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

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:

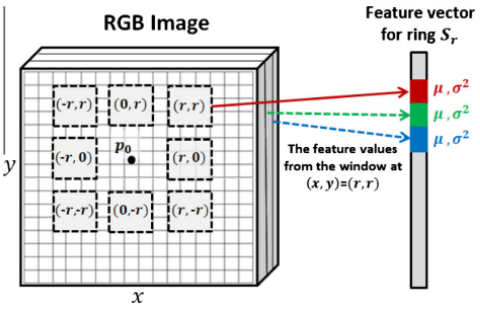
Several papers have been reviewed and their respective methods to detect solar panels have been studied. The project is based on two papers which give an insight of the latest techniques and benchmarks in the industry.

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

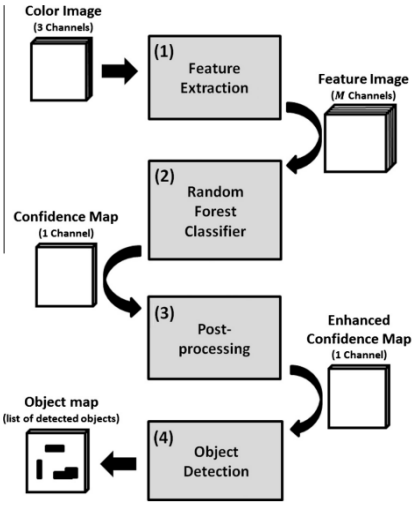
For the first paper, a slightly older approach is reviewed, that uses the classical feature extraction to process the images and extract the patterns and a Random Forest to predict the confidence levels. A team of human annotators is in charge of manually annotated PV arrays. The data used comes from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

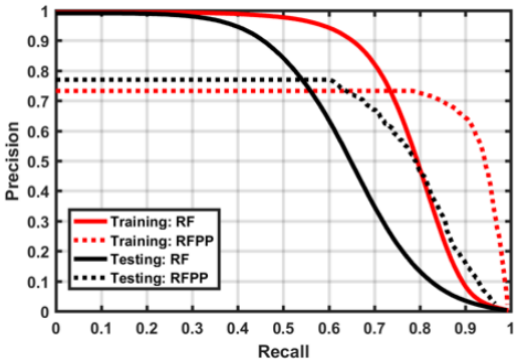


In order to link the automated detections to the human annotations, the Jaccard index is used. Depending on the goal of the problem, we would need either a really high value (almost perfect match) or just a value > 0 (it knows where the truth object is at, but can not detect it properly).

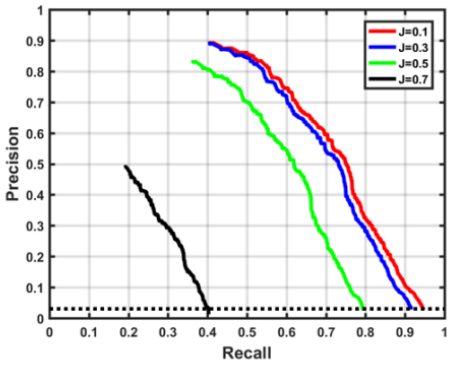


For the pixel-based performance, i.e. detecting if a pixel belongs to a solar panel array, the authors tried using only RF with and without post processing in both training and testing datasets.

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.

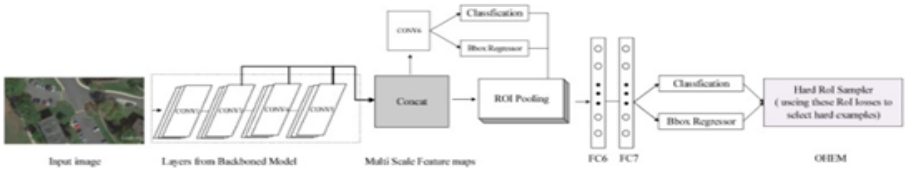
1.2 A light and faster regional convolutional neural network for object detection in optical remote sensing images (Ding et al. 2018)

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.

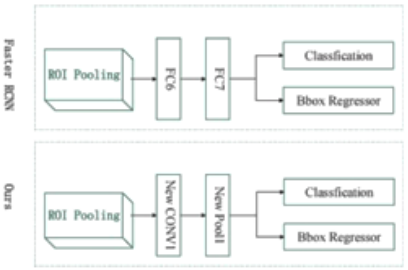


Download : Download high-res image (196KB) Download : Download full-size image

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

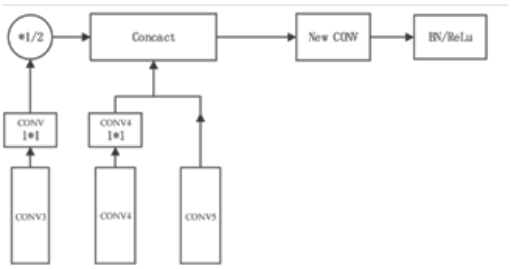
The structure is shown below:

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.

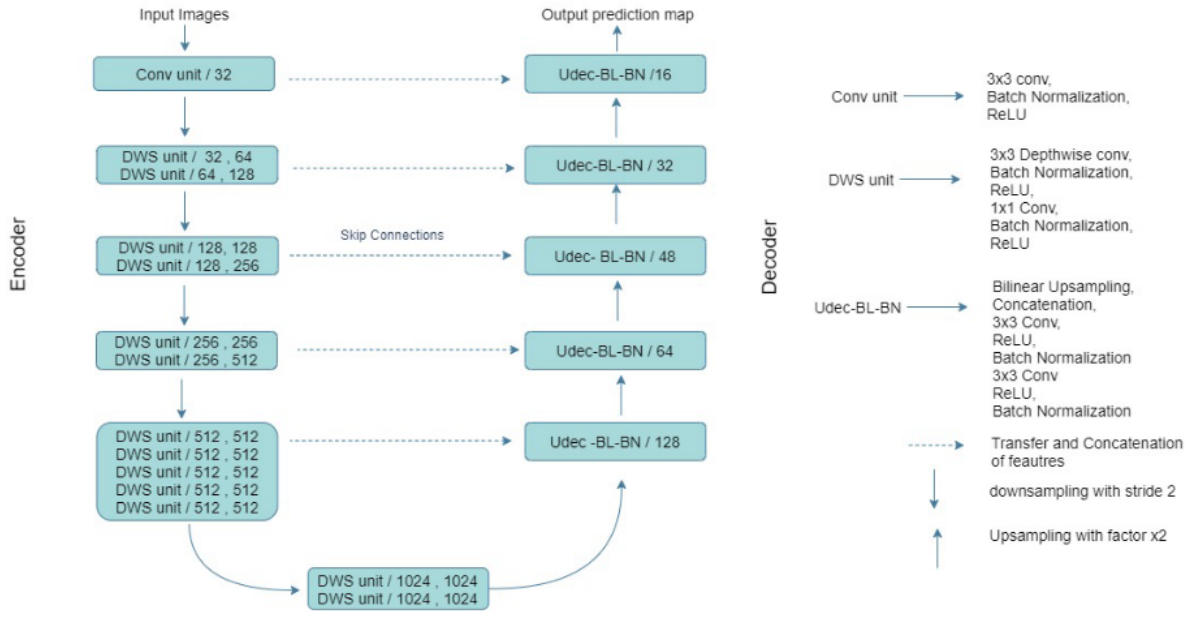
1.3 Segmentation of satellite images of solar panels using fast deep learning model. (Arif Wani et al. 2020)

This paper introduces a new approach to the problem that we want to solve in our research. It also considers instance segmentation the first step in order to provide a reliable knowledge on the production of energy that a certain area may have.

