DeepLabv3 Segmentation
Docker Guide

February 2025

1 Summary

This is a guide to the DeepLabv3 segmentation program used to segment aerial orthomosaics of hemp and grapevine. The multiple dependencies and scripts involved have been compiled into a docker image to simplify usage and promote batch processing. The Docker image should be able to run on a wide range of NVIDIA GPUs. A bash script (.sh file) is also provided to run these scripts from outside the Docker image and produce results.

The Docker image consists of a series of Python scripts which are called upon within a running instance of the Docker image. The scripts called directly are:

- **convert_to_rgbṗy**: converts the orthomosaic from a .tif file to a .png file using only specific wavelengths
- **ImageMask2Tiles_uniform.py**: breaks up a .png orthomosaic into various tiles for labeling. This may be used to further train the model.
- **train_new.py**: Used to train a model based on labelled images.
- **inference_new.py**: Used to run inference on (i.e. segment) images.

2 Setup

For Windows users:

- Install and Set-up WSL2 (Windows Subsystem for Linux)
- Install a Linux distro (preferably Ubuntu)
- Make sure a compatible NVIDIA driver is installed
- Install versions of CUDA Toolkit and CUDNN compatible with your NVIDIA GPU:

  - First, check your GPU name and go to `https://developer.nvidia.com/cuda-gpus#compute` to find its compute capability
  - Navigate to `https://en.wikipedia.org/wiki/CUDA#GPUs_supported` to check which CUDA toolkit is compatible with your GPU.
  - Find toolkit on NVIDIA website and install.
  - Check `https://docs.nvidia.com/deeplearning/cudnn/backend/latest/reference/support-matrix.html` and select CUDNN version to install based on the version of the CUDA toolkit installed.

- Install Docker Desktop
- Ensure Docker Desktop uses WSL2 as backend:
  - After installation, open Docker Desktop
  - Navigate to "Settings"
  - From the"General" tab, check if the"Use WSL 2 based engine" option is selected. If not, select it and then select "Apply and restart"
- Download custom DeepLabv3 Docker image and the corresponding bash script

  Docker image: `deeplab-segment_4.7_cuda12.1_published.tar`

  bash script: `Deeplab-updated_cuda12.1-published.sh`
- Make sure Docker engine is running
- Use the docker `load` command to load the image into Docker:

  ```
  docker load --input deeplab-segment_4.7_cuda12.1_published.tar
  ```

  via powershell, Ubuntu terminal or cmd
- Ensure the package x11-apps is installed in WSL. This can be done by executing

  `sudo apt update`

  and `sudo apt install x11-apps`

  in the WSL terminal

For Linux users:

- Make sure a compatible NVIDIA driver is installed
- Install compatible versions of the CUDA Toolkit and CUDNN as well as any dependencies needed.
- Install Docker Desktop for Linux

- Download custom DeepLabv3 Docker image and the corresponding bash script

  Docker image: `deeplab-segment_4.7_cuda12.1_published.tar`

  bash script: `Deeplab-updated_cuda12.1-published.sh`

- Make sure Docker engine is running
- Use the docker `load` command to load the image into Docker:

  ```
  docker load --input deeplab-segment_4.7_cuda12.1_published.tar
  ```

  via terminal.

- For some Linux users, Docker does not have access to the directory required to use the display. This can be accomplished by adding `/tmp/.X11-unix` to Docker resources. Open Docker, go to **Resources**, click on the **File Sharing** option and add: `/tmp/.X11-unix`.

3 Using the Docker image and bash script

To use the Docker image via the bash script for Linux or Windows:

- Start up the Docker Engine by opening Docker Desktop
- Ensure the engine is running
- Open the Linux Terminal (for Windows open the terminal of the WSL distro installed, e.g. Ubuntu)
- Navigate to the location of the bash script and run the bash script. The bash script is run by typing:

  ```
  bash Deeplab-updated_cuda12.1-published.sh --arg1 --arg2 ....
  ```

Details on how to configure the script are given in the next section.

3.1 Running the bash script

As shown above, the bash script can be configured with certain arguments, some of which are required, depending on what script(s) within the Docker image you want to run. The arguments are listed below:

- **-indir/--input-directory**: Path to the input directory, i.e. the images you want to process. For example, if you want to convert a multispectral .tif image to RGB in .png format, the `-indir` argument would refer to the folder containing the input multispectral .tif files. Ex:

  `\Desktop\Your_Project\tif_files\`

- **-outdir/--output-directory**: Path to the output directory, i.e where the processed images (or any other kind of output file) should be stored. Ex:

  `\Desktop\Your_project\outputs\`

- **-prog/--prog-value**: This is a numeric argument that dictates which program to run:

  1 for RGB conversion

  2 for tiling the orthomosaic (breaking it up into patches)

  3 for training a DeepLabv3 crop segmentation model

  4 for applying the trained model to a new set of images

- **-ep/--epochs**: The number of epochs required when training the DeepLabv3 model. Not necessary to input when there is no intention to train a new model. Default value is `30`.

