Distributed Deep Learning Network to Predict Location of Google Street View Data

Elijah Berberette
EECS
University of Tennessee
Knoxville, United States
elidberb@vols.utk.edu

Hunter Price
EECS
University of Tennessee
Knoxville, United States
hprice7@vols.utk.edu

***Abstract*—In this work, we attempt to create a deep Recursive Convolutional Neural Network that can play the popular online game Geoguessr [2]. We develop a regression model that given a pan of a particular location in the United States, can predict the Latitude and Longitude of the images. We build on previous work by using regression instead of classification. After analysis we have found that our model on average predicted locations approximately 260 miles closer than previous work and approximately 310 miles closer than a random guess.**

I. Introduction

The online game Geoguessr [2] has taken the world by storm. It consists of showing a user the Google Street View [5] of a random location on Earth then having the user predict the location. It will then show the user how far away the prediction was. In this work, we created a scalable model that will autonomously play this game. In previous work [1], the authors made a grid of their training data and trained the model to classify which grid the images resided in. This is not scalable as the more data for more countries the model is trained on the number of grids to predict will drastically increase; it also does not penalize a bad prediction based on how far away it was. We implemented a regression model that solves both of these issues: with more data the size of our predictions will not grow, and the model is now penalized for predictions far away from the actual location.

II. Methodology

The following content shows the process used to develop our model for determine the location of the given images.

A. Data Collection

We collected each of the images found in our dataset using Google Cloud Platform's Street View Static API. We improved upon previous work's [1] data scraping techniques by implementing the core algorithms in Apache Spark. This allows the process to be more scalable when we attempt to collect more data for more countries. We stored these images into a consistent directory structure with the metadata included in the filenames. We then tarred the directory and stored the tarball in our google drive.

B. Preprocessing

After collecting all of the images, we performed preprocessing on the data. This consisted of first parsing the directory structure created from the data collection and outputting each of the images file locations and features to a CSV file. From here, we stored the CSV file into a Pandas DataFrame, then separated the images into groups of 3 separate directions (0°, 90°, and 180°) at the given latitude and longitude values.

After splitting the data appropriately, we stored each of the images into a numpy array for the input. We also stored their latitude and longitude coordinates and grid values into two separate NumPy arrays. Lastly, we used min-max scaling to normalize the data, shuffled the data, and separated the testing data from validation data using an 0.8/0.2 split.

C. Model Architecture

For our model, we used inspiration from previous work [1]. Each sample of data our features consisted of 3 images combined in an array of shape (3, width, height, channels), and the labels were an array of shape (2,) holding the latitude and longitude values. Our model consisted of a Resnet model with pre-trained weights from imagenet to get the reduced features from each image. We then fed each of these feature vectors into a many to one LSTM. This output was fed into more dense and dropout layers. The final output dense layer had an output size of 2 and a linear activation function.

D. Training the Model

For training the model, we created a custom training loop such that we have more control over what happens in each epoch. This included room for adding a custom loss function that integrates geospatial distances which will be expanded upon in the Future Work section. Additionally, we opted to use a learning rate scheduler in an effort to more fine tune the model as the number of training epochs increase. We trained the model for 250 epochs at an initial learning rate of 2.5e-05 and a decay rate of 0.9.

E. Scalability

To make our project more scalable and distributable we implemented the core datascraping algorithms in Apache Spark. For our model, we used Tensorflow's distributive capabilities.

When training we used a Mirrored Strategy due to our limited resources; however, it has the capability to use more distributive strategies.

III. Results

In Figure 1, we can see the models average haversine distance between the predicted and actual location for the training and validation data. We see that the average distance error of the training data is 681.59 miles and increases on the validation data by about 200 miles to 884.15 miles. This increase is expected as it has not seen any of the validation data while training.

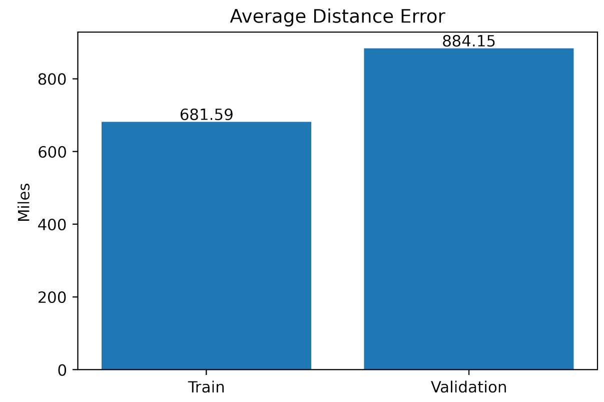

Fig. 1. Average Distance Between Predicted And Actual Locations

In Figure 2, we can see a comparison between the trained models prediction on the validation data and a random point prediction within the United States. Our trained model shows a vast improvement over the random prediction by over approximately 320 miles. This shows that our model learned and made better predictions based off the content of the images.

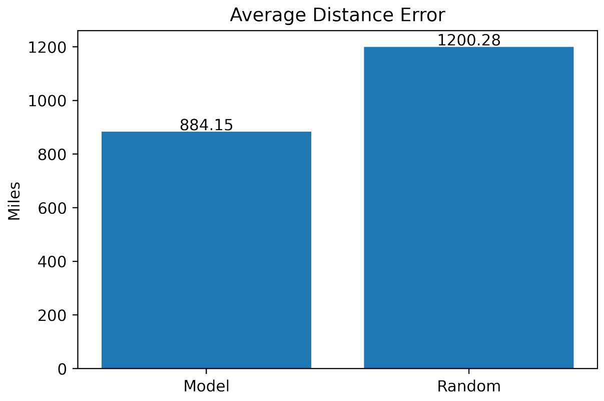

Fig. 2. Comparison Between Model And Random Predictions

Lastly, in Figure 3 we show the loss curve for the training of our model. Originally, the model starts at a very high loss due to the starting guess being between -1 to 1 for both the latitude and longitude. This is because all of the latitude and longitude coordinates within the United States are much higher than the initial guess; however, the model recognized this large difference quickly and began guessing more accurate numbers around the 10th epoch. This created the exponential drop shown in the loss curve.

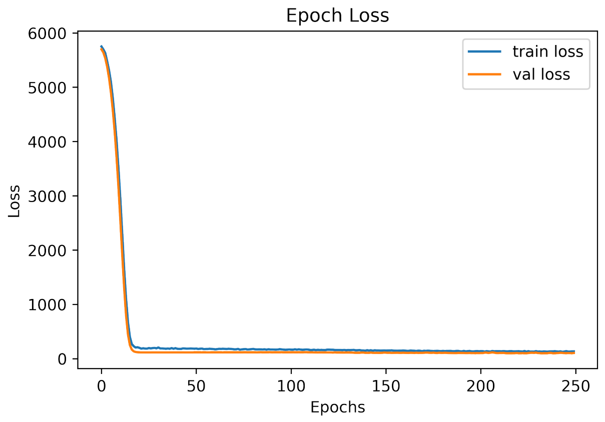

Fig. 3. Training and Validation Loss

IV. Conclusion and Future Work

In conclusion, we were satisfied with the outcome of the model with the given cost restraints for collecting the data and the computational resources available. Overall, we were able to improve the previous model used for reference by approximately 275 miles. These results showed us that regression is viable improvement upon previous work; to further improve the model a custom loss function can be used in combination with a deeper network architecture and more data.

In the future, we intend to integrate geospacial distance metrics, such as the Haversine Formula, into the loss function. We also have intentions to expand our model to more locations outside of the United States. If we can mitigate the cost associated with expanding the model, we would also like to collect a much larger amount of data and train the model for longer.

References

[1] N Theethira and D Ravindranath. GeoguessrLSTM. https://github.com/Nirvan66/geoguessrLSTM/blob/master/documentation/CSCI5922_ProjectReport.pdf , May 2020.
[2] Anton Wallén. Web-based geographic discovery game. https://www.geoguessr.com/, May 2013.
[3] United States Census Bureau. Cartographic boundary files - shapefile. https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html, 2018/
[4] Sean Gillies. Shapley, January 2020.
[5] Google Maps Platform. Street view static api. https://developers.google.com/maps/documentation/streetview/intro, 2020.