

Astroeco ML Guide

Login and setup

- This document is a short guide to the use of the astroecology ML software developed by Ross McWhirter during March 2020 and how to access data and resources on Mimir. This software can be used to automate much of the darknet machine learning training and evaluation.
- Users must login to the astroecology server 'Mimir' through the ARI external network.
- This is accomplished using the NoMachine software. Install this software and then create a connection to external.astro.ljmu.ac.uk using the username and password supplied by the ARI IT staff.
- Open a virtual desktop which will be the means to connect to Mimir. If you have an issue here (error message saying XFCE desktop cannot be created), please contact the ARI IT staff using ast-support@ljmu.ac.uk.
- Once connected to the virtual desktop, you must log into the Mimir server using the astroecology usernames provided.
- The next required step is to open a terminal window, this can be accomplished by clicking the black icon containing a dollar sign at the bottom of the screen. All actions will be conducted within the terminal window.
- The login to Mimir is accomplished by typing 'ssh 150.204.240.212 -l YOURUSERNAME -X'. You will be prompted for your password. If this is your first time connecting, it is strongly recommended you change your password once logged into Mimir using the 'passwd' command.
- Once you are logged in you may access the folders you are entitled to. You can type the command 'thunar' to open a file manager GUI for browsing.
- Standard users will have read only access to the /media/astroecology/raid drive which holds our data. The main astroecology data is held in the /media/astroecology/raid/ourdata folder.
- Users will have read and write access to the scratch disk at /media/astroecology/scratch and their own home folder at /home/USERNAME/. Both drives can be used to store datasets for ML which requires read/write access for the Astroeco ML python script.
- Your home directory is visible only to you whereas the scratch drive is visible to all users.
- There is also the 'astroecology' main account. This account has been setup with all the required software and full read/write access. Standard users have read only access to the /home/astroecology folder and subfolders EXCEPT for the darknet and yolov3 (ultralytics) folders required for machine learning.
- If full superuser access is required this must be done through the 'astroecology' user account. Login information can be provided if needed but make sure this is absolutely required as this account has read/write access to the full data.
- Standard XFCE GUI programs such as thunar and gedit may be launched from the command line by simply typing their names as the command. Both firefox and google chrome are installed as browsers and can be launched by typing 'firefox' or 'google-chrome'.

- If multiple programs are required, you can open additional terminal windows or tabs of one window within the File dropdown at the corner of the window. Each additional terminal window will need to be logged into Mimir using the method described above.

Running the astroecology ML software

- First the correct conda environment must be activated. For this software it is simply just the (base) environment. This should automatically be the active environment. If not, the environment can be activated by typing 'conda activate base'. In the event that you must deactivate the environment, type 'conda deactivate'.
- Next, you must copy the astroeco_ml folder from the /home/astroecology/Documents folder into your own home directory using the command 'cp -r /home/astroecology/Documents/astroeco_ml /home/YOURUSERNAME/'.
- Use the cd command to navigate to the astroeco_ml folder by typing 'cd /home/YOURUSERNAME/astroeco_ml'.
- This directory contains the required python scripts as well as some required supplementary files. You must always launch the astroecology ML scripts from this directory.
- You can launch the python program by simply typing 'python astroeco_ml.py'.
- Executing the above command will pop up a list of accepted inputs for this program of which multiple are required to execute a number of possible tasks.
- This software is capable of training object detection models using the darknet library, evaluating detection models using darknet on validation and testing sets and running detection on a new folder of images.
- In the next two subsections, the inputs and outputs of this software (as shown when you ran the above command) are defined.
- A full copy of this software is stored on the cloud at:
https://github.com/P-R-McWhirter/astroeco_ml/

Astroeco_ML software inputs

- The following options are input commands for the astroecology ML script and are input using a double hyphen followed by the name of the input (with no space) then a space and the value.
- For example: 'python astroeco_ml.py --gpu_id 1' which is the command to select GPU 1 (the second GPU) for training operations.
- If an argument is missing it may result in the program exiting during runtime. The error message will indicate which argument is missing so that the user may fix this.
- This software will NOT delete any files or folders it does not also create. If rerunning, for example, augmentation with new settings, it is recommended to delete the old augmentations folder inside the training data directory before proceeding.
- **INPUT COMMANDS :**
- --darknet : A path to the darknet folder. If running this code from Mimir, the correct path to the astroecology Documents directory version is automatically provided and is not required.

- `--ultra` : A path to the ultralytics (yolov3) folder. As with the darknet option, if running this code from Mimir, the correct path is provided and is not required.
- `--mode` : A required input. This indicates which mode you are running the script in. Type '`--mode`' train if you wish to train a new model, eval to run the evaluation scripts on an existing model or detect to run inference on a folder of new images.
- The next commands are separated into which mode they are under however some of these inputs may be required for more than one mode.

