**1. Introduction**

This project aims to detect the quality of welding in real-time using object detection techniques. The system classifies welding as either good, bad, or defective. By leveraging deep learning models in a mobile application, we can automate the quality control process, reducing manual inspection errors and increasing efficiency in industries where welding is critical.

**Why Welding Quality Detection is Important**

Welding is a vital process in various industries, including automotive, construction, shipbuilding, and aerospace. Poor welding quality can lead to structural failures, costly repairs, and even safety hazards. Traditionally, inspecting weld quality relies on human experts, which can be time-consuming and error prone. Automating this process with object detection not only speeds up inspections but also increases accuracy, ensuring that only high-quality welds pass through.

This book targets beginners who want to understand how welding quality detection works, as well as engineers and students interested in object detection applications. The goal is to provide a comprehensive, step-by-step guide that even someone with minimal experience can follow to implement this system.

**2. Background on Object Detection**

**Overview of Object Detection**

Object detection is a computer vision technique that identifies and classifies objects within an image or video stream. In this project, we use object detection to automatically assess welding quality by detecting different types of defects or features in welding lines.

**Why YOLOv5?**

For this project, we chose YOLOv5 (You Only Look Once) because it offers a good balance between accuracy and speed, which is crucial for real-time detection in mobile applications. YOLOv5 is widely adopted by developers due to its ease of use and efficient architecture, which allows for faster processing without sacrificing too much precision.

**How YOLOv5 Works**

YOLOv5 operates by dividing an image into a grid and predicting bounding boxes for objects within each grid cell. It outputs a list of objects, each with its probability score, bounding box coordinates, and class label. The small weights version of YOLOv5 was chosen for this project to reduce the model size, making it suitable for deployment on mobile devices. This helps ensure the app can run efficiently while maintaining accuracy.

**Challenges**

During training, we faced some challenges in ensuring that the model correctly identified "good" welding versus defects. Often, the model tended to over-focus on defects, which made it difficult to get accurate classifications for good welds.

**3. Dataset and Model Preparation**

For this project, I used six datasets—five from Roboflow and one from Kaggle—each ranging in size from 200MB to 400MB. The images are all 640x640 in resolution and stored in .jpg format. The datasets originally contained varying label sets, so one of the main challenges was merging these labels into a single, cohesive dataset for training.

**Preprocessing Steps**

1. **Dataset Handling**: First, I processed the Roboflow datasets to make them compatible with YOLOv5. This involved performing augmentations like horizontal and vertical flips, as well as rotating the images between -15° and +15° to increase the dataset’s diversity.
2. **Label Merging**: To standardize the labels across the different datasets, I created a custom Python function (def) to rename and merge the labels. Some labels were deleted if they didn’t align with the main goal of the project.
3. **Empty Labels and Duplicates**: I implemented additional Python functions to delete any images with empty labels and remove duplicate images. After this, I split the data into training (90%), validation (8%), and test (2%) sets.

**Challenges in Dataset Preparation**

One significant challenge was the inconsistent labels across the datasets. Each dataset had its own label structure, which had to be merged. The final, merged dataset contained the following labels: ['Defect', 'Good Welding', 'Porosity', 'Bad Welding'].

**4. Model Implementation in Flutter**

Convert the model to TFlite to put it in Flutter app

**5. Evaluation and Testing**

The model was evaluated on 1268 images, resulting in the following performance metrics:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Class** | **Images** | **Instances** | **Precision** | **Recall** | **mAP50** | **mAP50-90** |
| All | 1268 | 10039 | 0.847 | 0.808 | 0.876 | 0.561 |
| Defect | 1268 | 5995 | 0.889 | 0.863 | 0.913 | 0.55 |
| Good Welding | 1268 | 428 | 0.733 | 0.694 | 0.78 | 0.496 |
| Porosity | 1268 | 2617 | 0.856 | 0.83 | 0.881 | 0.519 |
| Bad Welding | 1268 | 999 | 0.909 | 0.845 | 0.931 | 0.678 |

**Challenges in Model Training**

I trained the model seven times before arriving at a version that balanced performance. In some iterations, the model detected all welds as bad, while in others, it overly focused on defects. Tweaking the dataset and model parameters helped fine-tune the results.

**Some test images:**

A close-up of a metal

Description automatically generated A comparison of a weld

Description automatically generated

**6. Use Case: Welding Quality Detection**

*(Placeholder for real-world use cases and screenshots showcasing the model’s effectiveness in detecting welding defects)*

The model can detect the following welding defects:

* Burn-through
* Crack
* Crater
* Incomplete penetration
* Overflow
* Porosity
* Spatter / Spatters
* Undercut
* Irregular weld line

**7. Optimization and Improvements**

To optimize the model for mobile deployment, I converted the YOLOv5 weights to ONNX format, then to TensorFlow model, and finally to TensorFlow Lite (TFLite). This process reduced the model size, allowing it to run smoothly on mobile devices. Additional optimizations, like quantization, were applied to minimize the performance impact.

**8. Conclusion**

This project demonstrated the power of deep learning in real-time welding quality detection. By using YOLOv5, we achieved a robust system capable of identifying various welding defects with high accuracy. The project's success highlights how AI can improve industrial processes, offering faster, more reliable quality control. Future enhancements may include expanding the model’s capabilities to detect more defect types and further optimizing its performance on mobile platforms.