Real-time unsupervised object localization on the edge for airport video surveillance*

Paula Ruiz-Barroso[1[0000−0003−0731−1074]], Francisco M. Castro[1[0000−0002−7340−4976]], and Nicolás Guil[1[0000−0003−3431−6516]]

University of Málaga, Málaga, Spain {pruizb,fcastro,nguil}@uma.es

Abstract. Object localization is vital in computer vision to solve object detection or classification problems. Typically, this task is performed on expensive GPU devices, but edge computing is gaining importance in real-time applications. In this work, we propose a real-time implementation for unsupervised object localization using a low-power device for airport video surveillance. We automatically find regions of objects in video using a region proposal network (RPN) together with an optical flow region proposal (OFRP) based on optical flow maps between frames. In addition, we study the deployment of our solution on an embedded architecture, *i.e.* a Jetson AGX Xavier, using simultaneously CPU, GPU and specific hardware accelerators. Also, three different data representations (FP32, FP16 and INT8) are employed for the RPN. Obtained results show that optimizations can improve up to 4.1x energy consumption and 2.2x execution time while maintaining good accuracy with respect to the baseline model.

Keywords: Edge Computing · Object Localization· Deep Learning

1 Introduction

The goal of *object localization* is to define the spatial extent of objects present in an image or video sequence. In this task, it is not necessary to know the class of the object since the result is usually used for object tracking where it is not important to know the kind of object. Object localization is typically carried out as the first step of the object detection problem in most object detection algorithms. Thus, an object detection algorithm can be used for object localization after discarding the class prediction step. Many different approaches have been proposed for this task [30, 9, 10, 24, 2]. Traditional approaches are based on hand-crafted descriptors that are computed and classified along the image in a sliding window setup [30, 9]. With the advent of deep learning models, the features are self-learnt by the model, which greatly boosted the performance of the proposed approaches [32, 34, 17].

* Supported by the Junta de Andalucía of Spain (P18-FR-3130 and UMA20-FEDERJA-059), the Ministry of Education of Spain (PID2019-105396RB-I00) and the University of Málaga (B1-2022_04).

However, all those approaches must be trained in a supervised way using labelled data. This requirement, together with the huge amount of data needed for training a deep learning model, limits the use of deep learning models with new classes that are not available in public datasets. Although this problem is less important for object localization since region proposal usually works fine with unknown objects, it is still a limitation for video surveillance tasks where cameras are placed at a certain distance and small objects appear in the scene. This is due to models are trained with public datasets where objects appear in the foreground thus, they are not able to localize small objects since they are not trained for that task. To solve this limitation, since we are working on video surveillance, we compute the optical flow maps between every 15 frames (since it is the frame step in the dataset) to discover areas with moving objects, allowing us to discover small objects in the scene.

Typically, the deployment of these deep learning models needs the use of high-performance architectures with discrete Graphics Processing Units (GPUs) to fulfil both inference time and energy consumption requirements. However, when these applications have to work in embedded platforms, energy consumption is a paramount issue that must also be taken into account. In these platforms, energy-efficient CPUs and accelerators are integrated on the same chip achieving a good trade-off between performance and consumption [26, 23]. Different techniques can be applied to reduce both the computational and energy consumption requirements. Thus, using pruning techniques [16] can reduce the number of arithmetic operations by removing filters and even layers of the model. Moreover, more simple data representation can be used using quantization methods [20]. Thus, instead of using 32-bit floating-point data representation, the model could employ 16-bit floating-point or even 8-bit integer data.

In this work, we propose an unsupervised approach for object localization in videos. In Fig. 1 we show a sketch of our pipeline. Firstly, we automatically find regions of objects in a sequence of frames by using the Region Proposal Network (RPN) of a pre-trained object detection model. In addition, we compute the optical flow maps between every 15 frames (it is the frame step in the dataset) to discover areas with small moving objects. Then, a non-maxima suppression step is carried out to filter and combine detections. Finally, the pipeline is optimized and deployed in an NVIDIA Jetson AGX Xavier embedded system.

