GeoGuessrAI: Image based GeoLocation - Final Report

Nirvan S.P. Theethira
CSCI6502 001
Boulder, CO
nith5605@colorado.edu

ABSTRACT

Geolocation can be defined as 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. Humans are not the best at guessing locations based on sensor data. But when it comes to location based image data, well travelled humans perform fairly well 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. The game Geoguessr [8] presents street view data to human players who attempt to guess the location of the street view image. Using the recent advances in image recognition technology this project aims to emulated and hopefully surpass the scores of human players in the game Geoguessr for the United States map.

MOTIVATION

The geolocation of a real-world location is the latitude and longitude of that location. One specific application of geolocation would be inferring the location of a tracked animals based, for instance, on the time history of sunlight brightness or the water temperature and depth measured by an instrument attached to the animal. While this is a useful, real world application, the ability to just use images to identify a location would greatly help in a myriad of tasks. The ability to geolocate images can help in geotagging new unseen data thereby facilitation the creation of new dataset. Another application of this technology could be to aid in identifying location with low or no connectivity. A person who is lost in a low connectivity region could snap an image of the surroundings to get an estimate of their location.

The motivation for this project comes form the game called Geoguessr [8]. The game presents players with street view images (see Figure 1) which the user, using any pre-existing knowledge, has to guess the location of the image. The use of pre existing knowledge offers well travelled players an advantage in the game. Prior knowledge may include the use of language on street signs, motor vehicle type or in some instances the fact that certain regions (military bases) cannot be reached by google maps can help in elimination of those locations. The game scores the user based on how closed the latitude an longitude of the guess was to the actual latitude and longitude of the location of the street view image. While this is an extremely challenging task for a machine that has no prior knowledge that an avid traveller possess, the use of CNN's for other image recognition tasks have given astonishing results. 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 images to extract locations. While humans can use prior knowledge as mentioned above, most humans who play the game use geographic features such as building styles, vegetation, mountains, weather etc to find locations. With the plethora of street view images available, a deep neural network can learn to notice these subtleties. As seen in Figure 1 on the left image, there are mountains and red stone buildings which can be used by a deep neural network to ascertain the image is from boulder.

RELATED WORK

There are three main approaches that have been researched:

- There are a few already existing approaches for the above mentioned task. One of the elementary approaches deals with the unsupervised k-NN algorithm [3]. 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 [4]. The next method therefore uses the CNN to geolocate images [10]. 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 (see Figure 3).
- The third method takes into account that multiple images can be used in a single classification [10]. 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 (see Figure 3).

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Figure 1. A comparison of a random location (Boulder,CO) on google street view to a famous location (Eiffel Tower) on google street view.

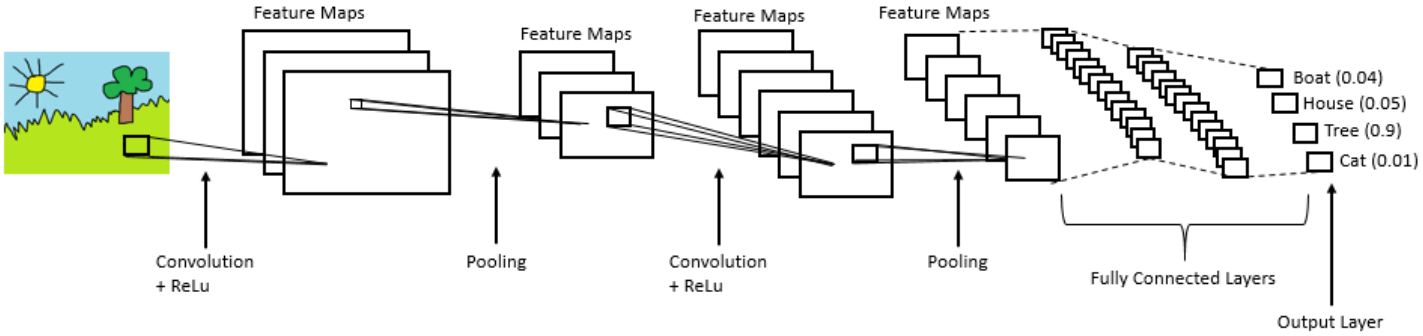

Figure 2. The CNN with soft maxed output across location grids.

