Street View Recognition

Hsiang-Sheng Huang*
Dept. of CS, NTHU
Hsinchu, Taiwan

Chiang Cheng-Han*
Dept. of EE, NTHU
Hsinchu, Taiwan

Chuang Bing-Cheng*
Dept. of EECS, NTHU
Hsinchu, Taiwan

Cheng-En Tang*
Dept. of EECS, NTHU
Hsinchu, Taiwan

Shan-Jou Huang*
Dept. of EECS, NTHU
Hsinchu, Taiwan

Yun Huang*
Dept. of EECS, NTHU
Hsinchu, Taiwan

***Abstract*—This study explores a machine learning approach to street view recognition, aiming to identify the country corresponding to a given street view image. Using images from 10 countries across the world, we collected street view images via the Google Street View API and employed data augmentation to enhance the dataset. A Vision Transformer architecture was selected for its ability to capture global patterns and complex features. Eventually, the model we trained achieved an F1 score of 0.87 and an accuracy of 92.9%, significantly outperforming human participants (46.4%) in the game that enable the model to compete with human. This research highlights the potential of deep learning in the classification of geographic images and its practical applications in interactive tools.**

I. Introduction

Inspired by GeoGuessr, an interactive game in which players can guess geographical locations, we decided to use machine learning techniques to achieve this task. Recognizing street-view images is a significant challenge for humans, as it requires identifying subtle features unique to different countries. However, machine learning models excel at focusing on key features, enabling them to achieve higher prediction accuracy.

In this work, we completed data collection, data augmentation, model training, evaluation, and integration with a game system. We utilized the Google Street View API to retrieve images and experimented with three different models: ResNet and Vision Transformer to predict the country based on the given images. Finally, we developed an interactive game that allows users to compete with machine learning models in guessing countries from street-view images.

II. Related Work

A. Training Geoguessr AI with CNN Model

In [1], the dataset is collected among 5 cities in the US using Google Street View API. They used Wideresnet as their base model. The author also proposed that using 3D CNN might further improve performance. The difference from our work is that we initially wanted to distinguish the street view image exclusively in Taiwan, and it turned out that it was much harder than the US version since the land area is much smaller, and the distance error was too large to be meaningful. Thus, we changed our task to distinguish among 10 countries around the world. It is also a more reasonable task for an AI model since we cannot be sure if such a distribution exists in street view pictures from Taiwan that can be learned by the models.

In the beginning, our goal was to predict the coordinates of the picture, so we treated it as a regression problem. [2] also had the same starting point but faced the same challenges as we did.

The relation between street view pictures and coordinates is too implicit for the model to learn, and it ended up predicting the mean value of coordinates so as to reach the minimum error value. We got a similar result in Taiwan's dataset, the model's prediction scattered evenly across the land, which made it a useless result. In [2], they proposed the solution to solve this by changing the problem to classification. They divided the world map into several regions and allowed the model to classify the pictures. What's different in our work is that we didn't split the regions by using cluster algorithm. Instead, we directly group them by the country. This is because distances between countries are larger in our data. This makes country label good enough to group the pictures.

III. Method

A. Data Collection

We selected ten countries and collected 6,000 street view images in each country. Finally, we collected totally 60,000 street view images and have a CSV file to store the information of each image. We split those images into 40,000 images of training set, 10,000 images of validation dataset, and 10,000 images of testing dataset. These images are captured using the Google Street View API. After given a pair of longitude and latitude coordinates, a request is sent through this API, and finally the street view image of this coordinate point is obtained.

In the beginning, we just used the Random function to get a pair of longitude and latitude within a rectangular block. However, there are many coordinate points that doesn't have a image (such as mountains and forests),

the hit rate of using this method to send a request to get non-blank image is only about 0.1. In addition to the very low hit rate, another problem is that there will be some images taken by unknown users (such as indoor photos).

In order to solve the problem and increase the hit rate of the request, we imported the `osmnx` package. After selecting a country and city, we can use it to generate a road map that can be driven in this city. After getting the road map, we can then randomly select a coordinate from the road map and then use the API to send the street view image request. Through the osmnx package, the hit rate of requests is increased from the original 0.1 to 0.9. Because a large proportion of street view cars appear in image from some countries, we have also increased the angle in the request to reduce the proportion of street view cars.

For some bad images that have problems such as overexposure, large mosaics, tunnels, large text billboards, etc., we used human judgment to pick out the index of bad images, and wrote a program that can batch delete these unsuitable images. This program can delete the unsuitable photos in the folder, reorder the index of the remaining photos, and update the csv file of the recorded data.

B. Model Training

We built our model using the `Pytorch` framework, running on NVIDIA GeForce RTX 2080 Ti to accelerate the training process. All of the street view pictures are located in the same folder, and the labels are located in one CSV file. We first inherited the *Dataset* class to make the loading data process more convenient. We then tried different models which are pre-trained on ImageNet as our base model.

During the training process, a checkpoint is saved every few epochs. By doing this, if something goes wrong afterward, we can select the best checkpoint.

Also, to better visualize the training process, we use TensorBoard to record the loss and metric.

Initially, our goal was to predict the coordinates of the street view picture so that the model could be trained as a regression problem. We first tried simple MSE as loss function, then up-scale the dataset size, finally also mixed regression and classification together. Unfortunately, those methods didn't work. The output of the model was uniformly scattered on the map. Thus, it was a random coordinate generator and didn't serve a meaningful purpose.

We then decided to turn this into a pure classification problem. We tried pre-trained ResNet and Vision Transformer. It then connects to 10 neurons as the output layer. We used cross-entropy as the loss function and F1 score as the metric because the dataset was not balanced in the beginning. We used learning rate 1e-4, batch size 128. The model could reach an F1 score of around 0.87 after 20 epochs, and it started to saturate. Figs. 1 and 2 are the loss function plots using ResNet and ViT as base model.

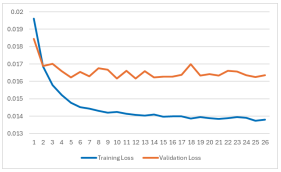

Fig. 1. ResNet Loss

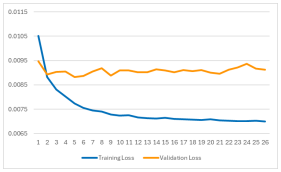

Fig. 2. ViT Loss

We also applied some regularization techniques such as data augmentation, label smoothing, and class weight in loss function. To verify the effectiveness of those measures, we did the ablation test. It turned out that only data augmentation had more impact on results. The class weight and label smoothing may not be needed once we have a balanced and larger dataset. Without data augmentation, the validation F1 score dropped slightly from 0.91 to 0.90.

Finally, to see the generalizability of our model, we collected another 10,000 of street view images as test set. The F1 score is 0.73 for the test dataset. Although there's a gap between validation F1 score and testing F1 score, implying there might be overfitting issue, by the analysis of IV, we can see that the model still learned some smart strategies just like human player would use in real GeoGuessr game.

C. Game

To demonstrate our model's capabilities and compare its performance with human players, we integrated the model into an interactive game between humans and machines. The game consists of ten rounds, where, in each round, players and the model are presented with a random street view image selected from a fixed pre-made testing set, and the task is to choose the correct answer from four given options. The answer generated by the model is to choose the one with the highest probability among the four options based on the guessing result. At the end of the game, it displays the winning rates of both the players and the model, and the battle results are saved into CSV files to facilitate further data analysis.

For development, we used the Python module `Pygame` to build the game. The game process is divided into the following scenes, controlled by a centralized `SceneManager`:

- **Start Scene:** The initial screen where the game begins.
- **Round-Begin Scene:** A 2-second countdown is displayed here, ensuring a smooth transition between rounds.
- **Game Scene:** The core scene in this game. Players and the model are shown a random street view image along with four answer options. Players make their guesses, while the model makes its decision based on the highest probability among the four options.

- **Result Scene:** After the player and the model make their guesses, the results for the current round are displayed, including:
  - The correct answer.
  - The player's choice.
  - The model's prediction and its probability distribution for the four options.
- **Final-Result Scene:** At the end of all rounds, this scene summarizes the game results. It displays the winning rates of the players and the model and saves detailed results into a CSV file for further analysis.

The scene-based design, managed by the `SceneManager`, ensures modularity and simplifies the flow between different scenes of the game.

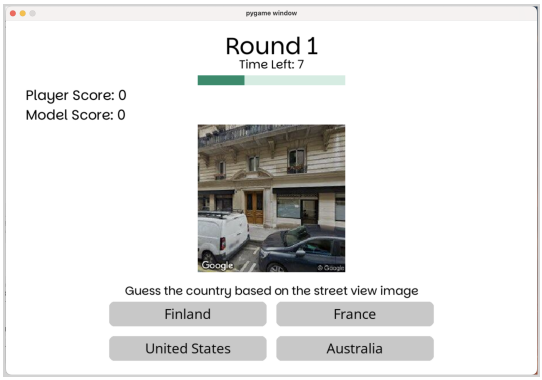

Fig. 3. Game screen of each round

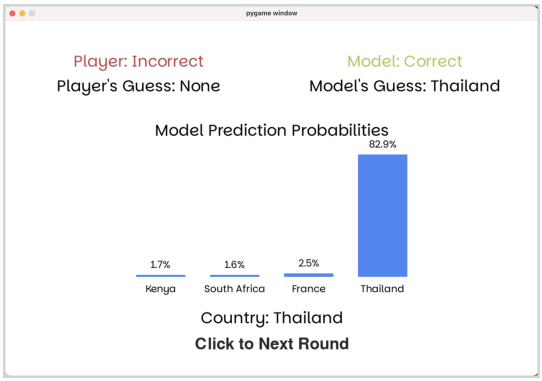

Fig. 4. Result of each round

The game workflow is shown in the flow-chart (Fig. 5). The model is first trained on a dataset with 10 classes, and its weights are saved. During the game, a pre-trained model is loaded, and a random image is selected. Four answer options are presented, and the model generates probabilities for all 10 classes. These are filtered to match the four options, with the final prediction made using the `argmax` function. This repeats for each round until the game ends.

The bar graph in Fig. 6 is the statistical result after we collect data from several test cases, showing the accuracy

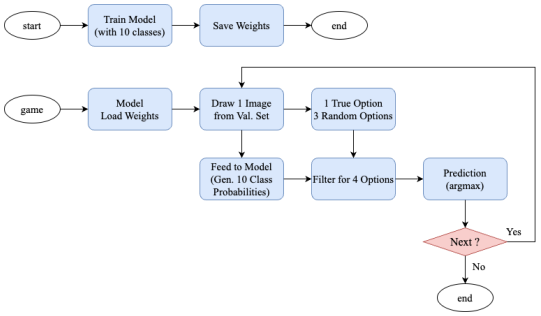

Fig. 5. Flow chart of the gameplay

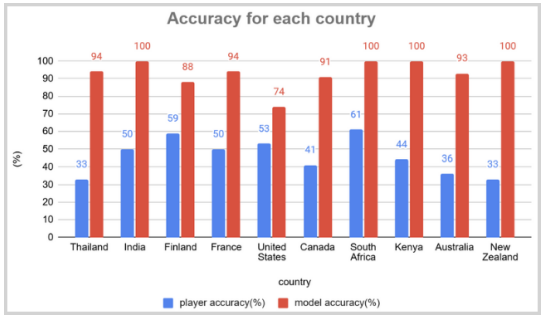

Fig. 6. Accuracy for each country

of humans and the model in predicting images from various countries. The average accuracy of the model is 92.9%, while the average accuracy of human players is 46.4%. We can see that in general the prediction accuracy of our model is better than that of humans in various countries. It is worth noticing that the accuracy of the United States and Canada is low compared to others; this is due to the similarities of their landscape.

IV. Result

A. Training Result

We achieved high accuracy using the ViT-based model (Vision Transformer), with 99% on the training dataset and 87% on the validation dataset (Table I). Although the test on the ResNet-based model outperforms ViT on the validation set slightly, we believe that this model lacks an attention mechanism, so we do not consider ResNet50 as our final model.

TABLE I
Performance of Model

Model	Dataset	Accuracy	F1 Score	Recall
ViT	Training Set	99%	0.99	0.99
ViT	Validation Set	87%	0.87	0.87
ViT	Testing Set	78%	-	-
ResNet50	Training Set	99%	0.99	0.99
ResNet50	Validation Set	92%	0.92	0.92
ResNet50	Testing Set	30%	-	-

B. Realizing the model

First, we will discuss the attention mechanism in the Vision Transformer (ViT). ViT works by **dividing the input image into $n \times n$ patches** (in our model, $n = 7$). These patches are then processed through self-attention, allowing the model to capture relationships between different parts of the image and learn global features effectively. This mechanism enables ViT to understand the overall structure of an image, rather than focusing solely on localized regions.

To analyze how the model attends to different parts of the image, we used two visualization techniques: heatmap occlusion and saliency maps. These methods revealed some intriguing patterns in the model's behavior.

Using heatmap, we observed that the model could capture rough global features of the image. This is likely due to the self-attention mechanism, which allows the model to focus on spatial relationships across the entire image. To illustrate, Fig. 7 shows the original input image, and Fig. 8 displays its corresponding heatmap result, highlighting the regions the model considers most significant.

Fig. 7. Original Image

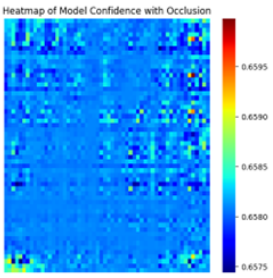

Fig. 8. Heatmap Result

The performance of heatmap is limited due to the occlusion size and its inability to account for pixel relationships effectively.

Next, we applied saliency maps (using gradient ascent) to analyze the model's behavior. Figs. 9 and 10 illustrate the first example, showing the original image and its corresponding saliency map. Similarly, Figs. 11 and 12 present another example for further analysis. These visualizations highlight the areas of the image that the model focuses on when making predictions.

Using the saliency map, we observed that the model can identify many important features in the image, such as cars, trees, motorcycles, and houses. These detailed features indicate the model's ability to focus on meaningful visual elements.

Through these two analysis methods, we confirmed that the model performs well in distinguishing countries based on a single photo.

Fig. 9. Original Image

Fig. 10. Saliency Map

Fig. 11. Original Image

Fig. 12. Saliency Map

C. Dataset

Looking at the confusion matrix (Figs. 13 and 14), we can see that some countries are challenging for the model to differentiate, such as Canada and the United States, or New Zealand and Australia. This is likely due to their similar landscapes and natural environments. On the other hand, the model performs exceptionally well in predicting countries like India, as its unique landscapes and street views are distinctly different from those of other countries.

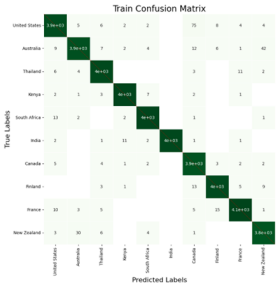

Fig. 13. Training Confusion Matrix

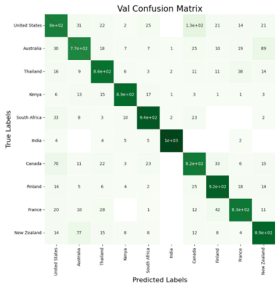

Fig. 14. Validation Confusion Matrix

D. Result

When generating the saliency map, gradient ascent is applied, which modifies the image. The following image (Fig. 15) illustrates an example. This process highlights the importance of different features within the image.

After running iterations on various images, we observed that the model tends to prioritize trees first, followed by cars,

Fig. 15. The iterations of the image

and finally people and buildings.

This finding indicates that the natural environment plays a significant role in the model's predictions. Moreover, this observation aligns with the characteristics discussed in IV-C.

V. Conclusion

Our ViT model can successfully distinguish the street view from ten different countries, and the evaluation works indicate that global patterns such as vegetation and regional structures play a significant role in how the model makes decisions. Although the model performs pretty well in recognizing specific features and competing with humans, there is still room for improvement in distinguishing the slight difference between countries with similar environmental characteristics, such as the United States and Canada.

For the application, it can be successfully applied to photo album categorization or social media photo tagging. Furthermore, we believe that our model has the potential to contribute to various interdisciplinary fields, such as urban planning by analyzing street views to identify infrastructure patterns or vegetation coverage.

VI. Data and Code Availability

We used GitHub to manage and maintain the development of this project. The complete codebase is publicly available at https://github.com/WeakGT/Street-View-Recognition. The main branch contains all essential files, and the code is properly structured and ready to run.

Additionally, we have provided the trained model weights through a Google Drive link mentioned in the README file. Users can download the weights and play the game seamlessly.

VII. Author Contribution Statement

- **Hsiang-Sheng Huang*:** The project manager (PM). Primarily contributed to game development and ensured the project's main branch functioned properly.
- **Chiang Cheng-Han*:** Responsible for visualizing the model's learning outcomes and conducting in-depth analysis.
- **Chuang Bing-Cheng*:** Focused on selecting suitable models and loss functions, and responsible for training the models.
- **Cheng-En Tang*:** Contributed to game development and oversaw the written report.
- **Shan-Jou Huang*:** Responsible for data collection and augmentation, filtering appropriate countries, and excluding unsuitable data.
- **Yun Huang*:** Responsible for data collection and augmentation, filtering appropriate countries, and excluding unsuitable data.

References

[1] Alex Stelath, "Geoguessr-AI". 2024. [Online]. Available: https://github.com/Stelath/geoguessr-ai.

[2] Sho Kiami, zchapman1. "GeoKnowr, A lightweight GeoGuessr AI". 2022. [Online]. Available: https://github.com/shokiami/GeoKnowr.