better results with negative labels, while a couple had the opposite outcome. However, removing ambiguous images from the training dataset generally yielded the highest test accuracy, except for VGG16, which exhibited inferior performance compared to other architectures.

Appendix C

Appendix: Improving Self-supervised Dimensionality Reduction: Exploring Hyperparameters and Pseudo-Labeling Strategies

This chapter is partially based on the paper of the same name [140].

C.1 Introduction

Visualization of high-dimensional data to find patterns and trends, and overall understand the data structure has become an essential ingredient of the data scientist's toolkit [90, 109]. Within the palette of such visualization methods, dimensionality reduction (DR) techniques, also called projections, have gained an established position due to their high scalability both in the number of samples and number of dimensions thereof. In the last decades, tens of DR techniques have emerged [131, 50], with PCA [86], t-SNE [118], and UMAP [125] having become particularly popular.

Neural-network-based techniques have been used to support DR, early examples of such approaches being self-organizing maps [94] and autoencoders [70]. More recently, the NNP technique [47] was proposed to mimic any DR technique. In parallel, the ReNDA method [14] was proposed to improve the projection quality offered by autoencoders.

Deep learning-based DR methods are very fast, simple to implement, generically work for any type of quantitative high-dimensional data, are parametric, thus stable to small-scale data variations and offering out-of-sample capability, and – in the case of autoencoders – also provide the inverse mapping from the low-dimensional projection space to the high-dimensional data space. However, such methods also have some limitations. Such methods cannot typically offer the same projection quality, measured *e.g.* in terms of neighborhood preservation or cluster delineation, as classical methods like t-SNE and

UMAP [47, 126, 43]. Inverse projection typically requires training a separate network [49]. NNP-class methods offer a higher quality than autoencoders but require supervision in terms of using a classical DR method to project a subset of the data [47].

Recently, the Self-Supervised Neural Projection (SSNP [45]) method was proposed to alleviate the above limitations of deep learned projections. SSNP uses a single neural network trained with two objectives – *reconstructing* the projected data (as an autoencoder does) and *classifying* the same data (based on pseudo-labels created by a clustering algorithm). In more detail, SSNP aims to provide the following characteristics:

Quality (C1): Better cluster separation than standard autoencoders, and close to state-of-the-art DR methods, measured by well-known metrics in DR literature;

Scalability (C2): Linear complexity in the number of samples and dimensions, allowing the projection of datasets of a million samples and hundreds of dimensions in a few seconds on consumer-grade GPU platforms;

Ease of use (C3): Minimal or no hyperparameter tuning required;

Genericity (C4): Projects any dataset whose samples are real-valued vectors;

Stability and out-of-sample support (C5): The trained SSNP model can project new samples along existing ones in a parametric fashion;

Inverse mapping (C6): Ability to infer the high-dimensional point corresponding to a low-dimensional point in the projection space;

Clustering (C7): Ability to label (cluster) unseen data. This feature of SSNP also supports requirement C1: Intuitively, clustering aggregates low-level distance information between sample points to a higher level, telling how groups of samples relate to each other. Next, this information is used by SSNP to produce projections that preserve such data clusters well in the low dimensional space.

In [45], it is shown that SSNP achieves the above requirements by evaluating it on four synthetic and four real-world datasets, using two clustering algorithms to produce pseudo-labels and compare its results with four existing DR techniques. However, this leaves the 'design space' of SSNP insufficiently explored. Similarly to [43], where the authors explored in detail the design space of NNP [47], in this paper we aim to provide more insights on how SSNP's results depend on its technical components and their hyperparameter settings. For this, we extend the evaluation in [45] by considering two additional projection techniques (MDS and Isomap) and four additional clustering algorithms (affinity propagation, DBSCAN, Gaussian mixture models, and spectral clustering). Separately, we study how SSNP's performance is influenced by the setting of the hyperparameters of both the clustering algorithms and the underlying neural network. All in all, our extended evaluation proves that SSNP indeed complies well with requirements C1-C7, being a serious contender in the class of deep-learning-based DR techniques.

We structure this chapter as follows: Section E.2 introduces notations and discusses related work. Section C.3 details the SSNP method. Section C.4 describes our experimental setup. Section C.5 presents the results of SSNP, including the additional experiments

outlined above. Section C.6 discusses the obtained findings. Section C.7 concludes the paper.

C.2 Background

Notations: Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i \in \mathbb{R}, 1 \leq i \leq n$ be a n-dimensional (nD) sample (also called a data point or observation). Let $D = \{\mathbf{x}_i\}$, $1 \leq i \leq N$ be a dataset of N such samples, *e.g.*, a table with N rows (samples) and n columns (dimensions). All datasets D used in this paper have class labels. Let C be the number of classes (or labels) in a dataset D. A DR, or projection, technique is a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q, \tag{C.1}$$

where $q \ll n$, and typically $q = 2$. The projection $\mathbf{p} = P(\mathbf{x})$ of a sample $\mathbf{x} \in D$ is a point $\mathbf{p} \in \mathbb{R}^q$. Projecting an entire dataset D yields a q-dimensional scatterplot, denoted next as $P(D)$. The inverse of P, denoted $\mathbf{x} = P^{-1}(\mathbf{p})$, maps a q-dimensional point $\mathbf{p}$ to the high-dimensional space $\mathbb{R}^n$, so that, ideally, $P(\mathbf{x}) = \mathbf{p}$, or in practice, $P(\mathbf{x})$ is close to $\mathbf{p}$.

Dimensionality reduction: Many DR methods have been proposed in the last decades [72, 119, 42, 184, 109, 38, 210, 131, 50]. We next outline how a few representative ones comply with the requirements mentioned in Sec. C.1, supporting our point that no DR method fully covers all those requirements. For further evidence for this statement, we refer to the above-mentioned surveys.

Principal Component Analysis [86] (PCA) is very popular due to its simplicity, speed (C2), stability and out-of-sample (OOS) support (C5), and ease of use (C3) and interpretation. PCA is also used as a pre-processing step for other DR techniques that require not-too-high-dimensional data [131]. Yet, due to its linear and global nature, PCA lacks quality (C1), especially for data of high intrinsic dimensionality.

Methods of the Manifold Learning family (MDS [194], Isomap [188], and LLE [167] and its variations [40, 220, 219]) aim to map to 2D the high-dimensional manifold on which data lives. Such methods generally yield higher quality (C1) than PCA. Yet, such methods can be hard to tune (C3), do not have OOS capability (C5), do not work well for data that is not restricted to a 2D manifold, and generally scale poorly (C2) with dataset size.

Force-directed methods (LAMP [85] and LSP [150]) can yield reasonably high visual quality (C1), good scalability (C2), and are simple to use (C3). However, they generally cannot do OOS (C5). For LAMP, a related inverse projection (C6) technique iLAMP [7] exists. Yet, LAMP and iLAMP are two different algorithms. Clustering-based methods, such as PBC [148], share many characteristics of force-directed methods, such as good quality (C1) and lack of OOS (C5).

SNE (Stochastic Neighborhood Embedding) methods, of which t-SNE [118] is the most popular, have the key ability to visually segregate similar samples, thus being very good for cluster analysis. While having high visual quality (C1), t-SNE has a high complexity of $O(N^2)$ in sample count (C2), is very sensitive to small data changes (C5), is hard to

tune (C3) [204], and has no OOS capability (C5). Tree-accelerated t-SNE [115], hierarchical SNE [154], approximated t-SNE [153], and various GPU accelerations of t-SNE [155, 25] improve computation time (C2). Yet, these methods require quite complex algorithms, and still largely suffer from the aforementioned sensitivity, tuning, and OOS issues. Uniform Manifold Approximation and Projection (UMAP) [125] generates projections with comparable quality to t-SNE (C1) but is faster (C2) and has OOS (C5). Yet, UMAP shares some disadvantages with t-SNE, namely the sensitivity to small data changes (C5) and parameter tuning difficulty (C3).

Deep learning: Autoencoders (AE) [70, 92] create a low-dimensional data representation in their bottleneck layers by training a neural network to reproduce its high-dimensional inputs on its outputs. They produce results of comparable quality (C1) to PCA. However, they are easy to set up, train, and use (C3), are easily parallelizable (C2), and have OOS (C5) and inverse mapping (C6) abilities.

ReNDA [14] is a deep learning approach that uses two neural networks, improving on earlier work from the same authors. One network implements a nonlinear generalization of Fisher's Linear Discriminant Analysis [54]; the other network is an autoencoder used as a regularizer. ReNDA scores well on quality (C1) and has OOS (C5). However, it requires pretraining of each network and has low scalability (C2).

Neural Network Projections (NNP) [47] select a training subset $D_s \subset D$ to project by any user-chosen DR method to create a so-called training projection $P(D_s) \subset \mathbb{R}^2$. Next, a neural network is trained to approximate $P(D_s)$ having D_s as input. The trained network then projects unseen data using 2-dimensional non-linear regression. NNP is very fast (C2), simple to use (C3), stable, and has OOS ability (C5). However, the projection quality (C1) is lower than the learned projection. The NNInv technique [49], proposed by the same authors as NNP, adds inverse projection ability (C6). However, this requires setting up, training, and using a separate network.

Table C.1 summarizes how the above DR techniques fare concerning each characteristic of interest. The last row highlights SSNP which we describe separately in Sec. C.3.

	Characteristic						
Technique	**Quality**	**Scalability**	**Ease of use**	**Genericity**	**Out-of-sample**	**Inverse mapping**	**Clustering**
PCA	low	high	high	high	yes	yes	no
MDS	mid	low	low	low	no	no	no
Isomap	mid	low	low	low	no	no	no
LLE	mid	low	low	low	no	no	no
LAMP	mid	mid	mid	high	no	no	no
LSP	mid	mid	mid	high	no	no	no
t-SNE	high	low	low	high	no	no	no
UMAP	high	high	low	high	yes	no	no
Autoencoder	low	high	high	low	yes	yes	no
ReNDA	mid	low	low	mid	yes	no	no
NNP	high	high	high	high	yes	no	no
SSNP	**high**	**high**	**high**	**high**	**yes**	**yes**	**yes**

Table C.1: *Summary of DR techniques and their characteristics. Names in* italic *are techniques we compare with SSNP.*

Clustering: As for DR, clustering is a field that goes back decades, with many techniques proposed over the years. Despite using different approaches, all techniques use some form of similarity measure to determine whether a sample belongs to a cluster or not. *Centroid-based* techniques, such as K-means [111], compute cluster centers and assign cluster membership based on closeness to a center. *Connectivity-based* techniques, such as Agglomerative clustering [88], group samples based on their relative distances rather than distances to cluster centers. *Distribution-based* techniques, such as Gaussian Mixture Models [39], fit Gaussian distributions to the dataset and then assign samples to each distribution. *Density-based* techniques, such as DBSCAN [51], define clusters as dense areas in the data space. More recent techniques use more specialized approaches, such as Affinity Propagation [57], which uses message passing between samples, and Spectral Clustering [181], which uses the eigenvalues of the data similarity matrix to reduce the dimensionality of the data to be clustered.

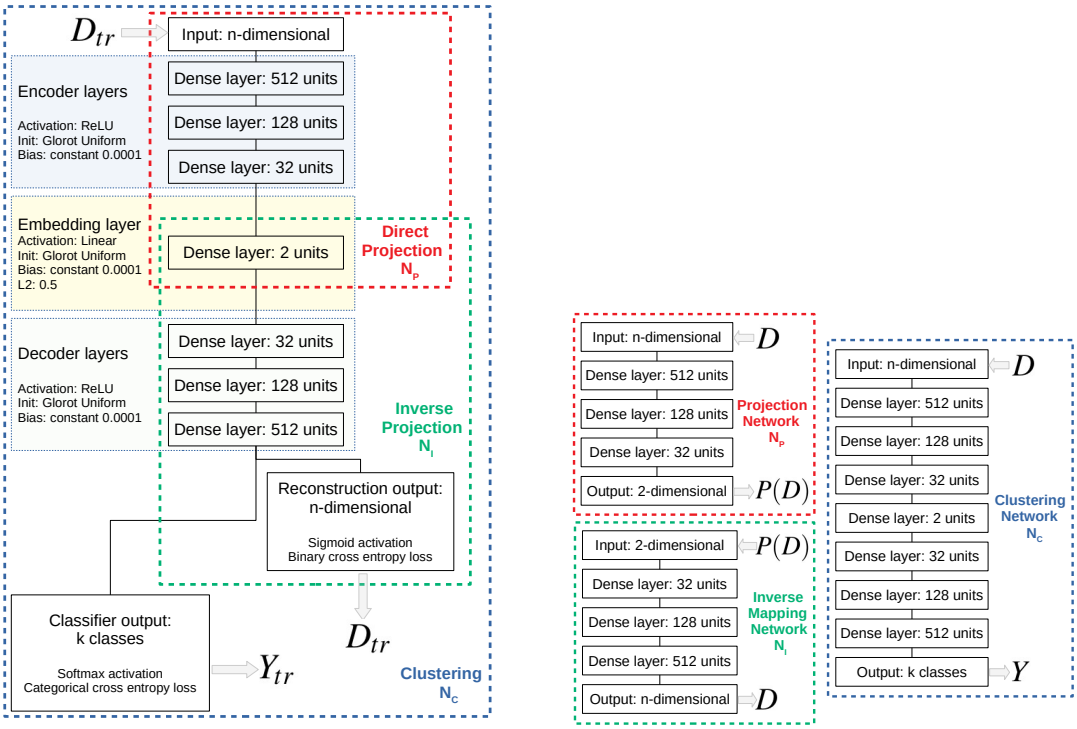
C.3 SSNP Technique

As stated in Sec. E.2, autoencoders have desirable DR properties (simplicity, speed, OOS, and inverse mapping abilities), but create projections of lower quality than, *e.g.*, t-SNE, and UMAP. A likely cause for this is that autoencoders do not use neighborhood information during training, while t-SNE and UMAP (obviously) do that. Hence, we propose to create an autoencoder architecture with a *dual* optimization target that explicitly uses neighborhood information. First, we have a *reconstruction* target, as in standard autoencoders; next, we use a *classification* target based on labels associated with the samples. These can be "true" ground-truth labels if available for a given dataset. If not, these are pseudo-labels created by running a clustering algorithm on the input dataset. The key idea behind this is that (pseudo)labels are a compact and high-level way to encode neighborhood information, *i.e.*, same-label data are more similar than different-label data. Since classifiers learn a representation that separates input data based on labels, adding an extra classifier target to an autoencoder learns how to project data with better cluster separation than standard autoencoders. We call our technique Self-Supervised Neural Projection (SSNP).

SSNP first takes a training set $D_{tr} \subset D$ and assigns to it pseudo-labels $Y_{tr} \in \mathbb{N}$ by using some clustering technique. We then take samples ($\mathbf{x} \in D_{tr}, y \in Y_{tr}$) to train a neural network with a reconstruction and a classification function, added to form a joint loss. This network (Fig. C.1a) contains a two-unit bottleneck layer like an autoencoder, used to generate the 2D projection when in inference mode. After training, we 'split' the layers of the network to create three new networks for inference (Fig. C.1b): a *projector* $N_P(\mathbf{x})$, an *inverse projector* $N_I(\mathbf{p})$, and a *classifier* $N_C(\mathbf{x})$, which mimics the clustering algorithm used to create Y_{tr}. The entire SSNP training-and-inference pipeline is summarized in Figure C.2.

C.4 Experimental Setup

In this section, we detail the experimental setup we used to evaluate SSNP's performance. The obtained results are discussed next in Sec. C.5.



(a) *Training* **(b)** *Inference*

Figure C.1: *SSNP network architectures used during training (a) and inference (b).*

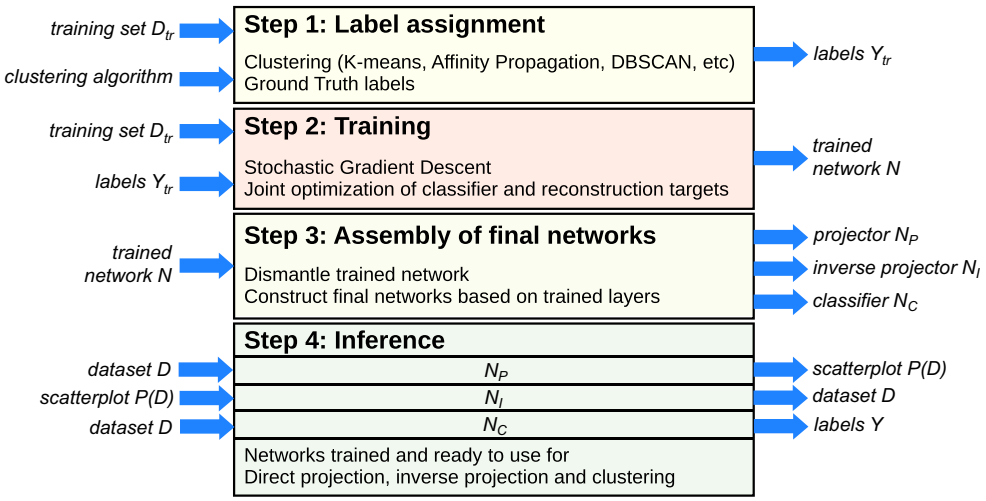


Figure C.2: *SSNP training-and-inference pipeline.*

C.4.1 Datasets

We first evaluate SSNP on synthetic datasets consisting of blobs sampled from a Gaussian distribution of different dimensionalities (100 and 700), number of clusters (5 and 10), and standard deviation σ, yielding datasets with cluster separation varying from very sharp to fuzzy clusters. All synthetic datasets have 5K samples. Next, we evaluate SSNP on four public real-world datasets that are high-dimensional, reasonably large (thousands of samples), and have a non-trivial data structure (same datasets as used in the original SSNP paper [45]):

MNIST [101]: 70K samples of handwritten digits from 0 to 9, rendered as 28x28-pixel grayscale images, flattened to 784-element vectors;

Fashion MNIST [209]: 70K samples of 10 types of pieces of clothing, rendered as 28x28-pixel grayscale images, flattened to 784-element vectors;

Human Activity Recognition (HAR) [9]: 10299 samples from 30 subjects performing activities of daily living used for human activity recognition grouped in 6 classes and described with 561 dimensions.

Reuters Newswire Dataset [191]: 8432 samples of news report documents, from which 5000 attributes are extracted using TF-IDF [173], a standard method in text processing.

All datasets had their attributes rescaled to the range $[0, 1]$, to conform with the sigmoid activation function used by the reconstruction layer (see Fig. C.1a).

C.4.2 Projection Quality Metrics

We measure projection quality by four metrics widely used in the projection literature (see Tab. C.2 for their definitions). All metrics range in $[0, 1]$ with 0 indicating the poorest, and 1 indicating the best, values:

Metric	Definition
Trustworthiness (T)	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in U_i^{(K)}} (r(i,j) - K)$
Continuity (C)	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in V_i^{(K)}} (\hat{r}(i,j) - K)$
Neighborhood hit (NH)	$\frac{1}{N} \sum_{\mathbf{y} \in P(D)} \frac{\mathbf{y}_k'}{\mathbf{y}_k}$
Shepard diagram correlation (R)	Spearman's ρ of $(\Vert\mathbf{x}_i - \mathbf{x}_j\Vert, \Vert P(\mathbf{x}_i) - P(\mathbf{x}_j)\Vert), 1 \leq i \leq N, i \neq j$

Table C.2: *Projection quality metrics used in evaluating SSNP*

Trustworthiness T [196] is the fraction of close points in D that are also close in $P(D)$. T tells how much one can trust that local patterns in a projection, *e.g.* clusters, represent actual data patterns. In the definition (Tab. C.2), $U_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in the 2D space but not among the K nearest neighbors of point i in $\mathbb{R}^n$; and $r(i, j)$ is the rank of the 2D point j in the ordered-set of nearest neighbors of i in 2D. We choose $K = 7$ following [119, 123];

Continuity C [196] is the fraction of close points in $P(D)$ that are also close in D. In the definition (Tab. C.2), $V_i^{(K)}$ is the set of points that are among the K nearest neighbors of

point i in $\mathbb{R}^n$ but not among the K nearest neighbors in 2D; and $\hat{r}(i, j)$ is the rank of the $\mathbb{R}^n$ point j in the ordered set of nearest neighbors of i in $\mathbb{R}^n$. As for T, we use $K = 7$;

Neighborhood hit NH [150] measures how well-separable labeled data is in a projection $P(D)$, in a rotation-invariant fashion, from perfect separation ($NH = 1$) to no separation ($NH = 0$). NH is the number $\mathbf{y}_K^l$ of the K nearest neighbors of a point $\mathbf{y} \in P(D)$, denoted by $\mathbf{y}_K$, that have the same label as $\mathbf{y}$, averaged over $P(D)$. In this paper, we use $K = 3$;

Shepard diagram correlation R [85]: The Shepard diagram is a scatter plot of the pairwise distances between all points in $P(D)$ *vs* the corresponding distances in D. The closer the plot is to the main diagonal, the better overall distance preservation is. Plot areas below, respectively above, the diagonal show distance *ranges* for which false neighbors, respectively missing neighbors, occur. We measure how close a Shepard diagram is to the diagonal by computing its Spearman rank correlation R. A value of $R = 1$ indicates a perfect (positive) correlation of distances.

C.4.3 Dimensionality Reduction Techniques Compared Against

We compared SSNP against six DR techniques, namely t-SNE, UMAP, MDS, Isomap, autoencoders (AE), and NNP (see also Tab. C.1). We selected these techniques based on popularity (t-SNE, UMAP, MDS, Isomap) or for having similar operations (AE and NNP are also deep learning based, like SSNP) and also on having desirable properties to compare against. For instance, t-SNE and UMAP are known to produce strong visual cluster separation by evaluating local neighborhoods. MDS, on the other hand, tries to preserve global distances between samples. Isomap can be seen as an extension of MDS that uses local neighborhood information to infer geodesic distances. AE produces results similar to PCA, which preserves global distances. Finally, NNP does not have specific built-in heuristics but rather aims to mimic and accelerate other DR techniques. For all these DR techniques, we used default values for their hyperparameters.

C.4.4 Clustering Techniques for Pseudo-labeling

In addition to using ground-truth labels in SSNP, we also used six clustering algorithms to generate the pseudo-labels for SSNP training (Sec. C.3). Table C.3 lists all clustering algorithms used, as well as the hyperparameters used in all experiments, except when noted otherwise. Hyperparameters not listed in Tab. C.3 used default values. We used these algorithms since they employ quite different approaches to clustering, which could produce different results for SSNP.

We selected two of these clustering algorithms alongside two datasets — K-means and DBSCAN, HAR, and MNIST — to further explore the effect of their main hyperparameters on the quality of the SSNP projection. For K-means, we studied the *n_clusters* parameter by choosing values well below and above the known number of clusters C in the data — $n\_clusters = \{5, 10, 15, 20, 30\}$ for MNIST ($C = 10$), $n\_clusters = \{3, 6, 9, 12, 18\}$ for HAR ($C = 6$). For DBSCAN, we explored the *eps* parameter, which determines the maximum distance between samples for them to be considered neighbors. We used $eps = \{6.1, 6.3, ..., 6.9\}$ for

MNIST and $eps = \{1.9, 2.1, ..., 2.7\}$ for HAR.

Algorithm	Acronym	Hyperparameters
Ground Truth Labels	SSNP(GT)	none
Affinity Propagation	SSNP(AP)	none
Agglomerative Clustering	SSNP(Agg)	$n\_clusters = 2 \times C$
DBSCAN	SSNP(DB)	$eps = 5$
Gaussian Mixture Model	SSNP(GMM)	$n\_components = 2 \times C$
K-means	SSNP(Km)	$n\_clusters = 2 \times C$
Spectral Clustering	SSNP(SC)	$n\_clusters = 2 \times C$

Table C.3: *Clustering algorithms used for pseudo-label creation and their hyperparameters used during testing. Ground truth is listed here as another labeling strategy.*

C.4.5 Neural Network Hyperparameter Settings

We further evaluated SSNP by using several hyperparameter settings for its neural network. To avoid a huge hyperparameter space, for each parameter explored, we kept the other parameters set to their defaults, similarly to the strategy used to explore NNP [43]. The explored hyperparameters are described next (see also Tab. C.4).

L2 regularization [98] decreases layer weights to small but non-null values, leading to every weight only slightly contributing to the model. It works by adding a penalization term $\lambda\|\mathbf{w}\|^2$ to the cost function, where $\mathbf{w}$ are the weights of a selected network layer. The parameter $\lambda \in [0, 1]$ controls the amount of regularization;

Embedding layer activation: The embedding (bottleneck) layer creates the 2D projection after training (Fig. C.1). Changing the activation function of this layer affects the projection's overall shape. We used four activation functions for this layer (see Tab. C.4);

Weight initialization: A neural network has thousands of parameters whose initialization can affect the training outcome. We used three common initialization types: random uniformly distributed in the range $[-0.05, 0.05]$, Glorot uniform [60] with the range $[-b, b]$ for $b = \sqrt{6/(l_{in} + l_{out})}$, where l_{in} and l_{out} are the number of input and output units in the layer, and He uniform [67], which uses the range $[-b, b]$ with $b = \sqrt{6/l_{in}}$;

Training epochs: We explored SSNP's performance for different numbers of epochs η ranging from 1 to 20.

Dimension	Values
L2 regularization	$\lambda = \{\mathbf{0}, 0.1, 0.5, 1.0\}$
Embedding layer activation	$\alpha = \{$ **ReLU**, sigmoid, tanh, Leaky RELU $\}$
Weight initialization	$\phi = \{$**Glorot uniform**, He uniform, Random uniform $\}$
Training Epochs	$\eta = \{1, 2, 3, 5, \mathbf{10}, 20\}$

Table C.4: *SSNP neural network parameters explored with default values in bold.*

C.5 Results

We next present the results for all experiments conducted to demonstrate SSNP's quality and robustness to hyperparameter selection.

C.5.1 Quality On Synthetic Datasets

Figure C.3 shows the SSNP projection of the synthetic blob datasets with SSNP(Km) with K-means set to use the correct (ground-truth) number of clusters alongside AE, t-SNE, and UMAP. In most cases SSNP(Km) shows better visual cluster separation than autoencoders. The t-SNE and UMAP projections look almost the same regardless of the standard deviation σ of the blobs, while SSNP(Km) shows more spread clusters for larger σ, which is the desired effect. We omit the plots and measurements for NNP for space reasons and since these are very close to the ones created by the learned technique [47].

Table C.5 shows the quality metrics for this experiment for datasets using 5 and 10 clusters. For all configurations, SSNP performs very similarly quality-wise to AE, t-SNE, and UMAP. Section C.5.2, which studies more challenging, real-world, datasets will bring more insight into this comparison.

		100 dimensions									700 dimensions							
		5 clusters				10 clusters					5 clusters				10 clusters			
Projection	σ	$T\uparrow$	$C\uparrow$	$R\uparrow$	$NH\uparrow$	$T\uparrow$	$C\uparrow$	$R\uparrow$	$NH\uparrow$	σ	$T\uparrow$	$C\uparrow$	$R\uparrow$	$NH\uparrow$	$T\uparrow$	$C\uparrow$	$R\uparrow$	$NH\uparrow$
AE	1.3	0.923	0.938	0.547	1.000	0.958	0.963	0.692	1.000	1.6	0.909	0.914	0.739	1.000	0.953	0.955	0.254	1.000
t-SNE		0.937	0.955	0.818	1.000	0.967	0.977	0.192	1.000		0.917	0.951	0.362	1.000	0.960	0.976	0.346	1.000
UMAP		0.921	0.949	0.868	1.000	0.957	0.970	0.721	1.000		0.906	0.933	0.878	1.000	0.954	0.965	0.471	1.000
SSNP(Km)		0.910	0.919	0.687	1.000	0.956	0.959	0.602	1.000		0.904	0.908	0.568	1.000	0.953	0.955	0.399	1.000
AE	3.9	0.919	0.926	0.750	1.000	0.959	0.963	0.484	1.000	4.8	0.910	0.914	0.615	1.000	0.953	0.954	0.354	1.000
t-SNE		0.931	0.953	0.707	1.000	0.966	0.978	0.227	1.000		0.914	0.950	0.608	1.000	0.960	0.977	0.331	1.000
UMAP		0.911	0.940	0.741	1.000	0.956	0.969	0.537	1.000		0.906	0.931	0.697	1.000	0.954	0.965	0.390	1.000
SSNP(Km)		0.910	0.918	0.622	1.000	0.955	0.958	0.549	1.000		0.905	0.907	0.612	1.000	0.953	0.954	0.296	1.000
AE	9.1	0.905	0.901	0.569	1.000	0.938	0.945	0.328	0.999	11.2	0.911	0.906	0.600	1.000	0.955	0.954	0.382	1.000
t-SNE		0.913	0.951	0.533	1.000	0.948	0.974	0.254	1.000		0.914	0.950	0.492	1.000	0.959	0.977	0.296	1.000
UMAP		0.888	0.939	0.535	1.000	0.929	0.966	0.342	1.000		0.905	0.931	0.557	1.000	0.953	0.965	0.336	1.000
SSNP(Km)		0.888	0.917	0.595	0.998	0.927	0.952	0.437	0.995		0.904	0.906	0.557	1.000	0.950	0.945	0.314	0.998

Table C.5: *Quality metrics, synthetic blobs experiment with 100 and 700 dimensions, 5 and 10 clusters, and* $\sigma \in [1.3, 11.2]$.

C.5.2 Quality On Real-World Datasets

Figure C.4 shows the projections of real-world datasets by SSNP with ground-truth labels (SSNP(GT)), SSNP with pseudo-labels created by the six clustering algorithms in Tab. C.3, and projections created by AE, t-SNE, UMAP, MDS, and Isomap. We omit again the results for NNP since they are very close to the ones created by t-SNE and UMAP. SSNP and AE were trained for 10 epochs in all cases. SSNP used twice the number of classes as the target number of clusters for the clustering algorithms used for pseudo-labeling.

