**COLONY COUNTING AND PICKING PACKAGE FOR OPENTRONS OT-2 (COPICK)**

**--------**

**COMPUTER VISION ENGINE**

This folder contains a purpose-made software package implementing the computer vision model required to analyze images of agar plates in order to identify bacterial colonies to pick. This package provides the hardware and software designs in different subfolders as described below:

* **Light frame CAD designs:** contains 3D models to build the photography setup structure for the acquisition of the images for the colony counting and picking dataset (COPICK dataset).
* **Detectron2:** the core of COPICK image recognition model. It contains the required libraries and scripts to work with a Convolutional Neural Network whose architecture was developed by Facebook research team. It is used to develop an inference model for colonies detection in agar plates. Scripts to set up the configuration for training, evaluation and optimization of the inference model, as well as the best trained model weights, are provided. Additionally, there are scripts to get and visualize model predictions.
* **Dataset preparation:** contains files related to the COPICK dataset preparation process. Scripts for image processing, segmentation, and automatic annotation and labelling are provided. There is also a \*.json file template to be employed during the labelling step (json\_template.json) and a \*.csv file containing the dataset information (Colony\_Picking\_database.csv).
* **Programs versions.pdf:** listed employed programs versions to develop the colony counting and picking system.

**DETECTRON2**

An inference model for bacterial colonies detection in agar plates was developed using Detectron2, a Facebook AI Research’s library for Computer Vision tasks (<https://github.com/facebookresearch/detectron2>). It is built for Linux and macOS with Python and it works with PyTorch and torchvision as Machine Learning Framework.

Installation of Detectron2 library for Linux or macOS can be checked here: <https://github.com/facebookresearch/detectron2>.

Additional instructions on the installation of Detectron2 for Windows are specified in Detectron2\_Windows.txt file (in Detectron2 folder). To build the colony detection model, Detectron2 for Windows with CPU only was used. A custom dataset for colony counting and picking purposes (COPICK dataset) was also created to train and evaluate the inference model.

From the different tasks Detectron2 offers, Panoptic Segmentation was chosen mainly because it allows segmentation of every pixel in an image**1**. Considering the scenario in which the COPICK dataset is built (colonies are sometimes very numerous, similar and small objects to detect in an image), Panoptic Segmentation was thought to be a good approach in this particular case to get an accurate inference model for bacterial colonies detection. Panoptic Segmentation model and weights previously trained on the COCO dataset (<https://cocodataset.org/#home>) were used as a starting point. The employed model was the Panoptic R101-FPN (3x) and its corresponding model weights to be found in Detectron2 MODEL ZOO:

<https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md>

To prepare and train the COPICK custom dataset, instructions on Detectron2 custom datasets and COCO dataset format for Panoptic FPN Segmentation models were followed:

* **Detectron2 custom datasets format:**

<https://detectron2.readthedocs.io/en/latest/tutorials/datasets.html>

* **Detectron 2 Panoptic FPN for COCO format:**

<https://detectron2.readthedocs.io/en/latest/tutorials/builtin_datasets.html>

* **COCO dataset format:**

<https://cocodataset.org/#format-data>

Overall, to prepare and register a custom dataset with COCO format for a Panoptic FPN model these are the files needed:

* **Original images**
* **Annotations:**
  + Panoptic masks (\*.png annotations): masks in which every pixel is assigned to a segment in the image (each segment has a unique id).
  + Semantic masks (\*.png annotations): refers to the stuff categories in the dataset (isthing=0), i.e. non-colonies objects in the image.
  + Panoptic JSON: includes panoptic segments annotations.
  + Instances JSON: includes things (isthing=1) segments annotations, i.e. colonies in the image.

There are different tools for image annotation and labeling, but the work has to be done manually in most cases for each object in the image. Therefore, traditional image processing techniques were used in order to generate automatic annotations of COPICK dataset. Instructions and scripts to generate the above files are detailed next in DATASET PREPARATION.