• **TRAINING ARGUMENTS :**

- `--train_folder` : The directory path to the folder containing the training data in the darknet format. This folder contains either jpg or png images along with txt files for the labels. Deeplabel outputs this format when exporting using the darknet command (consult Deeplabel documentation).
- `--val_folder` : The directory path to the folder containing the validation data in the darknet format. See Deeplabel comments above.
- `--model_type` : The type of YOLO computer vision model you wish to train. There are four input options, 'yolov3' for a basic YOLOv3, 'yolov3-spp' for YOLOv3 with Spatial Pyramid Pooling, 'yolov3-tiny' for Tiny YOLOv3 and 'yolov3-tiny-3l' a special variant of Tiny YOLOv3 with three detection layers (for smaller objects).
- `--gpu_id` : Which GPU to use for training (starting from 0). There are only two in Mimir so this can only be 0 or 1. Evaluation and detection only use GPU 0. If left empty, will default to GPU 0.
- `--subdivisions` : Models are trained with a batch size of 64 but this often is still too big for the GPU memory. Darknet can subdivide the batch even more at a cost of slower speed. The lower the better, trial values of subdivisions starting at 1, then 2, 4, 8, etc. Must be a sub-multiple of 64. If ignored, will default to 16 which will work for most astroecology datasets, but may not be the fastest at training.
- `--model_dim` : The dimensions of the input 'image' for the training. The larger this value is the longer it will take to train. Defaults to 416×416 and must be a multiple of 32. It is recommended to use a value similar to the largest axis of the input images. The aspect ratio will be maintained and empty pixels will be padded with zeros.
- `--num_of_epochs` : An epoch is the amount of training iterations required to complete one full pass over the training data. The more epochs trained for, the longer the training takes but the better the opportunity for the model to converge. Recommended to start with 100 and increase if required.
- `--data_folder` : Required, a path to a directory (existing or not existing) in which runtime data can be written. This folder MUST have read/write access. Recommend using a directory inside either the user home directory or the scratch drive. **Be careful not to over-write a previous run.**
- `--model_folder` : Output folder for the trained models saved every 1000 iterations. As with the above directory, this folder MUST have read/write access as well as sufficient storage space for the models. **Be careful not to over-write a previous run.**
- `--altaug` : A flag (does not require a value after it) to select that you wish to augment the height of your training data. Requires the `--init_height` and `--new_heights` arguments. Can be run with `--rotaug`.

- `--rotaug` : Another flag to select that you wish to augment the rotation of your training data. Requires the `--rot_angle` argument. Can be run with `--altaug`.
- `--init_height` : Input for the initial height of the altitude augmentation (the height the data was collected at).
- `--new_heights` : A list of numbers (input like `--new_heights 100 200 300` etc) which contain the new heights for the augmented data. **BE CAREFUL: THIS CAN TAKE UP ALOT OF SPACE IF HEAVILY AUGMENTED.**
- `--rot_angle` : The angle in which to incrementally rotate the data in degrees between 0 and 360. For example, 45 will turn one training image into 8 images at 0, 45, 90, 135, 180, 225, 270 and 315 degrees. **BE SUPER EXTRA CAREFUL: THIS CAN TAKE UP AN INSANE AMOUNT OF SPACE IF HEAVILY AUGMENTED (small value input).**
- `--blobaug` : A flag to select that you wish to populated your training, validation and optionally testing sets with Gaussian blobs simulating small animals. These new images will go into `blobaug` subfolders in the training, validation and testing directories.
- `--num_blobs` : Maximum number of blobs to place in each image.
- `--blob_minsize` : Minimum size of any of the Gaussian blobs (must be odd).
- `--blob_maxsize` : Maximize size of any of the Gaussian blobs (must be odd).
- `--blob_amplitude` : Brightness of the central pixel of the Gaussian blob.
- `--blob_skew_x` : Skewness of the Gaussian blobs in the x axis.
- `--blob_skew_y` : Skewness of the Gaussian blobs in the y axis.
- `--blob_prob` : Probability of adding a blob to an image (up to the maximum number). Defaults to 0.5 (50%).
- `--blob_class_id` : ID from `obj.names` file for the Gaussian blobs. It is not possible to automatically assign different class IDs to blobs in this version.

• **EVALUATION ARGUMENTS :**

- `--test_folder` : Directory path to the folder containing the testing set images and labels in darknet format. Note: `--val_folder` also required for evaluation despite listed under training arguments.
- `--prefix` : A string prefix to identify the models in a model folder for evaluation (in case multiple models using a different configuration are present). For example providing `yolov3` will mean only `.weights` model files with `yolov3` at the start are selected. Note: for string input it is not necessary to use single or double quotes.
- `--iou_thresh` : The IoU threshold to use for the calculation of the mAP statistic. Takes values between 0 and 1. The usual mAP values used are `mAP-50 = 0.5` and `mAP-75 = 0.75`. This threshold determines what IoU (Intersection over Union) is required between ground truth bounding boxes and model predicted bounding boxes to consider the detection a correct one.

• **DETECTION ARGUMENTS :**

- `--cfg` : File path of the configuration file of the inference model. Will have a filetype of `.cfg`. If a model trained using this script, it will likely be in the data folder with a name based on the model type selected and a `.cfg` extension.

