IT UNIVERSITY OF COPENHAGEN

Bachelor Project of BSc. Data Science

Development of an Analog Meter Reading System for Edge Devices

STADS code: BIBAPRO1PE

Author: Wei-Jie Wang, 19372 (wewa@itu.dk)

Professor: Paul D. Rosero-Montalvo (paur@itu.dk)

Date: 15-05-2023

Signature:

Table of content

Acknowledgement

I want to express my deepest appreciation to Professor Paul D. Rosero-Montalvo for his patience, helpfulness, and vast knowledge in this field. I am also very grateful to Professor Sebastian Büttrich for introducing me to this research topic. Many thanks to Adam Hartmann-Kruckow, the CMO of Eupry, for engaging in discussions with me about the project and borrowing essential components to help me complete this project. Lastly, I would like to acknowledge the support of my family, partner, and friends. Their belief in me has played a significant role in keeping my motivation high throughout the process.

Abstract

This project aims to develop an analog meter reading system that can be deployed on common edge devices, using a Convolutional Neural Networks called YOLOv5 to capture key objects in images and calculate values on the meters. The evaluation of the object detection model shows high precision and recall rates with solid performance metrics. The analog meter reading system achieved a ±0.584 bar error range on Meter A and a ±3.091 psi error range on Meter B in the worst-case scenario, outperforming previous results on Meter A. The system's size is approximately 2 MB, making it compatible with many edge devices. The automated system could contribute to the current manual meter reading procedure. Future research can focus on improving the system's robustness when handling images in harsh conditions.

1. Introduction

Analog gauges are still used in various industries for monitoring purposes, but this requires on-site personnel to read them and record the data manually, which can be tedious and labor-intensive. However, replacing all the analog gauges with digital solutions can be costly and time-consuming.

Previous research has investigated different methods to automate the analog gauge reading process. The Convolutional Neural Networks (CNN) method shows that it can capture accurate values from the meters and perform well in industrial environments. Since most research does not focus on implementing the gauge reading system into common edge devices, it remains an open challenge.

Considering the above challenges, this project aims to develop an analog meter reading system that can be deployed on common edge devices, which could liberate the manual procedure without heavy investments. The approach includes a CNN named YOLOv5 that captures the key objects in the images and uses the coordinates of the objects with basic trigonometry formula to calculate the values on the meters. Since the system needs to be compatible with edge devices, this project also focuses on downsizing the model and comparing the devices that can run the system.

To define the project's scope, it will focus on round dial meters with one pointer and a non-colored scale [Figure 1]. The highlight of this project is that the system achieves a

±0.584 bar error range on Meter A and a ±3.091 psi error range on Meter B in the worst-case scenario. Furthermore, the system size is around 2 MB, compared to state-of-the-art 300 MB, which is edge device friendly.

The complete implementation code for this project can be found on GitHub at https://github.com/weijiewang87/wei_jie_wang_bachelor_project.
Please refer to the repository for accessing the data and reproducing the system discussed in this paper.

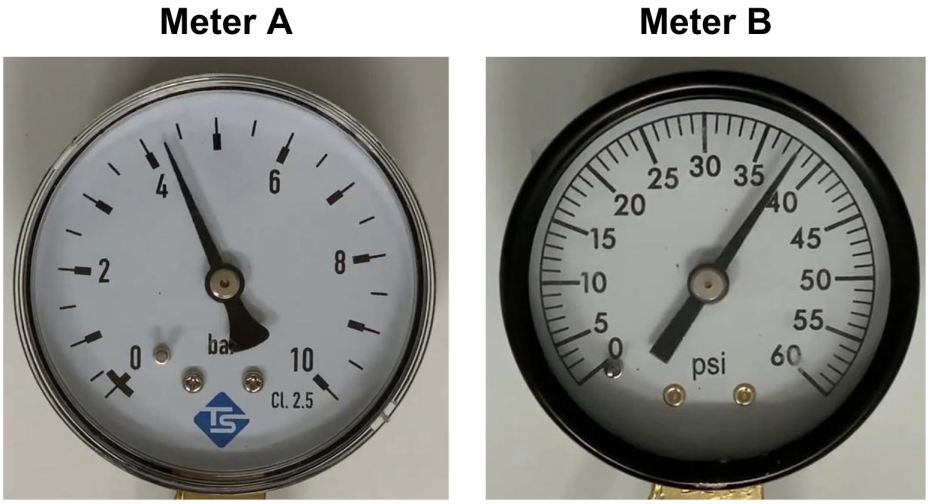


Figure 1. Examples of round dial meters with one pointer and a non-colored scale.

2. Related Work

Researchers have attempted to find a solution to reading the values from analog gauges, and several methods will be introduced that use different techniques, e.g., unsupervised algorithms, computer vision, and CNN.

Lauridsen et al. [1] introduced a solution incorporating two unsupervised algorithms. They used K-Mean Clustering (K-Mean) to detect the key objects from the meter and the Principal Component Analysis (PCA) to obtain the pointer angle. This solution performs well on particular meters, but when there are reflective materials on the meters or blurred motion in the images, the results will be highly influenced.

Peixoto et al. [2] provided a method that used pixel projection to determine the pointer angles. This method does not require much data, and it can capture the angles from various types of meters; however, it is also sensitive to reflection on the meters, and without hard-coded information, e.g., meter scale angle and meter scale value, it can not provide the results.

On the other hand, both Howells et al.[3] and Salomon et al. [4] use CNN in their methods. The biggest difference is that Howells et al. used a CNN called CenterNet to detect the key points from the meters and used the coordinates of the key points to calculate the pointed values. In contrast, Salomon et al. did not use CNN to capture key points and calculate the values; they used CNN to classify the pointed values on the meters. The fundamental

drawback of Salomon et al.'s method is that it can only provide values for a single type of meter. Overall, both methods perform well when there are reflections on the meters, which is more robust toward the actual industrial environments. However, both also require more data for training purposes.

In terms of implementing the gauge reading methods on edge devices, there needs to be more research that sets the focus on this topic. Howells et al. [3] have run their method on the iPhone 7 and the iPhone 11, which yield a decent result. However, the iPhone's memory capacity surpasses most common edge devices. In Peixoto et al.'s [2] research, the ESP32-CAM was used to take and send pictures to a local computer for calculating the pointed values, which does not involve computation from the device. Therefore, developing a method that can run on common edge devices remains an open challenge.

Howells et al. [3] and Lauridsen et al. [1] have significantly contributed to the field by providing publicly available datasets containing images of various types of meters. It is important to note that these datasets do not include corresponding annotations. Nevertheless, their efforts are still noteworthy as the availability of this type of data is limited in academic research.

3. Machine Learning Pipeline

3.1 System Overview

This section will provide an overview of the analog meter reading system. These are the four general steps in the process [Figure 2]. The details of them will be discussed in the upcoming sections.

Step 1: Camera Setup
To ensure that the system works effectively, it is crucial to position the camera correctly. This involves selecting a suitable spot that will enable the camera to capture clear images of the meter while considering the impact of factors such as background, lighting, and reflection.

Step 2: Meter Values Input
Users need to input the minimum and maximum values of the meter into the program. This will enable the system to calculate the value indicated by the meters. This step will only be executed once during the setup, which is acceptable to perform manually. However, in the future, this step can be automated by combining object detection and optical character recognition (OCR) to capture the values.

Step 3: Object Detection
The system will detect and capture specific objects such as the pointer, center, min, and max of the dial meter. This is a repetitive step that will be executed based on the frequency set by the user.

Step 4: Value Calculation

Once the objects are captured, the system will use the coordinates of the objects to calculate the pointer's angle and the pointed value. This step is the same as the previous one, being repetitively executed.

