Solar Panel Detection on Satellite Images: From Faster R-CNN to YOLOv10

Stefan Elbl Droguett
Stanford Graduate School of Business
selbl@stanford.edu

Camila Nicollier Sanchez
Stanford School of Humanities and Sciences
camilans@stanford.edu

Abstract

In this report, we cover the end-to-end process of our project that focuses on the detection of solar panels in satellite images. We began by identifying and describing a meaningful problem to solve using computer vision. Then, we conducted a literature review to explore existing approaches to similar tasks and their strengths and weaknesses. Afterwards, we conducted an Exploratory Data Analysis (EDA) to gauge the quality of the chosen dataset and performed data cleaning. Once the dataset was properly formatted, we tried different methods to achieve our goal, starting with a baseline model with a Mask R-CNN architecture, exploring different Convolutional Neural Network (CNN) architectures and implementing the newly released and state-of-the-art YOLOv10 model. We tuned the hyperparameters of our models using the number of false positive and false negatives in the Confusion matrix as our performance metric. Finally, we present our Conclusions and Future work in the last section.

1. Introduction

Our project focuses on identifying solar panels on satellite images. Automating solar panel identification is a relevant task in the context of renewable energies, where the need to keep track of these installations has increased exponentially and solar developers have little to no tools to quickly identify existing projects in a specific area. Our motivation to tackle this problem stemmed from one of the authors, who interviewed over 150 relevant players in this action space who constantly mentioned the need to keep track of new renewable facilities online as an unsolved problem. That context made us consider the beneficial impact of achieving a high performance on that task. To that end, we applied different approaches and state-of-the-art architectures in order to obtain a high performance.

We used a dataset of satellite solar panel images from Beijing, China [1], and we implemented both a Mask R-CNN architecture and the CNN architecture embedded in the You Only Look Once (YOLO) models [2] as the main algorithms for our goal. Finally, our model's output is an image with a mask or bounding box and a confidence score (according to the architecture) for the Solar Panel class that we aim to identify in a satellite image.

2. Related Work

The existing approaches that are relevant to our work can be grouped into 3 categories: Existing approaches for solar panel detection in satellite images or similar tasks, Mask-CNN Architectures, and YOLO models.

Within the solar panel detection in satellite images or similar tasks category, we find an interesting approach in SolarX: Solar Panel Segmentation and Classification [3], which uses a UNet architecture [4] designed in 2015 to process biomedical images. [3] provides a baseline for us to compare what the object detection pipeline looked like 2 years ago. The authors mention in their conclusion that further exploration of different architectures is recommended, which inspired us to use methods we describe in further sections. Secondly, Deep-Learning-for-Solar-Panel-Recognition project [5] also focuses on object detection on solar panels and expands the dataset that we use by adding 700 Google Maps images and manually labeling their masks. The authors use the YOLOv5 architecture and obtain more favorable results than [3]. We reference their work for the metrics to measure performance in object detection and instance segmentation. Since they published their results, YOLOv9 and YOLOv10 were released. This made us consider these new architectures as an even more favorable venue for improvement. Finally, Object Detection in Autonomous Driving Vehicles [6] provided us with another reference on how to use the YOLO architecture, in this case, for a different type of task: autonomous driving.

For the second category, related to the Mask-CNN Architecture, we referenced the original papers of the Faster Mask-CNN [7] and the Mask-CNN [8] architectures. The Faster Mask-CNN algorithm was state-of-the-art in 2015 as it solved the region proposal computation bottleneck of existing object detection networks. Mask-CNN was released in 2017 in order to perform object detection and instance segmentation simultaneously. It achieved this by adding

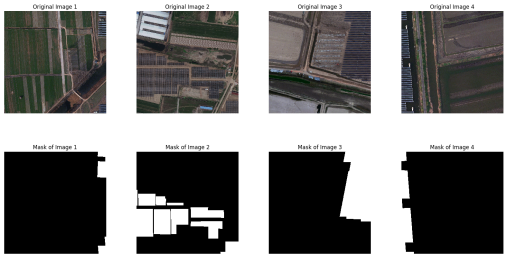

Figure 1. Example images and their masks

a branch for predicting an object mask in parallel with the existing branch for bounding box recognition in Faster Mask-CNN. After familiarizing ourselves with these architectures, we decided to use Faster Mask-CNN as a baseline model to compare with the state-of-the-art models in the final category.

Finally, we explored YOLO architectures and their evolution. We started by understanding the development timeline of the YOLO models with A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS [9]. We finished exploring the series with the last 2 versions released: YOLOv9 in February 2024 [10] and YOLOv10 in May 2024 [11]. After this review, we decided to use these two last versions as the core methods for our tasks, as we understand that this is the current predominant paradigm in object detection.

