ISSN: 2581-1274

A YOLO-Based Machine Learning Framework for Detection of Soft Pneumatic Actuator Bending Angles

Syahirul Alim Ritonga [a,1], Raditya Fadhil Arva [a,2], Sarah Iftin Atsani [b,3], Mohammad Ardyansah [a,4], Herianto [a,5*]

[a] *Departement of Mechanical and Industrial Engineering, Universitas Gadjah Mada, Jl. Grafika 2, Yogyakarta 55281, Indonesia*
[b] *Center of Additive Manufacturing and Systems, Universitas Gadjah Mada, Jl. Grafika 2, Yogyakarta 55281, Indonesia*
[1]*syahirul.alim.r@ugm.ac.id;* [2]*raditya.fad2003@mail.ugm.ac.id ;* [3]*sarah.iftin.a@mail.ugm.ac.id;*
[4]*mohammad.ardyansah@mail.ugm.ac.id;* [5]*herianto@ugm.ac.id**
*Corresponding author

ARTICLE INFO

Article history:
Published

Keywords:
Soft Pneumatic Actuator
Soft Robot
Python
Machine Learning
Angle Measurement

ABSTRACT

The bending angle of soft pneumatic actuator (SPA) is a critical parameter influencing their reliability and effectiveness across various applications. Conventional measurement methods are often labor-intensive and impractical for experiments requiring multiple trials, creating a need for efficient, non-invasive techniques. This study proposes a machine learning framework leveraging YOLO (You Only Look Once) models to detect SPA bending angles from image data, eliminating the need for additional hardware. A comprehensive dataset of SPAs under varying actuation pressures, with meticulously labelled bending angles, was curated to train a YOLO-based regression model. The results highlight the model's strong performance, achieving a recall of 99.1%, precision of 70%, and mean average precision (mAP) scores of 86.42% (IoU 0.5) and 84.35% (IoU 0.5–0.95). Low training and validation losses indicate high accuracy in bounding box predictions, object-background differentiation, and object classification. Optimized learning rates ensured efficient parameter updates, achieving convergence without overfitting. The proposed framework demonstrates a robust balance between accuracy, robustness, and efficiency, making it a practical solution for reliable SPA bending angle detection in real-world applications. This study underscores the potential of machine learning-driven techniques to streamline SPA characterization, offering a scalable and non-invasive alternative to traditional methods.

I. Introduction

Soft pneumatic actuators (SPAs) have garnered significant attention in recent years due to their inherent flexibility, compliance, and adaptability [1], which make them ideal for applications in robotics [2], healthcare devices [3], and assistive devices [4]. Unlike traditional rigid actuators, SPAs can safely interact with delicate objects [5] and conform to complex surfaces [6], enabling innovative solutions in fields such as soft robotics and wearable devices [7]. Despite their potential, accurately quantifying and controlling the bending motion of SPAs remains a critical challenge, as their behavior is influenced by nonlinear material properties [8], geometric design [9], and actuation pressures [10].

One critical aspect of SPA characterization is the measurement of bending angle performance, which directly impacts their reliability and efficacy in various applications [11]. Precise quantification of the bending angle is crucial to ensure consistent and predictable operation. Conventional methods for angle measurement often involve manual procedures [12], which are not only labor-intensive but also impractical for experiments requiring multiple trials. Another advance method is by implementing external sensors such as flex sensors [13] or strain sensors [14]. While effective, these methods can introduce complexities such as fabrication difficulties and increased system cost. As a result, there is a growing interest in non-invasive, data-driven techniques that leverage machine

learning (ML) to infer the bending angle from image or sensor data without requiring additional hardware.

Machine learning has established itself as a transformative tool in addressing complex problems by capturing intricate, nonlinear relationships within data [15]. Unlike traditional methods, which often rely on predefined assumptions and linear models, machine learning algorithms adapt to data patterns, uncovering hidden relationships and providing robust predictions [16]. Specifically, convolutional neural networks (CNNs) have emerged as a standout technique in image-based applications due to their ability to automatically learn hierarchical feature representations directly from raw image data [17]. This capability is particularly advantageous in tasks requiring high accuracy in feature extraction and regression, such as predicting continuous variables, where CNNs excel in identifying subtle variations and spatial patterns in images [18].

In practical applications, machine learning-based image processing has been successfully implemented in diverse fields. For instance, in healthcare, CNNs have enabled breakthroughs in disease detection, such as identifying skin conditions like melanoma, psoriasis, and eczema [19]. These systems analyse large datasets of skin images, learning to distinguish between healthy and diseased tissues based on nuanced visual features. Similarly, in agriculture, machine learning has been used to detect diseases in crops such as corn by analysing images of leaves and identifying symptoms like discoloration or texture changes caused by fungal infections or pest damage [20]. In engineering, CNNs have been employed for crack [21] and fracture detection [22] in materials and infrastructure by processing high-resolution images of surfaces.

