COE379L Project 4: Object in Image Detection
By Samarth Kabbur and Mukund Raman

This project explores the application of machine learning techniques to the field of robotics by developing a system that enables a robotic arm to detect and track objects in its environment. The objective is to use the robotic arm to identify a target object, such as a candy, coin, or water bottle, and move in such a way that the object remains centered in the arm's field of view.

We, Samarth and Mukund, share a common interest in using artificial intelligence and machine learning to solve real-world robotics challenges. Samarth previously built a robotic arm powered by an Arduino UNO microcontroller. This microcontroller is capable of accepting input parameters such as coordinates and orientation, which dictate how the robotic arm moves. In future work, we want to mount a camera directly onto the robotic arm to capture real-time images of its surroundings. These images would then be processed using a machine learning model trained to detect specific types of objects. Once an object is identified in the camera's field of view, the model would calculate its position and orientation and use that information to send movement commands to the robotic arm, enabling it to align itself with the detected object.

Three different image-based datasets were used to train the machine learning models. The first dataset consisted of approximately 162 labeled images of assorted candy items, which were obtained from EJ Technology Consultants. The second dataset contained around 750 labeled images of various coins, also sourced from EJ Technology Consultants. The third dataset comprised roughly 400 labeled images of water bottles, which were extracted and filtered from the Microsoft COCO dataset to include only instances of water bottles.

The machine learning model used in this project was part of the Ultralytics YOLOv11 family of object detection architectures. Among the available models, the nano variant was selected due to its small size and high speed, making it suitable for real-time inference tasks on limited computational resources. The implementation was carried out using Python, including the following modules such as `os`, `sys`, and `subprocess` for interacting with the file system and executing shell commands; `cv2`, `time`, and `numpy` for image processing, video frame capture, and inference loop execution; and `glob` and `IPython.display` for loading and visualizing image data.

Additionally, FiftyOne was used as a dataset and visualization tool. It was especially useful for filtering the COCO dataset to isolate only water bottle images and for generating and organizing corresponding label files. Docker was used to containerize the inference system, allowing it to run as a web-based service accessible via HTTP requests.

Our project followed several steps for dataset preparation, model training, and real-time inference. The first step involved preparing the datasets for training. Using FiftyOne, a script named `load_coco_dataset.py` was developed to download and filter the COCO dataset, ensuring that only images containing the "bottle" class were included. For the candy and coin datasets, the labeled images were manually downloaded and placed into appropriate directories.

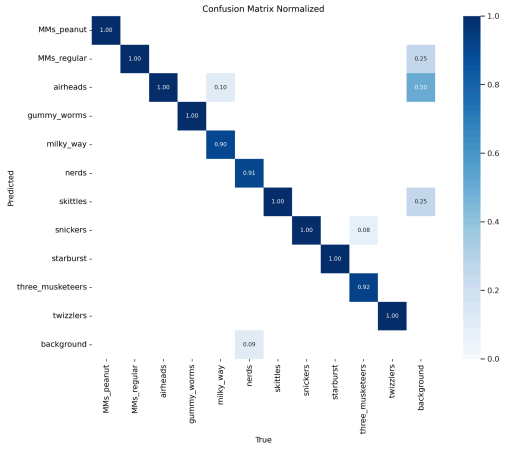
A second script, `data_splitter.py`, was written to automate the process of splitting each dataset into an 80/20 train-test split and organizing the images into a format compatible with the YOLO training procedure.

To facilitate model training, a third script called `data_yaml_generator.py` was used to create configuration files in YAML format. These configuration files contained essential information about dataset paths and class labels and were passed as input to the YOLO training command. The training process was executed within a Jupyter notebook using the `subprocess` module, which allowed system-level commands to be issued directly from within notebook cells. Mukund utilized an Nvidia RTX 4090 GPU, which significantly accelerated the training times for the models.

