T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

PART DETECTION AND FAULT ANALYSIS FOR APPLIANCE MANUFACTURING

BERRU LAFCI

SUPERVISOR
DR. YAKUP GENÇ

GEBZE
2024

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

PART DETECTION AND FAULT ANALYSIS FOR APPLIANCE MANUFACTURING

BERRU LAFCI

SUPERVISOR
DR. YAKUP GENÇ

2024
GEBZE

	GRADUATION PROJECT JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on .../.../2024 by the following jury.

JURY

Member
(Supervisor) : Dr. Yakup Genç

Member : Prof. Yusuf Sinan Akgül

ABSTRACT

This project develops an AI-based object detection system specifically for the buttons on ovens. The primary goal is to create an intelligent tool that can identify the presence and orientation of buttons on ovens during production, thereby reducing the need for human intervention.

Objectives

- **Detection of Buttons:** Develop an AI tool to determine whether ovens in production have buttons.
- **Orientation Analysis:** Analyze the direction of the buttons once detected.

Solution Approach

- **Dataset Annotation:** Label the buttons in the dataset containing images of ovens.
- **Model Training:** Use YOLOv8 for training and testing the model on the annotated dataset.
- **Direction Analysis:** Label the notches. Calculate the orientation of the buttons detected by the model.

Expected Outcomes

- An AI system capable of detecting buttons and notched on ovens.
- A tool to analyze the direction of buttons, ensuring correct assembly and functionality.
- Reduced human labor in the production process, enhancing efficiency and consistency.

Keywords: YOLOV8, Object Detection.

ÖZET

Bu projenin, fırın düğmelerine özel yapay zeka tabanlı nesne algılama sistemini geliştirmeyi hedefler. Temel amaç, üretim sırasında fırınlardaki düğmelerin varlığını ve yönünü tespit edebilen ve böylece insan müdahalesine olan ihtiyacı azaltan yapay zeka aracı yaratmaktır.

Hedefler

- **Düğmelerin Algılanması:** Üretimdeki fırınların düğmelerinin olup olmadığını belirlemek için bir yapay zeka aracı geliştirmek.
- **Yön Analizi:** Düğmeler algılandıktan sonra düğmelerin yönünü analiz etmek.

Çözüm Yöntemi

- **Data Etiketleme:** Fırın görsellerini içeren veri kümesindeki düğmeleri etiketlemek.
- **Model Eğitimi:** Etiketlenmiş veri kümesinde modeli eğitmek ve test etmek için YOLOv8'i kullanmak.
- **Yön Analizi:** Çentikleri etiketlemek. Model tarafından bulunan düğmelerin yönünü hesaplamak.

Beklenen Sonuçlar

- Fırınların üzerindeki düğmeleri ve çentikleri tespit edebilen bir yapay zeka sistemi oluşturmak.
- Düğmelerin yönünü analiz ederek doğru montaj ve işlevselliği sağlayan bir araç tasarlamak.
- Üretim sürecinde insan emeğinin azaltılıp, verimliliğin ve tutarlılığın arttırılmasını sağlamak.

Anahtar Kelimeler: YOLOV8, Nesne Algılama.

ACKNOWLEDGEMENT

I would like to acknowledge the support and resources provided by the open-source community, particularly the developers of the YOLO model and the Roboflow API, which greatly facilitated the creation of these scripts. Their contributions have been invaluable in advancing my work in object detection and image analysis.

Additionally, I extend my heartfelt gratitude to my supervisor, Yakup Genç, for their invaluable guidance, encouragement, and insights throughout this project. Their expertise and support have been instrumental in the successful completion of this work.

Berru Lafcı

CONTENTS

1. DATASET

I spent a considerable amount of time on finding and annotating the dataset, which is one of the most important and time-consuming parts of the project. The dataset is crucial because the quality and accuracy of the data directly impact the performance of the model. To annotate the dataset, I used Roboflow, a tool that significantly streamlined the annotation process and improved efficiency. Roboflow provided a user-friendly interface and a variety of features that made the task more manageable. Despite the tool's capabilities, careful and precise manual annotation was still required to ensure the highest quality data for training the model. This meticulous process was essential to achieve reliable and accurate results in the final project. [1]

1.1. Finding Dataset

Unfortunately, the dataset with the oven pictures was not provided by Vestel. So I found a dataset with oven images myself. Normally, in the dataset coming from Vestel, the oven images would be plain and clear. The dataset I found contained many blurry images from different angles. That's why some parts of the project were challenging for me. [2]

Figure 1.1: An image from my dataset.

