International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 25s,2025
https://theaspd.com/index.php

Embedded Artificial Neural Networks (Embedded-ANN) And YOLO-Edge For Embedded Systems

Halil Hüseyin Çalışkan [1*], **Talha Koruk** [2]
[1] Bursa Technical University, Faculty of Engineering and Natural Sciences, Department of Computer Engineering, Bursa, Türkiye. Email: caliskanhalil815@gmail.com
[2] Bursa Technical University, Faculty of Engineering and Natural Sciences, Department of Computer Engineering, Bursa, Türkiye. Email: talha.koruk@btu.edu.tr

Abstract: *In embedded systems, image classification with artificial neural networks and object detection with YOLO cause both excessive memory usage and low frame rate per second. In this study, in order to reduce these problems encountered in embedded systems, artificial neural networks were rewritten only with NumPy without using any deep learning library. The results of this study are that this artificial neural network model developed with NumPy specifically for embedded systems is approximately 265 times faster than artificial neural network models created with TensorFlow having the same neural network architecture and consumes 6 times less memory. In addition, the fact that these developed artificial neural networks operate at high speed using only the processor without using a graphics card in embedded systems shows the feature of this neural network model to work independently of hardware. Moreover, YOLOv8's architecture has been modified specifically for embedded systems and named as YOLO-Edge. According to results, YOLO-Edge is 4 times faster than YOLOv8m on Nvidia Jetson Xavier Nx and 8 times faster on Rockchip 3588 (NPU).*
Keywords: *Artificial Neural Networks, Embedded Systems, Image Classification, YOLO.*

1. INTRODUCTION

Embedded systems have various problems in implementing artificial intelligence models due to their limited processing power [1]. Since artificial neural network models created with TensorFlow require high computational power; It can be inefficient to run on embedded systems such as Raspberry Pi 5, Nvidia Jetson Xavier Nx, Orange Pi 5 Plus. Therefore, the need for high-speed and low-latency artificial neural network models optimized for embedded systems is increasing [2], [4]. In this study, an artificial neural network model that can work efficiently in embedded systems has the advantage of low power consumption and high speed and is created with NumPy. The matrix multiplications, transpose operations of matrices, and derivative operations of activation functions of this artificial neural network model developed for embedded systems are performed with NumPy. In addition, the matrices in the model are quantized and the Adam algorithm is rewritten using NumPy. All these added features allow the model to get an average of 1860 FPS in image classification when running only with the CPU without using a GPU or NPU in embedded systems [5]. Thus, image classification can be performed efficiently using only a standard CPU without the need for any GPU or NPU. The model's high FPS by working only with the CPU highlights the model's ability to work independently of hardware. On the other hand, standard YOLO models consist of too many layers and contain too many parameters, which can lead serious problems to occur in terms of speed and memory usage in embedded systems [3], [10]. Some architectural modifications have been made in the YOLOv8's architecture to cope with the difficulties encountered in the integration of standard YOLOv8 models to embedded systems [22], [24], [25]. By adding DWConv and GhostConv to the backbone section, the backbone section has been made capable of operating with high performance in embedded systems. In the neck section, the model has been provided to detect smaller objects better with UpSample operations. The model has been made lighter by adding LightConv to the head section. After all these optimization studies, the new YOLO model was named YOLO-Edge. Furthermore, the YOLO-Edge model uses 42% less GPU memory than YOLOv8m, according to tests conducted on RTX 3060, emphasizing that the model can run well on embedded systems with low GPU memory space [26], [27], [28], [29].

2. LITERATURE REVIEW

Problems such as high memory consumption and low frames per second caused by artificial intelligence models make it difficult to integrate such software into embedded systems with limited resources. In recent years, many researches have been conducted worldwide to reduce these problems encountered in the

integration of artificial intelligence models into embedded systems. In this review, the studies and their results are examined. In the following paper, resource-efficient neural networks for embedded systems are discussed. The effects of reducing neural network weights from 32-bit floating point values to lower bit levels are discussed in terms of reducing memory usage and increasing processing speed. It is explained that the model can be lightened by removing some unnecessary weights or layers after training [2]. One of the literature reviews, the classification of sensor data in small embedded systems with artificial neural networks, shows that we can successfully process sensor data using artificial neural networks in low-power microcontrollers. The best accuracy rate was obtained with a two-layer FFNN containing 1493 parameters; this model took 36 ms to run and FFNNs gave the most successful results with an accuracy rate of 83%. Thanks to this research, the integration of artificial neural networks in embedded systems has been further paved [41]. The following work aims to detect objects at different scales using optimized versions of YOLOv8. By removing some unnecessary layers in YOLOv8, a YOLO model that operates with lower energy consumption and lower computational power has been developed. The model optimized for small objects achieved a major improvement by reducing the model size from 6.3 MB to 1.4 MB. Additionally, the modified model runs faster than the original YOLOv8 model [12]. In the following paper, by developing two different modifications of the YOLOv8n model, feature extraction and prediction performance have been improved. By modifying the architecture of YOLOv8 and using layers such as GhostConv, faster and lighter YOLO models have been developed [13]. In the following work, various changes have been made to the architecture of YOLOv3. In addition to the standard 3 detection layers of YOLOv3, 2 new YOLO detection layers have been added. SPP network has been added to extract a uniform feature map for inputs of different sizes. Faster optimization has been achieved by using the tangent loss function instead of cross entropy. As a result, the newly added two YOLO layers and SPP module did not affect the speed of the model while improving the detection accuracy. The tangent function helped the model to provide better generalization performance by reducing the training time. The model achieved a speed of 60 fps and became suitable for real-time applications [23].

