

## Micro-module 1: Street view images (SVI) and urban analytics

Understanding the qualities of the built environment is crucial for a wide range of disciplines, including public health research, spatial social studies, real-estate and infrastructure development and urban planning. Street View Imagery (SVI) has gained a strong momentum in urban studies in the last few years, propelled by the proliferation of SVI data, advances in machine learning to extract a variety of information, and the growing computing power to facilitate processing large amounts of data.

This module will cover the whole process from street view data collecting, semantic segmentation processing, geographical visualization using google street view images as an example.

### 1. Street View Images and Application in Urban Analytics

#### - Street view images

Compared with tradition field observation methods and aerial data collected by remote sensing, street view images have the following advantages:

(1) large coverage thanks to omnipresent map service providers; (2) relatively homogeneous quality, sampling, and resolution; (3) free and efficient access to the data; (4) reliable and rich metadata; and (5) capture of the urban scenery from a human perspective.



(a) Aerial perspective.



(b) Street-level point of view.

#### - Platforms for collecting street view images

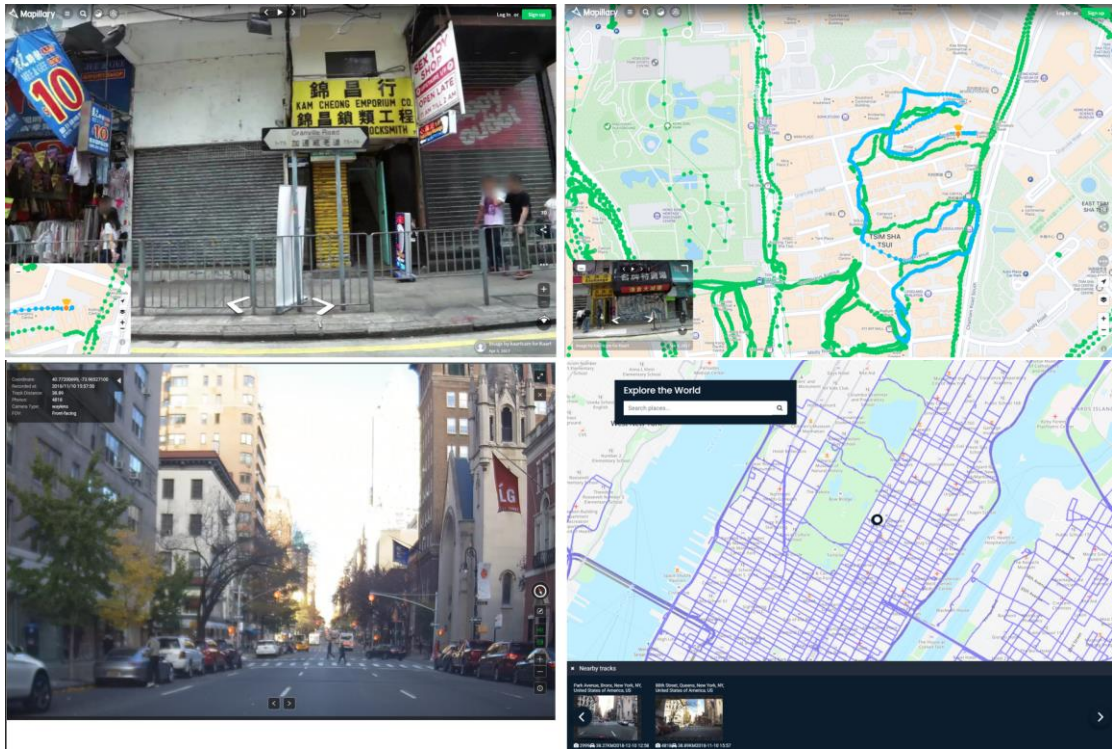
Google Street View Google Street View (GSV) is arguably the most well-known and widespread service providing SVI. Barring rare exceptions such as backpack-mounted cameras to survey narrow roads, the panoramic imagery is acquired in a standardised manner: from a car mounted with multiple cameras on its roof, accompanied with various sensors including lidar.



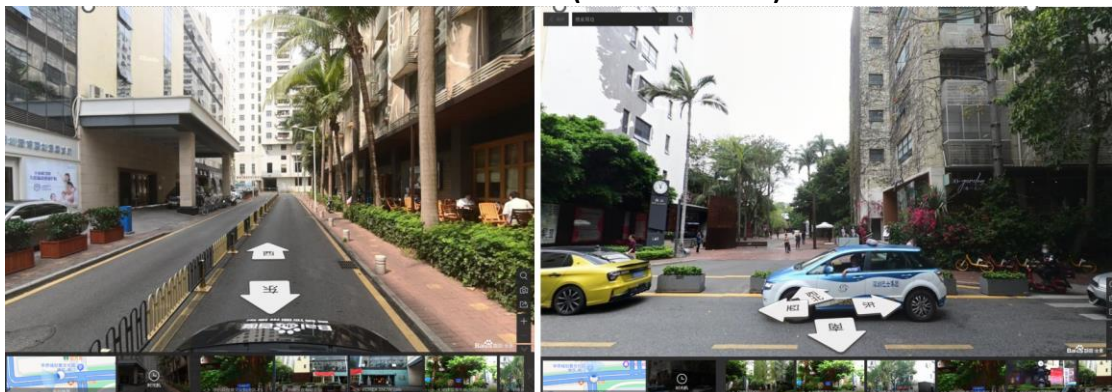
Mapillary and KartaView are the remaining two services with a global focus. They both rely on crowdsourced imagery and are owned and operated by commercial entities. Anyone can contribute to Mapillary and KartaView.

Pros: volunteered, high temporal resolution; views from pavements, cycle tracks and walkways;

Cons: not panoramic, not standard, quality of images (resolution of pixels).



- Tencent Street View and Baidu Total View (Mainland China)



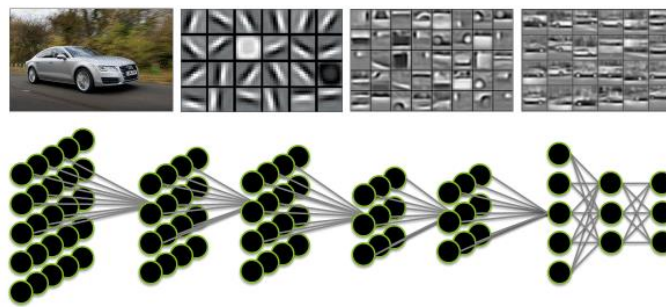
Baidu Maps is a web mapping service provided by Baidu, which can be considered as the counterpart of Google Maps for China. Since 2013 it offers a street view service — Baidu Total View. While the coverage of satellite imagery and maps in Baidu Maps spans beyond China, SVI is available only for China. Tencent Maps is a similar local service, provided by Tencent, and since 2011 it has been offering SVI under Tencent Street View.

- **Application of Street View Images in Urban Analytics**

In general, street view images can be used in research about Spatial data infrastructure, Greenery, Health and well-being, Urban morphology, Transportation and mobility, Walkability, Real estate, and Urban perception.

## 2. Computer Vision Techniques

### HOW A DEEP NEURAL NETWORK SEES



 NVIDIA

Computer vision analyzes images, and then creates numerical representations of what it ‘sees’ using a convolutional neural network (CNN). A CNN is a class of artificial neural network that uses convolutional layers to filter inputs for useful information. The convolution operation involves combining input data (feature map) with a convolution kernel (filter) to form a transformed feature map.



In this work, we present a densely annotated dataset ADE20K, which spans diverse annotations of scenes, objects, parts of objects, and in some cases even parts of parts. This dataset has both outdoor and indoor scenes, and it is typically used by CV communities as training and validation set to check their proposed DL models.

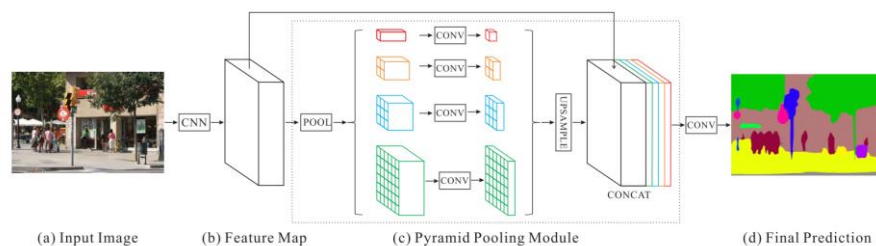


Figure 3. Overview of our proposed PSPNet. Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d).

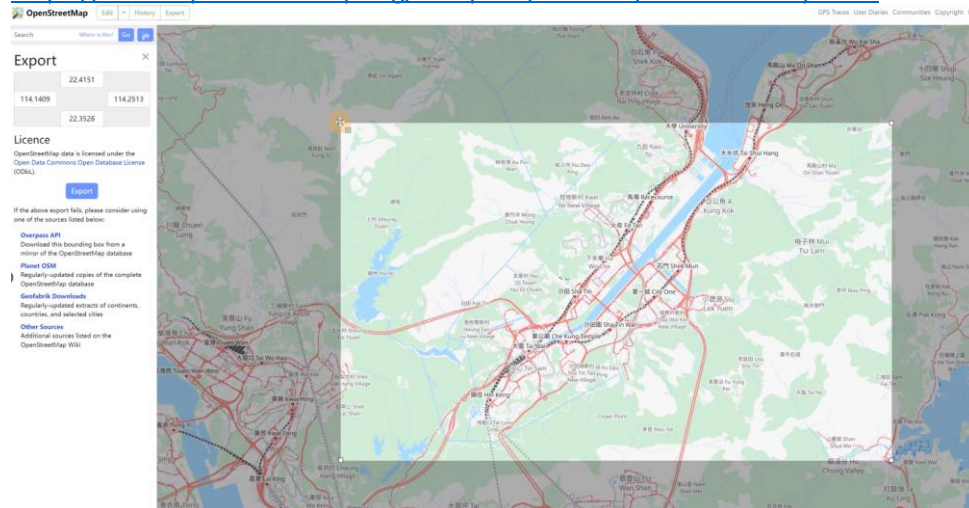
PSPNet, the Pyramid Scene Parsing Network makes great contribution to the improvement of CV. It adds a unique Pyramidal Pooling Module to the algorithm, and was able to improve the prediction accuracy compared to previous Fully Connected

Convolutional Models. And it is relatively fast, and considers the context of each pixels for better semantic segmentation. As you could see in this image above to the right, it is able to differentiate the pillows from the quilts, although it has a similar texture and color. Therefore, this algorithm has been used in many emerging studies.

### 3. Data Preprocessing

- **download OSM data**

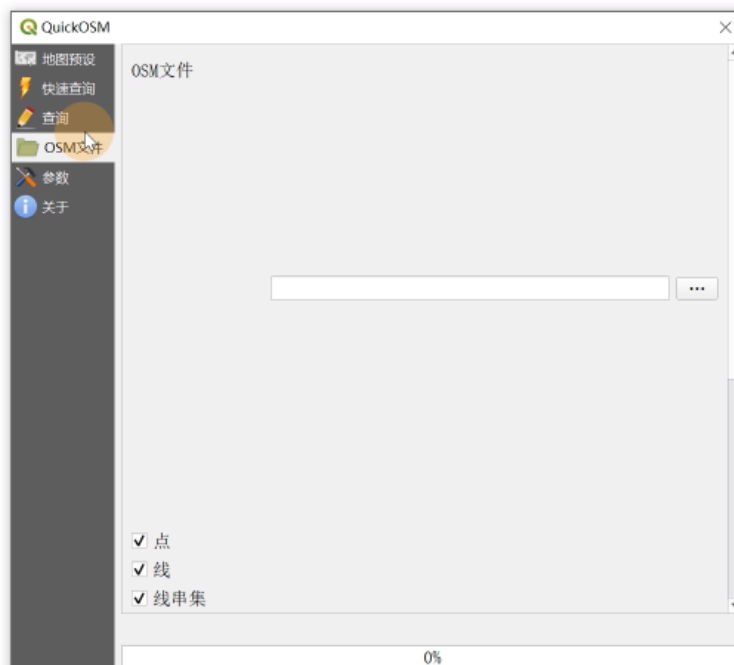
You can download OSM data via <https://www.openstreetmap.org/#map=15/22.3311/114.1585&layers=T>.



- **import OSM data**

open QuickOSM plugin, choose import from file, choose the path of your osm file, select all features and click OK.

For this case, we only use the polyline layers. You can remove other layers.



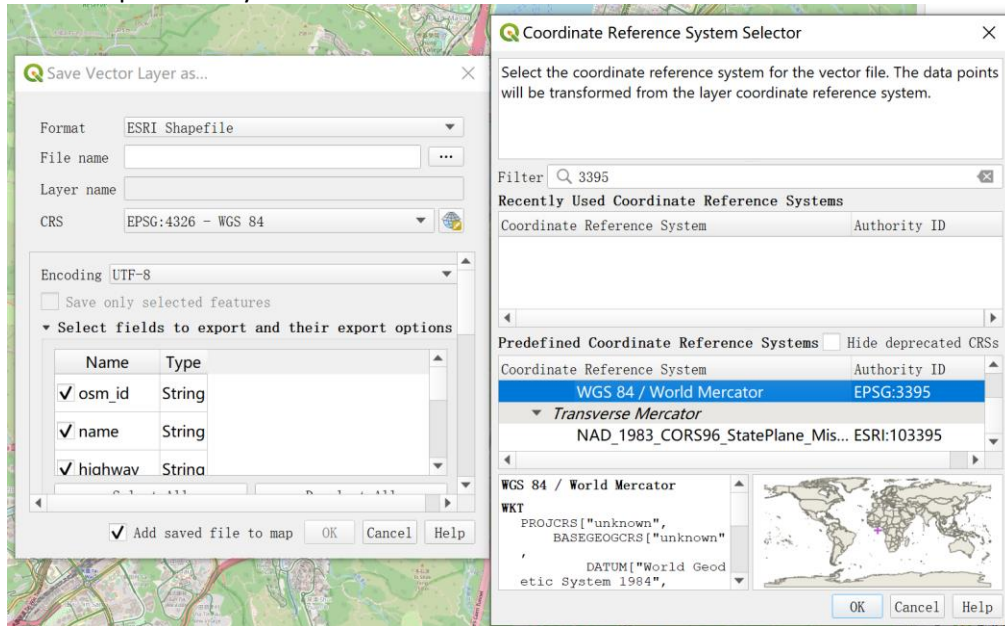
- **Clean the road network**

Use filter (right click the layer to active the menu), and you can copy paster the following syntax:

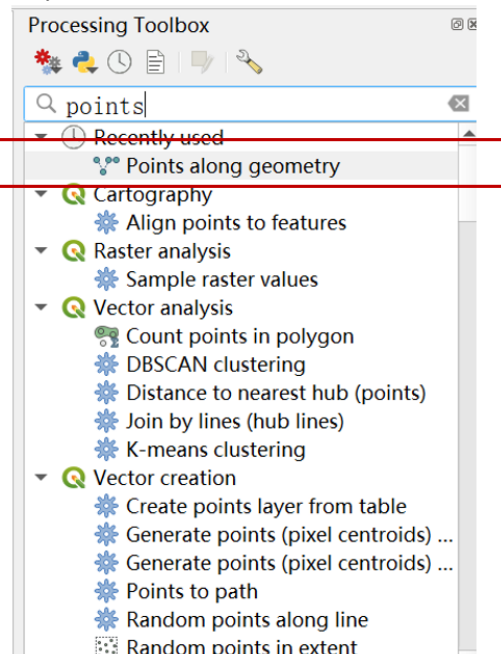
```
"highway" = 'primary' OR "highway" = 'residential' OR "highway" = 'secondary' OR
"highway" = 'tertiary' OR "highway" = 'tertiary_link' OR "highway" = 'trunk' OR
"highway" = 'trunk_link' OR "highway" = 'unclassified'
```

