panel-segmentation Documentation

Release 0+unknown

NREL PVP&R Team

May 12, 2022

CONTENTS

CHAPTER

ONE

GETTING STARTED

This page documents how to install the Panel Segmentation package and run automated metadata extraction for a PV array at a specified location. These instructions assume that you already have Anaconda and git installed.

1.1 Installing

First, clone the Panel-Segmentation repository to your computer with git. This will bring the source code of the package locally to your computer.

First, create a new conda environment and activate it (optional):

```
conda create -n panel-segmentation-dev python=3.7
conda activate panel-segmentation-dev
```

Now you should change the working directory to the Panel-Segmentation repository folder and install the package:

```
pip install .
```

If you want to use the precise package versions used in the example notebooks, you can use the `requirements.txt` file:

```
pip install -r requirements.txt
```

Now you should be able to import *panel_segmentation* in a python terminal

```
import panel_segmentation
```

The recommended way of running the code is through a normal python terminal so that the conda environment is kept clean, but if you want, you can install spyder in the environment too. Note that you'll have to start spyder in a terminal with the conda environment activated for it to have access to the packages we just installed.

1.2 Running a single system

A satellite image analysis is performed using the `panel_detection.PanelDetection` class. This class allows the user to generate a satellite image based on a set of latitude-longitude coordinates, run the satellite image through a image segmentation model to determine presence of a solar on a pixel-by-pixel basis, cluster individual solar arrays in an image, and estimate the azimuth of each detected solar array.

```
from panel_segmentation.panel_detection import PanelDetection
from panel_segmentation import panel_detection as pseg
import numpy as np
from tensorflow.keras.preprocessing import image as imagex
import matplotlib.pyplot as plt

#Example latitude-longitude coordinates to run the analysis on.
latitude = 39.7407
longitude = -105.1694
google_maps_api_key =  "YOUR API KEY HERE"
file_name_save = "sat_img.png"

#CREATE AN INSTANCE OF THE PANELDETECTION CLASS TO RUN THE ANALYSIS
panelseg = pseg.PanelDetection(model_file_path ='./panel_segmentation/VGG16Net_
↪ConvTranpose_complete.h5',
                                classifier_file_path ='./panel_segmentation/VGG16_
↪classification_model.h5')

#GENERATE A SATELLITE IMAGE USING THE ASSOCIATED LAT-LONG COORDS AND THE GOOGLE
#MAPS API KEY
img = panelseg.generateSatelliteImage(latitude, longitude,
                                       file_name_save,
                                       google_maps_api_key)
#Show the generated satellite image
plt.imshow(img)

#LOAD THE IMAGE AND DECLARE AS A NUMPY ARRAY
x = imagex.load_img(file_name_save,
                    color_mode='rgb',
                    target_size=(640,640))
x = np.array(x)

#USE CLASSIFIER MODEL TO DETERMINE IF A SOLAR ARRAY HAS BEEN DETECTED IN THE
#IMAGE
panel_loc = panelseg.hasPanels(x)

#Mask the satellite image
res = panelseg.testSingle(x.astype(float), test_mask=None,  model =None)
#Use the mask to isolate the panels
new_res = panelseg.cropPanels(x, res)
plt.imshow(new_res.reshape(640,640,3))

#check azimuth
az = panelseg.detectAzimuth(new_res)

#plot edges + azimuth
panelseg.plotEdgeAz(new_res,10, 1,
                     save_img_file_path = './')

#PERFORM AZIMUTH ESTIMATION FOR MULTIPLE CLUSTERS
#Cluster panels in an image. The image to be passed are the "isolated panels",
#mask and number of clusters

number_arrays = 5
clusters = panelseg.clusterPanels(new_res, res,
                                  number_arrays)
```

(continues on next page)

(continued from previous page)

```
for ii in np.arange(clusters.shape[0]):
    az = panelseg.detectAzimuth(clusters[ii][np.newaxis,:])
    print(az)


#Then we can find azimuth for each cluster
panelseg.plotEdgeAz(clusters, 10, 1,
                     save_img_file_path = './')
# Get the mounting configuration classification
panelseg.classifyMountingConfiguration(image_file_path = file_name_save)

# Run the site analysis pipeline on the image
panelseg.runSiteAnalysisPipeline("./panel_segmentation/examples/Panel_Detection_
↪Examples/sat_img.png")
```

CHAPTER

TWO

API REFERENCE

2.1 Classes

These classes perform analyses used in the Panel-Segmentation project.

panel_detection.PanelDetection([…])	A class for training a deep learning architecture, detecting solar arrays from a satellite image, performing spectral clustering, predicting azimuth, and classifying mounting type and configuration.
panel_detection.PanelDetection.generateSatelliteImage(…)	Generates satellite image via Google Maps, using a set of lat-long coordinates.
panel_detection.PanelDetection.classifyMountingConfiguration(…)	This function is used to detect and classify the mounting configuration of solar installations in satellite imagery.
panel_detection.PanelDetection.diceCoeff(…)	This function is used as the metric of similarity between the predicted mask and ground truth.
panel_detection.PanelDetection.diceCoeffLoss(…)	This function is a loss function that can be used when training the segmentation model.
panel_detection.PanelDetection.testBatch(…)	This function is used to predict the mask of a batch of test satellite images.
panel_detection.PanelDetection.testSingle(…)	This function is used to predict the mask corresponding to a single test image.
panel_detection.PanelDetection.hasPanels(…)	This function is used to predict if there is a panel in an image or not.
panel_detection.PanelDetection.detectAzimuth(in_img)	This function uses canny edge detection to first extract the edges of the input image.
panel_detection.PanelDetection.cropPanels(…)	This function basically isolates regions with solar panels in a satellite image using the predicted mask.
panel_detection.PanelDetection.plotEdgeAz(…)	This function is used to generate plots of the image with its azimuth It can generate three figures or one.
panel_detection.PanelDetection.clusterPanels(…)	This function uses connected component algorithm to cluster the panels

continues on next page

Table 1 – continued from previous page

panel_detection.PanelDetection.runSiteAnalysisPipeline(...)	This function runs a site analysis on a site, when latitude and longitude coordinates are given. It includes the following steps: 1. If generate_image = True, taking a satellite image in Google Maps of site location, based on its latitude-longitude coordinates. The satellite image is then saved under 'file_name_save_img' path. 2. Running the satellite image through the mounting configuration/type pipeline. The associated mount predictions are returned, and the most frequently occurring mounting configuration of the predictions is selected. The associated labeled image is stored under the 'file_name_save_mount' path. 3. Running the satellite image through the azimuth estimation algorithm. A default single azimuth is calculated in this pipeline for simplicity. The detected azimuth image is saved via the file_path_save_azimuth path. 4. If a mounting configuration is detected as a single-axis tracker, an azimuth correction of 90 degrees is applied, as azimuth runs parallel to the installation, as opposed to perpendicular. 5. A final dictionary of analysed site metadata is returned, including latitude, longitude, detected azimuth, and mounting configuration.
panel_train.TrainPanelSegmentationModel(...)	A class for training a deep learning architecture to perform image segmentation on satellite images to detect solar arrays in the image.
panel_train.TrainPanelSegmentationModel.loadImagesToNumpyArray(...)	Load in a set of images from a folder into a 4D numpy array, with dimensions (number images, 640, 640, 3).
panel_train.TrainPanelSegmentationModel.diceCoeff(...)	Accuracy metric is overly optimistic.
panel_train.TrainPanelSegmentationModel.diceCoeffLoss(...)	This function is a loss function that can be used when training the segmentation model.
panel_train.TrainPanelSegmentationModel.trainSegmentation(...)	This function uses VGG16 as the base network and as a transfer learning framework to train a model that segments solar panels from a satellite image.
panel_train.TrainPanelSegmentationModel.trainPanelClassifier(...)	This function uses VGG16 as the base network and as a transfer learning framework to train a model that predicts the presence of solar panels in a satellite image.
panel_train.TrainPanelSegmentationModel.trainMountingConfigClassifier(...)	This function uses Faster R-CNN ResNet50 FPN as the base network and as a transfer learning framework to train a model that performs object detection on the mounting configuration of solar arrays.
panel_train.TrainPanelSegmentationModel.trainingStatistics(...)	This function prints the training statistics such as training loss and accuracy and validation loss and accuarcy.

2.1.1 panel_segmentation.panel_detection.PanelDetection

class panel_segmentation.panel_detection.**PanelDetection**(*model_file_path='./VGG16Net_ConvTranpose_com classifier_file_path='./VGG16_classification_model.h5', mounting_classifier_file_path='./object_detection_model.*

A class for training a deep learning architecture, detecting solar arrays from a satellite image, performing spectral clustering, predicting azimuth, and classifying mounting type and configuration.

__init__(*model_file_path='./VGG16Net_ConvTranpose_complete.h5', classifier_file_path='./VGG16_classification_model.h5', mounting_classifier_file_path='./object_detection_model.pth'*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`([model_file_path, ...])	Initialize self.
`classifyMountingConfiguration`(image_file_path)	This function is used to detect and classify the mounting configuration of solar installations in satellite imagery.
`clusterPanels`(test_mask[, fig])	This function uses connected component algorithm to cluster the panels
`cropPanels`(test_data, test_res)	This function basically isolates regions with solar panels in a satellite image using the predicted mask.
`detectAzimuth`(in_img[, number_lines])	This function uses canny edge detection to first extract the edges of the input image.
`diceCoeff`(y_true, y_pred[, smooth])	This function is used as the metric of similarity between the predicted mask and ground truth.
`diceCoeffLoss`(y_true, y_pred)	This function is a loss function that can be used when training the segmentation model.
`generateSatelliteImage`(latitude, longitude, ...)	Generates satellite image via Google Maps, using a set of lat-long coordinates.
`hasPanels`(test_data)	This function is used to predict if there is a panel in an image or not.
`plotEdgeAz`(test_results[, no_lines, ...])	This function is used to generate plots of the image with its azimuth It can generate three figures or one.

continues on next page

Table 2 – continued from previous page

runSiteAnalysisPipeline(file_name_save_img)	This function runs a site analysis on a site, when latitude and longitude coordinates are given. It includes the following steps: 1. If generate_image = True, taking a satellite image in Google Maps of site location, based on its latitude-longitude coordinates. The satellite image is then saved under 'file_name_save_img' path. 2. Running the satellite image through the mounting configuration/type pipeline. The associated mount predictions are returned, and the most frequently occurring mounting configuration of the predictions is selected. The associated labeled image is stored under the 'file_name_save_mount' path. 3. Running the satellite image through the azimuth estimation algorithm. A default single azimuth is calculated in this pipeline for simplicity. The detected azimuth image is saved via the file_path_save_azimuth path. 4. If a mounting configuration is detected as a single-axis tracker, an azimuth correction of 90 degrees is applied, as azimuth runs parallel to the installation, as opposed to perpendicular. 5. A final dictionary of analysed site metadata is returned, including latitude, longitude, detected azimuth, and mounting configuration.
testBatch(test_data[, test_mask, . . .])	This function is used to predict the mask of a batch of test satellite images.
testSingle(test_data[, test_mask, model])	This function is used to predict the mask corresponding to a single test image.

2.1.2 panel_segmentation.panel_detection.PanelDetection.generateSatelliteImage

`PanelDetection.`**`generateSatelliteImage`**(*latitude*, *longitude*, *file_name_save*, *google_maps_api_key*)

Generates satellite image via Google Maps, using a set of lat-long coordinates.

Parameters

- **latitude** (`float`) – Latitude coordinate of the site.
- **longitude** (`float`) – Longitude coordinate of the site.
- **file_name_save** (`string`) – File path that we want to save the image to, where the image is saved as a PNG file.
- **google_maps_api_key** (`string`) – Google Maps API Key for automatically pulling satellite images.

Returns

- *Figure*
- *Figure of the satellite image*

2.1.3 panel_segmentation.panel_detection.PanelDetection.classifyMountingConfiguration

`PanelDetection.`**`classifyMountingConfiguration`**(*image_file_path*, *acc_cutoff=0.65*, *file_name_save=None*, *use_nms=True*)

This function is used to detect and classify the mounting configuration of solar installations in satellite imagery. It leverages the Detecto package's functionality (https://detecto.readthedocs.io/en/latest/api/index.html), to perform object detection on a satellite image.

