Geoguessr AI

Lawrie Feng - 2331513 & Max Fleurent - 2331173

Robert Vincent

420-LCW-MS Programming Techniques and Applications section 00002

Marianopolis College

2025 - 05 - 07

Mini-Manual:

Background Information:

Geoguessr is an online geography game, where the goal is to view a random google map street view coverage and guess where that place might be on a world map. For our project, we chose to use the NMPZ standard rule set. This means you cannot move around the street or look around: You get one image and that's it. Our Geoguessr AI is a program that takes in a user inputted Street View Image and outputs and guesses which country the image was taken in. We wanted to make this machine learning AI due to the recent popularity boom of geoguessr, and as a fun conceptual idea to make something really cool.

How to use it:

Our Youtube Tutorial: https://youtu.be/phZSOSW1HM0?si=PA9iV0QsAAYtXZ1c

1) Open the game… Obviously… (Use the website: https://openguessr.com)
    a) We use Openguessr rather than geoguessr because geoguessr cost money to play while Openguessr is just a Lite version of it and it's free for everyone
    b) Also please play in singleplayer because cheating isn't cool
2) Continue to the youtube guide to set up Google Colab as well and the necessary files for our code to work (5 minute process - all files are provided in the zip)
3) Using the Print Screen (PrtScn) or the command (win + shift + s), take a screenshot of you screen in Openguessr and save the image file somewhere
4) Drag and Drop upload the image file to Google Colab and rename it to "1" because its short and easy

5) Make sure for the input image the file path is correct and click run and voila! The AI should output a country name and you can now go back to Openguessr and guess your location on the map!

Design Guide:

Brief Introduction and Breakdown:

In this section, we will cover all the files in the zip and break down what each of them do as well as our process in making this Small Vision Transformer. Throughout we are also going to mention outside sources used and future improvements to our program.

The Main Code:

Tiny_vit.py and tiny_vit_model_geoguessr_(0.0048 - 92.88%).pth:

Tiny_vit.py is the model code we used on wkcn's tinyvit Github, the other file is the trained weights we saved throughout our epochs of training displaying its loss (0.0048) and its accuracy percentage (92.88%).

Geoguessr_AI_Final.ipynb:

This is an all inclusive code that can train and run the AI. It has about everything needed to run locally fully built in a contained environment (except for the tiny_vit code) The first code block contains the necessary pips (which allows us to keep track of which ones we have downloaded so far). Next, the import statements. It's important to have them all in one block so that you don't rerun them later on.

Next, we download our dataset from Kagglehub, a website made for downloading datasets like this. Even if we want to train on our own datasets, uploading them to Kagglehub and then downloading them again is good because it allows for flexibility when choosing which machine we run this on. Kagglehub automatically zips and unzips folders, allowing for faster file downloads if we choose to run this on google collabo or another computer. It's even smart enough to know if it has already downloaded the dataset, in which case it simply returns the directory of the download rather than redownloading it.

Then, we have our file paths. The first one is DATA_PATH, which is where we link our dataset. In this case, since we're only using the Kagglehub set, we can simply link it to that (specifically the "compressed dataset" folder inside the dataset). Next is MODEL_PATH. This is the directory where we save and load our own weights. This is important to actually keeping our training rather than starting from scratch every time we want to train or run the program.

After this we define all our classes and functions. Many of these are classes imported from PyTorch, which saves us a lot of time when programming. Neural networks work on a data class called "tensor". Tensors are a bundle of numbers that can be calculated at the same time (parallel processing). This means that many of our conventional programming techniques might not work here. This means we pretty much have to use PyTorch classes for everything. Right off the bat we see this with the "ImageFolder" class, which makes a dataset compatible with data loaders (more on that later). We also have to use "transforms" to change our image resolution to 224x224 for tiny_vit. We also need to make sure we turn all the country names into numbers. Since this model only outputs numbers (not strings) we need to assign a number to each country and save our indices for later use. We then split our dataset into training and testing subsets, and define our data loaders. The data loaders are what actually loads our dataset in batches to the AI. The base module for these is also imported from PyTorch.

