

# Street View Imagery Analysis in a Computational Notebook

Yuhao Kang<sup>1</sup>

<sup>1</sup> The University of Texas at Austin, Austin TX, USA

Received: November 4, 2024/Accepted: January 6, 2026

**Abstract.** Street view imagery, capturing detailed streetscapes at human eye-level, has received significant attention in the past decade. Street view images can be leveraged to observe the built environment at both the element and scene levels. This chapter provides an introduction to methodologies for street view image-based analytics, including downloading street view images using Google Street View API, and employing advanced computer vision techniques such as Deep Convolutional Neural Networks to detect and quantify urban elements (trees, cars, buildings) and scenes (park, street). Particularly, this chapter introduces the use of image semantic segmentation to identify distinct urban elements, and image classification techniques to categorize and predict urban scene types. Further, an example of the calculation of the Green View Index (GVI) is provided to demonstrate how street view imagery analysis could contribute to urban data analytics. Through these methods, street view imagery not only helps model the digital environment and enhances our understanding of urban environments, but also offers a variety of insights into geography and urban studies.

*Keywords:* Street view imagery, urban environment, deep learning, AI, urban data science

## 1 Introduction

Street view imagery has emerged as a crucial data source for auditing built environments across various domains, including geography, urban planning, environmental science, criminology, and public health (Rzotkiewicz et al. 2018, Kang et al. 2020, Ibrahim et al. 2020, Biljecki, Ito 2021). Street view images are typically collected as omni-directional panoramas, capturing detailed urban streetscapes, and providing visual information that closely reflects human eye-level perceptions (Angelov et al. 2010). Because these images are captured at specific points in time, they offer temporal “snapshots” of the built environment. To protect privacy, faces, license plates, and in some cases façade details are automatically blurred. Street view images thus offer comprehensive perspectives for examining various aspects of the urban built environment and have benefited many downstream urban applications.

Beyond allowing users to navigate urban landscapes remotely through panoramic images for an immersive experience, street view images have several significant advantages over traditional data collection methods such as surveys, questionnaires, and mental mapping. These benefits include high coverage and volume, high data quality, low cost, and eye-level scenery (Kang et al. 2020). First, multiple global mapping service providers, including commercial companies like Google, Bing, Baidu, Tencent, and Yandex, as

well as crowdsourced platforms such as [Mapillary](#), offer extensive volumes of street view images covering global cities. Despite that street view imagery might be limited in Global South, this extensive coverage enables a comprehensive depiction of the built environment and allows researchers to perform cross-national comparisons of urban landscapes.

Second, street view images typically exhibit relatively higher quality. Images obtained from mapping service providers generally adhere to uniform standards and processes, having a consistent format that ensures a reliable image quality. Even crowdsourced platforms require specific format standards so that images are preprocessed to ensure their quality standards. Third, compared with traditional methods, street view images are effective and easily accessible. Users can conveniently and efficiently download these images to meet their demand at relatively low costs or no cost via APIs (Application Programming Interfaces) provided by mapping services. Moreover, users can customize various parameters of APIs such as location, camera angle, and image size, to get the data to meet specific research objectives. This accessibility facilitates the study of built environments using street view images in diverse cities and regions.

Last, and most crucially, street view images provide an objective observation of the built environment from a human perspective. Unlike aerial photographs and remote sensing images that offer bird's-eye views, street view imagery captures the urban environment from ground level, aligning closely with human perceptions. This feature makes street view imagery particularly effective for observing urban spaces and everyday urban life. Collectively, these benefits make street view images a vital data source for in-depth analysis of urban-related topics and for making informed decisions in urban planning and policy development.

More importantly, the rapid development of Artificial Intelligence (AI) offers various opportunities for quantifying urban features. Researchers are increasingly utilizing advanced computer vision techniques, such as deep learning, to identify and measure various elements (e.g., trees, vehicles) and scenes (urban functions) within the built environment ([Li et al. 2015](#), [Zhang et al. 2018](#)). These efforts enable a comprehensive modeling of urban environmental settings, further enabling studies on both physical and socio-economic environments. For instance, researchers have quantified the proportion of urban greenery ([Li et al. 2018](#)), assessed the walkability and cyclability of environments ([Yin, Wang 2016](#), [Lu 2019](#)), and examined factors contributing to obesogenic environments ([Feuillet et al. 2016](#)). These studies leveraged AI to enhance our understanding of urban dynamics, offering critical insights into the complex interplay between urban design and societal outcomes. Additionally, researchers have also assessed the socio-economic dimensions of cities, such as measuring urban gentrification ([Ilic et al. 2019](#)), human mobility ([Zhang et al. 2019](#)), urban functions ([Ye et al. 2021](#)), and housing prices ([Kang et al. 2021](#)). Moreover, beyond analyzing objective elements and scenes, researchers are utilizing street view images to delve into residents' subjective perceptions of urban environments. Such studies have provided insights into feelings of safety and depression ([Zhang et al. 2018](#), [Kang et al. 2023](#)), thereby enhancing our understanding of the interplay between the built environment and human subjective experiences.

Given these advancements in large-scale street view images and AI methods for processing images, this chapter provides an overview of methods leveraged to utilize this valuable dataset. The following sections will provide detailed instructions about data collection and analysis at the object and scene levels. Several applications will be demonstrated to showcase how street view images might be leveraged to support urban and geospatial data analytics.

## 2 Computational Environment

In this section, we present the packages and computational environment used in our analysis. The *requests* and *ssl* libraries are used to communicate with web APIs (such as the Google Street View API) over secure connections. *NumPy* and *pandas* provide core numerical and tabular data structures, which we use to organize data. We utilize the *GeoPandas* package to handle and process geospatial data efficiently. We use the *OSMnx* package for downloading network data from OpenStreetMap. This tool helps in

generating points along road networks. For visualization, we rely on *matplotlib*, which allows us to create maps and plots. The *PIL* package is utilized to help process imagery data and display street view images. The following codes has been tested on Python 3.9.

```
[1]: import io
import json
import os
import requests
import ssl
import numpy as np
import pandas as pd
import osmnx as ox
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from PIL import Image
```

To effectively implement deep learning methods, it is essential to install relevant packages and establish suitable computational environments. Several tools such as *pip* or *conda* has been widely utilized. *Conda* is a package and environment manager that installs libraries and keeps track of their versions. These packages can help manage the packages and environment, supporting debugging of the environment.