Therefore, the main contributions of this work are: *(i)* a combination of two complementary region proposals specially designed for video sequences; *(ii)* an optimized pipeline for embedded systems; and, *(iii)* a thorough experimental study to validate the proposed framework and different optimizations.

In our experiments, we use videos obtained from a live RGB camera continuously recording the *apron area* (area where aeroplanes park to load passengers and luggage) of the Gdansk Airport. According to the results, our region proposal approach is robust, especially with small regions or classes with changes in shape such as persons, and can run in real-time in an embedded system.

The rest of the paper is organised as follows. We start by reviewing related work in Sec. 2. Then, Sec. 3 explains the proposed approach. Sec. 4 contains the experiments and results. Finally, we present the conclusions in Sec. 5.

2 Related Work

2.1 Objects Localization solutions

Object localization is an essential task in the field of object tracking and object classification. Gudovskiy *et al*. [12] present CFLOW-AD model, that is based on a conditional normalizing flow framework adopted for anomaly detection and localization using an encoder and generative decoders. Zimmerer *et al*. [36] complement localization with variational auto-encoders. Gong *et al*. [11] propose a two-step "clustering + localization" procedure. The clustering step provides noisy pseudo-labels for the localization step, and the localization step provides temporal co-attention models that in turn improve the clustering performance.

Moreover, some approaches are related to low-cost devices. In this area, Li *et al*. [15] propose a deep-reinforcement-learning-based unsupervised wireless-localization method. Chen *et al*. [4] present a robust WIFI-Based indoor localization, they use variational autoencoders and train a classifier that adjusts itself to newly collected unlabeled data using a joint classification-reconstruction structure. Chen *et al*. [3] show an unsupervised learning strategy to train the fingerprint roaming model using unlabeled onboard collected data which does not incur any labour costs.

2.2 Implementations on Embedded Systems

Nowadays, the use of embedded platforms to deploy deep learning applications has become more important. These systems are typically employed just for inference due to their limited computational capabilities [25, 14]. However, some authors have used them for collaborative training [29, 21]. Apply techniques such as pruning, quantization or matrix factorization, among others, that increase inference throughput on embedded platforms have led to multiple contributions. Model compression techniques can be found in surveys of the state-of-the-art [5, 7]. Following, we show some contributions in this field related to our work.

Quantization reduces the number of computations using simpler data representation. Some authors apply post-training quantization [25, 18]. Another option consists of the application of quantization during the training process [22, 35]. Others include quantization with other optimization techniques. Thus, some papers use quantization and pruning techniques separately [25]. Other works jointly use sparsity training, channel and layer pruning and quantization [19].

Furthermore, software development kits are available for optimizing deep learning models such as TensorRT or TensorFlow Lite for GPU or CPU, respectively. Focusing on TensorRT, parallelization methodology is used to maximize performance using GPU and deep learning accelerators (DLAs) [14]. Segmentation approaches and object detection networks can also be optimized using TensorRT [27, 33]. Turning our attention to TensorFlow Lite. It has two different targets: microcontrollers [6] and smartphones [1].

3 Methodology

In this section, we present our proposed method to localize small and big objects in real-time using an embedded device. Then, we describe hardware optimizations applied in this work.

3.1 Pipeline description

Our pipeline is shown in Figure 1. As input, we use one frame every 15 frames from a video since it is the frame step in the dataset. Then, the region proposal methodology is composed of two steps: (1) object localization using YOLOv4 [31] with ResNet50 [13] as the backbone and (2) object localization using region proposal based on Farneback optical flow (OF). Finally, non-maxima suppression is applied to combine objects from YOLOv4 and OF.

