

File Structure Outline

To utilize our provided codes with minimal changes required, we recommend following a specified file structure if possible.

Input Image Files Structure

Given that there is a limit to the number of images that can be processed at once, we encourage a file structure that is broken down. For example, for a given day of images from multiple (2-4) hours of collecting images, there should be a folder for about every 5,000 images. Within each of these folders containing the 5,000 360° images **along with a (.txt) or keyhole markup language (.kml) containing the latitude and longitude of all the images in the subfolder**. All these subfolders should then be consolidated into one folder for the particular day of images one would like to process. **We highly recommend that the run folder follow a naming structure of YYYYMMDD_XX at the beginning of the folder name, where YYYYMMDD is the run date and XX is the run number, otherwise the user must amend our uploaded code to reflect their folder naming structure.**

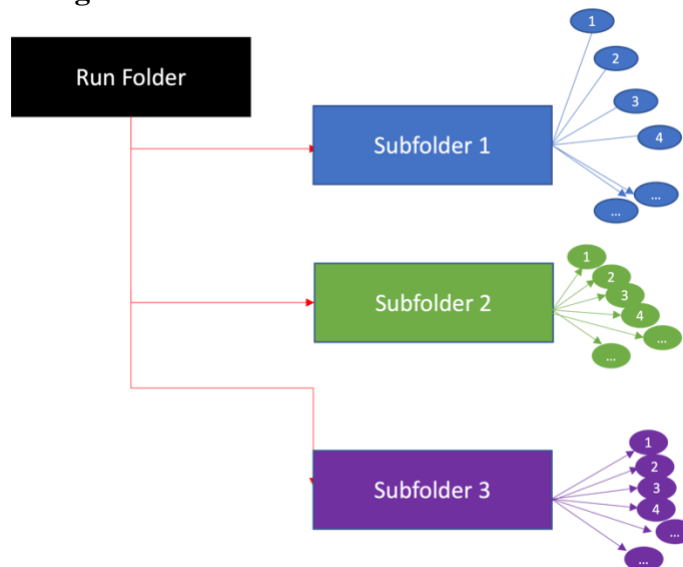


Image zipping and transferring

With this folder structure, given the size of the folder with all the images, we recommend zipping the run's contents together using a tar command, for example one could use the following command (this step can take quite a while (several hours) if your total images is over 1TB in size, make sure your computer does not disconnect from running this line of code):

```
!tar -cvf run03.tar 20200510_03_seattle_fullcity_processed
```

Once the data is zipped, you should transfer it (we recommend using a service such as Globus so that local systems are not running one line of code for hours) to and open it in a python environment on a supercomputer by unzipping using a command such as “tar -xf run03.tar” (note that this again could take a long time to run).

Setting up Python Environment

When setting up your Python environment on the supercomputer, please make sure the following modules are loaded. Also, make sure there is a specified folder assigned to \$HOME in your

environment and \$SCRATCH. Along with these modules, load the correct package versions listed in the *requirements.txt* uploaded file:

Module	Version
git	2.24.2
autotools	1.2
cmake	3.24.2
pmix	3.1.4
hwloc	1.11.12
xalt	2.10.34
gcc	6.3.0
cuda	10.1
cudnn	7.6.5
nccl	2.5.6

Running Analysis

Required Python Codes

Before running any analysis, you will need to load the *jobmaker.py* and the *CorrectAndDetect.py* python files along with the *runJobs.sh* shell script in the \$HOME directory. These files have been uploaded to the project GitHub folder.

Creating Jobs

In the first step of running the analysis, the user needs to run the *jobmaker.py* code. This code breaks up the subfolders of the images into separate job files so that multiple jobs can be run at the same time to optimize code run time. We recommend running the analyses in your \$SCRATCH directory.

```
python3 jobmaker.py $SCRATCH/source_folder  
$SCRATCH/output_folder your_email@uw.edu
```

Running Jobs

To run the jobs, change your directory to the *runXXjobfiles* which has been created by the *jobmaker.py* call above. Then, you will individual jobX.txt files. Each of these job files will have a bash call inside them to run the analysis on this job. To run a single job file, you can use the slurm command “*sbatch jobX.txt*”. You can also run a bash command to dispatch multiple jobs ranging from X (inclusive) to Y (exclusive). Please note your supercomputer environment may have a maximum number of concurrent jobs it can run, to your range of X to Y will be dependent on this.

```
bash runJobs.sh 1 16
```

As jobs finish, there will be a RunX.o and RunX.e output files. The .o file will show you the number of images the code has detected. The .e file will let you know of any errors/warnings. You can track the progress of the slurm jobs using “*squeue -u your_username*” (your username is your email provided in the jobmaker.py call).