3. Dataset

We use a dataset of solar photovoltaic samples from satellite and aerial photography taken in Beijing, China [1]. The dataset contains bitmap images collected at the spatial resolution of 0.8m, 0.3m and 0.1m and the respective bitmask of the area in the picture that corresponds to the solar panel (in case there is one). There are two main categories of the data that correspond to the background in which the panel is installed: Ground (and its subcategories of land use: shrubwood, grassland, cropland, saline-alkali, and water surface) and Rooftops (with its subcategories of roof material: flat concrete, steel tile, and brick). The richness of the dataset allows us to detect solar panels in a variety of settings and thus increase the use cases of our model. Figure 1 provides examples of images in our dataset along with their masks.

Our dataset contains 3480 images, where 2745 have a ground background and 735 have a rooftop background. Table 1 provides a breakdown of the number of images by the specific type of background.

In the Exploratory Data Analysis, we first transformed images and masks from bitmap format to .png with a resolution of 256 x 256, and then sanitized the original dataset by deleting images that did not contain a solar panel.

Ground	Rooftop
Not Specified 673	Not Specified 90
Cropland 859	Flat Concrete 413
Saline Alkali 352	Brick 138
Water Surface 625	Steel Tile 94
Grassland 117	
Shrubwood 119	

Table 1. Number of Images by Background Type

3.1. Pre-processing for Mask-CNN baseline model

We applied the following task-relevant augmentations, in order to increase robustness when training our Mask-CNN baseline model:

- Resize to 50x50 pixels.
- Resize to 100x100 pixels.
- RandomPerspective with a distortion scale of 0.6 and a probability of 1.0.
- ColorJitter with a brightness factor of 0.6 and a hue factor of 0.3.
- GaussianBlur with a kernel size ranging from 5x5 to 9x9 pixels and a sigma ranging from 0.1 to 5.0.

We finally split the original dataset in the following train-test split, by taking into account that the augmentations are applied as part of the training process:

- Train: 3415 images
- Test: 65 images

3.2. Pre-processing for YOLO models

For training the YOLO models, we used the same dataset that consists of 3480 images. It is important to note that YOLO models include further augmentation processes which we omit for brevity. The train-val-split is as follows:

- Train: 3360 images
- Validation: 90 images
- Test: 30 images

4. Methods

We explored 2 learning algorithms to perform our object detection task. The first one is a Mask-CNN architecture, and the second one is a current state-of-the-art technology

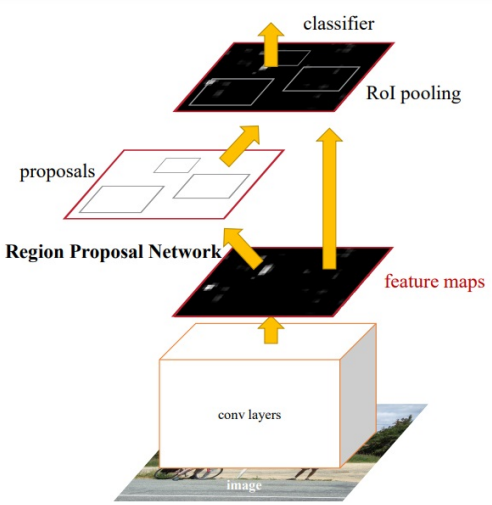

Figure 2. Faster R-CNN architecture (taken from [7]
)

for object detection YOLO architecture. Within the YOLO series, we implemented YOLOv9 and YOLOv10 which are the most recent versions released.

4.1. Faster Mask-CNN baseline model

As a baseline, we implement a Faster Mask R-CNN architecture [7], following an official TorchVision tutorial [12]. Faster Mask R-CNN builds upon the Fast R-CNN framework by introducing Region Proposal Networks (RPNs) that share convolutional layers with object detection networks such as Fast R-CNNs. As shown in Figure 2 from the original paper [7], the architecture begins with a CNN that extracts feature maps from the input image. Afterward, the algorithm uses a Region Proposal Network, which is also composed of several convolutional layers, to generate candidate boxes that undergo a Region of Interest (RoI) pooling operation that extracts fixed-size feature maps for each RoI. Finally, the extracted RoI features are fed into a final classification layer that predicts a bounding box with Class Probabilities for each class.

4.2. YOLO: You Only Look Once model series

The other learning algorithm we implemented is the YOLO architecture. We chose this algorithm because of its outstanding results and recent optimizations [13] that make it the state-of-the-art model for objection detection. In contrast to other architectures that use a pipeline of multiple models, YOLO is based on a single convolutional network that performs both detection and classification. Moreover, YOLO observes the entire image during both training and prediction, which in turn leads to fewer background errors.

