Enhancing embedded AI-based object detection using multi-view approach

Zijie Ning, Mostafa Rizk, Amer Baghdadi, Jean-Philippe Diguet

Enhancing embedded AI-based object detection using multi-view approach

Z. Ning[†], M. Rizk[†§], A. Baghdadi[†] and J-Ph. Diguet[‡]
[†]*IMT Atlantique, Lab-STICC UMR CNRS 6285, Brest, France*
[‡]*CNRS, IRL CROSSING, Adelaide, Australia*
[§] *Lebanese International University, CCE Department, Lebanon*
zijie.ning@imt-atlantique.net

***Abstract*—Object detection based on convolutional neural network (CNN) is widely used in multitude emergent applications. Yet, the deployment of CNNs on embedded devices at the edge with reduced resources and power budget poses a real challenge. In this paper, we address this issue by enhancing the detection performance without impacting the inference speed. We investigate the use of multi-view for the same scene to achieve better detection performance. A novel system of distributed smart cameras is proposed where each camera integrates a CNN for detection. Implementation results show that using light networks on the distributed cameras can lead to better detection performance and a reduction in the overall consumed power.**

***Index Terms*—Multi-view, Object detection, Embedded edge device**

I. Introduction

Object detection is widely used in several modern application fields such as autonomous vehicles, medical imaging, crowd detection, exploration, etc. Currently, artificial intelligence (AI) is the key trend to detect objects in images and videos. CNN architectures allow the detection of multiple objects in a scene, yet at a cost of high computational resources and power consumption. This fact creates a challenge when deploying CNNs on embedded devices with limited resources and power budget especially for applications that require real-time inference. Other parameters such as the size and weight of the computational platform as well as its power consumption are critical for trendy applications such as wearable devices, drones and autonomous robots.

Mainly, compared to light neural networks, dense networks lead to higher detection performance in terms of accuracy and precision but at the cost of inference speed. Often, systems integrating CNN-based detection methods have a trade-off between inference rate and detection performance. Several techniques have been recently proposed to optimize the trained networks towards embedded devices. These techniques are highly dependent on the target hardware, and usually produce degraded detection performance. In addition, the training cost in terms of power and utilized resources increases with the size of the target network.

In this work we address this issue with a totally new approach that makes use of multi-view of the same scene to increase the detection accuracy. We propose a system of distributed nodes, where each node integrates a camera and an embedded device that performs inference at the edge using light CNNs. The distributed nodes collaborate in order to achieve best predictions with reduced overall power consumption. Fig. 1 illustrates the proposed system for the application of detecting ships. The distributed nodes record the same object with different angles of view so that each one has its own prediction. Then, the predictions are all gathered and the final decision is taken. Note that in this case the nodes share only the detection information rather than the captured images. Hence, privacy, security and transmission bandwidth are conserved.

979-8-3503-9851-9/22/$31.00 ©2022 IEEE

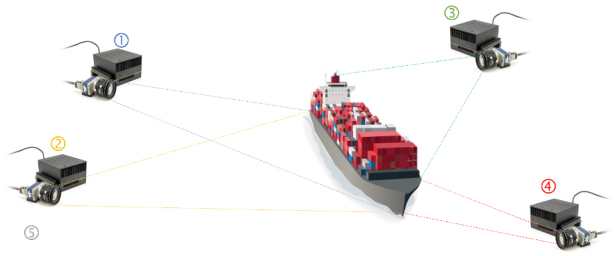


Fig. 1. Boat detected using a multi-view network

In addition to enhancing the detection, using multi-view concept during inference helps in avoiding occlusion, which can deteriorate the predictions. For example, as shown in Fig. 2, the pedestrian is hidden by a car. This generally brings a loss in the prediction confidence score. Multi-view can enhance the training of the network. Training only on one representation of an object can bring a bias, which damages the generality of the classifier. For instance, if the classifier is trained only on images of the front side of an object, it will be difficult for it to recognize the object from images of the backside. Therefore, the addition of this multi-view feature allows the classifier to train on vast various representations of an object, which brings a true added value.

