

The background features a white space with several colorful circles and dashed lines. On the left, there are teal and blue circles, some with dashed outlines. On the right, there are green, orange, and yellow circles, some with dashed outlines. A large yellow ring is at the bottom right. Dashed lines connect some of the circles, creating a path across the page.

Urban Forests on the UN Global Platform

Joe Peskett
@joepeskett

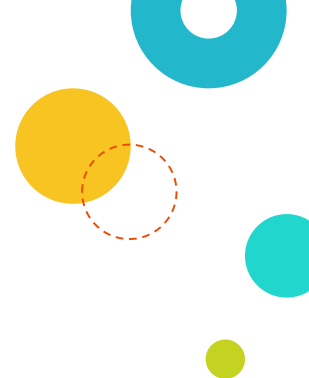
#UNGlobalPlatform

Overview

- Data Science Campus project and methodology
- Implementation on the UN Global Platform services



Urban Forests drivers

- Collaboration with the Natural Capital team within UK ONS.
 - There is a clear stakeholder.
 - There is are clear learnings that can be reused and novel techniques to be used.
- 



Alternative methodologies

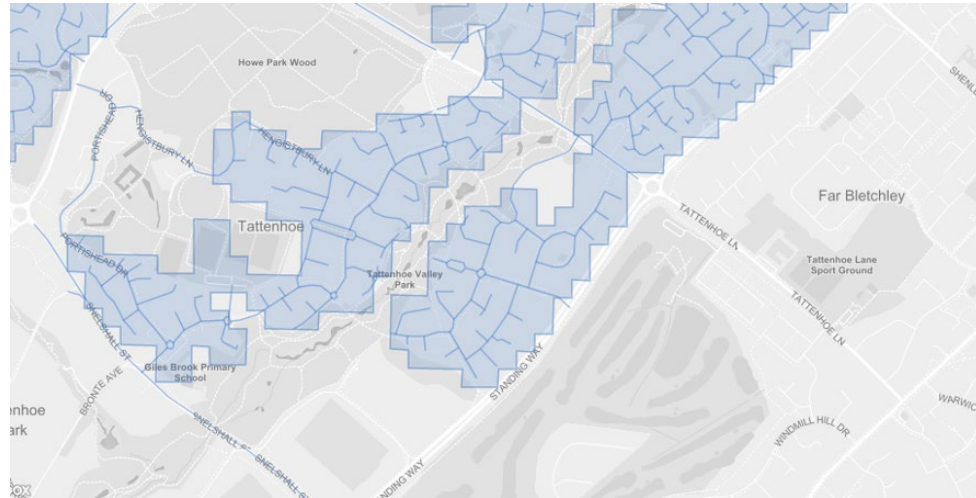
- Survey
- Crowdsourcing information
- Satellite imagery



Urban Forests Methodology

1. Use Open Street Map (OSM) to generate sample points around 112 towns & cities in the UK
2. Street view images taken at these points
3. Images are segmented to provide a value for percentage vegetation

Urban Forest Methodology



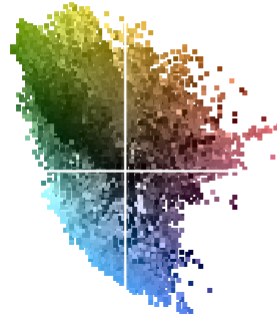


Urban Forest Methodology

1. Percentage of green pixels
 - a. Using LAB colour space, random forest used to increase accuracy
2. Pyramid -scene-parsing network (PSP - net) trained on CityScapes dataset to segment each image

Urban Forest Methodology

Vegetation



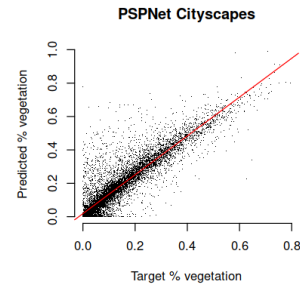
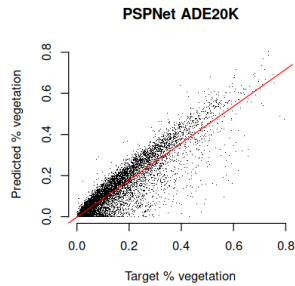
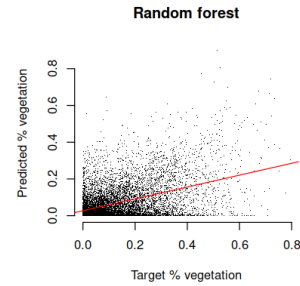
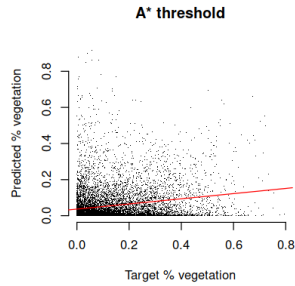
Cars



