Camera Trap Classifier – How-To Guide

Step-by-Step Instructions on how to set up, train and use a Model

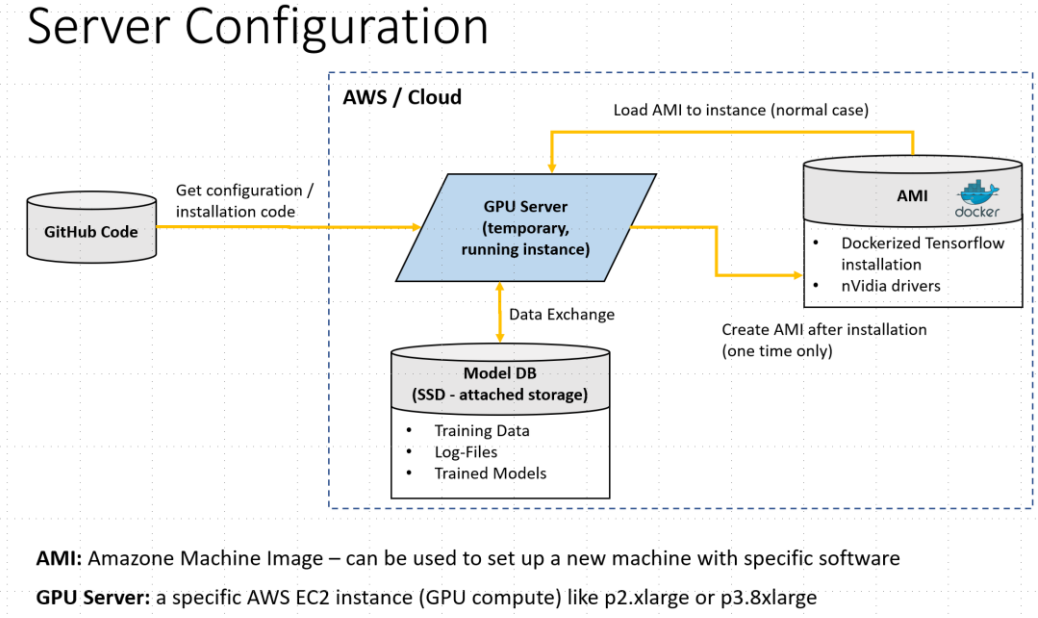
Contents

Prerequisites

1. Camera trap images with labels
2. Access to computer/server with graphics processing units (GPUs) for model training (e.g. AWS account)
3. Some (little) knowledge of Unix

Server Configuration – Example AWS EC2 instance with Docker

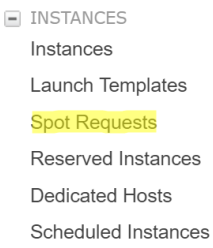
The goal is to set up a server with one or more GPUs that has the necessary software and configuration to run our experiments. See below figure for an overview. The explanations are based on using a Docker image which contains all the necessary software on an AWS ec2 instance. For other services or approaches this guide may not be sufficient.



