BrightSpot Automation

Final Report

Author:
Lin Zhang, Timothy Churchill, Logan Caffey, Lance Tobey

Colorado School of Mines - Advanced Software Engineering (CSCI 370

December 7, 2024

Table of Contents

1 Introduction

The installation of photovoltaic (PV) systems has experienced rapid growth due to the increasing demand for alternative, renewable, and clean energy sources. However, solar modules in the field are subjected to significant environmental stress, which can lead to module degradation. Specifically, environmental factors such as snow, hail, and rain can cause tensile stress that fractures the brittle silicon cells in solar modules. Additionally, oscillatory wind sources exacerbate these issues in already cracked cells. Among the various modes of degradation, cell fractures and glass cracks are particularly prominent. Notably, these defects can significantly reduce the photoelectric conversion efficiency of solar modules, thereby lowering their overall energy yield.

Currently, defects in solar modules are detected through routine inspections. However, module inspection remains a challenging and complex task. Defects in solar modules are difficult to identify with the naked eye, necessitating the use of non-disruptive imaging techniques such as Electroluminescence (EL), photoluminescence, and Ultraviolet Fluorescence (UVF). Still, the process is time-consuming and expensive, requiring trained experts to manually identify various cell defects. Since degradation rates and lifespan play a critical role in determining the levelized cost of electricity for PV systems, advanced techniques are needed to address these issues.

Our client, BrightSpot Automation, has partnered with our team to address the challenges of defect detection in solar modules. To streamline the inspection workflow, automation is a critical step. A key aspect of this automation involves the segmentation of solar modules and cells. Segmentation is essential because it enables object detection models to analyze individual cells, determining whether they contain defects. To this end, we are developing a machine learning model capable of accurately segmenting modules and cells in solar panels. This model will serve as a foundation for further advancements in defect detection using object detection models, a rapidly evolving area in the PV industry.

1.1 Client

Our client, BrightSpot Automation, is a global provider of specialized manufacturing and measurement tools for the PV industry. Their clients include R&D labs, cell and panel production facilities, certification testing labs, etc. BrightSpot Automation has a software platform, IMPEL, which interfaces with cameras to process and analyze images and store the resulting data.

For this project, BrightSpot Automation has provided us with approximately 200 EL images of solar modules and access to an Azure virtual machine(VM) with GPU for model training. The images serve as a sample dataset for model development, and the VM acts as the dedicated environment for experimentation. The segmentation models we developed may eventually be incorporated into their existing workflow or software such as IMPEL, but this decision will be at the client's discretion. Our primary focus is on developing and evaluating various computer vision models on the provided VM, rather than directly integrating them into their software.

1.2 Definition of Done

This project is research-focused and exploratory, with no requirement for integration into existing systems, model deployment, or the development of front-end or back-end infrastructure. All models are checkpointed on the provided Azure VM.

We define the Definition of Done as successfully developed:

1. "good" models for module segmentation (segmentation of modules from the background)
2. "good" model for cell segmentation (segmentation of individual cells within the segmented modules).

The definition of "good" in machine learning models is context-dependent. For BrightSpot Automation, this typically includes real-client feedback loops. However, since our models will not be tested by real users, we will rely on the sample dataset and use computer vision metrics and inference time as the basis to evaluate and highlight the models' strengths and limitations.

2 Requirements

2.1 Functional Requirements

At a minimum, this project requires the development of functional models for both module segmentation and cell segmentation. While traditional analytical methods and computer vision techniques exist for image segmentation, this project explicitly focuses on leveraging deep learning neural networks as the ultimate model choice. The functional requirements are outlined below:

1. **Leverage Neural Network Architectures:** Utilize established neural network architectures, such as those available through the Hugging Face's Transformers library or other relevant libraries, to perform segmentation tasks.
2. **Module Segmentation:** Develop a model to segment solar modules from the background in EL images. This includes separating and distinguishing modules even in challenging conditions, such as dark or noisy training samples
   - **Image Correction:** After module segmentation, apply perspective correction. Segmented modules should be cropped, masked, and transformed into a standardized rectangular format to prepare for cell segmentation.
   - This model could be instance, semantic, or panoptic.
3. **Solar Cell Segmentation:** Develop an instance segmentation model to distinguish individual solar cells within a segmented module.
   - Each cell should be uniquely identified and masked, with minimal false positives. For example, if a module contains 72 cells, the instance segmentation model should output approximately 72 instance outputs.
   - The model should include a mechanism (e.g., confidence scores or thresholds) to control and validate the predicted output layers.
4. **Training, Testing, and Logging:**
   - Include a clear train-test split to validate the model's performance.
   - Ensure proper logging of metrics during training and evaluation, including metrics such as Intersection over Union (IoU), loss, precision, and recall.

2.2 Non-Functional Requirements

The non-functional requirements focus on data quality and timing constraints:

1. **High-Quality Labels:** The performance of machine learning models is highly dependent on the quality of the labeled data provided. To ensure optimal results:
   - Labels for modules and cells should be precise and accurate. Minor discrepancies are acceptable but should be minimized.
   - Beyond visual inspection, which is prone to manual limitations, labels cannot be validated directly. Again, emphasizing the need for detailed and careful inspection during the labeling process.
   - Labels must be converted into COCO-style format, as this is a widely adopted standard for computer vision tasks. We used LabelMe to facilitate the labeling process.
2. **Model Bias:** Ideally, the model should generalize to various solar module images. However, due to the limited dataset provided by BrightSpot Automation, which lacks sufficient diversity, we acknowledge the inherent model bias.
   - Our focus will be on achieving generalization within the given dataset and minimizing overfitting, rather than attempting to generalize to unseen datasets.
3. **Inference Time:** The inference time for both module and cell segmentation must be efficient.
   - Predictions should take less than one second per image on a single NVIDIA GPU to ensure real-time usability.

3 System Architecture

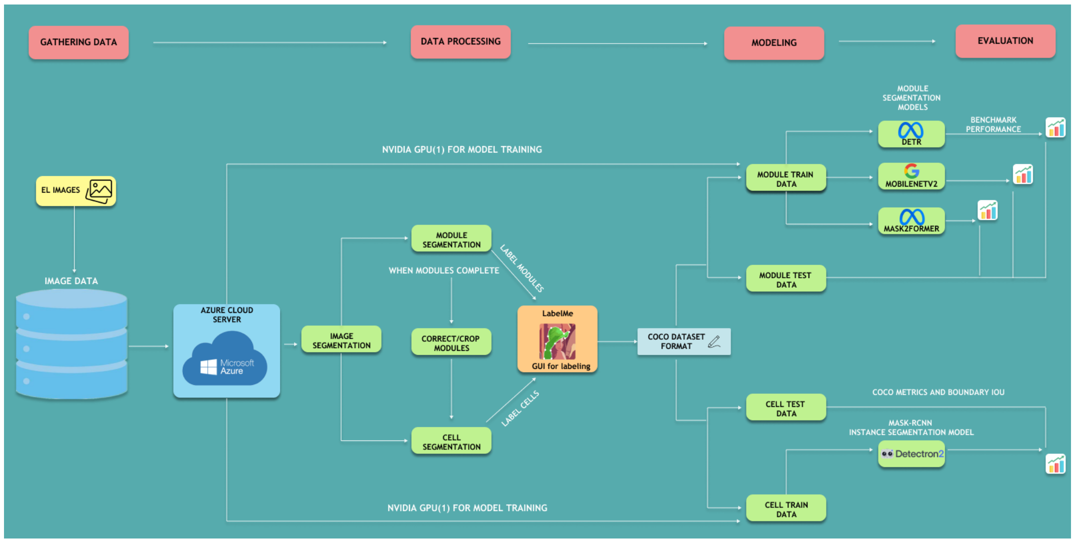

Figure 1: System Architecture Diagram

The architecture of the project is organized into four stages: Data collection, Data Processing, Modeling, and Evaluation. The data collection process is facilitated by BrightSpot Automation. EL images of solar panels were captured, processed and stored on their Azure VM Server.

For this project, a Microsoft Azure VM was set up to train and run neural network models. However, it is important to note that the VM is resource-constrained, with only one GPU available for training at any given time. Once the EL images are transported to the VM, the segmentation process begins. This process is divided into two primary stages, starting with module segmentation. Each EL image contains one solar module and up to 144 individual cells. Since ground truth labels are not available for these images, manual labeling is required as an initial step. While the project's long-term goal is to automate the labeling process, accurate ground truth labels are critical for effective model training and benchmarking.

