Anomaly detection with continuous data collection and modeling

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By
Manish Kovelamudi, Vamshidhar Reddy Parupally, Ravindar Reddy Siddenki,
Vikas Tadepu

December/2023

APPROVED

DocuSigned by:
kaikai liu
8D8C3598F1C8432...

Prof. Kaikai Liu, Project Advisor

ABSTRACT

Anomaly detection with continuous data collection and modeling
By
Manish Kovelamudi, Vamshidhar Reddy Parupally, Ravindar Reddy Siddenki, Vikas Tadepu

In this rapidly evolving world, in order to meet the demand, industries are transitioning to accelerated production processes. In such a fast-paced environment, ensuring that products are defect-free and of high quality is not just an advantage but a necessity. It's crucial to make sure that manufactured products are free from defects, in order to maximize desired output and save maintenance expenses. Although there are production standards and guidelines for various products, inherent flaws may remain undetected to human eye and lead to degradation. Conventional inspections in the workplace might overlook minute, tricky flaws, which take a lot of time to thoroughly examine from all sides. Traditional image processing algorithms may categorize products as defective or excellent using image classification and edge detection techniques, but they need professional assistance to fine-tune the parameters for effective outcomes. Furthermore, the lack of annotated data may affect the development of the model. To address the challenges in product inspection, Deep Learning technology can be utilized to develop an automated inspection system.

The proposed system will integrate a deep learning model, specifically Yolo v8 based, for the identification of defects in solar panels. To streamline the image pipeline and ensure efficient performance, we would like to integrate NVIDIA ROS platform that leverages GPU, and also have a web based user interface developed to view the output from the model, push new model and also control the pipeline from web(start/stop). Users can also classify and retrain the model with new anomalies, thereby enhancing defect detection accuracy. The project deliverables include framework design, data collection, modeling, evaluation and testing, and interface development. By using Yolo v8 for defect identification modeling and enabling users to rate the prediction and retrain the model, the system can address the challenge of limited data availability. The system's ability to be corrected and re-trained by users makes it a unique project capable of being deployed in the industry, thus significantly improving defect detection accuracy, and reducing maintenance costs. In this project, we will be concentrating on one such product i.e solar panels.

Acknowledgments

The authors are deeply indebted to Professor Kaikai Liu for his invaluable comments and assistance in the preparation of this study.

Anomaly detection with continuous data collection and modeling

Manish Kovelamudi, Vamshidhar Reddy Parupally, Ravindar Reddy Siddenki, Vikas Tadepu
Computer Engineering Department
San José State University (SJSU)
San José, CA, USA
Email: {manish.kovelamudi, vamshidharreddy.parupally, ravindarreddy.siddenki, vikas.tadepu}@sjsu.edu

***Abstract*—In this rapidly evolving world, in order to meet the demand, industries are transitioning to accelerated production processes. In such a fast-paced environment, ensuring that products are defect-free and of high quality is not just an advantage but a necessity. It's crucial to make sure that manufactured products are free from defects, in order to maximize desired output and save maintenance expenses. Although there are production standards and guidelines for various products, inherent flaws may remain undetected to human eye and lead to degradation. Conventional inspections in the workplace might overlook minute, tricky flaws, which take a lot of time to thoroughly examine from all sides. Traditional image processing algorithms may categorize products as defective or excellent using image classification and edge detection techniques, but they need professional assistance to fine-tune the parameters for effective outcomes. Furthermore, the lack of annotated data may affect the development of the model. To address the challenges in product inspection, Deep Learning technology can be utilized to develop an automated inspection system. The proposed system will integrate a deep learning model, specifically Yolo v8 based, for the identification of defects in solar panels. To streamline the image pipeline and ensure efficient performance, we would like to integrate NVIDIA ROS platform that leverages GPU, and also have a web based user interface developed to view the output from the model, push new model and also control the pipeline from web (start/stop). Users can also classify and retrain the model with new anomalies, thereby enhancing defect detection accuracy. The project deliverables include framework design, data collection, modeling, evaluation and testing, and interface development. By using Yolo v8 for defect identification modeling and enabling users to rate the prediction and retrain the model, the system can address the challenge of limited data availability. The system's ability to be corrected and re-trained by users makes it a unique project capable of being deployed in the industry, thus significantly improving defect detection accuracy, and reducing maintenance costs. In this project, we will be concentrating on one such product i.e solar panels.**

***Index Terms*—Anomaly, Solar Panels, YOLO, ROS, ROS Suite, Meta SAM, IOU, ONNX, Pytorch**

I. Introduction

With the increasing transition to clean energy, solar panels have become the core for the energy generation. The increased use of solar panels has created a need for faster and efficient production lines. Though the technology used for solar panels production is mature, there is a room for improving the efficiency and reducing cost for production. Presence of defects or faults can significantly impact the performance of the solar panels. Detecting and addressing these issues early in the production process is therefore critical to ensuring that solar panels perform optimally and have a long lifespan. Every production line has some techniques for identifying the defects and most of them are focused on manual or semi automated monitoring. The problem with these systems is that they might not be efficient for minute defect identification and cannot incorporate new defects discovered in the production line. As a result, there is growing interest in using artificial intelligence and machine learning techniques to automate the inspection process and improve the accuracy of defect detection. Applications for quality inspection are gaining importance and they are required to move towards a near minimal defect scenario. Deep learning has a great potential to achieve this system and can improve the accuracy of results obtained from computer vision techniques. This improvement would allow the machine learning model to incorporate latest features, and use transfer learning techniques to speed up the model development process. However, the efficiency of the inspection model would heavily rely on the amount of data available and generating the annotated data would be time consuming. Since the availability of data is limited and the data being quite similar to each other, there arises the problem of overfitting and reduced efficiency in the models developed. Of the 2.6 trillion dollars invested in renewable energy, most of it is invested in solar energy generation. This shows the importance of the solar panels, which are the primary source of energy generators. During the manufacturing process and assembly of the solar panels, events of mechanical stress or improper soldering, might lead to defective products and can impact the efficiency in the long run. Even a minute defect in the cell, after connecting with the panel group, would affect the performance and might spread across the module over time. This makes the process of inspection even more crucial and important. Thus an effective system has to be in place and this work provides a methodology to handle the issue. The current work addresses the issue by using a YOLO V8 model which allows anomaly detection from real time video feed. The model is a state of the art algorithm known for its capabilities in the field of fault detection from panels. With the help of real-time video feed and the use of GPU, the model can be quick and efficient. Also the use of ROS pipeline for image streaming would make this solution robust and efficient for the manufacturing industry. In order to utilize the ROS capabilities, Jetson developer kit has been employed which

will accelerate the development and deployment process with the power of GPU.