PROPOSED SOLUTION

The proposed solution takes the following steps:

- The first step involves the map and defining geographic girds to collect data from. For this project the map of the United States will be chosen. This is because the country has a diverse geographic landscape with definitive features in each region. This can be leveraged by the CNN to give good results. Another reason for this choice is because a plethora of street view images are available for most regions in the country. Confining the area to just the united states will also make comprehensive training possible as using data from the entire planet will require a lot more computing power for training than is available for this project.

- The next step is to split the map into logical grids. The size of these grids will be a hyper-parameter to be tuned. A view of the US map split up into grids that are the size of states is show in Figure 3. All geometry related tasks will be handles using the python package called Shapely [2]. Data points will then be scraped from each grid with the number of data points being scraped being a hyper parameter.

- Data will be scraped from google street view. This will be done using the google street view API [9]. Data scraped from a single location will act as a data point. Since google offers a 360 degree street view, multiple images will be taken at each location to act as a single data point for training. This list of images will be processed sequentially using the LSTM technique mentioned above.

- Once the data collection pipeline is ready, training will be conducted using the CNN machine learning techniques mentioned in the related work section.

- The technique involves using single images in training a CNN. After training, given a test image, the CNN will have to produce a soft maxed output across grids of locations Figure 3. The training of this network will be computationally heavy. To make training efficient the network will be trained in batches with images being scraped during training. Google Colab GPU's will most likely be used for training.

- The reach solution involves using a CNN/LSTM combination. Here a single training epoch will consist of multiple images obtained from a single 360 degree sweep of a location. The number of images collected from a single location will be a hyper-parameter. The list of images collected from a single location are processed sequentially using the CNN/LSTM logic mentioned in the related work section. The output will be a softmax across the grid of locations across the map. To make training efficient the network will be trained in batches with images being scraped during training. Google Colab GPU's will most likely be used for training. **Note: As discussed over email, due to time constraints this method (CNN/LSTM) will not be implemented in this course project. It will instead be implemented in the CSCI 5922 Neural Networks course project. For this project, the previously discussed CNN method will be implemented.**

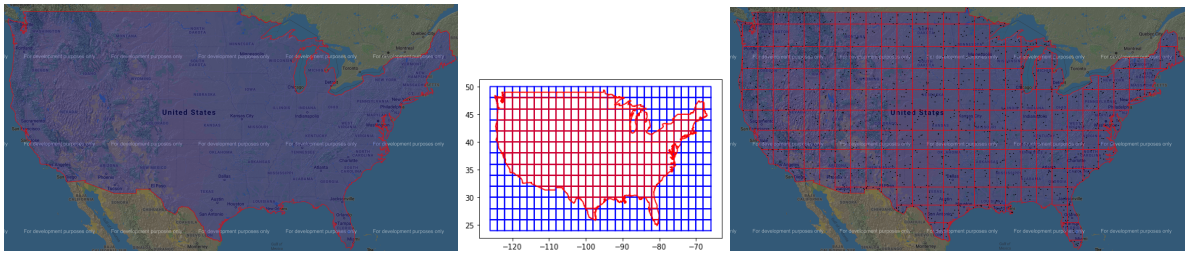

Figure 3. Left: Shape file of mainland US. Center: Fishnet used to split us into grids. Right: The map of mainland US split up into grids.

Figure 4. Sample images collected at grid 0 (Oregon).

DATA COLLECTION

As mentioned in the Proposed Solution section the following steps were taken to collect data points from across mainland USA:

- 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 [1] 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 3). The geometrical operations for this task were done using Shapely [2]. 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 3).The squares the intersected with the boundaries of the polygon were clipped at intersection points to split the polygon into grids (see right Figure 3). 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 [2].

