

# CSCI 1430 Final Project Report:

## Solving Geoguessr using Deep CNNs and Transfer Learning

*Hungry HOGs:* William Beakley, Paul Jeong.  
TA name: Eric Tang. Brown University

### Abstract

*Geoguessr is an online game where the player is given a random location via Google Streetview and must guess where the Streetview image was taken. Skilled players are able to figure out what country they are in nearly instantly, taking in clues from various memorized infrastructure, as well as a subjective feel that they get from certain areas. Our project focused on figuring out whether deep CNNs can learn these subjective qualities in order to beat Geoguessr. We used pretrained object detection weights from VGG16, and we trained dense layers totalling 26 million trainable parameters. Our testing showed that deep CNNs can learn subjective traits characteristic of different locations around the globe.*

### 1. Introduction

In our project, we will attempt to have a learning algorithm recognize features in Google Streetview images that are indicative of certain locations. This is a hard problem to solve, as it requires many hours of practice for humans to even approximate this skill, and many of the techniques involved would be incredibly tough to teach an algorithm without thousands of GPU hours. We cannot take advantage of reading signs, or recognizing certain languages and brands of cars. Solving this problem would not have much of an impact on anything in the real world, but answering this question may have implications for the abilities of deep learning algorithms in general.

### 2. Related Work

We strayed away from tutorials and other guides when making this project, while they do exist. We did, however, look at one implementation of a Geoguessr AI, and took away from it the tile prediction method of classification. This was from Nirvan S P Theethira's Geoguessr AI on GitHub [1]. In terms of software, we used a couple of datasets, plugin's, and GIS software applications to get to our finished product. We used Mapillary's [places](#) dataset. For the pur-

poses of creating demos, we used the Jupyter Google Maps plugin. We analyzed the dataset and its drawbacks in the open source GIS platform QGIS. We attempted to expand and refine our image dataset with Mapillary's API, but we did not end up using this in the final product. We used the Google Maps API for the purposes of creating demos as well.

### 3. Method

Our goal was the be able to feed a streetview image to our model and have it give us the location the streetview image is from. This is a very hard task, so in order to do it, we started small. We first created a relatively small dataset only containing images from Berlin and San Francisco. Simply applying binary classification, using VGG16 weights, and setting up the following head produced 95% accuracy:

```
1 self.head = [  
2     Flatten(),  
3     Dense(512, kernel_initializer='uniform', input_shape=(512,)),  
4     Dropout(0.5),  
5     Dense(100, kernel_initializer='uniform', input_shape=(512,)),  
6     Dropout(0.5),  
7     Dense(2, activation='softmax',  
8         kernel_initializer='uniform',  
9         input_shape=(100,)),  
10 ]
```

We then tested the model on Streetview images, and it appeared that the model was having a decently tough time adapting to the format of Google Streetview as opposed to the Mapillary dataset. However, it was still predicting significantly better than chance, so we assumed that the dataset would be able to adapt to general features rather than simply features present in the dataset. In order to create a better dataset, we turned to Mapillary's API. First we tried using Mapillary's function to get images close to a point in order to get data that was spread out. However, it turned out that there was no way to use this function in order to get

images that were spread out. If we queried the API to get images close to a point in San Francisco, it would return 60 thousand image URLs in a 1km radius, which took around a minute.



Figure 1. 60k points from San Francisco in a 1km radius.

There was no limit parameter built into the Python API, so we had to grapple with this runtime issue. There was a radius feature built into this function, so we tried to query the API using small radii to reduce runtime, but this resulted in similar runtime and unpredictable results, as sometimes the query would take a minute and not return any images. We realized that the API was built off of "tiles", which were essentially spatial indices. However, these tiles were small, but contained way too many images for our purposes, as our goal was to spread out our training data, as to not have our algorithm memorize certain buildings.

In order to get around these difficulties, we turned to creating grid tiles using QGIS. The goal was to populate one image per grid tile, as to be certain buildings were not in multiple images.