We primarily utilize the *PyTorch* framework, developed by the *Fundamental AI Research (FAIR)* team at Meta. PyTorch has become a popular platform in deep learning due to its comprehensive functionality and ecosystem. This framework provides a wide range of easily accessible pre-trained models, enabling users to apply advanced deep-learning techniques in a user-friendly way.

```
[2]: # Load packages of PyTorch
import torch
from torchvision import transforms
print(torch.__version__) # we want 2.1.2
```

```
[2]: 2.1.2+cu121
```

PyTorch can be installed in two main ways: a CPU-only version and a GPU-enabled version. The CPU (Central Processing Unit) is the main processor in your computer and can run all of the code in this notebook, but deep learning models may run slowly on large datasets. A GPU (Graphics Processing Unit) is a specialized processor originally designed for graphics, but it is now widely used to accelerate deep learning computations.

For optimal performance, the use of NVIDIA GPUs coupled with the installation of CUDA is recommended to ensure the environment runs efficiently. If you are unsure whether your machine has a suitable GPU or CUDA installed, you may need to check your system specifications or consult your IT support. Unlike traditional methods that are performed on CPUs, utilizing GPUs for model training could significantly accelerate computational tasks. It is crucial for running such computationally intensive experiments on GPUs to analyze place scenes and extract objects from large-scale street view image dataset. To verify if the system is properly configured for GPU utilization, the following command can be used:

```
[3]: print(torch.cuda.is_available())
```

```
[3]: True
```

In addition to PyTorch, we incorporate *OpenMMLab*, an extensive toolkit that offers a variety of pre-trained models and tools for deep learning applications. This toolkit provides valuable resources for users so that they do not need to develop and train models from the ground up. By leveraging these advanced computational tools, our research can efficiently identify objects in the physical environment and understand place scenes from street view images. The OpenMMLab provides its own tools *mmengine* to help install a variety of deep learning models. This enables the researchers to download model packages efficiently.

```
[4]: # !pip install -U openmim
      # !mim install mmengine
      # !mim install "mimcv>=2.0.0"
      # !pip install "mimsegmentation>=1.0.0"
```

After the installation commands above, the subsequent code loads an OpenMMLab model.

```
[5]: # Load packages of OpenMMLab
      import mmcv
      import mmseg
      from mmseg.apis import MMSegInferencer
```

### 3 Street View Images Collection

To demonstrate the practical applications of street view images in urban environment observation, we will use the Mueller neighborhood in Austin, TX, United States as a case study. This example will guide you through the process of acquiring street view images via the Google Street View API. Users must first obtain an API Key from Google Maps, which is utilized to verify authorization and manage data usage. To obtain this API Key, readers can visit the [Google Cloud Platform](#), register or log into the account, and follow the steps to generate a key. This key will further be utilized in their code to download street view images.

To download street view images, a prerequisite is to generate a set of geographic points from which street view images will be requested. To accomplish this, we use OSMnx to download nodes from OpenStreetMap that represent intersections and endpoints of streets and roads within the Mueller neighborhood. As illustrated below, nodes are downloaded within the Mueller neighborhood. Each node contains the unique OSM id and coordinates. In practice, beyond utilizing nodes downloaded from OpenStreetMap, users could generate their own geographic points for certain places manually. It should be noted that not all locations have street view images.

#### 3.1 Download node data

```
[6]: # Download nodes from OpenStreetMap using the OSMnx
      G = ox.graph_from_place('Mueller, Austin, TX')
      ox.save_graph_geopackage(G, filepath='data/mueller.gpkg')
      mueller = gpd.read_file("data/mueller.gpkg")
      mueller.head(5)
```

```
[6]:
```

	osmid	y	x	street_count	highway ref \
0	152680140	30.288202	-97.700334	3	
1	354665612	30.300224	-97.706409	4	
2	354665831	30.305895	-97.707555	5	traffic_signals
3	354667206	30.307465	-97.711078	4	crossing
4	354668466	30.302508	-97.708340	3	

```

      geometry
0 POINT (-97.70033 30.28820)
1 POINT (-97.70641 30.30022)
2 POINT (-97.70755 30.30589)
3 POINT (-97.71108 30.30746)
4 POINT (-97.70834 30.30251)
```

```
[7]: # Plot all nodes downloaded
      mueller.plot(markersize=2)
```

#### 3.2 Download street view images based on latitude and longitude

Once these geographic points are defined, two methods can be employed to download street view images. First, the [Google Street View API](#) allows users to retrieve street view images by providing latitude and longitude coordinates. This method is straightforward and convenient, returning the nearest street view image to the specific location. However,

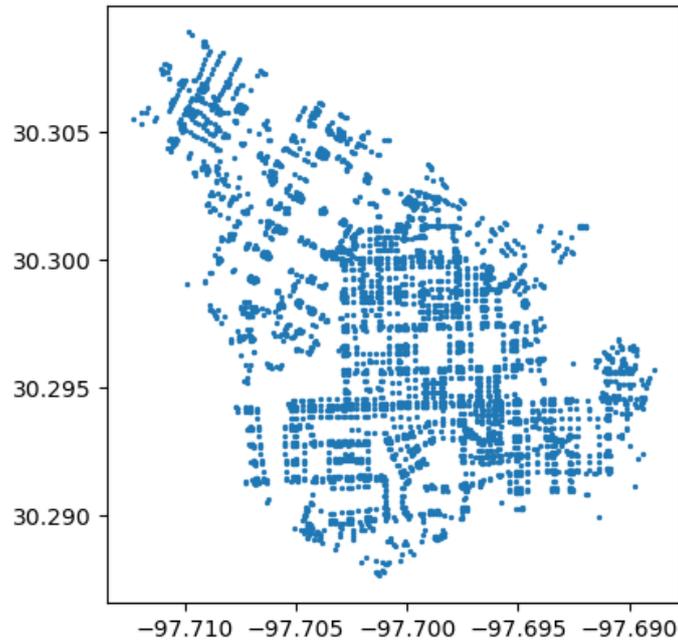


Figure 1: Example street view imagery points downloaded in Mueller, Austin. (Output of code chunk 7)

it should be noted that the image retrieved may not always precisely represent the given locations as it is the closest available image. More details of the parameters can be found in the [Google Street View API](#).

```
[8]: # The function to download the street view image
def download_sv_latlng(lat, lng, img_path):
    API_key = "-----" # Put your API key here
    sv_params = {
        "size": "400x400",
        "location": f"{lat}, {lng}",
        "heading": "150",
        "pitch": "0",
        "key": API_key
    }
    response = requests.get(url="https://maps.googleapis.com/maps/api/streetview",
    ↪params=sv_params)
    print("Download status: ", response.status_code)
    img = Image.open(io.BytesIO(response.content))
    img.save(f"{img_path}")
    return img
```

After running the function, the notebook prints an HTTP response status code (e.g., Download status: 200). This code indicates whether the request to the Google Street View API was successful. A status code of 200 means “OK,” indicating that the image was retrieved correctly. Other codes may indicate problems. For example, 400-series codes (such as 400 or 403) usually indicate client-side issues (e.g., invalid parameters, missing or incorrect API key, exceeding quota), whereas 500-series codes reflect server-side errors.

```
[9]: # Download a single street view image in the DataFrame mueller based on latitude and
    ↪longitude
sv_img = download_sv_latlng(30.288202, -97.700334, "node_152680140.jpg")
```

```
[9]: Download status: 200
```

Figure 2 shows an example street view image downloaded given a specific geographic coordinate. It is the first node collected in the *mueller* dataframe.

Street View Image

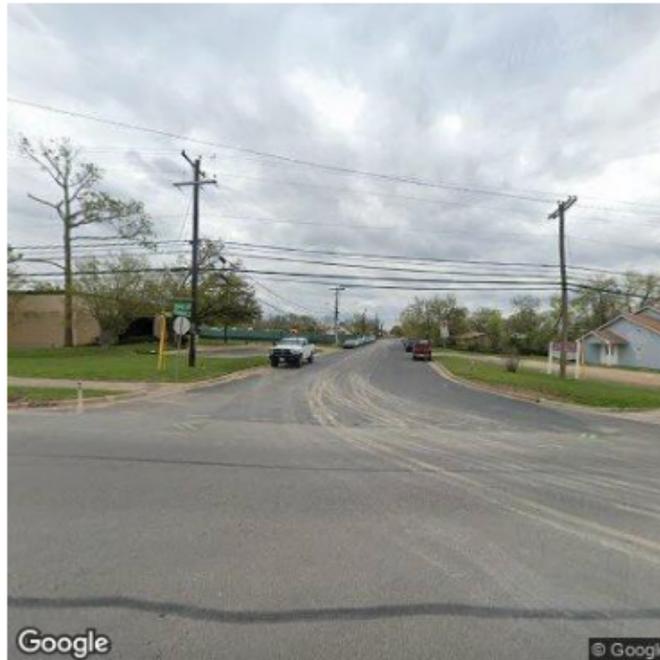


Figure 2: An example street view imagery. (Output of code chunk 10)

```
[10]: # Download a single street view image
plt.imshow(sv_img)
plt.axis('off') # Turn off axis numbers and ticks
plt.title('Street View Image')
plt.show()
```

### 3.3 Download street view images based on panoids

The second method involves two key steps. Instead of directly retrieving street view images, users first obtain the *panoid*, which is a unique identifier for each street view image, along with its precise latitude and longitude. Once the panoid of the nearest street view image of the location is acquired, users can use it to request the specific street view image. This step does not involve downloading the images themselves but rather gathering information to locate the images requested. After identifying the necessary metadata and removing duplicates to avoid redundancy, the street view image for each unique node is then downloaded.

```
[11]: # The function to get panoid of the street view image, given a specific coordinate
def get_panooid(lat, lng):
    API_key = "-----" # Put your API key here
    sv_params = {
        "size": "400x400",
        "location": f"{lat}, {lng}",
        "heading": "150",
        "pitch": "0",
        "key": API_key
    }
    response = requests.get(url="https://maps.googleapis.com/maps/api/streetview/
↳metadata", params=sv_params)
    panoid = json.loads(response.text)["pano_id"]
    return panoid
```

```
[12]: # Get panoid of the street view image, given a specific coordinate
panoid = get_panooid(30.288202, -97.700334)
print("panoid: ", panoid)
```

```
[12]: panoid: igiDU33L1L1_vYg9CMYZZw
```

```
[13]: # Get panoid for each osmid point
mueller["panoid"] = mueller.head(5).apply(lambda x: get_panoid(x["y"], x["x"]), axis=1)
↔# Just get panoids for the first five nodes
mueller.head(1)
```

```
[13]:      osmid      y      x  street_count highway ref \
0  152680140  30.288202 -97.700334      3

      geometry      panoid
0  POINT (-97.70033 30.28820)  igiDU33L1L1_vYg9CMYZZw
```

```
[14]: # Drop duplicates as different places may have the same street view image
panoids = mueller["panoid"].dropna().unique()
print(panoids)
```

```
[14]: ['igiDU33L1L1_vYg9CMYZZw' 'UpIFSkcTE-KYAepxqtXWjw'
'mzvCkPGNcD08bqMTam6pwA' 'Z7fSMVeWojQcJQG6N4nrGg'
'wiEr11l_CT64If0GL42BoQ']
```

```
[15]: # The function to download the street view image
def download_sv_panoid(panoid, img_path):
    API_key = "-----"
    sv_params = {
        "size": "400x400",
        "pano": f"{panoid}",
        "heading": "150",
        "pitch": "0",
        "key": API_key
    }
    response = requests.get(url="https://maps.googleapis.com/maps/api/streetview",
↔params=sv_params)
    print("Download status: ", response.status_code)
    img = Image.open(io.BytesIO(response.content))
    img.save(f"{img_path}")
    return img
```

```
[16]: # Download a single street view image
sv_img = download_sv_panoid("igiDU33L1L1_vYg9CMYZZw", "igiDU33L1L1_vYg9CMYZZw.jpg")
```

```
[16]: Download status: 200
```

Figure 3 shows an example street view image given a specific panoid. Both methods are valuable for accessing Google Street View images, with the choice of method depending on the user's specific needs regarding accuracy and the level of detail in imagery and metadata. By following this workflow, users can effectively leverage street view images to perform analysis of the urban environment.

```
[17]: # Display the downloaded street view image
plt.imshow(sv_img)
plt.axis('off') # Turn off axis numbers and ticks
plt.title('Street View Image')
plt.show()
```

This chapter's example focuses on Google Street View imagery. To collect crowd-sourced street view imagery from Mapillary, readers can refer to its website [Mapillary API](#).

#### 4 Urban Elements Extraction with Image Segmentation

With street view images retrieved, researchers could model the built environment at both the element- and scene-levels. Element-level observation refers to detecting and extracting individual objects, elements, or items from the urban environment, as well as calculating specific indices for these elements. One notable example refers to the calculation of urban greenery (Li et al. 2015). Other elements that have been analyzed include,

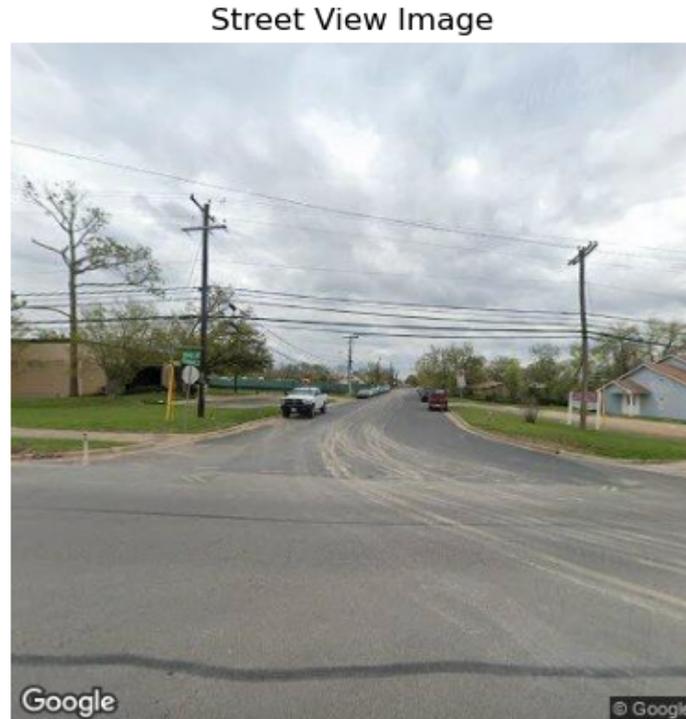


Figure 3: An example street view imagery. (Output of code chunk 17)

but are not limited to, features such as buildings (Gong et al. 2018) and traffic signs (Li et al. 2022). By detecting and measuring urban elements in streetscapes, researchers could further understand the composition and characteristics of urban spaces.