- `--weights` : File path of the weights model file used for the inference. Will have a filetype of `.weights` and if trained using this script will be in the model folder.
- `--obj_names` : File path to the `obj.names` file for the model inference. This file is a text file with the names of the classes on new lines. This file is required for Deeplabel export and is likely not far from the training and validation folders.
- `--input` : Directory path to a folder containing images to be classified using the inference model. This folder does not need to contain labels (for obvious reasons) and will be labelled by the inference.
- `--conf_thres` : A probability between 0 and 1 defining the confidence required for a prediction before it will be drawn on the image. This can vary depending on the confidence of the model. If it is too high you will not get any labels, too low and you will get many false positives. Do not set it to zero, you will probably blow the program up. Starting with 0.5 is a good idea and adjust from there.
- `--make_video` : If the input folder contains a thermal flight extracted by flirpy in the form `frame_000000.jpg`, `frame_000001.jpg` and so on, you can supply this flag argument to trigger the production of a video from the resulting predicted images. This can be used to make nice videos for outreach but other than that is pretty pointless!
- `--video_framerate` : Value for the framerate of the output video, only needed if the `--make_video` flag is used. For our recent datasets on the FLIR Duo camera this is usually 30 for 30 fps.
- `--output` : Directory path to a folder (existing or not existing) which will contain the classified and labelled output images. Again, be careful and make sure there is sufficient file space for this output. Pretty easy to predict the output folder size as it will be very similar to the input folder size.

Astroeco_ML software outputs

- This software outputs all required information into three folders, all supplied by the user as arguments: The '`--data_folder`', 'the `--model_folder`' and the '`--output`' directories.
- **TRAINING OUTPUTS :**
 - During the 'train' mode, the data folder will output text files containing training and validation file paths and `.data` files containing links to the model folder and the input training and validation file path text files. These are not needed by the user and can be ignored.
 - The configuration file for the training will also be output into the data folder named after the model type selected with the file extension '`.cfg`'. This configuration file is important as it will be needed for both the 'eval' and 'detect' modes using the `--cfg` command.
 - The model folder will also contain a set of `.weights` files. These are the models saved after 1000 iterations. Keep all these models, not just the final one as they will be important when used with the 'eval' mode to determine the model performance (and if later iterations are over-fitting).
 - The final file in this folder is an image named 'chart' which contains a plot of the training loss against iteration number. This is a good indicator that the model is training if it is dropping over time however is not an indicator of model performance.

• **EVALUATION OUTPUTS :**

- The evaluation mode will place two files, `map_output_val.csv` and `map_output_test.csv` into the data folder. These files contain raw unformatted output from darknet and are most likely not useful to the user, but may contain additional information if required.
- The main outputs of this mode are put into a subdirectory in the data folder named `eval_results`. It is recommended that you clear this folder before every evaluation mode execution.
- In this folder there are $2(N + 1)$ table files where N is the number of classes in the model. Each class has its own statistics file combined with an overall statistics file. This is applied to both the validation set and the testing set.
- Each class evaluation file contains as columns: Model file name, model file number (multiples of 1000 with a final and a last), The name of the class (shared by file name), the ID number of the class (starting at 0, contained in `obj.names` folder), the True Positive number, the False Positive number and the False Negative number and finally the accuracy percentage of that class from that model. The suffix of the file will be either `_val` or `_test` to indicate which dataset was used.
- The overall evaluation files contain as columns: Model file name, model file number (multiples of 1000 with a final and a last), overall Precision, overall Recall, overall F1-score, overall True Positive number, overall False Positive number and overall False Negative number, the average IoU of the predicted bounding boxes (in percentage) and finally the mAP value (in percentage) based on the IoU threshold provided (see `--iou_thresh` above).

• **DETECTION OUTPUTS :**

- The detection mode outputs all classified images into the `--output` folder.
- If the `--make_video` flag is used, a video will be produced with the name `output.mp4` within the `astroeco_ml` folder. **Be careful, this video will overwrite an existing output.mp4 in this folder.**

Runtime example

- For users of the Mimir computer, here is a toy example to show how to train and then evaluate a model in two commands to the `astroeco_ml.py` file.
- First, log into your ARI account with NoMachine and ssh into Mimir.
- In the `/home/astroecology/Documents` directory there are two directories named `astroeco_ml` and `data_arribada_cv_final`. Copy these two folders to your own home directory.
- This can be accomplished using either the `cp` command or running the thunar file browser by typing `thunar` in the command line and pressing enter.
- An example of these two folders in a user directory is shown in figure 1.
- It is important to note that despite this being a 'quick example', this approach will be similar for any dataset you may run in the future.
- First let's create a quick model. This will take between 5-10 minutes for GPU 0 on the server.
- Use the `cd` command to navigate into your copy of the `astroeco_ml` directory.

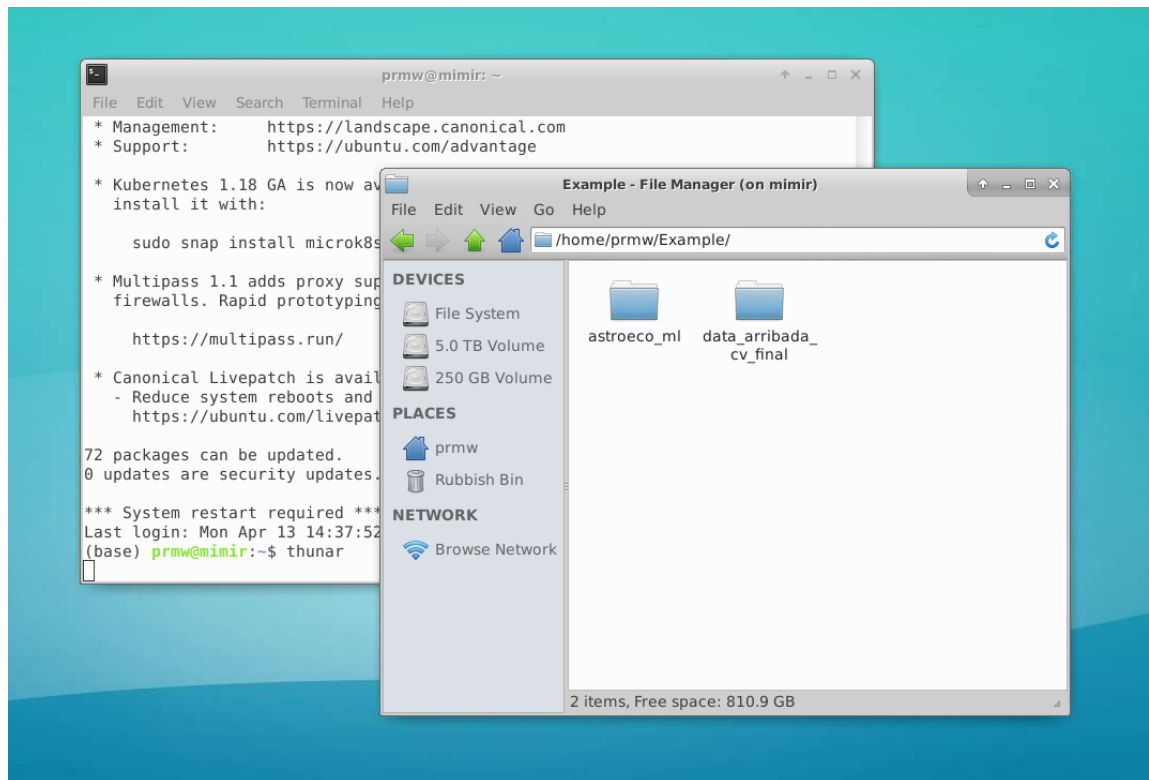


Figure 1: An example of the program folder and the data folder in a user directory.

- Then type: `python astroeco_ml.py --mode train --model_type yolov3-tiny-3l --obj_names ../data_arribada_cv_final/obj.names --train_folder ../data_arribada_cv_final/train/ --val_folder ../data_arribada_cv_final/val/ --model_folder model_test/ --num_of_epochs 50 --data_folder data_test/ --model_dim 160 --subdivisions 1`
- Then hit enter! Figure 2 shows an example of what this command looks like typed into the terminal window whilst in the `astroeco_ml` folder.
- You should see the program start and then a plot will appear showing the current training iteration and plotting new iterations as they finish. An example of this is shown in figure 3.
- A quick explanation of the inputs we used (detailed above):
- '--mode train' selects the training mode as we want to make a model.
- '--model_type yolov3-tiny-3l' selects the 3-layer tiny YOLO, this is a nice small model which trains quickly but is actually very powerful.