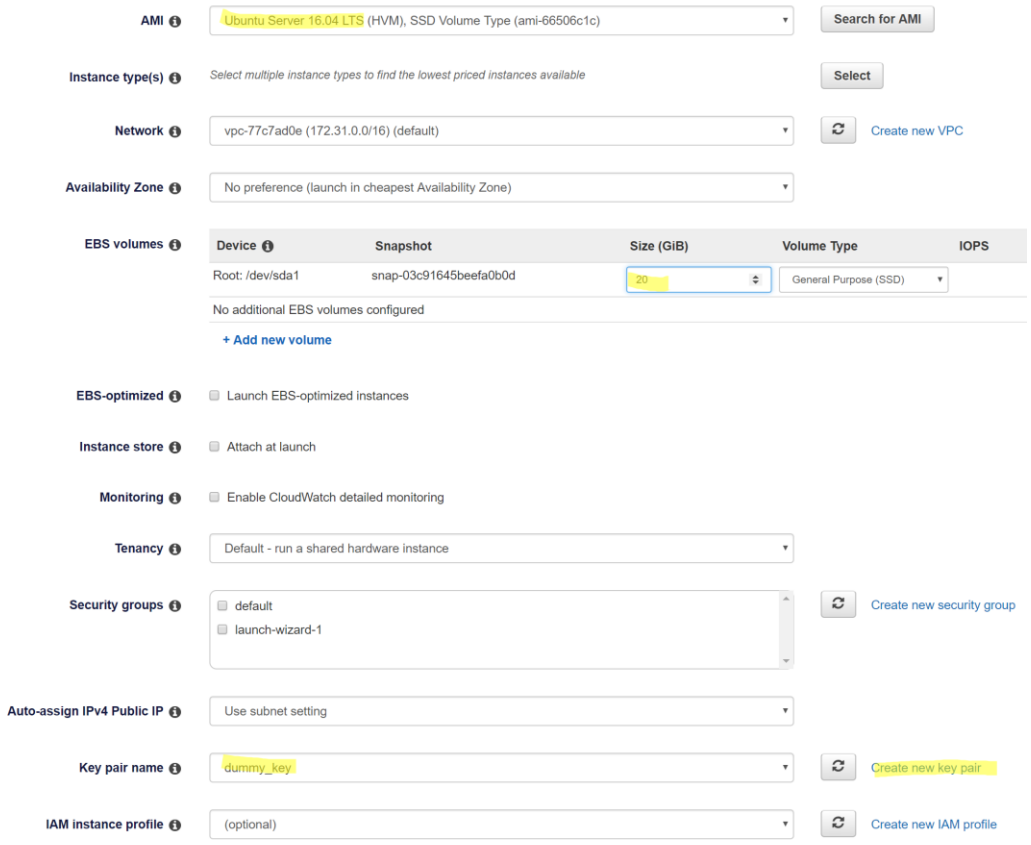
Spin up an AWS server

The following steps show how to spin up an AWS GPU server for you AWS web-interface.

1. Choose spot requests



2. FIRST TIME: Set up an instance with a fresh ubuntu base image, 20GB of disk space and a key to access the instance
3. FOLLOWING TIMES: choose your own Tensorflow AMI

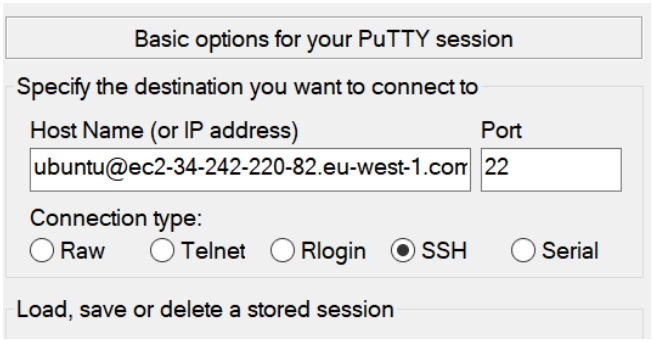


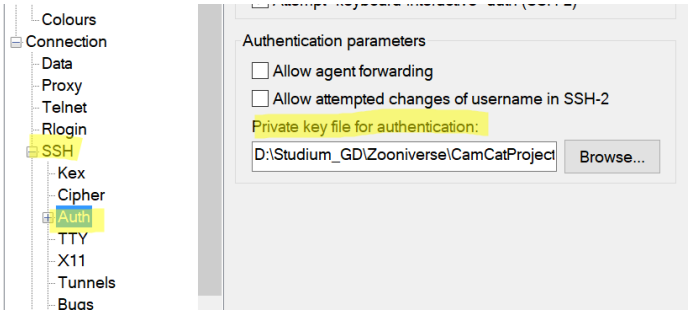
Option 1 – Install everything from scratch

This option allows you to create a base image for yourself (with potentially the newest version of Tensorflow and specifically tailored to AWS instances for improved performance).

