uc3m | Universidad Carlos III de Madrid

Data Science Project

Data Science and Engineering

Detection of solar panels in satellite images

Sergio Aizcorbe Pardo · Ricardo Chavez Torres ·
Daniel de las Cuevas Turel · Zijun He · Sergio Hidalgo López

Index

Initial Report on Solar Panels Detection Project

0. Introduction

In recent years, the breakthrough of **renewable energy** and the decreased cost of installing **photovoltaic panels (PV)**, many people have chosen to utilize this technology in their homes, factories, buildings and farms. The use of solar panels reduces the amount of carbon and other pollutants emitted into the environment. The reduction of the pollutants in the atmosphere help maintain clearer water and air, which are critical resources not only for humans, but for other living creatures on our planet.

However, while more and more PV are being installed, distribution networks are also being affected due to the panels' **intermittent nature**. The respective distribution company may receive certain information from these installations, but additional information for validation is important.

The aim of the project is providing a machine learning framework to detect solar panels from **aerial images**, and provide data about the location and status of the panels, including their coordinates, area and number of cells. The implemented **object detection model** is meant to be utilized by **ATOS**, a company leader in secure and decarbonized digital which has proposed the idea of bringing this project to fruition.

1. Literature Reviews and methodology:

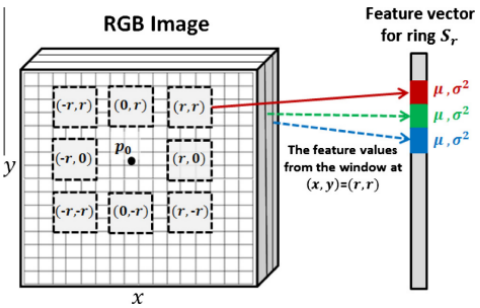
Several papers have been reviewed and their respective methods to detect solar panels have been studied. The project is based on two papers which give an insight of the latest techniques and benchmarks in the industry.

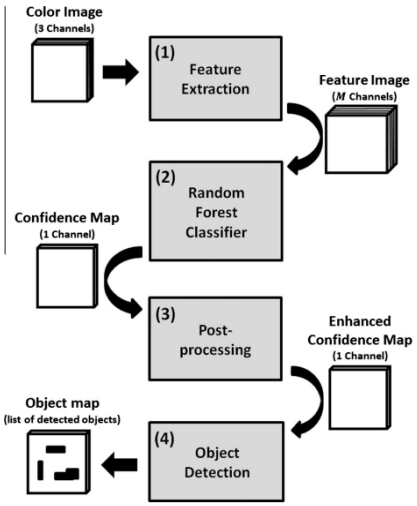
1.1 Automatic detection of solar photovoltaic arrays in high resolution aerial imagery (Malof et al. 2016): traditional feature extraction, RF

For the first paper, a slightly older approach is reviewed, that uses the classical feature extraction to process the images and extract the patterns and a Random Forest to predict the confidence levels. A team of human annotators is in charge of manually annotated PV arrays. The data used comes from the Aerial Dataset (Fresno, US.).

The process can be summarized as follows:

1. Extract features from the images around each pixel that characterize all the patterns
2. RF assigns a probability or confidence to each pixel of belonging to a PV array
3. Post processing to identify high confidence pixels (local maxima)
4. Object detection: identifying groups of contagious high confidence pixels

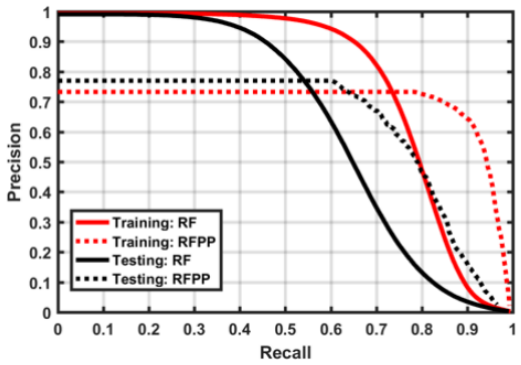




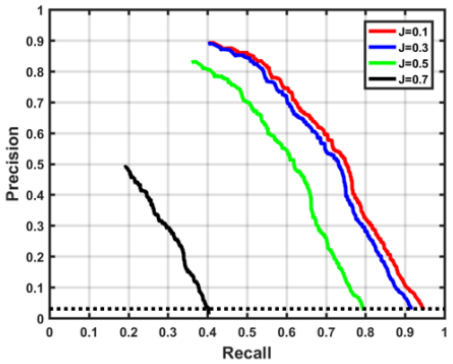
In order to link the automated detections to the human annotations, the Jaccard index is used. Depending on the goal of the problem, we would need either a really high value (almost perfect match) or just a value > 0 (it knows where the truth object is at, but can not detect it properly).

For the pixel-based performance, i.e. detecting if a pixel belongs to a solar panel array, the authors tried using only RF with and without post processing in both training and testing datasets.

The results showed that RF by itself overfits in the training set and its performance drops significantly in the testing set. On the other hand, using a post-processing technique afterwards fixes this issue and makes the model more able to generalize future predictions.



For the object-based case the goal is to identify the shape or size of the individual PV arrays. The performance here can be evaluated by varying the Jaccard index, since a lower one would be less demanding of exact precision and thus will provide a higher accuracy.



The results show that for J=0.1 the precision is 0.9 (90% of total detections are actually true detections). However, for a higher J index the performance drops, since it becomes a much more difficult task.

In conclusion, this paper represents a start-off point in object recognition using traditional methods. It is not the most efficient one, but one of the fastest. For our problem, we will not be worried about the exact location of the solar panel on the rooftop, but rather in the detection within a building.