This study focuses on developing a machine learning framework to detect the bending angle of SPAs using image data. Building upon the foundational principles of CNNs, YOLO (You Only Look Once) models have been developed to perform object detection and classification with remarkable efficiency. YOLO's ability to predict bounding boxes and object classes simultaneously offers an optimized framework for tasks that require real-time performance. By utilizing CNN-based YOLO models, the proposed framework aims to achieve robust angle prediction across various actuation pressures.

In this study, a comprehensive dataset of SPAs under varying actuation pressures was curated, with corresponding bending angles meticulously labelled. A YOLO-based regression model was designed, trained, and optimized to predict the bending angle from images. Finally, the proposed model was evaluated against established metrics, demonstrating its efficacy and potential for real-world applications.

By addressing the challenges associated with SPA angle detection through a CNN and YOLO-based approach, this study contributes to advancing in soft robotics and provides a foundation for further integration of data-driven methods in the field.

II. Method

A. Machine learning process

The machine learning process begins with recording a soft pneumatic actuator from various angles with specific rotation variations. The recorded videos are edited to replace the background with black, followed by frame extraction using a Python program ("frame_extractor.py") to isolate non-black pixels. An annotation program ("annotator.py") automatically generates .txt files for each image, providing necessary annotations for training. The dataset is split into training and validation sets using the "splitter.py" program, and images are converted to grayscale with "grayscaler.py." Next, the dataset is prepared for YOLO (You Only Look Once) training by defining dataset paths in a configuration file ("dataset.yaml"). The training process is initiated, and performance data is extracted. Finally, real-world testing is conducted using "predictor.py" to validate the results. The specifics of each process are outlined in the following section.

B. Dataset Preparation

The first step involved recording the object (soft pneumatic actuator) from various angles with different bending angles: 0°, 60°, 70°, 80°, and 90°. The recording was performed using an external

camera (JETE HD) and OBS Studio software. Each video had a duration of approximately one minute for each bending angle variation (Figure 1).

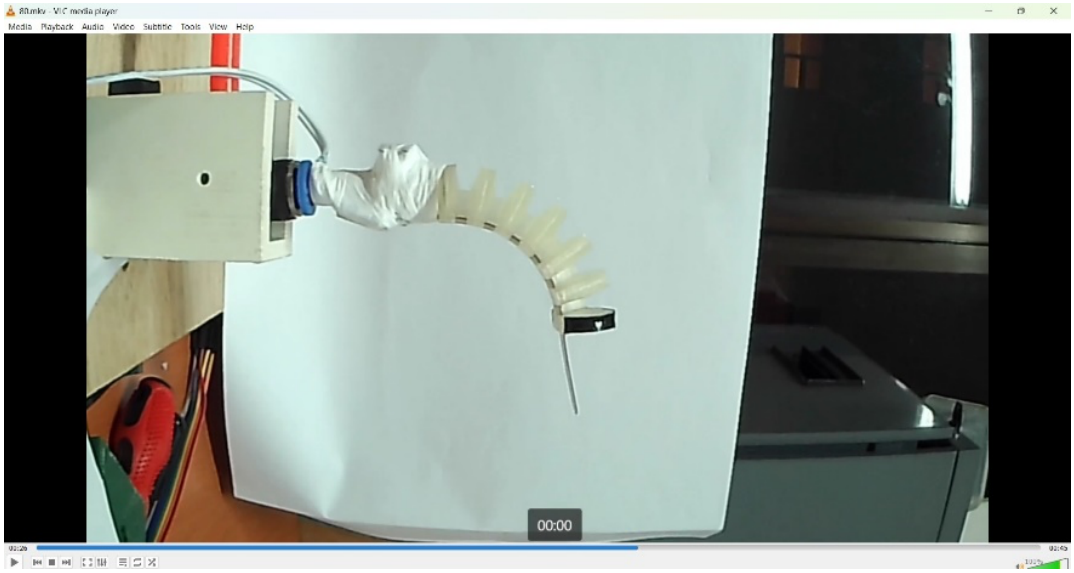

Fig. 1. Video recording of various bending angles.

C. *Dataset Preprocessing: Cropping*

The five resulting videos were cropped using ShotCut software. Cropping focused on creating a box containing only the object, with the background outside the box automatically turned black (Figure 2). This cropping was done manually, adjusting to the object's position in the video, which was naturally unstable. Additionally, irrelevant parts of the video, such as when the object moved out of the camera's frame, were cut, leaving video durations of approximately 40 seconds. The edited videos were rendered with a limit of 10 FPS to optimize the training process.