3. METHODS

Embedded Artificial Neural Networks (Embedded-ANN)

In this study, images in .jpg, .png and .jpeg formats that are intended to be trained with artificial neural networks are read with OpenCV, the pixel value of each image is divided by 255 and normalized and brought to the appropriate format. All these images brought to the appropriate format with OpenCV are put into a Python list and finally converted to a NumPy array [8]. Meanwhile, the class equivalent of each image is kept in another NumPy array. This NumPy array consisting of class numbers is also converted to a unit matrix and brought to the appropriate format to be used in updating the weights in backpropagation. In the artificial neural network model, instead of randomly initializing each weight, it is normalized with the He initialization method and created in matrix form. In forward propagation, the bias value to be added to each weight is created in matrix format depending on the number of neurons in each layer [9], [10], [11]. The schematic representation of artificial neural networks created with NumPy is shown in Fig. 1. According to this scheme, the matrix multiplication operations are performed in hidden layers. With the help of these hidden layers, activation functions enable the model to learn non-linear relationships instead of learning linear relationships.

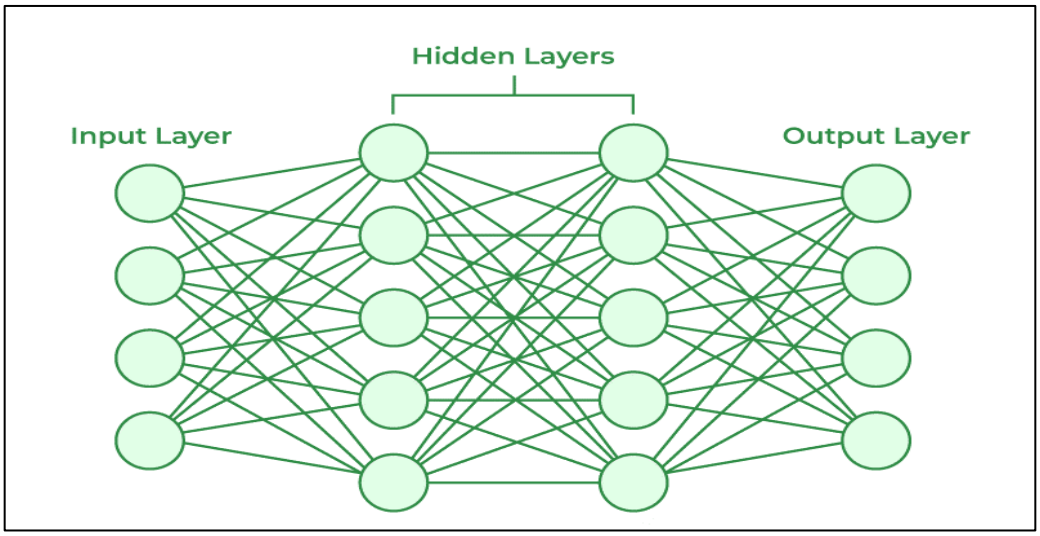

Fig. 1. The schematic representation of artificial neural networks

Images consisting of lists in NumPy format come to the neural network model as hints and matrix multiplication is performed with the first weight value. After this multiplication, the bias value is added. This output is passed to the ReLU activation function [7]. This resulting value is also matrix multiplied with the new weights and bias values are added. Then it is passed to the ReLU activation function again. In the last layer of forward propagation, each visual classified using Softmax instead of ReLU is found to have a class corresponding to it and the probability value of the class it belongs to. The matrix operations in forward propagation are shown in Figure 3. The reason for using ReLU in this neural network is that, unlike activation functions such as Silu and Tanh, it is linear and therefore faster than other activation functions [21]. In addition, another reason for using ReLU as the activation function is that it reduces the vanishing gradient problem, which is the problem of not being able to update the weights of the model as the gradients approach 0 while being transmitted to deeper layers during the training of the model. Briefly, ReLU is both a linear activation function and reduces the vanishing gradient problem, which has enabled ReLU to be used in Embedded-ANN. The images and label data of the model are divided into two as train and validation. During the training of train data, instead of sending all the data to forward propagation collectively in each epoch, the train data is divided into mini-batches and sent to each mini-batch for forward propagation. In this way, the model updates its weights better and learns better as a result. The probability values of the classes formed in the matrix form as a result of the Softmax function found in the last layer of forward propagation are sent to backpropagation in each epoch [6]. These sent values are extracted from the unit matrices containing the label information, normalized and brought to the form of the error matrix. This error is transferred to all layers with the help of the chain rule and the gradients of the weights are calculated by taking the derivative of the activation function affecting that weight and multiplying the matrix with the error matrix. The calculated weight and bias gradients provide the formation of new weight and bias values with the Adam optimizer algorithm. The beta1 value in the Adam optimizer algorithm is used to calculate the moving average of the gradients, while the beta2 value calculates the moving average of the square of the gradients [36], [37]. Gradients may show sudden changes at the beginning of the training, beta1 balances these changes and allows the gradient to take into account its past values. The beta2 value in the Adam optimizer algorithm provides better weight updates by scaling large and small gradient values. At the beginning of the training, the momentum values of all weight and bias values are brought into appropriate matrix formats. As a result, the gradients of the weights, the gradients of the biases, beta1, beta2 and momentum values provide the formation of our updated weight and bias matrices. The type of each weight matrix in the artificial neural network model created specifically for embedded systems is float64. Quantizing these weight matrices from float64 to float16 allows for less memory space to be taken up in embedded systems with limited resources [38], [39], [40]. Thus, these matrices converted to float16 take up 4 times less space in RAM compared to float64. The reason for using float16 instead of int8 in Embedded-ANN is that the weight matrices resulting from the He initialization are between 0 and 1, and if int8 is used in quantization, the weight matrices will only be integers, which causes a serious loss in the accuracy of the model. The areas occupied by data types in float type in