Figure 1.2: How the image should be.

1.2. Annotating Dataset

1.2.1. Button Annotating

As the first phase of the project, I annotated the buttons so that I could perform button object detection. As a result, I was able to annotate 202 oven images from the dataset I found, which had a button and were not excessively blurry.

Figure 1.3: Annotated buttons from Roboflow.

1.2.2. Getting Cropped Button Images

I obtained the button images by using the ”images” and ”labels” files that Roboflow created as a dataset, which contains the annotated images and their locations. First, I define the paths to the image and label directories and collect the list of image files and their corresponding label files. I implement a function, `yolo_to_pil`, which converts YOLO coordinates to PIL coordinates for cropping. For each image, I read the corresponding label file containing the bounding box coordinates for the buttons, convert these coordinates using the `yolo_to_pil` function, and crop the regions from the image. Each cropped button image is then saved to the specified output directory, and I store details about each cropped image in a list. Finally, I downloaded the cropped button images to use in notch detection.

Algorithm 1 Convert YOLO coordinates to PIL crop box

Require: x_center, y_center, $width$, $height$, $image\_width$, $image\_height$
Ensure: $(left, top, right, bottom)$
$x\_center \leftarrow x\_center \times image\_width$
$y\_center \leftarrow y\_center \times image\_height$
$width \leftarrow width \times image\_width$
$height \leftarrow height \times image\_height$
$left \leftarrow x\_center - (width/2)$
$top \leftarrow y\_center - (height/2)$
$right \leftarrow x\_center + (width/2)$
$bottom \leftarrow y\_center + (height/2)$
return $(left, top, right, bottom)$

1.2.3. Notch Annotating

As the second part of the project, I annotated the notches using the cropped button images.

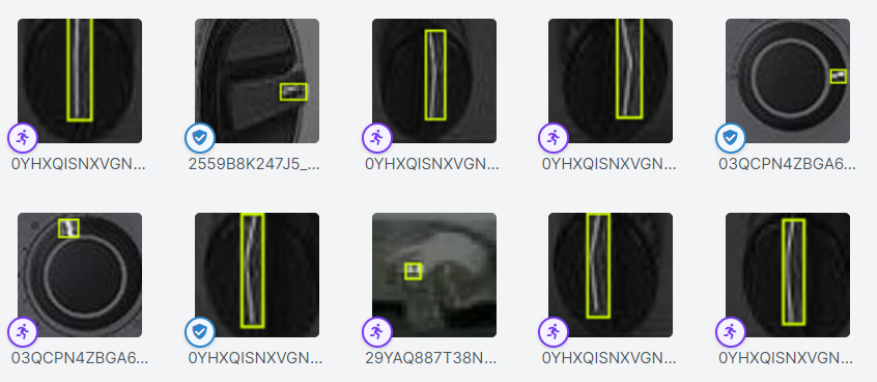


Figure 1.4: Annotated notches from Roboflow.

Since my actual dataset did not arrive, most of the button photos were either too blurry or had no notch. So, I was able to annotate about 16 percent of the cropped button images in the middle of the period. But after I annotated more data at the end of the period, I was able to annotate 276 of 1107 buttons. So, the ratio became 24 percent as a result.

Figure 1.5: A blurry button.

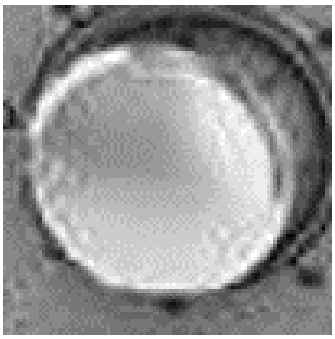


Figure 1.6: A button without notch.

2. OBJECT DETECTION

I did object detection with YOLOv8 via Google Colab. The reason I used YOLOv8 is its advanced accuracy and real-time detection capabilities, which are essential for my project. Google Colab provided a convenient and powerful platform for running these models, leveraging its cloud-based GPU resources to accelerate the training and inference processes. This combination of tools enabled me to achieve high-performance object detection without the need for extensive local hardware. [3]

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

The general flow of the project while performing object detection is in the form of the diagram given.