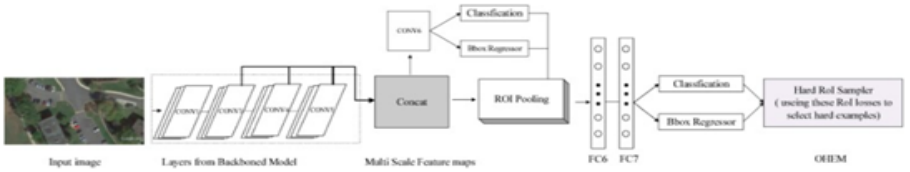
1.2 A light and faster regional convolutional neural network for object detection in optical remote sensing images (Ding et al. 2018)

This paper introduces a scheme of improvement based on the Faster R-CNN for detecting small objects from remote sensing images while trying to minimize the time spent. They test this model on vehicles and aircrafts detection.

Procedures:

1. Pre-trained model is used to train a limited amount of labeled data (VGG16).

2. Online Hard Example Mining (OHEM) is used to select the poorly classified samples by their loss, accumulate the gradients and pass them to the convolutional network. The structure is shown below:

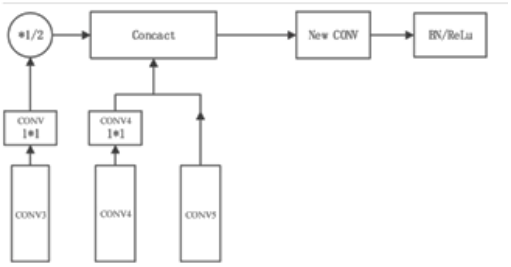


Download : Download high-res image (196KB) Download : Download full-size image

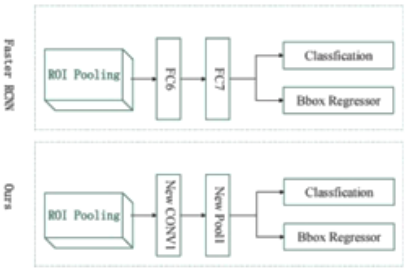
Fig. 4. Flow chart of Faster RCNN with OHEM and multi-scale prediction.

3. In order to keep the resolution while expanding the size of the receptive field, the pool4 layer is removed from the pretrained model and the size of filters in Conv5 are extended to 2.

4. Multi-scale representation is used. Instead of using fixed size feature maps of the last layer of the CNN part to extract candidate regions, it uses the concatenation of results from Conv3, Conv4 and Conv5 and generates a new convolutional layer, in which it applies a initialized method called Xavier, Batch Normalization and ReLu. Detailed flowchart is shown below:



5. The 2 fully connected layers to perform classification and regression are substituted by a convolutional layer and a pooling layer in order to save the computational time by reducing the number of parameters generated.



Evaluation

Average precision and recall are the main metrics used to evaluate the performance of the model. With all the previously mentioned procedures, 1000 images of aircrafts are trained resulting in an average precision of 0.907 and a recall of 0.9685. On a car dataset, the average precision is 0.879 and the recall is 0.8846. All results are improved, compared to the results of using only Faster R-CNN with VGG16-Net.

1.3 Paper data comparison

Paper number	Number of images in train/test	Type of Problem	Techniques	Evaluation Criteria	Performance
1	Training: 1780 (90km²) Testing: 1014 (45km²) + Annotations	Supervised (manually annotated)	Classical feature extraction + Random Forest	PR curves (precision - recall balance) + Jaccard index (for object detection)	Object (array) detection precision: 0.9 (J=0.1)
2	1000 images with about 7000 aircrafts 500 images with about 7000 cars	Supervised pre-trained model + Domain-specific fine-tuning	Faster R-CNN + OHEM + Multi-scale Representation	Average Precision(AP) + Recall Score	Aircraft dataset AP: 0.907 Recall: 0.9685 Car dataset AP: 0.879 Recall: 0.8846

Analysing these results we decided to base the first scope of the project on the use of a **Convolutional Neural Network** as a model to detect solar panels. This was in part due to the large number of architecture backbones that exist for object detection.

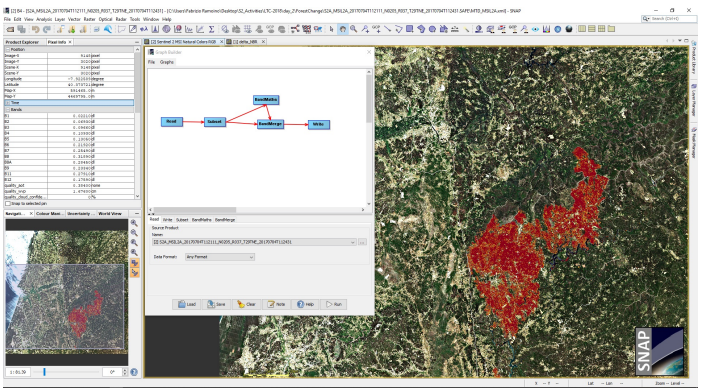
2. Data gathering

2.1 Copernicus - Mundi

One of the recommendations given by ATOS to obtain the images was utilizing **Copernicus API** which provides free and open access to Sentinel satellites data.

During the early stages of the project, a software called **SNAP Toolbox** was used to process data from **Sentinel-2** satellite, specifically from the region tile corresponding to Madrid (T30TVK). The resulting data contained multiple images in different spatial resolutions with their respective information of the electromagnetic spectrum.

The major issue found in this phase was the difficulty of processing **geospatial data**. We could not ensure that a deep learning model would be able to detect solar panels in these kinds of images. Therefore, due to the short period for the implementation of the project and the specificity of the task, an **alternative source** to get satellite data was considered.



