Automated Geogusser Meta Finder

Riedl Manuel[1]

[1]Graz University of Technology, Graz, Austria

November 30, 2025

Abstract

This work presents an automated approach for identifying locations containing country-specific bollards in Google Street View imagery for the online game GeoGuessr. For this purpose, a command-line application was developed. The approach was motivated by the goal of efficiently generating learning maps that include GeoGuessr-relevant metas. We constructed a custom dataset of annotated Street View images covering 20 different European bollards. A two-stage YOLO approach was employed: the first model detected all types of bollards in low-resolution panorama images, while the second model performed country-specific classification on the resulting crops. The results show that the system is capable of finding relevant locations for learning maps, eliminating the need for manual work.

1 Introduction

This project is inspired by the game GeoGuessr (nd), an online competitive game with over 80 million players. In the game, users must predict a location based on a randomly selected Google Street View image. The player's guess, which is closer to the actual location, gets more points. While visual clues like the landscape, vegetation, or the architectural style can provide a rough indicator of the region, infrastructure elements like power poles, road markings, bollards, or license plates give much more precise information about the location for the average player. The GeoGuessr community calls these kinds of clues "META" (**M**ost **E**ffective **T**actics **A**vailable). Hundreds of metas have been documented for all countries available in Google Street View on websites such as Plonkit. A new player can learn these metas by either studying them directly on websites, or in a more interactive way by using community-made maps, where locations are preselected with the meta types you want to learn. With the help of a Chrome extension called LearnableMeta, this approach is the most effective way for a new player to learn the game. The main limitation of these learnable maps is that they only offer a limited number of different locations per meta. Players quickly recognize the locations by hard, and stop paying attention to the metas themselves. Studies in Category Learning, such as an experiment where participants had to memorize different types of rocks, have shown that a greater variety of examples leads to better generalization. In particular, interleaved learning, where examples are shown in varied contexts, has been found to outperform blocked learning, which often results only in instance-based memorization (Do and Thomas, 2023). Because of the advantages of interleaved learning, this project aims to build an automated approach to find locations where a specific meta is present. A YOLO-based image detection model is trained to identify these metas and provide players with a varied set of examples for more effective learning.

2 Related Work

Using Google Street View images in combination with image detection has been used in a variety of prior studies. For example, Gebru et al.

(2017) analyzed images from the 200 largest U.S. cities and used a deep learning model to detect cars in the Google Street View images. Based on the different detected types of cars in a city, they predicted demographic attributes, like income, pollution levels, and crime rates.
Hara et al. (2014) used machine learning with the help of computer vision to detect curb ramps in Google Street View images. They trained their model on a labeled dataset consisting mostly of intersection images from the U.S., with annotations marking ramps and non-ramps.
Other examples for the use of Google Street View imagery include the study by Li et al. (2015), in which a modified version of the Green View Index (GVI) was applied to assess street-level greenery in New York City, as well as the work by Dubey et al. (2016) who trained a deep learning model to quantify perceived urban appearances such as safety, beauty, or how wealthy a street-level image looks. Furthermore, Haas et al. (2023) developed a model specifically trained to predict locations in a GeoGuessr style. Their model named PIGEON is trained on planet-scale Street View data and uses a multi-task architecture that combines CLIP embeddings and clustering techniques. Given a four-image panorama, it places over 40% of its predictions within 25 kilometers of the target location, outperforming even the most experienced GeoGuessr players.

3 Materials and Methods

3.1 Dataset Aggregation

Currently, there exists no publicly available dataset for GeoGuessr metas. Therefore, we created a dataset consisting of both official and unofficial Google Street View coverage, with a focus on european street bollards. Bollards are among the first metas new players learn, as they vary across countries, such as in color, shape, and reflectors.

(a) Germany **(b)** Spain **(c)** Poland

Figure 1: Bollards from European countries

In some countries, the differences between bollards are only minor; for example, German bollards have two nails on the reflector, whereas Luxembourg bollards have three. Or in the case of Austrian and Slovenian bollards, they only differ in the red tone used on the reflector.