- **-exp/--experiment_name**: The name of the experiment when training a model. Serves as the folder name for the training outputs. Ex:

  `experiment1_grapevine`

- **-br/--base_rate**: Sets the base learning rate for training a model. Default is `0.00025`

- **-wk/--workers**: Number of workers for model training. Default is `3`

- **-batch/--batch_size**: Batch size for model training. Default is `8`

- **-inf/--inference_model**: Path to the model (.pth file) used for inference. Ex:

  `\Desktop\Your_project\output\experiment1_grapevine\XXXX.pth`

- **-modeldir/--model-directory**: Path to the base model (.pth file) used for training. Ex:

  `\Desktop\Your_project\XXXX.pth`

- **-cr/--crop_size**: Dimensions (in pixels) of the annotated image patches used for model training in train_new.py. The annotated patches must be square, so the x and y dimensions have the same value. Default is `512`

- **-class/--class_num**: Number of classes for model training (train_new.py). Default is `3`

- **-dt/--datatype**: Type of input dataset used for model training. Accepts either: `grape` or `hemp`. Default is `grape`

4 Functions

These functions should be executed in the order in which they are listed as the latter function often depends on the reults of the former. Note that the output folder and the input folder should be two separate folders.

- **RGB conversion**: Implements `convert_to_rgb.py` to convert a multi-spectral .tif image to an RGB .png file. A new sub-folder called `rgb_png` is automatically created within the user-specified `output_directory` to store the output .png file:

  `{output_directory}\rgb_png\`

  The RGB conversion **-prog** value is 1.

  Required arguments: `-indir -outdir -prog`

- **Tiling**: Executes `ImageMask2Tiles_uniform.py` to split the RGB .png into smaller patches for manual annotation. The output patches are exported to a new sub-folder: `{output-directory}\tiles\`

  Patches can then be retrieved for manual labeling (or any other purpose). The Tiling **-prog** value is 2.

  Required arguments: `-indir -outdir -prog`

- **Training**: Using `train_new.py`, the user can fine-tune a DeepLabv3 model based on a set of annotated image patches.

  Before training, ensure that your image patches and annotated mask files are organized according to the following structure (`img_patch_x.jpg` represents the RGB image patch, and `mask_patch_x.png` represents the annotated crop mask corresponding to image patch x):

```
output
└── labeled_data
    ├── training
    │   ├── Image
    │   │   ├── img_patch_1.jpg
    │   │   ├── img_patch_2.jpg
    │   │   ├── img_patch_3.jpg
    │   │   └── ...
    │   └── Mask
    │       ├── mask_patch_1.png
    │       ├── mask_patch_2.png
    │       ├── mask_patch_3.png
    │       └── ...
    ├── testing
    │   ├── Image
    │   │   ├── img_patch_4.jpg
    │   │   ├── img_patch_5.jpg
    │   │   └── ...
    │   └── Mask
    │       ├── mask_patch_4.png
    │       ├── mask_patch_5.png
    │       └── ...
    └── validation
        ├── Image
        │   ├── img_patch_7.jpg
        │   ├── img_patch_8.jpg
        │   └── ...
        └── Mask
            ├── mask_patch_7.png
            ├── mask_patch_8.png
            └── ...
```

Note: Images should be in .jpg format and masks in .png format. The labeled_data folder **must** be located in the output directory.

The input to the -indir flag is arbitrary when running the training program. It is kept as a required argument to provide some uniformity.

The output of the model training is a .pth file that can be used to segment subsequent images. The .pth file is automatically exported to a sub-folder within the "-modeldir" folder, labeled with the experiment name. Ex:

```
Your_project\{model-directory}\experiment1_vineyard\{datetimestamp}\XXXX.pth
```

The Training **-prog** value is 3.

Required arguments: `-indir -outdir -prog -exp -modeldir -ep -dt`

Optional arguments: `-br -wk -batch -cr -class`

- **Inference** : Running `inference_new.py` will use the fine-tuned DeepLabv3 model to create a crop mask based on the RGB .png files generated in the RGB conversion step. It takes the following arguments: an input directory (in this case, input directory **must** be `output_dir\rgb_pngs`), an output directory, and a model filepath (see example filepath in previous section). The .pth file must be stored in the output directory or a sub-folder within the output directory. The Inference **-prog** value is 4.

  Required arguments: `-indir -outdir -prog -inf -modeldir`

5 Logging

To log each process for later reference, one can store the terminal outputs in a .txt file. This can be done by typing:

```
bash Deeplab-updated_cuda12.1-published.sh -arg1 -arg2 -arg3.. 2>&1
| tee log.txt
```