Here is an example of SNAP Toolbox, a software designed by the European Space Agency to analyze Sentinel satellite data.

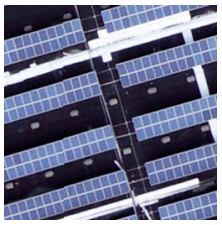
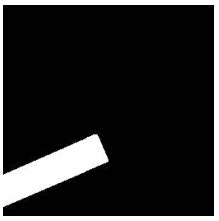
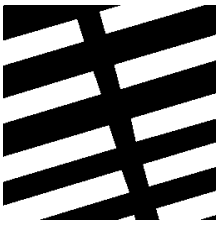
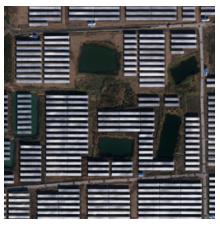
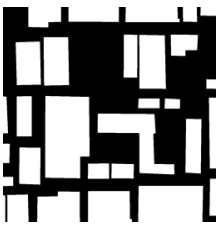
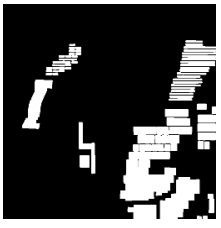
2.2 Dataset

In order to train a model to detect panels a relatively **large sample of images** is needed. Nevertheless, there is a lack of datasets that contain images from solar panels taken from an aerial point of view. This limitation greatly affects the development of deep learning based object detection methods. However, in August 2021, the School of Engineering of China released one of the most complete datasets of satellite images (7GB) containing photovoltaic cells (Jiang et al. 2021).

The dataset contains aerial images of solar panels with **spatial resolutions of 0.8 m, 0.3 m and 0.1 m**, a total of **3716 samples** of photovoltaic panels grouped by its surrounding terrain (ground solar panels and rooftop solar panels). There are several types of **terrains** (*shrub land, grassland, cropland, saline-alkali, water surface*) and **roofs** (*flat concrete, steel tile, brick roofs*).

Spatial Resolution	Ground	Rooftop	TOTAL
PV01 (0.1m)	0	645	645
PV03 (0.3m)	2122	186	2308
PV08 (0.8m)	673	90	763
			3716

One of the main advantages of the dataset with respect to the raw satellite data is the presence of **binary masks** representing each solar panel as well as the format of the images (bmp) which is more suitable for training a model.

Spatial Resolution	Sample images		Mask	
PV01 (0.1 m)				
PV03 (0.3 m)				
PV08 (0.8 m)				

2.3 Google Maps image tiles

Since acquiring aerial images directly from satellites has not been feasible, in order to evaluate our model with **real results**, a custom tool has been developed which interacts directly from **Google Maps**. Given a set of coordinates and a zoom parameter, a series of **256x256 regions tiles** are downloaded.



One of the major issues found utilizing Google Maps API, was its **rate limit** when making requests for the data. Therefore, a new implementation was developed which emulates a browser request and saves the respective responses in PNG images. Taking advantage of Python's **multiprocessing** library, the average download and processing speed of the data was increased (200 tiles/second).

2.4 Data pre-process

For ensuring a correct functioning of the model, all of the dataset images have been converted to PNG format. The initial approach for the **training and evaluation phases** was utilizing the solar panel dataset with the respective binary masks.

.

Train	Validation	Test
2377	595	744 images
80 %	20 %	
80 %		20 %

As there are not many images in this dataset that do **not contain** a solar panel, new images from Google Maps might be added to the training set in the future to make sure the network generalizes well enough and minimize the number of false predictions.

Furthermore, the resulting model has also been tested with some images from Google Maps.

3. Evaluation of different CNN architectures

Our initial approach is based on the general accuracy of each CNN architecture. In order to study which architecture we will be implementing, we took a look at *Object Detection in Optical Remote Sensing Images: A Survey and A New Benchmark* paper which compares several solar panel detection benchmarks and classifies the performance of the models by its backbone architecture.

Table 3
Detection average precision (%) of 12 representative methods on the proposed DIOR test set. The entries with the best APs for each object category are bold-faced.

c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
Airplane	Airport	Baseball field	Basketball court	Bridge	Chimney	Dam	Expressway service area	Expressway toll station	Golf course
c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
Ground track field	Harbor	Overpass	Ship	Stadium	Storage tank	Tennis court	Train station	Vehicle	Wind mill

