Phenotator-Toolbox (Object Detection)

The Phenotator-Toolbox can be used to train and use a neural object detection network (Faster RCNN). This document will instruct you how to install and use it.

1. Installation

The pipeline should be installed on a computer with a relatively powerful GPU. To install it, carry out the following steps:

1. Download the Phenotator Toolbox Github repository and extract it. This can either be done by downloading the repository as a zip file from https://github.com/tschutli/Phenotator-Toolbox and extracting it or by cloning the repository using the following command:

   ***git clone* https://github.com/tschutli/Phenotator-Toolbox**

2. Download Anaconda with Python 3.7 from https://www.anaconda.com/distribution/ and install it.

3. Open the Anaconda Prompt Program, change to the previously downloaded Phenotator-Toolbox folder and run the following command:

   cd /path/to/Phenotator-Toolbox/on/your/computer
   conda env create -f environment.yml

   This will install all python dependencies including the Tensorflow library with GPU support.

4. Run the following command to install the LabelMe application:

   pip install labelme

5. Done! These three steps should be sufficient to install everything you need to run the Image Segmentation Pipeline. Go to the Usage section to learn how to use it.

2. General Usage

Before being able to use the Phenotator-Toolbox, make sure that the Anaconda environment is activated. To do so, open Anaconda Prompt and activate the tf_gpu environment that you installed during the installation instructions:

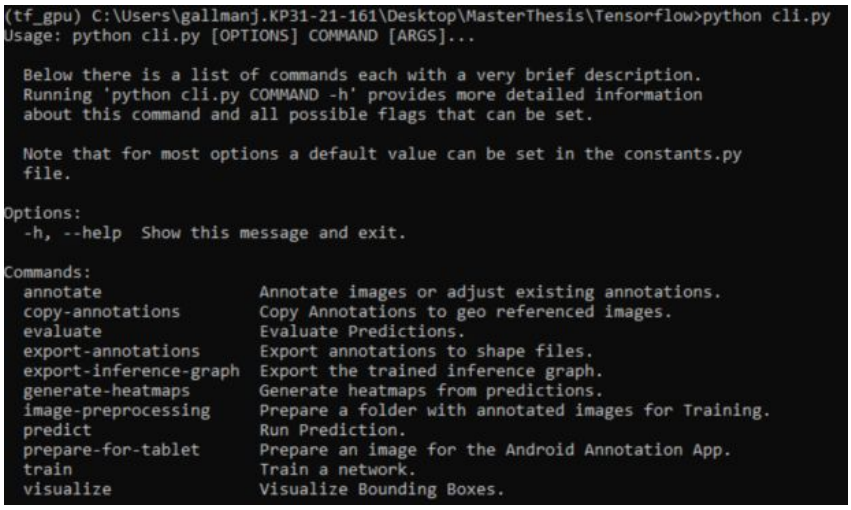
conda activate tf_gpu

Once the conda tf_gpu environment is activated, change to the Phenotator-Toolbox/Tensorflow directory:

cd /???/Phenotator-Toolbox/Tensorflow

Then execute the following command:

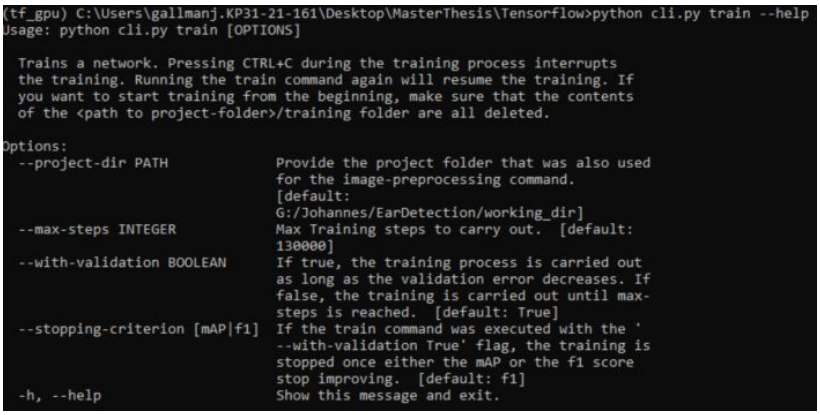
python cli.py



These three commands have to be executed before each usage of the Toolbox.
The last command starts a command line interface (CLI). It contains all functionality of the Toolbox. Typing the above command will present you with a list of commands that are available within the Toolbox. To get more information about a commands functionality, read the step-by-step instructions in this document or type:

python cli.py <command-name> --help

Example output of the above command:

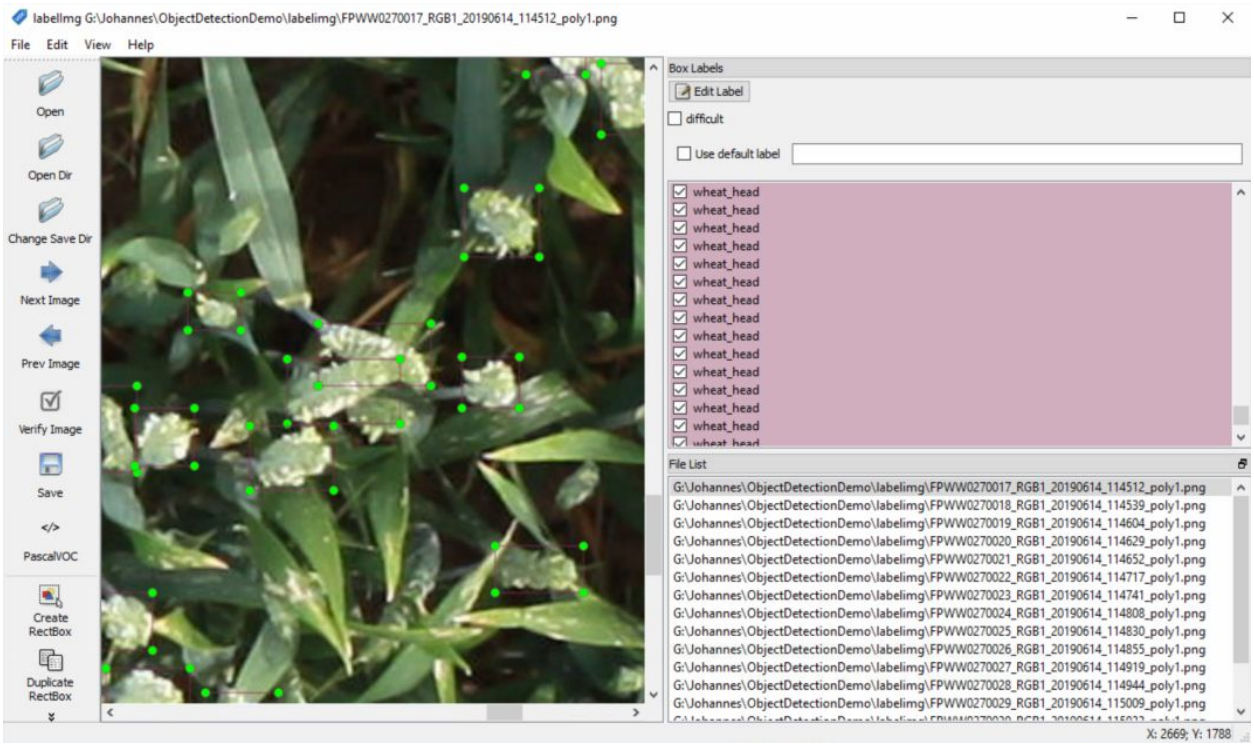


3. Step-by-step Instructions

3.1 Preparing Training Data

In order to train a neural network to detect one or multiple classes of objects, it has to be trained with a large amount of training data. So the first step is to prepare this training data. Follow the steps below to prepare your data.