- 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 images were saved into a file directory structured database. The structure format looked like: $data/ < grid-number > / < latitude, longitude > / < image-date.jpg >$. An example file directory structure looked like: $data/0/42.775957, -124.0667758/0 - 2009-07.jpg$. With the map being split up into 243 grids and 40 times 3 images collected per grid a total of 29160 images were collected per sweep. With each image averaging a size of 50 killo bytes, this lead to the collection of about 1.4 GB of data. At a price of 0.007 dollars per image, it cost about 204 dollars.

- The next step involved combining all images into a single to be uploaded to google Colab for training. The images were saved with names having the format of grid number followed by location followed by image number and then date. All of the data filds were separated by $+$ symbols. The image file data had the following format: $dataCombined/ < grid-number > + < latitude, longitude > + < image-date.jpg >$. A sample image file name looked like this: $dataCombined/0 + 42.775957, -124.0667758 + 0 - 2009-07.jpg$. All file names were then collected and split into train and test data. About 25000 images were used for training and the reset for testing . The train test data was stored in files named

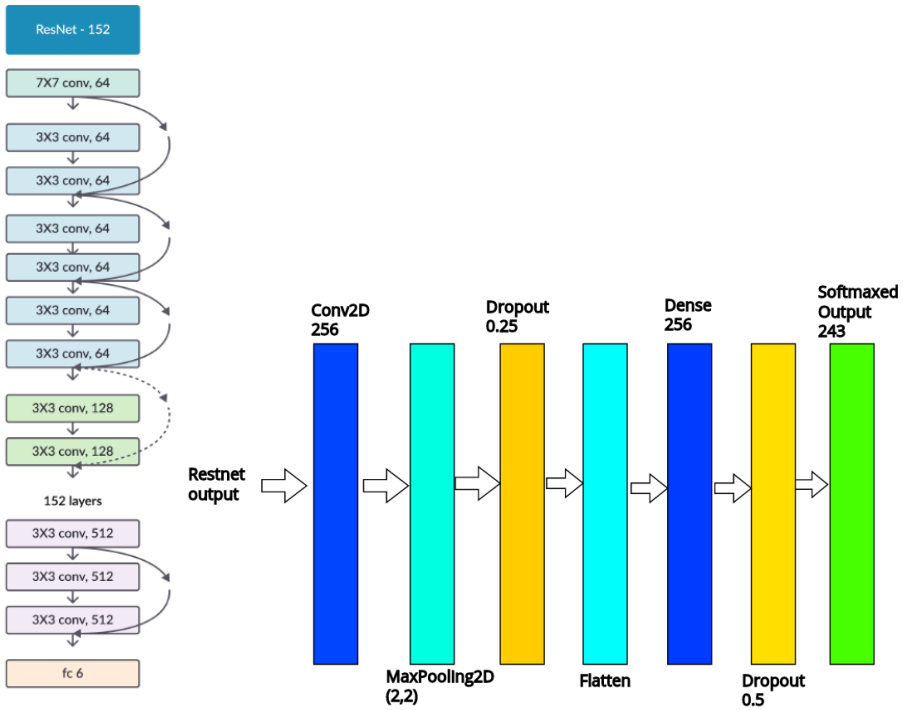

Figure 5. Left: Rest net pre-trained model. Right:MNIST model with trainable parameters

trainFiles.npy and *testFiles.npy* respectively. The 29160 data points that were collected were split into 27702 training and 1458 testing points. About 95 percent of the data was reserved for training and 5 percent for testing.

MACHINE LEARNING

Model Structure

The machine learning task at hand was to predict the grid an image from given a random image. 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 $(300, 600, 3)$. 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. The machine learning model trained on the data has two main components, the frozen pretrained RestNet model and the the trainable MNIST CNN architecture.

- The first part of the model is the ResNet [4] 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. This is an interesting question.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 5.