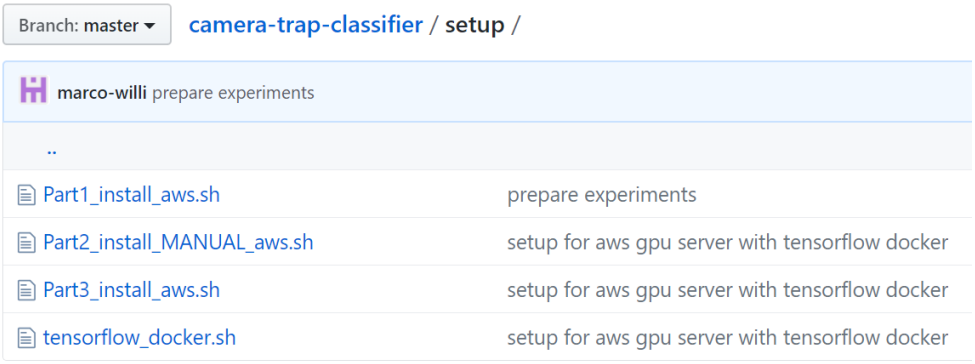
1. Spin up an AWS GPU instance (e.g. p2.xlarge)

2. Connect to your instance – Example with Putty (Windows users)
    a. Convert the key to a format compatible with Putty using puttygen.exe (SO_Link)
    b. Configure Putty with the instance ip and ubuntu as username and keyfile:





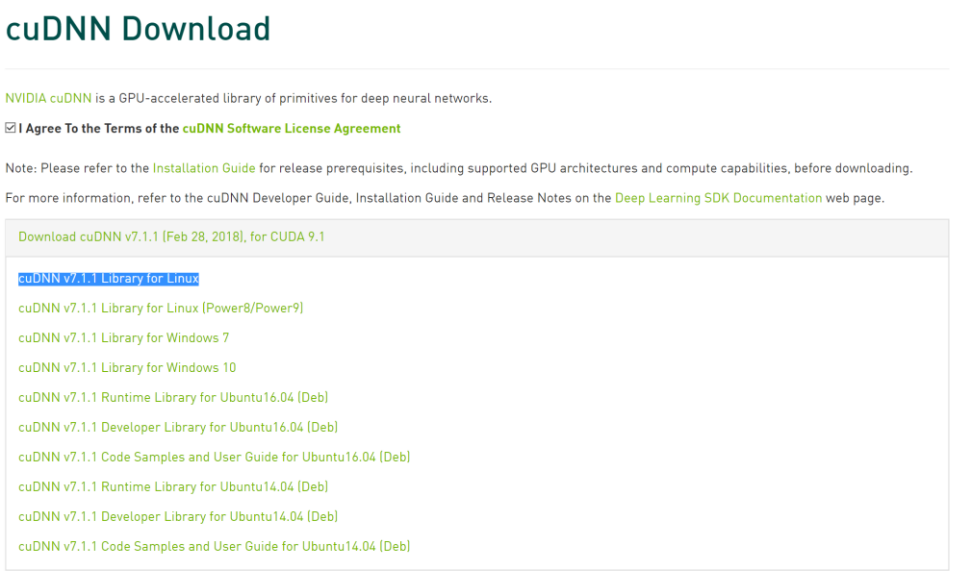
3. Install software on server using the provided code:



   a. Start with Part1 and follow the instructions – copy paste the code to the terminal as there are some manual steps (like uploading files) and reboots involved.

   b. Part2 installs the cuDNN software from nvidia which requires a developer account.

   https://developer.nvidia.com/rdp/cudnn-download



Option 1a – Use official Tensorflow Docker

Just follow the instructions in Part3 and install the official Tensorflow docker image.

Option 1b – Compile Tensorflow on the server

This is slightly more complicated but allows for a tailored installation of Tensorflow on the particular server, thus speeding up CPU operations. It takes a few hours to complete.

Follow the commented out sections in Part3 and use this configuration file:

camera-trap-classifier / setup / tensorflow_docker.sh

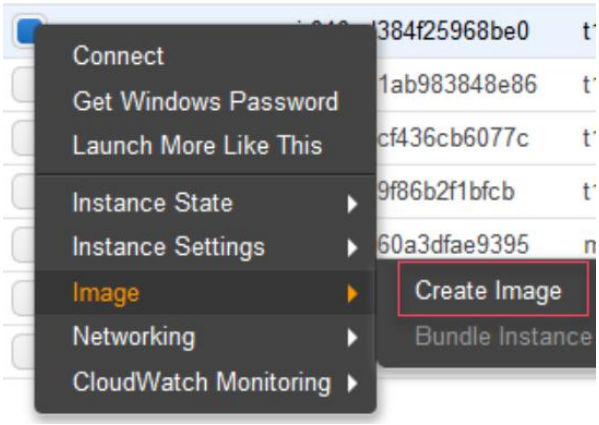
After finishing Part3 we have a completed installation.