**DATASET PREPARATION**

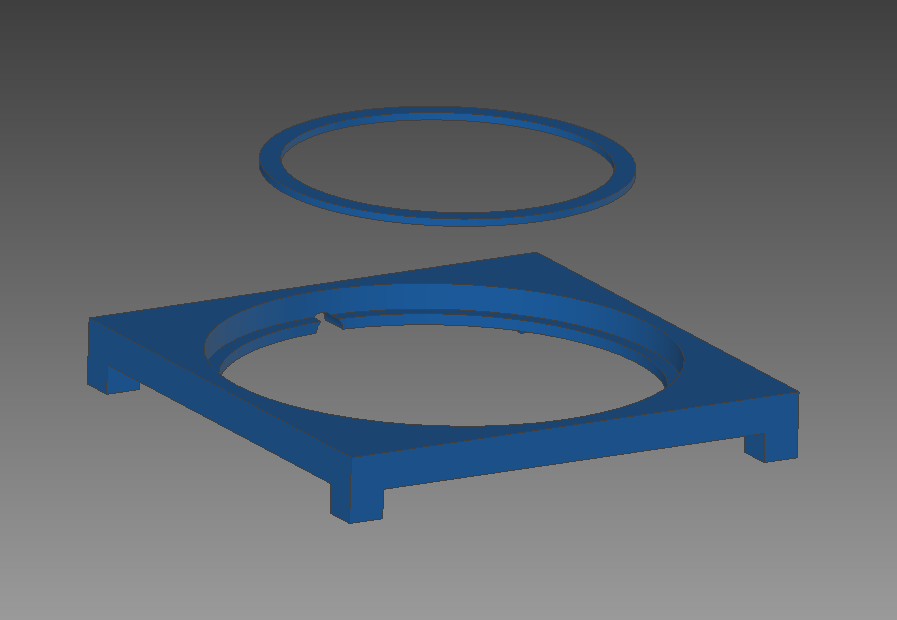
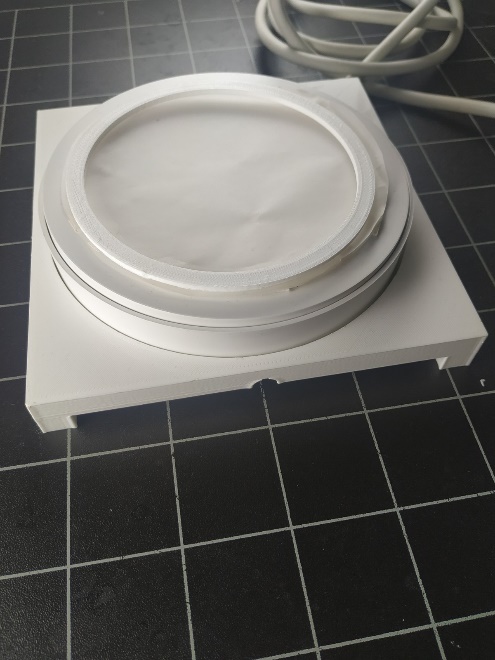
1. **Image acquisition**

To prepare the bacterial colony counting and picking dataset (COPICK dataset), a total of 200 petri dishes with different conditions were collected from lab colleagues (Víctor de Lorenzo’s lab, CNB-CSIC). The plates included different bacteria species (either *P. putida* KT-2440 strain colonies or *E. coli*) and mediums such as LB or M9 minimal medium*.* Total number of colonies in each plate were previously counted and added to the plates. Empty plates only with medium were also added.

Before image acquisition, plate preparation data (i.e. agar composition, strain, etc.) was annotated and associated with a plate number. All of this information was stored in a database (Colony\_Picking\_database.csv, in Dataset preparation folder) containing the following fields:

* **date:** image capture date.
* **id:** image number in the dataset.
* **filename:** assigned name of image using the file\_namer.py script in Dataset preparation folder.
* **category:** 0 for empty plates and 1 for plates with bacterial colonies.
* **colonies:** number of previously counted colonies in each plate.
* **strain:** None for empty plates, *P. putida* or *E. coli* for the rest of plates.
* **medium:** M9-cit (minimal M9 citrate medium) or LB.
* **multicolor:** 0 for plates with non-colored colonies and 1 for plates with colored colonies. This field was added for future additions to the dataset. In this case there were not plates with colored colonies.
* **obs:** plate observations.
* **colonysize**: general colonies size in the plate. Small, big or bigger categories.
* **recount**: number of colonies in plates after recounting some of them.

Image acquisition was performed with a Nikon reflex camera (model D60) with a macro lens (AF-S Micro NIKKOR 60 mm f/2.8G ED). To obtain good quality images, a custom-made illumination device was designed. It was composed by a led circular focus (123 mm, 6.7w, 1000 lumen) supported by a 3D holder designed using Autodesk Inventor Professional (2015) and printed with a Ultimaker3 3D printer (see **Figure 1**). Two light diffusing layers of translucid plastic and paper were added to the assembly in order to attenuate the amount of light. Both 3D models can be found in the Light frame CAD designs folder.