SSNP with pseudo-labels shows better cluster separation than AE but slightly worse than SSNP(GT). For the more challenging HAR and Reuters datasets, SSNP(GT) looks better than t-SNE and UMAP. In almost all cases, SSNP yields a better visual cluster separation than MDS and Isomap. We see also that, for almost all clustering algorithm-dataset combinations, SSNP creates elongated clusters in a star-like pattern. We believe this is so since one of the network's targets is a *classifier* (Sec. C.3) which is trained to partition the space based on the data. This results in placing samples that are near a decision boundary between classes closer to the center of the star; samples that are far away from a decision boundary are placed near the tips of the star, according to their classes.

Table C.6 shows the four quality metrics (Sec. C.4.2) for this experiment. SSNP with pseudo-labels consistently shows better cluster separation (higher NH) than AE as well as

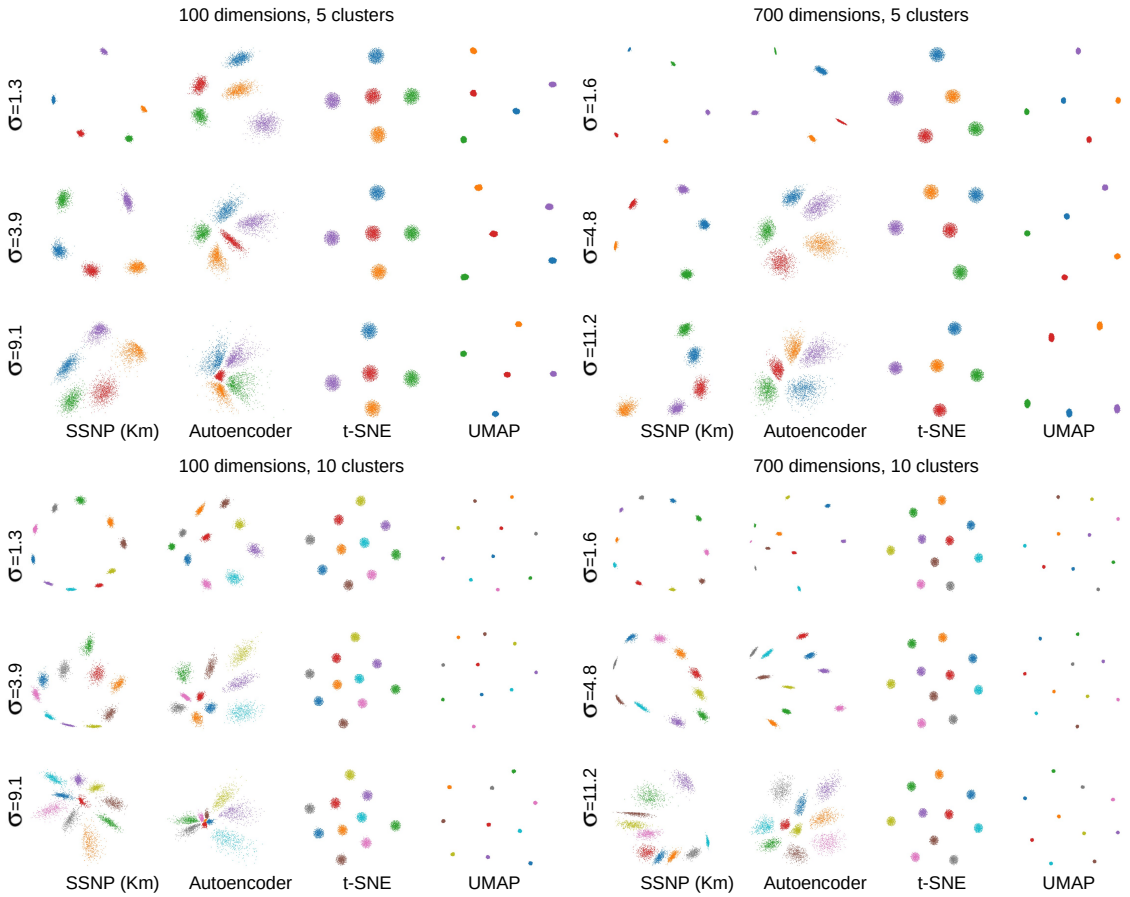


Figure C.3: *Projection of synthetic blobs datasets with SSNP(Km) and other techniques, with a different number of dimensions and clusters. In each quadrant, rows show datasets having increasing standard deviation σ.*

better distance preservation (higher R). For the harder HAR and Reuters datasets, SSNP(GT) shows NH results that are similar to and even higher than those for t-SNE and UMAP. Also, SSNP(GT) scores consistently higher than MDS and Isomap on all quality metrics, which correlates with these two projection techniques having been found as of moderate quality in earlier studies [50]. For the T and C metrics, SSNP(GT) outperforms again AE in most cases; for FashionMNIST and HAR, SSNP yields T and C values close to the ones for NNP, t-SNE, and UMAP. Separately, we see that the clustering algorithm choice influences the four quality metrics in several ways. DBSCAN (DB) yields in nearly all cases the lowest quality values while K-means (Km) and Agglomerative (AG) yield overall the best quality values. Spectral clustering (SC) is also a quite good option if one is mainly interested in cluster separation (high NH values). Finally, Affinity Propagation (AP) and Gaussian Mixture Models (GMM) score in between Km and AG (best overall) and DB (worst overall). From the above, we conclude that Km and AG are good default clustering methods that SSNP can use in practice.

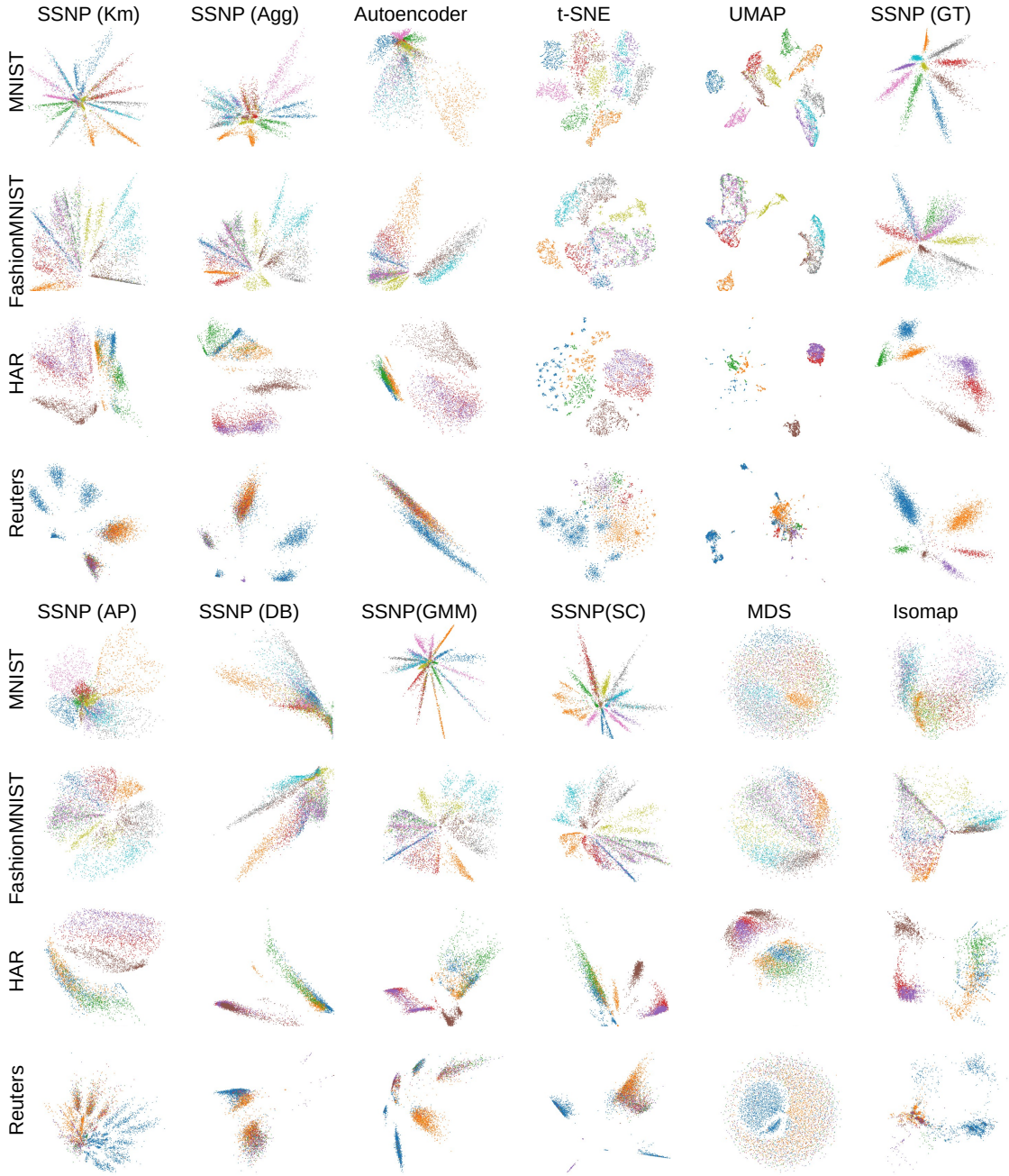


Figure C.4: *Projection of real-world datasets with SSNP (ground-truth labels and pseudo-labels computed by six clustering methods) compared to Autoencoders, t-SNE, UMAP, MDS, and Isomap.*

C.5.3 Quality *vs* Clustering Hyperparameters

Figure C.5 shows projections of the HAR and MNIST datasets created by SSNP with pseudo-labels assigned by DBSCAN and K-means and using the various clustering hyperparameter settings described in Sec. C.4.4.

For DBSCAN, we see that as the value of *eps* increases, the SSNP projection seems to vary between global- and local-distance preservation. This effect is more pronounced for the HAR dataset, where we see the number of clusters in the data varying from two

Dataset	Method	T	C	R	NH	Method	T	C	R	NH
MNIST	SSNP(Km)	0.882	0.903	0.264	0.767	SSNP(AP)	0.827	0.940	0.094	0.729
	SSNP(AG)	0.859	0.925	0.262	0.800	SSNP(DB)	0.689	0.802	0.032	0.588
	AE	0.887	0.920	0.009	0.726	SSNP(GMM)	0.880	0.895	0.257	0.755
	SSNP(GT)	0.774	0.920	0.398	0.986	SSNP(SC)	0.849	0.925	0.164	0.831
	NNP	0.948	0.969	0.397	0.891	MDS	0.754	0.862	0.618	0.580
	TSNE	0.985	0.972	0.412	0.944	Isomap	0.759	0.958	0.528	0.618
	UMAP	0.958	0.974	0.389	0.913					
FashionMNIST	SSNP(Km)	0.958	0.982	0.757	0.739	SSNP(AP)	0.947	0.986	0.750	0.728
	SSNP(AG)	0.950	0.978	0.707	0.753	SSNP(DB)	0.890	0.921	0.431	0.665
	AE	0.961	0.977	0.538	0.725	SSNP(GMM)	0.952	0.982	0.689	0.737
	SSNP(GT)	0.863	0.944	0.466	0.884	SSNP(SC)	0.957	0.981	0.706	0.756
	NNP	0.963	0.986	0.679	0.765	MDS	0.923	0.957	0.903	0.652
	TSNE	0.990	0.987	0.664	0.843	Isomap	0.920	0.976	0.749	0.685
	UMAP	0.982	0.988	0.633	0.805					
HAR	SSNP(Km)	0.932	0.969	0.761	0.811	SSNP(AP)	0.929	0.972	0.736	0.787
	SSNP(AG)	0.926	0.964	0.724	0.846	SSNP(DB)	0.852	0.909	0.759	0.690
	AE	0.937	0.970	0.805	0.786	SSNP(GMM)	0.924	0.966	0.768	0.796
	SSNP(GT)	0.876	0.946	0.746	0.985	SSNP(SC)	0.893	0.952	0.811	0.805
	NNP	0.961	0.984	0.592	0.903	MDS	0.911	0.890	0.941	0.765
	TSNE	0.992	0.985	0.578	0.969	Isomap	0.925	0.971	0.896	0.861
	UMAP	0.980	0.989	0.737	0.933					
Reuters	SSNP(Km)	0.794	0.859	0.605	0.738	SSNP(AP)	0.631	0.768	0.039	0.742
	SSNP(AG)	0.771	0.824	0.507	0.736	SSNP(DB)	0.574	0.650	0.360	0.705
	AE	0.747	0.731	0.420	0.685	SSNP(GMM)	0.622	0.788	0.460	0.793
	SSNP(GT)	0.720	0.810	0.426	0.977	SSNP(SC)	0.607	0.758	0.027	0.730
	NNP	0.904	0.957	0.594	0.860	MDS	0.575	0.757	0.551	0.699
	TSNE	0.955	0.959	0.588	0.887	Isomap	0.634	0.785	0.150	0.765
	UMAP	0.930	0.963	0.674	0.884					

Table C.6: *Quality measurements for the real-world datasets (Sec. C.5.2).*

Dataset	Technique	Parameter	T	C	R	NH
MNIST	DBSCAN	eps=6.1	0.685	0.821	0.097	0.555
		eps=6.3	0.679	0.798	0.012	0.570
		eps=6.5	0.722	0.812	0.044	0.614
		eps=6.7	0.698	0.801	0.022	0.576
		eps=6.9	0.729	0.825	0.011	0.605
	K-means	n_clusters=5	0.782	0.905	0.408	0.641
		n_clusters=10	0.834	0.916	0.379	0.697
		n_clusters=15	0.867	0.927	0.410	0.760
		n_clusters=20	0.880	0.909	0.047	0.755
		n_clusters=30	0.899	0.932	0.358	0.790
HAR	DBSCAN	eps=1.9	0.854	0.928	0.917	0.696
		eps=2.1	0.848	0.920	0.841	0.650
		eps=2.3	0.875	0.914	0.717	0.685
		eps=2.5	0.896	0.924	0.844	0.725
		eps=2.7	0.898	0.933	0.887	0.749
	K-means	n_clusters=3	0.887	0.939	0.932	0.693
		n_clusters=6	0.921	0.959	0.749	0.767
		n_clusters=9	0.920	0.965	0.877	0.812
		n_clusters=12	0.930	0.968	0.854	0.815
		n_clusters=18	0.937	0.972	0.840	0.812

Table C.7: *Quality measurements for the cluster hyperparameter experiment (Sec . C.5.3).*

($eps = 1.9$) and three ($eps = 2.7$). For the MNIST dataset, the increase in eps only makes the entire projection take a sharper shape, with no improvement in cluster separation. Overall, SSNP with DBSCAN having low eps values produces results similar to an autoencoder, which defeats the purpose of using SSNP. This correlates to the earlier findings in Sec. C.5.2 that showed that DBSCAN is not a good clustering companion for SSNP. The quality metrics in Tab. C.7 strengthen this hypothesis – we do not see any clear trend of these metrics being improved by varying eps in a specific direction.

For K-means, we see that the value of *n_clusters* has a great effect on the overall shape

of the SSNP projection. Particularly, when *n_clusters* is higher than the true number of classes in the data (10 for MNIST, 6 for HAR), we see that the cluster separation gets sharper. This suggests that, when the true number of clusters is not known, starting with a reasonably high number of clusters will produce better results for SSNP with K-means. This is confirmed by the quality metrics in Tab. C.7 which show higher values for higher *n_clusters* settings.

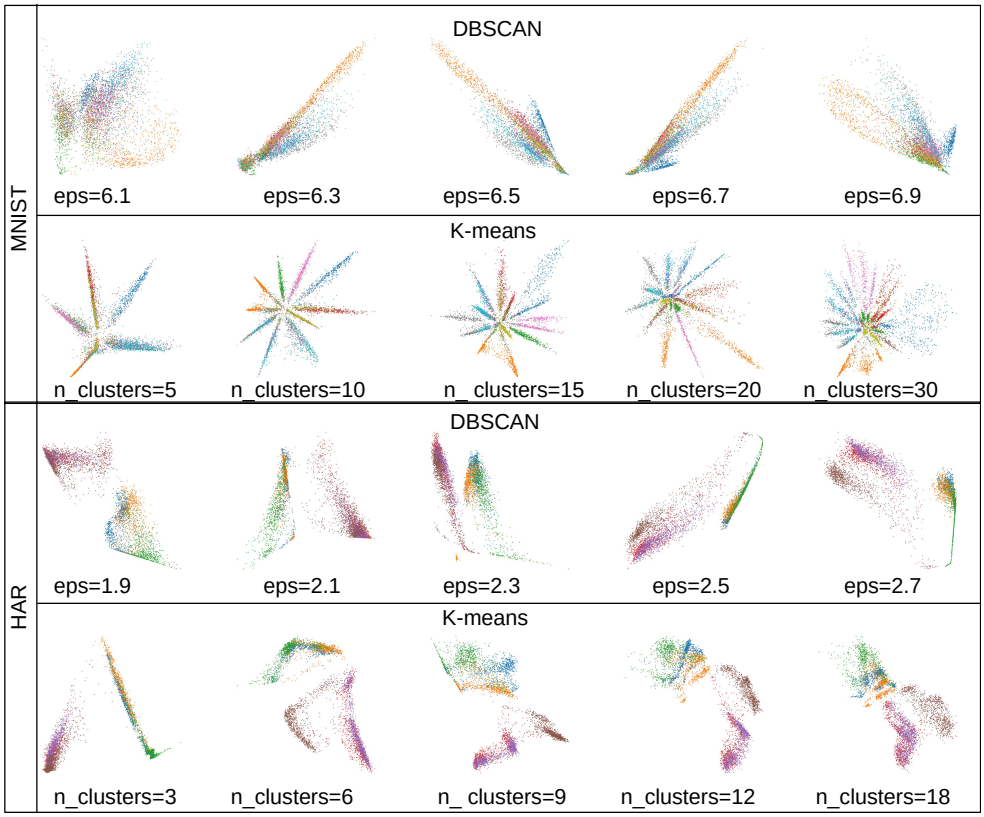


Figure C.5: *Projections of MNIST and HAR datasets using different hyperparameters for the DBSCAN and K-means clustering methods (see Sec. C.5.3 and Tab. C.7).*

C.5.4 Quality *vs* Neural Network Settings

We next show how the different neural network hyperparameter settings affect the SSNP results following the sampling of these parameters discussed in Sec. C.4.5. We also use this analysis to derive good default values for these parameters.

L2 regularization: Figure C.6 shows projections created with different amounts of L2 regularization during SSNP's training. We see that regularization has a detrimental effect on the visual quality of the projection. For values of $\lambda \geq 0.5$, the projection points collapse to a single point, marked by the red circles in the figure. Table C.8 shows the metric values for this experiment confirming that all quality values decrease with λ. We conclude that SSNP obtains optimal results without regularization.

Activation functions: Figure C.7 shows the effect of using different activation functions α in the embedding layer. We see that the ReLU and LeakyReLU activations produce similarly good results. Both produce visual cluster separation comparable to t-SNE and

Method	Parameter	Value	T	C	R	NH
SSNP(GT)	α	LeakyReLU	0.780	0.930	0.429	0.971
		ReLU	0.789	0.921	0.402	0.983
		sigmoid	0.703	0.891	0.088	0.746
		tanh	0.784	0.929	0.190	0.983
	η	2	0.781	0.924	0.428	0.903
		3	0.787	0.926	0.428	0.940
		5	0.786	0.925	0.419	0.966
		10	0.789	0.921	0.402	0.983
		20	0.797	0.920	0.391	0.989
	ϕ	Glorot	0.789	0.921	0.402	0.983
		He	0.789	0.928	0.328	0.982
		Random	0.758	0.905	0.071	0.927
	λ	0	0.789	0.921	0.402	0.983
		0.1	0.757	0.909	0.360	0.870
		0.5	0.538	0.502	NA	0.101
		1	0.538	0.502	NA	0.101
SSNP(Km)	α	LeakyReLU	0.863	0.919	0.177	0.748
		ReLU	0.888	0.916	0.119	0.768
		sigmoid	0.678	0.872	0.196	0.568
		tanh	0.884	0.928	0.265	0.774
	η	2	0.847	0.927	0.267	0.726
		3	0.827	0.926	0.244	0.714
		5	0.854	0.915	0.323	0.775
		10	0.881	0.908	0.188	0.770
		20	0.886	0.911	0.128	0.766
	ϕ	Glorot	0.884	0.915	0.333	0.784
		He	0.874	0.903	0.267	0.753
		Random	0.741	0.869	0.115	0.640
	λ	0	0.888	0.924	0.351	0.763
		0.1	0.872	0.910	0.352	0.753
		0.5	0.538	0.502	NA	0.101
		1	0.538	0.502	NA	0.101

Table C.8: *Quality measurements for SSNP for different training hyperparameters. NA indicates that the measurement failed for the respective experiment (Sec. C.5.4).*

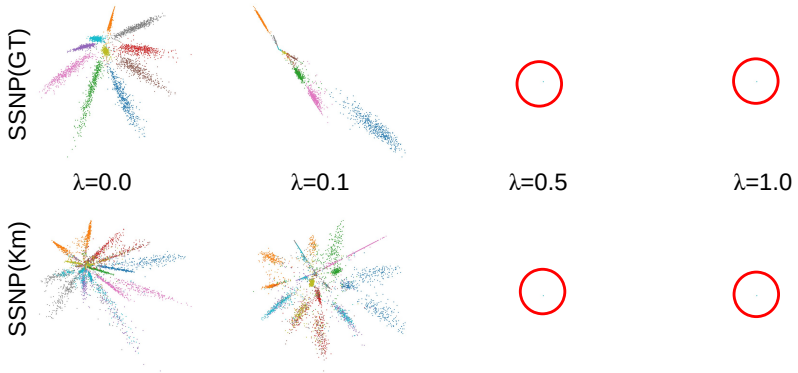


Figure C.6: *Projections created with SSNP(GT) and SSNP(Km) for the MNIST dataset varying the amount of L2 regularization λ (Sec. C.5.4).*

UMAP (see Fig. C.4), albeit with a distinct star or radial shape. The sigmoid activation collapses all data points into a single diagonal, making it a poor choice for the embedding layer. Finally, the tanh activation produced the best cluster separation of all, with results that look very close to the ones by t-SNE and UMAP for this dataset (see again Fig. C.4). We conclude that the tanh activation function is the best option for SSNP.

Initialization: Figure C.8 shows how weight initialization affects projection quality. We see that both Glorot and He uniform initializations produce good and comparable results,

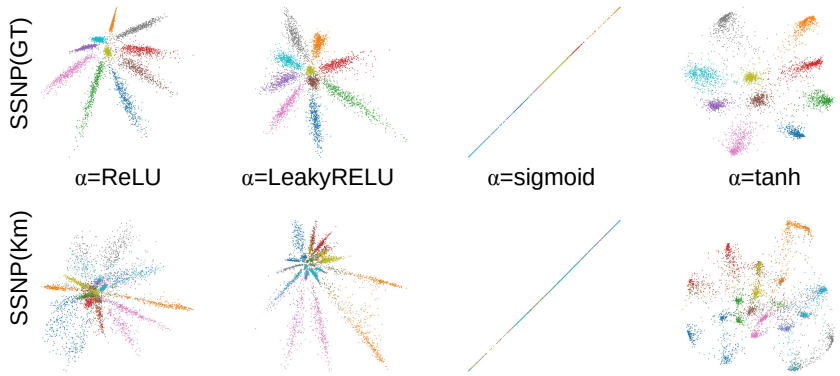


Figure C.7: *Projections of the MNIST dataset using SSNP(GT) and SSNP(Km) varying the activation function α (Sec. C.5.4).*

whereas random initialization yields very poor results. We opt for using He uniform as the default initialization, which correlates with the same choice (obtained by an independent investigation) for NNP [43].

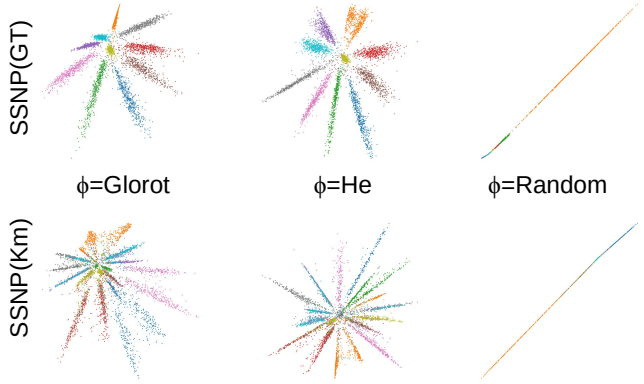


Figure C.8: *Projections of the MNIST dataset using SSNP(GT) and SSNP(Km) varying the weight initialization strategy ϕ (Sec. C.5.4).*

Training epochs: Finally, Figure C.9 shows projections created with SSNP trained for different numbers η of epochs. With as little as $\eta = 3$ training epochs, SSNP already produces good cluster separation. As η increases, the created visual clusters become sharper. However, there seems to be little improvement when going from $\eta = 10$ to $\eta = 20$. As such, we conclude that a good default is $\eta = 10$ training epochs. Interestingly, this is significantly less than the 50 epochs needed by NNP to achieve good projection quality [43], especially if we consider that SSNP has to train a more complex, dual-objective, network.

C.5.5 Computational Scalability

Using SSNP means (a) training the network and next (b) using the trained network in inference mode (see also Fig. C.1). We analyze these two times next.

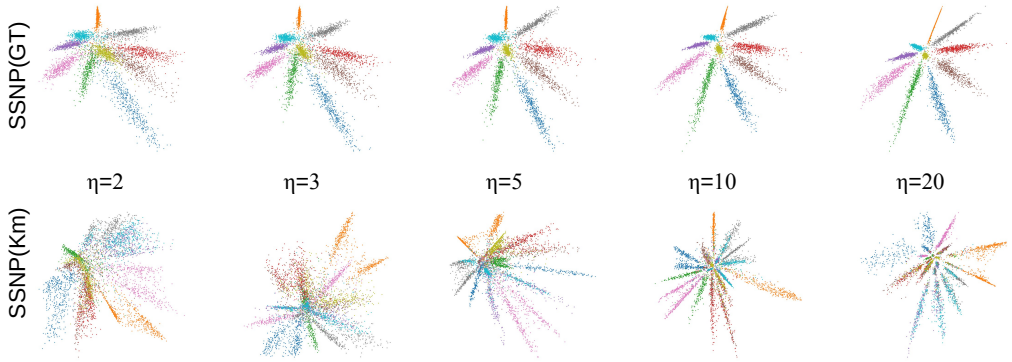


Figure C.9: *Projections of MNIST dataset using SSNP(GT) and SSNP(Km) varying the number of training epochs η (Sec. C.5.4).*

Setup time: Table C.9 shows the time needed to set up SSNP and three other projection techniques. For SSNP, NP, and AE, this is the training time of the respective neural networks using ten training epochs. Note that we used 10K training samples, but in practice, SSNP obtains good results (quality-wise) with as few as 1K samples. For UMAP and t-SNE, this is the time needed to project the data since these techniques do not have a training phase. We see that the SSNP variants using clustering take about the same time as t-SNE and UMAP and less time than NNP. SSNP(GT), which does not need clustering, is far faster than these competitors, except for AE, which is about twice faster. This is explainable since SSNP uses a dual-objective network (Sec. C.3), one of these being essentially the same as AE.

Method	Setup time (s)
SSNP(GT)	6.029
SSNP(Km)	20.478
SSNP(Agg)	31.954
AE	3.734
UMAP	25.143
t-SNE	33.620
NNP(t-SNE)	51.181

Table C.9: *Setup time for different projection methods for 10K training samples, MNIST dataset.*

Inference time: Figure C.10 shows the time needed to project up to 1M samples using SSNP and the other compared projection techniques. For SSNP, AE, and NNP, this is the inference time using the respective trained networks. For t-SNE and UMAP, this is the actual projection time, as described earlier in this section. Being GPU-accelerated neural networks, SSNP, AE, and NNP perform very fast, all being able to project up to 1M samples in a few seconds – an order of magnitude faster than UMAP, and over three orders of magnitude faster than t-SNE. We also see that SSNP, AE, and NNP have practically the same speed. This is expected since they have comparably large and similar-architecture neural networks which, after training, take the same time to execute their inference.

C.5.6 Inverse Projection

Recalling from Sec. E.2, an *inverse* projection $P^{-1}(\mathbf{p})$ aims to create a data point $\mathbf{x}$ so that its projection $P(\mathbf{x})$ is as close as possible to $\mathbf{p}$. Hence, we can test how well a method

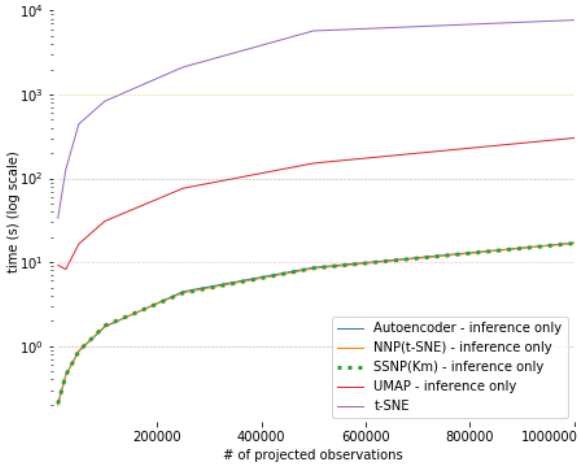


Figure C.10: *Inference time for SSNP and other techniques (log scale). Techniques using training use 10K samples from the MNIST dataset. Inference is done on MNIST upsampled up to 1M samples.*

computes P^{-1} for a given direct projection function P by evaluating how close $P^{-1}(P(\mathbf{x}))$ is to the data point $\mathbf{x}$ itself. To test this, we consider points $\mathbf{x}$ being images in the MNIST dataset and P and P^{-1} being computed by SSNP as described in Sec. C.3).