Image segmentation techniques have been widely used for element-level observation. Image segmentation refers to the process of partitioning an image into multiple segments, where each pixel is assigned a meaningful label or annotation. Several commonly detected objects include buildings, trees, roads, and vehicles. The emergence of deep learning methods, such as deep convolutional neural networks (DCNNs), has enabled researchers to extract high-dimensional visual features, facilitating a deeper understanding of images, and achieved high accuracy in image segmentation. Several notable methods include SegNet (Badrinarayanan et al. 2017) and ResNet (He et al. 2016).

In this chapter, we provide an example of using DeepLabV3 (Chen et al. 2018), a specific DCNN, for semantic segmentation, utilizing the ADE20K dataset. The ADE20K dataset is a large-scale annotated image dataset with over 27K images (Zhou et al. 2017). It depicts a variety of place scenes and provides comprehensive labels for numerous objects, such as buildings, vegetation, and vehicles, making it well-suited for urban element-level observation. Based on this dataset, a DeepLabV3 model has been trained to detect 150 objects from images. DeepLabV3 is a state-of-the-art DCNN model for semantic segmentation. It combines atrous (dilated) convolutions with an Atrous Spatial Pyramid Pooling (ASPP) module to capture multiscale contextual information, allowing the model to distinguish objects of different sizes in complex street scenes. It is highly effective for analyzing the urban environment by distinguishing different objects in place scenes, and has achieved competitive accuracy on the ADE20K dataset. We also acknowledge a few limitations of the current deep learning-based methods. The segmentation quality may vary across object classes and image conditions. For example, certain objects might be missing in the training data. Some images might be blurred, which will result in inaccurate results.

To perform image segmentation, the pretrained DeepLabV3 model is first downloaded using the MMSegmentation toolkit, which provides access to a variety of models for semantic segmentation with a simple command line (Chen et al. 2019). It should be

noted that, the readers could download other types of deep learning methods based on the needs. In this chapter, we take the following model as an example to show case how to perform image semantic segmentation.

#### 4.1 Image Semantic Segmentation based on ADE20K using DeepLabV3

```
[18]: # Create the model and output directory
      # !mkdir ./chec
      # !mkdir ./segmentation_results
```

```
[19]: # Download the deep learning model
      # !mim download mmsegmentation --config deeplabv3plus_r101-d8_4xb4-160k_ade20k-512x512
      ↪ --dest ./chec
```

```
[20]: # Set image file, model file and checkpoint path
      img= 'node_152680140.jpg'
      model = "chec/deeplabv3plus_r101-d8_4xb4-160k_ade20k-512x512.py"
      cpt = "chec/deeplabv3plus_r101-d8_512x512_160k_ade20k_20200615_123232-38ed86bb.pth"
      # Set output directory
      outdir = "segmentation_results/"
```

Rather than training the model from scratch, we load these pretrained weights and apply the model to our street view images to obtain pixel-level labels. This step is often called inference, i.e., running the trained model on new images to generate predictions (segmentation maps).

```
[21]: # Perform image semantic segmentation with the downloaded DeepLabV3 model
      mmseg_inferencer = MMSegInferencer(model, cpt, dataset_name= "ade20k")
      result = mmseg_inferencer(img, out_dir = outdir, opacity = 1, with_labels = False)
```

```
[21]: Inference -----
      /miniconda3/envs/geo/lib/python3.11/site-
      packages/mengine/visualization/visualizer.py:196: UserWarning: Failed to add
      <class 'mengine.visualization.vis_backend.LocalVisBackend'>, please provide the
      `save_dir` argument.
      warnings.warn(f'Failed to add {vis_backend.__class__}, '
```