For manual annotation, we used LabelMe, a GUI tool that allows dynamic annotations for both module and cell images. LabelMe generates annotations in JSON format, which are then aggregated and converted into a COCO-style dataset format. This format includes fields such as `'image_id'`, `'file_name'`, `'width'`, `'height'`, `'segmentation'`, `'bbox'`, `'area'`, and category mappings for the labels. COCO-style annotations are widely supported by image segmentation models, serving as the default format (similar to how pandas loads CSV files effortlessly). During the module segmentation phase, these COCO annotations are further converted into a `.jsonl` format, which is required by Hugging Face's dataloader function. This format ensures the proper setup of the dataset along with its associated metadata for the module segmentation task in Hugging Face.

It is important to highlight that module segmentation should ideally be performed first. Specifically, the segmented modules should 1) be masked or cropped out of the original images, and 2) undergo perspective transformation using OpenCV (cv2) to correct any angular distortions in the EL images. This ensures that modules are transformed into a rectangular format, which simplifies the labeling process for cells. For modeling and evaluation, this project leveraged Transformers from Hugging Face, an open-source Python library that provides a suite of computer vision models. For module segmentation, we employ several models, including MobileNetV2, a lightweight and efficient CNN, as well as DETR and Mask2Former, state-of-the-art transformer-based models developed by Meta. Training and testing for module segmentation are implemented using the Hugging Face Trainer base class. This framework simplifies the training pipeline while allowing for extensive customization and streamlined logging of metrics. In the context of module segmentation, all three segmentation tasks—instance, semantic, and panoptic—are supported. Specifically, we track Intersection over Union (IoU) metrics, which measure the overlap between predicted module masks and their ground truth labels.

For cell segmentation, we utilized Meta's Detectron2 API, a library designed for deep learning and computer vision tasks. Our approach is built around Mask R-CNN, a well-established model for instance segmentation. This model generates variable-sized masks for individual instances in an image, each accompanied by confidence scores. For evaluation, we use the COCO evaluator, which computes average precision (AP) for segmentation masks across various IoU thresholds. Additionally, we implement Boundary IoU, an edge-based evaluation metric that measures the accuracy of predicted boundaries by applying erosion to binary mask labels. This metric provides a finer-grained assessment of edge alignment, which is particularly useful in instance segmentation. These evaluation metrics, including Boundary IoU and COCO-based metrics, are used to benchmark the models' performance, helping to identify issues such as overtraining or undertraining.

Model training results, including optimized weights, are saved directly on the server for reproducibility and further analysis. For this project, it is important to note that model deployment is not a primary focus, as the current scope is centered around model training, evaluation, and performance for research and exploration.