RAM are presented in Table I. The artificial neural network model developed specifically for embedded systems is evaluated according to validation data. In each epoch, validation data is forward propagation and the matrix containing the validation results formed after forward propagation is compared with the matrix consisting of labels in the form of a unit matrix. This comparison returns the validation accuracy value. In each epoch, the validation accuracy value of the model is calculated and if the validation accuracy value is greater than the previous epochs, the weights of that model are recorded. Thus, the weights that reach the best validation accuracy value as a result of training are saved in NumPy format and become available for use in the testing phase. In addition, the fact that these weights created by NumPy can be used in any embedded system without any problems shows that the model is independent of the hardware. This neural network model, developed specifically for embedded systems, has been tested on 8 different datasets. Thus, more extensive information has been provided about the validation accuracy values of the model according to different datasets. Information about the datasets used is shown in Table II.

Table I. Data types and their memory usages

Type	Memory Usage
Float16	4
Float32	8
Float64	16
Float128	32
Float256	64

Table II. Properties of the utilized datasets

Dataset Type	Size of Training Dataset	Size of Validation Dataset	Number of Claseses	Resolutio
MNIST	60.000	10.000	10	28×28
Fashion MNIST	60.000	10.000	10	28×28
Cifar-10	60.000	10.000	10	32×32
Face Mask	10.800	992	2	160×160
Brain Tumor	2.934	711	2	224×224
Coffe Bean	1.200	400	4	96×96
Rock-Paper-Scissors	4.487	418	3	32×32
Fruits	347	92	2	224×224

YOLO-Edge for Embedded Systems

YOLO (You Only Look Once) is a deep learning-based object detection algorithm used for object detection [42], [45], [46], [47]. YOLO's basic architecture consists of three parts: backbone, neck and head [44], [48], [49], [50]. Backbone is a deep convolutional neural network architecture used to extract features such as edge, shape and texture from the input image. This layer learns the edge, texture and other object-specific distinguishing features required for object detection by generating a variety of feature maps as a result of filtering the image. In short, the backbone part of YOLO models processes the image gradually by performing filtering operations thanks to the Conv layers in it and extracts higher level features at each stage. The Neck component acts as an intermediate layer that improves object detection performance by processing the features extracted by the backbone. It improves the features to better detect objects at different scales. In short, the neck part of YOLO models uses multi-scale feature maps to detect objects of different sizes. The head section uses the feature maps processed by the backbone and neck to estimate which object types are in the image, their coordinates and confidence scores. In short, the head part of YOLO models produces final predictions from the processed features of the backbone and neck. YOLO models can estimate more than one box for the same object. Using the IOU (Intersection over Union) value, overlapping boxes are eliminated and the most reliable one is selected, and estimates with low confidence scores are filtered with thresholding. Intersection over Union is a metric used to measure the similarity between the area containing the bounding box coordinates estimated from the YOLO algorithm and the area containing the actual coordinates [33], [34], [35]. A visual representation of how Intersection over Union works is shown in Fig. 2.

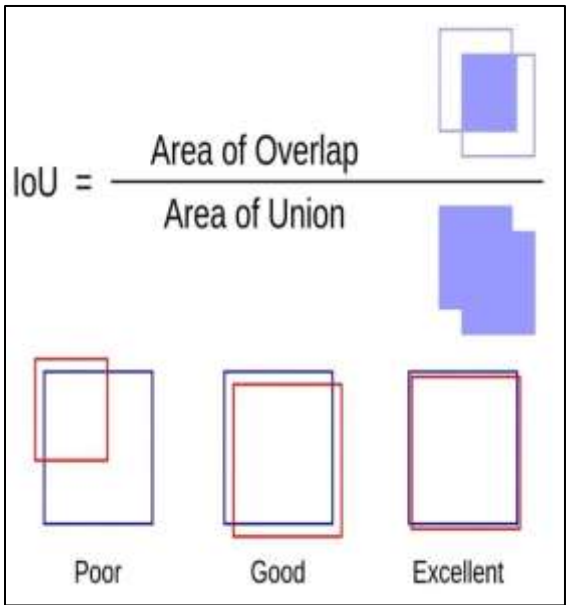

Fig. 2. The visual representation of how Intersection over Union works

In this study, the YOLO-Edge model is developed based on the YOLOv8 architecture [14], [20]. The C2F layers in the backbone section of YOLOv8 were deleted due to the excessive Conv2D layers they contained. Conv layers were replaced with DWConv and GhostConv. The computational cost of DWConv is lower than traditional convolutional layers and unlike traditional convolutional layers, each channel is processed with its own filter, thus having fewer parameters [16]. These features of DwConv cause the GFLOP value of the created model to decrease, thus making the model lighter for embedded systems. In GhostConv, the input image is processed with traditional convolutional layers to produce a small number of main feature maps. New feature maps are created by applying linear operations to the produced main feature maps. This method requires much less computation compared to standard convolutional layers but maintains the same output size [15]. Thus, the model is accelerated without using extra convolution weights. SPPF was preferred as the last layer in the backbone because it provides an easier connection to the neck section compared to other convolution layers and because a single MaxPool2D layer can be applied more than once, thus allowing larger-scale features to be extracted [17]. The schematic representation of the backbone of YOLO-Edge is shown in Fig. 3. Significant changes have been made to the neck section of YOLOv8. Concat and some Conv layers have been deleted due to the excessive computational load they contain. The model is prevented from having 3 head sections and the neck section is connected to a single head section. UpSample layers added to the neck section are used to convert low-resolution data to high resolution. The conv layer applied after UpSample operations is used to apply convolution operation to the data with increased resolution and the model becomes able to detect small-sized objects better. The schematic representation of the neck of YOLO-Edge is shown in Fig. 4. Some Conv layers in the head section of YOLOv8 were deleted and replaced by LightConv. Thus, the head section of YOLO-Edge was created. LightConv is a layer consisting of the combination of Conv2D and DWConv. Thus, by adding LightConv, the number of parameters of the model was reduced and the model became lighter [18], [19], [43]. The schematic representation of the architecture of YOLO-Edge is presented in Fig. 5.

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 25s,2025
https://theaspd.com/index.php

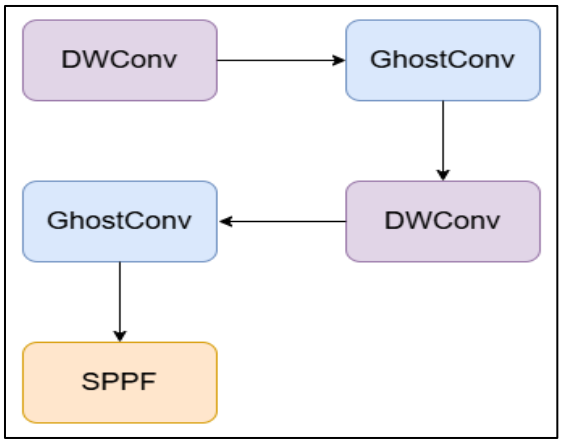

Fig. 3. The schematic representation of the backbone of YOLO-Edge

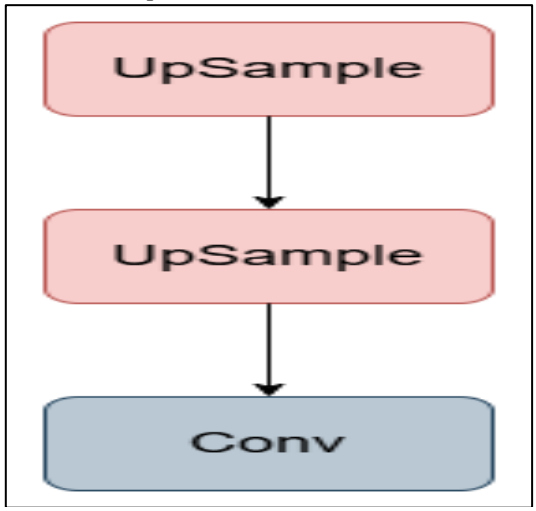

Fig. 4. The schematic representation of the neck of YOLO-Edge

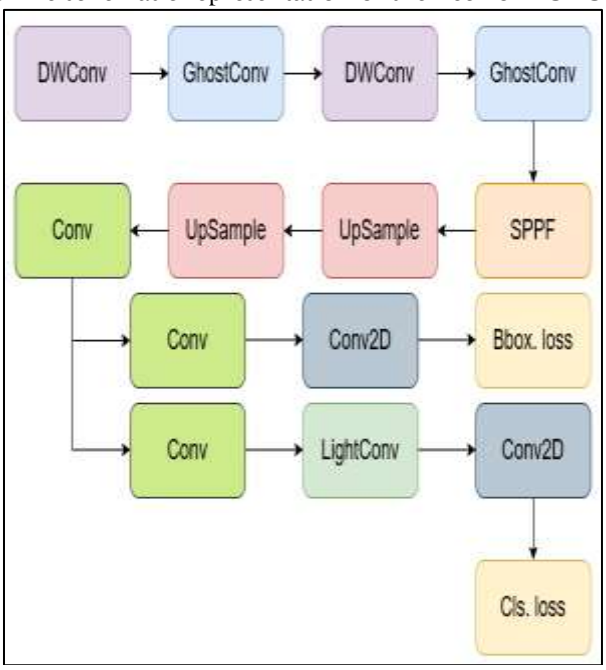

Fig. 5. The schematic representation of the architecture of YOLO-Edge

Table III. Comparison of YOLO-Edge and other YOLO models in terms of the number of layers, number of parameters and GFLOP values

Model	Number of Layers	Number of Parameters	GFLOP
YOLO-Edge	57	1.428.817	10.0
YOLOv8s	225	11.166.560	28.8
YOLOv8m	295	25.902.640	79.3
YOLOv8l	365	43.691.520	165.7
YOLOv8x	365	68.229.648	258.5

4. RESULTS

In this study, Embedded-ANN and the YOLO-Edge model were produced and their results were tested. Validation accuracy values of Embedded-ANN and artificial neural networks with the same architecture created with TensorFlow according to MNIST, Fashion MNIST, Cifar-10 datasets are shown in Table IV. Tests of Embedded-ANN with more datasets and the valuation accuracy values obtained according to these tests are shown in Table V. According to these results, the Embedded-ANN model has a validation accuracy value of more than 95% in 5 out of 8 datasets, indicating that the basic mathematical structure of the model (matrix multiplications, matrix transpose operations, derivatives of activation functions, correct setting of the parameters of the Adam optimizer algorithm) works appropriately. The FPS values and memory footprints of neural networks created with Embedded-ANN and TensorFlow on the Orange Pi 5 Plus are shown in Table VI. According to these results, artificial neural network models created with Embedded-ANN run on average 265 times faster and consume 6 times less memory than artificial neural network models created with TensorFlow. The high speed and low memory consumption of Embedded-ANN make it easy to integrate this neural network model to embedded systems. The main reason for this FPS and memory usage difference between Embedded-ANN and TensorFlow is that matrix multiplications, transpose operations, and derivatives of activation functions are performed using NumPy. Thus, image classification operations can be performed at high speed with Embedded-ANN in embedded systems with low memory capacity. In order to test the YOLO-Edge model, it was tested with the UAV dataset that we created ourselves. The UAV dataset consists only of UAV images and is a dataset consisting of 2616 images as a result of various data augmentation operations such as brightness change, blurring, vertical rotation, and contrast increase. The mAP, Precision, Recall and F1 values of the YOLO-Edge model trained with the UAV dataset at 640×640 resolution for 60 epochs are shown in Table VII. The GPU memories occupied by the YOLO-Edge model and other YOLO models on the RTX 3060 are compared in Fig. 4. According to these results, YOLO-Edge uses less GPU memory than other YOLO models, which makes the YOLO-Edge model suitable for embedded systems. Thus, deep learning-based target detection models will be able to work easily in embedded systems with low memory capacity in the defense industry, and high speed in image streaming will be provided with a high number of frames per second.

Table IV. Comparison of ANN with TensorFlow and Embedded-ANN

Model	MNIST (validation %)	Fashion MNIST (validation %)	Cifar-10 (validation %)
ANN with TensorFlow	96.35	87.03	19.19
Embedded-ANN	96.71	86.53	36.49

Table V. Validation accuracy values of Embedded-ANN based on various datasets and parameters

Dataset Type	Validatio Accuracy (%)	Epoch	Number of Hidden Layers	Number of Neurons of Each Layers
MNIST	96.71	100	1	32
Fashion MNIST	86.53	100	1	32
Cifar-10	36.49	100	1	32
Face Mask	96.06	100	2	32
Brain Tumor	96.48	100	2	32
Coffe Bean	99.25	100	2	32

Dataset Type	Validatio Accuracy (%)	Epoch	Number of Hidden Layers	Number of Neurons of Each Layers
Rock-Paper-Scissors	87.55	100	2	32
Fruits	98.91	100	2	32

Table VI. Comparison of artificial neural networks created with Embedded-ANN and TensorFlow in terms of FPS and memory usage

Model	FPS	RAM Usage (MB)
ANN with Tensorflow	7	930
Embedded-ANN	1860	160

Table VII. Performance metrics of the YOLO-Edge model based on the UAV dataset

Model	mAP	Precision	Recall	F1
YOLO-Edge	0.954	0.798	0.954	0.868

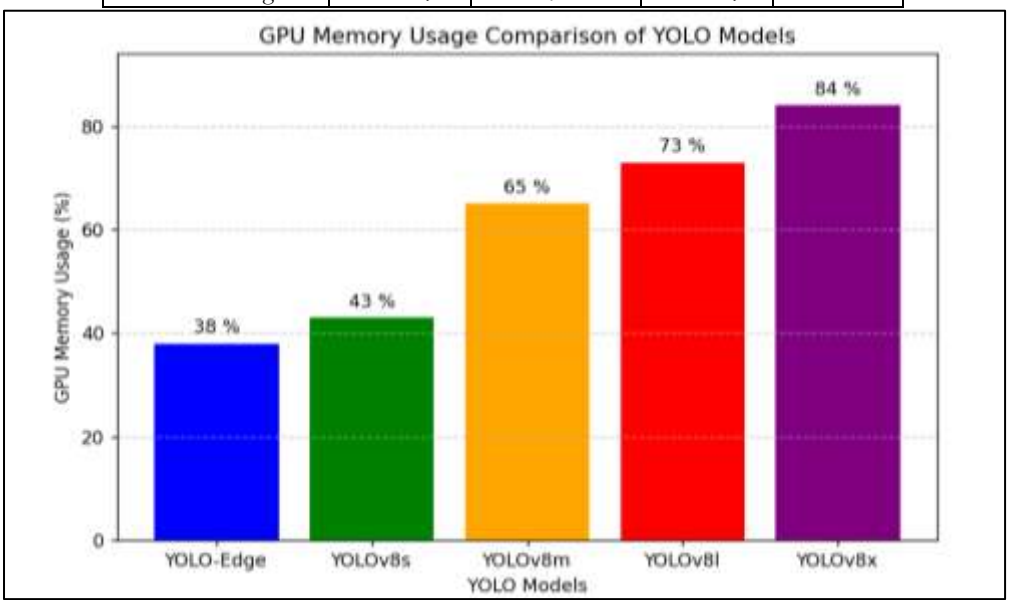

Fig. 4. GPU memory usage comparison of YOLO models

Sample images of real-time UAV detection and tracking using YOLO-Edge are shown in Fig. 5. These results show that the deep learning architecture developed for YOLO-Edge provides accurate results by correctly detecting UAV images taken in different environments and from different angles. By transferring the YOLO-Edge to gimbal cameras used for target detection and tracking in the defense industry, autonomous detection and tracking of unmanned aerial vehicles will be performed more quickly and at a lower cost. The FPS comparison of YOLO-Edge and YOLOv8 models on Raspberry Pi 5, Nvidia Jetson Xavier Nx, Rockchip 3588 is shown in Table VIII. According to these results, the YOLO-Edge model is 4 times faster than YOLOv8m on Nvidia Xavier NX (GPU) and 8 times faster on Rockchip 3588 (NPU).

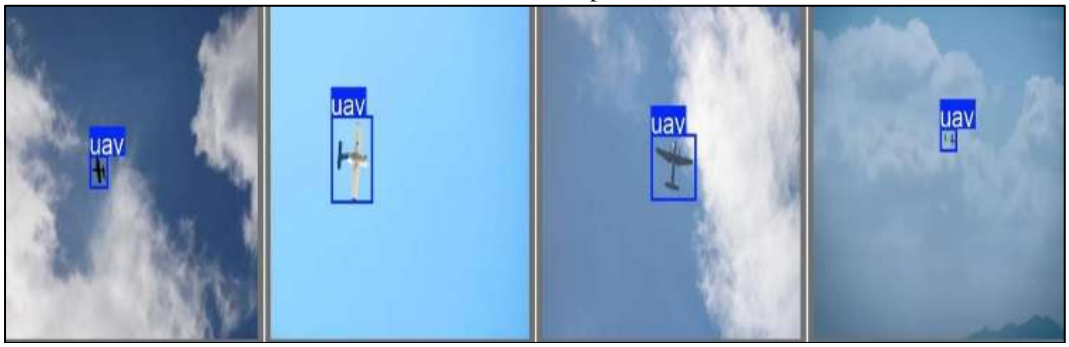

Fig. 5. Images of UAVs detected by YOLO-Edge

Table VIII. Comparison of FPS values of YOLO-Edge and other YOLO models on different embedded systems

Embedded System	Model	Resolution	Format	FPS
Raspberry Pi 5 (CPU)	YOLO-Edge	640	RGB	5
Raspberry Pi 5 (CPU)	YOLOv8s	640	RGB	1
Raspberry Pi 5 (CPU)	YOLOv8m	640	RGB	0.5
Raspberry Pi 5 (CPU)	YOLOv8l	640	RGB	0.2
Raspberry Pi 5 (CPU)	YOLOv8x	640	RGB	0.1
Orange Pi 5 Plus (CPU)	YOLO-Edge	640	RGB	10
Orange Pi 5 Plus (CPU)	YOLOv8s	640	RGB	0.6
Orange Pi 5 Plus (CPU)	YOLOv8m	640	RGB	0.3
Orange Pi 5 Plus (CPU)	YOLOv8l	640	RGB	0.1
Orange Pi 5 Plus (CPU)	YOLOv8x	640	RGB	0.1
Nvidia Jetson Xavier Nx (CPU)	YOLO-Edge	640	RGB	0.7
Nvidia Jetson Xavier Nx (CPU)	YOLOv8s	640	RGB	0.3
Nvidia Jetson Xavier Nx (CPU)	YOLOv8m	640	RGB	0.1
Nvidia Jetson Xavier Nx (CPU)	YOLOv8l	640	RGB	0.1
Nvidia Jetson Xavier Nx (CPU)	YOLOv8x	640	RGB	0.4
Nvidia Jetson Xavier Nx (GPU)	YOLO-Edge	640	RGB	11
Nvidia Jetson Xavier Nx (GPU)	YOLOv8s	640	RGB	7
Nvidia Jetson Xavier Nx (GPU)	YOLOv8m	640	RGB	3
Nvidia Jetson Xavier Nx (GPU)	YOLOv8l	640	RGB	2
Nvidia Jetson Xavier Nx (GPU)	YOLOv8x	640	RGB	1
Rockchip 3588 (NPU)	YOLO-Edge	640	RGB	42
Rockchip 3588 (NPU)	YOLOv8s	640	RGB	20
Rockchip 3588 (NPU)	YOLOv8m	640	RGB	10
Rockchip 3588 (NPU)	YOLOv8l	640	RGB	6
Rockchip 3588 (NPU)	YOLOv8x	640	RGB	4

5. CONCLUSION

In this study, it is explained how the Embedded-ANN and YOLO-Edge models developed for embedded systems with limited resources provide high FPS and low memory usage. The results of this new neural network model are that it is on average 265 times faster and consumes 6 times less memory than TensorFlow models with the same neural network architecture. Thanks to Embedded-ANN, we can reach an average of 1860 FPS in image classification using only a simple CPU without the need for any GPU, NPU or deep learning libraries such as TensorFlow, Keras, Pytorch. In addition, the developed YOLO-Edge model has been modified based on the architecture of YOLOv8 and has been made suitable for providing high FPS in embedded systems. YOLO-Edge will enable target detection and tracking software running on embedded systems in the defense industry to run faster and consume less memory [30], [31], [32]. In this way, energy saving and resource usage in embedded systems have become more efficient and more appropriate for defense industry. In future studies, it is aimed to develop new deep learning models with lower inference time while maintaining the high mAP value by integrating embedded artificial neural networks into the YOLO architecture. Thus, with the addition of Embedded-ANN to YOLO's architecture, a new era is planned to open for deep learning models running on embedded systems.