#### 4.2 Visualization of Results

After inferencing the image using the selected model, the results are presented as follows. The output is a matrix that matches the dimensions of the input image, with each pixel assigned a value that corresponds to a specific urban object. The segmented output is then visualized by rendering each element in a distinct color within the streetscape. The class names from the ADE20K dataset and their corresponding colors are predefined in the *mmsegmentation* tool, and details can be accessed through the provided links for [object names](#) and [color codings](#). This allows researchers to vividly understand and analyze the composition of the built environment.

```
[22]: # Display results
      print(result)
      print("Image size: ", result['predictions'].shape)
```

```
[22]: {'predictions': array([[2, 2, 2, ..., 2, 2, 2],
      [2, 2, 2, ..., 2, 2, 2],
      [2, 2, 2, ..., 2, 2, 2],
      ...,
      [6, 6, 6, ..., 6, 6, 6],
      [6, 6, 6, ..., 6, 6, 6],
      [6, 6, 6, ..., 6, 6, 6]]), 'visualization': []}
      Image size: (400, 400)
```

```
[23]: # mmsegmentation predefined object names, color schemes and palettes
# objects: https://github.com/open-mmlab/msegmentation/blob/main/mmseg/utils/
# class_names.py#L15-L42
# color schemes and palettes: https://github.com/open-mmlab/msegmentation/blob/main/
# mmseg/utils/class_names.py#L273-L312

obj_classes=['wall', 'building', 'sky', 'floor', 'tree', 'ceiling', 'road', 'bed ',
↳'windowpane', 'grass', 'cabinet', 'sidewalk', 'person', 'earth', 'door',
↳'table', 'mountain', 'plant', 'curtain', 'chair', 'car', 'water', 'painting',
↳'sofa', 'shelf', 'house', 'sea', 'mirror', 'rug', 'field', 'armchair',
↳'seat', 'fence', 'desk', 'rock', 'wardrobe', 'lamp', 'bathtub', 'railing',
↳'cushion', 'base', 'box', 'column', 'signboard', 'chest of drawers',
↳'counter', 'sand', 'sink', 'skyscraper', 'fireplace', 'refrigerator',
↳'grandstand', 'path', 'stairs', 'runway', 'case', 'pool table', 'pillow',
↳'screen door', 'stairway', 'river', 'bridge', 'bookcase', 'blind', 'coffee
↳table', 'toilet', 'flower', 'book', 'hill', 'bench', 'countertop',
↳'stove', 'palm', 'kitchen island', 'computer', 'swivel chair', 'boat', 'bar',
↳'arcade machine', 'hovel', 'bus', 'towel', 'light', 'truck',
↳'tower', 'chandelier', 'awning', 'streetlight', 'booth', 'television
↳receiver', 'airplane', 'dirt track', 'apparel', 'pole', 'land', 'bannister',
↳'escalator', 'ottoman', 'bottle', 'buffet', 'poster', 'stage', 'van', 'ship',
↳'fountain', 'conveyer belt', 'canopy', 'washer', 'plaything', 'swimming pool',
↳'stool', 'barrel', 'basket', 'waterfall', 'tent', 'bag', 'minibike', 'cradle',
↳'oven', 'ball', 'food', 'step', 'tank', 'trade name', 'microwave', 'pot', 'animal',
↳'bicycle', 'lake', 'dishwasher', 'screen', 'blanket', 'sculpture', 'hood',
↳'sconce', 'vase', 'traffic light', 'tray', 'ashcan', 'fan', 'pier', 'crt screen',
↳'plate', 'monitor', 'bulletin board', 'shower', 'radiator', 'glass', 'clock',
↳'flag']

palette=[[120, 120, 120], [180, 120, 120], [6, 230, 230], [80, 50, 50], [4, 200, 3],
↳[120, 120, 80], [140, 140, 140], [204, 5, 255], [230, 230, 230], [4, 250, 7],
↳[224, 5, 255], [235, 255, 7], [150, 5, 61], [120, 120, 70], [8, 255, 51],
↳[255, 6, 82], [143, 255, 140], [204, 255, 4], [255, 51, 7], [204, 70, 3],
↳[0, 102, 200], [61, 230, 250], [255, 6, 51], [11, 102, 255], [255, 7, 71],
↳[255, 9, 224], [9, 7, 230], [220, 220, 220], [255, 9, 92], [112, 9, 255],
↳[8, 255, 214], [7, 255, 224], [255, 184, 6], [10, 255, 71], [255, 41, 10],
↳[7, 255, 255], [224, 255, 8], [102, 8, 255], [255, 61, 6], [255, 194, 7],
↳[255, 122, 8], [0, 255, 20], [255, 8, 41], [255, 5, 153], [6, 51, 255], [235,
↳12, 255], [160, 150, 20], [0, 163, 255], [140, 140, 140], [250, 10, 15],
↳[20, 255, 0], [31, 255, 0], [255, 31, 0], [255, 224, 0], [153, 255, 0], [0,
↳0, 255], [255, 71, 0], [0, 235, 255], [0, 173, 255], [31, 0, 255], [11, 200, 200],
↳[255, 82, 0], [0, 255, 245], [0, 61, 255], [0, 255, 112], [0, 255, 133],
↳[255, 0, 0], [255, 163, 0], [255, 102, 0], [194, 255, 0], [0, 143, 255], [51, 255, 0],
↳[0, 82, 255], [0, 255, 41], [0, 255, 173], [10, 0, 255], [173, 255, 0], [0,
↳255, 153], [255, 92, 0], [255, 0, 255], [255, 0, 245], [255, 0, 102], [255, 173, 0],
↳[255, 0, 20], [255, 184, 184], [0, 31, 255], [0, 255, 61], [0, 71, 255],
↳[255, 0, 204], [0, 255, 194], [0, 255, 82], [0, 10, 255], [0, 112, 255], [51, 0, 255],
↳[0, 194, 255], [0, 122, 255], [0, 255, 163], [255, 153, 0], [0, 255, 10],
↳[255, 112, 0], [143, 255, 0], [82, 0, 255], [163, 255, 0], [255, 235, 0], [8, 184,
↳170],
↳[133, 0, 255], [0, 255, 92], [184, 0, 255], [255, 0, 31], [0, 184, 255], [0,
↳214, 255], [255, 0, 112], [92, 255, 0], [0, 224, 255], [112, 224, 255], [70, 184,
↳160],
↳[163, 0, 255], [153, 0, 255], [71, 255, 0], [255, 0, 163], [255, 204, 0],
↳[255, 0, 143], [0, 255, 235], [133, 255, 0], [255, 0, 235], [245, 0, 255], [255, 0,
↳122],
↳[255, 245, 0], [10, 190, 212], [214, 255, 0], [0, 204, 255], [20, 0, 255],
↳[255, 255, 0], [0, 153, 255], [0, 41, 255], [0, 255, 204], [41, 0, 255], [41, 255, 0],
↳[173, 0, 255], [0, 245, 255], [71, 0, 255], [122, 0, 255], [0, 255, 184], [0,
↳92, 255], [184, 255, 0], [0, 133, 255], [255, 214, 0], [25, 194, 194], [102, 255, 0],
↳[92, 0, 255]]

[24]: # Match objects and colors
unique_colors = np.unique(result['predictions']).reshape(-1, 1), axis=0)
exist_classes = [obj_classes[int(index)] for index in unique_colors]
exist_palette = [palette[int(index)] for index in unique_colors]
print("The following objects are detected in this image: ", exist_classes)
print("Their palettes are: ", exist_palette)
```



Figure 4: Example of a street view imagery (left) and its corresponding semantic segmentation map (right), where pixels are classified into built-environment elements (e.g., wall, buildings, and sky). (Output of code chunk 25)

```
[24]: The following objects are detected in this image: ['wall', 'building', 'sky',
'tree', 'road', 'grass', 'sidewalk', 'car', 'house', 'fence', 'signboard',
'streetlight', 'pole']
Their palettes are: [[120, 120, 120], [180, 120, 120], [6, 230, 230], [4, 200,
3], [140, 140, 140], [4, 250, 7], [235, 255, 7], [0, 102, 200], [255, 9, 224],
[255, 184, 6], [255, 5, 153], [0, 71, 255], [51, 0, 255]]
```

```
/tmp/ipykernel_59716/2720650808.py:3: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
    exist_classes = [obj_classes[int(index)] for index in unique_colors]
/tmp/ipykernel_59716/2720650808.py:4: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
    exist_palette = [palette[int(index)] for index in unique_colors]
```

```
[25]: # Plot the original and segmented images
out_img_path = os.path.join(outdir, 'vis', img.split('/')[-1])
out_img = mmcv.imread(out_img_path)
orig_img = mmcv.imread(img)

# Create a plot to compare the original and segmented images
fig, ax = plt.subplots(1, 2, figsize=(14,6))
ax[0].imshow(mmcv.bgr2rgb(orig_img))
ax[0].axis('off')
ax[1].imshow(mmcv.bgr2rgb(out_img))
ax[1].axis('off')

# Create a legend for every color
patches = [patches.Patch(color=np.array(exist_palette[i])/255.,
                                label=exist_classes[i]) for i in range(len(exist_classes))]
ax[1].legend(handles=patches, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.,
            fontsize='large')

# Plot the result
plt.tight_layout()
plt.show()
```