Parameters

- **image_file_path** (`string`) – File path of the image. PNG file.
- **acc_cutoff** (`float`) – Default set to 0.65. Confidence cutoff for whether or not to count a object detection classification as real. All returned classications greater than or equal to the accuracy cutoff are counted, and all classifications less than the accuracy cutoff are thrown out.

Returns tuple Tuple consisting of (scores, labels, boxes), where 'scores' is the list of object detection confidence scores, 'labels' is a list of all corresponding labels, and 'boxes' is a tensor object containing the associated boxes for the associated labels and confidence scores.

Return type Returns

2.1.4 panel_segmentation.panel_detection.PanelDetection.diceCoeff

`PanelDetection.`**`diceCoeff`**(*y_true*, *y_pred*, *smooth=1*)

This function is used as the metric of similarity between the predicted mask and ground truth.

Parameters

- **y_true** (`numpy array` of `floats`) – The true mask of the image
- **y_pred** (numpy array of `floats`) – the predicted mask of the data
- **smooth** (`int`) – A parameter to ensure we are not dividing by zero and also a smoothing parameter for back -ropagation. If the prediction is hard threshold to 0 and 1, it is difficult to back-propagate the dice loss gradient. We add this parameter to actually smooth out the loss function, making it differentiable.

Returns **dice** – Retuns the metric of similarity between prediction and ground truth

Return type `float`

2.1.5 panel_segmentation.panel_detection.PanelDetection.diceCoeffLoss

`PanelDetection.`**`diceCoeffLoss`**(*y_true*, *y_pred*)

This function is a loss function that can be used when training the segmentation model. This loss function can be used in place of binary crossentropy, which is the current loss function in the training stage.

Parameters

- **y_true** (`numpy array` of `floats`) – The true mask of the image
- **y_pred** (`numpy array` of `floats`) – The predicted mask of the data

Returns

- *float*
- *The loss metric between prediction and ground truth*

2.1.6 panel_segmentation.panel_detection.PanelDetection.testBatch

`PanelDetection.testBatch` (*test_data*, *test_mask=None*, *batch_size=16*, *model=None*)

This function is used to predict the mask of a batch of test satellite images. Use this to test a batch of images greater than 4.

Parameters

- **test_data** (`nparray float`) – The satellite images
- **test_mask** (`nparray int` or `float`) – The mask ground truth corresponding to the test_data
- **batch_size** (`int`) – The batch size of the test_data.
- **model** (`tf.keras.model.object`) – A custom model can be provided as input or we can use the initialized model

Returns

- **test_res** (`nparray float`) – The predicted masks
- **accuracy** (`float`) – The accuarcy of prediction as compared with the ground truth if provided

2.1.7 panel_segmentation.panel_detection.PanelDetection.testSingle

`PanelDetection.testSingle` (*test_data*, *test_mask=None*, *model=None*)

This function is used to predict the mask corresponding to a single test image. It takes as input the test_data (a required parameter) and two non-required parameters- test_mask and model. Use this to test a single image.

Parameters

- **test_data** (`nparray int` or `float`) – The satellite image. dimension is (640,640,3) or (a,640,640,3)
- **test_mask** (`nparray int` or `float`) – The ground truth of what the mask should be.
- **model** (`tf.keras model object`) – A custom model can be provided as input or we can use the initialized model

Returns

- **test_res** (`nparray float`) – The predicted mask of the single image. The dimension is (640,640 or (a,640,640))
- **accuracy** (`float`) – The accuracy of prediction as compared with the ground truth if provided

2.1.8 panel_segmentation.panel_detection.PanelDetection.hasPanels

`PanelDetection.hasPanels` (*test_data*)

This function is used to predict if there is a panel in an image or not. Note that it uses a saved classifier model we have trained and not the segmentation model.