Compressing Results for Analysis

Once all the jobs are successfully run, you should change your directory to the output folder provided in the jobmaker.py call. There, you will find outputX.pkl pickle files corresponding to

each jobX.txt file. Inside the pickle file is a list of images where the algorithm has detected a person, the probability of success of identifying the object as a person (from the algorithm), and the bounding box parameters for the person. Along with that, there will be a left and right hand side image from the 360° camera of all the images (which the pickle file findings refer to). In this process, the pictures are orthorectified and the size of the image is reduced. If you are going to analyze your results on the same supercomputer, you can now proceed, otherwise we now need to compress these which you can do with “`tar -cf runX.tar runx/`” and then transfer it to the post analysis location.

Analyzing the Images

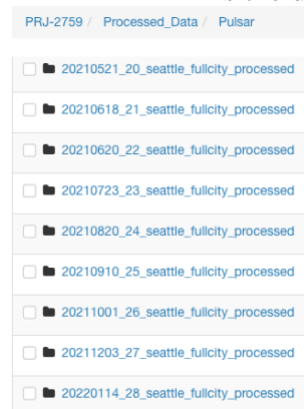
Next we need to analyze the images and detect the number of people in the images. To do so, we will run the *postProcessing.py* python code. **It is critically important that you have access to the original run folder while running this code. Please note that this is the one code the user must amend to reflect their file structure to be able to run. Within the *jobPickleToDataframe* function, the user needs to specify the *base* & *runFolder* values based on their file naming structure.** This code will take the pickle file and find all the detections of pedestrians within the pickle file and save them to a mysql database named *detections.db* (feel free to change the name if desired). Once the code is completed for a full run, there will be a database with all the detection counts for the run. In this database, along with the detection counts, will be the longitude and latitude of where the detection occurred, and the X&Y min & max which a user can utilize to draw a bounding box around the detection. With this database, a user can join on any demographic or other variables they wish for their analysis.

Information about our File Structure

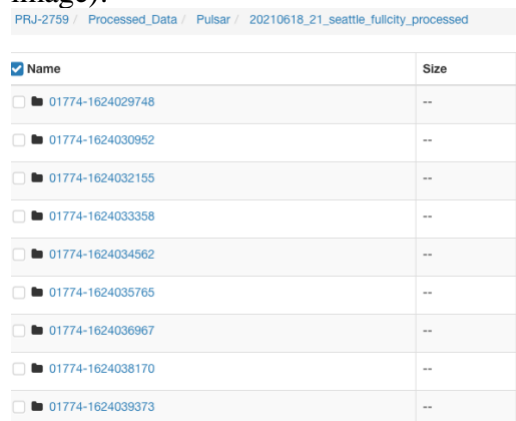
Below will be information about how your files are structured to make better sense of the environment we run our codes on.

Input Raw Run Files

Our raw run images exist under a folder named “Pulsar” which then contains all the runs where each run follows a YYYYMMDD_XX_structurename naming convention where the YYYYMMDD is the date of the run and the XX is the run number.



Here is how one sample run folder is structured (there are more subfolders than shown in the image):




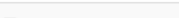
<input checked="" type="checkbox"/> Name	Size
<input type="checkbox"/> 01774-1624029748	--
<input type="checkbox"/> 01774-1624030952	--
<input type="checkbox"/> 01774-1624032155	--
<input type="checkbox"/> 01774-1624033358	--
<input type="checkbox"/> 01774-1624034562	--
<input type="checkbox"/> 01774-1624035765	--
<input type="checkbox"/> 01774-1624036967	--
<input type="checkbox"/> 01774-1624038170	--
<input type="checkbox"/> 01774-1624039373	--