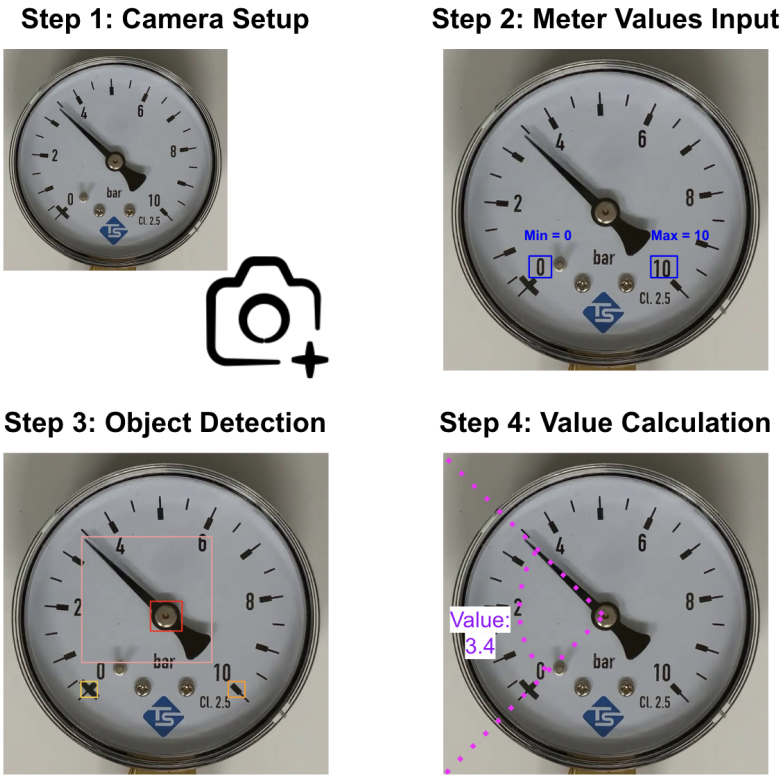


Figure 2. The four general steps of reading analog meters.

3.2 Data Collection

This project primarily utilizes the dataset provided by Howells et al. [3] as they provided high-resolution videos and their prediction results in JSON files. In order to train the model with images, the videos were further separated into one frame per image, which resulted in 450 images of Meter A and 450 images of Meter B. The images were split into training, validation, and test sets by using a 70:20:10 ratio. Note that since the dataset does not come with the true labels, this project has visually measured the true pointed value and used a protractor to measure the true pointed angle for the testing images.

There are many types of meters used in production. For example, some meters have two pointers, some have color scales in the background, and some include both pointer and a mechanical number counter [Figure 3(a)]. To define the scope, this project focused on round, one-pointer meters with no-color scales [Figure 3(b)]. The more diverse the meters are, the more training data there should be. It may cost a considerable amount of time to collect enough samples for various types of meters; therefore, the decision was made for this project to work with conventional pressure meters.

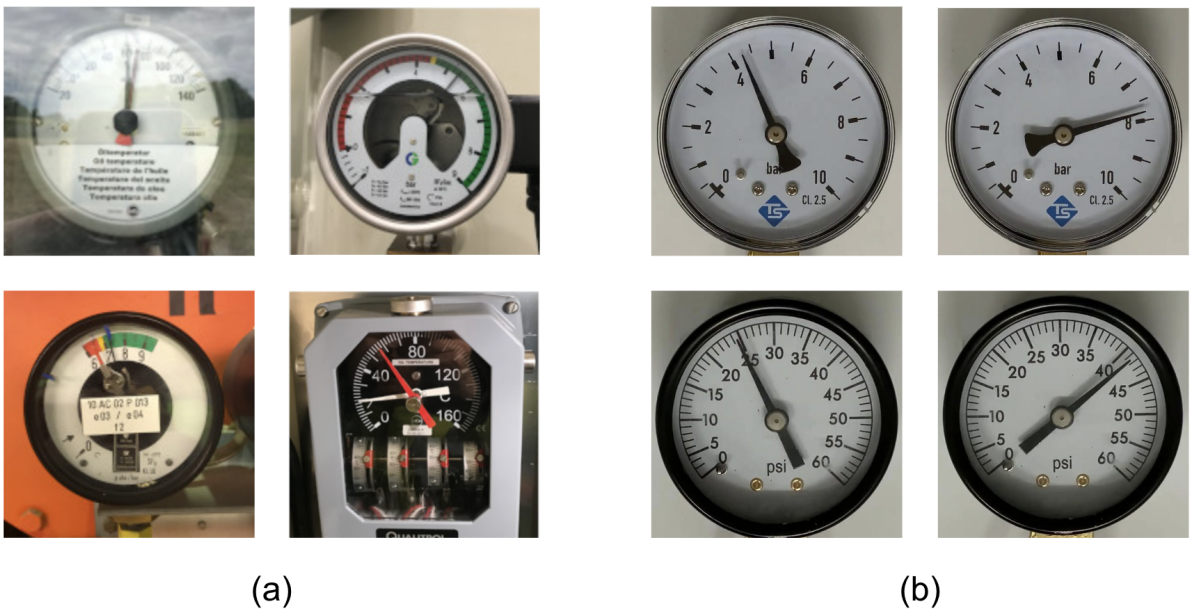


Figure 3. The various type of meters used in production sites (a), and the meters that this project chose to work with (b).

3.3 Data Processing

As Howells et al. [3] did not provide annotated images and annotation methods, annotating the images has been a trial-and-error process. For detecting the whole pointers, it is crucial to position the pointers diagonally inside the bounding boxes [Figure 4(a)]. This approach can help the model comprehend the pointer's orientation. For detecting the tips of the pointers, using a slim rectangular bounding box to locate the tip is crucial [Figure 4(b)]; it helps the model differentiate between the pointer tip and the tick marks. This project also attempted to annotate images with polygons and bounding boxes [Figure 4(c)]. However, this method proved confusing for the model, which is a pitfall that future projects should avoid. As this project mainly focuses on detecting the whole pointer, the annotated data from the other two approaches were not used during the training process.

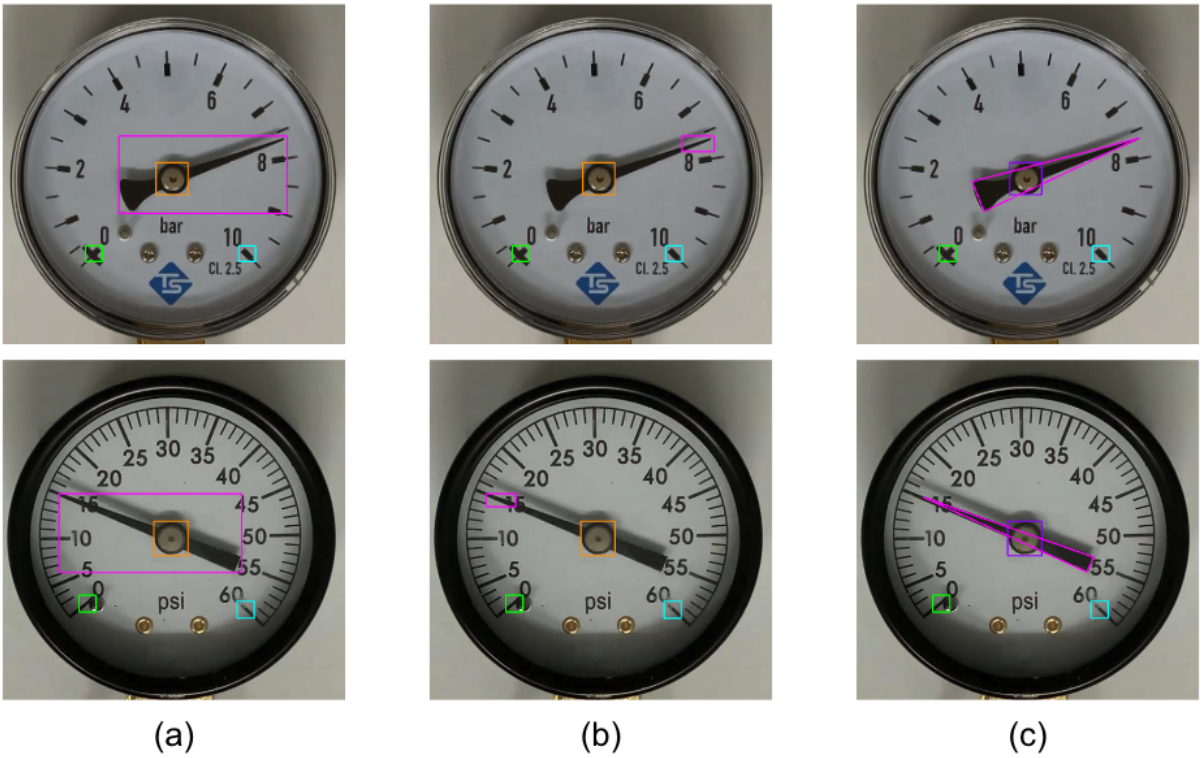


Figure 4. The three different methods of annotating the images.