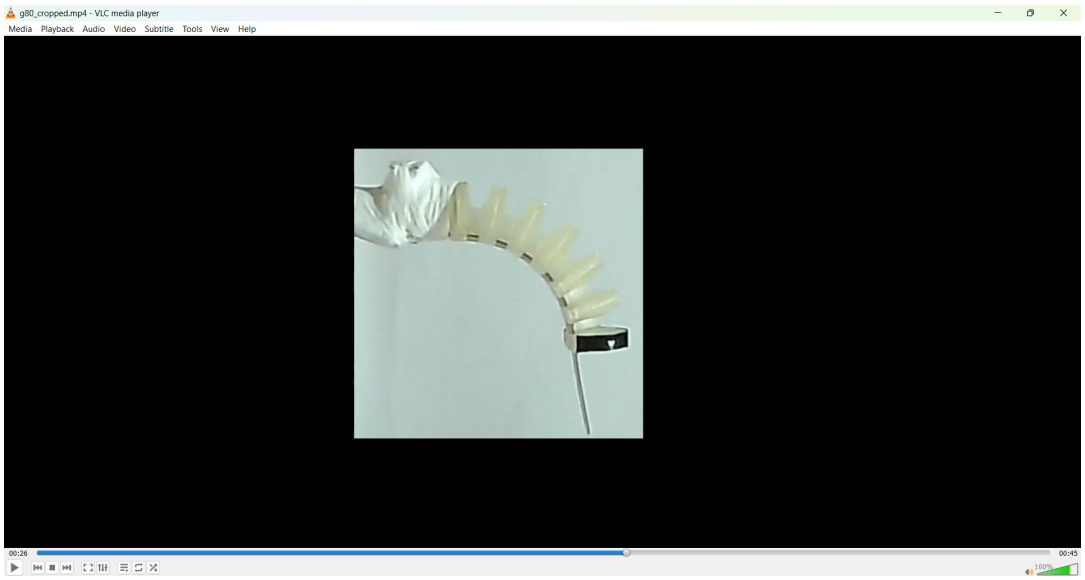

Fig. 2. Result of cropping process.

D. *Dataset Preprocessing: Frame Extraction*

The "frame_extractor.py" program was used to extract individual frames from the videos, excluding black areas. This process generated .jpg images for each frame, containing only the object. The same process was repeated for all videos. The extracted frames were then divided into five *classes* based on their bending angle variation. The file naming convention was automatically

generated, starting with "0_0000.jpg" for the first frame of the first *class* and ending with "90_0442.jpg" for the last frame of the last *class*. The *class* distribution and the number of datasets were listed in Table 1, which resulted in a total of 2212 datasets.

Table 1. Number of datasets for each angle

No	Angle (°)	Number of Datasets
1	0	446
2	60	439
3	70	433
4	80	451
5	90	443

E. Dataset Preprocessing: Annotation File

Frame annotation process is summarised in Figure 3. YOLO requires annotations for each dataset, which are stored in .txt files with names matching their corresponding dataset files. The annotation format used is: "*class*_id x_centre y_centre width height." Here, the `*class*_id` is an index number representing the dataset's *class*, where 0 corresponds to 0°, 1 to 60°, 2 to 70°, 3 to 80°, and 4 to 90°. The `x_centre` and `y_centre` denote the normalized coordinates (on a scale from 0 to 1) of the bounding box center. Since all datasets underwent prior editing, these values are set to 0.5. Meanwhile, the `width` and `height` represent the normalized dimensions of the bounding box, which are set to 1 for all files.

For example, in the first dataset of the first *class*, the image file is named "0_0000.jpg," and its corresponding annotation file is named "0_0000.txt." The content of the annotation file is "0 0.5 0.5 1.0 1.0." Similarly, for the last dataset of the last *class*, the image file is named "90_0442.jpg," with the annotation file named "90_0442.txt," containing the annotation "4 0.5 0.5 1.0 1.0." The generation of these annotation files was automated using a program called "annotator.py."

F. Dataset Preprocessing: Training and Validation Dataset

All datasets were randomly split into two groups: training datasets and validation datasets, with a ratio of 3:1. The training dataset was used for model training, while the validation dataset was used for performance evaluation metrics such as accuracy and precision. The datasets were separated into two directories: "_train" for the training dataset and "_valid" for the validation dataset. Each directory contained five folders, one for each *class*, with 1657 and 555 datasets respectively. This process was automated using the "splitter.py" program.

G. Dataset Preprocessing: Gray scaling

The next step applied a grayscale filter to the dataset images (Figure 4). This enhanced model performance by eliminating lighting factors, allowing the model to focus solely on the object's shape. The "grayscaler.py" program automatically converted colored images to grayscale without altering their location or filename, maintaining the dataset's directory structure.

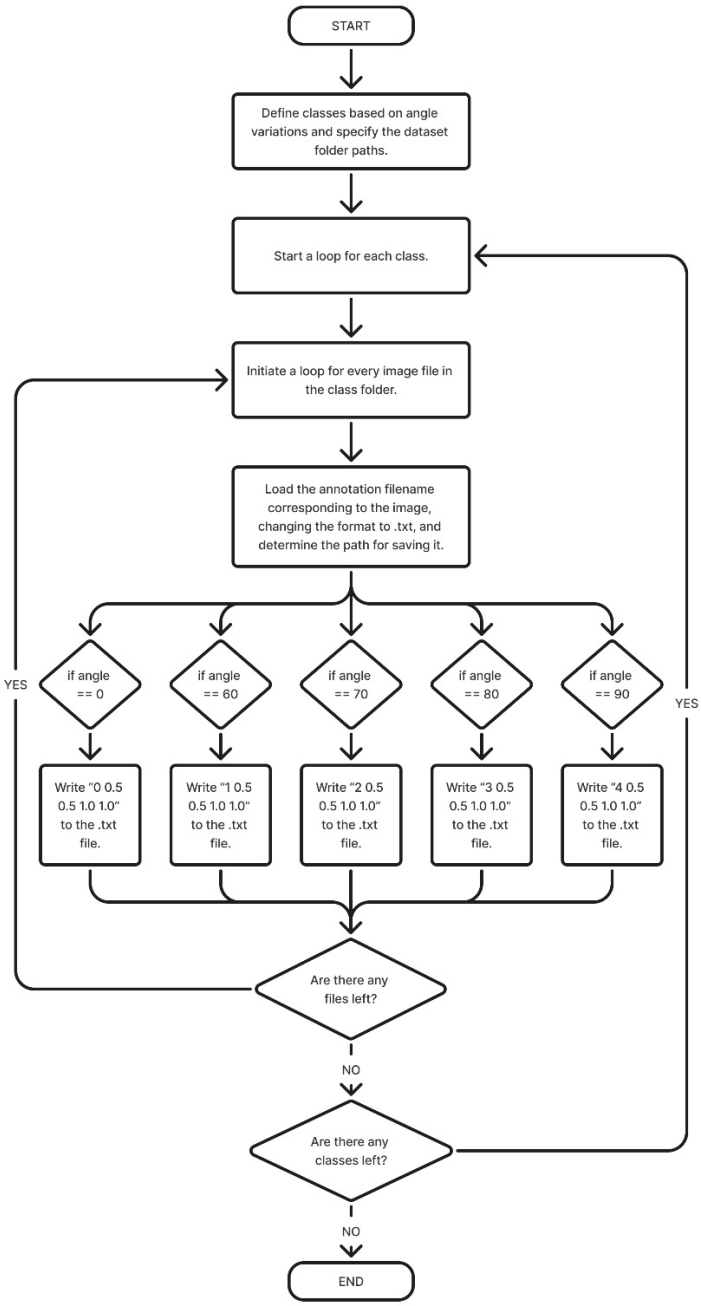

Fig. 3. Flowchart of annotation process.

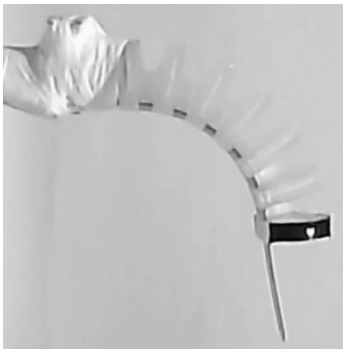

Fig. 4. Grayscale filter image.

H. YOLO Environment Setup

This step was divided into three parts. First, a Python-based environment for YOLO training was prepared by installing necessary modules, including PyTorch and Ultralytics. YOLOv5 was selected as the version. Next, the dataset directory was imported into the environment. Finally, a "dataset.yaml" file was created to define the dataset's folder paths and *class* distribution.

I. Machine learning program code

1) Frame Extractor

In this program, a *looping* is controlled by an if-else condition (Figure 5) to check whether cv2 (OpenCV) can still read frames from the video. The *looping* continues as long as cv2 detects frames. When the *looping* reaches the last frame of the video, on the next iteration, cv2 will no longer detect any frames, causing the loop to stop.

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

Fig. 5. If-else condition.

Within the *looping*, the program creates a mask to eliminate all black pixels (background) in the frame. The RGB value of black is (0, 0, 0), and the mask captures pixels with RGB values ranging from (1, 1, 1) to (255, 255, 255). Once the mask is created, contours are generated in the masked area using code in Figure 6. The frame is then cropped, retaining only the image within the contours.

```
mask = cv2.inRange(frame, (1, 1, 1), (255, 255, 255))
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Fig. 6. Masking and contour code.

The resulting image is saved in a predefined output folder. The file format is "class_count:04d.jpg". The "count:04d" format specifies that the count (numbering variable) is written in four digits, starting from 0000, 0001, 0002, and so on. The count value increments by one in each iteration.

2) Annotator

The first step is creating a dictionary where the keys are the names of class folders, and the values are variations of angles as integers (Figure 7).

```
class_folders = {
    'g0_cropped_frame': 0,
    'g60_cropped_frame': 60,
    'g70_cropped_frame': 70,
    'g80_cropped_frame': 80,
    'g90_cropped_frame': 90,
}
```

Fig. 7. Class folders.

Looping starts with each *class* in the dataset folder and continues into a deeper loop for each image in the *class* folder. For each image, a new variable is created to store the annotation file name (identical to the image name but with a .txt extension) and another variable to store its path. The text file is then filled with annotation data, following the format explained in the process description (Figure 8). The content is categorized based on the class where the file is located.

```
with open(annotation_file_path, 'w') as f:
    if angle == 0:
        f.write("0 0.5 0.5 1.0 1.0\n")
    elif angle == 60:
        f.write("1 0.5 0.5 1.0 1.0\n")
    elif angle == 70:
        f.write("2 0.5 0.5 1.0 1.0\n")
    elif angle == 80:
        f.write("3 0.5 0.5 1.0 1.0\n")
    elif angle == 90:
        f.write("4 0.5 0.5 1.0 1.0\n")
```

Fig. 8. Annotation file name

3) Splitter

The first step is defining paths for the dataset folder (overall), training dataset folder, and validation dataset folder. Then, a train_split variable is defined as an integer with a value of 0.75. For each class, a list is created containing all the files in the class, collecting each image and its annotations, shuffling all the files, and storing them in a zip.

Next, the files are divided with a 3:1 ratio (3 for training and 1 for validation) based on train_split. First, a split_index variable is created to determine the 0.75 position in the dataset (Figure 9). The training dataset includes all files to the left of the split_index (including split_index), while the validation dataset includes all files to the right of the split_index.

```
split_index = int(len(images) * train_split)
train_images = images[:split_index]
train_annotations = annotations[:split_index]
val_images = images[split_index:]
val_annotations = annotations[split_index:]
```

Fig. 9. Splitter code

The final step is moving the image and annotation files into their respective folders. Classes are no longer separated at this stage. All files from all classes are grouped in the same folder for both training and validation.

4) Grayscale converts

After defining the paths for the training and validation folders, the loop begins. The program processes only .jpg files. These files are read and converted to grayscale using OpenCV (Figure 10). Then, the old (coloured) file is replaced with the new (grayscale) file.

```
for filename in os.listdir(directory):
    if filename.endswith(".jpg"):
        img_path = os.path.join(directory, filename)
        img = cv2.imread(img_path)
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        cv2.imwrite(img_path, gray_img)
```

Fig. 10. Grayscale converter

5) Predictor

After loading the AI model trained using PyTorch via ultralytics/yolov5 into a “model” variable, the next step is activating video capture using OpenCV and entering the looping (Figure 11). In the looping, OpenCV reads frames. If an issue occurs, such as an incorrect camera port, a damaged camera, or an error preventing OpenCV from capturing frames, the looping stops automatically, and

the program terminates. Otherwise, the looping continues until interrupted by pressing the "q" key or forcibly stopping the program.

After reading a frame, OpenCV converts it to grayscale to match the data used for training the model. The model's prediction output provides confidence values for each class in the frame. Each class has its value, but the selected class is the one with the highest confidence, provided it surpasses the detection threshold. For example, when the camera is directed at an object forming a 60° angle, the 60° *class* will have the highest confidence and be selected. However, if no object is captured by the camera, all *class* values fall below the threshold, and the model considers no object detected, resulting in no class being chosen.

```
results = model(input_frame)
annotated_frame = np.squeeze(results.render())
cv2.imshow('YOLO Real-Time Detection', annotated_frame)
```

Fig. 11. Loading model

A new frame is then created based on the initial frame, with an added box indicating the object's location and the *class* selected by the model. OpenCV displays this frame on the screen. The process repeats very quickly (depending on the laptop and camera performance), presenting real-time prediction results.

III. Result and Discussion

A. Model Training

The training process utilized parameters detailed in Table 2, requiring approximately four hours to complete, depending on the computer's performance and the selected parameters. Following the training, the model's performance was evaluated across various metrics, illustrated in Figure 12.

Table 2. Parameters used in training process.

No	Parameters	Value
1	Image size	640
2	Batch	32
3	Epochs	10
4	Initial weights	yolov5s.pt

The Training Box Loss, which measures the error in predicting bounding box positions and sizes, achieved a low value of 0.76%, indicating good accuracy. The Training Object Loss, reflecting the error in object detection within prediction boxes, was 0.51%, demonstrating effective object-background differentiation. Additionally, the Training Class Loss, which captures classification errors of objects during training, reached 1.8%, signifying satisfactory accuracy. Precision, a measure of the model's ability to avoid false positives, was 70%, while Recall, indicating the ability to detect all objects, achieved an impressive 99.1%.

Further evaluation showed that the Mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 was 86.42%, and the mAP across IoU thresholds from 0.5 to 0.95 was 84.35%, highlight the model's consistent accuracy in object detection under varying conditions. On validation datasets, the Validation Box Loss, representing errors in bounding box predictions, was 0.25%, and the Validation Object Loss, indicating detection errors, was 0.22%. The Validation Class Loss, capturing classification errors during validation, was recorded at 0.68%. Finally, the learning rates, represented as x/lr0, x/lr1, and x/lr2 for optimizer components, ensured optimal parameter updates

during training, achieving a value of 0.2%. These results collectively highlight the model's robustness and accuracy in detecting and classifying objects.

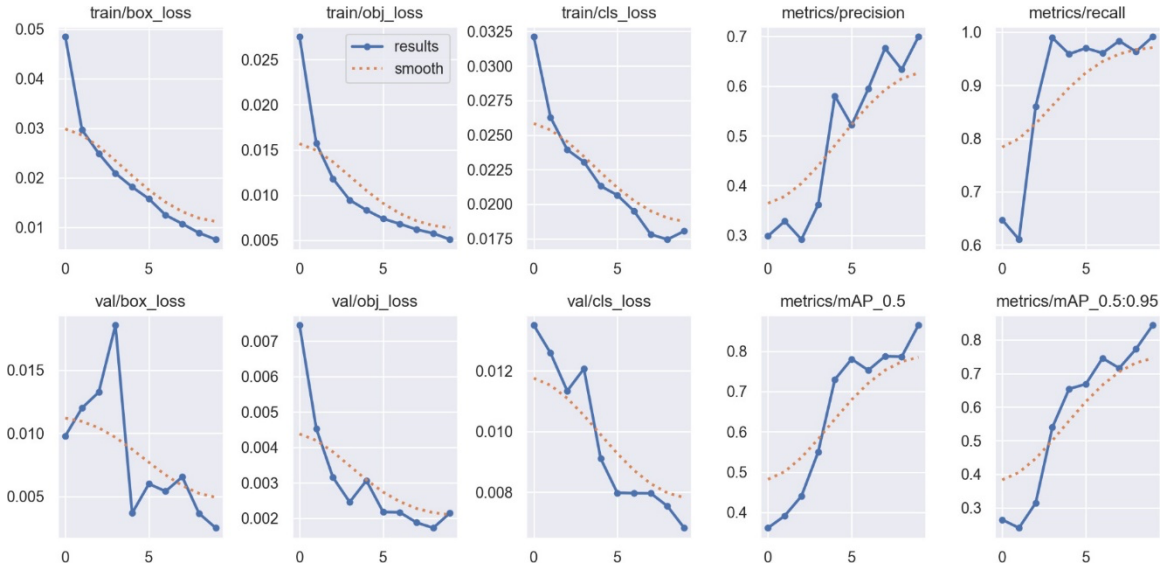

Fig. 12. Model's performance results.

B. *Model Testing*

The testing process was conducted using the "predictor.py" program to validate the results. The program captured video in real time, converted video frames to grayscale, and applied the trained model to detect and classify objects (soft pneumatic actuators) based on their bending angles. Real-time detection screenshots were analyzed as shown in Figure 13, showing detected classes (actuator angles) and their confidence values (0 to 1 range) in the top-left corner of the bounding box. In cases where YOLO detected more than one class simultaneously, the class with the highest confidence value was displayed prominently.

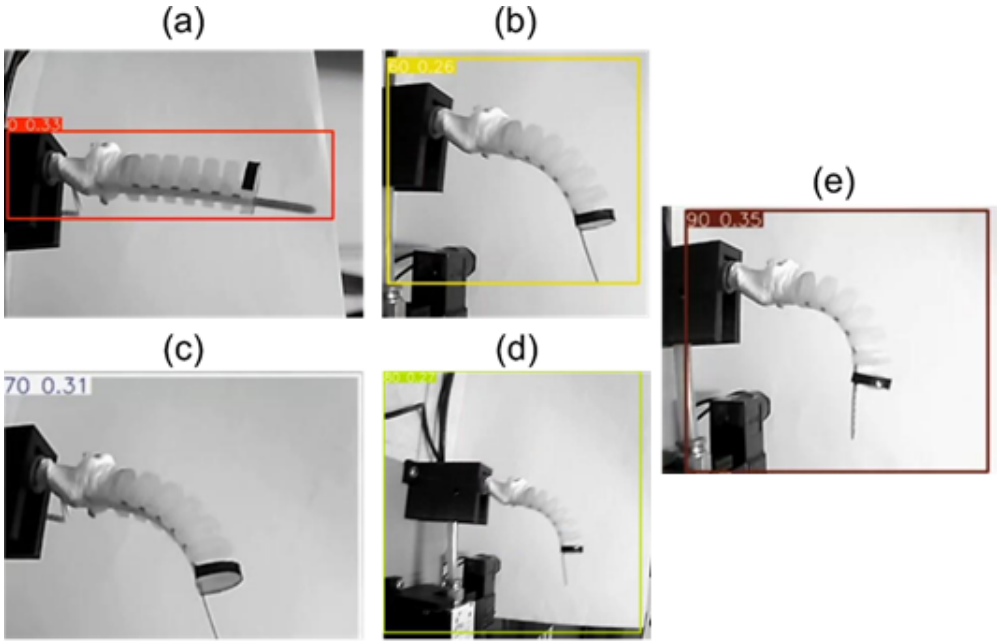

Fig. 13. Real-time detection of SPA in different condition: (a) 0° with confidence value of 0.33; (b) 60° with confidence value of 0.26; (c) 70° with confidence value of 0.31; (d) 80° with confidence value of 0.27; (e) 90° with confidence value of 0.35.

IV. Conclusion