II. Problem Statement

The solar panel industry is humongous and produces solar panels each day, any defects in the panels that are detected after the production line create a lot of overhead to retrospect the defect and also to maintain the quality and commitment for the quantity of the delivery. Solar panels play a role, in the shift towards energy by generating efficient electricity. Although solar panel production technology is well established there are opportunities to improve efficiency and reduce costs. A major challenge in the production process is identifying defects or faults that can significantly impact panel performance. Traditional defect detection methods on production lines are often manual or semi automated making it difficult to identify defects or adapt to ones. Limited data availability, similarity between data and the risk of model overfitting further complicate the development of inspection models. Given the investment in energy ensuring the quality and durability of solar panels is essential. Minor defects caused by factors like stress or improper soldering during manufacturing can compromise an entire panel over time. Therefore there is a pressing need for an automated inspection system of swiftly and accurately detecting anomalies. The proposed solution involves utilizing the YOLO V8 model renowned for its fault detection capabilities, alongside real time video feeds and GPU processing. Using the ROS pipeline to stream images and incorporating the Jetson developer kit can greatly improve the effectiveness and efficiency of this solution, in the manufacturing sector.

III. Method and Architecture

A. Overall Architecture

The architecture proposed in this paper involves the ROS platform and pipelines, the YOLO V8 model, tools for segmentation and annotation, along with the user interface to facilitate the live streaming of the detections. The high-level flow of the architecture starts from web interface, from the user can start or stop the detection process. After starting, the camera turns on and continuously sends the video feed that is consumed by the ISAAC ROS pipeline. Then the independent ROS nodes inside Isaac ROS pipeline consume the image data which is pre-processed and model detections are applied (in this paper we are demonstrating the use of a custom model trained on top of YOLO V8 as the detection model) on each frame from the feed and draw the bounding boxes around the detections with a class label helping to identify the defects identified. The ROS pipeline then publishes the detections to the ROS topic defined for this purpose and hence we can subscribe to the topic to get the details about the detections made by the YOLO V8 detection model.

Once the detection processing is done on the video frame, the data will be transferred to the user interface as a live stream with minimal latency to incorporate near-real-time detections. Over the network, data transfer often results in data packet loss and creates lag in the stream feed compared to the actual feed. In order to overcome this problem and to achieve low latency feed, we have selected the ROS bridge suite to expose the data from the ROS container and connect to it using a local network, in which the machine that is hosting the ROS pipelines and the machine which is running the user interface, are connected to the same newtork. The ROS bridge suite, which is running on the same machine where ROS pipelines are running, exposes the ROS topics via a WebSocket and allows external sources to connect to the specific port and subscribe to the ROS topic. Since the two machines are connected to the same network the IP address would be the IP address of the router or the gateway to which the machines are connected. The messages published to the ROS topics are base64 encoded before being exposed to the subscriber and secure the data transfer over the network.

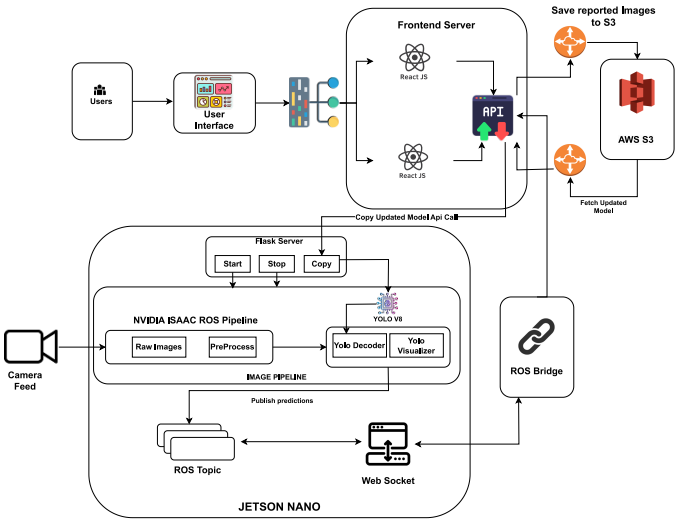

Fig. 1: Architecture

The data that is published to the ROS topic is now available to the user interface and can be subscribed to the topic using the IP address and the port number of the ROS bridge suite. To consume the messages on the user interface which is developed using React JS, a subscriber called RosLib package is used, which provides the UI with the capability to connect and subscribe to the topics exposed by the ROS bridge suite using the IP address of the network combined with the port that is exposed for this connection. This data stream is then displayed on the user interface, after decoding the base64 message received from the topic. The user interface further provides the capability to correct the detections and report the corrections to be used in the subsequent model training. Any reported corrections would be queued to be uploaded to the S3 bucket, without impacting the user interface application. An inbuilt annotation tool developed using Streamlit helps users correct the reported annotations, add new annotations, and modify the class of defects. To further extend the scope an image segmentation tool has been incorporated into the application to enable segmenting the defects from the panel. The reported data uploaded to the S3 bucket would be used to retrain the model to improve the accuracy. Detailed description of each component is listed below in this section.

B. Flow of the application

The application flow starts at the Home Screen, where users can view various graphs and have the option to upload new models. Moving to the Live Feed Screen, users have multiple functionalities: they can start the pipeline by selecting a model to be deployed on a Jetson device, which triggers a Flask server to initiate the pipeline and enables viewing of the live feed with detections. Users also have the ability to stop the running pipeline, report errors in detection, and toggle the view between raw and processed feed. Lastly, the Report page Screen presents a list of images reported by the user. Here, users can select an image and choose to either annotate or segment it, after which they are redirected to the respective screen for these tasks. This flow ensures an interactive and user-responsive system, catering to various operational needs within the application.