To investigate whether image augmentation can enhance object detection performance, this project utilized YOLOv5s and half of the available data to compare the accuracy scores of detecting the pointer object before and after applying image augmentation. Initially, the dataset consisted of 315 training, 90 validation, and 45 test images, and the model achieved a 60% accuracy score in detecting the pointer object. After applying the image augmentation techniques shown in Figure 5, the dataset became 945 training, 90 validation, and 45 test images. When retraining the model on this new dataset, it achieved a 90% accuracy score on pointer detection. The result shows that image augmentation can improve the model performance, and as a result, the remaining dataset underwent the same process, which finally obtained 1890 training, 180 validation, and 90 test images.

1. **Flip:** Horizontal, Vertical
2. **90 degree rotate:** Clockwide, Counter-Clockwise, Upside Down
3. **Crop:** 0% Minimum Zoom, 5% Maximum Zoom
4. **Rotation:** Between -5 degrees and +5 degrees
5. **Hue:** Between -6 degrees and +6 degrees
6. **Saturation:** Between -7% and +7%
7. **Brightness:** Between -9% and +9%
8. **Blur:** Up to 1px
9. **Noise:** Up to 5% of pixels
10. **Generate:** 3x augmentations

Figure 5. The details of image augmentation techniques.

3.4 YOLOv5 and Model Selection

YOLOv5 (You Only Look Once version 5) is an object detection system developed by Ultralytics, widely used for object detection and classification in computer vision. The network architecture consists of a backbone network, a neck network, and a head network. The backbone network extracts features from the input image, the neck network fuses the features from different levels, and the head network predicts the bounding boxes and class labels of the objects. The backbone network is pre-trained by using a large dataset called ImageNet, which allows it to learn general patterns and features that are useful for many applications and hence improve performance.

There are several variants of YOLOv5 [Figure 6]. The reason this project choose YOLOv5n is mainly due to model size. YOLOv5n's weight size is around 4MB. When compared to YOLOv5s' 16MB, YOLOv5n is four times smaller. As most microcontrollers and microprocessors have limited random-access memory (RAM), it is important to choose a model that can fit into the device.

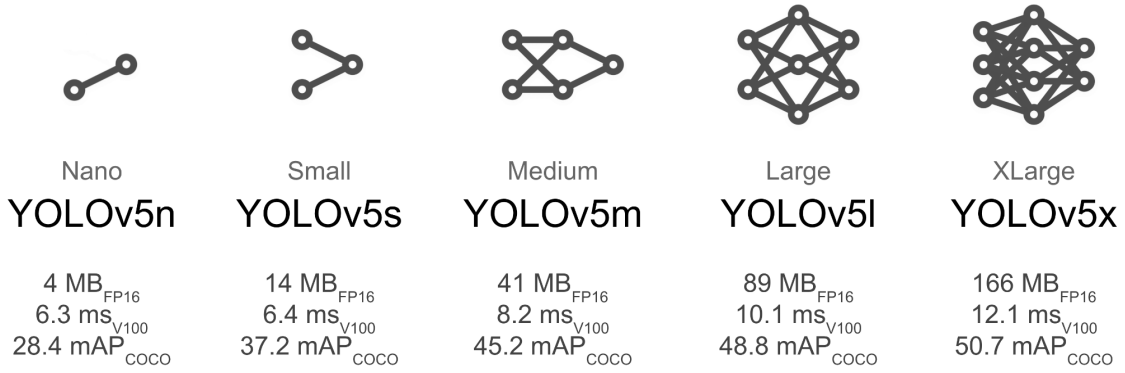


Figure 6, The different variants of YOLOv5.

However, since YOLOv5n has a smaller weight size, it has fewer neurons and a less complex structure. This leads to the result that YOLOv5n does not perform well in detecting specific objects. Table 1 shows that when it tried to detect the whole pointer (Pointer-Detection), it achieved a 91% accuracy score, compared to YOLOv5s' 100% accuracy, which is acceptable. However, when it tried to detect the tip of the pointer (Tip-Detection), it only obtained a 58% accuracy, which is significantly lower than YOLOv5s' accuracy score of 82%.

	Pointer-Detection	Tip-Detection
YOLOv5s	100%	82%
YOLOv5n	91%	58%

Table 1. Comparison between YOLOv5s and YOLOv5n on accuracy scores with two object detection methods.

The reason why YOLOv5n performs poorly on Tip-Detection can relate to the object's shape and the model's complexity. As many ticks in the meter have a similar shape as the tip of the pointer, the model may be confused between these two objects, and even though it can

detect the tip, the confidence level may also be low. Therefore, this is a more complicated task that requires a model with more neurons and a complex structure to be carried out. In contrast, since the shape of the whole pointer is more distinguishable than the tip, YOLOv5n can capture the object with high confidence [Figure 7]. Moreover, in most cases, the model will place the pointer diagonally inside the bounding box that indirectly gives the orientation of the pointer, which can be utilized when calculating the angle and the pointed value. This leads to choosing YOLOv5n with Pointer-Detection as the main object detection method.

Note that the experimental setup of this comparison only used half the amount of training data that has been through the image augmentation process. This is because, by the time this experiment was carried out, only half the data had been annotated.

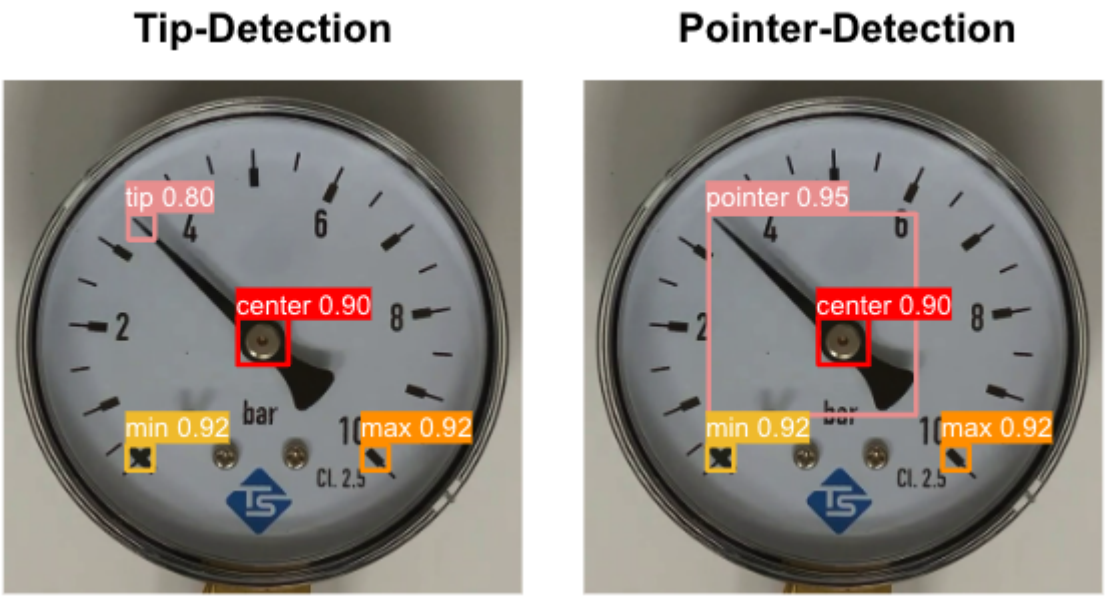


Figure 7. The difference between Tip-Detection and Pointer-Detection.

3.5 Model Deployment

Once the inference has been executed, the model will output the center coordinates of the detected objects, which include Pointer, Center, Min, and Max. For the sake of simplicity, the term "coordinate" will refer specifically to the center coordinate of each object from this point forward.

In Figure 8(a), the coordinates of the Pointer and Center objects are represented by the dots. When using a line to connect the two dots, there will be a slope, which implies the Pointer's orientation. To get the angle of the slope, it is important to first calculate the length of the opposite side (dy) and the adjacent side (dx) by using the coordinates, as seen in Figure 8(b). Then, using the arc tangent formula presented in Figure 9, the slope angle can be calculated. From this point, the angle of the slope will be called theta (θ).