Option 2 – Use a pre-made image

Ask the author of this module to provide access to a pre-installed image or use another one on AWS marketplace. This can then be copied to your AWS account.

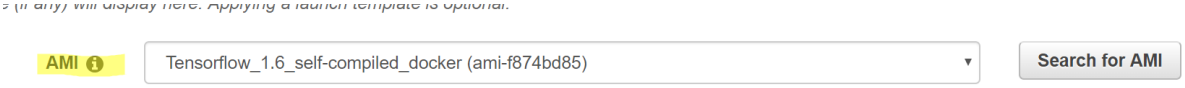
Save the image

Before shutting down the instance make sure to save the image by clicking on the running instance on the AWS web-interface:



Using the new Tensorflow image

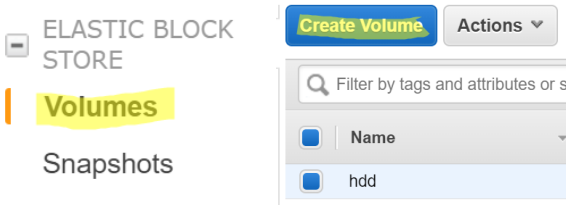
Whenever we start a new GPU server we use our new image instead of a generic Ubuntu base image:



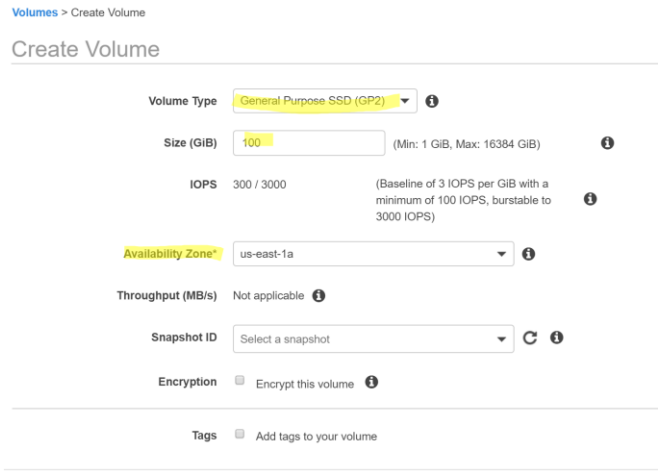
Attaching Permanent Storage to the Server

To save data permanently we create a local volume (SSD) which can be attached to a running instance.

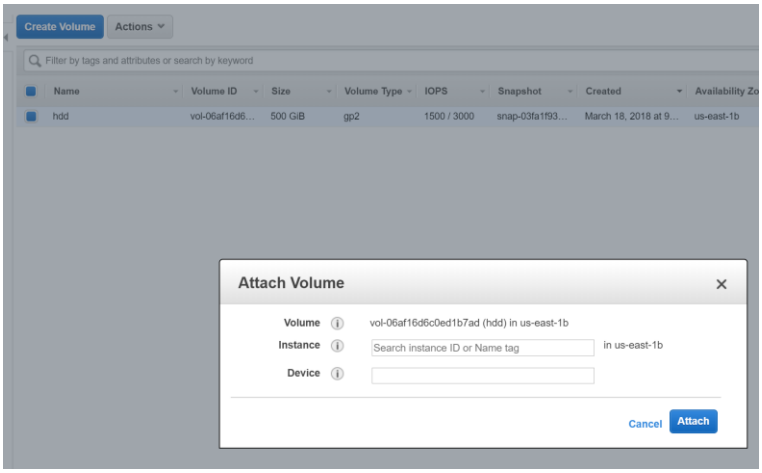
1. Go to volume creation page:



2. Create a volume:
    a. Choose an SSD as it allows for faster training
    b. Choose a size that is appropriate for your dataset size, allow for additional space for saved models, i.e. 50 GB
    c. The availability zone has to match the zone of the running instance, else it cannot be attached. You can also choose to start the instance in the availability zone of the SSD.



3. Attach the volume to a running instance:



Click on Actions and attach the volume. The running instance should be displayed in "Instance".

4. Format and mount the drive using following commands on the instance. Formatting (first two commands has to be done only once), mounting has to be done every time the volume is attached.

```
sudo file -s /dev/xvdf  # only do once for a specific disk

sudo mkfs -t ext4 /dev/xvdf # only do once for a specific disk

mkdir ~/data_hdd

sudo mount /dev/xvdf ~/data_hdd
```

Starting the Container (Docker) containing Tensorflow

1. When we are using a multi-gpu instance run this command once after the instance has been started.
   sudo nvidia-modprobe -u -c=0

2. To start a docker container we use following command:
    a. If we use a self-compiled AMI:

sudo nvidia-docker run -it -v ~/:/host root/tensorflow:latest-devel-gpu-py3 bash

    b. If we use the official Tensorflow docker:

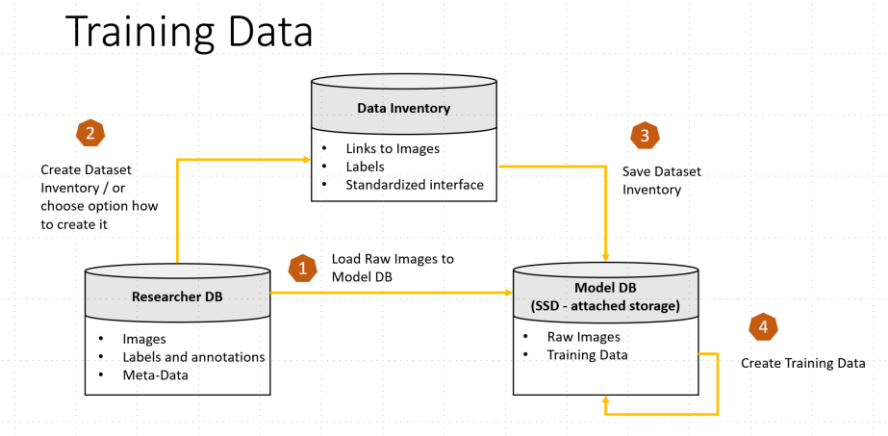
sudo nvidia-docker run -it -v ~/:/host tensorflow/tensorflow:nightly-devel-gpu-py3 bash

3. After executing this command we are inside the container. Type “python3” to check if it works. Type “import tensorflow as tf” to test the tensorflow installation. After that you can type “exit()” to exit python.

Data Preparation

Before being able to train a model, the data has to be prepared such that we can automatically transform it to a format suitable for model training.

The main idea is to create a dataset inventory which contains necessary information about the images like where they are stored and what labels they are associated with. From there on the code automatically processes, transforms, and trains on that data.

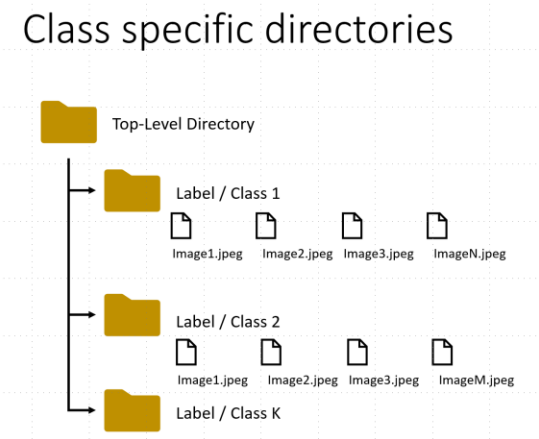


Choosing how to create a dataset inventory

Option 1 – Creating from Class Directories

This is the easiest option with the least flexibility.

