Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 8, No. 1, March 2020, pp. 204~214
ISSN: 2089-3272, DOI: 10.11591/ijeei.v8i1.1960

A Fast and Accurate Object Detection Algorithm on Humanoid Marathon Robot

Eko Rudiawan Jamzuri[1], Hanjaya Mandala[2], Jacky Baltes[3]
[1,2,3]Department of Electrical Engineering, National Taiwan Normal University, Taiwan
[1]Department of Electrical Engineering, Politeknik Negeri Batam, Indonesia

Article Info

Article history:

Received Jan 13, 2020
Revised March 30, 2020
Accepted March 31, 2020

Keywords:

Object detection
Deep learning
Convolution neural network
Robotic vision
Region proposal

ABSTRACT

This paper introduces a fast and accurate object detection algorithm based on a convolutional neural network for humanoid marathon robot applications. The algorithm is capable of operating on a low-performance CPU without relying on the GPU or hardware accelerator. A new region proposal algorithm, based on color segmentation, is proposed to extract a region containing a potential object. As a classifier, the convolution neural network is used to predict object classes from the proposed region. In the training phase, the classifier is trained with an Adam optimizer to minimize the loss function, using datasets collected from humanoid marathon competitions and diversified using image augmentation. An NVIDIA GTX 1070 training machine, with 500 batch images per epoch and a learning rate of 0.001, required 12 seconds to minimize the loss value below 0.0374. In the accuracy evaluation, the proposed method successfully recognizes and localizes three classes of marker with a training accuracy of 99.929%, validation accuracy of 99.924%, and test accuracy of 98.821%. As a real-time benchmark, the algorithm achieves 41.13 FPS while running on a robot computer with Intel i3-5010U CPU @ 2.10GHz.

Corresponding Author:

Eko Rudiawan Jamzuri,
Department of Electrical Engineering,
Politeknik Negeri Batam,
Jl. Ahmad Yani, Batam Centre, Batam 29461, Indonesia.
Email: ekorudiawan@polibatam.ac.id

1. INTRODUCTION

Object detection is the most common problem for robotic vision. Provided an image, the class and position of the object must be predicted. State of the art of Convolutional Neural Network (CNN) based object detection successfully addresses the problems in this domain, even though it requires a high computing platform. Santos et al. compared the performance of three CNN algorithms to detect tree species using an RGB camera on an Unmanned Aerial Vehicle (UAV) [1]. Faster R-CNN [2], YOLOv3 [3], and RetinaNet [4] were evaluated using a folding approach and successfully reached over 82.48 % validation accuracy. However, the detection process was offline, used an off-board computing platform, and was not performed in real-time.

In typical humanoid robotics applications, such as soccer-playing, marathon running, and obstacle avoidance, both of real-time performance and accuracy are crucial. In such cases, CNN algorithms prove challenging as robots require onboard computers capable of running such algorithms. Researchers address this by proposing embedded GPU modules or deep learning hardware accelerators. An embedded GPU NVIDIA Jetson TX1 on a humanoid robot to run YOLOv2 for ball and goal position detection on the soccer robot is introduced in [5]. By applying this approach, the detection process reaches 20 FPS and a training accuracy of over 60%. Further, [6] proposes two hardware options for deep learning object detection on a flying robot. With these methods, the detection speed reaches 8 FPS using Jetson TX1 with a Single Shot Detector (SSD) model and 5 FPS using the SSD Mobile Net model, implemented on Raspberry Pi with an additional Intel Neural Compute Stick (NCS) accelerator.

Some researchers address the problem through an efficient CNN algorithm, reducing the architecture and proposing an optimize region proposal layer. A region proposal layer that consists of efficient on-line convolutions and effective off-line optimization, followed by a detection layer using an MPGA-based CNN module and a TLD-based multi-frame fusion procedure, is proposed in [7]. The proposed approach results in robust and efficient detection without relying on GPU computation. A simplified CNN architecture, called FLODNet, which consists of seven convolution and max-pooling layers, followed by three fully connected layers, is described in [8]. The proposed architecture requires only 95 MB parameters and can be suitably run on a laptop CPU with detection speed around 8.85 FPS.

Our research focuses on addressing a problem in the humanoid robot marathon competition that requires a robot to detect three different markers, each a different symbol in the same dominant color. A marker must be identified using the robot camera in real-time, due it is used as feedback for the robot behavior controller. In this research, we propose a two-stage detection approach. On the first stage detection, a new method of region proposal, using a color segmentation technique, is proposed to extract a region of the object. The second is classifying the region using a shallow CNN classifier that consists of 10 layers with 83 MB parameters. The proposed approach is fast, accurate, and does not require GPU or hardware accelerator for inferencing the algorithm. The GPU only used for training the CNN classifier.

