Instruction Guide

Annotating and training an object detector for automated counting of species

IMPORTANT NOTE: Parameters in params.yaml are used throughout the repository.

Table of Contents

1. Definitions

The definitions below will be referenced throughout this document.

1.1. Raw Dataset

- **Project**: Also called a mission, whereby a drone is sent to collect images over a specified terrestrial area, using a push broom or mosaicking pattern. Images are saved in a folder (e.g. `project_name`).
- **Dataset**: A set of PNG or JPG images typically located in a single folder. These images must all have the same dimensions.
- **Overlap**: When using a push broom or mosaic pattern of overhead image collection, the overlap is the fractional overlap between vertically adjacent images.
- **Sidelap**: When using a push broom or mosaic pattern of overhead image collection, the overlap is the fractional overlap between horizontally adjacent images.

1.2. Machine Learning & Image Processing

- **Slices**: Due a combination of the limited memory capacity of processors as well as the relatively small objects of interest (e.g. birds), high-resolution images must be processed in smaller pieces. In this guide we refer to these pieces as slices. Click here for more information.
- **Training Set**: A set of slices that have been labelled with bounding boxes. This will typically consist of a data folder containing the sliced images themselves, and a labels.json file containing the label metadata.
- **Stride**: Object detection algorithms are less performant near the edges of images, therefore allowing a fractional overlap (a.k.a. stride) between slices allows a ML model to improve detection performance. **Note**: trimming the prediction area by the same stride value avoids double-counting.

2. Preparing and Labelling a New Training Set

2.1. Loading a Dataset

The first step to preparing a **Dataset** is to copy a **Project** into the root directory, which is set by default to `./data/raw`. With this pattern, raw images should be located `./data/raw/project_name/` directory.

2.2. Pre-Annotations Using DotDotGoose

Use DotDotGoose software to pre-annotate labels. The output of a DotDotGoose labelling job will be a PNT file with a `.pnt` extension, which must be placed inside the `./data/raw/project_name/` directory along with the raw images. **NOTE**: There must only be one PNT file per project.

2.3. Update `params.yaml`

To process the correct **Project**, the image_height, image_width and project_name fields must be updated to reflect the **Project** attributes. The project_name is simply the name of the folder inside of which the raw images are stored, while the image_height and image_width fields are the dimensions of the original high resolution (raw) images.

2.4. Prepare Data for Labelling

Once a **Dataset** is prepared, including a single PNT file, and the `params.yaml` is updated as described above, the following command should be executed to prepare the dataset for labelling.

```
python3 ./scripts/split_raw_dataset.py
```

2.5. Labelling Data using Label Studio

Following the preparation of a dataset for labelling, follow the instructions below to begin labelling data using Label Studio (a commercially available third party product).

NOTES:

- this repository is only compatible with Label Studio annotation formatting, use of other annotation software will require work to ensure compatibility
- the following steps should be run on a local computerl they are not guaranteed to work on cloud servers

1. Run the following command

```
python3 ./scripts/start_label-studio.py
```

2. Login to Label Studio

3. Load files.txt (located in the root directory) on the label studio GUI
4. Click Create (Project) and name your project
5. Click the Data Import tab*, then Upload Files button
6. Find and select files.txt (in root folder)
7. Once prompted to "Treat CSV/TST as:", Select "List of tasks"
8. Click on Labeling Setup and Remove Airplane and Car labels and add Bird (and/or other labels as required)
9. Click Save and begin labelling (click here for labelling best practices)
10. Once finished labelling, Click Export and select JSON format
11. Click Export and save the output file into `./data/labels/` directory. Update the relevant filename or filepath in `params.yaml` (located in `slices:labels`)

2.6. Prepare Training Set from Labelled Data

Once the steps in Section 2.5 have been followed to completion, first update the following fields in the **training** section of `params.yaml` as follows:

- `train_data_dir`: `./data/training-sets/project_name/data`
- `train_coco`: `./data/training-sets/project_name/labels.json`

Finally, run the following command to prepare the **Training Set**:

```
python3 scripts/prepare_training_set.py
```

2.7. Adding to an Existing Training Set

If all of the previous steps were followed for two projects (let's call them `project_1` and `project_2`), then there should be two different `labels.json` files each in their respective folders, as well as two corresponding `` `training-sets/project_name/data/ `` directories containing the image (slices) data inputs. The following methodology can be used to combine these two **Training Set** (this process can be used to combine any number of **Training Sets**):

1. Create a new folder called `./data/training-sets/combined_project_name/data/`
2. Copy contents of `./data/training-sets/project_1/data/` and `./data/training-sets/project_2/data/` into folder created in Step 1
3. Create an empty JSON file `./data/training-sets/combined_project_name/labels.json`
4. Concatenate the two JSON files from `project_1` and `project_2` and copy the output into the JSON file created in Step 3
5. Update the relevant fields in `params.yaml` (see Section 2.6)

3. Training a Model on a Training Set

Once a **Training Set** has been prepared according to the steps described in Section 2, first update the projectname_ of the following field in the **models** section of `params.yaml`:

- `best_model`: `./models/detection/weights/project_name.pth`

as well as the **training** section of `params.yaml`:

- `train_data_dir`: `./data/training-sets/project_name/data`
- `train_coco`: `./data/training-sets/project_name/labels.json`

Finally, run the following command to start a training job:

```
python3 scripts/train_mdba.py
```

While this should "just work", hyperparameters may be tuned, and data augmentation techniques can be added (see lines 96-107 in `./utils/train_utils.py` regarding the use of `torchvision.transforms`).

4. Counting Species using a Trained Model

4.1 Save Raw Images

Simply follow the instructions in Section 2.1. Keep the project name in mind for the next step.

4.2. Run Bird Counting Script

Update the relevant parameters in `params.yaml` then run the following command (arguments in square brackets are optional):

```
python3 scripts/bird_count.py --project-name project_name

[

*These optional arguments can be set here or in params.yaml

--data-root data_root
--output-folder path_to_output
--conf-thresh confidence_threshold
--overlap overlap_ratio
--sidelap sidelap_ratio
--stride stride_ratio
--model-path model_path
--save-slices save_slices # 0 for False, 1 for True

]
```

where the parameters are defined as follows:

- **--project-name:** This is the parent folder where image dataset is stored.
- **--data-root:** This is root directory inside which the project-folder lies (default: ./data/raw/).
- **--output-folder:** This is the path to store the counts CSV file (default: ./ models/results/).
- **--conf-thresh:** The confidence threshold (default: 0.95) allows for calibration of over- or under-counting.
- **--overlap:** This ratio determines the amount of overlap (default: 0.1) between adjacent whole images to avoid double counting.
- **--sidelap:** This ratio determines the amount of sidelap (default: 0.1) between adjacent whole images to avoid double counting.
- **--stride:** This factor determines the stride (default: 0.5) between adjacent slices during inferencing.
- **--model-path:** This is the path where the inferencing model is located (default: ./models/detection/weights/best.pth).
- **--save-slices:** This optional parameter enables saving visual predictions of non-empty slices (default: 0). Set to '1' to enable.
- WARNING: The **--save-slices** parameter if set to True can easily consume a large amount of storage so it is advised to create a small test folder containing 1 or only very few whole images for testing and calibration purposes.

See an example of a detection output (when save-slices is set to 1 or True).