	Backbone	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20	mAP
R-CNN	VGG16	35.6	43.0	53.8	62.3	15.6	53.7	33.7	50.2	33.5	50.1	49.3	39.5	30.9	9.1	60.8	18.0	54.0	36.1	9.1	16.4	37.7
RICNN	VGG16	39.1	61.0	60.1	66.3	25.3	63.3	41.1	51.7	36.6	55.9	58.9	43.5	39.0	9.1	61.1	19.1	63.5	46.1	11.4	31.5	44.2
RICAOD	VGG16	42.2	69.7	62.0	79.0	27.7	68.9	50.1	60.5	49.3	64.4	65.3	42.3	46.8	11.7	53.5	24.5	70.3	53.3	20.4	56.2	50.9
RIFD-CNN	VGG16	56.6	53.2	**79.9**	69.0	29.0	71.5	63.1	69.0	56.0	68.9	62.4	**51.2**	51.1	31.7	**73.6**	41.5	79.5	40.1	28.5	46.9	56.1
Faster R-CNN	VGG16	53.6	49.3	78.8	66.2	28.0	70.9	62.3	69.0	55.2	68.0	56.9	50.2	50.1	27.7	73.0	39.8	75.2	38.6	23.6	45.4	54.1
SSD	VGG16	59.5	72.7	72.4	75.7	29.7	65.8	56.6	63.5	53.1	65.3	68.6	49.4	48.1	59.2	61.0	46.6	76.3	55.1	27.4	65.7	58.6
YOLOv3	Darknet-53	**72.2**	29.2	74.0	78.6	31.2	69.7	26.9	48.6	54.4	31.1	61.1	44.9	49.7	**87.4**	70.6	**68.7**	**87.3**	29.4	**48.3**	78.7	57.1
Faster RCNN with FPN	ResNet-50	54.1	71.4	63.3	81.0	42.6	72.5	57.5	68.7	62.1	73.1	76.5	42.8	56.0	71.8	57.0	53.5	81.2	53.0	43.1	80.9	63.1
	ResNet-101	54.0	74.5	63.3	80.7	44.8	72.5	60.0	75.6	62.3	76.0	76.8	46.4	57.2	71.8	68.3	53.8	81.1	59.5	43.1	81.2	65.1
Mask-RCNN with FPN	ResNet-50	53.8	72.3	63.2	81.0	38.7	72.6	55.9	71.6	67.0	73.0	75.8	44.2	56.5	71.9	58.6	53.6	81.1	54.0	43.1	81.1	63.5
	ResNet-101	53.9	76.6	63.2	80.9	40.2	72.5	60.4	76.3	62.5	76.0	75.9	46.5	57.4	71.8	68.3	53.7	81.0	**62.3**	43.0	81.0	65.2
RetinaNet	ResNet-50	53.7	77.3	69.0	81.3	44.1	72.3	62.5	76.2	66.0	77.7	74.2	50.7	59.6	71.2	69.3	44.8	81.3	54.2	45.1	83.4	65.7
	ResNet-101	53.3	77.0	69.3	**85.0**	44.1	73.2	62.4	78.6	62.8	78.6	76.6	49.9	59.6	71.1	68.4	45.8	81.3	55.2	44.4	85.5	**66.1**
PANet	ResNet-50	61.9	70.4	71.0	80.4	38.9	72.5	56.6	68.4	60.0	69.0	74.6	41.6	55.8	71.7	72.9	62.3	81.2	54.6	48.2	**86.7**	63.8
	ResNet-101	60.2	72.0	70.6	80.5	43.6	72.3	61.4	72.1	66.7	72.0	73.4	45.3	56.9	71.7	70.4	62.0	80.9	57.0	47.2	84.5	**66.1**
CornerNet	Hourglass-104	58.8	**84.2**	72.0	80.8	**46.4**	**75.3**	**64.3**	**81.6**	**76.3**	**79.5**	**79.5**	26.1	**60.6**	37.6	70.7	45.2	84.0	57.1	43.0	75.9	64.9

Object detection in optical remote sensing images: A survey and a new benchmark (K. Li et al. 2020)

So far we have been working on **YOLO** (Bochkovskiy et al. 2020; Jocher et al. 2020)**, YOLACT** (Bolya et al. 2019) and **Mask-RCNN** (He et al. 2017) to find which one serves us best to later on based on the model on it.

3.1 Similarities and differences between approaches

We are aware of how each different architecture will affect our results based on the task that they are trying to tackle and thus, have decided to take a look at the similarities that they have like the creation of files where their bounding boxes are located to feed the same data to our model.

The end goal of this approach is to generalize our models as much as possible in order to be able to view the results in a more general manner so comparisons can be made easily.

3.2 Problems and challenges

Each of the selected implementations needed a different annotation format for the bounding boxes.

- YOLO (Bochkovskiy et al. 2020; Jocher et al. 2020) architecture has its own annotation format where its image has a corresponding *txt* file containing the bounding box and the number of the class, solar panels in this case.
- YOLACT (Bolya et al. 2019) architecture uses COCO annotation format, where the images and bounding boxes information is obtained from an annotation file written in JSON format. An annotation generator was made to transform the binary masks to bounding boxes and segments. Nevertheless, when YOLACT's model was being trained, the expected time to compute 50 epochs exceeded 20 days, thus this backbone architecture was discarded.
- The Mask-RCNN (He et al. 2017) library chosen for modeling was implemented using TensorFlow, a framework in which the team has not gathered as much experience as using Pytorch. We may still give it a last chance in the future, but so far, results using this approach have been worse than expected.

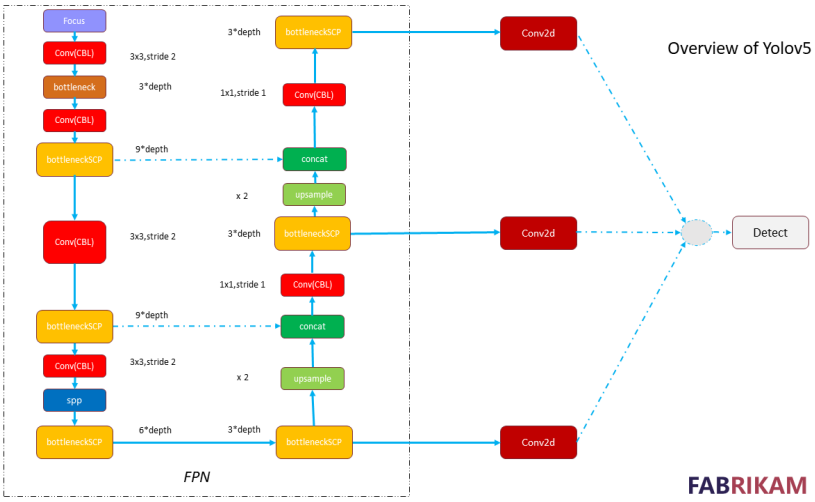
4. You Only Look Once method (YOLO)

With our approach with MaskRCNN being in trouble and YOLACT epochs taking several weeks to compute, we decided that our initial scope for the project would be to training our model based on YOLO network (Bochkovskiy et al. 2020; Jocher et al. 2020).