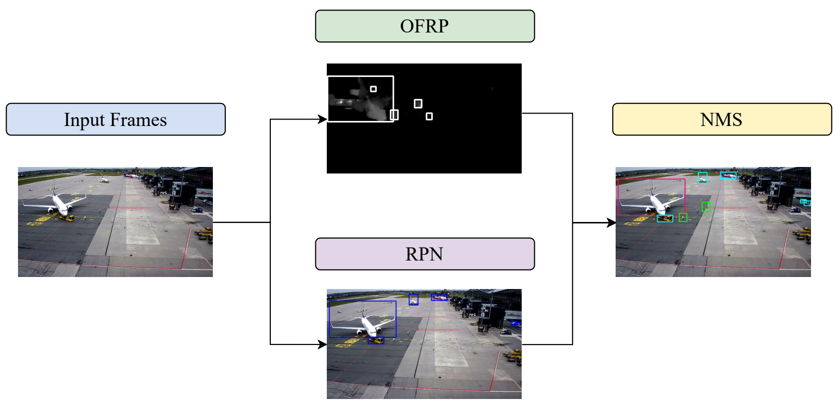

Fig. 1. Object localization method. The input is a subset of RGB video frames. Localizations are jointly obtained from both optical flow region proposal (OFRP) and region proposal network (RPN). Finally, non-maxima suppression (NMS) is applied to combine both localizations.

Input Data. Taking into account the input shape required by our RPN, the original video frames have been resized to 640 x 384 pixels. On the other hand, optical flow is calculated with a resolution of 720 x 405 pixels.

Region Proposal Network (RPN). In our approach, we use the Region Proposal Network (RPN) of a pretrained YOLOv4. It has been chosen because is one of the most widespread object detection networks nowadays and it obtains good accuracy for COCO dataset. To improve the performance of our embedded system, we have replaced the original CSPDarknet53 backbone with ResNet50 since it is better optimized for NVIDIA Jetson devices in terms of energy and time. YOLOv4 allows us to obtain the bounding box coordinates, mostly the largest objects which are common detections used for training this kind of network.

Optical Flow Region Proposal (OFRP). On the other hand, region proposal based on OF is obtained from optical flow maps using the FarneBack Dense Optical Flow technique [8]. OF maps F_t are computed every 15 frames (since it is the frame step in the dataset). To remove the optical flow noise produced by changes in the conditions of the scenario, all positions whose optical flow components (x and y) are smaller than a threshold T_F are set to 0. Then, contours containing the objects of the scene are found using the well-known algorithm proposed by Suzuki *et al*. [28]. Finally, to prevent excessively small regions, we remove those proposed regions whose area is smaller than a threshold T_A.

Non-maxima Suppression. To avoid duplicated localizations of a moving object that is localized by the RPN and the OFRP at the same time, we need to combine both localizations. To do that, we compute the Intersection over Union (IoU) metric between these two proposed regions. If the IoU is bigger than a threshold T_I and the aspect ratio of the biggest region divided by the smallest one is bigger than a threshold T_{AR} (*i.e.* the aspect ratio of both regions is similar), we keep the bounding box proposed by the RPN due to the fact that it is more accurate than the bounding box proposed by the OFRP. Moreover, we apply non-maxima suppression to each individual proposal algorithm to remove overlapped regions whose IoU is greater than the same threshold T_I.

3.2 Hardware optimizations

Hardware optimizations depend on the target device where the model is going to be deployed. In this paper, we employ an NVIDIA Jetson AGX Xavier that includes different processing units such as CPU, GPU, and DLA which can deal with different data precisions, i.e. FP32, FP16, and INT8. Thus, two different hardware-based optimizations can be applied to our platform:

Quantization: We explore, in addition to FP32 representation, FP16 and INT8 precision formats, which provide a more compact numerical representation that can reduce both the inference time and memory consumption. However, lower data precision can degrade model accuracy due to the fact that the numerical precision of the weights and activations is lower. In our case, FP16 optimization is performed with the TensorRT framework, but INT8 optimization requires an additional calibration to reduce the accuracy loss when the network is converted from FP32 to INT8. We have fine-tuned the model using PyTorch *quantization aware training*. During fine-tuning, all calculations are done in floating point, using fake quantization modules, which model the effects of quantization by clamping and rounding to simulate the effects of INT8. Then, the calibrated model is converted using TensorRT to complete the optimization process.

