Check for updates

RESEARCH ARTICLE

REVISED An in-depth analysis of data reduction methods for sustainable deep learning

[version 2; peer review: 2 approved]

Javier Perera-Lago[1], Victor Toscano-Duran[1], Eduardo Paluzo-Hidalgo[2], Rocio Gonzalez-Diaz[1], Miguel A. Gutiérrez-Naranjo[3], Matteo Rucco[1,4]

[1]Applied Mathematics I Department, University of Seville, Seville, Andalusia, Spain
[2]Quantitative Methods Department, Loyola University of Andalusia, Dos Hermanas, Andalusia, Spain
[3]Computer Science & Artificial Intelligence Department, University of Seville, Seville, Andalusia, Spain
[4]Data Science Department, Biocentis, Milan, Lombardy, Italy

v2 **First published:** 20 May 2024, **4**:101
https://doi.org/10.12688/openreseurope.17554.1
Latest published: 18 Sep 2024, **4**:101
https://doi.org/10.12688/openreseurope.17554.2

Abstract

In recent years, deep learning has gained popularity for its ability to solve complex classification tasks. It provides increasingly better results thanks to the development of more accurate models, the availability of huge volumes of data and the improved computational capabilities of modern computers. However, these improvements in performance also bring efficiency problems, related to the storage of datasets and models, and to the waste of energy and time involved in both the training and inference processes. In this context, data reduction can help reduce energy consumption when training a deep learning model. In this paper, we present up to eight different methods to reduce the size of a tabular training dataset, and we develop a Python package to apply them. We also introduce a representativeness metric based on topology to measure the similarity between the reduced datasets and the full training dataset. Additionally, we develop a methodology to apply these data reduction methods to image datasets for object detection tasks. Finally, we experimentally compare how these data reduction methods affect the representativeness of the reduced dataset, the energy consumption and the predictive performance of the model.

Keywords

Deep Learning, energy efficiency, sustainability, data reduction, dataset representativeness, classification, object detection.

Open Peer Review

Approval Status ✓ ✓

	1	2
version 2 (revision) 18 Sep 2024	✓ view	
version 1 20 May 2024	? view	✓ view

1. **Ekaterina Merkurjev**, Michigan State University, East Lansing, USA

2. **Tianzhixi Yin**, Pacific Northwest National Laboratory, Richland,, USA

Any reports and responses or comments on the article can be found at the end of the article.

This article is included in the Sustainable Development gateway.

This article is included in the Horizon Europe gateway.

Corresponding authors: Javier Perera-Lago (jperera@us.es), Victor Toscano-Duran (vtoscano@us.es), Eduardo Paluzo-Hidalgo (epaluzo@uloyola.es), Rocio Gonzalez-Diaz (rogodi@us.es), Miguel A. Gutiérrez-Naranjo (magutier@us.es), Matteo Rucco (matteo.rucco@biocentis.com)

Author roles: Perera-Lago J: Formal Analysis, Investigation, Methodology, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Toscano-Duran V**: Formal Analysis, Investigation, Methodology, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Paluzo-Hidalgo E**: Conceptualization, Investigation, Methodology, Project Administration, Supervision, Validation, Writing – Original Draft Preparation, Writing – Review & Editing; **Gonzalez-Diaz R**: Conceptualization, Funding Acquisition, Investigation, Methodology, Project Administration, Supervision, Writing – Review & Editing; **Gutiérrez-Naranjo MA**: Conceptualization, Investigation, Methodology, Project Administration, Supervision, Writing – Review & Editing; **Rucco M**: Conceptualization, Funding Acquisition, Investigation, Methodology, Project Administration, Supervision, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101070028 (REXASI-PRO).
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2024 Perera-Lago J *et al*. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Perera-Lago J, Toscano-Duran V, Paluzo-Hidalgo E *et al.* **An in-depth analysis of data reduction methods for sustainable deep learning [version 2; peer review: 2 approved]** Open Research Europe 2024, **4**:101 https://doi.org/10.12688/openreseurope.17554.2

First published: 20 May 2024, **4**:101 https://doi.org/10.12688/openreseurope.17554.1

REVISED **Amendments from Version 1**

Ekaterina Merkurjev's review showed that the language could sound incorrect to native English speakers. This new version serves to correct certain defects in the wording of the text. We have corrected typos, changed incorrect expressions, rewritten sentences whose meaning might be unclear and unified notations. The structure of the article and its scientific content remain the same.

Any further responses from the reviewers can be found at the end of the article

1 Introduction

Successful deep learning (DL) models require considerable consumption of resources for their development, partly due to the large volumes of data used for training. As explained in 1, most artificial intelligence (AI) research focuses solely on improving model performance at any cost. This line of research is known as Red AI. In contrast, Green AI considers the energy costs associated with AI development and seeks a balance between model performance and energy efficiency.

For example, recent publications such as 2 and 3 explain many different ways to improve the efficiency of intelligent agents, particularly DL models. In 4, the authors present experiments in which a dataset is reduced by random sampling with different percentages of reduction to train various types of models, such as k nearest neighbours (k-NN), decision trees (DT), support vector machines (SVM), random forests (RF), AdaBoost, and bagging classifier. These experiments suggest that reducing the size of the training set significantly reduces the training time in all cases and does not worsen the model's performance in specific cases of SVM, AdaBoost, and bagging classifier. For the other three algorithms (k-NN, DT and random forests), random reduction significantly decreases the F1-score.

The field of object detection in images has undergone a fascinating evolution over the years, driven by significant advances in computer vision and deep learning. In its early stages, the methods focused on traditional approaches, using features and regional classifiers. However, these methods could not efficiently handle large datasets and presented challenges in terms of speed and accuracy. The introduction of convolutional neural networks[5] (CNNs) marked a paradigm shift by addressing the automatic feature learning capability. R-CNN variants[6] introduced the concept of regions of interest (RoI), significantly improving accuracy but with a considerable computational cost. The true milestone came with the arrival of YOLO[7], which proposed an innovative approach by dividing the image into a grid and predicting bounding boxes and classes in a single pass. Although early versions of YOLO sacrificed some accuracy, they demonstrated revolutionary speed, making them suitable for real-time applications. Subsequent evolution, from YOLOv2 to YOLOv5 (the model we will focus on in this analysis from now on), has been marked by continuous improvements. Later versions refined the architecture, adding specialized layers and attention strategies, enhancing accuracy without significantly compromising speed.

In summary, we will focus on analyzing how data reduction affects the training of DL models, while expanding the list of reduction methods with additional algorithms developed in recent years. The two specific tasks on which we test the selected techniques are tabular data classification and object detection for image datasets. Specifically, we have tested eight different data reduction methods and compared their performance and efficiency in four different datasets: the Collision dataset and the Dry Bean dataset, both with tabular data and focused on classification, and the Roboflow dataset and the Mobility Aid dataset, composed of images and focused on object detection. Besides, we propose a specific methodology adapted to image processing, which is particularly crucial for detecting people in wheelchairs and pedestrians in the REXASI-PRO context (more information in https://rexasi-pro.spindoxlabs.com). Finally, we have published the Python code in two user-friendly GitHub repositories: one serving as a library for the data reduction methods[8], and the other showing the experiments carried out[9]. This paper is organized as follows: In Section 2, we present the Python library created to utilize data reduction methods, the proposed methodology to apply them to structured data, such as images, and a summary of the main results obtained and conclusions. Later, in Section 3, all the preliminary concepts are introduced, including key concepts about multi-layer perceptrons, how the classification of tabular data and object detection from images works, along with metrics to measure the performance of data reduction methods. In Section 4, we introduce different methods to reduce a dataset, categorizing them into four groups: statistic-based methods, geometry-based methods, ranking-based methods and wrapper methods. Finally, in Section 5, the results are tested experimentally for both tabular data and object detection, providing details about the databases, experimental setup and parameter settings.

2 Key findings and contributions

In this section, we highlight the main contributions developed for this paper.

2.1 Python Library: data reduction methods

We have released the beta version of a Python Library to use the data reduction methods. It is available on an open access GitHub repository, which includes instructions in the README.md file on how to use the data reduction methods.

Library repository	8
Experiments repository	9

The reduction methods implemented are listed in Table 1. We have included those state-of-the-art methods for data reduction that satisfy the following conditions:

1. The implementation is available or can be implemented easily.
2. The final size of the reduced dataset can be specified.
3. The reduction time offsets the training savings, as tested in small examples.

2.2 Data reduction for images

Some of the methods listed in Table 1 require their input to be n-dimensional vectors and cannot be directly applied to structured data such as images, which are stored as tensors of shape (height, width, channels). In this paper, we propose two methodologies (see Section 5.2.1) to extend the reduction methods to images, in the context of object detection. Specifically,

1. For statistic-based, geometry-based, and ranking-based reduction methods: The proposed methodology involves using a feature extraction model, such as the YOLOv5 backbone, and applying it to the given dataset. To apply data reduction techniques effectively, we use global average pooling (GAP)[10] to transform the images into n-dimensional vectors (768 dimensions, due to the backbone's structure, resulting in 768 feature maps). Finally, the images selected by the reduction method as the most important ones are used to train the model.
2. For the wrapper method: We create a classification DL model only to perform data reduction during its training. Then, the images selected as the most important by the reduction method are used to tune the YOLOv5 model.

2.3 Summary of the main results and conclusions

Our research on the effectiveness of reduction methods for achieving Green AI, aimed at minimizing CO_2 emissions and computational costs while maintaining performance, yielded promising results. One DL architecture was tested for each dataset in the experiments. The CO_2 emissions were estimated using a specific software, but no physical sensors were used (see Section 3.4.3).

- For tabular datasets: We have found that using reduced datasets notably decreases both the computation time and carbon emissions of neural network training. We have also found that these reduction methods can discard a large number of training examples without losing the good predictive properties of the DL models. Furthermore, we have found a significant statistical correlation between

Table 1. List of state-of-the-art data reduction methods selected for comparison.

STATISTIC-BASED	SRS	Stratified Random Sampling	4
	PRD	ProtoDash	11
GEOMETRY-BASED	CLC	Clustering Centroids	12
	MMS	Maxmin Selection	13
	DES	Distance-Entropy Selection	14
RANKING-BASED	PHL	PH Landmarks	15
	NRMD	Numerosity Reduction by Matrix Decomposition	16
WRAPPER	FES	Forgetting Events Score	17

the performance metrics of the trained models and a topological metric called ε-representativeness (discussed in detail in Section 3.4.1), which measures how close is the reduced dataset to the full one, for the Collision dataset.

- For images: The proposed methodology is an effective way to achieve greener DL models with good performance. In summary, regarding the Roboflow dataset, significant reductions in both CO_2 emissions and computation time (particularly for a 75% reduction rate) were achieved (approximately 60% time savings) without compromising the model's performance in object detection tasks. Among the reduction methods applied, SRS, MMS and RKM proved to be the most effective, while NRMD showed poorer performance, and FES resulted in longer processing times. Similarly, positive results were observed for the Mobility Aid dataset, with significant reductions in CO_2 emissions and computation time without compromising performance for specific methods. In this case, SRS, PRD, PHL and MMS showed superior effectiveness, while NRMD and RKM were identified as the least efficient methods.

3 Background

In this section, all preliminary concepts are introduced. We provide key concepts about multi-layer perceptrons and present the two specific problems we are addressing: the classification of tabular data and object detection from images. We then describe the YOLOv5 architecture. Finally, in Section 3.4, we discuss the different metrics used to measure the performance of the different data reduction methods.

3.1 Multi-layer Perceptrons

Multi-layer perceptrons (MLPs) are the simplest type of neural networks[18]. An MLP, denoted $\mathcal{N}$, is a function that transforms input vectors through a series of smaller functions called layers. Formally, $\mathcal{N}$ it can be represented as a composition $\mathcal{N} = f_l \circ \cdots \circ f_1$, where $f_j : \mathbb{R}^{d_{j-1}} \to \mathbb{R}^{d_j}$ is the j-th layer function. Each layer f_j can be decomposed into d_j smaller functions, $f_j = (f_{j,1},..., f_{j,d_j})$ called units or neurons. Each neuron $f_{j,m}$ is defined as $f_{j,m}(x) = g_j(W_{j,m}^T x + b_{j,m})$, where $W_{j,m} \in \mathbb{R}^{d_{j-1}}$ is the weight vector of the neuron, $b_{j,m} \in \mathbb{R}$ is the bias term, and $g_j : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function[19]. These components determine how information flows through the network and how it is transformed at each layer.

The design of an MLP depends on three elements: its architecture, a vector of parameters, and a set of learning hyperparameters. The architecture of an MLP is the choice for the number of layers (l), the number of dimensions of the layers ($d_1,..., d_l$) and the activation functions ($g_1, g_2,..., g_l$). Given an MLP architecture, the parameters are the weight vector entries $W_{j,m}$ and the bias terms $b_{j,m}$. All these parameters can be encoded in a parameter vector $\theta \in \mathbb{R}^p$, being p the number of adjustable parameters in the MLP. It is common to denote the MLP by $\mathcal{N}_\theta$ to state that $\mathcal{N}$ uses θ as a parameter vector. The set of all possible parameter vectors is denoted as Θ. Finally, the hyperparameters are related to the training procedure, that is, the search for a vector θ that makes $\mathcal{N}_\theta$ useful for the specific task and dataset. The hyperparameters must be defined prior to this search.

Given an MLP $\mathcal{N}_\theta$ and a tabular dataset $\mathcal{D}$ (as defined in Subsection 3.2), the suitability of the task is measured by a loss function $\mathcal{L}(\cdot, \mathcal{D}) : \Theta \to \mathbb{R}$[20], designed to have small values when $\mathcal{N}_\theta$ is useful and vice versa. Two typical examples of loss functions are the mean square error for regression tasks and the categorical cross entropy loss for classification tasks[21]. The search for a good choice of θ is made with an iterative training process that minimizes $\mathcal{L}$ over n_e successive epochs using the information from $\mathcal{D}$. In each epoch, $\mathcal{D}$ is randomly partitioned into a series of sub-datasets $\mathcal{B}_1, \mathcal{B}_2,...$, known as batches, with a maximum size equal to β. The model $\mathcal{N}_\theta$ is applied on each batch $\mathcal{B}_j$. After that, the performance of the model over the batch is evaluated, resulting in the calculation of the loss gradient $\nabla_\theta \mathcal{L}(\theta, \mathcal{B}_j)$. This gradient information is used by an optimization algorithm such as stochastic gradient descent (SGD), adaptative moment estimation (Adam) or root mean square propagation (RMSprop), to update θ and minimize $\mathcal{L}$[22]. These optimization algorithms may depend on other hyperparameters such as a learning rate $\gamma \in \mathbb{R}^+$ or a momentum $\mu \in \mathbb{R}^+$. Additionally, one can complement the training process with regularization techniques such as L1 or L2 regularization, dropout, or weight decay, which try to prevent overfitting (which occurs when the model fits the training dataset so well that it does not generalize well to new examples). The hyperparameters that define the training process are then the loss function $\mathcal{L}$, the number of epochs n_e, the batch size β, the optimization algorithm (together with other associated hyperparameters if needed), and the regularization techniques. For more details on feedforward neural networks, MLPs and how to train them, visit[23].

3.2 Classification of tabular data

A dataset $\mathscr{D} = (X, f)$ of a classification problem is a pair composed of a set of examples $X = \{x_1, \cdots x_N\} \subset \mathbb{R}^d$ and a function $f : X \to \{1,..., c\}$. The function $p_j : X \to \mathbb{R}$ that maps $x_i = (x_{i,1}, \cdots, x_{i,d})^T$ to $p_j(x_i) = x_{i,j}$ is called the feature j. The set of examples $X_k = \{x_i \in X : f(x_i) = k\}$ with $k \in \{1,..., c\}$ are called the class k. These types of datasets are usually known as tabular data because they can also be seen as a pair $(\mathbf{X}, \mathbf{y})$ where $\mathbf{X} \in \mathbb{R}^{N \times d}$ is a table or matrix whose i-th row corresponds to the example x_i and whose j-th column corresponds to the feature p_j, and $\mathbf{y} \in \mathbb{R}^n$ is a vector whose i-th component corresponds to $y_i = f(x_i)$. This tabular representation of the dataset is usual in computer science and is how it is used in Python code to design DL models. Given a dataset $\mathscr{D} = (X, f)$, the classification problem consists of finding an MLP: $\mathscr{N}: \mathbb{R}^d \to \{1,..., c\}$ that approximates f. If an MLP approximates f well, it is said to fit the dataset $\mathscr{D}$. If it also fits new unseen datasets with similar characteristics to $\mathscr{D}$, it is said to generalize well.

3.2.1 Data reduction

Given a dataset $\mathscr{D} = (X, f)$, the goal of data reduction is to find a reduced dataset $\mathscr{D}_R = (S, g)$, where $S = \{s_1, \cdots s_n\} \subset \mathbb{R}^d$ (being $n < N$) and $g : S \to \{1,..., c\}$. The class k in $\mathscr{D}_R$ will be denoted as $S_k = \{s_i \in S : g(s_i) = k\}$. In Section 4, some algorithms to extract a reduced dataset $\mathscr{D}_R$ from $\mathscr{D}$ are described. In most of them, the resulting $\mathscr{D}_R$ will be a sub-dataset of $\mathscr{D}$, which means that $S \subset X$ and $g = f|_S$. Assuming that $\mathscr{D}_R$ is representative of $\mathscr{D}$ and inherits its intrinsic properties, it should be possible to use it to train $\mathscr{N}$ instead of $\mathscr{D}$, gaining efficiency and obtaining a model $\mathscr{N}_R$ with similar performance. To test the correlation between representativeness, efficiency and performance it is necessary to define adequate metrics for all of them. The metrics used in our experiments can be consulted in Subsection 3.4.

3.3 Object detection from images

This Subsection is devoted to the definition of the problem and the description of YOLOv5.

3.3.1 Defining object detection

The problem of object detection[24] in the field of computer vision refers to the task of identifying the presence of specific objects in an image and providing accurate information on the spatial location of each of them. In other words, its main goal is to detect the presence of objects of interest within a visual scene and, at the same time, to delineate the exact region where they are located in the image.

To better understand this type of problem, it can be helpful to break it down into two key components:

- Object detection: This involves identifying and classifying the presence of objects within an image. This aspect addresses the fundamental question: "What objects are in the image?". Each detected object is usually associated with a specific class.

- Object location: Refers to providing information about the exact spatial location of the detected objects. This involves defining the coordinates or bounding boxes surrounding each object in the image, indicating exactly where they are located.

3.3.2 Object detection with YOLOv5

YOLO[7], an abbreviation for "You Only Look Once", is a DL model designed to address object detection with a comprehensive and efficient approach. Its main objective is to perform accurate detection of multiple objects in a single pass through the image, minimizing duplication of efforts and optimizing processing speed. YOLO is a single-stage architecture where object detection is performed by treating the problem as a regression problem to spatially separate the bounding box and the probability classes associated with the bounding box. YOLO predicts the bounding box and class directly from the entire image in a single evaluation. The fifth version of YOLO, named YOLOv5[25,26], is the first native release of models in the YOLO family written in Pytorch[27]. YOLOv5 is fast, with inference times of up to 0.007 seconds per image, meaning 140 frames per second. Figure 1 shows the detailed architecture of YOLOv5. Specifically, it consists of three main components: backbone, neck and head.

Backbone

The backbone extracts the essential features from the input image. In YOLOv5, it includes CSP-Darknet53, which is a convolutional neural network and incorporates a cross-stage partial network (CSPNet)[28] into Darknet to separate the base layer feature map into two parts and then combine them through a cross-stage hierarchy as shown in Figure 1. X is a variable in CSP1_X and CSP2_X, meaning the number of BottleNecks in the network. This enhanced CSPNet, built upon Darknet53, uses residual blocks, depthwise separable convolutions, and preactivation for improved efficiency and feature representation. For instance, given an RGB input

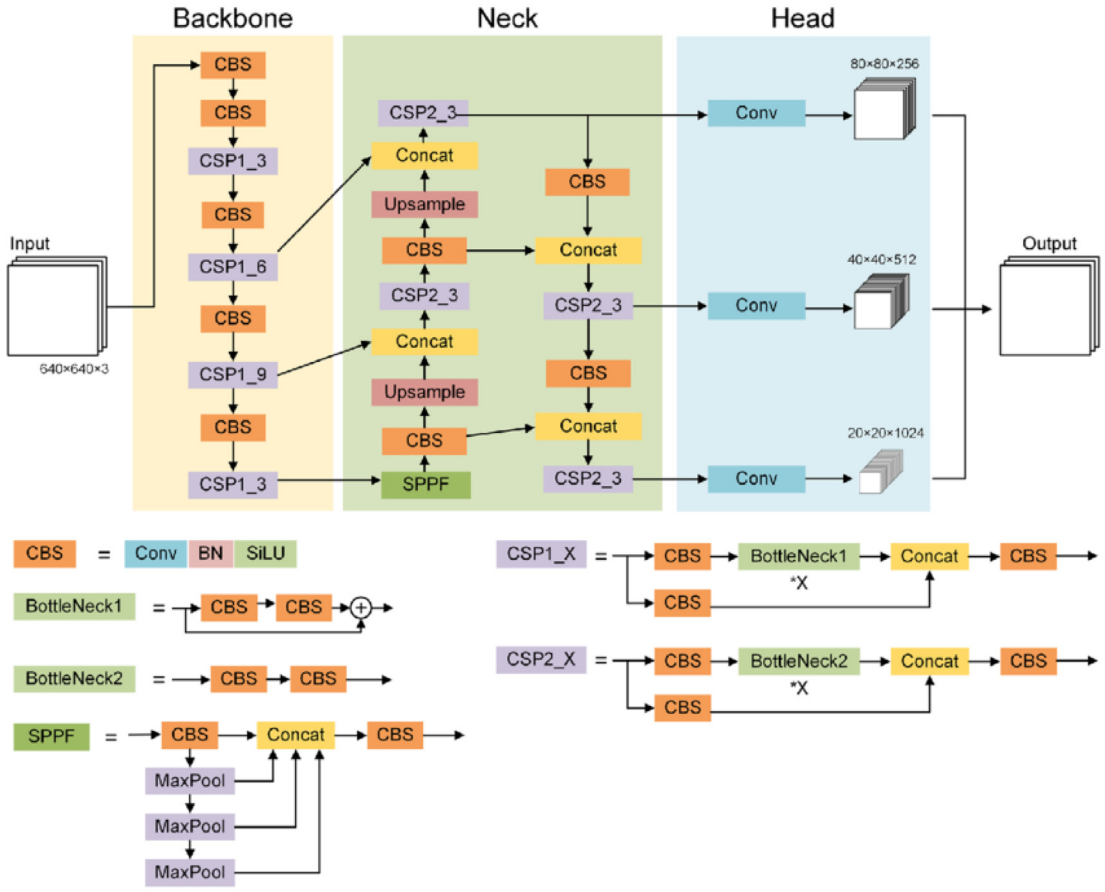

Figure 1. Architecture of YOLOv5[29], including three main parts: backbone, neck and head. The "backbone" is responsible for extracting fundamental features from the image, such as edges and textures. The "neck" is used to extract feature pyramids, which help the model to generalize well to objects of different sizes and scales. Finally, the "head" is responsible for the final prediction, generating the coordinates and classes of the detected objects.

image of 416 pixels in height and width, the backbone produces an output with 768 feature maps, each with dimensions of 13 pixels in height and width.

Neck

The neck acts as a bridge between the backbone and the head, performing operations to merge and refine features at different scales. The neck includes a spatial pyramid pooling-fast (SPPF) layer and a cross-stage partial path aggregation network (CSP-PAN), as shown in Figure 1. A spatial pyramid pooling (SPP)[30] layer is a pooling layer that removes the CNN limitation of fixed-size input images. The SPPF layer optimizes the SPP structure and improves the efficiency more than twofold. It aggregates the information received from inputs and returns a fixed-length output. PAN[31] is a feature pyramid network, used to improve information flow and help with the proper location of pixels in mask prediction task. In YOLOv5, this network has been modified applying the CSPNet strategy as shown in Figure 1.

Head

The network's head makes the final predictions, generating bounding boxes and classifications for each object. It is composed of four convolution layers that predict the location of the bounding boxes (x,y,height,width), the scores and the final classification. In addition, YOLOv5 uses several augmentations such as Mosaic, copy-paste, random affine, MixUp, HSV augmentation, random horizontal flip, as well as other augmentations from the `albumentations` package[32]. It also improves the grid sensitivity to make it more stable against runaway gradients.

YOLOv5 provides five scaled versions: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large), where the width and depth of the convolution modules vary depending on the specific applications and hardware requirements. From now on, we will focus on YOLOv5m, which is the one we have used for the experiments.

3.4 Metrics

In the following, we will discuss metrics for evaluating the performance of artificial intelligence models for both classification and object detection, as well as metrics for evaluating the representativeness of the reduced data over the entire dataset and evaluating the cost of the models.

3.4.1 Representativeness metrics

In this section, we present the two metrics that we will use in our experiments to measure the similarity between the original dataset $\mathscr{D}$ and its reduced version $\mathscr{D}_R$.

Reduction ratio

This is a common metric in the machine learning literature for comparing $\mathscr{D}$ and $\mathscr{D}_R$. The reduction ratio is just the quotient between the sizes of X_R and X. It has already been considered in publications such as 4 and 16 to measure the representativeness of $\mathscr{D}_R$ with respect to $\mathscr{D}$ and to study its impact on efficiency and performance, and we will also use it for our experiments.

ε-Representativeness

In 33, the authors introduced the concept of ε-representative datasets. The ε indicates how representative and, hence, how good is the representation of $\mathscr{D}_R$ for $\mathscr{D}$, smaller values being better. Consequently, we have $\varepsilon = 0$ if and only if $\mathscr{D}_R = \mathscr{D}$. Given a fixed isometry $i : \mathscr{D}_R \to \mathbb{R}^d$, the minimum ε such that $\mathscr{D}_R$ is a ε-representative dataset of $\mathscr{D}$ is:

$$\varepsilon^* = \max_{k=1,\ldots,c} \max_{x \in f^{-1}(k)} \max_{x' \in f_R^{-1}(k)} \|x - i(x')\| \tag{1}$$

It has been mathematically proven in 33 that, if $\mathscr{D}_R$ is ε-representative of $\mathscr{D}$ with ε small enough, $\mathscr{D}_R$ and $\mathscr{D}$ have the same accuracy (a performance metric whose definition can be read in 3.4.2) for a perceptron (which is an MLP with a single layer). Besides, it was proved experimentally in 33 that a similar relationship exists between the ε-representativeness of $\mathscr{D}_R$ with respect to $\mathscr{D}$ and the model performance for more complex neural architectures.

3.4.2 Performance metrics

Given a dataset $\mathscr{D} = (X, f)$ of a classification problem, one can design many different DL models with different architectures and parameters, that will give different approximations to f. It is then essential to measure how good these approximations are, assessing the overall performance of each model. Depending on the specific goals and characteristics of the classification task, some metrics might be more relevant than others, and the final choice of metrics is up to the model developers.

All the performance metrics that we will use in our experiments are derived from the confusion matrix. Given a dataset $\mathscr{D}$ and a DL model $\mathscr{N}$, the confusion matrix is a table with c rows and c columns where the cell (i, j) is filled with a non-negative integer $n_{i,j}$ equal to the number of examples in X whose actual class is $f(x) = i$ and whose predicted class is $\mathscr{N}(x) = j$. The sum of all the entries in row i is equal to the number of examples whose actual class is i, and it is denoted as A_i. The sum of all the entries in column j is equal to the number of examples whose predicted class is j, and is denoted as P_j. In addition, the sum of all the numbers in the confusion matrix is equal to N, the size of X. An example of a confusion matrix can be found in Table 2. Ideally, one would like to obtain a DL model $\mathscr{N}$ that exactly fits f, that is, $f(x) = \mathscr{N}(x)\ \forall x \in X$. In that case, the confusion matrix would be null out of the diagonal. In practice, it is not always possible to find such a perfect model (it may not be desirable due to the risk of overfitting X) but, in general, it is considered a good sign to get a confusion matrix with high values in the diagonal entries and lower values in the non-diagonal entries.

Accuracy

Accuracy is the most straightforward performance metric. It is the probability of correctly classifying a random example from X using $\mathscr{N}$. It is calculated as the quotient between the number of correctly classified examples and the total size of X, that is,

$$Acc = \frac{\sum_{k=1}^{c} n_{k,k}}{N} \tag{2}$$

Table 2. Confusion matrix for a classification problem with *c* classes, together with its marginal sums.

	Predicted 1	⋯	Predicted k	⋯	Predicted c	Total
Actual 1	$n_{1,1}$	⋯	$n_{1,k}$	⋯	$n_{1,c}$	A_1
⋮	⋮	⋱	⋮	⋱	⋮	⋮
Actual k	$n_{k,1}$	⋯	$n_{k,k}$	⋯	$n_{k,c}$	A_k
⋮	⋮	⋱	⋮	⋱	⋮	⋮
Actual c	$n_{c,1}$	⋯	$n_{c,k}$	⋯	$n_{c,c}$	A_c
Total	P_1	⋯	P_k	⋯	P_c	N

Accuracy is a good metric to get an overview of the model quality, but it can be misleading if the evaluation is reduced to it. When the training dataset is imbalanced (meaning that some classes are much more numerous than others) it is possible to find DL models that perform poorly at classifying the items from the less populated classes but yet have high accuracy because they perfectly fit f for the most populated ones. For this reason, it is also necessary to use other metrics that analyze the performance of $\mathcal{N}$ class by class.

Precision

Precision measures the probability that the model $\mathcal{N}$ is correct when it predicts that an example belongs to a specific class k. It tells us how confident we can be in the predictions obtained. The precision measure for a class k is obtained as the quotient between the number of examples correctly classified in class k and the total number of examples whose predicted class is k, that is,

$$Pre_k = \frac{n_{k,k}}{P_k} \tag{3}$$

When working with a dataset $\mathcal{D}$ with c classes, we can calculate c different precision values from the confusion matrix. We can aggregate all this information using the macro average precision, which is just the arithmetic mean of all of them:

$$MAPre = \frac{1}{c} \cdot \sum_{k=1}^{c} Pre_k \tag{4}$$

The macro average precision assigns the same relevance to the precision of all classes regardless of their size. It also mitigates the bias induced by the most populated ones.

Recall

Recall measures the probability that the model $\mathcal{N}$ correctly classifies the examples from X_k. It tells us how good the predictions are for that specific subset of X. For each class k the recall is obtained as the quotient between the number of examples correctly classified in class k and the total number of examples whose actual class is k, that is,

$$Rec_k = \frac{n_{k,k}}{A_k} \tag{5}$$

Just as all the precision values can be summarized with the macro average precision, the recall values can also be aggregated using the macro average recall, with the analogous formula:

$$MARec = \frac{1}{c} \cdot \sum_{k=1}^{c} Rec_k \tag{6}$$

Some bibliographic sources also refer to this metric as balanced accuracy[34].

In object detection, a high recall means the model is proficient at capturing most of the objects in the images, minimizing the number of false negatives.

F1-score

F1-score is a metric that gives us a trade-off between precision and recall. For each class k, it is obtained as the harmonic mean of the precision and recall values:

$$F1_k = 2 \cdot \frac{Pre_k \cdot Rec_k}{Pre_k + Rec_k} = 2 \cdot \frac{n_{k,k}}{P_k + A_k} \tag{7}$$

As a harmonic mean, $F1_k$ lies between Pre_k and Rec_k, always being less than or equal to the arithmetic mean $\frac{Pre_k + Rec_k}{2}$. In fact, $F1_k$ tends to approach the minimum value between Pre_k and Rec_k, resulting in a lower score when either precision or recall is low. Maximizing the F1-score is desirable because achieving a high value indicates that both precision and recall are high, proving that the model $\mathcal{N}$ has a good performance for that specific class.

As we already did with precision and recall, we can define the macro average F1-score as:

$$MAF1 = \frac{1}{c} \cdot \sum_{k=1}^{c} F1_k \tag{8}$$

An alternative definition for the macro average F1-score can be found in 35, but we prefer to use this one since it appears to be more suitable[36] and can be computed with standard Python libraries devoted to classification learning.

Intersection over Union and Mean Average Precision

Intersection over union (*IoU*)[37] is a performance measure specific to object detection tasks. It indicates the overlap of the predicted bounding box coordinates with the ground truth box. A higher *IoU* indicates that the predicted bounding box coordinates closely resemble the ground truth box coordinates. *IoU* is calculated by comparing the overlapped region between the prediction of the model and the ground truth with the total region covered by both. Mathematically, *IoU* is expressed as the ratio of the intersection area to the union area of the two regions:

$$IoU = \frac{IntersectionArea}{UnionArea}, \tag{9}$$

where:

- Intersection area: is the area where the model's prediction and the ground truth overlap.

- Union area: is the total area covered by both regions.

Normally, the prediction of a bounding box is considered correct if the *IoU* value is greater than 0.5, but other thresholds can be chosen.

The mean average precision (*mAP*)[38,39] is the current benchmark metric used by the computer vision research community to evaluate the robustness of object detection models. The *mAP* metric evaluates the overall accuracy of the model across multiple *IoU* thresholds, so the first thing to do when calculating the *mAP* is to select the *IoU* threshold. When calculating *mAP*, you have the flexibility to choose either a single *IoU* threshold or a range of thresholds. For instance, when you choosing a single IoU threshold, such as 0.5 (denoted *mAP*@0.5), you are assessing the model's accuracy when the predicted bounding box overlaps with the ground truth bounding box by at least 50%. However, setting a range of thresholds, like 0.5 to 0.95 with 0.05 increments (indicated as *mAP*@0.5:0.95), allows to evaluate the model performance across a range of IoU values.

The second thing to do is divide the object detections into classes based on the detected class. Next, compute the average precision (*AP*) for each class and calculate its mean, resulting in an *mAP* for a given *IoU* threshold. Then, calculate precision-recall curve points for different confidence thresholds and the *AP* for each class k using the following equation:

$$AP_k = \frac{1}{t} \int_0^1 Pre_k(Rec_k) dRec_k \tag{10}$$

where t is the number of *IoU* thresholds considered.

All the average precisions can be aggregated using the mean average precision, by the following formula:

$$mAP = \frac{1}{c} \cdot \sum_{k=1}^{c} AP_k \tag{11}$$

The *mAP* incorporates the trade-off between precision and recall and considers both false positives (FP) and false negatives (FN). This property makes *mAP* a suitable metric for most detection applications. A high *mAP* means that a model has both a low false negative rate and a low false positive rate.

3.4.3 Efficiency metrics

The cost of designing and using a model can depend on many factors. According to 1, the total cost of getting a result (R) is linearly related to the cost of processing a single example (E), the size of the training dataset (D) and the number of hyperparameters to be set (H), giving us the following equation:

$$\text{Cost}(R) \propto E \cdot D \cdot H \tag{12}$$

Following this idea, several metrics have been proposed to measure the amount of work performed during training, such as electricity consumption, the number of parameters to be adjusted, or the total number of floating-point operations performed. Some advantages and disadvantages of using these metrics can be read in 1. For our experiments, we will only focus on two specific metrics that are easy to calculate with Python code and give us an intuition about the impact that a DL model has on the environment. These are the elapsed computing time needed to build the DL model and the estimated carbon emission into the atmosphere during the process.

Elapsed computing time

Measuring the elapsed computing time is as simple as starting a timer at the beginning of the model-building process and using it to know how many seconds have passed until the whole process is finished. This time span can be influenced by factors independent of the training dataset and the model, such as hardware specifications, concurrent tasks on the same machine, and the use of multiple cores. However, it serves as a natural metric. When these factors are kept constant, the computation time serves as a direct indicator of energy consumption and carbon emissions, making it a meaningful measure of efficiency in the model-building process.

Estimated carbon emission

Carbon emission is the quantity we want to minimize, since carbon dioxide (CO_2) is one of the main gases involved in the greenhouse effect. An excessive release of CO_2 into the atmosphere contributes to changes in its composition, leading to an increase in the global average temperature[40,41]. Nevertheless, in practice, it is not easy to give an exact measure of carbon emission, since it depends on the sources of the energy used, the computer where the calculations are done, and the quality of the local electricity infrastructure. However, it is possible to give an approximate measure of carbon emission using the Python package CodeCarbon[42].

This approximation measure is the product of the energy consumed and its carbon intensity, which is the amount of CO_2 released per unit of energy. The amount of energy consumed by a computer is estimated by monitoring the power usage of its components, such as the central processing unit (CPU), graphics processing unit (GPU), and random access memory (RAM). To determine the carbon intensity of the energy consumed, it is necessary to know where the energy comes from. Each energy source emits a different amount of CO_2 for each kilowatt-hour of energy produced. Coal, petroleum and natural gas are three sources with high carbon intensity, while renewable sources such as solar power and hydroelectricity are characterized by lower carbon intensity. With the combination of energy sources used in the geographical area where the computer is located (the so-called energy mix), the average carbon intensity can be computed. This methodology is based on 43 and has already been used to estimate the carbon emission of machine learning development in 4.

4 Data reduction methods

In this section, we introduce different methods to reduce a dataset. According to the nature of the reduction algorithm, we categorize these methods into four groups: statistic-based methods, which reduce the dataset using probability or statistical concepts; geometry-based methods, which take into account the distances between examples to reduce the dataset; ranking-based methods, which sort the examples according to some criterion and reduce the dataset by selecting the best ones; and wrapper methods, which reduce the dataset during the training process.

4.1 Statistic-based methods

In this subsection, we introduce two data reduction methods that use concepts from probability and statistics to extract a reduced dataset $\mathscr{D}_R$ from $\mathscr{D}$.

Stratified Random Sampling (SRS). The simplest method for data reduction is Stratified Random Sampling (SRS), as proposed in 4, where the natural strata are the c classes of $\mathscr{D}$. Given a proportion $p \in [0, 1]$, the algorithm just selects for each class k a random subset $S_k \subset X_k$ with a reduction ratio of p. This ensures that $\mathscr{D}_R$ has the same class balance as $\mathscr{D}$. The pseudocode for SRS is shown in Algorithm 1.

Algorithm 1. SRS: Stratified Random Sampling

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$
Result: $\mathscr{D}_R = (S, g)$
1 for $k = 1, \cdots, c$ **do**
2 Set the class k as $X_k = \{x \in X : f(x) = k\}$;
3 Set the number of examples to be selected as $n_k = \lfloor p \cdot |X_k| \rfloor$;
4 Select a random subset $S_k \subset X_k$ with $|S_k| = n_k$;
5 Set $S = \mathsf{U}_{k=1,\cdots,c} S_k$;
6 Set $g = f|_S$;

ProtoDash Selection (PRD). ProtoDash Selection (PRD)[11] is an algorithm based on the concept of maximum mean discrepancy (MMD), which measures the dissimilarity between two probability distributions by comparing finite samples. Given a set of indices $I = \{1, \cdots, n_A\}$, let $A = \{a_i \in \mathbb{R}^d\}_{i=1}^{n_A}$ be the sample. Given a subset of indices $L \subset I$, a vector of non-negative weights $w = (w_1, \cdots, w_{n_A})^T$ with $w_j = 0 \ \forall j \notin L$, and a kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$, the empirical maximum mean discrepancy between A and $B = \{a_j : j \in L\}$ is:

$$\widehat{MMD}(K, A, B, w) = \frac{1}{n_A^2} \sum_{i,j \in I} K(a_i, a_j) - \frac{2}{n_A} \sum_{j \in L} w_j \sum_{i \in I} K(a_i, a_j) + \sum_{i,j \in L} w_i w_j K(a_i, a_j) \tag{13}$$

The aim of ProtoDash Explainer (PD) is to find a subset $L \subset I$ with size $|L| = m$ and a vector of weights $w = (w_1, \cdots, w_{nA})^T$ that minimize $\widehat{MMD}(K, A, B, w)$, which is equivalent to maximizing Equation (14).

$$l(w) = w^T \mu - \frac{1}{2} w^T K w \tag{14}$$

being $\mu_j = \frac{1}{n_A} \sum_i K(a_i, a_j)$ the j-th component of the vector μ and $K_{i,j} = K(a_i, a_j)$ the (i, j)-th component of the matrix K. Finding such an optimal subset L is infeasible in practice, and PD algorithm helps us to find an approximate solution heuristically. It starts by setting $L = \emptyset$ and $w_j = 0 \ \forall \ j \in I$. Each iteration of PD consists of two steps. In the first step, the index $j_0 \notin L$ that takes the maximum value in $g = \nabla l(w) = \mu - Kw$ is selected. In the second step, the set of weights w is updated to maximize $l(w)$, subject to $w_j \geq 0 \ \forall \ j \in I$ and $w_j = 0 \ \forall \ j \notin L$. The algorithm ends when $|L| = m$ and the output subset is $B = \{a_j \in A : j \in L\}$. PD does not necessarily give an optimal solution, but it is shown in 11 that the quality of the approximate solution is lower bounded by a fraction of the quality of the optimal solution. The pseudocode for the PD can be seen in Algorithm 2.

Algorithm 2. PD: ProtoDash Explainer

Data: $A = \{a_i\}_{i=1}^{n_A} \subset \mathbb{R}^d$, $K : A \times A \to \mathbb{R}$, $1 \leq m \leq n_A$
Result: $B \subset A$
1 Set $I = \{1, \cdots, n_A\}$ the set of indices in A;
2 Set $L = \emptyset$ the set of selected indices;
3 Set $\mathsf{K}_{i,j} = K(a_i, a_j) \ \forall i, j \in I$;
4 Set $\mu_j = \frac{1}{n_A} \sum_i K(a_i, a_j) \ \forall j \in I$;
5 Set $w_j = 0 \ \forall \ j \in I$;
6 Define $l(w) = w^T \mu - \frac{1}{2} w^T \mathsf{K} w$;
7 Define $\nabla l(w) = \mu - \mathsf{K} w$;

8 while $|L| < m$ **do**
9 Set $g = \nabla l(w)$;
10 Set $j_0 = \arg\max_{j \in \mathbb{N} \setminus L} g_j$;
11 Update $L = L \cup \{j_0\}$;
12 Solve $\xi = \arg\max_w l(w)$ subject to $w_j \geq 0\ \forall j \in I$, $w_j = 0\ \forall j \notin L$;
13 Update $w = \xi$;
14 Set $B = \{a_j \in A : j \in L\}$;

Then, the ProtoDash Selection algorithm (PRD) just applies PD to each class X_k, finding a subset $S_k \subset X_k$ with n_k examples that approximately minimizes $\widehat{MMD}(K, X_k, S_k, w)$. The reduced dataset $\mathscr{D}_R$ is the union of all S_k. The pseudocode for PRD can be seen in Algorithm 3.

Algorithm 3. PRD: ProtoDash Selection

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$, $K : X \times X \to \mathbb{R}$
Result: $\mathscr{D}_R = (S, g)$
1 for $k = 1, \cdots, c$ **do**
2 Set the number of examples to be selected as $n_k = \lfloor p \cdot |X_k| \rfloor$;
3 Apply Algorithm 2 with $A = X_k$ and $m = n_k$ to get $S_k = B$
4 Set $S = \bigcup_{k=1,\cdots,c} S_k$;
5 Set $g = f|_S$;

4.2 Geometry-based methods

In this subsection, we introduce three data reduction methods that use the distances between the examples in $\mathscr{D}$ to find a reduced dataset $\mathscr{D}_R$.

Clustering Centroids Selection (CLC). Clustering is a branch of unsupervised machine learning whose task is to partition a dataset into groups or clusters, where objects within the same cluster are highly similar and distinct from those in other clusters. The goal is to discover patterns or structures without prior knowledge or labels. Clustering algorithms produce different partitions depending on the approach. For a comprehensive overview of clustering, we refer to 44.

The Clustering Centroids Selection (CLC) algorithm uses k-means, one of the most-known clustering algorithms, for data reduction. This idea was stated in 45 and 46, among others. The general idea is to apply k-means on each class of $\mathscr{D}$ and include the resulting centroids in $\mathscr{D}_R$. This is the only data reduction method in this paper where the reduced dataset $\mathscr{D}_R$ is not necessarily a sub-dataset of $\mathscr{D}$. This method is easy to understand but can be computationally expensive for large datasets, unstable, and sensitive to outliers, as stated in 47,48. The pseudocode for CLC can be read in Algorithm 4.

Algorithm 4. CLC: Clustering Centroids Selection

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$
Result: $\mathscr{D}_R = (S, g)$
1 for $k = 1, \cdots, c$ **do**
2 Set the class k as $X_k = \{x \in X : f(x) = k\}$;
3 Set the number of examples to be selected as $n_k = \lfloor p \cdot |X_k| \rfloor$;
4 Apply k-Means on X_k with n_k clusters;
5 Select the set of centroids S_k;
6 Set $S = \bigcup_{k=1,\cdots,c} S_k$;
7 for $k = 1, \cdots, c$ **do**
8 Set $g(x) = k$ for each $x \in S_k$;

Maxmin Selection (MMS). Maxmin Selection (MMS) uses the distances between the examples to ensure that $\mathscr{D}_R$ is evenly spaced. It has been applied in 13 to reduce datasets and in 49 to create efficient data descriptors.

For each class k, the first step is to pick a random example $x_r \in X_k$ and add it to S_k. Then, given a distance function $d : X \times X \to \mathbb{R}^+$, each step picks the example in $X_k \setminus S_k$ that maximizes the function:

$$D : X_k \setminus S_k \to \mathbb{R}$$

$$x \mapsto \min_{x' \in S_k} d(x, x')$$

This is repeated until S_k has the required size. This method produces a subset that covers up well the dataset, but tends to pick extreme or outlier points. The pseudocode for MMS can be read in Algorithm 5.

Algorithm 5. MMS: Maxmin Selection

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$, $d : X \times X \to \mathbb{R}^+$
Result: $\mathscr{D}_R = (S, g)$
for $k = 1, \cdots, c$ **do**
  Set the class k as $X_k = \{x \in X : f(x) = k\}$;
  Set the number of examples to be selected as $n_k = \lfloor p \cdot |X_k| \rfloor$;
  Select a random example $x_r \in X_k$;
  Set $S_k = \{x_r\}$;
  while $|S_k| < n_k$ **do**
    Set $x = \arg\max_{x \in Xk \setminus Sk} \min_{x' \in Sk} d(x, x')$;
    Update $S_k = S_k \cup \{x\}$;
Set $S = \bigcup_{k=1,\cdots,c} S_k$;
Set $g = f|_S$;

Distance-Entropy Selection (DES). Distance-Entropy Selection (DES)[14] is a data reduction method that tries to ensure that the resulting dataset $\mathscr{D}_R$ has relevant examples. It is based on a distance-entropy indicator that measures how informative the different examples are for the classification task.

The algorithm begins by selecting a subset $X_{base} \subset X$, known as the base data. In our implementation, we have decided to select the base data via SRS, using a proportion $p_{base} < p$. The base data is used to calculate a prototype p_k for each class $k = 1, \cdots, c$. In our case, the prototype p_k is defined as the average of all points of class k in X_{base}. The algorithm then calculates the distances between the prototypes and all points in $X_{pool} = X \setminus X_{base}$, called the pool data. The distances $d_k = d(x, p_k)$ associated to $x \in X_{pool}$ are transformed into a probability distribution by the softmax function, with formula:

$$\text{Softmax}(d_k) = \frac{e^{d_k}}{\sum_{j=1}^{c} e^{d_j}} \tag{15}$$

The information entropy of this distribution is called the distance-entropy indicator of x:

$$E(x) = -\sum_{k=1}^{c} \text{Softmax}(d_k) \cdot \log_2 \text{Softmax}(d_k) \tag{16}$$

Finally, the reduced dataset $\mathscr{D}_R$ is formed by all the examples in X_{base} and the examples from X_{pool} with the highest values for the distance-entropy indicator. The pseudocode for DES can be seen in Algorithm 6.

Algorithm 6. DES: Distance-Entropy Selection

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$, $p_{base} \in [0, p]$, $d : X \times X \to \mathbb{R}^+$
Result: $\mathscr{D}_R = (S, g)$
for $k = 1, \cdots, c$ **do**
  Set the class k as $X_k = \{x \in X : f(x) = k\}$;
  Set the number of examples to be included in the base data as $n_{k,base} = \lfloor p_{base} \cdot |X_k| \rfloor$;
  Select a random subset $S_{k,base} \subset X_k$ with $|S_k| = n_{k,base}$;
  Calculate a prototype p_k for the examples in $S_{k,base}$;

6 Set $X_{base} = \bigcup_{k=1,\cdots,c} S_{k,base}$;
7 Set $X_{pool} = X \setminus X_{base}$;
8 for $x \in X_{pool}$ **do**
9 **for** $k = 1, \cdots, c$ **do**
10 Calculate the distance $d_k = d(x, p_k)$;
11 **for** $k = 1, \cdots, c$ **do**
12 Transform the distances into probabilities with $\text{Softmax}(d_k) = \frac{e^{d_k}}{\sum_{j=1}^{c} e^{d_j}}$;
13 Calculate the distance-entropy indicator as $E(x) = -\sum_{k=1}^{c} \text{Softmax}(d_k) \cdot \log_2 \text{Softmax}(d_k)$;
14 Set the number of examples to be added as $n_{add} = \lfloor p \cdot |X| \rfloor - |X_{base}|$;
15 Set $X_{add} \subset X_{pool}$ containing the n_{add} examples in X_{pool} with higher values for E;
16 Set $S = X_{base} \cup X_{add}$;
17 Set $g = f|_S$;

To justify why the best examples are those with higher entropies, the authors of 14 used the following reasoning. Suppose that an example $x \in X_{pool}$ is closer to one prototype p_k than to all the others. In that case, the distance-entropy indicator $E(x)$ will be low and x is likely to be classified in class k. By contrast, items with high entropy are informative because they are different from all prototypes and are not so easy to classify. Note that the examples from the pool data are selected regardless of their class, so it is possible that the reduction ratio of each class is different from the global reduction ratio. For this reason, we recommend selecting the base data using a sufficiently high p_{base} to make sure that all classes are well represented and then complementing the base data with the most informative examples from the pool data.

4.3 Ranking-based methods

In this subsection, we describe three methods that are based on a ranking system. Basically, these methods assign a score to the examples based on a particular criterion, sort them according to their score, and then select the best-ranked examples from this sorted list.

PH Landmarks Selection (PHL). PH Landmarks Selection (PHL)[15] is a subset selection method based on the concept of persistent homology. Roughly speaking, persistent homology is a common technique in topological data analysis (TDA) that builds a filtration of simplicial complexes over the dataset examples (such as the Vietoris-Rips filtration) and computes, for each $n \geq 0$, the evolution of certain mathematical features (called n-dimensional homology classes) along the filtration. The n-dimensional persistent homology of a data set can be encoded in a barcode $B_n = \{[b_i, d_i]\}_{i=1}^{I_n}$ that has a bar $[b, d)$ for each n-dimensional homology class that first appears in the stage b of the filtration and disappears at stage d.

PHL algorithm orders the examples in each class by evaluating how their removal changes its persistent homology. Given an example $x \in X_k$, the first step is to find its δ-neighbourhood $\Delta_x = \{\tilde{x} \in X_k \setminus \{x\} : d(x, \tilde{x}) \leq \delta\}$. If $|\Delta_x| \leq 2$, x is considered a super-outlier. If x is not a super-outlier, a Vietoris-Rips filtration is built over Δ_x and its persistent homology is computed for $n = 0, 1, 2$. Then, the *PH outlierness* of x is:

$$out_{PH}^{0,1,2}(x) = \max_{n=0,1,2} \max_{i} \{d_i - b_i : [b_i, d_i) \in B_n(\Delta_x)\} \tag{17}$$

A restricted version of PH outlierness that can be used in practice is:

$$out_{PH}^{1}(x) = \max_{i} \{d_i - b_i : [b_i, d_i) \in B_1(\Delta_x)\} \tag{18}$$

We denote the PH outlierness as out_{PH}. Small values for $out_{PH}(x)$ indicate that the persistent homologies of X_k and $X_k \setminus \{x\}$ are similar. The theoretical motivation for this statement can be found in 15. Two strategies are proposed to select examples from X_k. On the one hand, we can choose the examples that are not super-outliers and have smaller values for $out_{PH}(x)$, called representative landmarks. On the other hand, we can choose those with higher values for $out_{PH}(x)$, called vital landmarks. In case there are not enough examples in X_k that are not super-outliers to be chosen, the subset can include some random super-outliers. The reduced dataset $\mathscr{D}_R$ is generated by applying this procedure for each class. Algorithm 7 shows the pseudocode for PHL selection.

Algorithm 7. PHL: PH Landmarks Selection

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$, $d : X \times X \to \mathbb{R}^+$, $\delta > 0$, $o_{type} \in$ { multidimensional, restricted},
$l_{type} \in$ {representative, vital}
Result: $\mathscr{D}_R = (S, g)$
for $k = 1, \cdots, c$ **do**
  Set the class k as $X_k = \{x \in X : f(x) = k\}$;
  Set the number of examples to be selected as $n_k = \lfloor p \cdot |X_k| \rfloor$;
  Set $O_k = \emptyset$ the set of super-outliers;
  if $o_{type} = multidimensional$ **then**
    Set $out_{PH} \equiv out_{PH}^{0,1,2}$;
  else
    Set $out_{PH} \equiv out_{PH}^{1}$;
  for $x \in X_k$ **do**
    Find $\Delta_x = \{\tilde{x} \in X_k \setminus \{x\} : d(x, \tilde{x}) \leq \delta\}$;
    if $|\Delta_x| > 2$ **then**
      Compute the Vietoris-Rips filtration of Δ_x for $n = 0, 1, 2$;
      Compute $out_{PH}(x)$;
    else
      Update $O_k = O_k \cup \{x\}$;
  if $n_k \leq |X_k \setminus O_k|$ **then**
    if $l_{type} = representative$ **then**
      Set the subset $S_k \subset X_k \setminus O_k$ with the n_k lowest values for out_{PH};
    else
      Set the subset $S_k \subset X_k \setminus O_k$ with the n_k highest values for out_{PH};
  else
    Select a random subset $R_k \subset O_k$ with $|R_k| = |X_k| - n_k$;
    Set $S_k = X_k \setminus R_k$;
Set $S = \bigcup_{k=1,\cdots,c} S_k$;
Set $g = f|_S$;

Numerosity Reduction by Matrix Decomposition (NRMD). Numerosity Reduction by Matrix Decomposition (NRMD)[16] is a method that leverages matrix decomposition to rank examples in a dataset $\mathscr{D} = (X, f)$. To use this method, it is necessary to use the tabular representation of $\mathscr{D}$ that we saw in Subsection 3.2. The matrix $\mathbf{X}$ contains all the examples in X, and the submatrix $\mathbf{X}_k$ contains all the examples in X_k.

Given a matrix $A \in \mathbb{R}^{n \times d}$ with rows $a_1, \cdots, a_n$, a decomposition is just a factorization $A = UV$, where $U \in \mathbb{R}^{n \times r}$, $V \in \mathbb{R}^{r \times d}$ with rows $v_1, \cdots, v_r$, and $r = \min\{n, d\}$. Some typical matrix decompositions are singular value decomposition (SVD)[50], non-negative matrix factorization (NMF)[51], PLU decomposition[52], QR decomposition[52], dictionary learning (DICL)[53], supervised principal component analysis (SPCA)[54] and Fisher linear discriminant analysis (FLDA)[55]. From this decomposition, each row of A is assigned a score based on its similarity to the rows of V. The matrix Σ, with $\Sigma_{i,j} = |\cos(a_i, v_j)|_\varepsilon$, stores all the similarities ($|\cdot|_\varepsilon$ denotes the maximum between the absolute value and a certain $\varepsilon > 0$). The final score vector is $-\log(\Sigma)w$, where $w \in \mathbb{R}^r$ is a weight vector given by $w_i = \frac{1/\lambda_i}{\sum_i 1/\lambda_i}$ when the decomposition is based on eigenvalues (as in SVD, SPCA and FLDA) and by $w_i = \frac{2i}{r(r+1)}$ otherwise (as in NMF, DL, PLU and QR decompositions). Algorithm 8 shows the procedure to calculate scores from a matrix decomposition.

Algorithm 8. Calculate scores from a matrix

Data: $A \in \mathbb{R}^{N \times d}$, $d_{type} \in$ {SVD, NMF, PLU, QR, DL, SPCA, FLDA}
Result: $s \in \mathbb{R}^N$
Set $r = \min\{N, d\}$;
Calculate $A = UV$ using the d_{type} decomposition;
Set $\tilde{A}$ as the normalization of A;

4 Set $\tilde{V}$ as the normalization of V;
5 if $d_{type} \in \{SVD, SPCA, FLDA\}$ **then**
6 Set $w \in \mathbb{R}^r$ with $w_i = \frac{1/\lambda_i}{\sum_i 1/\lambda_i}$, being $\lambda_1 > \cdots > \lambda_r$ the eigenvalues given by the decomposition;
7 else
8 Set $w \in \mathbb{R}^r$ with $w_i = \frac{2i}{r(r+1)}$;
9 Calculate the scores vector $s = -\log(|\tilde{A}\tilde{V}^T|_e)w$;

Given a specific decomposition type, the NRMD method computes scores for all matrices $\mathbf{X}_1, \cdots, \mathbf{X}_c$. As a result, there exists a score $s_{\mathbf{X}}(x)$ for each $x \in \mathbf{X}$. In addition, the method calculates scores $s_D(x)$ for the matrix $D = [\mathbf{X}|E]$, where $E \in \mathbb{R}^{N \times c}$ represents the one-hot encoding matrix of f. In this encoding, $E_{ij} = 1$ if $f(x_i) = j$ and $E_{ij} = 0$ otherwise. The final score for an example $x \in X$ is $s(x) = s_{\mathbf{X}}(x) \cdot s_D(x)$. The dataset $\mathscr{D}_R$ is finally formed by the examples with the highest values for s, which are considered to be the most useful in terms of internal representation and discrimination between classes. The pseudocode for NRMD can be seen in Algorithm 9.

Algorithm 9. NRMD: Numerosity Reduction by Matrix Decomposition

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$, $d_{type} \in$ {SVD, NMF, PLU, QR, DICL, SPCA, FLDA}
Result: $\mathscr{D}_R = (S, g)$
1 Set $n = \lfloor p \cdot |X| \rfloor$;
2 for $k = 1, \cdots, c$ **do**
3 Set the class k as $X_k = \{x \in X : f(x) = k\}$;
4 Calculate s_{X_k} as the result of applying Algorithm 8 to the matrix $\mathbf{X}_k$;
5 Obtain $s_{\mathbf{X}}$ merging the score vectors $s_{X_1}, \cdots, s_{X_c}$;
6 Calculate E as the one-hot encoding matrix of f;
7 Set $D = [\mathbf{X}|E]$;
8 Calculate s_D as the result of applying Algorithm 8 to the matrix D;
9 Calculate the final scores $s = s_{\mathbf{X}} \odot s_D$;
10 Set the subset $S \subset X$ with the n highest scores in s;
11 Set $g = f|_S$;

4.4 Wrapper methods

All the data reduction methods described in the previous subsections are intended to be applied before training $\mathscr{N}$, since they only need the information from $\mathscr{D}$ itself to extract $\mathscr{D}_R$. In this section, we describe a method that uses the information obtained during the training of $\mathscr{N}$ to reduce $\mathscr{D}$. This means that the data reduction is not done before training, but is wrapped in the training process itself.

Forgetting Events Selection (FES). Forgetting Events Selection (FES) is a data reduction method that leverages the evolution of accuracy throughout DL models training. During the training process, an example $x \in X$ can be well classified after some epochs (we say that its current accuracy is $a_x = 1$) and misclassified after others (we say that $a_x = 0$). If $a_x = 0$ after epoch $t - 1$ but $a_x = 1$ after epoch t, we say that x has undergone a *learning event*. Conversely, if $a_x = 1$ after epoch $t - 1$ but $a_x = 0$ after epoch t, x has undergone a *forgetting event*. *Unforgettable examples* are those with $a_x = 1$ that never had a forgetting event.

The experiments in 17 show that unforgettable examples have less impact on network training than those that go through several forgetting events and that they can be removed from the training dataset without significantly affecting the model performance. Based on this idea, the FES algorithm counts how many forgetting events each example undergoes during training and discards the examples with the lowest number of forgetting events. Examples that never get well classified are assigned an infinite number of forgetting events.

Following the ideas from 56, our FES implementation only counts the forgetting events during the first $e_{initial}$ epochs of the training process. At that point, the algorithm reduces $\mathscr{D}$ by selecting the examples with more forgetting events and continues the training process for the remaining epochs using only $\mathscr{D}_R$ as a training dataset. To ensure that all classes are well represented in $\mathscr{D}_R$, the selection is made class by class. Algorithm 10 shows how to apply FES selection during the training of a DL model $\mathscr{N}$.

Algorithm 10. FES: Forgetting Events Selection

Data: $\mathscr{D} = (X, f)$, $p \in [0, 1]$, $\mathscr{N}: \mathbb{R}^d \rightarrow \{1, \cdots, c\}$, $e_{initial}$, e_{total}
Result: $\mathscr{D}_R = (S, g)$, $\mathscr{N}$
for $x \in X$ **do**
  Set the current accuracy $a_x = 0$;
  Set the number of forgetting events $f_x = 0$;
for $e = 1, \cdots, e_{initial}$ **do**
  Perform a training epoch on $\mathscr{N}$ using $\mathscr{D}$;
  for $x \in X$ **do**
    if $\mathscr{N}(x) = f(x)$ **then**
      Update $a_x = 1$;
    else if $a_x = 1$ **then**
      Update $f_x = f_x + 1$;
      Update $a_x = 0$;
for $x \in X$ **do**
  if $a_x = f_x = 0$ **then**
    Update $f_x = \infty$
for $k = 1, \cdots, c$ **do**
  Set the class k as $X_k = \{x \in X : f(x) = k\}$;
  Set the number of examples to be selected as $n_k = \lfloor p \cdot |X_k| \rfloor$;
  Select a subset $S_k \subset X_k$ with the n_k highest values for f_x;
Set $S = \bigcup_{k=1,\cdots,c} S_k$;
Set $g = f|_S$;
for $e = e_{initial} + 1, \cdots, e_{total}$ **do**
  Perform a training epoch on $\mathscr{N}$ using $\mathscr{D}_R = (S, g)$;

5 Experiments

In this section, we present the datasets used for the experiments, the parameter settings, the setup for the experiments, and finally, the results obtained. The source code of the experiments is available in the GitHub repository[9].

5.1 Experiments for tabular data classification

In this subsection, we describe the two experiments that we have developed to analyze the utility of data reduction for classification tasks with tabular datasets. In the first place, we detail the methodology to apply the different data reduction methods to a dataset and measure its efficiency, representativeness and performance. Then, we give some details about the two datasets we used. Finally, we show the results obtained for both experiments and discuss the main conclusions.

5.1.1 Datasets for classification

The two datasets that we have used in our experiments are:

Collision dataset This tabular dataset, generated for article 57, was provided by their authors. It can be downloaded from the repository[9], which also contains the code for the experiments and our results. The classification task consists of predicting whether a platoon of vehicles will collide based on features such as the number of vehicles and their speed. The dataset consists of 107,210 examples with 25 numerical features and 2 classes:

- collision = 1, with 69,348 examples.
- collision = 0, with 37,862 examples.

We decided to use this dataset to test the usefulness of data reduction methods to reduce resource consumption in a task related to safe mobility. Before the experiments, we discarded the two features "N" and "m" since they are constant and do not help us in the classification task.

Dry Bean dataset This dataset (see 58 and 59) was created by taking pictures of 7 different types of dry beans and calculating some geometric features from the images, such as the area, the perimeter and the eccentricity. The classification task consists of predicting the type of dry bean based on these geometric features. The dataset contains 13,611 examples with 16 features and 7 classes:

- Barbunya, with 1,322 examples.
- Bombay, with 522 examples.
- Cali, with 1,630 examples.
- Dermason, with 3,546 examples.
- Horoz, with 1,928 examples.
- Seker, with 2,027 examples.
- Sira, with 2,636 examples.

The classes were encoded from 0 to 6 for the experiments following the listed ordering. We decided to use this dataset to test the usefulness of data reduction methods for classification tasks with several unbalanced classes.

5.1.2 Methodology

The methodology of the experiments for both datasets consists of the following three steps:

1. **Dataset preprocessing:**

   It is a common practice to scale or standardize a dataset before building the DL model because this increases the likelihood that the training process will be fast and will not be conditioned by some features simply due to their greater magnitude[60,61]. In our case, we decided to apply the scikit-learn function *MinMaxScaler*[62].

   Each feature p_j has a maximum value $p_{j,max} = \max_{x_i \in X} p_j(x_i)$ and a minimum value $p_{j,min} = \min_{x_i \in X} p_j(x_i)$. We say that the range of p_j is the interval $[p_{j,min}, p_{j,max}]$ because all its possible feature values lie in it. Rescaling with *MinMaxScaler* is just changing each example $x_i \in X$ by:

   $$x_{i,scaled} = \left(\frac{x_{i,1} - p_{1,min}}{p_{1,max} - p_{1,min}}, \cdots, \frac{x_{i,j} - p_{j,min}}{p_{j,max} - p_{j,min}}, \cdots, \frac{x_{i,d} - p_{d,min}}{p_{d,max} - p_{d,min}} \right)^T \tag{19}$$

   After scaling, the range of all the features is [0, 1]. That means that all of them have similar values and can be compared between them.

2. **Fixing the architecture and hyperparameters:**

   In both experiments, we used a DL model architecture with 10 layers with the following dimensions:

   $$X \xrightarrow{f_1} \mathbb{R}^{50} \xrightarrow{f_2} \mathbb{R}^{45} \xrightarrow{f_3} \mathbb{R}^{40} \xrightarrow{f_4} \mathbb{R}^{35} \xrightarrow{f_5} \mathbb{R}^{30} \xrightarrow{f_6} \mathbb{R}^{25} \xrightarrow{f_7} \mathbb{R}^{20} \xrightarrow{f_8} \mathbb{R}^{15} \xrightarrow{f_9} \mathbb{R}^{10} \xrightarrow{f_{10}} O \tag{20}$$

   All layers except the last one, use the Rectified Linear Unit (ReLU) activation function $\mathrm{ReLU}(x) = \max(0, x)$. In addition, these layers use dropout as a regularization technique. The probability of zeroing a neuron during dropout each time is a hyperparameter called the *dropout probability*, which we have set equal to 0.50 for the experiment with the Collision dataset and equal to 0.25 for the experiment with the Dry Bean dataset. The differences in neural architecture for both experiments are in the last layer and in the output space O.

   For the Collision dataset, the last layer has only one neuron (that is, $O = \mathbb{R}$), with sigmoid activation function $\sigma(x) = 1/(1 + e^{-x})$. The output of $\mathcal{N}$ (also called the logit) for an input x_i is a number $z_i \in [0, 1]$. The predicted class for x_i is:

   $$\mathcal{N}_\theta(x_i) = \begin{cases} 0 & \text{if } z_i < 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{21}$$

To train the network for the Collision dataset, we used the Binary Cross Entropy as a loss function. Given a neural network $\mathcal{N}_\theta$ and a dataset $\mathcal{D}$, it has the following formula:

$$\text{BCELoss}(\theta,\mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N}(f(x_i)\cdot\log(z_i) + (1-f(x_i))\cdot\log(1-z_i)) \tag{22}$$

For the Dry Bean dataset, the last layer has 7 neurons, one for each class (that is, $O = \mathbb{R}^7$). The logits $z_{i,0}$, ⋯, $z_{i,6}$ for an input x_i are transformed into a probability distribution with the softmax activation function, given by $s_{i,k} = \text{Softmax}(z_{i,k}) = e^{z_{i,k}} / \sum_{m=0}^{6} e^{z_{i,m}}$, and the predicted class for x_i is:

$$\mathcal{N}_\theta(x_i) = \arg\max_{k=0,\cdots,6} s_{i,k} \tag{23}$$

To train the network for the Dry Bean dataset, we used the Categorical Cross Entropy loss function. If we denote $y_{i,k} = 1$ if $f(x_i) = k$ and $y_{i,k} = 0$ otherwise, the formula of Categorical Cross Entropy is:

$$\text{CCELoss}(\theta,\mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{6} w_k \cdot y_{i,k} \cdot \log(s_{i,k}) \tag{24}$$

Here, $w_k = N/N_k$ is a weight assigned to class k to give more importance to the least populated classes and prevent a bias toward the most populated ones.

In both cases, we used the Adam optimizer[63] to minimize the loss function, specifying a learning rate of $\gamma = 0.001$ and leaving the default values for the rest of the required hyperparameters.

Regarding the other learning hyperparameters, the network for the Collision dataset was trained for $n_e = 600$ epochs with a batch size of $\beta = 1,024$. When the FES reduction was applied, the model was trained for $n_i = 200$ epochs with the full training dataset and the remaining 400 epochs with the reduced dataset. For the Dry Bean dataset, the number of training epochs was $n_e = 150$ ($n_i = 50$ for the first part of training with FES reduction) and the batch size was of $\beta = 32$.

3. **Data reduction and model training:** Now that the dataset is scaled and the neural architecture and the learning hyperparameters have been set, in this step we analyze how the data reduction methods affect the efficiency and performance of the training of a neural network. This step is divided into the following 4 sub-steps:

    - **Train-Test dataset split:** The dataset $\mathcal{D}$ is randomly split into a training dataset $\mathcal{D}_{train}$ and a test dataset $\mathcal{D}_{test}$. The DL model is trained using $\mathcal{D}_{train}$ and its performance is evaluated using $\mathcal{D}_{test}$. The test dataset contains a proportion $p_{test} \in (0, 1)$ of the total number of examples in $\mathcal{D}$. For both experiments, we set $p_{test} = 0.25$.

    - **Training with no reduction:** In this step the DL model is trained using the whole training dataset $\mathcal{D}_{train}$ with no reduction, and then, the computation time and carbon emission of the training are calculated. After that, the model is used to classify the test dataset $\mathcal{D}_{train}$ and the accuracy, macro average precision, macro average recall and macro average F1-score are calculated.

    - **Training + reduction for non-wrapper methods:** In this step, $\mathcal{D}_{train}$ is reduced getting $\mathcal{D}_{train,R}$ as a result, and the ε-representativeness of $\mathcal{D}_{train,R}$ with respect to $\mathcal{D}_{train}$ is computed. The model is then trained for n_e epochs using $\mathcal{D}_{train,R}$, and the total computation time and carbon emission of the reduction and the training are calculated. The model is used to classify $\mathcal{D}_{test}$ as in the previous step. This is repeated for each non-wrapper data reduction method (all but FES) and for each reduction percentage $p \in \{0.1, 0.2, \cdots, 0.9\}$.

    - **Training + reduction for FES:** In this step, the DL model is trained using $\mathcal{D}_{train}$ for the first n_i epochs. After applying the FES reduction, the model is trained for the remaining epochs using $\mathcal{D}_{train,R}$, and the ε-representativeness of $\mathcal{D}_{train,R}$ with respect to $\mathcal{D}_{train}$ is also computed. The total computation time and carbon emission of the training and the reduction are computed. The model is used to classify $\mathcal{D}_{test}$ as in the previous steps. This step is repeated for each reduction percentage $p \in \{0.1, 0.2, \cdots, 0.9\}$.

Step 3 is repeated 10 times to test how the data reduction works for different train-test splits and mitigate possible overfitting or bias caused by a specific split of the dataset.

Algorithm 11 shows the experiment pipeline for tabular data classification.

Algorithm 11. Pipeline of the Experiments for Tabular Data Classification

Data: $\mathscr{D} = (X, f)$
1 Dataset Preprocessing;
2 Scale $\mathscr{D}$ using MinMaxScaler;
3 Fixing the architecture and hyperparameters;
4 Set a test size proportion $p_{test} \in (0, 1)$;
5 Set an architecture and create the DL model $\mathscr{N}$;
6 Set a loss function $\mathscr{L}: \Theta \rightarrow \mathbb{R}^+$;
7 Set an optimization algorithm to minimize $\mathscr{L}$ and its associated hyperparameters;
8 Set a regularization technique and its associated hyperparameters;
9 Set a number of training epochs $n_e \in \mathbb{N}$;
10 Set a number of initial training epochs for FES reduction $n_i \in \mathbb{N}$, with $n_i < n_e$;
11 Set a batch size $\beta \in \mathbb{N}$;
12 Set a number of iterations $n_{iter} \in \mathbb{N}$;
13 Data reduction and model training;
14 for $i = 1$ **to** *n_iter* **do**
15 **Train-Test dataset split**;
16 Set $N_{test} = \lfloor p_{test} \cdot N \rfloor$;
17 Split $\mathscr{D}$ into $\mathscr{D}_{train}$ and $\mathscr{D}_{test}$, being the size of $\mathscr{D}_{test}$ equal to N_{test};
18 **Training with no reduction**;
19 Train $\mathscr{N}$ for n_e epochs using $\mathscr{D}_{train}$;
20 Calculate the computing time and carbon emission of the training;
21 Validate the model with $\mathscr{D}_{test}$ and calculate the accuracy, macro average precision, macro average recall and macro average F1-score;
22 **Training + reduction for non-wrapper methods:**;
23 **foreach** *non-wrapper method* **do**
24 **for** $p \in \{0.1, 0.2, \cdots, 0.9\}$ **do**
25 Get $\mathscr{D}_{train,R}$ as the reduced dataset of $\mathscr{D}_{train}$ with the corresponding data reduction method and the reduction ratio p;
26 Calculate the ε-representativeness of $\mathscr{D}_{train,R}$ respect to $\mathscr{D}_{train}$;
27 Train $\mathscr{N}$ for n_e epochs using $\mathscr{D}_{train,R}$;
28 Calculate the computing time and carbon emission of the reduction and the training;
29 Validate $\mathscr{N}$ using $\mathscr{D}_{test}$ and calculate the accuracy, macro average precision, macro average recall and macro average F1-score;
30 **Training + reduction for FES**;
31 **for** $p \in \{0.1, 0.2, \cdots, 0.9\}$ **do**
32 Train $\mathscr{N}$ for n_i epochs using $\mathscr{D}_{train}$;
33 Get $\mathscr{D}_{train,R}$ as the reduced dataset of $\mathscr{D}_{train}$ with FES reduction and the reduction ratio p;
34 Calculate the ε-representativeness of $\mathscr{D}_{train,R}$ respect to $\mathscr{D}_{train}$;
35 Train $\mathscr{N}$ for $n_e - n_i$ epochs using $\mathscr{D}_{train,R}$;
36 Calculate the computing time and carbon emission of the reduction and the training;
37 Validate $\mathscr{N}$ using $\mathscr{D}_{test}$ and calculate the accuracy, macro average precision, macro average recall and macro average F1-score;

5.1.3 Results and discussion

All results in this section are the median values after 10 repetitions. We chose to use the median for this experiment because it provides a robust measure of central tendency that is less affected by outliers, ensuring that our analysis is not biased by extreme values.

Collision Dataset The median results we obtained for the efficiency metrics (computing time and carbon emission) can be seen in Figure 2 and Figure 3. The first thing that we can note is that both metrics express the same information since they are almost proportional. Approximately, each minute of computation during our experiment emitted 0.22 g of CO_2 into the atmosphere. This is because data reduction generally takes up a very small part of the total computing time, and the training time of the DL model is proportional to the dataset size. In general, the use of data reduction methods before model training helped to reduce the computing time and the carbon emission of model building with respect to the reference case (when the model is trained over the whole training dataset), but we find three particular exceptions. When the CLC reduction method is applied with a reduction ratio of 80% or superior, the efficiency metrics are worse than those obtained for the reference case. We have the same situation for MMS and DES when the reduction

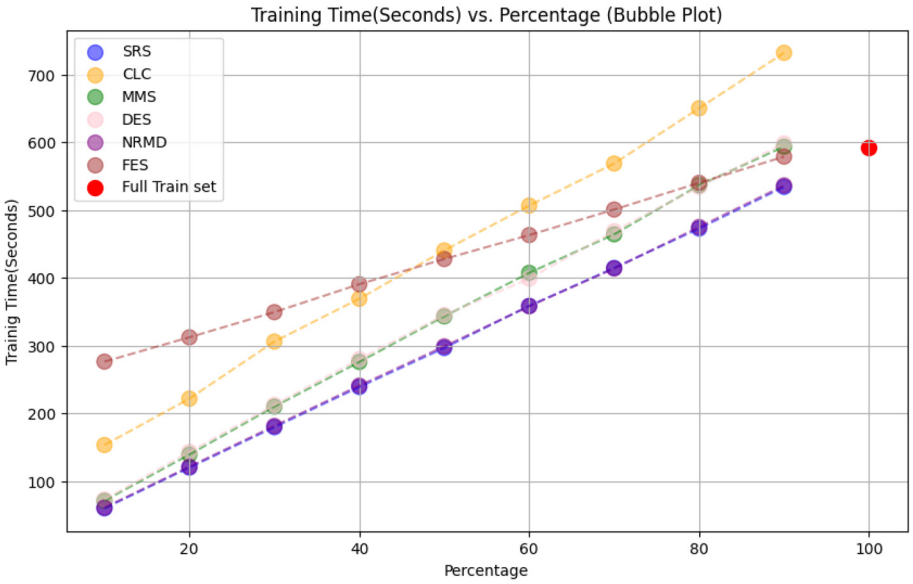

Figure 2. Collision: Reduction + training time.

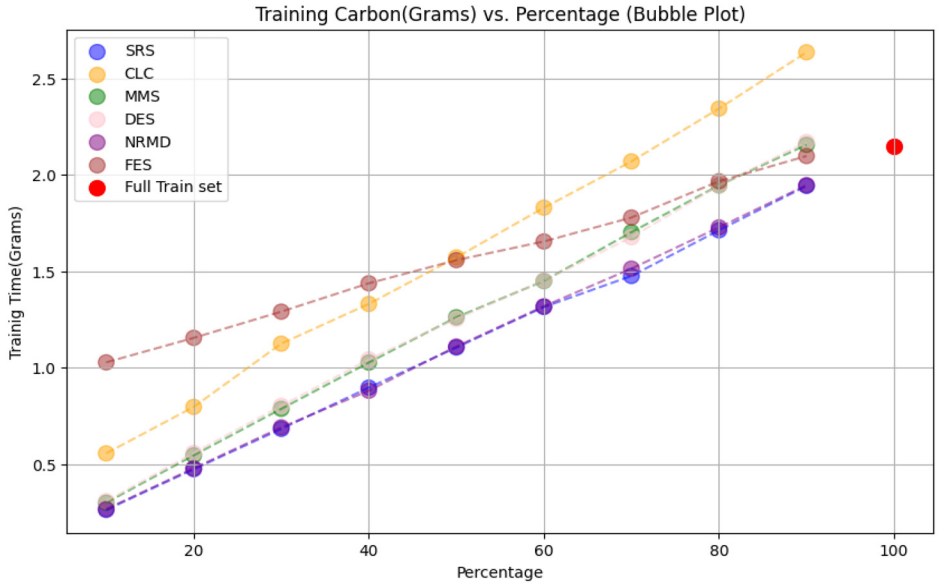

Figure 3. Collision: Reduction + training carbon.

ratio is equal to 90%. These exceptions suggest that, if we extract a reduced dataset with too many examples, it is possible that the time needed to compute the reduction does not compensate for the time saved during the network training. Because of that, if the size of the dataset is equal to or larger than the size of the Collision dataset, we recommend applying CLC, MMS and DES only for reductions with small reduction ratios. In all the other data reduction methods, we can observe that the efficiency metrics always improve those of the reference case. In terms of efficiency, the top two data reduction methods are SRS and NRMD, with similar results in both the computation time and carbon emission. We observed that the efficiency of NRMD reduction depends on the type of matrix decomposition selected. We decided to use SVD decomposition by trial and error, but the results may be different if we select another decomposition type.

Regarding the ε-representativeness of the reduced datasets with respect to the whole training dataset, the median results can be seen in Figure 4. The first thing we can observe is that MMS reduction is always the best at preserving the ε-representativeness for all the possible reduction ratios, which seems natural if we recall the definition of ε-representativeness and the way the MMS method selects each new example in the reduced dataset. CLC reduction also gives us datasets with good ε values. In contrast, NRMD is the data reduction method with the highest ε values for all the possible reduction ratios.

The results on accuracy, macro average precision, macro average recall and macro average F1-score can be seen in Figure 5, Figure 6, Figure 7 and Figure 8 respectively. Regarding accuracy, we can see that the model trained with the whole training dataset has a median success probability of 91%. In general, all the data reduction methods work very well for this dataset. In fact, there are many specific cases where the model obtained with a reduced dataset performs better on the test dataset than the one trained with the full training dataset. We can observe that when the reduction ratio is above 50% the best performing method is FES, while in other cases it is DES. Most of the compared methods manage to maintain accuracy almost intact despite the significant reduction in training size. If we look at the results when we reduce the training dataset to 10% of its size, the model trained after applying DES loses 1.8% of accuracy, while the loss is less than 3% when CLC and SRS are applied. In all cases, this loss in accuracy is more or less linear for all methods except for FES. In this case, the accuracy remains stable while the reduction ratio is high, but it undergoes a drastic drop when a high percentage of examples is removed.

We can observe a similar situation when analyzing the macro average precision. FES is the best method to preserve this metric (even improving the reference case) when the reduction ratio is greater than 50%, while for

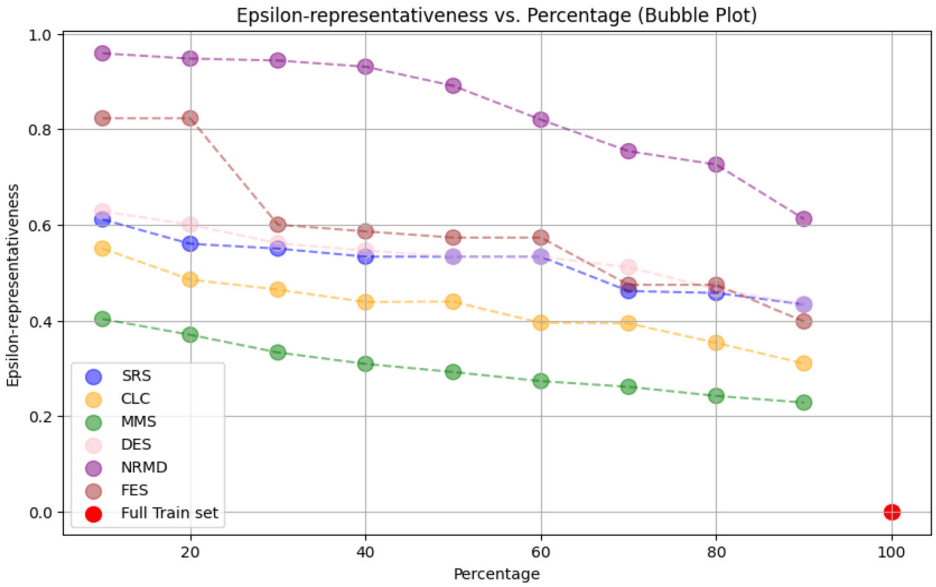

Figure 4. Collision: Reduction + ε-representativeness.

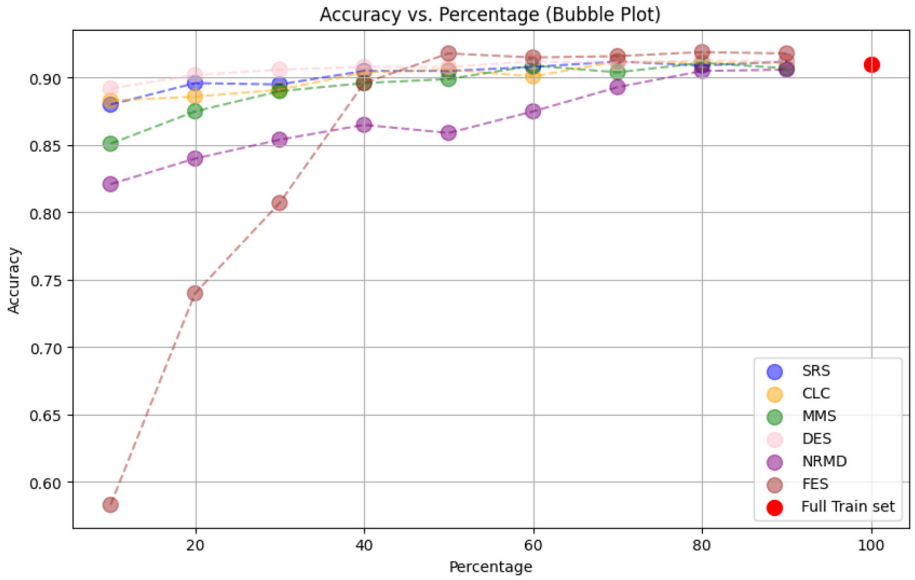

Figure 5. Collision: Reduction + Accuracy.

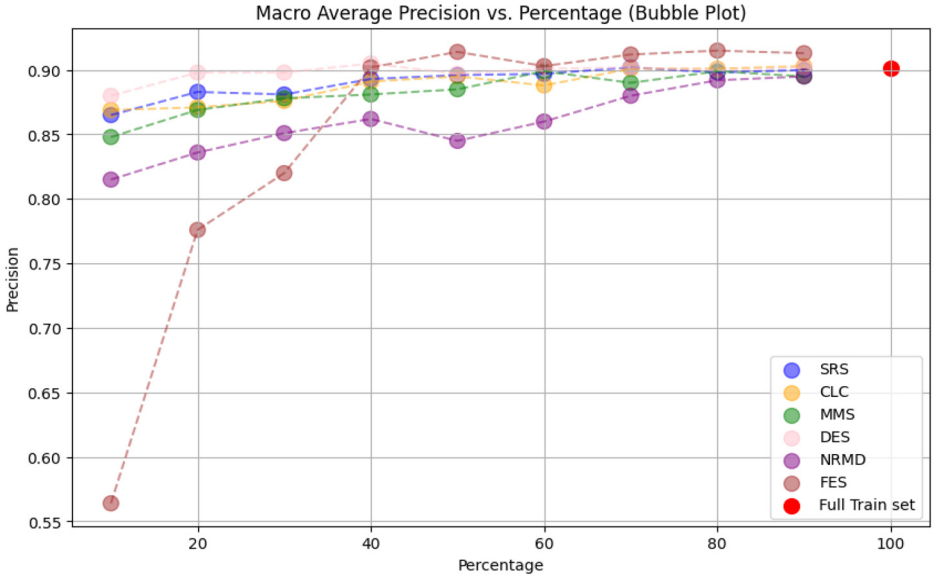

Figure 6. Collision: Reduction + macro average Precision.

other ratios the best one is DES. The relationship between macro average precision and training dataset size is also approximately linear for all methods except FES, which suffers a significant drop when the reduction ratio is under 30%.

The results that we get when we analyze the macro average recall are quite different. As we saw with accuracy and macro average precision, this metric is generally well preserved even at very low reduction ratios, although it suffers a very significant drop when FES is applied with a reduction ratio under 30%. But contrary to what we have seen for the previous metrics, no method clearly outperforms the others in terms of macro average recall. All methods except NRMD have given the best median result for some of the chosen reduction ratios.

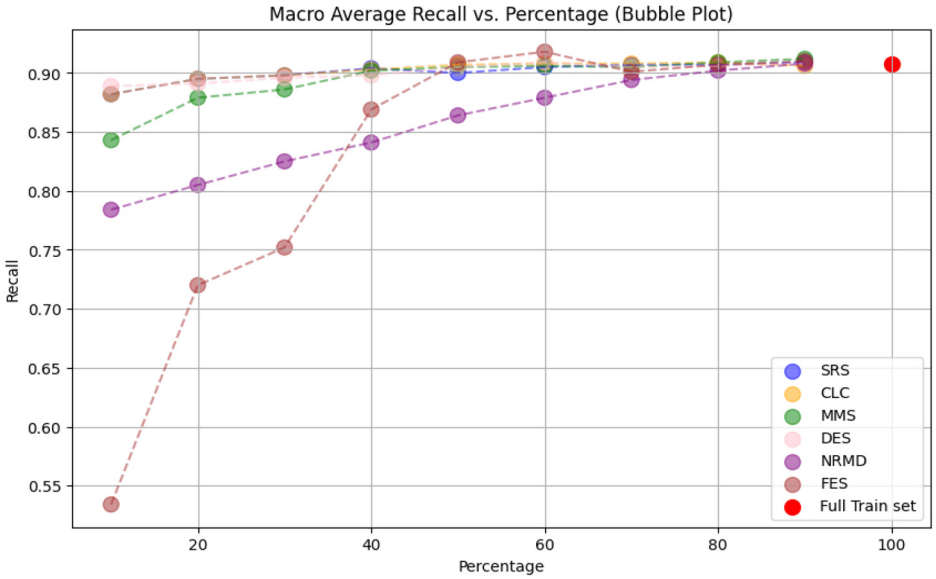

Figure 7. Collision: Reduction + macro average Recall.

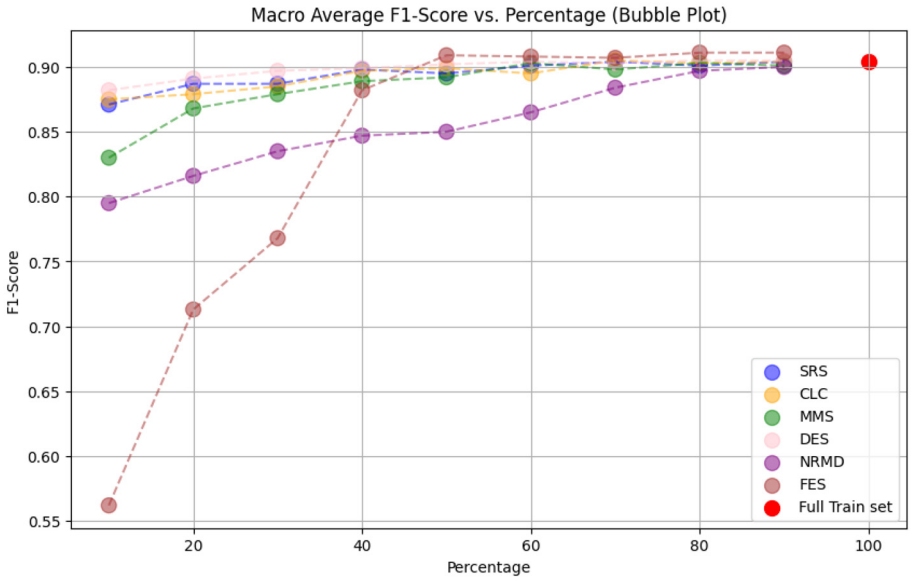

Figure 8. Collision: Reduction + macro average F1-score.

All the general observations we have made when analyzing accuracy and macro average precision can also be seen for the macro average F1-score. In general, all reduction methods preserve this metric well, being FES the best performing method when the reduction ratio is higher than 50% and DES otherwise. The drop in macro average F1-score is also noticeable when many examples are removed with the FES method, while this tendency is not as pronounced for the other data reduction methods.

Finally, we have found an interesting relationship between the ε-representativeness of the reduced datasets and the macro average F1-score of the models trained with them. Given any reduction ratio $p = 0.1, \cdots, 0.9$, we have got the ε-representativeness and the macro average F1-score for each reduction method and each

iteration in the experiment (in total there are 6 reduction methods × 10 iterations = 60 pairs (ε,F1) for each p). We computed for each p the Spearman's rank correlation coefficient[64] of its respective cloud of 60 points to test if there exists a dependence between ε-representativeness and the macro average F1-score that can be described with a monotonic (always increasing or always decreasing) function. This coefficient is a real number $\rho \in [-1, 1]$, where ρ close to 1 indicates a strong positive monotonic correlation, ρ close to -1 indicates a strong negative monotonic correlation, and ρ close to 0 indicates no monotonic correlation. We also compute the associated p-value to test if the correlation ρ is significantly different from 0. A p-value under a certain threshold (in our case 0.05, which is a standard choice) indicates that ρ is unlikely to be 0, while a p-value above it suggests that the observed correlation might be coincidental and not due to a true dependence between both variables. We performed this statistical analysis independently for each p to eliminate the possible effect that the reduction ratio could have if we used all the possible pairs (ε, F1) altogether.

The results that we got can be seen in Table 3. All the computed ρ values are negative, although they are only significantly different from zero when the reduction ratio is below 40%. That indicates that, when data reduction methods remove a large number of examples, the best performing models are those trained with the reduced datasets that best preserve the ε-representativeness of the entire training set. In a few words, when we reduce the Collision dataset with a small reduction percentage, the smaller the ε value, the better the model will perform.

Dry Bean Dataset The median results for the computing time and carbon emission can be seen in Figure 9 and Figure 10. There is also a proportional relation between the computation time and the carbon emission in this experiment since each minute of computations emitted approximately 0.21 g of CO_2 into the atmosphere. As happened with the Collision dataset, the use of data reduction methods prior to network training helped to reduce the computation time and the carbon emission of the model building with respect to the reference case. The only exception to this rule is when we apply PRD reduction with a reduction ratio greater than 70% (see in Figure 9 and Figure 10). In that situation, both the computation time and the carbon emission

Table 3. Collision: Correlation between ε-representativeness and macro average F1-score. This table displays the non-parametric Spearman correlation coefficient and its p-value. Note that when the reduction ratio is equal to or less than 40%, the p-value is below the significance level of 0.05.

	10%	20%	30%	40%	50%	60%	70%	80%	90%
Spearman's ρ	-0.38	-0.43	-0.42	-0.39	-0.22	-0.15	-0.19	-0.07	-0.14
p-value	0.0	0.0	0.0	0.0	0.1	0.24	0.14	0.58	0.3

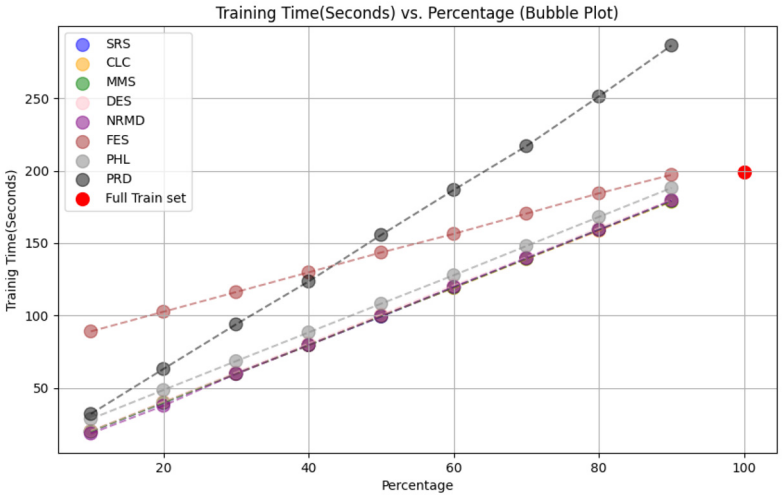

Figure 9. Dry Bean: Reduction + training time.

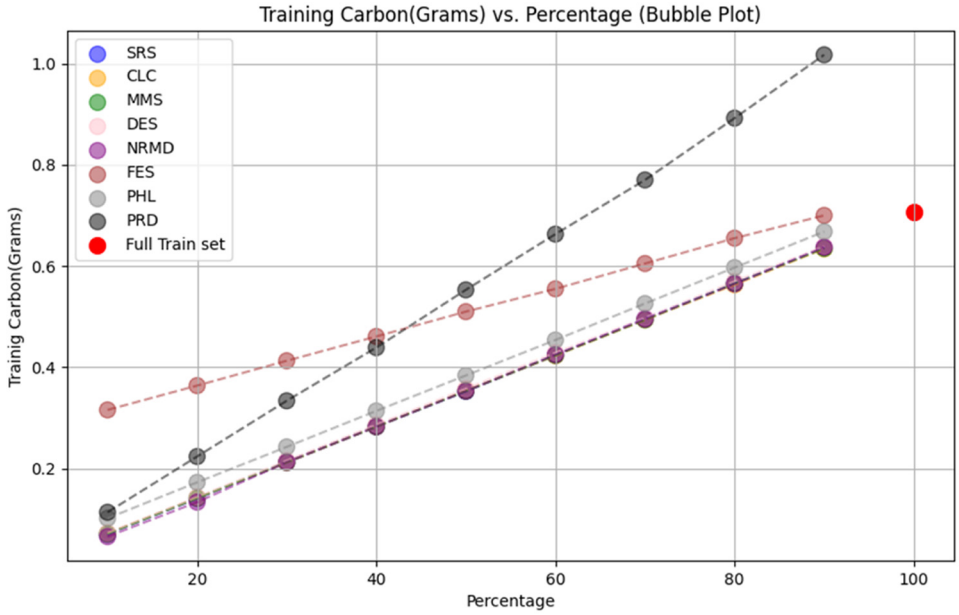

Figure 10. Dry Bean: Reduction + training carbon.

of reduction and training exceed those of the reference case. There is no reduction method that runs faster than all the others for this dataset. SRS, MMS, DES, NRMD and CLC run equally fast for the Dry Bean dataset. They hardly need any time to reduce the training data set, so almost all the measured time and, therefore, the carbon emission correspond to the network training.

The median values on the ε-representativeness, which can be seen in Figure 11, show us similar results to those observed with the Collision dataset. MMS is still the best data reduction method to preserve the ε-representativeness with respect to the full training dataset, being CLC the second best option. On the contrary, NRMD and PRD are the two methods that generally produce the less ε-representative reduced datasets.

The median results on accuracy, macro average precision, macro average recall and macro average F1-score for the Dry Bean dataset can be seen in Figure 12, Figure 13, Figure 14 and Figure 15 respectively. In this experiment, the reference case has a median accuracy of 89.9% on the test dataset and, contrary to the Collision dataset, no model trained on a reduced dataset improves this value. No reduction method outperforms the others for every reduction ratio in terms of accuracy. What we observe is that, while, in general, the accuracy is well preserved when the reduction ratio is high, it suffers a drastic drop when many examples are removed from the training dataset. If we wanted to lose at most 5% of accuracy (i.e., have at least 84.9%) we would have to select at least 40% of the data, and not all reduction methods would guarantee this maximum loss of accuracy.

Regarding the macro average precision, we also find that no method is clearly better than the rest for all reduction ratios, although FES seems to dominate the statistics for the central ratios ($30\% \leq p \leq 60\%$). Here, the drop in the metric as the reduction ratio decreases is not as pronounced as it is for accuracy. This could be because the macro average precision is a more robust measure as it is less influenced by the larger classes.

We can see similar results when we analyze the details on macro average recall. No method outperforms all the rest, and SRS, PRD, CLC, PHL and FES give the best result for at least one reduction ratio.

Finally, when analyzing the macro average F1-score, we observe that four methods (PRD, CLC, PHL and FES) give the best score for some reduction ratio, but FES seems to give the best performing reduced datasets for $30\% \leq p \leq 70\%$. As we said when we analyzed the accuracy, it is not possible to extract a reduced dataset with 30% of its size or less without losing more than 5% of macro average F1-score.

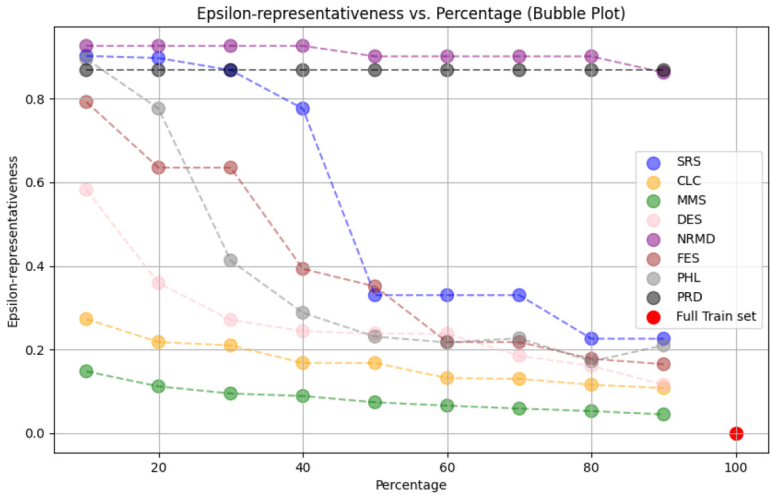

Figure 11. **Dry Bean: Reduction + Epsilon.**

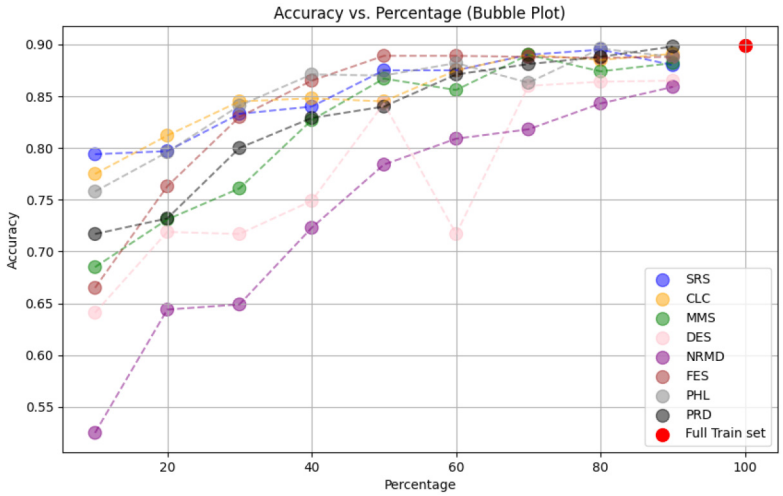

Figure 12. **Dry Bean: Reduction + Accuracy.**

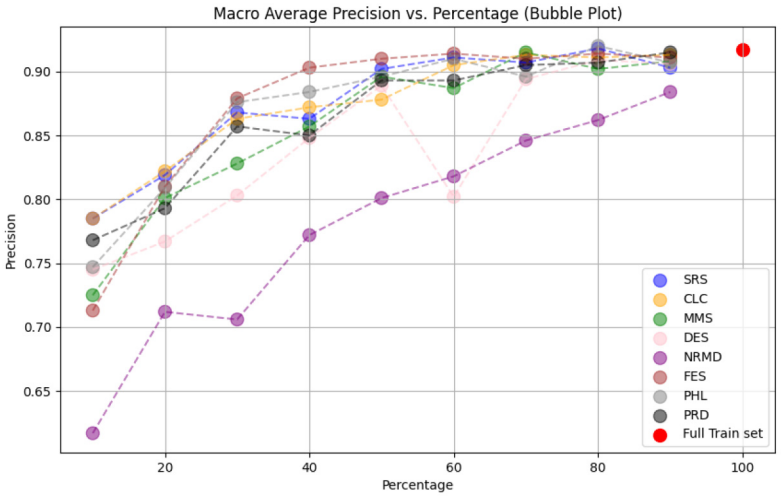

Figure 13. **Dry Bean: Reduction + Precision.**

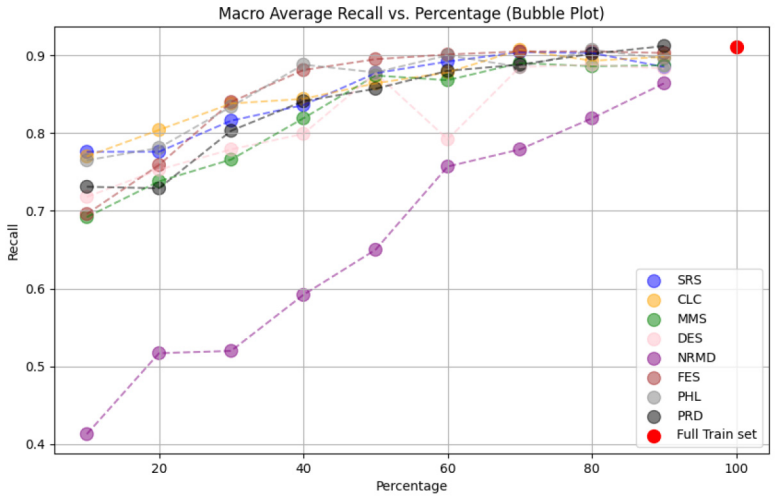

Figure 14. **Dry Bean: Reduction + Recall.**

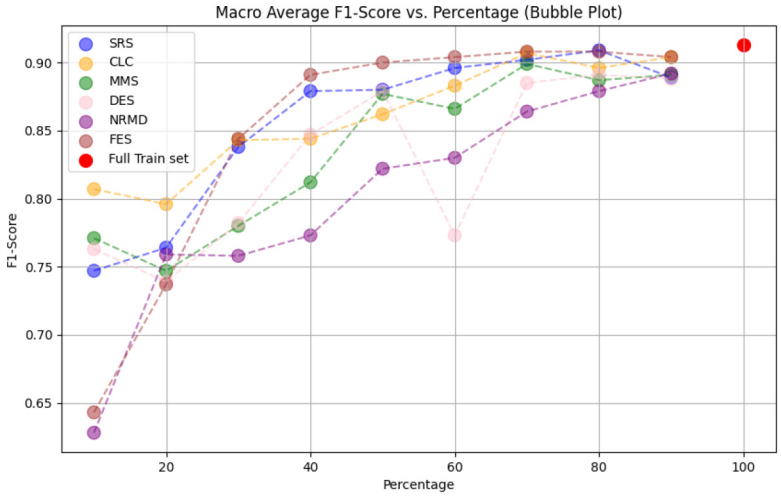

Figure 15. **Dry Bean: Reduction + F1-Score.**

Description of the data structure for accessing the final results The results of both experiments were stored in a Python dictionary with a nested structure, which we will call the *results dictionary*. On the first level, the result dictionary is composed of 10 dictionaries containing a dictionary for each iteration. The dictionary of each iteration has a dictionary for each data reduction method used. The dictionary of each data reduction method contains an item for each reduction ratio $p = 0.1, \cdots, 1.0$, and each one of them is a dictionary containing all the results obtained for that data reduction method and the reduction ratio p in that iteration. The results obtained when the training dataset is not reduced are stored in the item with key $p = 1.0$. Finally, the dictionary associated with a specific iteration, data reduction method, and reduction ratio, contains a key for the following metrics:

- **time:** To store the computing time in seconds of reduction and training (only training when $p = 1.0$)
- **carbon:** To store the carbon emission in kg of CO_2 of reduction and training (only training when $p = 1.0$)
- **epsilon:** To store the ε-representativeness of $\mathscr{D}_{\text{train},R}$ with respect to $\mathscr{D}_{\text{train}}$
- **acc:** To store the accuracy of the model over $\mathscr{D}_{\text{test}}$
- For each class k:
  - **pre_k:** To store the model precision for class k over $\mathscr{D}_{\text{test}}$

  - **rec_k:** To store the model recall for class k over $\mathscr{D}_{\text{test}}$
  - **f1_k:** To store the model F1-score for class k over $\mathscr{D}_{\text{test}}$
- **pre_avg:** To store the model macro average precision for class k over $\mathscr{D}_{\text{test}}$
- **rec_avg:** To store the model macro average recall for class k over $\mathscr{D}_{\text{test}}$
- **f1_avg:** To store the model macro average F1-score for class k over $\mathscr{D}_{\text{test}}$

Once we have this results dictionary, the next step is to summarize the information from all the iterations in a simpler dictionary. This object, which we call the *median results dictionary*, has the same structure as the entry that we got for each iteration in the results dictionary. For each data reduction method and each reduction ratio, the entry of a specific metric is the median value of the 10 metrics obtained during the 10 different iterations of the experiment. This way we can obtain a more stable representation of the performance of each method across iterations, mitigating the potential influence of outliers or variability in each individual run. All the figures we have seen in this subsection present the metrics from the median results dictionary.

5.2 Experiments for object detection

In this subsection, we describe the methodology we have used in our experiments to extend data reduction techniques to images. Note that we need to adapt the methodology depending on the type of data reduction method. Later, we present the datasets used for the object detection experiments, including the parameter settings and the setup, and finally, we discuss the results that we have obtained.

5.2.1 Methodology

The proposed methodology for the non-wrapper methods, illustrated in Figure 16, consists of the following five steps:

1. **Feature extraction**: The objective of this step is to convert the raw pixel values of the images into a set of meaningful and concise features that capture relevant information. These features should allow the model to distinguish between different patterns, objects, or structures within the images. This process involves utilizing a computer vision model, to extract features from all the images in the training set. In our case, we used a pre-trained YOLOv5 model on the COCO dataset (Common Objects in Context)[65], a widely used collection in computer vision. The COCO dataset, consisting of approximately 330,000 annotated images with object location and category information, is one of the largest and most diverse datasets available for object detection and segmentation tasks. Utilizing pre-trained models on COCO proves advantageous because of its scale and diversity. This pre-training allows models to learn generic features and representations from a vast array of real-world images, enhancing their ability to generalize across various downstream tasks. This approach can lead to improved performance and efficiency when fine-tuning or adapting these models to specific applications.

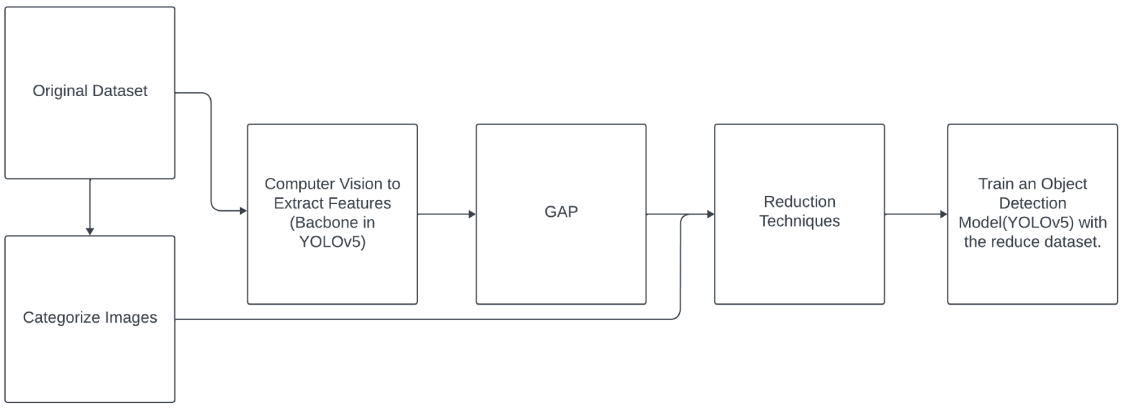

Figure 16. Diagram of the workflow for the proposed methodology to apply data reduction techniques on images dataset.

2. **Categorizing images**: In this step, we categorize each image based on the objects present in them. This categorization is essential for applying data reduction techniques, as explained in Section 5.2.2, where we detail how each dataset is categorized.

3. **Global Average Pooling**: Regarding the output of step 1, we apply global average pooling[10,66] to the output of the last layer of the backbone, in order to transform the feature maps into an n-dimensional vector representing their extracted features. Subsequently, these feature vectors can be used to calculate distances or similarities between images, and reduction methods can be applied to them.

4. **Applying data reduction technique**: Reduction techniques are applied to reduce the amount of samples in the dataset with a specified reduction rate on the matrix produced in step 3, comprising x images and n dimensions, along with the labels from step 2.

5. **Fine tuning with the reduced dataset**: This step allows us to assess whether satisfactory performance is achieved, potentially maintaining the same level as with the complete training set. The performance evaluation is conducted on the test set using YOLOv5 pre-trained on the COCO dataset, with the backbone frozen. In this context, "fine-tuning with a specific part frozen" implies that some of the model's parameters are kept fixed during the training process on the new task. This approach leverages prior knowledge gained during initial training, enabling more efficient adaptation to the new task without completely discarding previously learned information.

When using wrapper methods, we must adopt a slightly different methodology than the one previously described (see Figure 17). Initially, we need to categorize the images, similar to the preceding methodology, since these images are intended for object detection and lack specific labels, instead featuring multiple elements within them. Then, we build a straightforward classification model, incorporating the reduction technique during training to yield the reduced dataset. Finally, we train the YOLOv5 detection model using the reduced dataset, aligning with the objective of Step 5 in the aforementioned methodology.

In addition, to apply the CLC method to images, we introduce a slight modification by applying KMeans[67] on X with c (the number of classes) clusters. Then, we determine the number of samples closest (based on the Euclidean distance) to each centroid, resulting in our reduced dataset $\mathscr{D}_R$ guided by the specified reduction rate. This adjustment is necessary because the centroids generated by the KMeans method, derived from the representations obtained by our image methodology, do not correspond to specific images from our dataset. Consequently, they do not convey information about what we genuinely aim to detect and localize. This modification is called Representative KMeans (RKM).

5.2.2 Datasets for object detection

Roboflow The Roboflow dataset[1] [68] comprises 514 RGB images, each 416 pixels in both height and width. These images show pedestrians and people in wheelchairs. The training dataset comprises 463 RGB images, in

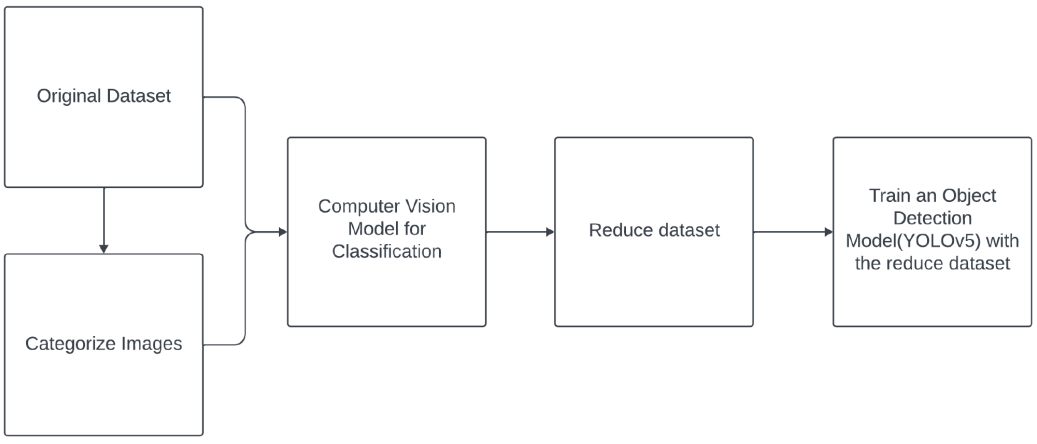

Figure 17. Diagram of the workflow for the proposed methodology to apply wrapper methods.

[1]https://universe.roboflow.com/2458761304-qq-com/wheelchair-detection

which a total of 499 pedestrians (annotated as P) and 616 wheelchair users (annotated as W) appear. The test dataset is composed of 51 RGB images, in which a total of 55 pedestrians and 65 wheelchair users appear. The following table shows the criteria that we have used to label the images in the Roboflow dataset according to their content.

Number of P	Number of W	Label
0	1	0
≥ 1	0	1
≥ 1	≥ 1	2
0	≥ 2	3

Mobility Aid The Mobility Aid dataset[2] [69] is composed of 17079 RGB images, of which 10961 are part of the training dataset and the 6118 remaining are part of the test dataset. In this dataset we can find five types of objects: pedestrians (8371, it will be annotated as P), wheelchair users (6458, it will be annotated as W), people pushing a wheelchair (3323, it will be annotated as PW), people with crutches (5374, it will be annotated as C) and people with a walking-frame (7649, it will be annotated as WF). The test dataset is composed of 6208 P, 1993 W, 782 PW, 1883 C, and 2174 WF. The following table shows the criteria that we have used to label the images in the Mobility Aid dataset according to their content.

Number of P/PW	Number of W/C/WF	Label
1	0	1
0	1	2
≥ 1	0	3
0	≥ 1	4
≥ 1	≥ 1	5

5.2.3 Parameter setting

For the fine-tuning of YOLOv5, we maintain the backbone frozen (pre-trained on the COCO dataset) while training the rest of the model. We configure the training with 100 epochs for the Roboflow dataset and 50 epochs for the Mobility Aid dataset. The batch size is set at 16, and the image size is fixed at 640. We use the SGD optimizer, and the learning rate is set to 0.01 for both datasets. The number of object classes to be detected depends on the specific dataset, being 2 for the Roboflow dataset and 5 for the Mobility Aid dataset.

5.2.4 Experiments setup

We used Python 3.9 and PyTorch[27] on Ubuntu 20.04 to perform our experiments. The training phase was executed on an NVIDIA QUADRO RTX 4000 with 8 GB of RAM and an Intel Xeon Silver 4210 preprocessor. Data partitioning followed the default specifications for each dataset.

To assess the performance difference between the full training set and the reduced set, we performed five training runs on the reduced training dataset. We calculated the arithmetic mean and standard deviation of the results for the test set to measure the performance of each reduction method. For the Roboflow dataset, due to its limited sample size, we applied rate reductions of 50% and 75%. In contrast, for the Mobility Aid dataset with a larger sample size, we applied rate reductions of 75% and 90%.

5.2.5 Comparison metrics

We use five metrics to assess the performance of training YOLOv5 with the full dataset compared to training it with different reduction methods and reduction rates. Initially, we use three performance metrics: precision, recall, and mean average precision setting a confidence threshold of 0.5 ($mAP@0.5$), which means that only detections with a confidence of 50% or higher are considered correct. These metrics are computed individually for each class and globally by averaging the values across all classes. Additionally, we consider the time required for data reduction, measured in seconds for each reduction method, and the model fine-tuning time. Our primary goal with these metrics is to determine whether we can maintain similar performance while considering the time saved.

[2] http://mobility-aids.informatik.uni-freiburg.de/

Furthermore, we calculate ε-representativeness to measure how well the reduced dataset $\mathscr{D}_R$ represents the original dataset $\mathscr{D}$. We also compute carbon emissions during both the data reduction and fine-tuning processes.

5.2.6 Results and discussion

The source code to implement data reduction techniques in these datasets is available at 9 in the ObjectDetection folder.

Roboflow Dataset In our initial experiment, we compared the performance of fine-tuning with the full training dataset against training with a reduced dataset using different reduction methods. Table 4 shows the reduction time, ε-representativeness of the $\mathscr{D}_r$, training time, model performance, and CO_2 emissions during the training and reduction phases at a 50% reduction rate. We can observe that, despite training with only 50% of the data, we maintained performance comparable to using the entire dataset. In addition, we reduced the training time by approximately 40%, decreasing from 9 minutes and 20 seconds to about 5 minutes and 45 seconds with 50% of the samples. This reduction in time was accompanied by a similar reduction (about 40%) in CO_2 emissions throughout the process. The emission during the application of the reduction methods was small compared to the training time. Notably, effective methods in this scenario include SRS, DES, MMS, RKM, and FES. On the contrary, NRMD and PHL perform worse, with a slight loss of performance. Data reduction times were generally low, except for FES, which showed excessive duration compared to other methods. Conversely, a lower ε-representativeness did not seem to be critical for performance improvement or degradation.

Table 4. Table results for Roboflow dataset and 50% reduction rate. The 'Precision', 'Recall' and '*mAP*@.5' columns show the mean and standard deviation values for the specified variables. The '$CO_2(g)$' column indicates the grams of CO_2 emitted during the application of the reduction method and during the fine-tuning. The 'R Time(s)' column shows the time in seconds for data reduction, while the 'FT Time' column displays the time spent on fine-tuning the model. We have highlighted in red the values obtained during fine-tuning with the full dataset, which serves as a reference. Additionally, we have highlighted in green the best reduction method for each metric.

Method	R Time(s)	ε	FT Time	Precision	Recall	*mAP*@.5	CO_2(g)
-	-	-	9m 19s	A: 0.951±0.001 P: 0.926±0.022 W: 0.976±0.015	A: 0.897±0.019 P: 0.832±0.031 W: 0.96±0.018	A: 0.944±0.009 P: 0.906±0.014 W:0.984±0.005	5.5
SRS	0.002	2.58	5m 44s	A: 0.945±0.021 P: 0.921 ±0.035 W: 0.97±0.011	A:0.897 ±0.019 P: 0.836±0.03 W: 0.958±0.015	A:0.897 ±0.019 P: 0.836±0.03 W: 0.958±0.015	0+3.3
DES	0.29	3.12	5m 44s	A: 0.94±0.001 P: 0.925±0.015 W: 0.95±0.015	A: 0.885±0.015 P: 0.795±0.03 W: 0.975±0.015	A: 0.945±0.005 P:0.905±0.005 W: 0.985±0.005	0.002+3.3
NRMD	0.09	2.27	5m 47s	A: 0.925±0.015 P: 0.905±0.02 W: 0.945±0.02	A: 0.89±0.016 P: 0.82±0.014 W: 0.956±0.019	A: 0.932±0.009 P: 0.892±0.016 W: 0.972±0.007	0.001+3.3
MMS	0.09	1.99	5m 46s	A: 0.951±0.01 P: 0.921±0.017 W: 0.981 ±0.005	A: 0.894±0.015 P: 0.821±0.029 W: 0.967±0.011)	A: 0.939±0.006 P:0.9±0.011 W: 0.981±0.006	0.0004+3.26
RKM	1.26	1.25	5m 45s	A: 0.948±0.005 P:0.907±0.015 W: 0.99 ±0.009	A: 0.895±0.017 P: 0.819±0.032 W: 0.971±0.008	A: 0.94±0 P: 0.894±0.003 W: 0.985±0.002	0.005+3.3
PRD	0.92	1.67	5m 44s	A: 0.944±0.02 P: 0.916±0.04 W: 0.97±0.008	A: 0.89±0.015 P: 0.814±0.025 W: 0.965±0.008	A: 0.937±0.007 P: 0.894±0.015 W: 0.974±0.008	0.002+3.29
PHL	0.64	2.85	5m 38s	A: 0.942±0.028 P: 0.9±0.045 W: 0.982±0.014	A: 0.863±0.21 P: 0.773±0.041 W: 0.954±0.01	A: 0.927±0.006 P: 0.875±0.017 W: 0.978±0.011	0.004+3.24
FES	8.14	2.28	5m 39s	A:0.921±0.013 P: 0.871±0.026 W: 0.972±0.017	A: 0.903±0.011 P: 0.844±0.018 W: 0.962±0.006	A: 0.948±0.004 P: 0.913±0.009 W: 0.986±0.003	0.08+3.24

Table 5 shows the same analysis with a 75% reduction rate, where the overall training time was reduced from 9.5 minutes to approximately 4 minutes, a 60% increase in speed. Data reduction time remained insignificant, with some methods displaying longer computation times, such as FES. Despite a slight decrease in performance across all metrics with all methods, SRS, MMS, RKMEANS, and PHL emerged as more robust options. In particular, SRS is the one that best maintains precision and *mAP*. On the contrary, NRMD showed the most significant loss of performance. A lower ϵ-representativeness did not appear to be a determining factor for better or worse performance. CO_2 emissions were also reduced by 60%. On average, the overall *mAP* for reduction methods showed only a 3% reduction compared to the substantial computational time and CO_2 emission savings of 60%. In Figure 18, the mean *mAP* values for each category and method, along with the full dataset, illustrate the best performing methods. At a 50% reduction rate, performance is nearly maintained, while at a 75% reduction rate, some performance loss is evident. SRS stands out as the most effective method. In addition, a general improvement in accuracy is observed for wheelcharis compared to people, potentially attributed to a slight imbalance in the dataset between the two object categories. We only present the *mAP* figure (Figure 18) because it is the most comprehensive performance metric for evaluating object detection.

Based on the results obtained from this dataset, we can affirm that the application of reduction methods within the proposed methodology, followed by fine-tuning of YOLOv5 for object detection, led to a significant reduction in CO_2 emissions and computation time. Importantly, this reduction did not adversely affect the model's performance in object detection tasks.

Table 5. Table results for Roboflow dataset and 75% reduction rate. The Precision', 'Recall' and '*mAP*@.5' columns show the mean and standard deviation values for the specified variables. The '$CO_2(g)$' column indicates the grams of CO_2 emitted during the application of the reduction method and during the fine-tuning. The 'R Time(s)' column shows the time in seconds for data reduction, while the 'FT Time' column displays the time spent on fine-tuning the model. We have highlighted in red the values obtained during fine-tuning with the complete dataset, which serves as a reference. Additionally, we have highlighted in green the best reduction method for each metric.

Method	R Time(s)	ϵ	FT Time	Precision	Recall	*mAP*@.5	CO_2(g)
-	-	-	9m 19s	A: 0.951±0.001 P: 0.926±0.022 W: 0.976±0.015	A: 0.897±0.019 P: 0.832±0.031 W: 0.96±0.018	A: 0.944±0.009 P: 0.906±0.014 W :0.984±0.005	5.5
SRS	0.001	2.7	4m 2s	A: 0.931±0.013 P:0.9±0.017 W: 0.963±0.021	A: 0.886±0.013 P: 0.815±0.016 W: 0.957±0.01	A: 0.937±0.005 P: 0.906±0.008 W: 0.968±0.004	0+2.25
DES	0.24	3.21	3m 59s	A: 0.894±0.022 P: 0.824±0.04 W: 0.964±0.03	A: 0.86±0.031 P: 0.761±0.045 W: 0.96±0.019	A: 0.919±0.006 P: 0.855±0.009 W: 0.984±0.005	0.002+2.24
NRMD	0.09	2.3	3m 58s	A: 0.901±0.016 P: 0.887±0.021 W: 0.914±0.023	A: 0.846±0.018 P: 0.743±0.043 W: 0.949±0.011	A: 0.908±0.009 P: 0.857±0.016 W: 0.958±0.003	0.001+2.23
MMS	0.05	2.3	4m	A: 0.935±0.012 P: 0.927±0.017 W: 0.943±0.015	A: 0.858±0.009 P: 0.776±0.002 W: 0.94±0.01	A: 0.922±0.01 P: 0.885±0.019 W: 0.958±0.006	0.0004+2.23
RKM	1.22	1.18	3m 52s	A: 0.908±0.015 P: 0.821±0.029 W: 0.995±0.006	A: 0.881±0.009 P: 0.829±0.019 W: 0.934±0.009	A: 0.927±0.007 P: 0.881±0.015 W: 0.972±0.004	0.005+2.18
PRD	0.38	2.68	4m	A: 0.895±0.024 P: 0.849±0.038 W: 0.941±0.02	A: 0.872±0.033 P: 0.793±0.066 W: 0.951±0.011	A: 0.916±0.013 P: 0.856±0.022 W: 0.967±0.014	0.001+2.22
PHL	0.67	3.74	3m 59s	A: 0.897±0.02 P: 0.816±0.034 W: 0.977±0.017	A: 0.887±0.016 P: 0.834±0.023 W: 0.941±0.016	A: 0.923±0.004 P: 0.877±0.007 W: 0.97±0.004	0.004+2.23
FES	8.63	3.12	3m 55s	A: 0.884±0.002 P: 0.824±0.033 W: 0.945±0.016	A: 0.874±0.014 P: 0.796±0.024 W: 0.951±0.009	A: 0.911±0.008 P: 0.854±0.013 W: 0.968±0.004	0.08+2.17

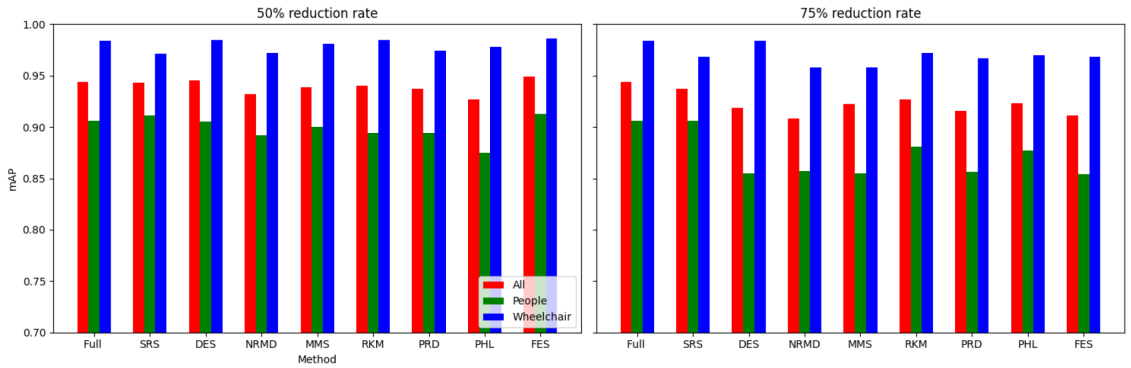

Figure 18. *mAP* values on Roboflow dataset when using a 50% reduction rate (first column) and when using a 75 percent reduction rate (second column).

Mobility Aid Dataset Initially, we observe the training benchmark results with the full dataset comprising 10,961 instances. This yields commendable results, such as a mean average precision of 0.93, with a narrow standard deviation of 0.003 across all classes. In particular, the performance stands out in the category of people in wheelchairs (W), surpassing the overall average, while the performance in other categories is noteworthy. Other categories were not included in Table 6 and Table 7 In general, superior performance is evident for the wheelchair, push-wheelchair, and walking-frame categories compared to the pedestrian and crutches categories, which show below-average performance.

Moving on to Table 6, which shows the results for a reduction rate of 75%, we observe a significant reduction in training time, approximately 50%. Reduction times are just seconds for most methods, extending to minutes for PRD, PHL and FES. CO_2 emissions also witness a substantial decrease, around 55% for all methods, except for PRD, PHL and FES, which emit more CO_2 due to a longer computation time during data reduction. Finally, we were able to maintain the performance achieved with the full training set for all data reduction methods. This underscores the practical significance of these methods in reducing computation time and consequently lowering CO_2 emissions during model fine-tuning. The exceptions are the RKM and NRMD methods, which show a decrease in performance.

In Table 7, we present similar results, but this time with a reduction rate of 90%. The primary observation is a decrease in performance across various metrics, including precision, recall, and mean average precision. Notably, while there is an overall loss in performance due to reduced metrics in other categories, the decline in the wheelchair category is comparatively less pronounced. Despite this loss in performance, we managed to cut down the training time to 43 minutes, representing about 67% of the training time without dataset reduction. A corresponding decrease in CO_2 consumption is observed. Despite the general decline, certain methods, such as SRS, MMS, PRD, PHL and FES, show a relatively robust maintenance of performance.

A visual representation of the mean *mAP* values for wheelchairs, overall, and for each method alongside the full dataset is shown in Figure 19. This visualization offers a clearer insight into the methods that yield optimal results. With a reduction rate of 75%, we almost maintain performance in all methods, except NRMD and RKM. However, at a rate of reduction 90%, some loss of performance is evident, highlighting the efficacy of methods such as SRS, MMS, PRD, PHL and FES. In particular, the drop in performance for the wheelchair category is less pronounced compared to other categories. We only present the *mAP* figure (Figure 19), as it serves as the most comprehensive performance metric for evaluating object detection.

With these results obtained for this dataset, we can confirm that the use of reduction methods within the proposed methodology, followed by fine-tuning YOLOv5 for object detection, led to a significant reduction in both CO_2 emissions and computation time. Importantly, this reduction did not affect the model's performance in object detection.

Table 6. Table results for Mobility Aid dataset and 75% reduction rate. The 'Precision', 'Recall' and '*mAP*@.5' columns show the mean and standard deviation values for the specified variables. The '$CO_2(g)$' column indicates the grams of CO_2 emitted during the application of the reduction method and during the fine-tuning. The 'R Time(s)' column shows the time in seconds for data reduction, while the 'FT Time' column displays the time spent on fine-tuning the model. We have highlighted in red the values obtained during fine-tuning with the complete dataset, which serves as a reference. Additionally, we have highlighted the best reduction method for each metric.

Method	R Time(s)	ϵ	FT Time	Precision	Recall	*mAP*@.5	$CO_2(g)$
-	-	-	2h 8m	A: 0.91±0.008 W: 0.994±0.001	A: 0.875±0.007 W: 0.896±0.01	A: 0.93±0.003 W: 0.941±0.003	73.85
SRS	0.006	0.86	1h 2m	A: 0.912±0.007 W: 0.979±0.029	A: 0.876±0.007 W: 0.887±0.011	A: 0.932±0.004 W: 0.94±0.003	0+34.12
DES	7.98	0.87	1h 2m	A: 0.91±0.009 W: 0.989±0.005	A: 0.876±0.006 W: 0.893±0.004	A: 0.93±0.006 W: 0.941±0.001	0.04+34.1
NRMD	3.83	1.04	1h 2m	A: 0.857±0.004 W: 0.986±0.007	A: 0.83±0.006 W: 0.865±0.031	A: 0.903±0.004 W: 0.932±0.002	0.015+33.94
MMS	5.88	0.66	1h 2m	A: 0.911±0.006 W: 0.994±0.003	A: 0.875±0.004 W: 0.891±0.007	A: 0.928±0.005 W: 0.939±0.005	0.03+34.13
RKM	31	0.74	1h 2m	A: 0.845±0.013 W: 0.961(0.008	A: 0.795±0.006 W: 0.871±0.008	A: 0.89±0.006 W: 0.92±0.005	0.023+34.1
PRD	457	0.59	1h 1m	A: 0.912±0.011 W: 0.989±0.005	A: 0.876±0.007 W: 0.889±0.008	A: 0.931±0.004 W: 0.939±0.002	1.82+33.91
PHL	314	0.9	1h 3m	A: 0.912±0.007 W: 0.994±0.003	A: 0.875±0.009 W: 0.896±0.003	A: 0.931±0.003 W: 0.38±0.001	1.79+35.09
FES	289	1	56m 33s	A: 0.912±0.004 W: 0.993±0.001	A: 0.875±0.005 W: 0.887±0.009	A: 0.928±0.004 W: 0.939±0.003	2.9+31.83

Table 7. Table results for Mobility Aid dataset and 90% reduction rate. The 'Precision', 'Recall' and '*mAP*@.5' columns show the mean and standard deviation values for the specified variables. The '$CO_2(g)$' column indicates the grams of CO_2 emitted during the application of the reduction method and during the fine-tuning. The 'R Time(s)' column shows the time in seconds for data reduction, while the 'FT Time' column displays the time spent on fine-tuning the model. We have highlighted in red the values obtained during fine-tuning with the complete dataset, which serves as a reference. Additionally, we have highlighted the best reduction method for each metric.

Method	R Time(s)	ϵ	FT Time	Precision	Recall	*mAP*@.5	$CO_2(g)$
-	-	-	2h 8m	A: 0.91±0.008 W: 0.994±0.001	A: 0.875±0.007 W: 0.896±0.01	A: 0.93±0.003 W: 0.941±0.003	73.85
SRS	0.018	1.15	48m 31s	A: 0.894±0.01 W: 0.993±0.003	A: 0.867±0.007 W: 0.886±0.015	A: 0.926±0.002 W: 0.942±0.002	0+26.13
DES	8.13	1	43m 28s	A: 0.875±0.014 W: 0.971±0.02	A: 0.862±0.007 W: 0.893±0.011	A: 0.917±0.006 W: 0.94±0.003	0.04+24.06
NRMD	2.69	1.18	43m 17s	A: 0.826±0.005 W: 0.976±0.008	A: 0.773±0.02 W: 0.846±0.02	A: 0.873±0.009 W: 0.923±0.005	0.017+23.93
MMS	2.9	0.84	43m 26s	A: 0.904±0.006 W: 0.996±0.001	A: 0.875±0.004 W: 0.87±0.01	A: 0.927±0.004 W: 0.935±0.002	0.017+24.06
RKM	22.92	0.95	43m 26s	A: 0.812±0.008 W: 0.968±0.007	A: 0.767±0.012 W: 0.855±0.024	A: 0.858±0.004 W: 0.918±0.003	0.024+24.04
PRD	66.4	1	43m 28s	A: 0.908±0.01 W: 0.99±0.006	A: 0.868±0.007 W: 0.883±0.01	A: 0.926±0.007 W: 0.938±0.004	0.29+24.05
PHL	317	1.04	43m 36s	A: 0.898±0.006 W: 0.989±0.005	A: 0.869±0.005 W: 0.889±0.007	A: 0.926±0.003 W: 0.937±0.002	1.35+24.11
FES	269	1.46	43m 29s	A: 0.906±0.007 W: 0.992±0.005	A: 0.87±0.005 W: 0.875±0.033	A: 0.927±0.003 W: 0.936±0.002	2.69+24.02

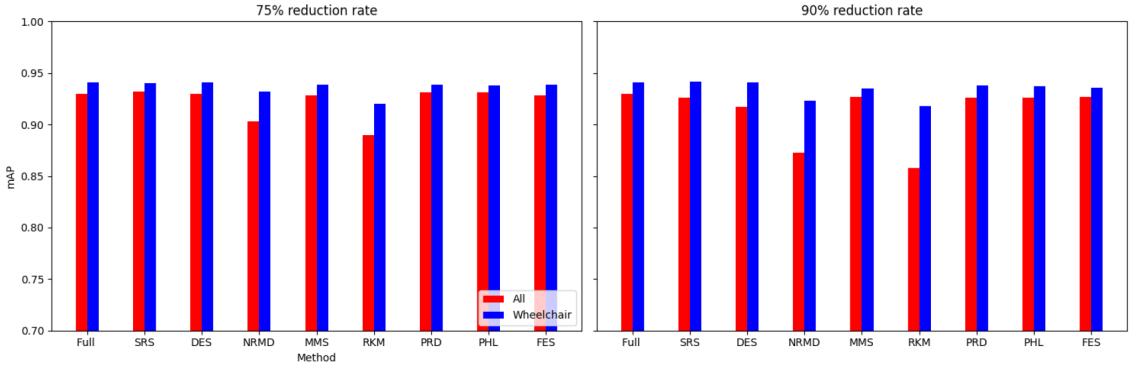

Figure 19. *mAP* values on Mobility Aid dataset when using a 75% reduction rate(first column) and when using a 90% of reduction rate (second column).

Data availability

- Collision: It consists of predicting whether a platoon of vehicles will collide based on features such as the number of vehicles and their speed. The dataset consists of 107,210 examples with 25 numerical features and 2 classes; collision = 1 and collision = 0. This dataset is available at https://doi.org/10.5281/zenodo.10844476[70].

  Data are available under the terms of the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

- Dry Bean: This dataset was created by taking pictures of dry beans from 7 different types and calculating some geometric features from the images, such as the area, the perimeter and the eccentricity. consists of predicting the type of dry bean based on these geometric features. The dataset contains 13,611 examples with 16 features and 7 classes. This dataset is available at https://doi.org/10.24432/C50S4B[58].

  Data are available under the terms of the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

- Roboflow: It comprises 514 RGB images, each 416 pixels in both height and width. These images feature pedestrians and people in wheelchairs. The training dataset comprises 463 RGB images, in which a total of 499 pedestrians (annotated as P) and 616 wheelchair users (annotated as W) appear. The test dataset is composed of 51 RGB images, in which a total of 55 pedestrians and 65 wheelchair users appear. This dataset is available at https://universe.roboflow.com/2458761304-qq-com/wheelchair-detection.

  Data are available under the terms of the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

- Mobility Aid: The dataset Mobility Aid 2[69] is composed of 17079 RGB images, of which 10961 are part of the training dataset and the 6118 remaining are part of the test dataset. In this dataset we can find five types of objects: pedestrians (8371), wheelchair users (6458), people pushing a wheelchair (3323), people with crutches (5374) and people with a walking-frame (7649). This dataset is available at http://mobility-aids.informatik.uni-freiburg.de/.

Code availability

Source code is available at https://doi.org/10.5281/zenodo.10844558[71]

https://github.com/Cimagroup/Experiments-SurveyGreenAI

Data are available under the terms of the Creative Commons Attribution 4.0 International license (CC-BY 4.0).

Acknowledgements

A previous version of this article was published on arXiv: https://arxiv.org/pdf/2403.15150.pdf.

References

1. Schwartz R, Dodge J, Smith NA, *et al.*: **Green AI.** *Commun ACM.* 2020; **63**(12): 54–63.
Publisher Full Text
2. Zha D, Bhat ZP, Lai KH, *et al.*: **Data-centric Artificial Intelligence: a survey.** arXiv: 2303.10158 [cs], 2023.
Publisher Full Text
3. Xu J, Zhou W, Fu Z, *et al.*: **A survey on green deep learning.** *arXiv.* 2021.
Publisher Full Text
4. Verdecchia R, Cruz L, Sallou J, *et al.*: **Data-centric green AI an exploratory empirical study.** In: *2022 International Conference on ICT for Sustainability (ICT4S).* IEEE, 2022; 35–45.
Publisher Full Text
5. O'Shea K, Nash R: **An introduction to Convolutional Neural Networks**. 2015.
Publisher Full Text
6. Ren S, He K, Girshick R, *et al.*: **Faster R-CNN: towards real-time object detection with Region Proposal Networks.** *IEEE Trans Pattern Anal Mach Intell.* 2017; **39**(6): 1137–1149.
PubMed Abstract | **Publisher Full Text**
7. Redmon J, Divvala SK, Girshick RB, *et al.*: **You only look once: unified, real-time object detection.** *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016; 779–788.
Publisher Full Text
8. Perera Lago J, Paluzo Hidalgo E, Toscano Durán V: **Repository survey green AI**. [Code], 2024.
Publisher Full Text
9. Toscano Durán V, Lago JP, Hidalgo EP: **Repository experiments Survey Green AI.** [Code], 2024.
Publisher Full Text
10. Gholamalinezhad H, Khosravi H: **Pooling methods in deep neural networks, a review.** *ArXiv.* 2020; abs/2009.07485.
Publisher Full Text
11. Gurumoorthy KS, Dhurandhar A, Cecchi G, *et al.*: **Efficient data representation by selecting prototypes with importance weights.** In: *2019 IEEE International Conference on Data Mining (ICDM).* IEEE, 2019; 260–269.
Publisher Full Text
12. Olvera-López JA, Carrasco-Ochoa JA, Martínez-Trinidad JF, *et al.*: **A review of instance selection methods.** *Artif Intell Rev.* 2010; **34**: 133–143.
Publisher Full Text
13. Lacombe C, Hammoud I, Messud J, *et al.*: **Data-driven method for training data selection for deep learning.** In: *82nd EAGE Annual Conference & Exhibition.* European Association of Geoscientists & Engineers, 2021; **2021**: 1–5.
Publisher Full Text
14. Li Y, Chao X: **Distance-entropy: an effective indicator for selecting informative data.** *Front Plant Sci.* 2022; **12**: 818895.
PubMed Abstract | **Publisher Full Text** | **Free Full Text**
15. Stolz BJ: **Outlier-robust subsampling techniques for persistent homology.** *J Mach Learn Res.* 2023.
Publisher Full Text
16. Ghojogh B, Crowley M: **Instance ranking and numerosity reduction using matrix decomposition and subspace learning.** In: *Canadian Conference on Artificial Intelligence.* 2019; 160–172.
Publisher Full Text
17. Toneva M, Sordoni A, des Combes RT, *et al.*: **An empirical study of example forgetting during deep neural network learning.** *arXiv.* 2018.
Publisher Full Text
18. Surdeanu M, Valenzuela-Escárcega MA: **Feed-forward neural networks.** Cambridge University Press, 2024; 73–86.
Publisher Full Text
19. Agostinelli F, Hoffman MD, Sadowski P, *et al.*: **Learning activation functions to improve deep neural networks.** *arXiv: Neural and Evolutionary Computing.* 2014.
Publisher Full Text
20. Wang Q, Ma Y, Zhao K, *et al.*: **A comprehensive survey of loss functions in machine learning.** *Ann Data Sci.* 2022; **9**: 187–212.
Publisher Full Text
21. Mao A, Mohri M, Zhong Y: **Cross-entropy loss functions: theoretical analysis and applications.** *ArXiv.* abs/2304.07288, 2023.
Publisher Full Text
22. Ruder S: **An overview of gradient descent optimization algorithms.** *ArXiv.* abs/1609.04747, 2016.
Publisher Full Text
23. Islam M, Chen G, Jin S: **An overview of neural network.** *American Journal of Neural Networks and Applications.* 2019; **5**(1): 7–11.
Publisher Full Text
24. Zaidi SSA, Ansari MS, Aslam A, *et al.*: **A survey of modern deep learning based object detection models.** *Digit Signal Process.* 2022; **126**: 103514.
Publisher Full Text
25. Jocher G: **Yolov5 by ultralytics**. 2020.
Publisher Full Text
26. Salawetz J: **What is yolov5? A guide for beginners**. 2020.
Reference Source
27. Paszke A, Gross S, Massa F, *et al.*: **Pytorch: an imperative style, high-performance deep learning library.** In: *Adv Neural Inf Process Syst.* Curran Associates, Inc, 2019; **32**: 8024–8035.
Publisher Full Text
28. Wang CY, Mark Liao HY, Wu YH, *et al.*: **CSPNet: a new backbone that can enhance learning capability of CNN.** *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops(CVPRW).* 2019; 1571–1580.
Publisher Full Text
29. Jiang T, Frøseth G, Rønnquist A: **A robust bridge rivet identification method using deep learning and computer vision.** *Eng Struct.* 2023; **283**: 115809.
Publisher Full Text
30. He K, Zhang X, Ren S, *et al.*: **Spatial pyramid pooling in deep convolutional networks for visual recognition.** *IEEE Trans Pattern Anal Mach Intell.* 2015; **37**(9): 1904–16.
PubMed Abstract | **Publisher Full Text**
31. Liu S, Qi L, Qin H, *et al.*: **Path Aggregation Network for instance segmentation.** In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2018; 8759–8768.
Publisher Full Text
32. Buslaev A, Iglovikov VI, Khvedchenya E, *et al.*: **Albumentations: fast and flexible image augmentations.** *Information.* 2020; **11**(2): 125.
Publisher Full Text
33. Gonzalez-Diaz R, Gutiérrez-Naranjo MA, Paluzo-Hidalgo E: **Topology-based representative datasets to reduce neural network training resources.** *Neural Comput Appl.* 2022; **34**(17): 14397–14413.
Publisher Full Text
34. Brodersen KH, Ong CS, Stephan KE, *et al.*: **The balanced accuracy and its posterior distribution.** In: *2010 20th International Conference on Pattern Recognition.* IEEE, 2010; 3121–3124.
Publisher Full Text
35. Sokolova M, Lapalme G: **A systematic analysis of performance measures for classification tasks.** *Inform Process Manag.* 2009; **45**(4): 427–437.
Publisher Full Text
36. Opitz J, Burst S: **Macro F1 and macro F1.** *arXiv.* 2019.
Publisher Full Text
37. Rezatofighi SH, Tsoi N, Gwak JY, *et al.*: **Generalized intersection over union: a metric and a loss for bounding box regression.** *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2019; 658–666.
Publisher Full Text
38. Henderson P, Ferrari V: **End-to-end training of object class detectors for Mean Average Precision.** In: *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision.* Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part V 13, 2017; 198–213.
Publisher Full Text
39. Shah D: **Mean average precision (map) explained: everything you need to know.** 2022; Retrieved November, 4:2022.
Reference Source
40. Myhre G, Shindell D, Pongratz J: **Anthropogenic and natural radiative forcing**. Cambridge University Press; 2014.
Publisher Full Text
41. Stocker TF, Qin D, Plattner GK, *et al.*: **Technical summary.** In: *Climate Change 2013: the Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change.* Cambridge University Press, 2013; 33–115.
Publisher Full Text
42. CodeCarbon contributors: **Codecarbon: A python library for carbon emission quantification**.
Reference Source
43. Lottick K, Susai S, Friedler SA, *et al.*: **Energy usage reports:**

environmental awareness as part of algorithmic accountability. In: *NeurIPS 2019 Workshop on Tackling Climate Change with Machine Learning.* 2019.
Publisher Full Text

44. Xu R, Wunsch D: **Clustering.** John Wiley & Sons, 2008.
Reference Source

45. Bezdek JC, Kuncheva LI: **Nearest prototype classifier designs: an experimental study.** *Int J Intell Syst.* 2001; **16**(12): 1445–1473.
Publisher Full Text

46. Liu H, Motoda H: **On issues of instance selection.** *Data Min Knowl Discov.* 2002; **6**(2): 115.
Publisher Full Text

47. Chawla S, Gionis A: **k-means–: a unified approach to clustering and outlier detection.** In: *Proceedings of the 2013 SIAM International Conference on Data Mining.* SIAM, 2013; 189–197.
Publisher Full Text

48. Li HG, Wu GQ, Hu XG, *et al.*: **K-means clustering with bagging and mapreduce.** In: *2011 44th Hawaii International Conference on System Sciences.* IEEE, 2011; 1–8.
Publisher Full Text

49. Silva V, Carlsson G: **Topological estimation using witness complexes.** *Proc Sympos Point-Based Graphics.* 2004; 157–166.
Publisher Full Text

50. Golub GH, Reinsch C: **Singular value decomposition and least squares solutions.** In: *Handbook for Automatic Computation: Volume II: Linear Algebra.* 1971; 134–151.
Publisher Full Text

51. Lee D, Seung HS: **Algorithms for non-negative matrix factorization.** *Adv Neural Inf Process Syst.* 2000; **13**.
Reference Source

52. Golub GH, Van Loan CF: **Matrix computations - 4th edition**. Johns Hopkins University Press, Philadelphia, PA, 2013.
Reference Source

53. Mairal J, Bach F, Ponce J, *et al.*: **Online dictionary learning for sparse coding.** In: *Proceedings of the 26th Annual International Conference on Machine Learning.* 2009; 689–696.
Publisher Full Text

54. Barshan E, Ghodsi A, Azimifar Z, *et al.*: **Supervised principal component analysis: visualization, classification and regression on subspaces and submanifolds.** *Pattern Recogn.* 2011; **44**(7): 1357–1371.
Publisher Full Text

55. Xanthopoulos P, Pardalos PM, Trafalis TB: **Linear discriminant analysis.** *Robust Data Mining.* 2013; 27–33.
Publisher Full Text

56. Coleman C, Yeh C, Mussmann S, *et al.*: **Selection via proxy: efficient data selection for deep learning.** *arXiv.* 2019.
Publisher Full Text

57. Mongelli M, Ferrari E, Muselli M, *et al.*: **Performance validation of vehicle platooning through intelligible analytics.** *IET Cyber-Physical Systems: Theory & Applications.* 2019; **4**(2): 120–127.
Publisher Full Text

58. **[Dataset] Dry bean dataset**. UCI Machine Learning Repository, 2020.
http://www.doi.org/10.24432/C50S4B

59. Koklu M, Ozkan IA: **Multiclass classification of dry beans using computer vision and machine learning techniques.** *Comput Electron Agric.* 2020; **174**: 105507.
Publisher Full Text

60. Ahsan MM, Mahmud MAP, Saha PK, *et al.*: **Effect of data scaling methods on machine learning algorithms and model performance.** *Technologies.* 2021; **9**(3): 52.
Publisher Full Text

61. Sharma V: **A study on data scaling methods for Machine Learning.** *Int J Global Acad Sci Res.* 2022; **1**(1): 31–42.
Publisher Full Text

62. Pedregosa F, Varoquaux G, Gramfort A, *et al.*: **Scikit-learn: machine learning in Python.** *J Mach Learn Res.* 2011; **12**: 2825–2830.
Publisher Full Text

63. Kingma DP, Ba J: **Adam: a method for stochastic optimization.** *CoRR.* 2014; abs/1412.6980.
Publisher Full Text

64. Spearman C: **The proof and measurement of association between two things.** *Am J Psychol.* 1904; **15**(1): 72–101.
Publisher Full Text

65. Lin TY, Maire M, Belongie S, *et al.*: **Microsoft coco: common objects in context**. 2015.
Publisher Full Text

66. Lin M, Chen Q, Yan S: **Network In Network**. 2014.
Publisher Full Text

67. Ikotun AM, Ezugwu AE, Abualigah L, *et al.*: **K-means clustering algorithms: a comprehensive review, variants analysis, and advances in the era of big data.** *Inform Sciences.* 2023; **622**: 178–210.
Publisher Full Text

68. 2458761304@qq.com: **[dataset] wheelchair detection dataset**. 2021.
Reference Source

69. Vasquez A, Kollmitz M, Eitel A, *et al.*: **Deep detection of people and their mobility aids for a hospital robot.** In: *Proc of the IEEE Eur Conf on Mobile Robotics (ECMR).* 2017.
Publisher Full Text

70. victosdur77: **Cimagroup/Experiments-SurveyGreenAI: V1.0 Experiments for REXASI-PRO (V1.0).** [Data], Zenodo. 2024.
http://www.doi.org/10.5281/zenodo.10844476

71. Perera-Lago J, EduPH: **Cimagroup/SurveyGreenAI: V1.0 Code for Deliverable 6.2 REXASI-PRO (V1.0).** [Code], Zenodo. 2024.
http://www.doi.org/10.5281/zenodo.10844558

Open Peer Review

Current Peer Review Status: ✔ ✔

Version 2

Reviewer Report 19 September 2024

https://doi.org/10.21956/openreseurope.20049.r44381

© **2024 Merkurjev E.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✔ **Ekaterina Merkurjev**
Michigan State University, East Lansing, Michigan, USA

The authors have addressed the concerns I have raised in my peer review report appropriately.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: graph-based machine learning

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Reviewer Report 19 July 2024

https://doi.org/10.21956/openreseurope.18970.r41935

© **2024 Yin T.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

✔ **Tianzhixi Yin**
[1] Pacific Northwest National Laboratory, Richland,, WA, USA
[2] Pacific Northwest National Laboratory, Richland,, WA, USA

This study addresses an important and timely issue by exploring methods to reduce data for deep learning, thus conserving energy during model training. The authors introduce eight data reduction methods and evaluate their effectiveness in both tabular classification and object

detection tasks using deep learning models. Notably, they employ the recently developed YOLO model for object detection, demonstrating its compatibility with the proposed reduction methods. However, I would appreciate further clarification on the justification for the chosen deep learning model architecture and hyperparameters in the tabular classification task. Specifically, the rationale behind selecting a 10-layer model with the given number of neurons needs to be elaborated, as classification results can be highly sensitive to these parameters. Additionally, considering the objective of reducing energy consumption, it would be beneficial to discuss the potential of employing a simpler model.
One minor comment pertains to the clarity of Figures 9 and 10. The overlapping data points make it challenging to distinguish between the different methods.
Overall, this paper is well-written and presents meaningful contributions. Effective data reduction methods not only save energy but also expedite the development of deep learning models.

Is the work clearly and accurately presented and does it cite the current literature?
Yes

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: deep learning, statistics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 18 July 2024

https://doi.org/10.21956/openreseurope.18970.r41936

© 2024 Merkurjev E. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

?

Ekaterina Merkurjev
[1] Michigan State University, East Lansing, Michigan, USA
[2] Michigan State University, East Lansing, Michigan, USA

Dear authors,

The work "An in-depth analysis of data reduction methods for sustainable deep learning" presents different methods for data reduction and applies them to different applications such as image data sets for object detection.

Overall, the paper has many good aspects, such as a good experimental section, organization, etc. My main concern is the exposition- the paper needs to be edited and polished before it can be indexed. I suggest that the authors carefully edit the paper to make sure the language and exposition are up to par with the journal.

In fact, there are many parts of the paper whose language can be edited. For example, in the Abstract and Introduction, some words do not need to be capitalized, such as deep learning and artificial intelligence. Moreover, many sentences are awkward/incorrect to a native English speaker, such as "We also introduce a representativeness metric based on topology to measure how similar are the reduced datasets and the full training dataset." in the Abstract, and “However, notice there is no reduction method” of Section 2.3. There are also typos such as ""For example, recent publications such as 2 and 3 explain various ways to improve”. I would also not cut Algorithms 2, 6 and 8 between two pages.

Overall, please polish the paper and go through the paper carefully to make sure the language is at the level it needs to be.

Is the work clearly and accurately presented and does it cite the current literature?
Partly

Is the study design appropriate and does the work have academic merit?
Yes

Are sufficient details of methods and analysis provided to allow replication by others?
Yes

If applicable, is the statistical analysis and its interpretation appropriate?
Yes

Are all the source data underlying the results available to ensure full reproducibility?
Yes

Are the conclusions drawn adequately supported by the results?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: graph-based machine learning

I confirm that I have read this submission and believe that I have an appropriate level of

expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.