1. The first step is to collect a large amount of images containing the objects you want to detect. Place all these images into one (or multiple) folders.
2. Inside the Phenotator-Toolbox folder, you can find an .exe file called **labelImg.exe.** Open the labelImg program.
3. Use the labelImg program to annotate all the objects within your images. To do so,
    - click on **Open Dir** and select the folder containing the images.
    - Click on **Change Save Dir** and select the same folder (containing the images)
    - By clicking **w**, annotate all objects within the first image.
    - Make sure to save your annotations by clicking on **Save**
    - Do the same for the other images.



Alternatives for Training Data Preparation:

Alternative 1: Alternatively, Training Data can be annotated on an Android Tablet using the PhenoAnnotator Application. To do so, the images have to be preprocessed by the following command:

python cli.py prepare-for-tablet <input-image> <output-folder>

Given an input-image (any format) and an output-folder, the command tiles the input-image into tiles of suitable size for an android tablet. If the input image is georeferenced (can be georeferenced tif or other image format with a imagename.imageformat.aux.xml file in the same folder(=>see gdal)), the script generates additional files with geo information that are read and used by the tablet app for displaying the user location. An additional advantage of using a georeferenced image as input is that after annotating on the tablet, all annotations can be copied onto other georeferenced images with the copy-annotations commmand.

Alternative 2: Annotate your training data with the LabelMe application. You can use LabelMe by typing the following command:

python cli.py annotate <input-folder>

Running this command will open the LabelMe Application with which all images in the input-folder can be annotated. If the images in the input folder are already annotated, these annotations can be viewed adjusted.

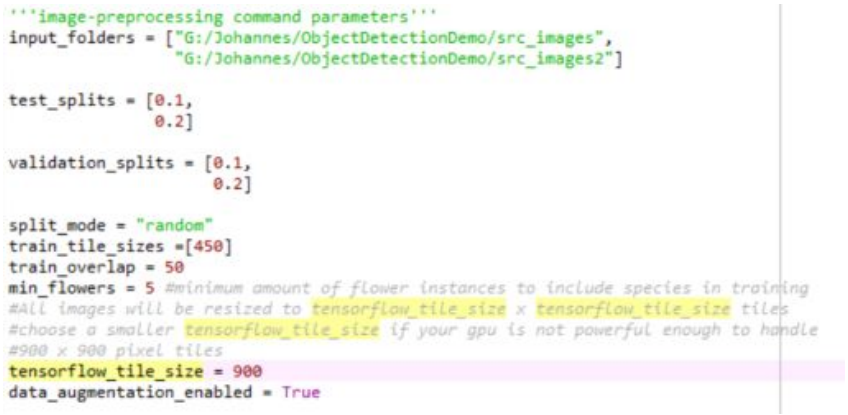
If the roi-strip flag is set to True, the user can select Regions of Interest (RoI) in the images. To do so, the user has to draw one or multiple polygons around the region(s) of interest and label them 'roi'. After the user has labelled all images accordingly, only the Regions of Interest (RoI) are kept in the images. The rest of the pixels are overriden with black.

3.2 Preparation for Training

Once the training data is available, you are ready to train a neural network. Before you can actually start the training process, some preparations have to be done, such as downloading a pretrained model, adjusting some hyperparameters in the model's config file, or converting and tiling the images and annotations into a tensorflow specific format. Fortunately all this is done by a script, which you can use as follows:

1. First, open ***/???/Phenotator-Toolbox/Tensorflow/utils/constants.py*** in a text editor.

2. Create an empty project folder anywhere on your computer. This folder will be used to save all intermediate and final results to. Inside the constants.py file set the project_folder variable to the path of your newly created project folder. Make sure to not use any backslashes, but only forward slashes.
3. Add all training data folders to the input_folders variable. Each input folder should contain images and annotation files. (As described in 3.1.)



```
'''image-preprocessing command parameters'''
input_folders = ["G:/Johannes/ObjectDetectionDemo/src_images",
                 "G:/Johannes/ObjectDetectionDemo/src_images2"]

test_splits = [0.1,
               0.2]

validation_splits = [0.1,
                     0.2]

split_mode = "random"
train_tile_sizes =[450]
train_overlap = 50
min_flowers = 5 #minimum amount of flower instances to include species in training
#All images will be resized to tensorflow_tile_size x tensorflow_tile_size tiles
#choose a smaller tensorflow_tile_size if your gpu is not powerful enough to handle
#900 x 900 pixel tiles
tensorflow_tile_size = 900
data_augmentation_enabled = True
```

4. In the test_splits and validation_splits variables, you have to define what portion of the data within each input folder should be used for testing (after training) and validating (during training). Normally these values should be chosen between 0.1 and 0.2.
5. OPTIONAL: Choose split_mode to be “random” or “deterministic”. It defines how the test set and validation set are created. If you are carrying out multiple experiments with different parameter configurations, “deterministic” is advantageous because it makes the results better comparable. Otherwise “random” should be preferred.
6. Define the tensorflow_tile_size. A value of 900 means, that the neural network will be trained with 900 x 900 pixel image tiles. Maximum is 1000 x 1000 pixels. Depending on your GPU, the tensorflow_tile_size has to be reduced to make sure the model can fit onto your GPU’s RAM. (GeForce GTX 1080 can handle 900x900 pixel tiles)
7. Define the train_tile_sizes variable. The train_tile_sizes variable defines the tile sizes your input images are tiled to. These can be multiple, but normally one tile_size should be sufficient. Note that during the training, these tiles are all resized to the tensorflow_tile_size defined in the previous step. If you set the train_tile_size to 450 and the tensorflow_tile_size to 900, your source images will all be tiled into 450 x 450 pixel images. During training however, they will be resized to 900 x 900 pixel images. This means that your images are scaled by a factor of 2 before the network trains on them. Upscaling your images has a positive effect on the performance of the network if you want to detect very small objects in your images (smaller than 40 pixel in diameter).
8. OPTIONAL: Define the train_overlap variable. It defines how many pixels of overlap your image tiles will have.
9. OPTIONAL: Define the min_flowers parameter. It defines how many instances of one class have to be present in your training data to include it in the training. If any class of object has less than min_flower instances present in the training data, it will be excluded from the training.
10. Inside the constants.py file also define whether data_augmentation should be enabled or not. Data-Augmentation-Options that are used are: Random vertical flips, Random

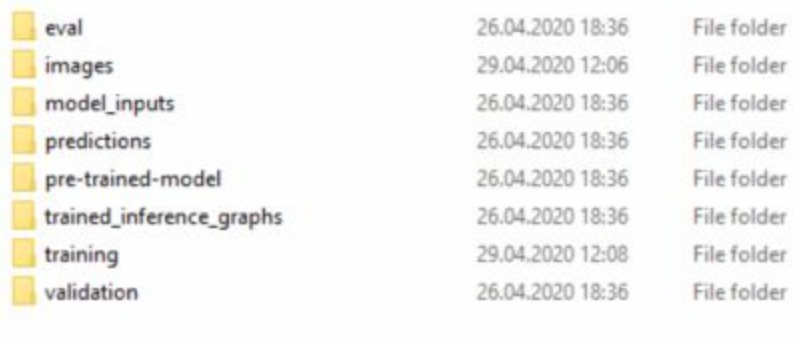
horizontal flips, Random brightness adjustments, Random contrast adjustments, random saturation adjustments, random box jittering. Note that brightness, contrast and saturation adjustments are only moderate. Your image colors will not be totally different, but only slightly.

11. Finally, everything is set up and you can open Conda Prompt and execute the following commands (see “General Usage” for explanations):

    conda activate tf_gpu
    cd /???/Phenotator-Toolbox/Tensorflow

12. To run the image-preprocessing command, execute the following line inside the conda prompt:

    python cli.py image-preprocessing



Once the image-preprocessing command has executed successfully, you should have a folder structure as in the image on the right. All image tiles can be found in the images folder. In the model_inputs folder, the tfrecord files are stored. These contain the image and annotation data in a tensorflow compatible format. Inside the pre-trained-model folder, the downloaded pretrained model can be found as well as the pipeline.config file. The rest of the folders should be empty and will be used by the subsequent commands.

3.3 Training a Network

Once the image-preprocessing command is executed, training a network is simple. Make sure the project_folder inside the ***/???/Phenotator-Toolbox/Tensorflow/utils/constants.py*** file is correct. Then execute the following command:

python cli.py train

You can stop the training process by pressing Ctrl+C. Later you can resume the training process by running the above command again.

OPTIONAL: Optionally you can set the max_steps variable inside the constants.py file. It defines for how many steps a network is trained at most. More than 100000 should normally not be necessary. Additionally, you can select whether the validation set should be used to determine when to stop training and when to adjust the learning rate (with_validation=True). If set to false, the network will train for max_steps steps. Also you can choose between “f1” and “mAP” as the criterion for the evaluations on the validation set. The training will stop (or change learning rate), once for 15000 no improvement on the performance on the validation set is made.

3.3 Exporting trained inference graph

Once you are done training. Either because the training script stopped by itself or you killed it by pressing Ctrl+C, you have to export the trained inference graph. This inference graph is basically the trained model which can be used to make predictions. Execute the following command:

python cli.py export-inference-graph

The trained inference graph will be saved to /???/project_dir/trained_inference_graphs/output_inference_graph_v1.pb

3.4 Evaluating Performance of Trained Network

Having the inference graph exported, you can use it to make predictions on images. To evaluate the performance of the model, you should first make predictions on the test images and then compare these predictions to the ground truth annotations. To do so, follow these steps:

1. Open the the ***/???/Phenotator-Toolbox/Tensorflow/utils/constants.py*** file.
2. Make sure that the project_folder path is correct.
3. Define the folder path containing the images you want to predict (images_to_predict). By default it is set to project_folder/images/test. This is the folder containing the test images.
4. Define an output folder, where the predictions should be saved to (predictions_folder). By default, it is set to project_folder/predictions.
5. Define the prediction_tile_size. Your images will be tiled into tiles of size prediction_tile_size x prediction_tile_size. Note that these tiles will be resized to tiles of size tensorflow_tile_size x tensorflow_tile_size before the prediction algorithm is executed. This gives you the opportunity to resize your images. The size of the objects in

the test images should be similar to the size of the objects in the train images (size in pixels).

6. Define a prediction_overlap. Your image tiles will overlap by prediction_overlap pixels.
7. Define min_confidence_score (between 0 and 1): Only predictions with a confidence score that is greater than min_confidence_score are considered.
8. OPTIONAL: Define a maximum intersection over union threshold. (max_iou) It should be a value between 0 and 1 indicating the maximal iou for the non maximum suppression algorithm. For all predictions with an iou greater than max-iou, only the one with the better score is kept.
9. Finally define your preferred visualization options:
    a. visualize_predictions: If True, the prediction bounding boxes are drawn onto copies of the images.
    b. visualize_groundtruth: If True, the ground truth bounding boxes are drawn onto copies of the images.
    c. visualize_name: If True, the name are drawn along with each bounding box.
    d. visualize_score: If True, the prediction score is drawn along with each bounding box.
10. Run the following command to make predictions:

    python cli.py predict

Once the prediction is done, you can evaluate the results by carrying out the following steps:

Open the the ***/???/Phenotator-Toolbox/Tensorflow/utils/constants.py*** file.
Optionally, define the following parameters (the default configuration should be sufficient for most use cases):

a. predictions_folder: The folder where the prediction results were saved to.
b. prediction_evaluation_folder: Defines where the evaluation results should be stored to. Default is predictions_folder/evaluations.
c. iou_threshold: Defines what is the minimum IoU (Intersection over Union) overlap to count a prediction as a True Positive.
d. min_confidence_score: the minimum score a prediction must have to be included in the evaluation.
e. generate_visualizations: If True, the erroneous predictions will be printed onto the images and saved to the evaluations_folder.
f. print_confusion_matrix: If True, the confusion matrix will be printed to the console in latex table format.
g. visualize_info: If True, in addition to the bounding boxes, info about the mispredictions is painted above the boxes.

Run the following command:

python cli.py evaluate

The results are printed to the console as well as saved to the prediction_evaluation_folder inside the results.txt file.

3.5 Make Predictions

Follow the steps in “3.4 Evaluating Performance of Trained Network” until step 10.

3.6 Generate Heatmaps from Predictions

Once predictions have been made, you can generate a heatmap from these predictions by executing the following command:

python cli.py generate-heatmaps <predictions-folder> <output-folder>

Replace <predictions-folder> with the path to the folder containing the predictions and replace <output-folder> to a path to an empty folder, where the heatmaps will be saved to. Most probably, executing this command will not generate a satisfying heatmap. Use the following flags to customize the command for your needs:

1. **--heatmap_width:** With this parameter you can adjust the granularity of your heatmap. It defines the number of pixels (buckets) the heatmap will have on the x axis. The height of the heatmap is chosen such that the width/height ratio is preserved. This heatmap will finally be resized to the size of the input (or background) image.
2. **--max_val:** If not set to None, it denotes the maximum value of the heatmap, meaning that all values in the heatmap that are larger than this max_val will be painted as red.
3. **--classname:** For which class the heatmap should be generated. If None is provided, only the overall heatmap for all classes is generated. This flag can be defined multiple times.
4. **--min-score:** The minimum score a prediction must have to be included in the heatmap.
5. **--overlay:** If True, the heatmap is drawn onpytto a copy of the input image. Otherwise it is drawn without any background.
6. **--output-image-width**: The width of the output image, the height is resized such that the width/height ratio is preserved.
7. **--generate-from-multiple**: If True, the script takes all predictions in the input folder and generates one heatmap from all of them. For this option, the input folder needs to contain georeferenced images and the background-image option has to be set.
8. **--back-ground-image**: The path to the image that should be used as background for the heatmap. (The background can still be deactivated with the --overlay flag but it needs to

be provided as a frame for the heatmap.) If generate-from-multiple is set to False, this option is ignored.

9. **--window**: Four float values indicating the [ulx, uly, lrx, lry] coordinates in the swiss coordinate system LV95+ of the area that should be used for the heatmap.
10. **--with-colorbar**: If True, a colorbar will be printed onto the output image.

Example:

python cli.py generate-heatmaps <predictions-folder> <output-folder> --heatmap-width 20 --max-val 10 --class "wheat_head" --min-score 0.2

3.7 Copy Annotations onto Georeferenced Images

If you have one folder with annotated images that are geo referenced, with the following command you can copy the annotations from that folder to other images that are also georeferenced:

python cli.py copy-annotations <annotated-folder> <to-be-annotated-folder>

An image can be georeferenced in the standard geotif format in any coordinate system. Alternatively, it can be a normal png or jpg image with a json file called imagename_geoinfo.json in the same folder. The imagename_geoinfo.json file must contain the following information in the WGS84 coordinate system: {"lr_lon": a, "lr_lat": b, "ul_lon": c, "ul_lat": d}

With CTRL+C the execution of the script can be interrupted. In the one-by-one mode, the execution can later be continued.

Type ***python cli.py copy-annotations --help*** for more information.

3.8 Convert annotations to shapefiles

The following command exports all annotations within the <annotation-folder> to shape files in the <output-folder>

python cli.py export-annotations <annotation-folder> <output-folder>