Concurrent computation: To accelerate our pipeline and take advantage of the different hardware resources available, we perform an efficient mapping of application modules on the available hardware. This way, our RPN and OFRP modules can be executed concurrently since the first one is mapped to the GPU and the second one to the CPU.

4 Experiments and Results

In this section, we first present the apron area dataset used in our experiments. Then, some implementation details are shown. Next, we define our pipeline evaluation, carry out hardware optimizations in our RPN, and show their effect on energy consumption and inference time. Finally, we examine the impact of our two region proposal algorithms according to their individual CorLoc metrics.

4.1 Dataset

In our experiments, we use a video dataset obtained from a live RGB camera continuously recording the apron area (the area where aeroplanes are parked, unloaded, loaded, boarded, refuelled or maintained) of the Gdansk Airport. This camera is publicly available online in a more modern version[1]. The dataset contains 96 video clips of one-minute length recorded by a FULLHD camera which provides a video stream with a resolution of 1920 x 1080 pixels and a frame rate of approximately 15 fps. In order to deal with different illumination conditions, 60% of the videos were recorded during the morning and the other 40% were recorded during the afternoon/evening. In our experiments, we consider the following categories: car ('car'), fire-truck ('ft'), fuel-truck ('fuel'), luggagetrain-manual ('lgm'), luggagetrain ('lg'), mobile-belt ('mb'), person ('pe'), plane ('pl'), pushback-truck ('pb'), stairs ('st') and van ('van'). Note that the abbreviated name of each class used in the tables is included in parentheses.

4.2 Implementation Details

As it has been indicated in the previous section, we ran our experiments on NVIDIA Jetson AGX Xavier. This platform is designed to be used as a low-power consumption system for embedded computer vision applications providing good performance. It includes an octa-core NVIDIA Carmel ARM V8.2 and a Volta GPU with 512 cores and 64 Tensor Cores accompanied by 32 GB of main memory shared by the ARM processors and the GPU. It also includes two deep learning accelerators (DLAs) and one vision accelerator (VA). Note that DLA is slower and more limited in functionality than GPU so it is expected that execution time increases using DLA and GPU with respect to performing the computation on just GPU. However, DLA consumes less energy than GPU.

Therefore, inference processes have been deployed on GPU or DLA+GPU using TensorRT 8.4.1 and cuDNN 8.4.1. Original YOLOv4 with ResNet50 as backbone has been trained based on Sacaled-YOLOv4 [31] code[2] on a computer with a discrete GPU using PyTorch 1.11. On the other hand, OF has been calculated using an OpenCV library implemented on CPU. Moreover, non-maxima suppression and preprocessing before introducing frames into OFRP or RPN have been calculated on CPU too.

Regarding parameters commented in Section 3, after a cross-validation process on a subset of the training data, we have established the following values: $T_F = 0.7$, $T_A = 100$, $T_I = 0.4$, $T_{AR} = 0.5$.

4.3 Performance Evaluation

On the one side, we report energy consumption and execution time for each implementation. Energy consumption is provided by internal sensors incorporated on Jetson boards. We employ a 500 Hz sampling frequency which is the

[1] http://www.airport.gdansk.pl/airport/kamery-internetowe
[2] https://github.com/WongKinYiu/ScaledYOLOv4

maximum value supported by the sensors. Also, the average values for 800 forward steps after 5 warm-up predictions are provided. We use the Energy Delay Product (EDP) metric, which is the product of energy and execution time, to measure hardware performance. Thus, the lower the EDP value, the better.

On the other side, we use the *Correct Localization* metric (CorLoc) to evaluate the precision for region localization. This metric is defined as the percentage of objects correctly localized according to the Pascal criterion: the IoU between the predicted region and the ground-truth region is bigger than a given threshold for the RPN and bigger than another given threshold for the OFRP.