## 5 Urban Scenes Evaluation using Image Classification

With street view images retrieved, researchers can also evaluate the entire urban scene to have an overall assessment of the environment and understand the quality of urban environments. Beyond individual elements, scene-level observation could offer insights into the overall characteristics of the urban landscape, such as place type classification, assessment of the neighborhood quality and functionality. To perform scene-level observation, researchers often leverage DCNNs to perform image classification (He et al. 2016). Image classification refers to assigning a label or category to an entire image

based on its visual content. Here, we utilize ResNet for scene-level observation, which is an advanced deep learning model that contains residual connections for effective understanding of complex images. ResNet has been widely leveraged for image classification in various image datasets such as ImageNet (Deng et al. 2009) and Places365 (Zhou et al. 2017). The ResNet model is leveraged to classify place scenes based on the Places365 dataset. Places365 dataset contains millions of labeled images representing a wide variety of scenes, including residential neighborhoods, parks, and hospitals. In total, 365 place scenes are defined. By leveraging Places365 dataset and ResNet, researchers can efficiently categorize urban scenes based on their visual characteristics, enabling scene-level observation. We first downloaded the model and the categories of place scenes.

```
[26]: # !wget https://raw.githubusercontent.com/CSAILVision/places365/refs/heads/master/
      ↪ categories_places365.txt -P ./chec
      # !wget http://places2.csail.mit.edu/models_places365/resnet50_places365.pth.tar -P ./
      ↪ chec
```

### 5.1 Identify urban scenes from street view images

```
[27]: from torch.autograd import Variable as V
      import torchvision.models as models
      from torchvision import transforms as trn
      from torch.nn import functional as F
```

```
[28]: # Load the pre-trained weights
      arch = 'resnet50'
      model = models._dict__[arch](num_classes=365)
      checkpoint = torch.load("chec/resnet50_places365.pth.tar", map_location=lambda storage,
      ↪loc: storage)
      state_dict = {str.replace(k,'module.',''): v for k,v in checkpoint['state_dict']}.
      ↪items()}
      model.load_state_dict(state_dict)
      model.eval()

      # Load the image transformer
      centre_crop = trn.Compose([
          trn.Resize((256,256)),
          trn.CenterCrop(224),
          trn.ToTensor(),
          trn.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
      ])

      # Load the class label
      classes = list()
      with open("chec/categories_places365.txt") as class_file:
          for line in class_file:
              classes.append(line.strip().split(' ')[0][3:])
      classes = tuple(classes)
```

```
[29]: # Read and process the input street view image
      img_array = Image.open(img)
      input_img = V(centre_crop(img_array).unsqueeze(0))

      # Forward pass
      logit = model.forward(input_img)
      h_x = F.softmax(logit, 1).data.squeeze()
      probs, idx = h_x.sort(0, True)

      # Display the input street view image
      plt.imshow(img_array)
      plt.axis('off') # Turn off axis numbers and ticks
      plt.title('Street View Image')
      plt.show()
      print('Input image', img)
      # Output the prediction
      print(f"Scene type: {classes[idx[0]]}")
      print(f"Top 5 scene type:")
      for i in range(0, 5):
          print(f'{classes[idx[i]]}: {probs[i]}')
```

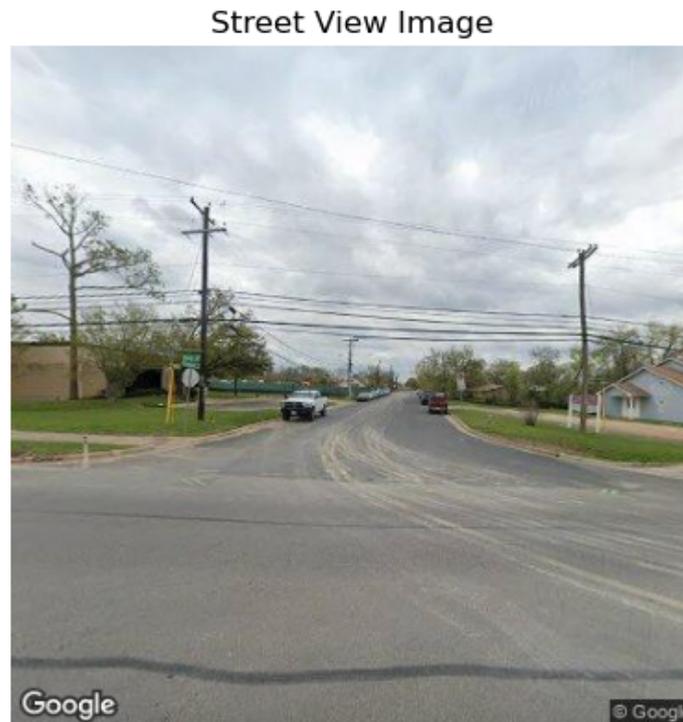


Figure 5: Example of a street view imagery and its corresponding semantic scene. (Output of code chunk 29)

```
[29]: Input image node_152680140.jpg
Scene type: industrial_area
Top 5 scene type:
industrial_area: 0.4464263319969177
residential_neighborhood: 0.2901725769042969
highway: 0.15815606713294983
field/wild: 0.025090044364333153
parking_lot: 0.018267285078763962
```

As shown in Figure 5, for a given street view image, the model generates an array of scene categories along with their corresponding probabilities, showing the confidence level of each classification. For instance, the top 5 predicted scenes of the image include an industrial area, residential neighborhood, highway, field, and parking lot. In particular, the top two categories of scenes have a combined probability of approximately 73%. This detailed output helps researchers effectively assess the place type of the contexts of the urban environments.

## 6 Urban Greenery Calculation

After performing element-level and scene-level observations of the urban environment, these valuable insights could be incorporated into urban data analytics to better understand and model the built environments. Here, we take the measurement of urban greenery as an example to demonstrate how greenery can be quantified from street view images. Urban greenery refers to the presence of vegetation, such as trees, shrubs, and grass, in urban areas. It has been widely studied in various urban research literature (Li et al. 2015, Yang et al. 2021, Lu et al. 2023). Researchers have developed the Green View Index (GVI) to measure the degree of physical greenness within the built environment. In the following paragraphs, we compute the average Green View Index (GVI) for the Mueller neighborhood using street view images. The equation for calculating the Green View Index is defined as follows:

$$GVI = \frac{N_{pixels_g}}{N_{pixels_a}}$$

where  $N_{pixels_g}$  refers to the number of pixels that have been identified as urban greenery (e.g., trees grass, plant, flower, palm, and canopy ) of the given image, and  $N_{pixels_a}$  indicates the total number of pixels of the street view image

Below, we present an example of how to calculate the Green View Index (GVI) for a given street view image. Specifically, the GVI calculation includes the following categories: trees, grass, plants, flowers, palms, and canopies. The proportions of each category are first computed individually, and then aggregated to determine the overall GVI percentage, providing a comprehensive measure of greenery in the image..

