LEVERAGING MAPILLARY'S OBJECT DETECTION CAPABILITY TO UPDATE KEY DATASETS

ALEXANDER YOUNG | DHAVAL JARIWALA

Introduction To Mapillary

"What if you could map the world with just your smartphone?"

- **What is Mapillary?**
  A global, crowdsourced platform turning street level imagery into actionable insights.

- **Why It Matters**:
  Streamlines mapping for urban planning, navigation, and infrastructure management by detecting features like traffic signs and crosswalks.

- **Key Details**:
  - Founded in 2013 in Malmö, Sweden.
  - Acquired by **Meta** (Facebook) in 2020.
  - **2+ billion** images from 190+ countries.

"Object detection on Mapillary (Image Source: Geo Week News)"

How Mapillary Works

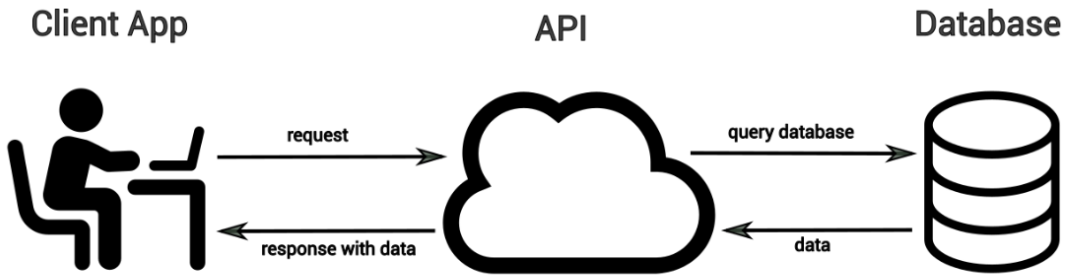
"From Photos to Actionable Data"

- **Crowdsourced Imagery Collection**:
  - Individuals/Organizations capture and upload geotagged street-level photos using devices like smartphones, dashcams, or action cameras.
- **3-D & Computer Vision Technology:**
  - **3-D Reconstruction:** Uses "SfM" (Surface from Motion) to map 3D locations by matching points across overlapping images.
  - **Semantic Segmentation**: Employs convolutional neural networks (CNNs) to analyze images at a pixel level and classify them.
  - **Object Detection**: Automatically identifies and maps infrastructure such as traffic signs, crosswalks, and hydrants for accurate mapping.
- **Data Validation and Quality Control:**
  - Ensures the accuracy of uploaded data using crowdsourced reviews. This process improves map reliability and reduces errors.

"Mapillary's end-to end workflow"

Unlocking Spatial Data with APIs

- An API (Application Programming Interface) acts as a bridge, enabling communication between systems and seamless data exchange.
- APIs enable users to connect to complex systems, retrieve data, and integrate it into their own applications.
- By leveraging APIs, Mapillary allows users to its extensive repository of spatial data.



"Illustration of API workflow (Image Source: Rob O'Leary)"

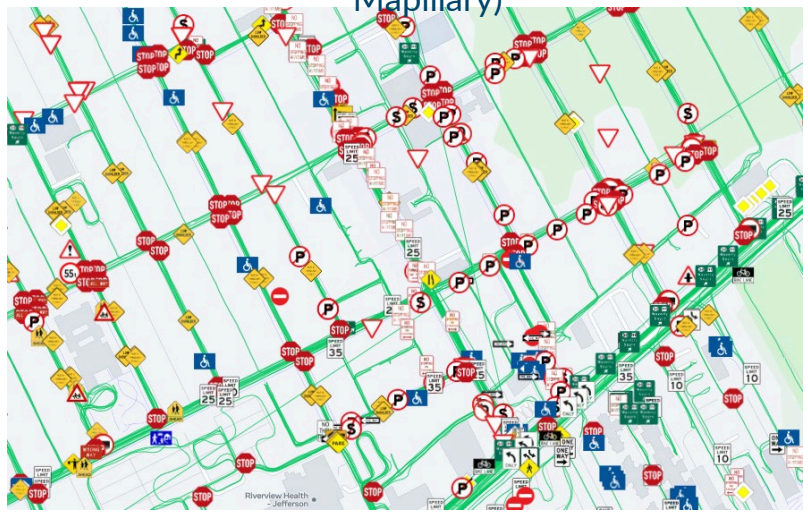
Utilizing the Mapillary API to Retrieve Spatial Data