4.4 Experimental Results

Baseline experiment. In this section, we show our baseline experiment without any optimization, *i.e.* the pipeline has been deployed with PyTorch on GPU using FP32 data representation. The obtained results in terms of energy and time can be observed in Table 1. The pipeline is divided into four stages: frames preprocessing ('Prep.') before introducing them into the next step, region proposal network ('RPN'), optical flow region proposal ('OFRP') and finally, non-maxima suppression ('NMS'). As we can see, the most time and energy consuming part is the RPN. The RPN using PyTorch is a bottleneck for the parallel OFRP+RPN process. On the other hand, energy consumption in preprocessing and NMS is almost negligible while inference time represents a 7.17% of the final time. Finally, our complete pipeline takes 94.25 ms with an energy consumption of 16.72 mJ without considering any hardware optimization.

Table 1. Baseline study: Time and energy. Each row represents a different stage: preprocessing (Prep.), parallel region proposal algorithms: region proposal network (RPN) and optical flow region proposal (OFRP) and non-maxima suppression (NMS). Columns represent inference time and energy consumption together with EDP (Energy Delay Product) value. The last row represents the total time, energy and EDP of our baseline method. Note that for the total time of the baseline, we only consider the highest time of the parallel region proposal part (i.e. 87.49 ms from RPN).

Stage		Time (ms)	Energy (mJ)		EDP
			GPU	CPU	
Prep.		3.32	-	0.05	0.16
Parallel	RPN	87.49	12.68	-	1109.23
	OFRP	73.01	-	3.98	290.89
NMS		3.44	-	0.01	0.03
Total		94.25	16.72		1400.32

Data quantization on GPU. In Table 2, we observe energy consumption, inference time and EDP for the RPN after the optimization carries out by TensorRT for different data precisions. The last column shows the EDP reduction ratio (EDP_{rr}) for FP16 and INT8 with respect to FP32 precision. As preprocessing and NMS are executed on the CPU, they have been omitted.

First of all, we compare baseline inference time and energy consumption (third row in Table 1) with FP32 optimization using TensorRT. It can be observed that there is a decrease in inference time of 19.35 ms and a reduction of 4.29 mJ in energy consumption.

Focusing on FP16 data precision, we can see a significative reduction (around 2×) of the computation time for FP16 w.r.t FP32, as it was expected. Energy consumption has also improved due to both the shorter inference time and the use of more compact computing units. Nevertheless, INT8 precision has an unexpected behaviour as it obtains a longer execution time than that achieved for FP16. We analysed the executed kernels with NVIDIA Profiler[3] for INT8 quantization and we discovered that TensorRT employs not only INT8 kernels but also FP32, probably because there is not INT8 implementation for some RPN layers. Thus, the execution improvement is lower than expected since mixed precisions are used together.

Comparing EDP_{rr} among different data precision, there is a great improvement when using FP16. However, the enhancement is not as significant when using INT8 due to the fact that also FP32 is being used as previously discussed.

Finally, we consider the RPN with FP16 data precision the best option due to lower energy consumption and inference time. Thus, the total energy consumption for our approach using FP16 quantization is 6.07 mJ and the total time is 79.77 ms taking into account preprocessing, OFRP+RPN and NMS values. In this case, as the OFRP takes longer than the RPN, it is a bottleneck for the parallel OFRP+RPN processing. In terms of frames per second (FPS), taking into account that during 79.77 ms we are processing 15 video frames, we are able to process 188 FPS.

Table 2. Hardware optimizations: Data quantization. Each row shows different data precision. Columns represent inference time and energy consumption together with EDP (Energy Delay Product) value for GPU. Finally, EDP_{rr} (Energy Delay Product reduction ratio) with respect to the FP32 model is included in the last column.

Quantization	Time (ms)	Energy (mJ)	EDP	EDP_{rr}
FP32	68.14	8.39	571.38	-
FP16	30.90	2.03	62.79	**9.10x**
INT8	52.91	5.52	292.06	1.96x

Deep Learning accelerator. Another hardware optimization available on Jetson Xavier is the use of both DLA and GPU to perform model inference using TensorRT. However, it has limitations in the amount and kind of layers supported. Our RPN is too large to be completely mapped on DLA. In this case, TensorRT implementation reaches an internal state which indicates that DLA has exceeded the number of layers in the network that it can handle, so the remaining layers are mapped in the GPU. Moreover, other models cannot be entirely mapped on DLA due to layer incompatibilities[4].

