GeoGuessr AI

Chloe Euston, Armaan Dave, Chongzhe Jin, Jirath Lojanarungsiri

CS4100: Artificial Intelligence
Northeastern University
December 2025

https://github.com/ChongzheJin/CS4100FinalProject

1 Abstract

Popular online game GeoGuessr is a game where players randomly spawn in an unknown location on Google Maps Street View and must determine their position using only visual context clues. Motivated by this geolocation challenge, we create a two-agent convolutional neural network system that can predict where players are in the United States and surround areas. The first agent will analyze a single Street View imagery and outputs a latitude and longitude coordinate, while the second agent classifies and outputs a probability distribution of the image originating from one of the grids that we divided the United States into. By combining the two outputs, our system aims to improve geolocation accuracy in ways that could be comparable to the strategies employed by professional GeoGuessr players.

2 Introduction

Image-based geolocation is a challenging computer vision problem. Unlike traditional object-identifying problems, geolocation depends on capturing more obscure features like vegetation density that can vary significantly across regions. In the hit game Geoguessr, geolocation is gamified into an application where players are randomly placed in unknown locations on Google Maps Street View, and they must use visual context clues to figure out where they are in the world. Despite the challenging nature of geolocation, professional GeoGuessr players managed to develop a sort of heuristic, or "metas," that allow them to pinpoint locations from a single image frame with remarkable accuracy. They often rely on similar obscure features such as the color of the dirt, the triangulation of the sun-cast shadows, and even the camera quality of Street View images [1]. Their inhuman capabilities have demonstrated that—with sufficient pattern recognition—it is possible to accurately pinpoint a location using a single image.

Given how computer-like these professional players are at recognizing image features, it poses the question: can an artificial intelligence beat these computer-like pros at a game of GeoGuessr? To answer this question, we first limit the scope to be contained within mainland United States (excluding Alaska, Hawaii, and territories) and some regions of Canada and Mexico instead of the whole world. This decision is driven by two key factors: computational constraints and visual diversity. Given our limited API credit available for collecting Street View images, we must make a strategic decision to limit our geographical focus. As such, we select a location that offers exceptional visual diversity. The region around the United States provides distinct terrains: from deserts, to the rocky mountains, and dense forest biomes—distinguishable shapes and colors that serve as potentially strong visual features for our model. Through this justification, we estimate that the United States and surrounding areas make for an ideal training environment.

In this environment, we will then construct two convolutional neural network (CNN) agents and combine their outputs to produce a single coordinate guess. The first agent, Agent 1, will accept an image as input and output a latitude and longitude prediction. The second agent, Agent 2, will accept an image as input and output a probability distribution of it being from one of the grids that we sectioned the map into. Using the two outputs, we will then combine the results to produce a single coordinate prediction. We hypothesize that by combining both agents, our result would prove to be more accurate as it would prevent either agents from overfitting to a singular guess.

This report will go over the methodologies approached, additional experiments that provided useful and interesting data, and the result of our model's final iteration.

3 Methods

3.1 Obtaining the Data

We section the United States into a 7x10 (row x column) grid and start randomizing coordinates within a bounded rectangle defined by the following coordinates: top left corner point (49.049081, -125.450687) and bottom right corner point (24.455005, -67.343249). Then, for each generated point, we call the Google Maps Static Street View API to download the image and increment our total image for that grid. The program then stops when each *valid grid*—a grid that has Google Maps coverage and is not in the ocean—has a minimum of 225 images. By the program's conclusion, we end up with roughly 35,000 images. Each of them is then named "[latitude],[longitude].jpg" where *latitude* and *longitude* is respectively labeled by the coordinates.

Also, it is important to note that the images called by the API *does* have a copyright watermark. This watermark should be left unedited, as it is included in each round of GeoGuessr. Instead of potentially skewing the result, it would actually be helpful if the model is able to extract the features of the watermark.

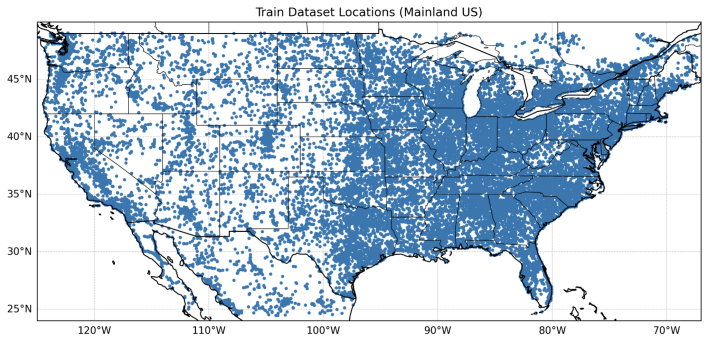


Figure 1: The dataset's data points distribution

3.2 Preparing the Dataset

Our goal is to split the dataset into three parts: training, validation, and testing. We first generate a CSV file containing the file names of all images with three columns: image_path, lat (short for latitude), lon (short for longitude). Then, we shuffle the CSV entries and split it into three subsets where we set aside 80% of the images for training, 10% for validation, and 10% for testing. This will serve as the dataset for the entire project.

3.3 Dataset Pre-processing

We need to pre-process the images before using them for training or evaluation. We first resize the image dimensions to 224x224. The reason being that the models we chose as our backbone pre-trained on images of that size. For training, we added additional transformations that slightly adjusts the brightness and contrast of the images to simulate random weather and camera quality conditions. The pixel values of each image are finally normalized to $[0, 1]$ and converted to tensors before being used by the model.

3.4 Agent 1 Model

Agent 1 is a CNN that directly predicts geographic latitude and longitude coordinates from Street View images. We chose EfficientNet-B0 as the backbone model for this agent. This backbone balances efficiency and accuracy while demonstrating stable performance when processing images [2]. For the regression head, it is a simple three-layer Multilayer Perceptron (MLP) that reduces the original 1280-dimensional features from EfficientNet-B0 to 512, then to 256, and finally to 2. Each hidden layer uses the ReLU as the activation function. This model outputs latitude and longitude normalized to the range $[-1, 1]$.

Due to the differing scales of latitude and longitude (where a degree change in either latitude or longitude corresponds to different distances), training the model directly on coordinates data can lead to suboptimal stability and accuracy. Therefore, we normalize the maximum and minimum latitude and longitude values from the training dataset to the range $[-1, 1]$. This ensures consistent gradients across dimensions, preventing our agent from developing bias towards any single dimension. For training and evaluation, we reverse the normalization to obtain accurate latitude and longitude coordinates.

3.5 Agent 1 Training

We set the distance between the predicted and correct locations as the loss function for training. To calculate that distance, we used the Haversine Formula: it calculates the distance between two coordinate points, having taken the curvature of the Earth into account [3]. The formula can be denoted as the following,

Given two points (ϕ_1, λ_1) and (ϕ_2, λ_2) (in radians), we first compute

$$a = \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right).$$

Then the Haversine distance is the function below where $R = 6371\text{km}$

$$d = 2R\arcsin\left(\sqrt{a}\right).$$

We trained the agent for 120 epochs with the Adam optimizer and a learning rate of 0.0001, a weight decay of 0.0001, and a batch size of 64. For early stopping, we set patience to 20 to prevent overfitting and also give the model more time to hit a better minima. As a result, we obtain the best checkpoint at epoch 31, with a total of 51 training epochs.

3.6 Agent 2's Model

As the goal of Agent 2 is to narrow the search down to a region, we constructed it using a CNN model to extract features of an input image. It then outputs a probability distribution of the image belonging to one of the 7x7 grids we divided the map into. To do this, we used ResNet-50—a model known for its strong feature extraction capabilites—as our backbone [4]. We then replace the final classification layer with a very lightweight head: a dropout layer followed by a linear projection onto the 49 grid classes.

3.7 Training Agent 2

While we tried out a few loss functions, the function with the highest accuracy is cross entropy loss: a function that rewards only correct classifications. The function can be denoted as the following:

$$L(\Theta) = -\sum_{i=1}^{N} y_i \cdot logSM(f(x_i; \Theta))$$

where N is the total number of classes, "y_i is a one-hot encoded representation of the true label of the i^{th} data point, and $SM(f(x_i; \Theta))$ is a vector of the same size as y_i, which contains the predicted probabilities of the i^{th} data point belonging to each class [5]."