YOLO models segment the input image into a square grid of S pixels. For each grid, the model predicts a fixed number B of bounding boxes and for each bounding box the model predicts: (i) the dimensions of the bounding box represented by its center and the width and height of the box relative to the entire image; and (ii) a confidence score of how likely it is that the predicted box contains and object as well as how accurate the box is. For multi-class settings, each grid cell predicts the probabilities of each box containing an object of a specific class conditional on the box containing an object.

YOLO models combine three different types of losses ([14]), depicted below:

$$\text{Total Loss} = \text{Loc Loss} + \text{Conf Loss} + \text{Class Loss} \quad (1)$$

Where:

$$\text{Loc Loss} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad (2)$$

$$\text{Conf Loss} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbf{1}_{ij}^{\text{obj}} (\hat{C}_i j - C_i j)^2 + \lambda_{\text{noobj}} \mathbf{1}_{ij}^{\text{noobj}} (\hat{C}_i - C_i j)^2 \right] \quad (3)$$

$$\text{Class Loss} = \sum_{i=0}^{S^2} \mathbf{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (4)$$

And:

- (x_i, j_i) predicted center coordinates of bounding box i with ground-truth $(\hat{x}_i, \hat{j}_i)$
- (w_i, h_i) predicted width and height of bounding box i with ground-truth $(\hat{w}_i, \hat{h}_i)$
- λ_k weight term to balance the loss in the event of k
- C_{ij} predicted confidence score for grid-cell i and bounding box j with ground-truth $\hat{C}_{ij}$
- $\mathbf{1}_k$ indicator function in case event k occurs
- $p_i(c)$ predicted probability of class c being present in grid-cell i with ground-truth $\hat{p}_i(c)$

Having a holistic loss function that combines multiple objectives helps YOLO to balance the accuracy of object localization, confidence of object presence and correctness of object classification.

During the project, we use the Ultralytics' Python API to implement YOLOv9. Because of how recent YOLOv10 was at the time of conducting our tests, there was no API from the Ultralytics package. So we cloned the repository of one of the authors of the pre-print of YOLOv10 [15].

4.2.1 YOLOv9

Building on the previously described architecture, YOLOv9 improves on its predecessors by addressing the information bottleneck principle: as data passes through more networks of a model, there is an increased potential for information loss. By implementing Programmable Gradient Information (PGI), introduced in the YOLOv9 paper that makes it so the target task receives complete input information, the authors argue that the impact of the bottleneck principle on YOLOv9 is reduced compared to its counterparts. YOLOv9 also expands on the usage of reversible functions (functions that can be inverted without loss of information) on its architecture, which ensures a reduction in the loss of information across layers. [10]

4.2.2 YOLOv10

YOLOv10 addresses two of the developers' main critiques with previous versions of the architecture: (i) computational redundancy which limits the model's performance; and (ii) the reliance on the non-maximum suppression (a technique in object detection that eliminates duplicate detections) for post-processing which impacts the inference latency and limits the deployment of previous models. The creators of YOLOv10 show that training without relying on non-maximum supression along with multiple other architectural improvements enhances the computational capabilities of YOLO and achieves new state-of-the-art performance [15].

5. Experiments and Results

5.1. Faster Mask-CNN baseline model

Figures 3, 4, 5, 6, 7 and 8 show the evolution of the different types of losses after training our baseline model for 6 epochs with the following hyperparameters:

- batch size = 1
- lr = 0.0005
- momentum = 0.9
- weight decay = 0.0005
- step size = 1
- gamma = 0.1

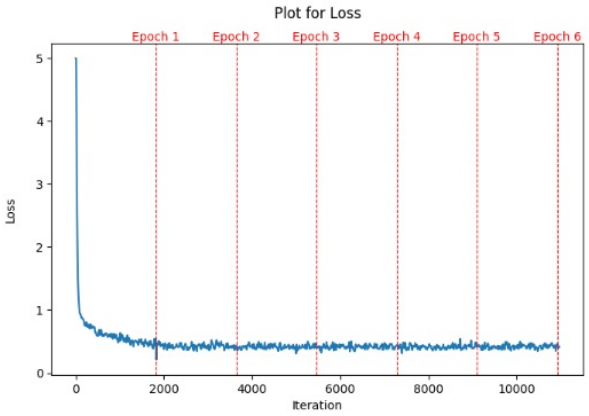

Figure 3. Overall Loss during training for 6 Epochs with 1820 iterations per epoch