[3] NVIDIA Nsight Compute documentation can be consulted: https://developer.nvidia.com/nsight-compute

[4] NVIDIA TensorRT documentation can be consulted to find mapping incompatibilities: https://docs.nvidia.com/deeplearning/tensorrt/archives/tensorrt-841/developer-guide/index.html

On the other hand, DLA data precision is currently limited to networks running in FP16 and INT8 modes. In our case, INT8 is not available due to previously commented layers incompatibilities with this data precision. Therefore we have only deployed our model on DLA using FP16. The results using DLA and GPU can be observed in the fourth row of Table 3. If we compare inference time and energy consumption with its GPU analogues in FP16 (third row in Table 3), DLA makes the inference process much slower, which increases the energy consumption with respect to GPU only. Thus, due to its lower computational capacity and data transfer rate between GPU and DLA, this combination also consumes more energy and time. However, it allows freeing GPU resources which could be employed for other computations if required.

Comparing EDP reduction ratios (EDP_{rr}) among baseline models without TensorRT optimizations and FP16 models using TensorRT, we can observe significant reduction ratios when using TensorRT and FP16 data precision. For the FP16 model on GPU, it achieves a value of 17.67x with respect to the baseline model and a value of 2.18x for GPU and DLA implementation.

Table 3. Hardware optimizations: DLA study. Columns represent inference time and energy consumption for DLA and GPU together with EDP (Energy Delay Product) for RPN. Finally, EDP_{rr} (Energy Delay Product reduction ratio) with respect to the baseline model is included in the last column. Each row represents a different model: baseline model without optimizations, FP16 model on GPU and FP16 model on DLA and GPU.

Model	Time (ms)	Energy (mJ)			EDP	EDP_{rr}
		GPU	DLA	Total		
Baseline	87.49	12.68	-	12.68	1109.23	-
FP16 on GPU	30.90	2.03	-	2.03	62.79	**17.67x**
FP16 on DLA - GPU	71.41	4.35	2.78	7.13	509.46	2.18x

Region proposal comparative. In this experiment, we evaluate the performance of each region proposal algorithm (RPN and OFRP) using the CorLoc metric, comparing the annotated ground-truth with the proposals obtained from each approach. Furthermore, we evaluate the performance of each one based on the category object to be detected.

In Table 4, we show the CorLoc results (where higher is better) for our two proposal algorithms. Concerning the RPN algorithm, we have four combinations: FP32, FP16, INT8 using GPU and FP16 using GPU with DLA. Finally, the combination of both has been added. The standard value of 0.5 is used as IoU threshold for the RPN algorithm and 0.1 is used for OFRP. OFRP requires a smaller threshold because Farneback optical flow is computed every 15 frames (since it is the frame step in the dataset), which generates large flow vectors which make the bounding boxes too big in comparison to the ground-truth bounding boxes. With regard to the results, RPN produces more accurate regions than OFRP due to OF is only able to detect moving objects. The RPN that uses INT8 data precision achieves the best results, thus this RPN together with OFRP is the best combination. Note that this improvement in the results for INT8 is because we have fine-tuned the model using PyTorch *quantization aware training*, as commented previously, during four epochs improving the accuracy of the model. The other combinations obtain almost the same CorLoc value.

Table 4. CorLoc results for different data quantization. Each row shows the CorLoc results using only RPN (second column), only OFRP (third column) or both RPN+OFRP (last column) for different data quantization.