This paper is organized as follows. Section 2 explains the method for conducting this research, starting with an introduction of the research environment, followed by the dataset collection process, then an explanation of the novel region proposal algorithm, the proposed CNN architecture, and lastly, the method to train and validate the classifier. Section 3 explains in detail the results of our research, starting with the region proposal, training and validation, inference on the robot computer, and comparisons with previous work. Section 4 outlines the main conclusions and an avenue for further research.

2. RESEARCH METHOD

This research was conducted beginning with dataset collection, followed by designing a region proposal algorithm and developing a CNN classifier architecture. The performance evaluation was then applied to measure the classification accuracy and detection speed of the proposed method.

2.1. Research Environment

The Federation of International Robot-soccer Association (FIRA) humanoid marathon competition was selected as a research benchmark for our object detection algorithm. In the marathon competition, a humanoid robot must able to recognize a line and a series of markers. A marker is a 10 cm x 10 cm sign for the robot to navigate in following the right track, consisting of three different directions (forward, left, and right). The robot must be able to recognize these markers and localize the position in the camera frame. Figure 1 illustrates an example of a marathon competition.

We used a modified version of the Darwin OP robot, which has 22 degrees of freedom [9]. A modification was applied by adding two grippers to the end effector of the robot arm, changing the default camera to the high-resolution web camera (Logitech C920), and changing the processing unit into a mini-computer with an Intel Core i3 processor. The robot architecture and specifications are shown in Figure 2 and Table 1.

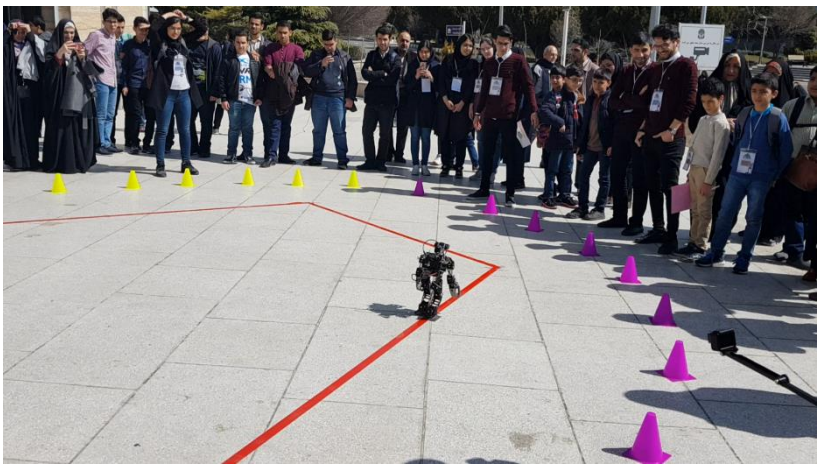
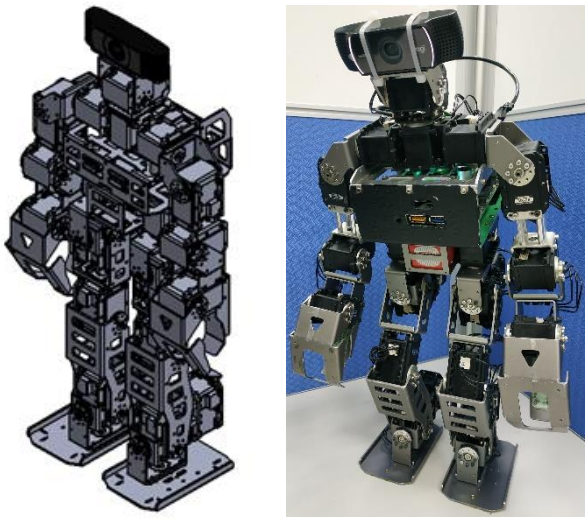


Figure 1. Humanoid robot in the FIRA marathon competition.



(a) (b)

Figure 2. (a) Mechanical design of the robot (b) real view of the robot.

Table 1. Humanoid robot specification.

Specification	Value
Height	450 mm
Weight	3 Kg
Battery	4-cell LiPo (14.8V, 2200mAh)
Main Controller	Intel NUC i3-5010U CPU @ 2.10GHz, RAM 4GB, SSD 250GB
Sub Controller	CM730, ARM Cortex M3 @ 72MHz
IMU Sensor	Gyro L3G4200D, Accelerometer LIS331DLH
Vision Sensor	Logitech C922 Pro
Actuator	22 x Robotics MX-28 Servo

2.2. Dataset Collection

The image dataset was collected from two different sources containing a total of 3,486 images. The first dataset was collected from the marathon field in our lab (Educational Robotics Centre, National Taiwan Normal University) using the robot camera and contains 660 images. The second dataset, containing 2,826 images, was collected from the marathon track of the Taiwan Humanoid 2019 robot competition and captured using a phone camera across different distances and perspectives. The image datasets were manually cropped by picture editor software to contain only the markers. These images were used as training data for our classifier model.