Figure C.11 shows a set of digits from the MNIST dataset – both the actual images $\mathbf{x}$ and the ones obtained by $P^{-1}(P(\mathbf{x}))$. We see that SSNP(Km) yields results very similar to AE, both of these being visually quite close images to the actual images $\mathbf{x}$, modulo a limited amount of fuzziness. Hence, SSNP's dual-optimization target succeeds in learning a good inverse mapping based on the direct mapping given by the pseudo-labels (Sec. C.3). Table C.10 strengthens this insight by showing the values of the Mean Squared Error (MSE) between the original and inversely-projected images $\frac{1}{|D|}\sum_{\mathbf{x}\in D}\|\mathbf{x}-P^{-1}(P(\mathbf{x}))\|^2$ for SSNP(Km) and AE for both the training and test sets. These errors, again, are very similar. Furthermore, the SSNP MSE errors are of the same order of magnitude – that is, very small – as those obtained by the recent NNInv technique [49] and the older iLAMP [7] technique that also computes inverse projections – compare Tab. C.10 with Fig. 2 in [49] (not included here for space reasons). Summarizing the above, we conclude that SSNP achieves a quality of inverse projections on par with existing state-of-the-art techniques.

	SSNP(Km)		Autoencoder	
Dataset	Train	Test	Train	Test
MNIST	0.0474	0.0480	0.0424	0.0440
FashionMNIST	0.0309	0.0326	0.0291	0.0305
HAR	0.0072	0.0074	0.0066	0.0067
Reuters	0.0002	0.0002	0.0002	0.0002

Table C.10: *Inverse projection Mean Square Error (MSE) for SSNP(Km) and AE, trained with 5K samples and tested with 1K samples, different datasets.*

C.5.7 Data clustering

Table C.11 shows how SSNP performs when doing classification or clustering, which corresponds respectively to its usage of pseudo-labels or ground-truth labels. We see that SSNP generates good results in both cases when compared to the ground-truth (GT) labels

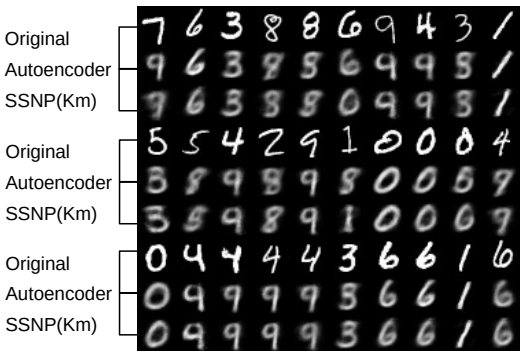


Figure C.11: *Sample images from MNIST inversely projected by SSNP and AE, both trained with 10 epochs and 5K samples, MNIST dataset. Bright images show the original images that the inverse projection should be able to reproduce.*

and, respectively, the underlying clustering algorithm K-means (Km), which emerged as one of the best clustering companions for SSNP (Sec. C.5.2). However, we should stress that classification or clustering is only a *side* result of SSNP, needed for computing the dual-objective cost that the network uses (Sec. C.3). While one gets this by-product for free, SSNP only *mimics* the underlying clustering algorithm that it learns, rather than doing data clustering from scratch. As such, we do not advocate using SSNP as a potential replacement for clustering algorithms.

	SSNP(GT)		SSNP(Km)	
Dataset	Train	Test	Train	Test
MNIST	0.984	0.942	0.947	0.817
FashionMNIST	0.866	0.815	0.902	0.831
HAR	0.974	0.974	0.931	0.919
Reuters	0.974	0.837	0.998	0.948

Table C.11: *Classification/clustering accuracy of SSNP when compared to ground truth (GT) and clustering labels (Km), trained with 5K samples, tested with 1K samples.*

C.5.8 Implementation details

All experiments discussed in this section were run on a 4-core Intel Xeon E3-1240 v6 at 3.7 GHz with 64 GB RAM and an NVidia GeForce GTX 1070 GPU with 8 GB VRAM. Table C.12 lists all open-source software libraries used to build SSNP and the other tested techniques. Our neural network implementations leverage the GPU power by using the TensorFlow Keras framework. The t-SNE implementation used is a parallel version of Barnes-Hut t-SNE [195, 116], run on all four available CPU cores for all tests. The UMAP reference implementation is not parallel but is quite fast (compared to t-SNE) and well-optimized. The implementation of MDS, Isomap, and all clustering techniques comes from Scikit-Learn [152]. Our implementation, plus all code used in this experiment, is publicly available at https://github.com/mespadoto/ssnp.

C.6 Discussion

We discuss next how the available hyperparameter settings influence the performance of SSNP concerning the seven criteria laid out in Sec. C.1.

Technique	Software used publicly available at
SSNP (our technique) Autoencoders	keras.io (TensorFlow backend)
t-SNE	github.com/DmitryUlyanov/Multicore-t-SNE
UMAP	github.com/lmcinnes/umap
Affinity Propagation Agglomerative Clustering DBSCAN Gaussian Mixture Model K-means Spectral Clustering	scikit-learn.org

Table C.12: *Software used for the SSNP implementation and evaluation.*

Quality (C1): As shown in Figures C.3 and C.4, SSNP provides better cluster separation than Autoencoders, MDS, and Isomap, and comparable quality to t-SNE and UMAP, as measured by the selected metrics (Tables C.5 and C.6). Interestingly, using ground-truth labels (SSNP(GT)) does not always yield the highest quality metrics as compared to using pseudo-labels produced by clustering. Related to the latter, K-means (Km) and Agglomerative clustering (AG) yield, overall, higher quality metrics for most tested datasets as compared to DBSCAN, Gaussian mixture models, Spectral clustering, and Affinity propagation. When we consider the neighborhood hit (NH) metric - which models the closest from all studied metrics - the ability of a projection to segregate similar samples into visually distinct clusters, SSNP(GT) performs better than all tested methods, including t-SNE and UMAP. Importantly, note that SSNP uses labels only during training and *not* during inference, so it can be fairly compared with other projection methods.

Scalability (C2): SSNP(GT) is roughly half the speed of Autoencoders during training which is expected given its dual-optimization target. Training SSNP with pseudo-labels is slower, roughly the speed of t-SNE or UMAP, which is explained by the time taken by the underlying clustering algorithm which dominates the actual training time. In our experiments, K-means seems to be faster than Agglomerative clustering, being thus more suitable when training SSNP with very large datasets. Inference time for SSNP is practically identical to Autoencoders and NNP, and one order of magnitude faster than UMAP and three orders faster than t-SNE, being also linear in the sample and dimension counts. This shows SSNP suitability to situations where one needs to project large amounts of data, such as streaming applications;

Ease of use (C3): SSNP yielded good projection results with little training (10 epochs), little training data (5K samples), and a simple heuristic of setting the number of clusters for the clustering step to twice the number of expected clusters in the data. Furthermore, we examined several hyperparameters of SSNP and found good default values (in terms of obtaining high-quality metrics) as follows: no L2 regularization, tanh activation function for the embedding layer, and He uniform weight initialization. The clustering algorithm default is K-means or Agglomerative, with K-means slightly preferred for speed reasons. As such, SSNP can be used with no parameter tweaking efforts needed.

Genericity (C4): We show results for SSNP with different types of high-dimensional data, namely tabular (HAR), images (MNIST, FashionMNIST), and text (Reuters). As these

datasets come from quite different sources and as the SSNP method itself does not assume anything about the nature or structure of the data, we believe that SSNP is generically applicable to any high-dimensional real-valued dataset.

Stability and out-of-sample support (C5): All measurements we show for SSNP are based on inference, *i.e.,* we pass the data through the trained network to compute them. This is evidence of the out-of-sample capability, which allows one to project new data without recomputing the projection, in contrast to t-SNE and other non-parametric methods.

Inverse mapping (C6): SSNP shows inverse mapping results which are, quality-wise, very close to results from Autoencoders, NNInv, and iLAMP, these being state-of-the-art methods for computing inverse projections. Additionally, SSNP computes the inverse projection at no extra cost or need for a separate implementation, in contrast to NNInv and iLAMP.

Clustering (C7): SSNP can mimic the behavior of the clustering algorithm used as its input, as a byproduct of its training with labeled data. We show that SSNP produces competitive results when compared to pseudo- or ground truth labels. Although SSNP is not a clustering algorithm, it provides this for free (with no additional execution cost), which can be useful in cases where one wants to do both clustering and DR. However, we stress that SSNP should not be considered as a replacement for state-of-the-art clustering algorithms, since it only learns to *mimic* the actual clustering. This is similar to the distinction between a classifier and an actual clustering technique.

In addition to the good performance shown for the aforementioned criteria, a key strength of SSNP is its ability to perform all its operations after a *single training phase*. This saves effort and time in cases where all or a subset of those results (*e.g.*, direct projection, inverse projection, clustering) are needed.

Limitations: While scoring high on several criteria, SSNP also has several limitations. Quality-wise, its operation in pseudo-labeling mode cannot reach the high-quality values for all metrics that are delivered by t-SNE or UMAP for challenging datasets (Tab. C.6). We believe that this is affected by the number of clusters used during training, which is related to the neighborhood size that t-SNE and UMAP use. More involved strategies in setting this number of clusters can be explored to further increase SSNP's quality. Visually, while we argue for the reason for the star-shaped cluster structures produced by SSNP (Sec. C.5.2), such patterns can be less suitable for visual exploration than the blob-like patterns produced typically by t-SNE. Using a tanh activation function partially alleviates this issue (Sec. C.5.4). However, more studies are needed to explore other activation functions that allow even better control of the visual cluster shapes. Most importantly, however, SSNP is a *learning* method. As with any such method, its quality will decrease when inferring on (that is, projecting) datasets that are too far away from the ones used during training, an issue also present for NNP and autoencoders. In contrast, methods that do not use training can obtain similar quality for any input dataset. Yet, the price to pay for such methods is that they cannot guarantee stability and out-of-sample behavior, which come with SSNP by default.

C.7 Conclusion

We presented an in-depth analysis of a dimensionality reduction (DR) method called Self-Supervised Neural Projection (SSNP) recently proposed by us. SSNP uses a neural network with a dual objective – reconstruction of the high-dimensional input data and classification of the data – to achieve several desirable characteristics of a general-purpose DR method. SSNP is, to our knowledge, the only technique that jointly addresses *all* characteristics listed in Section 1 of this paper, namely producing projections that exhibit a good visual separation of similar samples, handling datasets of millions of elements in seconds, being easy to use (no complex parameters to set), handling generically any type of high-dimensional data, providing out-of-sample support, and providing an inverse projection function.

Our evaluation added two additional dimensionality reduction methods, and four clustering algorithms, and also explored the hyperparameter space of both the clustering algorithms and neural network training to gauge SSNP's behavior. The evaluation results led to establishing default values for all these hyperparameters which obtain high quality values and also turn SSNP into a parameter-free method. Additionally, the obtained results show that SSNP with ground-truth labels yields higher quality in terms of visual cluster separation than all tested projections including the state-of-the-art t-SNE and UMAP methods. When pseudo-labels are used due to the lack of true labels, SSNP achieves lower but still competitive results with t-SNE and UMAP, slightly to significantly higher quality than autoencoders, and significantly higher quality than MDS and Isomap.

In future work, we consider studying better heuristics for controlling the clustering process, which we believe are low-hanging fruits towards improving SSNP. Another interesting direction is to explore other activation function designs that can offer control to the end users on the shape of the visual clusters that the projection creates, which would be, to our knowledge, a unique feature in the family of projection techniques. A more ambitious, but realizable, goal is to have SSNP learn its pseudo-labeling during training and therefore remove the need for using a separate clustering algorithm.

Acknowledgments

This study was financed in part by FAPESP grants 2015/22308-2, 2017/25835-9 and 2020/13275-1, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Appendix D

Appendix: Stability Analysis of Supervised Decision Boundary Maps

This chapter is partially based on the paper of the same name [135].

D.1 Introduction

As Machine Learning (ML) techniques develop and address increasingly many application domains, so does their complexity and difficulty in understanding their working. This poses problems for their adoption in contexts where transparency and accountability of inference are required [163]. Such issues are especially important for Deep Learning (DL) models which handle very high dimensional datasets and operate essentially as black boxes having millions of hidden parameters [58].

To explain ML *classifier* models, several approaches have been proposed, using variable importance [112], locally interpretable models [163], and surrogate models [130]. Visualization techniques complement such approaches by mapping the model's predictions or internal states to various visual representations [160, 161]. A recent survey of visualization techniques for the explanation of DL models was proposed by Garcia *et al.* [58].

In the above family, *Decision Boundary Maps* (DBMs) [164] are a particular visualization technique for ML classifiers. Given a multidimensional projection [131] that shows how a classifier handles some input dataset, DBMs fill the whitespace with classification results (color-mapped labels) for data points that would project at those locations. The result is a dense image that shows how the visual projection space is partitioned into per-class decision zones. Decision zone boundaries – where two or more label colors adjoin – depict locations where the classifier changes its output. DBMs offer a simple to interpret and visually scalable way to depict the working of any classification model.

In recent work [134], we proposed Supervised Decision Boundary Maps (SDBM), which extends the DBM method [164] to address several of the latter method's shortcomings, as

follows:

Quality (C1): SDBM produces decision maps that create a clearer, and far less noise-prone, visual separation of a higher number of decision zones from real-world, complex, datasets, than DBM;

Scalability (C2): SDBM has linear complexity in the number of samples and dimensions and runs on the GPU; this allows creating megapixel maps in a few seconds on commodity hardware in contrast to the minutes needed by DBM;

Ease of use (C3): SDBM produces good results with minimal or no parameter tuning;

Genericity (C4): Like DBM, SDBM can construct decision boundaries for *any* single-value classifier.

Despite these attractive points, the interpretation of the maps produced by both DBM and SDBM relies fundamentally on a *stability* assumption: Indeed, users examine such maps to determine, for instance, the size and adjacency of decision zones, to *e.g.* decide whether a classifier is well trained and/or where to add more training samples to improve it [164]. If the maps – and in particular, the borders where decision zones meet – are unstable to small changes in the training data, their interpretation can easily go wrong. Such effects were already found in earlier work on DBMs [165, 48] in terms of noise-like 'islands' that appear in DBMs constructed for complex classifier models. SDBM successfully removes such small-scale artifacts. Yet, it is still unknown how stable, thus trustworthy, are the *large scale* patterns (decision zones, decision boundaries) that SDBM creates. If these patterns are not stable, then the overall interpretation of the SDBM maps is of limited value.

In this work, we address the above open question by performing a multi-faceted stability analysis on SDBM. For this, we train three classifiers on several perturbed versions of three real-world datasets and compute and visualize the resulting decision maps as well as their changes. We also propose two novel visualizations to summarize the stability of SDBMs in the presence of several training-set changes. Our analysis shows that SDBM has an additional desirable property, namely

Stability (C5): SDBM constructs decision maps that are stable to changes. The amount of visual change – in terms of positions and sizes of the decision zones – is following the amount of change present in the input data. In particular, small data changes only yield small visual changes which do not adversely affect the interpretation of the computed decision maps.

We structure this paper as follows: Section D.2 discusses related work on visual explanation of classification models. Section D.3 details the SDBM method. Section D.4 presents results that support our contributions C1-C4 outlined above, as well as our new stability analysis and novel visualizations designed to explore it (C5). Section D.5 discusses SDBM. Finally, Section D.6 concludes the paper.

D.2 Background

We next introduce the notations used in further this paper. Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i \in \mathbb{R}, 1 \leq i \leq n$ be an n-dimensional (nD) real-valued sample. Let $D = \{\mathbf{x}_j\}$, $1 \leq j \leq N$ be a dataset of N such samples, *e.g.*, a table with N rows (samples) and n columns (dimensions). As we focus on classification models, we assume D is labeled by K label values in $C = \{c_k\}$, $1 \leq k \leq K$. Specifically, let $\mathbf{y} = \{y_j | y_j \in C\}$, $1 \leq j \leq N$ be the labels of D where sample $\mathbf{x}_j$ has label y_j. A classification model is a function

$$f : \mathbb{R}^n \rightarrow C \tag{D.1}$$

that maps between data samples and label values. The model f is typically obtained by using a training algorithm over the dataset D, such as Logistic Regression [36], SVM [35], Random Forests [20], or Neural Networks, to name a few.

A Dimensionality Reduction (DR), or projection, technique is a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q \tag{D.2}$$

that maps a sample $\mathbf{x} \in \mathbb{R}^n$ to a point $\mathbf{p} = P(\mathbf{x}), \mathbf{p} \in \mathbb{R}^q$ (typically, $q = 2$). Projecting a dataset D yields a qD scatterplot denoted next as $P(D)$. The inverse of P, denoted $P^{-1}(\mathbf{p})$, maps, or backprojects, a qD point $\mathbf{p}$ to the high-dimensional space $\mathbb{R}^n$.

Decision Boundary Maps: A Decision Boundary Map (DBM) is an image that depicts how a given model f partitions the projection space $\mathbb{R}^2$ into decision zones. A *decision zone* is a set of points $\mathbf{p} \in \mathbb{R}^2$ for which $f(P^{-1}(\mathbf{p})) = c_k$, *i.e.*, back-project to data points classified by f to the same label c_k, and is colored by the label c_k. Decision zones are separated by *decision boundaries*, which are pixels $\mathbf{p}$ whose labels (colors) differ from those of at least one 8-neighbor pixel. A DBM shows, among other things, how f partitions the high-dimensional space into decision zones, how large these zones are, how they are adjacent to each other, and how smooth the decision boundaries between classes are [165]. This gives insights on whether the model f has overfitted the training data, and how well separated the data is, *i.e.*, how difficult is the classification task. DBMs are a step forward from the key observation of Rauber *et al.* [160] who showed how projections aid in deciding whether a high-dimensional dataset is easily classifiable or not. Simply put, DBMs support the same tasks but provide more information by 'filling in' the white gaps between the points of a 2D scatterplot $P(D)$ by extrapolating the classifier f.

The DBM technique of Rodrigues *et al.* [165] relies heavily on direct and inverse projections. The direct mapping is used to create a 2D scatterplot $P(D)$ from a dataset D. The inverse mapping P^{-1} creates synthetic data points from all pixels $\mathbf{p}$ in the 2D bounding box of $P(D)$. These data points $P^{-1}(\mathbf{p})$ are then classified by f, and colored by the assigned labels $f(P^{-1}(\mathbf{p}))$. DBM has two main issues: (1) The inverse projection technique P^{-1} used, iLAMP [7], scales poorly to the hundreds of thousands of pixels a DBM has. This was addressed in [165] by using low-resolution DBMs. To increase accuracy, several points were sampled over a pixel in these maps and the pixel class (color) was set by majority-voting on the labels assigned by f to the back-projections of these points. This

scheme however creates artifacts visible as highly jagged decision boundaries. (2) Since DBM uses an unsupervised projection P, outliers in a dataset D can generate spurious 'islands' of pixels having a different label (color) than their neighbors, thus appearing as spurious decision zones that confuse the user.

Improved Decision Boundary Maps: Several improvements were proposed to address the above-mentioned issues of DBMs. Rodrigues *et al.* [48] examined the DBMs generated for four classifiers and using 28 projection techniques P and found that suitably parameterized t-SNE [118] and UMAP [125] projections limit the spurious islands in the decision maps. Next, the same authors proposed a simple filtering technique to eliminate poorly projected points from $P(D)$ and use only the remaining ones to construct the inverse mapping P^{-1} [164]. They also increased the accuracy and speed of computing the DBMs by using a deep learning technique [49] to construct P^{-1} from a given direct mapping $P(D)$. Finally, they proposed ways to visualize the distance-to-closest-boundary of all points inside a decision zone to highlight areas prone to misclassification.

Related Dense Maps: Besides DBMs used to visualize the working of a classifier model, dense maps have been used to analyze high-dimensional data in other contexts. Closer to our application, OptMap [44] uses dense maps to explore the optimization process of generic regressors $r : \mathbb{R}^2 \rightarrow \mathbb{R}$. StrategyAtlas [33] projects high-dimensional datasets to 2D using UMAP and creates dense maps showing the value of a user-selected dimension over the projection space using Shepard interpolation around the projected points [180]. Similar interpolation is used to construct dense maps showing errors at the projected points [11, 122] or dimensions that explain how neighbor projection points are related [192]. Among these techniques, only OptMap uses an inverse projection to map from the image space to the data space – all other techniques only interpolate data values at the sample points in the image space. As explained in [164], such image-space interpolation can be misleading since distances in the projection space usually do not directly reflect distances in the data space.

Dimensionality reduction in DBMs: As explained above, DBMs rely heavily on Dimensionality Reduction (DR) or projection techniques. For the DBM context, such a technique should ideally

1. Work generically for any type of high-dimensional dataset D;
2. Be computationally fast, ideally linear in the number of samples and dimensions of D;
3. Provide both the direct (P) and inverse (P^{-1}) projection;
4. Be simple to parameterize (for easy usage in practice);
5. Provide a high-accuracy projection;
6. Be stable and have out-of-sample (OOS) capability.

The first four requirements above are, we believe, evident. Requirement 5 (accuracy)

means that $P(D)$ can successfully preserve the structure of the data (clusters, neighbors, outliers) present in D. If this is not the case, any (visual) inference done on $P(D)$ – such as reasoning about the sizes, shapes, and relative positions of decision zones – may be misleading. Accuracy is typically gauged by measuring several so-called projection quality metrics [196, 176, 11, 85]. Such quality metrics have been used to filter poorly projected points to improve the DBM quality [164], as outlined earlier.

Requirement 6 also deserves a separate explanation: A projection technique P is called *stable* if small changes in its input dataset D cause only small changes in the created scatterplot $P(D)$. As a special case, a projection is called *deterministic* if it outputs the same $P(D)$ for the same input D. Stable and deterministic projection techniques are preferred for many visualization applications as they simplify the user's task – one *e.g.* can exactly reproduce the output of a projection for the same dataset D; and small-scale noise or inaccuracies in the input D do not massively affect the obtained visualization. Separately, a projection P is called to have *out of sample* (OOS) ability if it can project new, unseen, samples along those earlier provided in some dataset D, without modifying the projection $P(D)$. OOS is desirable when one needs to project a sequence of related datasets [131, 50, 198]. OOS projections are typically also stable, though the converse is not necessarily true. For DBM construction, we ideally want both P and P^{-1} to be stable and deterministic. If not, one could obtain radically different decision maps from the *same* classifier, *e.g.* when trained with slightly different data D. In turn, this would make the visual interpretation of the respective classifier via the DBMs very challenging if not hardly possible.

Measuring projection *stability* is a relatively new and little explored topic, as most quantitative studies on DR focused so far on ensuring high projection quality (see the survey in [50]). Key difficulties for stability measurement are defining the 'allowable' change in the data D and in the projection $P(D)$. Vernier *et al.* [198] present, to our knowledge, the first attempt to quantify stability for dynamic projection techniques by measuring the correlation of changes in the 2D distances between points in $P(D)$ and their nD distances in D. Data changes are implicitly given by the application domain as D is a time-dependent dataset. Bredius *et al.* [19] gauge the stability of a specific projection method [47] by explicitly synthesizing noise-like changes of D and depicting the changes in $P(D)$. However, they perform no quantitative stability measurements. Espadoto *et al.* [43] use similar noise-like changes to train a projection method to behave less sensitively (thus, be more stable) in their presence. However, they do not explicitly measure or reason about projection stability. For inverse projections P^{-1} or DBMs, we are not aware of any stability study. Our work here is, to our knowledge, the first study that explicitly measures the stability of a DBM pipeline involving both direct and inverse projections.

Many DR techniques have been proposed over the years, as reviewed in various surveys [72, 119, 42, 184, 109, 38, 210, 131, 50]. Below we describe a few representative ones from DBM computation perspective and outline how these fare concerning requirements 1-5 mentioned above.

Principal Component Analysis [86] (PCA) is one of the most popular DR techniques for many decades, and complies well with all requirements except 5 (accuracy), especially

for data of high intrinsic dimensionality. PCA was used to compute both P and P^{-1} by the OptMap visualization method for regressor analysis [44]. However, the authors noted that higher-quality results could be obtained by using a more accurate projection technique.

The Manifold Learning family of methods contains techniques such as MDS [194], Isomap [188], and LLE [167], which aim to capture nonlinear data structure by mapping to 2D the high-dimensional manifold on which data is located. These methods generally yield better results than PCA (5) but do not scale well computationally (2), and also yield poor results when the intrinsic data dimensionality is higher than two. Also, many such methods require careful parameter tuning (4) to obtain suitable results.

The SNE (Stochastic Neighborhood Embedding) family of methods, of which the most popular member is t-SNE [118], are best known for the high quality of the projections they produce (5). Yet, they can be hard to tune [204], and typically have no OOS capability and/or stability (6). Parametric t-SNE [117] adds OOS and stability at the expense of a significantly slower and more complex implementation. Several refinements of t-SNE improve speed (2), such as tree-accelerated t-SNE [115], hierarchical SNE [154], and approximated t-SNE [153], and various GPU accelerations of t-SNE [155, 25]. Uniform Manifold Approximation and Projection (UMAP) [125], while not part of the SNE family, generates projections with comparable quality to t-SNE (5), but much faster (2), and with OOS capability (6).

All the above projection techniques work in an *unsupervised* way – they use only distance information between points in D to compute $P(D)$. Recently, Espadoto *et al.* [47] proposed Neural Network Projection (NNP) to learn the projection $P(D)$, computed by any user-selected technique P, from a small subset $D' \subset D$, using a deep learning regressor. While slightly less accurate than the original P (5), NNP is computationally linear in the size and dimensionality of D (2), has OOS ability (5), and is simple to implement and parameter-free (4). A recent study [19] showed that NNP is very stable to a wide range of perturbations of its input data D. NNP was further refined [126] to use neighborhood information between samples in D and further increase the projection accuracy (5). A related idea to NNP was used by NNInv [49] to learn the inverse mapping P^{-1}. NNP and NNInv were next extended by Self-Supervised Network Projection (SSNP) [46], which can be used either in a *self-supervised* fashion, by computing pseudo-labels by a generic clustering algorithm on D, or in a *supervised* fashion (similar to NNP), using ground truth labels $\mathbf{y}$ coming with D. We choose SSNP to create our proposed SDBM as it complies well with our earlier stated six requirements for a projection method in the DBM context:

1. SSNP works generically for any high-dimensional dataset;
2. SSNP is GPU-accelerated, which makes it one to two magnitude orders faster than DBM (see next Sec. D.4.4);
3. SSNP provides both the direct and inverse mappings (P and P^{-1}) needed by the DBM method;
4. SSNP is parameter-free (after its training phase has been completed);
5. SSNP provides good cluster separation by partitioning the data space D as a classifier would do, which is closely related to the original goal of DBM;

6. SSNP is parametric, thus has OOS and, as we show next in Sec. D.4.3, leads to a DBM computation stable to changes in the input dataset D.

D.3 Method

We next describe our proposed SDBM technique and how it is different from its predecessor, DBM. Our technique has five steps as illustrated by the pipeline in Fig. D.1. Below, we detail all these steps.

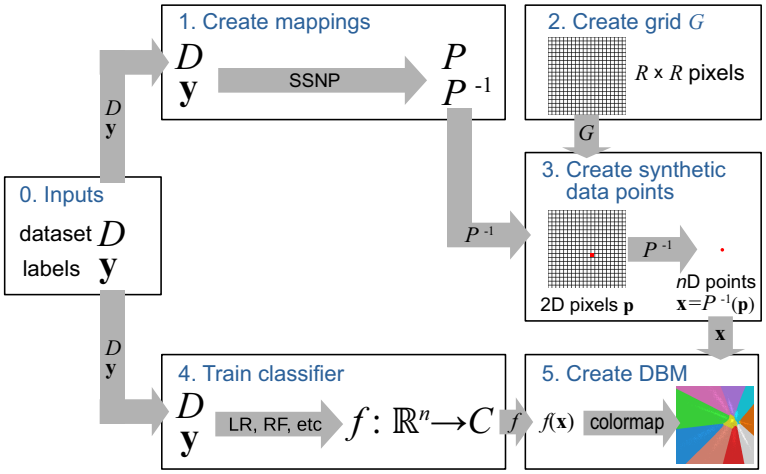


Figure D.1: *SDBM pipeline (see Sec. D.3).*

0. Input data: SDBM needs only two inputs – a high-dimensional dataset D and its label vector $\mathbf{y}$. The definitions of these are given in Sec. D.2. As stated earlier, no restrictions exist on the data dimensionality n, data nature, or the number of labels K used in $\mathbf{y}$. Simply put, any labeled dataset $(D, \mathbf{y})$ that can be used to build a classification model for some problem is acceptable as input for SDBM. Therefore, SDBM is applicable to visualize the decision boundaries and zones of any classifier.

1. Create mappings: We train SSNP to create the direct and inverse projections P and P^{-1} based on D and $\mathbf{y}$. This step is fundamentally different from DBM. In detail: DBM requires the user to supply a projection technique P to map D to a 2D scatterplot $P(D)$. Next, DBM uses $P(D)$ to learn the inverse mapping, or inverse projection P^{-1}. For this, DBM uses various inverse projection techniques such as NNInv [49] or iLAMP [7] (see also Sec. D.2). The problem with this is that, depending on the direct projection P chosen by the user, these inverse projection techniques may have difficulties in computing an accurate inverse projection P^{-1}. That is, for several points $\mathbf{x}$ in the input domain of the classifier, $P^{-1}(P(\mathbf{x})) \neq \mathbf{x}$, *i.e.*, P^{-1} is not the exact inverse of P. In practice, this leads to jagged decision boundaries and noise-like small islands scattered all over the dense maps created by DBM (see examples in Fig. D.5 later on). SDBM does not have this problem as it uses the SSNP method to *jointly* compute both P and P^{-1}, as mentioned in Sec. D.2.