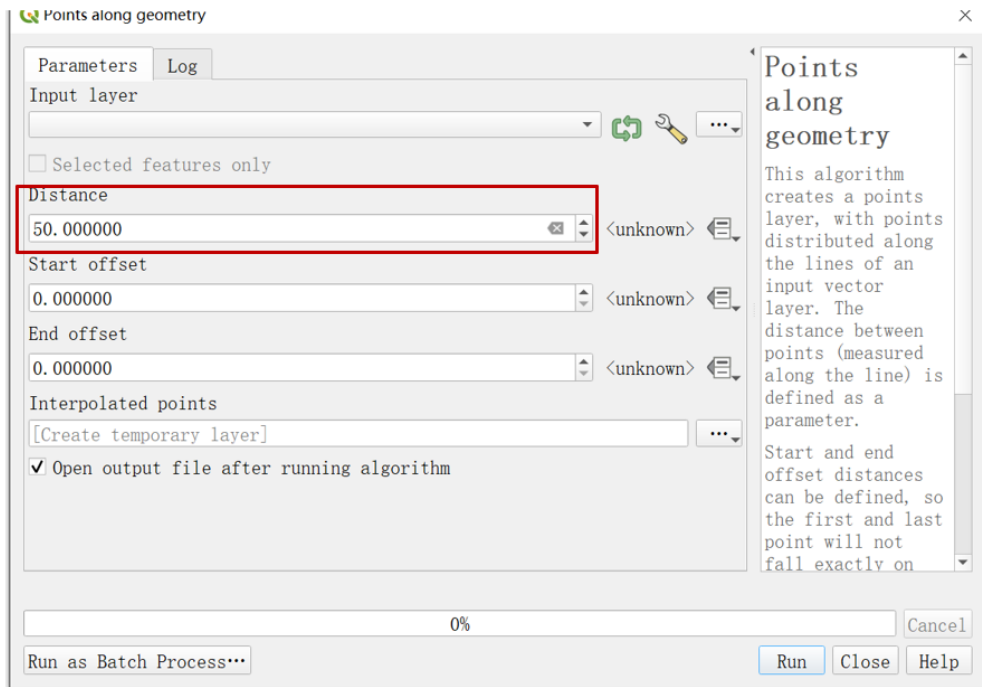
- **Create Sampling Points**

We need to make sure the unit of the layers is meter (not degree, instead). Here we save a duplicated layer and set the CRS as '3395'.



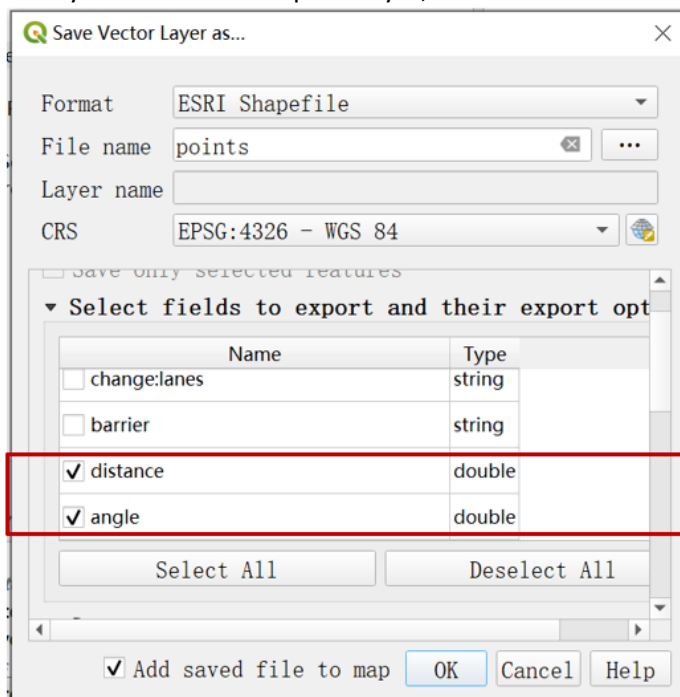
Then we use the function 'Points along geometry'. You can set your own interval. We usually use 50, 100, 200 meters as the interval.





- **Relocated the point layer**

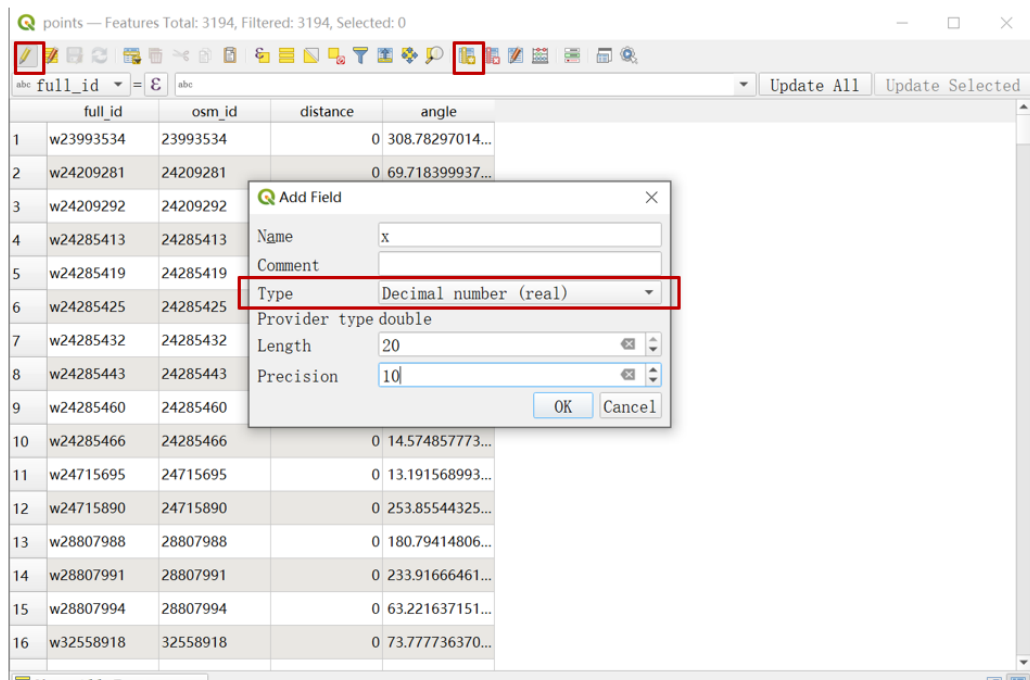
We may need to save the point layer, and relocated it with the CRS 4326.