4 Technical Design

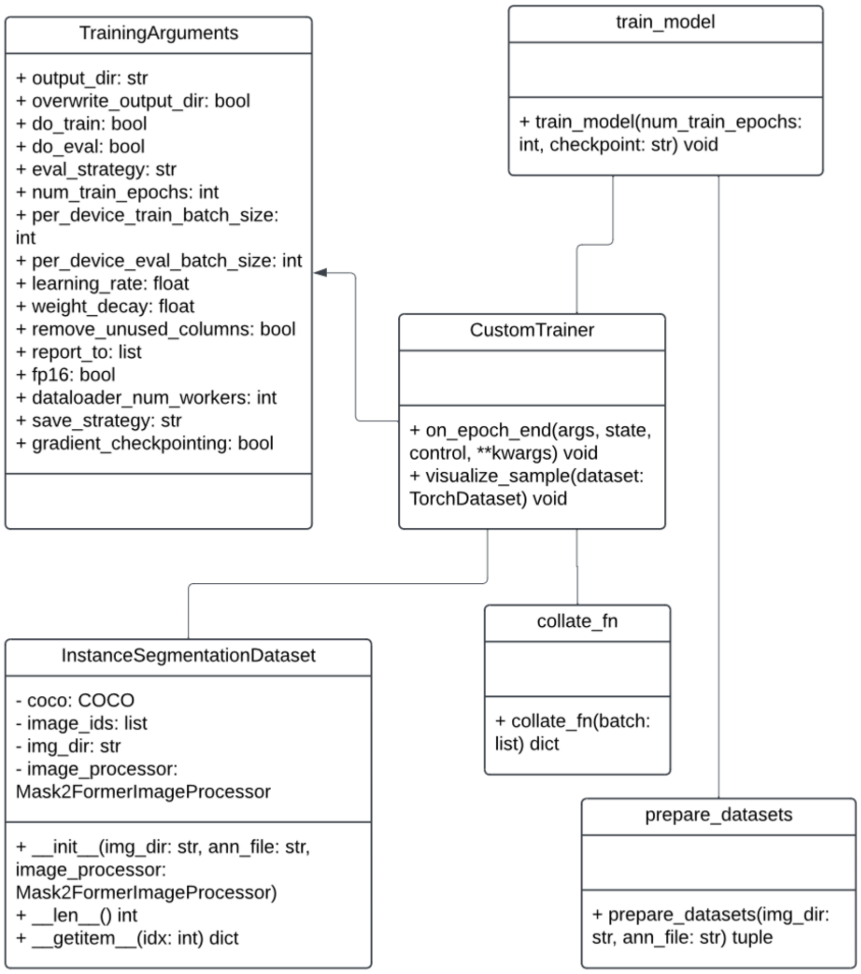
TrainingArguments
+ output_dir: str
+ overwrite_output_dir: bool
+ do_train: bool
+ do_eval: bool
+ eval_strategy: str
+ num_train_epochs: int
+ per_device_train_batch_size: int
+ per_device_eval_batch_size: int
+ learning_rate: float
+ weight_decay: float
+ remove_unused_columns: bool
+ report_to: list
+ fp16: bool
+ dataloader_num_workers: int
+ save_strategy: str
+ gradient_checkpointing: bool
train_model
+ train_model(num_train_epochs: int, checkpoint: str) void
CustomTrainer
+ on_epoch_end(args, state, control, **kwargs) void
+ visualize_sample(dataset: TorchDataset) void
collate_fn
+ collate_fn(batch: list) dict
InstanceSegmentationDataset
- coco: COCO
- image_ids: list
- img_dir: str
- image_processor: Mask2FormerImageProcessor
+ __init__(img_dir: str, ann_file: str, image_processor: Mask2FormerImageProcessor)
+ __len__() int
+ __getitem__(idx: int) dict
prepare_datasets
+ prepare_datasets(img_dir: str, ann_file: str) tuple

Figure 2: Model UML Diagram

The technical design of the solar panel segmentation project focuses on creating a highly efficient and accurate pipeline for processing high-resolution solar panel imagery. The overarching goal is to segment solar modules and their constituent cells, enabling the identification of defects with precision and minimal latency. This requires a thoughtful integration of model selection, data preprocessing, and resource optimization while ensuring scalability and robustness. Key aspects of the project are discussed in detail below, including model selection, optimization, data preprocessing, and refinement. Relevant diagrams are provided to illustrate the architecture and workflows.

A UML diagram, as shown in Figure 2, details the system's architecture, highlighting the relationships between components such as the TrainingArguments class, CustomTrainer, and the InstanceSegmentationDataset. The TrainingArguments class, in particular, is central to configuring hyperparameters like learning rate, batch size, and gradient checkpointing. The custom training loop implemented through CustomTrainer allows for epoch-end evaluations and visualizations of segmentation results, enabling iterative improvements. This modular design ensures that the system remains adaptable and extensible for future enhancements.

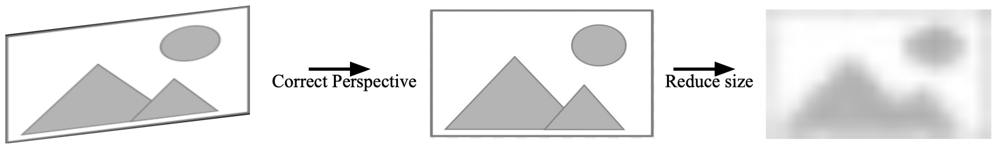

Figure 3: Model UML Diagram

The technical pipeline begins with high-resolution input images that often exceed 6000x4000 pixels. These images require substantial preprocessing to meet memory constraints while retaining critical features necessary for segmentation. The preprocessing pipeline, as illustrated in Figure 3, starts with image resizing, where dimensions are scaled down. This step reduces computational demands while preserving edge details. Data augmentation techniques such as rotations, flips, and brightness adjustments are applied to increase the diversity of the dataset, which is essential for improving the model's ability to generalize to different lighting conditions and camera angles. Annotating the images in COCO format ensures compatibility with the chosen segmentation models. These annotations undergo rigorous validation to maintain alignment with the actual structures of solar modules.

To further enhance the quality of the input data, distortion correction techniques are applied. Solar panels are often photographed at angles that introduce perspective distortions, which can mislead segmentation models. These distortions are corrected using projective transformations, standardizing the images for accurate analysis. The InstanceSegmentationDataset class manages these preprocessing steps dynamically during training, ensuring consistency across all batches. A utility function, collate_fn, structures the data into variable-sized batches compatible with the training pipeline.

Performance metrics play a critical role in evaluating the system's efficacy. IoU scores quantify the overlap between predicted and ground truth segmentation masks, while precision and recall metrics assess the system's ability to detect individual solar cells accurately. Beyond these traditional metrics, defect detection accuracy is introduced as a custom metric to evaluate the system's ability to identify minute imperfections within the segmented cells. These metrics guide the iterative training process, helping the team pinpoint areas for improvement.

The project's design also prioritizes computational efficiency. The training pipeline is optimized for limited GPU resources, with image processing scheduled sequentially to prevent memory overflow. During inference, the system achieves a latency of under one second per image, meeting real-time performance requirements. Efforts to reduce this latency further are ongoing, with a goal of achieving processing times closer to 500 milliseconds per image. This optimization is crucial for deploying the system in operational environments where rapid analysis of solar panels is necessary.

Integration and validation are key to ensuring the success of the system. Each component of the pipeline undergoes rigorous testing, from the manual verification of annotated data to the validation of model outputs against ground truth labels. Logging mechanisms are embedded within the training process to monitor loss functions and evaluation metrics in real time, allowing the team to identify and resolve anomalies promptly. By combining advanced segmentation models, robust preprocessing techniques, and stringent evaluation criteria, the solar panel segmentation project delivers a reliable and efficient solution for analyzing photovoltaic systems. The technical design reflects a systematic approach to addressing the unique challenges of solar panel analysis, ensuring the system's adaptability to diverse scenarios and its scalability for future advancements.

5 QA

5.1 Code Reviews

Our team is focusing on specific module testing, our approach emphasizes the integration of different components within the development process. Code reviews in our team are conducted systematically, with sessions held at the end of every sprint to ensure code quality and alignment with project standards. The review process begins internally, where team members examine each other's code, identifying potential issues, suggesting improvements, and verifying that all changes adhere to agreed-upon guidelines. Following this internal review, we engage with our client, to collect feedback and confirm that changes meet the client's requirements. This ensures that the code not only functions properly but also aligns with the user's expectations. After incorporating feedback from both internal and client reviews, the changes are finalized, and we obtain our client's approval to merge them. This process promotes transparency, accountability, and high-quality integration across the codebase.

5.2 Code Metrics

Our choice of metrics reflects our integration-focused approach, where we focus on model performance. Metrics like inference time, IoU, and loss tracking during training were selected to provide insight into the trade-offs between speed, precision, and learning efficiency. Inference time measures how quickly a model can make predictions, which is vital for real-time applications, while IoU evaluates the accuracy of predictions, particularly relevant for tasks involving object detection or segmentation. Loss tracking, on the other hand, helps monitor the model's learning process over time, revealing patterns of convergence and highlighting potential issues like overfitting or stagnation both of which we dealt with. These metrics allow us to assess how well each model balances the demands of our project objectives, whether prioritizing speed, accuracy, or overall robustness, ensuring that the chosen models align with the intended outcomes and deliver reliable performance in production environments.