Introduction to the Mapillary API

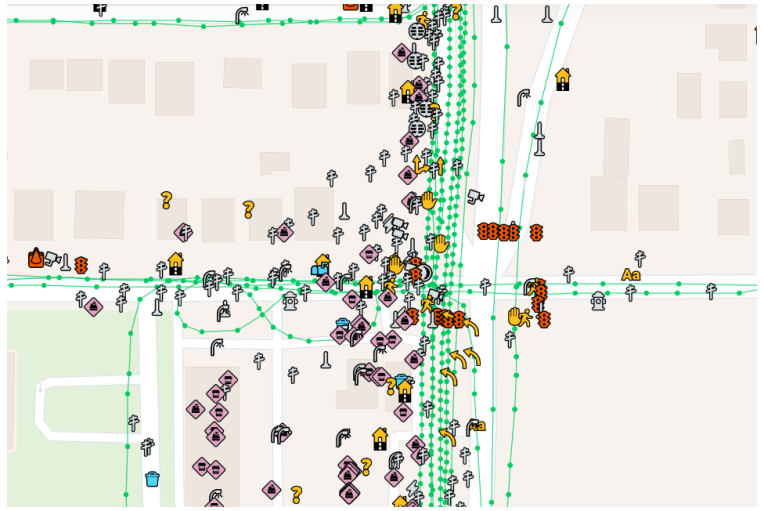
The Mapillary API allows for interaction with Mapillary data in two different forms:

- Vector Tiles – allow for geographic data
  - Coverage Tiles
  - Map Feature Tiles (1,500+ types)
- Entity Endpoints – interact with metadata and images
  - Image
  - Map feature
  - Detection
  - Organization
  - Sequence

Examples of Tile Traffic Signs (credit Mapillary)



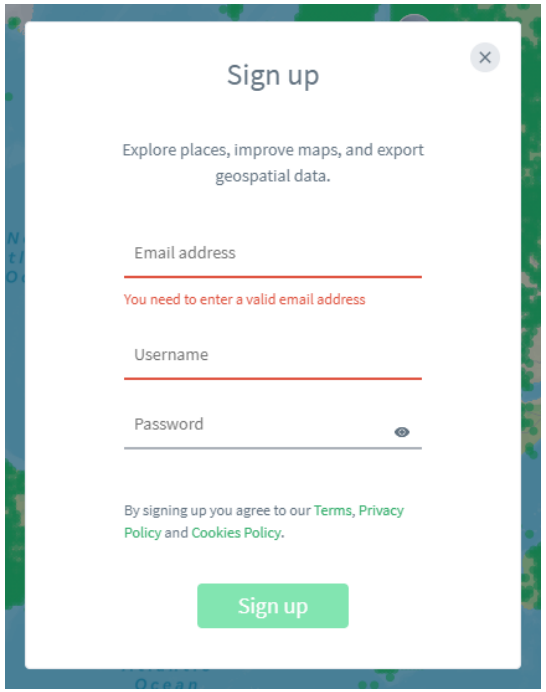
Examples of Tile Points (credit Mapillary)