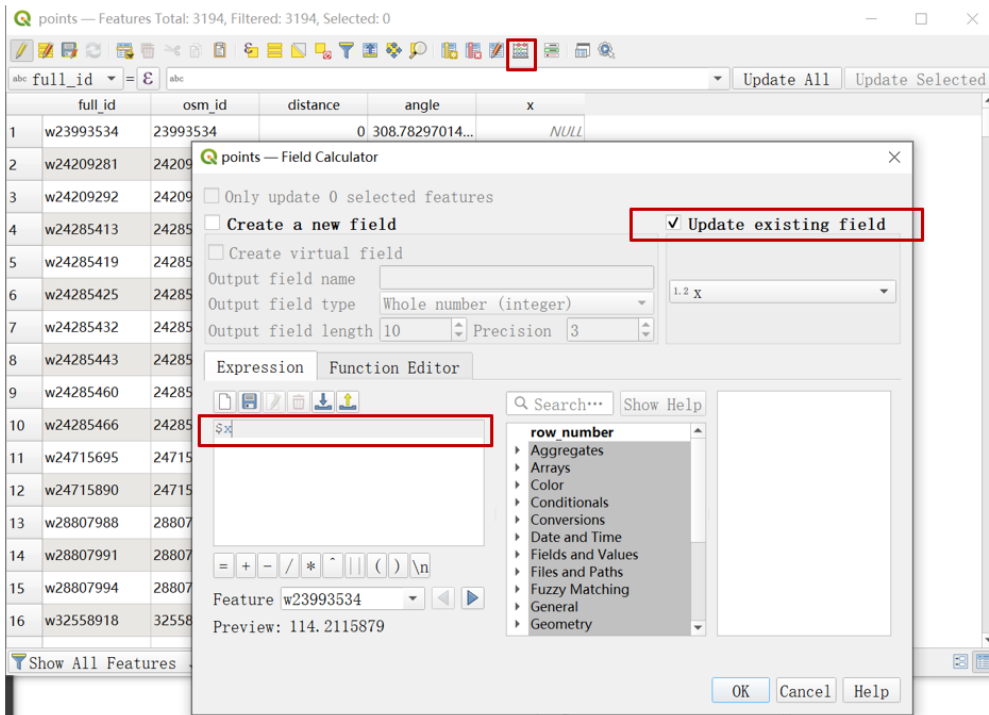
- **Add coordinate information and ID for each point**

As this layer is used to search the google street view images at each location. But for now, we can check the attribute table, no information of the specific longitude and latitude are included.

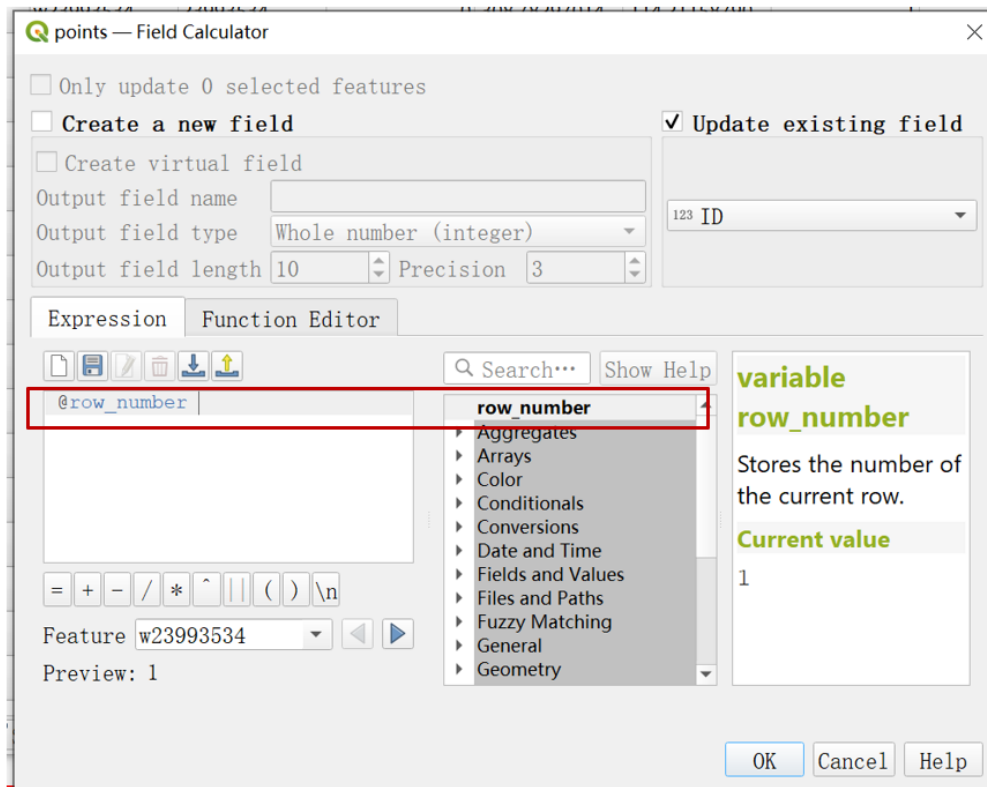
We can active the toggle editing, select 'add filed'. Do the same for x and y columns.



Then open the field calculator, we use '\$x' and '\$y' function, and you need to select 'update existing field'.



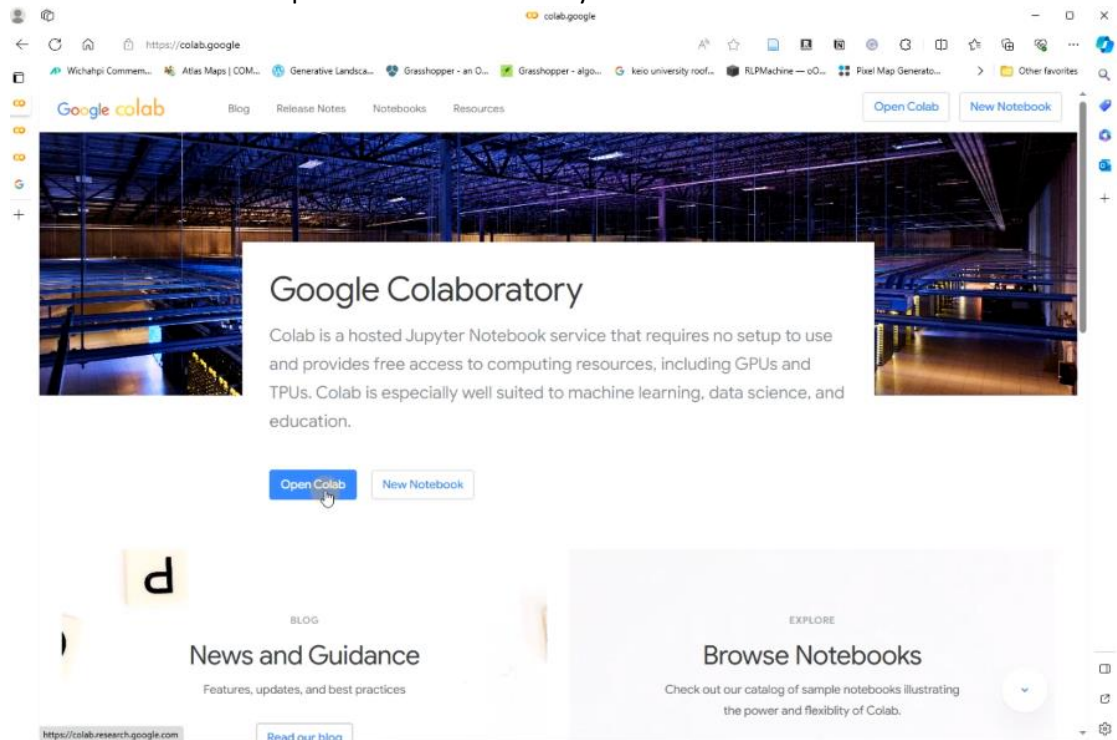
Using the same function, create a new column named 'ID', and this time, we use '@row\_number' function in field calculator.