5.3 User Acceptance Testing

One major aspect to making sure we cover all of the tests that the end user will want to attempt. These can be categorized into two main sets, the expected cases and the edge cases. The results of these cases will be similar, as the program will execute and go through the machine learning process for any library of images. Expected case results will have an input of a folder of different solar panel files, and through our different machine learning models, we will output the accuracy, loss, and images of the masks overlaying panels that show high accuracy from the model itself. For our edge cases, the machine learning model will still take images as normal, but under the user assumption that these images will not be of solar panels or have the elements that we are looking for. As a result, we would expect non-accurate results due to the images not matching what we expected.

A large part of our product is to find the result our stakeholders are satisfied with. This will be a model that is both accurate and fast. Part of the process of model selection of the multiple models that have been created includes what our stakeholders would want to prioritize. We would then work with them to find the most accurate model of the set that we have and work alongside them to figure out what they want the model's specifications around that accuracy to be, to find the model that fits the specific mold that they want.

5.4 Ethics

The ethical implications of this project are primarily limited to the developmental aspects of the code rather than the broader societal impact of the project itself. As it stands, the design does not pose any inherent risks or potential harm to individuals. Therefore, our ethical responsibility is centered around ensuring that the code we develop meets high standards of quality and integrity. One of our key obligations, as outlined in ACM Code of Ethics section 2.6, is to ensure that we are sufficiently knowledgeable and skilled in the technical domain we are working in, so that we operate within our area of competence. This includes being mindful of our limitations and seeking advice or collaboration where necessary.

Furthermore, it is critical that our testing processes are thorough, valid, and provide an accurate reflection of the model's performance in real-world scenarios. This is essential not only for the functionality of the project but also for maintaining transparency and building trust with our client. If the model is not tested rigorously or the results are misrepresented, it could lead to unintended consequences down the line. Equally important is ensuring that the code we produce is both robust and maintainable, meeting standards of effectiveness and usability to support long-term success and adaptability. By adhering to these ethical guidelines and ensuring competence, accuracy, transparency, and quality in our development process we ensure that our project remains ethically sound and trustworthy.

6 Results

For the results, we present model performance metrics for both module segmentation and cell segmentation. The dataset was split with 80% used for training and 20% for testing, and evaluation metrics were logged throughout the model training process.

6.1 Module Segmentation

Model	IoU	Inference Time
DETR	0.9790	0.135
MobileNetV2	0.9762	0.040
Mask2Former	0.9382	0.208

Table 1: Module Segmentation Model Performance

For evaluation, we tracked IoU, which measures the overlap between predicted and ground truth masks for each of our models in Hugging Face. Inference time refers to the duration required for the model to predict the segmentation of a module within each image.

6.2 Cell Segmentation

6.2.1 Boundary IoU

At the cell level, we place particular emphasis on the precision of the edges in the model's output. Specifically, IoU alone is not sufficient to assess edge-level accuracy. This is because the area measured by IoU increases quadratically over time, while the boundaries of an object grow linearly. As a result, for large objects, achieving a high IoU primarily reflects capturing a large portion of the object, but not necessarily its edges. To address this issue, we introduce Boundary IoU, a metric designed to account for this distinction.

Boundary IoU for cell segmentation is calculated as follows: after the model generates n instance segmentation masks, both the ground truth labels and the predicted masks are eroded inwards. The eroded masks are then subtracted from the original outputs, leaving only the boundaries of the predicted and ground truth masks. The intersection of these boundary-only regions is then computed. Each ground truth label is compared against the n predicted masks, and the best-matching mask is considered the correct prediction.

Below are visualizations that illustrate Boundary IoU. The size of the boundaries can be adjusted using a dilation ratio.