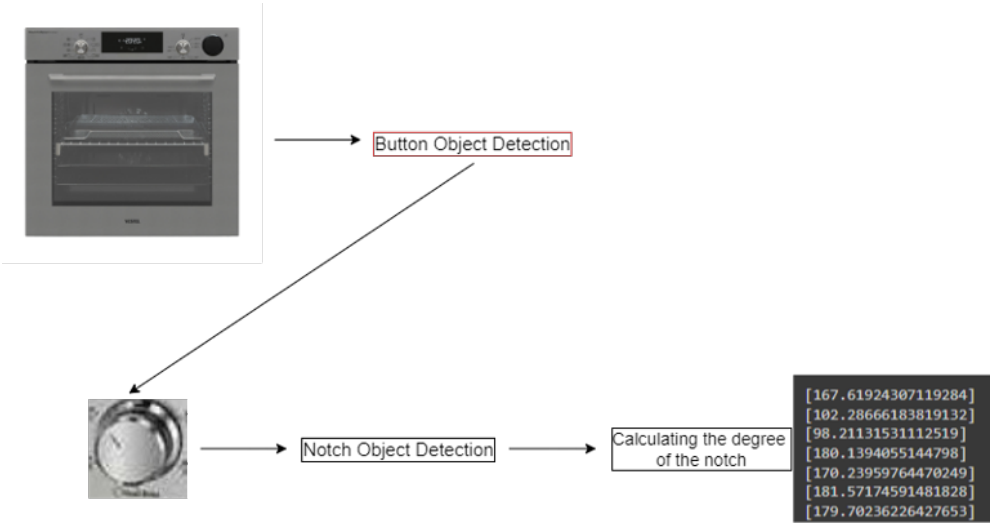


Figure 2.1: Diagram of the project.

2.1. Button Detection

First, I loaded the image data from Roboflow, which involved reading in a dataset of images containing various buttons. Then, I trained the model with 40 epochs with images resized to 800 pixels. I used yolov8m model for this detection. The medium-sized YOLOv8 model (yolov8m) offers a good balance between accuracy and computational efficiency. It provides better performance than the smaller variants (yolov8s or yolov8n) while being less computationally demanding than the larger ones (yolov8l or yolov8x).

After training, I evaluated the model on the validation set to check its performance. I also plotted the training and validation loss/accuracy curves to visualize the model's learning process and ensure there was no overfitting.

Then finally, I tested the images with confusion threshold as 0.5.

2.1.1. Evaluation Results

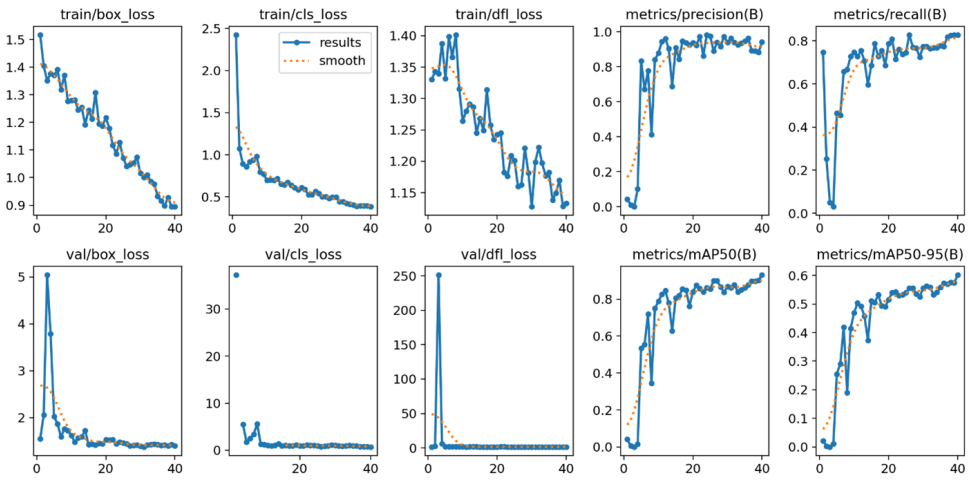


Figure 2.2: Loss graphs.