#### 4. GSV Data Collecting

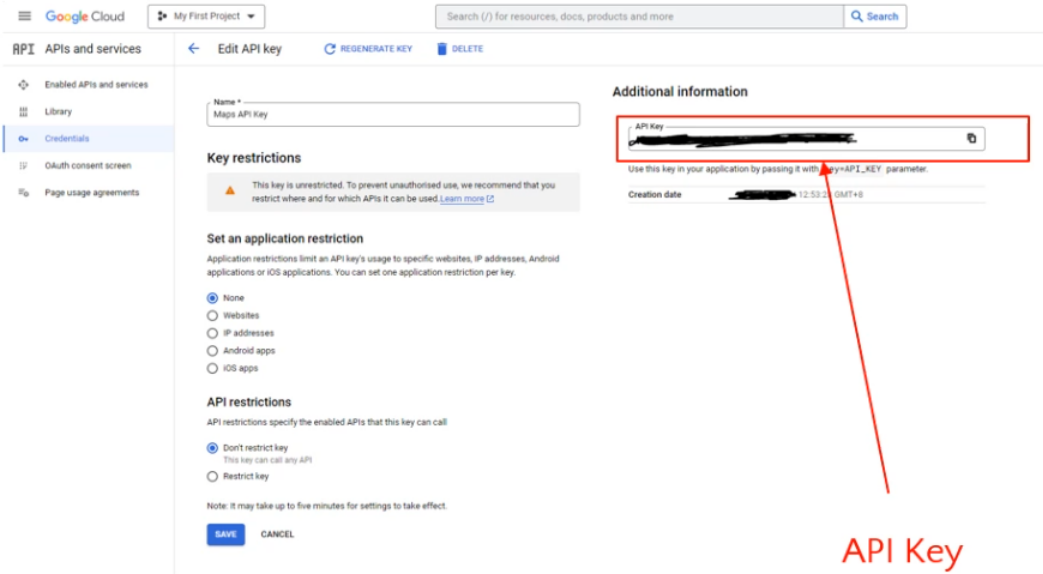
##### - Loading the code into Google Colab Notebook

Here, we use Google Colab to open our shared code, the platform is relatively simple and convenient to set up the environment for Python code.



Then, in order to request SVI from the Google Street View API, you need to register a Google API Key (which can be found in the Google Cloud).





And we should also be aware of the monthly free quota given to the Google Account to avoid overuse by accident that could incur additional charge to your credit card account.

#### Pricing for the Street View Static API

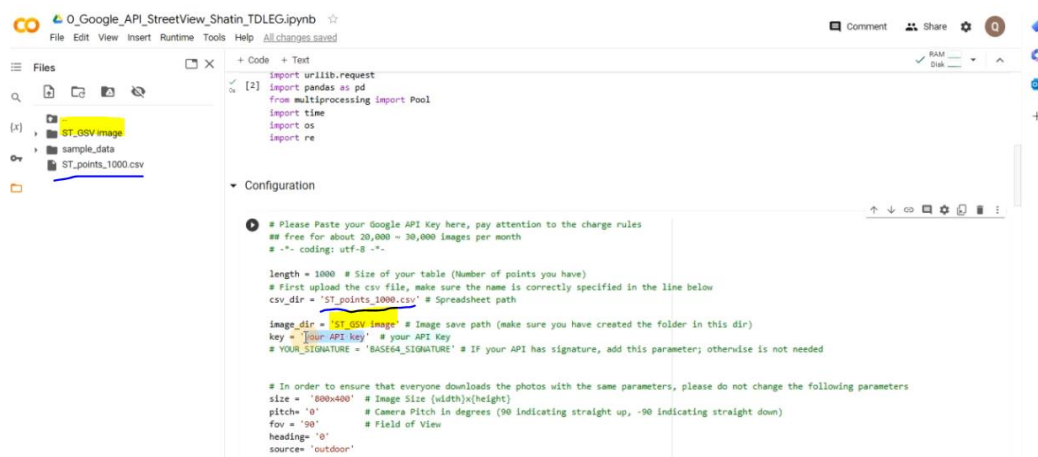
##### SKU: Static Street View

Street View panoramas and map loads are now charged separately. A static Street View panorama is charged for each request to the [Street View Static API](#) to embed a static (non-interactive) Street View panorama. Usage of the [Street View Image Metadata](#) endpoint is not charged.

MONTHLY VOLUME RANGE (Price per PANORAMA)		
0–100,000	100,001–500,000	500,000+
0.007 USD per each (7.00 USD per 1000)	0.0056 USD per each (5.60 USD per 1000)	<a href="#">Contact Sales</a> <a href="#">🔗</a> for volume pricing

#### - Configuring the basic setup and specifying parameters

Here, we need to upload the csv file to the left panel area, and also create a folder 'ST\_GSV\_Image' to receive the requested street views. Furthermore, add your API key to this line of code below. Then run the cell.



Here, we need to upload the csv file to the left panel area, and also create a folder 'ST\_GSV\_Image' to receive these images.

When requesting images, there are a few parameters that you could specify, for

instance, the heading indicates the compass heading of the camera. Accepted values are from 0 to 360 (both values indicating North, with 90 indicating East, and 180 South).

Size: this indicates the exact image size you want to download, for instance, 600 pixels by 400 pixels

fov: (default is 90) determines the horizontal field of view of the image expressed in degrees, with a maximum allowed value of 120.

pitch: (default is 0) specifies the up or down angle of the camera relative to the Street View vehicle.

Source: it is either default or outdoor, if it is outdoor, it will search the scenes that were taken outdoors.

```

Configuration

# Please Paste your Google API Key here, pay attention to the charge rules
## free for about 20,000 ~ 30,000 images per month
# -*- coding: utf-8 -*-

length = 1000 # Size of your table (Number of points you have)
# First upload the csv file, make sure the name is correctly specified in the line below
csv_dir = 'ST_points_1000.csv' # Spreadsheet path

image_dir = 'ST_GSV_image' # Image save path (make sure you have created the folder in this dir)
key = 'your API key' # your API Key
# YOUR_SIGNATURE = 'BASE64_SIGNATURE' # IF your API has signature, add this parameter; otherwise is not needed

# In order to ensure that everyone downloads the photos with the same parameters, please do not change the following parameters
size = '800x400' # Image Size (width)x(height)
pitch = '0' # Camera Pitch in degrees (90 indicating straight up, -90 indicating straight down)
fov = '90' # Field of View
heading = '0'
source = 'outdoor'

[ ] def download(url, save_dir, ID):
    try:
        conn = urllib.request.urlopen(url)
    except Exception as e:
        print('wrong', str(ID), e)
        return 0
    f = open(save_dir, 'wb')
    f.write(conn.read())
    f.close()
    print('image save', str(ID))
    return 0
  
```

Then we present the csv data to confirm the data frame it is loading.

```

Read SVI sampling points

# understand SVI sampling
data = pd.read_csv(csv_dir)
data_of_your_area = data
data_not_downloaded = data_of_your_area[~ data_of_your_area.ID.isin(filepath_downloaded)]
data2 = data_not_downloaded.reset_index(drop=True)

# now it is time to print a snapshot of the csv file to understand how it looks and all the column names.
data.head(length)
  
```

	ID	turn	surface	oneway	maxspeed	angle	latitude	longitude	highway	Name
0	1	NaN	asphalt	yes	70.0	174.686492	22.373663	114.214026	motorway	Sha TinIn
1	2	NaN	asphalt	yes	70.0	209.859314	22.357963	114.173269	motorway	Sha TinIn
2	3	NaN	asphalt	yes	80.0	249.977709	22.348256	114.153362	motorway	Sha TinIn
3	4	NaN	NaN	NaN	NaN	157.932289	22.375369	114.180683	cycleway	Sha TinIn
4	5	NaN	NaN	NaN	NaN	37.913937	22.428734	114.208534	cycleway	Sha TinIn
...	...	...	...	...	...	...	...	...	...	...
995	996	NaN	asphalt	yes	NaN	41.302881	22.398489	114.192823	tertiary	Sha TinIn
996	997	NaN	NaN	NaN	NaN	309.284687	22.430405	114.245649	residential	Sha TinIn
997	998	NaN	NaN	yes	80.0	56.776369	22.378425	114.204267	motorway	Sha TinIn
998	999	NaN	asphalt	yes	NaN	280.058160	22.399924	114.198714	tertiary	Sha TinIn
999	1000	NaN	NaN	yes	80.0	227.347201	22.405429	114.220528	motorway	Sha TinIn