**Figure 1. Led focus support structure and plates ring holder.** **a)** 3D pieces design; **b)** final structure for image acquisition.

**a**

**b**

Images of the 200 collected plates were taken under the same conditions without lid for to avoid lines and reflections in the images.

1. **Dataset augmentation**

After image acquisition, dataset augmentation techniques were applied in order to increase the dataset size. Using the augmentation\_stage.py script (in Dataset preparation folder), flip and rotation transformations were applied to the original set of images generating 15 replicas of each one and resulting in a total of 3000 images in the COPICK dataset. Every image was assigned a different name using the file\_namer.py script (in Dataset preparation folder) during the augmentation process. Replicas of the same image share a common code at the beginning of their names for a better identification:

**Code** 🡪 A0000**X,** where the **X** corresponds to the assigned number of plate during image acquisition. Therefore, **X** values range from 1 to 200.

The 3000 images were divided in two subsets: train dataset (2250 images) and val dataset (750 images) for training and validation steps respectively. Previously mentioned database file (Colony\_Picking\_database.csv) contains the information of the whole augmented COPICK dataset.

1. **Image processing and annotation**

In order to produce panoptic and semantic annotations for each image in the dataset (as mentioned previously), two specific scripts (PROCESS\_DATASET\_PANOPTIC\_SEGMENTATION.py and PROCESS\_DATASET\_INSTANCE\_SEGMENTATION.py) and auxiliary functions were created (see Dataset preparation folder).

First, PROCESS\_DATASET\_PANOPTIC\_SEGMENTATION.py was executed. In this script, the steps are:

1. Define dataset images paths and choose train or val datasets to be processed.
2. Then load and set fields to the provided json\_template.json file (-json and csv files- folder in Dataset preparation folder) to store panoptic annotations.
3. Access database data (Colony\_picking\_database.csv in -json and csv files- in Dataset preparation folder) for panoptic segments annotations.
4. Define color combinations for panoptic \*.png annotations.
5. Process dataset (train or val) to get panoptic annotations.
   1. Load images and create new folders to store panoptic \*.png annotations.
   2. Call image\_pretreatment.py function (in Dataset preparation folder) to crop images in order to avoid unnecessary regions and get region-based masks and data to be used in next segmentation steps.
   3. Call segmentation\_stage.py function (in Dataset preparation folder) to generate final segmented panoptic masks \*.png annotations.
   4. Store panoptic masks \*.png annotations.
   5. Get panoptic segments annotations and store them in the json\_template. Then save the \*.json annotation file to specified folder.
6. Process dataset (train or val) to get semantic \*.png annotations from panoptic \*.png annotations.
   1. Access panoptic annotations (both \*.png and \*.json files) and create new paths to store semantic annotations.
   2. Define dataset categories and call modified Detectron2 function separate\_semantic\_from\_panoptic.py (in Dataset preparation) to get and store semantic masks \*.png annotations.

Then, PROCESS\_DATASET\_INSTANCE\_SEGMENTATION.py script was executed to generate instances annotations. In this script the steps are:

1. Choose train or val dataset to process and load previously created panoptic \*.json annotations.
2. Then load and set fields to the provided json\_template.json file (-json and csv files- folder in Dataset preparation folder) to store instance annotations.
3. Process dataset (train or val) to get instance annotations from panoptic info.
4. Get instances annotations and store them in the json\_template. Then save the \*.json annotation file to specified folder.

Further explanation on each mentioned function can be found inside its respective script in the Dataset preparation folder.

**DATASETS DESCRIPTION**

COPICK dataset images and annotations files as well as some trial smaller datasets can be found in this repository to download:

REPO url

In the repository, the -Complete dataset (COPICK)- folder contains the images and annotations of the whole colony counting and picking dataset used to generate the final trained model on Detectron2 for bacterial colonies detection in agar plates. The original images from the whole dataset are stored and separated in "train" and "val" folders. The corresponding annotations of both datasets follow the Panoptic Segmentation format for Detectron2 with COCO dataset and PanopticFPN models previously described:

* **\*.json files for both panoptic and instances annotations:** generated with -PROCESS\_DATASET\_PANOPTIC\_SEGMENTATION.py- and -PROCESS\_DATASET\_INSTANCES\_SEGMENTATION.py- scripts respectively (in Dataset preparation folder).
* **\*.png panoptic masks annotations:** generated with -PROCESS\_DATASET\_PANOPTIC\_SEGMENTATION.py- script (in Dataset preparation folder). Created for train and val datasets in -panoptic\_trainval (panoptic annotations)- folder.
* **\*.png semantic mask annotations:** generated from panoptic masks annotations with modified Detectron2 -prepare\_panoptic\_fpn.py- script (inside -PROCESS\_DATASET\_PANOPTIC\_SEGMENTATION.py- script (in Dataset preparation folder). Created for train and val datasets in panoptic\_stuff (semantic annotations) folder.