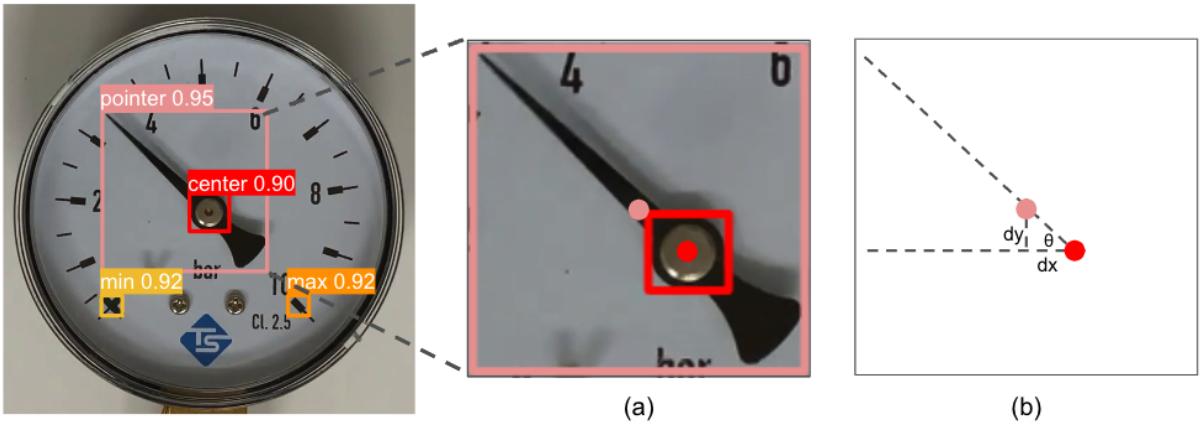


Figure 8. An example of calculating the theta by using the coordinates of Pointer and Center objects.

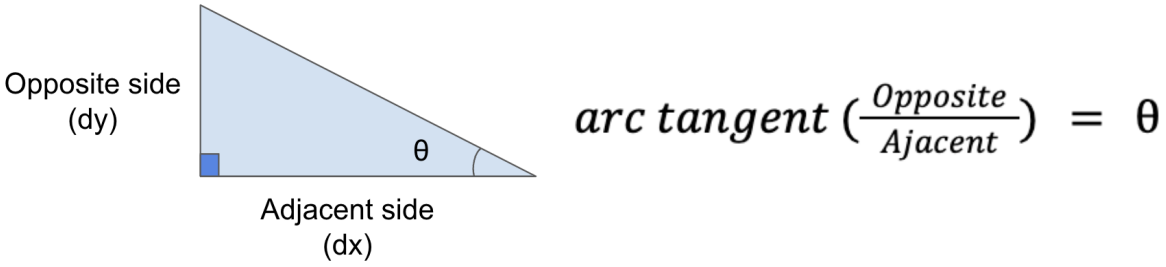


Figure 9. The arc tangent formula for calculating the theta.

However, the angle of interest, in this case, is the one formed between the Min coordinates and the slope, which will be referred to as the "adjusted theta."

To determine the adjusted theta, the first step is to calculate the angle between the Min and Max objects. For instance, in the given example, the angle between these two objects is 88 degrees. The next step is to divide this angle by two, which results in the angle between the Min coordinates and a vertical line passing through the Center, which equals 44 degrees [Figure 10(a)]. This angle can determine the angle between the Min coordinates and a horizontal line passing through the Center, which equals 46 degrees [Figure 10(b)]. Since both theta and the 46-degree angle share the same horizontal line [Figure 10(c)], they can be added together to obtain the adjusted theta, which equals 91 degrees [Figure 10(d)].

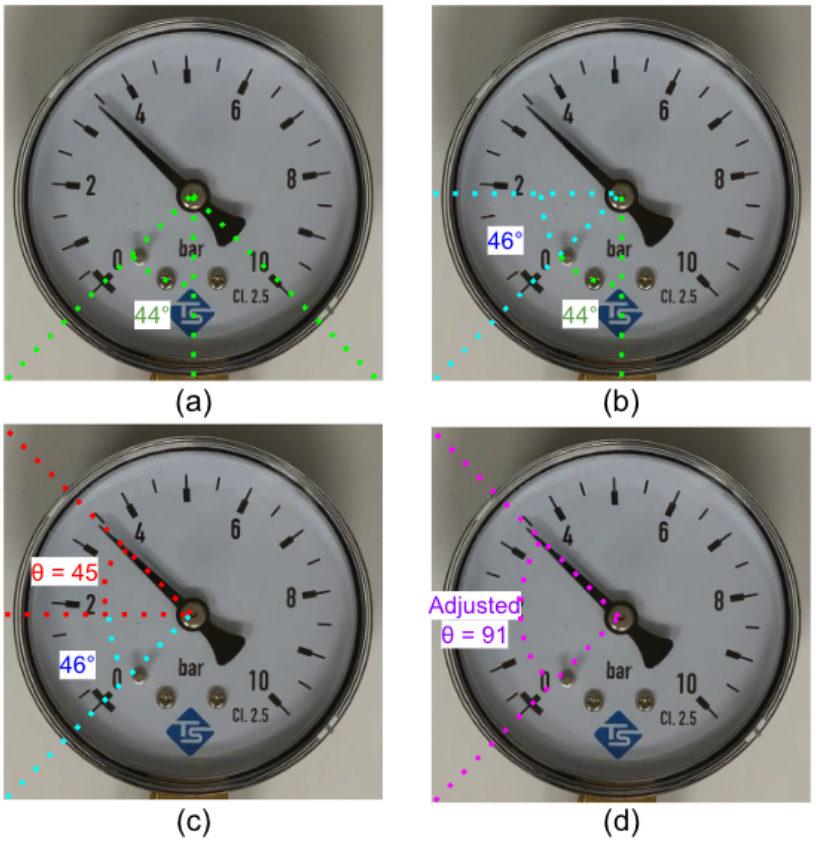


Figure 10. The process of finding adjusted theta.

In the example above, the Pointer is pointing at the top-left position; hence, the adjusted theta is 46 degrees plus theta. When the Pointer is pointing at other positions, however, there will be other ways to calculate the adjusted theta [Figure 11].

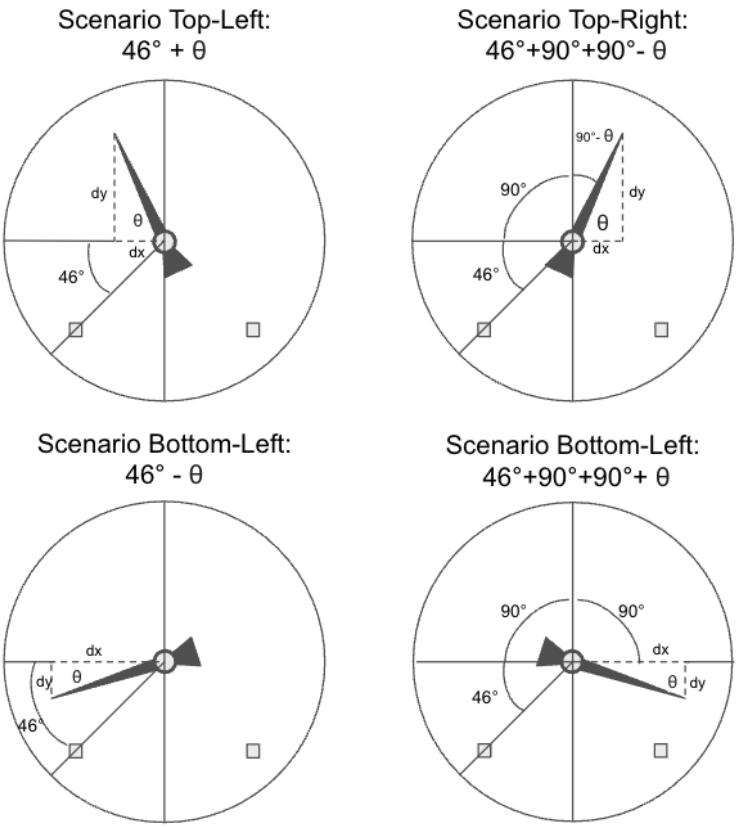


Figure 11. Different ways to calculate the adjusted theta in different scenarios.

The adjusted theta can be used to calculate the pointed value. By dividing the adjusted theta by the degree of the incomplete circle, a ratio is obtained which represents the amount the pointer has moved. This ratio can be mapped onto the scale between the values of Min and Max to yield the pointer value [Figure 12].

$$Ratio = \frac{Adjusted\,Theta}{Degree\ of\ Incomplete\ Circle}$$

$$Pointed\,Value = Ratio \times (Max - Min)$$

Figure 12. The formula for calculating the pointed value.

3.6 Model Optimization

In machine learning, post-training quantization is a technique that reduces the precision or bit of numerical values in a model, such as weights and activations. This could reduce the size of the model without sacrificing too much accuracy, which is beneficial for deploying models on edge devices that have limited memory.

The project has explored three quantization techniques by following TensorFlow's instructions [5]. First, Dynamic range quantization only quantizes the model's weights from floating point to integers with 8-bit precision. It supposes to reduce the model size four times and increase in speed by up to two to three times. Second, Float16 quantization should make the model two times smaller by quantizing the weights from Float 32 to Float 16. Last, 16x8 quantization quantizes the activations to 16-bit integers and the weights to 8-bit integers, making the model 4 times smaller and 3 times faster. [Table 2]

Please note that the target device for this project is the Raspberry Pi Zero HW, which does not necessitate the use of integer-only quantization. Therefore, this project does not investigate further into the topic of integer-only quantization. However, this aspect could be considered for future research.

Technique	Benefits	Details
Dynamic range quantization	4x smaller, 2x-3x speedup	Activation: No change Weight: 8-bit integers
Float16 quantization	2x smaller	Activation: No change Weight: Float 16
16x8 quantization	4x smaller, 3x+ speedup	Activation: 16-bit integer Weight: 8-bit integer

Table 2. The benefits and details of the quantization techniques.

3.7 Model Training Details

For training the model, this project has followed a Google Colab Notebook (notebook) that Roboflow provided. Roboflow provides data management and data annotation tools and is an official partner of the creators of YOLOv5. The notebook provides the foundation for the training process, e.g., installing all required packages, downloading YOLOv5 models, and creating a data.ymal file for training and validation sets. The notebook has a default training configuration that sets image size as 416, batch size as 16, and epochs as 100 [Figure 13].

YOLOv5 provides two training options: using pre-trained weights or training from scratch. Pre-trained weights are model parameters that have been trained on a large-scale dataset, capturing learned representations from other domains. When using pre-trained weights, the model starts with prior knowledge and can recognize common object classes. On the other hand, training from scratch initializes the model with randomly initialized parameters, requiring it to learn everything from the beginning. While both methods yielded good

performance in this project, training from scratch slightly outperformed pre-trained weights and required less training time [Figure 13]. As a result, the project decided to follow the training from scratch approach for its better results and efficiency.

Training with pre-trained weights

```
!python train.py --img 416 --batch 16 --epochs 100 --data {dataset.location}/data.yaml --weights yolov5n.pt --name yolov5n_results --cache
```

```
Model summary: 157 layers, 1764577 parameters, 0 gradients, 4.1 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 6/6 [00:06<00:00,  1.06s/it]
                   all        180        720      0.997      0.999      0.994      0.813
         centerrotation       180        180      0.996          1      0.995        0.9
        pointerrotation       180        180      0.998          1      0.995      0.908
      scale-maxrotation       180        180      0.994      0.994      0.993      0.725
      scale-minrotation       180        180      0.999          1      0.995      0.719
Results saved to runs/train/yolov5n_results2
CPU times: user 25.3 s, sys: 2.69 s, total: 28 s
Wall time: 35min 17s
```

Training from scratch

```
!python train.py --img 416 --batch 16 --epochs 100 --data {dataset.location}/data.yaml --cfg ./models/yolov5n.yaml --weights '' --name yolov5n_results  --cache
```

```
YOLOv5n summary: 157 layers, 1764577 parameters, 0 gradients, 4.1 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 6/6 [00:06<00:00,  1.09s/it]
                   all        180        720      0.999          1      0.995      0.789
         centerrotation       180        180      0.998          1      0.995      0.894
        pointerrotation       180        180      0.999          1      0.995      0.899
      scale-maxrotation       180        180          1          1      0.995       0.68
      scale-minrotation       180        180          1          1      0.995      0.684
Results saved to runs/train/yolov5n_results
CPU times: user 23 s, sys: 2.55 s, total: 25.5 s
Wall time: 33min 55s
```

Figure 13. This figure shows the training details and performance of two training options: training with pre-trained weights and training from scratch.

4. Results

The following section presents the performance of the system for reading values from the analog meters. Firstly, the overall performance of the object detection model is presented. Secondly, a comparison is made between the performance of the quantized models. Finally, a comparison is made between the systems of this project and that of Howells et al. [3].

4.1 Performance of the object detection model (YOLOv5n)

Figure 14 displays three types of loss: box loss, objectness loss, and classification loss. The box loss (box_los) measures the model's ability to locate an object's center and predict bounding box coverage by using Mean Square Error (MSE). Objectness loss refers to the confidence level that an object exists inside the bounding box, where higher confidence results in lower objectness loss (obj_los). Classification loss (cla_los) provides insight into the algorithm's ability to identify the correct class of an object, which uses Cross Entropy as the underlining calculation [6].

By analyzing the graphs from Figure 14, the losses generally become very low after 100 epochs. Since MSE is a metric ranging from zero to infinity, the very low box losses in the training and validation process imply that the predicted bounding box is closely aligned with the true annotation in terms of location and coverage. Additionally, a low objectness loss indicates that the algorithm has high confidence in detecting objects. Finally, the result of

Cross Entropy is supposed to range from 0 to 1, with the class loss that is approaching 0 reflecting the model's ability to classify objects very well.

The precision and recall on the right-hand side of Figure 14 are both around 0.999, indicating that the model generates very few False Positive and False Negative results, and is highly optimized toward True Positive when performing classification. For more information on the performance of each class, refer to the confusion matrix in Figure 15.

Since the losses from both the training and validation processes were steadily decreasing, the model should not have an overfitting issue. When running the inference on the test set, the model successfully provided correct bounding boxes for the images, meaning there were no wrongly labeled objects. This shows that the model can generalize well to the unseen images from the controlled environment.
As the model already performs well, this project does not further experiment with hyperparameter tuning. Note that, in future research, there can be more focus on increasing the robustness of the model since the training data does not include images with harsh conditions.

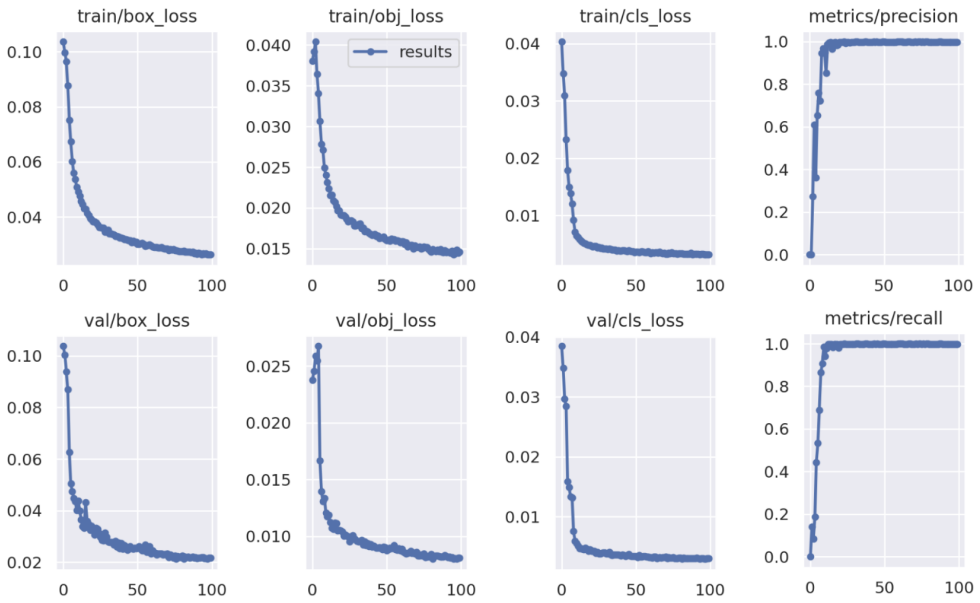


Figure 14. Graphs depicting box loss, objectness loss, classification loss, precision, and recall in training and validation process over 100 epochs.

