Urban 3D Challenge Software User Guide

Finalists of the Urban 3D Challenge were required to submit a dockerized version of their system so that the stakeholders of the contest can verify their submission in a clean and standardized fashion. Chapter 1 of this document gives the requirements for such submissions. Chapter 2 provides background information on Docker. Chapter 3 provides an example by showing how a simple baseline solution created in Java could be dockerized.

NOTE: This document was written for the contestants of the Urban 3D Challenge. It can now be used to understand how to use the open source solutions for training and testing purposes. Note that some contestants deviated slightly from the guidelines provided below (including the winning solution – albu – so please be sure to read the individual readme files associated with each solution.

# Requirements

## Submission format

The submission must be a zip package that contains:

* All your code (training and inference) that are needed to reproduce your results.
* A Dockerfile that will be used to build your system. The Dockerfile must include a comment whether it is meant to be built on a CPU- or GPU-based system, see the section ‘Hardware specification’ below.
* All data files that are needed during training and inference, with the exception of
  + the contest’s own training and testing data. You may assume that the training and testing data will be vailable on the machine where your docker container runs, already unpacked, in the same structure as it is available for download during the contest,
  + large data files that can be downloaded automatically either during building or running your docker script.
* Your trained model file(s). Alternatively your build process may download your model files from the network. Either way, you must make it possible to run inference without having to execute training first. An exception to this rule is the case when your model files can be quickly recreated by your training process. As an example, if your model files are several gigabytes in size but your training process can recreate them in 10 minutes then it makes more sense to run training instead of adding the files to the package.

Either the submission package or a link to its public address must be sent to the contest admins in email (walrus71@copilots.topcoder.com and tim@copilots.topcoder.com) within 10 days after the end of the provisional testing phase.

After unpacking your submission the

docker build -t <id> .

command will be used to build your docker image (the final ‘.’ is significant), where <id> is your TopCoder handle. (Note that depending on docker setup each command starting with “docker …” may need to be issued as “sudo docker …”.)

The build process must run out of the box, i.e. it should download and install all necessary 3rd party dependencies, either download from internet or copy from the unpacked submission all necessary external data files, your model files, etc.

Your container will be started by the

docker run -v <local\_data\_path>:/data -it <id>

command, where the -v parameter mounts the contest’s data to the container’s /data folder. This means that all the raw contest data will be available for your container within the /data folder.

## Training and test scripts

Your container must contain a train and test script having the following specification:

* train.sh <train\_data\_path> should create any data files that your algorithm needs for running test.sh later. The supplied <train\_data\_path> parameter points to a folder that has all the training files of the fmow-rgb or fmow-full AWS bucket in the same structure as is available for you during the coding phase.
* The allowed time limit for the train.sh script is 7 days.
* Training should be possible to do using only raw training data of the contest and publicly available external data. This means that this script should do all the preprocessing and training steps that are necessary to reproduce your complete training work flow.
* A sample call to your training script:  
  ./train.sh /data/train  
  In this case you can assume that the training data looks like this:  
   data/  
   train/  
   JAX\_Tile\_004\_DSM.tif  
   JAX\_Tile\_004\_DTM.tif  
   JAX\_Tile\_004\_GTC.tif  
   ...  
   JAX\_Tile\_005\_DSM.tif  
   ...
* test.sh <train\_data\_path> <test\_data\_path> <output\_file> should run your prediction code for a given set of test images.
  + <train\_data\_path> is as defined above for train.sh
  + <test\_data\_path> points to a folder that contains test data files. The format of these files will be the same as for the provisional test files.
  + <output\_file> is the name of a file (without extension) your code should generate. The full name of the generated file must be either <output\_file>.txt or <output\_file>.zip.
* The allowed time limit for the test.sh script is 8 hours.
* Testing should be possible to do without running training first, i.e. using only your prebuilt model files.
* It should be possible to execute your test script multiple times on the same set of test images or on a different set. You must make sure that these executions don't interfere, each execution leaves your system in a state in which further executions are possible.
* A sample call to your testing script :  
  ./test.sh /data/train /data/test out1  
  In this case you can assume that the training data looks like shown above and the testing data looks like this:  
   data/  
   test/  
   JAX\_Tile\_000\_DSM.tif  
   JAX\_Tile\_000\_DTM.tif  
   JAX\_Tile\_000\_RGB.tif  
   JAX\_Tile\_001\_DSM.tif  
   ...

## Code requirements

* Your training and test scripts must output progress information. This may be as detailed as you wish but at the minimum it should contain the number of tiles processed so far.
* The algorithm and models within your container should be identical to what you used in your final submission during the online submission phase of the contest. You must not improve your algorithm any further after the end of the submission phase (4th December).
* Your testing code must process the test and validation data the same way, that is it must not contain any conditional logic based on whether it works on images that you have already downloaded or on unseen data.

## Hardware specification

Your docker image will be built and run on a Linux AWS instance, having one of these two configurations:

* m4.10xlarge (for CPU-based systems)
* p2.xlarge (for GPU-based systems)

Please see [here](https://aws.amazon.com/ec2/instance-types/) for the details of these instance types.

# Docker introduction

### What is Docker?

From the Docker website: [https://www.docker.com/ “](https://www.docker.com/)Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.”

### Why do we use it?

Docker is a great way to control your build and deployment environment. It ensures that the tools necessary are well documented and algorithms are easy to transfer from one location to the next.

### Installation of Docker

First we need to install Docker on your computer. Follow the instructions at<https://www.docker.com/community-edition> to setup Docker.

If your algorithm requires a GPU you must next install NVIDIA-Docker at <https://github.com/NVIDIA/nvidia-docker>

NOTE: All six (6) winning solutions are built for use with a GPU. Therefore, the nvidia-docker package is required for running all solutions. **IMPORTANT**: In all code examples in this file, use ‘nvidia-docker’ in place of ‘docker’ if running with a GPU.

# Dockerization example

As a real life example here is a way to dockerize a simple solution written in Java that works by predicting rectangles that look interesting: they have little difference in their colour and z coordinate inside but they differ a lot from their surroundings. Such solution doesn’t need any training and doesn’t produce any model files, but hopefully it illustrates most of the work flow of dockerization.

Download the package from [here](https://drive.google.com/file/d/0BzQFrCvRK-MORjhqVmVaaVhJRU0/view?usp=sharing) and extract it anywhere.

Building the Dockerfile with

docker build -t urban3d .

executes these steps:

* Installs some general purpose development tools on top of a ubuntu image.
* Installs java8 which is needed to build and run this solution.
* Copies everything from the build context (i.e. the whole folder where the Dockerfile is, recursively) to the container’s /work folder.
* Creates a bin directory and calls javac to compile the java source files into bin.
* Makes the train and test scripts executable.

(Depending on your OS and docker setup, you may need to write ‘sudo docker …’ instead of ‘docker …’.)

Running the image built above with the

docker run -v //c/Users/urban3d\_data:/data -it urban3d

command will start a container that

* Makes the contents of my local data files at c:\Users\urban3d\_data available at /data (note the strange syntax to be used for Windows based local paths. Also, Windows users should be aware that by default they can only use the c:\Users folder to share data with containers).
* Has Java-8 installed in ubuntu.
* Has the application already built in /work/bin.
* Puts the user in the /work folder.

From within the container you can run the testing script by executing the

./test.sh /data/train/ /data/test-provisional/ out-provisional

command. This should create the /work/out-provisional.txt output file. If you want the output file to be available also out of the container then either

* copy it to the mounted volume while you are still within the container (somewhere below /data), or
* use the ‘docker cp …’ command after you exited from the container.