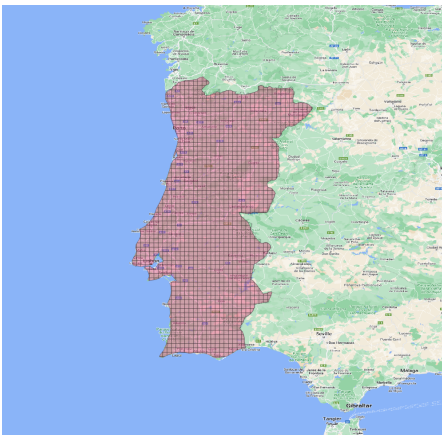


Figure 2. Grid created in Portugal.

Mapillary's API had a function to use bounding boxes to query images. We figured we would use each bounding box we had created to query images. We would then grab only one image from each bounding box in order to create

this dataset. However, it turned out that the bounding box function in the API would query all of the tiles intersecting the bounding box. As such, we were essentially asking the API for every single image in Portugal. If we wanted to apply this methodology globally, we would have to query every single one of Mapillary's 1.8 billion images, which is certainly infeasible. Battling the API documentation took up a large chunk of our time, and we eventually gave up on using it. Reflecting on the issues that the dataset created, it would have been better to simply deal with the long runtimes, and create a better dataset. Another option we could have used is using our Google Cloud coupon, and the free trial of Google cloud to create a dataset. Google Cloud has a feature where it simply returns one Streetview image close to a lat and lng, which was perfect for our purposes. We could have done this without spending any money, but we were afraid that it would have been a dead end. The issues with our dataset lay in the fact that the images were in sequences of Streetview images. The dataset was offered to train computer vision algorithms to figure out how the images fit together. As such, when we randomly split the images into train and test images, some of the test images would be very similar to the train images. This created erroneously high test accuracy. However, we underestimated the problems this would cause with overfitting.

We then proceeded to adapt our 2 city classification model to a 24 city classification model, using all 1.2 million of the images in our dataset. We used python scripts to adapt our dataset into the directory structure expected by Keras' flow function. In order to identify more features complex features, we added dense layers.

```

1 self.head = [
2     Flatten(),
3     Dense(1024, kernel_initializer='
4         uniform', input_shape=(512,)),
5     Dropout(0.5),
6     Dense(512, kernel_initializer='
7         uniform', input_shape=(512,)),
8     Dropout(0.5),
9     Dense(100, kernel_initializer='
10        uniform', input_shape=(512,)),
11        Dropout(0.5),
12        Dense(24, activation='softmax',
13            kernel_initializer='uniform',
14            input_shape=(100,)),
15    ]

```

After many hours of training, we achieved a 90% test accuracy on our dataset. We incorporated our model into a Jupyter notebook, so that we could query our model with Streetview images from our clipboard, and from the Google Maps API, both for the purposes of building demos and for sanity checking our results. We noticed that the model was

having a hard time identifying images that were outside of the range of our training dataset. Even a road one away from a road it was trained on would not be identified correctly. However, we did notice that the algorithm was clearly picking up on some of the features of certain areas. If we showed model an image of Austin TX, it would either predict Austin TX or Phoenix AZ, which is a very human way to classify Austin TX. Limited on time we could use for perfecting our dataset, we chose to proceed with latitude and longitude prediction for images.

We adapted the keras model to use raw labels rather than class labels. We used Pandas to create a dataframe with 1.2 million labels lat-lng labels in the correct formats. We quickly noticed that our model could not learn lat-lng labels on a global scale, so we switched to predicting lat-lng labels within one city. We chose Budapest, as we had the most images from it. We used the same training head, switching the final dense layer to two outputs instead of 24, and changed the activation to linear rather than softmax. We changed the loss function to mean squared error rather than sparse categorical error. Our model was able to make some predictions correctly, but it was clearly biased towards the center of the city. We used the Google Maps API to draw the predicted latitude and longitude contrasting with the actual coordinates.

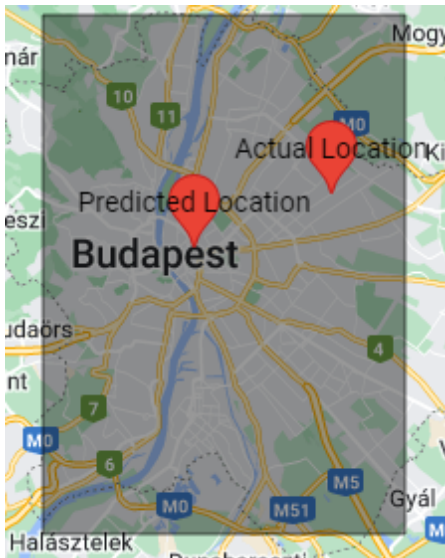


Figure 3. Erroneous prediction by our model using an image from streetview.

Unsatisfied by these results, we decided to implement grid tile prediction instead of direct latitude and longitude prediction. We got this idea from a project titled Geoguessr AI on GitHub [1].

We used a 10 x 10 grid for the purpose of predicting the latitude and longitude. We used the same head as our first model, changing the number of outputs to 97, as there

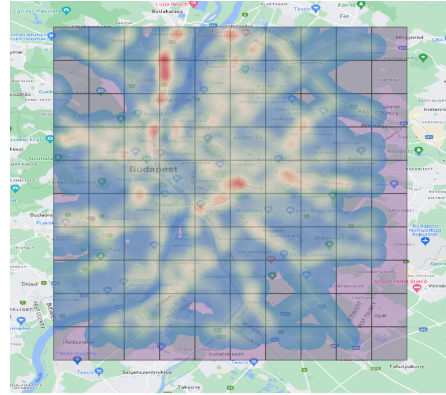


Figure 4. The grid used. The heatmap shows our dataset's coverage.

were 97 grid tiles represented in our dataset. We used Pandas to create grid labels based on the latitude and longitude of the images. This prediction method was less biased than our direct latitude and longitude prediction methods. It seemed to pick up on some of the more human features of the images. If the image looked more suburban, it would generally predict grid squares farther from the center of the city. We used the Google Maps API to draw the predicted grid square in addition to the actual location.

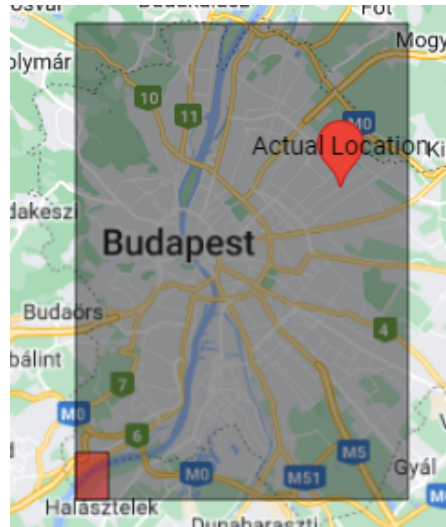


Figure 5. Drawn prediction from our third model.

Having created these three models, we then refined our models into an interactive Jupyter notebook for the purposes of creating a demo.

## 4. Results

Displayed below is an example of the predictions (Figures 7, 8, and 9) of the 3 CNNs on an image of Budapest (Figure 6). The corresponding LIME explainer images are included as well.



Figure 6. Initial image from Budapest.

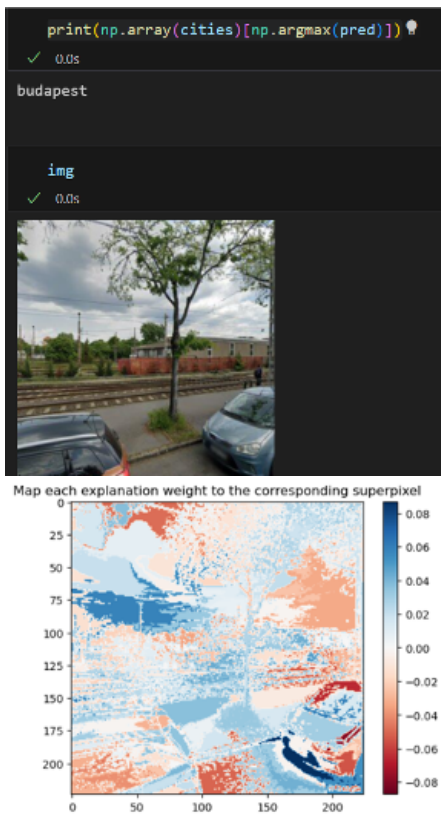


Figure 7. *Top*: Our CNN's prediction of the image's city. *Bottom*: Corresponding LIME explainer image.

