Future Generation Computer Systems 174 (2026) 108000

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A deep learning-based method for efficient floating garbage debris recognition on high-performance edge computing platform

Diego Romano [a,*], Carlo Mennella [b], Marco Lapegna [c]

[a] Inst. for High Performance Computing and Networking, National Research Council, Naples, Italy
[b] Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy
[c] Department of Mathematics and Applications, University of Naples Federico II, Naples, Italy

ARTICLE INFO

Keywords:
Deep learning
High-performance edge computing
Energy efficiency
Floating garbage debris recognition

ABSTRACT

Our research introduces a novel method to identifying floating garbage debris using deep learning and High Performance Edge Computing (HPEC). We utilize a convolutional neural network (CNN) to classify debris from images captured by an RGB camera on a vessel, aiming for high accuracy and efficiency on low-power devices.

We conducted a comparative analysis of various models, implementing optimization techniques like transfer learning and pruning. Each model was evaluated for accuracy, inference time, and energy consumption, leading us to develop a multi-objective function to determine the best approach.

Our findings show that the proposed method effectively detects and classifies floating debris on desktop GPUs and low-power devices such as the Jetson Nano. Furthermore, it maintains accuracy by optimizing memory and computational power requirements while adapting to different energy needs.

These advancements can enhance the capacity to combat marine plastic pollution and promote intelligent systems integration within environmental initiatives. They facilitate precise, real-time detection of floating debris on energy-constrained edge platforms, thereby effectively addressing deployment challenges in marine environments.

1. Introduction

The issue of marine litter represents a significant environmental challenge that inflicts severe damage on marine ecosystems. Nearly 8 million tons of plastic waste are deposited annually into the ocean, and this figure continues to rise [1]. The sources of marine litter are diverse, encompassing land runoff, maritime activities, and fishing practices, all of which have detrimental effects on marine life. Marine organisms can become entangled in debris, ingest, or suffer from toxic exposure. Furthermore, coral reefs and other marine habitats also face degradation due to marine debris.

Several methods exist for monitoring marine litter, including remote sensing. This technique utilizes satellites, aircraft, or drones to gather data about the Earth's surface and identify ocean debris, even in remote areas [2]. By employing specialized sensors like Synthetic Aperture Radar (SAR), it can effectively track large volumes of plastic in the oceans [3]. However, it is unsuited for detecting individual floating debris and poses specific computational challenges [4]. Tracking individual objects aids in understanding the sources and pathways of marine pollution, identifying and mitigating associated hazards to coastal ecosystems, such as the transfer of harmful chemicals and pollutants, and enhancing the quality of coastal recreational activities. Indeed, remote sensing monitoring of oceanic litter does not significantly impact the health of coastal regions, home to a substantial portion of the global population and essential to the worldwide economy.

Thanks to *data crowdsourcing* [5] efforts, now citizens on their boats can implement an automatic tool to identify and track single pieces of floating garbage debris using an Edge Computing platform [6].

Recent advancements in computing technology have led to the development of devices that integrate multi-core CPUs, GPU-based parallel architectures, and microcontrollers for sensor management, all on a single board. These devices can serve as High-Performance Edge Computing (HPEC) platforms, enabling the implementation of remote Machine Learning (ML) and Deep Learning (DL) algorithms to process images from sea cameras, even aboard vessels without network connectivity in navigation. However, creating applications for these devices is challenging, as it requires finding the right balance between high performance and low power consumption [7,8].

In recent years, advanced DL techniques have profoundly transformed the fields of image processing and classification. Among these

* Corresponding author.
E-mail addresses: diego.romano@cnr.it (D. Romano), mennellacarlo1@gmail.com (C. Mennella), marco.lapegna@unina.it (M. Lapegna).

https://doi.org/10.1016/j.future.2025.108000
Received 30 April 2025; Received in revised form 10 June 2025; Accepted 26 June 2025
Available online 10 July 2025

methods, Convolutional Neural Networks (CNNs) have demonstrated particular effectiveness in analyzing standard optical, RGB, and infrared (IR) images. These networks adeptly perform various tasks, including object detection, classification, and semantic segmentation [9–11]. By training a CNN on a labeled image dataset, we can accurately distinguish between sea camera images containing litter and those that do not. This capability can be effectively employed on an HPEC device on board a vessel, offering valuable insights into the prevalence of floating debris and aiding the development of more effective strategies for locating and removing it. However, recent deep learning-based studies for marine waste detection rely on high-power GPU platforms and seldom address the constraints of real-time edge deployment, such as power consumption, memory footprint, and processing latency.

This work addresses these limitations. More precisely, innovative contributions in this paper are:

- we introduce a novel approach that directly combines High-Performance Edge Computing with lightweight Convolutional Neural Networks optimized for inference speed and energy efficiency. Our method enables real-time floating debris classification using RGB imagery from vessel-mounted cameras during navigation;
- we select an appropriate HPEC platform and leverage its features by applying transfer learning and pruning techniques;
- we tailor the CNN model through a newly defined multi-objective selection metric to balance accuracy, speed, and energy use—three critical constraints for autonomous, field-deployable systems;
- we present through experimental evaluation of the effectiveness of our approach concerning the state of the art.

Section 2 explores related work, explaining the diverse sources of marine debris, their spectral and thermal properties, remote sensing techniques, and deep learning-based methods for detecting and classifying floating waste. It also mentions the limitations of such works and how our approach aims to address them.

Section 3 presents the dataset employed for training and testing the CNN, the choices made for dataset optimization, the definition of the CNN architecture, how we divided the dataset into training, validation, and test sets, and the utilization of transfer learning and pruning techniques to enhance CNN performance.

Section 4 describes the computational environment chosen for the project and introduces the methodology we used to identify the best models in an Edge Computing environment.

Section 5 delves into the results of the experiments. It shows and discusses the inference times of the models on the hardware platforms, including an RTX3060Ti GPU and the Jetson Nano at two power levels (10 W and 5 W).

Finally, Section 6 summarizes the project's objectives, original contributions, encountered limitations and challenges, and potential future development prospects. It emphasizes the critical role of innovative solutions based on deep learning and edge computing in tackling marine plastic pollution.

2. Related work

The identification of plastics in the ocean is possible through the utilization of spectral reflectance [12]. The remote sensing process involves measuring the environment and objects through electromagnetic radiation. To characterize terrestrial surfaces, one can rely on the Visible, Shortwave Infrared (SWIR), and Thermal Infrared (TIR) spectra. These spectra derive from a combination of energy emitted by the Earth, energy reflected by the sun, and energy from devices like microwave generators.

Research has delved into the spectral reflectance of marine macroplastics using visible and SWIR ranges to enhance remote sensing technologies in detecting plastic waste [13]. Spectral bands in the visible and SWIR range are essential in recognizing wet plastic, representing a typical situation in natural aquatic environments. By examining spectral signatures of water-borne garbage debris [14], various materials in marine waste can be distinguished and identified, thus streamlining the classification of plastic waste. Nonetheless, the hurdles to overcome include developing algorithms to differentiate plastic waste from other materials and obtaining a high-resolution dataset.

Remote sensing using thermal infrared radiation (TIR) has the potential to detect marine plastic waste, but it has received less attention than visible and shortwave infrared wavelengths. In [15], the authors point out that TIR signals are highly responsive to temperature differences between the water surface and its surroundings. Mapping techniques that utilize sea surface temperatures (SSTs) and 2-m air temperatures (t2m) demonstrate TIR's efficacy in regions with notable temperature disparities. However, certain obstacles exist, such as temperature fluctuations due to wind and waves, and the need to incorporate TIR with other technologies to improve accuracy.

Recent scientific studies have demonstrated the potential of deep learning-based techniques in detecting plastic pollution in water bodies such as rivers [16] and oceans [17]. These techniques exhibit the ability to differentiate between various categories of waste and enhance the precision of monitoring marine plastic waste floating on the surface. Nonetheless, there is a need to further augment the datasets employed in such research, fine-tune algorithms, and select models that align with hardware architecture. In addition to the cited works, a recent review article collects a comprehensive list of over 100 contributions on the topic [18]. In all these cases, the authors give particular emphasis to the aspects of accuracy and efficiency. They give little or scarce attention to energy consumption issues, particularly challenging for low-power devices often powered by a battery. As already mentioned in the introduction, in this contribution, we intend to address this aspect that, in our opinion, has been poorly addressed in other works.

3. DL models for edge computing

After opting to employ Convolutional Neural Networks to detect floating garbage debris, the main challenge was identifying suitable models that function effectively within the constraints of a low-power and limited-resources edge computing device. Three key requirements emerged:

- Reduced number of parameters: The models needed to be lightweight, with a limited number of parameters, to ensure optimal performance on a low-power device.
- Accuracy preservation: Maintaining satisfactory accuracy in the analysis was essential, even with the reduced number of parameters.
- Computational efficiency: The models had to be efficient and capable of quick inference on edge computing device, thereby minimizing processing times.

Our strategy involved selecting efficient CNN architectures, utilizing transfer learning from CNN models pre-trained on image datasets similar to ours, analyzing and augmenting the training dataset to enhance accuracy, and tentatively pruning the model to reduce the number of parameters.

To identify the most effective resolution strategy for our recognition problem, we systematically evaluated each identified model using a range of parameters and methods.

Initially, we established a reliable dataset partition to minimize biases in the randomization process. Subsequently, we evaluated the effectiveness of transfer learning by measuring the reduction in parameters, early stopping training by assessing the training loss and the number of epochs, and pruning strategies by measuring the energy consumption and inference time.

3.1. Choice of CNN architectures

Following the approach of evaluating various pre-trained models for transfer learning [19], we began by focusing on the deep learning platform. We aimed to select a programming language with a clear and concise syntax, facilitating rapid development and experimentation. It was also essential for the language to be optimized for GPU utilization, as this significantly enhances both the training and inference phases of complex deep learning models. Additionally, we sought a language supported by a large and active community of developers and researchers. Consequently, we opted for the PyTorch open-source library for deep learning, built on Python and the Torch library [20]. Compared to other open source environments for deep learning such as Keras and Tensorflow, for our application we have chosen the PyTorch library for its flexibility features that allow rapid development of prototypes, for its wide diffusion in the scientific community and because it represents a good compromise between efficiency and resource consumption, a key aspect in edge computing applications [21].

In the initial phase, we selected the following CNN models for evaluation:

- MobileNetV2 [22] is a lightweight and efficient CNN architecture designed for mobile and embedded devices. It is known for its balance of accuracy and computational efficiency, making it a popular choice for applications where resource constraints are a concern.
- MobileNetV3 (Small and Large) [23] is an improved version of MobileNetV2, available in two variants: Small and Large. Both variants offer enhanced performance compared to MobileNetV2 while maintaining their focus on resource efficiency. The Small variant is particularly suitable for devices with limited resources, while the Large variant provides higher accuracy at the cost of increased complexity.
- ShuffleNetV2 [24] utilizes the "shuffle" concept to reduce computational cost without sacrificing significant accuracy, making it suitable for resource-constrained environments.
- SqueezeNet1.1 [25] is a highly compact architecture that minimizes the number of parameters without sacrificing performance. Version 1.1 represents an even more efficient iteration compared to the original version.
- EfficientNet (B0 and B1) [26] is a family of CNN models that balances computational efficiency and performance. B0 and B1 are two available sizes, with B0 being the lightest and B1 offering slightly better performance.
- ResNet (18, 34, 50) [27] is a family of CNN models known for using residual blocks, which help alleviate the vanishing gradient problem during the training of deep neural networks. ResNet-18 is the most lightweight variant, while ResNet-50 offers the highest accuracy but with increased complexity.
- VGG (11, 16, 19) [28] is a family that has a very deep and uniform architecture. While offering high accuracy, VGG models can be computationally expensive. The variants listed here represent different depths within the VGG family.

We chose these models based on their reputation for accuracy, efficiency, and suitability for resource-constrained devices. Later, we will evaluate them to identify the best balance between performance and computational requirements.

Given the limited computational power of edge computing devices, we opted to decouple the more computationally intensive model training from the inference process. For optimal efficiency in the first step (described in this section), we utilized a workstation featuring an AMD Ryzen 7 3700X CPU, 32 GB of RAM, and an RTX 3060 Ti GPU. In addition, this setup included CUDA 11.7, Python 3.10.6, PyTorch 2.0.0, and TorchVision 0.15.1 for initial optimization, training, and evaluation of the models. Following this, we assessed the most promising solutions for the low-power device used in our study, described in the following Section 5.

Fig. 1. A tin can image before and after augmentation.

3.2. Training dataset construction

We focused on the dataset produced by The Ocean Cleanup, an organization committed to developing technologies for removing plastic pollution from the oceans, in collaboration with Kili Technology, a data labeling platform [29]. The dataset focuses on images of floating debris on rivers and contains 4000 images with an average resolution of 1280 × 720 pixels. Each image depicts objects protruding from the background, each labeled with a bounding box. Subsequently, we extracted these pre-selected objects from the original image and saved them at a resolution of 200 × 200 pixels. This cropping process concentrates on the objects of interest, improving the quality and clarity of the images.

The original dataset comprised four categories: plastic bottles, plastic bags, other plastic waste, and non-plastic waste. To improve classification precision and reduce noise within these categories, we refined the dataset into five distinct classes to more effectively differentiate between various types of waste: (a) Containers, (b) Plastic bags, (c) Plastic bottles, (d) Tin cans, and (e) Unknown.

We initially applied random affine transformations to the dataset to enhance the CNN's performance. The transformations included rotation and shear, generating new data variations and increasing diversity in fewer classes.

The dataset analysis revealed that water in the background significantly hampers the recognition of transparent bags. We implemented data augmentation techniques to tackle this issue and create new synthetic data from the existing dataset. Specifically, we combined the Unsharp Mask and Detail image filters from the Pillow Python library to enhance image sharpness.

The first transformation applies a Gaussian blur to a duplicate of the original image, which is then compared to the original. If the detected difference between the two images exceeds a threshold value, the filter effectively subtracts the images, enhancing the details. The second filter employs a 2D convolution to highlight objects' contours and details.

Fig. 1 illustrates the application of these transformations to a tin can, demonstrating improved visibility and affine rotation.

To test the choice of the augmentation strategies described above, we measured accuracy using the ResNet-50 model, the deepest and most complex among the previously selected ones. In Fig. 2, we can notice how the introduction of additional augmentation strategies improved the accuracy of the prediction. We expected similar behavior using other models, so we used the same augmented dataset for all the following testing.

3.3. Dataset partitioning

We adopted a specific dataset partitioning strategy to optimize classification performance and standardize the evaluation results of our models. We provided each model with three distinct subsets of the dataset: training, validation, and testing. To address this challenge, we employed a random data-splitting approach.

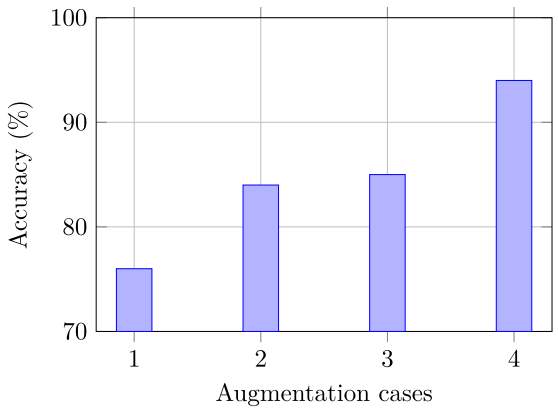

Fig. 2. Measured accuracy on ResNet-50 with four cases of augmentation strategies: 1 - Original classes, no augmentation; 2 - New five classes; 3 - Addition of affine transformed images; 4 - Addition of contour highlighting.

Since we needed to conduct ten separate tests on models for later analysis, we generated ten identical copies of the entire dataset, each stored in its folder. Each folder was then randomly divided into three subsets in the following proportions: 70% for training, 20% for validation, and 10% for testing.

This procedure yielded 30 unique folds of the dataset, each representing a portion designated for training, validation, or testing.

We monitored various evaluation metrics, including accuracy and F1 score, during the training process. The F1 score is a metric that considers both precision and recall, and it is beneficial for evaluating the performance of a classification model in a balanced manner:

$$\mathrm{F1} = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

By analyzing the F1 score obtained across the ten tests performed for each model, we could assess the performance of convolutional neural networks on different data splits.

Next, we analyzed the model's performance for individual classes by calculating each class's minimum, maximum, and mean F1 scores across all data splits. After obtaining these values, we calculated the average F1 score for each class across all models and datasets. This average score illustrates how a single data split affected the performance of all models.

Finally, we compared the average F1 scores by class for each dataset split (see Table 1) with the overall averages of the individual classes. We aimed to identify the dataset split that produced the most reliable performance across all classes. Conversely, if we selected the dataset split closest to the maximum F1 score of the individual classes, it could result in a split with fewer representative images in the testing dataset.

3.4. Transfer learning with pre-trained CNNs

Our study used various strategies to optimize the models for our HPEC context. One of the primary strategies we adopted was using pre-trained networks, which helped us improve our performance results.

After identifying suitable CNN models, we implemented *transfer learning* [30] to leverage pre-trained weights for image classification. This approach offers several advantages. Firstly, it eliminates the need for extensive training from scratch, saving significant computational resources and time. Training a complex CNN model on a large dataset like ImageNet [31], which we used in this study, can be resource-intensive. Transfer learning allows us to bypass this initial training phase and utilize the knowledge encoded in the pre-trained weights.

Secondly, pre-trained models have already learned to recognize general image features such as edges, shapes, and textures, enabling them to accurately classify all categories in the ImageNet dataset. This foundational knowledge can be effectively transferred to our case study, even if our dataset is smaller, enhancing the model's ability to learn and generalize, especially when working with limited data.

Furthermore, reducing the number of parameters in a machine learning model can lead to several advantages, particularly regarding performance and efficiency. Fewer parameters occupy less memory and reduce the computational burden, which, in turn, lowers the energy consumption of the models. A model with too many parameters may overfit the training data, making it difficult to generalize to new data. By concentrating on essential parameters, the model can focus on the most relevant features, becoming less susceptible to noise and variations in the input data. This reduction in model complexity enhances generalization abilities, helps prevent overfitting, and maintains high performance even on new or slightly different data.

A lower parameter count reduces computational complexity, minimizing resource requirements during model inference. This results in faster inference times and lower resource demands, which is crucial in our HPEC context.

We utilized transfer learning techniques to harness the knowledge acquired from pre-trained models and adapt it to the specific classes and data relevant to our problem. This process involves fine-tuning the pre-trained model to suit our needs, emphasizing the final layer, primarily responsible for assigning class probabilities to input images.

To align with our dataset, we modified this final layer, as our case requires the classification of five distinct categories. Consequently, we adjusted the final layer to generate a five-dimensional output. The pre-trained weights in the earlier layers of the model were frozen, meaning their values remained unchanged during training. This strategy helps preserve the valuable general image features learned from the extensive pre-training dataset. Ultimately, the modified final layer effectively classified our specific task.

Table 2 displays the results obtained after implementing transfer learning. The models exhibited a notable reduction in the number of parameters and the corresponding memory usage compared to those that did not utilize transfer learning.

3.5. Early stopping in training

After implementing transfer learning on the selected dataset, we retrained the networks for a maximum of 200 epochs. We utilized an early stopping criterion during training to mitigate overfitting and minimize unnecessary computational costs.

Early stopping is a monitoring technique that tracks the model's performance throughout training. A counter increments each time the model's performance fails to improve by at least 1% compared to the previous epoch. This counter continues to increase for each successive epoch without improvement, capped at a maximum of ten consecutive increments.

When the counter reaches this limit, training halts. However, if performance improves during the monitoring phase, the counter resets, and training continues. This cycle of monitoring and training persists until the model has either completed the maximum of 200 training epochs or training terminates otherwise.

This approach effectively curtails unnecessary overfitting by stopping the training process once substantial performance gains diminish, thereby saving computational resources.

As depicted in Fig. 3, some models, such as EfficientNetB1 and MobileNet v3 Large, took longer to satisfy our Early Stopping criterion, even though they achieved a lower final validation loss with minimal fluctuations. In contrast, other models, including ResNet34 and VGG19, exhibited more significant oscillations and had a higher final loss.

Table 1
The upper rows show the average F1 scores by classes among the different models for each dataset split. The last three rows show each class's minimum, average, and maximum values of the above F1 scores.

	CONTAINERS	PLASTIC_BAG	PLASTIC_BOTTLE	TIN_CAN	UNKNOWN
Dataset 1	0.9096857	0.8866810	0.9221571	0.8381000	0.8857762
Dataset 2	0.9149333	0.8838429	0.9211333	0.8400667	0.8878667
Dataset 3	0.9113000	0.8845048	0.9229095	0.8463952	0.8793571
Dataset 4	0.9122286	0.8899238	0.9248143	0.8568619	0.8911429
Dataset 5	0.9170952	0.8897571	0.9266762	0.8479952	0.8853619
Dataset 6	0.9073667	0.8818571	0.9216952	0.8426190	0.8807143
Dataset 7	0.9033190	0.8796905	0.9174762	0.8304571	0.8754524
Dataset 8	0.9023524	0.8748476	0.9201714	0.8337190	0.8720429
Dataset 9	0.9116524	0.8879381	0.9255190	0.8490333	0.8877095
Dataset 10	0.9036905	0.8796286	0.9171095	0.8284048	0.8775714
Min	0.9023524	0.8748476	0.9171095	0.8284048	0.8720429
Avg	0.9093624	0.8838671	0.9219662	0.8413652	0.8822995
Max	0.9170952	0.8899238	0.9266762	0.8568619	0.8911429

Table 2
Ordered list of models by reduction in parameters and memory footprint. For each model: number of parameters and its corresponding size in memory in MB, with and without Transfer Learning (TL), with relative reduction percentage.

Model	Total params			Params size (MB)		
	Without TL	With TL	Reduc.	Without TL	With TL	Reduc.
SqueezeNet1.1	1.235.496	725.061	−41,31%	4,71	2,77	−41,19%
MobileNet v3 Small	2.542.856	1.522.981	−40,11%	9,70	5,81	−40,10%
MobileNet v2	3.504.872	2.230.277	−36,37%	13,37	8,51	−36,35%
ShuffleNet v2 x2.0	7.393.996	5.355.241	−27,57%	28,21	20,43	−27,58%
EfficientNet B0	5.288.548	4.013.953	−24,10%	20,17	15,31	−25,57%
MobileNet v3 Large	5.483.032	4.208.437	−23,25%	20,92	16,05	−23,28%
EfficientNet B1	7.794.184	6.519.589	−16,35%	29,73	24,87	−16,35%
ResNet50	25.557.032	23.518.277	−7,97%	97,49	89,72	−7,97%
ResNet18	11.689.512	11.179.077	−4,37%	44,59	42,64	−4,37%
VGG11	132.863.336	128.786.821	−3,07%	506,83	491,28	−3,07%
VGG16	138.357.544	134.281.029	−2,95%	527,79	512,24	−2,95%
VGG19	143.667.240	139.590.725	−2,84%	548,05	532,5	−2,84%
ResNet34	21.797.672	21.287.237	−2,34%	83,15	81,2	−2,35%

3.6. Pruning optimization

We also examined model pruning [32] as an optimization technique. Pruning involves the selective removal of the least significant weights from a neural network, thereby reducing the overall number of parameters. This approach can substantially lower resource consumption regarding memory and computational power. However, the effectiveness of pruning may vary depending on the specific use case and the hardware employed.

To identify the least significant weights, we utilized criteria such as absolute value and sensitivity to changes in output. We employed the `prune.L1Unstructured` utility in PyTorch, which prunes (unpruned) units in a tensor by zeroing out those with the lowest $L1$-norm.

In this analysis, we present the results of testing conducted on the VGG16 model to assess its impact on inference time and energy consumption. Our conclusions align with those drawn from the other models we investigated.

Fig. 4 illustrates the differences in inference energy consumption between the original and pruned models on the RTX 3060 Ti GPU platform. The blue line depicts the power consumption of the original model, while the orange line represents the pruned model. Notably, the orange line consistently demonstrates a lower average power consumption, approximately 20 W less than the original model's blue line. Specifically, the original model has an average power consumption of 130 W, whereas the pruned model consumes 110 W.

The inference process using the original VGG16 model terminated in about 11 s. In contrast, the pruned model continued processing, resulting in a longer execution time and higher energy consumption. The original model's inference consumed 1313.766 J, while the pruned model consumed 2995.512 J.

We also applied random and structured pruning techniques to the models using corresponding PyTorch utilities: `prune.RandomUnstructured`, which prunes units in a tensor at random, and `prune.LnStructured` with $n = 2$, which prunes entire tensor channels based on their L_n-norm.

However, the results were comparable to those achieved through unstructured pruning. When the pruning percentage was below 50%, we did not observe significant changes in the outcomes. Fig. 5 presents a comparison of these three techniques. These findings made us reconsider the exploration of custom pruning techniques, as energy consumption and execution time were equivalent across all three methods.

After thoroughly evaluating all the models using the three pruning techniques, we ultimately concluded that their application did not benefit our specific Edge Computing case. Although pruning techniques could induce savings in terms of memory footprint and reduce the power required to run the model, we noted a considerable increase in overall inference time, resulting in greater total energy consumption than the unpruned model. Therefore, we decided against implementing them on our HPEC platform.

4. Methodology to identify the best model

The concept of bringing computing capabilities to the edge is increasingly gaining traction due to its wide array of potential applications, including autonomous vehicles, robotics, industrial automation, and smart cities. This approach uses sensor data like radar, lidar, and cameras to control motors, navigate environments, interact with objects, and make real-time decisions. Thanks to advancements in embedded computing platforms, ML and DL algorithms can now perform data inference at the network edge.

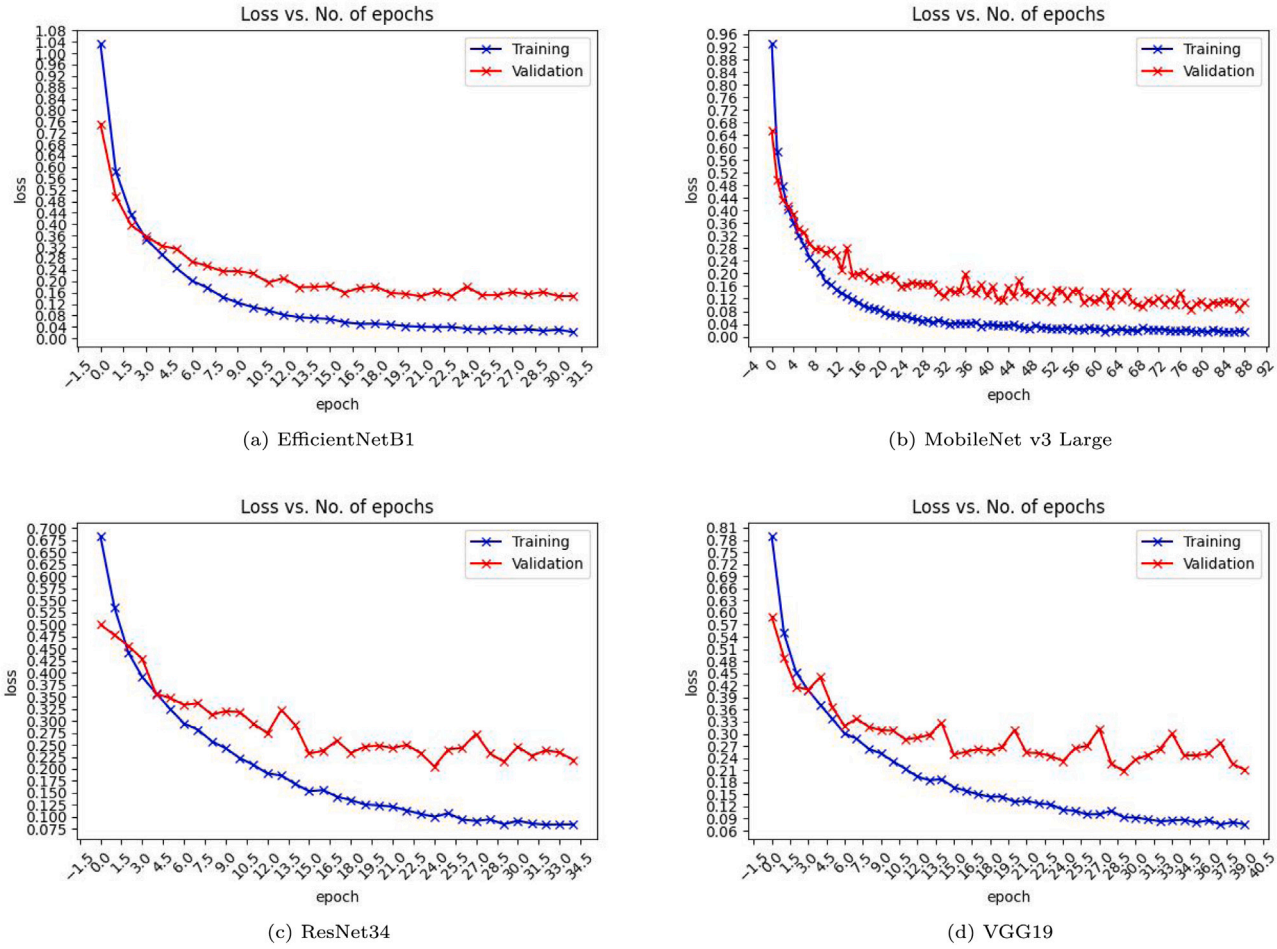

Fig. 3. Loss of EfficientNetB1, MobileNet v3 Large, ResNet34, VGG19.

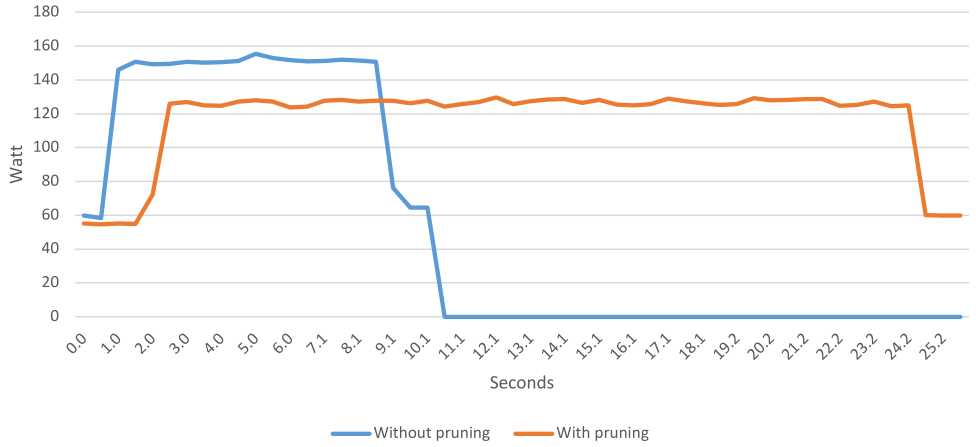

Fig. 4. Inference time and energy absorption of VGG16 model with and without unstructured $L1$-norm pruning applied to weights, running on the RTX 3060 Ti GPU platform.

We used the Jetson Nano Developer Kit developed by NVIDIA for our experiments. This kit features a quad-core CPU and a GPU with 128 CUDA cores, and it offers two power configurations: 10 W and 5 W. The default configuration is 10 W, designed for applications requiring higher processing power. This mode can execute TensorFlow Lite and PyTorch models at up to 2.5 TOPS (tera operations per second). The 5 W configuration, on the other hand, is a lower-power option intended for applications that prioritize energy efficiency. More specifications for the platform are available in Table 3, while Fig. 6 maps the software components on our testing hardware.

To identify the most effective and efficient resolution strategy within this HPEC context, we assessed the frame-per-second (FPS) inference speed, relative energy consumption, and accuracy of each model with the two different power configurations of the Jetson Nano board (Tables 4, 5).

We faced a significant challenge in selecting the optimal model due to the differing scales of the parameters we analyzed, which made direct comparisons difficult. Moreover, the measures reported did not highlight a single best-performing model on all the reference parameters. We aimed to maximize accuracy and FPS while minimizing

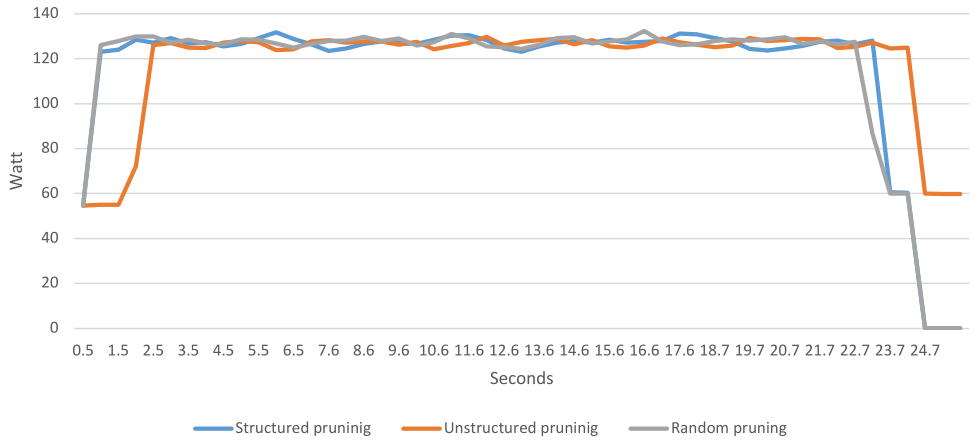

Fig. 5. Inference time and energy absorption of VGG16 model with unstructured $L1$-norm, structure $L2$-norm, and random pruning applied to weights, running on the RTX 3060 Ti GPU platform.

Table 3
NVIDIA Jetson Nano developer kit specifications.

GPU: 128-core Maxwell GPU (921.6 MHz on 10 W, 640 MHz on 5 W)	CPU:ARM Cortex-A57 CPU quad-core 1.43 GHZ on 10 W, two cores 918 MHz on 5 W
Memory:4 GB 64-bit LPDDR4	Storage:Micro SD card slot (min. 16 GB)
Power: • Micro USB (5 V 2 A) • DC jack (5 V 4 A)	Dimensions: • Module: 69.6 mm × 45 mm • Complete Kit:100 mm × 80 mm × 29 mm
Video: • Encode: 4K@30FPS (H.264/H.265) • Decode: 4K@60FPS (H.264/H.265)	Software: • Jetson Linux version 32.7.1 (JetPack 4.6.1), including: - Ubuntu 18.04.6 LTS (Kernel 4.9.253-tegra) - CUDA 10.2 (Architecture 5.3) • Python 3.6.9 • PyTorch 1.10.0 • TorchVision 0.11.0
Interfaces: • Ethernet: 10/100/1000Base-T auto-negotiation • Camera: 12-ch (3x4 OR 4x2) MIPI CSI-2 DPHY 1.1 • Display: HDMI 2.0, DP (DisplayPort) • USB: 4x USB 3.0, USB 2.0 (Micro USB) • Other: GPIO pins, I2C, I2S, SPI, UART	

Table 4
Accuracy, frames per second, average watts, and joules per image for inference with all the DL models considered for the Jetson Nano 10 W platform. Numbers highlighted in red indicate the worst performance, while those in green indicate the best performance.

Model	Accuracy	FPS	AVG Watts	Joules per IMG
efficientnet_b0	96.4654	12.5129	0.4557	0.0364
efficientnet_b1	96.1671	9.0880	0.5201	0.0572
mobilenetv2	94.2506	**17.6321**	0.3867	0.0219
mobilenetv3_large	**97.6904**	14.4485	0.3425	0.0237
mobilenetv3_small	93.9801	15.8339	**0.1766**	**0.0111**
resnet18	91.8428	11.8949	0.8983	0.0755
resnet34	90.8600	9.1039	1.6539	0.1816
resnet50	96.3145	8.5394	1.9723	0.2309
shufflenetv2	95.2808	14.2592	0.4419	0.0309
squeezenet1_1	**87.9658**	17.3234	0.2923	0.0168
vgg11	92.7273	7.5104	1.8249	0.2429
vgg16	93.5135	6.7022	2.3955	0.3574
vgg19	91.2039	**6.1815**	**2.7445**	**0.4439**

Table 5
Accuracy, frames per second, average watts, and joules per image for inference with all the DL models considered for the Jetson Nano 5 W platform. Numbers highlighted in red indicate the worst performance, while those in green indicate the best performance.

Model	Accuracy	FPS	AVG Watts	Joules per IMG
efficientnet_b0	96.5111	2.5116	0.2403	0.0957
efficientnet_b1	95.9386	**2.1680**	**0.2567**	**0.1184**
mobilenetv2	94.2998	**6.7971**	0.2317	0.0341
mobilenetv3_large	**97.4447**	5.6705	0.2433	0.0429
mobilenetv3_small	93.9558	6.0508	**0.1559**	**0.0257**
shufflenetv2	95.0369	3.4490	0.2512	0.0728
squeezenet1_1	**88.8452**	5.6832	0.1724	0.0303

the average watts and joules consumed per image. To tackle this challenge, we utilized the z-score technique and a linear transformation to normalize the values, bringing them within a range of 0 to 1.

The z-score, or the standardized score, is a statistical measure indicating how much a particular value deviates from the mean, expressed in terms of standard deviations. Suppose we have a data sample $X = x_1, x_2, \ldots, x_n$. We can calculate the z-score for an individual value x_i using the following formula:

$$z_i = \frac{x_i - \mu}{\sigma}$$

where μ represents the mean of the sample, and σ is the standard deviation of the sample. After calculating the z-scores z_i for each value, we obtain a new data distribution with a mean of zero and a standard deviation of one. We can then normalize these z-scores to get new values z_i^* in the interval $[0, 1]$ through a canonical linear transformation.

After normalizing the values of the four parameters — accuracy, FPS, average watts, and joules per image — within the interval $[0, 1]$, we adopted a different approach by transforming the problem into a maximization problem. We preserved the parameters z_i^* that required maximization and inverted those that needed minimization, resulting in $1 - z_i^*$. This method allowed us to assign weights to the parameters based on their significance.

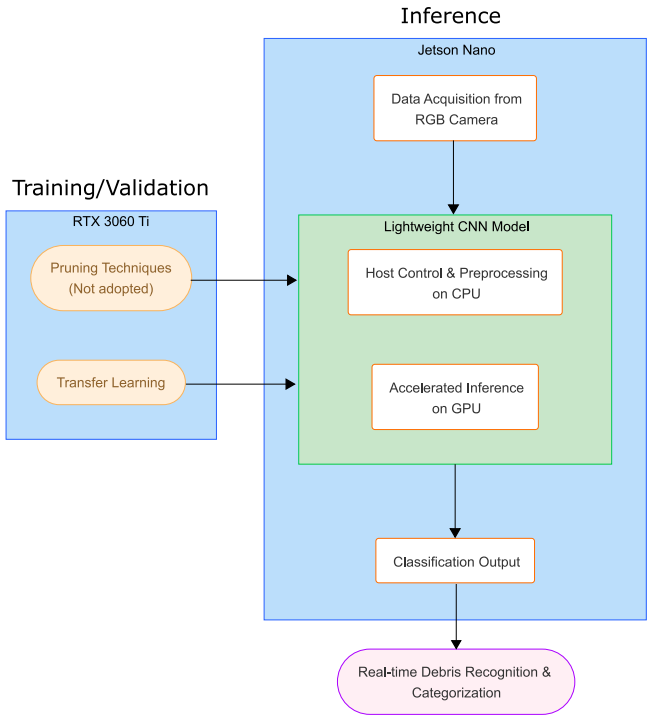

Fig. 6. Software architectural implementation chart of the proposed method.

We assigned heuristically the following weights to the parameters:

- Accuracy: 0.1
- FPS: 0.4
- Average Watts: 0.1
- Joules per image: 0.4

These weights reflect our evaluation of the relative importance of each parameter in our application context. We aim to develop a tool to continuously process a data stream on a vessel with limited energy resources. Observing that no model exhibited an accuracy below 88%, as seen in Tables 4 and 5, and considering the hardware's constraints on watt absorption, we chose to assign greater importance to the FPS and Joules-per-image parameters, viewing them as more critical than the other two. This approach ensures the application can process as many images as possible while keeping energy consumption low.

We multiplied the normalized parameter values z_i^* by their corresponding assigned weights. Next, we summed the weighted results to obtain an overall score for each model. This overall score reflects the model's performance, expressed as follows:

$$\Phi_T = w_{acc}\Phi_{acc} + w_{fps}\Phi_{fps} + w_{asm}\Phi_{asm} + w_{enf}\Phi_{enf} \quad (1)$$

where:

Φ_T : Total score
Φ_{acc} : Normalized accuracy score
Φ_{fps} : Normalized FPS score
Φ_{asm} : Normalized average power consumption score
Φ_{enf} : Normalized energy per frame score
w_{acc} : Accuracy weight
w_{fps} : FPS weight
w_{asm} : Average power consumption weight
w_{enf} : Energy per frame weight

In the following, we will rank the models based on the calculated scores and identify the model with the highest score as the best performer. This model maximizes accuracy and FPS while minimizing watt consumption and joules per image, as specified by the chosen weightings.

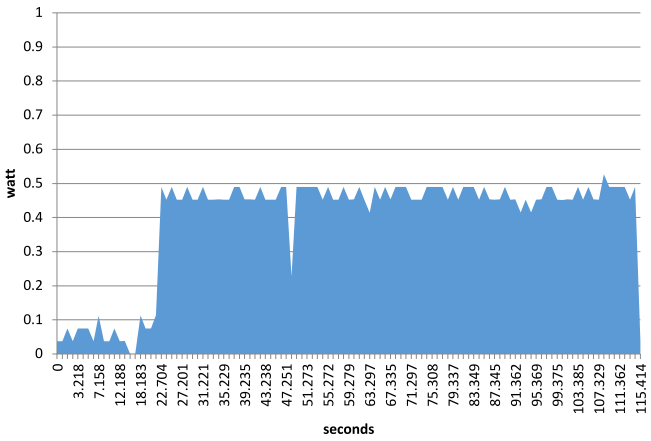

Fig. 7. Energy absorption of inferencing with MobileNetV2 on the Jetson Nano 10 W platform.

5. Experimental results and discussion

5.1. Identifying the best models on Jetson Nano (10 W)

We utilized the Jetson Nano in its 10 W absorption mode for the initial testing phase and operated it without a cooling fan. This limitation, which mirrors realistic conditions onboard vessels, resulted in overheating, prompting a reduction in power to cool the device.

The overheating issue became particularly pronounced when using neural networks with many parameters. Despite implementing the aforementioned transfer learning strategies, models such as ResNet18, ResNet34, ResNet50, VGG11, VGG16, and VGG19 exhibited signs of overheating and excessive energy consumption, as detailed in Table 4. In contrast, the Jetson Nano performed effectively with networks with fewer parameters. The table highlights the trade-offs between speed, energy consumption, and accuracy.

To estimate the energy in Joules for each inference test, we utilized the following integral:

$$E_{tot} = \int_o^t P(t)dt \quad (2)$$

In this equation, the total energy E_{tot} is expressed in Joules, while the power $P(t)$ at a specific time t is indicated in Watts.

To compute the integral, we systematically recorded the absorbed power in watts at regular intervals throughout the test, tracking the time elapsed since the last sample and the power readings from the Jetson Nano's GPU internal meter. We then applied the trapezoidal rule for numerical integration, as depicted in the graph in Fig. 7. The numbers presented in the tables reflect averages derived from ten test runs.

MobileNetV2 was the best performer when using the multi-objective function in (1) (see Table 6). Due to the Jetson Nano's hardware and power limitations, the light-weight design of this model proved highly efficient. Indeed, MobileNetV2 was designed by its authors to minimize computational operations and reduce the load on the CPU and GPU, allowing the device to perform effectively even with limited resources.

5.2. Identifying the best models on Jetson Nano (5 W)

We encountered thermal throttling issues with the ResNet18, ResNet34, ResNet50, VGG11, VGG16, and VGG19 networks when operating under the 10 power configuration. Therefore, we decided to exclude these models from further testing with the 5 W configuration. This reduced power setting curtails energy consumption by turning off two CPU cores and limiting GPU performance. Unfortunately, attempts

Table 6
Ordered list of models by total score obtained with our multi-objective function using results from inferencing on the Jetson Nano 10 W platform.

Model	Accur_Score	FPS_Score	Watts_Scores	Joules_Score	Total
mobilenetv2	0.64627	1	0.91818	0.97509	0.94648
mobilenetv3_small	0.61846	0.84296	1	1	0.89901
squeezenet1_1	0	0.97303	0.95491	0.98677	0.87941
mobilenetv3_large	1	0.72196	0.93536	0.97099	0.87071
shufflenetv2	0.75221	0.70544	0.89668	0.95416	0.82873
efficientnet_b0	0.87403	0.55292	0.89131	0.94163	0.77435
resnet18	0.39868	0.49896	0.71893	0.85128	0.65185
efficientnet_b1	0.84335	0.25382	0.86622	0.89354	0.62990
resnet34	0.29761	0.25521	0.42470	0.60604	0.41673
resnet50	0.85851	0.20591	0.30071	0.49215	0.39515
vgg11	0.48963	0.11606	0.35811	0.46439	0.31695
vgg16	0.57048	0.04547	0.13592	0.20002	0.16884
vgg19	0.33298	0	0	0	0.03329

Table 7
Ordered list of models by total score obtained with our multi-objective function using results from inferencing on the Jetson Nano 5 W platform.

Model	Accur_Score	FPS_Score	Watts_Scores	Joules_Score	Total
mobilenetv3_small	0.59429	0.83877	1	1	0.89493
mobilenetv2	0.63429	1	0.24780	0.91011	0.85225
squeezenet1_1	0	0.75935	0.83609	0.95062	0.76760
mobilenetv3_large	1	0.75661	0.13351	0.81506	0.74202
shufflenetv2	0.72001	0.27673	0.05510	0.49209	0.38504
efficientnet_b0	0.89143	0.07422	0.16256	0.24529	0.23320
efficientnet_b1	0.82486	0	0	0	0.08248

at inference with the overheating models resulted in system crashes. As a result, we shifted our focus to the remaining networks (refer to Table 5).

The Jetson Nano operating at a 5 W configuration produced unexpected results, as we anticipated its performance would align more closely with that of the 10 W configuration. However, after implementing the multi-objective function (1) (refer to Table 7), the findings indicated that MobileNetV3_small emerged as the top-performing model for this particular scenario. Contrary to our initial expectations, MobileNetV2 secured the second position among the evaluated solutions.

If we had to select a single model suitable for both 5 W and 10 W modes, MobileNetV2 would be a commendable option due to its consistently solid performance. However, in this instance, the objective was to pinpoint the optimal solution, and MobileNetV3_small delivered the best results in the 5-W modality.

5.3. Comparison on Jetson Nano - 10 W vs. 5 W

We conducted a comprehensive final test to determine which of the two optimal models is better suited for our application on a vessel. We specifically evaluated overall energy consumption along with CPU and GPU power usage. The models under comparison were MobileNetV3_small in the 5 W configuration and MobileNetV2 in the 10 W configuration. We aimed to identify the optimal combination of settings and models for enhanced performance and energy efficiency.