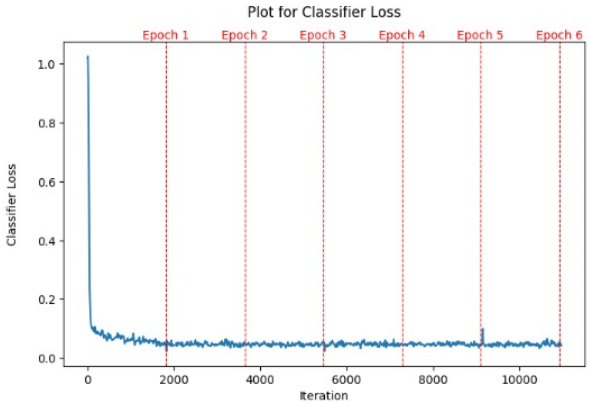

Figure 4. Classifier Loss during training for 6 Epochs with 1820 iterations per epoch

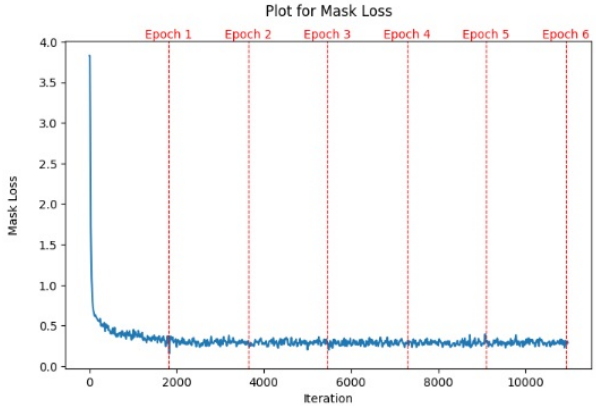

Figure 5. Mask Loss during training for 6 Epochs with 1820 iterations per epoch

We chose these hyperparameters after a manual process of trial and error because they were the ones that resulted in a reasonable loss decay over the iterations.

From the figures, we observe that the Overall Loss (Fig-

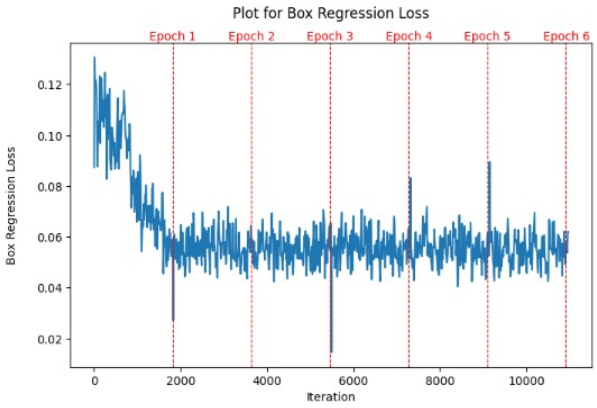

Figure 6. Box Regression Loss during training for 6 Epochs with 1820 iterations per epoch

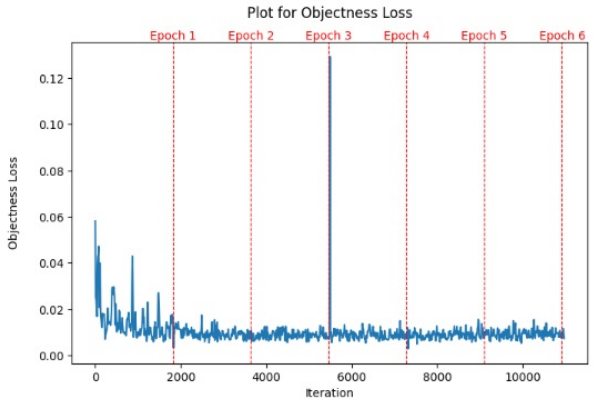

Figure 7. Objectness Loss during training for 6 Epochs with 1820 iterations per epoch

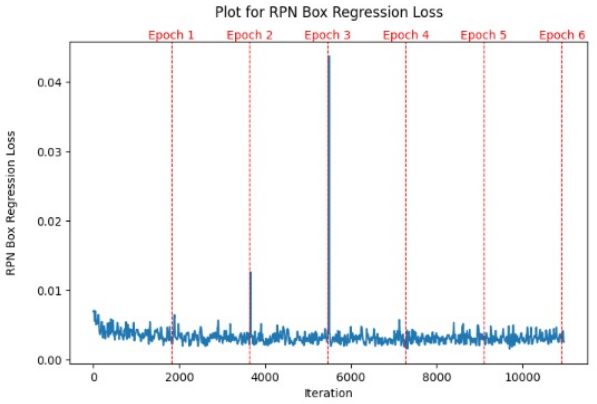

Figure 8. RPN Loss during training for 6 Epochs with 1820 iterations per epoch

ure 3), Classifier Loss (Figure 4), and Mask Loss (Figure 5) start at 5, 1.0, and 4.0 respectively. Then, they present a strong decay by iteration 20 and afterward, they continue to decrease at a slower pace. On the other hand, the Box Regression Loss (Figure 6), Objectness Loss (Figure 7), and