(a) Germany **(b)** Luxenburg **(c)** Austria **(d)** Slovenia

Figure 2: Bollard reflectors with minor differences

To construct the dataset, we conducted the following steps:

- Relevant Google Street View locations containing bollards were manually searched and selected using MapMakingApp.
- A Python script was used to automatically capture Street View images from the saved latitude/longitude coordinates (1904x933).
- Bollards were labeled using rectangular bounding boxes in YOLO format via MakeSenseAI (2019). For some countries, also the reflector on the bollard received an extra label.

Based on these annotations, two separate datasets were created. The first dataset contains

the Street View images with the corresponding bounding box labels for all annotated objects. The second dataset consists of cropped bollard images, where each labeled bollard is extracted from the original image and stored as an individual image. This cropped dataset is intended for more fine-grained image/bollard classification.

Country	Full-Image	Cropped	Reflector
Austria	200	311	–
Croatia	120	131	–
Czechia	130	140	–
Denmark	151	159	–
Estonia	151	147	–
Finland	150	152	–
France	150	184	–
Germany	200	302	136
Hungary	120	126	–
Iceland	110	111	–
Italy	150	183	–
Latvia	150	162	–
Lithuania	150	159	–
Luxembourg	102	54	52
Norway	150	153	–
Poland	120	134	–
Portugal	89	107	–
Slovenia	150	185	127
Spain	151	134	–
Sweden	150	159	–
Total	**2884**	**3268**	**315**

Table 1: Number of labeled images per country for the full-image dataset, cropped-bollard dataset, and reflector-labeled bollards.

3.2 Model Implementation

This section focuses on the detection pipeline used to identify locations that contain at least one bollard. It consists of three main parts: location selection, automated Street View capture, and image-based bollard detection.

3.2.1 Location selection

Because a completely random selection of locations often results in areas with no Street View coverage, the selection process uses latitude–longitude coordinates along specific road types. Major roads (e.g., federal roads and motorways) were primarily selected, as these are more likely to contain bollards than smaller roads. This approach results in fewer candidate locations but still provides sufficient samples to ensure randomness. For example, the preselected datasets contain 461.691 locations for Austria, 1.717.492 for Germany, 564.628 for Italy, and 88.020 for France. This preselection guarantees fewer invalid locations and thus speeds up the location collection process. For the selection of coordinates, we used Overpass Turbo. It is a web-based data mining tool for OpenStreetMap that allows querying geospatial data based on specific criteria such as road type or geographic region.

3.2.2 Automated Street View capture

This project focuses on a fully free and open-source approach; because of that, the Google Street View API was not used, even though it would have significantly simplified the process. Instead, we used the Selenium (2025) Python package, which enables automated control of a simulated browser environment. Actions such as panning (e.g., pressing the left arrow key for a certain duration) or zooming (simulated through JavaScript-based keypress events, since Selenium's built-in functions did not work) were emulated. Selenium was used in two stages of our pipeline:

- **Panorama image capture:** Load the randomly chosen location (lat/lon) in Google Street View, pan around the location, and take four screenshots (4 pan steps - 360 degrees).
- **Zoomed image capture:** For every detected bollard in one of the panorama images, the same location is reloaded using the identical pan angle. The offset between the image center and the detected bollard is calculated and used to center the bollard horizontally on the screen. Centering the bollard before zooming is necessary because objects near the image borders may become partially invisible after zooming. That's due to the spherical projection of the Street View imagery, where spatial relationships are not linear.