Train/Box Loss:

This graph shows the decrease in box loss during training, indicating the model is improving its predictions for bounding box coordinates. The steady decline suggests effective learning.

Val/Box Loss:

The validation box loss decreases, although there is more fluctuation compared to training loss. This suggests that while the model is improving, it may be encountering more varied or challenging data during validation.

The overall trend in both training and validation metrics indicates that the model is stable and not overfitting significantly, as both training and validation losses decrease and performance metrics improve.

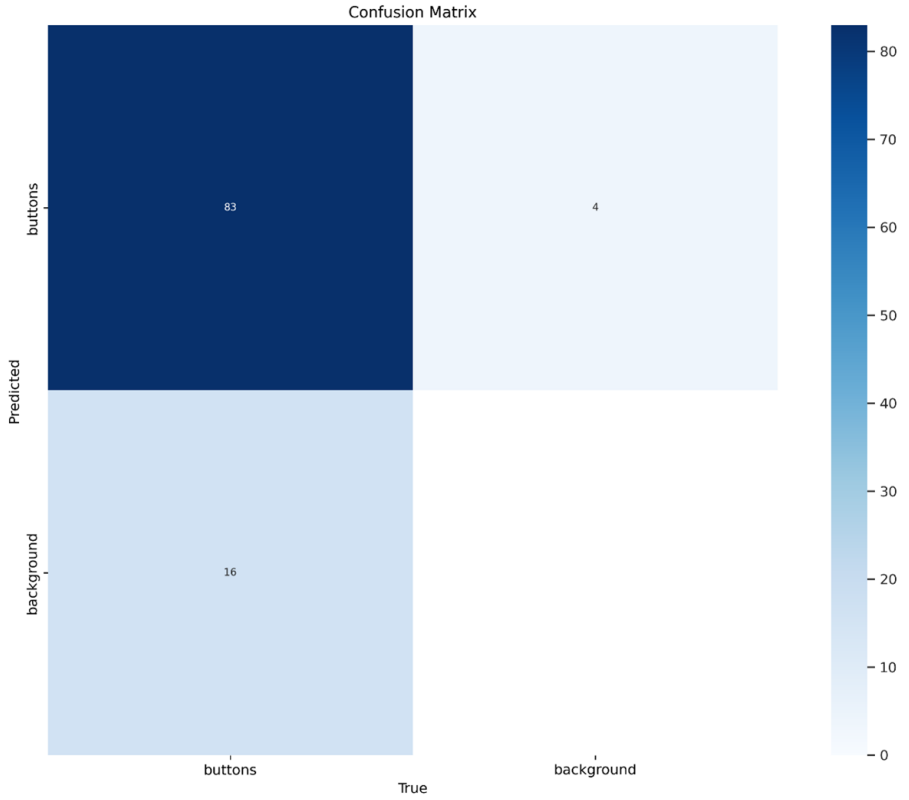


Figure 2.3: Confusion matrix.

True Positives (Top-Left: 83): The model correctly predicted 83 instances as ”buttons” that are actually ”buttons”. This shows a strong performance in correctly identifying buttons.

False Positives (Top-Right: 4): There are 4 instances where the model incorrectly predicted the absence of buttons (background) as ”buttons”. These are false positives, indicating areas where the model mistakenly identifies background regions as buttons.

False Negatives (Bottom-Left: 16): The model predicted 16 instances as background (no buttons) that actually contain ”buttons”. These are false negatives,

indicating that the model missed these buttons.

True Negatives (Bottom-Right): This cell does not contain any value because the absence of buttons is not labeled, so true negatives are not tracked or relevant in this context.

Finally here is a couple of test results with their accuracies and bounding boxes. As you can see, the model did a great job while detecting the buttons.

2.1.2. Getting Cropped Button Images

As I explained above in Datasets chapter, I wrote a function to get cropped button images from already annotated button images to upload them on Roboflow to start a new model. 1

2.2. Notch Detection

In this model, I transferred the cropped button images to Roboflow and used the annotated notches as a dataset. Then, I trained the model with 40 epochs with images resized to 800 pixels. I used yolov8m model for this detection again same with the previous model. After training, I evaluated the model on the validation set to check its performance. I also plotted the training and validation loss/accuracy curves to visualize the model's learning process and ensure there was no overfitting. Then finally, I tested the images with confusion threshold as 0.5.

