474 Final Project - GeoGuesser AI

Brayden Smith
Department of Computer Science
Brigham Young University

Abstract

The game of GeoGuesser tasks the player to consider images taken from around the world and then guess as to the location of the images. In this report, I cover the iterative building of multiple convolutional neural networks (CNNs) to a recurrent CNN (RCNN). After much testing and iteration, the RCNN performs better than the average player on the substantial dataset that I have attained from Google's Street View API, but still only scores on average an 18,000 out of 25,000 in a game of GeoGuesser with large variation.

1 Introduction

GeoGuessr is an online geographic discovery game originally designed by Swedish IT consultant Anton Wallén. The premise of the game is that the player is put into google street view on a random street somewhere in the world. Through navigating around for a few seconds the player then must guess as to their location. Afterwards points are then allocated to the player with respect to the distance of the guess and the actual random location. Usually the player is given 5 positions to guess, and then some final score is tallied up.

The problem of using DNNs to determine the location that a photo was taken has been tackled many times by many different groups. Models range in the scope of CNNs like those built by Peddada and Hong that focused primarily on images from a certain city and achieved fairly high accuracy, and models like PlaNet (Weyand et al., 2016) are CNNs amplified with Long-Term, short-term memory units (LSTMs) used to classify images from all over the world.

Thus, after much research into how previous models had been built, I was curious as to the efficacy of the Recurrent Convolutional Layers as built and tested by Liang and Hu for object recognition. I felt that with the great improvements that PlaNet had made when augmented with LSTMs that in giving my model a sort of built in memory to build a final answer when looking at the 4 images, then I could build the most accurate model.

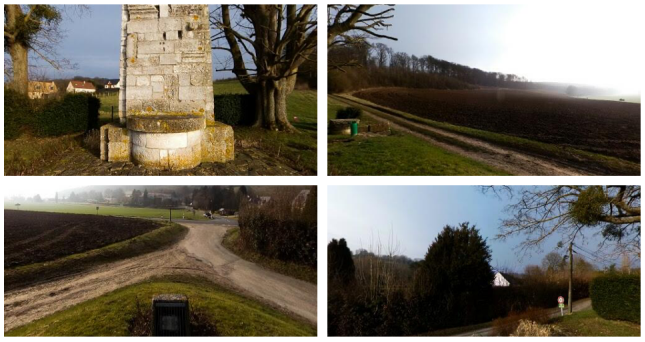
2 Methods

The following section follows closely to the timeline of events as I iteratively built models, beginning with data collection through to the final best model and evaluation.

2.1 Data

Through Google's Street View API, I was able to download roughly 84,000 street view images from around the world. Downloading 4 images at each location, I had roughly 21,000 locations throughout the world that I could train and test models on. While originally I felt this was plenty of data to train the model on, I noticed a fair amount of bias in the model to more heavily populated areas of the world, and I feel that would be due to how I downloaded most of the data, by choosing cities and then picking random points in the general area. Thus, when presented with a more rural image, the model suffered greatly. Likewise, considering the resources of models like PlaNet (Wayend et al., 2016) with more than 126M images to use, the models I built are lacking on total training data.

Figure 1: Example single round input into the model, these 4 images are taken in the 4 cardinal directions outside of Rouen, France, player must make a guess as to the location of the picture and will receive a score out of 5000 points, each game consists of 5 locations.



2.2 Score and Loss

In the original GeoGuesser game, after a guess is made and submitted, the player receives some score based off of the distance from the guess and actual location and the size of the map. The maximum score a player can receive in a single guess is 5000 points. While the exact formula used by the official game is not released to the public, I decided to follow in the footsteps of observations of other people that the score of a guess would be calculated as $Score = 5000e^{\frac{-x}{2000}}$ Originally I had been using MSE as loss, and then tested Huber loss as my loss functions during early training, but I decided to test out using this scoring system as my loss by $Loss = 5000 - Score$. This appeared to be learning fairly well, likely since my goal was high scoring models. The models of Peddada and Hong used a linear loss and I found there to be fairly good results by using the following $Loss = \frac{254 \times DisplacementOfGuess}{280}$. However after awhile, I returned to the GeoGuesser scoring model and found the best results using the following $Loss = 180 - 180^{\frac{-Displacement^2}{20000}}$ where the order of displacement was increased to further punish the more wild distant guesses and also give a bit more leniency in guesses that were closer to the mark.

2.3 Early Models

I was hoping to get a better understanding of the difficulty and nature of the problem by tackling it first by evaluating different shapes and structures of CNNs trained on my data. Firstly, I took to fine tuning the last few layers of the ResNet-152 model as a quick exploration as to the usefulness of CNNs. Notably, the ResNet-152 is a classification model, however; I had decided that models built for this project would output a latitude and longitude location for their guess, rather than the systems used frequently in models like PlaNet where Earth's surface was sectioned off into a multitude areas that then were used as labels. I decided to do the more regression approach as this is how a human player would play the game, inputting a guess at a very specific location on Earth's surface rather than guessing an entire area.