1. For each class / label create a directory with the class name containing all the corresponding images for that class.



2. Upload / create this structure on the model DB
3. Specify the corresponding option in the cfg file

Option 2 – Creating from Json file

The json file has to have this structure:

```
{
 "10296725":{
  "labels": {"species": ["cat"],
             "color": ["brown", "white", "black"]
         },
  "images": ["\\images\\4715\\all\\cat\\10296725_0.jpeg",
          "\\images\\4715\\all\\cat\\10296726_0.jpeg",
          "\\images\\4715\\all\\cat\\10296727_0.jpeg"
         ]
 },
 "10296741":{
  "labels": {"species": ["cat"],
             "color": ["black"]},
  "images": ["\\images\\4715\\all\\cat\\10296741_0.jpeg",
          "\\images\\4715\\all\\cat\\10296742_0.jpeg"
         ]
 }}
```

"10296725": a unique id of a record / capture event

"labels": key-list pairs (dictionary) of all the labels of that capture event

{"species": ["cat"]: For each label type (e.g. species or color) store the labels of that particular capture event in a list. In this case the image features a cat with colors brown , white and black.

Option 3 – Custom Data Importer

If you want to create your own data importer, e.g. to create a dataset inventory from a csv file, you can do that by modifying the code in XYZ. There is a working example of a data importer implemented for Pantheras specific csv format.

Place Data Importer into the Code

Adjust the main_train.py code here with your data importer:

logging.info("Building Dataset Inventory")

dataset_inventory = DatasetInventory()

dataset_inventory.create_from_class_directories(cfg.current_paths['inventory'])

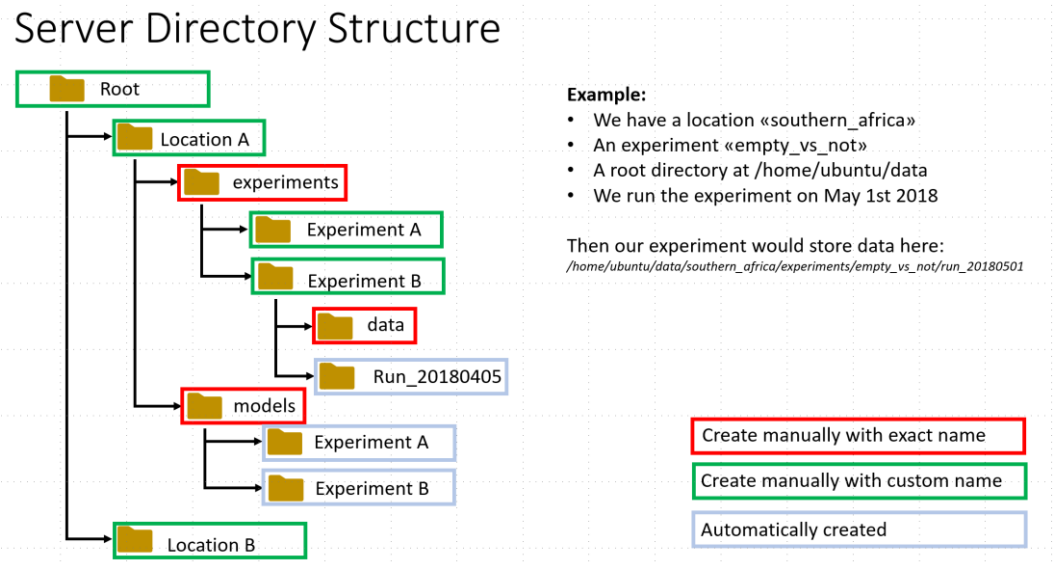
dataset_inventory.label_handler.remove_multi_label_records()

dataset_inventory.log_stats()

Make sure the config.yaml entry of 'inventory' points to the file this function needs. In the above case it would be a directory with class specific image directories.

Directory Structure on Server

The following directory structure has to be created for the model runs. The root directory can be anywhere. The directories in red boxes need exact naming, green boxes have flexible names, and the light blue boxes are automatically created directories.



Training a Model

Pre-Requisites

1. GPU server up and running with attached storage (model db)
2. Raw-data and (potentially) data inventory on model db.
3. Directory structure created.

Configuration File

To run a specific model the configuration file has to be adjusted. The easiest way to do so is to clone the GitHub directory to your local machine.