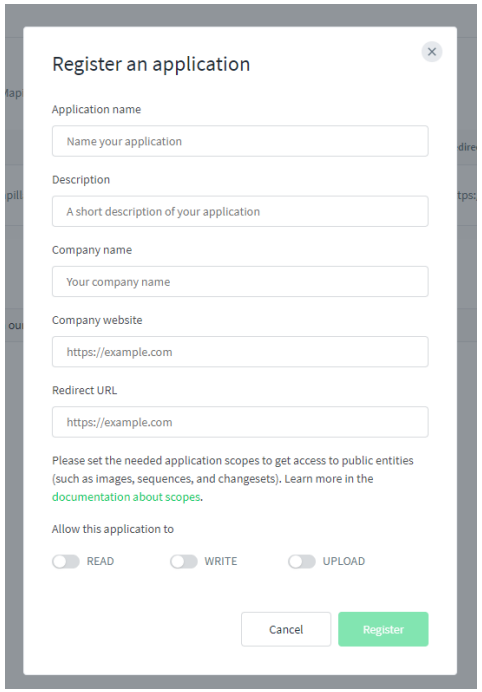


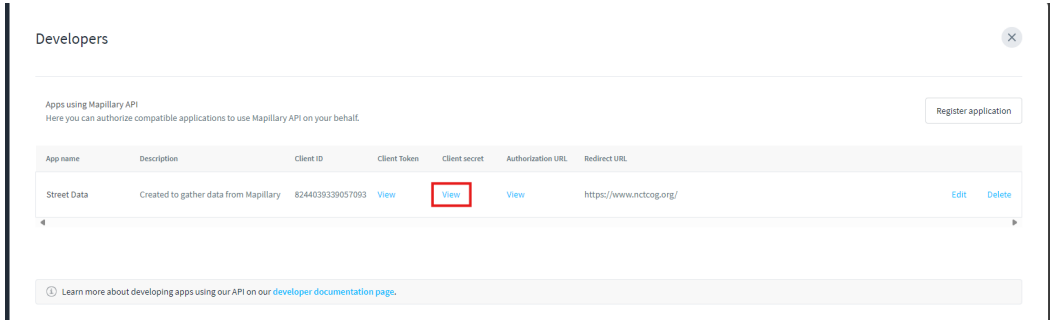
Getting Started with the Mapillary API

To begin using the Mapillary API:

- Sign up to create a new Mapillary account
- Once created, go to your profile settings and click “Developers”
- From there you can register an application with Mapillary that has Read/Write scopes enabled
- Once registered you will be able to view the access token by clicking “view” below “Client Token”
- This token will be used in the script as it allows for authorized requests to be made







Scripting with Mapillary

- Mapillary API scripts vary greatly in complexity, with the simplest ones requiring only the endpoint URL.
- Depending on the requested data type, a certain endpoint URL will be used to make the request.
  - Endpoint for metadata - https://graph.mapillary.com
  - Endpoint for vector tiles - https://tiles.mapillary.com
- Scripting for Mapillary in a python environment requires importing the “requests” and usually the “json”, “Mercantile”, and “mapbox_vector_tile” libraries.

https://tiles.mapillary.com/maps/vtp/{}/2/{}/{}/{}?access_token={}

https://tiles.mapillary.com/maps/vtp/mly_map_feature_traffic_sign/2/14/3729/6578?access_token=MLY|00000000000000000|9856ad9cc629a15337480e7344792d78

Example: Using Mapillary API to Gather All-Way Stop Locations

- Firstly, all relevant libraries are imported:
  - Mercantile – allows for retrieving more than 2,000 features when using a bounding box.
  - Mapbox_vector_tile – allows for the reading and interpretation of vector tiles.
- Then variables are created that will be used to complete the endpoint URL:
  - tile_traffic_signs
  - tile_layer
  - access_token
  - west, south, east, north
  - filter_values
- The mercantile library is then used to create a list of vector tiles that intersect with the search area.

```
import arcpy, requests, mercantile, mapbox_vector_tile, json
from vt2geojson.tools import vt_bytes_to_geojson

# define an empty geojson as output
output= { "type": "FeatureCollection", "features": [] }

# vector tile endpoints -- change this in the API request to reference the correct endpoint
tile_traffic_signs = 'mly_map_feature_traffic_sign'

# tile layer depends which vector tile endpoints:
# 1. if map features or traffic signs, it will be "point" or "traffic_sign" respectively
# 2. if looking for coverage, it will be "image" for points, "sequence" for lines, or "overview" for far zoom
tile_layer = "traffic_sign"

# Mapillary access token -- user should provide their own
access_token = 'MLY|0000000000000000000|9856ad9cc629a15337480e7344792d78'

# a bounding box in [west_lng,_south_lat,east_lng,north_lat] format
west, south, east, north = [-98.060889,32.051951,-95.862486,33.409515]

# list of values to filter for and keep -- update this if changing to traffic signs
filter_values = ['regulatory--all-way--g1']

# get the list of tiles with x and y coordinates which intersect our bounding box
# MUST be at zoom level 14 where the data is available, other zooms currently not supported
tiles = list(mercantile.tiles(west, south, east, north, 14))
```