As shown by our results in Sec. D.4, this joint computation of P and P^{-1} used by SDBM significantly reduces the above-mentioned artifacts in the decision maps.

2. Create 2D grid: Create an image $G \subset \mathbb{R}^2$ with a resolution of R pixels, where R is chosen by the user. Higher R values capture more details in the decision maps but take longer to compute – more precisely, the computation time is linear in the number of pixels of the map. This is different from DBM. In detail, DBM uses the full resolution of G to compute the direct projection $P(D)$ but then evaluates P^{-1} on a subsampled version of G of a lower resolution than R to reduce computation time (see Sec. D.2). In contrast, SDBM uses the full user-specified resolution R to compute both P and P^{-1} (for all experiments in this paper, we set this to $R = 300^2$ pixels). SDBM does not need to use subsampling since its underlying direct-and-inverse projection technique, SSNP, is fast enough to treat the full resolution specified by the user.

3. Create synthetic data points: Use the trained P^{-1} (delivered by SSNP in step 1) to map each pixel $\mathbf{p} \in G$ to a high-dimensional data point $\mathbf{x} \in \mathbb{R}^n$. This is similar to DBM, except for the use of a dense pixel grid and the jointly-trained P and P^{-1} mappings delivered by SSNP (see step 1).

4. Train classifier: Train the classifier f to be visualized using the dataset D and its labels $\mathbf{y}$, as in a usual machine learning setting. This step is identical to DBM. Any single-class-output classifier $f : \mathbb{R}^n \rightarrow C$ can be used generically, *e.g.*, Logistic Regression (LR), Random Forests (RF), Support Vector Machines (SVM), or neural networks. Moreover, no restrictions are placed on the design or architecture of f. Also, note that the classifier training occurs *after* the construction of the mappings P and P^{-1} in step 2. That is, these mappings have no knowledge of the class labels. Hence, we can reuse these mappings computed in step 2 to next construct decision maps to visualize any classifier to be trained on the given inputs $(D, \mathbf{y})$. Simply put, once step 2 is executed, we can next quickly construct decision maps to compare how several classifiers perform on a given $(D, \mathbf{y})$. We illustrate this further in our results (Sec. D.4).

5. Create DBM: Color all pixels $\mathbf{p} \in G$ by the values of $f(P^{-1}(\mathbf{p}))$, *i.e.*, the inferred classes of their corresponding (synthetic) data points, using a categorical color map. In this paper we use the 'tab20' color map [79]. This is the same as DBM.

5b. Encode classifier confidence (optional part of step 5): For a classifier f that provides the probability of a sample $\mathbf{x}$ belonging to a class c_k, we encode this probability in the brightness of the pixel $\mathbf{p}$ that back-projects to $\mathbf{x}$. The lower the confidence of the classifier is, the darker the pixel appears on the map. This informs the user of the confidence of the decision zone in that area – dark areas in the map, typically close to decision boundaries, indicate regions in the data space where the classifier is less confident. This is the same as DBM.

5c. Draw scatterplot (optional part of step 5): If desired, one can visualize the projection $P(D)$ of the training set D by drawing it as a scatterplot atop the DBM. Note that this is the only place where SDBM uses the projection P. The added value of showing this scatterplot is showing users where are the actual data points in the decision map. If users do not wish to see this scatterplot, we only use the inverse projection P^{-1} computed in

step 3 above. Since we use SSNP to jointly compute P and P^{-1}, we obtain P for free, so there is no additional cost to drawing this scatterplot.

Summarizing the above pipeline in simple words, SDBM creates an image G, takes every pixel of this image, and backprojects it to the data space $\mathbb{R}^n$ to obtain a data point, computes a label for this point by using the classifier we wish to explore, and finally colors the pixel to show the class label and, optionally, the classifier's confidence, at that location. The result is a densely colored map where same-color regions indicate regions in the data space where the classifier yields the same output (label), and color boundaries between adjacent regions indicate the decision boundaries of the classifier.

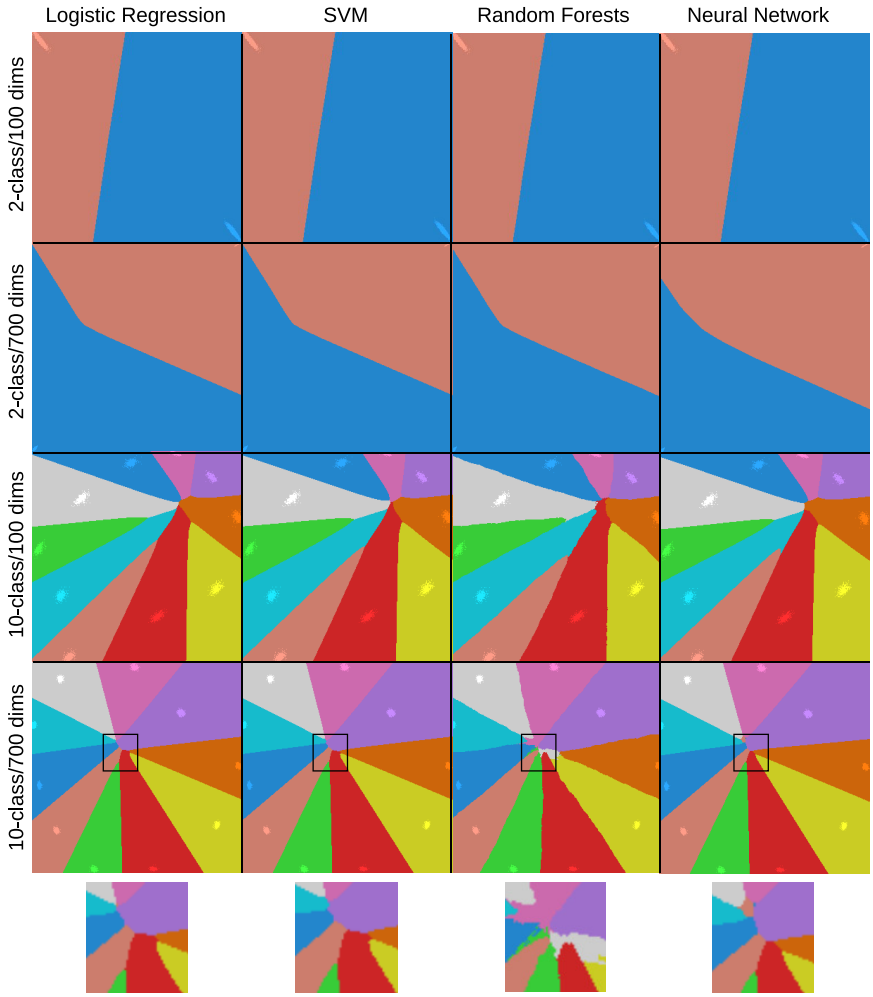


Figure D.2: *Decision Boundary Maps (DBMs) created with SDBM for several classifiers (columns) and synthetic datasets (rows). Lighter pixels represent training samples from the datasets D. Insets show decision map details around the region where all decision zones meet for the 10-class, 700-dimensional dataset.*

D.4 Results

We next evaluate SDBM against the desirable criteria C1-C5 introduced in Sec. D.1. Specifically, we analyze quality (C1) by first using SDBM with synthetic data in a controlled

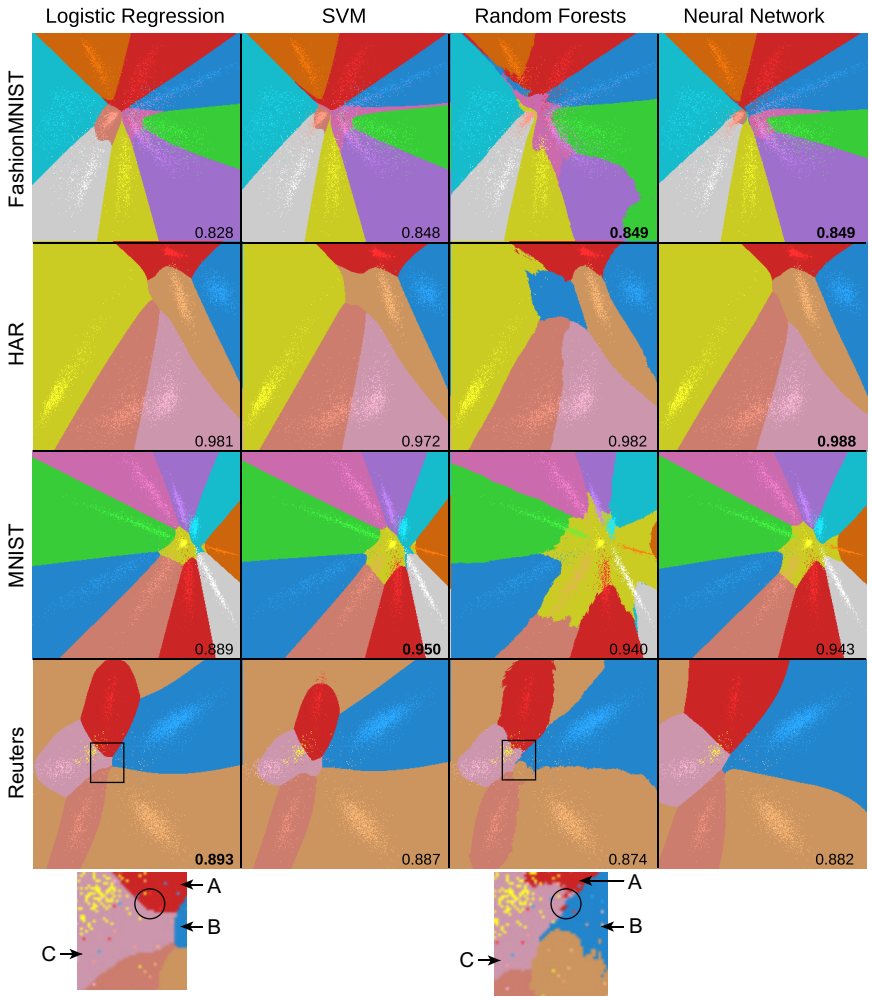


Figure D.3: *Decision Boundary Maps (DBMs) created with SDBM for several classifiers (columns) and real-world datasets (rows). Numbers inside each map indicate test accuracy obtained by each classifier, bold indicating top performers. Lighter pixels represent training samples from the datasets D. Insets show details in the decision zones for Logistic Regression and Random Forests for the Reuters dataset.*

setting, as we know what the 'ground truth' shapes of the decision zones are for a given synthetic dataset and given classifier (Sec. D.4.1). We next assess quality for more complex real-world datasets and additional classifiers (Sec. D.4.2) and also compare SDBM with DBM. This shows also that SDBM is generic (C4) and that it increases quality as compared to DBM. Next, and in addition to [134], we present several experiments that measure SDBM's stability in the presence of different amounts and types of data change to support our stability claims (C5). Finally, we show how SDBM compares to DBM speed-wise and thereby justify our scalability claims (C2, Sec. D.4.4). We end this section by providing full implementation details for SDBM (Sec. D.4.5).

D.4.1 Quality on Synthetic Datasets

To assess how SDBM performs in a controlled situation, we consider several synthetic Gaussian blobs with 5000 samples, with varied dimensionality (100 and 700), and varied number of classes (2 and 10). All points in a blob have the same class label. These are,

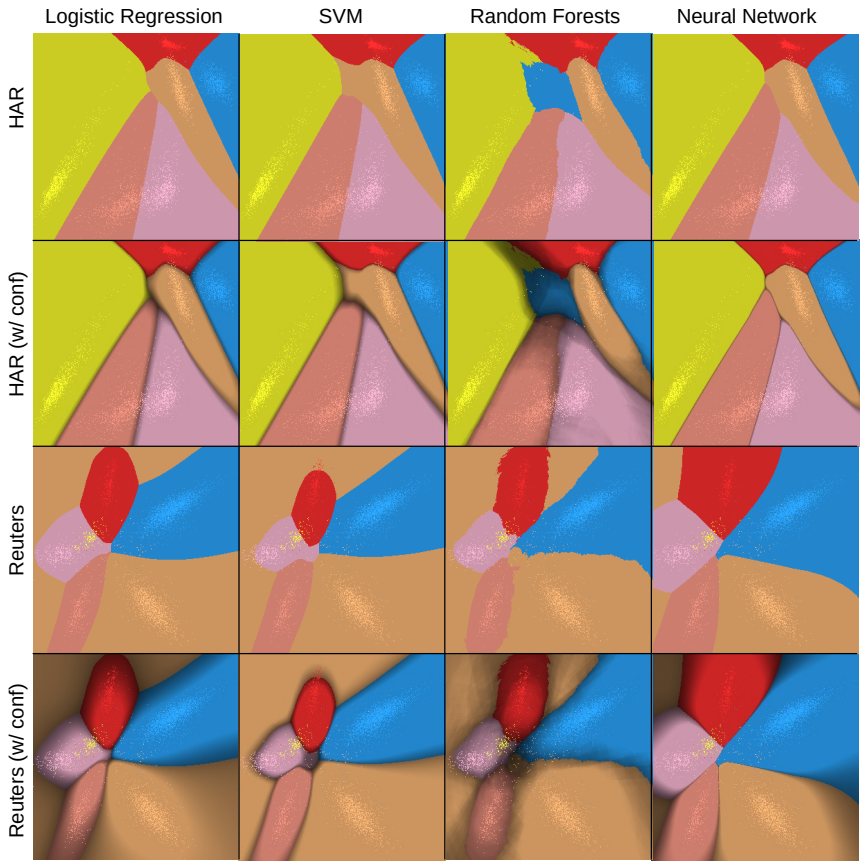


Figure D.4: *Decision Boundary Maps created with SDBM for several classifiers, HAR, and Reuters datasets. Columns show different classifiers. Rows show different datasets, with and without confidence encoded into brightness.*

thus, easily classifiable datasets, for which we expect the decision zones to 'surround' the respective blobs. We construct decision maps for four classifiers, namely Logistic Regression [36], SVM [35] (with an RBF kernel), Random Forests [20] (200 estimators), and a Neural Network (multi-layer perceptron with 3 layers of 200 units each). All these classifiers can handle the synthetic datasets with 100% accuracy. Consequently, as said above, we expect to see clearly-separated decision zones surrounding the data blobs in the projection.

Figure D.2 shows the SDBM maps for all the dataset *vs* classifier combinations, with decision zones colored by class labels. Projected samples in $P(D)$ are colored by their class too, but slightly brighter than the maps so they are visible around their respective decision zones. We first see that the projections (bright 'spots' in the figure) indicate well-separated blobs, which confirms the easy structure of these datasets. We also see that all decision zones are compact and with smooth boundaries, as expected for such simple datasets, and enclose the Gaussian blobs with the same respective labels. For example, the red and blue zones for the 2-class, 100-dimensional dataset (Fig. D.2, top row), contain two clusters of light red, respectively light blue, projected points. The maps for Logistic Regression show almost perfectly straight boundaries, which is a known fact for this classifier. In contrast, the more sophisticated classifiers, Random Forests, and Neural Networks create

boundaries that are slightly more complex than Logistic Regression and SVM for the most complex dataset. The differences are best visible for Random Forests and Neural Networks in the small wiggles of the decision zone shapes in the center of the maps in Fig. D.2, bottom row (see also insets).

D.4.2 Quality on Real-World Datasets

We next show how SDBM performs on three real-world datasets. To select such datasets, we look at candidates who are (a) challenging for classification problems; (b) quite diverse in terms of data provenance, dimensionality (hundreds of dimensions), and size (thousands of samples); (c) well-known, and openly accessible, to the machine learning community, for comparison and replication purposes. This quickly leads us to select datasets used in many ML evaluation benchmarks. Additionally, we note that using such datasets makes sense in our context since we want to evaluate a technique (SDBM) that is designed to visualize the behavior of classifier models.

With the above requirements, we selected the following datasets for evaluating SDBM:

FashionMNIST [209]: 10K samples of $K = 10$ types of clothing images, rendered as 28x28-pixel gray scale images, flattened to 784-element vectors. We also use a subset of this dataset containing only two classes, namely *Ankle Boot* and *T-Shirt*, to show an example where classes are more easily separable.

Human Activity Recognition (HAR) [9]: 10299 samples from 30 subjects performing $K = 6$ daily activities, and used for human activity recognition. The samples have 561 dimensions that encode in the time and frequency domains 3-axial linear acceleration and 3-axial angular velocity measured on the subjects.

MNIST [101]: 70K samples of $K = 10$ handwritten digits from 0 to 9, rendered as 28x28-pixel gray scale images, flattened to 784-element vectors. This dataset was downsampled to 10K observations for all uses in this paper.

Reuters Newswire Dataset [191]: 8432 samples of news report documents, from which 5000 attributes were extracted using the standard TF-IDF [173] text processing method. From the full dataset, we use only the $K = 6$ most frequent classes.

Figure D.3 shows the SDBM maps for these datasets for the same classifiers used in Sec. D.4.1. These datasets are considerably more complex than the synthetic ones (Sec. D.4.1), also seen by the varying accuracies they achieve for the different classifiers. Still, for all combinations, the classifiers' decision zones are visible in Fig. D.3. Also, we see – like for the synthetic datasets – how these decision zones surround the blobs of training-set samples, depicted as lighter-colored points in Fig. D.3. As for the synthetic datasets, simpler classifiers (Logistic Regression and SVM) show decision zones that are more contiguous and have smoother, simpler, boundaries. More complex classifiers (Random Forests and Neural Networks) show more complex shapes and topologies of the decision zones. The maps for the Random Forest classifiers show very jagged boundaries. This can be a result of having an ensemble of classifiers working together. An interesting insight can be obtained when comparing the maps for different classifiers trained on the same

dataset. Consider, *e.g.*, for the Reuters dataset (bottom row), the best classifier (Logistic Regression (LR), accuracy 0.893) and the poorest classifier (Random Forests (RF), accuracy 0.874). The projected points are the *same* for the two maps since we use the same training set. Still, we see different shapes and sizes of the decision zones. Consider now a small area in the two maps (see insets at the bottom of Fig. D.3). As described earlier, both training-set points and decision zones are colored by label values. Hence, misclassified points will have different colors from their surrounding zones, while correctly classified points have the same (slightly lighter) colors as the surrounding zones. Comparing the map details for LR and RF, we see that the red decision zone (A) is smaller for RF than for LR, while the blue zone (B) is comparatively larger. In the RF inset, we see that several red points in the black circle fall in the blue (B) and pink (C) decision zones, indicating misclassifications. These red points fall under the large red decision zone for the LR map. Hence, we conclude that the shapes of the LR decision zones, in this region, are more correctly following the training data than those of RF. Similar reasoning can be done to compare other decision map areas.

Encoding classifier confidence: Figure D.4 shows SDBM maps with classifier *confidence* encoded as brightness, as described in Sec. D.3. We see the added value of depicting confidence if we compare the first-*vs*-second (HAR), respectively third-*vs*-fourth (Reuters), rows in Fig. D.4. The confidence maps show a brightness gradient, dark close to the decision boundaries (where colors change in the maps) and bright deep in the decision zones. This shows that confidence increases as we go deeper into the decision zones, *i.e.*, closer to the training samples. For the HAR dataset, these dark bands are quite thin for Logistic Regression and SVM, thicker for Random Forests, and extremely and uniformly thin for Neural Networks. This tells us that Neural Networks have an overall very high confidence everywhere (except very close to the decision boundaries); Logistic Regression and SVM are less confident close to the boundaries; and Random Forests have a higher variation of confidence over the data space. Note how these findings match the classification accuracy values (Fig. D.3). For Random Forests, the darkest region covers the central blue decision zone and the top-right of the left yellow zone. These are exactly the areas where the map for Random Forests significantly differs from those of all the other three classifiers. Hence, we can infer that the island-like blue decision zone that Random Forests created is likely wrong, as it is low confidence *and* different from what the other three classifiers created in that area. For the Reuters dataset (Fig. D.4 bottom row), all classifiers produced a beige region at the top left corner. Brightness shows us that all classifiers except SVM treat this region as a low-confidence one. This can be explained by the total absence of training samples in that region. This also tells us that the behavior of SVM in this region is likely wrong.

Confidence visualization also helps to quickly assess the *overall difficulty* of classifying a dataset. Consider *e.g.* the Reuters dataset (Fig. D.4 bottom row). Compared to HAR (Fig. D.4, second row), the decision maps for this dataset are darker for all four classifiers. This shows that it is harder to *extrapolate* (during inference) from a model trained on Reuters than one trained on HAR. Note that this is not the same as the usual testing-after-training in ML. Indeed, for testing, one needs to 'reserve' a set of labeled samples that cannot be used during training. In contrast, SDBM does not need to do this as it synthesizes 'testing' samples on the fly via the inverse projection P^{-1}. Also, classical ML testing only

gives a global or per-class accuracy. In contrast, SDBM gives a per-region-of-the-data-space confidence, encoded by brightness.

Comparison with DBM: Figure D.5 shows the SDBM maps side-by-side with maps created by the original DBM technique for Logistic Regression, Random Forest, and k-NN classifiers and three real-world datasets. For DBM, we used UMAP [125] for the direct projection and iLAMP [7] for the inverse projection. Several observations can be made, as follows.

First, we see that the SDBM and DBM projections $P(D)$ of the same datasets are not the same – compare the bright-colored dots in the corresponding figures. This is expected since DBM employs a user-chosen projection technique P (UMAP in our case) while SDBM *learns* P from the label-based clustering of the data using the SSNP method (see Sec. D.3). Since the DBM and SDBM projections $P(D)$ differ, it is expected that the overall shapes of the ensuing decision maps will also differ – see *e.g.* the nearly horizontal decision boundary between the blue and red zones for Random Forests with DBM for FashionMNIST (2-class) *vs* the angled boundary between the same zones for the same classifier, same dataset, with SDBM (Fig. D.5, middle row, two leftmost images). For the relatively simple classification problem that FashionMNIST (2-class) is, this is not a problem. Both DBM and SDBM produce useful and usable renditions of the two resulting decision zones, showing that this classification problem succeeded with no issues.

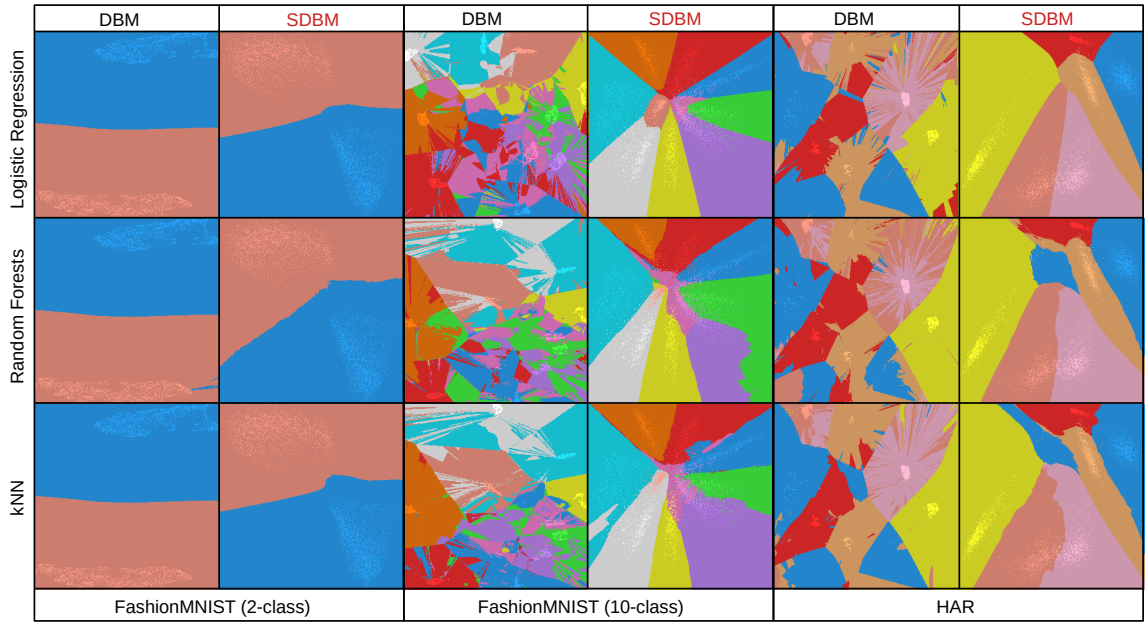


Figure D.5: *Comparison between SDBM and DBM using three different datasets and three classifiers.*

For more difficult datasets (FashionMNIST 10-class or HAR), the situation is very different: DBM shows *highly* noisy pictures, in which it is very hard to say where and which are the actual decision zones. If these images were correct, this would mean that none of the three tested classifiers could correctly handle these two datasets. Indeed, such noise-like rapid changes as the DBM images show would mean that the classifiers would change decisions extremely rapidly and randomly as points only slightly change over the data space. This is *known* not to be the case for these classifiers. In more detail: Logistic

Regression has built-in limitations of how quickly its decision boundaries can change [164]. k-NN essentially constructs a Voronoi diagram around the same-class samples in the *n*D space, partitioning that space into cells whose boundaries are smooth manifolds. DBM does not show any such behavior (Fig. D.5, third and fifth columns). In contrast, SDBM shows a far lower noise level and far smoother, contiguous, decision zones and boundaries. Even though we do not have formal ground truth on how the zones and boundaries of these dataset-classifier combinations look, SDBM matches better the prior knowledge we have on these problems than DBM.

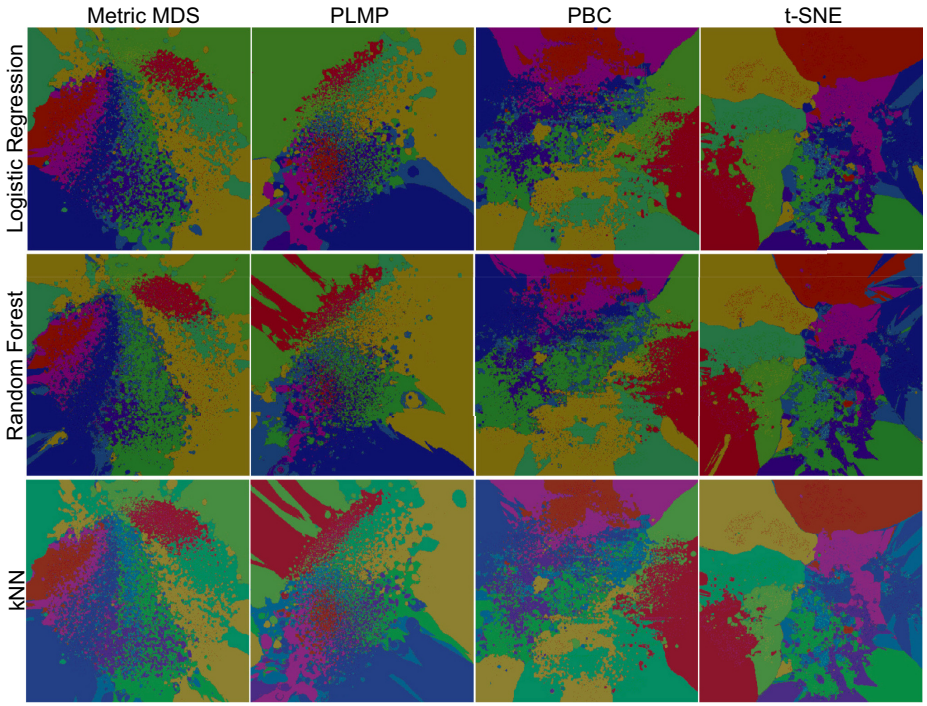


Figure D.6: *DBM images using more direct and inverse projection methods, FashionMNIST (10 class). Compare these images with SDBM for the same classifiers and dataset in Fig. D.5 (column 4).*

However, DBM's results depend on the choice of the direct projection P and inverse projection P^{-1} it uses (see Sec. D.3). To compare SDBM with DBM under these degrees of freedom, we ran DBM for the three classifiers shown in Fig. D.5 on the FashionMNIST 10-class dataset but used four different projection methods P (Metric MDS [99], PLMP [149], Projection by Clustering (PBC) [148], and t-SNE [118]), and used NNInv [49] instead of iLAMP for the inverse projection P^{-1}. Figure D.6 shows the decision maps created by DBM for these configurations. We see that these are practically as noisy as the DBM results shown in Fig. D.5 (column 3). In contrast, the SDBM results (Fig. D.5, column 4) show better separated, less noisy, smoother-boundary decision zones. This strengthens our claim that SDBM produces higher-quality maps than DBM.

D.4.3 Stability Analysis

We now turn to the stability desirable criterion (C5, Sec. D.1). As explained there and also in Sec. D.2, a stable decision map algorithm shows only small changes in the output decision map when its input, *i.e.*, the labeled dataset (D, y) it was constructed to show,

change little. By extension, when this input does not change, the output map should also not change. If this is not the case, then the decision map may show patterns that are highly influenced by irrelevant (small) changes in the input data or algorithm parameters which, in turn, can be highly misleading.

Figures D.5 and D.6 show two reasons why the original DBM algorithm is *unstable*. First, we see that DBM creates very noisy, discontinuous, decision maps. However, as we explained in Sec. D.4.2, the visualized classifiers are known to change their decision *slowly* as their inputs change. The DBM images in Figs. D.5 and D.6 show a different picture, suggesting that the classifiers rapidly change outputs as inputs only slightly change. Thus, DBM itself introduces instabilities in the computation of the decision maps which are not genuinely there in the visualized classifiers. In contrast, SDBM shows far smoother, less noisy, decision maps, for the *same* classifiers and datasets. Secondly, Fig. D.6 shows that DBM creates very different (and still noisy and discontinuous) decision maps when we change its two hyperparameters, namely the direct projection P and inverse projection P^{-1}, for the *same* dataset-classifier combination. This is by definition an unstable algorithm.