1. Clone the GitHub directory to your local machine - example unix command:

git clone https://github.com/marco-willi/camera-trap-classifier.git ~/code/camera-trap-classifier

2. Adjust the configuration file :

   config.yaml

3. The configuration file contains options to specify paths, server configurations, and which dataset to run with which model. The options are commented in the config file. The most important parts are:
   a. **general**: specify general options on logging, number of GPUs and CPUs and batch size to use in model training
   b. **paths**: specify the path to the root, experiment and models directories
   c. **locations**: specify the locations (datasets) for which to run a model, these names link to directories in the directory structure
   d. **experiments**: specify experiments in each location and their run configuration
4. Upload the configuration file to your GitHub clone of the repository

Start Model Training

1. Start a GPU server and attach the model DB (SSD / hard drive)
2. Change to the code directory and pull/ clone the repo from GitHub with the adjusted config file
3. Start the Tensorflow docker image
4. Change to the code directory from within the docker image
5. Execute the main file:

   python3 main_train.py > output.log &

6. Check whether the process is running using the “top” command (CPU usage)
7. Tail the log file to check status messages with:
   tail -f output.log

Use a Model on Local Machine

Pre-Requisites

1. Trained model file for prediction - example:

   model_prediction_run_201804060404_incept_res_species.hdf5

2. Model configuration files:

   label_mappings.json, image_processing.json

3. JPEG/JPG images in a directory (can include subdirectories with images)
4. GitHub code of the repo

Installing Python

To apply the models on a local machine the correct python installation must be available. It is strongly recommended to create a virtual environment for this task and install all dependencies to it.

Some background information if needed:

https://packaging.python.org/guides/installing-using-pip-and-virtualenv/

The easiest way is to follow the instructions here:

https://www.tensorflow.org/install/

The package was developed with Tensorflow 1.6 and Python 3.5. Python 3.4 works, but Python 3.6 may not.

Additionally, following modules have to be installed using the requirements.txt:

pip install -r requirements.txt

For windows users with Anaconda – there are instructions on the GitHub repository

setup/install_conda_windows.txt

Applying the model

To run the model a command line program can be used. Meaning, we specify a command with all the options and execute it using python 3. The program will predict a directory of images and write results to a csv file.

1. Prepare the command to execute the program:

   Arguments:

   -**image_dir**: path to root of image directory, can contain subdirectories

   with images, the program will search for all images and predict them

   - **results_file**: path to the file to store the predictions in

   -**model_path**: path to the model to use (hdf5 file)

   -**class_mapping_json**: path to 'label_mappings.json'

   -**pre_processing_json**: path to the image_processing.json

   -**export_file_type** (optional, default csv): currently only csv

   -**batch_size** (optional, default 128): the number of images once at a time

   to predict before writing results to disk

   -**check_images** (optional, default 0): whether to check all images for

corruption before starting the training (most likely not necessary)

Example:

```
python3 main_prediction.py -image_dir /user/images/ \
-results_file /user/predictions/output.csv \
-model_path /user/models/my_super_model.hdf5 \
-class_mapping_json /user/models/label_mappings.json \
-pre_processing_json /user/models/image_processing.json
```

2. Activate the virtual environment with the relevant python installation.
3. Execute the command above.

Feature Wishlist

- Create and export predictions for test data after model training as part of the whole model training process
- Integrate label mappings into the config file instead of a code file
- Provide option to calculate multi-crop predictions
- Improve specification of how to create dataset inventory, e.g. move to config file instead of main code file

Open Issues

Reading TFRecord with slow speed for the first time

It can happen that reading the training data as TFRecord can be very slow when done for the first time. This is a bug and has not yet been resolved. The solution is to simply wait for that slow initial read to finish (CPU usage at approx. 20%).

Tipps and Tricks

Transferring files between instances

To transfer files the target directory has to have the appropriate permission:

sudo chmod -R 777 dir_to_transfer_files_to

GPU run out of memory vs Batch Sizes

Maximum batch sizes depend on how much GPU memory is available. The larger the batch size, the larger the model itself, the more GPU memory is required. If not enough GPU memory is available Tensorflow will throw an error like this (and additional messages):

```
Resource exhausted: OOM when allocating tensor with shape[128,1
```

The solution is to simply reduce batch size.