PSP-net Segmentation



Methodology Comparison



Results from the Campus

- Total dataset ~17.1 million images
- Technical report
- Open source pipeline on GitHub
- [Collaboration with ONS visualisation team](#)
- Implementation on the UN Global Platform

Implementation within the UNGP Methods Service

- Hosting algorithms, methods and microservices
- Dependencies managed per method using containers
- Run on cloud infrastructure, allowing quick scaling
- Methods are called using APIs
- Easy access to cloud datastores

1. Image processing

- Same PSPnet as used for the original pipeline
- You can see that Phil, one of the lead Data Scientists developed this code.
- The model file is loaded in using a function – the model file is saved in the developers storage, though is made available to be used in methods that he has developed.



#UNGlobalPlatform



2. Considerations for cloud implementation

- Keep outputs small
- Can outputs be formatted as inputs for the following method in the pipeline?
- Keep scaling in mind

3. Methods for the pipeline

OSM points

Generate points for requesting images

Street-view images

Download images into cloud storage, filed into way ids.


Segment images

Analyse images/return value of vegetation for each image

Create composites

Use segmentation data with original image to show classes with colours

3. Methods for the pipeline

 [joepeskett_ungp](#)/**HighwayScrapeR**/2.0.0 copy ☆ STAR 0 🗨️ FOLLOW 1

Run Docs Discussion Source →

HighwayScrapeR

Sample a queried area for OSM highways and return points at defined intervals(in metres) along these highways.

API calls 395

Tags

Experimental Geo geospatial OSM Urban Forests

Permissions

Algorithmia Platform License • [Internet Access](#) • [Calls Other Algorithms](#)



[HighwayScrapeR method](#)