In order to increase the variation of the dataset, we created synthetic image data from the original images using an image augmentation method [10] by applying image transformations, diversifying brightness, changing contrast value, and placing random black rectangles on the images. Figure 3(a) shows an example of an image from the robot camera, Figure 3(b) a cropped image, and Figure 3(c) a synthetic image from the data augmentation.

Overall, the dataset contains 55,776 images from both of the original datasets and image augmentation results. The dataset distribution of each class is 17,696 images (31.727%) of forward markers, 18,000 images (32.272%) of right markers, and 20,080 images (36.001%) of left markers.

(a) (b) (c)

Figure 3. (a) Raw image (b) cropped images (c) synthetic images.

2.3. Color-Based Region Proposal

Region proposal is commonly used to propose a region that contains a potential object for the two-stages of the object detection algorithm. Prior work in [11] proposed a selective search algorithm for generating possible object locations, and it has been used with the state of the art R-CNN algorithm [12]. In contrast, a selective search algorithm creates a bottleneck, extracting 2,000 candidate regions and classifying each. In this work, we proposed a simple region proposal algorithm by using color segmentation. We assume that each image frame contains a single object with a dominant color that can be identified by classifying color. For marker detection, the marker has a dominant color, black or white. The black color is chosen as a reference color to propose a region of interest (ROI) of the object. A pipeline for color-based region proposal is shown in Figure 4.

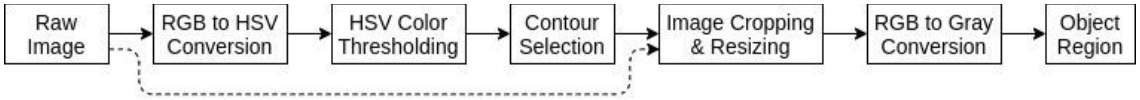


Figure 4. Region proposal pipeline.

A raw image is captured from the robot camera in red, green, blue (RGB) color space and converted to hue, saturation, and value (HSV) color space. HSV color conversion is started by normalizing the R, G, and B color channels as follows:

$$R' = \frac{R}{255} \tag{1}$$

$$G' = \frac{G}{255} \tag{2}$$

$$B' = \frac{B}{255} \tag{3}$$

R, G, and B are the red, green, and blue color intensities, respectively, in RGB color space in the range of 0–255 and R', G', and B' are normalized red, green, and blue color intensities, respectively, in the range of 0–1. From the normalized RGB color space, a range of minimum and maximum value (Δ) can be calculated as follows:

$$C_{min} = \min(R', G', B') \tag{4}$$

$$C_{max} = \max(R', G', B') \tag{5}$$

$$\Delta = C_{max} - C_{min} \tag{6}$$

C_{min} and C_{max} represent the minimum and maximum value of the normalized RGB color space. Hue (H), saturation (S), and value (V) of HSV color space are defined in (7), (8), and (9).

$$H = \begin{cases} 0^o, & \Delta = 0 \\ 60^o \left(\frac{G'-B'}{\Delta} mod(6)\right), & C_{max} = R' \\ 60^o \left(\frac{B'-R'}{\Delta} + 2\right), & C_{max} = G' \\ 60^o \left(\frac{R'-G'}{\Delta} + 4\right), & C_{max} = B' \end{cases} \tag{7}$$

$$S = \begin{cases} 0, & C_{max} = 0 \\ \frac{\Delta}{C_{max}}, & C_{max} \neq 0 \end{cases} \tag{8}$$

$$V = C_{max} \tag{9}$$

In order to apply color classification, an HSV color space image is converted into a binary image by applying a color thresholding function in (10).

$$f(x,y) = \begin{cases} 255, & H_{min} < H \leq H_{max}\ AND\ S_{min} < S \leq S_{max}\ AND\ V_{min} < V \leq V_{max} \\ 0, & otherwise \end{cases} \tag{10}$$

H_{min}, S_{min}, and V_{min} are minimum threshold parameters and H_{max}, S_{max}, and S_{max} are maximum threshold parameters for each H, S, and V value in pixel coordinates x and y. The color thresholding process will result in either 0 or 255, where 0 indicates a black pixel and 255 a white pixel. A group of white pixels on a binary image (contour) is selected based on the area and ratio of the width and height to get an ROI of the potential object. The parameters of the area and the ratio of width and height are predefined and adjusted manually. The contour that matches the predefined parameter is cropped from the raw RGB image, resize to 100 x 100 pixels, then converted into grayscale color space to reduce the number of color channels. The gray color intensity (Y) is defined as follows:

$$Y = 0.299R + 0.587G + 0.114B \tag{11}$$

This grayscale image is the output of the region proposal algorithm and becomes the input for the CNN classifier to predict the class of an object.