2.2.1. Evaluation Results

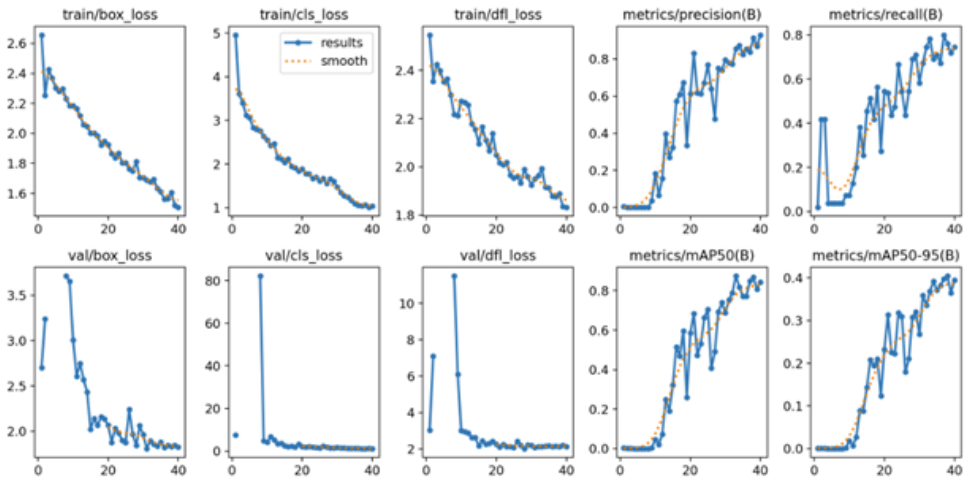


Figure 2.4: Loss graphs.

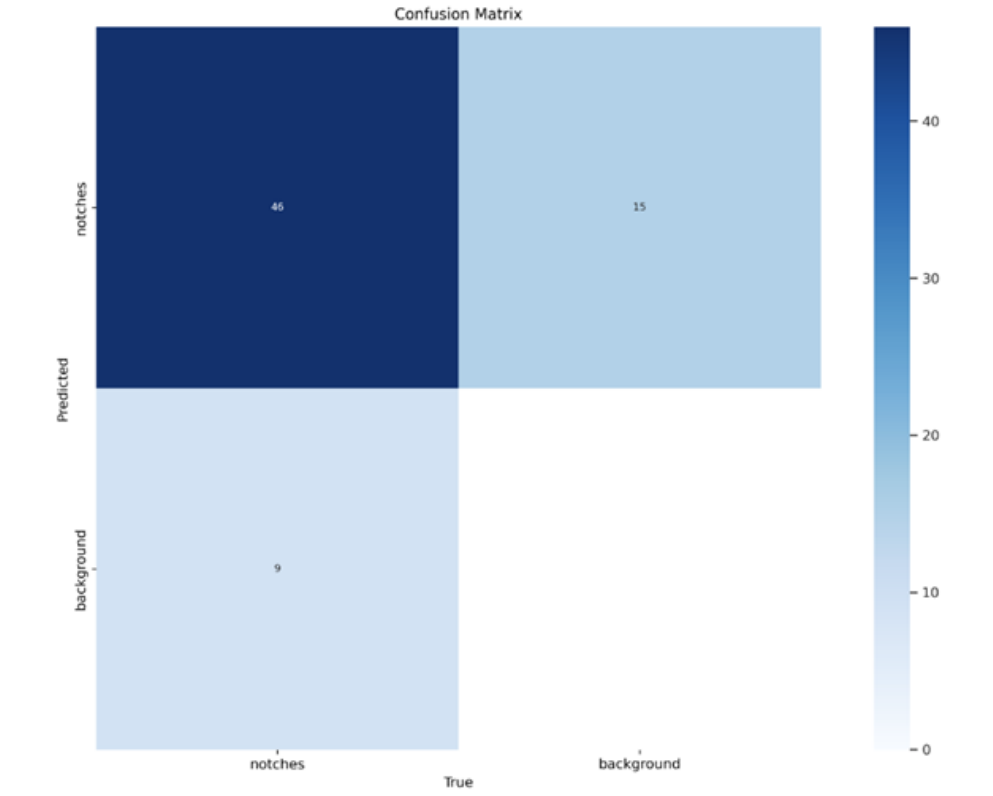


Figure 2.5: Confusion matrix.