There is also a "Trial datasets" folder including smaller samples of the whole dataset following the same format as the complete dataset. These datasets are created in order to do faster testing with data. The folder includes:

* **train\_val\_short:** contains 10 training images and 3 validation images and corresponding annotations files.
* **train\_val\_test\_sample:** contains 160 training images and 40 validation images and corresponding annotations files.
* **val\_short:** contains 1 image for inference/validation only and corresponding annotations files.

**DETECTRON2 CONFIGURATION**

Some examples on Detectron2 configuration for training, evaluation and inference can be found in here:

<https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5>

As Panoptic Segmentation is a relatively new task, there is not much available information nor examples on how to set up the configuration for training and evaluation purposes on custom datasets. Here is a list of additional resources that were consulted in order to set up Detectron2 to train with the COPICK dataset:

* **Train Panoptic Segmentation model on custom dataset:**

<https://github.com/facebookresearch/detectron2/issues/1691>

* **Panoptic Segmentation dataset organization:**

<https://github.com/facebookresearch/detectron2/issues/1201#issuecomment-879864483>

* **Dataset registration:**

<https://github.com/facebookresearch/detectron2/issues/327>

* **Load custom datasets for evaluation:**

<https://github.com/facebookresearch/detectron2/issues/552>

Additionally, based on information from the above sites and personal experience, changes were applied to some of the Detectron2 package scripts. Those can be found in -Changes in Detectron2 scripts.txt- file (in Detectron2 folder).

Detectron2 configuration setup scripts for training, evaluation, inference and optimization procedures are provided and detailed down below.

1. **Training and Evaluation**

Detectron2 configuration to train and evaluate the inference model on the COPICK dataset can be found in detectron2\_train\_eval.py script (in Train, evaluation and optimization folder from the Detectron2 folder). In this script, both train and val datasets are first registered with -register\_coco\_panoptic\_separated- function from Detectron2. For the register, corresponding images and annotations files are required. After that, training configuration parameters are set to train the inference model on the COPICK dataset. Then, trained model weights are stored and evaluation parameters are set for evaluation, obtaining Panoptic Segmentation metrics (see <https://cocodataset.org/#panoptic-eval>).

1. **Model Inference**

Model inference can be done in different ways to check model predictions. Additionally to Detectron2 Visualizer function for registered datasets, scripts to get predicted colonies in the images and visualize them were created:

* detectron2\_get\_model\_predictions.py: it is used to get predictions from the trained model and store them. Dataset registration is only necessary if Detectron2 Visualizer function is going to be used. If not, then just the trained model and weights as well as the test parameters are required to set the model predictor for inference. After that, multiple or single images can be processed in order to get and store panoptic predictions.
* display\_colonies\_overlay.py: it is used to process panoptic predictions to get only the colonies footprint and display an overlay of it on top of the original image for visualization purposes.

Both scripts can be found in the -Train, evaluation and optimization- folder (inside Detectron2 folder).

1. **Hyperparameters optimization**

To search for best hyperparameters combinations and improve the colony detection model accuracy, hyperparameter optimization techniques were used. For that purpose, the Optuna library was chosen (<https://optuna.org/>). By default, this library uses Bayesian Optimization algorithms but it can be changed if wanted.

The script -detectron2\_optimization.py- (in Train, evaluation and optimization- folder from Detectron2 folder) is provided to perform hyperparameters optimization. The configuration of parameters values was done following Optuna’s instructions. In this case, the parameters to be optimized were:

* SOLVER.IMS\_PER\_BATCH: defined as number of images per batch across all machines.
* SOLVER.BASE\_LR: learning rate.
* SOLVER.MAX\_ITER: maximum number of iterations.
* MODEL.ROI\_HEADS.BATCH\_SIZE\_PER\_IMAGE: number of regions of interest per image during training.