As shown by the images, the city CNN correctly predicts the image as being from Budapest. While the latitude and longitude and city grid predictions are not as accurate, both CNNs seem to pick up on the fact that the location shown in the image is within the vicinity of the center of Budapest. Within all three LIME explainer images, it can be observed that all three CNNs consider the blotch in the sky and the front of the car in the bottom right corner to be telling of

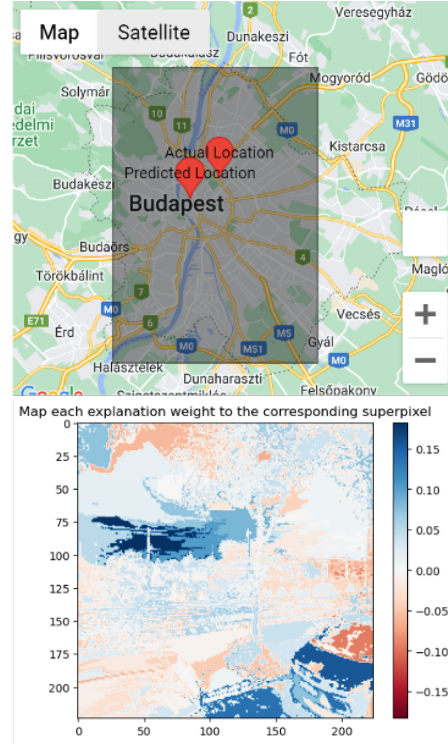


Figure 8. *Top*: Our CNN's prediction of the image's latitude and longitude. *Bottom*: Corresponding LIME explainer image.

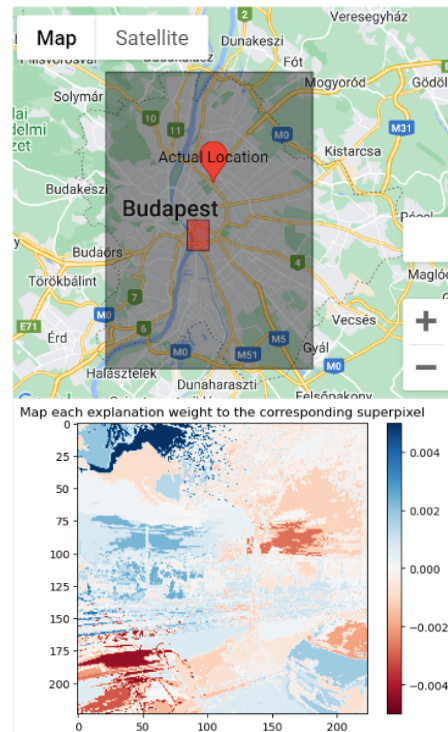


Figure 9. *Top*: Our CNN's prediction of the image's location via grid square. *Bottom*: Corresponding LIME explainer image.

Metric of Prediction	Accuracy Measure
Latitude & Longitude	MSE: 0.05992
City	Categorical Accuracy: 90.20%
Grid Square	Categorical Accuracy: 74.64%

Table 1. Accuracy results for our three different models.

Budapest.

By testing various Google Streetview images and using the LIME explainer, it was observed that the CNNs tended to look to patterns in the sky or the texture of the road/streets as indicators of a scene’s location in the world.

We assessed the performance of our CNNs in two different ways: categorical accuracy for city and grid square prediction and mean squared error for latitude and longitude prediction (Table 1).

While the models reached relatively high and satisfactory accuracies with the testing dataset, when asked to evaluate images that were screenshots from Google Streetview, the three CNNs exhibited moderate performance. A specific behavior displayed by the CNNs was that with images that did not have a clear view of the road, the CNNs would rarely predict the correct city displayed in the image, let alone the latitude and longitude. Because the images in the dataset often displayed the road as they were taken from a car, it is likely that the models were trained to perform well on images with roads and would be confused on images that lacked roads.

Because the weights for our CNNs are pretrained, predicting the city/latitude and longitude/grid square of an image requires negligible computer performance.