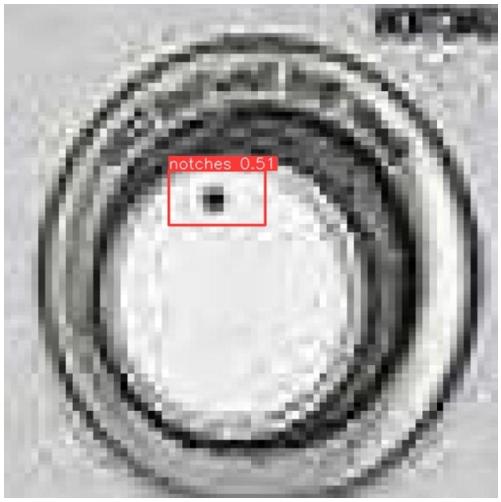
True Positives (Top-Left: 46): The model correctly predicted 46 instances as ”notches” that are actually ”notches”.

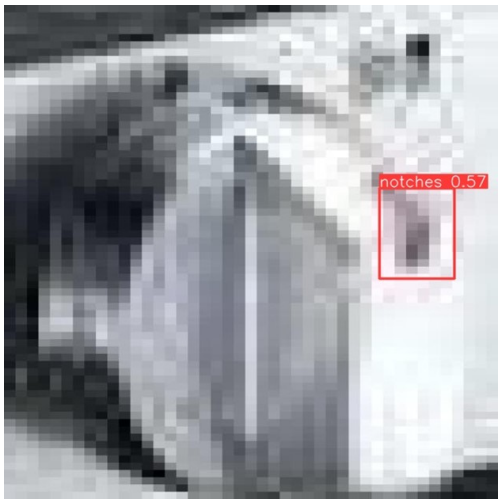
False Positives (Top-Right: 15): There are 15 instances where the model incorrectly predicted the absence of notches (background) as ”notches”.

False Negatives (Bottom-Left: 9): The model predicted 9 instances as background (no notches) that actually contain ”notches”. These are false negatives, indicating that the model missed these notches.

True Negatives (Bottom-Right): This cell does not contain any value because the absence of notches is not labeled, so true negatives are not tracked or relevant in this context.

Finally here is a couple of test results with their accuracies and bounding boxes. As you can see, the model did an average job while detecting the buttons. This may be because of the blurry and different structures of the buttons and the notches.





3. DEGREE CALCULATING

In this part, I developed a comprehensive script that integrates image preprocessing, object detection using the YOLO (You Only Look Once) model, and a graphical user interface (GUI) for visualizing and interacting with the results. I utilized various Python libraries such as ultralytics, cv2, PyQt5, and standard libraries to accomplish these tasks.

1. I prepared the images to ensure compatibility with the YOLO model and to see the buttons bigger by resizing them to a standard size (640x640).

2. After preprocessing the images, I exported the YOLO model **(best.pt)** and making predictions on the resized images. [4]

3.1. Getting Bounding Box Coordinates

For each prediction made by the exported YOLO model, I retrieved the image path and constructed the corresponding label file path by appending a .txt extension to the image file name. I then checked if there were any bounding boxes detected in the prediction. If bounding boxes were present, I extracted the first bounding box's coordinates in the format $[x\_center, y\_center, width, height]$ and the class of the detected object. These values were converted from PyTorch tensors to a list and then written to the label file in the YOLO format: $class, x\_center, y\_center, width, height$.

Algorithm 2 Write Bounding Box to Label File

```
try
if number of bounding boxes > 0 then
    extract the first bounding box coordinates as (x, y, w, h)
    extract the class index as cls
    convert coordinates to list format
    open label file in write mode
    write class and coordinates to file in format: cls x y w h
end if
except Exception as e
print error message with image path
```

I then created a function to read the annotation files and extract the bounding box coordinates.This function opens a label file and reads its contents. It extracts the center coordinates of the bounding boxes and returns them as a list of tuples.