We find this last paper to be of great importance because of the amount of different architectures that it implements in order to further understand the results obtained with their proposed approach.

Procedures:

1. Develops a deep learning segmentation architecture based on Mobilenet. This will mean that the architecture will have an encoder and decoder method with the following structure:



2. The study implements different models with various encoders in order to find the most reliable method while also checking how time efficient they are. Some models that are checked are **Unet, Segnet, DilatedNet, PspNet, DeepLab v3+, Dilated Resnet** and the chosen encoders are different variations of **VGGNet and Resnet.** The proposed approach consists of Unet-Mobilenet + Mobilenet as the encoder.

3. Takes as input 601 high quality RGB images of 5000x5000 that are segmented to 224x224 in order to better capture the details in images. At the same time obtain their binary mask in order to use it as input too.

4. One of the key features of this paper is the implementation of a dice loss layer. This layer will approach the bounding boxes with a different approach to more naive methods like cross entropy loss. Because of the great importance of this layer, we will further explain this feature after the evaluation step.

Evaluation

The precision and recall varies highly depending on the implementation that is used but is greatly summarized in the chart below:

Model		Encoder	Recall	Precision
Unet		Vggnet 16	0.8695	0.8773
Segnet		Vggnet 16	0.816	0.8252
DilatedNet		Vggnet 16	0.6427	0.6813
PspNet		Resnet 50	0.7002	0.6113
DeepLab v3+		Resnet 50	0.7536	0.7319
Dilated Resnet		Resnet 18	0.6615	0.6557
Our other Implementations	Unet-Vggnet-BN	Vggnet 16 (with batch normalization)	**0.9266**	0.9227
	Unet-Vggnet-DWS	Vggnet 16 (with DWS convolution)	0.9115	0.9045
	Unet_Resnet	Resnet Blocks	0.9078	0.9198
	Fully Unet-Resnet	Resnet Blocks	0.8994	0.9421
Proposed	Unet-Mobilenet	Encoder (Mobilenet)	0.8498	**0.9595**

Implementation and usefulness of dice loss

As explained before, the dice loss approach is being implemented in this paper. Its usefulness goes far beyond some of the reasons that they mention in the paper and we'll try to explain why they chose it. The formula that defines the Dice coefficient is the following:

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

In this formula, Pi and Gi represent the pairs of respective values of pixels in prediction and ground truth respectively. In a boundary detection scenario, their values are either 0 or 1 and represent if the pixel is a boundary (1) or not (0). As a result of this, the formula represents the sum of correctly classified bounding pixels divided by sum of total boundary pixels from the prediction and ground truth.

As a result of this, when comparing this approach to some more usual approaches, we see that our method is better at considering closeness to adjacent pixels. Dice loss will also do a better job at working with unbalanced datasets where not so many boundary pixels are found as is the case with our problem regarding detection of solar panels.

1.4 Paper data comparison

Paper number	Type of approach	Number of images in train/test	Type of Problem	Techniques	Evaluation Criteria	Performance
1	Classical approach	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	Object detection	1000 images with about 7000 aircrafts 500 images with about 7000 cars	Supervised pre-trained model + Domain-specific	Faster R-CNN + OHEM + Multi-scale	Average Precision(AP) + Recall Score	Aircraft dataset AP: 0.907 Recall: 0.9685 Car dataset

			fine-tuning	Representation		AP: 0.879 Recall: 0.8846
3	Instance segmentation	601 images of size 5000x5000 in subsets of 224x224	Supervised learning with masks	Unet-Mobilenet + Dice loss	Recall + Precision + Time	Solar panel dataset: Precision: 0.8498 Recall: 0.9595

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.

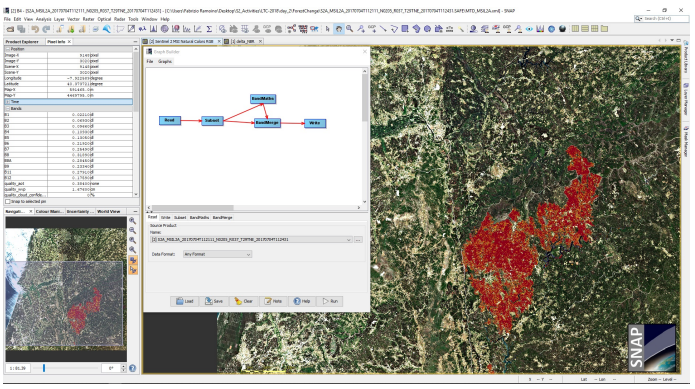
The second consideration that we had to do, regarded the kind of approach that our method would implement and since we know that a CNN would be our best bet, we decided to implement **Object detection** and **image segmentation** models in order to compare the both approaches.

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



The major issue found when using this data source was the lack of reference samples of solar panels in **geospatial data**. In order to train a model to recognize objects, several examples are needed to characterise common features in the model and ensure a correct detection.

Additionally, the **resolution** of the processed data was not sufficiently large to ensure a correct detection in this type 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.

2.2 Solar Panel Labeled Dataset

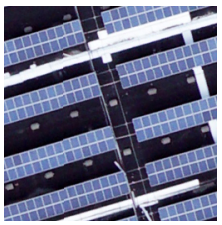
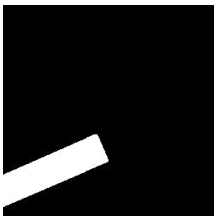
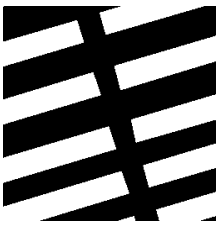
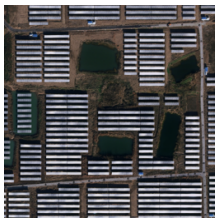
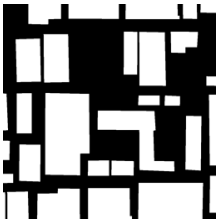
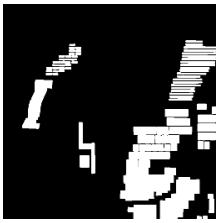
In order to train a model to detect panels a relatively **large sample of images** is needed. Nevertheless, there is a lack of datasets that contain images from solar panels taken from an aerial point of view. However, in August 2021, the School of Engineering of China released one of the most complete datasets of satellite images (7GB) containing photovoltaic cells (*Multi-resolution dataset for photovoltaic panel segmentation from satellite and aerial imagery,* 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