The results indicated that MobileNetV3_small exhibits lower average energy consumption in the 5 W configuration than MobileNetV2 in the 10 W configuration. Nevertheless, our primary aim was to identify an optimal solution by considering various evaluation parameters.

Table 8 demonstrates a significant disparity in performance between the models, particularly in terms of images processed per second and energy consumption measured in joules per image. MobileNetV2, operating at the 10 W configuration, is considerably more efficient, consuming nearly half the energy of MobileNetV3_small at the 5 W configuration. Even when we do not consider accuracy as an evaluative parameter, MobileNetV2 remains the superior choice.

We can better circumstantiate our findings by introducing the Energy Efficiency parameter, which we define as the ratio of performance achieved to the energy consumed. The following equation can represent this ratio:

$$H = \frac{frames\ per\ second}{Joules\ per\ frame} \tag{3}$$

Table 8
Accuracy, frames-per-second, average watts, joules per, and corresponding energy efficiency for image inferencing on Jetson Nano with MobileNetV2 using 10 W mode and MobileNetV3_small using 5 W mode.

Model	Accuracy	FPS	AVG Watts	Joules per IMG	H
mobilenetv2 (10 W)	94.2506	17.696	4.89614	0.27867	63.5014
mobilenetv3_small (5 W)	93.9558	6.0633	3.26589	0.53862	11.2571

As shown in Table 8, MobileNetV2 operating in 10 W mode exhibits significantly greater energy efficiency than MobileNetV3_small in 5 W mode.

Following our thorough energy consumption and performance evaluation, MobileNetV2 configured at 10 W emerges as the optimal choice for our deep learning tool. It offers an ideal balance between performance and energy efficiency, making it well-suited for resource-constrained environments.

5.4. Identifying the best models on RTX 3060 Ti platform

As an ancillary result (see Table 9), we present the measures obtained on the RTX 3060 Ti platform, as described at the end of Section 3.1. We calculated (2) using a procedure similar to the one used on the Jetson Nano. At first sight, we notice that even if efficientnet_b1 is the least absorbing model, its inference is the slowest, causing the highest energy consumption.

In this scenario (refer to Table 10), the application of the multi-objective function (1) indicated that ResNet-18 was the optimal model for the platform considered. This finding reinforces our conclusion that while MobileNets are well-suited for the HPEC platform, they do not perform as effectively in a workstation environment.

Upon comparing the energy efficiency (Eq. (3)) of the top-performing network on Jetson Nano with that of the best model on the RTX 3060 Ti platform, it is clear from Table 11 that both systems

Table 9
Accuracy, frames-per-second, average watts, and joules per image for inference with all the DL models on the RTX 3060 Ti platform, considering overall platform energy consumption. Numbers highlighted in red indicate the worst performance, while those in green indicate the best performance.

Model	Accuracy	FPS	AVG Watts	Joules per IMG
efficientnet_b0	96.1671	112.655	69.0844125	0.613238736
efficientnet_b1	94.4472	**90.171925**	**66.6786259**	**0.739461047**
mobilenetv2	94.742	139.86254	70.2919005	0.502578454
mobilenetv3_large	**98.0344**	130.89342	72.0536678	0.550475859
mobilenetv3_small	94.2998	135.32385	70.1216271	0.518176427
resnet18	92.285	270.50379	81.9444149	**0.302932596**
resnet34	92.0393	253.77229	95.1608659	0.37498525
resnet50	95.9214	119.30586	81.5743566	0.683741425
shufflenetv2	95.774	119.38285	72.7250236	0.609174817
squeezenet1_1	**89.1892**	193.18398	74.3604154	0.384920204
vgg11	92.6781	**311.87739**	134.000196	0.429656648
vgg16	93.3661	270.25232	**144.19057**	0.533540536
vgg19	91.3022	213.60344	137.899258	0.645585373

Table 10
Ordered list of models by total score obtained with our multi-objective function using results from inferencing on the RTX 3060 Ti platform.

Model	Accur_Score	FPS_Score	Watts_Scores	Joules_Score	Total
resnet18	0.34999	0.81338	0.80305	1	0.84065
vgg11	0.39440	1	0.13146	0.70970	0.73647
resnet34	0.32220	0.73791	0.63254	0.83494	0.72460
squeezenet1_1	0	0.46463	0.90089	0.81218	0.60081
vgg16	0.47222	0.81225	0	0.47172	0.56081
mobilenetv2	0.62777	0.22412	0.95338	0.54265	0.46482
mobilenetv3_large	1	0.18367	0.93065	0.43292	0.43970
mobilenetv3_small	0.57778	0.20365	0.95558	0.50691	0.43756
vgg19	0.23888	0.55673	0.08116	0.21505	0.34072
shufflenetv2	0.74444	0.13175	0.92199	0.29846	0.33873
efficientnet_b0	0.78889	0.10141	0.96896	0.28915	0.33200
resnet50	0.76111	0.13140	0.80782	0.12764	0.26051
efficientnet_b1	0.59444	0	1	0	0.15944

Table 11
Frames-per-second, joules per image, and corresponding energy efficiency for inferencing on the Jetson platform on 10 W and the RTX 3060 Ti platform, considering only GPU energy consumption.

Model	FPS	Joules per IMG	H
mobilenetv2 (10 W)	17.6322	0.021932	803.95
resnet18 (RTX)	270.504	0.302933	892.95

exhibit comparable energy efficiency. They showcase similar performance and energy consumption trade-offs, highlighting that selecting an energy-efficient model for a deep learning task is closely linked to the specific platform and characteristics of the application.

6. Conclusions

This study introduced a novel deep learning-based approach for the real-time detection and classification of floating marine debris on low-power, HPEC platforms. Unlike existing methods that depend on satellite remote sensing with limited resolution or deep learning models requiring high-power GPUs, our method enables efficient, in-situ debris recognition directly onboard marine vessels using consumer-grade edge devices by employing CNNs to classify various waste categories. To assess the effectiveness of our approach, we compared different models and optimization techniques, including transfer learning and pruning, with accuracy and efficiency as key evaluation criteria.

We also formulated a multi-objective function to identify the optimal model for each hardware configuration, considering accuracy, inference time, and energy consumption factors. Additionally, we established an energy efficiency parameter tailored to our application context and measured it across various hardware configurations to determine the most suitable settings.

Our results indicated that our approach can effectively detect and accurately classify floating objects on a desktop GPU and a low-power edge device, specifically the Jetson Nano. We observed that transfer learning and pruning are effective strategies for reducing resource consumption in terms of memory and computational power while preserving accuracy. However, it is noteworthy that pruning may lead to increased computation time. Furthermore, developing an appropriate multi-objective function enables us to select the most fitting model for diverse requirements and operating conditions.

Our research contributes to the scientific understanding of floating waste pollution monitoring by providing a versatile approach applicable to various areas, including selective waste collection, environmental awareness initiatives, and the prevention of risks to marine life and human health. Additionally, our work paves the way for the development of intelligent and autonomous systems to detect floating marine waste. Potential research directions include:

- Expanding the dataset to incorporate additional waste categories and environmental factors, such as water turbidity and the presence of white foam.
- Integrating diverse sensor modalities, including spectral reflectance and thermal infrared, to enhance the discrimination of waste materials.
- Implementing advanced algorithms for localizing and segmenting floating waste alongside classification.
- Exploring innovative applications, such as employing drones or aquatic robots to monitor and remove floating marine waste.
- Applying the design approach for energy-efficient DL inference at the edge in other application contexts.

The expansion of our dataset, the integration of various sensor modalities, and the exploration of innovative applications present significant opportunities and represent an exciting prospect for future advancements in intelligent and autonomous systems within the environmental sector and above.

We conclude this work by observing that new low-power edge computing devices are continuously emerging. Despite their architectural differences, they all share the need for a careful choice of energy-aware models and algorithms. For such a reason, while we mainly aimed this work at presenting a general and flexible method for choosing AI-based models that are accurate and energy-efficient, we will address future activities to validate and apply such an approach to additional edge computing platforms, even with different characteristics.