Further information on Detectron2 configuration parameters can be found in here:

<https://detectron2.readthedocs.io/en/latest/modules/config.html>

Parameters to be optimized had to be included inside a function along with Detectron2 necessary configuration parameters for training and evaluation. This function returns a Panoptic Segmentation metric value SQ (Segmentation Quality) after evaluation to be maximized. The higher the SQ value is, the better the combination of hyperparameters of that optimization iteration are supposed to be to train the colony detection model and get more accurate predictions. Initially, PQ (Panoptic Quality) metric was used for optimization but better results were observed with SQ.

The hyperparameter optimization was performed by executing runs of 6 trials (n\_trials) followed by the addition of the results to a \*.pkl file as described in Optuna library to increase the number of data points in each batch and improve the accuracy of the next prediction. The reason of such procedure was a recurrent memory error that systematically interrupted the execution of larger optimization runs. We hypothesize that the issue is located at some point of inner memory allocation managed by torch libraries required to execute Detectron.

Once optimization was finished, best trained model weights were stored in -Model weights- folder (in Detectron2 folder) to be used for bacterial colonies detection in agar plates with the Opentrons OT-2 robot.

**LIMITATIONS AND ISSUES**

Limitations about use, configuration and performance of the presented inference model are described next:

* Documentation: public documentation provided by both Detectron2 and COCO dataset regarding Panoptic Segmentation was found to be confusing at some points. The lack of information and examples on how to train Panoptic Segmentation models on custom datasets made it difficult to set the final configuration to train and evaluate our model and we ended up using a mix of different sources as mentioned before.
* COPICK dataset: the COPICK dataset was built using the most common conditions for plates preparation in the lab they were collected from. Those are not representative of other conditions and it has to be considered, even though Neural Networks models are very versatile and can discriminate different types of similar objects under variable circumstances. Also, light settings for image acquisition in the dataset are very important for a good image quality and we found our model to be very sensitive to light conditions when changing them inside the Opentrons robot. Finally, we considered the dataset size to be appropriate as a starting point. However, images with previously unincluded information such as different mediums or colonies fluorescence, color, and shape, as well as extra information on colonies size and number (regarding more information of groups of 2 or more colonies too), would be required to increase the dataset data and variability for the model to learn from and be more complete in the future.
* Image segmentation: image segmentation was probably the main problem to tackle when preparing the COPICK dataset annotations. We had to deal with difficult regions such as the border of the plates where the plastic is thicker and both the agar and the colonies often have a different morphology. Moreover, the agar is not a homogeneous medium: it has bubbles, wrinkles or traces of particles that add unspecific noise to the apparent uniformity of the plate. Therefore, light conditions vary in those regions even for the same agar composition. Additionally plates with very small and hundreds of colonies had to be treated differently for a higher contrast of the colonies in the images in order to detect them. For those reasons, we had to consider all of the possible regions and cases to create the segmentation algorithm. We managed to segment as many colonies as possible in the dataset images, including the problematic ones, still the segmentation was not perfect as it is not a trivial task to perform and it conditioned the posterior model training.
* Memory usage: recurrent memory errors were found at some point during the inference model training and evaluation process and also during hyperparameters optimization. For the installation of Detectron2 and Pytorch we could not manage to set it up for GPU usage and therefore had to use only the CPU. This problem, as well as the inner memory allocation of torch libraries mentioned before, were limitations to build the inference model.
* Model inference results: after several training and optimization runs of the model, we observed it was good at segmentation but not that much at identification. This means it was accurate when segmenting every different object in an image, mainly varying on the contours of the segments from one training to another, but it worked worse when assigning a category to those segmented objects, in this case, the colonies. Hence, model predictions have some colonies classified as colonies and the rest only segmented and uncategorized. For our purposes, this is not a problem as we get the segmentation footprint of the colonies in the images anyway and that is enough to identify them and pick them. For the implementation of the colony picking application in the Opentrons, we ended up using a combination of the best trained model weights to get the best segmentation results possible.
* Model metrics: during evaluation, we found Panoptic Segmentation metrics to fail for the stuff categories as every metric value was always 0. We tried to solve this issue but we could not figure out what was causing it. Also, we observed during optimization that larger values of both PQ and SQ metrics were not necessarily correlated with a realistic prediction of colonies in the images at naked eye. As Panoptic Segmentation codes for evaluation task provided by COCO are relatively new and have been only tested on the COCO dataset (check <https://cocodataset.org/#panoptic-eval>), evaluation metrics and configuration might not be working well for our custom dataset and model.

**REFERENCES**

1. Panoptic Segmentation task paper: <https://arxiv.org/abs/1801.00868>