#### 4.1. Technical Discussion

One aspect of our program that was not thoroughly considered until the towards the end of the project was the use of VGG-16 weights pretrained on ImageNet. Because the base of all of the CNNs is the pretrained VGG-16 weights, the CNNs are predisposed to pick up on objects in the scene that would be helpful for classifying images from ImageNet, which contains classes that may not be helpful for determining location of an image with many details and elements. This choice could have had a huge impact on the accuracy performance of the CNNs. One alternative would have been to train a CNN built from scratch. While this approach would have been more time consuming, it would have produced a model that would be more tailored to examining scenes of the outside world.

Dataset curation was another issue that was faced. Within our dataset from Mapillary, it was found that images were very condensed around roads and streets and some images were extremely close in proximity to other images. This could explain why the test accuracy was extremely high for

city prediction, while showing moderate performance for random images taken from Google Streetview.

#### 4.2. Socially-responsible Computing Discussion via Proposal Swap

We believe that the first concern that Geoguessr content creators will be displaced by our CNN is somewhat invalid. Geoguessr content creators attract consumers because they are entertaining and/or because the level of skill needed to become extremely proficient at the game is laudable and impressive. While a Geoguessr bot that achieves high accuracy at the game would be extraordinary, it is unlikely that it would displace Geoguessr content creators due to the fact that having a computer perform well at Geoguessr is not as impressive or as entertaining as a human who performs well at Geoguessr. It is possible that a “Geoguessr AI” can bolster the popularity and social media presence of Geoguessr content creators as the other team has mentioned, and AI can even be used to improve at Geoguessr by uncovering what features to look for in a scene, further boosting the game’s popularity. In the end, having a CNN negatively impact content creators seems unlikely.

We believe that the second concern about the geolocation CNN infringing on privacy is a valid concern. Certainly if computers were able to accurately determine the location of the scene based off an image and such a program were accessible to the public, it would compromise the privacy of many of those who post pictures online which could result in some using such a model for malicious purposes. One way to combat this issue would be to limit the use of the program to only organization/people who are trustworthy and will not abuse the power. However, such a task is not simple and it is likely that people will find ways to use the program anyways. An alternative would be to advise people not to share photos or videos of their surroundings if they do not want others to discover their location.

We believe that the third concern about the existence of bias within machine learning is fair. In fact, a considerable amount of bias was encountered when working on the project. We found that the grid square CNN tends to predict central locations within Budapest as most of the data is concentrated within that area. One method of mitigating this bias is to create a more representative and balanced dataset. If the training and test datasets could both contain images equally spread across the world and with similar frequencies, it could account for certain areas of the world being excluded from recognition. Another way to minimize bias within the AI is to use analysis tools like the LIME explainer to obtain a better understanding of why the model is biased and what specifically the model is biased towards.

## 5. Conclusion

Our project aimed to answer the question of whether neural networks can reliably use subjective features in order to predict the location of Streetview images. In the end, our work showed that this prediction is very possible. While our project did not produce the most satisfying results, it is safe to say that CNNs can get a feel for certain locations. In order to make a very powerful Geoguessr bot, the dataset would need to be much robust. It would need to have noise filtered out, and it would need to have a well monitored test dataset in order to curb overfitting. This work was outside the scope of our project, and we are proud of what we created. Our visuals created a nice demo, generating interesting conversations in our poster presentation, made our work explainable, and makes it possible for others to pick up where we left off. It is possible that AIs like this one will be able to create content for Geoguessr content creators. Aside from that, we think our project only has minimal and positive impacts on the world and the field of computer vision.

## References

- [1] Nirvan S P Theethira. Geoguessr ai, 2020. Grid identification. Supplied as additional material. 1, 3

## Appendix

### Team contributions

**William Beakley** William generated many of the creative ideas used in the project, such as training a CNN on grid squares of latitude and longitude and having the CNNs predict based on screenshots off of Google Streetview for our presentation demo. Much of the visual presentation work to display the predictions of the CNNs was coded by William. William has also worked on building the model architecture for the city prediction and grid square CNNs while also training the CNNs on his GPU. William also used GIS applications in order to build visuals and conduct analysis.

**Paul Jeong** Paul has done work on obtaining the data used for the CNN as well as splitting up the data for training and testing. Preprocessing was handled by Paul, ensuring that the dataset images and class labels were in a format that was able to be read by the program. Paul worked on building the model architecture for the city and latitude and longitude CNNs and designed most of the poster.