### 6.1 Compute the Green View Index

```
[30]: # Display the input street view image
plt.imshow(img_array)
plt.axis('off') # Turn off axis numbers and ticks
plt.title('Street View Image')
plt.show()

# Compute the Green View Index of a given street view image
total_pixel = result['predictions'].shape[0]*result['predictions'].shape[1]

tree_percentage = (np.sum(result['predictions'] == 4) / total_pixel)*100
print(f"{obj_classes[4]} percentage: {tree_percentage:.2f}%")

grass_percentage = (np.sum(result['predictions'] == 9) / total_pixel)*100
print(f"{obj_classes[9]} percentage: {grass_percentage:.2f}%")

plant_percentage = (np.sum(result['predictions'] == 17) / total_pixel)*100
print(f"{obj_classes[17]} percentage: {plant_percentage:.2f}%")

flower_percentage = (np.sum(result['predictions'] == 66) / total_pixel)*100
print(f"{obj_classes[66]} percentage: {flower_percentage:.2f}%")

palm_percentage = (np.sum(result['predictions'] == 72) / total_pixel)*100
print(f"{obj_classes[72]} percentage: {palm_percentage:.2f}%")

canopy_percentage = (np.sum(result['predictions'] == 106) / total_pixel)*100
print(f"{obj_classes[106]} percentage: {canopy_percentage:.2f}%")

print(f"Total Green: ↵
↵{tree_percentage+grass_percentage+plant_percentage+flower_percentage+
palm_percentage+canopy_percentage:.2f}%")
```

```
[30]: tree percentage: 8.44%
grass percentage: 4.52%
plant percentage: 0.00%
flower percentage: 0.00%
palm percentage: 0.00%
canopy percentage: 0.00%
Total Green: 12.96%
```

For instance, as illustrated in Figure 6, the proportion of trees in the street view image is 8.44%, and grass accounts for 4.5%, resulting in an overall GVI of approximately 13% when all types of urban greenery are considered. Researchers can then iterate over all GVI values within the study area and aggregate them to the neighborhood level for future spatial analysis. These GVI calculations can then be integrated into broader urban research to provide insights into the relationships between urban greenery and various environmental and social aspects of urban environments.

Additionally, the following cell provides codes that combines image semantic segmentation with the calculations of the Green View Index (GVI). This allows for the efficient computation of the GVI from any provided street view image, enabling researchers to assess the extent of urban greenery present in street view images.

Street View Image

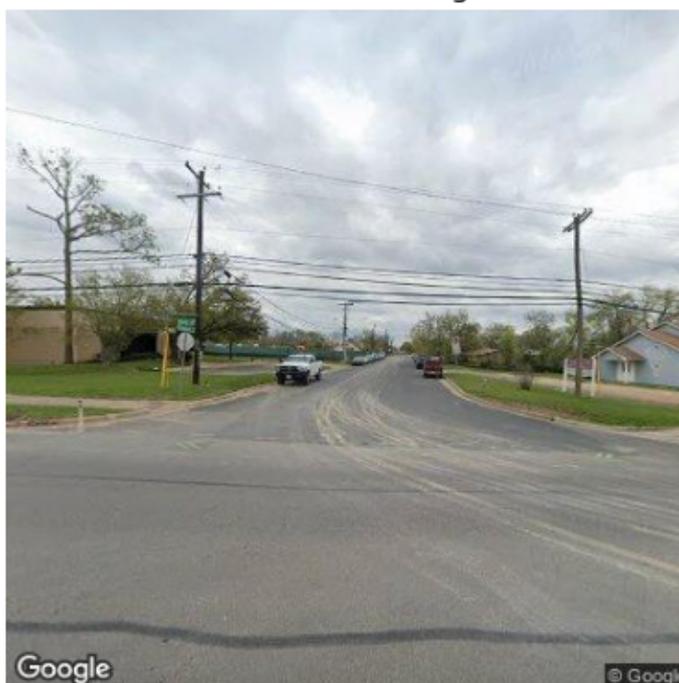


Figure 6: Example of a street view imagery and its corresponding element percentages. (Output of code chunk 30)

We acknowledge potential limitations for the GVI. When extending this calculation to many images along a road network, it is important to recognize that overlapping scenes may occur. Images captured at nearby points can contain very similar or even nearly identical views. If a particularly green segment of a street is sampled many times, this overlap may inflate neighborhood-level GVI estimates.

```
[31]: # The function to calculate the Green View Index of a given street view image
def cal_gvi(result):
    total_pixel = result['predictions'].shape[0]*result['predictions'].shape[1]
    tree_percentage = (np.sum(result['predictions'] == 4) / total_pixel)*100
    grass_percentage = (np.sum(result['predictions'] == 9) / total_pixel)*100
    plant_percentage = (np.sum(result['predictions'] == 17) / total_pixel)*100
    flower_percentage = (np.sum(result['predictions'] == 66) / total_pixel)*100
    palm_percentage = (np.sum(result['predictions'] == 72) / total_pixel)*100
    canopy_percentage = (np.sum(result['predictions'] == 106) / total_pixel)*100
    gvi = tree_percentage+grass_percentage+plant_percentage+flower_percentage+
    palm_percentage+canopy_percentage
    return gvi
```

```
[32]: # Given an image, perform image semantic segmentation and calculates the GVI
img= 'node_152680140.jpg'
print("The GVI is: ", cal_gvi(mmseg_inferencer(img, out_dir = outdir, opacity = 1,
↳with_labels = False)), "%")
```

```
[32]: Inference -----
The GVI is: 12.959375 %
```

## 7 Conclusion

The emergence of street view images provides a valuable data source for auditing the built environment. Compared to traditional methods such as field surveys and secondary data sources, street view images capture the urban physical environment from

an eye-level perspective, significantly reducing costs and increasing efficiency. In this chapter, we provided detailed guidance on retrieving street view images using the Google Street View API. Then, we demonstrated how deep learning techniques can be effectively leveraged for both element-level and scene-level observations of the built environment. At the element level, individual urban features, such as trees, buildings, and vehicles, can be identified and detected through image segmentation. We showcased the use of the DeepLabV3 model for image segmentation, where up to 150 categories of objects can be identified from street-view images. At the scene level, we employed image classification techniques using the ResNet model, trained on the Places365 dataset, to categorize urban scenes based on their place types. This classification enables researchers to analyze different urban settings. After illustrating element- and scene-level observations, we calculated the Green View Index (GVI) of a given street view image to demonstrate how street view image analysis can benefit urban studies to advance our understanding of the built environment. The GVI computation allows for quantifying the presence of greenery within the urban environment, supporting a comprehensive understanding of streetscapes, and may benefit a wide range of potential urban applications. Overall, this chapter highlights the significant potential of street view imagery and deep learning in urban environment observation. By integrating advanced methods such as image segmentation and image classification, street view imagery ultimately contributes to more informed geospatial and urban studies.