2.4. Convolutional Neural Network Classifier

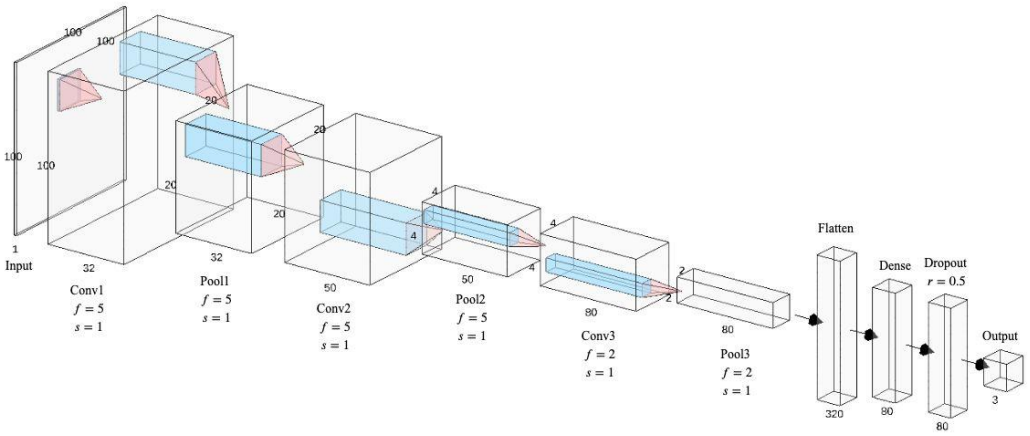


Figure 5. CNN classifier architecture.

A CNN classifier is used to predict the class of the proposed region by the color-based region proposer in Section 2.3. Generally, a CNN classifier consists of a convolution layer, a subsampling/polling layer, and a dense/fully connected layer. Our CNN architecture is adopted from classical CNN architecture LeNet-5 [13] and adds an extra dropout layer to the dense layer. The CNN architecture is detailed in Figure 5.

Here, l is the layer number in the CNN architecture, $a^{[l-1]}$ the previous input layer, $W^{[l]}$ the convolution filter or weight, and $b^{[l]}$ the bias. The output of the convolution layer $a^{[l]}$ can be formulated as:

$$z^{[l]} = W^{[l]} * a^{[l-1]} + b^{[l]} \tag{12}$$

$$g\left(z^{[l]}\right) = \max(0, z^{[l]}) \tag{13}$$

$$a^{[l]} = g(z^{[l]}) \tag{14}$$

Where $z^{[l]}$ is the result of the convolution operation previous layer with weight and added bias parameter. $z^{[l]}$ is mapped onto the next layer $a^{[l]}$ by applying the non-linear activation function ReLU $g(z^{[l]})$. The dimension of the convolution layer output is represented as $(n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]})$ tensor, where $n_c^{[l]}$ is the number of the convolution filter, $n_h^{[l]}$ and $n_w^{[l]}$ are height and width of output tensor, that defined as:

$$n_h^{[l]} = \frac{n_h^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \tag{15}$$

$$n_w^{[l]} = \frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \tag{16}$$

Where $f^{[l]}$ is filter size, $p^{[l]}$ padding size, and $s^{[l]}$ the stride of the convolution filter. The convolution layer is followed by a max-pooling layer that selects a maximum value from the previous layer with specific window size and stride. The dimension of the max-pooling output is represented as $(n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]})$ tensor, where $n_c^{[l]}$ is same as $n_c^{[l-1]}$, $n_h^{[l]}$ and $n_w^{[l]}$ are:

$$n_h^{[l]} = \frac{n_h^{[l-1]} - f^{[l]}}{s^{[l]}} + 1 \quad (17)$$

$$n_w^{[l]} = \frac{n_w^{[l-1]} - f^{[l]}}{s^{[l]}} + 1 \quad (18)$$

The output tensor of the last max-pooling layer is flattened into a 1D vector to be an input neuron in the dense layer. We added a dropout layer before the output layer to reduce overfitting in the training process [14]. The dropout layer discards neurons with probability less than the rate (r). In the output layer, a softmax activation function is used to predict the class of input images, defined as follows:

$$g\left(z^{[l]}\right) = \frac{e^{z^{[l]}}}{\sum_{j=1}^{c} e^{z^{[l]}}} \quad (19)$$