For that, the dataset images were converted to PNG format and the bounding boxes of the binary masks were computed and normalized to be fed into the network training process.

4.1 Implementation

Our vanilla YOLO implementation (Bochkovskiy et al. 2020; Jocher et al. 2020) used the different data augmentation techniques in order to make a better use out of the imputed images and run in complex scenarios. YOLO has the following architecture:



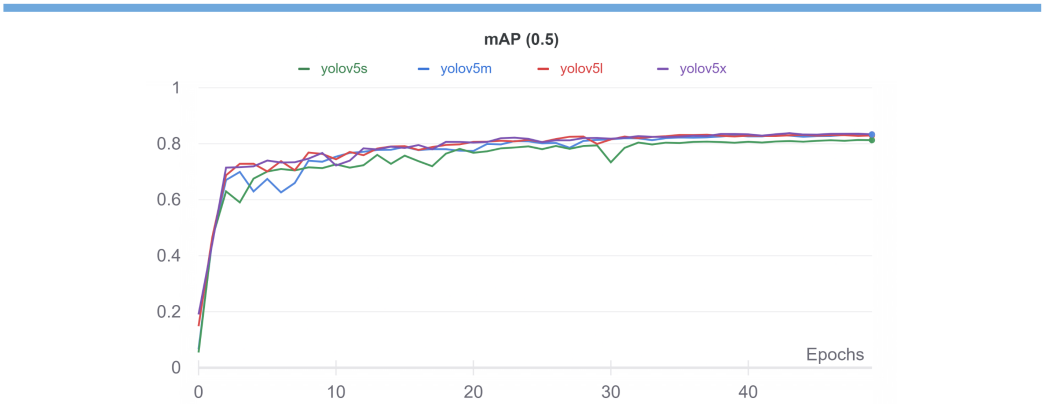
Tuning the following hyperparameters the models can perform some data augmentations such as HSV, translation, recaling, shearing, flipping, mosaic techniques. The following are the default hyperparameters of the network:

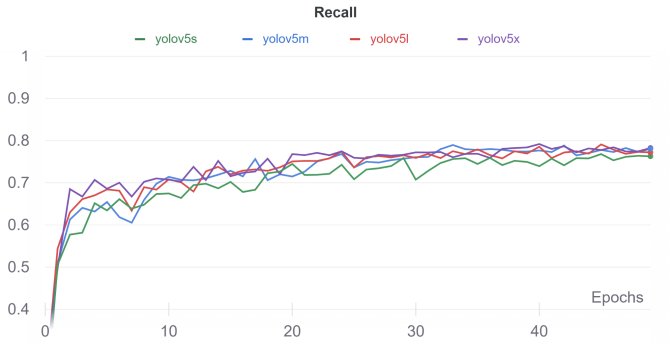
DEFAULT HYPERPARAMETERS

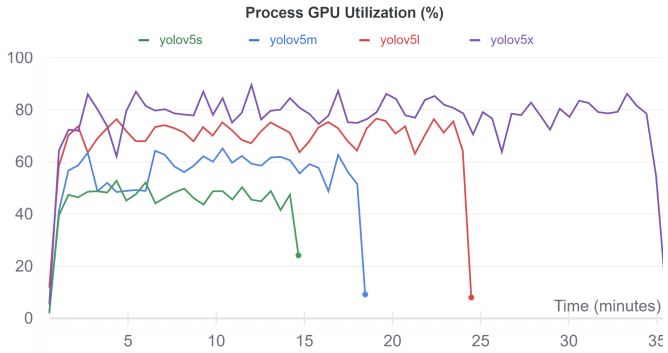
lr0	0.01	**lrf**	0.1	**momentum**	0.937
weight_decay	0.0005	**warmup_epochs**	3.0	**warmup_momentum**	0.8
warmup_bias_lr	0.1	**box**	0.05	**cls**	0.5
cls_pw:	1.0	**obj**	1.0	**obj_pw**	1.0
iou_t:	0.2	**anchor_t:**	4.0	**fl_gamma:**	0.0
hsv_h	0.015	**hsv_s**	0.7	**hsv_v**	0.4
degrees	0.0	**translate**	0.1	**scale**	0.5
shear	0.0	**perspective**	0.0		
flipud	0.0	**fliplr**	0.5		
mosaic	1.0	**mixup**	0.0	**copy_paste**	0.0

4.2 Results and analysis

The model has been tested with a random sample of 20% size (744 images) of the total dataset described above, the scores that we got were the following:



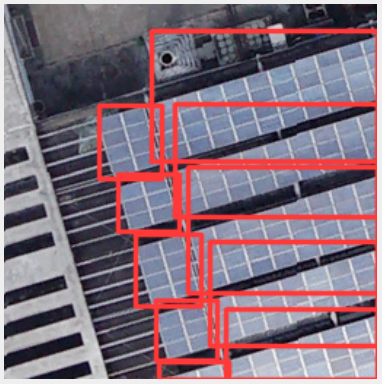


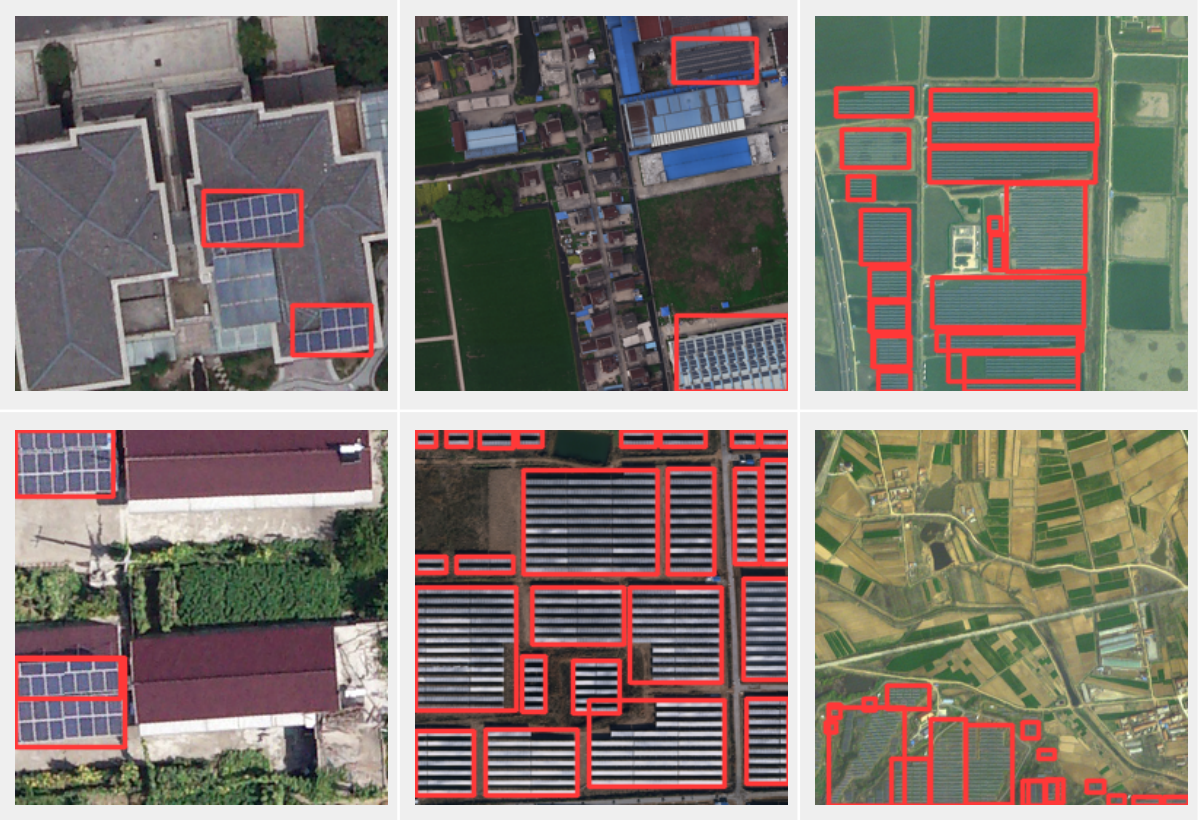


Batch size: 16	*Yolov5s*				*Yolov5m*			
EPOCHS	**Precision**	**Recall**	**mAP**	**F1 score**	**Precision**	**Recall**	**mAP**	**F1 score**
25	0.869	0.778	0.825	0.821	0.879	0.772	0.821	0.813
50	0.893	0.794	0.836	0.841	0.904	0.763	0.829	0.827
100	0.918	0.786	0.841	0.847	0.918	0.786	0.841	0.847

Batch size: 16	*Yolov5l*				*Yolov5x*			
EPOCHS	**Precision**	**Recall**	**mAP**	**F1 score**	**Precision**	**Recall**	**mAP**	**F1 score**
25	0.895	0.782	0.831	0.834	0.895	0.782	0.831	0.834
50	0.899	0.775	0.828	0.832	0.893	0.794	0.836	0.841
100	0.922	0.775	0.836	0.842	0.921	0.784	0.840	0.847

We can see that the results are pretty good for a first implementation but the images that the algorithm outputs look even more promising. Let's take as example the following images:

PV01	PV03	PV08
		



YOLO models have also been tested with Google Maps images, which contain less photovoltaic panels.

GOOGLE MAPS SATELLITE IMAGES

As we can see when evaluating the model with Google Maps images the network can still detect solar panels. However these detections are less accurate than the ones from the actual dataset.

4.3 Hyperparameter Evolution

The YOLO (Bochkovskiy et al. 2020; Jocher et al. 2020) library offers hyperparameter evolution. In every generation of evolution, it has 90% probability and 0.4 variance to mutate and create new offspring based on a combination of the best parents from all previous generations. We have performed it on the *Yolov5m* model to seek improvements from the baseline result.

Within 25 generations, the best result with mAP 0.828 was obtained. The improvement was not big because the number of generations we used, compared to the recommendation of 300 generations of evolution, was relatively small. However, the improvements demonstrated the usefulness and necessity of conducting this process.

Batch Size	**Precision**	**Recall**	**mAP 0.5**
Baseline	0.89336	0.76198	0.82432
Tuned	0.9055	0.7582	0.82818

Below we show some of the hyperparameters that have been tuned:

	Initial learning rate	**Momentum**	**Weight_decay**	**hsv-h**	**hsv-s**	**hsv-v**
For Baseline	0.01	0.937	0.0005	0.015	0.7	0.4
After Evolution	0.01162	0.9334	0.00035	0.01002	0.55308	0.31491

	Translate	**Scale**	**Mosaic**	**Mixup**	**Copy-paste**	**Anchors**
For Baseline	0.1	0.9	1	0.1	0.1	3
After Evolution	0.129	0.77614	0.82128	0.09367	0.0979	3.556