Inside the first folder is 4,392 images with framepos.txt file looking like this:

```
systemtime_sec,frame_index,lat,lon,altitude,distance,heading,pitch,roll,track,jpeg_filename
1624029752.09913,0,47.6525830473553,-122.304252150533,12.2962800490039,0,-46.4666472518189,1.8414538981561,2.9691973819328,-47.0995109133976,0000000000.jpg
1624029753.24197,1,47.6524983448491,-122.304241127031,12.1264828903696,1.01953793766742,-31.3113911892116,1.60759680327445,3.63384996683125,-30.3859976969052,0000000001.jpg
1624029754.24195,2,47.6524919193746,-122.304232323319,12.0915472769722,2.02415401015054,-15.6626264311549,0.91325656259188,4.22895365249896,-15.7682841279695,0000000002.jpg
1624029755.24193,3,47.6524849399477,-122.304226531308,11.9570716794727,2.99273158426751,-1.45803037102948,0.103636791338852,4.56325301144669,-3.27888951441573,0000000003.jpg
1624029756.81333,4,47.652477863118,-122.304223347284,11.6852979681711,3.94508846050708,12.7108012985705,-0.005510628332593,4.69229051225133,9.00947503158695,0000000004.jpg
```

So to process this run, we would zip the “20210618_21_seattle_fullcity_processed” folder into a tar file named “run21.tar” and transfer it to our supercomputer. After unzipping it on there and running the jobmaker.py code on it, we get 24 jobs to run. After we process those jobs we get left and right pictures of all pictures within each subfolder and 24 pickle files in our fulloutput folder. For example, here is the exact same subfolder from above but in the fulloutput folder:

PRJ-2759 / Computer_Vision / CorrectedImages / 20210618_21_seattle_fullcity_processed / 01774-1624029748

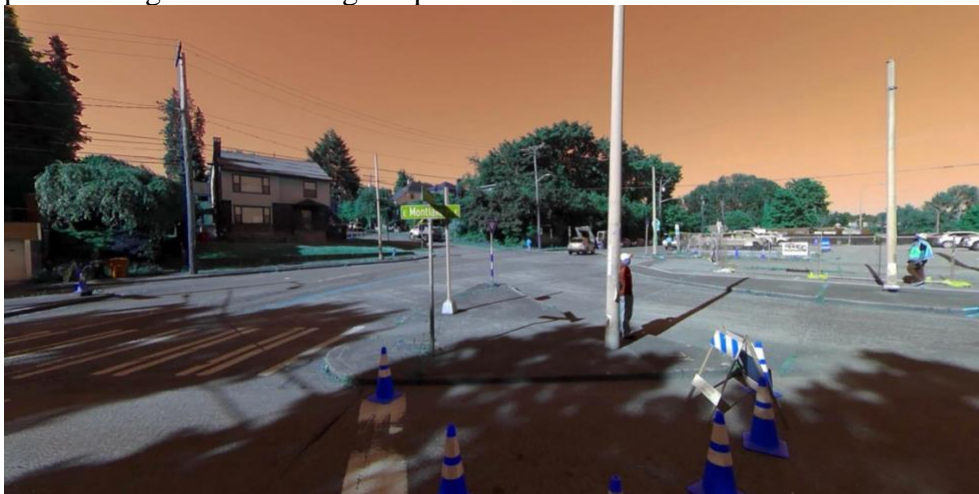
<input type="checkbox"/>  0000004387_l.jpg	1.3 MB	9/3/22 4:08 PM
<input type="checkbox"/>  0000004388_l.jpg	1.3 MB	9/3/22 4:08 PM
<input type="checkbox"/>  0000004388_r.jpg	1.6 MB	9/3/22 4:08 PM
<input type="checkbox"/>  0000004389_l.jpg	1.3 MB	9/3/22 2:27 PM
<input type="checkbox"/>  0000004389_r.jpg	1.6 MB	9/3/22 2:27 PM
<input type="checkbox"/>  0000004390_l.jpg	1.3 MB	9/3/22 3:00 PM

In the end, we found that this run contains 110,000 pedestrian detections (at 80% confidence threshold).

For example purposes, here is a raw image our Streetview campaign took on a run:



We can notice on the left side there are some pedestrians, here is the orthorectified left hand picture we get after running the process:



Here are the pedestrian detections our algorithm detected with their respective bounding boxes:

