D6.2 - Topological Techniques for Dataset Compression

Due Date of Deliverable: 31/03/2024

Responsible Partner: University of Seville (USE)

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Document Details

Title	Topological techniques for dataset compression
Deliverable No.	6.2
Work Package	6
Task	2
Deliverable Type	R - Document report
Lead Partner	University of Seville (USE)
Contributing Partner(s)	SPXL, HSOL, AIT and CNR
Due Date of Deliverable	31/03/2024
Actual Submission Date	31/03/2024
Abstract	Review and implementation for different methodologies and metrics for data reduction, as well as their impact in terms of time of execution and CO2 emissions in the context of tabular data classification and object detection
Keywords	Artificial Intelligence, energy efficiency, sustainability, data reduction

Dissemination Level		
PU	Public	X
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Change History

Version	Date	Changed by	Changes Made
1.0	13/03/2024	USE team	First draft for internal review
1.1	22/03/2024	USE team	Revisions following internal review by DFKI and SUPSI.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Table of Contents

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

List of Figures

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

List of Tables

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Acronyms Table

Acronym	Expanded Form
AI	Artificial Intelligence
Adam	Adaptive Moment Estimation
AP	Average Precision
COCO	Common Objects in Context
CLC	CLustering Centroids Selection
CPU	Central Processing Unit
CNN	Convolutional Neural Network
CSP-PAN	Cross-Ctage Partial Path Aggregation Network
DES	Distance-Entropy Selection
DICL	Dictionary Learning
DL	Deep Learning
DT	Decision Tree
FES	Forgetting Events Selection
GAP	Global Average Pooling
GPU	Graphics Processing Unit
IoU	Intersection over Union
k**-NN**	k Nearest Neighbours
MLP	Multi-Layer Perceptron
MMS	MaxMin Selection
NMF	Nonnegative Matrix Factorization
NRMD	Numerosity Reduction by Matrix Decomposition
PHL	Persistent Homology Landmarks Selection
PRD	ProtoDash Selection
RAM	Random Access Memory
ReLU	Rectified Linear Unit
REXASI-PRO	REliable & eXplainable Swarm Intelligence for People with Reduced mObility
RF	Random Forest
RMSProp	Root Mean Square Propagation

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Acronym	Expanded Form
RoI	Regions of Interest
RKM	Representative KMeans
SGD	Stochastic Gradient Descent
SPP	Spatial Pyramid Pooling
SPPF	Spatial Pyramid Pooling-Fast
SRS	Stratified Random Sampling
SVM	Support Vector Machine
SVD	Singular Value Decomposition
TDA	Topological Data Analysis
YOLO	You Only Look Once

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

1. Overview of the deliverable

1.1. Scope

In this deliverable, we will focus on the task of reducing the amount of data used to train models, making the training process less costly, and ensuring that the obtained models ultimately have similar performance.

1.2. Audience

The content of this deliverable is intended for the following audience:

- Consortium members
- European Commission
- Any other person interested in the topic

1.3. Summary

In summary, we will focus on analyzing how data reduction affects the training of deep neural networks, expanding the list of reduction methods with other algorithms developed in recent years. The two specific tasks in which we test the selected techniques are tabular data classification and object detection for image datasets. We have tested eight different data reduction methodologies and compared their performance and efficiency in four different datasets: the Collision dataset and the Dry Bean dataset, both containing tabular data focused on classification, and the Roboflow dataset and the Mobility Aid dataset, which consist of images and focused on object detection (localizing people in wheelchairs and pedestrians). Besides, we propose a specific methodology tailored for dealing with images, which is particularly important in Task 3.4 in the REXASI-PRO project, where ML models for people detection are training. Finally, we have unified the Python implementations into a user-friendly GitHub repository [1].

1.4. Structure

This deliverable is organized as follows: First, in Section 2, we present the state of the art regarding the topic of the deliverable. 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. Moving on to Section 4, we introduce different methodologies to reduce a dataset, categorizing them into four groups: statistic-based methods, geometry-based methods, ranking-based methods and wrapper methods. Later, in Section 5, the results are tested experimentally for both tabular data and object detection, providing details about the databases, the experiment setup and the parameter settings. Finally, in Section 6, we present the Python library created to utilize data reduction methods, the proposed methodology for applying them to structured data, such as images, and a summary of the main results obtained and conclusions.

[1] https://github.com/Cimagroup/Experiments-SurveyGreenAI

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

2. State of the Art

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 [47], most 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 [65] and [62] explain various ways to improve the efficiency of intelligent agents, particularly DL models. In [58], 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 decreases the training time in all cases and does not worsen the model's performance in specific cases of SVM, AdaBoost, and Bagging Classifier. In the other three algorithms, random reduction significantly decreases the F1-score.

The field of object detection and localization 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 more 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 [39] (CNNs) marked a paradigm shift by addressing the automatic feature learning capability. R-CNN variants [43] 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 [42], which proposed an innovative approach by dividing the image into a grid and making predictions of bounding boxes and classes in a single pass. Although early versions of YOLO slightly sacrificed 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, and incorporated specialized layers and attention strategies, enhancing accuracy without significantly compromising speed.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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. Finally, in Section 3.4, we discuss the different metrics to measure the performance of the different data reduction methods.

3.1. Multi-layer Perceptrons

Multi-layer perceptrons (MLP) are the simplest kind of neural networks [55]. An MLP, denoted $\mathcal{N}$, is a function that transforms input vectors through a series of smaller functions called layers. Formally, $\mathcal{N}$ 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 called units or neurons, $f_j = (f_{j,1}, \ldots, f_{j,d_j})$. 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 [2]. 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 a neural architecture, the network 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 network. 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 and are usually set by trial and error.

Given an MLP $\mathcal{N}_\theta$ and a tabular dataset $\mathcal{D}$ (whose definition can be read in Subsection 3.2), the suitability of the task is measured by a loss function $\mathcal{L}(\cdot, \mathcal{D}) : \Theta \to \mathbb{R}$ [60], 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 [35]. The search for a good choice of θ is made with an iterative training process that minimizes $\mathcal{L}$ across n_e successive epochs using the information from $\mathcal{D}$. In each epoch, $\mathcal{D}$ is randomly partitioned into a series of sub-datasets $\mathcal{B}_1, \mathcal{B}_2, \ldots$, known as batches, with a maximum size equal to β. Each batch $\mathcal{B}_j$ is then fed into $\mathcal{N}_\theta$, where its performance 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 $\mathcal{L}$ [45]. 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 so closely to the training dataset that it does not generalize well to new examples). The hy-

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

perparameters 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 [20].

3.2. Classification of Tabular Data

A dataset $\mathcal{D} = (X, f)$ of a classification problem is a pair composed of a set of examples $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$ and a function $f : X \to \{1, \ldots, c\}$. The function $p_j : X \to \mathbb{R}$ (used in Subsubsection 5.2.2) that maps $x_i = (x_{i,1}, \ldots, x_{i,d})^T$ 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 $k \in \{1, \ldots, c\}$ 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 common in computer science and is how it is used in Python code to design DL models. Given a dataset $\mathcal{D} = (X, f)$, the classification problem consists of finding an MLP $\mathcal{N} : \mathbb{R}^d \to \{1, \ldots, c\}$ that approximates f.

3.2.1. Data Reduction

Given a dataset $\mathcal{D} = (X, f)$, the goal of data reduction is to find a reduced dataset $\mathcal{D}_R = (S, g)$, where $S = \{s_1, \cdots s_n\} \subset \mathbb{R}^d$ (being $n < N$) and $g : S \to \{1, \ldots, c\}$. The class k in $\mathcal{D}_R$ will be denoted as $S_k = \{s_i \in S : g(s_i) = k\}$. In Section 4, some algorithms to extract a reduced dataset $\mathcal{D}_R$ from $\mathcal{D}$ are described. In most of them, the resulting $\mathcal{D}_R$ will be a sub-dataset of $\mathcal{D}$, which means that $S \subset X$ and $g = f|_S$. Assuming that $\mathcal{D}_R$ is representative of $\mathcal{D}$ and inherits its intrinsic properties, it should be able to be used to train $\mathcal{N}$ instead of $\mathcal{D}$, gaining efficiency and obtaining a model $\mathcal{N}_R$ 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 that we use 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.

3.3.1. Defining Object Detection

The problem of object detection [64] and localization 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, 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 of

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

"What objects are in the image?". Each detected object is usually associated with a specific class.

- Object Location: Refers to providing information about the precise spatial location of the detected objects. This involves defining the coordinates or bounding boxes surrounding each object in the image, indicating their exact location.

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 include the two metrics that we will use in our experiments to measure the similarity between the original dataset $\mathcal{D}$ and its reduced version $\mathcal{D}_R$.

Reduction Ratio

This is a common metric in the machine learning literature for comparing $\mathcal{D}$ and $\mathcal{D}_R$. The reduction ratio is just the quotient between the sizes of X_R and X. It has already been considered in publications such as [58] and [11] to measure the representativeness of $\mathcal{D}_R$ with respect to $\mathcal{D}$ and to study its effect on efficiency and performance, and we will also use it for our experiments.

ε-Representativeness

In [15], the authors introduced the concept of ε-representative datasets. The ε indicates how representative and, hence, how good is the representation of $\mathcal{D}_R$ for $\mathcal{D}$, smaller values being better. Consequently, we have $\varepsilon = 0$ if and only if $\mathcal{D}_R = \mathcal{D}$. Given a fixed isometry $i : \mathcal{D}_R \to \mathbb{R}^d$, the minimum ε such that $\mathcal{D}_R$ is a ε-representative dataset of $\mathcal{D}$ is:

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

A graphical explanation on how to calculate the ε-representativeness can be seen in Figure 1.

It has been mathematically proven in [15] that, if $\mathcal{D}_R$ is ε-representative of $\mathcal{D}$ with ε small enough, $\mathcal{D}_R$ and $\mathcal{D}$ have the same accuracy (a performance measure whose definition can be read in 3.4.2) for a perceptron (which is a MLP $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}$ with a single layer). Besides, it was demonstrated experimentally in [15] that a similar relationship exists between the ε-representativeness of $\mathcal{D}_R$ with respect to $\mathcal{D}$ and the model performance for more complex neural architectures.

3.4.2. Performance Metrics

Given a dataset $\mathcal{D} = (X, f)$ 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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

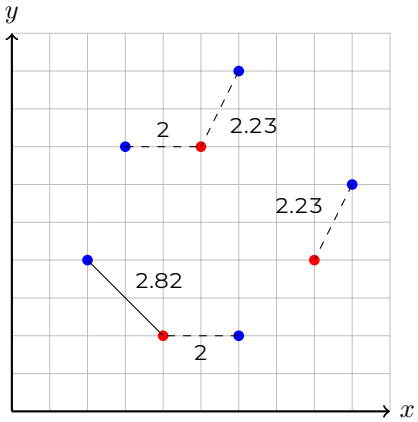

Figure 1: *Graphical explanation of the calculation of ε-representativeness for a 2D dataset with only one class. In blue, the examples of $\mathcal{D}$. In red, the examples of $\mathcal{D}_R$. For each example in $\mathcal{D}$, a line is drawn to the nearest example in $\mathcal{D}_R$ together with its distance. The ε-representativeness is equal to the maximum of these distances, which in this case is $\varepsilon = 2.82$*

	Predicted 1	$\cdots$	**Predicted k**	$\cdots$	**Predicted c**	**Total**
Actual 1	$n_{1,1}$	$\cdots$	$n_{1,k}$	$\cdots$	$n_{1,c}$	A_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
Actual k	$n_{k,1}$	$\cdots$	$n_{k,k}$	$\cdots$	$n_{k,c}$	A_k
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
Actual c	$n_{c,1}$	$\cdots$	$n_{c,k}$	$\cdots$	$n_{c,c}$	A_c
Total	P_1	$\cdots$	P_k	$\cdots$	P_c	N

Table 3: *Confusion matrix for a classification problem with c classes, together with its marginal sums.*

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 $\mathcal{D}$ and a DL model $\mathcal{N}$, 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 $\mathcal{N}(x) = j$. 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 3. Ideally, one would like to obtain a DL model $\mathcal{N}$ that fits exactly f, that is, $f(x) = \mathcal{N}(x)\ \forall x \in X$. In that case, the confusion matrix would be null except on 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.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Accuracy

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

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

Accuracy is a good metric for getting an overview of the model quality, but it can be misleading if the evaluation is reduced to it. When the training dataset is imbalanced (that is, it has some classes much more numerous than others) it is possible to find DL models that are bad at classifying the items from the least 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 performance 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. For each class k, the precision 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 resume 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 largest 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 precisions can be summarized with the macro average precision, the recall values can also be aggregated using the macro average recall, with the analogous formula:

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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

Some bibliographic sources also refer to this metric as Balanced Accuracy [6].

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 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 [51], but we prefer to use this one since it appears to be more suitable [38] and can be computed with standard Python libraries devoted to classification learning.

Intersection over Union and Mean Average Precision

Intersection over Union (IoU) [44] indicates the overlap of the predicted bounding box coordinates with the ground truth box. 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: Area where the model's prediction and the ground truth overlap.
- Union Area: Total area covered by both regions.

IoU is commonly set at 0.5, which means that the metric considers a detection successful if at least 50% of the predicted region overlaps with the actual region of the object.

The Mean Average Precision (mAP) [18, 48] is the current benchmark metric used by the computer vision research community to evaluate the robustness of

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

object detection models. The mAP metric evaluates the overall accuracy of the model across multiple Intersection over Union thresholds, so the first thing you need to do when calculating the Mean Average Precision (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 choose a single IoU threshold, such as 0.5 (denoted as 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%. On the other hand, selecting a range of thresholds, like 0.5 to 0.95 with 0.05 increments (indicated as mAP@0.5:0.95), allows you to evaluate the model's performance across a spectrum of IoU values.

The second thing we need to do is divide our detections into classes based on the detected class. We then compute the Average Precision (AP) for each class and calculate its mean, resulting in an mAP for a given IoU threshold. We calculate Precision-Recall curve points for different confidence thresholds and then we calculate the AP for each class k by 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 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 [47], 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 [47]. 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 our DL model has on the environment. These are the elapsed computing time needed to train the DL model and the estimated carbon emission into the atmosphere during the process.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Elapsed Computing Time

Measuring the elapsed computation time is as simple as setting a timer at the beginning of the model building 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 utilization 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 the change in its composition, leading to an increase in the global average temperature [53, 36]. Nevertheless, in practice, it is not easy to give an exact measure of carbon emission, since it depends on the sources of the used energy, the computer where the calculations are being 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 [10].

This approximate measure is the product of the energy consumed by 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 obtain the carbon intensity of the energy, it is necessary to know where it comes from. Each energy source emits a different amount of CO_2 for each kilowatt-hour of energy generated. 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 [33] and has already been used to estimate the carbon emission of machine learning development in [58]. The estimated carbon emission is proportional to the computing time, as we would see in Section 5.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

4. Data Reduction Methods

In this section, we introduce different methodologies 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 of probability and statistics to extract a reduced dataset $\mathcal{D}_R$ from $\mathcal{D}$.

Stratified Random Sampling (SRS)

The most simple method for data reduction is Stratified Random Sampling (SRS), as proposed in [58], where the natural strata are the c classes of $\mathcal{D}$. 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 $\mathcal{D}_R$ has the same class balance as $\mathcal{D}$. The pseudocode for SRS is shown in Algorithm 1.

Algorithm 1: SRS: Stratified Random Sampling

Data: $\mathcal{D} = (X, f),\ p \in [0, 1]$
Result: $\mathcal{D}_R = (S, g)$
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 subset $S_k \subset X_k$ with $|S_k| = n_k$;
Set $S = \bigcup_{k=1,\cdots,c} S_k$;
Set $g = f|_S$;

ProtoDash Selection (PRD)

ProtoDash Selection (PRD) [16] 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:

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

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 2: 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: $B \subset A$
Set $I = \{1, \cdots, n_A\}$ the set of indices in A;
Set $L = \emptyset$ the set of selected indices;
Set $\mathsf{K}_{i,j} = K(a_i, a_j)\ \forall i, j \in I$;
Set $\mu_j = \frac{1}{n_A} \sum_i K(a_i, a_j)\ \forall j \in I$;
Set $w_j = 0\ \forall j \in I$;
Define $l(w) = w^T \mu - \frac{1}{2} w^T \mathsf{K} w$;
Define $\nabla l(w) = \mu - \mathsf{K} w$;
while $|L| < m$ **do**
    Set $g = \nabla l(w)$;
    Set $j_0 = \arg\max_{j \in I \setminus L} g_j$;
    Update $L = L \cup \{j_0\}$;
    Solve $\xi = \arg\max_w l(w)$ subject to $w_j \geq 0\ \forall j \in I$, $w_j = 0\ \forall j \notin L$;
    Update $w = \xi$;
Set $B = \{a_j \in A : j \in L\}$;

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

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 the ProtoDash Explainer 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 the ProtoDash Explainer 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 $B = \{a_j \in A : j \in L\}$. The ProtoDash Explainer does not necessarily give an optimum solution, but it is proven in [16] that the quality of the approximate solution is lower-bounded by a fraction of the quality of the optimum solution. The pseudocode for the ProtoDash Explainer can be seen in Algorithm 2.

Then, the ProtoDash Selection algorithm just applies the ProtoDash Explainer 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 $\mathcal{D}_R$ is the union of all S_k. The pseudocode for Protodash Selection can be seen in Algorithm 3.

Algorithm 3: PRD: ProtoDash Selection

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

4.2. Geometry-based Methods

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

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 yield diverse partitions depending on the approach. For a comprehensive overview of clustering, we refer to [63].

The Clustering Centroids Selection (CLC) algorithm proposes to use k-means, one of the most-known clustering algorithms, for data reduction. This idea was stated in [5] and [31], among others. The general idea is to apply k-means on each class of $\mathcal{D}$ and include the resulting centroids in $\mathcal{D}_R$. This is the only data reduction method in this deliverable where the reduced dataset $\mathcal{D}_R$ is not necessarily a sub-dataset of $\mathcal{D}$. This method is easy to understand but can be computationally expensive for large datasets, unstable, and sensitive to outliers, as stated in [8, 27]. The pseudocode for CLC can be read in Algorithm 4.

Algorithm 4: CLC: Clustering Centroids Selection

Data: $\mathcal{D} = (X, f),\ p \in [0, 1]$
Result: $\mathcal{D}_R = (S, g)$
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 $n_k = \lfloor p \cdot |X_k| \rfloor$;
  Apply k-Means on X_k with n_k clusters;
  Select the set of centroids S_k;
Set $S = \bigcup_{k=1,\cdots,c} S_k$;
for $k = 1, \cdots, c$ **do**
  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 $\mathcal{D}_R$ is evenly spaced. It has been used in [25] for the reduction of datasets and in [50] 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:

$$\begin{aligned} D : X_k \setminus S_k &\to \mathbb{R} \\ x &\mapsto \min_{x' \in S_k} d(x, x') \end{aligned}$$

This is repeated until S_k has the required size. This method gives 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.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 5: MMS: Maxmin Selection

Data: $\mathcal{D} = (X, f),\ p \in [0,1],\ d : X \times X \to \mathbb{R}^+$
Result: $\mathcal{D}_R = (S, g)$
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 $x = \arg\max_{x \in X_k \setminus S_k} \min_{x' \in S_k} d(x, x')$;
    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) [28] is a data reduction method that tries to ensure that the resulting dataset $\mathcal{D}_R$ has relevant examples. It is based on a distance-entropy indicator that measures how informative are the different examples 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 Stratified Random Selection, 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 $\mathcal{D}_R$ 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 Distance-Entropy Selection can be seen in Algorithm 6.

To justify why the best examples are those with higher entropies, the authors of [28] use 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 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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 6: DES: Distance-Entropy Selection

Data: $\mathcal{D} = (X, f),\ p \in [0,1], p_{base} \in [0, p]$, $d : X \times X \to \mathbb{R}^+$
Result: $\mathcal{D}_R = (S, g)$
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 $S_{k,base}$;
Set $X_{base} = \bigcup_{k=1,\cdots,c} S_{k,base}$;
Set $X_{pool} = X \setminus X_{base}$;
for $x \in X_{pool}$ **do**
  for $k = 1, \cdots, c$ **do**
    Calculate the distance $d_k = d(x, p_k)$;
  for $k = 1, \cdots, c$ **do**
    Transform the distances into probabilities with $\mathrm{Softmax}(d_k) = \frac{e^{d_k}}{\sum_{j=1}^{c} e^{d_j}}$;
  Calculate the distance-entropy indicator as $E(x) = -\sum_{k=1}^{c} \mathrm{Softmax}(d_k) \cdot \log_2 \mathrm{Softmax}(d_k)$;
Set the number of examples to be added as $n_{add} = \lfloor p \cdot |X| \rfloor - |X_{base}|$;
Set $X_{add} \subset X_{pool}$ containing the n_{add} examples in X_{pool} with higher values for E;
Set $S = X_{base} \cup X_{add}$;
Set $g = f|_S$;

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) [54] 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 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:

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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

$$out^1_{PH}(x) = \max_i \{d_i - b_i : [b_i, d_i) \in B_1(\Delta_x)\} \tag{18}$$

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 [54]. 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 $\mathcal{D}_R$ is created by applying this procedure for each class. Algorithm 7 shows the pseudocode for PHL selection.

Algorithm 7: PHL: PH Landmarks Selection

Data: $\mathcal{D} = (X, f)$, $p \in [0, 1]$, $d : X \times X \to \mathbb{R}^+$, $\delta > 0$, $o_{type} \in \{$ multidimensional, restricted$\}$, $l_{type} \in \{$representative, vital$\}$
Result: $\mathcal{D}_R = (S, g)$

```
for k = 1, ··· , c do
    Set the class k as X_k = {x ∈ X : f(x) = k};
    Set the number of examples to be selected as n_k = ⌊p · |X_k|⌋;
    Set O_k = ∅ the set of super-outliers;
    if o_type = multidimensional then
        Set out_PH ≡ out^{0,1,2}_PH;
    else
        Set out_PH ≡ out^1_PH;
    for x ∈ X_k do
        Find Δ_x = {x̃ ∈ X_k \ {x} : d(x, x̃) ≤ δ};
        if |Δ_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 ∪ {x};
    if n_k ≤ |X_k \ O_k| then
        if l_type = representative then
            Set the subset S_k ⊂ X_k \ O_k with the n_k lowest values for out_PH;
        else
            Set the subset S_k ⊂ X_k \ O_k with the n_k highest values for out_PH;
    else
        Select a random subset R_k ⊂ O_k with |R_k| = |X_k| − n_k ;
        Set S_k = X_k \ R_k;
Set S = ⋃_{k=1,··· ,c} S_k;
Set g = f|_S;
```

Numerosity Reduction by Matrix Decomposition (NRMD)

Numerosity Reduction by Matrix Decomposition (NRMD) [11] is a method that leverages matrix decomposition to rank examples in a dataset $\mathcal{D} = (X, f)$. To use this method, it is necessary to use the tabular representation of $\mathcal{D}$ that we saw in

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 8: Calculate scores from a matrix

Data: $A \in \mathbb{R}^{N \times d}$, $d_{type} \in \{\text{SVD}, \text{NMF}, \text{PLU}, \text{QR}, \text{DL}, \text{SPCA}, \text{FLDA}\}$
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;
Set $\tilde{V}$ as the normalization of V;
if $d_{type} \in \{SVD, SPCA, FLDA\}$ **then**
    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;
else
    Set $w \in \mathbb{R}^r$ with $w_i = \frac{2i}{r(r+1)}$;
Calculate the scores vector $s = -\log(|\tilde{A}\tilde{V}^T|_\varepsilon)w$;

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) [13], Non-negative Matrix Factorization (NMF) [26], PLU Decomposition [14], QR decomposition [14], Dictionary Learning (DICL) [34], Supervised Principal Component Analysis (SPCA) [4] and Fisher Linear Discriminant Analysis (FLDA) [61]. 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.

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 $\mathcal{D}_R$ is finally formed by the examples with the highest values for s, which are considered the most useful in terms of internal representation and discrimination between classes. The pseudocode for Numerosity Reduction by Matrix Decomposition can be seen in Algorithm 9.

4.4. Wrapper Methods

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 9: NRMD: Numerosity Reduction by Matrix Decomposition

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

Forgetting Events Selection (FES)

Forgetting Events Selection (FES) is a data reduction method that leverages the evolution of accuracy throughout neural network 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 [56] 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 [9], our FES implementation only counts the forgetting events during the first $e_{initial}$ epochs of the training process. At that point, the algorithm reduces $\mathcal{D}$ by selecting examples with more forgetting events and performs the remaining epochs using only $\mathcal{D}_R$ as a training dataset. To ensure that all classes are well represented in $\mathcal{D}_R$, the selection is made class by class. Algorithm 10 shows how to apply FES selection during the training of a DL model $\mathcal{N}$.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 10: FES: Forgetting Events Selection

Data: $\mathcal{D} = (X, f)$, $p \in [0, 1]$, $\mathcal{N} : \mathbb{R}^d \to \{1, \cdots, c\}$, $e_{initial}$, e_{total}
Result: $\mathcal{D}_R = (S, g)$, $\mathcal{N}$

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 $\mathcal{N}$ using $\mathcal{D}$;
  for $x \in X$ **do**
    if $\mathcal{N}(x) = f(x)$ **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 $\mathcal{N}$ using $\mathcal{D}_R = (S, g)$;

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

5. Experiments

In this section, we present the datasets used for the experiments, the parameter settings, the setup for the experiments, and finally, the obtained results.

5.1. Code Availability

The source code of the experiments is available in the GitHub repository `https://github.com/Cimagroup/Experiments-SurveyGreenAI`.

5.2. 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 on the two datasets we used. Finally, we show the results obtained for both experiments and discuss the main conclusions.

5.2.1. Datasets for Classification

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

Collision dataset

Provided by our colleagues Maurizio Mongelli and Sara Narteni from the CNR REXASI-PRO partner. The classification task consists of predicting whether a platoon of vehicles will collide based on features such as the number of cars 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 in experiments 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. “N” is the number of vehicles in the platoon, while “m” is the vehicle mass expressed in kg.

Dry Bean dataset

This dataset (see [1] and [24]) 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. 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.

This project has received funding from the European Union’s Horizon Europe research and innovation program under grant agreement **No 101070028-2**

- 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 experimentation 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.

5.2.2. Methodology

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

1. **Dataset Preprocessing:**

   It is 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 [3, 49]. In our case, we decided to apply the scikit-learn function *MinMaxScaler* [41].

   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 values lie in it. Rescaling with *MinMaxScaler* is just changing the 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 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 neural 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 $\text{ReLU}(x) = \max(0, x)$. Additionally, 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:

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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

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:

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

For the Dry Bean dataset, the last layer has 7 neurons, one for each class (that is, $O = \mathbb{R}^7$). The logits $z_{i,0}, \cdots, z_{i,6}$ 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 towards the most populated ones.

In both cases, we used the Adam optimizer [23] to minimize the loss function, specifying a learning rate of $\gamma = 0.001$ and letting 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 will be trained using $\mathcal{D}_{train}$ and its performance will be 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 $p_{test} = 0.25$.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

- **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 computed.
- **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 $\mathcal{D}_{train,R}$ 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 $p \in \{0.1, 0.2, \cdots, 0.9\}$.

This last step 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.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Algorithm 11: Pipeline of the Experiments for Tabular Data Classification

Data: $\mathcal{D} = (X, f)$
Dataset Preprocessing;
Scale $\mathcal{D}$ using MinMaxScaler;
Fixing the architecture and hyperparameters;
Set a test size proportion $p_{test} \in (0, 1)$;
Set a neural architecture and create the DL model $\mathcal{N}$;
Set a loss function $\mathcal{L} : \Theta \rightarrow \mathbb{R}^+$;
Set an optimization algorithm to minimize $\mathcal{L}$ and its associated hyperparameters;
Set a regularization technique and its associated hyperparameters;
Set a number of training epochs $n_e \in \mathbb{N}$;
Set a number of initial training epochs for FES reduction $n_i \in \mathbb{N}$, with $n_i < n_e$;
Set a batch size $\beta \in \mathbb{N}$;
Set a number of iterations $n_{iter} \in \mathbb{N}$;
Data reduction and model training;
for $i = 1$ **to** n_iter **do**
    Train-Test dataset split;
    Set $N_{test} = \lfloor p_{test} \cdot N \rfloor$;
    Split $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, being the size of $\mathcal{D}_{test}$ equal to N_{test};
    Training with no reduction;
    Train $\mathcal{N}$ for n_e epochs using $\mathcal{D}_{train}$;
    Calculate the computing time and carbon emission of the training;
    Validate the model with $\mathcal{D}_{test}$ and calculate the accuracy, macro average precision, macro average recall and macro average F1-score;
    Training + reduction for non-wrapper methods:;
    foreach *non-wrapper method* **do**
        for $p \in \{0.1, 0.2, \cdots, 0.9\}$ **do**
            Get $\mathcal{D}_{train,R}$ as the reduced dataset of $\mathcal{D}_{train}$ with the corresponding data reduction method and the reduction ratio p;
            Calculate the ε-representativeness of $\mathcal{D}_{train,R}$ respect to $\mathcal{D}_{train}$;
            Train $\mathcal{N}$ for n_e epochs using $\mathcal{D}_{train,R}$;
            Calculate the computing time and carbon emission of the reduction and the training;
            Validate $\mathcal{N}$ using $\mathcal{D}_{test}$ and calculate the accuracy, macro average precision, macro average recall and macro average F1-score;
    Training + reduction for FES;
    for $p \in \{0.1, 0.2, \cdots, 0.9\}$ **do**
        Train $\mathcal{N}$ for n_i epochs using $\mathcal{D}_{train}$;
        Get $\mathcal{D}_{train,R}$ as the reduced dataset of $\mathcal{D}_{train}$ with FES reduction and the reduction ratio p;
        Calculate the ε-representativeness of $\mathcal{D}_{train,R}$ respect to $\mathcal{D}_{train}$;
        Train $\mathcal{N}$ for $n_e - n_i$ epochs using $\mathcal{D}_{train,R}$;
        Calculate the computing time and carbon emission of the reduction and the training;
        Validate $\mathcal{N}$ using $\mathcal{D}_{test}$ and calculate the accuracy, macro average precision, macro average recall and macro average F1-score;

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

5.2.3. 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 influenced by extreme values.

Collision Dataset

The median results we obtained for the efficiency metrics (computing time and carbon emission) can be seen in Tables 4 and 5. 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. It is important to clarify that this occurs because we are primarily measuring CO2 emissions during the training time, which present the same characteristics regardless of the method used. However, as will be seen in the experiments for object detection, the CO2 emitted during the data reduction phase is not proportional to the computing time, because the reduction methods are not similar in terms of code. In general, the use of data reduction methods before network 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 ratio is equal to 90%. That suggests 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 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 the three reduction methods only to perform 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 two top data reduction methods are SRS and NRMD, with similar results in both the computation time and carbon emission. If we observe the respective columns in Tables 4 and 5, we can see that the total time and carbon emission from the reduction and training are proportional to the number of examples in the reduced dataset. That indicates that the reduction takes a small time in the process, and almost all the measured time and carbon correspond to the network training. We observed that the efficiency of NRMD reduction depends on the type of matrix decomposition selected. We used SVD decomposition but the results may be different if we select another decomposition type.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	CLC	MMS	DES	NRMD	FES
10%	59.68	153.03	70.76	72.62	60.76	276.12
20%	119.73	221.47	138.94	142.89	121.15	311.98
30%	179.33	305.67	209.66	212.18	181.22	349.14
40%	238.67	368.85	275.61	279.7	240.73	390.32
50%	296.24	440.14	342.89	344.82	298.63	427.5
60%	358.38	506.34	407.13	398.71	357.88	462.85
70%	414.35	568.4	463.99	468.55	413.52	500.52
80%	472.5	649.97	536.28	534.12	474.81	539.93
90%	533.91	731.77	593.33	597.94	536.11	578.62
100%	592.48	592.48	592.48	592.48	592.48	592.48

Table 4: *Collision: Reduction + training time. This table displays in seconds the time during the reduction process and during the training of the model. In the columns, we can see the different methods, and in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and in gray, the best reduction method for each percentage of data used in training.*

	SRS	CLC	MMS	DES	NRMD	FES
10%	0.263	0.556	0.3	0.308	0.267	1.028
20%	0.474	0.797	0.546	0.558	0.479	1.155
30%	0.683	1.125	0.787	0.802	0.691	1.291
40%	0.897	1.331	1.027	1.043	0.881	1.438
50%	1.105	1.572	1.262	1.254	1.11	1.558
60%	1.315	1.83	1.45	1.452	1.319	1.655
70%	1.476	2.07	1.703	1.677	1.514	1.779
80%	1.713	2.344	1.946	1.945	1.728	1.967
90%	1.943	2.636	2.156	2.17	1.947	2.098
100%	2.146	2.146	2.146	2.146	2.146	2.146

Table 5: *Collision: Reduction + training carbon. This table displays the grams of CO_2 emitted during the training of the model. In the columns, we can see the different methods, and in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and in gray, the best reduction method for each percentage of data used in training.*

About the ε-representativeness of the reduced datasets with respect to the whole training dataset, the median results can be seen in Table 6. 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, the data reduction method that has the highest ε values for all the possible reduction ratios is NRMD.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	CLC	MMS	DES	NRMD	FES
10%	0.611	0.55	0.403	0.628	0.959	0.823
20%	0.56	0.485	0.37	0.6	0.948	0.823
30%	0.55	0.464	0.333	0.561	0.944	0.6
40%	0.533	0.438	0.309	0.546	0.931	0.586
50%	0.533	0.439	0.292	0.533	0.891	0.573
60%	0.533	0.395	0.273	0.533	0.82	0.573
70%	0.461	0.394	0.261	0.511	0.754	0.474
80%	0.457	0.353	0.242	0.465	0.726	0.474
90%	0.433	0.31	0.228	0.433	0.612	0.398
100%	0.0	0.0	0.0	0.0	0.0	0.0

Table 6: *Collision: Percentage vs epsilon. This table displays the ε-representativeness of $\mathcal{D}_R$ for $\mathcal{D}$ using different reduction methods and percentages of data. In columns, we can see the different methods and, in rows, the different percentages of data used for the training. We have marked in red the values doing the training with the complete dataset, that is, the reference, and, in gray, the best reduction method for each percentage of data used in the training.*

The results on accuracy, macro average precision, macro average recall and macro average F1-score can be seen in Tables 7, 8, 9 and 10 respectively. With respect to the 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 complete training dataset. We 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 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 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 other ratios the best one is DES. macro average precision dropping as training data set size decreases also appears to be approximately linear except for all the methods but FES, which suffers a significant drop when the reduction ratio is under 30%.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	CLC	MMS	DES	NRMD	FES
10%	0.88	0.883	0.851	0.892	0.821	0.583
20%	0.896	0.886	0.875	0.902	0.84	0.74
30%	0.895	0.891	0.89	0.906	0.854	0.807
40%	0.905	0.904	0.896	0.908	0.865	0.896
50%	0.905	0.906	0.899	0.908	0.859	0.918
60%	0.908	0.901	0.909	0.912	0.875	0.915
70%	0.912	0.912	0.904	0.911	0.893	0.916
80%	0.909	0.911	0.911	0.913	0.905	0.919
90%	0.912	0.911	0.907	0.912	0.906	0.918
100%	0.91	0.91	0.91	0.91	0.91	0.91

Table 7: *Collision: Percentage vs accuracy. This table displays the accuracy of the model. In the columns, we can see the different methods, and in the rows, the different percentages of data used for training. We have marked in red the values obtained during the training with the complete dataset, which is the reference, and, in gray, the best reduction method for each percentage of data used in training.*

	SRS	CLC	MMS	DES	NRMD	FES
10%	0.865	0.869	0.848	0.88	0.815	0.564
20%	0.883	0.871	0.869	0.898	0.836	0.776
30%	0.881	0.876	0.878	0.898	0.851	0.82
40%	0.893	0.891	0.881	0.905	0.862	0.902
50%	0.896	0.895	0.885	0.897	0.845	0.914
60%	0.897	0.888	0.899	0.901	0.86	0.903
70%	0.902	0.901	0.89	0.902	0.88	0.912
80%	0.898	0.901	0.899	0.9	0.892	0.915
90%	0.9	0.903	0.895	0.902	0.895	0.913
100%	0.901	0.901	0.901	0.901	0.901	0.901

Table 8: *Collision: Percentage vs macro average precision. This table displays the macro average precision of the model. In the columns, we can see the different methods, and, in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and, in gray, the best reduction method for each percentage of data used in training.*

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 if the reduction ratio is very low, 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 rest in terms of macro average recall. Depending on the reduction ratio, the method that best preserves the macro average recall is one or another. All methods except NRMD have given the best median result for some of the chosen percentages.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	CLC	MMS	DES	NRMD	FES
10%	0.882	0.882	0.843	0.889	0.784	0.534
20%	0.895	0.895	0.879	0.891	0.805	0.72
30%	0.898	0.898	0.886	0.896	0.825	0.752
40%	0.904	0.903	0.902	0.898	0.841	0.869
50%	0.9	0.907	0.905	0.905	0.864	0.909
60%	0.905	0.908	0.906	0.909	0.879	0.918
70%	0.907	0.908	0.905	0.905	0.894	0.901
80%	0.907	0.909	0.909	0.906	0.902	0.907
90%	0.909	0.906	0.912	0.908	0.908	0.91
100%	0.908	0.908	0.908	0.908	0.908	0.908

Table 9: *Collision: Percentage vs macro average recall. This table displays the macro average recall of the model. In the columns, we can see the different methods, and in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and, in gray, the best reduction method for each percentage of data used in training.*

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.

	SRS	CLC	MMS	DES	NRMD	FES
10%	0.871	0.875	0.83	0.882	0.795	0.562
20%	0.887	0.879	0.868	0.891	0.816	0.713
30%	0.887	0.885	0.879	0.897	0.835	0.768
40%	0.898	0.897	0.889	0.899	0.847	0.882
50%	0.895	0.899	0.892	0.902	0.85	0.909
60%	0.901	0.895	0.903	0.904	0.865	0.908
70%	0.904	0.905	0.898	0.903	0.884	0.907
80%	0.901	0.903	0.903	0.905	0.897	0.911
90%	0.904	0.904	0.901	0.905	0.9	0.911
100%	0.904	0.904	0.904	0.904	0.904	0.904

Table 10: *Collision: Percentage vs macro average F1-score. This table displays the macro average F1-score of the model. In the columns, we can see the different methods, and, in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and in gray, the best reduction method for each percentage of data used in training.*

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

and each iteration in the experiment (in total there are 6 reduction methods $\times$ 10 iterations $= 60$ pairs $(\varepsilon, \text{F1})$ for each p). We computed for each p the Spearman's rank correlation coefficient [52] 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, implying that as one variable increases, the other variable also increases, ρ close to -1 indicates a strong negative monotonic correlation, and ρ similar 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 likely not null, 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 $(\varepsilon, \text{F1})$ altogether.

	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

Table 11: *Collision: Correlation between ε-representativeness and macro average F1-score. This table displays the non-parametric Spearman correlation coefficient and its p-value. We have marked in gray the columns with a significative correlation setting a significance level of 5%, that is with a p-value less or equal than 0.05*

The results that we got can be seen in Table 11. 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 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 Tables 12 and 13. 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 Tables 12 and 13). In that situation, both the computation time and the carbon emission 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 even CLC reduction, which was the slowest method for the Collision dataset, 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.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	20.04	32.2	20.21	20.12	20.59	18.68	28.7	89.11
20%	39.79	63.17	40.08	39.99	40.61	37.83	48.66	102.66
30%	59.68	93.95	59.75	60.03	60.73	59.92	68.57	116.22
40%	79.61	123.54	79.58	79.68	80.78	79.66	88.4	129.85
50%	99.43	155.65	99.85	99.53	100.7	99.52	108.22	143.55
60%	119.51	186.72	119.07	119.42	120.85	119.79	127.85	156.39
70%	139.62	216.93	139.02	139.25	140.27	139.51	148.02	170.35
80%	159.27	251.16	158.59	159.39	160.42	159.05	168.07	184.37
90%	179.19	286.36	178.53	178.83	179.95	179.44	188.09	197.12
100%	198.8	198.8	198.8	198.8	198.8	198.8	198.8	198.8

Table 12: *Dry Bean: Reduction + training time. This table displays in seconds the time during the reduction process and during the training of the model. In the columns, we can see the different methods, and in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and in gray, the best reduction method for each percentage of data used in training.*

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	0.071	0.114	0.072	0.071	0.073	0.066	0.102	0.316
20%	0.141	0.224	0.142	0.142	0.144	0.134	0.173	0.364
30%	0.212	0.334	0.212	0.213	0.216	0.213	0.243	0.413
40%	0.283	0.439	0.283	0.283	0.287	0.283	0.314	0.461
50%	0.353	0.553	0.354	0.353	0.358	0.353	0.384	0.51
60%	0.424	0.663	0.423	0.424	0.429	0.425	0.454	0.555
70%	0.496	0.77	0.494	0.494	0.498	0.495	0.526	0.605
80%	0.565	0.892	0.563	0.566	0.57	0.565	0.597	0.655
90%	0.636	1.017	0.634	0.635	0.639	0.637	0.668	0.7
100%	0.706	0.706	0.706	0.706	0.706	0.706	0.706	0.706

Table 13: *Dry Bean: Reduction + training carbon. This table displays the grams of CO_2 emitted during the training of the model. In the columns, we can see the different methods, and in the rows, the different percentages of data used for training. We have marked in red the values obtained during training with the complete dataset, which is the reference, and in gray, the best reduction method for each percentage of data used in training.*

The median values on the ε-representativeness, which can be seen in Table 14, 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 give the less ε-representative reduced datasets.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	0.902	0.868	0.273	0.148	0.582	0.926	0.895	0.792
20%	0.897	0.868	0.218	0.112	0.359	0.926	0.776	0.635
30%	0.868	0.868	0.21	0.095	0.271	0.926	0.413	0.635
40%	0.776	0.868	0.168	0.089	0.244	0.926	0.288	0.393
50%	0.33	0.868	0.168	0.074	0.238	0.901	0.231	0.351
60%	0.33	0.868	0.132	0.066	0.238	0.901	0.217	0.218
70%	0.33	0.868	0.13	0.059	0.185	0.901	0.227	0.218
80%	0.226	0.868	0.116	0.053	0.161	0.901	0.172	0.178
90%	0.226	0.868	0.108	0.045	0.116	0.862	0.21	0.165
100%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 14: *Dry Bean: Percentage vs epsilon. This table displays the ε-representativeness of $\mathcal{D}_R$ for $\mathcal{D}$ using different reduction methods and percentages of data. In columns, we can see the different methods, and in the rows, the different percentages of data used for the training. We have marked in red the values doing the training with the complete dataset,that is the reference, and in gray, the best reduction method for each percentage of data used in the training.*

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 Tables 15, 16, 17 and 18 respectively. In this experiment, the reference case has a median accuracy in the test dataset of 89.9% 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 that maximum accuracy loss.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	0.794	0.717	0.775	0.685	0.641	0.525	0.758	0.665
20%	0.797	0.732	0.812	0.731	0.719	0.644	0.796	0.763
30%	0.833	0.8	0.845	0.761	0.717	0.649	0.841	0.83
40%	0.84	0.829	0.848	0.827	0.749	0.723	0.871	0.865
50%	0.875	0.84	0.845	0.867	0.843	0.784	0.87	0.889
60%	0.875	0.871	0.875	0.856	0.717	0.809	0.882	0.889
70%	0.89	0.881	0.89	0.89	0.86	0.818	0.863	0.888
80%	0.895	0.888	0.885	0.874	0.864	0.843	0.896	0.887
90%	0.88	0.898	0.891	0.882	0.865	0.859	0.888	0.888
100%	0.899	0.899	0.899	0.899	0.899	0.899	0.899	0.899

Table 15: *Dry Bean: Percentage vs accuracy. This table displays the accuracy of the model. In the columns, we can see the different methods, and, in the rows, the different percentages of data used for training. We have marked in red the values obtained during the training with the complete dataset, which is the reference, and in gray, the best reduction method for each percentage of data used in training.*

With respect to 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.

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	0.785	0.768	0.785	0.725	0.745	0.617	0.747	0.713
20%	0.819	0.793	0.822	0.801	0.767	0.712	0.809	0.81
30%	0.868	0.857	0.863	0.828	0.803	0.706	0.876	0.879
40%	0.863	0.85	0.872	0.857	0.848	0.772	0.884	0.903
50%	0.902	0.893	0.878	0.896	0.889	0.801	0.896	0.91
60%	0.911	0.893	0.905	0.887	0.802	0.818	0.91	0.914
70%	0.907	0.905	0.913	0.915	0.894	0.846	0.896	0.91
80%	0.918	0.907	0.911	0.902	0.909	0.862	0.92	0.914
90%	0.903	0.915	0.913	0.908	0.906	0.884	0.907	0.911
100%	0.917	0.917	0.917	0.917	0.917	0.917	0.917	0.917

Table 16: *Dry Bean: Percentage vs macro average precision. This table displays the macro average precision of the model. In the columns, we can see the different methods, and, in the rows, the different percentages of data used for training. We have marked in red the values obtained during the training with the complete dataset, which is the reference, and, in gray, the best reduction method for each percentage of data used in training.*

We can see similar results when analyzing 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.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	0.776	0.731	0.77	0.692	0.718	0.413	0.765	0.696
20%	0.776	0.729	0.804	0.738	0.753	0.517	0.781	0.759
30%	0.816	0.803	0.838	0.766	0.779	0.52	0.834	0.84
40%	0.836	0.841	0.844	0.819	0.799	0.592	0.888	0.881
50%	0.877	0.857	0.864	0.874	0.877	0.65	0.878	0.895
60%	0.892	0.88	0.878	0.868	0.792	0.757	0.9	0.901
70%	0.904	0.888	0.907	0.89	0.885	0.779	0.885	0.905
80%	0.903	0.902	0.893	0.886	0.888	0.819	0.907	0.905
90%	0.885	0.912	0.898	0.887	0.883	0.864	0.896	0.903
100%	0.911	0.911	0.911	0.911	0.911	0.911	0.911	0.911

Table 17: *Dry Bean: Percentage vs macro average recall. This table displays the macro average recall of the model. In the columns, we can see the different methods, and, in the rows, the different percentages of data used for training. We have marked in red the values obtained during the training with the complete dataset, which is the reference, and, in gray, the best reduction method for each percentage of data used in training.*

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.

	SRS	PRD	CLC	MMS	DES	NRMD	PHL	FES
10%	0.747	0.727	0.807	0.771	0.763	0.628	0.724	0.643
20%	0.764	0.721	0.796	0.747	0.739	0.759	0.803	0.737
30%	0.838	0.797	0.843	0.78	0.782	0.758	0.834	0.844
40%	0.879	0.84	0.844	0.812	0.847	0.773	0.883	0.891
50%	0.88	0.862	0.862	0.877	0.879	0.822	0.883	0.9
60%	0.896	0.882	0.883	0.866	0.773	0.83	0.901	0.904
70%	0.902	0.892	0.907	0.899	0.885	0.864	0.882	0.908
80%	0.909	0.903	0.896	0.887	0.891	0.879	0.912	0.908
90%	0.889	0.914	0.904	0.891	0.888	0.892	0.9	0.904
100%	0.913	0.913	0.913	0.913	0.913	0.913	0.913	0.913

Table 18: *Dry Bean: Percentage vs macro average F1-score. This table displays the macro average F1-score of the model. In the columns, we can see the different methods, and, in the rows, the different percentages of data used for training. We have marked in red the values obtained during the training with the complete dataset, which is the reference, and, in gray, the best reduction method for each percentage of data used in training.*

Description of data structure for accessing the final results

The results obtained can be consulted and reproduced using the code provided in the GitHub repository `https://github.com/Cimagroup/Experiments-SurveyGreenAI`.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

The results of both experiments were saved 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 p value in that iteration. The results obtained when the training dataset is not reduced are saved 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 $\mathcal{D}_{\text{train},R}$ with respect to $\mathcal{D}_{\text{train}}$
- **acc:** To store the accuracy of the model over $\mathcal{D}_{\text{test}}$
- For each class k:
  - **pre_k:** To store the model precision for class k over $\mathcal{D}_{\text{test}}$
  - **rec_k:** To store the model recall for class k over $\mathcal{D}_{\text{test}}$
  - **f1_k:** To store the model F1-score for class k over $\mathcal{D}_{\text{test}}$
- **pre_avg:** To store the model macro average precision for class k over $\mathcal{D}_{\text{test}}$
- **rec_avg:** To store the model macro average recall for class k over $\mathcal{D}_{\text{test}}$
- **f1_avg:** To store the model macro average F1-score for class k over $\mathcal{D}_{\text{test}}$

Once we have this results dictionary, the next step is to summarize the information of all the iterations in a more simple dictionary. This object, which we will 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 tables that we have seen in this subsection present the metrics from the median results dictionary.

5.3. Experiments for Object Detection

In this subsection, we describe the methodology we have used in our experiments to extend data reduction techniques to images. Observe that we need to adapt the methodology depending on the type of data reduction method. Later, we present the datasets used for the experiments done on object detection, including the parameter settings and the setup, and finally, we discuss the results that we have obtained. For our experiments we use YOLOv5, so first we are going to explain this arquitecture in detail.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

5.3.1. Object Detection with YOLOv5

YOLO [42], an abbreviation for "You Only Look Once", is conceived to address object detection with a comprehensive and efficient approach. Its main objective is to perform accurate detections of multiple objects in a single pass through the image, minimizing duplication of efforts and optimizing processing speed. YOLO is a single-stage architecture with which object detection is performed by viewing the problem as a regression problem to spatially separate the bounding box and the probability classes associated with the bounding box. A neural network predicts the bounding box and prediction class directly from the entire image from a single evaluation. The fifth version of YOLO, named YOLOv5 [22, 46], is the first native release of models in the YOLO family written in Pytorch [40]. YOLOv5 is fast, with inference times up to 0.007 seconds per image, meaning 140 frames per second. Figure 2 shows the detailed architecture of YOLOv5. Specifically, YOLOv5 consists of three essential components:

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) [59] into Darknet to separate the base layer feature map into two parts and then combine them through a cross-stage hierarchy as shown in Fig. 2. X is a variable in CSP1_X and CSP2_X, meaning the number of BottleNecks in the network. This enhanced CSPNet, built upon Darknet53, residual blocks, depthwise separable convolutions, and preactivation for improved efficiency and feature representation. For instance, given an RGB input 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 Fig. 2. A spatial pyramid pooling (SPP) [17] 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 twice. It aggregates information received from inputs and returns a fixed-length output. PAN [32] 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 Fig. 2.

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 [7]. It also improves the grid sensitivity to make it more stable to runaway gradients.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

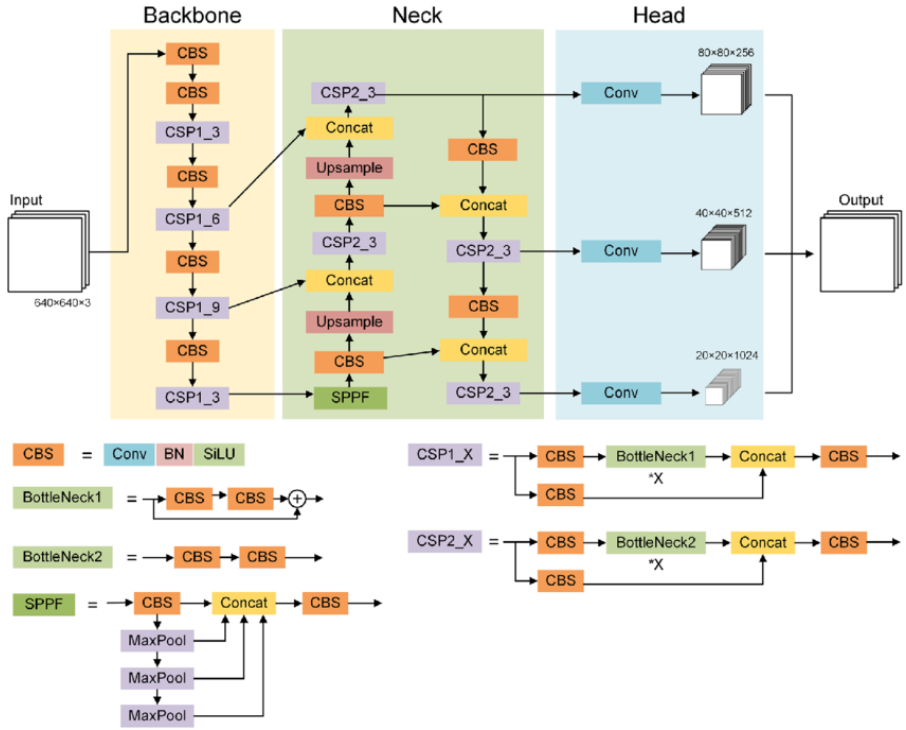

Figure 2: *Architecture of YOLOv5 [21], including three main parts: backbone, neck and head. The "backbone" is responsible for extracting fundamental features from the image, such as edges and textures. The "neck" is used to extract feature pyramids, which helps the model to generalize well to objects of different sizes and scales. Finally, the "head" is responsible for the final prediction, generating the coordinates and classes of the detected objects.*

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 experimentation.

5.3.2. Methodology

The proposed methodology for the data-dependent methods, illustrated in Figure 3, consists of the following five steps:

1. **Feature extraction**: The objective of this stage is to convert the raw pixel values of images into a set of meaningful and concise features that capture pertinent information. These features should empower the model to distinguish between different patterns, objects, or structures within the images. This process entails 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) [30], 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 project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

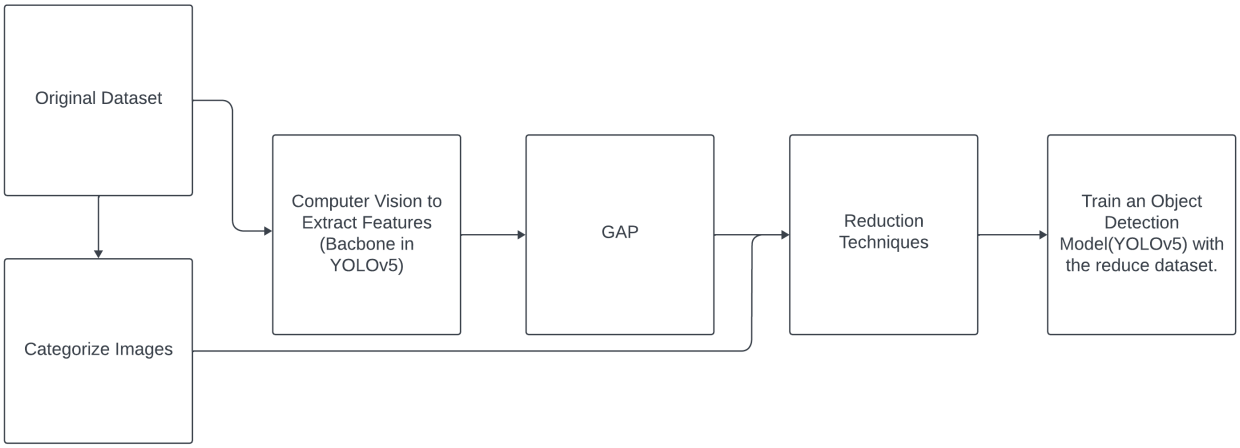

Figure 3: *Diagram of the workflow for the proposed methodology to apply data reduction techniques on images dataset.*

This approach can lead to improved performance and efficiency when fine-tuning or adapting these models to specific applications.

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 5.3.3, where we detail how each dataset is categorized.

3. **Global Average Pooling**: Regarding the output of Step 1, we apply Global Average Pooling [12, 29] to the output of the last layer of the backbone, in order to transform the features 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 decrease 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. 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. We kept the backbone frozen and and we did the fine-tuning on the rest of the model. This approach leverages prior knowledge gained during initial training, enabling more efficient adaptation to the new task without completely discarding previously learned information.

When employing wrapper methods, we must adopt a slightly different methodology than the one previously described (see Figure 4). Initially, we must categorize the images, similar to the preceding methodology, as these images are

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

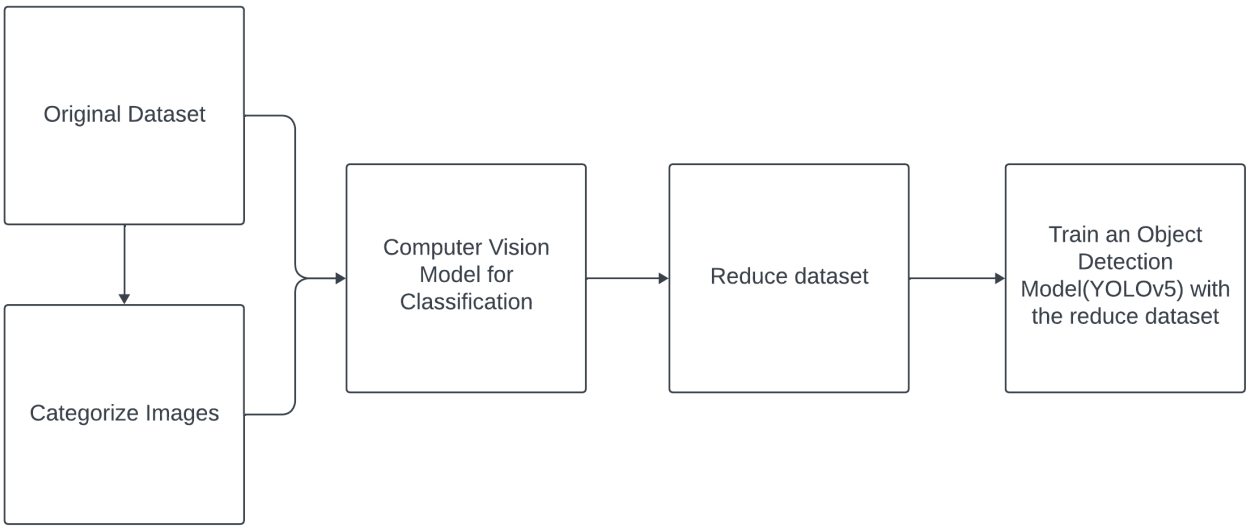

Figure 4: *Diagram of the workflow for the proposed methodology to apply wrapper methods.*

intended for object detection and lack specific labels, instead featuring multiple elements within them. Subsequently, we create 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.

Additionally, to apply the CLC method to images, we introduce a slight modification by employing KMeans [19] 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 $\mathcal{D}_R$ 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).

5.3.3. Datasets for Object Detection

Roboflow

hThe dataset Roboflow[2] 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. To label this dataset, we stick to the following criteria:

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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

Mobility Aid

The dataset Mobility Aid [3] [57] is composed of 17079 RGB images, 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), person pushing a wheelchair (3323, it will be annotated as PW), person with crutches (5374, it will be annotated as C) and person 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. To label this dataset, we stick to the following criteria:

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

5.3.4. Parameter Setting

For the fine-tuning of YOLOv5, we maintain the backbone frozen (pretrained on the COCO dataset) while training the remaining parts 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 employ the SGD optimizer, and the learning rate is set to 0.01 for both datasets. The number of classes (nc) is adjusted based on the specific dataset, such as 2 for the Roboflow dataset and 5 for the Mobility Aid dataset.

5.3.5. Experiments Setup

We utilized Python 3.9 and PyTorch [40] on Ubuntu 20.04 to conduct 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 conducted training five times for the reduced training dataset. We calculated the arithmetic mean and standard deviation of the results for the test set to gauge the performance of each reduction method. For the Roboflow dataset,

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

given 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 implemented rate reductions of 75% and 90%.

5.3.6. Comparison Metrics

We use five metrics to assess the performance of training YOLOv5 with the complete dataset compared to training it with different reduction methods and reduction rates. Initially, we employ 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 more are considered. 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.

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

5.3.7. Results and Discussion

The source code to implement data reduction techniques in these datasets is available at `https://github.com/Cimagroup/ExperimentsOD-SurveyGreenAI` in the folder ObjectDetection.

Roboflow Dataset

In our initial experiment, we compared the performance of fine-tuning with the complete training dataset against training with a reduced dataset using various reduction methods. Table 19 displays the reduction time, ε-representativeness of the $\mathcal{D}_r$, 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. Additionally, we reduced the training time by approximately 40%, decreasing from 9 minutes and 20 seconds to around 5 minutes and 45 seconds with 50% of the samples. This time reduction was accompanied by a similar decrease (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 light, except for FES, which exhibited excessive duration compared to other methods. Conversely, a lower ε-representativeness did not seem decisive for performance improvement or degradation.

Table 20 presents 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 performance reduction across all metrics with all methods, SRS, MMS, RKMEANS, and PHL emerged as more resilient options. In particular, SRS is the one that best

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

maintains precision and mAP. On the contrary, NRMD showed the most significant performance loss. 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 saw only a 3% reduction compared to the substantial computational time and CO_2 emission savings of 60%.

In Figure 5, 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, some performance loss is evident. SRS stands out as the most effective method. Additionally, a general improvement in accuracy is observed for Wheelchairs compared to People, potentially attributed to a slight imbalance in the dataset between the two categories. We only present the mAP figure (Figure 5) because it is the most comprehensive performance metric for object detection evaluation.

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

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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: 0.943±0.01 P: 0.911±0.014 W: 0.971±0.005	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 19: *Table results for Roboflow dataset and 50% reduction rate. The 'Precision', 'Recall' and 'mAP@.5' columns display 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 serve as the reference. Additionally, in gray, we highlight the best reduction method for each metric.*

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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

Table 20: *Table results for Roboflow dataset and 75% reduction rate. The 'Precision', 'Recall' and 'mAP@.5' columns display 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 serve as the reference. Additionally, in gray, we highlight the best reduction method for each metric.*

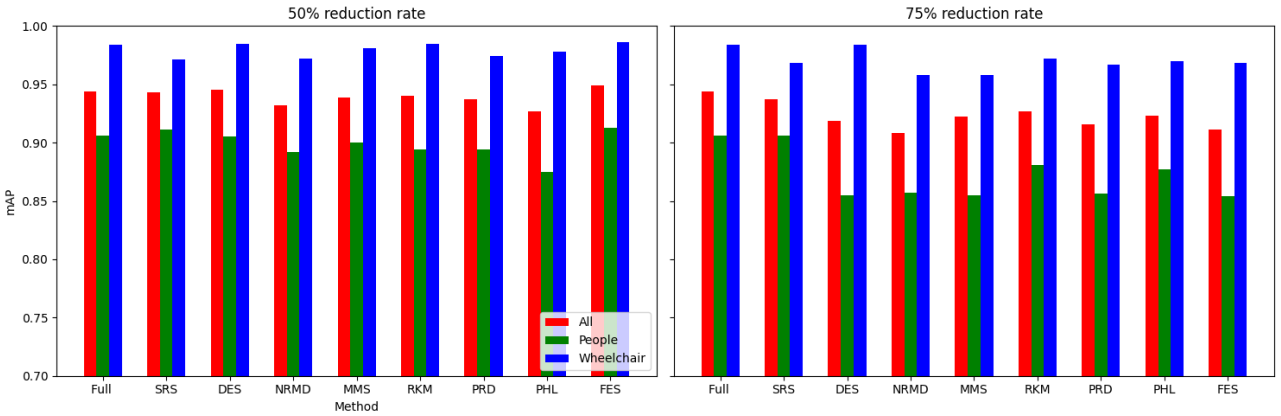

Figure 5: *mAP values on Roboflow dataset when using a 50% reduction rate (first column) and when using a 75 percent reduction rate (second column).*

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

Mobility Aid Dataset

Initially, we observe the training benchmark results with the complete dataset comprising 10,961 instances. This yields commendable outcomes, such as a mean average precision of 0.93, with a narrow standard deviation of 0.003 across all classes. Notably, the performance stands out in the category of people in wheelchairs (W), surpassing the overall average, while performance in other categories is noteworthy. Other categories were not included in Table 21 and Table 22. In general, superior performance is evident for the wheelchair, push-wheelchair, and walking-frame categories compared to the pedestrian and crutches categories, which exhibit below-average performance.

Moving to Table 21, which illustrates 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 an extended computation time during data reduction. Finally, across all methods, we managed to maintain the performance achieved with the complete training set. 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 exhibit a performance drop.

In Table 22, we present similar findings, 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 performance loss due to reduced metrics in other categories, the decline in the wheelchair category is comparatively less pronounced. Despite this reduction 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, demonstrate 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 provided in Figure 6. 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 performance loss 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 6), as it serves as the most comprehensive performance metric for object detection evaluation.

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 substantial reduction in both CO_2 emissions and computation time. Importantly, this reduction did not affect the model performance in object detection and localization.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

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 21: *Table results for Mobility Aid dataset and 75% reduction rate. The 'Precision', 'Recall' and 'mAP@.5' columns display 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 serve as the reference. Additionally, in gray, we highlight 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

Table 22: *Table results for Mobility Aid dataset and 90% reduction rate. The 'Precision', 'Recall' and 'mAP@.5' columns display 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 the reference. Additionally, in gray, we highlight the best reduction method for each metric.*

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

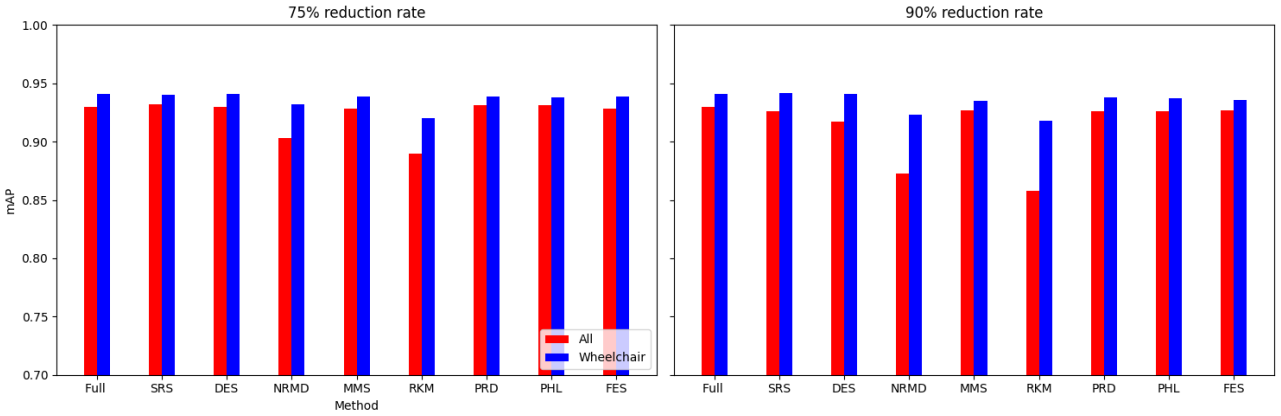

Figure 6: *mAP values on Mobility Aid dataset when using a 75% reduction rate(first column) and when using a 90% of reduction rate (second column).*

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

6. Key Findings and Contributions

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

6.1. Python Library: Data Reduction Methods

We have released the Beta version of a Python Library to use the data reduction methods. It is placed on an Open-access GitHub repository provided with instructions from the `README.md` file of the repository to use the data reduction methods.

Python Package	`https://github.com/Cimagroup/SurveyGreenAI` `https://doi.org/10.5281/zenodo.10844557`
Experiments	`https://github.com/Cimagroup/Experiments-SurveyGreenAI` `https://doi.org/10.5281/zenodo.10844476`

The reduction methods implemented are listed in Table 23. We have included those state-of-the-art methods for date reduction that satisfy the following conditions:

1. The implementation is available or their implementation is straightforward.
2. The final size of the reduced dataset can be chosen.
3. The reduction time offsets the training savings tested in small examples.

STATISTIC-BASED	SRS	Stratified Random Sampling	[58]
	PRD	ProtoDash	[16]
GEOMETRY-BASED	CLC	Clustering Centroids	[37]
	MMS	Maxmin Selection	[25]
	DES	Distance-Entropy Selection	[28]
RANKING-BASED	PHL	PH Landmarks	[54]
	NRMD	Numerosity Reduction by Matrix Decomposition	[11]
WRAPPER	FES	Forgetting Events Score	[56]

Table 23: *List of state-of-the-art data reduction methods selected for comparison.*

6.2. Data Reduction for Images

Some of the methods listed in Table 23 need their input to be n-dimensional vectors and cannot be directly applied to structured data such as images that have a tensor of shape (height, width, channels). In this deliverable, we propose two methodologies (see Section 5.3.2) to extend the reduction methods to images, in the context of this research project (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. Then, we

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

employ Global Average Pooling (GAP) [12] to transform the images into n-dimensional vectors (768 dimensions, due to the backbone's structure, which yields 768 feature maps). This allows us to apply data reduction techniques effectively. Finally, the images selected by the reduction method as the most important ones will be used to train the model.

2. For the wrapper method: We create a classification network to apply these techniques internally within the classification model. Subsequently, the images selected as the most important by the reduction method will be used to tune the YOLOv5 model with the reduced dataset.

6.3. Summary of the Main Results and Conclusions

The outcome of the investigation on the effectiveness of reduction methods to achieve green AI, with the aim of minimizing CO_2 emissions and computational costs while maintaining performance, yielded promising results. The CO2 emissions were estimated using a specific software implementation 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. However, notice there is no reduction method that always performs better than the rest. Furthermore, for one of the analyzed datasets, we have found a significal statistical correlation between 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 images: The proposed methodology is an effective way to achieve greener artificial intelligence models with good performance. In summary, regarding the Roboflow dataset, substantial 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 localization and object detection tasks. Among the reduction methods applied, SRS, MMS and RKM proved most effective, while NRMD exhibited poorer performance, and FES incurred longer processing times. Similarly, positive results were observed with 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 demonstrated superior effectiveness, while NRMD and RKM were identified as less efficient methods.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

References

[1] Dry Bean Dataset. UCI Machine Learning Repository, 2020. doi:10.24432/C50S4B.

[2] Forest Agostinelli, Matthew D. Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv: Neural and Evolutionary Computing*, 2014. doi:10.48550/arXiv.1412.6830.

[3] Md Manjurul Ahsan, MA Parvez Mahmud, Pritom Kumar Saha, Kishor Datta Gupta, and Zahed Siddique. Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9(3):52, 2021. doi:10.3390/technologies9030052.

[4] Elnaz Barshan, Ali Ghodsi, Zohreh Azimifar, and Mansoor Zolghadri Jahromi. Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds. *Pattern Recognition*, 44(7):1357–1371, 2011. doi:10.1016/j.patcog.2010.12.015.

[5] James C Bezdek and Ludmila I Kuncheva. Nearest prototype classifier designs: An experimental study. *International journal of Intelligent systems*, 16(12):1445–1473, 2001. doi:10.1002/int.1068.

[6] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *2010 20th international conference on pattern recognition*, pages 3121–3124. IEEE, 2010. doi:10.1109/ICPR.2010.764.

[7] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. URL: https://www.mdpi.com/2078-2489/11/2/125, doi:10.3390/info11020125.

[8] Sanjay Chawla and Aristides Gionis. k-means–: A unified approach to clustering and outlier detection. In *Proceedings of the 2013 SIAM international conference on data mining*, pages 189–197. SIAM, 2013. doi:10.1137/1.9781611972832.21.

[9] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *arXiv*, 2019. doi:10.48550/arXiv.1906.11829.

[10] CodeCarbon contributors. Codecarbon: A python library for carbon emission quantification. URL: https://codecarbon.io/.

[11] Benyamin Ghojogh and Mark Crowley. Instance ranking and numerosity reduction using matrix decomposition and subspace learning. In *Canadian Conference on Artificial Intelligence*, pages 160–172. Springer, 2019. doi:10.1007/978-3-030-18305-9_13.

[12] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *ArXiv*, abs/2009.07485, 2020. doi:10.48550/arXiv.2009.07485.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[13] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Handbook for Automatic Computation: Volume II: Linear Algebra*, pages 134–151. Springer, 1971. doi:10.1007/BF02163027.

[14] Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA, 2013. doi:10.1137/1.9781421407944.

[15] Rocio Gonzalez-Diaz, Miguel A. Gutiérrez-Naranjo, and Eduardo Paluzo-Hidalgo. Topology-based representative datasets to reduce neural network training resources. *Neural Computing and Applications*, 34(17):14397–14413, September 2022. doi:10.1007/s00521-022-07252-y.

[16] Karthik S Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient data representation by selecting prototypes with importance weights. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 260–269. IEEE, 2019. doi:http://dx.doi.org/10.1109/ICDM.2019.00036.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37, 06 2014. doi:10.1109/TPAMI.2015.2389824.

[18] Paul Henderson and Vittorio Ferrari. 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*, pages 198–213. Springer, 2017. doi:10.1007/978-3-319-54193-8_13.

[19] Abiodun M. Ikotun, Absalom E. Ezugwu, Laith Abualigah, Belal Abuhaija, and Jia Heming. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622:178–210, 2023. doi:10.1016/j.ins.2022.11.139.

[20] Mohaiminul Islam, Guorong Chen, and Shangzhu Jin. An overview of neural network. *American Journal of Neural Networks and Applications*, 5(1):7–11, 2019. doi:10.11648/j.ajnna.20190501.12.

[21] Tengjiao Jiang, Gunnstein Frøseth, and Anders Rønnquist. A robust bridge rivet identification method using deep learning and computer vision. *Engineering Structures*, 283:115809, 05 2023. doi:10.1016/j.engstruct.2023.115809.

[22] Glenn Jocher. Yolov5 by ultralytics, 2020. URL: https://github.com/ultralytics/yolov5, doi:10.5281/zenodo.3908559.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. doi:10.48550/arXiv.1412.6980.

[24] Murat Koklu and Ilker Ali Ozkan. Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, 174:105507, 2020. doi:10.1016/j.compag.2020.105507.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[25] C Lacombe, I Hammoud, J Messud, H Peng, T Lesieur, and P Jeunesse. Data-driven method for training data selection for deep learning. In *82nd EAGE Annual Conference & Exhibition*, volume 2021, pages 1–5. European Association of Geoscientists & Engineers, 2021. doi:10.3997/2214-4609.202112817.

[26] Daniel Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13, 2000.

[27] Hai-Guang Li, Gong-Qing Wu, Xue-Gang Hu, Jing Zhang, Lian Li, and Xindong Wu. K-means clustering with bagging and mapreduce. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–8. IEEE, 2011. doi:10.1109/HICSS.2011.265.

[28] Yang Li and Xuewei Chao. Distance-entropy: An effective indicator for selecting informative data. *Frontiers in Plant Science*, 12, 01 2022. doi:10.3389/fpls.2021.818895.

[29] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014. doi:10.48550/arXiv.1312.4400.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. doi:10.48550/arXiv.1405.0312.

[31] Huan Liu and Hiroshi Motoda. On issues of instance selection. *Data Mining and Knowledge Discovery*, 6(2):115, 2002. doi:http://dx.doi.org/10.1023/A:1014056429969.

[32] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018. doi:10.1109/CVPR.2018.00913.

[33] Kadan Lottick, Silvia Susai, Sorelle Friedler, and Jonathan Wilson. Energy usage reports: Environmental awareness as part of algorithmic accountability. In *NeurIPS 2019 Workshop on Tackling Climate Change with Machine Learning*, 2019. URL: https://www.climatechange.ai/papers/neurips2019/8.

[34] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696, 2009.

[35] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. *ArXiv*, abs/2304.07288, 2023. doi:10.48550/arXiv.2304.07288.

[36] Gunnar Myhre, Drew Shindell, and Julia Pongratz. Anthropogenic and natural radiative forcing. *Cambridge University Press*, 2014. doi:10.1017/CBO9781107415324.018.

[37] J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artificial Intelligence Review*, 34:133–143, 2010. doi:http://dx.doi.org/10.1007/s10462-010-9165-y.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[38] Juri Opitz and Sebastian Burst. Macro f1 and macro f1. *arXiv*, 2019. doi:10.48550/arXiv.1911.03347.

[39] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. doi:10.48550/arXiv.1511.08458.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. doi:10.48550/arXiv.1912.01703.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.48550/arXiv.1201.0490.

[42] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2015. doi:http://dx.doi.org/10.1109/CVPR.2016.91.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. doi:10.1109/TPAMI.2016.2577031.

[44] Seyed Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019. doi:10.1109/CVPR.2019.00075.

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016. doi:10.48550/arXiv.1609.04747.

[46] Jacob Salawetz. What is yolov5? a guide for beginners, 2020. URL: https://blog.roboflow.com/yolov5-improvements-and-evaluation/.

[47] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020. doi:10.1145/3381831.

[48] Deval Shah. Mean average precision (map) explained: Everything you need to know. *Retrieved November*, 4:2022, 2022. URL: https://www.v7labs.com/blog/mean-average-precision.

[49] Vinod Sharma. A study on data scaling methods for machine learning. *International Journal for Global Academic & Scientific Research*, 1(1):31–42, 2022. doi:10.55938/ijgasr.v1i1.4.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[50] Vin Silva and Gunnar Carlsson. Topological estimation using witness complexes. *Proc. Sympos. Point-Based Graphics*, 06 2004. doi:10.2312/SPBG/SPBG04/157-166.

[51] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009. doi:http://dx.doi.org/10.1016/j.ipm.2009.03.002.

[52] C Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101, 1904. doi:10.1093/ije/dyq191.

[53] Thomas F Stocker, Dahe Qin, G-K Plattner, Lisa V Alexander, Simon K Allen, Nathaniel L Bindoff, F-M Bréon, John A Church, Ulrich Cubasch, Seita Emori, 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*, pages 33–115. Cambridge University Press, 2013. doi:10.1017/CBO9781107415324.

[54] Bernadette J Stolz. Outlier-robust subsampling techniques for persistent homology. *Journal of Machine Learning Research*, 24, 2023. doi:10.48550/arXiv.2103.14743.

[55] Mihai Surdeanu and Marco Antonio Valenzuela-Escárcega. *Feed-Forward Neural Networks*, page 73–86. Cambridge University Press, 2024. doi:10.1017/9781009026222.006.

[56] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv*, 2018. doi:10.48550/arXiv.1812.05159.

[57] Andres Vasquez, Marina Kollmitz, Andreas Eitel, and Wolfram Burgard. Deep detection of people and their mobility aids for a hospital robot. In *Proc. of the IEEE Eur. Conf. on Mobile Robotics (ECMR)*, 2017. doi:10.1109/ECMR.2017.8098665.

[58] Roberto Verdecchia, Luís Cruz, June Sallou, Michelle Lin, James Wickenden, and Estelle Hotellier. Data-centric green ai an exploratory empirical study. In *2022 international conference on ICT for sustainability (ICT4S)*, pages 35–45. IEEE, 2022. doi:10.1109/ICT4S55073.2022.00015.

[59] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1571–1580, 2019. doi:10.1109/CVPRW50498.2020.00203.

[60] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020. doi:10.1007/s40745-020-00253-5.

[61] Petros Xanthopoulos, Panos M Pardalos, Theodore B Trafalis, Petros Xanthopoulos, Panos M Pardalos, and Theodore B Trafalis. Linear discriminant analysis. *Robust data mining*, pages 27–33, 2013. doi:10.1007/978-1-4419-9878-1_4.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**

[62] Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, Hao Zhou, and Lei Li. A survey on green deep learning. *arXiv*, 2021. doi:10.48550/arXiv.2111.05193.

[63] Rui Xu and Don Wunsch. *Clustering*. John Wiley & Sons, 2008.

[64] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *Digital Signal Processing*, 126:103514, 2022. URL: https://www.sciencedirect.com/science/article/pii/S1051200422001312, doi:10.1016/j.dsp.2022.103514.

[65] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. Data-centric Artificial Intelligence: A Survey, June 2023. arXiv:2303.10158 [cs]. URL: http://arxiv.org/abs/2303.10158, doi:10.48550/arXiv.2303.10158.

This project has received funding from the European Union's Horizon Europe research and innovation program under grant agreement **No 101070028-2**