Acknowledgment:

This study was carried out within the scope of TUBITAK 2224-A Programme for Supporting Participation in International Scientific Activities.

REFERENCES

1. Zhang, Z., & Li, J. (2023). A review of artificial intelligence in embedded systems. Micromachines, 14(5), 897.

2. Roth, W., Schindler, G., Klein, B., Peharz, R., Tschiatschek, S., Fröning, H., ... & Ghahramani, Z. (2024). Resource-efficient neural networks for embedded systems. Journal of Machine Learning Research, 25(50), 1-51. Voss RF, Clarke J. (1986) Algorithmic Musical Composition, Silver Burdett Press, London.
3. Khanam, R., & Hussain, M. (2024). Yolov11: An overview of the key architectural enhancements. arXiv preprint arXiv:2410.17725.
4. Venzke, M., Klisch, D., Kubik, P., Ali, A., Missier, J. D., & Turau, V. (2020). Artificial neural networks for sensor data classification on small embedded systems. arXiv preprint arXiv:2012.08403.
5. Cittadini, E., Marinoni, M., & Buttazzo, G. (2025). A hardware accelerator to support deep learning processor units in real-time image processing. Engineering Applications of Artificial Intelligence, 145, 110159.
6. Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., & Hinton, G. (2020). Backpropagation and the brain. Nature Reviews Neuroscience, 21(6), 335-346.
7. Shin, Y., & Karniadakis, G. E. (2020). Trainability of relu networks and data-dependent initialization. Journal of Machine Learning for Modeling and Computing, 1(1).
8. Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.
9. Datta, L. (2020). A survey on activation functions and their relation with xavier and he normal initialization. arXiv preprint arXiv:2004.06632. (pp. 1733-1738). IEEE.
10. Koturwar, S., & Merchant, S. (2017). Weight initialization of deep neural networks (DNNs) using data statistics. arXiv preprint arXiv:1710.10570.
11. Hanin, B., & Rolnick, D. (2018). How to start training: The effect of initialization and architecture. Advances in neural information processing systems, 31.
12. Rasheed, A. F., & Zarkoosh, M. (2025). Optimized YOLOv8 for multi-scale object detection. Journal of Real-Time Image Processing, 22(1), 6.
13. Muna, A. S., & Ramo, F. M. (2024). Performance evaluation for face mask detection based on mult modification of yolov8 architecture. Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska, 14(2), 89-95.
14. Terven, J., Córdova-Esparza, D. M., & Romero-González, J. A. (2023). A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. Machine learning and knowledge extraction, 5(4), 1680-1716.
15. Cao, J., Bao, W., Shang, H., Yuan, M., & Cheng, Q. (2023). GCL-YOLO: A GhostConv-based lightweight yolo network for UAV small object detection. Remote Sensing, 15(20), 4932.
16. Zhang, P., Lo, E., & Lu, B. (2020, April). High performance depthwise and pointwise convolutions on mobile devices. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 04, pp. 6795-6802).
17. Zhang, G., Wang, C., & Xiao, D. (2024). A novel daily behavior recognition model for cage-reared ducks by improving SPPF and C3 of YOLOv5s. Computers and Electronics in Agriculture, 227, 109580.
18. Liu, P., Fu, H., & Ma, H. (2021). An end-to-end convolutional network for joint detecting and denoising adversarial perturbations in vehicle classification. Computational Visual Media, 7, 217-227.
19. Li, Y., Li, Q., Pan, J., Zhou, Y., Zhu, H., Wei, H., & Liu, C. (2024). Sod-yolo: Small-object-detection algorithm based on improved yolov8 for uav images. Remote Sensing, 16(16), 3057.
20. Zhao, T., Feng, R., & Wang, L. (2025). SCENE-YOLO: A One-stage Remote Sensing Object Detection Network with Scene Supervision. IEEE Transactions on Geoscience and Remote Sensing.
21. Bai, Y. (2022). RELU-function and derived function review. In SHS web of conferences (Vol. 144, p. 02006). EDP Sciences.
22. Hussain, M. (2023). YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection. Machines, 11(7), 677.
23. Kaushal, M. (2022). Rapid-YOLO: A novel YOLO based architecture for shadow detection. Optik, 260, 169084.
24. Hasan, R. H., Hassoo, R. M., & Aboud, I. S. (2023). Yolo Versions Architecture. International Journal of Advances in Scientific Research and Engineering, 9(11), 73.
25. Ferrante, G. S., Vasconcelos Nakamura, L. H., Sampaio, S., Filho, G. P. R., & Meneguette, R. I. (2024). Evaluating YOLO architectures for detecting road killed endangered Brazilian animals. Scientific reports, 14(1), 1353.
26. Kalinina, M., & Nikolaev, P. (2020, November). Research of YOLO architecture models in book detection. In 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS 2020) (pp. 218-221). Atlantis Press.
27. Vijayakumar, A., & Vairavasundaram, S. (2024). Yolo-based object detection models: A review and its applications. Multimedia Tools and Applications, 83(35), 83535-83574.
28. Jovanovic, L., Bacanin, N., Zivkovic, M., Mani, J., Strumberger, I., & Antonijevic, M. (2023, October). Comparison of yolo architectures for face mask detection in images. In 2023 16th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS) (pp. 179-182). IEEE.
29. Alif, M. A. R., & Hussain, M. (2024). YOLOv1 to YOLOv10: A comprehensive review of YOLO variants and their application in the agricultural domain. arXiv preprint arXiv:2406.10139.
30. Sumit, S. S., Awang Rambli, D. R., Mirjalili, S., Ejaz, M. M., & Miah, M. S. U. (2022). Restinet: On improving the performance of tiny-yolo-based cnn architecture for applications in human detection. Applied Sciences, 12(18), 9331.
31. Zhang, Z., Lu, X., Cao, G., Yang, Y., Jiao, L., & Liu, F. (2021). ViT-YOLO: Transformer-based YOLO for object detection. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 2799-2808).
32. Gunawan, T. S., Ismail, I. M. M., Kartiwi, M., & Ismail, N. (2022, September). Performance Comparison of Various YOLO Architectures on Object Detection of UAV Images. In 2022 IEEE 8th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA) (pp. 257-261). IEEE.