3.2. Degree Calculation

I developed a function to calculate the angle of the bounding box center (notches) relative to the image center.

- The function first calculates the center coordinates of the image.
- For each annotation containing normalized center coordinates (x, y) of a bounding box, the function converts these normalized coordinates to absolute pixel coordinates.
- The function calculates the angle between the image center and the notch center using the $math.atan2$ function, which returns the angle in radians. This angle is then converted from radians to degrees using the $math.degrees$ function.
- Since angles can be negative, the function ensures all angles are positive by adding 360 degrees if necessary and adjusts the image orientation with take modulo 360 to ensure it falls within the 0-360 degree range.
- Finally, the function returns the list of formatted angles, providing the angular positions of notches in the image.

Algorithm 3 Calculate Degree

```
function CALCULATE_DEGREE(image, annotations)
    height, width, _ ← image.shape
    center_x, center_y ← width // 2, height // 2
    degrees ← [ ]
    for (x, y) in annotations do
        notch_x, notch_y ← x * width, y * height
        angle ← MATH.ATAN2(notch_y - center_y, notch_x - center_x)
        degree ← MATH.DEGREES(angle)
        if degree < 0 then
            degree ← degree + 360
        end if
        degree ← (degree + 90) % 360
        formatted_value ← FORMAT(degree, ".3f")
        degrees.append(formatted_value)
    end for
    return degrees
end function
```

3.3. PyQt5 GUI Application

Finally, I implemented a PyQt5 application to provide a GUI for loading images and labels, displaying images, and showing the calculated degrees. I processed each image and its corresponding label 2, calculated degrees 3, and populated the list widget with the results. Then, I displayed the selected image and its calculated degrees in the GUI.

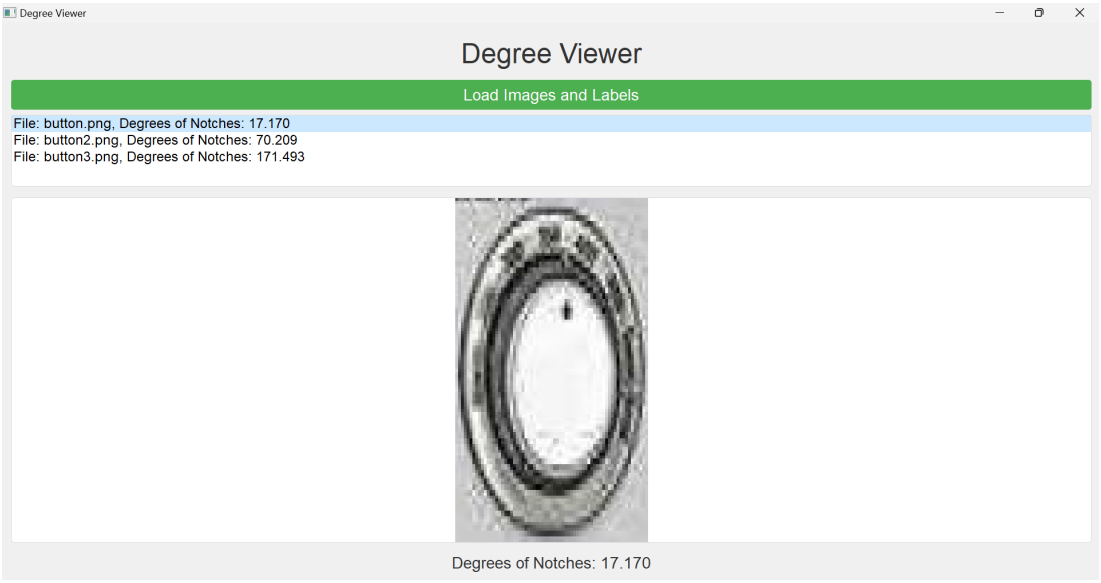


Figure 3.1: Screenshot from GUI.

4. CONCLUSIONS

In conclusion, I have developed two models that effectively detect and analyze buttons and notches on images using YOLO models. For button detection, my model focuses on training a YOLO model and providing detailed visualizations of the training results. Additionally, I included code to crop detected button images from the original images based on YOLO annotations.