After receiving fairly good results using ResNet's pre-trained weights, I took to building a CNN of my own to tackle classifying these images and their locations. Up until this point, I had been concatentating the 4 cardinal direction images on top of each other and had also tried stitching them together into a large panorama. I found the best results when stitching everything into one large image, but the best model, the original fine-tuned ResNet-152, I had the testing accuracy of an average score at around 6,351 per game out of the 25,000 maximum points (5 rounds of 5,000 maximum points per round). Considering that 6,000 is even less than the average score on GeoGuesser's official world map of 9,000 per game, I was fairly disappointed by the results.

2.4 Recurrent Convolutional Neural Network

Thus, in desire to greatly improve the model, I turned to work similar to what had been done to achieve better results of PlaNet, I decided I needed to implement some sort of memory system in the model. Since each input in the model is four images, I wanted to create a memory system so that the model would build up to its final answer after seeing all 4 images in random order. I turned to the idea of Recurrent Convolutional Layers (RCLs). My now Recurrent CNN consists of a few Convolutional layers similar to that of the first number of layers of ResNet-34, and then 3 layers of RCLs followed by 2 fully connected linear layers of 1000 hidden nodes, and then a final output layer for the longitude and latitude guess.

The RCL was by far the most difficult thing to build in this project, and to be completely honest, I am fairly sure that the implementation that I have so far is possibly not entirely correct. I am confident that there is learning and memory going on, but I feel I could still improve on my implementation, however; so far with my limited data set, I've achieved much better results with the RCNN than with the fine-tuned ResNet or my custom Convolutional neural networks.

Each Recurrent Convolutional Layer activates based off of the following.

$$z_{t,m,i,j,k} = (\mathbf{w}^b_{m,k})^\top \mathbf{h}_{(t,m-1,i,j)} + (\mathbf{w}^l_{m,k})^\top \mathbf{h}_{(t-1,m,i,j)} + b_{m,k} \quad (1)$$

Where $\mathbf{h}_{(t,m,i,j)}$ is the vectorized input on the patch centered at location i, j, and in layer m, when computed at time t for the kth RCL layer, thus m corresponds to the 3 RCLs in my model, t varies from 1 to 4 over the 4 images. Note that while $\mathbf{h}_{(t,0,i,j)}$ in the paper I am pulling this from is defined as the input image, I have instead defined it as the final output of the first few convolutional layers. Then that in RCL feature map k has the convolutional connections as vectorized format $\mathbf{w}^b_{m,k}$. Notably, the paper I pull this from also included an input of the RCL above k and previous in time $t-1$ to be included in the final calculation of $z_{t,m,i,j,k}$ this was omitted due to time, I imagine it could help, but due to time constraints I left it out.

Evaluating this new RCNN initially gave similar results to that of previous work, but after changing the initial layers a little bit, reducing the learning rate, changing the optimizer to Adam, and then correcting some issues with my implementation of RCLs, the model not only began to converge much more quickly in training but also began performing much better on my smaller testing samples. After never breaking a score of more than 7,000 per game, the model scored an average of roughly 16,200 each game over 10 testing runs of 10,000 images (2,500 locations).

3 Results

Considering my average scores, a fairly skilled GeoGuesser player, are around 22,000 out of 25,000, the model still has a bit of work to do on beating me, however; a score of on average 16,200 per game is quite respectable, with out of the 10 test runs, 20,487 the highest, I could see it being a fairly good opponent to a new player. Nevertheless, the model appears to be performing many times better than the initial naive CNNs and I imagine that it could perform even better with more data. I was fairly worried about overfitting on the training set, which it's still very possible that it is doing, especially considering that in all of the training and testing splits, cities that are trained on also have images that are tested on as well. It would be interesting to see if the model could train on images from a subset of locations from around the world, and then be tested on images of a different subset of cities and locations from around the world that are entirely new to the network's eyes.

4 Conclusion

As a student fairly new to the world of building and training deep neural nets (DNNs) I feel I have barely scratched the surface of the field of image classification and feature recognition in this final project. However, I am fairly proud of the results that I have achieved, for my final recurrent convolutional neural network (RCNN) can in certain situations outperform my human guesses. I look forward to continuing work in this and similar projects.

References

[1] A. V. Peddada and J. Hong. Geo-Location Estimation with Convolutional Neural Networks. 2015

[2] Ming Liang, and Xiaolin Hu. (2015). Recurrent convolutional neural network for object recognition. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2015.7298958

[3] Weyand, T., Kostrikov, I., and Philbin, J. (2016). PlaNet - photo geolocation with convolutional neural networks. Computer Vision – ECCV 2016, 37-55. doi:10.1007/978-3-319-46484-8