Where e is the exponential number and c the number of classes. The predicted output of the CNN classifier $(\hat{y})$ is taken from the output of the softmax activation function. The loss function between predicted output and truth label is calculated using categorical cross-entropy loss in (20).

$$L(y, \hat{y}) = -\sum_{j=1}^{c} y_j \log(\hat{y}_j) \quad (20)$$

The cost function J in (21) is used to measure the performance of the CNN classifier, provided weight parameter (W), bias parameter (b), and the number of training examples (m). For better performance, W and b parameters are updated in several numbers of iteration using a stochastic optimization algorithm to get the minimum value of J. This process usually called the learning phase or backpropagation step.

$$J = \frac{1}{m} \sum_{j=1}^{m} L(\hat{y}_j, y_j) \quad (21)$$

In this research, Adam [15] is used as the optimization algorithm. Adam has a parameter learning rate (α), decay rate for first moment estimates $(\beta 1)$, decay rate for second-moment estimates $(\beta 2)$, and a small number (ϵ) to prevent division error. The learning parameter is adjusted manually for a faster training process and better performance of the CNN classifier.

2.5. Training and Validation

K-fold cross-validation [16] is used to evaluate the performance of the CNN classifier in the training stage. First, the dataset is split—90% training data and 10% testing data. In the training stage, training data is split into five-folds (Figure 6). We used five classifiers with the same architecture, and each classifier uses four-folds for training and one-fold for validation. We also varied the learning rate parameter to expedite the training process and performance. The performance of the classifier was measured by averaging the training accuracy and validation accuracy from those classifiers. In the testing stage, the final performance of the classifier was evaluated by picking the ideal classifier and testing it with a dataset that was never used in the learning phase. The training stage used a computer with GPU with detail specification is listed in Table 2. In the inference stage, the ideal classifier was used on the robot computer with an input image from the robot camera to evaluate the performance of the classifier's processing time.

Table 2. Computer specification for training CNN.

Hardware	Specification
Processor	Intel(R) Core(TM) i7-8750H CPU@2.20GHz, 6 CPU Cores
RAM	16 GB
Hard disk	SSD 234 GB
GPU	GeForce GTX 1070, 1920 CUDA Cores

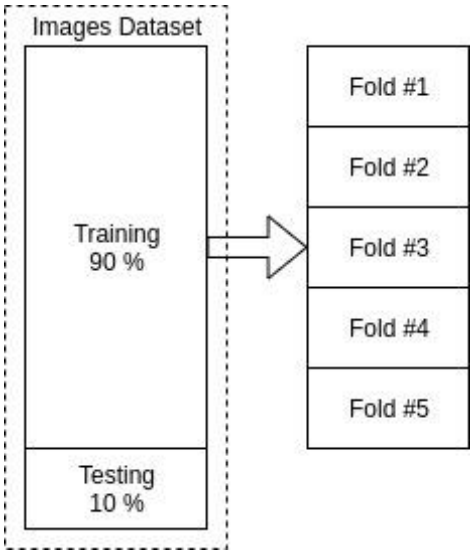


Figure 6. Dataset for training and testing process.

3. RESULTS AND DISCUSSION

3.1. Region Proposal Results

Figure 7 illustrates the result of the color-based region proposal algorithm with parameters H_{min} = 26, S_{min} = 29, V_{min} = 55, H_{max} = 121, S_{max} = 255, V_{max} = 121, $Area_{min}$ = 4,651, and $Area_{max}$ = 37,500. Figure 7(a) shows the color thresholding process that results in a binary image. Figure 7(b) is a proposed region successfully cropped from the raw image and containing a potential object. In this case, the region proposal parameter depends on environmental lighting conditions and must be adjusted manually for a proper region. Some slight noise remains from the color thresholding process (Figure 7(a)) and can be removed by adjusting the parameters $Area_{min}$ and $Area_{max}$.