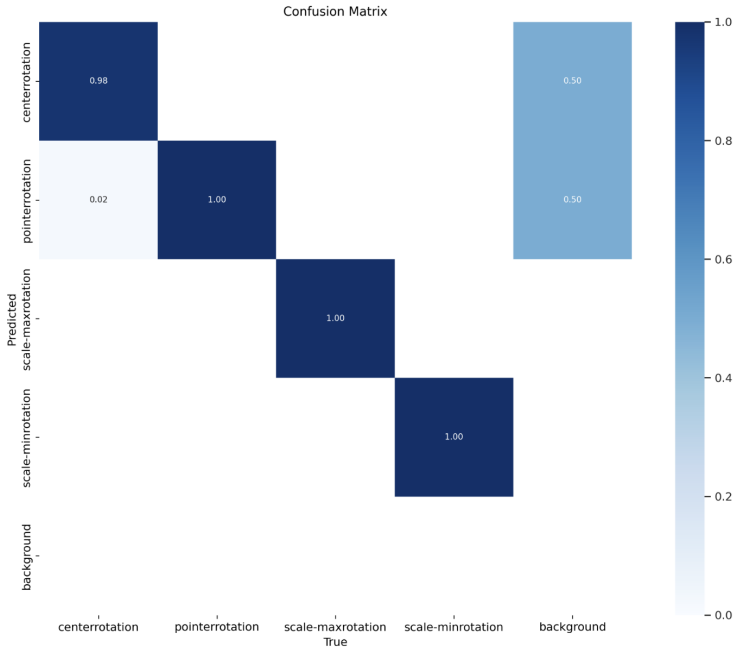


Figure 15. Confusion matrix of the object detection model.

4.2 Performance comparison between the quantization technique

To evaluate the performance of reading the pointed angle and value, this project has used five different metrics. Figure 16 shows the formulas of the four metrics. The fifth metric is the model size which is self-explanatory.

$$\frac{\sum_{i=0}^{n} |A_i - a_i|}{n}$$

(a) Average Absolute Angular Error

$$\frac{\sum_{i=0}^{n} |V_i - v_i|}{n}$$

(b) Average Absolute Deviation Value

$$max(|A_i - a_i|)$$

(c) Max Absolute Angular Error

$$max(|V_i - v_i|)$$

(d) Max Absolute Deviation Value

Figure 16. The four metrics used to evaluate the performance of gauge reading; where A_i is the true angle that measured manually, a_i is the predicted angle from the system, V_i is the true value that determined visually, v_i is the predicted value from the system, and n is the total number of test images.

As mentioned in the model optimization section, this project has explored different quantization techniques. Figure 17 presents a performance comparison between the three techniques. The green color in the figure indicates the lowest value in each metric category. By analyzing the results from Meter A, dynamic range quantization outperforms the other two techniques in every metric. In the case of Meter B, although dynamic range quantization

does not perform the best on the Average Absolute Deviation Value, it still performs better than the other two on the rest of the metrics. In short, dynamic range quantization performs well in both Meter A and Meter B; it is recommended for use in the system.

Meter A

Technique	Average Absolute Angular Error	Max Absolute Angular Error	Average Absolute Deviation Value (bar)	Max Absolute Deviation Value (bar)	Model Size (MB)
Dynamic range quantization	8.277°	16.5°	0.261	0.584	2.0
Float16 quantization	8.511°	16.5°	0.270	0.647	3.7
16x8 quantization	8.455°	16.5°	0.269	0.610	2.0

Meter B

Technique	Average Absolute Angular Error	Max Absolute Angular Error	Average Absolute Deviation Value (psi)	Max Absolute Deviation Value (psi)	Model Size (MB)
Dynamic range quantization	6.811°	18.5°	1.438	3.091	2.0
Float16 quantization	6.900°	19.0°	1.437	3.160	3.7
16x8 quantization	6.777°	18.5°	1.448	3.131	2.0

Figure 17, The comparison of three quantization techniques' performance on Meter A and Meter B.

4.3 This project vs. Howells et al.[3]: Performance Comparison

As the data of Meter A and Meter B are from Howells et al. [3], they have also provided their predicted results. To have a benchmark on how the system performs, this section will compare the performance of this project to Howells et al..

Figure 18 shows that in the case of Meter A, this project's Average Absolute Deviation Value is at 0.261 bar, which is worse than Howells et al. at 0.162 bar. In terms of Max Absolute Deviation Value, this project performs better at 0.584 bar than Howells et al. at 0.750 bar. When looking into other metrics, this project outperforms Howells et al.. Hence, the system from this project is better at reading values from Meter A.

In the case of Meter B, Howells et al. perform better than this project on most of the metrics. They achieve 0.993° on Average Absolute Angular Error and 0.385 psi on Average Absolute Deviation Value, which is very impressive. When analyzing the error range in the worst-case scenario, in terms of Max Absolute Angular Error, Howells et al. perform approximately three times better, with their result at 6.9° and 18.5° from this project. Regarding Max Absolute Deviation Value, Howells et al. are around two times better than this project, with their result at 1.659 psi and 3.091 psi from this project. Undoubtedly, Howells et al. perform very well on Meter B. However, when considering the model from this project is 2 MB compared to

Howells et al.'s model, which is 300 MB, it is still impressive to see that the model from this project is only two to three times worse than Howells et al. in terms of error range in the worst-case scenario.

To summarize, this project outperforms Howells et al. on Meter A, but Howells et al. perform better than this project on Meter B. As the model size from this project is significantly small, it can be perceived as a success in achieving this performance.

Meter A

Model	Average Absolute Angular Error	Max Absolute Angular Error	Average Absolute Deviation Value (bar)	Max Absolute Deviation Value (bar)	Model Size (MB)
This Project	8.277°	16.5°	0.261	0.584	2
Howells et al.	11.404°	42.1°	0.162	0.750	300

Meter B

Model	Average Absolute Angular Error	Max absolute Angular Error	Average Absolute Deviation Value (psi)	Max Absolute Deviation Value (psi)	Model Size (MB)
This Project	6.811°	18.5°	1.438	3.091	2
Howells et al.	0.993°	6.9°	0.385	1.659	300

Figure 18. Comparing the metrics between this project and Howells et al on Meter A and Meter B.

5. Discussion

5.1 Visualizing the deviation value and discussing the findings

By looking at Figure 19, it can be difficult to get an actual feeling of how different the two systems are; thus, the figure visualizes the Average Absolute Deviation Value (green lines) and Max Absolute Deviation Value (red lines) on the Meters A and B.

When looking at Meter A, the green lines from Howells et al. [3] have a narrower gap than the green lines from this project, which means the Average Absolute Deviation Value is smaller. However, the red lines from Howells et al. have a larger gap than the red lines from this project, which means the Max Absolute Deviation Value is larger. In the case of Meter B, it is clear that Howells et al. perform better on both Average Absolute Deviation Value and Max Absolute Deviation Value since the gaps from both green lines and red lines are narrower than the gaps from this project.

When looking at Meters A and B from this project, it can be observed in Figure 19 that the gaps of the deviations are similar, which might be derived from the annotation approach. Since placing the pointer diagonally inside bounding boxes will naturally incorporate an error range, if the annotation is done correctly, the error ranges of these two meters should not be far from each other. However, when looking at Meters A and B from Howells et al. in Figure

19, there is an evident difference between them. In the following paragraphs, two findings can potentially explain this.