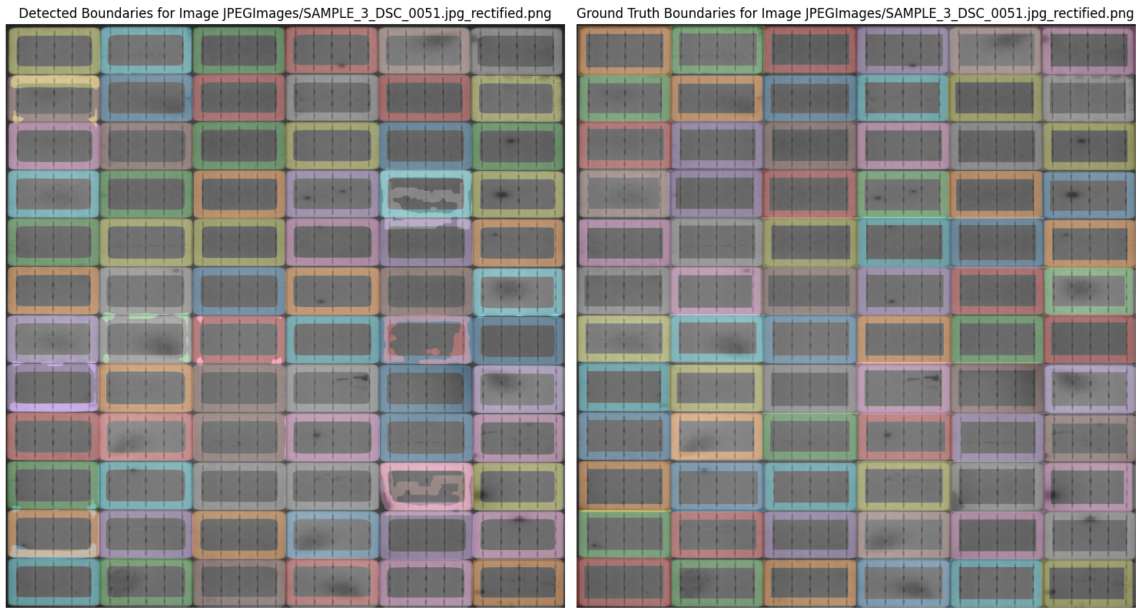
Detected Boundaries for Image JPEGImages/SAMPLE_3_DSC_0051.jpg_rectified.png
Ground Truth Boundaries for Image JPEGImages/SAMPLE_3_DSC_0051.jpg_rectified.png

Figure 4: Boundary IoU Example

Model	lr sched	bbox AP	segm AP	segm Recall	Boundary IoU
R101-C4	3x	67.9	69.1	73.5	66.5
R50-FPN	3x	72.91	72.68	76	71.1
R100-FPN	3x	70.80	72.04	76.3	71.9
R101-D5	3x	73.23	73.46	77.8	72.3
X101-FPN	3x	71.79	72.47	76.7	72.0

Table 2: Detectron2 Model Performance Metrics

Note: We also tracked Boundary IoU on some module segmentation models.

7 Future Work

7.1 Features Implemented

1. **Module Segmentation:** Various models were implemented and evaluated for module segmentation with Hugging Face compatibility, as requested by the client.

- Models such as DETR, Mask2Former, and MobileNetV2 were trained and tested on the sample dataset, achieving high IoU scores.
- Post-segmentation corrections, including perspective transformation of modules into rectangular form, were successfully completed to improve the quality of downstream tasks like cell segmentation.
- Metrics and performance evaluations, including IoU, loss and inference times, were logged consistently for all models during training and testing phases.

2. **Cell Segmentation:** Instance segmentation was implemented using Detectron2's Mask R-CNN models, which were benchmarked and evaluated for performance.
   - Multiple architectural variations within Detectron2 were tested and evaluated, with performance metrics logged using a customized version of the COCO Evaluator. This evaluator tracks traditional computer vision metrics such as segmentation AP and segmentation recall, along with a custom implementation of Boundary IoU

7.2 Features to Complete

1. **Module Segmentation Edge Refinement:** Refining the edges of masked outputs from module segmentation models remains a key challenge
   - MobileNetV2, in particular, struggles with severe resolution compression during image processing. It resizes images to 513x513 dimensions and produces output layers with a resolution as low as 65x65. When working with original images of 6000x4000 pixels, this leads to jagged edges on the masked outputs. Despite its high IoU scores, it failed to capture the edges of modules. While DETR and Mask2Former also face edge detection issues, their resolution compression is less pronounced compared to MobileNetV2. (Image compression is necessary due to memory constraints)
   - To mitigate these issues, post-processing algorithms such as contour refinement or edge-specific adjustments (e.g., finding and refining rectangular module edges) may be required. Additionally, modifying the output layers of models to improve resolution could help enhance edge-level precision.
2. **Model Optimization and Feature Engineering:** Further optimization and feature engineering are necessary to enhance the performance of cell segmentation models.
   - While Detectron2's Mask R-CNN models provided an alternative to Hugging Face models, further improvements are needed for cell segmentation, which remains a challenging task due to the complexity of cell boundaries and dense object layouts.
   - Feature Engineering was proposed during client meetings as a viable solution.
3. **Defect Detection:** Once cell segmentation challenges are addressed, the next step will involve defect detection using object detection models. This is the ultimate goal of this project.
   - The defect detection process will follow a similar workflow, including manual annotation, model training, and metric logging. However, labeling defects requires domain expertise.
   - Andrew Gabor, CTO and President of BrightSpot Automation, is currently responsible for defect labeling at BrightSpot Automation, leveraging his years of expertise in the field.