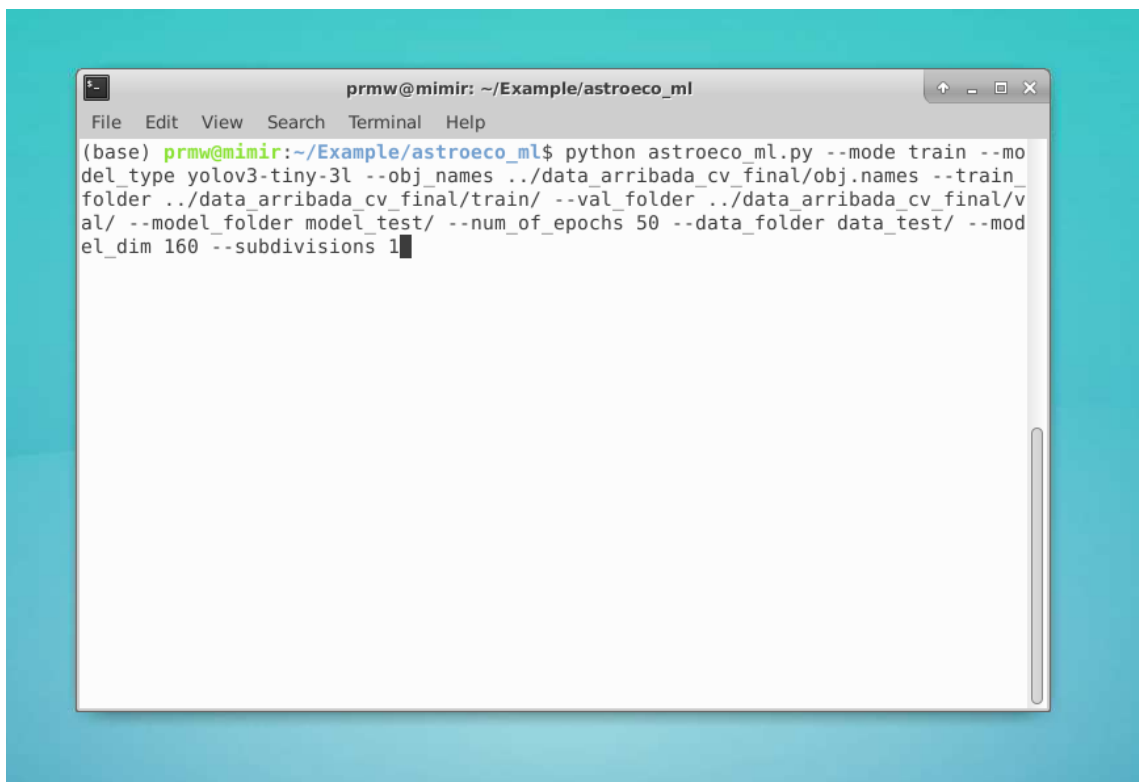


Figure 2: An example of the training command typed into the terminal window.

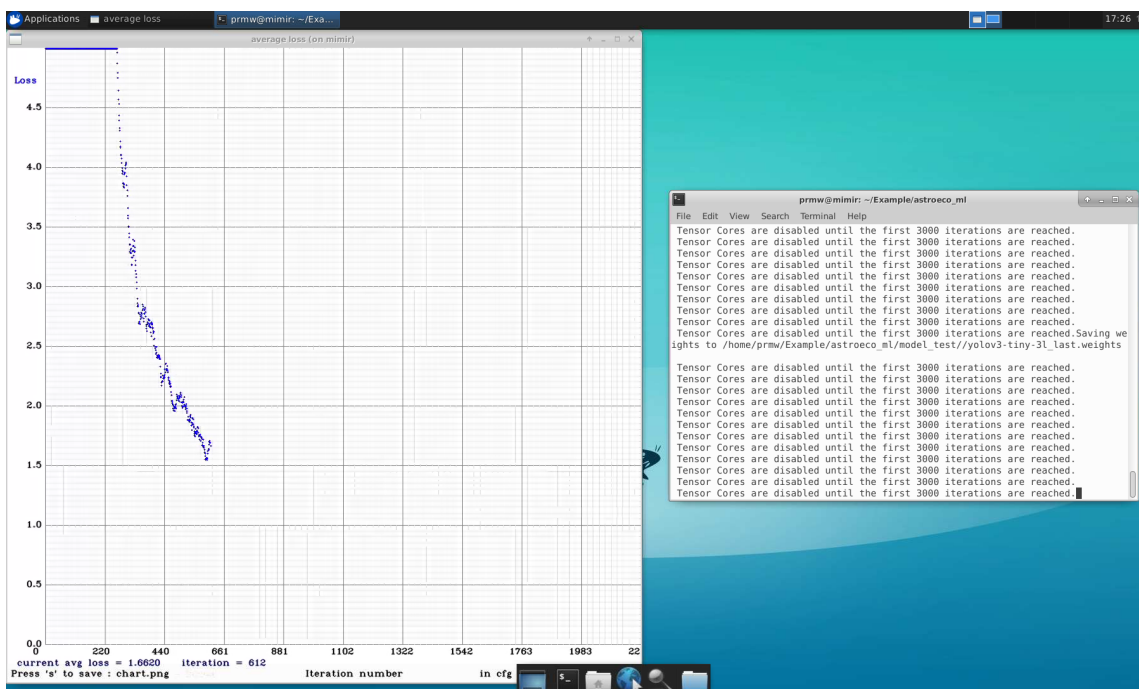
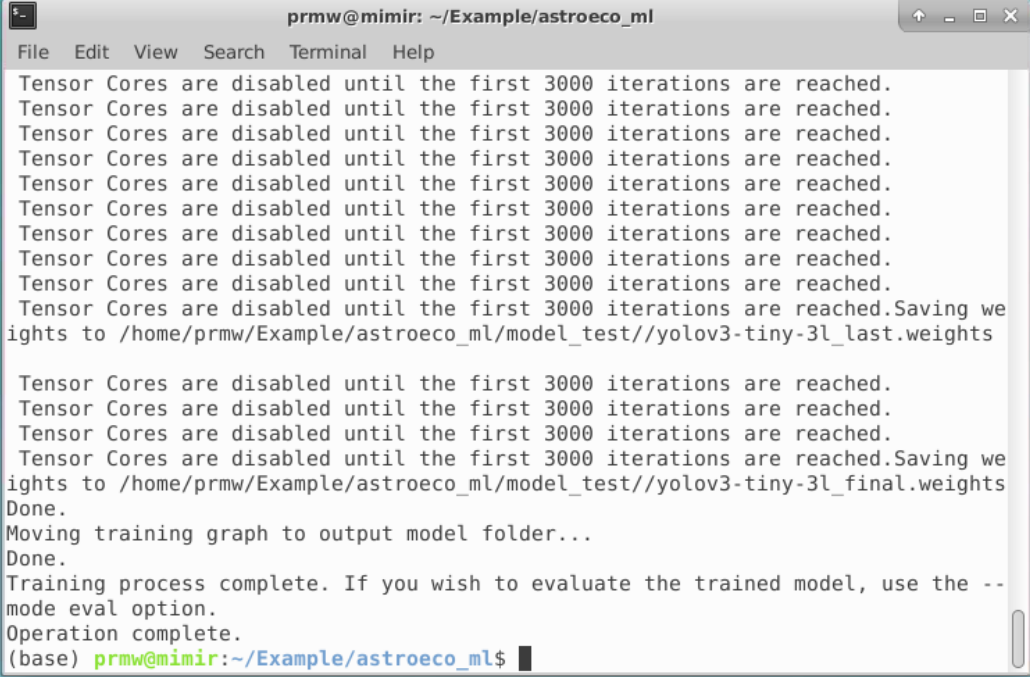


Figure 3: An example of the training mode running in the terminal window.

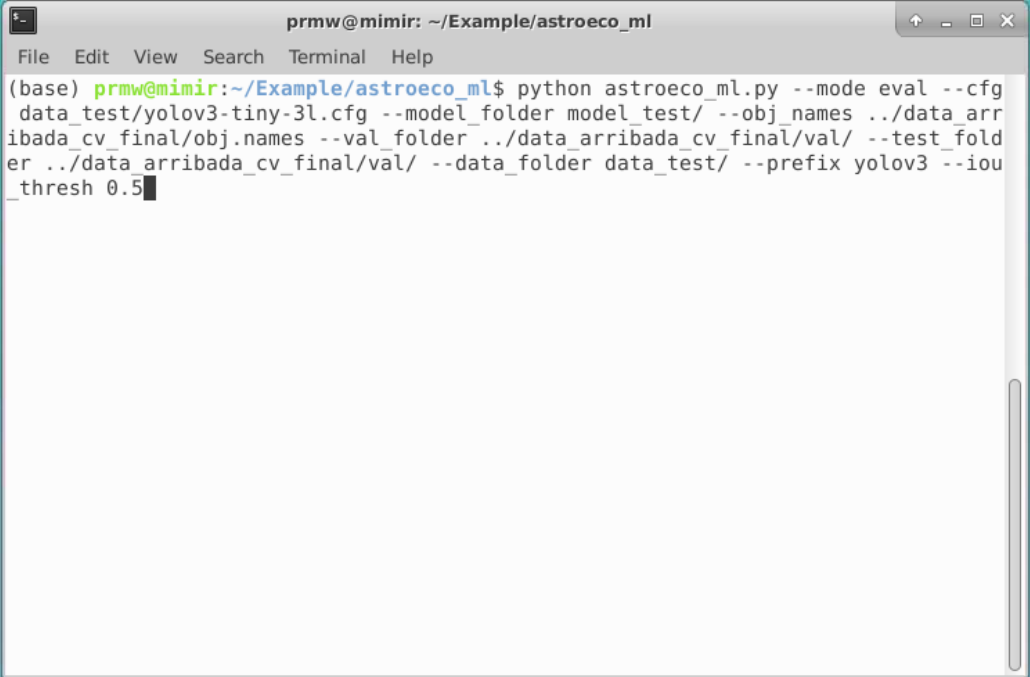
- ‘--obj_names ../data_arribada_cv_final/obj.names’: this selects the obj.names file for our example dataset, the ../ means go up one folder (as astroeco_ml and data_arribada_cv_final are currently in the same directory).
- ‘--train_’ folder and ‘--val_’ folder select the training and val directories in the data_arribada_cv_final directory. If you look inside these folders you will see .png images with .txt label files.
- ‘--model_folder’ and ‘--data_folder’ are directories which hold our model and some configuration files. We just set these to model_test and data_test which appear in the astroeco_ml directory.
- ‘--num_of_epochs 50’ trains the model for 50 epochs, this takes about 10 minutes and is just long enough to see the model learning to detect elephants and humans.
- ‘--model_dim 160’: Normally YOLOv3 resizes the input images to 416x416 pixels but it can cope with any multiple of 32. As our example has 160x120 images, we can resize to 160x160 without a loss in resolution. This helps speed up the training of the model. (608x608 works well for our drone data).
- ‘--subdivisions 1’: If the data cannot fit in the computer memory, we can subdivide it allowing us to train but slowing the training down slightly. As this data is so small we don’t need to subdivide so we set this to 1. Can take values of 1,2,4,8,16,32,64. Usually 8 or 16 is required for our drone data at 608x608.
- Once the model is trained you should see the program output two final lines reading : ‘Training process complete. If you wish to evaluate the trained model, use the --mode eval option. Operation complete.’
- Figure 4 shows an example of this program complete output.
- There should now be 5 files in the model_test folder located in the astroeco_ml directory: chart.png, an image of the plot you saw previously and four models (same model, saved at different points), one at 1000 iterations, one at 2000 and a final and last model. Figure 5 shows these files.
- You will notice that we did not make use of any augmentation in this example. This can be applied by adding the --rotaug and/or the --altaug arguments at the end of the above command along with their required inputs as defined in this document.
- Now we can run the script again but with the evaluation mode, to get some statistics on the performance the model we just trained.
- While still in the astroeco_ml folder, type the following command to call astroeco_ml.py again to do this. Again, this will be similar for most other datasets you may wish to run:
- `python astroeco_ml.py --mode eval --cfg data_test/yolov3-tiny-3l.cfg --model_folder model_test/ --obj_names ../data_arribada_cv_final/obj.names --val_folder ../data_arribada_cv_final/val/ --test_folder ../data_arribada_cv_final/val/ --data_folder data_test/ --prefix yolov3 --iou_thresh 0.5`
- hit enter again. The program will display multiple numbers on the command line and should take about 1 minute to finish.
- Figure 6 shows this command in the terminal window.
- Another quick explanation for the input arguments we used:
- ‘--mode eval’ selects the evaluation mode as we want to test a set of models (the ones we just trained in this case).



```
prmw@mimir: ~/Example/astroeco_ml
File Edit View Search Terminal Help
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.Saving weights to /home/prmw/Example/astroeco_ml/model_test//yolov3-tiny-3l_last.weights

Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.
Tensor Cores are disabled until the first 3000 iterations are reached.Saving weights to /home/prmw/Example/astroeco_ml/model_test//yolov3-tiny-3l_final.weights
Done.
Moving training graph to output model folder...
Done.
Training process complete. If you wish to evaluate the trained model, use the --mode eval option.
Operation complete.
(base) prmw@mimir:~/Example/astroeco_ml$
```