While SDBM shows far smoother, more continuous, decision maps than DBM, we would like more evidence to claim that SDBM is stable. In this section, we address this by explicitly measuring SDBM's stability, as follows (see also Fig. D.7; compare to Fig. D.1 that shows the baseline SDBM method):

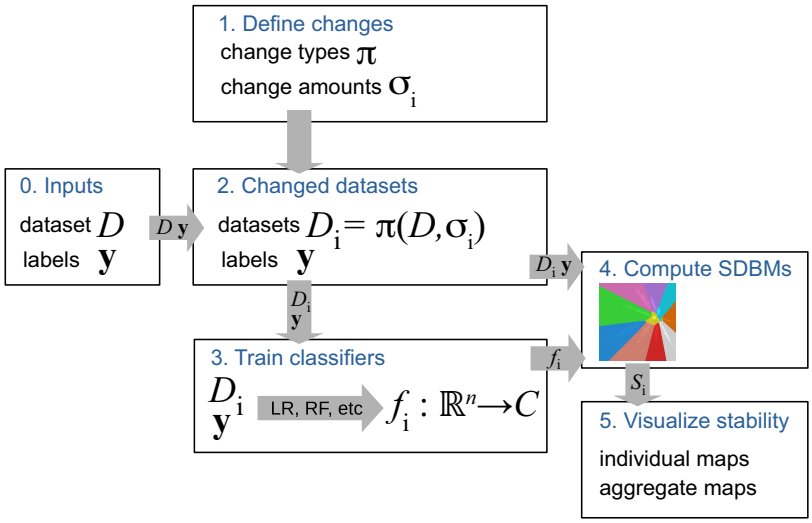


Figure D.7: *Pipeline for assessing SDBM stability.*

- let $D \in \mathbb{R}^n$ be a training dataset with labels $\mathbf{y}$ (Fig. D.7 step 0);
- let $f(D)$ be a classification model trained on D and $\mathbf{y}$;
- let $S(f)$ be the decision map computed by SDBM on $f(D)$;
- let $\pi : \mathbb{R}^n \times [0,1] \to \mathbb{R}^n$ be a change, or perturbation function. That is, $\pi(D,\sigma)$ is the dataset D changed by π with a change intensity $\sigma \in [0,1]$. Larger σ values change D more, and $\pi(D,0) = D$, *i.e.* a value $\sigma = 0$ means no change. We record this change intensity σ by a set of samples, or change amounts, σ_i (Fig. D.7 step 1);

- compute $D_i = \pi(D, \sigma_i)$, a set variations of the dataset D, changed by perturbation π, with change intensities σ_i (Fig. D.7 step 2);
- train the models $f_i = f(D_i)$. Labels $\mathbf{y}$ of D_i stay the same as those of D, only the sample values change (Fig. D.7 step 3). Note that we apply the changes on the *training* sets of the visualized classifiers, not the test sets since changing a test set will not change the decision map of a trained classifier;
- construct the decision maps $S_i = S(f_i)$ (Fig. D.7 step 4);
- visualize the maps S_i to interpret the stability of SDBM (Fig. D.7 step 5).

The intuition of the above procedure is simple: Let S_0 be the decision map computed by SDBM for a dataset D and some classifier f. Let S_i be the decision maps computed by SDBM for the same classifier but for increasingly perturbed versions D_i of the dataset. If SDBM is a stable method, then it should produce decision maps S_i which are similar to S_0 for low i values and increasingly different from S_0 as i increases. Note that a similar definition of stability – small input data changes should lead to small output visualization changes – was used to assess other visualization techniques for high-dimensional data such as projections [198, 19] and treemaps [199, 197].

We apply this procedure to three types of data changes π defined as follows:

- *Add constant:* π adds a fixed bias value σ to all dimensions of D. For the image datasets (MNIST, FashionMNIST), we used $\sigma_i \in \{0.07, 0.15, 0.3\}$, which corresponds to a 'brightening' of the images with up to 30%. For the Reuters text dataset, we used $\sigma_i \in \{0.05, 0.08, 0.2\}$;
- *Drop dimensions:* π sets to zero a given number of randomly chosen dimensions from the n ones of D. We used here $\sigma_i \in \{0.1n, 0.2n, 0.3n\}$, which means that π has an effect similar to removing up to 30% of D's dimensions;
- *Random noise:* π adds to all D's dimensions random noise sampled from a normal distribution with mean 0 and standard deviation $\sigma_i \in \{0.01, 0.05, 0.1\}$.

These changes are related to the ones used in [19] to test the stability of the NNP projection – not the same as our classifier decision maps, but related in spirit. For additional rationale referring to the purposefulness of these changes, we refer to [19].

Visualizing stability: We next apply these three change types π, each sampled for three change intensities σ_i, for the SDBM maps constructed for Logistic Regression and Neural Networks trained with the MNIST, FashionMNIST, and Reuters datasets – thus, we compute and evaluate a total of $3 \times 3 \times 2 \times 3 = 54$ decision map images. Figures D.8, D.9, and D.10 show these images for the three datasets with confidence encoded into brightness (see Sec. D.4.2). In each figure, the leftmost column (labeled 'without noise') shows the decision map of the original, unperturbed, dataset. The rightmost three columns show, per row, the decision maps for the respective (classifier, dataset, noise-type) combination, for increasing amounts of noise amounts. As explained earlier, if the decision maps slowly and increasingly change as the noise level increases relative to the noise-free map, this means that the SDBM method is stable.

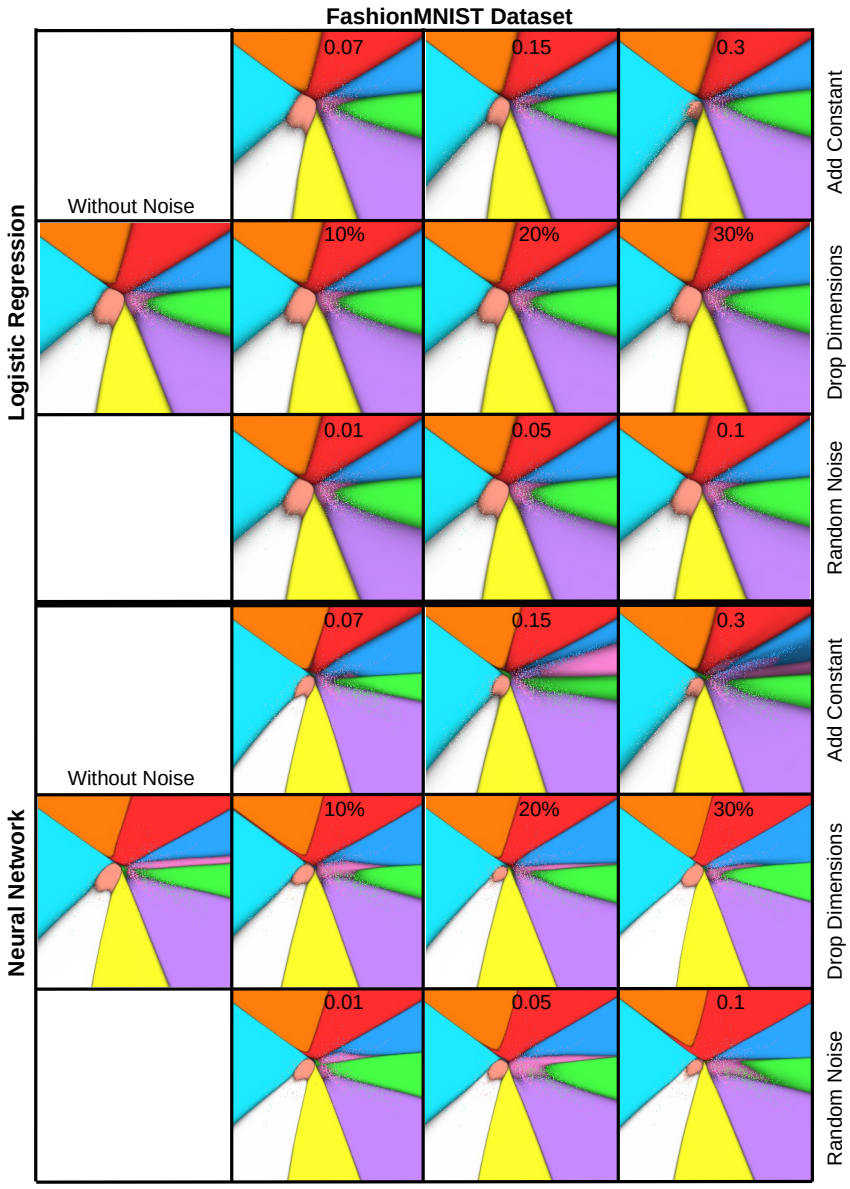


Figure D.8: *SDBM decision maps for two classifiers trained with varying types and amounts of noise, FashionMNIST dataset. Confidence is encoded into brightness. The leftmost column shows the maps for the original, un-noised, datasets. As more noise is added (columns 2 to 4, noise amounts marked inside images), the decision maps start progressively diverging from the original, leftmost, map.*

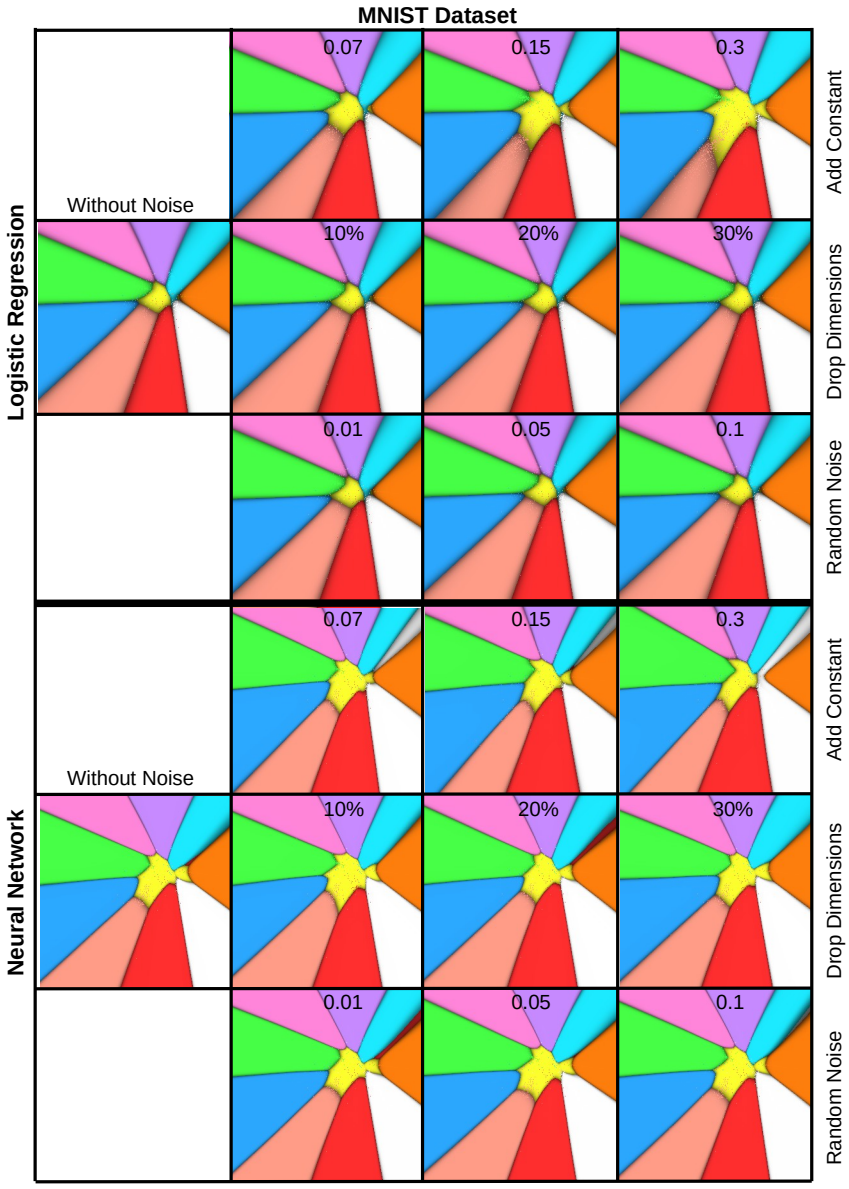


Figure D.9: *SDBM decision maps for two classifiers trained with varying types and amounts of noise, MNIST dataset. Confidence is encoded into brightness. The leftmost column shows the maps for the original, un-noised, datasets. As more noise is added (columns 2 to 4, noise amounts marked inside images), the decision maps start progressively diverging from the original, leftmost, map.*

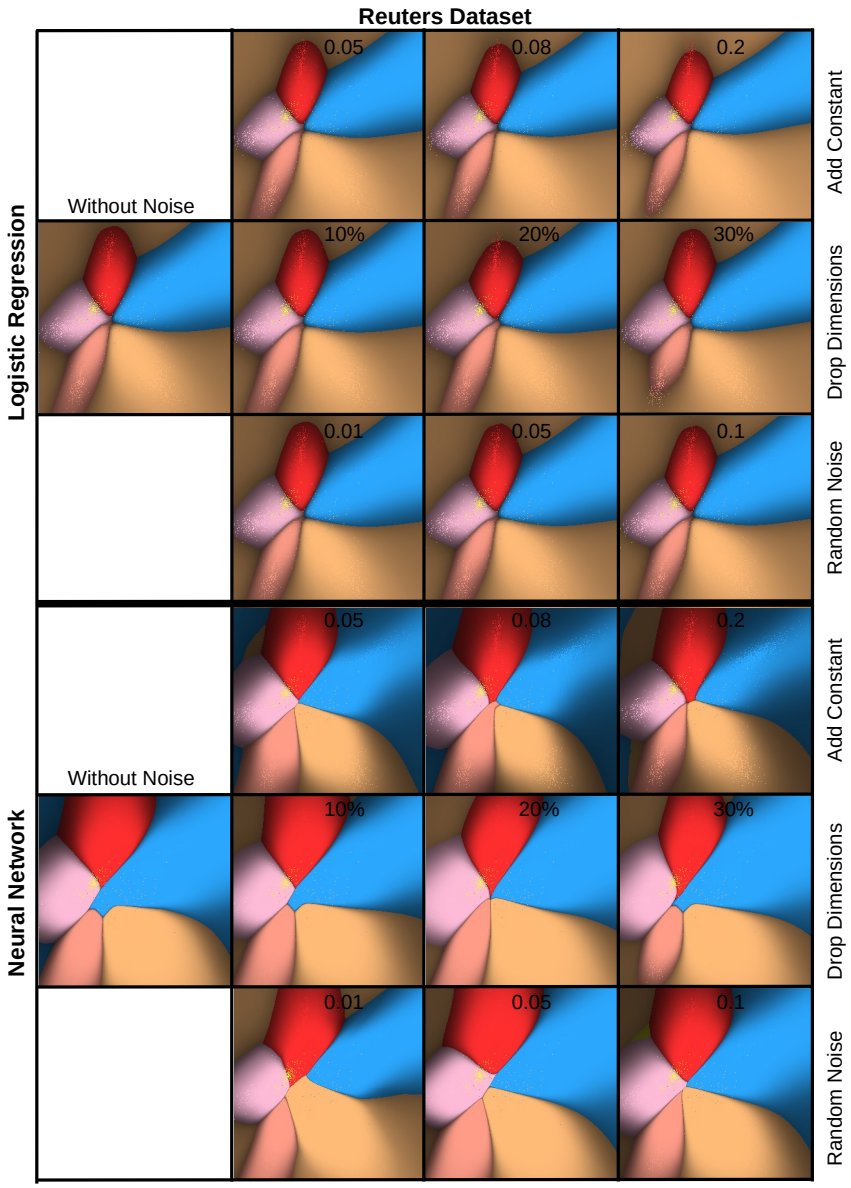


Figure D.10: *SDBM decision maps for two classifiers trained with varying types and amounts of noise, Reuters dataset. Confidence is encoded into brightness. The leftmost column shows the maps for the original, un-noised, datasets. As more noise is added (columns 2 to 4, noise amounts marked inside images), the decision maps start progressively diverging from the original, leftmost, map.*

Figures D.8, D.9, and D.10 show indeed this stability. If we scan each row left-to-right, we see how the images become progressively more different from the leftmost image (decision map for the unchanged dataset). Different rows for a classifier show the effect of the three different change types. Interestingly, in terms of the overall amount of visual change, these effects are quite similar. Take, for example, Logistic Regression trained with FashionMNIST (Fig. D.8, top three rows): The nine images to the right are quite similar among themselves and also similar to the decision map of the unchanged dataset (shown in the left column). Also, we see that the changes in the maps do not seem to differ – in terms of amount – for the three datasets. The fact that SDBM appears to be quite stable for different change types *and* for different datasets is a quite unexpected result, as the nature of the three change types and the three tested datasets is quite different. Related work [19] has shown that, when testing the stability of the NNP deep-learned projection [47], different change types (similar to ours) have quite different effects and also that the effects differ strongly as a function of the dataset. In other words, SDBM appears to be a more stable method than NNP. There can be many factors that make SDBM and NNP different, including the supervised nature of NNP *vs* self-supervised one of SDBM and the fact that SDBM learns and next applies both a direct and inverse projection, whereas NNP only learns a direct projection.

Figures D.8, D.9, and D.10 also outline two other important aspects of SDBM. First, we see that not just the shapes and sizes, but also the relative *positions* in the image of the various decision zones are quite stable over change. This is important for practical SDBM usage. Indeed, if the decision zones would maintain similar sizes and shapes but wildly change positions, interpreting the maps would be hard. Moreover, such position changes would confuse the user as they would imply instability of the underlying classification model. Secondly, we see that the *confidence* of the maps changes only very little as with the dataset changes. This is a desirable result that confirms indirectly SDBM's stability, as follows: Small changes of the confidence indicate that the trained classifiers for the various changed datasets $\pi(D, \sigma_i)$ behave *similarly* to the classifier trained on the unchanged dataset. Since these classifiers are similar, their decision maps also should be similar – and this is what we observe in the above-mentioned images.

Aggregated change maps: Visualizing individual SDBM maps for increasing amounts of change can be difficult as each such image needs to be compared with the original map (for the unchanged dataset). This becomes even harder to do when one wants to consider more than a few sample values of the change amount – which is useful when one wants to discern a clearer *trend* in terms of visual (map) change *vs* data change. To address this, we propose two ways to *aggregate* multiple SDBM maps, as follows. Consider all maps S_i computed multiple values σ_i, $1 \leq i \leq N$, for a single change type π. We compute a single aggregated map by analyzing, at each pixel location $\mathbf{p}$, the label values f_i of the N images S_i at location $\mathbf{p}$, using a 'hard voting' procedure. The color assigned to the aggregated map at $\mathbf{p}$ will map the label appearing most frequently in the set $\{f_1, \ldots, f_n\}$ at that location. We also set the luminance of the aggregated map at $\mathbf{p}$ to the fraction of the N maps that have 'voted' for this value.

Figure D.11 shows SDBM maps for the same classifiers and datasets as discussed above, aggregated using hard voting for each change type. We do not aggregate different change

types together as we believe this would be confusing to interpret. For each change type, we use $N = 10$ different change values σ_i, sampled uniformly between the minimum and maximum values used earlier to generate Figs. D.8, D.9, and D.10. We interpret the images in this figure as follows: Bright colored areas indicate *stable* decision zones which do not change as the classifier's training set is perturbed. Darker areas indicate decision zones that change as the training set is perturbed – the darker the area, the more that decision zone changes during the applied data changes. The images exhibit a 'color banded' structure because there are at most $K = 10$ possible brightness levels, where K is the number of classes of the problem. These range from fully bright, indicating 100% agreement of all decision zones for all N trained classifiers, to a luminance of $1/K$, which indicates that the N classifiers are uniformly split into K groups each voting for a different label value. The brightness inside a decision zone has a similar gradient to that shown earlier in Figures D.8, D.9, and D.10 – that is, points close to a decision boundary appear darkest while points deep inside a decision zone are brightest. However, the meaning of the brightness is different: In the earlier image, brightness encoded *confidence* of classification at a given map location; in the aggregated map, brightness encodes *stability* of the decision maps at a given location. Confidence and stability are positively correlated – a map changes the least in areas where a classifier is very confident of its inference and conversely. However, the two visualizations convey different types of insights, as explained above.

The above discussion suggests that it would be useful to create an aggregated map that visualizes *both* confidence and stability. We do this by using, by analogy with the technique presented above, a 'soft voting' procedure, as follows. The color of each pixel in this soft-voting map is determined identically to the hard-voting map as the most frequent label in the set $\{f_1, \ldots, f_n\}$ at that location. However, we now set the brightness to depict the average value, at that location, of the confidences of the N classifiers whose SDBM maps we want to aggregate. This way, the soft-voting map depicts the overall confidence of the aggregated maps across all applied change levels σ_i.

Figure D.12 shows the SDBMs for the same classifiers and datasets as in Fig. D.11 aggregated with soft voting. The interpretation of the aggregated images in Fig. D.12 is different from those in Fig. D.11 – darker regions indicate now areas where the aggregated decision maps have overall low confidence. We see that such areas follow the decision zone borders in the aggregated maps, just as in the original, unperturbed, maps (shown in the leftmost column in Fig. D.12). We also see that the dark areas in Fig. D.12 correlate well with the dark areas in Fig. D.11. This tells us that, for the studied classifiers and datasets, the decision maps are less stable in areas where the classifiers are low confidence, and conversely, decision maps are *stable in areas of high classifier confidence*. The latter answers our question from Sec. D.1 positively.

D.4.4 Computational Scalability

We next study the scalability of SDBM and compare it to the original DBM method. For this, we created maps using synthetic Gaussian blobs datasets with 5 clusters, varying the dimensionality from 10 to 500, and varying the map size from 25^2 to 300^2 pixels. We did not use larger maps since the speed trends were already clear from these sizes, with DBM getting considerably slower than SDBM. We used this synthetic data approach, rather than

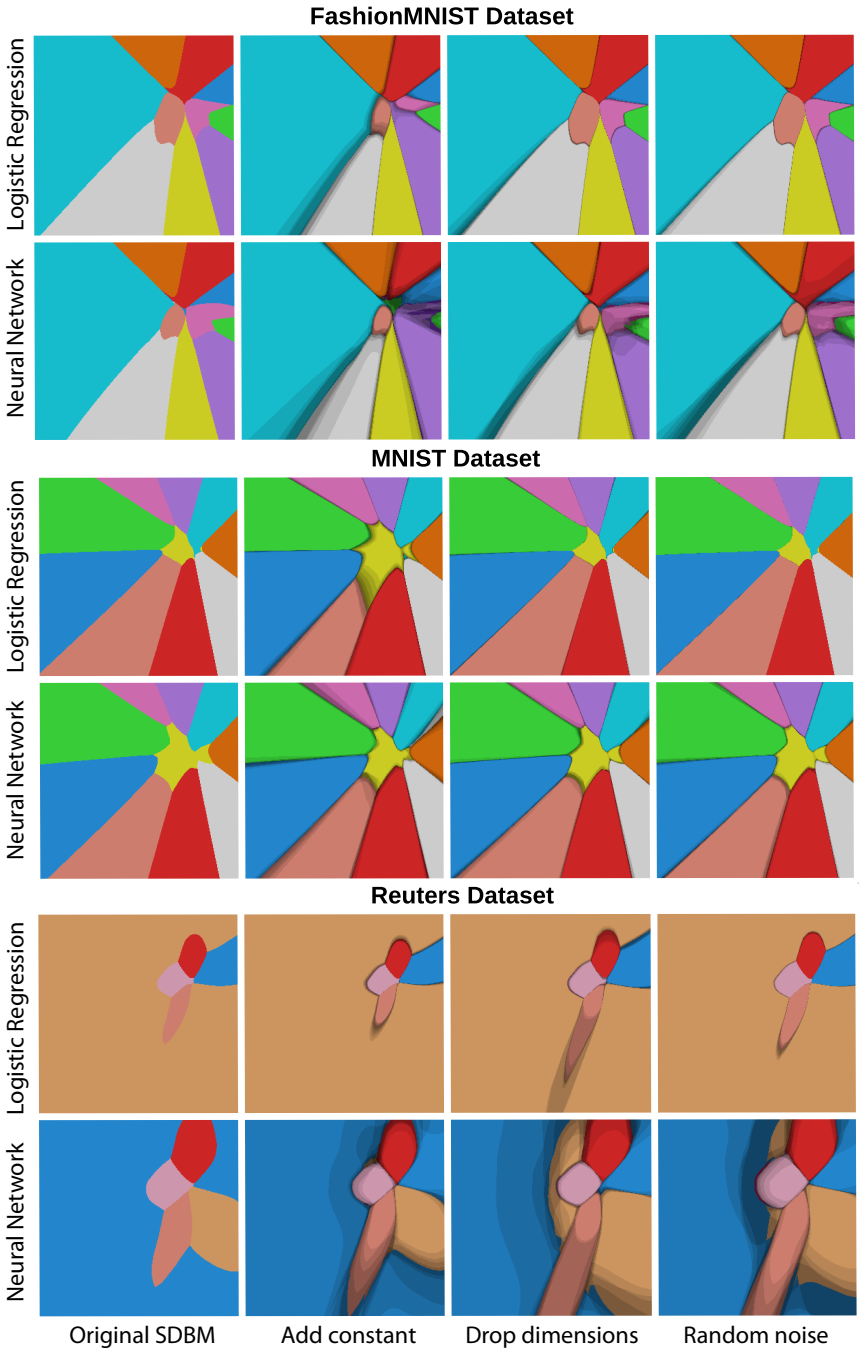


Figure D.11: *Aggregated decision maps computed by hard voting for two classifiers trained with three change types, ten change amounts, on three datasets. The leftmost column shows the decision maps of the unchanged datasets.*

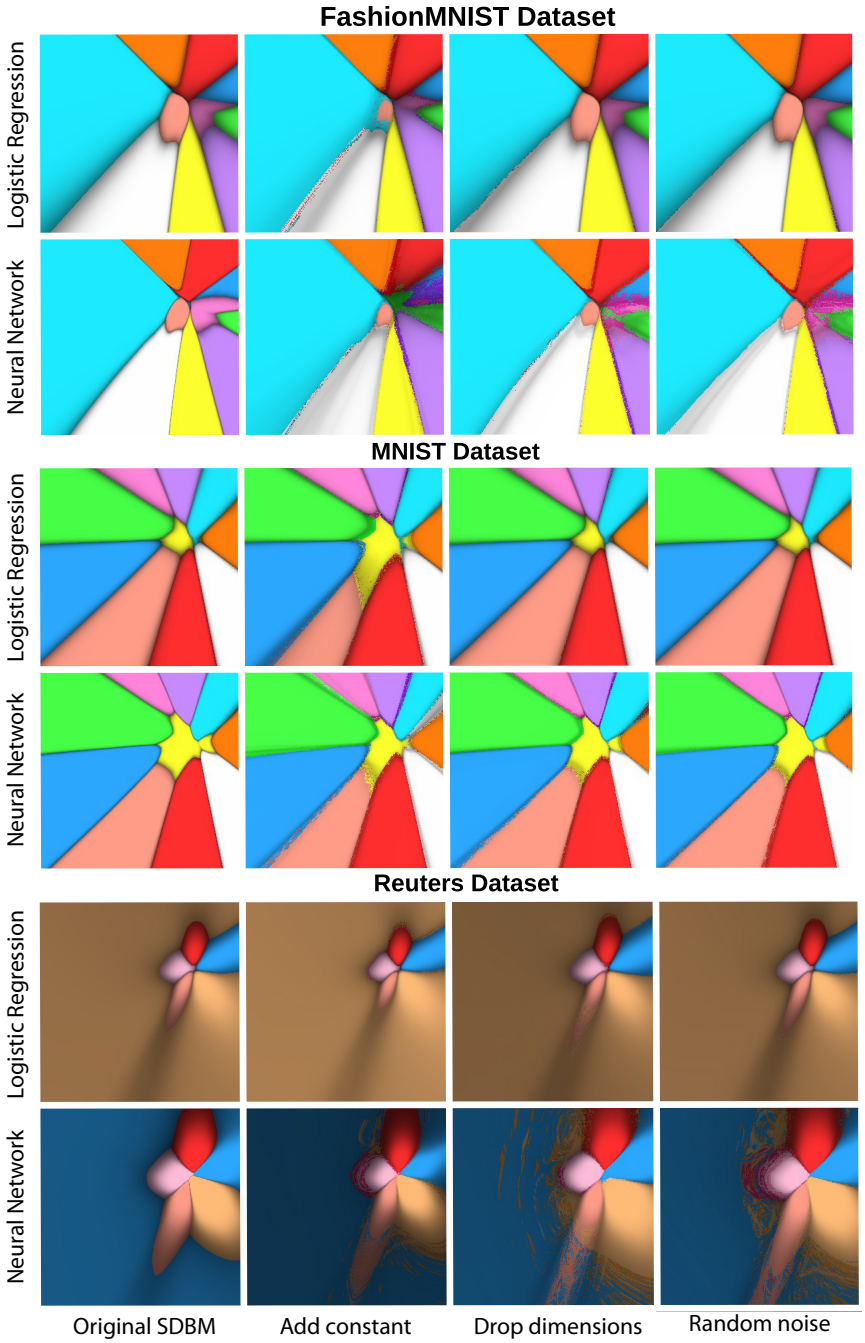


Figure D.12: *Aggregated decision maps computed by soft voting for two classifiers trained with three change types, ten change amounts, on three datasets. The leftmost column shows the decision maps of the unchanged datasets.*

real-world datasets, as it allowed us to control the data dimensionality in a fine-grained way, which is the key factor influencing computing speed for both methods. Note that the number of samples does not heavily influence DBM and DBM's computing times. Both methods only need to project a dataset *once* after which they need to apply the inverse projection P^{-1} for each map *pixel*. For typical situations, the pixel count is far larger than the sample count, which makes the former dominate the map computation cost.