During training, we followed a two-phase fine-tuning strategy with a batch size of 64. The first phase used an Adam optimizer for an accelerated learning curve and a learning rate of 0.0003 with a weight decay of 0.01. The second phase, which began at epoch 15, used a Stochastic Gradient Descent (SGD) optimizer for fine tuning and a learning rate of 0.001 with a weight decay of 0.0005. In total, we trained the agent up to epoch 31 as the training loss began to plateau around epoch 27—meaning that training-wise (with this approach), this was the minimum obtainable loss.

3.8 Combining the Agents

Both agents prioritize different strengths that were important to take into account. We want to combine agents 1 and 2 in a way that leverages the spacial precision of agent 1 with the distributional certainty of agent 2. The general method for this was to start at the coordinate achieved by agent 1, look at surrounding grids within some reasonable radius, and pick the grid with the highest probability. We accomplished this through an objective function that picks the grid with the highest score and returns its center coordinates. Our score for each grid was calculated through two terms: likelihood term

and probability term. Grids that had a smaller haversine distance to the coordinate picked by agent 1 get a higher likelihood term. We use a radial basis function to compute this:

$$L(g) = e^{-\frac{d(g)^2}{2\sigma^2}}$$

where $d(g)$ is the haversine distance between agent 1's coordinate and the grid g. The term σ determines how quickly the likelihood term decays as grids move away from the coordinate. If $\sigma = n$km, the likelihood term will decay at a much higher rate once grids are further than $n$km from the coordinate. Probabilities among grids $P(g)$ comes directly from agent 2's output. For computing the score, we normalize each term then sum them. Since probability terms are already normalized, we only normalize the likelihood term.

$$\text{score}(g) = w\frac{L(g)}{\sum_g L(g)} + (1-w)p(g)$$

The term w is a term between 0 and 1 that weighs probabilities and likelihood terms. A higher w favors the likelihood term from agent 1 whereas a lower w favors the probability term from agent 2.

4 Experiments

4.1 Agent 1: EfficientNet-B0 vs ResNet50

Although ResNet50 offers larger feature sizes, during training we found it to be very slow and not accurate enough. In contrast, EfficientNet-B0 gives us a lower mean error even with its smaller feature sizes. Under the same conditions, the mean error using EfficientNet-B0 was 768 km, while the error from ResNet-50 was 891 km. More importantly, its training speed reaches 12 it/s, twice that of ResNet50's 5 it/s. Therefore, we picked EfficientNet-B0 as agent 1's backbone for all subsequent training and experiments.

4.2 Agent 1: Compare Data Pre-processing Strategies

We experimented with several data pre-processing strategies, including slight rotations, horizontal flipping, and random cropping. These transformations were intended to improve the accuracy of Agent1's results, but instead reduced accuracy. Through our discussions, this is likely caused by cropping as it resulted in the loss of important features. In addition to this, rotating and flipping the images added unnecessary computations as our models were not expected to extract features where rotations and direction mattered (like texts or signage). So from these experiments, we shifted to a more conservative approach. We retained only brightness and contrast dithering. This process enhances the model's adaptability to weather and lighting conditions. As a result, it improves training outcomes by about 3%(20 km).

4.3 Agent1: Test number of training epochs

Initial training was set to 20 epochs with a patience value of 5, but the model did not reach a stable state. Based on the training data, there was still room for improvement. As training time increased, performance should continue to improve. Therefore, we increased the epochs to 30 with a patience value of 10. The results improved, but since early termination had not yet been triggered, we suspected it still had room for improvement. Finally, we increased the epochs to 120 with a patience value of 20 to achieve the best training results for this model. Early stopping occurred at epoch 51, but the improvement in performance was negligible. This outcome implies that the current training results represent Agent1's best achievable performance. To further enhance results, increasing the image size could give us a slight improvement, but this would require significantly more training time.