(a) (b)

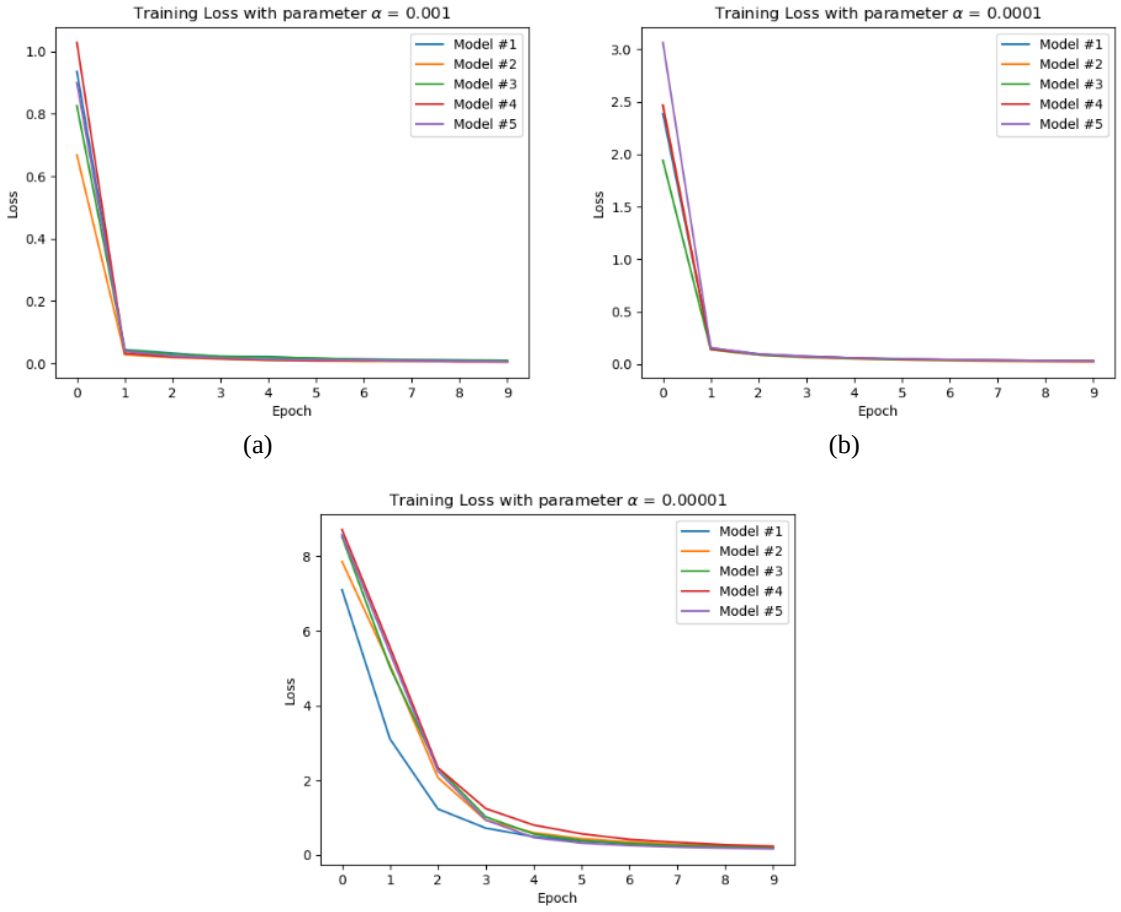
Figure 7. (a) Binary thresholding result (b) a proposed region of the potential object.

3.2. Training and Evaluation Results

Figure 8 illustrates the result of the loss function (21) during the training phase using an Adam optimization algorithm with learning parameters $\beta 1 = 0.9$, $\beta 2 = 0.999$, $\epsilon = 0.1$, and varying learning rate (α). Figure 8(a) demonstrates that training loss drops significantly after several epochs, where $\alpha = 0.001$ is the proper learning rate and can decrease to less than 0.2 after two epochs. On the other hand, a small α requires the training algorithm more epoch to converge, as shown in Figure 8(c) where $\alpha = 0.00001$—the algorithm takes more epochs to reduce loss function compared to Figures 8(a) and 8(b). In this research, the time to learn per 1 epoch is 6 seconds using batch size = 500 images while training. By choosing the proper parameters, as in Figure 8(a), the learning phase takes 12 seconds to generate a proper classifier model using a training computer, as described in Table 2.

Table 3 shows an evaluation of the accuracy of CNN classifier during the training process. Overall, the training and validation accuracy increase above 98.652% during training and above 99.550% during validation. By using $\alpha = 0.001$, the accuracy increases the most, at 99.929% in training and 99.924% in the validation. This result suggests that learning rate $\alpha = 0.001$ is the ideal parameter in the training phase, resulting in a faster training process and a more accurate classifier.

The confusion matrix from the classification result is illustrated in Figure 9, where the ideal classifier with $\alpha = 0.001$ is used to predict a test dataset. Overall, test accuracy is 99.821% in the test set, which generated ten misclassified results: a forward marker predicted as a right marker, four right markers predicted as a left marker, and five left markers predicted as a right marker. This result shows that the classifier fits with the model as both validation and test accuracy result in values above 99%.