Figure D.13 shows the running times of both methods as a function of both the grid size (horizontal axis) and dataset dimensionality (different-color lines). We see that DBM's runtime increases quickly with dimensionality, taking about 5 minutes to create a 300^2 map for the 500-dimensional dataset. In contrast, SDBM is over an order of magnitude faster, taking roughly 7 seconds to run for the same dataset. Also, we see that SDBM's speed depends far less on the data dimensionality, whereas this is a major slowdown factor for DBM. All in all, this shows that SDBM is significantly more scalable than DBM. This can be explained by the fact that SSNP, which underlies SDBM, *jointly* trains both the direct and inverse projections by deep learning. SDBM is GPU-accelerated, linear in the sample and dimension counts both for training and inference and does not need to use different resolutions and sampling tricks for accelerating the 2D to nD mapping (see Sec. D.3). In contrast, DBM uses UMAP and iLAMP for the direct, respectively, inverse projections (as mentioned earlier). None of these techniques is GPU-accelerated. Also, while UMAP is close to linear in the sample count and dimensionality, iLAMP is superlinear in dimensionality and sample count. Together, these aspects make DBM significantly slower than SDBM.

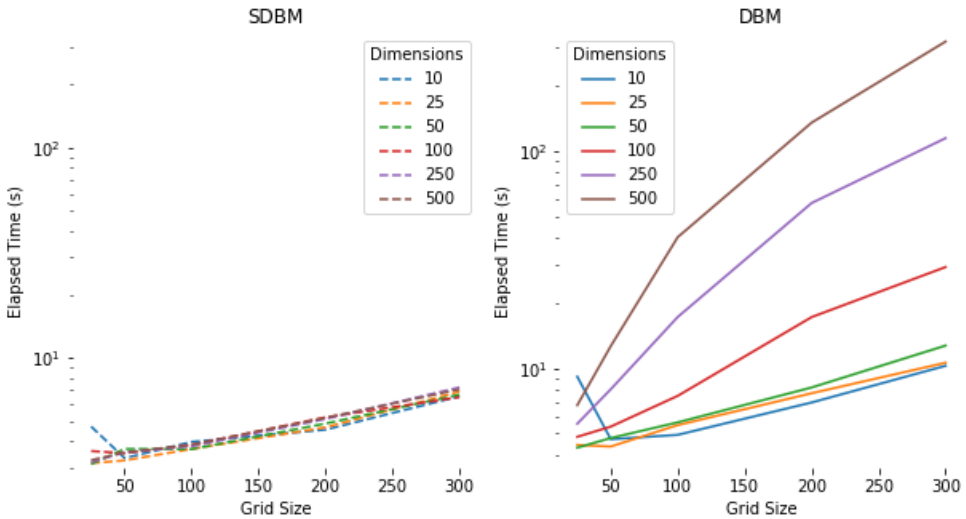


Figure D.13: *Computation time to create decision maps of increasing size by SDBM and DBM using synthetic datasets of varying dimensionality. The vertical axis is on a logarithmic scale.*

D.4.5 Implementation details

All experiments presented above were run on a dual 8-core Intel Xeon Silver 4110 with 256 GB RAM and an NVidia GeForce RTX 2070 GPU with 8 GB VRAM. Table D.1 lists all

open-source software libraries used to build SDBM and the other tested techniques. Our implementation and all datasets used in this work are publicly available at [190].

Technique	Software used publicly available at	Reference
SSNP	http://keras.io (TensorFlow backend)	[30]
UMAP	http://github.com/lmcinnes/umap	[125]

Table D.1: *Software packages used in the evaluation.*

D.5 Discussion

We discuss how our technique performs for the criteria C1-C5 introduced in Sec. D.1.

Quality (C1): SDBM can create maps that show classifier decision boundaries very clearly, and, most importantly, much clearer than the maps created with the original DBM. For the same dataset-classifier combinations, SDBM's maps show significantly less noise, more compact decision zones, and smoother decision boundaries, than DBM. These results are in line with what we expect for dataset-classifier combinations for which we have ground truth knowledge about their decision zones and boundaries (see Fig. D.5 and related text). As such, we conclude that SDBM captures the actual decision zones better than DBM.

Scalability (C2): SDBM is an order of magnitude faster than DBM. Since SDBM scales linearly in the number of observations during inference/drawing, and it is end-to-end GPU-accelerated, it can generate maps having hundreds of thousands of pixels in a few seconds, which makes it practical for handling large datasets and rendering highly detailed decision maps.

Ease of use (C3): SDBM produces good results with practically no need for hyperparameter tuning. In more detail, there are only two such hyperparameters. First, there is the number of epochs used to train SSNP to construct the direct and inverse projections. Following [46], we set this to a default value of 10. Secondly, there is the resolution R of the output decision map. Note, however, that this parameter does not influence the *stability* of the method, but only the level of detail of the produced decision maps. As Fig. D.13 shows, the computation time is linear with the output resolution – which is expected, since SDBM needs to execute an inverse projection and classifier model inference per output pixel, and both these operations have a constant cost. We also note that compared to SDBM, the parameter-setting of DBM is far more complex. Briefly put, DBM is very slow and as such uses a low resolution. However, this implies a sparse sampling of the input data space. To counter this, DBM creates multiple randomly-distributed 2D sample points in each grid cell, backprojects these, classifies the backprojections, and aggregates the resulting labels to compute the final pixel color. To obtain good results, DBM requires careful tuning of the number of such sample points inside every pixel (for more details, we refer to [164]). SDBM does not have any of these problems as it can directly construct high-resolution images.

Genericity (C4): As for the original DBM method, SDBM is agnostic to the nature and dimensionality of the input data, and to the classifier being visualized. We show that

SDBM achieves high quality on datasets of different natures and coming from a wide range of application domains and with classifiers based on quite different algorithms. As such, SDBM does not trade any flexibility that DBM already offered, but increases quality, scalability, and ease of use, as explained above.

Stability (C5): We have described a range of experiments that show that SDBM is stable to changes in the training dataset of the classifiers it visualizes. For quite significant changes amounting to additive bias up to 30% of the data range, dropping up to 30% dimensions, and adding noise up to a 0.1 standard deviation, SDBM creates decision maps that differ visually little from the ones for the unperturbed datasets. Additionally, we showed that the type of perturbation does not significantly influence the amount of change in the produced decision maps. The maps are also stable in the sense that decision zones are plotted in (roughly) the same areas of the map regardless of the perturbation. Moreover, the overall visual appearance of the decision maps, *e.g.* in terms of decision boundary smoothness and island-like small-scale disconnected regions, is not influenced by perturbing the training set. Interestingly, the stability of SDBM is far higher in the presence of similar input perturbations than that of a related NNP technique that also uses deep learning for projecting high-dimensional data [19]. Interpreting SDBM's stability needs, potentially, a few extra words: What we showed, is that classifiers trained by changed training data produce similar decision maps to classifiers trained by unperturbed data. We argue that this makes sense in the formal definition of stability of a function (SDBM in our case) since we change the input of that function which, in our case, is the trained classifier f (see Sec. D.2). In turn, f's behavior depends solely on its training set. One could argue that a trained classifier also depends on the test set and that such a test set should be varied as well to assess the classifier. While this is true, changing a test set does not change the formal decision zones or boundaries of a trained classifier model, hence it does not change its SDBM visualization. As such, the main variable we can change to assess SDBM's stability is the classifier's training set. Importantly, we also showed that SDBM is more stable than its predecessor, DBM – which can be ascribed to the deterministic nature of the deep-learned direct and inverse projections in SSNP as opposed to the direct and inverse projections (UMAP, respectively iLAMP) used by DBM.

Limitations: SDBM shares a few limitations with DBM. First and foremost, it is hard to *formally* assess the quality of the decision maps it produces for dataset-classifier combinations for which we do not have clear ground truth on the shape and position of their decision zones and boundaries. Our work showed that SDBM produces results fully in line with known ground truth for such simple situations. However, this does not formally guarantee that the same is true for more complex datasets and any classifiers. Finding ways to assess this is an open problem to be studied in future work. Secondly, the interpretation of the SDBM maps can be enhanced. Examples shown in this paper outlined how such maps can help to find out whether a trained classifier can generalize well and how far, from its training set, and how different classifier-dataset combinations can be compared by such maps. Yet, such evidence is qualitative. A more formal study showing how users actually *interpret* such maps to extract quantitative information on the visualized classification problems is needed. Finally, while our stability study outlined that SDBM is surprisingly stable to significant variations of a classifier's training set, a full understanding of such a stability concept needs further work. For instance, one would like to test SDBM stability

in the presence of varying the training hyperparameters of the classifier. Also, in such a stability study, one would arguably want to use more data-domain-dependent change types than the generic ones that we explored in Sec. D.4.3.

Applications: Decision maps are not an end by themselves but a *tool* that allows ML engineers to study a given classification model and, in the case, the model performs poorly, obtain insights on how to improve it. The current paper has shown that SDBM can produce high-quality decision maps that have all the requirements needed for their application in practice (as discussed above). As such, SDBM is now ready to be deployed in concrete scenarios. In this respect, we believe that *imaging* applications are one of the domains where SDBM would best fit. Examples of ML applications in this domain include transfer learning for image classification [159], understanding important features for classification of histopathology images [28], analysis of misclassification results in cell image classification [110], microorganism image segmentation [217], and data augmentation for cancer image classification [158].

What all these applications have in common – from a technical perspective – is the (a) usage of complex multi-stage, deep learning, models to (b) analyze image data. As such, fine-tuning the respective models is a complex task, for which projections are typically used. We believe that using decision maps can significantly augment the insights shown by projections as one can effectively see how decision zones and decision boundaries relate to the training-set and test-set points. Moreover, since the targeted data are *images*, one can effectively display such images *e.g.* as users move a tooltip over the decision map image. This can show not only which existing images fall in specific decision zones (or close to decision boundaries), but also synthesize *new* images, via backprojection, that fall in the 'empty' spaces between existing samples. These synthesized images can next help understand and improve how the trained classification models work, *e.g.*, by user-supervised data augmentation.

D.6 Conclusion

In this paper, we have presented and explored the behavior of SDBM, a new method for producing classifier Decision Boundary Maps. Compared to the only other similar technique we are aware of – DBM – our method has several desirable characteristics. First, it can create decision maps that are far smoother and less noisy than those created by DBM and also match the known ground truth of the visualized classification problems far better than DBM, therefore allowing users to interpret the studied classifiers with less confusion. Secondly, SDBM is about an order of magnitude faster than DBM due to its joint computation of direct and inverse projections on a fixed-resolution image by deep learning. Thirdly, SDBM has virtually no parameters to tune (apart from the resolution of the desired final image) which makes it easier to use than DBM. Finally, in addition to our earlier work [134], we have presented a comprehensive study of the stability of SDBM in the presence of several types and amounts of changes of the examined classifiers' training sets. Our study shows that SDBM is quite stable for a wide range of such changes, irrespective of the classifier used or the nature of the training set. We have also presented new methods to compactly visualize SDBM's stability using aggregated maps which summarize the

changes in several SDBM maps.

Future work can target several directions. A very relevant direction is the generation of maps for multi-output classifiers, *i.e.*, classifiers that can output more than a single class for a sample. Secondly, we consider organizing more quantitative studies to gauge which are the interpretation errors that SDBM maps generate when users consider them to assess and/or compare the behavior of different classifiers, which is the core use-case of decision maps. Thirdly, proposing new methods to measure and visualize SDBM's stability can not only help to increase trust in this method but also help to understand the stability of other regressors for changing high-dimensional data [198, 19, 44]. In this sense, proposing formal metrics to characterize SDBM's stability, akin to measuring directional derivatives of a multi-variable function, or sensitivity analysis [172], is a key direction to follow. Last but not least, using SDBM to understand and improve existing complex classification models, especially for image data, is an important direction we aim to pursue.

Acknowledgments

This study was financed in part by FAPESP grants 2015/22308-2, 2017/25835-9, and 2020/13275-1, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Appendix E

Appendix: SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers

This chapter is partially based on the paper of the same name [136].

E.1 Introduction

In recent years, Machine Learning (ML) techniques have become very popular in many fields to support pattern recognition and predictive modeling. Despite their popularity, the inner workings of trained ML models are hard to explain, which can hamper their adoption where transparency and accountability of inference are required [163]. For Deep Learning (DL) models, explainability is an even harder concern, as such models have millions of parameters that contribute jointly to the generation of many levels of latent features [58].

For the more specific case of ML *classifiers*, several approaches for model explanation have been proposed, using variable importance [112], locally interpretable models [163], and a variety of visualization-based techniques [160, 161]. Garcia *et al.* [58] recently presented a survey of visual techniques oriented towards the explanation of DL models.

A particular visual explanation technique in this set is the *Decision Boundary Map* (DBM) [164]. DBM extends classical multidimensional projections [131] by filling in the gaps between projected points from a labeled dataset used to train a classifier with synthesized, classified, data points. This effectively creates a 2D dense image that shows how the classifier partitions its high-dimensional data space into per-class decision zones. DBM is, to our knowledge, the first technique that succeeds in visually depicting such classifier decision zones for any classifier. However, DBM has several limitations – it is slow, sensitive to parameter settings, and produces noisy visualizations from which it is hard to understand which are the shapes, topologies, and extents of the decision zones.

In this paper, we propose Supervised Decision Boundary Maps (SDBM), a *supervised* technique, which improves DBM in four key directions:

Quality (C1): SDBM produces decision maps that allow for a clearer, and far less noise-prone, visual separation of a higher number of decision zones from real-world, complex, datasets, than DBM;

Scalability (C2): SDBM is GPU accelerated and has linear complexity in the number of samples and dimensions, allowing the creation of megapixel maps in a few seconds on commodity hardware, in contrast to the minutes needed by DBM;

Ease of use (C3): SDBM produces good results with minimal or no parameter tuning;

Genericity (C4): SDBM can construct decision boundaries for *any* single-value classifier.

We structure this paper as follows: Section E.2 discusses related work on classifier visualization. Section E.3 details our SDBM method. Section E.4 presents the results that support our contributions outlined above. Section E.5 discusses our method. Finally, Section E.6 concludes the paper.

E.2 Background

We next introduce the notation used in this paper. Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i \in \mathbb{R}, 1 \leq i \leq n$ be an n-dimensional (nD) real-valued, labeled observation, and $D = \{\mathbf{x}_j\}$, $1 \leq j \leq N$ be a dataset with N samples, *e.g.*, a table with N rows (samples) and n columns (dimensions). Let $C = \{c_k\}$, $1 \leq k \leq K$ be the set of K class labels used in D. Let $\mathbf{y} = \{y_j | y_j \in C\}$, $1 \leq j \leq N$ be the class labels associated with each sample $\mathbf{x}_j$.

A classifier is a function

$$f : \mathbb{R}^n \rightarrow C, \tag{E.1}$$

that maps between data samples and class labels and is learned using a training algorithm over the dataset D. Logistic Regression [36], SVM [35], Random Forests [20], and Neural Networks are examples of ML algorithms.

A Dimensionality Reduction (DR), or projection, technique is a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q, \tag{E.2}$$

where $q \ll n$, and typically $q = 2$. The projection $P(\mathbf{x})$ of a sample $\mathbf{x} \in \mathbb{R}^n$ is a qD point $\mathbf{p} \in \mathbb{R}^q$. Projecting a set D yields thus a qD scatterplot, which we denote next as $P(D)$. The inverse of P, denoted $P^{-1}(\mathbf{p})$, maps a qD point $\mathbf{p}$ to the high-dimensional space $\mathbb{R}^n$.

Decision Boundary Maps: Given a classifier f, a Decision Boundary Map (DBM) is a 2D image that shows a representation of how f partitions the $\mathbb{R}^n$ data space into decision zones. A *decision zone* is a set of 2D points $\mathbf{p}$ for which $f(P^{-1}(\mathbf{p})) = \{c_k | c_k \in C\}$ – that is, map high-dimensional points which are classified by f to the same class c_k. Class labels c_k are color-coded in the decision maps. Decision zones are separated by *decision boundaries*,

which are pixels **p** whose labels (colors) differ from those of at least one 8-neighbor pixel in the DBM. The DBM shows, among other things, how the high-dimensional space is effectively partitioned by f into decision zones, how large these zones are, how they are adjacent to each other, and how smooth the decision boundaries between classes are [164]. This gives insights into whether the classifier f has overfitted the training data, how well separated the data is, *i.e.*, and how difficult is the task of partitioning the high-dimensional space to obtain good classification accuracy. DBMs are a step forward atop of the key observation in Rauber *et al.* [160], which showed how multidimensional projections aid in deciding whether a high-dimensional dataset is easily classifiable or not. Simply put, DBMs support the same task but provide more information by 'filling in' the white gaps between the points of a 2D scatterplot $P(D)$ by extrapolating the classifier f.

The DBM technique, as introduced by Rodrigues *et al.* [164], relies heavily on direct and inverse projections, to create the mappings P and P^{-1}. The direct mapping is used to create a 2D scatterplot $P(D)$ from the dataset D. The inverse mapping P^{-1} creates synthetic nD data points from all pixels **p** in the 2D bounding box of $P(D)$. These points $P^{-1}(\mathbf{p})$ are then classified by f, and colored by the assigned class labels $f(P^{-1}(\mathbf{p}))$. While this approach is conceptually sound, it has two main issues: (1) The inverse projection technique P^{-1} used, iLAMP [7], scales poorly to the hundreds of thousands of points a dense pixel map has. This was addressed in [164] by subsampling the 2D projection space into cells larger than one pixel, sampling a few 2D pixels from each cell, and next deciding the label (and thus color) of each cell by majority voting on the classification of the inverse-projections of these samples. This subsampling creates artifacts that are visible in the highly jagged boundaries of the decision zones. (2) Since the direct projections P used are unsupervised, outliers in the data D can generate 'islands' of pixels having a different label (and thus color) than their neighbors. This creates spurious decision zones and decision boundaries which next make the resulting DBMs hard to analyze by the user, in particular when the problem has several classes.

Dimensionality reduction: Both the original DBM technique and our improved version SDBM rely heavily on Dimensionality Reduction (DR) techniques. Many DR techniques have been proposed over the years, as reviewed in various surveys [72, 119, 42, 184, 109, 38, 210, 131, 50]. Below we describe a few representative ones, referring to the aforementioned surveys for a more thorough discussion.

Principal Component Analysis [86] (PCA) is one of the most popular DR techniques for many decades, being easy to use, easy to interpret, and scalable. However, PCA does not perform well for data of high intrinsic dimensionality and is thus not the best option for data visualization tasks.

The Manifold Learning family of methods contains techniques such as MDS [194], Isomap [188], and LLE [167], which aim to capture nonlinear data structure by mapping the high-dimensional manifold on which data is located to 2D. These methods generally yield better results than PCA for visualization tasks, but do not scale well computationally, and also yield poor results when the intrinsic data dimensionality is higher than two.

The SNE (Stochastic Neighborhood Embedding) family of methods, of which the most popular member is t-SNE [118], are very good for visual tasks due to the visual

cluster segregation they produce. Yet, they can be hard to tune [204], and typically have no out-of-sample capability. Several refinements of t-SNE improve speed, such as tree-accelerated t-SNE [115], hierarchical SNE [154], and approximated t-SNE [153], and various GPU accelerations of t-SNE [155, 25]. Uniform Manifold Approximation and Projection (UMAP) [125], while not part of the SNE family, generates projections with comparable quality to t-SNE, but much faster, and with out-of-sample capability.

All the above projection techniques work in an *unsupervised* fashion, by using information on distances between data points in D to compute the projection $P(D)$. Recently, [47] proposed Neural Network Projection (NNP) to learn the projection $P(D)$, computed by any user-selected technique P, from a small subset $D' \subset D$, using a deep learning regressor. While slightly less accurate than the original P, this technique is computationally linear in the size and dimensionality of D, has out-of-sample capability, is stable, and is simple to implement and parameter-free. The same idea was used by NNInv [49] to learn the inverse mapping P^{-1}. These approaches were next extended by Self-Supervised Network Projection (SSNP) [46], which can be used either in a *self-supervised* fashion, by computing pseudo-labels by a generic clustering algorithm on D, or in a *supervised* fashion (similar to NNP), using ground truth labels $\mathbf{y}$ coming with D. SSNP's supervised mode is key to the creation of our proposed SDBM for the following reasons:

- SSNP provides good cluster separation by partitioning the data space D as a classifier would do, which is closely related to the original goal of DBM;
- SSNP provides both the direct and inverse mappings (P and P^{-1}) needed by DBM to generate synthetic data points;
- SSNP is GPU-accelerated, which makes SDBM one to two magnitude orders faster than DBM.

E.3 Method

We next describe our proposed SDBM technique and how it is different from its predecessor, DBM (see also Fig. E.1 for step-by-step details of the SDBM pipeline):

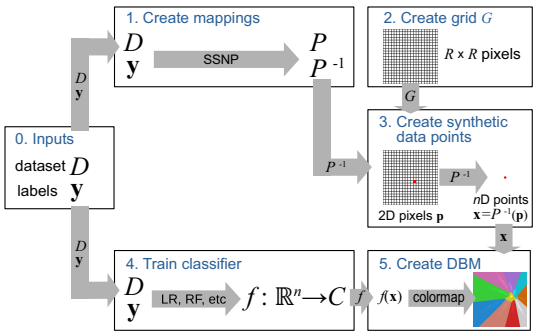


Figure E.1: *SDBM pipeline.*

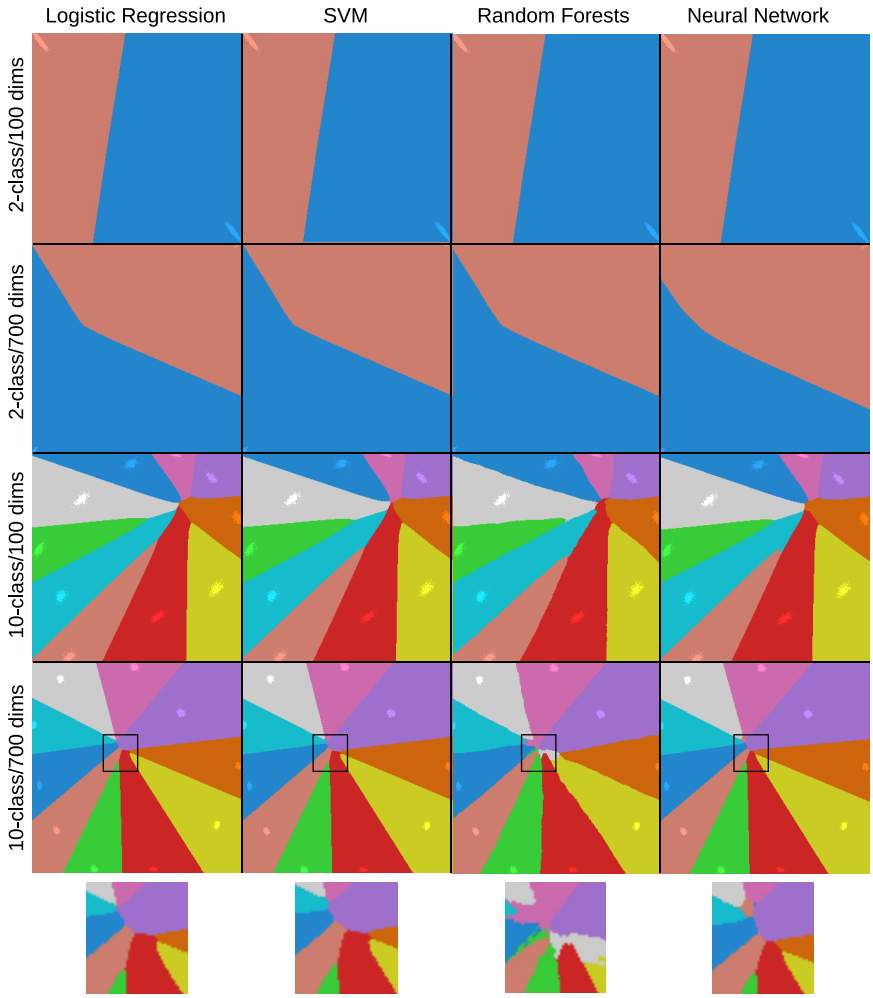


Figure E.2: *Decision Boundary Maps (DBMs) created with SDBM for several classifiers (columns) and synthetic datasets (rows). Lighter pixels represent training samples from the datasets D.*

1. Train classifier: Train the classifier f to be visualized using the dataset D and its class labels $\mathbf{y}$. This step is identical to DBM. Any single-class-output classifier $f : \mathbb{R}^n \to C$ can be used generically, *e.g.*, Logistic Regression (LR), Random Forests (RF), Support Vector Machines (SVM), or neural networks.

2. Create mappings: Train SSNP to create the direct and inverse projections P and P^{-1} based on D and $\mathbf{y}$. This step is fundamentally different from DBM which accepts any user-selected projection P and then constructs P^{-1} by deep learning the 2D to nD mapping using deep learning [49] (see also Sec. E.2). This asymmetric design of DBM makes P^{-1} significantly differ from the mathematical inverse of P for several points $\mathbf{x}$, *i.e.*, $P^{-1}(P(\mathbf{x})) \neq \mathbf{x}$, which is visible as jagged decision boundaries and noise-like small islands scattered all over the dense maps (see Fig. E.5 later on). As we shall see in Sec. E.4, the *joint* computation of P and P^{-1} used by SDBM significantly reduce such artifacts.

3. Create 2D grid: Create an image $G \subset \mathbb{R}^2$. This is different from DBM which uses subsampling of the 2D projection space (see Sec. E.2). In detail, SDBM uses the full resolution of G to compute $P(D)$ but then evaluates P^{-1} on a subsampled version thereof. In our case,

both P and P^{-1} use the full resolution image G. For the experiments in this paper, we set the resolution of G to 300^2 pixels.

4. Create synthetic data points: Use the trained P^{-1} to map each pixel $\mathbf{p} \in G^2$ to a high-dimensional data point $\mathbf{x} \in \mathbb{R}^n$. This is similar to DBM, except for the use of a dense pixel grid and jointly-trained P and P^{-1} (see above).

5. Color pixels: Color all pixels $\mathbf{p} \in G$ by the values of $f(P^{-1}(\mathbf{p}))$, *i.e.*, the inferred classes of their corresponding (synthetic) data points, using a categorical color map. In this paper we use the 'tab20' color map [79]. This is the same as DBM.

6. Encode classifier confidence (optional): For classifiers f that provide the probability of a sample $\mathbf{x}$ belonging to a class c_k, we encode that probability in the brightness of the pixel $\mathbf{p}$ that back-projects to $\mathbf{x}$. The lower the confidence of the classifier is, the darker the pixel appears on the map, thereby informing the user of the confidence of the decision zone in that area. This is the same as DBM.

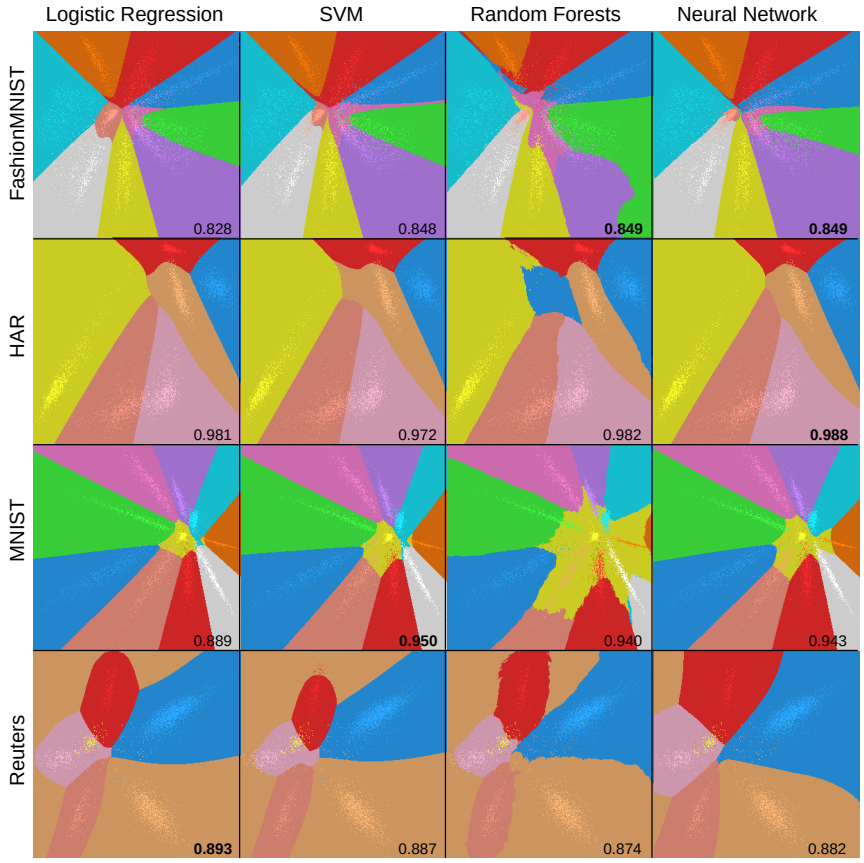


Figure E.3: *Decision Boundary Maps (DBMs) created with SDBM for several classifiers (columns) and real-world datasets (rows). Numbers inside each map indicate test accuracy obtained by each classifier, bold indicating top performers. Lighter pixels represent training samples from the datasets D.*

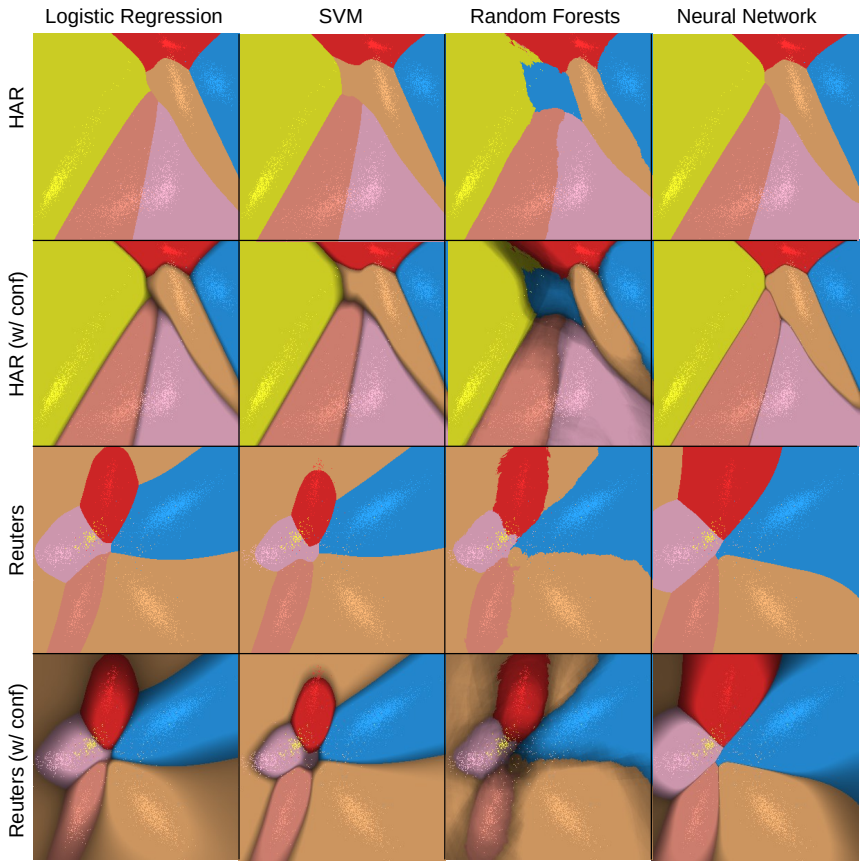


Figure E.4: *Decision Boundary Maps created with SDBM for several classifiers, HAR, and Reuters datasets. Columns show different classifiers. Rows show different datasets, with and without confidence encoded into brightness.*

E.4 Results

We next present the results that support our claims regarding SDBM. First, we show how our method performs with synthetic data, where perfect class separation is possible by most classifiers (Sec. E.4.1). This allows us to verify how the technique performs under a controlled setting where we know the 'ground truth' shapes of the decision zones. Next, we show how SDBM performs on more complex real-world datasets and additional classifiers (Sec. E.4.2) and also how it compares with DBM. This supports our claim that our technique can be generically used and that it improves quality *vs* DBM. We next show how SDBM compares to the original DBM speed-wise, thereby supporting our claims of improved scalability (Sec. E.4.3). Finally, we provide full implementation details for SDBM (Sec. E.4.4).