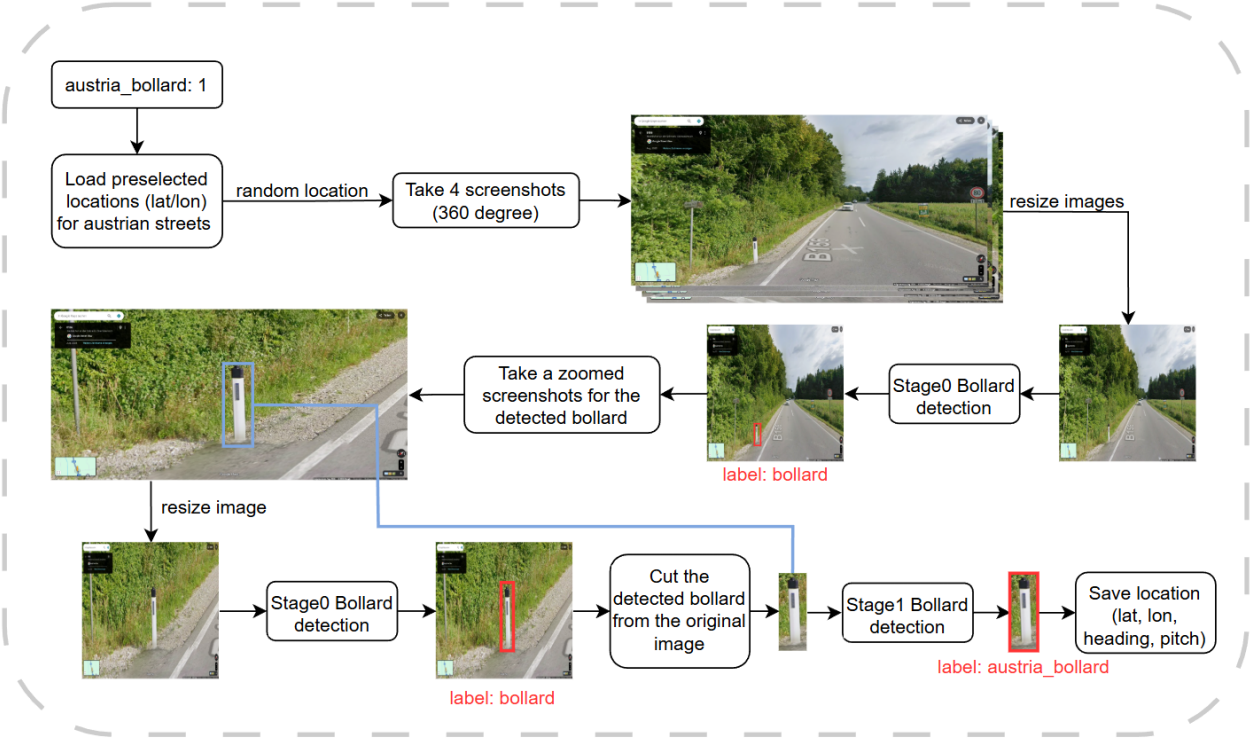

Figure 3: Example of the pipeline architecture for finding a location containing an Austrian bollard

After the bollard is centered, a zoomed screenshot is taken to capture as many details of the bollard as possible.

3.2.3 Bollard detection

For the image detection, we used the well-established YOLO (**Y**ou **O**nly **L**ook **O**nce) model introduced by Redmon et al. (2015). YOLO is a deep learning-based object detection framework that processes the entire image in a single convolutional neural network, making it fast and suitable for real-time applications. A YOLO model can be trained completely from scratch, or a pretrained version can be used, which already detects objects such as cars, people, or bicycles. The advantage of using a pretrained model is that it has already learned to recognize edges, shapes, and basic object features, which allows the model to adapt to a new class much faster. For this project, we used the pretrained YOLOv8nano model as our basis, because tests with larger models did not result in a significant performance increase. Since training a model on a screenshot resolution of 1904x933 would be really computationally expensive, we went for a two-stage pipeline approach in which both stages use a reduced resolution of 640x640.

- **Stage 0:** The first stage detects all kinds of bollards, without distinguishing between countries. This is because in the resized, low-resolution screenshots, the details of the bollards are barely visible, making it hard to differentiate between countries. All country-specific labeled bollards were therefore assigned with the general label "bollard" during training.

- **Stage 1:** The second stage receives the cropped bollards from Stage 0. These cropped images have much better resolution since they are cut directly from the original zoomed screenshot. This allows the model to classify each bollard according to its country-specific label. The model is trained on bollard crops extracted from the original screenshots, including twenty country-specific bollard labels and three different reflector labels.

