CSCI5922 Neural Networks Group Project: GeoguessrLSTM Project Report

Nirvan S P Theethira
nith5605@colorado.edu

Dheeraj Ravindranath
dhmu3474@colorado.edu

Abstract

Geolocation is the identification or estimation of real-world geographic location using location based data. The location based data used for geolocation includes but is not limited to satellite images, sensor data, traffic movement data, pollution data etc. While humans cannot effectively guess locations based on sensor data, well travelled humans are good at guessing the location of where the image was taken. With google street view, it is now possible to obtain image data from most location in the United States. This project hopes to use neural networks to emulate the results of a human when it comes to geolocation of google street view images.

Motivation

The motivation for this project comes form the game called Geoguessr [1]. The game presents players with street view images (see Figure 1), which the player, using any pre-existing knowledge, has to guess the location of the image. The game scores the user based on how closed the latitude and longitude of the guess was to the actual latitude and longitude of the location of the street view image. This is an extremely challenging task for a machine that has no prior knowledge that an avid traveller possess, therfore making this an exciting problem to tackle using neural networks. While previous projects have aimed to geolocate prominent landmarks of locations such as the Eiffle tower, The Taj Mahal etc. this project aims to use generic street view images. As seen in Figure 1, there are mountains and red stone buildings which can hopefully be used by a deep neural network to ascertain the image is from boulder.

Figure 1: Google street view of pearl street.

Preprint. Under review.

Previous Work

There are a few already existing approaches for the above mentioned task. One of the elementary approaches deals with the unsupervised k-NN algorithm [2]. It uses the clustering algorithm to cluster similar images. This idea is used to cluster images from a single location into a single cluster. A new image is run through the K-NN algorithm to find how close the image is to the centroids of the location cluster. The sofmax of these distances denotes how likely the image is from a particular location. There has been drastic improvements in the field of image recognition with the use of ResNet [3]. The next method therefore uses the CNN to geolocate images [4]. For the dataset, geotagged images scraped from Flickr was used in training and testing. One drawback of using Flickr images is that it results in indoor images and image of people that could clutter the dataset during training. The entire map is split into grids and images are collected from each grid. The model is trained to give a soft maxed output across the grids. Without diving into the data collection portion of the project, we can see that multiple images can be collected from a single location's google street view. This is because google street view presents a 360 degree view of a location and so images can be collected from multiple camera heading angles. The paper [4] discusses a technique were multiple images collected from a single location can be processed sequentially. To process multiple images in one training epoch, the list of images are fed into CNN's that give the vector representations of the list of images. These vectors are then fed into a sequential LSTM and the output is soft maxed across regions from which data is collected. For this project we intend to use more generic google street view images as our dataset and restrict the area of study to the United States.

Data collection

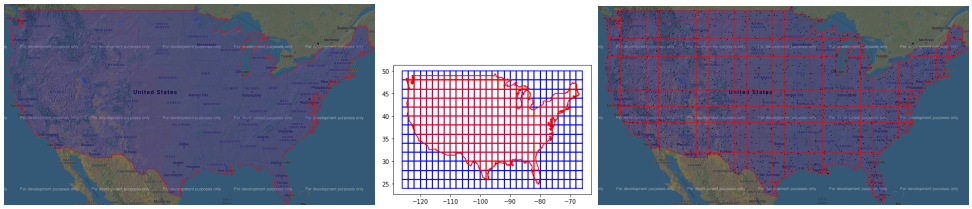

Figure 2: Left: Shape file of mainland US. Center: Fishnet used to split map into grids. Right: The map of mainland US split up into grids.

- The first step involved finding a shape file of the US mainland boundaries. The most recent shape file of the US published by the United states Census bureau [5] was downloaded. The shape file contained boundaries of not just mainland USA but also boundaries of Alaska and all the islands and union territories part of the US. To focus data collection efforts on just the mainland the boundary of mainland US had to be isolated. To do this, all shapes in the shape file were converted into polygons. The polygon containing the central geographic coordinate of mainland was found to isolate the boundaries of mainland US (see left Figure 2). The geometrical operations for this task were done using Shapely [6]. Once the latitudes and longitudes of the mainland US boundary were isolated, the coordinates were saved into a pickle file.
- The next step involved splitting the mainland polygon into grids. To do this a fishnet or mesh of square boxes were first overlayed on the boundary polygon (see center Figure 2).The squares that intersected with the boundaries of the polygon were clipped at intersection points to split the polygon into grids (see right Figure 2). Each grid had a maximum area of four square units which translated to an area of 138.265 sq units with grids at borders being smaller. Grids that were too small (less that 0.1 times area of the largest grids) were combine with the neighbouring larger grid. This was done to avoid some grids having no data. The entire process resulted in the map of the US being split up into 243 grids. The geometrical operations for this task were done using Shapely [6].
- The final step involved image collection from multiple locations in a single grid. Image from forty locations were collected per grid. The date the image was take was also stored along