Parameters **test_data** (`nparray float` or `int`) – The satellite image. The shape should be [a,640,640,3] where 'a' is the number of data or (640,640,3) if it is a single image

Returns

- *boolean*

- *True if solar array is detected in an image,*
- *and False otherwise.*

2.1.9 panel_segmentation.panel_detection.PanelDetection.detectAzimuth

`PanelDetection.`**`detectAzimuth`**(*in_img*, *number_lines=5*)

This function uses canny edge detection to first extract the edges of the input image. To use this function, you have to first predict the mask of the test image using testSingle function. Then use the cropPanels function to extract the solar panels from the input image using the predicted mask. Hence the input image to this function is the cropped image of solar panels.

After edge detection, Hough transform is used to detect the most dominant lines in the input image and subsequently use that to predict the azimuth of a single image.

Parameters

- **in_img** (`nparray uint8`) – The image containing the extracted solar panels with other pixels zeroed off. Dimension is [1,640,640,3]
- **number_lines** (`int`) – This variable tells the function the number of dominant lines it should examine. We currently inspect the top 5 lines.

Returns **azimuth** – The azimuth of the panel in the image.

Return type `int`

2.1.10 panel_segmentation.panel_detection.PanelDetection.cropPanels

`PanelDetection.`**`cropPanels`**(*test_data*, *test_res*)

This function basically isolates regions with solar panels in a satellite image using the predicted mask. It zeros out other pixels that does not contain a panel. You can use this for a single test data or multiple test data.

Parameters

- **test_data** (`nparray float`) – This is the input test data. This can be a single image or multiple image. Hence the dimension can be (640,640,3) or (a,640,640,3)
- **test_res** (`nparray float`) – This is the predicted mask of the test images passed as an input and used to crop out the solar panels. Dimension is (640,640)

Returns **new_test_res** – This returns images here the solar panels have been cropped out and the background zeroed. It has the same shape as test data. The dimension is [a,640,640,3] where a is the number of input images.

Return type `nparray uint8`

2.1.11 panel_segmentation.panel_detection.PanelDetection.plotEdgeAz

`PanelDetection.`**`plotEdgeAz`**(*test_results*, *no_lines=5*, *no_figs=1*, *save_img_file_path=None*, *plot_show=False*)

This function is used to generate plots of the image with its azimuth It can generate three figures or one. For three figures, that include the input image, the hough transform space and the input images with detected lines. For single image, it only outputs the input image with detected lines.

Parameters

- **test_results** (`nparray float64` or `unit8`) – 8-bit input image. This variable represents the predicted images from the segmentation model. Hence the dimension must be [a,b,c,d] where [a] is the number of images, [b,c] are the dimensions of the image - 640 x 640 in this case and [d] is 3 - RGB
- **no_lines** (`int`) – default is 10. This variable tells the function the number of dominant lines it should examine.
- **no_figs** (`int`) – 1 or 3. If the number of figs is 1, it outputs the mask with Hough lines and the predicted azimuth. However, if the number of lines is 3, it gives three plots.
  1. The input image,
  2. Hough transform search space
  3. Unput image with houghlines and the predicted azimuth
- **save_img_file_path** (`string`) – You can pass as input the location to save the plots
- **plot_show** (`boolean`) – If False, it will supress the plot as an output and just save the plots in a folder

Returns

- *Figure*
- *Plot of the masked image, with detected Hough Lines and azimuth*
- *estimate.*

2.1.12 panel_segmentation.panel_detection.PanelDetection.clusterPanels

`PanelDetection.`**`clusterPanels`**(*test_mask*, *fig=False*)

This function uses connected component algorithm to cluster the panels

Parameters

- **test_mask** (`boolean` or `float`) – The predicted mask. Dimension is (640,640) or can be converted to RGB (640,640,3)
- **fig** (`boolean`) – shows the clustering image if fig = True

Returns

- *uint8*
- *Masked image containing detected clusters each of*
- *dimension (640,640,3)*
- *uint8*
- *The optimal number of clusters*

2.1.13 panel_segmentation.panel_detection.PanelDetection.runSiteAnalysisPipeline

`PanelDetection.runSiteAnalysisPipeline`(*file_name_save_img*, *latitude=None*, *longitude=None*, *google_maps_api_key=None*, *file_name_save_mount=None*, *file_path_save_azimuth=None*, *generate_image=False*)

This function runs a site analysis on a site, when latitude and longitude coordinates are given. It includes the following steps:

1. **If generate_image = True, taking a satellite image in** Google Maps of site location, based on its latitude-longitude coordinates. The satellite image is then saved under 'file_name_save_img' path.
2. **Running the satellite image through the mounting** configuration/type pipeline. The associated mount predictions are returned, and the most frequently occurring mounting configuration of the predictions is selected. The associated labeled image is stored under the 'file_name_save_mount' path.
3. **Running the satellite image through the azimuth estimation** algorithm. A default single azimuth is calculated in this pipeline for simplicity. The detected azimuth image is saved via the file_path_save_azimuth path.
4. **If a mounting configuration is detected as a single-axis** tracker, an azimuth correction of 90 degrees is applied, as azimuth runs parallel to the installation, as opposed to perpendicular.
5. **A final dictionary of analysed site metadata is returned,** including latitude, longitude, detected azimuth, and mounting configuration.

Parameters

- **file_name_save_img** (`string`) – File path that we want to save the raw satellite image to. PNG file.
- **latitude** (`float`) – Default None. Latitude coordinate of the site. Not required if we're using a pre-generated satellite image.
- **longitude** (`float`) – Default None. Longitude coordinate of the site. Not required if we're using a pre-generated satellite image.
- **google_maps_api_key** (`string`) – Default None. Google Maps API Key for automatically pulling satellite images. Not required if we're using a pre-generated satellite image.
- **file_name_save_mount** (`string`) – File path that we want to save the labeled mounting configuration image to. PNG file.
- **file_name_save_azimuth** (`string`) – File path that we want to save the predicted azimuth image to. PNG file.
- **generate_image** (`bool`) – Whether or not we should generate the image via the Google Maps API. If set to True, satellite image is generated and saved. Otherwise, no image is generated and the image saved under the file_name_save_img path is used.

Returns Dictionary containing the latitude, longitude, classified mounting configuration, and the estimated azimuth of a site.

Return type Python dictionary

2.1.14 panel_segmentation.panel_train.TrainPanelSegmentationModel

class panel_segmentation.panel_train.**TrainPanelSegmentationModel**(*batch_size*, *no_epochs*, *learning_rate*)

A class for training a deep learning architecture to perform image segmentation on satellite images to detect solar arrays in the image.

__init__(*batch_size*, *no_epochs*, *learning_rate*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(batch_size, no_epochs, learning_rate)	Initialize self.
diceCoeff(y_true, y_pred[, smooth])	Accuracy metric is overly optimistic.
diceCoeffLoss(y_true, y_pred)	This function is a loss function that can be used when training the segmentation model.
loadImagesToNumpyArray(image_file_path)	Load in a set of images from a folder into a 4D numpy array, with dimensions (number images, 640, 640, 3).
trainMountingConfigClassifier(train_path, …)	This function uses Faster R-CNN ResNet50 FPN as the base network and as a transfer learning framework to train a model that performs object detection on the mounting configuration of solar arrays.
trainPanelClassifier(train_path, val_path[, …])	This function uses VGG16 as the base network and as a transfer learning framework to train a model that predicts the presence of solar panels in a satellite image.
trainSegmentation(train_data, train_mask, …)	This function uses VGG16 as the base network and as a transfer learning framework to train a model that segments solar panels from a satellite image.
trainingStatistics(results, mode)	This function prints the training statistics such as training loss and accuracy and validation loss and accuarcy.

2.1.15 panel_segmentation.panel_train.TrainPanelSegmentationModel.loadImagesToNumpyArr

TrainPanelSegmentationModel.**loadImagesToNumpyArray**(*image_file_path*)

Load in a set of images from a folder into a 4D numpy array, with dimensions (number images, 640, 640, 3).

Parameters **image_file_path** (string) – Path to folder where we want to process png images.

Returns

- *nparray*
- *4D numpy array with dimensions*
- *(number images in folder, 640, 640, 3).*

2.1.16 panel_segmentation.panel_train.TrainPanelSegmentationModel.diceCoeff

`TrainPanelSegmentationModel.diceCoeff`(*y_true*, *y_pred*, *smooth=1*)

Accuracy metric is overly optimistic. IOU, dice coefficient are more suitable for semantic segmentation tasks. This function is used as the metric of similarity between the predicted mask and ground truth.

Parameters

- **y_true** (`nparray float`) – the true mask of the image
- **y_pred** (`nparray float`) – the predicted mask of the data
- **smooth** (`int`) – a parameter to ensure we are not dividing by zero and also a smoothing parameter. For back propagation. If the prediction is hard threshold to 0 and 1, it is difficult to back propagate the dice loss gradient. We add this parameter to actually smooth out the loss function, making it differentiable.

Returns **dice** – The metric of similarity between prediction and ground truth

Return type `float`

2.1.17 panel_segmentation.panel_train.TrainPanelSegmentationModel.diceCoeffLoss

`TrainPanelSegmentationModel.diceCoeffLoss`(*y_true*, *y_pred*)

This function is a loss function that can be used when training the segmentation model. This loss function can be used in place of binary crossentropy, which is the current loss function in the training stage.

Parameters

- **y_true** (`nparray float`) – The true mask of the image
- **y_pred** (`nparray float`) – The predicted mask of the data

Returns

- *float*
- *The loss metric between prediction and ground truth*

2.1.18 panel_segmentation.panel_train.TrainPanelSegmentationModel.trainSegmentation

`TrainPanelSegmentationModel.trainSegmentation`(*train_data*, *train_mask*, *val_data*, *val_mask*, *model_file_path='/home/docs/checkouts/readthedocs.org/user_bui segmentation/envs/stable/lib/python3.7/site-packages/panel_segmentation/VGG16Net_ConvTranpose_complet*

This function uses VGG16 as the base network and as a transfer learning framework to train a model that segments solar panels from a satellite image. It uses the training data and mask to learn how to predict the mask of a solar array from a satellite image. It uses the validation data to prevent overfitting and to test the prediction on the fly. The validation data is also use to validate when to save the best model during training.

Parameters

- **train_data** (`nparray float`) – This should be the training images.
- **train_mask** (`nparray int/float`) – This should be the training images mask - ground truth
- **val_data** (`nparray float`) – This should be the validation images
- **val_mask** (`nparray float`) – This should be the validation images mask - ground truth

Notes

Hence the dimension of the four variables must be [a,b,c,d] where [a] is the number of input images, [b,c] are the dimensions of the image - 640 x 640 in this case and [d] is 3 - RGB

Returns

- **results** (`tf.keras.fit_generator History object`) – This varaiale contains training history and statistics
- **custom_model** (`tf.keras model object`) – The final trianed model. Note that this may not be the best model as the best model is saved during training

2.1.19 panel_segmentation.panel_train.TrainPanelSegmentationModel.trainPanelClassifier

`TrainPanelSegmentationModel.`**`trainPanelClassifier`**(*train_path*, *val_path*, *model_file_path='/home/docs/checkouts/readthedocs.org/user segmentation/envs/stable/lib/python3.7/site-packages/panel_segmentation/VGG16_classification_model.h*

This function uses VGG16 as the base network and as a transfer learning framework to train a model that predicts the presence of solar panels in a satellite image. It uses the training data to learn how to predict the presence of a solar array in a satellite image. It uses the validation data to prevent overfitting and to test the prediction on the fly. The validation data is also use to validate when to save the best model during training.

Parameters

- **train_path** (`string`) – This is the path to the folder that contains the training images Note that the directory must be structured in this format:

  train_path/

  . . . has panel/ . . . . . . a_image_1.jpg . . . . . . a_image_2.jpg

  . . . no panels/ . . . . . . b_image_1.jpg . . . . . . b_image_2.jpg

- **val_path** (`string`) – This is the path to the folder that contains the validation images Note that the directory must be structured in this format:

  val_path/

  . . . has panel/ . . . . . . a_image_1.jpg . . . . . . a_image_2.jpg

  . . . no panels/ . . . . . . b_image_1.jpg . . . . . . b_image_2.jpg

Returns

- **results** (`tf.keras.fit_generator History object`) – This varaiale contains training history and statistics
- **final_clas_model** (`tf.keras model object`) – The final trianed model. Note that this may not be the best model as the best model is saved during training

2.1.20 panel_segmentation.panel_train.TrainPanelSegmentationModel.trainMountingConfigClas

`TrainPanelSegmentationModel.`**`trainMountingConfigClassifier`**`(`*train_path*, *val_path*, *device=device(type='cuda')*`)`

This function uses Faster R-CNN ResNet50 FPN as the base network and as a transfer learning framework to train a model that performs object detection on the mounting configuration of solar arrays. It uses the training data to locate and classify mounting configuration of the solar installation. It uses the validation data to prevent overfitting and to test the prediction on the fly.

