Satellite Imagery DataSet Toolkit

The Satellite Imagery DataSet is important part to train, validation the model of different mission. This toolkit work for download different datasources and use specific layer (class) in OSM Vector Data to generate dataset for train or validation model.

Support Vector Datasource type:

- MBTiles
- Shapefile
- Pbf
- Geojson

First of all , the layer name & class should be know as prior knowledge that mean the same class maybe has different keyword in OSM data and definition. like 'water' as a classname, but same classname in OSM data will be 'waterway','water','lake'...

Support Raster Dataset key:

- Google
- Google China,
- Google Maps,
- Google Satellite,
- Google Terrain,
- Google Terrain Hybrid,
- Google Satellite Hybrid
- Stamen Terrain
- Stamen Toner
- Stamen Toner Light
- Stamen Watercolor
- Wikimedia Map
- Wikimedia Hike Bike Map
- Esri Boundaries Places
- Esri Gray (dark)
- Esri Gray (light)
- Esri National Geographic
- Esri Ocean,
- Esri Satellite,
- Esri Standard,
- Esri Terrain,
- Esri Transportation,
- Esri Topo World,
- OpenStreetMap Standard,
- OpenStreetMap H.O.T.,
- OpenStreetMap Monochrome,
- OpenTopoMap,
- Strava All,
- Strava Run,
- Open Weather Map Temperature,
- Open Weather Map Clouds,
- Open Weather Map Wind Speed,

- CartoDb Dark Matter,
- CartoDb Positron,
- Bing VirtualEarth

Usage:

Step 1:

Download the tile file is the first step. But the almost data resources supporter didn't write the projection information to tile file. So the compute tile projection infomation & write to file is most import part in process of download flow.

In [7]:
```
from Data.IO.Downloader import DOWNLOADER
UTAH=DOWNLOADER("Google Satellite")
```

```
# ------------------------------------------------------------------------ #
#                            MAP Production Toolkit                        #
# ------------------------------------------------------------------------ #
# ---------------------- MAP Serverv Init Successful by ---------------------- #
# ---------------------- Google Satellite ----------------------------------- #
```

Demo:

We could choose a Position like Saltlake. Salt Lake City is located in United States country, in North America continent (or region). DMS latitude longitude coordinates for Salt Lake City are: 40°45'38.81"N, 111°53'27.78"W. • Latitude position: Equator ⇐ 4532km (2816mi) ⇐ Salt Lake City ⇒ 5475km (3402mi) ⇒ North pole. • Longitude position: Salt Lake City ⇐ 8644km (5371mi) ⇐ Prime meridian. GMT: -6h. • Local time in Salt Lake City: Friday 1:35 am, May 22, 2020. [*time info]

We need plan a area that describe by WGS84 lonlat,like: Cord1=(-111.89105,40.76078) # Left Top Lonlat Cord2=(-111.8,40.7)# Right Bottom Lonlat In addition, we need set the `zoom level` that mean resolution of each map tile. Relative info:

Level	Degree	Area	m / pixel	~Scale	# Tiles
0	360	whole world	156,412	1:500 million	1
1	180		78,206	1:250 million	4
2	90		39,103	1:150 million	16
3	45		19,551	1:70 million	64
4	22.5		9,776	1:35 million	256
5	11.25		4,888	1:15 million	1,024
6	5.625		2,444	1:10 million	4,096
7	2.813		1,222	1:4 million	16,384
8	1.406		610.984	1:2 million	65,536
9	0.703	wide area	305.492	1:1 million	262,144
10	0.352		152.746	1:500,000	1,048,576
11	0.176	area	76.373	1:250,000	4,194,304
12	0.088		38.187	1:150,000	16,777,216
13	0.044	village or town	19.093	1:70,000	67,108,864
14	0.022		9.547	1:35,000	268,435,456
15	0.011		4.773	1:15,000	1,073,741,824
16	0.005	small road	2.387	1:8,000	4,294,967,296
17	0.003		1.193	1:4,000	17,179,869,184
18	0.001		0.596	1:2,000	68,719,476,736
19	0.0005		0.298	1:1,000	274,877,906,944

The data will generate as tiles (256*256 image), you also could use `DOWNLOADER_INSTANCE` .merge() to merge all the tiles to whole tiff file.

addcord() as a function ,input is WGS cord of left-top point & right-bottom point x1,y1,x2,y2,additional zoom level that mean different level density of data grid.

```
left, top : left-top coordinate, for example (100.361,38.866)

right, bottom : right-bottom coordinate

z : zoom
filePath : File path for storing results, TIFF format
```

In [8]:
```
Cord1=(-111.89105,40.76078) # Left Top Lonlat
Cord2=(-111.8,40.7)# Right Bottom Lonlat
Zoom=13 # Zool Level

UTAH.addcord(*Cord1,*Cord2,Zoom)
UTAH.download()
tiles=UTAH.savetiles(path="./Image",format="tif")
```

```
-----Total tiles number：3 X 3
Tiles downloading......
<ForkProcess(ForkPoolWorker-15, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-13, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-21, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-16, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-17, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-20, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-14, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-19, started daemon)>
-----Buffer DataSize: 72
<ForkProcess(ForkPoolWorker-18, started daemon)>
-----Buffer DataSize: 72

100%|██████████| 9/9 [00:00<00:00, 244.66it/s]
Tiles download complete
TileSize: 9
Size: 3 X 3
```

The Vector & Raster Class could do some IO,transform to raster or vector object. For instance, we use a shapefile that download from https://gis.utah.gov/ (https://gis.utah.gov/) as label to generate groundtruth. if the timestamp of two datasources (vector & raster) is almost same, you could get the high quality dataset.