Example Continued

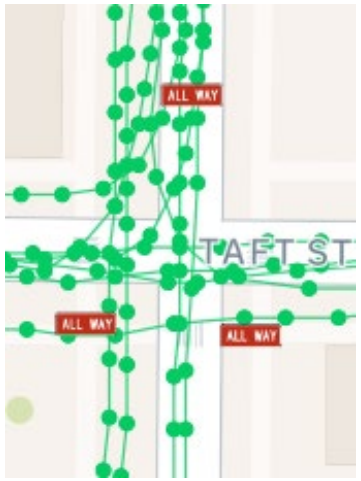
- Loop through all tiles that were previously selected using the variables to send a unique request for each tile
- The retrieved data is then added to “data” which is then filtered by values and geography.
- The filtered features are then appended to output and then saved locally as a geojson.

```
# loop through list of tiles to get tile z/x/y to plug in to Mapillary endpoints and make request
for tile in tiles:
    tile_url = r'https://tiles.mapillary.com/maps/vtp/{}/2/{}/{}/{}?access_token={}'.format(tile_traffic_signs,tile.z,tile.x,tile.y,access_token)
    print(tile_url)
    response = requests.get(tile_url, verify=False)
    data = vt_bytes_to_geojson(response.content, tile.x, tile.y, tile.z,layer=tile_layer)

    # filter for only features matching the values in filter list above
    filtered_data = [feature for feature in data['features'] if feature['properties']['value'] in filter_values]
    print(filtered_data)
    # check if features are inside bbox, and push to output geojson object if yes
    for feature in filtered_data:
        print(feature)
        if (west < feature['geometry']['coordinates'][0] < east) \
            and (south < feature['geometry']['coordinates'][1] < north):
            output['features'].append(feature)
# save a local geojson with the filtered data
with open(r'I:\GIS\GIS Project Assistance\2024\Mapillary API\MapillaryAPIOutput\mydata.geojson', 'w') as f:
    json.dump(output, f)
```

Uploading Data to ArcGIS Pro

- Take the exported geojson and run the tool “JSONToFeatures”
- Snapped multiples to an intersection generated from the TxDOT roadway inventory
- Dissolved to removed redundant duplicate features

```
geoJson = r'I:\GIS\GIS Project Assistance\2024\Mapillary API\MapillaryAPIOutput\mydata.geojson'
mapillaryFeatures = r"I:\GIS\GIS Project Assistance\2024\Mapillary API\mapillaryFeatures"
snappingIntersections = r"I:\GIS\GIS Project Assistance\2024\Mapillary API\MapillaryAPITask\MapillaryAPITask.gdb\TxDOT_Road_Intersections"
outputDissolved = r"I:\GIS\GIS Project Assistance\2024\Mapillary API\MapillaryAPITask\MapillaryAPITask.gdb\myDataDissolved"

arcpy.conversion.JSONToFeatures(geoJson, mapillaryFeatures, "POINT")
arcpy.edit.Snap(mapillaryFeatures, [(snappingIntersections, "END", "125 FEET")])
arcpy.management.Dissolve(mapillaryFeatures, outputDissolved,"", "", "SINGLE_PART")
```



(credit Mapillary)

Resources

- Mapillary
- API Documentation
- Getting Started with the new Mapillary API v4

Questions?

Is anyone else working with Mapillary or other similar resources to update their inventories?

CONTACT US

Alexander Young

North Central Texas Council of Governments

GIS / Data Solutions Analyst I

AYoung@nctcog.org | (817) 704-2501

Dhaval Jariwala

North Central Texas Council of Governments

GIS / Data Solutions Analyst II

djariwala@nctcog.org | (682) 433-0342