CRediT authorship contribution statement

Diego Romano: Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Methodology, Formal analysis, Data curation, Conceptualization. **Carlo Mennella:** Writing – review & editing, Writing – original draft, Software, Resources, Investigation, Formal analysis, Data curation. **Marco Lapegna:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Center for HPC, Big Data, and Quantum Computing (M.L. and D.R.), the PRIN-PNRR 2022 project STRUDEL: “A sustainable and trusted Transfer Learning platform for Edge Intelligence” (M.L.) in the framework of Next Generation EU program, and within the activities of D.R. as a member of the INdAM Research group GNCS and the ICAR-CNR INdAM Research Unit.

Data availability

Data will be made available on request.

References

[1] H. Ritchie, M. Roser, Plastic pollution, Our World Data (2018) URL https://ourworldindata.org/plastic-pollution.

[2] V. Martínez-Vicente, J.R. Clark, P. Corradi, S. Aliani, M. Arias, M. Bochow, G. Bonnery, M. Cole, A. Cózar, R. Donnelly, F. Echevarría, F. Galgani, S.P. Garaba, L. Goddijn-Murphy, L. Lebreton, H.A. Leslie, P.K. Lindeque, N. Maximenko, F.-R. Martin-Lauzer, D. Moller, P. Murphy, L. Palombi, V. Raimondi, J. Reisser, L. Romero, S.G. Simis, S. Sterckx, R.C. Thompson, K.N. Topouzelis, E. van Sebille, J.M. Veiga, A.D. Vethaak, Measuring marine plastic debris from space: Initial assessment of observation requirements, Remote. Sens. 11 (20) (2019) http://dx.doi.org/10.3390/rs11202443.

[3] M. Arii, M. Koiwa, Y. Aoki, Applicability of SAR to marine debris surveillance after the great east Japan earthquake, IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens. 7 (5) (2014) 1729–1744, http://dx.doi.org/10.1109/JSTARS.2014.2308550.

[4] D. Romano, M. Lapegna, V. Mele, G. Laccetti, Designing a GPU-parallel algorithm for raw SAR data compression: A focus on parallel performance estimation, Future Gener. Comput. Syst. 112 (2020) 695–708, http://dx.doi.org/10.1016/j.future.2020.06.027.

[5] A. Misra, A. Gooze, K. Watkins, M. Asad, C.A. Le Dantec, Crowdsourcing and its application to transportation data collection and management, Transp. Res. Rec. 2414 (1) (2014) 1–8, http://dx.doi.org/10.3141/2414-01.

[6] C.D. Vita, G. Mellone, D.D. Sanchez-Gallegos, G. Coviello, D. Romano, M. Lapegna, A. Ciaramella, Citizen science for the sea with information technologies: An open platform for gathering marine data and marine litter detection from leisure boat instruments, in: 2023 IEEE 19th International Conference on E-Science (E-Science), IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 1–9, http://dx.doi.org/10.1109/e-Science58273.2023.10254866.

[7] G. De Lucia, M. Lapegna, D. Romano, Unlocking the potential of edge computing for hyperspectral image classification: An efficient low-energy strategy, Future Gener. Comput. Syst. 147 (2023) 207–218, http://dx.doi.org/10.1016/j.future.2023.05.003.

[8] G. De Lucia, M. Lapegna, D. Romano, Towards explainable AI for hyperspectral image classification in edge computing environments, Comput. Electr. Eng. 103 (2022) 108381, http://dx.doi.org/10.1016/j.compeleceng.2022.108381.

[9] N. Audebert, B.L. Saux, S. Lefèvre, Semantic segmentation of earth observation data using multimodal and multi-scale deep networks, in: Asian Conference on Computer Vision, Springer, 2016, pp. 180–196, http://dx.doi.org/10.1007/978-3-319-54181-5_12.

[10] M. Volpi, D. Tuia, Dense semantic labeling of subdecimeter resolution images with convolutional neural networks, IEEE Trans. Geosci. Remote Sens. 55 (2) (2016) 881–893, http://dx.doi.org/10.1109/TGRS.2016.2616585.

[11] D. Marmanis, J.D. Wegner, S. Galliani, K. Schindler, M. Datcu, U. Stilla, Semantic segmentation of aerial images with an ensemble of CNSS, ISPRS Ann. Photogramm. Remote. Sens. Spat. Inf. Sci. 2016 3 (2016) 473–480, http://dx.doi.org/10.5194/isprs-annals-III-3-473-2016.

[12] V. Raizer, Optical Remote Sensing of Ocean Hydrodynamics, CRC Press, 2019, http://dx.doi.org/10.1201/9781351119184.

[13] M. Moshtaghi, E. Knaeps, S. Sterckx, S. Garaba, D. Meire, Spectral reflectance of marine macroplastics in the VNIR and SWIR measured in a controlled environment, Sci. Rep. 11 (1) (2021) 5436, http://dx.doi.org/10.1038/s41598-021-84867-6.

[14] P. Tasseron, T. Van Emmerik, J. Peller, L. Schreyers, L. Biermann, Advancing floating macroplastic detection from space using experimental hyperspectral imagery, Remote. Sens. 13 (12) (2021) 2335, http://dx.doi.org/10.3390/rs13122335.

[15] L. Goddijn-Murphy, B. Williamson, On thermal infrared remote sensing of plastic pollution in natural waters, Remote. Sens. 11 (18) (2019) http://dx.doi.org/10.3390/rs11182159.

[16] C. van Lieshout, K. van Oeveren, T. van Emmerik, E. Postma, Automated river plastic monitoring using deep learning and cameras, Earth Space Sci. 7 (8) (2020) e2019EA000960, http://dx.doi.org/10.1029/2019EA000960.

[17] K. Kylili, I. Kyriakides, A. Artusi, C. Hadjistassou, Identifying floating plastic marine debris using a deep learning approach, Environ. Sci. Pollut. Res. 26 (2019) 17091–17099, http://dx.doi.org/10.1007/s11356-019-05148-4.

[18] N. Prakash, O. Zielinski, AI-enhanced real-time monitoring of marine pollution: Part 1-A state-of-the-art and scoping review, Front. Mar. Sci. 12 (2025) N. 1486615, http://dx.doi.org/10.3389/fmars.2025.1486615.

[19] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W.X. Zhao, J. Zhu, Pre-trained models: Past, present and future, AI Open 2 (2021) 225–250, http://dx.doi.org/10.1016/j.aiopen.2021.08.002.

[20] R. Collobert, S. Bengio, J. Mariéthoz, Torch: A Modular Machine Learning Software Library, Idiap - Report RR 02-46, 2002.

[21] M.C. Chirodea, O.C. Novac, C.M. Novac, N. Bizon, M. Oproescu, C.E. Gordan, Comparison of tensorflow and PyTorch in convolutional neural network - based applications, in: 2021 13th International Conference on Electronics, Computers and Artificial Intelligence, ECAI, 2021, pp. 1–6, http://dx.doi.org/10.1109/ECAI52376.2021.9515098.

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520, http://dx.doi.org/10.1109/CVPR.2018.00474.

[23] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324, http://dx.doi.org/10.1109/ICCV.2019.00140.

[24] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 116–131, http://dx.doi.org/10.1007/978-3-030-01264-9_8.

[25] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size, 2016, http://dx.doi.org/10.48550/arXiv.1602.07360, arXiv preprint arXiv:1602.07360.

[26] M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114, http://dx.doi.org/10.48550/arXiv.1905.11946.

[27] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778, http://dx.doi.org/10.1109/CVPR.2016.90.

[28] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, http://dx.doi.org/10.48550/arXiv.1409.1556, arXiv preprint arXiv:1409.1556.

[29] Kili Technology, plastic_in_river dataset, 2022, URL https://kili-technology.com/data-labeling/saving-the-earth-one-annotated-plastic-bottle-at-a-time. (Accessed 30 June 2024).

[30] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks? Adv. Neural Inf. Process. Syst. 27 (2014) http://dx.doi.org/10.48550/arXiv.1411.1792.

[31] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90, http://dx.doi.org/10.1145/3065386.

[32] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2736–2744, http://dx.doi.org/10.1109/ICCV.2017.298.