(a) (b)

(c)

Figure 8. Loss value during the training phase.

Table 3. Training and validation accuracy.

Learning Parameter	Training Acc.	Validation Acc.
α = 0.001	99.929 %	99.924 %
α = 0.0001	99.889 %	99.888 %
α = 0.00001	98.652 %	99.550 %

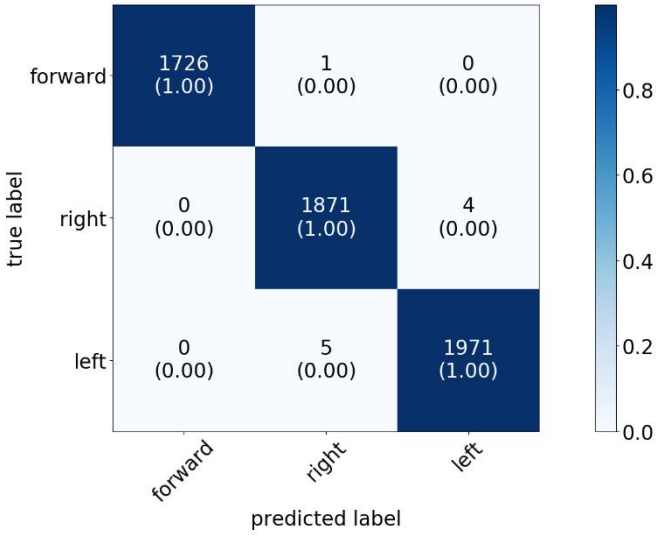
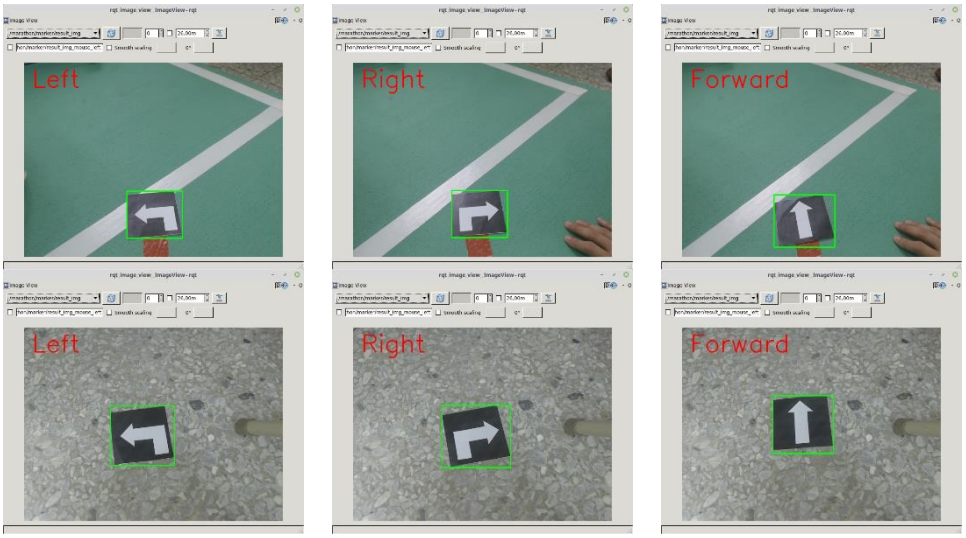


Figure 9. Confusion matrix of classification on the test dataset.

3.3. Inference Results

The inference results for our object detection approach were evaluated on robot computing hardware (Figure 10) for each marker: left, right, and forward. The marker object is identified and delimited by the green bounding box, while text at the top left corner of the image shows the predicted label from the CNN classifier. Based on the experimental results, the algorithm is capable of recognizing a marker object on different backgrounds. To evaluate the processing speed, we used a statistical approach with ten sampling measurements to determine the average processing time of our algorithm. Based on the measurement, it takes an average of 24.313 ms or 41.13 FPS to process one frame image, excluding the process of acquiring the image from the camera device.



(a) (b) (c)

Figure 10. (a) Left marker, (b) right marker, and (c) forward marker.

3.4. Comparison Benchmark

Table 4 shown a comparison benchmark of our approach to prior work by researchers. Based on the comparison, our approach is the fastest algorithm, reaching 41.13 FPS compares to previous approaches, even while runs on an Intel Core i3 CPU. Moreover, the validation accuracy is the highest compared to other approaches.

Table 4. Comparison benchmark with the prior work.