II. Related Work

Few research works have addressed object detection and classification in the context of multi-view. Arsalan Mousavian et al. [1] have proposed a method to construct 3D bounding boxes based on a single image and geometric relationships of

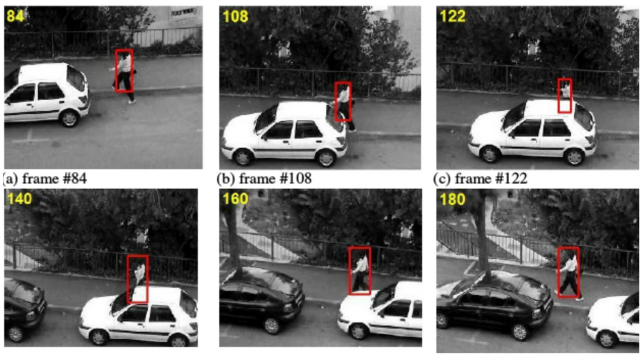


Fig. 2. Pedestrian hidden by a car

(a) book (b) shoe

Fig. 3. Example of objects

the object, which can perform 3D object detection and pose estimation from a single image. Their main contribution is limited to the prediction of a 3D bounding box from a 2D one. The method proposed in [2] has been able to accurately estimate 3D bounding boxes for even small objects, using both mature 2D object detectors and advanced 3D deep learning for object localization. The authors in [3] have proposed a framework for multi-camera 3D object detection called DETR3D, which extracts 2D features from multiple camera images and then uses a transformation matrix to relate 3D positions to multi-view images. Recently, the work proposed in [4] has used multi-view approach during training on a skier's pose segmentation system, to address the lack of large datasets.

To the best of our knowledge, no published work has used a multi-view approach in the detection phase, and no one has investigated the enhancement of multi-view approach on the performance of object detection system.

III. Methods

A. Available multi-view datasets

In order to apply our proposed idea of using multiple views, we need a dataset that includes several images for the same scene, which are captured from different points of view. As a starting point, we have investigated the published datasets.

The well known COCO (Common Objects in Context) [5] is a dataset of 80 common objects provided by Microsoft and has been widely used as a baseline for object detection problems. The amount of data in this dataset is sufficient for a general target detection system training task. However, each object in this dataset usually appears only once per scene and there is no information about multiple views.

Several research works have already provided multi-perspective datasets. The multi-view car dataset provided by EPFL [6] includes 2000 images of 20 cars with different positions and angles of view. However, all the images have the same background and constant light intensity. In our context above, this amount of data is not big enough for the training. In addition, the stated images' identical properties tend to cause overfitting of the model.

Another dataset, ELKI Multi-View Clustering Data Sets [7], from ALOI (Amsterdam Library of Object Images) contains 1000 objects and 100 photos for each object. The variables in this dataset contain different angles with different light as well as the RGB information. However, this dataset uses only black background which has the same default as the previous one. They have only one object for each class, so they may not be capable of making good decisions on other objects of the same class.

The PASMVS dataset [8] is generated with scripts, which do not include images, but the code and materials used to reproduce 18,000 scenes and backgrounds. It has only four classes, including imaginary animals which are not usable in real-life detection.

There are also other multi-view datasets that we don't mention here as they are not convenient in our case. Therefore, we decided to create our own dataset.

B. Creation of multi-view dataset

To build a multi-view dataset, we firstly define 7 classes to be included: mug, shoe, keyboard, wallet, pencil case, book, and mouse. For each class, we define its coordinate system to express the object's orientation by yaw, pitch, and roll. The mugs have their centers as the origin, the handle as the X-axis, the Z-axis facing outward, and the left-handed system to establish the corresponding Y-axis. The origin of the shoe is defined at the heel, with the X-axis along the body of the shoe. We define the short side of the keyboard as the Y-axis and the long side as the X-axis. When the opening of the wallet is on the right side, the coordinate origin of the wallet is defined as the vertex in the lower left corner. Along the bottom edge to the right is the X-axis and up is the Y-axis. For the pencil case, the origin is defined as the position of the zipper lock when it is closed, and the Y-axis is along the long axis. We define the long side of the book as the Y-axis and the short side as the X-axis. The definition of the mouse axis is somewhat special, with the scroll wheel as the origin, when placed forward, pointing to the left for the Y-axis, pointing forward for the X-axis.

We choose three different rooms and fix several cameras at different locations. Different scenes are generated by changing the types, angles, and positions of the placed objects and changing the lighting conditions of the rooms. We pre-define more than ten positions in each room where objects could be placed, and use a randomized method to generate a table containing the placement of objects corresponding to the scenes. Then we manually enter the room to place the items

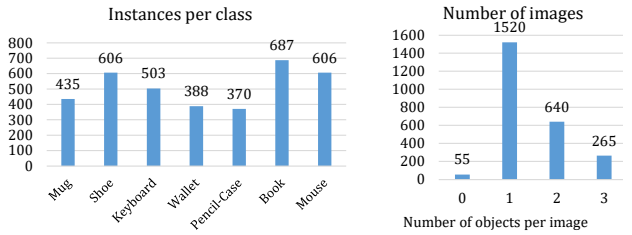


Fig. 4. Statistic of the constructed dataset

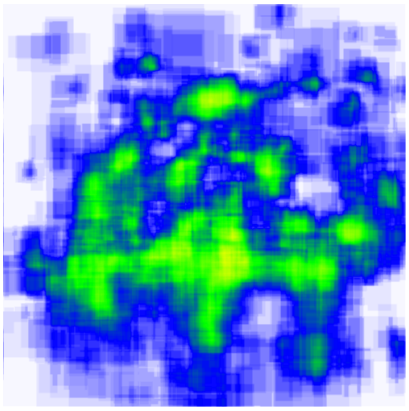


Fig. 5. Heat map of object position (high occurrence in green, lower in blue)

according to the generated table and then use a program to control all cameras to take pictures at the same time and store the images as png files. After the shooting has been completed, we manually annotate them using CVAT [9]. At the same time, blurring is used to erase parts of the images that might affect the learning process, such as the shoes worn by the person taking the picture.

Fig. 4 shows the number of times each class appears in the dataset and the statistics of the number of objects in each image. Most of the images (1520) have only one object, others (640) have two objects, a few (265) have three objects, and a very few (55) have no objects and are used as background images for training.

In most cases the objects are placed at different positions on the ground, whereas sometimes we use objects outside our defined classes (e.g. cardboard boxes) to increase the height of the object positions. Fig 5 shows the heat map of the object positions. Most of the objects are placed in the center of the view (green area). The blue color corresponds to the area with less number of objects.

Fig. 6 shows some example images from the dataset.

C. Baseline experiment

YOLO [10] is a state-of-the-art CNN-based model for object detection, with great advantages in accuracy and speed. It is the acronym of "You Only Look Once", which means that you only need to browse the image once to recognize the class and location of the objects in the picture. It also offers pre-trained models on the COCO dataset. YOLOv5 [11] is an extension of the YOLO series, and we can also consider it as an improved work based on YOLOv3 [12] and YOLOv4 [13]. Although there is no corresponding paper description for YOLOv5, the authors are actively working on this open source project and have gained a lot of experience and advantages in practice together with other users. The model uses the PyTorch architecture, which is very extensible and convenient. Five versions are given in the official YOLOv5 code, namely YOLOv5n(nano), YOLOv5s(small), YOLOv5m(medium), YOLOv5l(large), and YOLOv5x(extreme). These different variants represent a good trade-off between accuracy and speed, making YOLOv5 convenient for users to implement.

We select a number of scenes in the approximate ratio of 3:5:4 based on the amount of data in the three rooms, then choose the images recorded by all cameras in these scenes as the test set so that the number of images in the test set is about 20% of the total. Another 80 scenes containing 8 cameras are selected from the total dataset, and these images are taken out and kept as the test set for the Camera position experiment that will be presented later. The remaining scenes are divided into training and validation sets according to the ratio 3:1, and all images of the same scene always belong to the same set.

In order to improve the training efficiency, we have conducted research on hyperparameters by doing a sweep using Wandb [14]. For each set of hyperparameters, 5 training cycles have been conducted to observe the training effect. Then we have discarded the unreasonable and undesirable hyperparameter combinations and selected the ones with good training effects to continue the training for 300 complete training cycles.

After the training is completed, we select the best-performing weights for each network size during the training process and test the model performance on the test set.

All trainings are done with a computer equipped with NVIDIA GeForce GTX 1060 6GB. All subsequent experiments were performed on the same machine.