1000 rows | 10 columns

We also need to make sure the names called in these following cells could perfectly match the respective column names (e.g., latitude) read from the csv file. After making them consistent, we could successfully run these cells.

```

print('total image you need to download:',len(data_of_your_area))
print('have downloaded:', len(filepath_downloaded))
print('have not downloaded:',len(data2))

total image you need to download: 1000
have downloaded: 0
have not downloaded: 1000

▼ Appending Data to request url

run_list = []
for i in range(len(data2)):
    ID = data2.loc[i, 'ID'] # Sampling point number

    latitude = data2.loc[i, 'latitude'] # Latitude ; make sure the name corresponds with the csv file.
    longitude = data2.loc[i, 'longitude'] # Longitude; make sure the name corresponds with the csv file.

    heading = data2.loc[i, 'angle'] # Road/Street Angle;needed to manually change the angle number in the excel if you want to use other angle(90,2

    image_save_dir = image_dir + '/' + str(ID) + '.jpg'

    url = 'https://maps.googleapis.com/maps/api/streetview?size='+ size + '&location=' + str(latitude) + ',' + str(longitude) + '&key='+ key +

    # url = "https://maps.googleapis.com/maps/api/streetview?size=" + size + "&location=" + str(latitude) + "," + str(longitude) + "&heading=" + str
    run_list.append((url,image_save_dir,ID))
  
```

If everything is correctly specified, then we could start downloading the images into the image folder. Patiently wait until all the images are downloaded.

```

▼ Downloading Images

# In case of errors such as "403 Forbidden", most likely your Google API key is incorrect or problematic
print('start')

for k in range(len(run_list)):
    # for k in range(0,10):
    url_run = run_list[k][0]
    save_dir_run = run_list[k][1]
    id_run = run_list[k][2]
    download(url_run,save_dir_run, id_run)

print('finish')

start
image save 1
image save 2
image save 3
image save 4
image save 5
image save 6
image save 7
image save 8
image save 9
image save 10
image save 11
image save 12
image save 13
image save 14
image save 15
image save 16
image save 17
image save 18
  
```

- **Saving and downloading the collected Street View Images**

Here, we need to run the last cell in this Python notebook to mount your own Google Drive to the left side, the reason is that when you have many downloaded images saved to the temporary storage folder shown to the left panel, it is not possible to download the entire folder, you could only save each image one by one.

Therefore, after mounting your own Drive, you could drag the image folder to the Drive folder, and you could go back to your own Google Drive to download these images there.



```

[12] #this line of code has some issue, need to be fixed.
print('have not downloaded:',len(data2))

total image you need to download: 1000
have downloaded: 107
have not downloaded: 1000

Mount to google drive and manually move into that main folder, and then download the entire image folder from google drive. remember to refresh

from google.colab import drive
drive.mount('/content/drive')

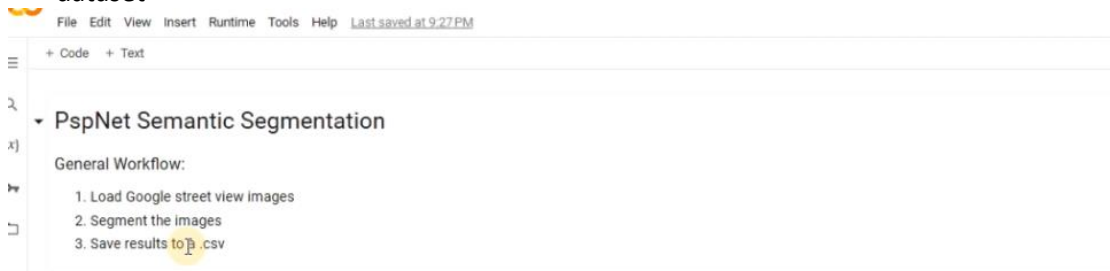
Mounted at /content/drive

Please also manually check how many photos have been downloaded to avoid duplicates
  
```

**5. Semantic Segmentation Using ADE20K and PSPNet**

- **Loading the ADE20K dataset**

Here, we need to read the labels from the pre-determined and widely used ADE20K dataset



▼ Defining the Color Palette for different street features

Use the ADE20K pretrained dataset

```
[ ] # Defining Colour Palette
from collections import namedtuple
import scipy.io
Label = namedtuple('Label', ['name', 'id', 'color']) # Colour is in RGB format
labels_ade = [Label('wall', 0, (120, 120, 120)),
              Label('building', 1, (180, 120, 120)),
              Label('sky', 2, (6, 230, 230)),
              Label('floor', 3, (80, 50, 50)),
              Label('tree', 4, (4, 200, 3)),
              Label('ceiling', 5, (120, 120, 80)),
              Label('road', 6, (140, 140, 140)),
              Label('bed', 7, (204, 5, 255)),
              Label('windowpane', 8, (230, 230, 230)),
              Label('grass', 9, (4, 250, 7)),
              Label('cabinet', 10, (224, 5, 255)),
              Label('sidewalk', 11, (235, 255, 7)),
              Label('person', 12, (150, 5, 61)),
              Label('earth', 13, (120, 120, 70)),
              Label('door', 14, (8, 255, 51)),
              Label('table', 15, (255, 6, 82)),
              Label('mountain', 16, (143, 255, 140)),
```

- **Install packages and load image data**

Here, we need to pip install related libraries and CV modules

▼ 2. Generate the downloaded jpg list

```
[ ] # pip install library
!pip install mxnet

Requirement already satisfied: mxnet in /usr/local/lib/python3.10/dist-packages (1.9.1)
Requirement already satisfied: numpy<2.0.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from mxnet) (1.23.5)
Requirement already satisfied: requests<3,>=2.20.0 in /usr/local/lib/python3.10/dist-packages (from mxnet) (2.31.0)
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from mxnet) (0.8.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet) (2.0.7)
Requirement already satisfied: certifi<2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.20.0->mxnet) (2023.7.22)

[ ] !pip install gluoncv

Requirement already satisfied: gluoncv in /usr/local/lib/python3.10/dist-packages (0.10.5.post0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from gluoncv) (1.23.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gluoncv) (4.66.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from gluoncv) (2.31.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from gluoncv) (3.7.1)
Requirement already satisfied: portalocker in /usr/local/lib/python3.10/dist-packages (from gluoncv) (2.8.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from gluoncv) (9.4.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from gluoncv) (1.11.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from gluoncv) (3.7.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from gluoncv) (1.5.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from gluoncv) (6.0.1)
Requirement already satisfied: autocf in /usr/local/lib/python3.10/dist-packages (from gluoncv) (0.0.0)
```

Here, we need to mount the google drive and read the image folder located in the Drive or we could simply upload the image folder directly to this temporary storage area to the left panel.

```

1_CV_Shatin_workshop.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Files
- drive
- sample_data
[3] Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (4.44.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib-gluoncv) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas-gluoncv) (2023.3.post1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests-gluoncv) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests-gluoncv) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests-gluoncv) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests-gluoncv) (2023.7.22)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib-gluoncv) (1.16.0)
Installing collected packages: yacs, portalocker, autocfg, gluoncv
Successfully installed autocfg-0.0.6 gluoncv-0.10.5.post0 portalocker-2.0.2 yacs-0.1.8

[4] #Use a path from Google drive could help you load the images more easily
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

# Image path config
path='/content/drive/MyDrive/SV_GSV_Images' # Change the path to the folder where you put Street view images in the google drive
# or you could just simply use a folder location that you uploaded temporarily to this notebook.
#path = 'foldername' #no need for slashes

print("Import GSV photo path is: %s"%path)
Import GSV photo path is: /content/drive/MyDrive/SV_GSV_Images/

If everything is working the cell below should show how many images are in the folder and the name of the first two images.
  