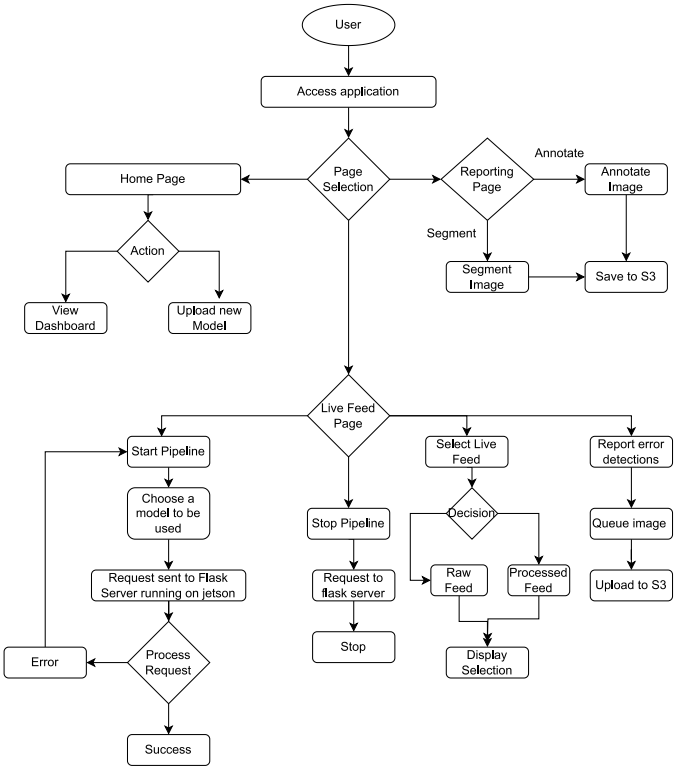

Fig. 2: Flow of the application

1) Upload new model: The accuracy of the model increases with the addition of the new data and continuous retraining of the model. Since the aim of this project is to integrate the continuous learning technique, we need a way to keep updating the new models and this is achieved through the feature where the application allows the users to select the models available to be pushed to the target device. The models that are trained on the gathered data would be pushed to the S3 bucket and are then retrieved and shown to the user to make a selection and push the model. This feature makes it convenient to replace the older models with the newer models. Refer Fig 6

2) Pipeline Startup : Application with an easy and intuitive user interface is expected. It is tedious to run more commands and scripts to start a pipeline and hence we have introduced a feature to control the pipeline from the user interface so that users can start and stop the ROS pipeline to start the detection model and also choose a specific model to be used for the launch. This utilizes the flask server APIs exposed to the user interface to send the request for running the scripts responsible for starting the pipeline in the background. The flask server is hosted on the Jetson device and the APIs are exposed through a shared network address. This feature gives users the control to select the model they want to use while deployment and can use the latest trained models by selecting them from the drop-down section available while starting the pipeline. Refer Fig 7

3) Live feed selection: ROS Pipeline provides two types of feed from the pipeline, one is the raw feed and the other is the processed feed with the bounding boxes of the detected classes. The application offers a control to switch between raw and processed feed and vice versa. This control helps users monitor the production line efficiently and would also play a key role in uploading the reported frames.

4) Stopping the pipeline: In order to maintain the ease of control, the application also provides user interface control to stop the pipeline that is running and would stop the streaming by killing the ROS Bridge, ROS Nodes, and the containers that are created as part of the startup process. This control is present in the live feed page and can be used effortlessly.

5) Report inaccurate predictions: The model might make mistakes in analyzing and detecting the errors in the solar panels, and hence the application provides an option to the users to report the inaccurate predictions which would be added to the queue and would be uploaded to the S3 bucket asynchronously.

The uploaded images can be viewed from the third page which provides the list of images reported and provides an option to either annotate the images with proper bounding boxes or segment the images. The selection made by the user would redirect the application to the specified page to either annotate or segment the desired image.

6) Annotation and segmentation: A larger dataset is required for the training of the model to make it accurate in detecting the defects on the panels. To overcome this constraint we have developed an application enabling users to report the images which are not accurately detected and facilitates them to correct annotations and upload them. This process is achieved with the help of Streamlit, where the application developed will allow the users to edit, modify, and create new bounding boxes and add the class labels to the region. These images are utilized for future model training.

Moving a step further, we have used an open-source code for image segmentation and integrated it into the same anomaly detection application to allow the users to capture the defects and segment it as shown in Fig. 4. These specific segments can be used by the model in the training phase.

7) API's involved: The flask server deployed on the jetson device provides the following API end points. Refer table I

C. ROS Platform

This section discusses details about the ROS (Robot Operating System) platform in the project, this platform is

TABLE I: API's Used for communication

API	***Purpose***
/run-script	Triggers the launch of the container in the jetson device.
/run-docker	Triggers the start of the container and make necessary startup installations in the container
/run-camera	Triggers the launch of the camera module once the ROS container is successfully started

crucial in this project as the main processing of images and detections happens in this platform. In ROS, there is a way for communicating between nodes in a publish-subscribe method. The publisher publishes data and subscriber receives that data. All the communication in this platform uses this method to send/receive data. This platform consists of nodes that work independently and perform specific operations. This platform consists of the following nodes and pipelines.

- Camera feed pipeline
- Nvidia Isaac ROS pipeline
  - Isaac ROS DNN Encoders node
  - Isaac ROS TensorRT Inference node
  - Isaac ROS DNN Decoder Node
- Visualizer node

Camera Feed Pipeline

This pipeline is launched using a python script, after launching, the pipeline gathers camera feed from the camera by capturing frames and sends it to Issac ROS pipeline. This pipeline is launched using GStreamer which leverages GPU instead of CPU, giving way for better and efficient processing.

Nvidia Isaac ROS Pipeline

This pipeline consists of three nodes and are combinedly responsible for processing the image, send the image to model for detection and retrieve the predicted detections. This pipeline uses DNN inference technique to process the image and output the detections. Internal working of each node is discussed below.

Isaac ROS DNN Encoders Node

In order for DNN inference to work, it requires input images as tensors, this node converts the image to tensors using pytorch and sends it to tensor RT inference node. This node also has inbuilt capabilities to perform pre-processing that is required by model such as grey scale conversions. This node can be replaced by custom encoder node, currently default encoder node is used.

Isaac ROS TensorRT Inference Node

This node performs the inference using the trained model. Tensor RT SDK developed by Nvidia is used to perform high performance deep learning inference. The trained model is loaded on the Jetson device and name of the file is sent as parameter while launching the node. Following types of models are supported ONNX Runtime, TensorRT Engine Plan, TensorFlow, PyTorch. Currently in this project, ONNX model is used as it gives better performance.

Isaac ROS DNN Decoder Node

This node gets the input from inference node and converts the output tensors to results that are human readable and can be easily used by application. Similar to encoder node, this node can be replaced by custom decoder node. In this project, custom decoder node is used.

Visualizer Node

This node gets input from decoder node and construct bounding boxes by extracting details from the results. After construction of bounding boxes, they are plotted on the image and sent to front end application.