4.4 Agent 2: Other Loss Functions

It is worth mentioning that for Agent 2, we tried a few other loss functions: TopK and grid adjacency. However, their performance did not measure up to cross entropy loss. To evaluate the agent's performance, we will be using top-1, top-3, and top-5 metrics. In other words, the percentage at which the correct grid is within top-1, top-3, and top-5 guesses.

TopK is the loss function where for a given integer k, we reward guesses that had the correct grid listed within the top k results. Initially, this loss function made sense as we were working on trying to narrow down our guesses to a region and selecting the correct grid is not as important as getting the general area right. The accuracy of the model with this loss function was quite low for top-1, top-3, and top-5 guesses. The accuracy was 14.07%, 30.28%, and 44.37% respectively. In hindsight, the reasoning is correct that we want to narrow down the search to a general area, but we overlooked a very simple fact: a grid is already a form of *general area*. This meant that we were essentially just enlarged the scope of our search—hurting the overall accuracy.

Grid adjacency is when we fully reward the correct grid guess and partial rewards for grids adjacent to it. The hope was that it would look to be a smoother heatmap. The accuracy for top-1, top-3, and top-5 are 7.4%, 14.2%, and 19% respectively. The accuracy here is quite low. Randomly guessing a 7x7 grid is a $\frac{1}{49}$ chance or 2.04% — meaning that it was only performing 3.5x better than randomly choosing a grid in contrary to TopK's 7x. A potential issue with this loss function is that some images can be equally as likely to belong to a grid that aren't in the proximity of one another. An example of this is the common mistake that GeoGuessr pros make: mistaking Northwest US with Northeast US and vice versa. Within those regions, due to the climate similarity, the overall environments are hard to differentiate. As such, this was a loss function that detracted from the agent's desired behavior.

4.5 Combining Agents

Since agent 1 had error usually under 500-800km, for computing likelihood terms we set $\sigma = 800$km. We found that when σ was smaller, such as $\sigma = 500$, not enough grids were taken into account for their probability values. As a result, some grids within 500-800km that had high probabilities were ignored, since the likelihood term had decayed so much by that

distance. When σ was larger, we were looking at too many grids, and there were more cases where grids far from agent 1's coordinate were chosen because it was agent 2's top-1 probability (which we saw is often incorrect). So, with $\sigma = 800$km, we are mainly looking at probabilities of grids within this 800km radius. For the score function, we played around with different weightings for likelihood and probability terms and concluded that agent 1 had a higher accuracy, so weighing it heavier was more effective. In fact, when the terms were evenly weighted, we actually saw that error increased by 3% from agent 1. So, we ultimately set $w = 0.7$, in favor of agent 1.

5 Results

5.1 Agent 1: Overall Result

Agent 1's mean error is $\approx$ 720 km, with a median error of $\approx$ 545 km. The chart in Figure **??** shows that the model performs poorly at certain edge cases.

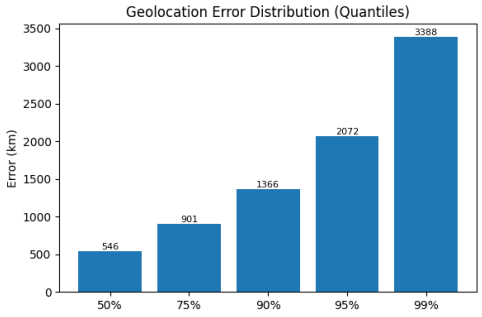


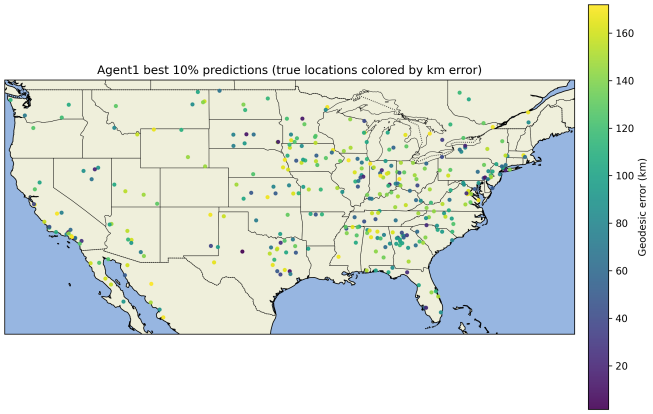
Figure 2: Geolocation Error Distribution

5.2 Agent 1: Best Predictions

The top 10% of prediction results are concentrated in the Midwest region of the United States. These regions feature simple and consistent landscapes, such as plains, farmland, and straight highways. Because similar street scenes cluster in these adjacent states, the model is able to recognize them and predict the correct coordinates at a much higher rate. Additionally, our training dataset shows higher coverage in this region than in the western area, potentially creating bias in the model. The unbalanced training data caused the model to skew towards regions with more data points and reduces its confidence in the western region.

5.3 Agent 1: Worst Predictions

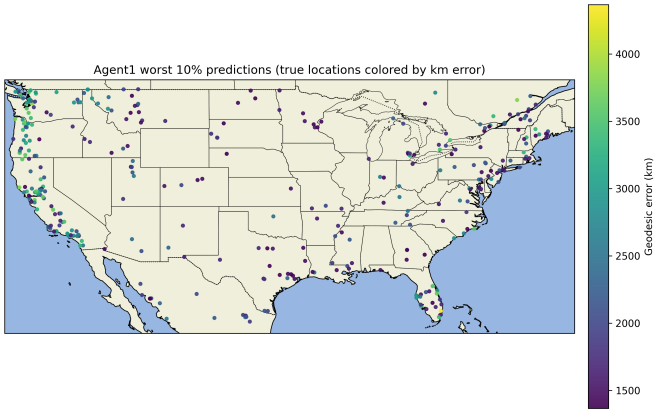
The worst 10% of prediction results typically originate from coastal regions, particularly the West Coast, East Coast, and Florida. The model occasionally confuses two visually similar yet geographically distant areas, such as California and Florida, or the forests of the Northwest and Northeast regions. Additionally, a small percentage of images do not have any features for Agent 1 to extract from. While these edge cases



[!h]

Figure 3: Best Results Agent 1

are relatively rare, when the model makes an incorrect prediction, the error often exceeds 2000 km, significantly skewing the overall accuracy.



[!h]

Figure 4: Worst Results Agent 1

5.4 Agent 2: Performance

Agent 2's final accuracy using cross-entropy loss for top-1, top-3, and top-5 is 19.44%, 43.62%, and 60.12% respectively. That's almost a 10x improvement from random guess and $\tilde{5}$% higher than the TopK loss function. To see an example of the model's out, refer to Figure 6 where the probability distribution is more accurate to our expectation.

Figure 5: Input Image

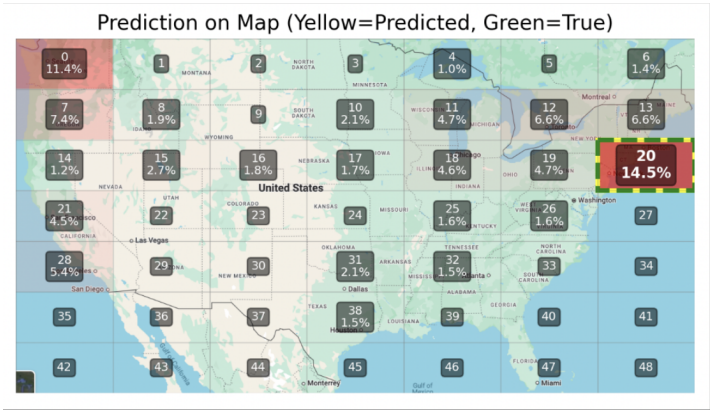


Figure 6: Agent 2's Output from the Input Image

5.5 Combining Agents

Our results from combining agents were quite close to the results from agent 1, with only a 1.8 % improvement in loss. This makes sense, as the coordinate achieved from combining agents is often very close to the coordinate from agent 1—sometimes just a grid or two away. While the confidence values from agent 2 can pull the coordinates from Agent 1 towards the correct location, it has also—at times—moved it further away. Since the average distance between points was relatively low, the average error of combined agents and agent 1 leveled out to a very similar value. Below shows the error distributions of combined agents and agent 1, and we can see that they are effectively the same.

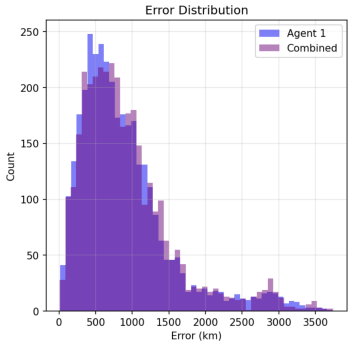


Figure 7: Combined Agents vs. Agent 1

In order to significantly improve combined agents from agent 1, our combined agents method must be able to recognize when agent 1 picks a coordinate in the wrong region, or when error is greater than 1000 km. This could be accomplished through greater agent 2 accuracy and more even weight values. Since probability is weighted lower than likelihoods, even if agent 2 has a strong probability in the correct grid, if that grid is far from agent 1's coordinate, it will almost never be picked. We could also adjust the way the score is computed by perhaps adding another agent. This agent would detect signs when agent 1 is likely in the wrong region. For example, maybe when agent 1 has high error, it is also true that there is a high probability distribution far from agent 1's coordinate. If an agent can recognize these cases, then it can decide when to weigh probabilities vs likelihoods more.

5.6 Analyzing the Features Extracted

Let's take a look at a feature visualizations in our CNN. Figure 8 shows the original input image on the top left and feature maps from convolutional layers 1-4, with multiple kernel outputs (K0 - K10) from the first layer, and the final Class Activation Map (CAM)—a CAM is a map that shows which regions of the input contributed most to the model's output: the brighter the color, the higher the focus.

The input image (in the top left corner) contains a clear sky, a defined horizon, and two vehicles next to a parking area. Examining the feature maps, we can see that most of the kernels (some are fainter than others) are able to detect horizontal edges from the horizon. K1 and K2 especially are able to see colors and contrasts quite clearly—emphasizing the the horizon. K0-K3, K5, and K9-K10 are able to detect the vehicles particularly well.

As for the deeper layers, the feature maps are quite difficult to interpret. Layer 2 shows some color separation between the sky and the ground. Layer 3 isn't really seeing much except for some faint activations around the horizon and the top of the image. Layer 4 displays clear activation on the ground, with almost entirely white pixels focused around the gray sedan. Throughout the layers, it can be concluded that in this example, the model was able to extract the vehicles.

This is reflected in the CAM as well, where it is even more apparent that the model focused on regions centered around the gray sedan. The finding was quite unexpected as we hypothesized that the model would be able to extract features relating to the terrains and biome. In this particular example, it used the car to make its final decision, which is odd as geographic locations do not typically correlate with specific car models. In other words, it is possible that the model has learned unrelated correlations beyond just geographical features.

While we took a look at a specific example that may be an outlier case in this report, this pattern actually persists across other tests: the CNN identifies the sky (if any), extract the horizon, and zoom in on large objects. In images where there *are* more noises like denser forest areas in the Northern coasts of the United States, the model is not able to find the sky, but instead, they identify large shrubs in areas that a sky *would* typically be (a typical sky region in this case would be similar to the input image in Figure 8 where the horizon meets about mid-way through the height of the image) in addition to large objects like cars and houses. For these cases, they align closer to our hypothesis as it does use the geographical features to finalize its decision—albeit due to the *absence of sky* more so than specific vegetation and colors.

The unexpected behavior—particularly on the model's tendency to identify large objects like cars that have no correlation to the location—requires further investigating to fully understand if the model is learning features that even we don't know about or if it is relying on coincidental correlations in the training dataset.

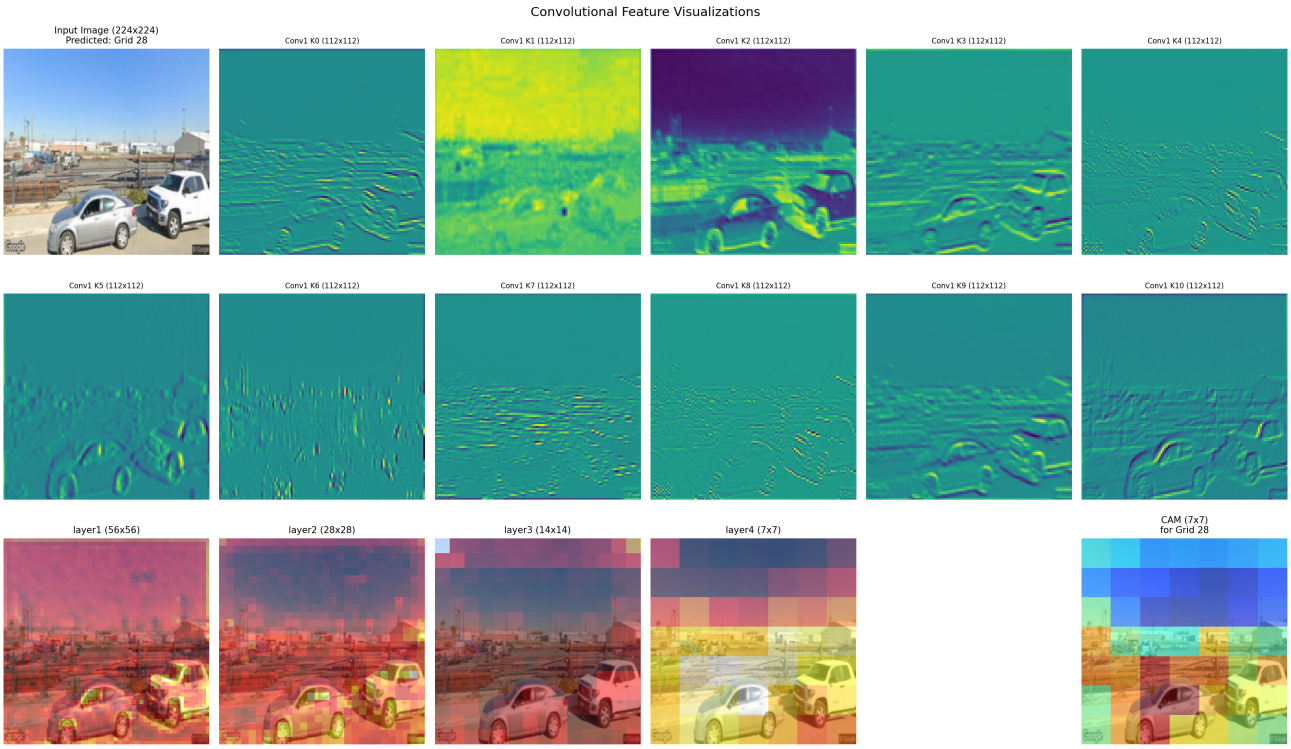


Figure 8: Feature Map of a Correct Grid Guess

6 Contributions

Jin wrote data preparation scripts, collected data/graphs for Agent-1, and experimented with Agent-1. Jin and Dave then equally contributed to Agent-1's development. Lojanarungsiri provided the dataset, gathered data/graphs for Agent-2, and experimented with Agent-2. Euston created the initial framework for Agent-2, combined the agents, and gathered the combined agent's data. Except for Dave, the team equally contributed to the presentation, final report, and code submission.

7 Conclusion and Discussions

Our project explored two complementary approaches to playing GeoGuessr: direct coordinate regression and grid-based confidence classification. Individually, each agent produced moderately high accuracies that perform better than an average player (although not better than pros). Professional GeoGuessr players often use signs, license plates, and vegetation to make their guess. Neither agents, however, learned these heuristics. In the future, our model could be improved by adding more agents to identify vegetation and license plates. Additionally, a larger and more detailed dataset could improve the function of both agents. As we discussed earlier in our report, there is an uneven distribution of data points, with data more concentrated along the eastern half of the US. We saw that these areas had lower error, so increasing the size of our dataset should improve our model's accuracy.

A feature that we did not account for in GeoGuessr is that players are able to move and look around. Our dataset only included one image frame per location. Without being restricted by API credit limitations, adding in several images per location or a panorama image would likely increase performance for both models as there would be more features to extract from.

Lastly, we also saw that our combined agents only modestly improved from agent 1. It may be helpful to train another smaller model that can adjust weights of agents, especially when the agents disagree significantly. The accuracies that we did accomplish with our models shows us that creating an accurate GeoGuessr is definitely feasible. With sufficient data and richer feature signals, there should be no barrier to extending this architecture to cover the map globally.

References

[1] Plonk It, "Beginner's Guide to GeoGuessr," 2025. Accessed: Dec. 8, 2025. URL: https://www.plonkit.net/beginners-guide.

[2] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," arXiv.org, 2019. Accessed: Dec. 9, 2025. URL: https://arxiv.org/abs/1905.11946?utm_source=chatgpt.com.

[3] S. Ferraris, "Haversine's distance mathematics – A Geospatial Data Science Blog," Github.io, Jan. 13, 2024. Accessed: Dec. 9, 2025. URL: https://sebastianof.github.io/GeoDsBlog/posts/gds-2024-01-10-haversine-dist/?utm_source=chatgpt.com.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[5] R. Venkat, "Machine Learning," in *CS4100 Lecture Notes*, 2025. Accessed: Dec. 9, 2025. URL: https://rajagopalvenkat.com/teaching/resources/AI/ch7.html

A AI-Assisted Writing Clarifications

This appendix documents all instances where AI was used to improve grammar and writing clarity, as required by the course policy. It is important to note, however, that while AI assistance was used for improving writing clarity, there were no direct copy-pasting and every sentence outputted by the AI was moderately, manually edited and reviewed before they made it to the report.

A.1 Abstract

Revised: Popular online game GeoGuessr is a game where players randomly spawn in an unknown location on Google Maps Street View and must determine their position using only visual context clues.
Original: Popular online game GeoGuessr is a game where players randomly spawn somewhere on Google Maps Streetview, and the goal of the game is to use context clues around the location to figure out where they are in the world.

A.2 Introduction

Revised: Image-based geolocation is a challenging computer vision problem. Unlike traditional object-identifying problems, geolocation depends on capturing more obscure features like vegetation density that can vary significantly across regions.
Original: Geolocation is a difficult task to tackle. It requires utmost diligence and a good eye to identify weird features like the color of the dirt and types of vegetation.

Revised: Despite the challenging nature of geolocation, professional GeoGuessr players managed to develop a sort of heuristic, or "metas," that allow them to pinpoint locations from a single image frame with remarkable accuracy. They often rely on similar obscure features such as the color of the dirt, the triangulation of the sun-cast shadows, and even the camera quality of Street View images. Their inhuman capabilities have demonstrated that—with sufficient pattern recognition—it is possible to accurately pinpoint a location using a single image.
Original: Professional players of this game have learned to identify locations by the color of the dirt, the triangulation of the sun-cast shadows, and even the copyright mark on the Google Maps Street View images. In fact, they were able to pinpoint locations without even having to move around, using only a single image and sometimes only seeing the image for 0.1 seconds.

Revised: Given our limited API credit available for collecting Street View images, we must make a strategic decision to limit our geographical focus. As such, we select a location that offers exceptional visual diversity. The region around the United States provides distinct terrains: from deserts, to the rocky mountains, and dense forest biomes—distinguishable shapes and colors that serve as potentially strong visual features for our model.
Original: Because of our limited API credits, we had to make a smart decision in choosing a location to practically and realistically obtain the dataset and train the agent in. With that being said, we chose those United States and surrounding regions due to its great diversity in biomes and terrains. These distinguishable characteristics of the area is not only difficult to find in other countries, but also serve as exceptional features for our model's training.

A.3 Analyzing the Features Extracted

Revised: Let's take a look at a feature visualizations in our CNN. Figure 8 shows the original input image on the top left and feature maps from convolutional layers 1-4, with multiple kernel outputs (K0 - K10) from the first layer, and the final Class Activation Map (CAM).
Original: Let's take a look at a feature map. In Figure 8, the original input image is listed on the top left, and each visualizations in the first and second rows shows different convolutions feature maps.