```

Then, we add modules and libraries that we need.

```

[8] # Generate jpg list
from os import listdir

from mxnet import image
from os.path import isfile, join
import pandas as pd

all_files = [f for f in listdir(path)]
### Get only text files
jpg_files = list(filter(lambda x: x[-5] == ('.jpeg') or x[-4] == ('.jpg'), all_files))
jpg_files.sort()

#print(jpg_files)
print("1.Total Image Files in the folder:",len(jpg_files))

#show the first ten images
print(jpg_files[0:10])

1.Total Image Files in the folder: 1000
['1.jpg', '10.jpg', '100.jpg', '1000.jpg', '101.jpg', '102.jpg', '103.jpg', '104.jpg', '105.jpg', '106.jpg']
  
```

2.1 Test whether the images can be called and what are there format

```

If everything is working the cell below should show image shape: (400, 640, 3)

### 2.1 Get image
for filename in jpg_files[10:11]:
    print((filename.split('.')[0]))
### 1. Read the jpg file
img = image.imread(path+filename)
print(filename, 'image shape:',img.shape)

107
107.jpg image shape: (400, 640, 3)
  
```

- **Applying the PSPNet and defining the features we want to extract**  
Here, we define the street elements we want to extract based on the labels from ADE20K.

3. Apply PSPNet to images and generate a csv

```

# Apply PSPNET to images and generate a .csv
import mxnet as mx
from mxnet import image
from mxnet.gluon.data.vision import transforms
import gluoncv

from datetime import datetime
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
import matplotlib.patches as mpatches
from matplotlib.font_manager import FontProperties
import numpy as np
import collections
from gluoncv.utils.viz import get_color_pallette

# Define the features you want to detect
keywords=['mountain','water','lake','sky','bridge','pier',
'ceiling','building','skyscraper','wall','fence','windowpane','glass',
'tree','grass','plant','road','sidewalk','earth',
'person','minibike','bicycle','car','van',
'sofa','chair','booth','fountain','railing','signboard','column','awning','desk','lamp',
'streetlight','sculpture','ashcan','bulletin board']
  
```

And we could then create two folders, one for the segmented images without labels, and the other containing labels. After getting the pre-trained model, we simply run this cell.

```

! PLEASE CHECK BEFORE RUNNING FOLLOWING CELL !

Make sure you have created a folder to save processed segmentation results without labels

Also, you have created a separate folder to save processed segmentation result WITH labels

Remember to set path_out and path_out_lgd to the respective folders

# Process Using CPU, and Define Folders to save in
ctx = mx.cpu(0)
path_out="1" # The Folder you put segment results without labels
path_out_lgd="/content/drive/MyDrive/ST_GSV_PSPOutLabeled/" # The Folder you put segment results with a legend

# Define a dataframe to save analysis results
df_pspnet=pd.DataFrame()

# Get pre-trained model | PSPNET ADE
model = gluoncv.model_zoo.get_model('psp_resnet101_ade', pretrained=True)
# Set up legend font
fontP = FontProperties()
fontP.set_size('x-small')
#print(jpg_files)
print("1.Image Files in the folder:",len(jpg_files))

Downloading /root/.mxnet/models/resnet101_v1s-bd93a83c.zip from https://apache-mxnet.s3-accelerate.dualstack.amazonaws.com/gluon/models/resnet101_v1s-bd93a83c.zip: 100% |#####| 266773/266773 [00:10<00:00, 26095.05KB/s]
1.Image Files in the folder: 1000
  
```

Next, define the start and end index of the images.

```

!!! Please check before running the following cell

Change the number assigned to start_image_index, and end_image_index to segment smaller batches of images

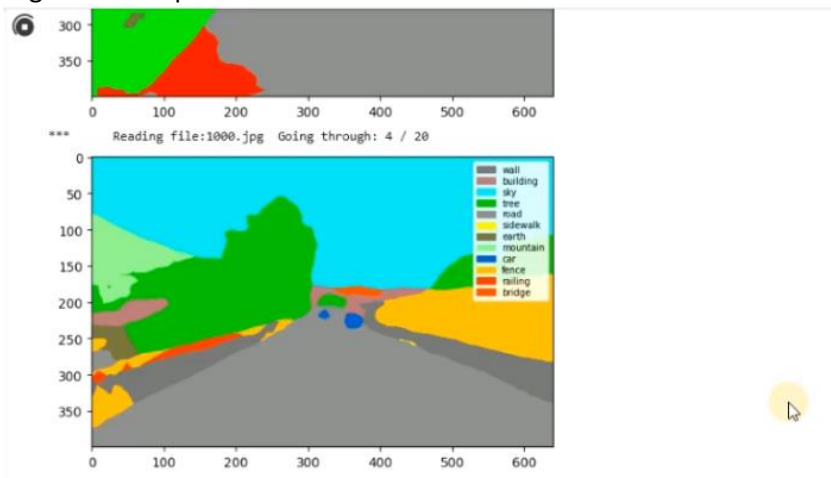
Please refer to the output from cells above for the numbers of available images in your folder

# Change file numbers to be segmented
start_image_index = 0 # Images before this index will not be processed, first image in folder is index 0
end_image_index = 20 #the image at & after this index will not be processed, in our case we have 1k images

ID=0
for filename in jpg_files[start_image_index:end_image_index]:
    # 1.Read the jpg file
    img = image.imread(path+filename)
    #print(path+filename,img.shape)
    size=img.shape[0]*img.shape[1]
    # print("1",img.shape)
    # # Display origin image
    # plt.imshow(img.asnumpy())
    # plt.show()

    # Normalize the image using dataset mean
    transform_fn = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([.485, .456, .406], [.229, .224, .225])
    ])
    img = transform_fn(img)
    # print("2",img.shape)
    img = img.expand_dims(0).as_in_context(ctx)
  
```

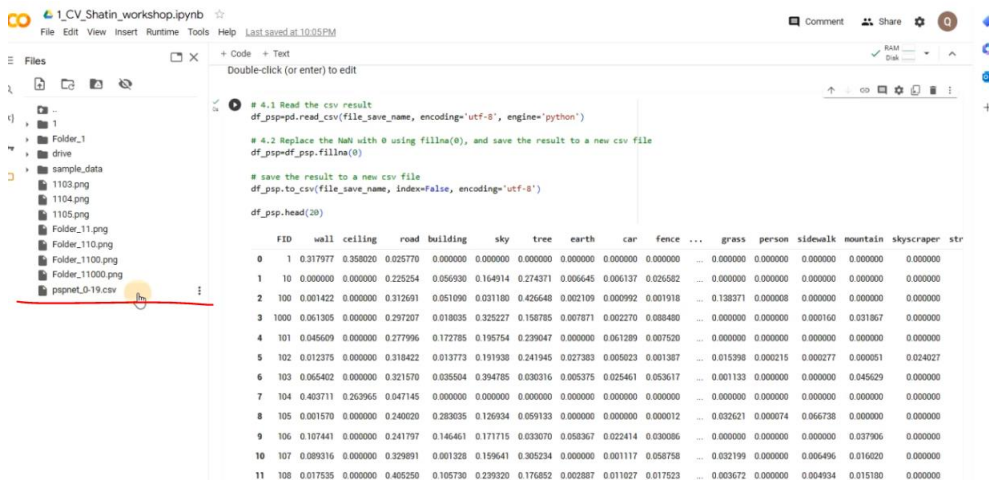
If all previous steps are run successfully, the segmented results can be obtained, and you will see the progress. Each image may need 10-20 seconds if you are using CPU, typically GPU is more powerful in terms of computing power and could accelerate this segmentation process.