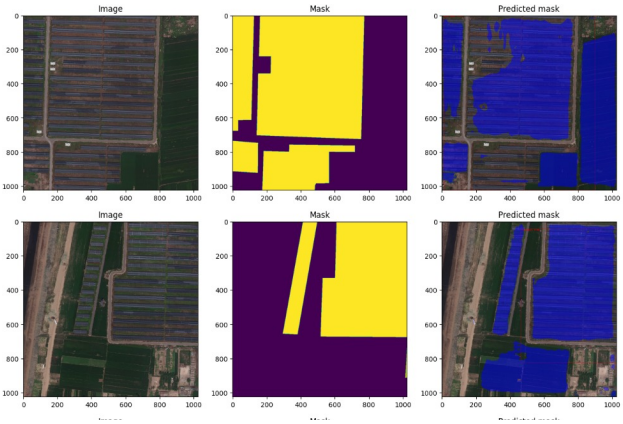

Figure 9. Performance metrics for Epoch 1

Metric	IoU	Area	Max Dets	Bbox	Segm
AP	0.5:0.95	All	100	0.115	0.086
AP	0.5	All	100	0.219	0.188
AP	0.75	All	100	0.109	0.065
AP	0.5:0.95	Small	100	-1.000	-1.000
AP	0.5:0.95	Medium	100	0.056	0.030
AP	0.5:0.95	Large	100	0.140	0.116
AR	0.5:0.95	All	1	0.208	0.184
AR	0.5:0.95	All	10	0.442	0.350
AR	0.5:0.95	All	100	0.548	0.412
AR	0.5:0.95	Small	100	-1.000	-1.000
AR	0.5:0.95	Medium	100	0.575	0.375
AR	0.5:0.95	Large	100	0.546	0.415

Table 2. Average Precision and Recall by IoU Metric, after training for 6 epochs

Region Proposal Network Box Regression Loss (Figure 8) start at 0.12, 0.06, and 0.01 and then present a much noisier decay with fluctuations over iterations. This is the expected behavior of the model and it is explained by the fact that we are using an optimized model that is already good at identifying Regions of Interest and doing inference on them, and where the main components of the Overall Loss come from the Classifier and Mask Losses, that are being learned from the new dataset. We stopped the training process at 6 epochs because the Losses did not decrease significantly in the last epochs.

When testing the trained model's performance on the test set, we obtain the metrics presented in Table 5.1. We observe that, for the bounding box creation task, the model has an Average Precision of 0.115 for IoU 0.5:0.95, with better performance in IoU up to 0.5 (AP= 0.219) and for larger objects (AP= 0.140). Besides that, the Average Recall metrics indicate that the model can correctly create a bounding box for 55% of the solar panels in the images, when considering a maximum of 100 detections per image.

Additionally, for the instance segmentation task, the Average Precision is lower for IoU 0.5:0.95 (AP = 0.086), and

we also observe better performance for IoU up to 0.5 (AP= 0.188) and for larger objects (AP= 0.116). In that sense, the Average Recall values suggest that the model correctly creates a segmentation mask for 41% of the solar panels in the images, when considering a maximum of 100 detections per image.

Finally, as a complement to the presented metrics, we can find in Figure 9 some examples of the original images and masks, together with the predicted masks and bounding boxes in blue and red, respectively. The baseline model is not as accurate as we need for this task. Even though it could be further optimized, we decided to use it as a starting point to increase performance with respect to it, with the other approach that uses YOLO models.

5.2. YOLO: You Only Look Once model series

In this section, we present the best results obtained after several iterations of different hyperparameters.

5.2.1 YOLOv9

When we started working on this project, YOLOv9 was the latest version of the YOLO and, based on the documentation found in Ultralytics, we chose to implement the 2 most suitable variants to our task: YOLOv9c and YOLOv9e[16]

- **YOLOv9c**: As mentioned in [16], the YOLOv9c model, in particular, highlights the effectiveness of the architecture's optimizations. It operates with 42% fewer parameters and 21% less computational demand than YOLOv7 AF, yet it achieves comparable accuracy, demonstrating YOLOv9's significant efficiency improvements.
  We trained our optimized model by setting the following hyperparameters:
  - batch size = 8
  - lr = 0.01
  - epochs = 30
  - patience = 10

  Figures 10 and 11 show our training results. In the first figures, we observe a steady decrease in the train and validation losses which indicates a good choice for the learning rate and a healthy learning process. Additionally, we observe that the model reaches a precision of 0.86 and a recall of 0.76 when taking into account the number of objects detected. We also observe a mAP 0.5:0.95 of 0.69 which is highly superior to the one obtained using Mask-CNN. Finally, analyzing the F1 score curve, the model achieves an F1 score of 0.82, which suggests an overall good performance balance between precision and recall.