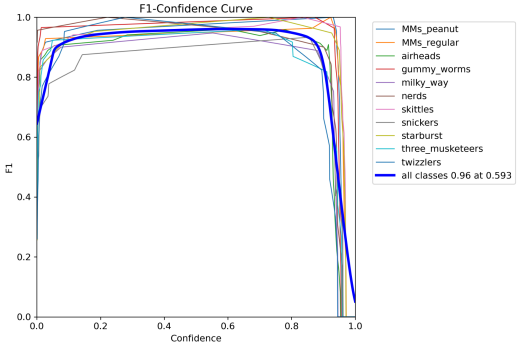
Once training was complete, the models were evaluated using the validation portion of each dataset. Metrics such as the confusion matrix, F1 curve, and precision-recall statistics were analyzed to assess performance. The YOLO framework automatically saved the best-performing model weights during the training process as a `.pt` file, which would later be used for inference.

To perform inference on new data, we developed a Docker container that acted as a web server. This server was capable of handling HTTP GET and POST requests, receiving an image, resizing and preprocessing it, running the trained YOLO model, and returning an annotated image with bounding boxes drawn around detected objects. This container was uploaded to Docker Hub to enable easy reuse and deployment.

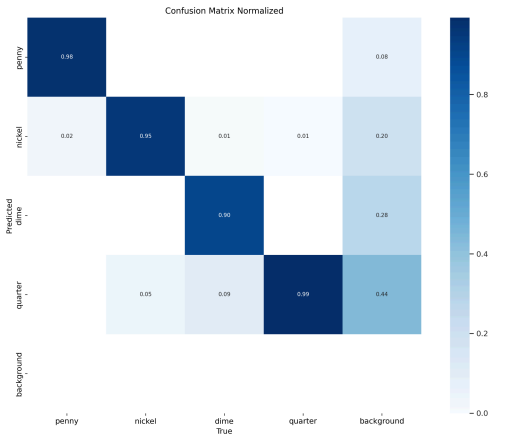
In addition to image inference, we also implemented functionality to process entire videos. A loop was created that read an input video file frame by frame. Each frame was passed through the model to detect objects, and the results were used to draw bounding boxes in real time. The annotated frames were stored in a memory buffer and then written to a new MP4 file using OpenCV's `VideoWriter` functionality. In this implementation, the bounding box will only be written if the detection confidence is above 50%.

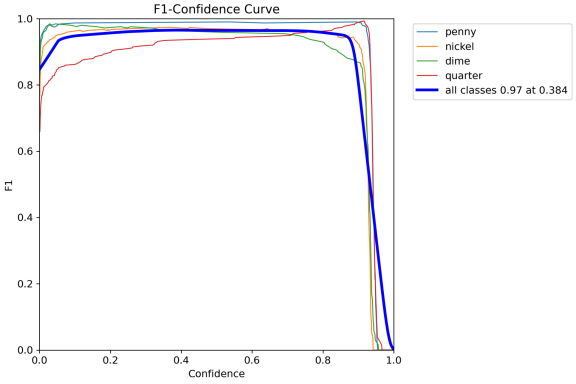


Confusion Matrix of the Candy Dataset



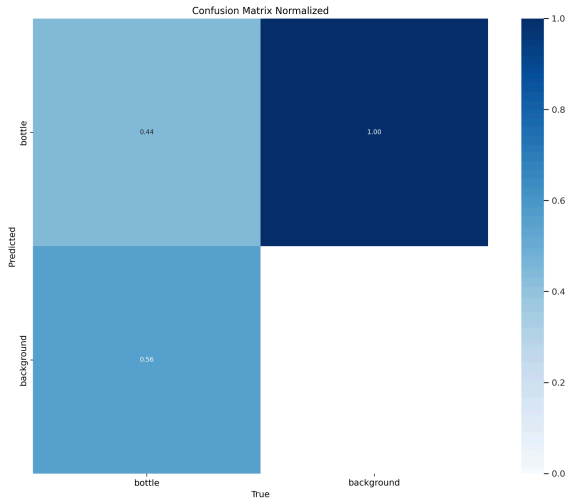
F1 Confidence Curve of the Candy Dataset