D. Multi-view experiment

In this experiment, we want to verify that multiple cameras from different views working together can improve prediction accuracy. Unlike the previous experiment, we now ask the system to give predictions for one scene at a time, instead of predicting each image individually. In our example, each scene contains four images seen by four cameras at different locations. Therefore, we run four programs simultaneously, each using the model trained in the previous experiment, with four images of different viewpoint of the same scene as input. Each program gives all the bounding boxes above the set threshold and their corresponding prediction categories and confidence, and then the system considers these predictions together to give the final prediction.

Since each camera observes from a different angles of view, we first transform all the bounding boxes to the same plane as shown in Fig. 7(a), using Homography Transformation [15]. This is a matrix transformation that allows us to project images obtained from cameras into the same coordinate system. This approach has been widely used in the field of computer vision.

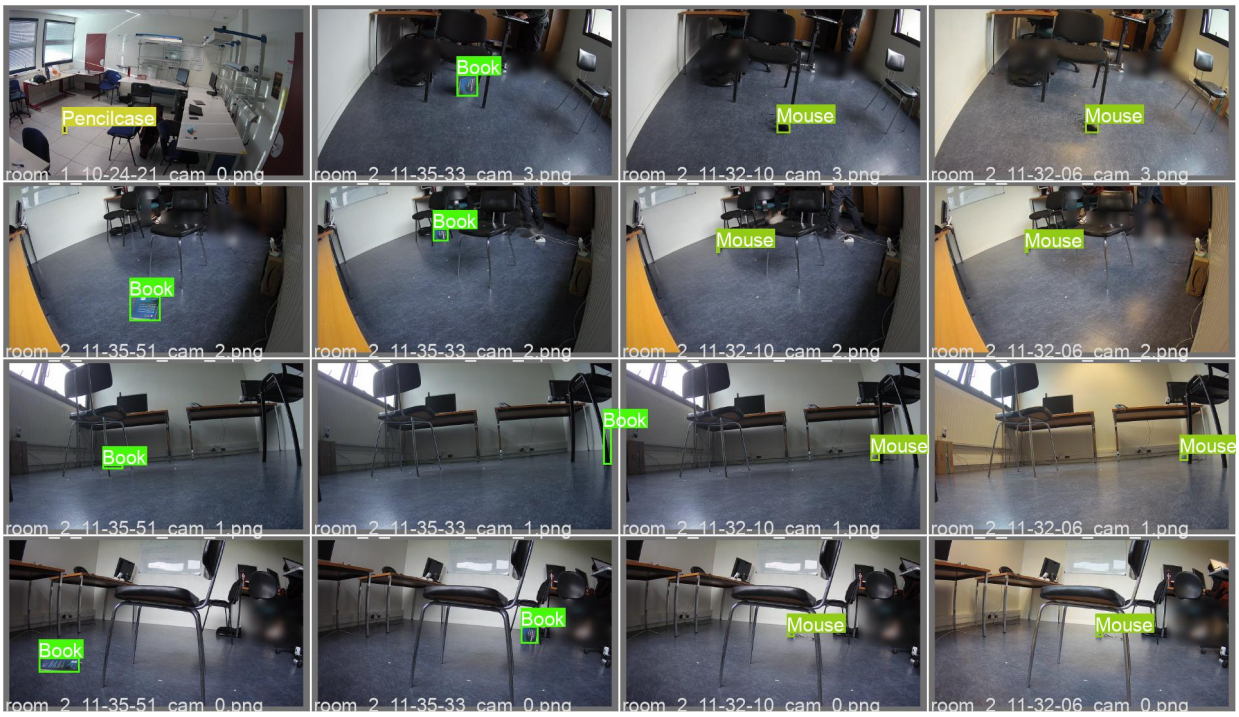


Fig. 6. Examples from the constructed dataset

Algorithm 1 Algorithm for merging bounding boxes and predictions

Input: Some combinations of bounding box and prediction from different nodes

Remove the bounding boxes with confidence $<$ threshold$_1$

repeat

  B = untreated bounding box with the highest confidence

  for bounding box B' **in** untreated bounding boxes **do**

    Calculate IOU of B and B'

    if IOU $>$ threshold$_2$ **then**

      Remove the bounding box of B'