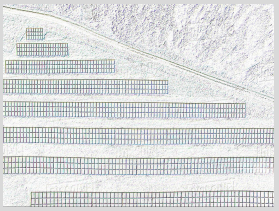
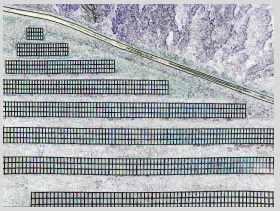
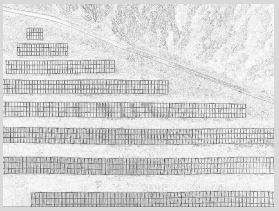
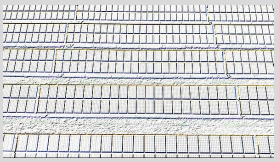
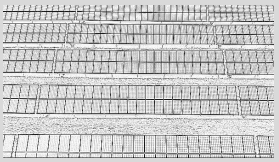
5. Future updates

5.1 Transformation to the input images

During all of our experiments our data has been a set of images with their respectives annotated labels that have been inputted to the model. Apart from the data augmentation techniques that are applied when loading the batches to train the models, no other modification has been done before extracting the features maps (in the case of the YOLO) with the convolutional layers.

This straightforward approach was the initial idea that the YOLO developers had. They did not want to use complicated manually programmed algorithms that extracted relevant features from the input images. They wanted the CNN to learn those relevant features by itself, but, although this philosophy has been very successful in recent years, we can still make use of this studied classical techniques to try to improve the performance of our model.

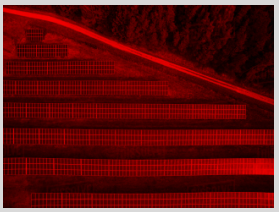
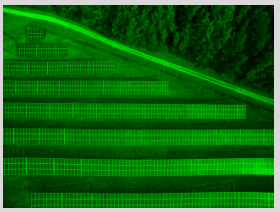
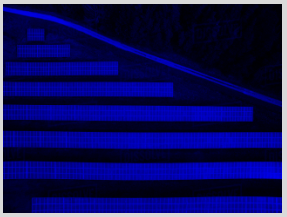
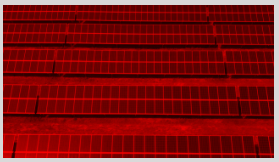
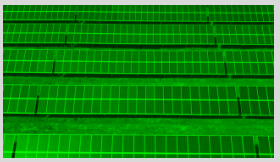
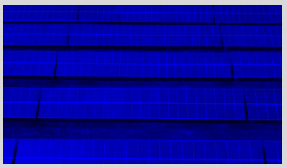
One of the most meaningful features is the mAP of the edges of the image. By applying a convolution with a specific filter over the image we can get another image (B/W) in which the pixel values indicate whether the pixels around are alike or not in the original image. That way, if they are alike it means that that pixel is not in the boundary of an object on the image, otherwise it will mean that it is. Different filters might yield different results.

Sample Images	Robert Cross Filter	Sobel Filter	Laplacian Filter
			
			

These filtered images could be used to substitute the original images and check if we can achieve good results. Another way to make use of these images is by adding this information to the RGB representation of the original images having as input an image composed of four 2D layers: red color, blue color, green color and the filtered image.

The addition of these filters to have more information about the images could improve the performance of our models, but it does not come without problems.

Currently we are taking advantage of pretrained models that we retrain over our training set to fine tune the parameters. But if we change the structure of the inputs of the models (by adding one more dimension to the RGB representation) we would have to change the initial layers of the model as the input would have a different dimension.

Sample Images	R	G	B
			
			

Previous input: R | G | B

Augmented input: R | G | B | Robert Cross Filter

This means either that we reinitialize the parameters of the initial layers or we create new parameters that immediately would change the output of the initial layers.

Something that we have to take into consideration if we want to do this is that there exists the possibility that the performance is not significantly improved. The first layers of most of the models that we are looking at are convolutional layers. Thus, we cannot confirm that the models did not already learned a similar filter that gives a boundary representation of the image in a feature map (in fact, most of the times when CNN are working with images, the feature maps of the first layers are geometrical representations of edges and boundaries of different parts or shapes of objects in image)

Another way of adding extra information to the images is by using spectral information of the satellite images of light frequencies that do not lie in the visible spectrum.

Thinking about the specific nature of our task, it might be a good idea to try to use the spectral information of the images to find solar panels.

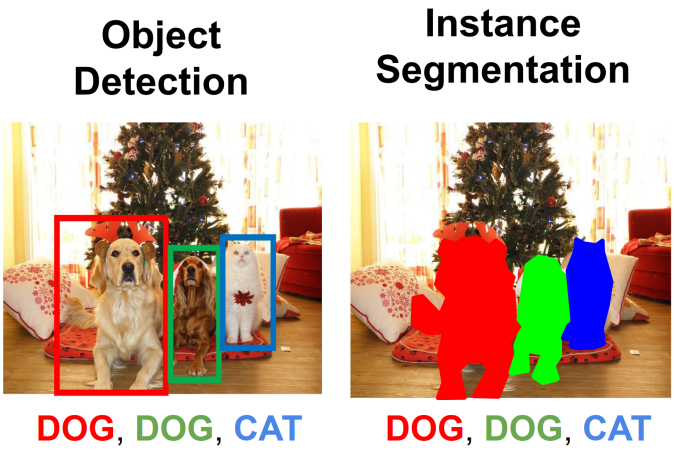
Solar panels absorb light of specific wave-lengths leaving a very distinctive mark on the frequency spectrum that the satellites capture (Czirjak 2017).

However, we will still encounter the following problems if we go through this path.

- As previously commented, we do not know how to process this data to make use of it and input it on a model.
- Similarly as with the edge detection filters, we would have to retrain the models from zero, and modify them to accept this extra information as input alongside with the images.
- The Copernicus - Mundi datasets do not have labeled examples of solar panels, so we would have to manually create this dataset (maybe partially automate the task by using our current model with the rgb part of the images)

5.2 Instance segmentation

The YOLO model that we run performs object detection over an image by outputting a bounding box where the solar panel is located. This approach has worked well but we could also tackle the problem by applying instance segmentation, predicting a mask with the silhouette of the target instead of a bounding box.