33. Rahman, M. A., & Wang, Y. (2016, December). Optimizing intersection-over-union in deep neural networks for image segmentation. In International symposium on visual computing (pp. 234-244). Cham: Springer International Publishing.
34. Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 658-666).
35. Zhang, H., & Zhang, S. (2024). Focaler-iou: More focused intersection over union loss. arXiv preprint arXiv:2401.10525.
36. Chiranjeevi, V. R., Dhanasekaran, S., Murugan, B. S., & Pandi, S. S. (2024, April). ADAM Optimizer Based Convolutional Auto Encoder for Detecting Anomalies in Surveillance Videos. In 2024 International Conference on Communication, Computing and Internet of Things (IC3IoT) (pp. 1-5). IEEE.
37. Swathika, R., & Kumar, S. D. (2024, February). RSS-Based Localization using Deep Learning Models with Optimizer in LoRaWAN-IoT Networks. In 2024 IEEE International Conference for Women in Innovation, Technology & Entrepreneurship (ICWITE) (pp. 242-246). IEEE.
38. Roy, S. (2023). Understanding the impact of post-training quantization on large language models. arXiv preprint arXiv:2309.05210.
39. Xiao, P., Zhang, C., Guo, Q., Xiao, X., & Wang, H. (2024). Neural networks integer computation: Quantizing convolutional neural networks of inference and training for object detection in embedded systems. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.
40. Xiao, P., Lin, X., & Wang, H. (2022, July). Research on Neural Network Post-Quantization Method for Ship Detection in SAR Images. In IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium (pp. 2155-2158). IEEE.
41. Kır, S., & Günay, E. E. (2022). Augmented Artificial Neural Network Model for the COVID-19 Mortality Prediction: Preliminary Analysis of Vaccination in Turkey. Sakarya University Journal of Computer and Information Sciences, 5(1), 22-36.
42. Yolcu Öztel, G., & Öztel, İ. (2023). Deep Learning-based Road Segmentation & Pedestrian Detection System for Intelligent Vehicles. Sakarya University Journal of Computer and Information Sciences, 6(1), 22-31.
43. Özatılgan, A., & Kaya, M. (2024). A Lightweight Convolutional Neural Network for Classification of Brain Tumors Using Magnetic Resonance Imaging. Sakarya University Journal of Computer and Information Sciences, 7(3), 482-493.
44. Bakhytov, Y., & Öz, C. (2024). Cigarette Detection in Images Based on YOLOv8. Sakarya University Journal of Computer and Information Sciences, 7(2), 253-263.
45. Patel, K., Patel, V., Prajapati, V., Chauhan, D., Haji, A., & Degadwala, S. (2023, June). Safety helmet detection using YOLO v8. In 2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN) (pp. 22-26). IEEE.
46. Xiong, C., Zayed, T., & Abdelkader, E. M. (2024). A novel YOLOv8-GAM-Wise-IoU model for automated detection of bridge surface cracks. Construction and Building Materials, 414, 135025.
47. Bawankule, R., Gaikwad, V., Kulkarni, I., Kulkarni, S., Jadhav, A., & Ranjan, N. (2023, June). Visual detection of waste using YOLOv8. In 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS) (pp. 869-873). IEEE.
48. Bakirci, M. (2024). Utilizing YOLOv8 for enhanced traffic monitoring in intelligent transportation systems (ITS) applications. Digital signal processing, 152, 104594.
49. Soylu, E., & Soylu, T. (2024). A performance comparison of YOLOv8 models for traffic sign detection in the Robotaxi-full scale autonomous vehicle competition. Multimedia Tools and Applications, 83(8), 25005-25035.
50. Gašparović, B., Mauša, G., Rukavina, J., & Lerga, J. (2023, June). Evaluating Yolov5, Yolov6, Yolov7, and Yolov8 in underwater environment: Is there real improvement?. In 2023 8th International Conference on Smart and Sustainable Technologies (SpliTech) (pp. 1-4). IEEE.