Figure 4: An example of the output of the completed training mode in the terminal window.



A terminal window titled "prm@mimir: ~/Example/astroeco_ml" with a menu bar (File, Edit, View, Search, Terminal, Help). The command entered is: `(base) prm@mimir:~/Example/astroeco_ml$ python astroeco_ml.py --mode eval --cfg data_test/yolov3-tiny-3l.cfg --model_folder model_test/ --obj_names ../data_arribada_cv_final/obj.names --val_folder ../data_arribada_cv_final/val/ --test_folder ../data_arribada_cv_final/val/ --data_folder data_test/ --prefix yolov3 --iou_thresh 0.5`

Figure 6: An example of the evaluation command typed into the terminal window.

- ‘--cfg data_test/yolov3-tiny-3l.cfg’ : We need to supply the configuration file of the models we just trained which was automatically created when we ran the training mode. This is located in the data_folder we supplied previously (data_test) and has a name of yolov3-tiny-3l.cfg as we selected this model type.
- ‘--model_folder model_test/’ : We need to provide the folder containing the models we want to test. This was the model_test folder so we input that.
- ‘--obj_names ../data_arribada_cv_final/obj.names’ : We need to provide the same obj.names file as we used for training, so this is identical to the training mode command.
- ‘--val_folder’ and ‘--test_folder’ : Okay... The code assumes we are good data scientists and have made a training, validation and test split of our data. However, we do not have a testing set for this example data. We will put in the validation folder twice just so the code will run. For an actual project, make sure to have a test split.
- ‘--data_folder data_test/’ : We need to supply a folder for the results of our evaluation. Give it the data_test folder we used before as it will place an eval_results directory into this folder containing the output. There is also some ‘half-way-point’ runtime files which get dropped in here so might as well have it alongside the junk we produced for training!
- ‘--prefix yolov3’ : Some folders can contain models from multiple different model_types. This lets us supply a prefix match for the model filenames so we only evaluate appropriate models (it will crash if there is a mismatch between the .weights model files and the .cfg file). We just put yolov3 for the moment as all the models start with that in this example.
- ‘--iou_thresh 0.5’ : We need to supply a value which lets us know what amount of overlap between the model predicted bounding box and the ground truth box is required for us to consider the box a good match. This is a value between 0 and 1 indicating this overlap. Setting it to 0.5 indicates that any predicted box which has more than 50% of its area intersecting (overlapping) with the ground truth box relative to the union (combination) of their areas. Commonly papers report this as either 0.5 or 0.75 (mAP50 and mAP75). The value used has implications for the final two columns in the ‘overall’ results files from this run.
- If the program completes successfully you should see the messages: ‘These statistics can be found in a directory names ‘eval_results’ inside the chosen data output directory. Operation complete.’
- Figure 7 shows this output in the terminal window.
- As our chosen data output directory was selected by --data_folder we can go take a look at it by either using the cd and ls commands to navigate to the folder or just typing thunar and using the GUI file manager to navigate to the eval_results folder inside data_test.
- Figure 8 shows the eval_results folder from this operation open in thunar.
- We should see 8 files. Results on the val and test data (which were identical in this example) for the 3 classes, Elephant, Goat and Human combined with an overall result. The meaning of the columns is described in this document.
- You can open these text files to see the results and import them into any data analysis program required for further analysis and creation of nice plots.
- Congratulations! If you reached this point, you have successfully trained a computer vision model using the software and then tested its performance on some (theoretically) independent data.
- More advanced operations should still follow a similar form to those shown in this example.

```
prmw@mimir: ~/Example/astroeco_ml
File Edit View Search Terminal Help

23 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.0
0
24 route 21 -> 10 x 10 x 256
25 conv 128 1 x 1/ 1 10 x 10 x 256 -> 10 x 10 x 128 0.007 BF
26 upsample 2x 10 x 10 x 128 -> 20 x 20 x 128
27 route 26 6 -> 20 x 20 x 256
28 conv 128 3 x 3/ 1 20 x 20 x 256 -> 20 x 20 x 128 0.236 BF
29 conv 24 1 x 1/ 1 20 x 20 x 128 -> 20 x 20 x 24 0.002 BF
30 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.0
0
Total BFLOPS 1.051
Allocate additional workspace_size = 52.43 MB
Loading weights from /home/prmw/Example/astroeco_ml/model_test/yolov3-tiny-3l_la
st.weights...Done! Loaded 31 layers from weights-file
708Total Detection Time: 1.000000 Seconds
Done.
Computing final statistics from the testing set run...
Done.
These statistics can be found in a directory named 'eval_results' inside the cho
sen data output directory.
Operation complete.
(base) prmw@mimir:~/Example/astroeco_ml$
```