Parameters

- **train_path** (`string`) – This is the path to the folder that contains the training images Note that the directory must be structured in this format:

  train_path/

  ...images/ ......a_image_1.png ......a_image_2.png

  ...annotations/ ......b_image_1.xml ......b_image_2.xml

- **val_path** (`string`) – This is the path to the folder that contains the validation images Note that the directory must be structured in this format:

  val_path/

  ...images/ ......a_image_1.png ......a_image_2.png

  ...annotations/ ......b_image_1.xml ......b_image_2.xml

- **device** (`string`) – This argument is passed to the Model() class in Detecto. It determines how to run the model: either on GPU via Cuda (default setting), or on CPU. Please note that running the model on GPU results in significantly faster training times.

Returns **model** – The final trained mounting configuration object detection model.

Return type `detecto.core.Model object`

2.1.21 panel_segmentation.panel_train.TrainPanelSegmentationModel.trainingStatistics

`TrainPanelSegmentationModel.`**`trainingStatistics`**`(`*results*, *mode*`)`

This function prints the training statistics such as training loss and accuracy and validation loss and accuarcy. The dice coefficient was only used for segmentation and not panel classification. We use mode to decide if we should print out dice coefficient.

Parameters

- **results** (`tf.keras.fit_generator History object`) – This is the output of the trained classifier. It contains training history.
- **mode** (`int`) – If mode = 1, it assumes we want plots for the semantic segmentation and also plots the dice coefficient results. For any other value of mode, it does not show plots of dice coefficients.

Returns

- *figures*
- *Figures based on the model training statistics*

2.2 Models

The following deep learning models are included in the Panel-Segmentation package.

CHAPTER

THREE

CHANGE LOG

3.1 Version 0.0.2 (May 12, 2022)

Added the mounting object detection algorithm to detect and classify the mounting configuration of solar installations in satellite imagery. Updated several of the Github workflows to provide more rigorous testing protocols (requirements.txt check and flake8 check).

3.1.1 Documentation

- Updated Sphinx Documentation to account for new functions.
- Updated the Jupyter notebooks to reflect pipeline changes: adding in the mounting configuration detection classifier and running it on satellite imagery.

3.1.2 Scripts

- Add the function `panel_segmentation.panel_train.TrainPanelSegmentationModel.trainMountingConfigClassifier()`.
- Add the functions `panel_segmentation.panel_detection.PanelDetection.runSiteAnalysisPipeline()` and `panel_segmentation.panel_detection.classifyMountingConfiguration()`.
- Add unit testing for the `panel_segmentation.panel_train.TrainPanelSegmentationModel.trainMountingConfigClassifier()`, `panel_segmentation.panel_detection.runSiteAnalysisPipeline()`, and `panel_segmentation.panel_detection.PanelDetection.classifyMountingConfiguration()` functions.

3.1.3 Other Changes

- Add Github workflow checks to include requirements.txt checks and flake8 checks.

3.2 Version 0.0.1 (October 27, 2020)

Created initial package Panel-Segmentation for public release.

3.2.1 Documentation

- Add Sphinx documentation.
- Add commenting for each of the functions in the package.
- Add example Jupyter notebooks for panel detection and training the classifier and segmentation models.

3.2.2 Scripts

- Add PanelDetection class, where the user can generate a satellite image and run it through the pre-generated models.
- Add TrainPanelSegmentationModel() class, where the user can independently train segmentation and classifier models.
- Add unit testing for the PanelDetection() and TrainPanelSegmentationModel() classes, stored in the /tests/ folder. The pytest package was used.

3.2.3 Other Changes

- Add versioneer and setup scripts to perform pip installs of the package.

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

C

D

G

H

L

P

R

T