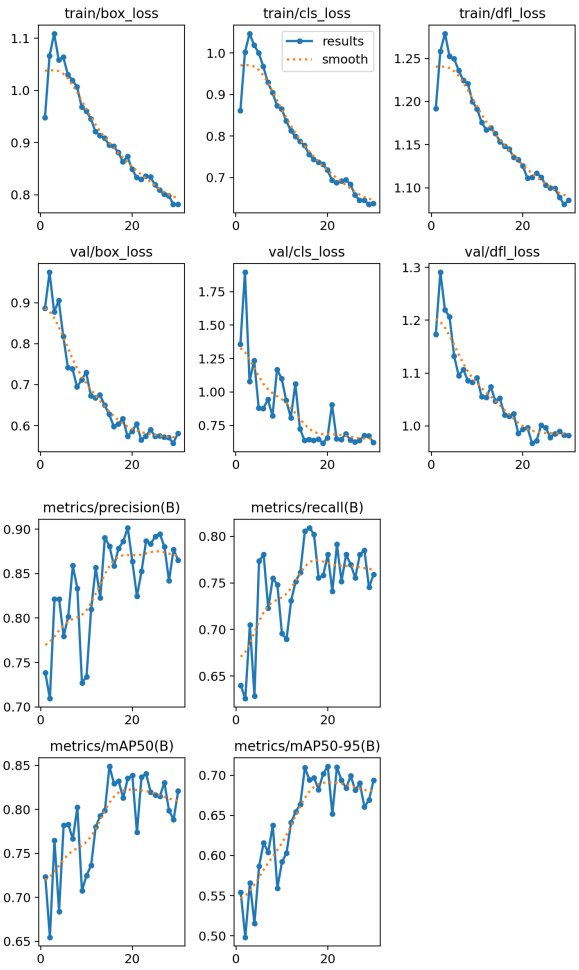

Figure 10. Results obtained after training YOLOv9c for 30 Epochs

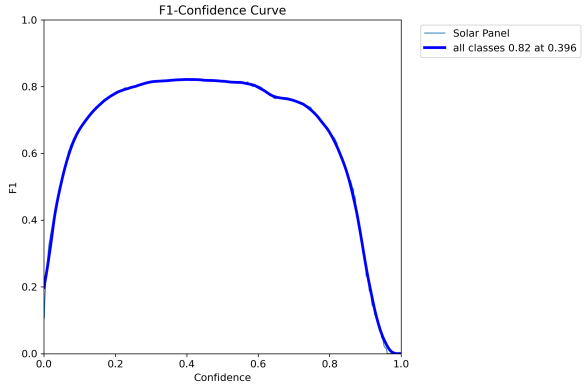

Figure 11. F1 curve obtained after training YOLOv9c for 30 Epochs

- **YOLOv9e**: YOLOv9e sets a new standard for large models, with 15% fewer parameters and 25% less computational need than YOLOv8x, alongside a incremental 1.7% improvement in AP [16]. We trained

this model with the same hyperparameters mentioned above, for 40 Epochs without early stopping. The results obtained are presented in Figure 12 and 13. In the first graphs, we observe similar behavior of the losses and a precision of 0.88 and a recall of 0.73, which is slightly higher than the previous model. We also observe a mAP 0.5:0.95 of 0.74 which is, again, superior to YOLOv9c. Finally, the model achieves an F1 score of 0.86. In Figure 14, we observe an example of the model predicting on the test set.