E.4.1 Quality on Synthetic Datasets

We assess how SDBM performs in a controlled situation where the ground truth is known, *i.e.*, datasets with clear class separation and known shapes of the expected decision zones. The datasets contain synthetic Gaussian blobs with 5000 samples, with varied dimensionality (100 and 700), and varied number of classes (2 and 10). We used

four different classifiers, namely Logistic Regression, SVM (with an RBF kernel), Random Forests (200 estimators), and a Neural Network (multi-layer perceptron having 3 layers of 200 units).

Figure E.2 shows the maps created using SDBM for all the different classifiers and dataset combinations. Decision zones are categorically colored. Projected samples in $P(D)$ are drawn colored also by their class, but slightly brighter, to distinguish them from the maps. We see that the decision zones are compact and with smooth boundaries, as expected for such simple classification problems. They enclose the Gaussian blobs with the same respective labels – *e.g.*, the red and blue zones for the 2-class, 100-dimensional dataset in Fig. E.2, top row, contain two clusters of light red, respectively light blue, projected points. We also see that the maps for Logistic Regression show almost perfectly straight boundaries, which is a known fact for this classifier. In contrast, the more sophisticated classifiers, such as Random Forests and Neural Networks, create boundaries that are slightly more complex than the others for the most complex dataset (Fig. E.2, bottom row, at the center of the maps for those classifiers).

E.4.2 Quality on Real-World Datasets

We next show how SDBM performs on real-world datasets. These datasets are selected from publicly available sources, matching the criteria of being high-dimensional, reasonably large (thousands of samples), and having a non-trivial data structure. They are also frequently used in ML classification evaluations and projection evaluations.

FashionMNIST [209]: 70K samples of $K = 10$ types of pieces of clothing, rendered as 28x28-pixel gray scale images, flattened to 784-element vectors. We also use a subset of this dataset containing only two classes, namely *Ankle Boot* and *T-Shirt*, to provide an example of a problem where classes are more easily separable. This dataset was downsampled to 10K observations for all uses in this paper.

Human Activity Recognition (HAR) [9]: 10299 samples from 30 subjects performing $K = 6$ activities of daily living used for human activity recognition, described with 561 dimensions that encode 3-axial linear acceleration and 3-axial angular velocity measured on the subjects.

MNIST [101]: 70K samples of $K = 10$ handwritten digits from 0 to 9, rendered as 28x28-pixel gray scale images, flattened to 784-element vectors. This dataset was downsampled to 10K observations for all uses in this paper.

Reuters Newswire Dataset [191]: 8432 observations of news report documents, from which 5000 attributes were extracted using TF-IDF [173], a standard method in text processing. This is a subset of the full dataset which contains data for the $K = 6$ most frequent classes.

Figure E.3 shows the maps created by SDBM for these datasets, with the same types of classifiers used in Sec. E.4.1. Even though the current real-world datasets are considerably more complex and harder to separate into classes, the classifiers' decision boundaries are visible. Simpler classifiers (Logistic Regression and SVM) show decision zones that are more contiguous and have smoother, simpler, boundaries. More complex classifiers

(Random Forests and Neural Networks) show more complex shapes and topologies of the decision zones. In particular, the maps created for the Random Forest classifiers show very jagged boundaries. This can be a result of having an ensemble of classifiers working together.

Encoding classifier confidence: Figure E.4 shows maps created by SDBM with classifier confidence encoded as brightness, as described in Sec. E.3. This allows us to see how different classifiers model probability very differently, and thus produce different results. The added value of encoding confidence can be seen if we compare the first-*vs*-second, respectively third-*vs*-fourth, rows in Fig. E.4. The confidence-encoding maps show a smooth brightness gradient, dark close to the decision boundaries (where colors change in the images) and bright deep in the decision zones. The effect is slightly reminiscent of shaded cushion maps [206], *i.e.*, and it enhances the visual separation of the color-coded decision zones. More importantly, the shading gradient effectively shows how confidence increases as we go deeper into the decision zones for different classifiers: For example, for the HAR dataset, these shaded bands are quite thin for Logistic Regression and SVM, thicker and less informative for Random Forests, and extremely and uniformly thin for Neural Networks. This tells us that Neural Networks have an overall very high confidence everywhere (except very close to the decision boundaries); Logistic Regression and SVM are less confident close to the boundaries; and Random Forests have a higher variation of confidence over the data space. For Random Forests, we see that the darkest region falls in the area of the central blue decision zone and the top-right of the left yellow zone. These are precisely the areas where the map of this classifier significantly differs from those of all the other three classifiers. Hence, we can infer that the isolated blue decision zone that Random Forests created is likely wrong, as it is low confidence *and* different from what all the other three classifiers created in that area. For the Reuters dataset (Fig. E.4 bottom row), we see that all classifiers produced a beige region at the top left corner. The confidence information (brightness) shows us that all classifiers but one (SVM) treat this region as a low-confidence one. This can be explained by the total absence of training samples in that region. More importantly, this tells us that the behavior of SVM in this region is likely wrong.

Confidence visualization also serves in quickly and globally assessing the *overall quality* of a trained classifier. Consider *e.g.* the Reuters dataset (Fig. E.4 bottom row). Compared to all other three rows in Fig. E.4, the decision maps for this dataset are darker. This shows that this dataset is harder to *extrapolate from* during inference. Note that this is not the same as the usual testing-after-training in ML. Indeed, for testing, one needs to 'reserve' a set of samples unseen during training to evaluate the trained classifier on. In contrast, SDBM decision maps do not need to do this as they synthesize 'testing' samples on the fly via the inverse projection P^{-1}. Moreover, classical ML testing only gives a global or per-class accuracy. In contrast, SDBM gives a per-region-of-the-data-space confidence, encoded by brightness.

Comparison with the Original DBM: Figure E.5 shows maps created by SDBM side-by-side with maps created by the original DBM technique, using Logistic Regression, Random Forest, and k-NN classifiers, for three real-world datasets. In this experiment, we used UMAP [125] as the direct projection for DBM, and iLAMP [7] for the inverse projection,

respectively Several important observations can be made, as follows.

First, we see that the projections $P(D)$ of the same datasets are not the same with DBM and SDBM – compare the bright-colored dots in the corresponding figures. This is expected, since, as explained in Sec. E.3, DBM employs a user-chosen projection technique P, whereas SDBM *learns* P from the label-based clustering of the data, following the SSNP method (see Sec. E.3). Since the projections $P(D)$ of the same datasets differ for the two methods, it is expected that the overall shapes of the ensuing decision boundaries will also differ – see *e.g.* the difference between the nearly horizontal decision boundary between the blue and red zones for Random Forests with DBM for FashionMNIST (2-class) and the angled boundary between the same zones for the same classifier, same dataset, with SDBM (Fig. E.5, middle row, two leftmost images). For the relatively simple classification problem that FashionMNIST (2-class) is, this is not a problem. Both DBM and SDBM produce useful and usable renditions of the two resulting decision zones, showing that this classification problem succeeded with no issues.

When considering more difficult datasets (FashionMNIST 10-class or HAR), the situation is dramatically different: DBM shows *highly* noisy pictures, where it is even hard to say where and which are the actual decision zones. These images suggest that none of the three tested classifiers could correctly handle these datasets, in the sense that they would change decisions extremely rapidly and randomly as points only slightly change over the data space. This is *known* not to be the case for these datasets and classifiers. Logistic Regression has built-in limitations of how quickly its decision boundaries can change [164]. k-NN is also known to construct essentially a Voronoi diagram around the same-class samples in the nD space, partitioning that space into cells whose boundaries are smooth manifolds. DBM does not show any such behavior (Fig. E.5, third and fifth columns). In contrast, SDBM shows a far lower noise level and far smoother, contiguous, decision zones and boundaries. Even though we do not have formal ground truth on how the zones and boundaries of these dataset-classifier combinations look, SDBM matches better the knowledge we have on these problems than DBM.

E.4.3 Computational Scalability

We next study the scalability of SDBM and compare it to the original DBM method. For this, we created maps using synthetic Gaussian blobs datasets with 5 clusters, varying the dimensionality from 10 to 500, and varying the map size from 25^2 to 300^2 pixels. We did not use larger maps since the speed trends were already clear from these sizes, with DBM getting considerably slower than SDBM. Figure E.6 shows the running times of both methods as a function of both the grid size (horizontal axis) and dataset dimensionality (different-color lines). We see that DBM's runtime increases quickly with dimensionality, taking about 5 minutes to create a 300^2 map for the 500-dimensional dataset.

In contrast, SDBM is over an order of magnitude faster, taking roughly 7 seconds to run for the same dataset. Also, we see that SDBM's speed only *marginally* depends on the dimensionality, whereas this is a major slowdown factor for DBM. Concerning the number of samples, we see that both methods exhibit similar trends, with SDBM being closer to a linear trend than DBM. However, the slope of the SDBM graphs is smaller than those of DBM for the same dimensionality. All in all, this shows that SDBM

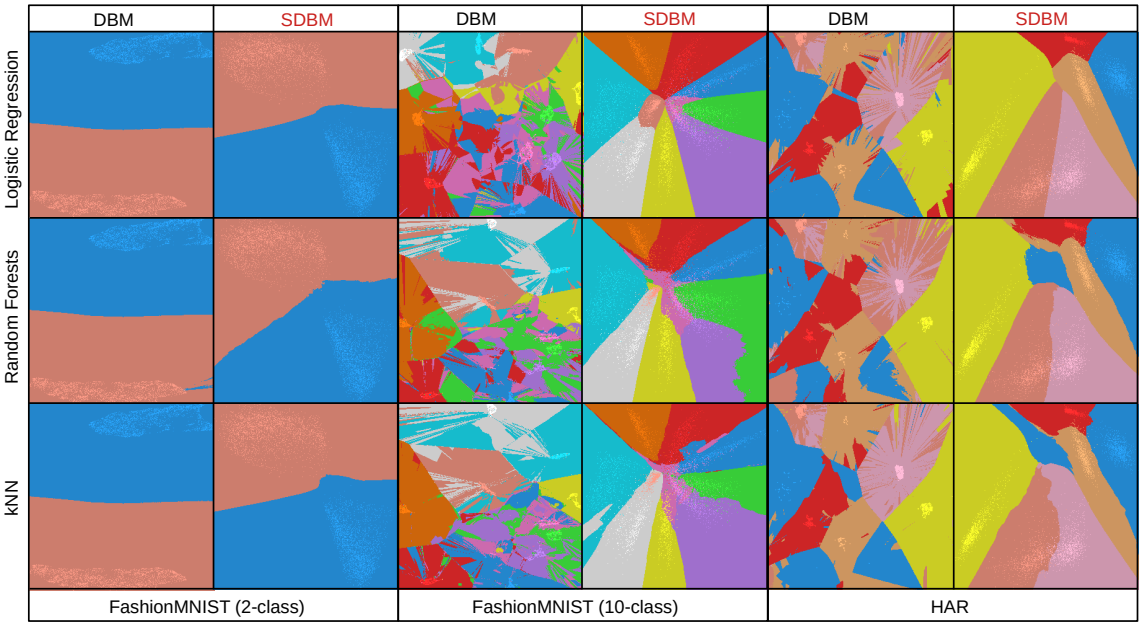


Figure E.5: *Comparison between SDBM and DBM using three different datasets and three classifiers.*

is significantly more scalable than DBM. This can be explained by the fact that SSNP, which underlies SDBM, *jointly* trains both the direct and inverse projections by deep learning. As this is GPU-accelerated, linear in the sample and dimension counts both for training and inference and does not need to use different resolutions and sampling tricks for accelerating the 2D to nD mapping (see Sec. E.3). In contrast, DBM uses UMAP and iLAMP for the direct, respectively, inverse projections (as mentioned earlier). None of these techniques is GPU-accelerated.

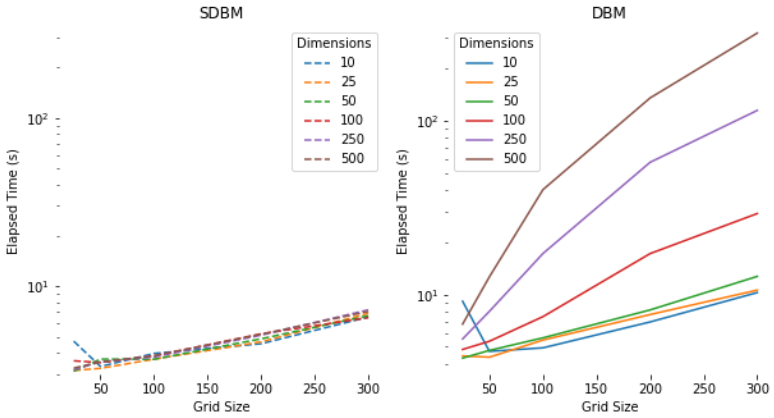


Figure E.6: *Plot showing the order of growth of time used to create maps of increasing size using DBM and SDBM, using synthetic datasets of varying dimensionality. The vertical axis is on a logarithmic scale.*

E.4.4 Implementation details

All experiments presented above were run on a dual 8-core Intel Xeon Silver 4110 with 256 GB RAM and an NVidia GeForce RTX 2070 GPU with 8 GB VRAM. Table E.1 lists all open-source software libraries used to build SDBM and the other tested techniques. Our implementation, plus all code used in this experiment, is publicly available at [190].

Technique	Software used publicly available at
SSNP	keras.io (TensorFlow backend) [30]
UMAP	github.com/lmcinnes/umap [125]

Table E.1: *Software packages used in the evaluation.*

E.5 Discussion

We discuss how our technique performs for the criteria laid out in Section E.1.

Quality (C1): SDBM can create maps that show classifier decision boundaries very clearly, and, most importantly, much clearer than the maps created with the original DBM. For the same dataset-classifier combinations, SDBM's maps show significantly less noise, more compact decision zones, and smoother decision boundaries, than DBM. These results are in line with what we expect for dataset-classifier combinations for which we have ground truth knowledge about their decision zones and boundaries (see Fig. E.5 and related text). As such, we conclude that SDBM captures the actual decision zones better than DBM.

Scalability (C2): SDBM is an order of magnitude faster than DBM. Since SDBM scales linearly in the number of observations during inference/drawing, and it is end-to-end GPU-accelerated, it can generate maps having hundreds of thousands of pixels in a few seconds, which makes it practical for handling large datasets and rendering highly detailed decision maps.

Ease of use (C3): SDBM produces good results with minimal tuning. The single performance-sensitive setting is the size of the map image. All maps in this paper have 300^2 pixels. As the figures show, this resolution is already sufficient for rendering detailed decision maps for all the tested dataset-classifier combinations. Compared to DBM, SDBM tuning is far simpler, as it does not require tuning of cell and sample sizes required by the former (for details of DBM tuning, we refer to [164]).

Genericity (C4): As for the original DBM method, SDBM is agnostic to the nature and dimensionality of the input data, and to the classifier being visualized. We show that SDBM achieves high quality on datasets of different natures and coming from a wide range of application domains and with classifiers based on quite different algorithms. As such, SDBM does not trade any flexibility that DBM already offered, but increases quality, scalability, and ease of use, as explained above.

Limitations: SDBM shares a few limitations with DBM. First and foremost, it is hard to *formally* assess the quality of the decision maps it produces for dataset-classifier combinations for which we do not have clear ground truth on the shape and position of their

decision zones and boundaries. Current testing shown in this paper has outlined that SDBM produces results fully in line with known ground truth for such simple situations. However, this does not formally guarantee that the same is true for more complex datasets and any classifiers. Finding ways to assess this is an open problem to be studied in future work. Secondly, the interpretation of the SDBM maps can be enhanced. Examples shown in this paper outlined how such maps can help to find out whether a trained classifier can generalize well and how far, from its training set, and how different classifier-dataset combinations can be compared by such maps. Yet, such evidence is qualitative. A more formal study showing how users interpret such maps to extract quantitative information on the visualized classification problems is needed.

E.6 Conclusion

We have presented SDBM, a new method for producing classifier Decision Boundary Maps. Compared to the only similar technique we are aware of – DBM – our method presents several desirable characteristics. First and foremost, it can create decision maps that are far smoother and less noisy than those created by DBM and also match the known ground truth of the visualized classification problems far better than DBM, therefore allowing users to interpret the studied classifiers with less confusion. Secondly, SDBM is about an order of magnitude faster than DBM due to its joint computation of direct and inverse projections on a fixed-resolution image. Finally, SDBM has virtually no parameters to tune (apart from the resolution of the desired final image) which makes it easier to use than DBM.

Future work can target several directions. We believe a very relevant one to be the generation of maps for multi-output classifiers, *i.e.*, classifiers that can output more than a single class for a sample. Secondly, we consider organizing more quantitative studies to gauge which are the interpretation errors that SDBM maps generate when users consider them to assess and/or compare the behavior of different classifiers, which is the core use-case for decision maps. Thirdly, we consider adapting SDBM to help the understanding of semantic segmentation models. Last but not least, the packaging of SDBM into a reusable library that can be integrated into typical ML pipelines can help it gain widespread usage.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and by FAPESP grants 2015/22308-2, 2017/25835-9 and 2020/13275-1, Brazil.

References

[1] ©2013-2023 Docker Inc. *Docker overview*. https://docs.docker.com/engine/docker-overview/. Retrieved: 2023-01-13. 2023 (cit. on p. 24).

[2] ©2023 Neo4j, Inc. *Neo4j Graph Platform*. https://neo4j.com/. Retrieved: 2023-01-13. 2023 (cit. on pp. 23, 29).

[3] ®Grab and OpenStreetCam Contributors. *Hello, Kartaview! - OpenStreetMap @ Grab*. https://blog.improveosm.org/en/hello-kartaview/. Access Ago 22nd, 2021. 2020 (cit. on pp. 1, 19, 25).

[4] Abien Fred Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018) (cit. on p. 13).

[5] Junaid Ahmad et al. "Vegetation encroachment monitoring for transmission lines right-of-ways: A survey". In: *Electric Power Systems Research* 95 (2013), pp. 339–352 (cit. on p. 19).

[6] Artur André Almeida de Macedo Oliveira and Roberto Hirata. "INACITY - INvestigate and Analyze a CITY". In: *SoftwareX* 15 (2021), p. 100777. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2021.100777. URL: https://www.sciencedirect.com/science/article/pii/S2352711021000911 (cit. on p. 23).

[7] E. Amorim et al. "iLAMP: Exploring high-dimensional spacing through backward multidimensional projection". In: *Proc. IEEE VAST*. 2012, pp. 53–62 (cit. on pp. 71, 86, 93, 97, 104, 123, 129).

[8] Dragomir Anguelov et al. "Google street view: Capturing the world at street level". In: *Computer* 43.6 (2010), pp. 32–38 (cit. on pp. 1, 19, 20).

[9] D. Anguita et al. "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine". In: *Proc. Intl. Workshop on Ambient Assisted Living*. Springer. 2012, pp. 216–223 (cit. on pp. 75, 102, 128).

[10] Alvaro Arcos-Garcia, Juan A Alvarez-Garcia, and Luis M Soria-Morillo. "Evaluation of deep neural networks for traffic sign detection systems". In: *Neurocomputing* 316 (2018), pp. 332–344 (cit. on p. 25).

[11] M. Aupetit. "Visualizing distortions and recovering topology in continuous projection techniques". In: *Neurocomputing* 10.7 (2007). Publisher: Elsevier, pp. 1304–1330 (cit. on pp. 94, 95).

[12] ©2023 Baidu. *Baidu map (Translated from chinese)*. https://map.baidu.com/. Last access 10/03/2023 (cit. on pp. 20, 28).

[13] Robert Baldock, Hartmut Maennel, and Behnam Neyshabur. "Deep learning through the lens of example difficulty". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 10876–10889 (cit. on p. 22).

[14] Martin Becker et al. “Robust dimensionality reduction for data visualization with deep neural networks”. In: *Graphical Models* 108 (2020), p. 101060 (cit. on pp. 69, 72).

[15] Adam Berland and Daniel A Lange. “Google Street View shows promise for virtual street tree surveys”. In: *Urban Forestry & Urban Greening* 21 (2017), pp. 11–15 (cit. on p. 20).

[16] Filip Biljecki and Koichi Ito. “Street view imagery in urban analytics and GIS: A review”. In: *Landscape and Urban Planning* 215 (2021), p. 104217. ISSN: 0169-2046. DOI: https://doi.org/10.1016/j.landurbplan.2021.104217 (cit. on pp. 19–21).

[17] Filip Biljecki and Koichi Ito. “Street view imagery in urban analytics and GIS: A review”. In: *Landscape and Urban Planning* 215 (2021), p. 104217 (cit. on p. 25).

[18] TK Boehmer et al. “Perceived and observed neighborhood indicators of obesity among urban adults”. In: *International journal of obesity* 31.6 (2007), p. 968 (cit. on p. 19).

[19] C. Bredius, Z. Tian, and A. Telea. “Visual Exploration of Neural Network Projection Stability”. In: *Proc. MLVis*. Eurographics. 2022 (cit. on pp. 95, 96, 107, 111, 117, 119).

[20] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001). Springer, pp. 5–32 (cit. on pp. 93, 101, 122).

[21] H. Butler et al. *The GeoJSON Format*. RFC 7946. Aug. 2016. DOI: 10.17487/RFC7946. URL: https://rfc-editor.org/rfc/rfc7946.txt (cit. on pp. 25, 26, 29, 31).

[22] Bill Yang Cai et al. “Treepedia 2.0: applying deep learning for large-scale quantification of urban tree cover”. In: *2018 IEEE International Congress on Big Data (BigData Congress)*. IEEE. 2018, pp. 49–56 (cit. on pp. 19, 20, 61).

[23] Grupo CEEE. *CEEE - Companhia Estadual de Energia Elétrica - Rio Grande do Sul - Compartilhamento de Infraestrutura*. https://ceee.equatorialenergia.com.br/compartilhamento-de-infraestrutura. Access Jan 22nd, 2023. 2021 (cit. on p. 38).

[24] CEMIG. *Queda de árvores é a principal causa de desligamentos acidentais em Minas Gerais*. https://www.cemig.com.br/noticia/acidentes-com-arvores-na-rede-eletrica/. Última visualização em 26/08/2023. 2023 (cit. on p. 1).

[25] D. Chan et al. “T-SNE-CUDA: GPU-Accelerated t-SNE and its Applications to Modern Data”. In: *Proc. SBAC-PAD*. 2018, pp. 330–338 (cit. on pp. 72, 96, 124).

[26] Aditya Chattopadhyay et al. “Grad-CAM++: Generalized Gradient-based Visual Explanations for Deep Convolutional Networks”. In: *CoRR* abs/1710.11063 (2017). arXiv: 1710.11063. URL: http://arxiv.org/abs/1710.11063 (cit. on pp. 20, 43).

[27] Beidi Chen et al. “Angular visual hardness”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1637–1648 (cit. on p. 21).

[28] Haoyuan Chen et al. “GasHis-Transformer: A multi-scale visual transformer approach for gastric histopathological image detection”. In: *Pattern Recognition* 130 (2022), p. 108827 (cit. on p. 118).

[29] Wengang Cheng and Zhengzheng Song. “Power pole detection based on graph cut”. In: *2008 Congress on Image and Signal Processing*. Vol. 3. IEEE. 2008, pp. 720–724 (cit. on p. 19).

[30] François Chollet et al. *Keras*. 2015. URL: https://keras.io (cit. on pp. 116, 132).

[31] Donna J Clarke and John G White. “Towards ecological management of Australian powerline corridor vegetation”. In: *Landscape and urban planning* 86.3-4 (2008), pp. 257–266 (cit. on p. 19).

[32] Simon Clode and Franz Rottensteiner. “Classification of trees and powerlines from medium resolution airborne laserscanner data in urban environments”. In: *Proceedings of the APRS Workshop on Digital Image Computing (WDIC), Brisbane, Australia*. Vol. 21. 2005 (cit. on p. 20).

[33] D. Collaris and J. J. van Wijk. “StrategyAtlas: Strategy analysis for machine learning interpretability”. In: *IEEE TVCG* (2022). DOI:10.1109/TVCG.2022.3146806 (cit. on p. 94).

[34] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 47).

[35] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995). Springer, pp. 273–297 (cit. on pp. 93, 101, 122).

[36] David R Cox. “The regression analysis of binary sequences”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 20.2 (1958). Wiley Online Library, pp. 215–232 (cit. on pp. 93, 101, 122).

[37] Ekin D. Cubuk et al. “RandAugment: Practical data augmentation with no separate search”. In: *CoRR* abs/1909.13719 (2019). arXiv: 1909.13719. URL: http://arxiv.org/abs/1909.13719 (cit. on p. 17).

[38] J. Cunningham and Z. Ghahramani. “Linear Dimensionality Reduction: Survey, Insights, and Generalizations”. In: *JMLR* 16 (2015), pp. 2859–2900 (cit. on pp. 71, 95, 123).

[39] Arthur P Dempster, Nan M Laird, and Donald B Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22 (cit. on p. 73).

[40] D. L. Donoho and C. Grimes. “Hessian Eigenmaps: Locally Linear Embedding techniques for high-dimensional data”. In: *Proceedings of the National Academy of Sciences* 100.10 (2003). Publisher: National Academy of Sciences, pp. 5591–5596 (cit. on p. 71).

[41] M V et al. Eitzel. “Citizen Science Terminology Matters: Exploring Key Terms. Citizen Science: Theory and Practice”. In: 2(1):1 (2017), pp. 1–20. DOI: https://doi.org/10.5334/cstp.96 (cit. on p. 19).

[42] D. Engel, L. Hattenberger, and B. Hamann. “A Survey of Dimension Reduction Methods for High-dimensional Data Analysis and Visualization”. In: *Proc. IRTG Workshop*. Vol. 27. Schloss Dagstuhl. 2012, pp. 135–149 (cit. on pp. 71, 95, 123).

[43] M. Espadoto et al. “Improving Neural Network-based Multidimensional Projections”. In: *Proc. IVAPP*. 2020 (cit. on pp. 70, 77, 84, 95).

[44] M. Espadoto et al. “OptMap: Using Dense Maps for Visualizing Multidimensional Optimization Problems”. In: *Proc. IVAPP*. SciTePress. 2021 (cit. on pp. 94, 96, 119).

[45] Mateus Espadoto, Nina ST Hirata, and Alexandru C Telea. “Self-supervised Dimensionality Reduction with Neural Networks and Pseudo-labeling”. In: *Proc. IVAPP*. SCITEPRESS, 2021, pp. 27–37 (cit. on pp. 70, 75).

[46] Mateus Espadoto, Nina ST Hirata, and Alexandru C Telea. “Self-supervised Dimensionality Reduction with Neural Networks and Pseudo-labeling.” In: *Proc. IVAPP*. SCITEPRESS. 2021, pp. 27–37 (cit. on pp. 96, 116, 124).

[47] Mateus Espadoto, Nina Sumiko Tomita Hirata, and Alexandru C Telea. "Deep learning multidimensional projections". In: *Information Visualization* 19.3 (2020). SAGE, pp. 247–269 (cit. on pp. 69, 70, 72, 78, 95, 96, 111, 124).

[48] Mateus Espadoto, Francisco C. M. Rodrigues, and Alexandru C. Telea. "Visual Analytics of Multidimensional Projections for Constructing Classifier Decision Boundary Maps". In: *Proc. IVAPP*. SCITEPRESS. 2019, pp. 132–144 (cit. on pp. 92, 94).