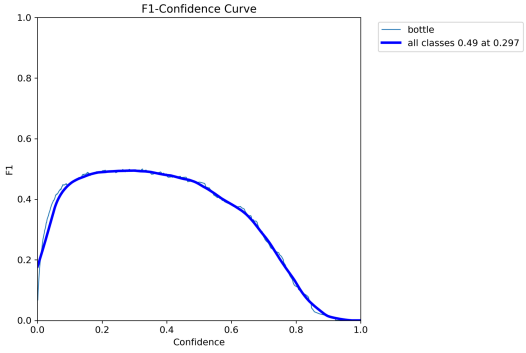




Confusion Matrix of the Coin Dataset F1 Confidence Curve of the Coin Dataset

The results for the candy and coin datasets were really good. When evaluated using confusion matrices and precision-recall curves, the models trained on these datasets performed with high accuracy. The candy model confidently identified all target objects with only minimal confusion, such as occasional misclassification between the candy labeled as "Airheads" and the image background. The simplicity and consistency of the candy and coin images played a significant role in this strong performance. Most of these images featured clear subjects placed against uniform or non-distracting backgrounds, often with a human hand holding the object in view.



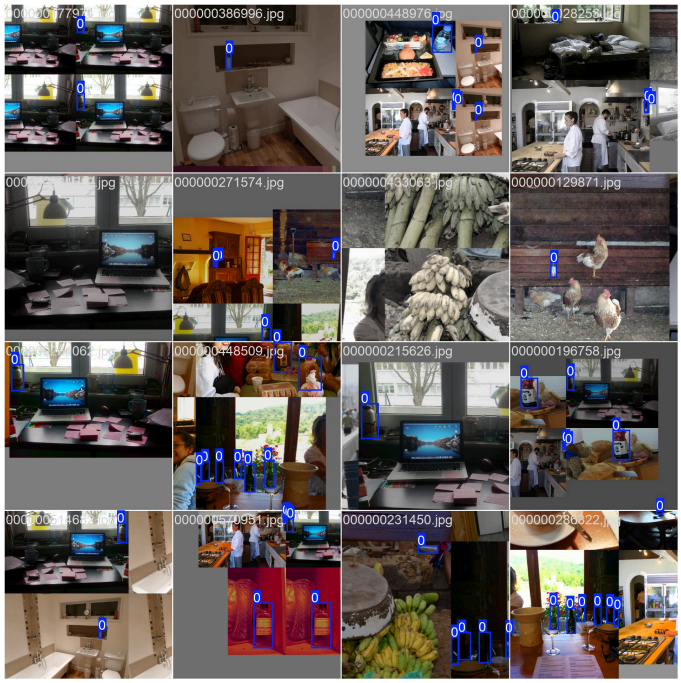


Confusion Matrix of the Bottle Dataset F1 Confidence Curve of the Bottle Dataset

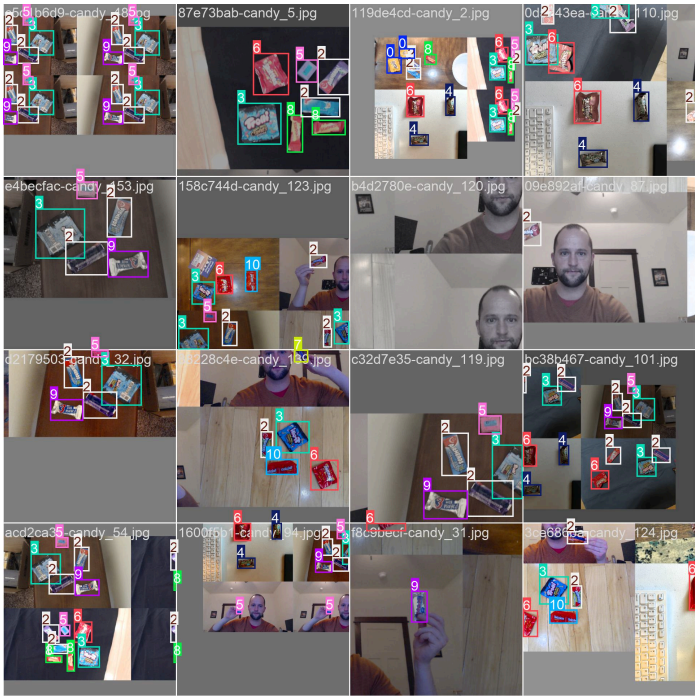
On the other hand, the performance of the water bottle model was significantly weaker. The model successfully detected water bottles in only 44 percent of cases. Looking at the confusion matrix revealed that the model often failed to distinguish between water bottles and background elements. The F1 confidence climbed to barely over 50%. This issue was probably due to the high complexity of the COCO dataset, which is designed to feature objects in a wide

variety of real-world contexts. The water bottles in these images appeared in scenes ranging from sports fields to cluttered indoor environments, often with significant variation in lighting and scenery.

The underlying cause of this poor performance appears to be the complexity and inconsistency of the image backgrounds in the COCO dataset. Unlike the candy and coin datasets, which had clean and focused backgrounds and objects, the COCO water bottle images were highly diverse. As a result, the model may have struggled to learn a consistent representation of what constitutes a water bottle versus the background. This suggests that for tasks involving object detection in real-world environments, a more curated dataset with clearer distinctions between object and background may be necessary. It is likely that if a new dataset were constructed with consistent backgrounds and focused water bottle imagery, the model would achieve significantly higher accuracy.



A sample train batch of the COCO bottle data subset, with annotated labels. You can see a strong variety of backgrounds and contexts.



A sample train batch of the candy dataset, with annotated labels. You can see a clear distinction between background and the subject.

A sample train batch of the coin dataset, with annotated labels. You can see a clear distinction between background and the subject.

As a final step to encapsulate our model, we created an inference.ipynb file that could run inference in a modular fashion on a variety of input images depending on the input, by

interfacing with the inference server. Simply update the data directory and the script will do the following. It will run the docker container, send a GET request to the server,


```
    "yaml_file": "yolo11s.yaml"
  },
  "description": "Detects locations of water bottles in images.",
  "name": "water-bottle-detector",
  "number_of_parameters": 9428179,
  "version": "v1"
}
```


Sample JSON Response from the Inference Server

It'll then load up the input image, and send a POST request to the server with the image. The server will respond with the image with predictions drawn onto the image.

References

The candy and coin datasets used for this project were published by Evan Juras of EJ Technology Consultants. The candy dataset can be found at https://s3.us-west-1.amazonaws.com/evanjuras.com/resources/candy_data_06JAN25.zip, and the coin dataset is available at https://s3.us-west-1.amazonaws.com/evanjuras.com/resources/YOLO_coin_data_12DEC30.zip.

The water bottle dataset was extracted from the Microsoft COCO dataset (Common Objects in Context), which is described in more detail in the following paper: https://doi.org/10.48550/arXiv.1405.0312. We created a custom script to isolate only the images and annotations containing instances of water bottles.

The YOLOv11n model was sourced from Ultralytics: Jocher, G., Qiu, J., & Chaurasia, A. (2023). Ultralytics YOLO (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

This project successfully demonstrated a complete pipeline for integrating machine learning with robotic systems. However, the challenges faced with the COCO water bottle dataset also showed the importance of dataset quality and context when training deep learning models. With a more targeted and consistent dataset, we hope that future iterations of the system could yield even better results and serve as a foundation for more advanced robotic vision systems.