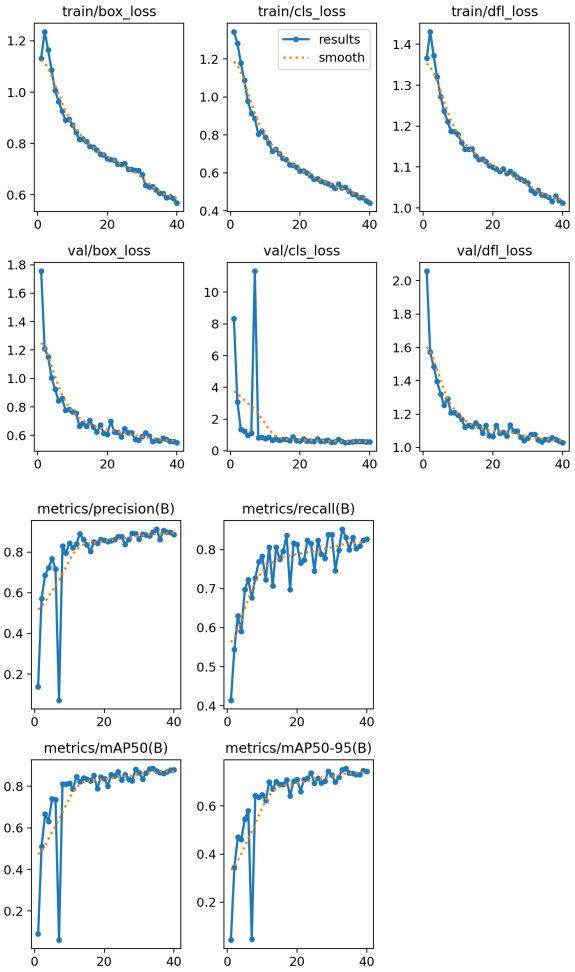

Figure 12. Results obtained after training YOLOv9e for 40 Epochs

Overall, we observe that with these hyperparameters YOLOv9e achieves a sufficient performance for the task of this project, which is superior to Mask-CNN baseline model and YOLOv9c.

5.2.2 YOLOv10

During the development of our project, Ultralytics released YOLOv10 on May 25, 2024. Our pursuit of state-of-the-art performance led us to test its performance in our task even though the official API had not yet been released. We implemented the following 2 variants:

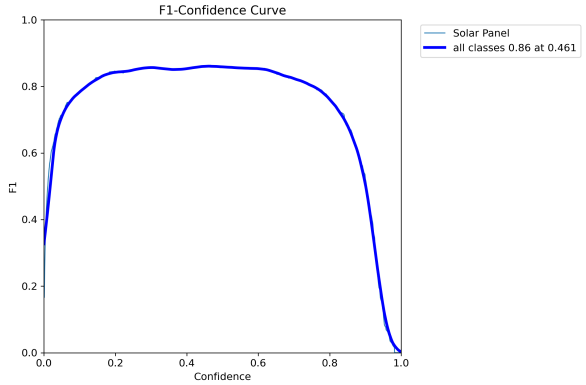

Figure 13. F1 curve obtained after training YOLOv9e for 40 Epochs

Figure 14. Predictions obtained with YOLOv9e trained for 40 Epochs

- **YOLOv10x**: This model is the extra-large version and it is optimized for maximum accuracy and performance [17]. It is important to highlight that we trained this model for 26 epochs but we only present the results from the last 6, as we had to train it in multiple incremental stages. The results obtained in the last 6 epochs are presented in Figure 15 and 16. In the first graphs, we observe an oscillation of the losses which is explained by the fact that we are only evaluating the last 6. The overall trend for the losses is similar to that presented in YOLOv9. In terms of precision and recall, we obtained 0.88 and 0.82 respectively. These metrics are better than those obtained with the YOLOv9 implementations and we were able to achieve them with only 26 epochs. We also observe a mAP 0.5:0.95 of 0.73 which is close to the YOLOv9e benchmark. Finally, the model achieves an F1 score of 0.84.
- **YOLOv10l**: It is important to mention that we also im-

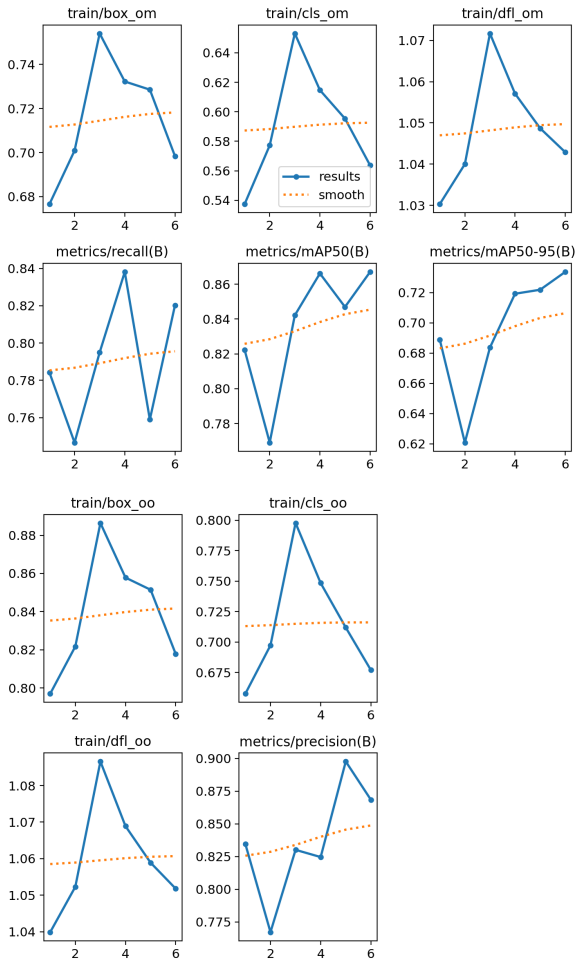

Figure 15. Results obtained for the last 6 epochs, after training YOLOv10x for 26 Epochs