- For the trainable model, a multi layer convolutional neural net was used. The model used dropout between layers to prevent over fitting. The model structure can be seen in figure **??**. The model structure achieved a 99 percent accuracy with the MNIST dataset [6]. This was the reason for choosing this model to be trained. The model consists of a 2 dimensional convolutional layer that takes in an image of shape $(300, 600, 3)$ which covers the width and height of the image and the three RGB values for each pixel. This followed by a max pooling layer of pool size $(2, 2)$ followed by dropout to prevent overfilling. The output from this layer is flattened to make it compatible with softmaxed output layer. The flattened output is passed through a hidden dense layer followed by a dropout layer. The final output layer is

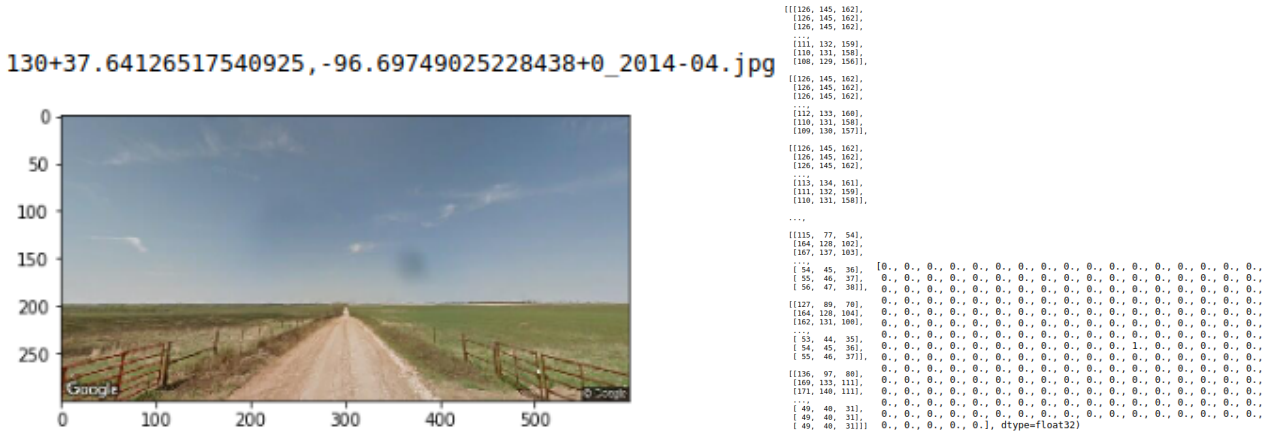

Figure 6. Left: Sample input image from grid 130 and its file name. Center: Input image converted to training input vector. Right: The output one-hot vector with position 130 set to 1.

a dense layer which produces a soft maxed output across 243 grids.

- The model is then compiled before training can take place. As the task is to predict across multiple categories with each grid being a single category, categorical cross entropy is used for the loss. Adam optimizer is used as it is know to give good results in deep networks. Categorical accuracy is the metric that is set to be tracked as the task is to predict across multiple categories with each grid being a single category. A birds eye view of the CNN structure can be seen in figure 5

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 file names to train on were obtained from the *trainFiles.npy* file that has a list of image file name designated for training as mentioned in the data collection section. The model was trained in batches using a generator function. The generator function was used to generate data in batches. A list of training image file 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 file names into batches of the specified size. The batch of file names are passed to a read Data function to read images related to the batch of file names. The $tensorflow.keras.preprocessing.image.load_img$ function is used to load specific images (whose file names look like $< grid - number > + < latitude, longitude > + < image - date.jpg >$) with a target size obtained from the built models input layer. The input image is converted to a vector using $tensorflow.keras.preprocessing.image.img_to_array$ 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 6. This generates a list of training input data which has the shape $(batchSize, 300, 600, 3)$. To generate the target outputs for training, the file names themselves are used. As mentioned in the data scraping section the file names look like $< grid - number > + < latitude, longitude > + < image - date.jpg >$ an example of which looks like $0 + 42.775957, -124.0667758 + 0 - 2009 - 07.jpg$ as seen in figure 6. 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 6. 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.

- A single batch of 1000 input output pairs were used to train a model for 20 epochs with each epoch having running for 1000 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 27 times to train across 27000 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 1000 input output pairs. This happened at each epoch to generate 27 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.