3.2.4 Detection Pipeline

The complete detection pipeline, which is built upon the above-described components, consists of several steps. First, a random candidate location is chosen from the target coun-

try where a location containing a bollard is desired. After that, each of the four images from the panorama screenshot is processed by Stage 0. For every detected bollard, a new zoomed-in bollard-centered screenshot is taken and processed again by Stage 0. This second pass of Stage 0 helps to filter out false positives, as objects that are falsely detected as bollards in the low-resolution image (e.g., street signs) are unlikely to be detected again in a higher-resolution, zoomed version. Next, each detected bollard in the zoomed screenshot is cropped from the screenshot (1904x933) and passed to Stage 1, which performs the country-specific bollard labeling. Since the model is used in a location collection scenario, the label of the bollard is known in advance. Therefore, for each country, a predefined list of acceptable labels is used to handle false negatives better. For example, when collecting locations in Austria, we anticipate austria_bollard detections. Because of the visual similarity between Austrian and Slovenian bollards, the model occasionally confuses them, as seen in the confusion matrix from Stage 1 (Figure 4). To speed up the collection process, both austria_bollard and slovenia_bollard are therefore valid detections. If a valid bollard is detected by Stage 1, the corresponding heading, pitch, and zoom of the location are stored in a JSON file. This JSON file can then be imported directly into GeoGuessr. An example illustrating the complete detection pipeline for a location is shown in Figure 3.

4 Results

This section presents the results of the two trained bollard classifiers (Stage 0 and Stage 1), followed by the results of the full application pipeline.

4.1 Bollard classifiers

Each model was trained for a total of 100 epochs, with an 80/20 train–validation split for both Stage 0 and Stage 1.

4.1.1 Stage 0

The Stage 0 model achieves solid performance, with a precision of 0.956, a recall of 0.788, an mAP@50 of 0.883, and an mAP@50–95 of 0.673. Although the mAP@50–95 is relatively low, this is not critical for the current usecase, since only the approximate bollard location is needed to set the camera correctly.

4.1.2 Stage 1

Stage 1 reached a precision of 0.943, a recall of 0.951, an mAP@50 of 0.961, and an mAP@50–95 of 0.920. Stage 1 is used more like an image classificator, which explains the high mAP@50–95. Overall, the model shows good performance in accurately classifying the bollard classes.

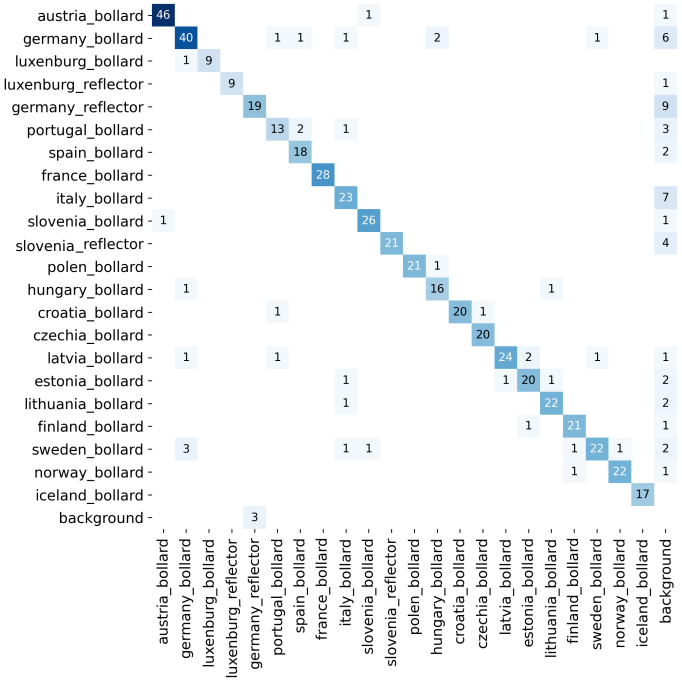

Figure 4: Confusion matrix of Stage 1

4.2 Bollard Detection Application

We developed a command-line application which serves as a user interface for the bollard detection pipeline. The application consists of three main screens: a home screen (Figure 6), a meta selection screen (Figure 7), and a fetching screen (Figure 8). To evaluate the application, we searched for at least 40 locations per country and manually reviewed how many of

these were invalid (locations which not contain a bollard). As shown in Table 2, the GeoGuessr Meta Finder application performs very well, with only a few invalid locations. Because bollards are much more common in some countries than in others, the time per location varies heavily between countries. We have to note that these statistics are not representative, as we only sampled around 40 locations per country. An overview of the geographic distribution of the found locations can be seen in Figure 9 and Figure 10 shows examples of the playable map in GeoGuessr. The generated map is publicly available on GeoGuessr (subscription required). Free sample challenges can be accessed here: Sample 1, Sample 2, and Sample 3. The source code is available on GitHub (Riedl, 2025).

Country	Checked	Time/Loc (s)	Invalid (%)
Austria	2.18	81.8	5.88
Germany	2.88	108.8	0.00
Luxembourg	3.85	135.0	2.50
Slovenia	2.45	84.1	0.00
France	16.43	360.4	7.50
Spain	6.18	162.9	2.50
Portugal	11.35	267.3	10.00
Italy	10.43	237.9	0.00
Poland	14.25	158.6	8.89
Hungary	3.30	99.3	2.50
Czechia	4.30	121.2	2.50
Croatia	3.68	99.2	2.50
Lithuania	3.83	117.8	3.17
Latvia	4.90	92.1	2.50
Estonia	2.45	87.0	1.32
Sweden	2.70	82.5	9.52
Finland	7.15	165.0	0.00
Norway	19.47	390.4	7.50
Iceland	5.27	108.0	0.00

Table 2: Average checked locations, average time per location (in seconds), and average invalid locations (in percentage) for each country to find one valid location

5 Conclusion and Future Work

The results of our work show that we developed a reliable two-stage detection pipeline that can automatically find locations containing bollards. The multi-stage architecture worked well and made a large, complex YOLO model unnecessary. Future work can extend the labeled dataset with additional object classes, for example various types of street signs like “Give Way”, “One Way Street”, or “Stop” signs. Also, a language-detection stage could be integrated to find location with country-specific word endings or letters. For example, Nordic countries can often be distinguished by their characteristic letters, such as å, æ, ø, or þ. Furthermore, a large language model can be used as an autonomous AI agent that can navigate Google Street View on its own and search for locations of interest. The LLM could use country-specific meta information, for example, the descriptions from Plonkit as context for identifying relevant locations.

Appendix

(a) Full-Image dataset

(b) Cropped dataset

Figure 5: Examples from the training dataset for Stage 0 and Stage 1

Figure 6: Home screen of the Meta-Finder Application

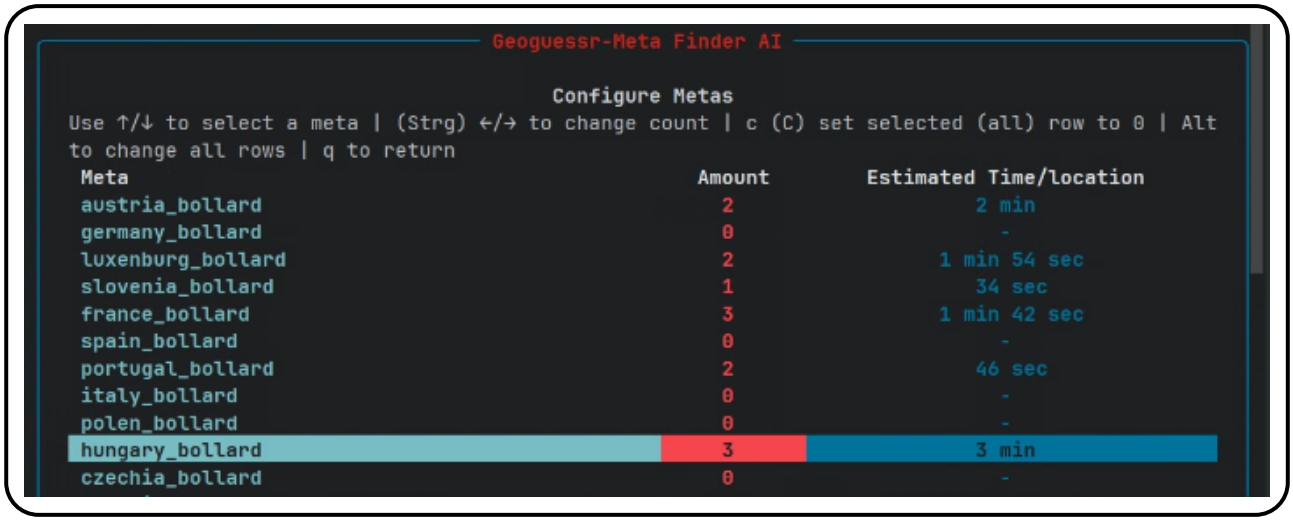

Figure 7: Selection screen of the Meta-Finder Application

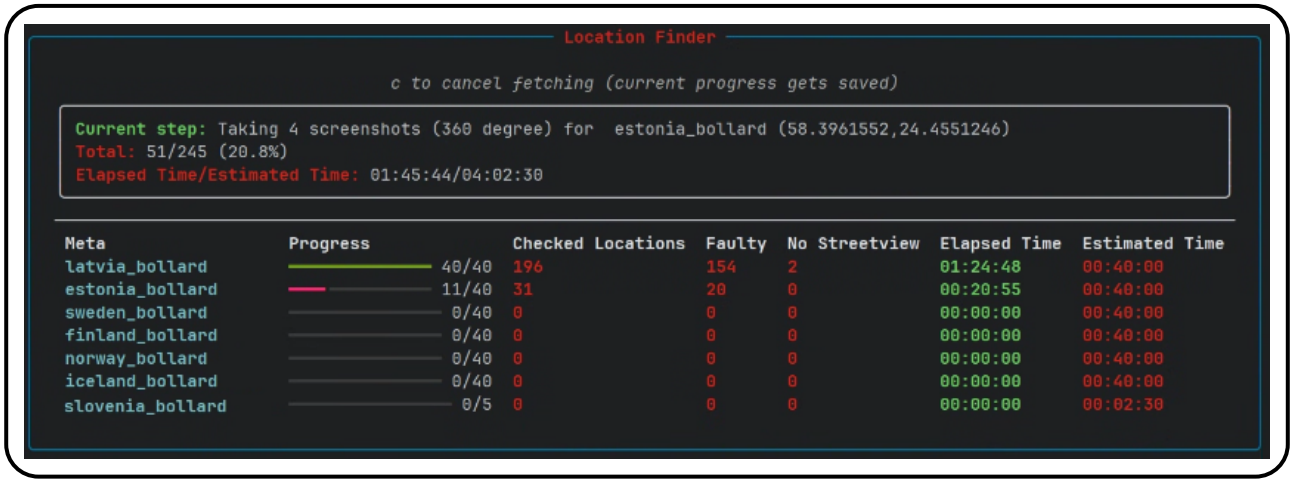

Figure 8: Fetching screen of the Meta-Finder Application

Figure 9: Overview of the geographic distribution of the found locations

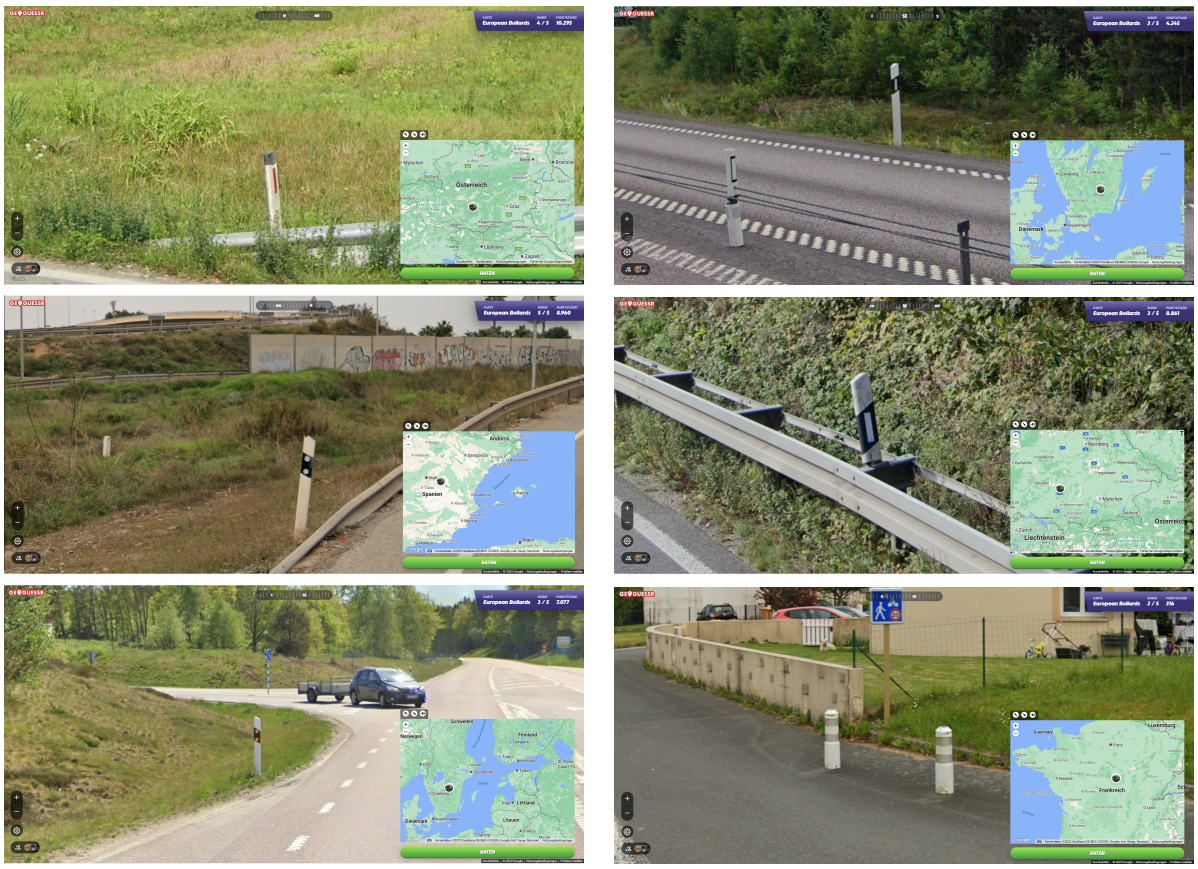

Figure 10: Example locations from the playable map in GeoGuessr

References

Do, L. A. and Thomas, A. K. (2023). The underappreciated benefits of interleaving for category learning. *Journal of Intelligence*, 11(8):153.

Dubey, A., Naik, N., Parikh, D., Raskar, R., and Hidalgo, C. A. (2016). Deep learning the city: Quantifying urban perception at a global scale. In *European conference on computer vision*, pages 196–212. Springer.

Gebru, T., Krause, J., Wang, Y., Chen, D., Deng, J., and Fei-Fei, L. (2017). Fine-grained car detection for visual census estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

GeoGuessr (n.d.). Geoguessr – deutschland. `https://www.geoguessr.com/de`. Accessed: 2025-11-05.

Haas, L., Skreta, M., Alberti, S., and Finn, C. (2023). Pigeon: Predicting image geolocations.

Hara, K., Sun, J., Moore, R., Jacobs, D., and Froehlich, J. (2014). Tohme: detecting curb ramps in google street view using crowdsourcing, computer vision, and machine learning. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 189–204, New York, NY, USA. Association for Computing Machinery.

LearnableMeta (n.d.). Maps. `https://learnablemeta.com/maps`. Accessed: 2025-11-05.

Li, X., Zhang, C., Li, W., Ricard, R., Meng, Q., and Zhang, W. (2015). Assessing street-level urban greenery using google street view and a modified green view index. *Urban forestry & urban greening*, 14(3):675–685.

MakeSenseAI (2019). Makesenseai — free browser-based image labeling tool. `https://makesense.ai/`. Accessed: 2025-11-07.

MapMakingApp (n.d.). MapMakingApp. `https://map-making.app/`. Accessed: 2025-11-07.

Overpass Turbo (n.d.). Overpass turbo. `https://overpass-turbo.eu/`. Accessed: 2025-11-12.

Plonkit (n.d.). Guide to geoguessr. `https://www.plonkit.net/guide`. Accessed: 2025-11-05.

Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640.

Riedl, M. (2025). Geoguessrai application. `https://github.com/ManuelRiedl/GeoGuesserAI_new`.

Selenium (2025). Selenium. `https://www.selenium.dev/`. Accessed: 2025-11-12.