      Remove B' from the list of untreated bounding boxes

      Add the predictions of B' to B

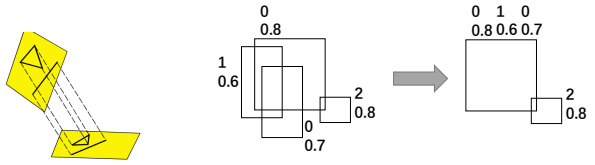
    end if

  end for

until No untreated bounding box left

Output: Merged bounding boxes with its corresponding predictions

For each object, each camera may provide a bounding box, so the object will have multiple bounding boxes. We want to merge the bounding boxes so that all predictions for one object can be treated together. Inspired by the NMS (non-maximum suppression) algorithm that has been used in the YOLO [10], we propose a similar method as described in Algorithm 1. Each subsystem outputs all bounding boxes above the threshold, projecting them onto the plane where the object is located. Then, the merge algorithm is applied to these projected bounding boxes together. If some of these boxes are merged, then their corresponding predictions are also merged and treated together in later process. Fig. 7(b) shows the bounding boxes before and after merge.



(a) Homography transform (b) Bounding boxes before and after merge

Fig. 7. Pre-processing of node output results

All the predictions from the nodes are pre-processed to obtain some bounding boxes, each of which also contains several predicted categories and corresponding confidence scores.

Now we can focus on verifying that multi-view can improve the accuracy of prediction. For that, we consider first a simple situation, in which there is only one object within each scene and each camera sees only this one object, i.e., all the predicted bounding boxes belong to this only object. For each scene, we first compare the prediction of each node with the true value to get the accuracy of the prediction for the images. Then we combine the predictions of all nodes to give the final prediction and compare this prediction with the true value to get the accuracy of the prediction for the scene.

We currently propose several methods to give the final

TABLE I
PREDICTION EXAMPLE OF A 7 CAMERAS SYSTEM

	cam0	cam1	cam2	cam3	cam4	cam5	cam6
Prediction	0	0	1	1	2	2	2
Confidence	0.8	0.2	0.7	0.5	0.5	0.4	0.4

prediction: the maximum value method, the average method, and the summation method. The maximum value method means that the prediction with the highest confidence is directly selected as the final prediction. The average method and the summation method average or sum the confidence of all predictions of the same class, respectively, and then select the largest one as the final prediction. In practice, these three methods may yield different final predictions, for example, in the 7 cameras system example shown in the Table I, using the three methods will yield 0, 1, 2 as the prediction respectively.

The overall process[1] of this multi-view experiment is shown in Fig. 8.

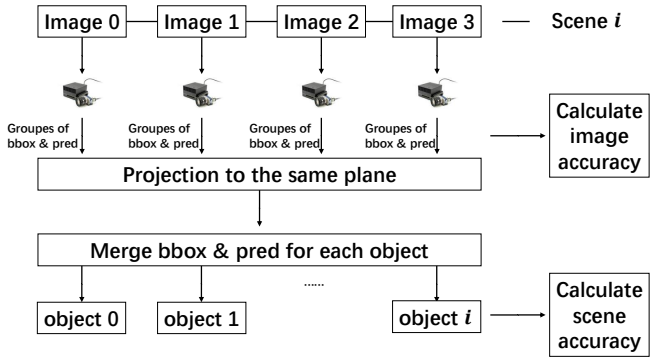


Fig. 8. Process of multi-view experiment

E. Camera position experiment

This experiment is designed to investigate the effect of camera placement on accuracy in the multi-view approach. We want to demonstrate that the camera placement affects the correct prediction rate. In other words, it is the multiple views that improve the correct prediction rate, not simply the increase in the number of cameras that brings the improvement.

We use scenes with 8 cameras as the test set for this experiment, which have been selected when segmenting the dataset before training and have not been used as the training or validation set. The relative relationship between the 8 cameras and the zone of objects is shown in Fig. 9. Four of the cameras are placed on the same table as the objects, and the other four cameras are placed higher than the table. For each scene, we select images taken by four of the eight cameras as input, give the final output using the same method as in the multi-view experiment, and then repeat the process by replacing the selected camera combination. Finally, we compare the prediction accuracy of the different camera combinations.

[1]Code published at https://github.com/mm0806son/Enhance_embedded_object_detection_system_using_multi_view

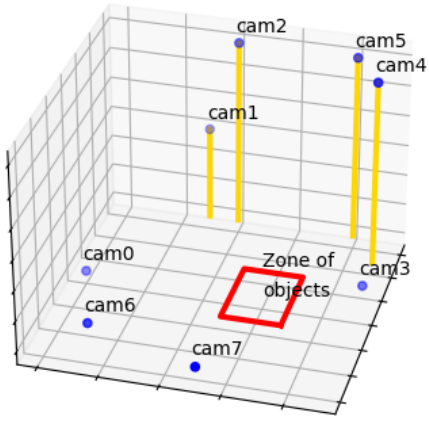


Fig. 9. Cameras' position

IV. RESULTS

A. Baseline experiment

In the baseline experiment, we verify the training and inference of different size models of YOLOv5 and record their accuracy on the multi-view dataset we produced. The results are considered as a reference for subsequent experiments. At the same time, we confirm that using smaller size models is more advantageous in terms of energy consumption and memory requirements, and thus more suitable for embedded systems.

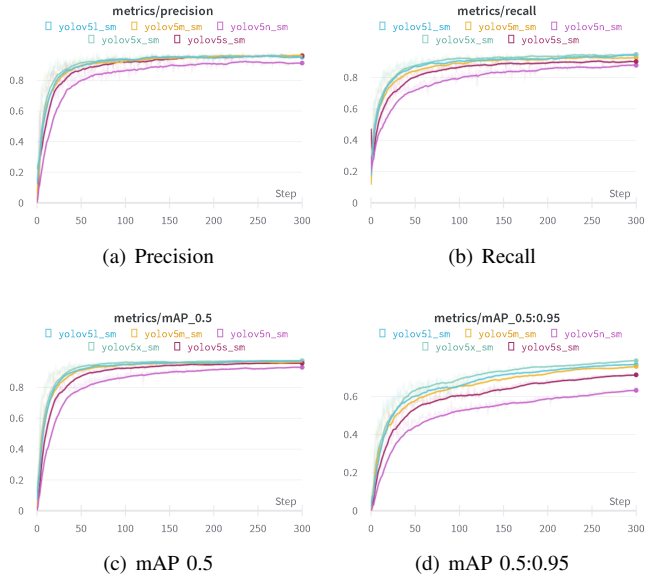
We have trained 300 epochs on the training set for 5 models of different sizes and then used the validation set to evaluate them. The size of the models and their performance on the validation set are shown in Table II[2]. In general, the larger is the model, the better is the performance, yet the training and inference time is longer. The evaluation metrics are shown in Fig. 10.

TABLE II
RESULTS OF BASELINE EXPERIMENT

Model	Params (M)	FLOPs @640 (B)	Speed (ms)	mAP_0.5	mAP_0.5:0.95
YOLOv5x	86.7	205.7	43.7	0.9760	0.7913
YOLOv5l	46.5	109.1	23.7	0.9730	0.7707
YOLOv5m	21.2	49.0	13.9	0.9681	0.7618
YOLOv5s	7.2	16.5	5.6	0.9583	0.7164
YOLOv5n	1.9	4.5	2.6	0.9349	0.6394

The GPU usage during training is shown in Fig. 11. For the three smaller models (i.e., medium, small, nano) we have used 16 as the batch size. For the two larger models (i.e., extreme, large) we have used 4 as the batch size due to the GPU memory limitation. The training of the larger models consumes 1.5× to 2× the energy required to train the smaller models and takes up a large amount of memory, placing higher demands on the GPU. Smaller models are also faster to train and require less computing power from the hardware.

[2]Visualised results can be found at https://wandb.ai/smart_cam/Enhance_embedded_object_detection_system_using_multi_view?workspace=user-mm0806son



(a) Precision (b) Recall

(c) mAP 0.5 (d) mAP 0.5:0.95

Fig. 10. Results of baseline experiment

B. Multi-view experiment

In the multi-view experiment, we use each model separately for image prediction, and then different strategies for scene prediction. The results of the experiment are shown in Table III.

All three methods described in Section III-D to give the final multi-view prediction have achieved good results, greatly improving the correctness of inference and enabling small models to outperform large ones. In the baseline experiment, we investigate the energy consumption (Fig. 11(a)), GPU memory requirements (Fig. 11(b)) of different models. We used 4 nano (YOLOv5n) models for scene prediction, and their total number of parameters and computation is about the same as 1 small model, yet they perform better. These 4 nano models can even outperform 1 extreme (YOLOv5x) model, while having only one-tenth of its number of parameters and executes 20× faster. It is also noticed from the Fig. 11(a) that the larger models (i.e. x,l) consume about twice as much energy as the smaller models when their batch size is a quarter of that of the smaller models (i.e. s, n). Therefore, under our hardware conditions, using 4 small models saves about half the energy. Using the multi-view system approach allows to achieve better detection performance while requiring much less computational power.

TABLE III
RESULTS OF MULTI-VIEW EXPERIMENT

Model	Prediction by image	Prediction by scene		
		Max	Average	Sum
YOLOv5x	0.821	0.917	0.917	0.933
YOLOv5l	0.835	0.933	0.917	0.933
YOLOv5m	0.798	0.917	0.908	0.925
YOLOv5s	0.792	0.908	0.900	0.833
YOLOv5n	0.767	0.925	0.933	0.900

TABLE IV
SYSTEM ACCURACY OF CAMERA POSITION EXPERIMENT

Model	Cams	Prediction by image	Prediction by scene		
			Max	Average	Sum
YOLOv5x	**0,1,3,7**	0.635	0.763	0.775	0.763
	1,2,4,5		0.763	0.775	0.763
YOLOv5l	**0,1,3,7**	0.641	0.813	0.788	0.763
	1,2,4,5		0.813	0.813	0.800
YOLOv5m	**0,1,3,7**	0.598	0.733	0.725	0.750
	1,2,4,5		0.763	0.763	0.763
YOLOv5s	**0,1,3,7**	0.575	0.688	0.675	0.663
	1,2,4,5		0.725	0.713	0.700
YOLOv5n	**0,1,3,7**	0.528	0.713	0.713	0.738
	1,2,4,5		0.588	0.575	0.600

C. Camera position experiment

Regarding this third experiment, we choose the cameras 0,1,3,7 as the first group and the cameras 1,2,4,5 as the second group. The first group of cameras is able to observe around the object, while that of the second group gather on one side of the object. The accuracy of each model for detection alone on images and for detection using multi-view approach on scenes is shown in Table IV.

When using the smallest nano network, we find that camera placement has a significant impact on performance. Regardless of which of the three strategies is used, the accuracy of the two groups of cameras differs by about 0.13. When using the other larger networks, the performance of the two groups of cameras fluctuates, with no one group being able to significantly outperform the other, but both outperforming the predictions made for each image individually.

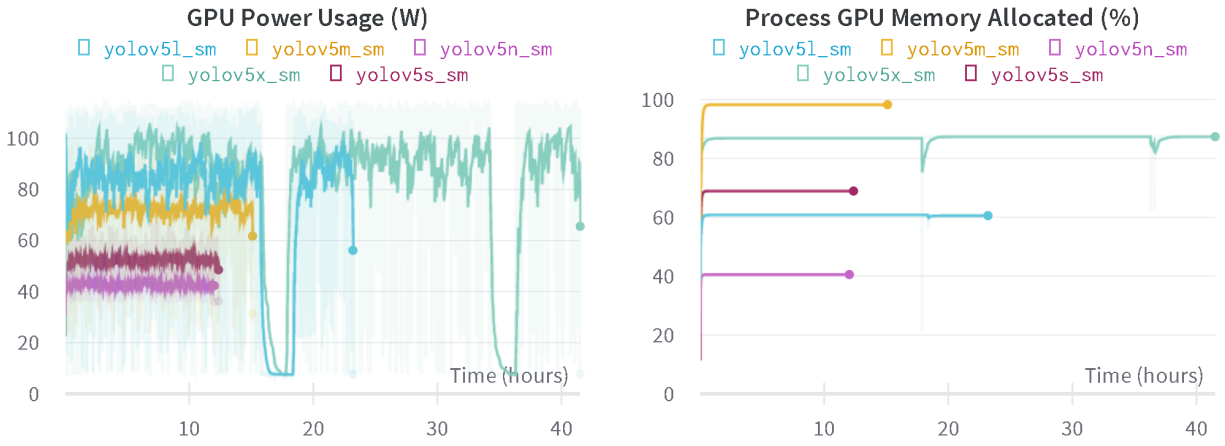
It can also be noted from the results that until the YOLOv5l model, using a more complex model helps significantly for the second group of cameras, as the large model has a 30% improvement in accuracy over the nano model. This is very likely due to the fact that the images observed by these cameras did not have enough details to enable a very small network to perform accurate decisions.

The results show that the improvement in performance by changing the position can be comparable to the improvement by using a larger network. If the cameras can be placed in the right position to capture more details at the time of observation, its prediction results will be better. This makes a lot of sense for saving energy and computational power.

V. CONCLUSION

In this paper, we present the idea of using multiple subsystems with different views to work together to improve the overall accuracy of an object detection system. To that end, we have designed and collected a multi-viewpoint dataset on our own, containing several common objects in an office environment. For each scene in this dataset, we have multiple images from different angles of view to validate multi-view related experiments.

Using this dataset, we have conducted three experiments: baseline, multi-view, and camera position. The performance, energy consumption, and computational power requirements of different models of YOLOv5 have been measured. Smaller



(a) GPU power usage

(b) GPU memory usage

Fig. 11. Results of baseline experiment

YOLOv5 models have less demand and faster training and inference, yet perform relatively worse. In the multi-view experiments, we have presented the complete flow of how the multi-view object detection system works. The predictions of the nodes are first preprocessed, and then the final prediction of the scene is obtained using three simple strategies: max, average, and sum. We found that a distributed system using several small models has been able to greatly improve detection performance, outperforming large models. This approach requires less computational power and is much faster. In the camera placement experiments, we have found that the location of the cameras can have a significant impact on system performance, especially for small networks. A good placement of the cameras can achieve an enhancement effect similar to the use of large models.

Overall, our work demonstrates that using multi-view nodes to form a global object detection system can save computational power and increase speed. This has great interest for models deployed on edge and embedded systems. Sometimes people also want to build prototypes first with few hardware resources. This approach can help them get reliable predictions before committing more resources. In future work, we can also take the orientation of the object into consideration to pursue further performance enhancement, since the observed images from different views have a correspondence of orientation.

ACKNOWLEDGMENT

This work was supported in part by the Regional Council of Bretagne through the ODESSA FEDER project. The multi-view dataset[3] used in this paper was built by Aziz Sellami, Yuming Wang, Sebastian Heyer and Zijie Ning. They worked together to design the collection scheme and completed the tasks of item placement, photography, and labeling.

[3]https://www.kaggle.com/datasets/zijiening/smart-cam-multiview

REFERENCES

[1] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," 2016. [Online]. Available: https://arxiv.org/abs/1612.00496

[2] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," 2017. [Online]. Available: https://arxiv.org/abs/1711.08488

[3] Y. Wang, V. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. Solomon, "Detr3d: 3d object detection from multi-view images via 3d-to-2d queries," 2021. [Online]. Available: https://arxiv.org/abs/2110.06922

[4] I. Katircioglu, H. Rhodin, J. Spörri, M. Salzmann, and P. Fua, "Human detection and segmentation via multi-view consensus," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 2855–2864.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie et al., "Microsoft coco: Common objects in context," 2014. [Online]. Available: https://arxiv.org/abs/1405.0312

[6] M. Ozuysal, V. Lepetit, and P. Fua, "Pose estimation for category specific multiview object localization," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 778–785.

[7] E. Schubert and A. Zimek, "ELKI Multi-View Clustering Data Sets Based on the Amsterdam Library of Object Images (ALOI)," Jun. 2010. [Online]. Available: https://doi.org/10.5281/zenodo.6355684

[8] P. J. Broekman, André; Gräbe, "Pasmvs: a dataset for multi-view stereopsis training and reconstruction applications," 2020.

[9] Boris Sekachev, Nikita Manovich, Maxim Zhiltsov et al., "Opencv/cvat: v1.1.0," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4009388

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015. [Online]. Available: https://arxiv.org/abs/1506.02640

[11] Glenn Jocher, Alex Stoken, Jirka Borovec et al., "Ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements," Oct. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4154370

[12] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[13] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[14] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

[15] E. Dubrofsky, "Homography estimation," 2009.