with the image. The google street view static API [7] was used to collect image data. Since each location has a 360 degree view, 3 images were collected per location. The three images per location were collected from heading 0^o, 90^o and 180^o per location. The images were saved into a file directory structured database. The structure format looked like: $data/ < grid - number > / < latitude, longitude > / < number - image - date.jpg >$. An example file directory structure looked like: $data/0/42.775957, -124.0667758/0 - 2009 - 07.jpg$. For this project, the map is split up into 243 grids and 40 locations are queried per grid. This data scraping sweep summed up to 9720 location data points. Three images are collected per location and all of this totals to 29160 images that were collected. With each image averaging a size of 30 killo bytes, this lead to the collection of about 0.8 GB of data. At a price of 0.007 dollars per image, it cost about 204 dollars.

- The next step involved combining the data to a format that's easy to query for straining. Since each location has 3 image files and and each location is a training data point, that's how they are stored. A single folder is used to store one folder per location with each file containing three images. Folders are named with the format $< grid - number > + < latitude, longitude >$. Each location folder contains an image file with format $< number - image - date.jpg >$ with three images per location folder. The file structure for a single image file looks like $dataCombined/ < grid - number > + < latitude, longitude > / < number - image - date.jpg >$. An example looks like $dataCombined/0 + 42.775957, -124.0667758/0 - 2009 - 07.jpg$. All location folder names were then collected and split into train and test data. The train test data was stored in files named $trainFiles.npy$ and $testFiles.npy$ respectively. The 9720 data points that were collected were split into 8748 training and 972 testing points. About 90 percent of the data was reserved for training and 10 percent for testing.

CNN/LSTM model

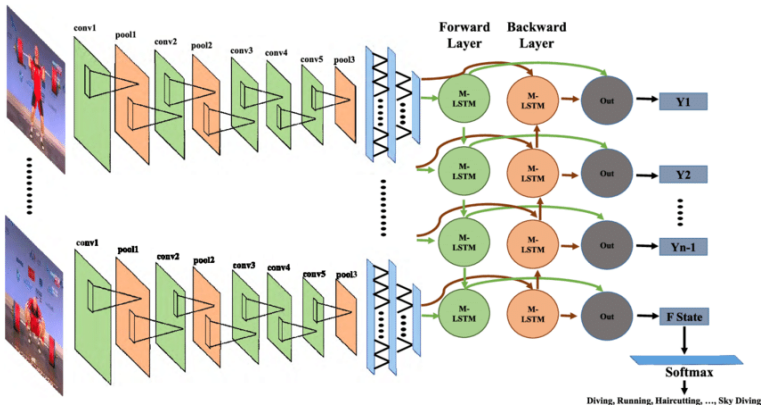

Figure 3: Structure of the proposed deep learning model.

The neural network structure consists of a CNN/LSTM combination (see Figure 3). This idea was inspired by PlaNet paper [4]. Here a single training epoch will consist of multiple images obtained from a single 360 degree sweep of a location. To process multiple images in one training epoch, the list of images are fed into CNN's that give the vector representations of the list of images. These vectors are then fed into a sequential LSTM and the output is soft maxed across grids from which data is collected.

Model Structure

The machine learning task at hand was to predict the grid a sequence of images came from. To do this the input images are loaded and converted to numpy arrays using the $tf.keras.preprocessing.image.loadimg$ function. The array is made up of rgb values and has the following shape $(300, 600, 3)$. Since the input consists of three such images, the shape of the

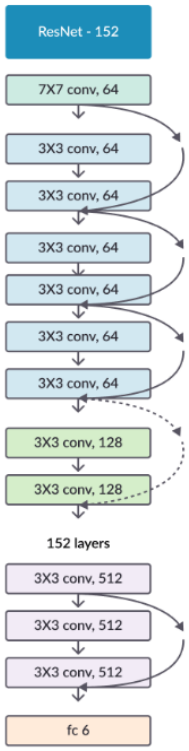

Figure 4: Restnet pre-trained model.

input turns into $(3, 300, 600, 3)$. The model uses a softmaxed output for prediction. Therefore for training, the grid numbers corresponding to given input image vector are converted to one hot vectors. This is done using the $tf.keras.utils.tocategorical$ function. The one hot vector has a shape of $(243,)$ with all values zero except the location representing the grid that has a one. There were two models trained to compare the perfomance of one with another. The first model had a restnet CNN whose weights were frozen during training. The restnet model was connected to an LSTM to process a sequence of three images. The next model had a trainable CNN connected to an LSTM. Below are the structures of both models explained in greater detail

- The first part of the first model is the ResNet [3] pre-trained model. ResNet, short for Residual Networks is a neural network used as a model for many computer vision tasks. This model was the winner of ImageNet challenge in 2015. The fundamental breakthrough with ResNet was that it made training of extremely deep neural networks with 150+layers possible. ResNet first introduced the concept of skip connection. RestNet stacks convolution layers together one after the other just like regular models. But RestNet also adds the original input to the output of the stacked convolution block. This is called skip connection. One of the reasons skip connections work are because they mitigate the problem of vanishing gradient by allowing this alternate shortcut path for gradient to flow through.They also allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse. The pretrained Keras Restnet model is used for this project. The model is loaded and the weights are frozen and trainable is set to false. This is done as the restnet model is only used to convert the image into a meaning full vector representation. This helps use transfer learning from the restnet to the aid the trainable CNN in the next step. The restnet structure can be seen in 4. The restnet CNN model is connected to an LSTM through a time distributed layer to process a sequence of images. The model has a total of 24,809,715 total parameters with 1,222,003 being trainable parameters.


```
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.], dtype=float32)
```

Figure 5: Top: Sample Input images for a single location. Bottom: Output categorical vector.

- The second model has a similar structure with the only difference being a trainable CNN model is used instead of the pre trained non trainable restnet. The trainable CNN model uses $tensorflow.keras.layers.Conv2D$ layers stacked over one another with $tensorflow.keras.layers.MaxPool2D$ and $tensorflow.keras.layers.BatchNormalization$ used between layers. The trainable CNN model is connected to an LSTM through a time distributed layer to process a sequence of images. The model has a total of 5,518,003 total parameters with 5,516,083 being trainable parameters.

Training

There are two main steps in training with the first being generation batches of data to train on and the second being training each batch of generated for a certain number of epochs and a certain number of steps.The list of folder names to train on were obtained from the $trainFiles.npy$ file that has a list of folder names that are named after the latitude and longitude of the location the three images were collected from. Three images are present in each folder that contain google images from different headings at that location. The list of training folder names are used to load three training images per location. A single training data point is seen in figure 5. Data from all training locations are loaded in batches. The model was trained in batches using a generator function. A list of training location folder names from $trainFiles.npy$ are given to the the generator function. The chunks to split the data into is specified using batch size. The generator split the list of location folder names into batches of the specified size. The batch of location folder names are passed to a read Data function to read images related to the batch of file names. The $tensorflow.keras.preprocessing.image.loadimg$ function is used to load three images per location folder with a target size obtained from the built models input layer. The input images are converted to a vector using $tensorflow.keras.preprocessing.image.imgtoarray$. The shape for this project is $(300, 600, 3)$ per image which covers the width and height of the image and the three RGB values for each pixel as seen in figure 5. Since a single data point covers three images per location, the shape of a single three image data point is $(3, 300, 600, 3)$. A single batch from the generator is a list of training input data which has the shape $(batchSize, 3, 300, 600, 3)$. To generate the target outputs for training, the file names themselves were used. As mentioned in the data scraping section the location folder names look like $< grid - number > + < latitude, longitude >$ an example of which looks like $0 + 42.775957, -124.0667758$ as seen in figure 5. This can be used to extract the grid number by splitting on $+$ and getting the first element. This is done to the list of file names provided in the specific batch to generate of list of grid numbers corresponding to the input image vectors. The $tf.keras.utils.to_categorical$ function is used to convert the list of grid numbers to one hot vectors. Each output vector is of size $(243,)$ with most positions of the vectors being 0. The position corresponding the grid number for a particular data point is set to one as seen in figure 5. This process is used to generate a batch of input image output grid one hot vector pairs. For this project a batch size of 1000 was used to produce a thousand input output pairs for training per batch.

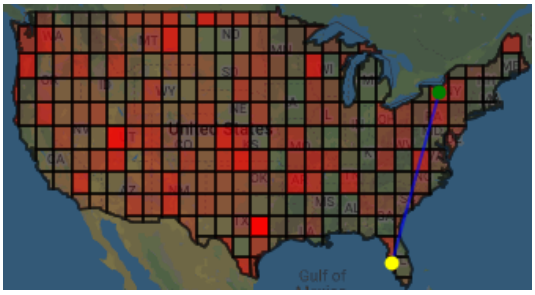

Figure 6: Predictions weights for each grid plotted using google maps API. The green dot indicates the actual grid the image came from and the yellow dot indicates the predicted grid with the highest weight. The blue line indicates the distance between the predicted and actual grid

A single batch of 300 input output pairs were used to train a model for 20 epochs with each epoch having running for 300 steps. This gave the model sufficient epochs to train on a single batch before moving onto the next batch of data. This process was repeated 29 times to train across 8748 training points.

An additional feature know as a callback was implemented during training. The callback record the model loss at the end of every epoch. The callback was used to save the model at the end of a single batch of training. This means the model saved a copy after 20 epochs of training on a single batch of a 300 input output pairs. This happened at each epoch to generate 29 models by the end of training completion.
To accomplish the computationally intensive training task, google Colab was used. The geoguessr model creation code was uploaded to Colab along with the entire dataset. A Colab notebook was use to collect data files and run the model creation and training code. Saving at the end of a single batch of training helps prevent loss of all progress and the model if Colab crashes at any point during training. As Colab has a training cap of 12 hours and the training took more time that that, the saved models were used as checkpoints and reloaded to continue training after the end of a single 12 hour period.

Prediction

The test files are used to test the predictive capacity of the model. Three image files for a location along with the expected output grid number is used to create an input, expected output pair. The three input image vectors of shape $(3, 300, 600, 3)$ is feed into the trained models prediction function. The function outputs a soft maxed array of shape $(243,)$. The entire array sums up to 1. A single index of the list has the models assigned score or prediction for that numbered grid.

To properly visualize this array of prediction scores, the python google maps API gmaps [8] is used. The every grid is plotted as a polygon using the $gmaps.Polygon$. The dimensions of each polygon of the grid and the entire grid itself is a result of the map splitting process talked about in the data collection section. The opacity of each grid polygon is set to the weight at that index in the array of predictions as seen in figure 6 as different opacity of reds. The numbers in the predictions array were very small and could be directly used to set opacity. For the purpose of getting good visuals, the predictions weights were re scaled to make each value between zero to one. This made the highest confidence prediction in the array have a opacity value of one and the lowest confidence prediction a value of zero. The center of the grid with the highest prediction score and the center of the grid that the image is actually from is used to find the actual and prediction locations. This is seen in 6 with the actual location denoted by a green point and the predicted a yellow point. The distance between these points is calculated using haversine distance which is discussed in detail in the evaluation section. This distance is used to draw a line between the start and end points which is seen in figure 6 as the blue line.

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 3, 2048)	23587712
lstm (LSTM)	(None, 64)	540928
dense (Dense)	(None, 1024)	66560
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 128)	65664
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 243)	15795

Total params: 24,809,715
Trainable params: 1,222,003
Non-trainable params: 23,587,712

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 3, 512)	4689216
lstm (LSTM)	(None, 64)	147712
dense (Dense)	(None, 1024)	66560
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 128)	65664
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 243)	15795

Total params: 5,518,003
Trainable params: 5,516,083
Non-trainable params: 1,920

Figure 7: Left: Model with .

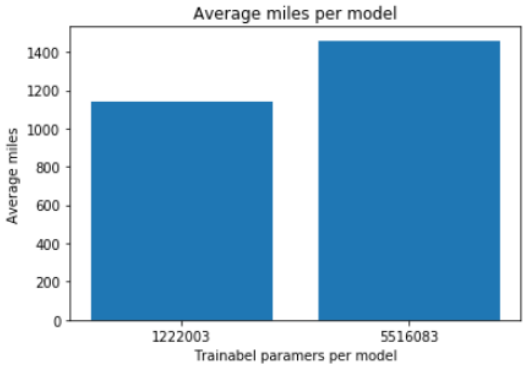

Figure 8: .

Evaluation

The list of test location folders are used to evaluate the model. The test location folder names are sent to the read data function that loads the three image vectors related to the location along with the expected output grid number to create an input, expected output pair. The 1458 test input image vectors of shape $(3, 300, 600, 3)$ is feed into the trained models prediction function one at a time. The evaluate function predicts a soft maxed array of shape $(243,)$ for every one of the 1458 test locations. An argmax is used on each output prediction to get the number of grid with the highest prediction score or the predicted grid. Since we already know the grid it was supposed to be, this is considered the actual grid. The centroid of the actual and predicted grid is calculated and the distance between the two points are calculated using haversine distance.
Since the earth is a rough sphere, regular euclidean distance cannot be used for distance calculation. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. The law of haversines relates the sides and angles of spherical triangles.

$$2R \arcsin\left[\sqrt{\sin^2\left(\frac{x_1 - x_2}{2}\right) + \cos(x_1)\cos(x_2)\sin^2\left(\frac{y_1 - y_2}{2}\right)}\right]$$

In the above formula x_1 and x_2 represent the latitude and longitude of the actual grid centroid and y_1 and y_2 represent the latitude and longitude of the predicted grid centroid. The actual and predicted grid locations for a single image along with the distance between them can be seen in figure 6. Using haversine distance the distance between actual and predicted location grids for a list of test image files can be found. The list of distances are averaged to give a single distance score for a model evaluated on a list of test image files.
In this project, two models were trained and tested using the methods previously discussed. The two models had the structure discussed in the model structure section. The keras specification for each of the two models can be seen in figure 7. From figure 8 we see that the predictions of model with

the non trainable pretrained restnet CNN model is off by an average distance of around 1200 miles while the model with the the trainable CNN is on average off by a distance of 1400 miles. While neither model seems to perform exceptionally well, the restnet model seems to perform better that the non restnet model. This could be because the model with the trainable CNN has more trainable parameters and thus might be over fitting on the data.

Conclusion

From the performance chart show in figure 8 we see that even the best model has an average error in distance of 1200 miles. Even with each grid having a 138 sq mile radius, an 1200 mile error rate is not great. One way to improve this would be to relax evaluation metrics on the model. To do this, instead of taking the argmax of the predicted output and only considering the grid number with the highest score, the actual confidence score can be considered. This way we can see how certain the model is of its prediction. The confidence scores themselves can be used to consider other top scoring grids in evaluation instead of just the best one. This can be used to find how the model predicts an image to see if the predicted grid has images that look similar to the actual grid.
Another reason for the models poor performance could be the similarity in training data. The google street view data collected looks very similar as the images are mostly of roads with trees next to it. If the model was given images of defining landmarks of each location it could perform better. The model was also just trained for 20 epochs on each batch of data. Increasing the number of epochs could help improve the model.
Another noteworthy fact is that human players of the game geoguessr are allowed to walk around (specific number of steps) using google street view. If the machine learning model was allowed to do the same and collected data as a larger stream of images per location, there could be a performance improvement with the model learning more about a location just like a human player.

References

[1] Anton Wallén. Web-based geographic discovery game. `https://www.geoguessr.com/`, May 2013.

[2] James Hays and Alexei A. Efros. im2gps: estimating geographic information from a single image. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet - photo geolocation with convolutional neural networks. *CoRR*, abs/1602.05314, 2016.

[5] Unites States Census Bureau. Cartographic boundary files - shapefile. `https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html`, 2018.

[6] Sean Gillies. Shapely, January 2020.

[7] Google Maps Platform. Street view static api. `https://developers.google.com/maps/documentation/streetview/intro`, 2020.

[8] John Kleint. googlemaps 1.0.2. `https://pypi.org/project/googlemaps/1.0.2/`, 2009.