#UNGlobalPlatform



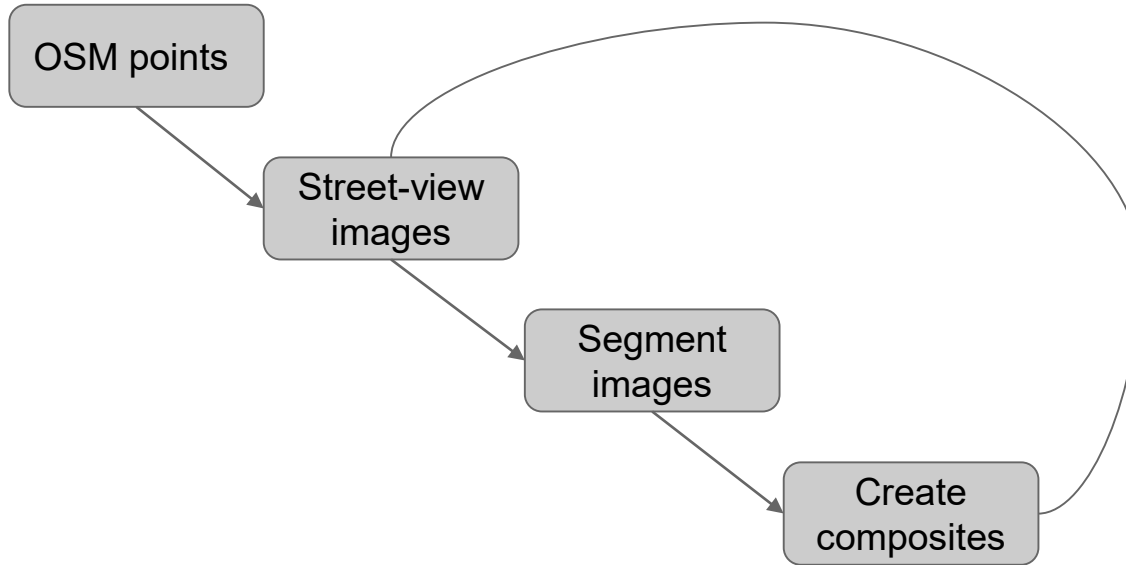
#UNGlobalPlatform



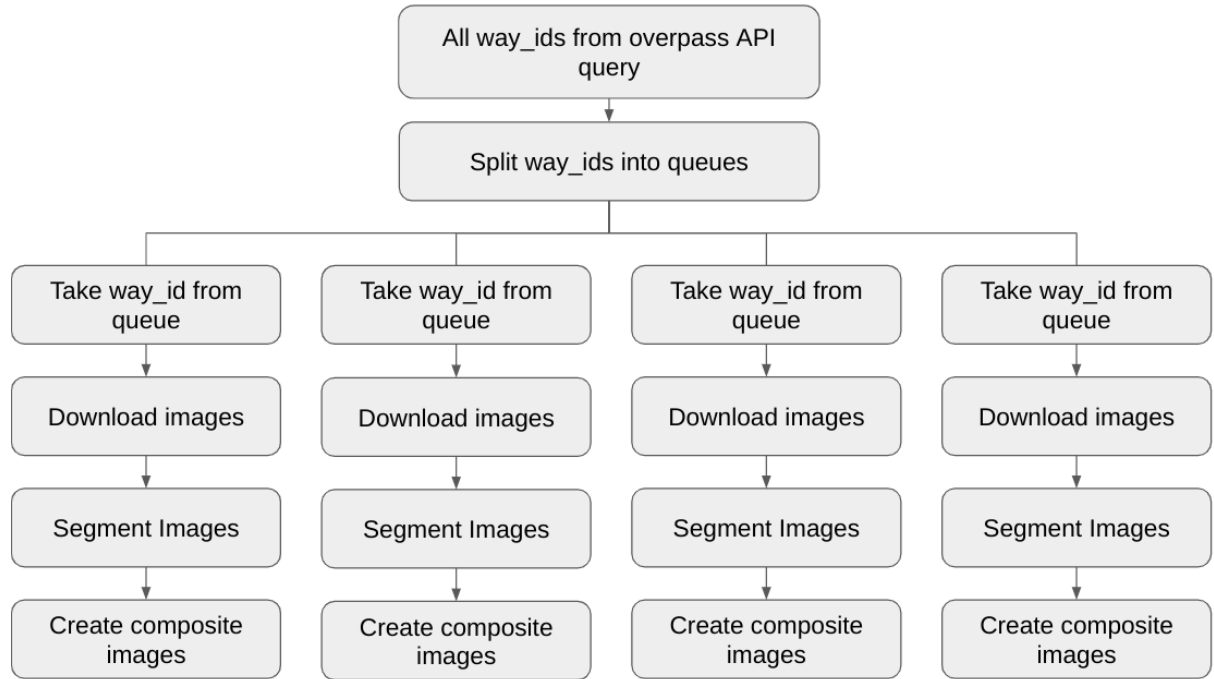
3. Methods for the pipeline: Image downloader

- All images are saved in cloud -based storage
- Coordinates are kept in the image filename, way_id is in the folder name

4. Composing the pipeline



5. Asynchronous processing



6. Calling the pipeline locally

```
22 #tidy_tibbles
23 input_tibble<-do.call(rbind, result %>%
24   lapply(FUN = function(x) rbind(map(x, unlist)))
25   lapply(as.tibble)) %>% mutate(way_id = rownames)
26
27 input_tibble
28 input_tibble %>% unnest(V1, V2, V3)
29 leaflet() %>%
30   addTiles() %>%
31   addCircles(data = input_tibble %>%
32     unnest(V1, V2, V3),
33     lng = ~as.numeric(V1),
34     lat = ~as.numeric(V2))
35 #create a pipeline
36 #create algorithm objects
37 call_creator<- function(func_name){
38   algo<-clientsalgo(func_name)
39   algo$setOptions(timeout = 3000)
40   return(algo)
41 }
42 #create a list of algorithm objects
43 down_segment <- function(functions = c("joepeskett_ungp/st
44   "DSC/vegetation/0.2
45   "DSC/Colourise_seg
46   functions_list<-lapply(as.list(functions), call_creator)
47   functions_list
48 }
49 call_list<-down_segment()
50 #Wrap the functions into the pipeline:
51 function caller<- function(input, call_list){
```

The screenshot shows the RStudio environment with a script editor on the left and a map viewer on the right. The script defines a pipeline for processing spatial data. The map viewer displays a map of Penylan, Cardiff, with a green dashed line indicating the output of the pipeline. The map includes labels for various roads and areas, such as Penylan Hill, Penylan, Roath, and Cathays. The RStudio interface includes a menu bar (File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help) and a toolbar with icons for file operations, zooming, and exporting. The console at the bottom shows the execution of the script.



Pros and Cons of this pipeline

- Use of street -view images
 - Incomplete coverage using street -view images
 - Timing of street -view image capture
 - Ensuring access to data
1. Learning how to work in new ways
 2. Modular design of pipeline, allowing reuse of code



Thanks for listening.

Any questions?