KU LEUVEN

Annotation-Efficient Machine Learning for Marine Litter Detection

Giorgia Milli

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

Supervisor:
Prof. Matthew Blaschko

Assessors:
Prof. Renaud Detry
Robrecht Moelans

Assistant-supervisors:
Mehrdad Moshtaghi
Robrecht Moelans

Academic year 2021 – 2022

Preface

This work has been to me a brand new start in a field very far from my comfort-zone. I am grateful I had the chance to give my small contribution to a much broader cause and to work in a context which is in line with my personal beliefs. Thus, I would like to write few lines to thank everybody who supported this work and my progress in this journey.

First, I would like to thank my supervisor Prof. Matthew Blaschko and my daily advisors Robrecht Moelans and Mehrdad Moshtaghi from VITO for choosing me to carry out this study and for their constant and valuable support along the way.

I must also thank Xenia Ivashkovych for the insightful suggestions and for the guidance in my understanding of recent Semi-Supervised and Self-Supervised frameworks.

Thanks to Gianluca, for the valuable external observations concerning this work and for helping me out with improving my programming skills during happy-hours.

My deep gratitude goes also to Gabriele, who has always supported my ambitions and my work, both personally and intellectually, with constant guidance and patience.

Finally, I want to express a warm thank to my family who, even in the distance, has always encouraged both my life and professional choices.

As symbol of this journey, I would like to dedicate this Thesis to Delfina, whose light and presence never abandon me.

Giorgia Milli

Abstract

Marine litter pollution is one of the most severe consequences of the human impact on our planet. Transported by water, waste residuals of different nature reach the most remote corners of Earth and threaten the safety of the marine ecosystem.

To limit the damage and develop concrete solutions for cleaning the marine environment, many Computer Vision and Machine Learning-based research studies have been proposed to detect and track marine litter at the harbor, river, and coastal areas. However, the lack of annotations needed to train and validate Machine Learning algorithms represents a major burden for researchers. Labeling is a time-consuming yet crucial procedure to obtain good and robust models that could be exploited in real-world scenarios.

The study carried out in this Thesis aims at providing possible solutions to deal with the lack of annotations in the context of marine litter detection.

To this end, 24 UAV images acquired at 50m above the Vietnamese coast were split into 7056 patches, out of which 2070 were annotated as "litter"/"no litter". The experiments carried out on this dataset involved both Machine Learning and Deep Learning algorithms trained in Supervised, Semi-Supervised and Self-Supervised learning settings. The experimental results were compared to the accuracy (88%) and "litter" class Recall (82%) scored with a Random Forest taken as reference. The tests involved a Supervised training for a CNN and a pre-trained MobileNetV2, both alone and in combination with a Random Forest. The combined options were also tested in Semi-Supervised mode. Finally, a novel Self-Supervised framework called DINO was tested in combination with a k-NN and a Linear classifier. For all the options involving the CNN, the results obtained were very close to the baseline both in terms of accuracy (from -2% to +4%) and of "litter" Recall (from -3% to +0%). The solutions involving the MobileNetV2 were less robust and even, and seemed to suffer more from class unbalance. The performance related to DINO in combination with a k-NN perfectly matched the accuracy of the baseline and scored only a -1% in "litter" Recall.

The results highlighted the possibility of obtaining feature extractors as good as those of Supervised methods, even with a minimal amount or no labeled data provided. This suggests that less annotation-eager learning approaches could effectively be exploited in the marine litter detection context. Given the potential of these methodologies in classification tasks, the next step for this study should focus on testing label-free techniques for semantic segmentation. To this end, a first qualitative proof-of-concept is also provided in this Thesis.

Contents

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations

Abbreviations

Adam	Adaptive momentum estimation
BGD	Batch Gradient Descent
BN	Batch Normalization
CE	Cross-Entropy
CNN	Convolutional Neural Network
CRF	Connected Random Field
DINO	Knowledge DIstillation with NO labels
DL	Deep Learning
DNN	Deep Neural Network
DT	Decision Tree
FCN	Fully Connected Network
GD	Gradient Descent
GUI	Graphic User Interface
k-NN	k-Nearest Neighbors
MBGD	Mini-Batch Gradient Descent
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Square error
MSP	Multi-Layer Self-Attention
NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RF	Random Forest
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SSL	Self-Supervised
STEGO	Self-Supervised Transformer with Energy-Based Graph Optimization
UAV	Unmanned Aerial Vehicle
ViT	Vision Transformer

Chapter 1

Introduction

Marine litter consists of solid materials of different nature (e.g., plastic, metals, textiles, glass, paper, wood), which are intentionally discarded or unwillingly lost in coastal, river and harbor areas. Transported by water, marine litter ends up polluting the marine ecosystem even in the most remote corners of the Earth [23]. This issue is one of the most severe consequences of the human impact on our planet.

The alarming increase of litter concentrations in the marine environment prompted many research studies to focus their efforts on developing Computer Vision algorithms able to detect and track those polluting residuals. As stated in [4] *"Being able to detect larger floating plastics in coastal waters before they become entangled, ingested, exported and/or fragmented, may help to answer key questions about sources, pathways and trends."*. The aim is to prevent the problem to spread any further and to develop concrete solutions to remove as much as possible litter items from the marine ecosystem. However, it is still largely unclear which would be a proper way to tackle the problem. Among the challenges researches have to deal with, detecting the amount of litter entering the ocean and the accumulated volume in the marine environment, mapping the source location and gathering data related to floating debris transport are among the greatest ones [23].

To this end, a useful source of information are images collected via remote sensing satellites. Those systems are in fact designed to provide observation of global scope with continuous temporal coverage [23] and can potentially be used for global marine litter monitoring. However, the detection of debris from space is greatly compromised by atmospheric and sea-surface effects, spatial and temporal resolutions of the satellite and the lack of ground-truth data.

An alternative solution, which avoids some of the challenges related to images collected from space, are data acquired with Unmanned Aerial Vehicles (UAV). Drones can cover relatively large areas and provide information at high geo-spatial resolution. In the last years, several attempts of using Machine Learning and Deep Learning methods for automated detection and quantification of marine litter have been tested on UAV images [1], [16], [2], [14], [9]. Deep Learning-based approaches have also been proposed to segment and classify litter objects according to their materials as in the study carried out in [25].

However, despite collecting UAV images is relatively easy, the lack of annotations needed to train and validate Machine Learning and Deep Learning algorithms represents a major burden for the research. Labeling is well known to be the most tedious and time consuming procedure when dealing with new datasets. Yet, this step is crucial to obtain good and robust results and models that could be exploited in real-world scenarios.

On the other hand, the advent of learning approaches such as Semi-Supervised, Self-Supervised and new architectures such as Vision Transformers [8] shed the light on the possibility of (partially) renouncing to the need of large amounts of labeled data for achieving good performance. Exploiting these new options and working with few or no annotations would be a great step forward for research progress in the context of marine litter detection on UAV images.

To this end, the goal of this Thesis is providing preliminary results in this direction by comparing the performance obtained with both Machine Learning and Deep Learning algorithms in Supervised, Semi-Supervised and Self-Supervised learning modalities when few annotations are available for a large dataset. More specifically, the study aims at performing a patch level binary classification ("litter"/"no litter") by means of different models and training modalities. The first experiments involve a simple Random Forest taken as baseline, a CNN, a pre-trained MobileNetV2 [20] architecture. After testing those models separately, a combination of a Random Forest with both CNN and MobileNetV2 is tested in both Supervised and Semi-Supervised settings. A third experiment covers a more advanced approach by means a label-free framework called DINO [6]. Last, a new promising option for label-free litter segmentation will be briefly touched upon in the Appendix section.

The source code developed in this Thesis is available at `https://github.com/giollimirgia/litter_thesis.git`

1.1 Main Challenges

Beside the lack of labels, other challenges that might have an impact on the achievement of good results are listed below.

Distance from the ground

The height at which the drone collects the images determines the amount of details that can be captured and exploited to discern litter from any other object or surrounding environment. Therefore, the higher the flight the harder it becomes to confidently spot litter items/areas. The following points are strongly related to this challenge.

Class unbalance

Despite the great amount of litter items found in the coastal areas, the more distant from the ground the drone is the more the surrounding environment and other elements (e.g., cars, boats, trees, stones) will be captured in the data. Although

covering larger areas is beneficial for detecting and estimating the size of wide cluttered litter patches, the latter would represent a rather small portion of the whole picture. Consequently, the amount of "no litter" examples will be considerably greater than the one of "litter" examples. This class unbalance might lead the models to strongly favor the largest of the two classes.

Litter pattern

With "litter" we refer to a multitude of items that differ in size, shape, material and color. Being this category extremely sparse, finding robust and representative features related to it is a rather tough task. Moreover, the farther from the ground the images are collected, the more the search of optimal attributes becomes challenging. This is both because sometimes it is non-trivial to tell whether a patch contains litter or, for instance, just stones, and because the "litter" examples will contain additional unrelated information such as different backgrounds (e.g. water, sand, stones, vegetation).

Quality of annotations

This Thesis focuses on the detection of large cluttered litter areas, discarding small scattered pieces of residual materials. For the annotation procedure, this choice might represent an additional challenge and produce labels that could "confuse" the learning process. In fact, assuming that a model learns a proper feature representation for the "litter" category, it may be misleading providing it with patches containing waste items labeled as "no litter". Furthermore, the annotating process based on this concept may lead to poor quality labels, since the evaluation of how much litter a patch contains is qualitative rather than quantitative and, therefore, based on the annotator's perception.

1.2 Thesis Outline

This Thesis presents different learning methodologies and algorithms that have been tested in the attempt to deal with lack of annotations in the context of marine litter detection.

Chapter 2 will cover the Technological Background needed to understand the methodology and the experiments carried out in this study. First, some notions concerning different learning settings, from Supervised to Self-Supervised to Transfer-Learning and Fine-Tuning, will be given. Second, the traditional Machine Learning algorithms used in this Thesis will be briefly introduced, whereas a stronger focus will be addressed on covering the Deep Learning approaches. Finally, an overview of more recent topics and technologies, such as Vision Transformers and Knowledge Distillation, will be provided to introduce a novel and promising Self-Supervised framework called DINO [6].

The methodology at the base of this study will be introduced in Chapter 3. After describing the annotation procedure, the process of moving from a Supervised

learning towards a label-free approach will be discussed by providing information concerning the model architectures and learning settings chosen for the different strategies tested. Results will be presented and discussed in Chapter 4 after a detailed introduction about the dataset used in this project. Chapter 5 will cover Conclusions and possible future works.

Finally, an Appendix section will briefly introduce a preliminary work on a label-free semantic segmentation approach as a possible continuation of the study carried out in this Thesis.

Chapter 2

Technological Background

This chapter provides background information and some technical details on the techniques, dataset structures and learning approaches that are useful for a proper understanding of this Thesis.

2.1 Annotations and Learning Approaches

When dealing with images, the prediction task might have two different natures: **classification** or **semantic segmentation**. In the first case, the annotations consist on the true classes each input belongs to, whereas in the second case the annotations consist in pixel-level labeled masks of same size as the input. The creation of the ground truth, taken care by some human annotator, is often the most time consuming and demanding task in data preparation. Depending on the amount of available annotations, different learning approaches can be chosen:

- In **Supervised** settings, each input must be provided with its annotation. This approach generally works very well when large amounts of labels are available. In fact, Machine Learning (ML) and Deep Learning (DL) models are extremely eager for data, but annotating a large amount of examples is not always feasible. Moreover, the labeling procedure is likely to be biased and error-prone and the consequent label quality heavily impacts the performance of the model.

- The **Unsupervised** approaches do not make use of any annotation to perform their task. In this case, the model learns patterns starting from the data and outputs a more compact and clustered representation of the inputs. This method can be coupled with classifier trained in Supervised mode in order to assign labels to the obtained clusters.

- **Semi-Supervised** learning is halfway between the Supervised and Unsupervised worlds. This approach is suitable when having few labeled examples and many unlabeled ones. In this context, two different learning strategies can be pursued: **Inductive** and **Transductive**. In the first case, the model is trained in Supervised mode on few labeled data and used to generalize to unseen data,

whereas in the second case the model observes all the data and uses the few labeled examples to label the unlabeled ones. The latter procedure is called **Label Propagation**.

- **Self-Supervised** learning (SSL) attempts to approximate a form of realistic common sense that could reflect the background knowledge used by humans to annotate the data. This procedure is a form of Unsupervised learning where the data themselves provide the supervision. The main idea here is to hide a portion of the data and try predicting it making use of the remaining portion. This approach can reveal some hidden relationships and underlying structures of the data in an unbiased fashion. To this end, the model is fed with unlabelled data and forced to learn unbiased meaningful representations. Then, the pre-trained model is usually Fine-Tuned according to the specific task.

In case the dataset size or the computational resources are limited, a very useful approach to deal with such situations is represented by **Transfer Learning**. This practice consists of taking advantage of a model trained on larger data sets (such as ImageNet [7]) and with powerful computational resources. More specifically, the features of this model are leveraged on a new, similar task. The idea behind Transfer Learning is related to the observation that low level features often share similar properties that are independent of the task. On the other hand, when moving to higher levels, the model representations become more and more goal-specific. This means that the capabilities of a powerful model that learned very rich feature representations can be exploited in different tasks. This is usually achieved by replacing the top prediction layer with some new ones more suitable for the new goal. First, the structure is trained by freezing the pre-trained model weights to only train the new top layers. Then, a **Fine-Tuning** step is usually carried out to refine the whole model weights specifically for the new task. To do so, all the layers are trained at a very low learning rate for a limited number of epochs.

2.2 Datasets

Before training a Neural Network, the dataset is usually split into three subsets meant for different purposes and exploited at different phases:

- **Training set:** this set is used for the learning task. It is common for this partition to contain most of the entire dataset examples. The size of the training set should be adequate for the learning task: the more complex the task is, the more data will be needed to achieve good learning performance.

- **Validation set:** this dataset is used to simulate a preliminary test of the model during the learning phase. The performance associated with the validation set should give an idea of what to expect at testing time. Therefore, it is important that the distribution of this dataset matches the one of the test set, i.e., it

should be as representative as possible of the target task. The validation set is much smaller than the training set.

- **Test set:** this dataset is used once the model has been fully trained to assess its ability to generalize and so perform well on unseen data. The size of the test set is usually comparable to the one of the validation set.

2.3 Traditional Machine Learning algorithms

Despite the advent of DL, traditional ML algorithms still represent a good way to gather some basic insights about the data and solve simple tasks. Moreover, these algorithms are often combined with more advanced approaches from the DL family and, for their simplicity, they can be chosen as baseline methods to which refer any further development and improvement. In the following paragraphs, the three ML algorithms used in this Thesis will be presented: Logistic Regression, k-Nearest Neighbors and Random Forest.

2.3.1 Logistic Regression

Logistic Regression translates the output of a Linear Regression model into probabilities. A Linear Regression model learns a linear mapping between input and output variables. Thus, the goal of this algorithm is to fit a linear function between data points. At training time, the Mean Squared Error (MSE) between the predicted values and the targets is computed to adjust the model parameters. This approach is very simple and involves a limited number of parameters. The Logistic Regression model can indeed be implemented through a simple Perceptron.

2.3.2 *k*-Nearest Neighbors

k-Nearest Neighbors (k-NN) is a proximity-based classifier. Given a query point, the goal of k-NN is to assign it to the closest neighbor and, consequently, label it with the same class the chosen neighbor belongs to. The most used distance-based similarity metric is the Euclidean distance, but others, such as Manhattan distance or Hamming distance, can be chosen to find the closest neighbor. The k-NN algorithm requires the user to provide the k number of neighbors to check. This selection is generally non-trivial, as lower values might lead to over-fitting (low bias but high variance) whereas higher values to under-fitting (low variance but large bias).

2.3.3 Random Forest

Random Forest consists of an ensemble of Decision Trees (DTs) whose predictions are combined together with strategies such as majority voting to generate a final and unique result. At each node of a DT, the input vector is split according to a particular feature; the splitting continues until a leaf is reached and the final prediction occurs. To select the best feature to use for partitioning the data, the quality of the split is evaluated at each node via metrics such as Gini impurity, Information Gain or MSE.

The idea behind Random Forest and, more generally, ensemble models is to combine the effort of multiple weak learners that could focus on specific aspects of the input vector and learn different representations. This approach makes the Random Forest particularly robust to over-fitting compared to a single DT and typically leads to better performance. At training time, the Random Forest algorithm selects a random subset of features in order to ensure low correlation among the DTs. Therefore, the evaluators are trained with a subset of features instead of the whole input vector. A Random Forest requires the user to set three hyper-parameters prior to training: node size, number of DTs and number of features to sample.

2.4 Introduction to Deep Neural Networks

Deep Neural Networks (DNN) are Neural Networks carachterized by multiple hidden layers able to capture from finer to higher level features. Each layer of a DNN produces an input-output mapping which, differently from traditional Fully Connected Networks (FCN), allows to learn very complex functions thanks to a greater achievable depth. Moreover, DNNs are able to capture deeper details and to self-learn the most representative features with no need for the user to pre-define them. The following section will be dedicated to a specific type of DNNs called Convolutional Neural Networks (CNN) that are particularly meant for images and that were used in this Thesis.

2.5 Convolutional Neural Networks

The traditional FCNs are not suitable to handle complex inputs such as images. A FCN would interpret each pixel as a single feature for which weights must be learned. Even for small image sizes, this approach would involve an enormous amount of parameters. Such a condition makes the training prohibitive even when dealing with small architectures consisting of a couple of layers. Furthermore, FCNs are not able ot capture the image context, because neighbouring pixels that add important information for the interpretation of the input are not taken into account.

Conversely, CNNs are well suited for dealing with complex inputs such as images. Those architectures belong to the DL algorithm family and overcome the limitations of FCN, particularly thanks to the additional usage of Convolutional and Pooling layers. The main layers composing a CNN are explained in detail in the following sections.

Convolutional layer

Consists of a kernel which is translated over the image and acts as a filter. At each position, a convolution operation is performed between the image pixels and the kernel weights to output the feature map. The 2D convolution of an input image I and a kernel K of size $m \times n$ can be defined as a cross-correlation operation 2.1:

$$S(i,j) = \sum_m \sum_n I(m,n)K(i+m, j+n) \tag{2.1}$$

This operation reduces the size of the input image according to 2.2:

$$\frac{I_{rows} - K_{rows} + 2 \times padding}{stride} + 1 \tag{2.2}$$

padding refers to the operation used to fill the image borders in order to prevent the output dimensions from becoming too small, and *stride* defines how many pixels the kernel should move to reach the next position. The third dimension is determined by the number of filters being used in the convolutional layer.

Since the kernel is smaller than the image, its units interact only with a limited region of the entire input. This characteristic is defined **sparsity of connections**, which makes the output of the convolution operation only dependent on a small portion of the input. Furthermore, since the kernel slides all over the image, its weights are re-used in all input regions. This condition, known as **parameter sharing**, greatly reduces the number of network parameters and allows training much deeper and more complex architectures.

The Convolutional layer confers a last important property to CNNs, which is the ability of catching shapes and taking into account the surrounding context. In fact, during the learning process, the kernel weights specialize in particular shapes and characteristics of the image and act as feature detectors. The first layers focus on low level features, such as edges and lines, whereas deeper layers catch more complex and high level features, such as shapes.

Pooling layer

A Pooling layer is a non trainable layer (i.e., does not involve parameters) usually placed in cascade to a Convoloutional layer to reduce its output feature map dimensions. This operation is useful to allow a GPU with limited memory storing deeper architectures, but also to increase the coverage on which the convolution kernel acts. Moreover, the presence of this layer speeds up the training. The operations performed in a Pooling layer are either linear, such as average, or statistic, such as maximum extraction. After a pooling operation, only the most meaningful features are retained. As in the Convolutional layer, a kernel is slid over the feature map. The output dimensions will be reduced according to 2.3 at preserved number of channels:

$$\frac{I_{rows} - K_{rows}}{stride} + 1 \tag{2.3}$$

Fully Connected layer

This layer, also called Dense layer, consists of units all connected to the previous layer neurons, as in a traditional FCN. the Fully Connected layers layers are usually placed towards the end of the CNN architecture to carry out the classification task. The 2D feature map is flatten to form a 1D feature vector before being passed as input to the Fully Connected layer.

Activation Function layer

The Activation Function layer is characterized by a non-linear function (i.e., the activation function) which is applied to the output of the precedent linear layer (i.e., Convolutional or Fully Connected). There exist several kind of activation functions, and generally the choice depends on the task and on the depth of the network. *Sigmoid* and *Hyperbolic tangent* functions, for instance, are bounded in the range of [0, 1] and [-1, 1] respectively and are characterized by plateau regions towards their extremes. This means that in those locations the resulting gradient is so small that might vanish across the backward propagation, leading to small or no updates of the weights for deep architectures. To prevent this problem, a more suitable activation function for deep networks is the *Rectified Linear Unit* (ReLU) [17] which only bounds negative values to 0 but does not have an upper bound, so that plateau regions leading to small gradients are avoided. For multi-classification tasks, a *Softmax* function is generally used in the last layer to output probabilities of belonging to each class. This function is defined as follows:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{2.4}$$

By setting a probability threshold, each input can be ultimately assigned to a specific class.

2.5.1 Loss Functions

In the context of an optimization algorithm, an objective function is used to evaluate a candidate solution (i.e., a set of parameters). The goal can be either maximizing or minimizing such an objective function. In the first case, we would like to find the candidate solution that brings to the highest score; in the second case, we seek the candidate solution that brings to the lowest score. When dealing with Neural Networks, we aim at minimizing the error. Therefore, the objective function is often referred to as a cost (or loss) function, and the value obtained from the latter is called "loss".

Maximum Likelihood and Cross-Entropy

When using **maximum likelihood**, Cross-Entropy is used to measure the error between two probability distributions. When the learning task consists of mapping the inputs to class labels, the problem can be modeled as predicting the probability of an example to belong to a class. In case of binary classification, the prediction concerns the probability of an example to belong to one class, whereas in a multi-class context each example will be provided with a probability of belonging to each of the classes. Therefore, during the training phase, maximum likelihood estimation allows to find the optimal parameters able to minimize the difference between the probability distribution predicted by the model and the given true class probability distribution. This is called **Cross-Entropy** (CE).

The maximum likelihood approach is the preferred choice in the Neural Network context, because classification models that use a *Sigmoid* or a *Softmax* activation function in the output layer tend to train more robustly and faster when combined with the Cross-Entropy loss function. The latter, also known as Softmax loss, is defined in equation 2.5:

$$CE = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot p_{model}(y_i|x_i, \theta) \tag{2.5}$$

where p_{model} is the probability distribution estimated by the model, x_i is the i^{th} example drown from the true distribution, θ is the network parameters distribution, and y_i the label x_i example is associated with.

2.5.2 Regularization

Over-fitting is one of the major risks when training a ML model. This issue occurs when the network memorizes the training data and, consequently, compromises its ability to generalize on unseen data. As a result, the model performs very well during the training phase, but considerably worse in test phase, proving itself unreliable to work on real-case scenarios. Regularization techniques are meant to prevent the risk of over-fitting and preserve the generalization capability of the model. The regularization approaches used in this Thesis are presented below.

Data augmentation

Is a strategy that allows generating new training examples by applying some user-defined transformations (e.g., horizontal/vertical translations, flips, rotation, zoom in/out, contrast change, etc.) to the available training data. The augmented examples will be associated with the same label as the examples they were generated from. Thus, it is fundamental to choose transformations that are realistic and that do not compromise the semantic of the example. This approach allows to include some variations to the dataset and to prevent the network from focusing too much on specific and unusual details.

Dropout layer

This layer is often placed in cascade to Convolutional and Fully Connected layers. Dropout layers are characterized by a user-defined probability representing the proportion of units of the functional layer that should be turned off during a training iteration. A dropout probability of 0.5, for instance, will randomly turn off half of the neurons of the functional layer placed before the Dropout layer. Those units will not contribute to either the forward nor the backward pass of the current iteration. The in-going/out-going weights of the units not being considered will be preserved and reused once the neurons will be turned on again in the following iteration.

The Dropout layer is only active during the training phase to prevent over-fitting and reduce the variance of the network. At testing time, all the units will be turned

on and contribute to the achievement of the goal. This strategy is equivalent to training multiple models that focus on different characteristics of the data. As happens with ensemble models, combining the contribution of several classifiers generally improves the performance compared to that of a single classifier.

Batch Normalization layer

This layer is usually placed between a linear layer and an activation layer. If present, each dimension of the linear layer output is standardized by subtracting the mean and dividing by the standard deviation of the mini-batch currently being processed. Then, the normalized features are re-scaled and re-translated according to a scale and a shift parameters which are also learned during the training.

The Batch Normalization (BN) layer guarantees that cases where none or all responses manage to surpass the activation threshold are avoided. This condition makes the learning more stable.

2.5.3 Optimization

In the context of Neural Networks, the optimization procedure consists of minimizing the objective function, which determines the difference between the true label and the prediction. The negative gradient of the loss error is used for updating the weights during the learning process. Such a procedure allows to descend the cost slope and to move closer and closer to the minimum after each parameter update, until convergence is reached. The methods that base their learning on this procedure belong to the gradient descent algorithm family and are presented below.

Gradient Descent

Gradient Descent (GD) is the baseline technique used for learning in Neural Networks. Once the loss is computed after the forward pass, its gradient is calculated and the parameters of the network are updated following its opposite direction according to the rules 2.6 and 2.7:

$$w_i^{k+1} = w^k - \eta \frac{\partial J(w,b)}{\partial w_i} \tag{2.6}$$

$$b_i^{k+1} = b^k - \eta \frac{\partial J(w,b)}{\partial b_i} \tag{2.7}$$

where w_i^{k+1} and b_i^{k+1} are the i^{th} weight and bias at iteration $k+1$, and η is the learning rate, which defines the size of the step to take in descending the cost slope $J(w,b)$. There are three main variants of the GD that differ in the size of the data set being used in the update:

- **Batch GD (BGD):** The parameter update occurs after the whole training set has been exploited. This approach is rarely used, because using all the training examples for computing the gradient is computationally expensive and memory heavy. Therefore, it is unfeasible to proceed with this strategy for large training sets.

- **Stochastic GD (SGD):** The parameter update occurs after exploiting only one example of the training set. Despite this strategy avoids memory issues and allows training on huge datasets, it causes large fluctuations of the output loss and, consequently, promotes large updates that make the learning very unstable.

- **Mini Batch GD (MBGD):** This solution is halfway between the BGD and SGD variants. In this case, the parameter update occurs after a mini-batch (i.e., a subset of the training set) has been passed through the network. MBGD avoids memory issues are without compromising the stability of the learning process.

Adaptive Methods

Despite MBGD overcomes some limitations, it does not handle other hyper-parameter related issues. In particular, the selection of a suitable learning rate η is non-trivial and problem-dependent. The learning process would greatly benefit from an automatic adjustment of η across the training phase. To this end, more advanced techniques, such as the Momentum [19], the Nesterov accelerated gradient [18], the Adaptive Gradient (Adagrad) [3], the Root Mean Squared Propagation (RMSProp) [22] and the Adaptive momentum estimation (Adam) [12], have been proposed to properly handle the learning rate.

In this Thesis, Adam was selected as optimization function in all the experiments carried out. This optimization strategy stores exponentially decaying averages of past squared gradients and gradients and computes adaptive learning rates for each parameter.

2.5.4 Training with *k*-fold cross-validation

This technique consists of randomly partitioning the training set into k subgroups and repeating the training procedure k times. Each time a different partition is used as validation set, whereas the remaining $k-1$ partitions are used as training set. This means that a total of k models will be learned, each one based on a different validation set. At the end of this procedure, the predictions obtained from each model will be combined together to get a single output. Typical approaches consist of averaging the outputs or using a majority voting strategy. At the end, probabilities are thresholded to get the final classification. k-fold cross-validation is particularly useful to achieve a better generalization, so that a good performance can be obtained also on unseen data.

2.6 Knowledge DIstillation with NO labels (DINO)

Knowledge **DI**stillation with **NO** labels **(DINO)** [6] is a Vision Transformers-based framework trained in SSL settings. Before diving into DINO structure and

functioning, some more details about the main actors of this framework are provided in the following paragraphs.

2.6.1 Vision Transformers

Transformers were first implemented for Natural Language Processing (NLP) applications [24]. The important advantages Transformers brought in the context of NLP prompted the scientific community to adapt them to computer vision tasks.

Vision Transformers (ViTs) [8] partition the input image into patches called "visual tokens", embedded into a set of encoded vectors of fixed dimension that also keep track of the patch positions. Such embeddings are fed into a Transformer encoder where a Multi-Head Attention network assigns them weights according to their importance, in order to also include some context. The Multi-Head Attention network outputs attention maps indicating to the model which regions are the most relevant for the prediction. Last, a Multi-Layer Perceptron (MLP) head is placed at the end of the encoder to output logits that can be converted to probabilities via activation layers. Transformers are usually pre-trained on large unlabelled data sets and only Fine-Tuned afterwards in a Supervised fashion to refine the learned data representations [11].

Self-Attention mechanism is one of the key ideas behind the ViTs that grants several advantages compared to traditional CNNs. Thanks to Self-Attention, ViTs are able to learn long-range relationships, as well as local and global features. The model filters are dynamically calculated according to the input and, therefore, they are not static as for convolutions. Moreover, [11] reports that *"self-attention is invariant to permutations and changes in the number of input points"* . Some more details on the functioning of Self-Attention mechanism are provided in the following paragraph. Other advantages of ViTs compared to CNNs include their scalability to high-complexity models and large-scale data sets.

Self-Attention mechanism in detail

The Self-Attention layer takes into account the information of the entire input sequence to update each sequence component. A sequence of n elements $(x_1, x_2, \ldots, x_n)$ can be indicated by $X \in \mathbb{R}^{n \times d}$, with d being the embedding dimension for representing each entity. Self-Attention tries to encode each of the n entities according to global contextual information in order to capture any existing underlying interaction among the data. Such a goal is achieved by means of three weight matrices called Queries $W^Q \in \mathbb{R}^{d \times d_q}$, Keys $W^K s \in \mathbb{R}^{d \times d_k}$ and Values $W^V \in \mathbb{R}^{d \times d_v}$, where $d_q = d_k$. These matrices are used to project the input sequence X in order to obtain $Q = X \cdot W^Q$, $K = X \cdot W^K$ and $V = X \cdot W^V$. The final output $Z \in \mathbb{R}^{n \times d_v}$ of the Self-Attention layer is then obtained by applying the *Softmax* to such projections [11]:

$$Z = Softmax(\frac{Q \cdot K^T}{\sqrt{d_q}}) \cdot V \tag{2.8}$$

This procedure assigns attention scores to all the entities of the sequence X. In order to embed more complex relationships among the entities of a sequence, it is possible to include more Self-Attention blocks. Such a structure is referred to as **Multi-Head Attention**. Each block will be characterized by its own learnable weight matrices and will output a different output Z. All these outputs will then be concatenated into a single matrix and projected onto a weight matrix to get a final unique result.

2.6.2 Knowledge Distillation

Knowledge Distillation refers to the process of transferring knowledge gathered with a large model to a smaller one. This approach is helpful because models characterized by a larger knowledge capacity often cannot be used in practice due to their size and computational costs. Knowledge Distillation follows a Teacher-Student structure: the Student is represented by the small network that tries to mimic the Teacher, which is the larger and pre-trained model. The goal is to transfer the knowledge from the Teacher to the Student while maintaining, or even improving, the accuracy.

2.6.3 Teacher and Student networks

In DINO the Teacher and the Student networks have exactly the same structure. The model consists of a ViT backbone followed by a 3-layer MLP as projection head. The only difference between the two models is that the Teacher is also provided with a **centering** step which prevents one dimension to dominate over the others. Moreover, since centering promotes the network to collapse to a uniform distribution, an additional **sharpening** strategy is included to contrast this effect. Sharpening is obtained by modulating a parameter called *temperature* included in the *Softmax* computation.

2.6.4 DINO approach

In the usual Knowledge Distillation paradigm the Student is trained to match the Teacher's output. Given an input image x, the two networks output probability distributions of fixed dimensions thanks to a *Softmax* function. In particular, the input image is first cropped with a multi-crop strategy that generates 2 *global views* (i.e., crops covering an area $> 50\%$ of the image) and several *local views* (i.e., crops covering an area $< 50\%$ of the image). The Student network is fed with all the crops, whereas the Teacher is only provided with the global views. This strategy encourages "local-to-global" correspondences.

The choice of having a *Softmax* layer despite no labels are provided for classification is meant to force the networks to come up with a "classification guess" by themselves. Therefore, instead of providing a classification task, the models are asked to come up with one. In this context, the *Softmax* layer is used to obtain a distribution of the input which has no semantic meaning.

For the Student network the output distribution will be more spread across the output "classes", whereas for the Teacher it will be more peaked and localized within

a smaller range. The loss function will compute the similarity via Cross-Entropy loss between the Student and the Teacher's outputs, which will be used to update the network parameters. In particular, a stop gradient operator applied to the Teacher will prevent this network to immediately update its parameters and to fully address the gradient propagation to the Student. Only after the Student's parameters will be updated an exponential moving average will be applied to them and used to update the Teacher's weights.

As opposed to traditional approaches, where it is usual to have a pre-trained Teacher network from which the Student learns, here the Teacher is dynamically built during training. This means that Knowledge Distillation is expressed as a SSL objective instead of being used as a post-processing step to SSL pre-training. It was observed that with this structure it is still guaranteed that the Teacher has a better performance than the Student during training and, consequently, it is able to guide the learning by providing features of higher quality [6]. Figure 2.1 was taken from the original paper and shows how DINO Student-Teacher training is carried out.

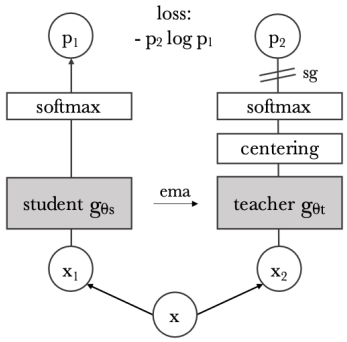

Figure 2.1: DINO Student-Teacher structure. The figure illustrates the example of a single pair of views (x_1, x_2). Two different random transformations of the input image are passed to the Teacher and Student networks. The latter have the same architecture and only differ in a centering step, which is only performed in the Teacher network. Both models output a K-dimensional feature vector normalized via Softmax computation. The similarity between the outputs is obtained via cross-entropy loss. A stop gradient (sg) operator allows the gradients propagating only through the Student, whereas the Teacher's parameters are updated subsequently with an exponential moving average (ema) of the Student's parameters [6].

2.6.5 Notes on DINO

Despite no labels are provided to DINO, the framework is able to produce accurate attention maps. This can be explained by the fact that the background knowledge is actually inferred through augmentations that play a central role. By means of augmentations it is possible to indirectly tell the model what is important or not and where to focus on. For instance, if a brightness augmentation is applied, we are telling the model such a change in the input does not affect the classification. As a consequence, the model will learn not to focus on brightness changes. It is important to underline that, for DINO to work properly, the dataset construction is another

key point. If the framework is provided with images representing some main subjects, the model will learn to pay attention to them. On the contrary, if DINO is provided with random, independent and identically distributed (iid) data, it will struggle to understand on what to focus the attention.

Chapter 3

Method

This chapter provides details concerning the methodology followed in this Thesis. The study explores different strategies to perform a binary patch level "litter"/"no litter" classification task when few or no labels are provided. First, the annotation process and data preparation will be explained in detail. Then, from a simple Random Forest taken as baseline model, more advanced strategies involving CNN and Stacked models, Transfer Learning, Semi-Supervised and Self-Supervised approaches will be presented in depth.

3.1 Data preparation

The dataset used for this study consists of non-overlapping patches of dimension 256×256 pixels obtained from UAV RGB images. The patch dimension was chosen in order to include enough context for the models to distinguish litter from water, sand, stones and vegetation. For annotating the examples, a Graphic User Interface (GUI) named Annotator was implemented. The GUI shows the patch alongside with the whole image it belongs to, where its position is highlighted by a red square to provide some more context details. For each patch, the user is asked to indicate whether litter is present or not. The GUI presents the adjacent examples in sequence (i.e., as a window sliding over the current image), but the user can decide whether to change image or skip the current patch, in case they want to label some specific images or areas. Since the goal is to detect regions containing cluttered and dense litter areas, the patches characterized by small sparse waste residuals were annotated as "non-litter". Figure 3.1 provides an overview of the Annotator GUI.

3.2 Supervised Learning Models

3.2.1 Random Forest

A Random Forest model was selected as baseline algorithm to which the other experiments are compared. This choice was motivated by the fact that Random Forests are generally robust to over-fitting, they train fast and usually provide pretty

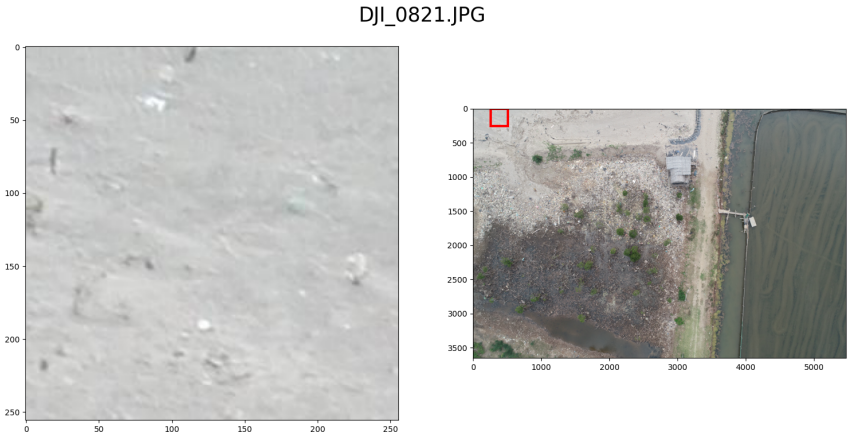

Figure 3.1: Annotator GUI. The left side shows the patch, whereas the right side the patch location on the image it belongs to. The name of the image is reported on top.

good results thanks to their DT ensemble structure. As traditional ML models, they require a feature vector as input. Therefore, the patches are first mapped to feature vectors before being fed to the Random Forest ensemble. Since feature selection is well known to be a challenging task, a typical and easier procedure for dealing with images consists of flattening the 2D pixel grid onto a 1D vector. However, in this case the patch size would lead to a 1D feature vector of dimension 65536. Therefore, in order to compress the input, the patch information are mapped onto histograms of 100 bins to obtain 1D feature vectors of size 100. The Random Forest hyper-parameters, such as the depth of the DTs and the maximum number of features, were selected through a Randomized Search on a hyper-parameter grid. This search is much quicker than a Grid Search because not all possible combinations of hyper-parameters are tested. Despite the hyper-parameter values are sampled from the distributions provided in the grid, this approach is still able to find a rather suitable combination to achieve good performance.

3.2.2 CNN

The next model tested was a small CNN architecture built from scratch specifically for the "litter"/"no litter" binary classification task. As reported in Chapter 2, CNNs are well suited for dealing with images, as they are able to also take into account the context within their field of view. Before being passed to the network, the inputs are normalized and then augmented with horizontal flips, random rotations and contrast changes to extend the training set. The built CNN architecture includes three blocks, each one characterized by a Convolutional layer followed by a BN step, a MaxPooling layer and a Dropout layer. The Convolutional layer filters are set to 16 in the first block, doubled to 32 in the second and again to 64 in the last one. The

Dropout fraction is maintained at 0.25 in all the blocks. A final portion predicting the "litter"/"no litter" classes is placed at end of the chain. The feature map output by the last Convolutional block is first flattened into a 1D vector and then passed as input to a Fully Connected layer composed by 64 units. Such a Dense layer is followed by a Dropout layer at 0.50 before being passed to the final Fully Connected classification layer, where a *Sigmoid* activation function is used to obtain the final predictions. The CNN architecture is shown in Figure 3.2.

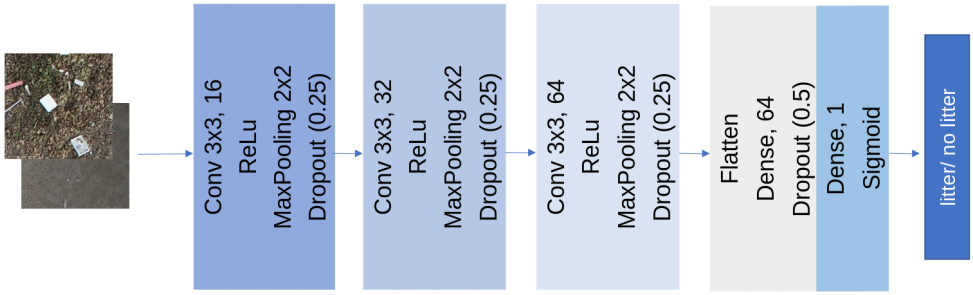

Figure 3.2: CNN architecture

3.2.3 MobileNetV2

Transfer Learning is usually a recommended practice for taking advantage of models that have been trained with major computational resources on very large datasets (such as ImageNet [7]) and, therefore, learned richer feature representations. After Fine-Tuning a pre-trained model on a relatively small dataset representative of the task, the final network is usually able to perform pretty well on the new goal. Given the limited amount of labeled data available for training, this practice was considered suitable for this Thesis task.

The pre-trained network selected was the MobileNetV2 [20]. This model was implemented to run on mobile devices, therefore its architecture is rather light and small compared to other pre-trained models. The MobileNetV2 pre-trained on ImageNet was taken from the Keras database [21]. The top classification layers of the model were replaced with a Global Average Pooling (which averages the feature map to obtain a single vector to be passed to the Softmax layer) and a final Fully Connected layer with activation function *Sigmoid*.

3.2.4 Stacked Models

A practice that generally boosts performance is using a DL model as feature extractor and then feeding the learned representations into a ML classifier to obtain the final predictions. In this Thesis, the models presented in the paragraphs above were combined to create a stacked model structure. In particular, two structures were tested: CNN with Random Forest, and MobileNetV2 with Random Forest. The CNN and the MobileNetV2 were used as feature extractors, whereas the Random

Forest was used as binary classifier. Therefore, the feature mappings from the second last layer were extracted and passed to the Random Forest to train on. As in the case of the Random Forest alone, the hyper-parameters of the ensemble were tuned via Randomized Search onto a hyper-parameter grid.

3.3 Semi-Supervised Learning Models

Another attempt to achieve good performance without labelling more data consisted of a Transductive-based Semi-Supervised approach. To this end, the tests involved the stacked structures of CNN and MobileNetV2 combined with a Random Forest. The previously trained CNN extracts the features from training and unlabelled data, then the Random Forest is used to perform the label propagation. Specifically, the classifier learns from unlabeled data by iteratively predicting pseudo-labels (i.e., labels assigned on the basis of annotated data) for some unlabeled data and adding them to the training set, until no more pseudo-labels can be added. The same procedure was followed for the combination MobileNetV2 with Random Forest previously trained.

3.4 Self-Supervised Learning Models

3.4.1 DINO

The last approach tested to deal with the lack of labeled data was DINO [6]. As explained in Chapter 2, this framework learns input representations in a SSL mode and is completely label-free. To validate this approach on the "litter"/"no-litter" classification task, DINO was used as feature extractor whose output was processed, in a first experiment, by a k-NN and, in a second experiment, by a Linear model. Those classifiers were chosen to follow the same evaluation procedure of the original paper.

Chapter 4

Results and Discussion

This chapter presents and discusses the results obtained by evaluating the strategies and combinations proposed in Chapter 3. First, details concerning the dataset used in this study for training and testing will be provided. Second, the subsequent paragraphs will cover experiment setups both in terms of parameters and computational resources used. Last, the results obtained will be presented and discussed in detail.

4.1 Dataset

The datatset provided for this Thesis consisted of 218 UAV RGB images acquired with a DJI Phantom 4 drone at 50 meters above the ground in two different areas of the Vietnamese coast. Images have dimensions of $(3648, 5472, 3)$ pixels and resolution of 2 m/pixel. Figure 4.1 shows one of the selected images.

This dataset was completely unlabeled and images acquired in time-sequence showed big overlapping areas ($\sim$40% of the image). Therefore, before proceeding with the experiments, 12 non-overlapping images with visible litter content were selected from each of the two covered areas (for a total of 24 images) and partitioned into patches of size 256×256 pixels, as explained in Chapter 3. This procedure led to a total of 7056 patches, out of which 2070 were manually annotated using the Annotator GUI (Section 3.1). The annotated examples were then divided into the training, validation and test sets by maintaining the dataset class balance as follows:

- **Training set**: 1079 "no litter" patches, 681 "litter" patches
- **Validation set**: 94 "no litter" patches, 61 "litter" patches
- **Test set**: 99 "no litter" patches, 56 "litter" patches

4.2 Hyper-parameter setup

The whole study was carried out in the Google Colab environment [5]. In this online workspace, the user can ask for a GPU device to use for their experiments. Despite

Figure 4.1: Example of a selected image of the Vietnam dataset.

the usage and power limitations of the provided GPU, Colab is a useful platform for running some experiments by taking advantage of additional computational resources.

Random Forest

As explained in Chapter 3, Random Forest hyper-parameters were selected through a Randomized Search on a hyper-parameter grid. This search led to the selection of 100 as the maximum depth of the DTs, 5 as the minimum number of samples required to split a node, and 73 as the number of estimators.

The accuracy obtained on the training set and on the test set was 100% and 88% respectively. The Receiver Operating Characteristic (ROC) computed on the test set is reported in Figure 4.4.

CNN

In order to get more robust and unbiased results, the training and validation sets defined above were merged together to perform the training through k-fold cross-validation with $k = 3$. Adam was chosen as optimizer and the binary Cross-Entropy selected as loss. The training was carried out for 300 epochs for each fold with a batch size of 32 examples and a starting learning rate of 10^{-3}, which is exponentially decreased in case a plateau region on the validation loss is detected. After the training, the predictions of the three models obtained were combined via majority

voting strategy. The CNN validation and training average accuracy and loss curves are shown in Figure 4.2.

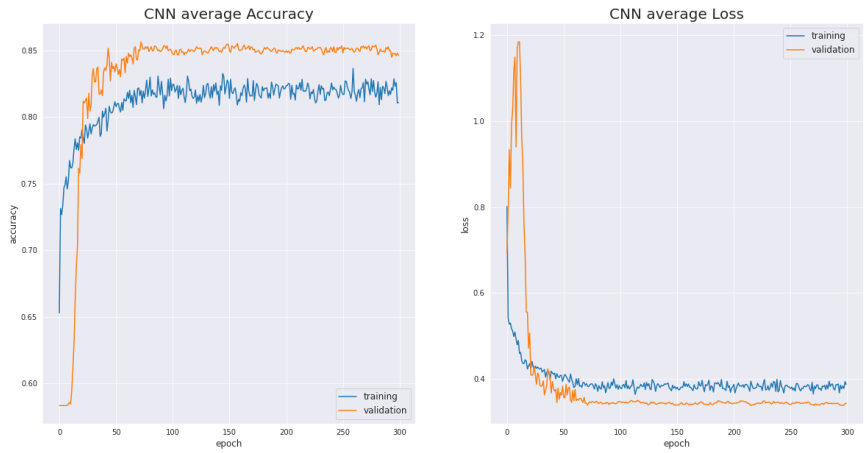

Figure 4.2: Average CNN Training performance of the 3-folds. The plot on the left reports the average Accuracy curves, whereas the right plot reports the average Loss trends for both training and validation sets.

The accuracy obtained on the training set and on the test set was 84% and 86% respectively. The ROC computed on the test set is reported in Figure 4.4.

MobileNetV2

The considerably different class performance obtained in preliminary Transfer Learning experiments suggested that the model was pretty sensitive to class unbalance. Since Fine-Tuning is usually carried out with relatively small datasets, discarding a substantial amount of "no litter" examples was considered, in this case, an acceptable solution. Therefore, the "no litter" class was downsized to 681 examples to match the "litter" class prior to training. The training was performed following the Transfer Learning procedure. First, all the MobileNetV2 pre-trained weights were frozen in order to train the parameters only for the newly added classification layers, for 30 epochs at learning rate of 10^{-3}. Then, all the weights were unfrozen to proceed with the Fine-Tuning step, for 10 epochs at learning rate of 10^{-5}. The optimization strategy selected was Adam in both steps. Figure 4.3 shows the training performance of MobileNetV2.

The accuracy obtained on the training set and on the test set was 97% and 84% respectively. The ROC computed on the test set is reported in Figure 4.4.

CNN and Random Forest

For this combination the CNN, which performed best on the test set out of the three models produced with the k-fold, was selected as feature extractor. The

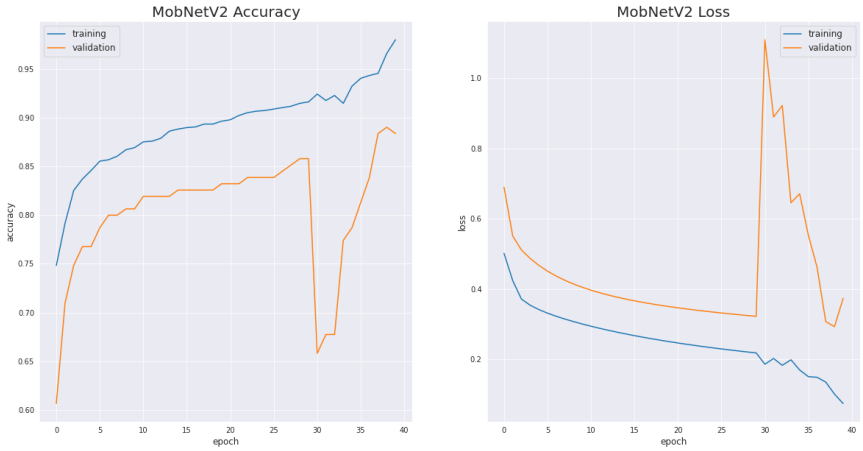

Figure 4.3: MobileNetV2 training. The plot on the left reports the accuracy trends, whereas the right plot reports the loss trends for both training and validation sets. A visible performance degradation can be noticed around the 30^{th} epoch. This is due to the switch from Transfer Learning, where only the top layers are trained, to the Fine-Tuning procedure, where the whole network is trained at low learning rate.

hyper-parameters of the Random Forest were again selected through Randomized Search and the classifier trained on the CNN features. The selected hyper-parameters consisted of 31 DTs, a maximum tree depth of 70 and a minimum number of samples for splitting a node of 10. The accuracy obtained on the training and test sets was 96% and 92% respectively. The ROC computed on the test set is reported in Figure 4.4.

MobileNetV2 and Random Forest

The procedure followed for this experiment was the same as the one above, i.e., the CNN and Random Forest combination. In this case, the MobileNetV2 model previously trained was used as feature extractor instead of the CNN. The hyper-parameters selected by the Randomized grid Search were 52 estimators, a maximum depth of 30 and a minimum number of samples required for node splitting of 4.

The accuracy obtained on the training set and on the test set was 100% and 83% respectively. The ROC computed on the test set is reported in Figure 4.4.

Semi-Supervised Experiments

The Stacked models obtained were further trained in Semi-Supervised settings. In each experiment, the feature extractor was used to extract features from the training and unlabelled data, which were then passed to the Random Forest. As explained in Chapter 3, the classifier trained in Semi-Supervised mode provides unlabelled data with pseudo-labels and progressively adds them to the training set during the learning process.

The accuracy obtained on the test set was 90% for the combination CNN and Random Forest and 83% for the combination MobileNetV2 and Random Forest. The ROC computed on the test set is reported in Figure 4.4.

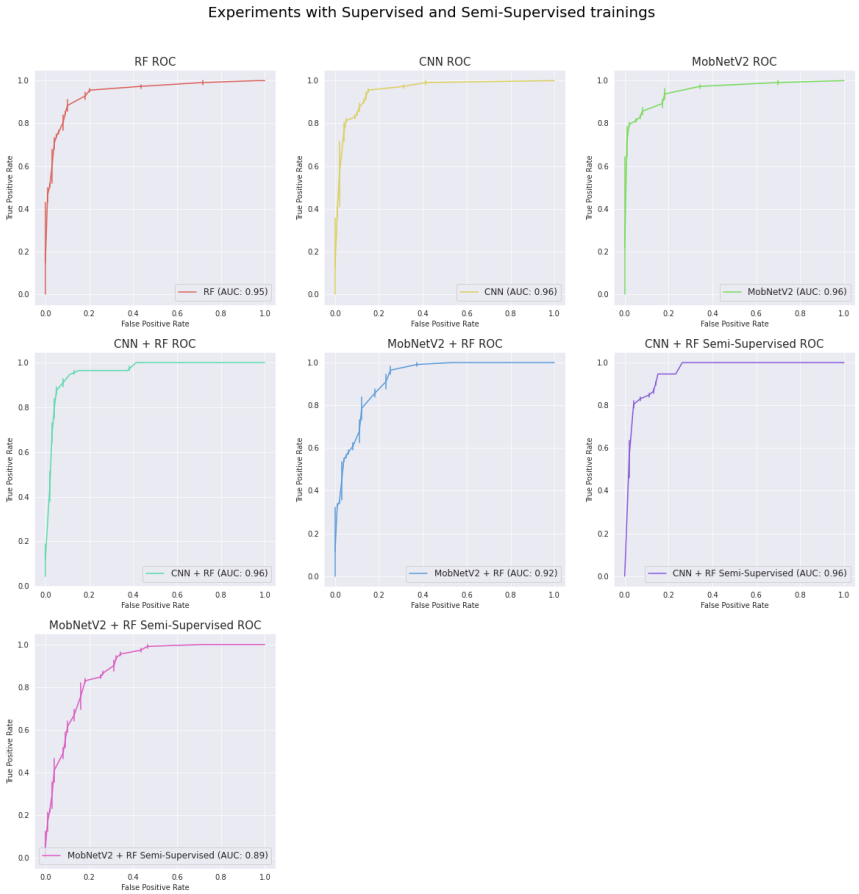

Figure 4.4: ROC curves with error bars for Supervised and Semi-Supervised learning experiments.