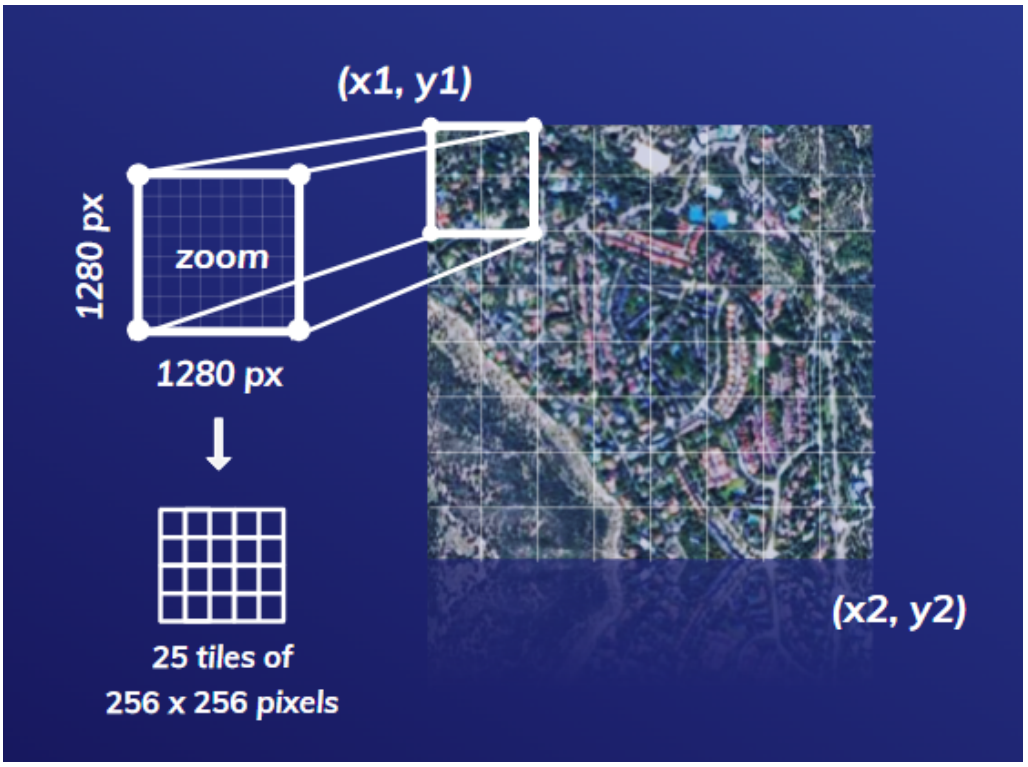
One of the main advantages of the dataset with respect to the raw satellite data is the presence of **binary masks** representing each solar panel as well as the format of the images (bmp) which is more suitable for training a model.

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.

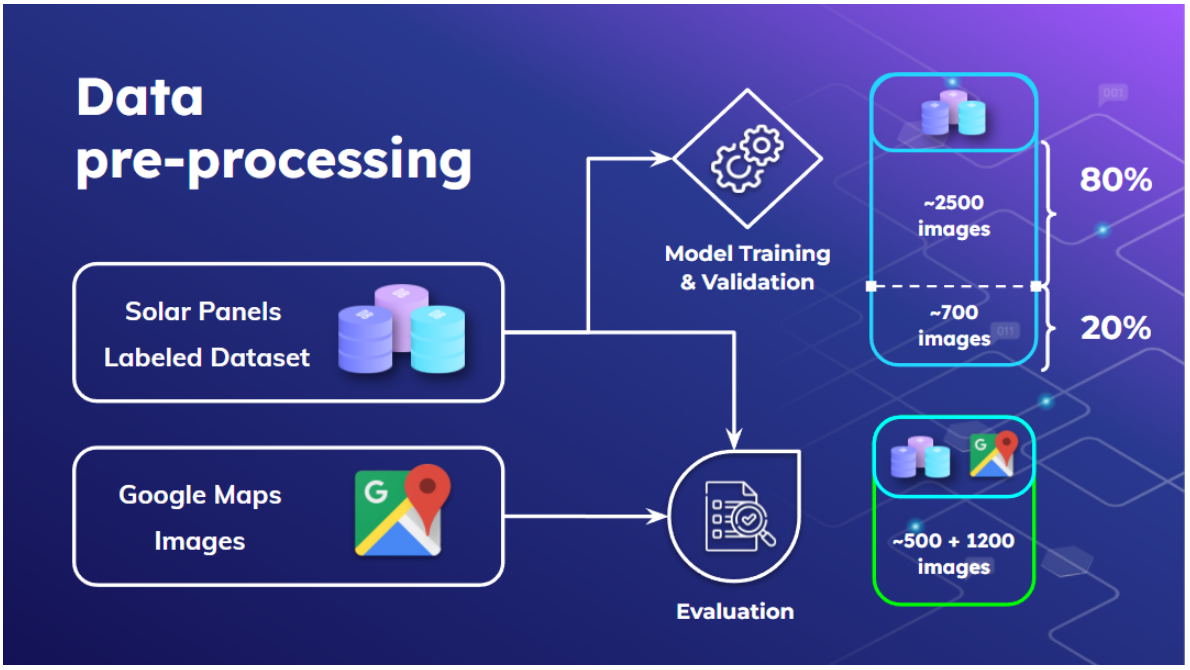




2.4 Data pre-processing

2.4.1 Datasets

For the training and validation phase of the models, the labeled solar panel dataset has been used (80% training, 20% validation).



For the evaluation of phase, two datasets are defined. The first one is a partition of the labeled dataset and the second one contains images from Google Maps.

2.4.2 Data Augmentation

To better train the models using the dataset that is relatively small, we applied data augmentation techniques from the library Albumetation.

The augmentations used include:

- Horizontal flip
- Shift Scale Rotate
- CLAHE
- Padding
- Brightness / contrast / colors manipulations
- Image blurring and Sharpening
- Gaussian noise
- Random crops
- Data normalization with respect to different backbones

4. Evaluation of different CNN architectures

Our initial approach was based on comparing 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. Object detection vs image segmentation

Let us comment on some advantages and disadvantages of both approaches to explain the reason for our decision and understand both techniques.

Object detection is the task of finding and classifying objects in an image.

Image segmentation is the task of dividing an image into multiple regions, or segments.

Some advantages of CNN object detection and CNN image segmentation include:

1. They are able to learn representations of objects that are more discriminative and invariant to transformations than hand-crafted features.
2. They are able to segment/detect the targeted objects in images with high accuracy.

However, there are also some disadvantages of using CNN object detection and CNN image segmentation architectures. These include:

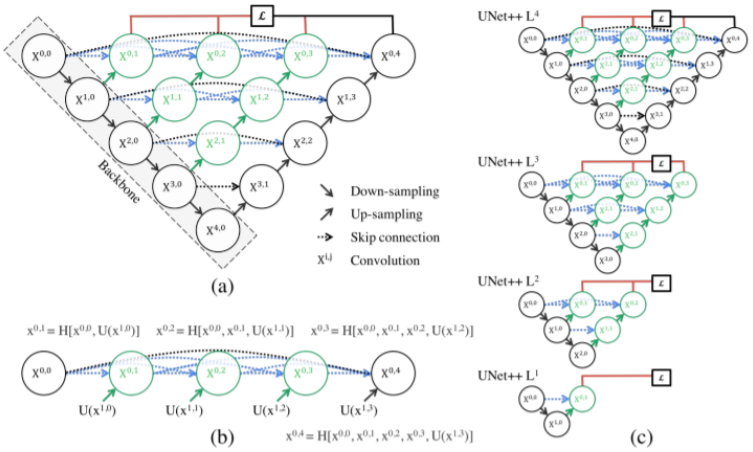
1. They are often slow to train and require a lot of computational resources.
2. They can be difficult to adapt to new tasks or object classes.
3. They are not always accurate when detecting objects in challenging images, such as those with complex backgrounds or cluttered scenes.

5. Instance Segmentation Models

5.1 Implementation and tested architectures

UNET++

Unet++ is a fully convolutional neural network for image semantic segmentation. It consists of the encoder and decoder, which are connected through a series of nested dense convolutional blocks with skip connections. The encoder extracts features of different spatial resolution (skip

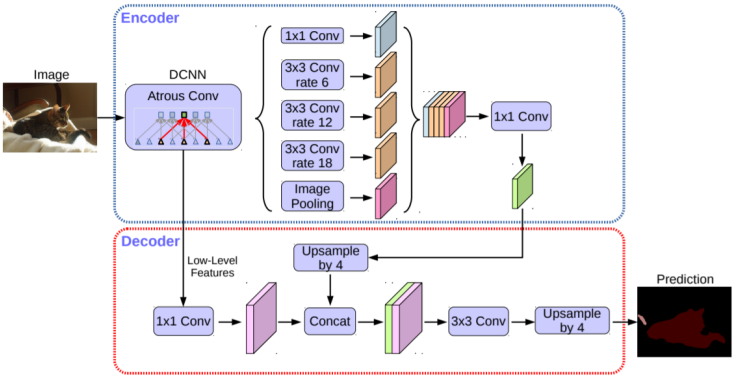


connections) which are used by the decoder to define an accurate segmentation mask.