For notch detection, my model extends the functionality to detect notches on oven buttons, showcasing the versatility of YOLO models in different object detection tasks.

Furthermore, in degree calculating code in VSCode, I integrated PyQt5 to develop a user-friendly GUI for visualizing detection results and calculating notch degrees. These codes collectively form robust tools for practical applications in image analysis and object detection, demonstrating my capability to handle comprehensive detection tasks from start to finish.

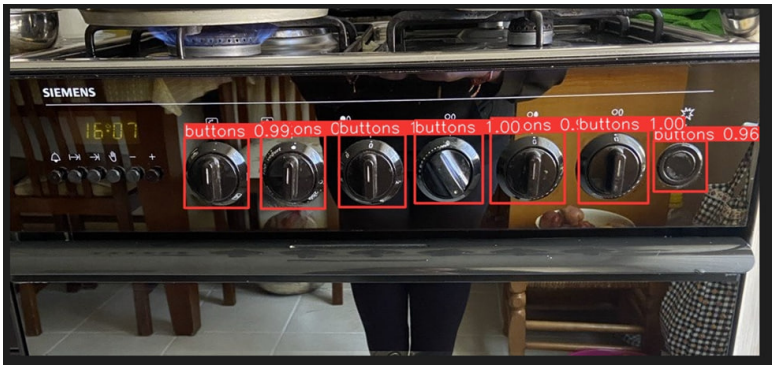


Figure 4.1: Predicted buttons in my house.

BIBLIOGRAPHY

[1] J. Nelson, *Train test split guide and overview*, https://blog.roboflow.com/train-test-split-with-roboflow/, 2020.

[2] images.cv, *Oven labeled image datasets*, https://docs.ultralytics.com/modes/export/.

[3] Y. Yoon, *Fine-tuning yolov8 with custom dataset generated by open-world object detector*, https://medium.com/@yongsun.yoon/fine-tuning-yolov8-using-custom-dataset-generated-by-open-world-object-detector-5724e267645d, 2023.

[4] Burhan-Q and Kayzwer, *Model export with ultralytics yolo*, https://docs.ultralytics.com/modes/export/, 2023.

CV

BERRU LAFCI

lafciberru@gmail.com
+905543834398
linkedin.com/in/berru-lafcı/
github.com/xemeriusss

Education

Gebze Technical University
Computer Science and Engineering Student
Istanbul, Türkiye

Experience

ARDICTECH
Mobile Application Developer Intern (*July 2022 - September 2022*)
- Developed a front-end application for an IoT project using the Flutter framework to monitor and display real-time room temperature and humidity data.

University of Amsterdam
Machine Learning Intern (*July 2023 - September 2023*)
- Created a user-friendly front-end using Streamlit to help explain pre-trained ML models or build self-explanatory ML models under the topic of XAI.

- Evaluated necessary ML models, rule-based classifiers, and datasets for this interface.

Skills

- Programming Languages: C/C++, Python, Java, Dart, Verilog, Git, SQLite.

- Tools and Frameworks: Streamlit, Flutter, Dash, Procreate, Quartus, Mars MIPS Simulator, Postman, Pandas, Sklearn, Linux.

- Languages: Turkish (Native), English (Proficient), Japanese (Beginner).

Projects

Online Library System (Flutter)

Software Engineering Course Project *(2022)*

- Developed a mobile and web application for borrowing and returning books in libraries in an online system.

Telco Customer Churn Analysis (Python)

Data Mining Course Project *(2023)*

- Analyzed Telco Customer Churn dataset, preprocessed the data, built several classifier models, and evaluated their performances.

Part Detection and Fault Analysis for Appliance Manufacturing (Python, YOLOv8)

Graduation Course Project *(2024 - Present)*

- Developing an AI tool to detect whether ovens under production have buttons and to find the directions of these buttons using YOLOv8.

Hobbies

- Digital drawing, Swimming, Tennis, Japanese culture, Reading
- My drawings: instagram.com/xemeriusss

APPENDICES

Appendix 1: Youtube demo

https://youtu.be/GekUPFl4aNU?si=aS3xsSbpjS2OAIp0

Appendix 2: Github Repository

My source code: https://github.com/xemeriusss/YOLO-Button-Notch-Detection-Degree-Calculation