## References

- Anguelov D, Dulong C, Filip D, Frueh C, Lafon S, Lyon R, Ogale A, Vincent L, Weaver J (2010) Google street view: Capturing the world at street level. *Computer* 43: 32–38. [CrossRef](#)
- Badrinarayanan V, Kendall A, Cipolla R (2017) Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence* 39: 2481–2495. [CrossRef](#)
- Biljecki F, Ito K (2021) Street view imagery in urban analytics and GIS: A review. *Landscape and Urban Planning* 215: 104217. [CrossRef](#)
- Chen K, Wang J, Pang J, Cao Y, Xiong Y, Li X, Sun S, Feng W, Liu Z, Xu J, Zhang Z, Cheng D, Zhu C, Cheng T, Zhao Q, Li B, Lu X, Zhu R, Wu Y, Dai J, Wang J, Shi J, Ouyang W, Loy CC, Lin D (2019) Mmdetection: Open mmlab detection toolbox and benchmark. arXiv preprint, 190607155
- Chen LC, Zhu Y, Papandreou G, Schroff F, Adam H (2018) Encoder-decoder with atrous separable convolution for semantic image segmentation. Proceedings of the european conference on computer vision (eccv) 2018. [CrossRef](#)
- Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. 2009 IEEE conference on computer vision and pattern recognition, IEEE. [CrossRef](#)
- Feuillet T, Charreire H, Roda C, Ben Rebah M, Mackenbach J, Compernelle S, Glonti K, Bárdos H, Rutter H, De Bourdeaudhuij I, McKee M, Brug J, Lakerveld J, Oppert JM (2016) Neighbourhood typology based on virtual audit of environmental obesogenic characteristics. *Obesity Reviews* 17: 19–30. [CrossRef](#)
- Gong FY, Zeng ZC, Zhang F, Li X, Ng E, Norford LK (2018) Mapping sky, tree, and building view factors of street canyons in a highdensity urban environment. *Building and Environment* 134: 155–167. [CrossRef](#)
- He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016. [CrossRef](#)

- Ibrahim MR, Haworth J, Cheng T (2020) Understanding cities with machine eyes: A review of deep computer vision in urban analytics. *Cities* 96: 102481. [CrossRef](#)
- Ilic L, Sawada M, Zarzelli A (2019) Deep mapping gentrification in a large canadian city using deep learning and Google Street View. *PloS one* 14: e0212814. [CrossRef](#)
- Kang Y, Abraham J, Ceccato V, Duarte F, Gao, S. and LL, Zhang F, Näsman P, Ratti C (2023) Assessing differences in safety perceptions using geoai and survey across neighbourhoods in Stockholm, Sweden. *Landscape and Urban Planning* 236: 104768. [CrossRef](#)
- Kang Y, Zhang F, Gao S, Lin H, Liu Y (2020) A review of urban physical environment sensing using street view imagery in public health studies. *Annals of GIS* 26: 261–275. [CrossRef](#)
- Kang Y, Zhang F, Gao S, Peng W, Ratti C (2021) Human settlement value assessment from a place perspective: Considering human dynamics and perceptions in house price modeling. *Cities* 118: 103. [CrossRef](#)
- Li X, Ning H, Huang X, Dadashova B, Kang Y, Ma A (2022) Urban infrastructure audit: An effective protocol to digitize signalized intersections by mining street view images. *Cartography and Geographic Information Science* 49: 32–49. [CrossRef](#)
- Li X, Ratti C, Seiferling I (2018) Quantifying the shade provision of street trees in urban landscape: A case study in Boston, USA, using Google Street View. *Landscape and Urban Planning* 169: 81–91. [CrossRef](#)
- Li X, Zhang C, Li W, Ricard R, Meng Q, Zhang W (2015) Assessing street-level urban greenery using Google Street View and a modified green view index. *Urban Forestry & Urban Greening* 14: 675. [CrossRef](#)
- Lu Y (2019) Using Google Street View to investigate the association between street greenery and physical activity. *Landscape and Urban Planning* 191: 103435. [CrossRef](#)
- Lu Y, Ferranti EJS, Chapman L, Pfrang C (2023) Assessing urban greenery by harvesting street view data: A review. *Urban Forestry & Urban Greening* 83: 127917. [CrossRef](#)
- Rzotkiewicz A, Pearson AL, Dougherty BV, Shortridge A, Wilson N (2018) Systematic review of the use of Google Street View in health research: major themes, strengths, weaknesses and possibilities for future research. *Health & Place* 52: 240–246. [CrossRef](#)
- Yang J, Rong H, Kang Y, Zhang F, Chegut A (2021) The financial impact of street-level greenery on New York commercial buildings. *Landscape and Urban Planning* 214: 104162. [CrossRef](#)
- Ye C, Zhang F, Mu L, Gao Y, Liu Y (2021) Urban function recognition by integrating social media and street-level imagery. *Environment and Planning B: Urban Analytics and City Science* 48: 1430–1444. [CrossRef](#)
- Yin L, Wang Z (2016) Measuring visual enclosure for street walkability: Using machine learning algorithms and Google Street View imagery. *Applied Geography* 76: 147–153. [CrossRef](#)
- Zhang F, Wu L, Zhu D, Liu Y (2019) Social sensing from street-level imagery: A case study in learning spatio-temporal urban mobility patterns. *ISPRS Journal of Photogrammetry and Remote Sensing* 153: 48–58. [CrossRef](#)
- Zhang F, Zhang D, Liu Y, Lin H (2018) Representing place locales using scene elements. *Computers, Environment and Urban Systems* 71: 153–164. [CrossRef](#)
- Zhang F, Zhou B, Liu L, Liu Y, Fung HH, Lin H, Ratti C (2018) Measuring human perceptions of a large-scale urban region using machine learning. *Landscape and Urban Planning* 180: 148–160. [CrossRef](#)

- Zhou B, Lapedriza A, Khosla A, Oliva A, Torralba A (2017) Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence* 40: 1452–1464. [CrossRef](#)
- Zhou B, Zhao H, Puig X, Fidler S, Barriuso A, Torralba A (2017) Scene parsing through ADE20K dataset. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 633–641



© 2026 by the authors. Licensee: REGION – The Journal of ERSA, European Regional Science Association, Louvain-la-Neuve, Belgium. This article is distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).

---