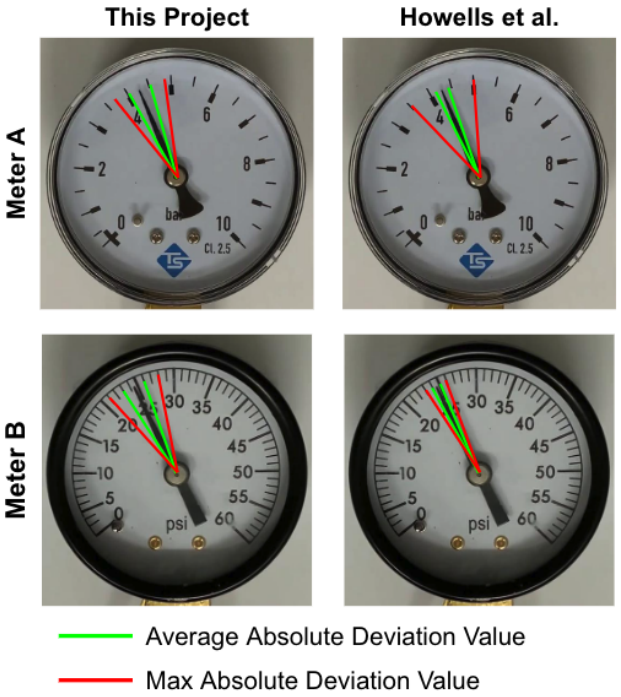


Figure 19. Visualization of deviation range on Meter A and Meter B.

Since the pointed value is calculated from the pointed angle, it is peculiar that Howells et al. [3] obtain a "higher" Average Absolute Angular Error at 11.404° but a "lower" Average Absolute Deviation Value at 0.162 in Meter A. One explanation is that they have adjusted the starting location of the pointer in Meter A, which lowers the Average Absolute Deviation Value. The scale of Meter A is from 0 to 10; thus, the pointer's starting location is supposed to be at 0. However, after investigating their JSON files containing the predicted values related to Meter A, two of them show the starting location at 0.5. In one of the JSON files, the starting location is at 2. Considering the formula in Figure 12, changing the meter's starting location will influence the degree of the incomplete circle and consequently change the predicted value.

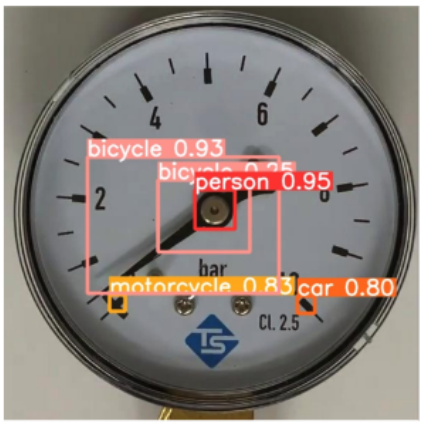
Another finding is that when looking at Meter A, Howells et al. [3] have a low Average Absolute Deviation Value of 0.162 bar. However, its Max Absolute Deviation Value is very high at 0.750 bar. Intuitively, the max value should also be lower when the average value is lower. However, since there are two test images, the predicted values are 0.75, but the true values are 0, which makes the Max Absolute Deviation Value 0.75. This issue may derive from its object detection process.

5.2 The issue of excessive bounding boxes

In the ideal situation, when the model runs inference, there should be only four bounding boxes on the image. But, often, due to light and reflection, the model will provide more bounding boxes than it is supposed to have, as seen in the Figure 20 meter images.

This problem can be solved by selecting the coordinate labels that have the highest confidence level. When running the inference on the test set, 12 out of 90 images have this problem, but by using the labels with the highest confidence level, the system can still calculate the angle and the value. By looking into the label file, the labels are sorted ascendingly by using confidence level [tables in Figure 20], which indicates that the last four rows of labels are the ones to be used for locating the objects.

There are two things that need further explanation when looking at Figure 20. First, the coordinate information of each label is presented as normalized values; in order to obtain the exact pixels' coordinates, "x_center" and "width" need to be multiplied by image width, and "y_center" and "height" need to be multiplied by image height. Second, by looking at the inference images, the texts on the top of the bounding boxes are not correct. This is due to the TensorFlow lite model using a default label mapping [7], which will map class 0 to person, class 1 to bicycle, class 2 to car, and class 3 to motorcycle; when exporting the model to tflite file, it does not come with any file that allows adjusting the labels' text. This problem will be further investigated in a future project. Nevertheless, by using the class number, it is still possible to distinguish them.



class	x_center	y_center	width	height	confidence
1	0.481	0.513	0.221	0.171	0.252
2	0.728	0.724	0.041	0.040	0.796
3	0.273	0.720	0.037	0.039	0.826
1	0.432	0.535	0.462	0.324	0.929
0	0.508	0.498	0.096	0.093	0.952

class	x_center	y_center	width	height	confidence
1	0.490	0.480	0.170	0.261	0.336
0	0.495	0.483	0.152	0.195	0.459
2	0.713	0.715	0.038	0.037	0.779
3	0.254	0.691	0.038	0.037	0.794
0	0.498	0.507	0.102	0.105	0.923
1	0.484	0.439	0.319	0.499	0.953

Figure 20. This visualization demonstrates that in certain instances the model will detect more than four bounding boxes, and by selecting the labels with the highest confidence level, it is still able to locate the objects.

5.3 The mismatching expectation during the optimization process

There is a discrepancy between what TensorFlow claims quantization can achieve and the actual size of the model after quantization. TensorFlow claims that using Dynamic range quantization can get a model that is four times smaller and two to three times faster [5]. However, after the quantization process, the model is only two times smaller, and the speed does not significantly improve. This topic has been discussed by the author of YOLOv5 in several Github posts. The author wrote that when quantizing the model to 16 bits, the size will remain the same, and when quantizing to 8 bits the model will be half-size. He also mentioned that during the training process, there is a function called strip optimized, that already made the size of YOLOv5n decrease from 12 MB to 4 MB [8]. In terms of inference speed, the author also claimed that it would not be improved after the quantization process; however, this statement does not come along with a concrete explanation and therefore is still open for discussion [10].

Additionally, this project has tried to use Pruning Technique to remove less important parameters from the model to further downsize it, but this attempt does not make the model size smaller. This is because the structure does not change, and the parameter count is the same. The weights are set to zeros, but they are not removed after pruning, according to the author [9]. Although this project could not make the model four times smaller and faster,

though the model size is 2 MB, it is still very memory efficient. In future research, there can be a comparison between different devices' inference speeds by using the same model.

5.4 Combinations of computing methods and model weights

As mentioned in the related work section, previous research has tried to automate reading values from analog gauges. However, some of the solutions that these researchers provided include heavy-weighted models that can mainly run on powerful devices such as servers or local computers. Alternatively, using edge computing combined with light-weighted models can also perform similar tasks. The section below will discuss in depth the pros and cons of using different kinds of models and computing methods, and which use case suits which solution.

There are pros and cons between the heavy-weighted and light-weighted models [Table 3]. The heavy-weighted model has more neurons and hence can execute more complicated tasks, e.g., Tip-Detection, but it requires more memory when it is running. Since its weight size is larger, it can mainly run on servers instead of edge devices. In contrast, the light-weighted models do not use as much memory while running and can be executed on both servers and edge devices, but since the model is simpler, it is more suitable to carry out simpler tasks, e.g., Pointer-Detection.

	Pros	Cons
Heavy-weighted model	● Can run more complicated prediction tasks	● High memory usage ● Can be challenging to run on edge devices
Light-weighted model	● Low memory usage ● Can run on both edge devices and remote servers	● Can not run complicated prediction tasks

Table 3, The pros and cons of Heavy-weighted and Light-weighted models.

There are also pros and cons to where to run the model [Table 4]. On the one hand, when using cloud computing, it can run both heavy-weighted and light-weighted models. It also has enough memory space to save the images as a backup plan in case the users want to verify the prediction. But, when the internet connection fails, then there will be no incoming data and thus no prediction. And, since the cameras need to send images back to servers, which usually have larger file sizes, the transaction requires a consistent internet connection. This means some areas with poor internet connection perhaps need to install new routers to serve the cameras, which can be an extra expense. On the other hand, when using edge computing, it can still run prediction when there is no connection, and it will not cost too much memory to save the predicted value on the device. Since the file size is very small, it could still be sent with a poor internet connection. However, edge devices usually can only run light-weighted models and it can be challenging to save images due to limited memory on the device.