Stanford CS231n | Lecture 15: Detection and Segmentation (F.-F. Li et al. 2021)

As our main dataset is made of binary masks, utilizing networks for instance segmentation can increase both the accuracy of the box and the detection of solar panels. Mask-RCNN is an architecture that allows us to do this.

5.3 Other models

Up to this point we have successfully trained a YOLOv5 model, but there are other models that could improve our performance.

We have already seen YOLACT and Mask RCNN are interesting alternatives that we could try but there are a lot of other architectures that could achieve a good performance like PANet (S. Liu et al. 2018), SSD (W. Liu et al. 2016) or CornerNet (Law and Deng 2019).

The original creators of YOLO did not build YOLOv5 unlike YOLOv4. It seems that on some occasions the version 4 outperforms version 5 (Jocher et al. 2020; Nelson et al. 2020; PhD 2020). Trying YOLOv4 might be a way to improve our performance.

The problem with testing each of the models is that their implementations are quite different and require different annotations formats. Three frameworks have been studied that let us experiment with different network backbones having to restructure the main pipeline of the code every time a new model has to be tested.

These are some interesting frameworks for object detection.

- **MMDetection** (Chen et al. 2019): MMDetection, an object detection toolbox that contains a rich set of object detection and instance segmentation methods as well as related components and modules.

- **Detectron2** (Wu et al. 2019): is Facebook AI Research's software system that implements state-of-the-art object detection algorithms.

- **TensorFlow Object Detection API** (Huang et al. 2017): is a framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models.

All three of these frameworks are open source and have a broad documentation. MMDetection and Detectron2 are built on top of PyTorch while the TensorFlow Object Detection API is built on top of TensorFlow.

6. Conclusion and final thoughts

From the very first moment we spent thinking about this project, our main goal was to develop a model that would prove to be useful either to us, the company that we are working with or the general population of our world. We acknowledge that we are still pretty far from the optimal solution that we seek to achieve but still know that big improvements have been made in order to get us closer to solving the problem that we want to tackle.

This data science project is being proven to be helpful in developing real world skills that will one day help us solve even bigger tasks and we as a team are really happy that this opportunity has been given to us so we can show what we are worth.

Even though we are happy with the obtained results so far, progress will still be made and we hope that anyone that reads this first report stays tuned in order to find what we are capable of achieving with persistence, hard work and great ideas in mind.

References

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.

Bolya, D., Zhou, C., Xiao, F., & Lee, Y. J. (2019). YOLACT: Real-time Instance Segmentation. In *ICCV*.

Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., et al. (2019). MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155*.

Czirjak, D. W. (2017). Detecting photovoltaic solar panels using hyperspectral imagery and estimating solar power production. *Journal of Applied Remote Sensing*, *11*(2), 1–25. https://doi.org/10.1117/1.JRS.11.026007

Ding, P., Zhang, Y., Deng, W.-J., Jia, P., & Kuijper, A. (2018). A light and faster regional convolutional neural network for object detection in optical remote sensing images. *ISPRS Journal of Photogrammetry and Remote Sensing*, *141*, 208–218. https://doi.org/10.1016/j.isprsjprs.2018.05.005

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 2980–2988). https://doi.org/10.1109/ICCV.2017.322

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv:1611.10012 [cs]*. http://arxiv.org/abs/1611.10012. Accessed 16 November 2021

Jiang, H., Yao, L., Lu, N., Qin, J., Liu, T., Liu, Y., & Zhou, C. (2021). Multi-resolution dataset for photovoltaic panel segmentation from satellite and aerial imagery. *Earth System Science Data Discussions*, *2021*, 1–17. https://doi.org/10.5194/essd-2021-270

Jocher, G., Nishimura, K., Mineeva, T., & Vilariño, R. (2020). Yolov5. *Code repository https://github. com/ultralytics/yolov5*.

Law, H., & Deng, J. (2019). CornerNet: Detecting Objects as Paired Keypoints. *International Journal of Computer Vision*, *128*(3), 642–656. https://doi.org/10.1007/s11263-019-01204-1

Li, F.-F., Krishna, R., & Xu, D. (2021). Lecture 15: Detection and Segmentation, 114.

Li, K., Wan, G., Cheng, G., Meng, L., & Han, J. (2020). Object detection in optical remote sensing images: A survey and a new benchmark. *ISPRS Journal of Photogrammetry and Remote Sensing*, *159*, 296–307. https://doi.org/10.1016/j.isprsjprs.2019.11.023

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path Aggregation Network for Instance Segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016* (pp. 21–37). Cham: Springer International Publishing.

Malof, J. M., Bradbury, K., Collins, L. M., & Newell, R. G. (2016). Automatic detection of solar photovoltaic arrays in high resolution aerial imagery. *Applied Energy*, *183*, 229–240. https://doi.org/10.1016/j.apenergy.2016.08.191

Nelson, J., JUN 12, J. S., & Read, 2020 16 Min. (2020, June 12). Responding to the Controversy about YOLOv5. *Roboflow Blog*. https://blog.roboflow.com/yolov4-versus-yolov5/. Accessed 16 November 2021

PhD, I. I. (2020, June 30). YOLOv4 vs YOLOv5. *Deelvin Machine Learning*. https://medium.com/deelvin-machine-learning/yolov4-vs-yolov5-db1e0ac7962b. Accessed 16 November 2021

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019). Detectron2. https://github.com/facebookresearch/detectron2