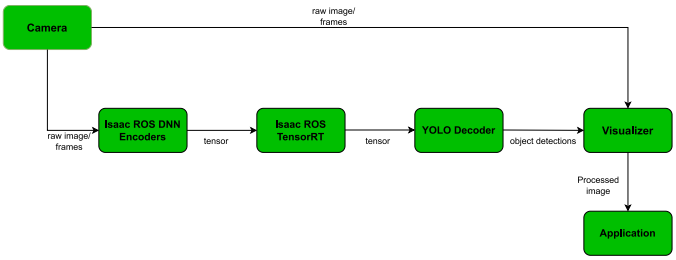

Fig. 3: Nodes

D. YOLO V8

YOLO model, popularly known as You Only Look Once is an object detection algorithm, which is popular for object detections. YOLO uses a single neural network on the image, unlike other existing models which uses classifiers on different parts of the picture/image with the help of protocols like sliding window. This feature of the YOLO makes it faster and more suitable for real-time detection. TensorFlow and PyTorch are the frameworks used by YOLO for its implementation. There are several versions of YOLO and in this paper we are using YOLO V8 which is the latest version of YOLO available.

The current version of YOLO V8 is made available by Ultralytics and can be installed using the pip commands. In this paper, we have used YOLO V8 for the purpose of detecting the defects on the solar panel from a video stream fed to the model using the ROS Image pipeline, and the model is deployed on NVIDIA Jetson to enhance the performance by utilizing cuda environment present in Jetson.

Ultralytics provides different ways to interact with the YOLO model through CLI as well as Python SDK. It supports different sets of operations ranging from classification, and detection through tracking poses, etc. For the purpose of defect identification in this paper, we have used a detection model which is trained on the image dataset of solar panels with annotations on the class of defects marked using the bounding boxes. By default torch model is obtained after training, this model is used to convert pytorch model to onnx (Open Neural Network Exchange) format, which is the only format

supported by Isaac ROS TensorRT Inference Node currently. The details of the dataset have been outlined in the dataset section of this paper.

The dataset is pre-processed and classified into training, validation, and test sets, using the 60:20:20 split before being used for the training of the model. Images are resized to 1024 and next, the annotations were adjusted to match the image dimensions and converted to YOLO-specific format and were used for the training of the model. The aim of this phase of the project is to develop a model that can accurately detect the defects from a solar panel image.

Pre-trained weights are used to leverage the advantage of transfer learning and to ensure stable convergence. The model is trained to learn to predict the bounding boxes with an associated confidence score, which shows the likelihood of the box containing the class of defect identified. The model was trained to minimize the loss and increase the confidence associated with the detection. Post-training, the model is evaluated for its performance using the test dataset of the images that are not used in the training phase. This evaluation ensured the model's reliability in detecting the defects. The challenge faced during this phase of the project is associated with the limited availability of the data, and this would be overcome by the concept of continuous learning where the users, with the help of the user interface, report the faulty detections and upload the corrected annotations which would be used to train the model in the subsequent phases to improve the accuracy of the model in identifying the defects.

E. Automated Image Segmentation

This section discusses about the research work done on possible integration of automated image segmentation to this project. In order to get better accuracy results while predicting anomalies on images, good data-set is required to train models, this image data-set must be annotated or segmented before training a model. When dealing with large data-sets it is often time consuming to manually segment each image and verify it. So, we researched on a possible way to automate segmentation or annotation of images. During this research we found cutting-edge models available in the market currently, these are Meta SAM(Segment Anything Model) and Yolo V8 (mask generator). These stock models cannot be used directly for our specific anomaly detection purpose. As these models support fine-tuning, a dataset of 100 images and its corresponding masks, which are manually segmented are used to fine-tune these models, which in-turn are used for predicting masks for labelled anomalies. After predicting, model gives IOU(Intersection of Union) and dice coefficient scores, which are the score to measure how well a predicted object aligns with the actual object annotation or measure of similarity between two expected and predicted data. After mask prediction, the idea is to extract this mask boundary and convert them to bounding boxes, and store them so that these images can later be used as a dataset for training new anomaly prediction models. The results of this approach is discussed in the results section.

IV. Application Setup

A. Dataset

The data set consists of solar panel grey scale images along with their segmentation separately in another file. The segmentation's are classified into 12 different classes "black_core", "crack", "finger", "star_crack", "thick_line", "corner", "fragment", "scratch", "printing_error", "horizontal_dislocation", "vertical_dislocation", "short_circuit". The total dataset size is 2.4Gb and consists of images, but only a few images are annotated or segmented. The segmentations initially are in the COCO format, they are converted to Yolo format.

B. Hardware and software

1) ***ROS setup****:* Nvidia Isaac ROS pipeline is the mainline pipeline used in this project. It will be run on the nvidia jetson device that contains all the libraries and SDK required to run the pipeline. Firstly, all the required code base is cloned into the jetson from github, then we run the container from Nvidia. Inside the container, we will be building the ROS2 humble package. Using this package, pipeline is launched using some set of commands.

2) ***Pipelines designed****:* There are total two pipelines that will be running on the jetson. First one is the camera pipeline, which will run using Gstreamer pipeline and will be taking the camera feed and sending it to Isaac ROS pipeline for processing.

3) ***ROS Integration with UI****:* ROS pipeline is in place to send the stream to the model and then publish the predictions to the ROS topic. But the topics are available in the machine that runs ROS nodes. These topics should be exposed to the outer world in a secure way. This brings us to the description of the ROS bridge suite, which exposes the ROS topics to outer consumers located in the network using web sockets. The setup includes the installation of the ROS bridge suite on the ROS machine and then the ROS lib js on the client machine. The ROS Bridge is configured to expose the topics on a specified port, which is subscribed on the client side with the help of ROS Lib JS, and the messages, which in this case is a video stream, are displayed on the user interface. Refer 4

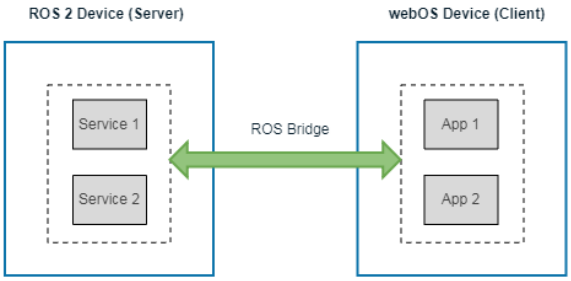

Fig. 4: ROS Bridge Suite

V. Results

The application contains different features like Live anomaly detections, reporting the detections, annotating, and segmenting the reported images.