[49] Mateus Espadoto et al. "Deep Learning Inverse Multidimensional Projections". In: *Proc. EuroVA*. Eurographics. 2019 (cit. on pp. 70, 72, 86, 94, 96, 97, 105, 124, 125).

[50] Mateus Espadoto et al. "Toward a quantitative survey of dimension reduction techniques". In: *IEEE TVCG* 27.3 (2019), pp. 2153–2173 (cit. on pp. 69, 71, 79, 95, 123).

[51] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Proc. KDD*. Vol. 96. 34. 1996, pp. 226–231 (cit. on p. 73).

[52] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136 (cit. on p. 21).

[53] Judy Fakes. "Practical issues in line clearance and street trees". In: *TREENET* (2000), p. 39 (cit. on p. 19).

[54] Ronald A Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188 (cit. on p. 72).

[55] Django Software Foundation. *About the Django Software Foundation*. https://www.djangoproject.com/foundation/. Retrieved: 13/03/2023 (cit. on p. 24).

[56] Django Software Foundation. *Managers - Django Documentation - Django*. https://docs.djangoproject.com/en/2.2/topics/db/managers/. Retrieved: 13/01/2023 (cit. on p. 24).

[57] Brendan J Frey and Delbert Dueck. "Clustering by passing messages between data points". In: *Science* 315.5814 (2007), pp. 972–976 (cit. on p. 73).

[58] R. Garcia et al. "A Task-and-Technique Centered Survey on Visual Analytics for Deep Learning Model Engineering". In: *Computers and Graphics* 77 (2018). Elsevier, pp. 30–49 (cit. on pp. 91, 121).

[59] Andreas Geiger et al. "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237 (cit. on p. 20).

[60] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proc. AISTATS*. 2010, pp. 249–256 (cit. on p. 77).

[61] Clément Godard et al. "Digging into self-supervised monocular depth estimation". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 3828–3838 (cit. on p. 20).

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 13, 15, 16).

[63] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014) (cit. on pp. 21, 22).

[64] Chuan Guo et al. "On calibration of modern neural networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1321–1330 (cit. on p. 56).

[65] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on pp. 15, 67).

[66] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034 (cit. on p. 1).

[67] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proc. IEEE ICCV*. 2015, pp. 1026–1034 (cit. on p. 77).

[68] Nan He and Guanghao Li. “Urban neighbourhood environment assessment based on street view image processing: A review of research trends”. In: *Environmental Challenges* 4 (2021), p. 100090. ISSN: 2667-0100. DOI: https://doi.org/10.1016/j.envc.2021.100090. URL: https://www.sciencedirect.com/science/article/pii/S266701002100069X (cit. on pp. 1, 20).

[69] Dan Hendrycks et al. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15262–15271 (cit. on pp. 1, 21).

[70] G. E. Hinton and R. R. Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *Science* 313.5786 (2006). Publisher: AAAS, pp. 504–507 (cit. on pp. 69, 72).

[71] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* 2.7 (2015) (cit. on p. 15).

[72] P. Hoffman and G. Grinstein. “A survey of visualizations for high-dimensional data mining”. In: *Information Visualization in Data Mining and Knowledge Discovery* 104 (2002). Morgan Kaufmann, pp. 47–82 (cit. on pp. 71, 95, 123).

[73] Andrew Howard et al. “Searching for mobilenetv3”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1314–1324 (cit. on pp. 15, 55).

[74] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV] (cit. on p. 40).

[75] Gao Huang et al. “Deep Networks with Stochastic Depth”. In: *CoRR* abs/1603.09382 (2016). arXiv: 1603.09382. URL: http://arxiv.org/abs/1603.09382 (cit. on p. 17).

[76] Gao Huang et al. “Deep networks with stochastic depth”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer. 2016, pp. 646–661 (cit. on p. 52).

[77] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708 (cit. on pp. 15, 67).

[78] Yingsong Huang et al. “Uncertainty-aware Learning Against Label Noise on Imbalanced Datasets”. In: (2022) (cit. on p. 21).

[79] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.3 (2007). IEEE, pp. 90–95 (cit. on pp. 98, 126).

[80] *imglab*. https://github.com/NaturalIntelligence/imglab/. Access Dec 17th, 2022. 2017 (cit. on pp. 62, 65).

[81] Google Inc. *Google Street View*. https://www.google.com/maps/streetview/. Last access 13/01/2023 (cit. on pp. 20, 25, 32).

[82] Infonet.com.br. *Manutenção de árvores pode prevenir acidentes com rede elétrica*. https://infonet.com.br/noticias/cidade/manutencao-de-arvores-pode-prevenir-acidentes-com-rede-eletrica/. Última visualização em 26/08/2023. 2022 (cit. on p. 1).

[83] Ziheng Jiang et al. "Characterizing structural regularities of labeled data in overparameterized models". In: *arXiv preprint arXiv:2002.03206* (2020) (cit. on p. 22).

[84] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5 (cit. on p. 1).

[85] P. Joia et al. "Local affine multidimensional projection". In: *IEEE TVCG* 17.12 (2011), pp. 2563–2571 (cit. on pp. 71, 76, 95).

[86] I. T. Jolliffe. "Principal component analysis and factor analysis". In: *Principal Component Analysis*. Springer. 1986, pp. 115–128 (cit. on pp. 69, 71, 95, 123).

[87] Y Jwa, G Sohn, and HB Kim. "Automatic 3d powerline reconstruction using airborne lidar data". In: *Int. Arch. Photogramm. Remote Sens* 38.Part 3 (2009), W8 (cit. on pp. 19, 20, 49).

[88] L Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 2005 (cit. on p. 73).

[89] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. "Shallow-deep networks: Understanding and mitigating network overthinking". In: *International conference on machine learning*. PMLR. 2019, pp. 3301–3310 (cit. on p. 63).

[90] J. Kehrer and H. Hauser. "Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey". In: *IEEE TVCG* 19.3 (2013), pp. 495–513 (cit. on p. 69).

[91] W Andy Kenney, Philip JE van Wassenaer, Alexander L Satel, et al. "Criteria and indicators for strategic urban forest planning and management". In: *Arboriculture & Urban Forestry* 37.3 (2011), pp. 108–117 (cit. on p. 19).

[92] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *CoRR* abs/1312.6114 (2013). eprint: 1312.6114 (cit. on p. 72).

[93] Yoshihiro Kobayashi et al. "The utilization of satellite images to identify trees endangering transmission lines". In: *IEEE Transactions on Power Delivery* 24.3 (2009), pp. 1703–1709 (cit. on pp. 19, 49).

[94] T. Kohonen. *Self-organizing Maps*. Springer, 1997 (cit. on p. 69).

[95] Basma Korchani and Kaouthar Sethom. "Real-Time Littering Detection for Smart City using Deep Learning Algorithm". In: *2020 International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*. 2021, pp. 1–5. DOI: 10.1109/ICCSPA49915.2021.9385721 (cit. on p. 20).

[96] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. "Handling imbalanced datasets: A review". In: *GESTS international transactions on computer science and engineering* 30.1 (2006), pp. 25–36 (cit. on p. 21).

[97] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105 (cit. on p. 55).

[98] Anders Krogh and John A Hertz. "A simple weight decay can improve generalization". In: *Proc. NIPS*. 1992, pp. 950–957 (cit. on p. 77).

[99] J. B. Kruskal. "Multidimensional Scaling by optimizing goodness of fit to a nonmetric hypothesis". In: *Psychometrika* 29.1 (1964). Publisher: Springer, pp. 1–27 (cit. on p. 105).

[100] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 21).

[101] Y. LeCun and C. Cortes. *MNIST Handwritten Digits Dataset*. http://yann.lecun.com/exdb/mnist. 2010 (cit. on pp. 75, 102, 128).

[102] Yann LeCun et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems* 2 (1989) (cit. on p. 19).

[103] Xiaojiang Li and Carlo Ratti. "Mapping the spatial distribution of shade provision of street trees in Boston using Google Street View panoramas". In: *Urban Forestry & Urban Greening* 31 (2018), pp. 109–119. ISSN: 1618-8667. DOI: https://doi.org/10.1016/j.ufug.2018.02.013 (cit. on p. 19).

[104] Rodrigo Alves Lima and Wallace Faveron de Almeida. "API de navegação no Google Street View e análise de imagens da paisagem urbana". In: () (cit. on p. 62).

[105] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2013. arXiv: 1312.4400 [cs.NE] (cit. on p. 40).

[106] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988 (cit. on pp. 8, 16, 21, 51, 62).

[107] Dongwei Liu et al. "Establishing a citywide street tree inventory with street view images and computer vision techniques". In: *Computers, Environment and Urban Systems* 100 (2023), p. 101924 (cit. on p. 20).

[108] Li Liu et al. "Deep learning for generic object detection: A survey". In: *International journal of computer vision* 128.2 (2020), pp. 261–318 (cit. on p. 21).

[109] S. Liu et al. "Visualizing High-Dimensional Data: Advances in the Past Decade". In: *IEEE TVCG* 23.3 (2015), pp. 1249–1268 (cit. on pp. 69, 71, 95, 123).

[110] Wanli Liu et al. "CVM-Cervix: A hybrid cervical Pap-smear image classification framework using CNN, visual transformer and multilayer perceptron". In: *Pattern Recognition* 130 (2022), p. 108829 (cit. on p. 118).

[111] Stuart Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137 (cit. on p. 73).

[112] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Proc. NIPS*. 2017, pp. 4768–4777 (cit. on pp. 91, 121).

[113] Ulrich Lüttge and Marcos Buckeridge. *Trees: structure and function and the challenges of urbanization*. 2020 (cit. on p. 1).

[114] Jun Ma et al. "Real-time detection of wildfire risk caused by powerline vegetation faults using advanced machine learning techniques". In: *Advanced Engineering Informatics* 44 (2020), p. 101070 (cit. on p. 19).

[115] L. van der Maaten. "Accelerating t-SNE using Tree-Based Algorithms". In: *JMLR* 15 (2014), pp. 3221–3245 (cit. on pp. 72, 96, 124).

[116] L. van der Maaten. "Barnes-Hut-SNE". In: *arXiv preprint arXiv:1301.3342* (2013) (cit. on p. 87).

[117] L. van der Maaten. "Learning a parametric embedding by preserving local structure". In: *Proc. AI-STATS*. 2009 (cit. on p. 96).

[118] L. van der Maaten and G. Hinton. "Visualizing data using t-SNE". In: *JMLR* 9 (Nov 2008), pp. 2579–2605 (cit. on pp. 69, 71, 94, 96, 105, 123).

[119] L. van der Maaten and E. Postma. *Dimensionality Reduction: A Comparative Review*. Tech. rep. Tilburg University, Netherlands, 2009 (cit. on pp. 71, 75, 95, 123).

[120] Mappilary. *About | Mappilary*. https://www.mapillary.com/about. Last access 13/01/2023 (cit. on p. 28).

[121] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 15).

[122] R. Martins et al. "Visual Analysis of Dimensionality Reduction Quality for Parameterized Projections". In: *Computers & Graphics* 41 (2014). Publisher: Elsevier, pp. 26–42 (cit. on p. 94).

[123] Rafael Messias Martins, Rosane Minghim, Alexandru C Telea, et al. "Explaining Neighborhood Preservation for Multidimensional Projections." In: *CGVC*. 2015, pp. 7–14 (cit. on p. 75).

[124] David Mayo et al. *How hard are computer vision datasets? Calibrating dataset difficulty to viewing time*. https://objectnet.dev/flash/how_hard_are_computer_vision_datasets_calibrating_dataset_difficulty_to_viewing_time.pdf. Access Oct 17th, 2022. 2022 (cit. on p. 21).

[125] L. McInnes and J. Healy. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *arXiv:1802.03426v1 [stat.ML]* (2018) (cit. on pp. 69, 72, 94, 96, 104, 116, 124, 129, 132).

[126] Terri S. Modrakowski et al. "Improving Deep Learning Projections by Neighborhood Analysis". In: Springer. 2020 (cit. on pp. 70, 96).

[127] Sina Mohseni et al. "Taxonomy of Machine Learning Safety: A Survey and Primer". In: *arXiv e-prints* (2021), arXiv–2106 (cit. on pp. 1, 21).

[128] Gerhard Neuhold et al. "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes". In: *International Conference on Computer Vision (ICCV)*. 2017. URL: https://www.mapillary.com/dataset/vistas (cit. on p. 19).

[129] Quynh C. Nguyen et al. "Using Google Street View to examine associations between built environment characteristics and U.S. health outcomes". In: *Preventive Medicine Reports* 14 (2019), p. 100859. ISSN: 2211-3355. DOI: https://doi.org/10.1016/j.pmedr.2019.100859 (cit. on p. 20).

[130] Caio Nóbrega and Leandro Marinho. "Towards explaining recommendations through local surrogate models". In: *Proc. ACM/SIGAPP Symp. on Applied Computing*. 2019, pp. 1671–1678 (cit. on p. 91).

[131] L. Nonato and M. Aupetit. "Multidimensional Projection for Visual Analytics: Linking Techniques with Distortions, Tasks, and Layout Enrichment". In: *IEEE TVCG* (2018). DOI: 10.1109/TVCG.2018.2846735 (cit. on pp. 69, 71, 91, 95, 121, 123).

[132] Caitlin O'Kane. *How much of Maui has burned in the wildfires? Aerial images show fire damage as containment efforts continue*. https://www.cbsnews.com/news/how-much-of-maui-has-burned-wildfires-aerial-images-acres-containment/. Última visualização em 26/08/2023. 2023 (cit. on p. 1).

[133] Travis Oliphant. *NumPy: A guide to NumPy*. USA: Trelgol Publishing. Retrieved: 2023-01-13. 2006. URL: http://www.numpy.org/ (cit. on p. 32).

[134] A. A. M. Oliveira et al. "SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers". In: *Proc. IVAPP*. SciTePress. 2022, pp. 77–87 (cit. on pp. 91, 100, 118).

[135] Artur AAM Oliveira et al. “Stability Analysis of Supervised Decision Boundary Maps”. In: *SN Computer Science* 4.3 (2023), p. 226 (cit. on p. 91).

[136] Artur AM Oliveira et al. “SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers.” In: *VISIGRAPP (3: IVAPP)*. 2022, pp. 77–87 (cit. on p. 121).

[137] Artur André Oliveira, Marcos S Buckeridge, and Roberto Hirata Jr. “Detecting tree and wire entanglements with deep learning”. In: *Trees* (2022), pp. 1–13 (cit. on pp. 32, 37).

[138] Artur André A. M. Oliveira. *INACITY use cases*. https://youtu.be/K525hS7SsAg. Last access 13/01/2023. 2021 (cit. on p. 30).

[139] Artur André A. M. Oliveira, Zhangyang Wang, and Roberto Hirata. “Locating Urban Trees near Electric Wires using Google Street View Photos: A New Dataset and A Semi-Supervised Learning Approach in the Wild”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2022, pp. 4285–4293. DOI: 10.1109/CVPRW56347.2022.00474 (cit. on p. 51).

[140] Artur André AM Oliveira et al. “Improving Self-supervised Dimensionality Reduction: Exploring Hyperparameters and Pseudo-Labeling Strategies”. In: *Computer Vision, Imaging and Computer Graphics Theory and Applications: 16th International Joint Conference, VISIGRAPP 2021, Virtual Event, February 8–10, 2021, Revised Selected Papers*. Springer. 2023, pp. 135–161 (cit. on p. 69).

[141] OpenStreetMap contributors. *Overpass API/Overpass QL - OpenStreetMap Wiki*. https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL. Last access 13/01/2023. 2019 (cit. on p. 30).

[142] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017 (cit. on pp. 24, 29, 32, 61).

[143] Camilo Ordóñez and Peter N Duinker. “An analysis of urban forest management plans in Canada: Implications for urban forest management”. In: *Landscape and Urban Planning* 116 (2013), pp. 36–47 (cit. on p. 19).

[144] Pertti Pakonen. “Characteristics of partial discharges caused by trees in contact with covered conductor lines”. In: *IEEE Transactions on Dielectrics and Electrical Insulation* 15.6 (2008), pp. 1626–1633 (cit. on pp. 1, 19).

[145] Sinno Jialin Pan and Qiang Yang. “A survey on transfer learning”. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359 (cit. on p. 39).

[146] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 15).

[147] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. “Deep learning on a data diet: Finding important examples early in training”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 20596–20607 (cit. on pp. 21, 22).

[148] F. V. Paulovich and R. Minghim. “Text map explorer: a tool to create and explore document maps”. In: *Proc. Intl. Conference on Information Visualisation (IV)*. IEEE, 2006, pp. 245–251 (cit. on pp. 71, 105).

[149] F. V. Paulovich, C. T. Silva, and L. G. Nonato. “Two-phase mapping for projecting massive datasets”. In: *IEEE TVCG* 16.6 (2010), pp. 1281–1290 (cit. on p. 105).

[150] F. V. Paulovich et al. “Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping”. In: *IEEE TVCG* 14.3 (2008), pp. 564–575 (cit. on pp. 71, 76).

[151] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 24).

[152] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research (JMLR)* 12 (2011), pp. 2825–2830 (cit. on p. 87).

[153] N. Pezzotti et al. “Approximated and User Steerable t-SNE for Progressive Visual Analytics”. In: *IEEE TVCG* 23 (2017), pp. 1739–1752 (cit. on pp. 72, 96, 124).

[154] N. Pezzotti et al. “Hierarchical stochastic neighbor embedding”. In: *Computer Graphics Forum* 35.3 (2016). Wiley Online Library, pp. 21–30 (cit. on pp. 72, 96, 124).

[155] Nicola Pezzotti et al. “GPGPU Linear Complexity t-SNE Optimization”. In: *IEEE TVCG* 26.1 (2020), pp. 1172–1181 (cit. on pp. 72, 96, 124).

[156] Lindsey Pucell. *Trees and Electric Lines*. https://extension.purdue.edu/extmedia/FNR/FNR-512-W.pdf. Access Jan 22nd, 2021. 2015 (cit. on p. 19).

[157] PySimpleGUI. *PySimpleGUI*. https://www.pysimplegui.org/en/latest/. Access May 05th, 2023. 2023 (cit. on p. 65).

[158] Mamunur Rahaman et al. “DeepCervix: A deep learning-based framework for the classification of cervical cells using hybrid deep feature fusion techniques”. In: *Computers in Biology and Medicine* 136 (2021), p. 104649 (cit. on p. 118).

[159] Mamunur Rahaman et al. “Identification of COVID-19 samples from chest X-Ray images using deep learning: A comparison of transfer learning approaches”. In: *Journal of X-Ray Science and Technology* 28.5 (2020), pp. 821–839 (cit. on p. 118).

[160] P. E. Rauber, A. X. Falcao, and A. C. Telea. “Projections as Visual Aids for Classification System Design”. In: *Information Visualization* 17.4 (2017). SAGE, pp. 282–305 (cit. on pp. 91, 93, 121, 123).

[161] P. E. Rauber et al. “Visualizing the hidden activity of artificial neural networks”. In: *IEEE TVCG* 23.1 (2017), pp. 101–110 (cit. on pp. 91, 121).

[162] George Reese. *Database Programming with JDBC and JAVA*. " O’Reilly Media, Inc.", 2000 (cit. on p. 23).

[163] M. T. Ribeiro, S. Singh, and C. Guestrin. “Why should I trust you?: Explaining the predictions of any classifier”. In: *Proc. ACM SIGMOD KDD*. 2016, pp. 1135–1144 (cit. on pp. 91, 121).

[164] Francisco Rodrigues et al. “Constructing and Visualizing High-Quality Classifier Decision Boundary Maps”. In: *Information* 10.9 (2019). MDPI, p. 280 (cit. on pp. 91, 92, 94, 95, 105, 116, 121, 123, 130, 132).

[165] Francisco Caio M Rodrigues, Roberto Hirata, and Alexandru Cristian Telea. “Image-based visualization of classifier decision boundaries”. In: *Proc. IEEE Conf. on Graphics, Patterns and Images (SIBGRAPI)*. 2018, pp. 353–360 (cit. on pp. 92, 93).

[166] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961 (cit. on p. 13).

[167] S. T. Roweis and L. L. K. Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *Science* 290.5500 (2000). AAAS, pp. 2323–2326 (cit. on pp. 71, 96, 123).

[168] Andrew G Rundle et al. “Using Google Street View to audit neighborhood environments”. In: *American journal of preventive medicine* 40.1 (2011), pp. 94–100 (cit. on pp. 19, 30).

[169] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252 (cit. on pp. 1, 21, 39, 67).

[170] BryanC. Russell et al. “LabelMe: A Database and Web-Based Tool for Image Annotation”. English. In: *International Journal of Computer Vision* 77.1-3 (2008), pp. 157–173. ISSN: 0920-5691. DOI: 10.1007/s11263-007-0090-8. URL: http://dx.doi.org/10.1007/s11263-007-0090-8 (cit. on pp. 62, 65).

[171] Brianna Sacks. *Power lines likely caused Maui’s first reported fire, video and data show*. https://www.washingtonpost.com/climate-environment/2023/08/15/maui-fires-power-line-cause/. Última visualização em 26/08/2023. 2023 (cit. on p. 1).

[172] A. Saltelli et al. *Global Sensitivity Analysis: The Primer*. John Wiley & Sons. 2008 (cit. on p. 119).

[173] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill. 1986 (cit. on pp. 75, 102, 128).

[174] Mark Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520 (cit. on pp. 15, 37, 39, 62).

[175] Prefeitura de São Paulo. *Mapa digital da cidade de São Paulo*. http://geosampa.prefeitura.sp.gov.br/PaginasPublicas/_SBC.aspx. Access Jan 22nd, 2021. 2023 (cit. on pp. 29, 32, 38, 61).

[176] C. Seifert, V. Sabol, and W. Kienreich. “Stress maps: analysing local phenomena in dimensionality reduction based visualisations”. In: *Proc. IEEE VAST*. 2010 (cit. on p. 95).

[177] Ramprasaath R Selvaraju et al. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626 (cit. on pp. 8, 62).

[178] Ramprasaath R Selvaraju et al. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626 (cit. on p. 43).

[179] Claude E Shannon. “A mathematical theory of communication”. In: *The Bell system technical journal* 27.3 (1948), pp. 379–423 (cit. on p. 15).

[180] D. Shepard. “A two-dimensional interpolation function for irregularly-spaced data”. In: *Proc. ACM National Conference*. 1968, pp. 517–524 (cit. on p. 94).

[181] Jianbo Shi and Jitendra Malik. “Normalized cuts and image segmentation”. In: *IEEE TPAMI* 22.8 (2000), pp. 888–905 (cit. on p. 73).

[182] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on pp. 15, 67).

[183] Michael R Smith, Tony Martinez, and Christophe Giraud-Carrier. “An instance level analysis of data complexity”. In: *Machine learning* 95.2 (2014), pp. 225–256 (cit. on p. 21).

[184] C. Sorzano, J. Vargas, and A. Pascual-Montano. *A survey of dimensionality reduction techniques*. arXiv:1403.2877 [stat.ML]. 2014 (cit. on pp. 71, 95, 123).

[185] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on p. 17).

[186] Maarten Sukel, Stevan Rudinac, and Marcel Worring. “Multimodal classification of urban micro-events”. In: *Proceedings of the 27th ACM International Conference on Multimedia*. 2019, pp. 1455–1463 (cit. on pp. 1, 35).

[187] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114 (cit. on pp. 15, 67).

[188] J. B. Tenenbaum, V. De Silva, and J. C. Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* 290.5500 (2000). AAAS, pp. 2319–2323 (cit. on pp. 71, 96, 123).

[189] tensorflow. *tensorflow/mobilenet_v2.py at GitHub*. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/keras/applications/mobilenet_v2.py. Access Jan 22nd, 2021. 2018 (cit. on p. 39).

[190] The Authors. *SDBM Implementation*. https://github.com/mespadoto/sdbm. 2021 (cit. on pp. 116, 132).

[191] Martin Thoma. *The Reuters Dataset*. https://martin-thoma.com/nlp-reuters. July 2017 (cit. on pp. 75, 102, 128).

[192] Z. Tian et al. “Using Multiple Attribute-Based Explanations of Multidimensional Projections to Explore High-Dimensional Data”. In: *Computers and Graphics* 98 (2021), pp. 93–104 (cit. on p. 94).

[193] Eric K. Tokuda, Roberto M. Cesar, and Claudio T. Silva. “Quantifying the Presence of Graffiti in Urban Environments”. In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2019, pp. 1–4. DOI: 10.1109/BIGCOMP.2019.8679113 (cit. on pp. 19, 61).

[194] W. S. Torgerson. *Theory and Methods of Scaling*. Wiley. 1958 (cit. on pp. 71, 96, 123).

[195] Dmitry Ulyanov. *Multicore-TSNE*. 2016. URL: https://github.com/DmitryUlyanov/Multicore-TSNE (cit. on p. 87).

[196] J. Venna and S. Kaski. “Visualizing gene interaction graphs with local multidimensional scaling”. In: *Proc. ESANN*. 2006, pp. 557–562 (cit. on pp. 75, 95).

[197] E. Vernier et al. “Quantitative Comparison of Time-Dependent Treemaps”. In: *Computer Graphics Forum* 39.3 (2020), pp. 393–404 (cit. on p. 107).

[198] E. Vernier et al. “Quantitative Evaluation of Time-Dependent Multidimensional Projection Techniques”. In: *Proc. EuroVis*. 2020 (cit. on pp. 95, 107, 119).

[199] E. F. Vernier, J. Comba, and A. Telea. “Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization”. In: *Proc. IEEE VISSOFT*. 2018 (cit. on p. 107).

[200] John Vlissides et al. “Design patterns: Elements of reusable object-oriented software”. In: *Reading: Addison-Wesley* 49.120 (1995), p. 11 (cit. on p. 24).

[201] *VoTT (Visual Object Tagging Tool)*. https://github.com/microsoft/VoTT/. Access Dec 17th, 2022. 2018 (cit. on pp. 62, 65).

[202] Stéfan van der Walt et al. “scikit-image: image processing in Python”. In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453 (cit. on p. 32).

[203] D.W. Wanik et al. “Using vegetation management and LiDAR-derived tree height data to improve outage predictions for electric utilities”. In: *Electric Power Systems Research* 146 (2017), pp. 236–245. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2017.01.039. URL: https://www.sciencedirect.com/science/article/pii/S0378779617300482 (cit. on p. 20).

[204] M. Wattenberg. *How to use t-SNE effectively*. https://distill.pub/2016/misread-tsne. 2016 (cit. on pp. 72, 96, 124).

[205] Jan D Wegner et al. “Cataloging public objects using aerial and street-level images-urban trees”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 6014–6023 (cit. on p. 19).

[206] J. J. van Wijk and H. van de Wetering. “Cushion Treemaps: Visualization of Hierarchical Information”. In: *Proc. InfoVis*. 1999 (cit. on p. 129).

[207] Wikipedia. *Dixie Fire*. https://en.wikipedia.org/wiki/Dixie_Fire. Última visualização em 26/08/2023. 2021 (cit. on p. 1).

[208] Jeffrey S Wilson et al. “Assessing the built environment using omnidirectional imagery”. In: *American journal of preventive medicine* 42.2 (2012), pp. 193–199 (cit. on p. 19).

[209] H. Xiao, K. Rasul, and R. Vollgraf. *Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv:1708.07747. 2017 (cit. on pp. 75, 102, 128).

[210] H. Xie, J. Li, and H. Xue. *A survey of dimensionality reduction techniques based on random projection*. arXiv:1706.04371 [cs.LG]. 2017 (cit. on pp. 71, 95, 123).

[211] Qizhe Xie et al. “Self-training with noisy student improves imagenet classification”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10687–10698 (cit. on pp. 8, 11, 15, 17, 51, 52, 55).

[212] Wenhan Yang et al. “Advancing image understanding in poor visibility environments: A collective benchmark study”. In: *IEEE Transactions on Image Processing* 29 (2020), pp. 5737–5752 (cit. on pp. 20, 21).

[213] Xiangli Yang et al. “A survey on deep semi-supervised learning”. In: *IEEE Transactions on Knowledge and Data Engineering* (2022) (cit. on p. 17).

[214] Wei Yao and H Fan. “Automated detection of 3D individual trees along urban road corridors by mobile laser scanning systems”. In: *Proceedings of the International Symposium on Mobile Mapping Technology, Tainan, Taiwan*. 2013, pp. 1–3 (cit. on p. 20).

[215] Lucas Henrique Bahr Yau and Roberto Jr. Hirata. *Reconhecimento de Ações Humanas em Imagens com Deep Learning*. https://www.linux.ime.usp.br/~rukasu/mac0499/Monografia_v3.pdf. Access May 05th, 2023. 2023 (cit. on p. 65).

[216] Matthew D Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012) (cit. on p. 67).

[217] Jinghua Zhang et al. “LCU-Net: A novel low-cost U-Net for environmental microorganism image segmentation”. In: *Pattern Recognition* 115 (2021), p. 107885 (cit. on p. 118).

[218] Quan-shi Zhang and Song-chun Zhu. “Visual interpretability for deep learning: a survey”. In: *Frontiers of Information Technology & Electronic Engineering* 19.1 (Jan. 2018), pp. 27–39. ISSN: 2095-9230. DOI: 10.1631/FITEE.1700808. URL: https://doi.org/10.1631/FITEE.1700808 (cit. on p. 43).

[219] Z. Zhang and J. Wang. “MLLE: Modified Locally Linear Embedding using multiple weights”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2007, pp. 1593–1600 (cit. on p. 71).

[220] Z. Zhang and H. Zha. “Principal manifolds and nonlinear dimensionality reduction via tangent space alignment”. In: *SIAM Journal on Scientific Computing* 26.1 (2004). Publisher: SIAM, pp. 313–338 (cit. on p. 71).

[221] Tianyi Zhou, Shengjie Wang, and Jeffrey Bilmes. “Curriculum learning by dynamic instance hardness”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 8602–8613 (cit. on p. 21).