This study develops a machine learning framework to detect the bending angle of soft pneumatic actuators (SPAs) using image data. A curated dataset of SPAs under varying actuation pressures, with labelled bending angles, forms the basis for training a YOLO-based regression model designed to predict bending angles from images. The model's performance, evaluated using established metrics, demonstrates its effectiveness and potential for practical applications.

The process involves recording videos of SPAs from multiple angles with rotational variations, editing the videos to replace the background with black, and extracting frames with a Python program. Non-black pixels are isolated, and annotations for training are generated. The dataset is then split into training and validation sets, and images are converted to grayscale with. The dataset is prepared for YOLO training by defining paths in a configuration file. Finally, training is conducted, and performance metrics are analysed.

The training results demonstrate that the model is highly effective in detecting and classifying objects, with strong performance across multiple metrics. The low training and validation losses, including box, object, and class losses, indicate the model's high accuracy in predicting bounding box positions, distinguishing objects from the background, and classifying objects correctly. The high recall of 99.1% reflects the model's exceptional ability to detect nearly all objects, while the precision of 70% shows moderate success in minimizing false positives.

The mAP scores, particularly 86.42% at an IoU threshold of 0.5 and 84.35% across a range of thresholds from 0.5 to 0.95, highlight the model's consistent accuracy in object detection under varying conditions. Additionally, the optimized learning rates contributed to efficient parameter updates during training, ensuring convergence without overfitting. Overall, the model demonstrates a balance between accuracy, robustness, and efficiency, making it well-suited for applications requiring reliable SPA's bending angle detection and classification.

Acknowledgment

This research was funded by the Academic Excellence Improvement Scheme C Program Universitas Gadjah Mada Number 6530/UN1.P1/PT.01.03/2024.

References

[1] M. S. Xavier et al., 'Soft Pneumatic Actuators: A Review of Design, Fabrication, Modeling, Sensing, Control and Applications', IEEE Access, vol. 10, pp. 59442–59485, 2022, doi: 10.1109/ACCESS.2022.3179589.

[2] W. Hu, R. Mutlu, W. Li, and G. Alici, 'A Structural Optimisation Method for a Soft Pneumatic Actuator', Robotics, vol. 7, no. 2, 2018, doi: 10.3390/robotics7020024.

[3] M. G. Antonelli, P. Beomonte Zobel, F. Durante, and T. Raparelli, 'Additive Manufacturing Applications on Flexible Actuators for Active Orthoses and Medical Devices', J. Healthc. Eng., vol. 2019, no. 1, p. 5659801, 2019, doi: 10.1155/2019/5659801.

[4] P. H. Nguyen and W. Zhang, 'Design and Computational Modeling of Fabric Soft Pneumatic Actuators for Wearable Assistive Devices', Sci. Rep., vol. 10, no. 1, p. 9638, Jun. 2020, doi: 10.1038/s41598-020-65003-2.

[5] C. Tawk and G. Alici, 'A Review of 3D-Printable Soft Pneumatic Actuators and Sensors: Research Challenges and Opportunities', Adv. Intell. Syst., vol. 3, no. 6, p. 2000223, 2021, doi: 10.1002/aisy.202000223.

[6] M. Li, A. Pal, A. Aghakhani, A. Pena-Francesch, and M. Sitti, 'Soft actuators for real-world applications', Nat. Rev. Mater., vol. 7, no. 3, pp. 235–249, Mar. 2022, doi: 10.1038/s41578-021-00389-7.

[7] H. A. Sonar, J.-L. Huang, and J. Paik, 'Soft Touch using Soft Pneumatic Actuator–Skin as a Wearable Haptic Feedback Device', Adv. Intell. Syst., vol. 3, no. 3, p. 2000168, 2021, doi: 10.1002/aisy.202000168.