The most important things we learned were the optimizations. We took at least 6 iterations to fully get this working well. The first thing we did was add weights to the countries. Some countries, like the United States, have way more images than some other countries. This means the AI can get stuck in a local minimum by constantly guessing the US. Our code accounts for the fact that some countries appear more than others and makes it so that guessing a common country is less rewarded. The second optimization we did was switching to the AdamW optimizer, since it is simply more modern and worked the best for us out of all the other optimizers we tried. Finally, we added a learning rate scheduler that tweaks the optimizer's learning rate based on how the training is going.

The training loop is fairly standard for all python AI models. The specific steps are in the file annotations, but to be honest we don't 100% understand some of this stuff as many of the optimization steps were greatly inspired through online aid and pretrained models. Loss is how we define the model's accuracy and monitor its improvement over time, but we still want to see how effective it is at actually guessing some images. So, we have it run on the testing dataset to see its tangible accuracy. This isn't the most important thing to the training, and can technically be entirely removed. However, it's still fun to see the actual improvement over time.

Current AI Limitations and errors:

Currently our AI displays a 92.88% accuracy, which doesn't give us the full picture. This is because in geoguessr when you guess a country, you also have to determine where exactly it is in that country. While the AI is very accurate when it comes to guessing the country, it can't tell you the exact location inside the country. This means that its in-game accuracy is lower than 92%. On the other hand, this accuracy does not account for close calls. When we tested our code, we saw that it was always close to the correct answer (geographically speaking). It might guess Switzerland instead of Italy or etc. This accuracy value does not reward proximity, so this effect contributes to the in-game accuracy being higher than predicted.

Future Improvements:

Geoguessr_csv_dataset_gen.ipynb (adding more data):

This code is what we are currently working on for future improvements. It uses the same concept and techniques used in our main code, but adds the ability to create our own datasets with geopandas. Geopandas contain every world country as a 2D shape, which allows us to know what country a coordinate belongs to.

The website we used to obtain raw street view coordinates is called MapGenerator, a tool made by the geoguessr community to generate random coordinates from random street views around the world. We then put all the coordinates inside of a CSV file and fed it to the Google Street View API, which returned images corresponding to the provided coordinate list. We then can save these images inside another local file and start expanding on our dataset. Furthermore, we can’t just have a folder full of random images without knowing which country they are from. To fix this, we used geopandas to read each coordinate. Before a picture is downloaded, we use geopandas to see which country the coordinates are in. When the image is downloaded, it is saved to the folder of its corresponding country. Unfortunately, we never got the time to train on our massive dataset since it took too long to train our model. Thus instead of making a better dataset we decided to use one online (Kaggle 50k dataset). However, the model is training as of writing this, and we could share improved results at a later date.

Furthermore, a major improvement we are hypothesising on is to split the world into different cells (similar to the battleship board game) where you can predict more precise regions by calling A4 (example). We want to do this because right now our programming only tells us what country we are in but not where inside the country.
This is especially harsh for the United States, which is so vast that getting the country right does not always translate to more points. So if we can label the world map into different grids then it would be even better as we can pinpoint not only the exact country, but also the location (CNN AI Research Paper & County geoguessr Research Paper)

CSV_Code and Geopanda_dependencies:
CSV_Code is just a file where we put all our csv coordinate data. Geopanda_dependencies contain all world countries as geometric shapes. It is necessary to the geopandas module, which allows us to know the sovereignty of any coordinate.

Online Sources:
Our Inspiration: https://www.youtube.com/watch?v=7bKLl1NoKbQ
US County Geoguessr Research Paper: https://nirvan66.github.io/geoguessr.html
Geoguessr CNN AI Research Paper: https://arxiv.org/pdf/1602.05314
KaggleDataset: https://www.kaggle.com/datasets/ubitquitin/geolocation-geoguessr-images-50k
wkcn’s TinyVit Model: https://github.com/wkcn/TinyViT

PlonkIt CSV: https://map-generator-nsj.vercel.app

Pytorch Documentation: https://docs.pytorch.org/docs/stable/index.html

*To be more specific when we need help with defining the model so torch.nn, you can just search up torch.nn on any search engine and the specific code documents will appear. This is just a link to the whole documentation or else there will be too many links. (eg. Documentation for devices, PyTorch built in Optimizers, Forward backward pass etc.)