Quantization	RPN	OFRP	RPN + OFRP
FP32 GPU	19.60	3.92	23.52
FP16 GPU	19.58	3.87	23.45
INT8 GPU	21.46	3.87	**25.33**
FP16 GPU + DLA	19.58	3.93	23.51

In addition, we measure the CorLoc metric over the true positive set of each class. We obtain the CorLoc metric separately (second and third rows) in Table 5) for each algorithm and each class. Finally, the last row in Table 5 shows the results using both region proposal algorithms together. For brevity, we only show the results per class for the best RPN algorithm, i.e. using INT8 data precision and GPU. It can be observed that RPN proposes better regions for all classes because most of the objects remain stationary, thus, there is no optical flow in this situation. The major contribution of the optical flow is for the person class as very small bounding boxes are generated and sometimes may not be detected by the RPN algorithm. Finally, the OFRP algorithm increases the CorLoc metric of 6 classes and allows us to obtain the best results in our pipeline.

Table 5. CorLoc results per class using true positives. Each row represents a different proposal algorithm and each column represents a different class.

Precision	Car	ft	Fuel	lgm	lg	mb	pe	pl	pb	st	Van
OFRP	8.5	8.0	5.5	2.3	12.0	9.2	36.3	1.4	7.4	37.8	86.7
RPN INT8	**67.6**	97.8	**97.8**	31.2	**15.2**	56.3	37.9	**98.1**	13.3	66.5	**93.3**
OFRP+RPN INT8	**67.6**	**97.9**	**97.8**	**31.3**	**15.2**	**56.7**	**39.7**	**98.1**	**14.0**	**67.1**	**93.3**

5 Conclusions

We have proposed a real-time unsupervised object localization approach using a low-power device for airport video surveillance. Our method has two principal components which produce the bounding boxes: a region proposal network (RPN) and an optical flow region proposal (OFRP). We have evaluated our pipeline on video sequences of the *apron area* of an airport showing that our approach is able to localize objects that appear on videos. Moreover, hardware optimizations allow us to improve energy consumption and inference time in order to obtain real-time localization in addition to low consumption. Regarding the CorLoc metric, it can be observed that the combination of OFRP and RPN improves the results obtained using only one of them. Furthermore, with regard to hardware optimizations, we have demonstrated that the use of different data precisions i.e. FP16 or INT8 or the employment of GPU together with DLA can achieve less energy consumption and inference time than the baseline pipeline.

As future work, we plan to use these localizations to obtain descriptors and group them using a clustering algorithm in order to find similar objects. Finally, we want to label these clusters and a human may optionally revise some samples from each cluster.

References

1. Ahmed, S., Bons, M.: Edge computed nilm: a phone-based implementation using mobilenet compressed by tensorflow lite. In: NILM 2020. pp. 44–48
2. Cai, Z., Vasconcelos, N.: Cascade R-CNN: Delving into high quality object detection. In: CVPR. pp. 6154–6162 (2018)
3. Chen, M., Liu, K., Ma, J., Zeng, X., Dong, Z., Tong, G., Liu, C.: Moloc: Unsupervised fingerprint roaming for device-free indoor localization in a mobile ship environment. ieee internet of things journal **7**(12), 11851–11862 (2020)
4. Chen, X., Li, H., Zhou, C., Liu, X., Wu, D., Dudek, G.: Fidora: Robust wifi-based indoor localization via unsupervised domain adaptation. IEEE Internet of Things Journal **9**(12), 9872–9888 (2022)
5. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. arXiv preprint:1710.09282 (2017)
6. David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Wang, T., et al.: Tensorflow lite micro: Embedded machine learning for tinyml systems. PMLR **3**, 800–811 (2021)
7. Deng, L., Li, G., Han, S., Shi, L., Xie, Y.: Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE **108**(4), 485–532 (2020)
8. Farnebäck, G.: Two-frame motion estimation based on polynomial expansion. In: Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13. pp. 363–370. Springer (2003)
9. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. IEEE PAMI **32**(9), 1627–1645 (2010)
10. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR. pp. 580–587 (2014)
11. Gong, G., Wang, X., Mu, Y., Tian, Q.: Learning temporal co-attention models for unsupervised video action localization. In: CVPR. pp. 9819–9828 (2020)
12. Gudovskiy, D., Ishizaka, S., Kozuka, K.: Cflow-ad: Real-time unsupervised anomaly detection with localization via conditional normalizing flows. In: WACV 2022. pp. 98–107
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016)
14. Jeong, E., Kim, J., Tan, S., Lee, J., Ha, S.: Deep learning inference parallelization on heterogeneous processors with tensorrt. IEEE Embed. Syst. Lett. **14**(1), 15–18 (2021)
15. Li, Y., Hu, X., Zhuang, Y., Gao, Z., Zhang, P., El-Sheimy, N.: Deep reinforcement learning (drl): Another perspective for unsupervised wireless localization. ieee internet of things journal **7**(7), 6279–6287 (2019)
16. Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing **461**, 370–403 (2021)
17. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. In: ICCV. pp. 10012–10022 (2021)