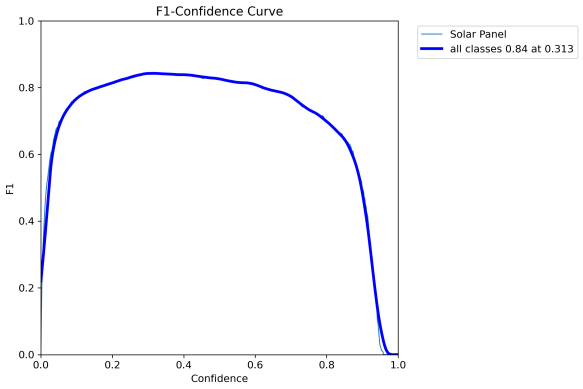

Figure 16. F1 curve obtained for the last 6 epochs, after training YOLOv10x for 26 Epochs

plemented the YOLOv10l variant, which is the large version model for higher accuracy at the cost of increased computational resources. We trained for 20 epochs and the results we obtained for this implementation were similar to the ones presented before for YOLOv10x.

6. Conclusion and Future Work

We present multiple approaches to perform the task of detecting solar panels in satellite images. As described in the Methods and Results section, we chose to use a Faster Mask-CNN architecture as a baseline and then explored the YOLO models. For our task and dataset, YOLOv9e presented the best performance and suitability for the task.

We were also excited to implement YOLOv10 a few days after its release and, even though we did not have enough time to optimize it as much as we did with YOLOv9, we believe that its results are very promising and should set the base for future work, especially once the API is released and initial bugs are corrected.

As [3] mentions, there are currently documented approaches that complete this task using U-Net architecture, Faster Mask-CNN, and YOLO. However, there is still room for exploration of other architectures such as Single Shot MultiBox Detector [18] and RetinaNet [19].

7. Contributions and Acknowledgements

This research project builds on the strong base created by the cited references and literature. Additionally, the code for this project was created using the following existing libraries: numpy [20], matplotlib ([21]), seaborn ([22]), Ultralytics ([10]) and boto3 [23].

Regarding the division of tasks among the teammates, each one owned different workstreams, which came with the responsibility of enforcing the respective deadlines and we also worked together in weekly sessions. Although we split the workload, we made sure to be involved with each other's tasks as a way to support and learn from each other. Camila took charge of formatting the dataset for both Fast R-CNN and YOLO models, training the YOLOv9e architecture, optimizing hyperparameters to use and constantly looking for references for our project. Stefan set up the AWS and Colab environments for training our models and oversaw the training of the Fast R-CNN, YOLOv9c, and YOLOv10 models.

References

[1] Liu Yujun Jiang Hou, Yao Ling. Multi-resolution dataset for photovoltaic panel segmentation from satellite and aerial imagery (v1.0), 2021.

[2] Ayush Chaurasia View RizwanMunawar, Glenn Jocher. Ultralytics yolo. software, 2024.

[3] Ethan Hellman Spencer Paul, Rodrigo Nieto. Solarx: Solar panel segmentation and classification. 2022.

[4] Thomas Brox Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, 2015. Conditionally accepted at MICCAI 2015.

[5] Deep-learning-for-solar-panel-recognition. Technical report.

[6] Rui Chen Adil Sadik, Qianli Song. Object detection in autonomou driving vehicles. Technical report, 2022.

[7] Ross Girshick Jian Sun Shaoqing Ren, Kaiming He. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. Extended tech report.

[8] Piotr Dollar Ross Girshick Kaiming He, Georgia Gkioxari. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017. Open source; appendix on more results.

[9] Diana Cordova-Esparza Juan Terven. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5:1680–1716, 2023.

[10] Hong-Yuan Mark Liao Chien-Yao Wang, I-Hau Yeh. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*, 2, 2024.

[11] Lihao Liu Kai Chen Zijia Lin Jungong Han Guiguang Ding Ao Wang, Hui Chen. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024.

[12] Torchvision object detection finetuning tutorial. Technical report.

[13] Khwab Kalra. Yolo: A state-of-the-art object detection model, 2023. Medium.

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[15] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024.

[16] Chien-Yao Wang and Hong-Yuan Mark Liao. YOLOv9: Learning what you want to learn using programmable gradient information. 2024.

[17] Lihao Liu et al. Ao Wang, Hui Chen. Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*, 2024.

[18] Dumitru Erhan Christian Szegedy Scott Reed Cheng-Yang Fu Alexander C. Berg Wei Liu, Dragomir Anguelov. Ssd: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2016.

[19] Ross Girshick Kaiming He Piotr Dollár Tsung-Yi Lin, Priya Goyal. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[21] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[22] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[23] Amazon Web Services. Boto3 documentation. `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`, 2024.