Figure 7: An example of the output of the completed evaluation mode in the terminal window.

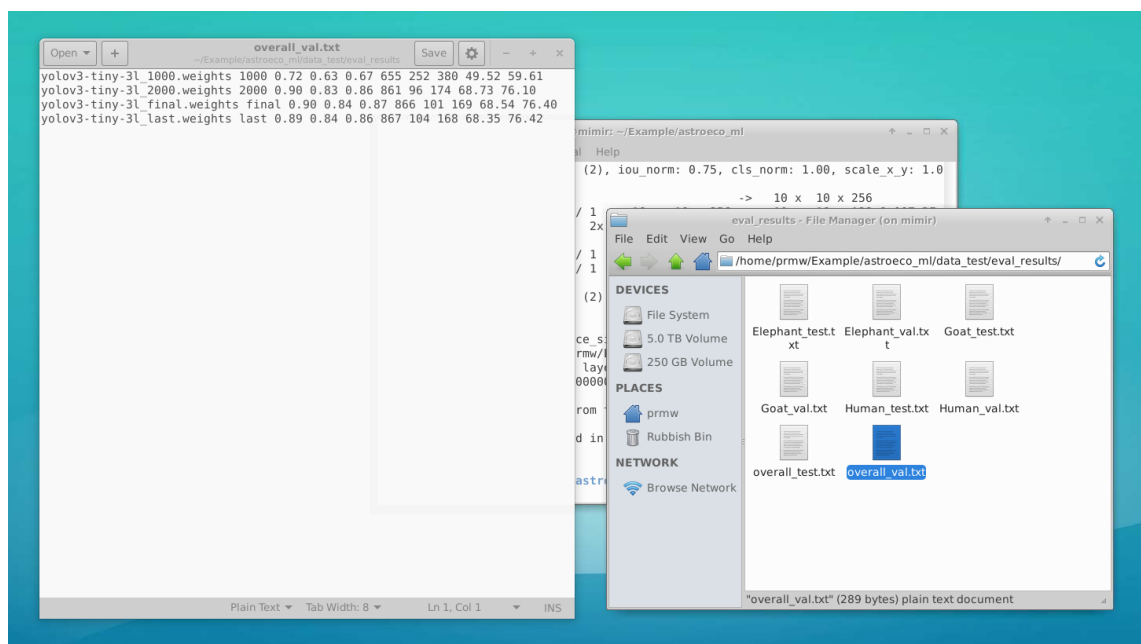


Figure 8: The output table files of the evaluation example operation with the overall table of the validation set open.

Final remarks

- Following this guide and addressing error messages from the operation of this software should be sufficient to mitigate any major issues.
- Feel free to inspect any of the .py python scripts in the astroeco_ml folder but take care not to alter them or delete any of them as the software is dependent on their use.
- This software is limited in capability and designed to offer users access to some powerful computer vision machine learning but it is not designed to replace the users education!
- For further information on how to use darknet independently of this software please study closely the wise words of AlexeyAB on his github darknet repository (used extensively by this software) found at: <https://github.com/AlexeyAB/darknet>
- Take care and I hope this software is useful! Ross McWhirter, 15th April 2020.