8 Lessons Learned

- **Drawbacks of High Levels of Abstraction:** Abstraction is considered a key factor in improving software development productivity, allowing developers to use functions as black boxes by simply inputting specific parameters and receiving results. However, in our project, we encountered challenges with using Hugging Face's `Trainer` function. This high-level function, while incredibly versatile, required us to understand its inner workings to apply it effectively. Although we preferred having a comprehensive function rather than writing everything ourselves, it might have been helpful if `Trainer` could be divided into smaller, modular functions. This would have allowed us more control and flexibility while retaining the benefits of abstraction.

- **Importance of Abundant and Quality Data:** During the latter half of our project, when we began labeling individual cells, we discovered that our dataset was heavily biased toward one type of cell. This imbalance affected the model's performance on less common cell types, underscoring the importance of having not only a large quantity of data but also a balanced representation across all categories. Our experience emphasized that diverse data is crucial for model generalization and consistent accuracy.

- **Importance of Computational Resources:** Resource limitations were a recurring challenge throughout our project. We frequently encountered issues such as VM storage shortages, insufficient system RAM to handle data, and the restriction of having only one GPU, which meant only one person could test their model at a time. Each of these limitations slowed down our progress and highlighted how much more efficiently we could work with a larger infrastructure. This experience demonstrated that adequate computational resources, including storage, memory, and GPU availability, are essential to maintaining momentum and making real progress.

- **Data Preprocessing:** Each image in our dataset required extensive preprocessing for optimal model performance. We carefully labeled each solar panel and cell, scaled images down, turned labeled boxes into maps, normalized each image to square dimensions, cropped them, and corrected their perspectives to ensure consistency. This intensive, detailed preprocessing process was necessary for our model to learn effectively and accurately, reinforcing the idea that high-quality, well-prepared data is as important as the model itself.

- **Benefits of Version Control:** Throughout the project, GitHub version control was essential in maintaining an organized workflow. Each team member worked on their branch, and when ready, we merged changes into the main branch after a team code review and a pull request that required client approval. This approach allowed us to track changes systematically, minimize conflicts, and ensure high-quality code before each integration. Version control was a cornerstone of our collaboration and organization, and we used it to the fullest extent.

9 Acknowledgment

We would like to extend our thanks first and foremost to BrightSpot automation for providing us with both the scope and data required for completing this project. In addition, we would like to thank Rich Landy for his guidance and suggestions. Without him providing starting points along the way for us to explore we would not have been able to achieve nearly as much as we did.

Additionally, we express our appreciation towards the open source community on hugging face which provided us with excellent image segmentation models that we worked to improve upon. Specifically, we thank Meta and Google for the production of the DETR, Mask2, and MobileNet V2 models. These models were the basis of everything we did so this project would not have been possible without these initial models to train from.

10 Team Profile

Logan Caffey:

Role: Developer

Bio: Junior, Computer Science Major with a focus on Data Science.

Lin Zhang:

Role: Developer, Advisor Liaison

Bio:Senior, dual major: Statistics + Computer Science(general track)

Timothy Churchill:

Role: Developer, Client Liaison

Bio:Sophomore: Computer Science with a focus on robotics and intelligent systems.

Lance Tobey:

Role: Developer

Bio: Senior, Computer Science Major with a focus on Data Science.

11 Appendix

Our project uses a Python virtual environment that can be created from our requirements.txt in the root directory of our code. It consists of every function that we either need directly or that is needed by something we use indirectly.