The main idea behind UNet++ is to bridge the semantic gap between the feature maps of the encoder and decoder prior to fusion. Decoder of Unet++ is more complex than in usual Unet.

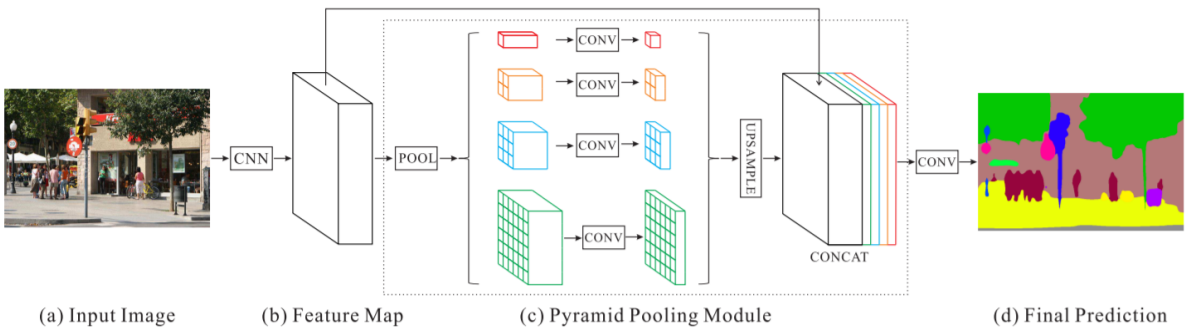
DEEPLABV3+

"DeepLabv3+" employs the encoder-decoder structure where DeepLabv3 is used to encode the rich contextual information and a simple yet effective decoder module is adopted to recover the object boundaries. The atrous convolution can also be applied to extract the encoder features at an arbitrary resolution, depending on the available computation resources.



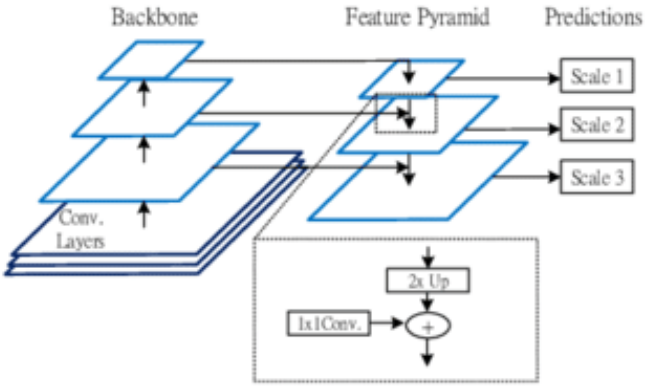
PSPNet

PSPNet is a fully convolutional neural network for image semantic segmentation. Consist of encoder and Spatial Pyramid (decoder). Spatial Pyramid is built on top of the encoder and does not use "fine-features" (features of high spatial resolution). PSPNet can be used for multiclass segmentation of high resolution images, however it is not good for detecting small objects and producing accurate, pixel-level mask.



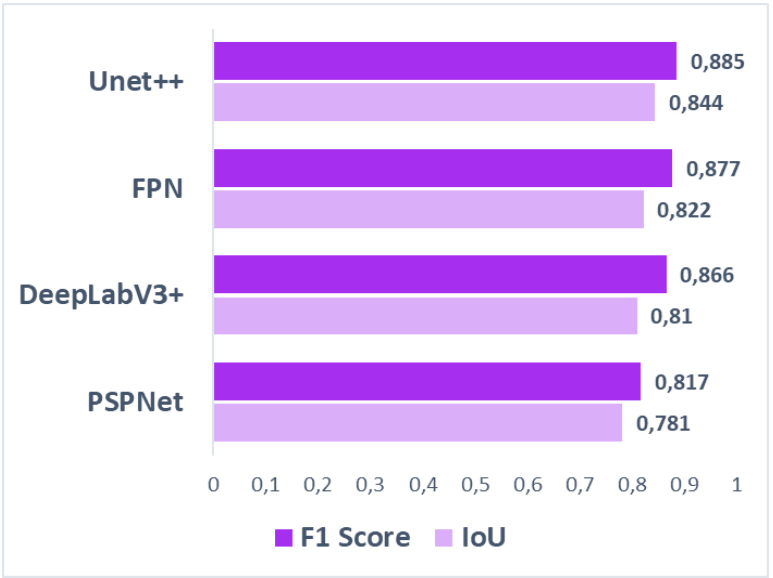
FPN

A feature pyramid network (FPN) is a type of deep learning network that is used to learn and identify features in data. It is a pyramid-shaped network that contains multiple layers of nodes, with each layer containing more nodes than the one below it. The nodes in the lower layers are used to identify basic features in the data, and the nodes in the upper layers are used to identify more complex features. This type of network is used to improve the accuracy of deep learning networks.



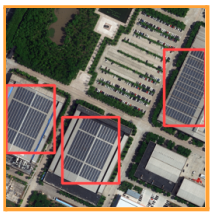
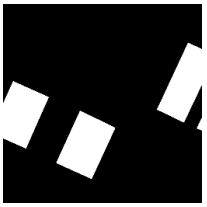
5.2 Result Comparison between Different Architectures

We trained the four models on the same dataset. The plot below shows the F1 score and IoU score of each model respectively. It also ranks the models by the scores in descending



order. Unet++ performed the best and reached a F1 score of 0.885.

Using these models, we predicted solar panels on some samples from the test dataset. Bounding boxes were used to locate the solar panels. A binary segmentation mask was generated for each sample, displaying the corresponding area.

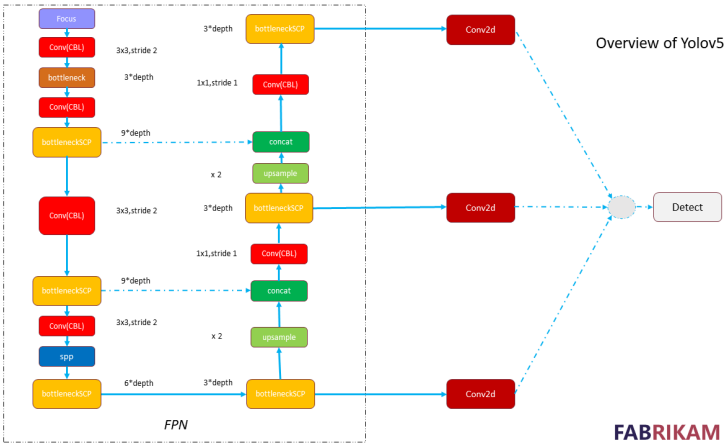
PV01			
PV03			
PV08			

6. You Only Look Once method (YOLO) – Object detection

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.

6.1 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:



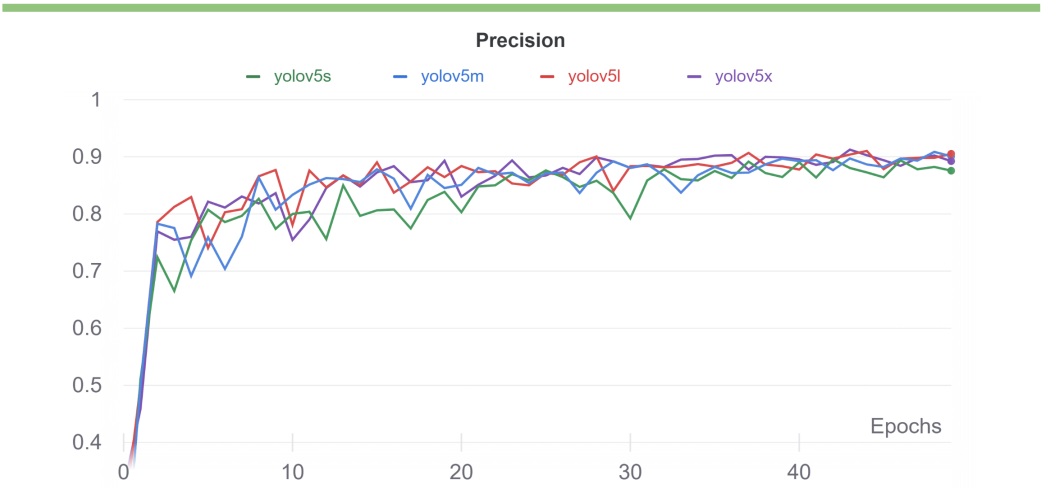
Tuning the following hyperparameters the models can perform some data augmentations such as HSV, translation, recaling, shearing, flipping, mosaic techniques. The following are the default hyperparameters of the network:

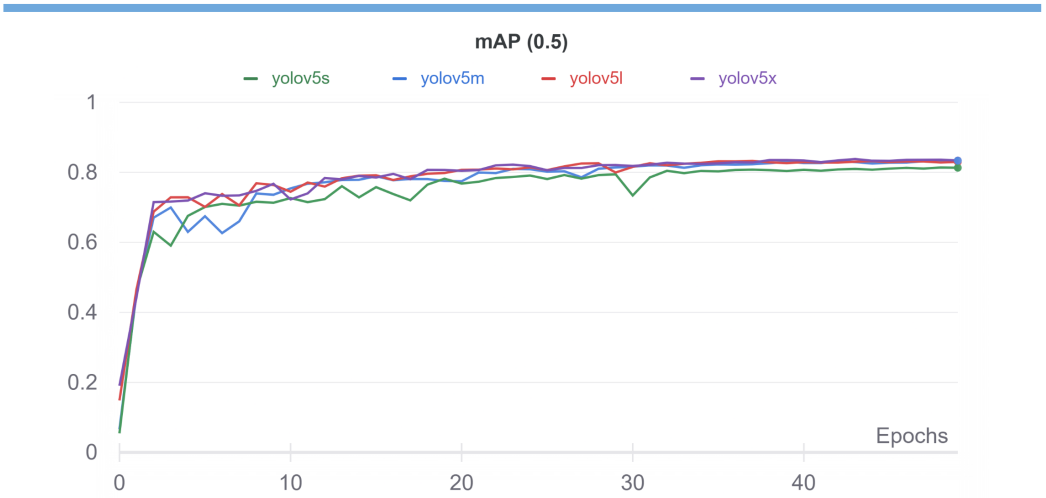
DEFAULT HYPERPARAMETERS

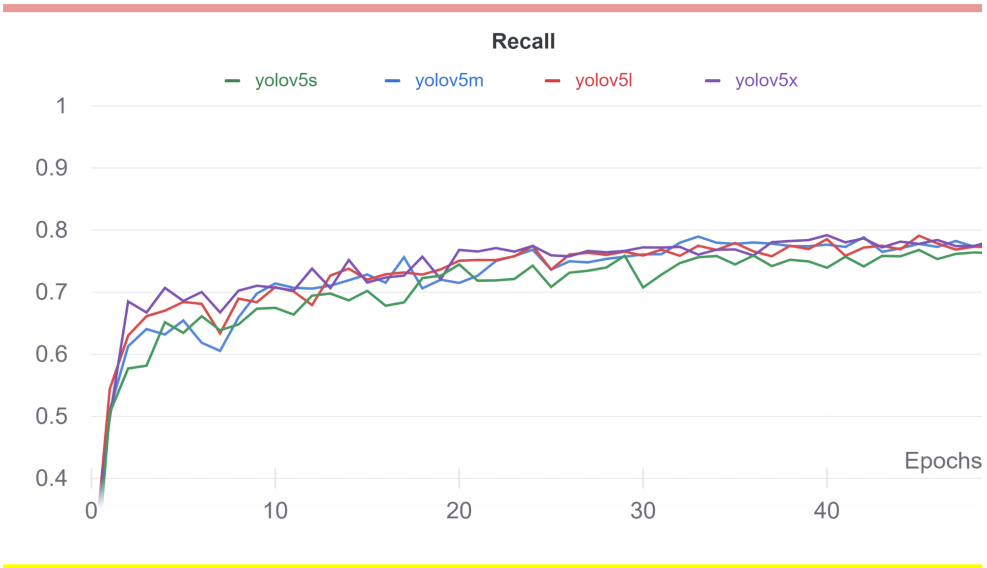
lr0	0.01	**lrf**	0.1	**momentum**	0.937
weight_decay	0.0005	**warmup_epochs**	3.0	**warmup_momentum**	0.8
warmup_bias_lr	0.1	**box**	0.05	**cls**	0.5
cls_pw:	1.0	**obj**	1.0	**obj_pw**	1.0
iou_t:	0.2	**anchor_t:**	4.0	**fl_gamma:**	0.0
hsv_h	0.015	**hsv_s**	0.7	**hsv_v**	0.4
degrees	0.0	**translate**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

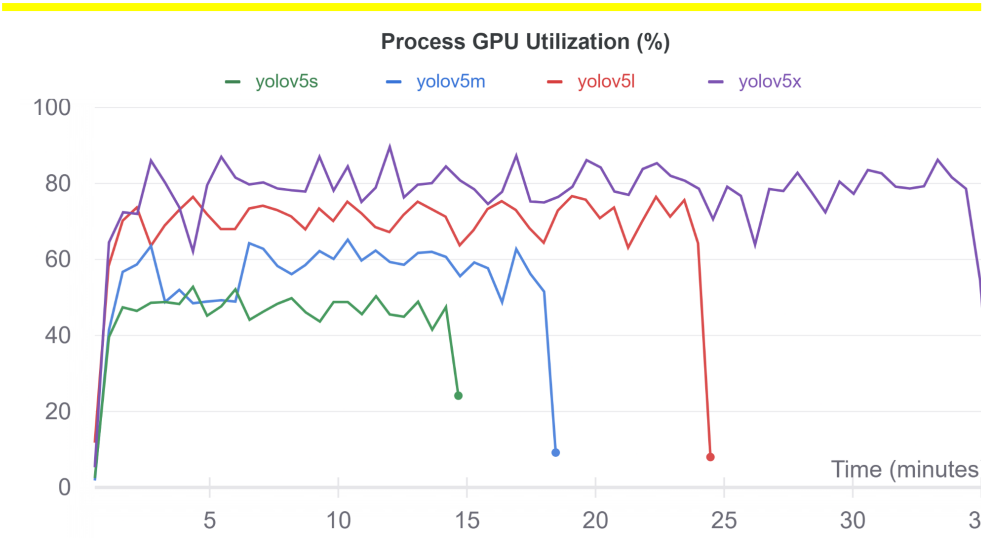
6.2 Results and analysis

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:





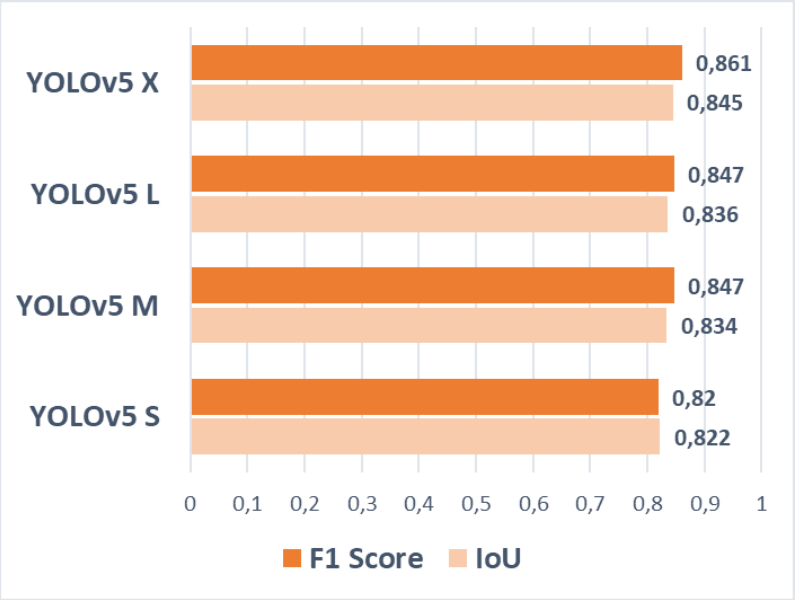




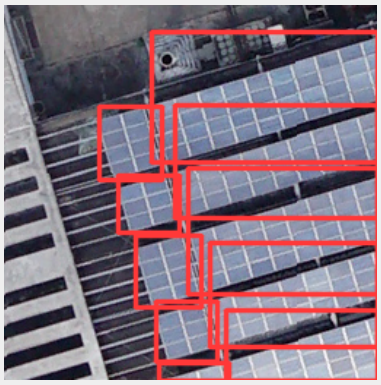
Batch size: 16	*Yolov5s*				*Yolov5m*			
EPOCHS	**Precision**	**Recall**	**mAP**	**F1 score**	**Precision**	**Recall**	**mAP**	**F1 score**
25	0.869	0.778	0.825	0.821	0.879	0.772	0.821	0.813
50	0.893	0.794	0.836	0.841	0.904	0.763	0.829	0.827
100	0.918	0.786	0.841	0.847	0.918	0.786	0.841	0.847

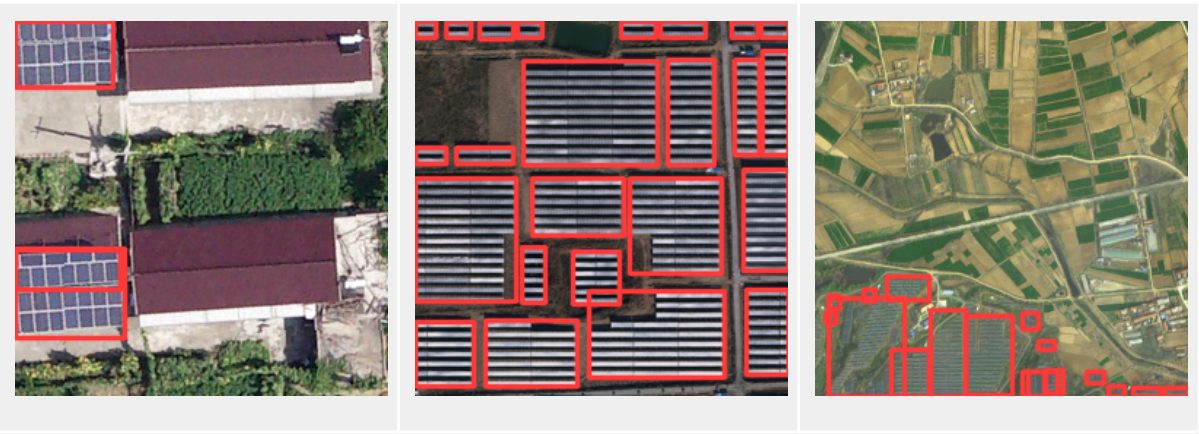
Batch size: 16	*Yolov5l*				*Yolov5x*			
EPOCHS	**Precision**	**Recall**	**mAP**	**F1 score**	**Precision**	**Recall**	**mAP**	**F1 score**
25	0.895	0.782	0.831	0.834	0.895	0.782	0.831	0.834
50	0.899	0.775	0.828	0.832	0.893	0.794	0.836	0.841
100	0.922	0.775	0.836	0.842	0.921	0.784	0.840	0.847

With a plot summarizing the scores from the tables above, it shows that the YOLOv5X performed the best among the 4 models, with a F1 score of 0.861.



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 used to be less accurate than the ones from the actual dataset.

6.3 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 *YOLOv5 M* 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:

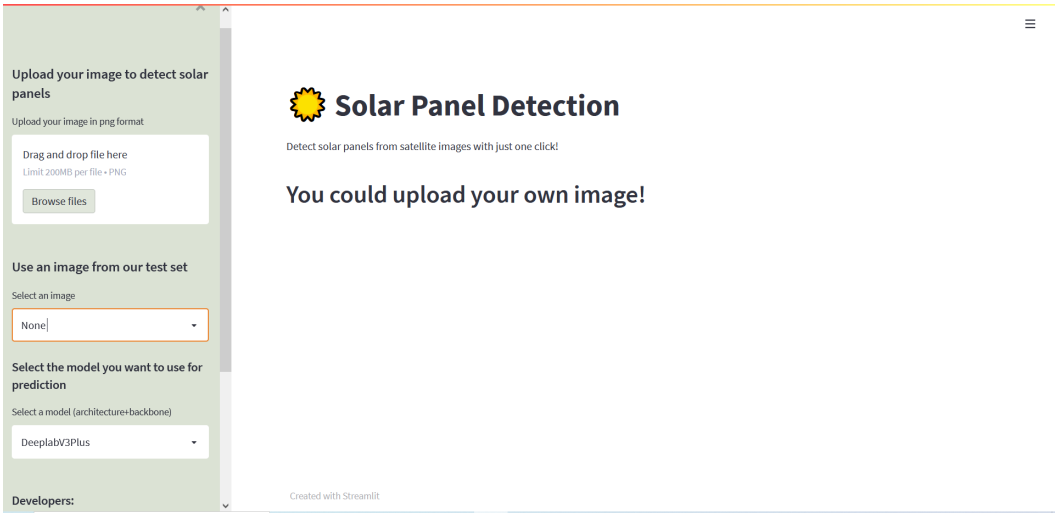
	Initial learning rate	Momentum	Weight_decay	hsv-h	hsv-s	hsv-v
For Baseline	0.01	0.937	0.0005	0.015	0.7	0.4
After Evolution	0.01162	0.9334	0.00035	0.01002	0.55308	0.31491

	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

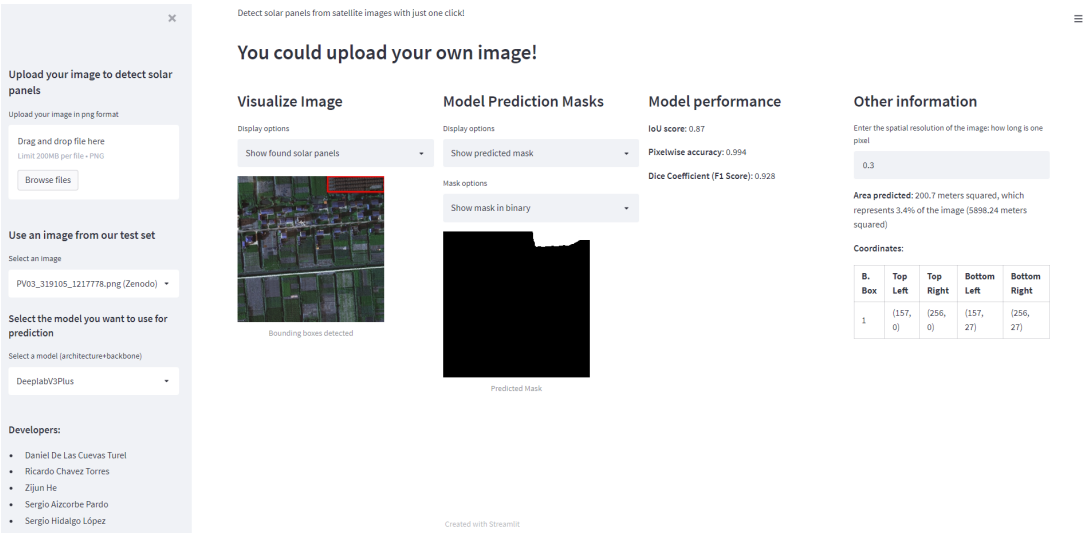
7. Streamlit development

In order to make the most out of our research, we decided to deploy our successful models with Streamlit to showcase our results. In this app, users will be able to detect solar panels from the image they upload or select, compare the performance of different models and obtain information about area and coordinates.

Something important to note is that the system works with **256x256** images and resizes them every time to that resolution. In any case, we recommend uploading images with that resolution to avoid possible edge problems or errors.

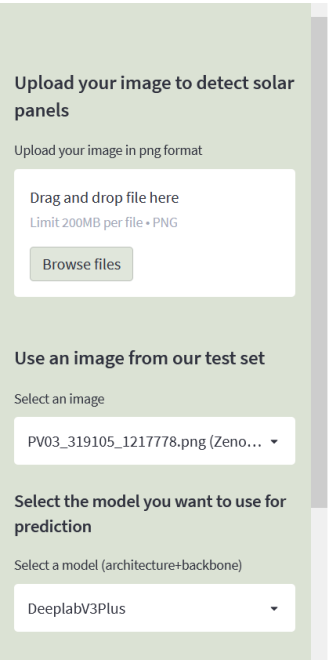


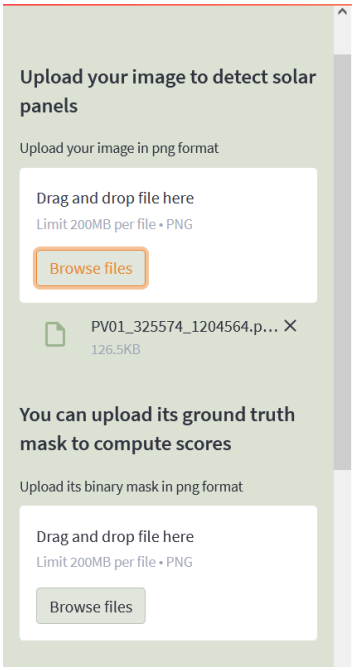
We will explain how this dashboard works illustrated with an image from the Zenodo dataset. After selecting the image and the desired model, a page is shown like the one below:



7.1 Sidebar utilities

In this part, users can either upload their image or choose one image provided by us. Four different models are provided by us so the user can choose the one that best suits their needs.





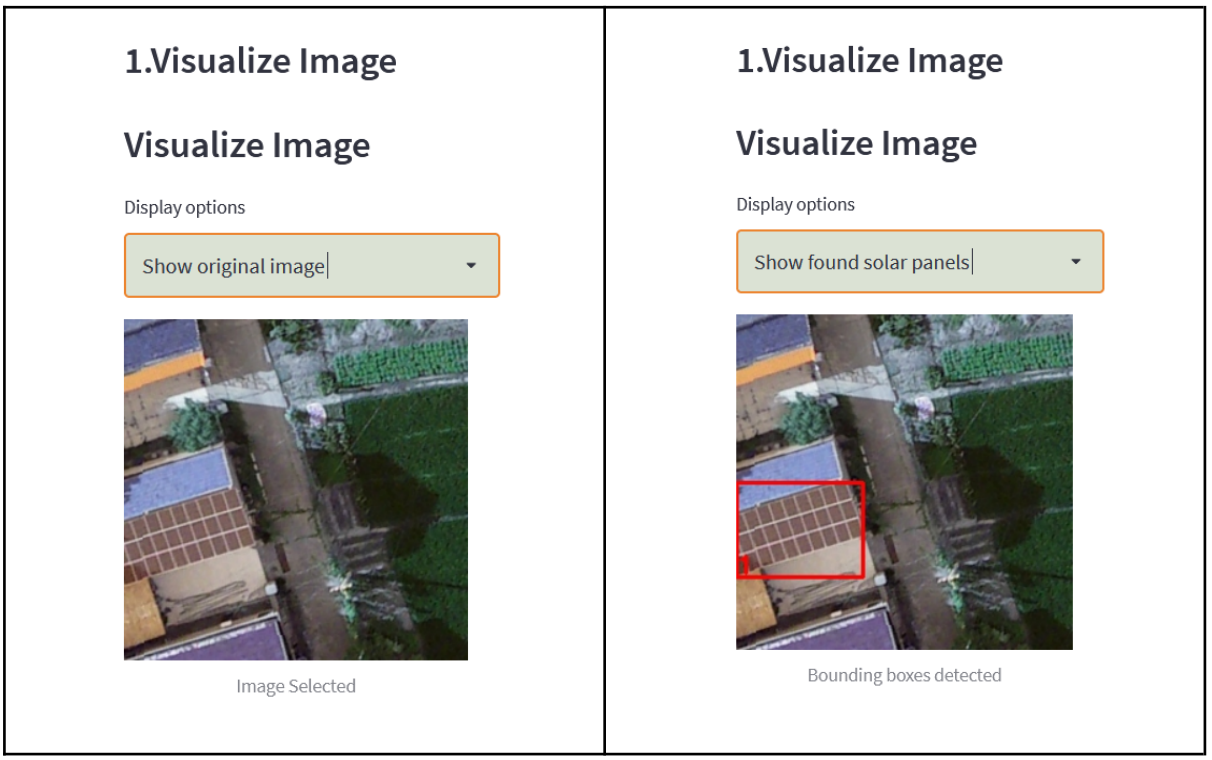
The ground truth mask of the image can also be provided if users choose to use their own images, in order to see the model performance.

7.2 Main view

In this region of our website, we have separated the space into 4 columns. They are the following:

First column

In the first column, users can choose to show the original uploaded image or selected image or the same image with bounding boxes locating solar panels on it. These are just two examples of the options that we show on our website.



Second column

In the second column, users can decide whether to display the predicted mask, the ground truth mask, if there is any, or both. As an example, the four images below show different combinations of the parameters “Display options” and “Mask options” where the first row shows the results with a predicted mask vs the ground truth mask in the second row. In comparison, the first row will change how the mask is outputted (either in binary or with its original colors).

Model Prediction Masks

Display options

Show predicted mask

Mask options

Show mask in binary

Predicted Mask

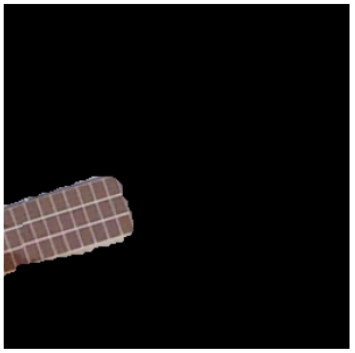
Model Prediction Masks

Display options

Show predicted mask

Mask options

Show mask applied to origi…



Detection applied to image

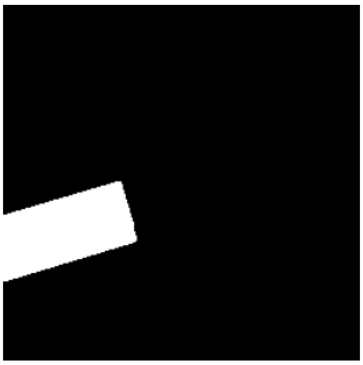
Model Prediction Masks

Display options

Show ground truth mask

Mask options

Show mask in binary



Ground Truth Mask

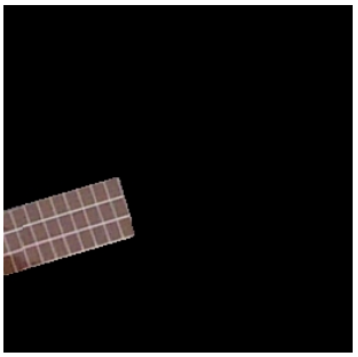
Model Prediction Masks

Display options

Show ground truth mask

Mask options

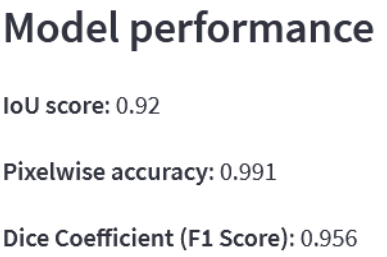
Show mask applied to origi…

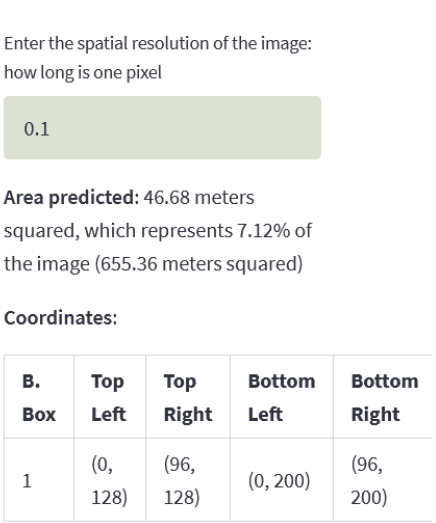


Ground Truth Mask applied

Third column

This column shows the performance of the selected model on the image. In order to do the closest to optimal performance comparison of the model to the real image, the chosen metrics are Intersection over Union (IoU) score, pixel-wise accuracy and F1 score. This will show it this third column of our Streamlit main view.





B. Box	Top Left	Top Right	Bottom Left	Bottom Right
1	(0, 128)	(96, 128)	(0, 200)	(96, 200)

Fourth column

In the last column, we'll study the area of the solar panels detected in square meters according to the spatial resolution of the image which will be imputed by the end user as a parameter. A table is generated for showing the coordinates of the four corners of bounding boxes, which locate the solar panels.

The implementation of different mathematical methods and use of different neural network models produced results that are great for the person using our website while also providing useful tools that can be used in real world problems like checking the exact size of the solar panels in an image taking into account the closeness of the image.

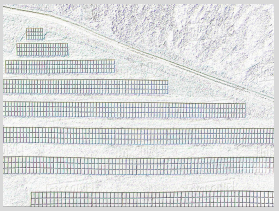
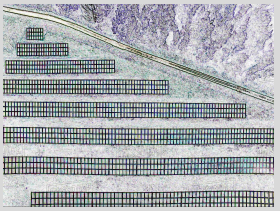
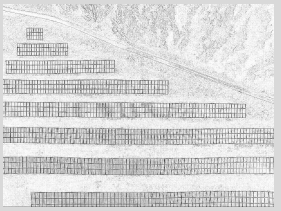
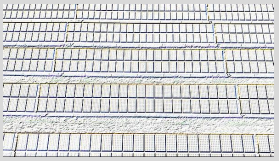
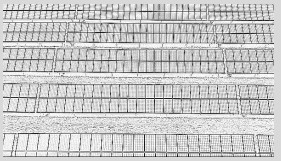
8. Future updates

8.1 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.

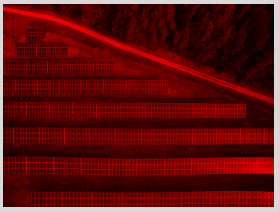
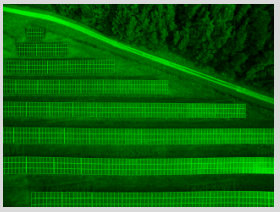
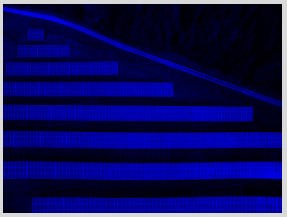
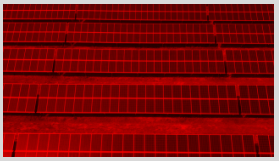
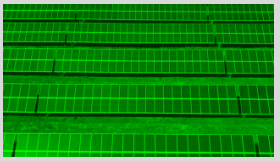
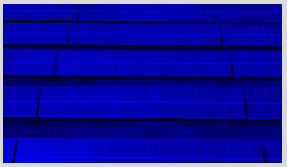
Sample Images	Robert Cross Filter	Sobel Filter	Laplacian Filter
			
			

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.

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.

- 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)

8.2 Other models

Up to this point, in terms of object detection, we successfully trained a YOLOv5 model, but there are other models that could improve our performance.

We have already seen YOLACT is an interesting alternative 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.

Regarding segmentation, we were happy with the results obtained from DeepLabV3+, Unet++, and FPN. Some other alternative to try could be MaskRCNN, which outperforms FastRCNN and FasterRCNN in terms of efficiency and accuracy.

8.3 Extra features

We successfully implemented the most important features. However, some functionalities we could add to this project would be computing real coordinates (latitude and longitude), segmenting the building and computing the number of modules that a photovoltaic cell has.

For the coordinates, we would need an informative dataset that contained more geo-spatial information. With satellite data, we would get this problem fixed (we have already talked

about the difficulty of the problem of getting this data set up). For segmenting the building, we would just need a full dataset of buildings with ground truth masks, train the model and finally use it to detect buildings on Zenodo's or Google Maps data. Finally, for computing the modules that form a solar panel array, we could use classical techniques to segment the detection using edge filters, maybe clustering, etc. Overall, it could be added without much complexity.

9. Conclusion and final thoughts

From the very first moment, the main goal was to develop a model that would predict solar panels in aerial images and their estimated area. We acknowledge that we are still 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 tackled problem.

This data science project is being proven to be helpful in developing real world skills that will help us solve bigger tasks. As a team we are really happy that this opportunity has been given to us

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

UNet++: A Nested U-Net Architecture for Medical Image Segmentation
https://arxiv.org/pdf/1807.10165.pdf (!!!)

FPN: A Unified Architecture for Instance and Semantic Segmentation
http://presentations.cocodataset.org/COCO17-Stuff-FAIR.pdf (!!!)

PSPNet: Pyramid Scene Parsing Network
https://arxiv.org/pdf/1612.01105.pdf

DeepLabV3+:Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation
https://arxiv.org/pdf/1802.02611.pdf

Understanding Dice Loss for Crisp Boundary Detection
https://medium.com/ai-salon/understanding-dice-loss-for-crisp-boundary-detection-bb30c2e5f62b