Double-click (or enter) to edit

Remember to run the last piece of the code, as it saves the ratio of each street element

(also called visual index in urban studies literature) you chose previously for all the images into a combined csv file, that you could use to match back to the QGIS for future use.



```

# 4.1 Read the csv result
df_psp=pd.read_csv(file_save_names, encoding='utf-8', engine='python')

# 4.2 Replace the NaN with 0 using fillna(0), and save the result to a new csv file
df_psp=df_psp.fillna(0)

# save the result to a new csv file
df_psp.to_csv(file_save_name, index=False, encoding='utf-8')

df_psp.head(20)

```

	FID	wall	ceiling	road	building	sky	tree	earth	car	fence	...	grass	person	sidewalk	mountain	skyscraper	str
0	1	0.317977	0.358020	0.025770	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	10	0.000000	0.000000	0.225254	0.056930	0.164914	0.274371	0.006645	0.006137	0.026582	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

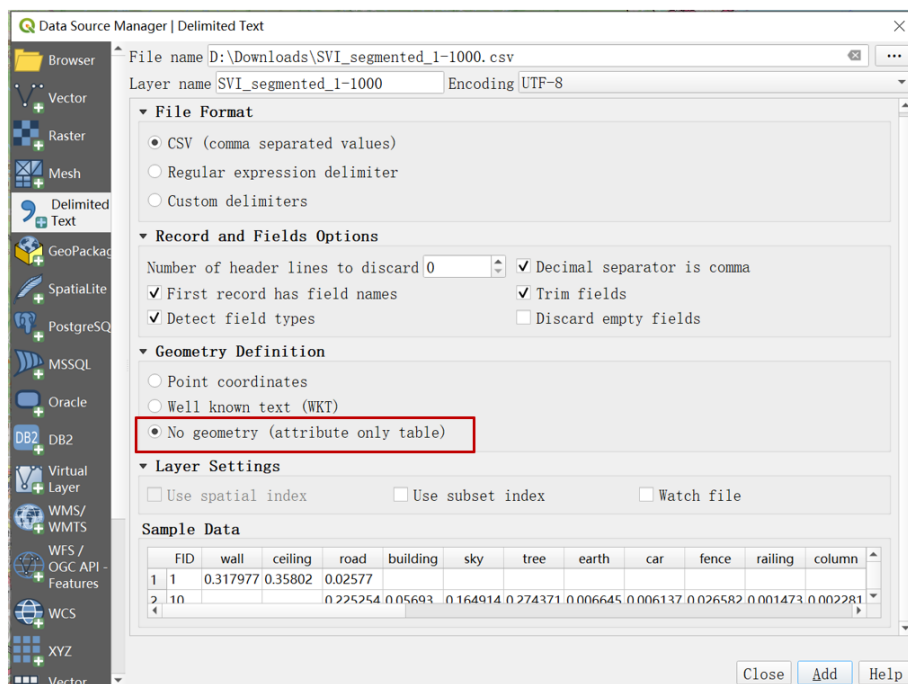
## 6. Data Visualization

### - Add 'delimited text layer'

Here we use two sample files.

First, we import 'st\_points.csv'. You can notice that this layer has coordinate information. We import it as a point layer.

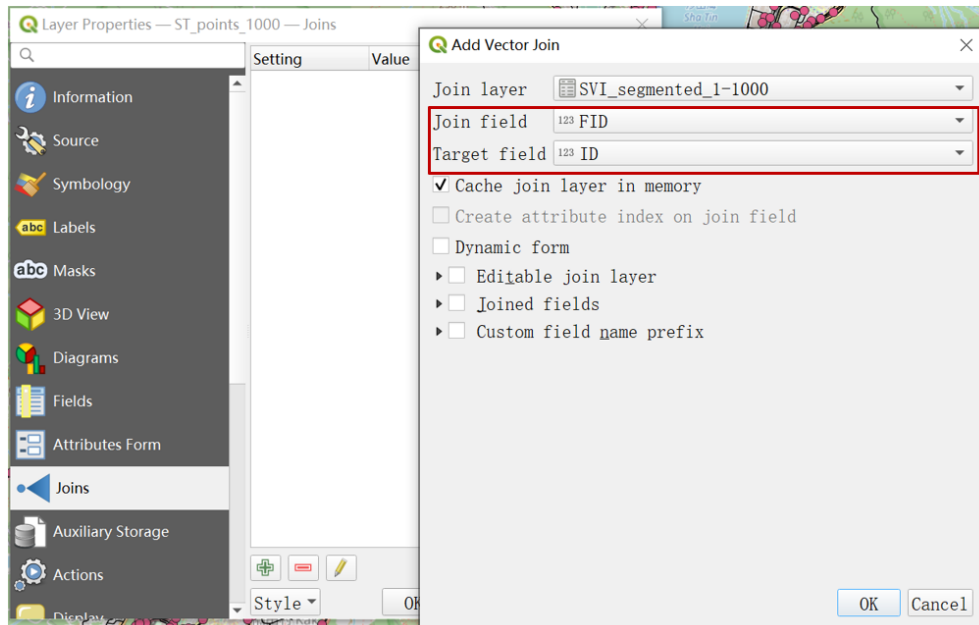
After that, we import 'st\_segmented\_1-1000.csv'. This time we choose 'no geometry'.



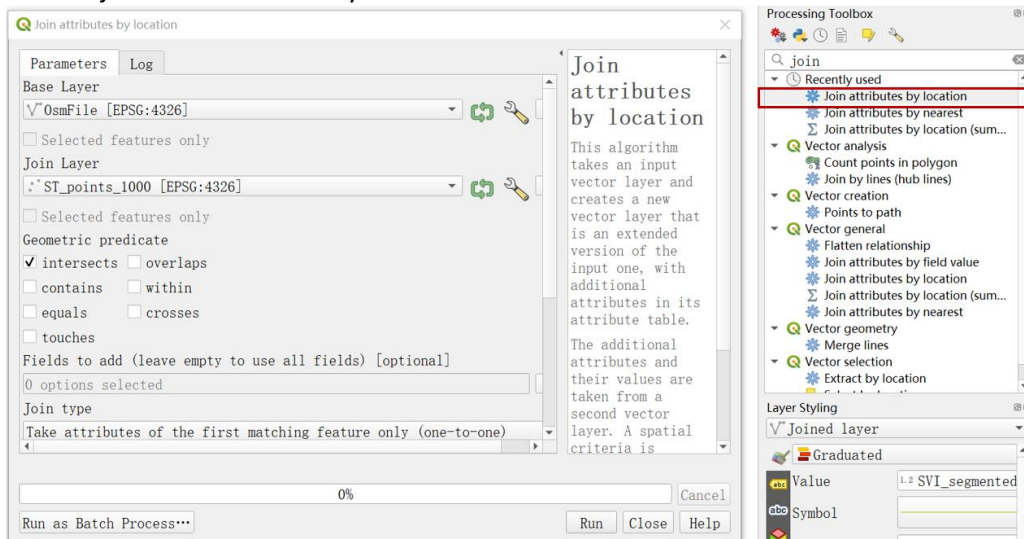
### - Join tables

For this case, we want to add the values of different visual indices into the point layer. For this purpose, we can notice that there are columns have equal values. 'ID' in the point layer and 'FID' in the segmented result.

Right click the point layer, find 'properties', click 'Joins'. The join layer is the segmentation result. FID is the join field; ID is the target field. We can select joined fields, which depends on your tasks.



- **Assign value to road sections**  
We use join attribute table by location function.



- **Visualization**  
To visualize the one parameter by heatmap. You can right click the newly join layer, find 'Properties', select 'Symbology'. You can either use categorized or graduated. Select the parameter you take as the target, click 'classify'. You can change the classes and mode for classification. You may need to try multiple intervals to achieved a nicer visual :).