DINO

Since DINO works in SSL mode and is completely label-free, the framework was trained in two different settings: the first exploits the same training set used for the Supervised experiments, the second the same training set used for the Semi-Supervised experiments, consisting of all the labeled and unlabeled data except for those belonging to the test set.

DINO offers the possibility to choose between different ViTs structures for training, which differ in the size of the embedding output and in the number of ViT heads.

k	training	test
5	88%	88%
10	84%	85%
20	83%	84%
100	80%	81%
200	77%	76%

Table 4.1: Performance of DINO and k-NN on the training and test sets. The training set is the same one used for the Supervised experiments.

k	training	test
5	88%	86%
10	85%	83%
20	84%	84%
100	80%	80%
200	76%	75%

Table 4.2: Performance of DINO and k-NN on the training and test sets. The training set is the same used for the Semi-Supervised experiments, consisting of both labeled and unlabeled examples.

Specifically, for the *ViT-tiny*, *ViT-small* and *ViT-base* architectures, the embedding dimensions are 192, 384 and 768, while the number of heads 3, 6 and 12, respectively. The DINO framework was adapted to run in Colab and, due to computational resource limitations of such an environment, the architecture used in this Thesis was *ViT-tiny*, further simplified by setting the number of heads to 1 instead of 3. The batch size was set to 8 and the training carried out for 100 epochs with Adam as optimizer. All the other training hyper-parameters were kept to the default values suggested in the paper [6].

The two trained DINO models were then used to extract the two training dataset features to feed a k-NN and a Linear classifier. Concerning the k-NN, multiple values for k were tested and the change in performance was observed and reported in Figure 4.5. The accuracy obtained on the training and test sets using different values of k for both experiments are reported in Tables 4.1 and 4.2.

The accuracy obtained for DINO combined with a Linear classifier was 80% on the training set and 79% on the test set, in both the training settings tested. The results of the combination of DINO and Linear classifier are reported on Figure 4.6.

4.3 Evaluation

Results were first evaluated in terms of accuracy differences on the test set, as compared to the Random Forest model selected as baseline. The accuracy performance of all the experiments and their relative improvements are reported in Table 4.3.

Except for the experiments involving the MobNetV2 (-4%/-5%) and DINO combined with the Linear Classifier (-9%), the performance of all the other models

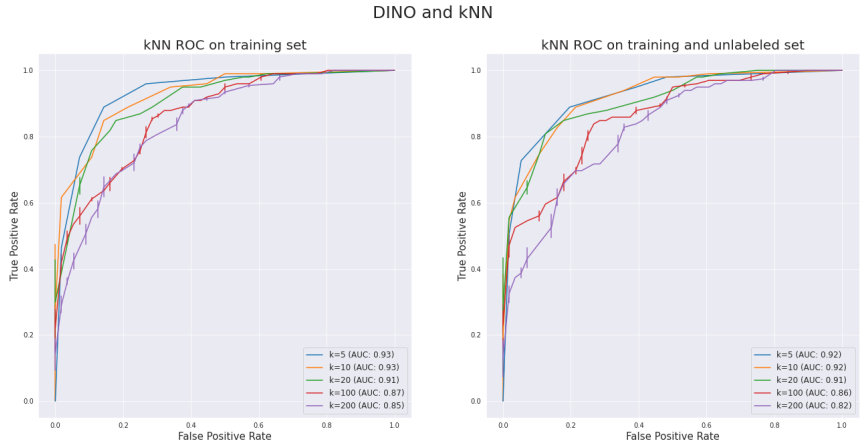

Figure 4.5: ROC curves with error bars of Self-Supervised learning experiments with DINO and k-NN classifier. The trends are related to different number of neighbors k set.

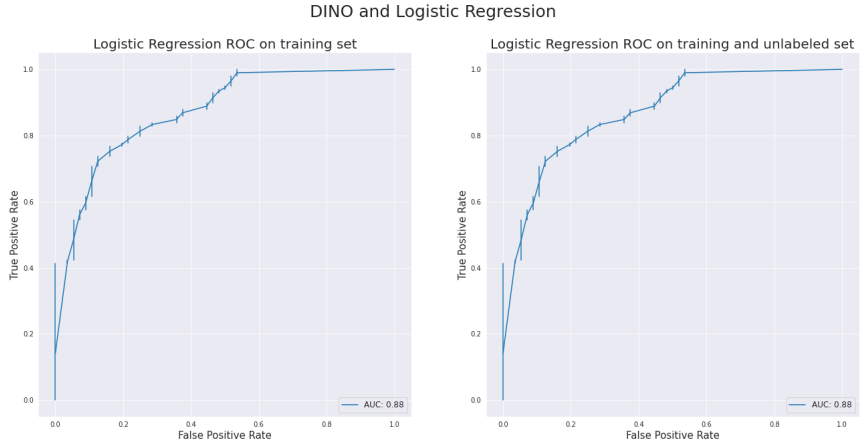

Figure 4.6: ROC curves with error bars for Self-Supervised learning experiments with DINO and Linear classifier.

was very close to the one obtained with the Random Forest (±2%). A more significant improvement was made by the combination CNN and Random Forest (+4%).

These results highlight the different model capability of extracting the proper features crucial to distinguish the "litter" from the "no litter" class. Specifically, the CNN and DINO seem to be pretty good and robust feature extractors: The CNN presents improvements in both Supervised and Semi-Supervised settings when combined with the Random Forest, whereas DINO combined with the 5-NN was able to achieve the same performance as the baseline, despite training completely label-free.

However, it is also clear that the choice of the classifier heavily affects the outcome: as it can be noticed, the Linear classifier is not able to fully exploit the information provided by DINO features. This is also highlighted by the fact that results are exactly the same when using DINO trained on the fully annotated training set or the one trained on the Semi-Supervised training set, whereas small differences can be noticed in the two 5-NN experiments.

As for the experiments involving the MobileNetV2, it can be observed that the training performance was considerably better than the one obtained at testing time. Such an outcome suggests that model was probably over-fitting due to the fact that the training set was too small even for Fine-Tuning. In fact, it should be taken into account that the MobileNetV2 was trained on ImageNet, which is a completely different dataset from the one used in this Thesis. Therefore, the Fine-Tuning phase should probably have been "heavier" in terms of learning rate and epochs, but this would also have required a larger dataset to avoid the over-fitting issue.

The models were also evaluated in terms of Recall performance for the "litter/no litter" classes of the test set. Specifically, the aim of this evaluation was to get a clearer idea about the model ability to detect the "litter" class. The results of this analysis are reported in Table 4.4, along with the performance differences for the "litter" class compared to the baseline. Overall, the Recall results confirm the points already highlighted with the accuracy-based evaluation.

On the other hand, the inconsistency in performance for the experiments involving the MobileNetV2 needs a further explanation. The MobileNetV2 alone was trained on a balanced training set, whereas for the experiment in combination with the Random Forest the model was used to extract the features from the unbalanced training set in order to train the downstream classifier. In the experiment in Semi-Supervised setting, however, the Random Forest benefits from label propagation learning and is able to find a better balance between the capability of the Fine-Tuned , which is particularly good on the "litter" class, and the Random Forest, that favors the "no litter" class.

In general, the effect of the class unbalance on the performance can be noticed in all the experiments, where the "no litter" Recall differs quite substantially from the "litter" one. This problem should be taken into account (and possibly addressed) when dealing with ML and DL models trained in Supervised mode. On top of that, as mentioned in the introduction, an additional challenge is represented by the fact that litter class has high variance in terms of color, brightness, shape, size and so on. Detecting such a random pattern and finding representative features that allow

model	**training**	**test**
Random Forest (baseline)	100%	88%
CNN	84%	86% **(-2%)**
CNN + Random Forest	96%	92% **(+4%)**
CNN + Random Forest (Semi-Supervised)	/	90% **(+2%)**
MobileNetV2	97%	84% **(-4%)**
MobileNetV2 + Random Forest	100%	83% **(-5%)**
MobileNetV2 + Random Forest (Semi-Supervised)	/	83% **(-5%)**
DINO + 5-NN (1)	88%	88% **(+0%)**
DINO + 5-NN (2)	88%	86% **(-2%)**
DINO + Linear Classifier	80%	79% **(-9%)**

Table 4.3: Performance summary. The results on the test set are sided by the accuracy improvement relatively to the Random Forest taken as baseline model. The results reported for DINO and k-NN are those for k=5 on the labeled training set (1) and on the Semi-Supervised training set (2).

spotting litter in the water, within the vegetation or over the stones is a non-trivial task per se.

Overall, this evaluation highlights the potential of the Semi-Supervised and, in particular, the SSL approaches. In fact, the main interest of this study was to assess whether good performance could be achieved with few or no labels available. Thus, even no improvement or a small difference compared to the baseline can be regarded as a promising outcome for future studies and developments.

model	Recall non-litter	Recall litter
Random Forest (baseline)	92%	82%
CNN	91%	79% **(-3%)**
CNN + Random Forest	97%	82% **(+0%)**
CNN + Random Forest (Semi-Supervised)	94%	82% **(+0%)**
MobileNetV2	77%	96% **(+14%)**
MobileNetV2 + Random Forest	91%	71% **(-11%)**
MobileNetV2 + Random Forest (Semi-Supervised)	84%	79% **(-3%)**
DINO + 5-NN (1)	92%	81% **(-1%)**
DINO + 5-NN (2)	89%	80% **(-2%)**
DINO + Linear Classifier	85%	71% **(-11%)**

Table 4.4: Class Recall summary. For the litter class, the result is sided by the Recall improvement relatively to the Random Forest taken as baseline model. The results reported for DINO and k-NN are those for k=5 on the labeled training set (1) and on the Semi-Supervised training set (2).

Chapter 5

Conclusions and Future Works

The work of this Thesis aimed at exploring some possible options to cope with the lack of annotations in the context of marine litter detection. To this end, the performance obtained with both ML and DL algorithms in Supervised, Semi-Supervised and Self-Supervised learning modalities were compared among them on a "litter"/"no litter" patch-level classification task. The study was carried out on 24 selected UAV RGB images acquired at 50m above the Vietnamese coast.

First, the dataset was prepared by partitioning such images in non-overlapping patches of dimensions 256×256 pixels. Out of 7056 patches, 2070 were manually labeled as "litter"/"no litter". Second, experiments were split into three groups according to the learning approach used. For the Supervised algorithms, a Random Forest, a CNN built and trained from scratch, a MobileNetV2 trained via Transfer Learning and combinations of CNN and MobileNetV2 with the Random Forest were tested. Those stacked model solutions were then also tested in Semi-Supervised mode performed via label propagation. Last, experiments with a recent ViTs and Knowledge Distillation-based framework called DINO combined with a k-NN and a Linear classifier were carried out for the Self-Supervised learning category.

The results were evaluated in terms of accuracy and "litter" class Recall, as compared to the ones obtained with the Random Forest taken as baseline: 88% and 82% respectively.

All the options involving the CNN reported an accuracy very close to the baseline (between -2% and +4%). This similarity was noticed also in terms of Recall, where the solutions with the CNN performance differed from the reference between -3% and +0%.

On the other hand, the experiments ran with the MobileNetV2 resulted on a slightly worse performance in accuracy (-4%/-5%) and very different "litter" Recall outcomes. Specifically, the MobileNetV2 alone reported an improvement of +14%, which dropped at -11% when stacking the model with a Random Forest and to -3% when the latter structure was trained in Semi-Supervised mode.

As for DINO, in one case it was trained on the same training set used for the Supervised approaches, in the other on the same training set exploited for the Semi-Supervised learning settings. In both training scenarios, DINO in combination with

the k-NN (with k=5) reported results very close to the baseline in both accuracy (+0% and -2%) and "litter" Recall (-1% and -2%). For the combination with a Linear classifier, no difference in performance were detected for the two DINO models and in both cases accuracy and Recall were considerably worse than the baseline (-9% and -11% respectively).

These results highlighted the possibility of obtaining well trained algorithms able to properly select representative features crucial to distinguish the "litter" from the "no litter" class, even with a minimal amount or no labeled data provided. The potential of the Semi-Supervised and, in particular, of the Self-Supervised approaches stands out in the evaluation outcomes, as it was possible to obtain a performance extremely close to the baseline even in those learning settings. The results of this study suggest that less annotation-eager learning approaches, along with novel architectures such as ViTs, could effectively be exploited in the marine litter detection context.

Given the potential of these methodologies in classification tasks, the next step for this study should focus on testing label-free techniques for semantic segmentation tasks. Pixel-level information would in fact be crucial to proceed with further downstream analyses and being able to accurately detect and locate marine litter. To this end, Appendix A of this Thesis provides some preliminary results on the usage of a novel framework able to exploit the information of the feature representations produced by DINO (or any other feature extractor) to output semantically meaningful and consistent segmentation clusters.

However, it is important to underline that a larger and independent test set as compared to the one annotated in this Thesis would help drawing more robust and trustworthy conclusions.

Appendices

Appendix A

STEGO

Self-Supervised **T**ransformer with **E**nergy-Based **G**raph **O**ptimization **(STEGO)** [10] is a novel framework meant to improve the results of Unsupervised semantic segmentation approaches. As opposed to such methodologies, which usually attempt to achieve segmentation outcomes through an end-to-end framework, STEGO separates the feature learning process from their aggregation into semantically meaningful clusters. This approach makes STEGO agnostic to the algorithm being used as feature extractor in the first step. Consequently, existing or even novel SSL-based algorithms could be integrated into this framework with relatively low effort.

The idea behind STEGO is based on the assumption that *"the current unsupervised feature learning frameworks already generate dense features whose correlations are semantically consistent"* [10]. Therefore, such a characteristic can be exploited to encourage similar features to group in clusters while preserving their relationships across the corpora.

STEGO distills the features into the different labels thanks to a novel formulation of contrastive loss, which is used to train a segmentation head. Given two input images x and y, such a loss function tries to push together the segmentation features associated to them if a significant similarity between their feature tensors is detected.

The final loss consists of a linear combination of the individual correspondence losses related to the features of three different input pairs. Given an image, the first loss term will be formed by the correlation among its features (self-correlation), the second by the correlation with features of another image within its k-NN, and the third by correlation with those belonging to a random image. The first two loss terms will provide an attractive contribution, whereas the last one a repulsive effect. At prediction time, the clusters are extracted using a cosine distance-based mini-batch K-Means [15] on the segmentation head output and finally refined with a fully Conditional Random Field (CRF) [13].

A.1 Preliminary Results

In the original paper, the authors of STEGO tested their framework in combination with DINO [6]. Their experiments showed promising segmentation results, which

encouraged running some preliminary tests in the context of this Thesis.

As done for DINO, the STEGO framework was adapted to run in Colab [5] and make use of the architecture *ViT-tiny* trained in DINO experiments. In this case, being the pixel level labeling the ultimate goal, the number of classes (i.e., clusters) was set to 5: "water", "sand", "stones", "vegetation" and "litter".

The preliminary results highlighted that a SSL approach can be potentially exploited also for semantic segmentation tasks. In fact, if gathering image/patch level annotations is time consuming, collecting pixel level annotations is much more demanding, error prone (i.e., it is harder to get good quality annotations) and time consuming. Therefore, in the context of marine litter detection (and in many other fields), being able to benefit of a label-free segmentation approach without losing fine-grained results would be extremely valuable. In fact, thanks to pixel level information, it would be possible to quantify and accurately locate waste residuals, hence providing important details for further actions and analyses.

To conclude, this preliminary study suggests that it might be worth exploring this path further in future works.

An example of segmentation clusters obtained with STEGO relatively to a litter patch is shown in Figure A.1. As it can be noticed, STEGO seems to distinguish "litter", "vegetation" and "sand" classes pretty well.

Figure A.1: Example of patch segmentation output by STEGO.

Bibliography

[1] T. Acuña-Ruz, D. Uribe, R. Taylor, L. Amézquita, M. C. Guzmán, J. Merrill, P. Martínez, L. Voisin, and C. Mattar. Anthropogenic marine debris over beaches: Spectral characterization for remote sensing applications. *Remote Sensing of Environment*, 217:309–322, 2018.

[2] S. Bak, D. Hwang, H. Kim, and H. Yoon. Detection and monitoring of beach litter using uav image and deep neural network. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2019.

[3] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8624–8628. IEEE, 2013.

[4] L. Biermann, D. Clewley, V. Martinez-Vicente, and K. Topouzelis. Finding plastic patches in coastal waters using optical satellite data. *Scientific reports*, 10(1):1–10, 2020.

[5] E. Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[6] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9650–9660, 2021.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[9] G. Gonçalves, U. Andriolo, L. Pinto, and F. Bessa. Mapping marine litter using uas on a beach-dune system: a multidisciplinary approach. *Science of the Total Environment*, 706:135742, 2020.

[10] M. Hamilton, Z. Zhang, B. Hariharan, N. Snavely, and W. T. Freeman. Unsupervised semantic segmentation by distilling feature correspondences. *arXiv preprint arXiv:2203.08414*, 2022.

[11] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 2021.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *Advances in neural information processing systems*, 24, 2011.

[14] K. Kylili, I. Kyriakides, A. Artusi, and C. Hadjistassou. Identifying floating plastic marine debris using a deep learning approach. *Environmental Science and Pollution Research*, 26(17):17091–17099, 2019.

[15] J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[16] C. Martin, S. Parkes, Q. Zhang, X. Zhang, M. F. McCabe, and C. M. Duarte. Use of unmanned aerial vehicles for efficient beach litter monitoring. *Marine pollution bulletin*, 131:662–673, 2018.

[17] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[18] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[19] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[21] K. Team. Keras documentation: Keras applications.

[22] T. Tieleman, G. Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[23] K. Topouzelis, D. Papageorgiou, G. Suaria, and S. Aliani. Floating marine litter detection algorithms and techniques using optical remote sensing data: A review. *Marine Pollution Bulletin*, 170:112675, 2021.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[25] M. Wolf, K. van den Berg, S. P. Garaba, N. Gnann, K. Sattler, F. Stahl, and O. Zielinski. Machine learning for aquatic plastic litter detection, classification and quantification (aplastic-q). *Environmental Research Letters*, 15(11):114042, 2020.