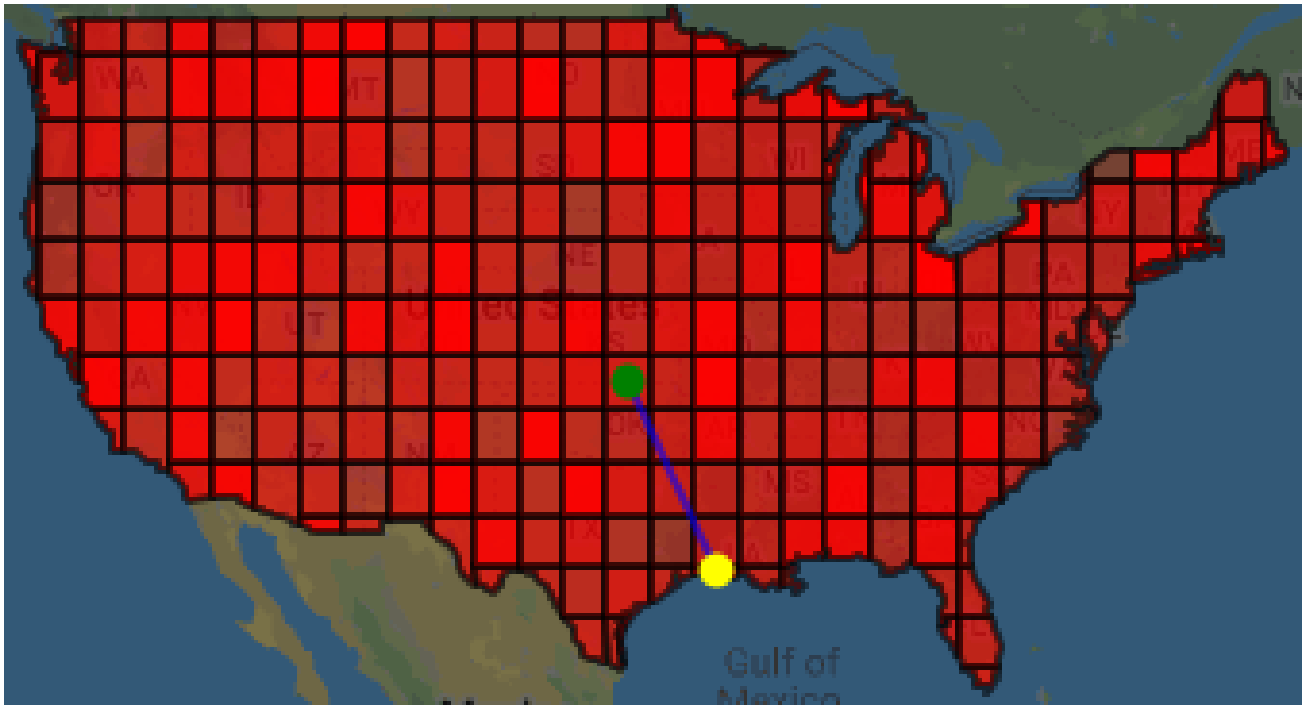

Figure 7. Predictions weights for each grid plotted using google maps API. The green dot indicates the actual grid the image came from and the yellow dot indicates the predicted grid with the highest weight. The blue line indicates the distance between the predicted and actual grid

Prediction

The test files are used to test the predictive capacity of the model. A single test file is sent to the read data function that loads the image related to the file along with the expected output grid number to create an input, expected output pair. The input image vector of shape $(300, 600, 3)$ 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. This score is the models confidence that the image came from the grid of that number.

To properly visualize this array of prediction scores, the python google maps API gmaps [5] 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 7 as different hues 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 7 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 7 as the blue line.

EVALUATION

The list of test files are used to evaluate the model. The test files are sent to the read data function that loads the image vectors related to the file along with the expected output grid number to create an input, expected output pair. The input image vectors of shape $(1458, 300, 600, 3)$ is feed into the trained models prediction function. The function outputs a soft maxed array of shape $(1458, 243)$. An argmax is used on each output prediction to get the number the grid with the highest prediction score or 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

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 10, 19, 2048)	23587712
conv2d (Conv2D)	(None, 8, 17, 64)	1179712
max_pooling2d (MaxPooling2D)	(None, 4, 8, 64)	0
dropout (Dropout)	(None, 4, 8, 64)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262272
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 243)	31347