	Pros	Cons
Cloud Computing	● Can store larger models ● Can save images and do verification	● Cannot run prediction when no wifi connection ● Need strong internet connection to send images
Edge Computing	● Can still run prediction when no internet connection ● Can still send data when internet connection is poor	● Can only store smaller models ● Cannot save images and do verification

Table 4. The pros and cons of Cloud computing and Edge computing.

When combining heavy-weighted models with cloud computing, it can perform complicated tasks and provide accurate results. This is suitable for the use case in which capturing accurate data from the meters is crucial. In contrast, the solution that combines light-weighted models with edge computing can perform simpler tasks and provide approximate results, which is suitable for the use case that does not require accurate values but wants to detect deviations when they happen. In a future project, it can develop an alarm system that informs users that the value on the meter has deviated from its designated range.

In Howells et al.'s research [3], their model requires 300 MB of RAM while running, which is perceived as a heavy-weighted model and can be challenging for edge devices like Arduino Portenta H7 to run. Therefore, this project mainly focuses on training a light-weighted model that can be run by most edge devices while still providing a decent predicted result. Note that what model to use, where to run the model, and which device to use are still dependent on the use case of the end user.

5.5 Comparison between different devices

This project has looked into several edge devices that theoretically can run the system. Each device has its own pros and cons due to its technical specifications. The choice of device will highly depend on the specific use case. For instance, both Raspberry Pi Zero HW and Raspberry Pi 4 have sufficient RAM and Flash which can run larger models like YOLOv5s, YOLOv5m, or the model that Howells et al. [3] created. However, they are usually power-hungry, meaning there should be wall sockets or extension cords next to the meters. In terms of price, the setup of Raspberry Pi 4 is slightly more expensive than Raspberry Pi Zero HW, because Raspberry Pi 4 has more RAM. If some use cases require running other programs at the same time, then Raspberry Pi 4 will be more suitable, otherwise Raspberry Pi Zero HW should be sufficient to carry out the analog meter reading.

On the other hand, Arduino Protenta H7 is power efficient and can be operated by batteries. It is suitable for the use case where the meters are placed at a remote location with no

power supply nearby. Due to its small memory capacity, it can only run smaller models like YOLOv5n. Hence, the use case should accept that there will be a certain error range of the prediction. Figure 21 includes the details about the iPhone 7 and iPhone 11 since Howells et al. [3] use them to test out their system. It is obvious that they both have sufficient RAM and Flash to run most of the programs, but they are in a higher price range compared to Raspberry Pi and Arduino. In future research, there can be more focus on implementing the system from this project to the edge devices. It is foreseeable that different devices need different system requirements, which may lead to adjusting the code.

	Raspberry Pi Zero HW	Raspberry Pi 4	Arduino Portenta H7	iPhone 7	iPhone 11
					
RAM	512 MB	1 GB	8 MB	2 GB	4 GB
Flash	16 GB	16 GB	16 MB	32 GB	64 GB
Total price	*75 €	*95 €	*144 €	583 €	628 €
Pros	Sufficient Memory	Sufficient Memory	Limited Memory	Sufficient Memory	Sufficient Memory
Cons	Power Hungry	Power Hungry	Power Efficient	High Price Range	High Price Range

*Figure 21. Comparison between the devices on RAM, Flash, Total price, Pros and Cons. *Components are already included in the prices.*

6. Conclusion

This project has developed an analog meter reading system that can run on common edge devices. The system starts with setting up a camera that takes images from meters regularly. Then, the object detection model, YOLOv5n, will capture the crucial objects inside the images. Finally, the system will launch the meter reading process by using the coordinates of the objects to calculate the pointed values.

The evaluation of the object detection model shows that it performs well in locating objects, detecting their presence, and identifying their correct classes according to the various types of low losses. The model also shows high precision and recall rates and no overfitting issues. It can generalize well to unseen images from a controlled environment. However, future research is needed to improve the model's robustness to the images with harsh conditions. Overall, YOLOv5n is a promising object detection model with solid performance metrics.

Regarding meter reading, the system developed from this project has an average ±0.261 bar deviation and a maximum error range of ±0.584 bar on Meter A. In the case of Meter B, it has

an average ±1.438 psi deviation and a maximum error range of ±3.091 psi. Compared to Howells et al. [3], this project outperforms their results on Meter A, and Howells et al. perform better than this project on Meter B. Considering the system size is approximately 2 MB, it can be perceived as a success in achieving this performance. Meanwhile, the system size is small enough to fit into many edge devices, which also achieves the goal of being edge device friendly.

The system created from this project can liberate personnel from reading analog dial meters manually, as this tedious and labor-intensive task can be automated. Moreover, it also provides the possibility to sense value deviation immediately, which can provide alarms to the responsible person and minimize the risk of production loss.

References

[1] J. S. Lauridsen, J. A. G. Grassmé, M. Pedersen, D. G. Jensen, S. H. Andersen, and T. B. Moeslund, “Reading circular analogue gauges using digital image processing,” Proceedings of the 14th International Conference on Computer Vision Theory and Applications, vol. 4, pp. 373-382), Jan. 2019, doi: 10.5220/0007386003730382.

[2] J. Peixoto et al., “Development of an analog gauge reading solution based on computer vision and deep learning for an IoT application,” Telecom, vol. 3, no. 1, pp. 564-580, Oct. 2022, doi: 10.3390/telecom3040032.

[3] B. Howells, J. Charles, and R. Cipolla, “Real-time analogue gauge transcription on mobile phone,” presented at 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, June 2021, doi: 10.1109/CVPRW53098.2021.00269.

[4] G. Salomon, R. Laroca, and D. Menotti, “Deep learning for image-based automatic dial meter reading: Dataset and baselines,” presented at 2020 International Joint Conference on Neural Networks, May 2020, doi: 10.1109/IJCNN48605.2020.9207318.

[5] “Post-training quantization.” TensorFlow. https://www.tensorflow.org/lite/performance/post_training_quantization (accessed March 25, 2023).

[6] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” ArXiv, Apr. 2018. [Online]. Available: https://arxiv.org/pdf/1804.02767.pdf (accessed March 10, 2023).

[7] EdjeElectronics. “TensorFlow-lite-object-detection-on-Android-and-Raspberry-Pi.” GitHub. Sep. 06, 2022. https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/doc/local_training_guide.md (accessed April 2, 2023).

[8] G. Jocher. “The size of the trained model is inconsistent with the official model.” GitHub. https://github.com/ultralytics/ultralytics/issues/1374 (accessed May 3, 2023).

[9] G. Jocher. “Pruning/sparsity tutorial.” GitHub. https://github.com/ultralytics/yolov5/issues/304 (accessed May 6, 2023).

[10] G. Jocher. “8bit quantisation.” GitHub. https://github.com/ultralytics/yolov5/issues/211(accessed May 7, 2023).

Appendix A: Details of devices’ prices

- **Raspberry Pi Zero WH (with pre-soldered header)**
  - Board (512 MB RAM): 21,75 €
  - Flash (16GB Micro SD card): 13,54 €
  - Camera: 34,07 €
  - Camera Cable: 5,28 €
  - Total Price: 74,64 €
- **Raspberry 4**
  - Board (1 GB RAM): 47,80 €
  - Flash (16GB Micro SD card): 13,54 €
  - Camera: 34,07 €
  - Total Price: 95,41 €
- **Arduino Protenta H7**
  - Board with (8MB RAM and 16 MB Flash): 99 €
  - Camera Shield: 45 €
  - Total Price: 144 €
- **iPhone 7**
  - RAM: 2GB
  - FLASH: 32GB
  - Total Price: 583 € (649 USD)
- **iPhone 11**
  - RAM: 4GB
  - Flash: 64GB
  - Total Price: 628€ (699 USD)

Appendix B: License of Howells et el.'s dataset

Appendix C: The data and code from this project

The code and the test data from this project can be both found on this GitHub page: https://github.com/weijiewang87/wei_jie_wang_bachelor_project.

Since training and validation data have larger file sizes which could not be stored in GitHub, they are separately stored in this Google Drive: https://drive.google.com/drive/folders/1ONyU1p6EDwMY8ug632hYzcuPlLoNHgDo?usp=sharing