18. Liu, Z., Wang, Y., Han, K., Zhang, W., Ma, S., Gao, W.: Post-training quantization for vision transformer. NeurIPS **34**, 28092–28103 (2021)
19. Ma, X., Ji, K., Xiong, B., Zhang, L., Feng, S., Kuang, G.: Light-yolov4: An edge-device oriented target detection method for remote sensing images. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens **14**, 10808–10820 (2021)
20. Mathew, M., Desappan, K., Kumar Swami, P., Nagori, S.: Sparse, quantized, full frame cnn for low power embedded devices. In: CVPR (July 2017)
21. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
22. Park, E., Yoo, S., Vajda, P.: Value-aware quantization for training and inference of neural networks. In: ECCV. pp. 580–595 (2018)
23. Qasaimeh, M., Denolf, K., Khodamoradi, A., Blott, M., Lo, J., Halder, L., Vissers, K., Zambreno, J., Jones, P.H.: Benchmarking vision kernels and neural network inference accelerators on embedded platforms. Journal of Systems Architecture **113**, 101896 (2021)
24. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. pp. 91–99 (2015)
25. Ruiz-Barroso, P., Castro, F.M., Delgado-Escaño, R., Ramos-Cózar, J., Guil, N.: High performance inference of gait recognition models on embedded systems. Sustain. Comput. Informatics Syst. **36**, 100814 (2022)
26. Saddik, A., Latif, R., Elhoseny, M., Elouardi, A.: Real-time evaluation of different indexes in precision agriculture using a heterogeneous embedded system. Sustain. Comput. Informatics Syst. **30**, 100506 (2021)
27. Seichter, D., Köhler, M., Lewandowski, B., Wengefeld, T., Gross, H.M.: Efficient rgb-d semantic segmentation for indoor scene analysis. In: ICRA. pp. 13525–13531. IEEE (2021)
28. Suzuki, S., et al.: Topological structural analysis of digitized binary images by border following. CVGIP **30**(1), 32–46 (1985)
29. Tao, Z., Li, Q.: esgd: Commutation efficient distributed deep learning on the edge. HotEdge p. 6 (2018)
30. Viola, P., Jones, M., et al.: Rapid object detection using a boosted cascade of simple features. CVPR **1**, 511–518 (2001)
31. Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Scaled-yolov4: Scaling cross stage partial network. In: CVPR. pp. 13029–13038 (2021)
32. Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv: 2207.02696 (2022)
33. Xia, X., Li, J., Wu, J., Wang, X., Wang, M., Xiao, X., Zheng, M., Wang, R.: Trt-vit: Tensorrt-oriented vision transformer. arXiv preprint:2205.09579 (2022)
34. Zhai, X., Kolesnikov, A., Houlsby, N., Beyer, L.: Scaling vision transformers. In: CVPR. pp. 12104–12113 (2022)
35. Zhao, K., Huang, S., Pan, P., Li, Y., Zhang, Y., Gu, Z., Xu, Y.: Distribution adaptive int8 quantization for training cnns. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 3483–3491 (2021)
36. Zimmerer, D., Isensee, F., Petersen, J., Kohl, S., Maier-Hein, K.: Unsupervised anomaly localization using variational auto-encoders. In: MICCAI 2019. pp. 289–297. Springer