Total params: 25,061,043
Trainable params: 1,473,331
Non-trainable params: 23,587,712

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 10, 19, 2048)	23587712
conv2d (Conv2D)	(None, 8, 17, 256)	4718848
max_pooling2d (MaxPooling2D)	(None, 4, 8, 256)	0
dropout (Dropout)	(None, 4, 8, 256)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 1024)	8389632
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 243)	249075

Total params: 36,945,267
Trainable params: 13,357,555
Non-trainable params: 23,587,712

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 10, 19, 2048)	23587712
conv2d (Conv2D)	(None, 8, 17, 4)	73732
max_pooling2d (MaxPooling2D)	(None, 4, 8, 4)	0
dropout (Dropout)	(None, 4, 8, 4)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 4)	516
dropout_1 (Dropout)	(None, 4)	0
dense_1 (Dense)	(None, 243)	1215

Total params: 23,663,175
Trainable params: 75,463
Non-trainable params: 23,587,712

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 10, 19, 2048)	23587712
conv2d (Conv2D)	(None, 8, 17, 256)	4718848
max_pooling2d (MaxPooling2D)	(None, 4, 8, 256)	0
dropout (Dropout)	(None, 4, 8, 256)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 243)	62451

Total params: 30,466,419
Trainable params: 6,878,707
Non-trainable params: 23,587,712

Figure 8. Left to Right: models with approximately a 1 million trainable parameter, 13 million trainable parameters, 6 million trainable parameters and 75 thousand trainable parameters.

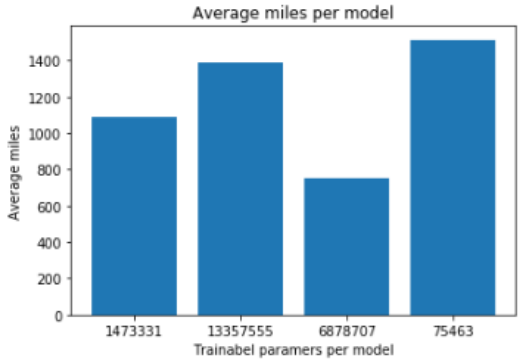

Figure 9. The graph shows the evaluation scores for each model. The x axis indicates the average distance in miles between actual and prediction grids for the list of test files.

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 7.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, four models were trained and tested using the training methods previously discussed. All the models had a similar structure consisting of a pre-trained restnet model which was frozen along with the trainable MNIST CNN. The difference between the models are the number of hidden layers in the trainable CNN. Figure 8 shows the different model structures and the number of trainable parameters. From figure 9 we see that the model with around six million trainable parameters has the least average distance between actual and predicted grids thereby achieving the best score. The model with 13 million seems to be too large and overfits the test data thereby not performing well on the training data. The model with 75 thousand training parameters seems to underfit the training therefore also not performing well in evaluations.

REFERENCES

[1] Unites States Census Bureau. 2018. Cartographic Boundary Files - Shapefile. (2018). **https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html**

[2] Sean Gillies. 2020. Shapely. (Jan. 2020). **https://pypi.org/project/Shapely/**

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

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). **http://arxiv.org/abs/1512.03385**

[5] John Kleint. 2009. googlemaps 1.0.2. **https://pypi.org/project/googlemaps/1.0.2/**. (2009).

[6] Yann LeCun. 2020. MNIST hadwritten. **http://yann.lecun.com/exdb/mnist/**. (2020).

[7] Google Maps Platform. 2020. Street View Static API. **https://developers.google.com/maps/documentation/streetview/intro**. (2020).

[8] Anton Wallén. 2013. Web-based geographic discovery game. (May 2013). **https://www.geoguessr.com/**

[9] Richard Wen. 2019. Google street view API. (2019). **https://pypi.org/project/google-streetview/**

[10] Tobias Weyand, Ilya Kostrikov, and James Philbin. 2016. PlaNet - Photo Geolocation with Convolutional Neural Networks. *CoRR* abs/1602.05314 (2016). **http://arxiv.org/abs/1602.05314**