Home Page: This is the landing page for the Anomaly detection application. It has details about the images reported, and annotated, and the status of the detections. It has links to navigate to other features of the application including live anomaly detections, annotating the images, and segmenting the images. Refer Fig. 5

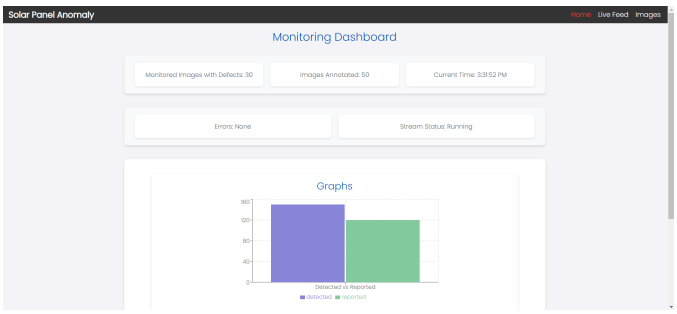

Fig. 5: Monitoring Dashboard

This page also hosts the functionality to upload the retrained model to be utilized in the subsequent runs of the pipeline. This has been achieved with the help of the S3 bucket and flask server APIs to push the model to the designated folder in the Jetson device. Refer Fig. 6

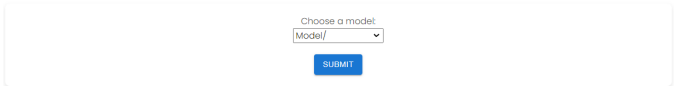

Fig. 6: New Model upload

After uploading new model, pipeline needs to be started with this latest model using pipeline startup option. Web application contains the functionality to start the pipeline with the required model. Refer Fig. 7

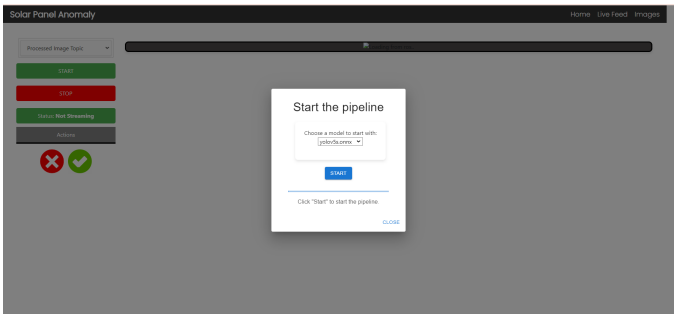

Fig. 7: Pipeline Startup

Live Anomaly Detection Page: This page streams the live detections of the solar panels from the ROS pipeline using ROS LIB. This allows the monitoring of the defects in the solar panels and reporting them as needed. Once the user reports the detection the image from the video stream is captured and stored to S3 bucket, this allows us to retrain the model based on the corrections suggested. Refer to Fig. 8. This page hosts the options to start, and stop the pipeline and to select the model to be utilized in each of the pipeline startups.

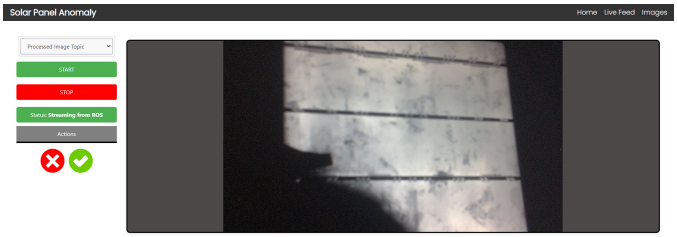

Fig. 8: Live Anomaly Detection Page

Annotation and Segmentation Page: This page allows users to annotate and segment the images that are previously reported from the live stream and would be stored in the S3 bucket with their respective bounding boxes and the segmentation information, which is further utilized in retraining the model. Refer Fig. 11 and 10 for the details.

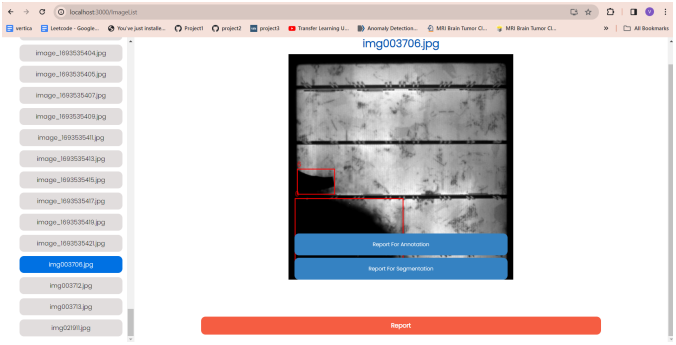

Fig. 9: Reported Images List

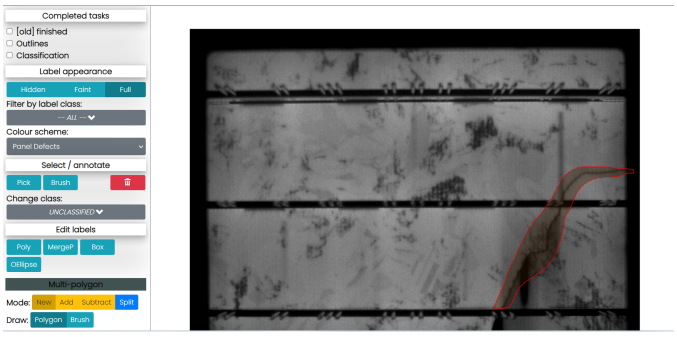

Fig. 10: Segmentation Tool

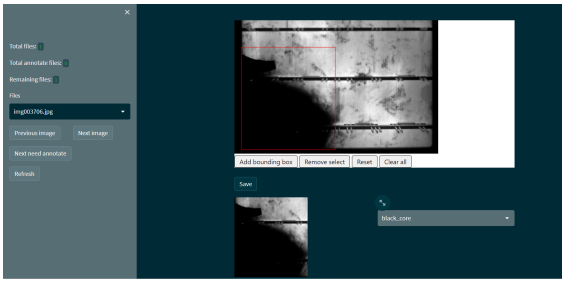

Fig. 11: Annotation Tool

A. *Latency Analysis*

The application relies on the network for the exchange of information from the Jetson device and the user interface and uses ROS Bridge Suite to publish and subscribe to the topics. Ultimately introducing some network latency in the process. The network latency though might be less, is worth making some analysis, to ensure that the data is not delayed longer than the required threshold. The latency for different topics has been plotted on the graph using a histogram, as below in Fig.12.

