Image Classification for GeoGuessr Application

Kevin Ramos, Bruce Kirschbaum, and Meredith Westrich

Abstract

GeoGuessr is a web-based geographic discovery game. A player is dropped at a random location in Google Street View and must pinpoint where they are based solely on the image clues provided. For our project, we are using image classification to identify where in the United States an image was taken. We divided the United States into eight distinct sections based on latitude and longitude, considering international borders and geographical similarities. We are using data from Google Street View to train an image classification system based on these eight sections. In order to collect this data, we used a Street View API and connected it to a Python program. For the image classification part of the project we collaborated on a Jupyter Notebook in Colab. We used a convolutional neural network (CNN) to classify these images, and ended up with a result of around 60% accuracy with our validation training set. This was a fascinating introduction to image classification.

Introduction

Image classification is an important field in the realm of machine learning. From self-driving cars, to improving search results, teaching a computer to "see" an image the same way as a human is something that computer scientists have been perfecting for decades. The most efficient way to do this is with a convolutional neural network (CNN). CNN's extract high levels of information about an image's content, inferring objects by extracting features like textures or colors.

For our project, we have trained a CNN using sets of photos from Google Street View in order to classify the image into one of eight distinct sections of the United States. We divided the country based on latitude and longitude into rectangular sections, considering geographic similarity and international boundaries. The goal of our project is to have an algorithm that can classify an image from Google Street View into one of the eight sections in the United States with at least 25% accuracy, double the random chance.

Using Python, we were able to create an algorithm that predicted the location with a validation loss of over 60% accuracy, much more than our initial goal.

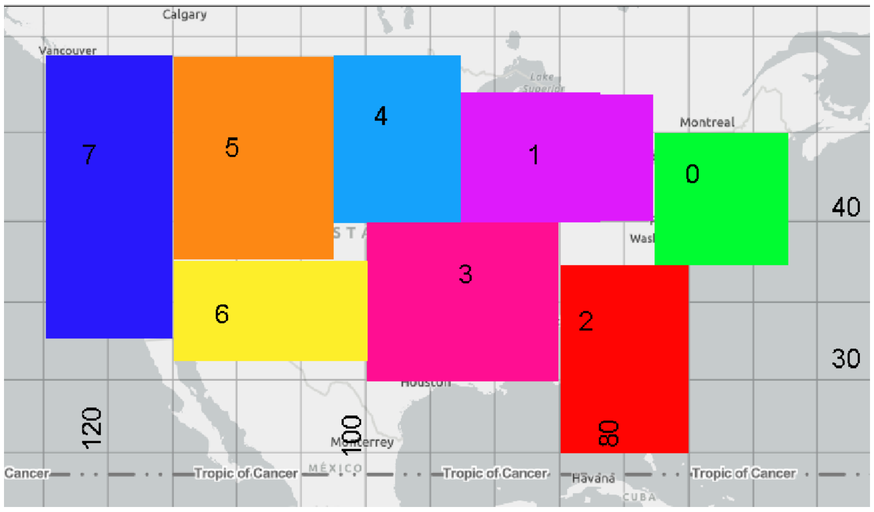
Related Work

Our initial instinct was to look for other AI algorithms that had mastered Geoguesser, and work off of their code. One example of this is a Youtube video labeled AI Learns to Play GeoGuessr by adumb. This video was not ultimately helpful in our solution, but gave us a better idea of how to structure a project like this. For their AI, adumb focuses on the United States, instead of the world as a whole. In addition, they divide the US into squares 340 miles tall and 360 miles wide. We also chose to divide the country into squares, however we made some adjustments. Our squares are not of equal size to each other, and are grouped together on the basis of geographic and environmental similarities, as opposed to linear sectioning. This video was instrumental in creating our methodology, but provided us with little information regarding the actual coding or data.

Another piece of related work that assisted in our project was another Jupyter Notebook in Colab that accomplishes something similar to the classification problem we were attempting, a binary classification example classifying cats versus dogs. The example used a CNN to classify it's images, and used numpy and matplot to plot its iterations of learning. We used this code as a framework for our project, making a few notable alterations to the code. We kept things like image size and the code for the network consistent, but in order to adapt the algorithm for a categorical classification problem, we needed to change the size of our datasets, as well as the type of regression algorithm used by the program.

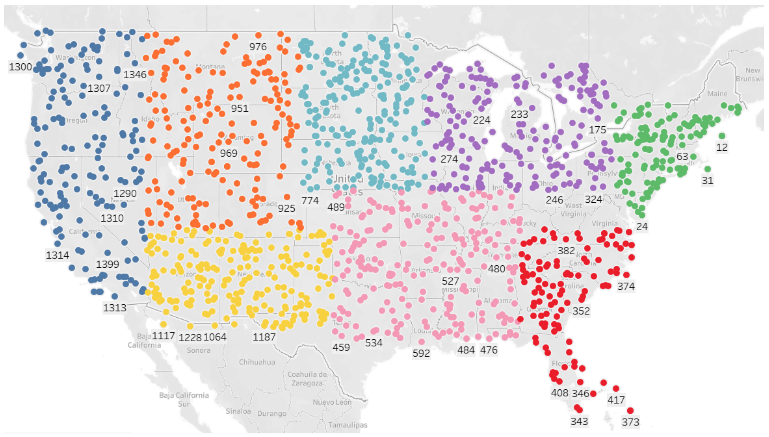
Data

Our training dataset consists of nearly 1400 images collected through Google Street View. In order to collect our data, we divided the United States into eight different rectangles of latitude and longitude, grouping areas based on common flora, agricultural, and architectural features. In order to obtain "rectangular" sections of latitude and longitude, we acknowledge two errors in our data.



section	latitude	longitude
0	37.5-45N	67.5-77.5W
1	40-47.5N	77.5-92.5W
2	25-37.5N	75-85W
3	30-49N	85-100W
4	40-49N	92.5-102.5W
5	37.5-49N	102.5-115W
6	31-37.5N	100-115W
7	33-49M	115-125W

The first error is that not all of the contiguous United States were included in our dataset. In order to primarily exclude Canada and Mexico from our data while still taking reasonably sized rectangles, we had to exclude some land mass of the US, mainly in Maine, West Virginia, Virginia, Kentucky, and Texas. This leads to our second error, that it was impossible to exclude neighboring countries entirely from our dataset. Out of the over 1300 images, approximately 50 of them are of neighboring countries.



In order to collect data, our group made a Python script that scraped images through Street View Static API. Our python script consisted of eight requests to the API, one for each section dependent on the latitude and longitude. We repeated this process 222 times per section, resulting in our dataset. We did not receive exactly 222 images per section, due to the fact that it is impossible to section off locations in the US without including some latitude and longitude that is without Google Streetview images, such as oceans or lakes. We did not consider this to be an issue, however, as the disparity results in a similar density of data per section.

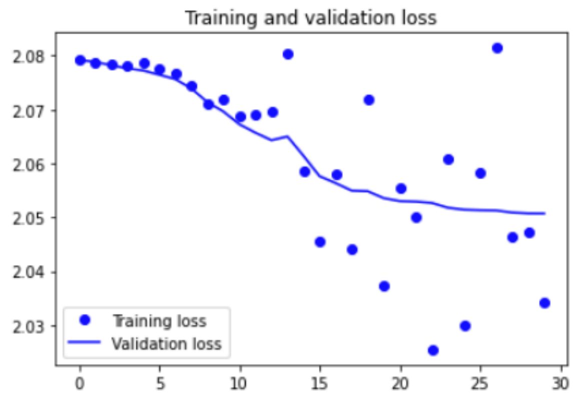
We repeated this process for our validation dataset, however, we shrunk the size of the dataset. We preprocessed the data as part of the final program, scaling every image to a 150x150 ratio.

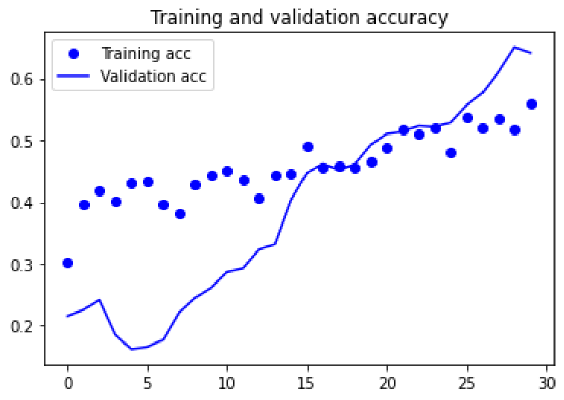
Methods

We used a CNN for this project, modeling our project off of the cats and dogs example mentioned previously. Using Python, we imported the Keras module and used its API to create a CNN. Our CNN underwent many iterations, resulting in various issues in our results. First, we didn't scale the image before running through, resulting in wildly skewed data. Next, we didn't add enough layers to our CNN. This resulted in nearly 100 million trainable parameters for the model, which was too many. We adapted and changed our CNN based on the results we were getting from the experiments. When we didn't have enough layers, our model was incredibly overfitted, resulting in accuracies of 90%, but with incredibly high losses. After adding more layers, we reached between 3 million and 4 million trainable parameters, similar to the example.

Experiments

After collecting our data and figuring out how to use Python's Keras framework to implement our convolutional neural network, we were able to run the experiment. This was the resulting plot after running through thirty epochs:





Clearly, the convolutional neural network had worked. The top plot shows that, throughout the thirty epochs, our AI was able to increase in accuracy in both training and validation categories. It ended with a 64% and 56% accuracy in validation and training respectively. The plot below shows that the loss was on a slow but steady decline throughout the thirty epochs. This is incredibly significant for the small size of our dataset.

We ran this experiment multiple times, resulting in similar results after many attempts at gaining uniform results in our code. We think the size of our dataset is the biggest reason for the variation of results we had when we ran the algorithm multiple times. If we had more time and more resources, we would have included more images in both our training and validation datasets. In the end, the results still greatly exceeded our expectations of 25% accuracy, as the AI was able to correctly predict in which region an image was taken with over 50% accuracy.

Key Takeaways

Overall, we feel that working with a CNN was a valuable experience, and was the best method for our project. However, there are components of our project that we would adapt if we were to continue with this project. For our project, considering both the timeframe and the scale of the class, our dataset is ultimately very small. We would love to be able to run our simulation with a bigger dataset and see our results. Currently we believe our model is relatively overfitted to the data we have collected, even though our validation set shows good results.

One part of our project that we would like to expand on is the way that it can be adaptable to any input dataset. For example, the same code could be applied to any country in the world, as long as it was split into eight sections. You could feed in data from eight major cities in the US, or eight different countries, or eight different national parks. Given the positive results from our dataset, our simulation could differentiate with 60% accuracy between any dataset containing eight different sectors of images.

This project was an incredible introduction into the world of CNNs and machine learning and allowed us to not only complete a goal, but create a tool that could be applied to many different fields. The field of image classification is bigger than any of us could have imagined and we are proud of the work we created and the contributions we made towards the field.