Method	Object Classes	Inference Hardware	CNN Model	Detection Speed	Validation Accuracy
Santos et al. [1]	Tree species	Intel Xeon CPU E3-1270@3.80GHz, NVIDIA Titan V	Faster R-CNN	6.13 FPS	82.48 %
			YOLOv3	38.46 FPS	85.88 %
			RetinaNet	14.93 FPS	92.64 %
Susanto et al. [5]	Ball and goal	NVIDIA Jetson TX1	YOLOv2	20 FPS	60.00 %
Hossain and Lee [6]	Multiple objects	NVIDIA Jetson TX2	YOLOv2	7 FPS	-
			YOLOv2 Tiny	15 FPS	-
			YOLOv3	3 FPS	-
			YOLOv3 Tiny	12 FPS	-
			SSD	11 FPS	-
		Raspberry Pi + Movidius NCS	YOLO	1 FPS	-
			SSD MobileNet	5 FPS	-
Oliveira et al. [8]	Foods	Intel Core i7-3610QM CPU	FLODNet + Sliding Window	0.16 FPS	-
			FLODNet + Conv. Feature Map	8.85 FPS	-
Proposed Method	**Marathon marker**	**Intel NUC i3-5010U CPU@2.10GHz**	**Color-based region proposal + CNN Classifier**	**41.13 FPS**	**99.92 %**

4. CONCLUSION

In this paper, two stages of a CNN-based object detection algorithm are introduced to solve an object detection problem within a typical humanoid marathon robot competition. The detection algorithm consists of a color-based region proposal, and CNN classifier with six convolution and max-pooling layers, followed by four dense layers. An Adam optimizer is used to optimize the classifier model with a dataset that was collected and augmented. In the experimental results, the proposed algorithm was able to detect three categories of markers with a training accuracy of 99.929%, validation accuracy of 99.924%, and test accuracy of 99.821%. The algorithm can be implemented on an onboard robot computer with an Intel i3-5010U CPU @ 2.10GHz with a maximum detection speed of 41.13 FPS. However, setting up the color segmentation parameters should be further considered and is an area for future work.

ACKNOWLEDGMENTS

This work was financially supported by the Chinese Language and Technology Center of National Taiwan Normal University (NTNU) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan, and Ministry of Science and Technology, Taiwan, under Grants No. MOST 108-2634-F-003-002, MOST 108-2634-F-003-003, and MOST 108-2634-F-003-004 (administered through Pervasive Artificial Intelligence Research (PAIR) Labs), as well as MOST 107-2811-E-003-503-. We are grateful to the National Center for High-performance Computing for computer time and facilities to conduct this research.

REFERENCES

[1] A. A. dos Santos *et al.*, "Assessment of CNN-Based Methods for Individual Tree Detection on Images Captured by RGB Cameras Attached to UAVs," *Sensors*, vol. 19, no. 16, p. 3595, 2019.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[4] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[5] Susanto, E. Rudiawan, R. Analia, P. D. Sutopo, and H. Soebakti, "The deep learning development for real-time ball and goal detection of barelang-FC," in *Proceedings IES-ETA 2017 - International Electronics Symposium on Engineering Technology and Applications*, 2017, vol. 2017-Decem, pp. 146–151.

[6] S. Hossain and D. Lee, "Deep Learning-Based Real-Time Multiple-Object Detection and Tracking from Aerial Imagery via a Flying Robot with GPU-Based Embedded Devices," *Sensors*, vol. 19, no. 15, p. 3371, 2019.

[7] K. Lu, X. An, J. Li, and H. He, "Efficient deep network for vision-based object detection in robotic applications," *Neurocomputing*, vol. 245, pp. 31–45, 2017.
[8] B. A. G. de Oliveira, F. M. F. Ferreira, and C. A. P. da Silva Martins, "Fast and Lightweight Object Detection Network: Detection and recognition on resource constrained devices," *IEEE Access*, vol. 6, pp. 8714–8724, 2018.
[9] I. Ha, Y. Tamura, H. Asama, J. Han, and D. W. Hong, "Development of open humanoid platform DARwIn-OP," *Proc. SICE Annu. Conf.*, pp. 2178–2181, 2011.
[10] D. M. Montserrat, Q. Lin, J. Allebach, and E. J. Delp, "Training object detection and recognition CNN models using data augmentation," *Electron. Imaging*, vol. 2017, no. 10, pp. 27–36, 2017.
[11] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vis.*, vol. 104, no. 2, pp. 154–171, 2013.
[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.
[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[15] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," pp. 1–15, 2014.
[16] T. Fushiki, "Estimation of prediction error by using K-fold cross-validation," *Stat. Comput.*, vol. 21, no. 2, pp. 137–146, 2011.