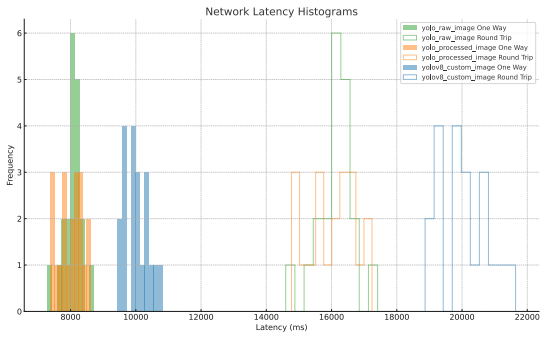

Fig. 12: Network Latency for Topics

The latency comparison for the ROS topics was performed by calculating the time intervals between the publishing and receiving of messages. Timestamp data were extracted from the ROS topics, specifically the 'Published At' and 'Received At' times. These times were parsed into datetime objects to facilitate precise calculations.The latency difference between the topic being published to the ROS topic and received in the user interface is around 6 seconds. The analysis is performed by comparing the time stamps at the time of publishing the event and the timestamp at the time of consuming it in the user interface. A sample of data of size 20 is used to perform the statistical analysis as shown in the Table II.

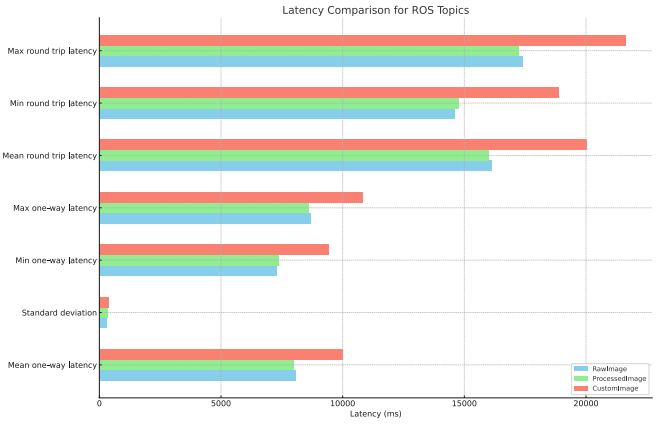

Fig. 13: Statistical Comparison of Latency

Since we are using the Flask server for the API requests, it is worth computing the latency, and the time difference between API calls made and API calls received.The response time

TABLE II: Latency Comparison for ROS Topics

Metric	**RawImage(Raw Frame)**	**ProcessedImage (Processed Frame)**	**CustomImage (Processed + Raw Frame)**
Mean one-way latency	8070.98 ms	8008.41 ms	10022.25 ms
Standard deviation	290.91 ms	354.26 ms	398.90 ms
Min one-way latency	7297.44 ms	7386.89 ms	9434.09 ms
Max one-way latency	8706.27 ms	8619.75 ms	10820.81 ms
Mean round trip latency(20 Frames)	16141.97 ms	16016.83 ms	20044.49 ms
Min round trip latency(20 Frames)	14594.88 ms	14773.79 ms	18868.19 ms
Max round trip latency(20 Frames)	17412.54 ms	17239.49 ms	21641.61 ms

depends on the type of command being passed to the server, in this case, the command passed is to list all the *.onnx* files present in the folder, which took around 1 minute to get the response back. The Table III shows the statistical distribution of the latency for two API's, and the data for the calculations has been obtained from 20 sample calls of the API's using postman tool. The latency calculations are performed based on the total time taken to process the request by removing the server processing time which gives us the network latency.

For run-command API, the mean total response time is relatively low at 1170.70 ms, with a tight standard deviation, indicating consistent and fast processing. In contrast, run-docker API exhibits a much higher mean response time of 128,902.89 ms. This is because the run-docker is performing more complex operations, encountering more variable network conditions, which result in a longer response time compared to the more streamlined run-command API call.

TABLE III: Latency Comparison for API Calls

Metric	**run-command API**	**run-docker API**
Mean	1270.70 ms	110,902.89 ms
Standard Deviation	69.65 ms	7462.40 ms
Min	1060.50 ms	118,821.63 ms
Max	1324.63 ms	146,969.27 ms
One-way Latency	1170.70 ms	64,451.45 ms
Round-Trip Latency	2344.40 ms	128,902.89 ms

B. Automated Mask Prediction Comparison

This section discusses about the output results, IOU (Intersection of Union) and dice coefficient scores given by the mask predictors of each model for 100 images. Table IV shows the comparison between two models (SAM and Yolo).

TABLE IV: Mask Prediction Comparison

Metrics	Meta SAM	Yolo V8 Mask Predictor
IOU Score (0-1)	0.24	0.18
Dice Co-efficient (0-1)	0.0063	0.0051

Fig. 14 compares the image and predicted mask from SAM model. The original image consists of a crack, which should

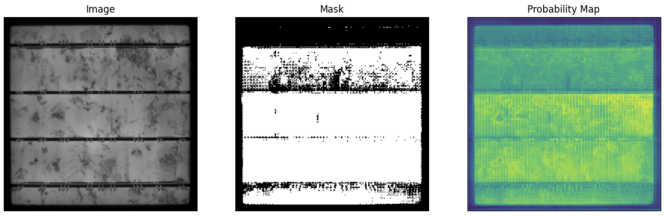

Fig. 14: Predicted Mask

be identified and mask should be generated on that area by fine tuned SAM. But from above visualized predicted mask, we can see that SAM model could not identify required anomalies and whole image is given as mask. Thus, this section concludes that SAM and Yolo V8 is not suitable for mask prediction and generation for this data-set.

VI. Discussion and Future Work

In this study, we have focused on the significance of the detection algorithms in the context of the solar industry. The research in this field led to the exploration of various techniques for detection including YOLO. The emphasis was laid on the development of infrastructure and a platform for the detection process packed in a containerized environment. Integrating ROS and YOLO models, as well as ROS and React Js, took a great amount of work and research in the background. To start with, we developed the YOLO V8 model and stored the weights, integrated it with the ROS pipelines, and connected it to ROS topics to store the results and detections into topics. This process helped us get meaningful insights in a standardized methodology.

A further extension to the project includes the addition of an automated pipeline for retraining the model, which now is not completely automated and requires minimal human interaction. Additionally integrating with Azure Ml-Ops or AWS Sage Maker could significantly increase the speed of development and deployments. In order with the current domain which is the solar industry, this solution can be extended further to different domains with just the input data being provided. For example, the user can provide the dataset with the images for a particular use case, and the system should train the model based on the data generate the model, and make it available to use.

VII. Related Works

Solar panel anomaly detection using deep learning models has been a popular topic of research in recent years. The effectiveness of transfer learning in training pre-trained models like ResNet50 or YOLOv3 on solar panel images has been investigated in multiple studies. Anomaly detection in images is a challenging task due to the presence of complex features, variations in lighting conditions, and the need for continuous data collection and modeling. Autoencoders, a type of neural network, have been used for anomaly detection in surface defect inspection of solar panels. According to Tsai, D. M., & Jen, P. H. (2021), autoencoder-based methods can learn to represent normal data effectively, allowing them to detect anomalies accurately [1]. In many studies, infrared images have been used for defect detection in photovoltaic modules and are proven to be accurate. A study proposed by Waqar Akram et al.(2020) on automatic detection of defects in PV cells describes the use of isolated deep learning and develop-model transfer techniques for anomaly detection, which uses infrared images to train the model and have achieved the accuracy of 99Furthermore, with the introduction of YOLO, much research has been done in this area and one such study we referred to for this project is the use of a multi-stage model for defect detection using a YOLO V3 model [3]. This paper talks about the combination of YOLO with computer vision and aims at performing two tasks with both detecting the panels in the frame and also detecting the defects from the detected panels. This study has encouraged us to consider the probability of detecting the defects from a video feed. Further research has revealed that the use of electroluminescence images for the detection of defects, is also considered to be a potential option and the paper published by W.Tang et al. (2020) on “Deep learning based automatic defect identification of photovoltaic modules using electroluminescence images” [4] reinforces the fact that the electroluminescence images coupled with deep learning leads to an effective detection algorithm. This paper uses a convolution neural network which is trained on the EL images and the resultant model is compared against the VGG16, ResNet50, and concludes that the model performs efficient detection compared to other models. A recent paper by Minhhuy Le et al . titled ”Remote anomaly detection and classification of solar photovoltaic modules based on deep neural network” proposed using IR camera to capture the images and residual network structure coupled with ensemble techniques to improve the accuracy of detecting the classes of defects [5]. This paper has demonstrated the performance of the model by using an IR dataset of solar farms, with 12 classes of defects and the model has recorded an accuracy of 94%. In addition to the above mentioned references, the paper “Machine learning framework . . . ” by V. S. B. Kurukuru et al . proposes to utilize a combination

of texture features and color information to classify images into healthy or defective categories [6]. The proposed framework employs various machine learning algorithms, including support vector machines (SVM), decision trees, and random forests, to achieve accurate detection and classification results. The authors reported a high classification accuracy of 96.83% using the proposed framework, demonstrating its effectiveness for photovoltaic module defect detection. In one of the works by Balzategui et al. , the authors have detected potentially defective areas in solar cell images by a sliding window approach with a convolutional neural network (CNN) designed for classification is often utilized [7]. This approach involves processing the cell images by breaking them down into smaller patches, which are then analyzed by CNN. The results are accumulated in a heatmap-like image, which highlights areas that have a higher probability of being defective.

The field of solar panel anomaly detection has seen significant progress in recent years, with various advanced techniques being developed for this purpose. One of the leading-edge tools is the use of convolutional neural networks (CNNs) and transfer learning. The paper on “Automated Detection of Solar Cell Defects with Deep Learning”, demonstrates how CNN can be used for inspection of solar panels for defects [8]. Transfer learning is particularly useful for small datasets as it enables the transfer of learned features from pre-trained models to new models, thereby reducing the need for large amounts of annotated data. Conventional algorithms for image processing rely extensively on human feature identification and extraction where the defect’s differentiating qualities will be utilized to analyze and interpret the images in order to expose the flaws. In order to separate the faults in the image, the background can be smoothened by “anisotropic diffusion filters” [9,10], “modified steerable filters” [11,12]. Alternatively, using “anisotropic diffusion” [13] or “frequency domain filters” [16] to eliminate faults, and then using the discrepancy between the processed and actual picture to map the problems. In paper “Classification of Manufacturing Defects in Multi crystalline Solar Cells with Novel Feature Descriptor” [14], the features from the faulty portions of the cells are extracted using a “CS-LBP” model, which is then utilized to build the models like K Means.

Autoencoder-based methods have also been used for surface defect inspection, with the ability to learn low-dimensional feature representations of input images. Multi-stage models on YOLO have been developed to identify defects in panels with the help of Infrared imaging by UAV vehicles. YOLOv3 is a popular real-time object detection system that uses a single CNN for both object detection and classification. Additionally, infrared (IR) imaging has emerged as an effective tool for detecting photovoltaic module defects, while electroluminescence imaging has been used for deep learning-based detections in solar panels. These techniques and tools have shown promising results and are expected to further improve the accuracy and efficiency of anomaly detection in solar panels, ultimately leading to improved performance and reliability of solar energy systems.

Recent advances in GAN-based models have shown great potential for automating the annotation of solar panel data, thereby reducing the time and cost associated with manual annotation. The authors of the paper ”Anomaly Detection and Automatic Labelling” propose a method for automatically labelling images of solar cells using a GAN [15]. The GAN is trained to generate images of normal solar cells, which are then compared to actual images of solar cells to identify anomalies. The generated images are used to provide annotations for the actual images, making it possible to train a deep learning model for anomaly detection without the need for extensive manual annotation. In another study conducted by Tsai et al. [16] and Zhang et al. [17], a technique known as Independent Component Analysis (ICA) is used to construct a demixing matrix from samples of solar cells that are known to be free of defects. This demixing matrix is then used to reconstruct images during the inspection stage, using the basis images that have been learned from the ICA process. By comparing the reconstructed images to the original images, the reconstruction error can be calculated and used to detect the presence of any defects. In this way, ICA can be a useful tool for detecting defects in solar cells and ensuring that they are functioning optimally. The paper “Automatic Solar Cell Diagnosis and treatment” [18], outlines how random forest is used in pixel classification in the defect regions which can also be considered as an important consideration.

VIII. Conclusion

In conclusion, this project represents a significant leap forward in the field of industrial anomaly detection. Our team successfully built the Anomaly detection system with a full-fledged platform based on ROS and a user interface based on React JS. This application seamlessly blended YOLO V8 detection algorithm, ROS Pipelines, and User interface to transform the way defects are detected in the industry.

The proposed system addresses the critical need for efficient, accurate, and automated detection of defects in a variety of industrial settings. By employing NVIDIA Jetson, we successfully utilized the power of GPU and reduced the dependency on the cloud to limit the latency in the system. We ensured that the application is both intuitive and performance-effective.

The application boasts a holistic production management system, simplified defect handling, and an efficient production line, all enriched by continuous training models specific to the industry.

Moreover, the application offers detailed product analytics, empowering users with the data they need to manage their products effectively. This leads to smarter production decisions, optimization of stock levels, and an overall enhancement in business operations. The amalgamation of Machine Learning-driven detection and our extensive set of features culminates in a unique and enriched production line experience, significantly benefiting, and boosting the success of the industry.

The comprehensive deliverables of the project, encompassing framework design, data collection, modeling, evaluation, testing, and interface development, promise a transformative impact on industrial operations. Ultimately, this system sets a

new standard in anomaly detection, promising to significantly elevate operational efficiency, ensure product quality, and reduce maintenance costs across various industries.

ACKNOWLEDGMENT

We would like to take this opportunity to thank Professor Kaikai Liu for his invaluable guidance, support, and feedback throughout the duration of this research project. His expertise and knowledge in the field have played a pivotal role in shaping our ideas and enhancing the overall quality of our work. Additionally, we would like to express our gratitude to San Jose State University for providing us with the necessary resources and facilities to conduct this research. We thank NVIDIA AI IOT team for developing a sample prototype that integrates Yolo and Isaac ROS. We also thank roboflow and ultralytics team for providing platforms to annotate and train models which we have done in initial phase of the project.

REFERENCES

[1] D.-M. Tsai and P.-H. Jen, 'Autoencoder-based anomaly detection for surface defect inspection', Advanced Engineering Informatics, vol. 48, p. 101272, 2021.

[2] M. W. Akram, G. Li, Y. Jin, X. Chen, C. Zhu, and A. Ahmad, 'Automatic detection of photovoltaic module defects in infrared images with isolated and develop-model transfer deep learning', Solar Energy, vol. 198, pp. 175–186,2020.

[3] A. Di Tommaso, A. Betti, G. Fontanelli, and B. Michelozzi, 'A multi-stage model based on YOLOv3 for defect detection in PV panels based on IR and visible imaging by unmanned aerial vehicle', Renewable Energy, vol. 193, pp. 941–962, 2022.

[4] W. Tang, Q. Yang, K. Xiong, and W. Yan, 'Deep learning based automatic defect identification of photovoltaic modules using electroluminescence images', Solar Energy, vol. 201, pp. 453–460, 2020.

[5] M. Le, V. S. Luong, D. K. Nguyen, V.-D. Dao, N. H. Vu, and H. H. T. Vu, 'Remote anomaly detection and classification of solar photovoltaic modules based on deep neural network', Sustainable Energy Technologies and Assessments, vol. 48, p. 101545, 2021.

[6] V. S. B. Kurukuru, A. Haque, A. K. Tripathy, and M. A. Khan, 'Machine learning framework for photovoltaic module defect detection with infrared images', International Journal of System Assurance Engineering and Management, vol. 13, no. 4, pp. 1771–1787, Aug. 2022 .

[7] Balzategui J., Eciolaza L., Arana-Arexolaleiba N., Altube J., Aguerre J., Legarda-Ereño I., Apraiz A. Semi-automatic quality inspection of solar cell based on Convolutional Neural Networks; Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA); Zaragoza, Spain. 10–13 September 2019; pp. 529–535.

[8] A. Bartler, L. Mauch, B. Yang, M. Reuter, and L. Stoicescu, 'Automated Detection of Solar Cell Defects with Deep Learning', in 2018 26th European Signal Processing Conference (EUSIPCO), 2018, pp. 2035–2039.

[9] J. Ko and J. Rheem, "Anisotropic diffusion based micro-crack inspection in polycrystalline solar wafers," in Proc. World Congress on Engineering 2012, London, UK, Jul. 4-6, 2012, vol. 2188, pp. 524-528.

[10] S. A. Anwar and M. Z. Abdullah, "Micro-crack detection of multicrystalline solar cells featuring an improved anisotropic diffusion filter and image segmentation technique," EURASIP Journal on Image and Video Processing, vol. 2014, pp. 1-17, 2014.

[11] H. Chen, H. Zhao, D. Han, and K. Liu, "Accurate and robust crack detection using steerable evidence filtering in electroluminescence images of solar cells," Opt. Lasers Eng., vol. 118, pp. 22-33, 2019.

[12] H. Chen, H. Zhao, D. Han, H. Yan, X. Zhang and K. Liu, "Robust Crack Defect Detection in Inhomogeneously Textured Surface of Near Infrared Images," in Proceedings of the Chinese Conference on Pattern Recognition and Computer Vision (PRCV), Guangzhou, China, 23-26 November 2018, pp. 511-523, Berlin/Heidelberg, Germany: Springer, 2018.

[13] D.M. Tsai, C.C. Chang, and S.M. Chao, "Micro-crack inspection in heterogeneously textured solar wafers using anisotropic diffusion," Image Vis. Comput., vol. 28, pp. 491-501, 2010.

[14] D.M. Tsai, S.C. Wu, and W.C. Li, "Defect detection of solar cells in electroluminescence images using Fourier image reconstruction," Sol. Energy Mater. Sol. Cells, vol. 99, pp. 250-262, 2012.

[15] B. Su, H. Chen, Y. Zhu, W. Liu, and K. Liu, 'Classification of Manufacturing Defects in Multicrystalline Solar Cells With Novel Feature Descriptor', IEEE Transactions on Instrumentation and Measurement, vol. 68, no. 12, pp. 4675–4688, 2019.

[16] J. Balzategui, L. Eciolaza, and D. Maestro-Watson, "Anomaly detection and automatic labeling for solar cell quality inspection based on generative Adversarial Network," Sensors, vol. 21, no. 13, p. 4361, 2021.

[17] D.-M. Tsai, S.-C. Wu, and W.-Y. Chiu, "Defect detection in solar modules using ICA basis images," IEEE Transactions on Industrial Informatics, vol. 9, no. 1, pp. 122–131, 2013.

[18] X. Zhang, H. Sun, Y. Zhou, J. Xi, and M. Li, "A novel method for surface defect detection of photovoltaic module based on Independent Component Analysis," Mathematical Problems in Engineering, vol. 2013, pp. 1–8, 2013.

[19] A. Rodriguez, C. Gonzalez, A. Fernandez, F. Rodriguez, T. Delgado, and M. Bellman, "Automatic Solar Cell Diagnosis and treatment," Journal of Intelligent Manufacturing, vol. 32, no. 4, pp. 1163–1172, 2020. Dwyer, B., Nelson, J. (2022), Solawetz, J., et. al. Roboflow (Version 1.0) [Software].