[8] G. Alici, T. Canty, R. Mutlu, W. Hu, and V. Sencadas, 'Modeling and Experimental Evaluation of Bending Behavior of Soft Pneumatic Actuators Made of Discrete Actuation Chambers', Soft Robot., vol. 5, no. 1, pp. 24–35, Feb. 2018, doi: 10.1089/soro.2016.0052.

[9] Y. Elsayed, C. Lekakou, T. Geng, and C. M. Saaj, 'Design optimisation of soft silicone pneumatic actuators using finite element analysis', in 2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Jul. 2014, pp. 44–49. doi: 10.1109/AIM.2014.6878044.

[10] D. Maruthavanan, A. Seibel, and J. Schlattmann, 'Fluid-Structure Interaction Modelling of a Soft Pneumatic Actuator', Actuators, vol. 10, no. 7, Art. no. 7, Jul. 2021, doi: 10.3390/act10070163.

[11] G. Zhong, W. Dou, X. Zhang, and H. Yi, 'Bending analysis and contact force modeling of soft pneumatic actuators with pleated structures', Int. J. Mech. Sci., vol. 193, p. 106150, Mar. 2021, doi: 10.1016/j.ijmecsci.2020.106150.

[12] A. Mészáros and J. Sárosi, 'Increasing the Force Exertion of a Soft Actuator Using Externally Attachable Inter-Chamber Plates', Actuators, vol. 12, no. 6, 2023, doi: 10.3390/act12060222.

[13] K. Elgeneidy, N. Lohse, and M. Jackson, 'Bending angle prediction and control of soft pneumatic actuators with embedded flex sensors – A data-driven approach', Mechatronics, vol. 50, pp. 234–247, Apr. 2018, doi: 10.1016/j.mechatronics.2017.10.005.

[14] M. C. Yuen, R. Kramer-Bottiglio, and J. Paik, 'Strain sensor-embedded soft pneumatic actuators for extension and bending feedback', in 2018 IEEE International Conference on Soft Robotics (RoboSoft), Apr. 2018, pp. 202–207. doi: 10.1109/ROBOSOFT.2018.8404920.

[15] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, 'Machine learning on big data: Opportunities and challenges', Neurocomputing, vol. 237, pp. 350–361, May 2017, doi: 10.1016/j.neucom.2017.01.026.

[16] P. Choudhury, R. T. Allen, and M. G. Endres, 'Machine learning for pattern discovery in management research', Strateg. Manag. J., vol. 42, no. 1, pp. 30–57, 2021, doi: 10.1002/smj.3215.

[17] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, 'A survey of the recent architectures of deep convolutional neural networks', Artif. Intell. Rev., vol. 53, no. 8, pp. 5455–5516, Dec. 2020, doi: 10.1007/s10462-020-09825-6.

[18] Y. Liu, H. Pu, and D.-W. Sun, 'Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices', Trends Food Sci. Technol., vol. 113, pp. 193–204, Jul. 2021, doi: 10.1016/j.tifs.2021.04.042.

[19] N. S. ALKolifi ALEnezi, 'A Method Of Skin Disease Detection Using Image Processing And Machine Learning', Procedia Comput. Sci., vol. 163, pp. 85–92, 2019, doi: 10.1016/j.procs.2019.12.090.

[20] B. S. Kusumo, A. Heryana, O. Mahendra, and H. F. Pardede, 'Machine Learning-based for Automatic Detection of Corn-Plant Diseases Using Image Processing', in 2018 International Conference on Computer, Control, Informatics and its Applications (IC3INA), Nov. 2018, pp. 93–97. doi: 10.1109/IC3INA.2018.8629507.

[21] N. Aravind, S. Nagajothi, and S. Elavenil, 'Machine learning model for predicting the crack detection and pattern recognition of geopolymer concrete beams', Constr. Build. Mater., vol. 297, p. 123785, Aug. 2021, doi: 10.1016/j.conbuildmat.2021.123785.

[22] A. Müller, N. Karathanasopoulos, C. C. Roth, and D. Mohr, 'Machine Learning Classifiers for Surface Crack Detection in Fracture Experiments', Int. J. Mech. Sci., vol. 209, p. 106698, Nov. 2021, doi: 10.1016/j.ijmecsci.2021.106698.