In [9]:
```
from Data.IO.Vector import Vector
Building=Vector("/workspace/data/UTAH/Buildings_shp/Buildings/Buildings.shp")
```

```
# ------------------------------------------------------------------------------ #
#                                Vector Toolkit                                  #
# ------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------ #
#                             TIFF process Toolkit                               #
# ------------------------------------------------------------------------------ #
-----Class TIF init without filename
-----Valid vector format : shp
-----Meta :  {}
-----Description :  /workspace/data/UTAH/Buildings_shp/Buildings/Buildings.shp
-----LayerCount: 1
-----Layer : 0  Define :  Buildings
-----LayerDictByName:
 {'Buildings': <osgeo.ogr.Layer; proxy of <Swig Object of type 'OGRLayerShadow *' at 0x7f8e5c9e5120> >}
-----Alread Load: /workspace/data/UTAH/Buildings_shp/Buildings/Buildings.shp
# ------------------------------ DEFINE DONE ------------------------------- #
```

The most of SQLite based mbtiles vector database will have multi-layer, but wkt based shapefile & geojson almost have single layer. Normally,Name of layer is class name that must set as default layer by `getDefaultLayerbyName` function.

In [14]:
```
Building.getDefaultLayerbyName("Buildings")
```

Out[14]:
```
<osgeo.ogr.Layer; proxy of <Swig Object of type 'OGRLayerShadow *' at 0x7f8e5c9e5120> >
```

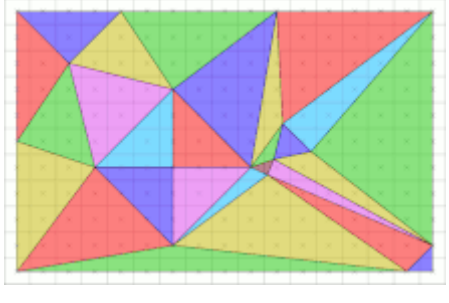
Step 2:

If the data use for model training, we should have label that could be generate by rasterize vector file. Normally, the data will label by artificial work.But human resources has limit in huge object label with high resolution imagery. The OSM Vector data has a worldwide version that save in sqlite based mbtiles file system that could be decode by GDAL library.

The Class Vector and Raster is important part of data I/O. Rasterisation (or rasterization) is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (a series of pixels, dots or lines, which, when displayed together, create the image which was represented via shapes).[1][2] The rasterised image may then be displayed on a computer display, video display or printer, or stored in a bitmap file format. Rasterisation may refer to the technique of drawing 3D models, or the conversion of 2D rendering primitives such as polygons, line segments into a rasterized format.

The map data has better relative accuracy than temporary human label work that mean the vector map has potential to be ground truth. So, transform the exist vector to raster data that is indispensable method for generate training data in DL-based computer vision mission.

Rasterize:

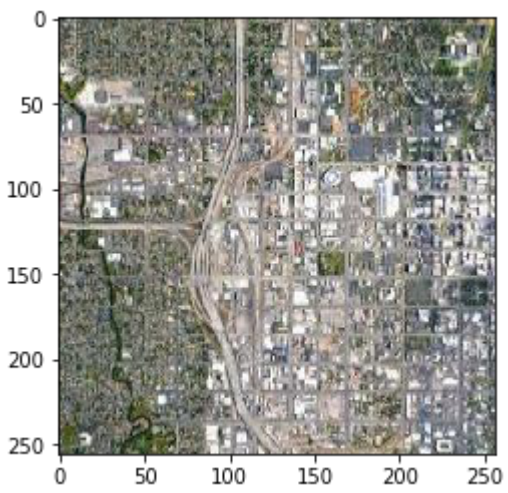


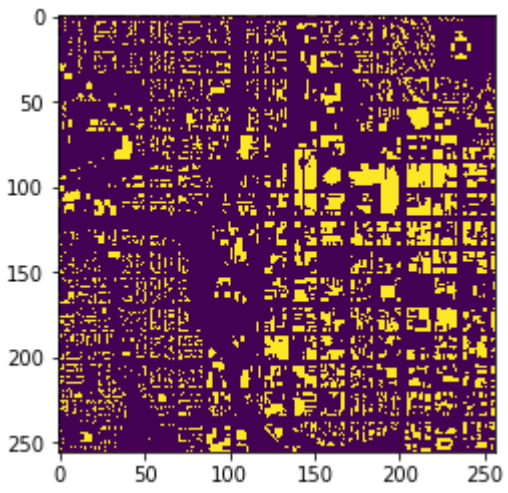
In [11]:
```
label=Building.generate(tiles,output_path="./Label")
```

```
  0%|          | 0/9 [00:00<?, ?it/s]
-----Start Generate.....
100%|██████████| 9/9 [00:40<00:00,  4.50s/it]
```

If we write the 'image' & 'label' to csv / json that could be great dataset for deeplearning training work flow. We could show the label&image like that.

In [12]:
```
import tifffile as tif
import matplotlib.pyplot as plt
image=tif.imread(tiles[0])
label=tif.imread(label[0])
plt.imshow(image),plt.show()
plt.imshow(label),plt.show()
```





Out[12]: (<matplotlib.image.AxesImage at 0x7f8e5d848f28>, None)

In [ ]: