Outperforming Humans in GeoGuessr with Deep Learning

Valdrin Nimani, Timothy Veigel, Bastian Rothenburger, Markus Fiedler
Institut für Informatik
Goethe-Universität Frankfurt am Main

Abstract

Extracting geographic information from images is a highly complex task. While the game GeoGuessr[1] has been featured in related works as a benchmark to evaluate models' performance in extracting geographic information from images, our approach is different. Rather than prioritizing location accuracy as our main objective, we focus on achieving the highest performance in GeoGuessr. Location accuracy is treated as a proxy measure of our model's ability to excel in this task.

Building upon existing literature we tried several different computer vision techniques and tested further alterations to improve our models. Thereby we found regular ResNets using an End-to-end learning approach to be the most effective model among our candidates. We also present additional positive results for the use of data augmentation and a custom loss function tailored to this task. Additionally, we found that increasing the dataset size led to significant improvements.

1 Introduction

When tasked with determining the location of an image, human accuracy can be quite inconsistent. Each person's guess may be influenced by a wide variety of cues, such as flags, signs displaying the location name, recognizable buildings, or unique flora, among others.

It is not surprising that this task is highly complex for machines, which often struggle to perform good in just one of the various aspects involved. Geolocating images is a task in the field of computer vision, in which the used model has to be able to identify many different aspects of an image to be able to achieve good results.

As many of the more complex tasks in machine learning geolocating images has a relatively short history and earlier works have had mixed results with a variety of different approaches discussed in more detail in the "Related work" section. Therefore, no single model has emerged as the most effective for this task, providing us with the opportunity to evaluate and compare a selected group of models and add to the overall body of evidence.

This problem can be framed as a classification task in which the goal is to determine the correct cell within a grid overlaid on a world map. We will use some of these rougher models as a part of our own sequential models that predict a region first and then narrow the location down to one single point. But in the case of trying to play GeoGuessr, we need the coordinates of an image and only predicting the country will often result in very bad approximations of the location.

In the game GeoGuessr you see a three-dimensional view from a point in Google's Street View. We feed the initial 360 degree view into our models but don't allow walking around. The model then has to produce a latitude and longitude from this image. From the difference between the real location and the output from the model a difference is computed which represents the distance of

[1] GeoGuessr is a game based around correctly geolocating a place in Google Street View (Geo)

the two points on a sphere (Haversine distance). The formula to calculate the score per guess can be approximated with the following equation

$$\text{score} = 5000 \cdot \exp(\frac{-\text{distance}}{2000})$$

although the real formula is not officially known and gets changed regularly. The score per guess is always in the interval $[0, 5000]$. To put this in perspective, an average human player scores an average of 2000 points per guess[2] and the top players score an average of 4000 per guess.

1.1 Motivation

The main motivation for this topic is the huge potential it still has for improvement, while also being a quite new research topic. In popular games such as chess and go, AI has already surpassed human ability, becoming unbeatable for even the most skilled players. In GeoGuessr pro players still have a significant edge against AI.

Apart from that, geolocating images may lay the groundwork for further location-based information in images. This could include geological information or information about the weather or even social science data like population density.

It can also be used in geolocating images afterwards, if the location of an image is not known anymore. While being useful for private use, this feature may be crucial in digital image forensics when this can lead to finding criminals and therefore also preventing crime. At the moment, the "Geotag" of an image is used to geolocate it - but it often does not exist. Locating an image with computer vision requires only the image and may be a powerful tool to tackle image geolocation if a Geotag is missing.

1.2 Related work

Beginning with our main inspiration for this topic: A video by the YouTuber Traversed[3]. He made an AI for GeoGuessr achieving around 3000 points per round. This corresponds to a score better than the average human player (2000) but pro players often reach scores over 4000. He describes using a CNN as his model and reached an average of roughly 3000 points using this approach. He trained his neural network purely on GeoGuessr data by using a browser extension allowing him to train directly in the browser. When we reached out to him and told him about our plans, he generously shared his own training dataset with us, enabling us to start right away.

The paper Hays and Efros [2008] sits at the foundation of most other current papers discussing this topic. With a dataset of 6 million images from Flickr, they achieved decent results with the simple method of nearest neighbor matching.

In the paper Weyand et al. [2016] the authors train a model called PlaNet on 126 million simple images mined from Flickr. Their additional test set includes further 2.3 million images. After partitioning the earth in so called "S2" cells[4] they are using a CNN with batch normalization. It is directly comparable with Im2GPS and is clearly better than its predecessor:

Table 1: Im2GPS versus PlaNet

Method	**Street 1km**	**City 25km**	**Region 200km**	**Country 750km**	**Continent 2500km**
Im2GPS (orig) 2008		12%	15%	23%	47%
Im2GPS (new) 2015	2.4%	21.9%	32.1%	25.4%	51.9%
PlaNet (900K)	0.4%	3.8%	7.6%	21.6%	43.5%
PlaNet (6.2M)	6.3%	18.1%	30%	45.6%	65.8%
PlaNet (91M)	8.4%	24.5%	37.6%	53.6%	71.3%

In a direct match against humans in GeoGuessr's challenge mode where the CNN just guesses the center-point of the classified S2 cell, PlaNet won 28 out of 50 matches with an average distance of

[2] `https://www.geoguessr.com/maps/world`

[3] `https://www.youtube.com/@TraversedTV`

[4] `https://s2geometry.io/`

1131.7km against 2320.75km. They additionally found positive results for using LSTMs for image sequence/ album localization.

In the paper Suresh et al. [2018b] different image integration techniques were explored. The dataset here only consisted of images from the USA and they only trained to correctly guess the US-state the image was taken in. Their dataset consisted out of 500,000 Google Street View images, which were evenly distributed with 10,000 images per state. This presented the idea that a evenly distributed dataset can yield better results.

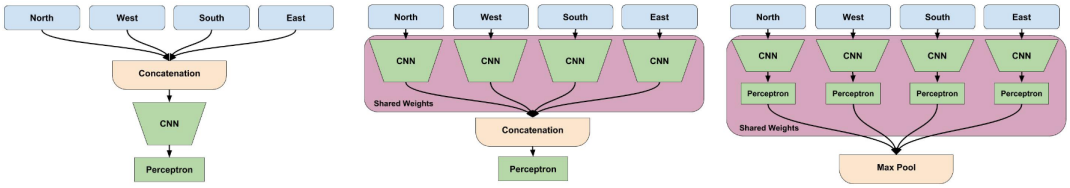

Figure 1: From left to right - Early, Medium and Late Integration(Suresh et al. [2018b]).

They used ResNets throughout all tests and evaluated a series of different image integration techniques shown in Figure 1. In the early integration scheme, the four images facing the cardinal directions are concatenated to create a 12-channel input image, which allows for information to be shared between the images. With the medium integration, a single CNN is used to extract features from each input image. These are then combined and passed through a single layer perceptron for the final classification. In the late integration scheme, they also use a shared CNN combined with an additional fully connected layer. These predictions are then integrated by taking the maximum over each output class, this way the image with the highest confidence is used to make the final prediction. The best results were achieved using the medium integration scheme. They also tested this over GeoGuessr against in humans and were able to win in 4 out of 5 rounds.

The paper Alamayreh et al. [2022] focuses just on detecting the country the image was taken in rather than the exact coordinates. They also made a new dataset called VIPPGeo for this task containing 4 million urban images they filtered to remove images containing mostly non-significant objects. With a model based on a ResNet-101 structure and applying data augmentation, they found that this attempt works better in classifying the country than predicting coordinates first and then calculate the country afterwards. These results were made by comparing their results to the ones from Muller-Budack et al. [2018] who also use a S2 cell based classification to determine the location.

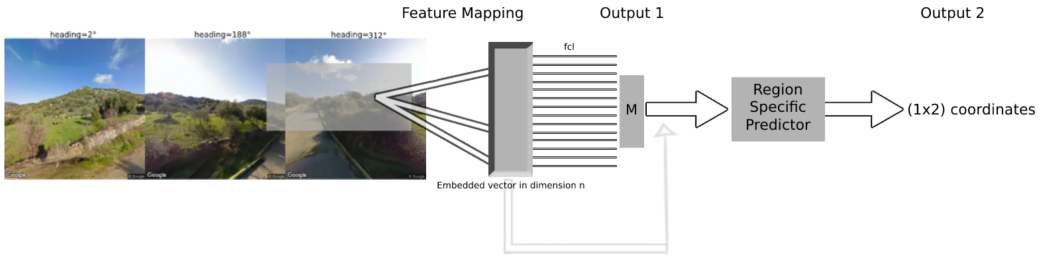

Figure 2: The architecture used in the article Salehalwer [2023].

The article Salehalwer [2023] discusses a problem very similar to ours: Inspired by GeoGuessr they made a dataset containing Google Street View images from 15 countries in Europe and trained a model on them. Their method involved overlaying a grid on each country and ensuring that their dataset had an equal number of images for each cell. By utilizing both a VGG and an Inception ResnetV2 model, they were able to achieve good results. After experiencing overfitting, they tried data augmentation and noticed a decent boost in performance.

2 Methods

2.1 Dataset Description

The success of data-driven machine learning approaches heavily relies on the availability of a large amounts of data. We were fortunate to have access to a dataset consisting of 128,000 images, which was specifically created for this task by the Youtuber "Traversed". The images were obtained by scraping them directly from the GeoGuessr website. As a result, the dataset is not publicly available, as GeoGuessr uses images from the Google Street View API, which requires a paid subscription to access. Each image in the dataset is composed of five snapshots taken from a 360 degree Street View panorama in GeoGuessr, which are concatenated horizontally. The label associated with each image is the geographic location represented by its latitude and longitude.

The images in the dataset are representative of locations all over the world, as the images have been sourced from the "world" map on GeoGuessr. It is important to note the the dataset does not include images from regions with limited Google Street View coverage.

2.2 Dataset Preprocessing

In order to turn the task of predicting the location of an image into a classification problem, we convert the geographic coordinates of the location, represented by latitude and longitude, into geohashes. This is a specific technique for encoding geographic locations into a base-32 alphabet character string. The process of mapping locations to geohashes involves dividing the world into a grid of cells, where each cell is represented by a unique geohash string. At geohash level 1, the world is divided into 32 cells, each defined by a single character geohash string. At geohash level 2 the level 1 cells are further divided into 32 parts, each defined by a two-character geohash string. This can be repeated up to a level of 12 to achieve the desired level of accuracy. Figure 3 visualizes how geohashes divide the world into a grid at precision levels one, two and three. We decided to use a precision of three, which resulted in a total of 32768 geohashes, each representing a unique class. Additionally, with this precision, each geohash cell has an approximate height and width of 156 kilometers. As a final step after the classification, the geohash can be converted back to geographic coordinates by returning the latitude and longitude of the center of the geohash cell. This means that if our model predicts the correct geohash label, there is at most and error of around 76 kilometers along each axis. To reduce the large number of classes, we filtered out any geohashes that did not contain samples. Since a majority of the geohashes were on locations that Google Street View did not cover, such as the oceans or uninhabited areas. After applying this processing we could reduce the number of classes by a factor of 10 from 32768 to 3139. The reasoning behind this was to keep the size of our models as small as possible. With the classification models, most of the learnable parameters are in the final fully connected output layer and by reducing the number of output neurons we can make the training faster and more efficient.

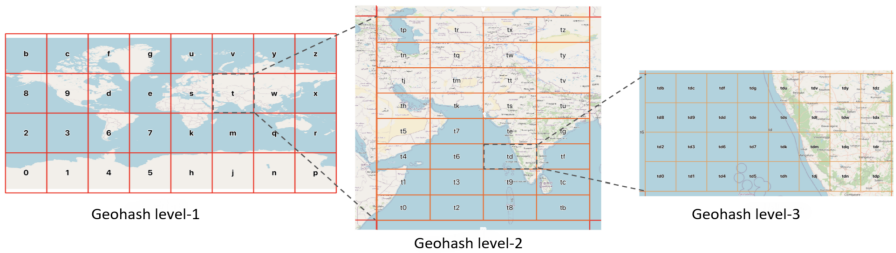

Figure 3: Result after applying geohashes at different levels of precision (Khadka and Singh).

Furthermore, we extended the original dataset by sourcing additional data from the Google Street View API. In total, this resulted in an approximate increase of 50,000 new images to our dataset. Unlike the images from the original dataset, these newly acquired images were composed of only four snapshots taken from the main cardinal directions.

We also incorporated continent labels into our samples, which were necessary for our sequential model. This model will be discussed in more detail in the following section 2.6.

2.3 Implementation Details

All models have been implemented using pytorch. We used Adam optimizer with a learning rate of 0.0001 and all other values on default. Furthermore we applied a learning rate scheduler that decreased the learning rate by a factor of 0.5 every 6 epochs. Models have been trained for 14 epochs using a 90/10 training to validation split. The model parameters which achieved the highest validation accuracy during training were then tested within the game. The images were resized from 512 x 2560 pixels to 250 x 1000 pixels. The average training time for a ResNet50 on the initial dataset was 8 hours on a NVIDIA® RTX™ A4500 GPU enabling a maximum batch size of 16 for the given image size (ResNet50).

2.4 Regression

The regression model represents the most trivial solution to this problem: Feed in the image and get out the two coordinates. Early small models made apparent that normal loss functions can't reach its full potential here, as the score depends on the differing distance across a sphere. Therefore a haversine-based loss function was implemented to improve learning, explained in greater detail in section 2.8. Furthermore, the small models quickly converged to a single point, which we believed was due to the complexity of the problem in comparison to the simplicity of the test model.

Hence we tested a pretrained ResNet18 with modified output layer for the two output coordinates and the haversine loss function. After initial predictions showed high variance, the model suffered from extreme overfitting. On the test set an average distance of 2500 km was the best result while the training set reached 1000 km. Also the model converged to one single "average" output guess. As other models appeared better suited for this task, we abandoned the regression model in favor of better performing models.

2.5 End-to-End

In the end-to-end approach, a single model was used to predict the geohash, and therefore the geographic location, directly from the input image. This model took the concatenated Street View images as input and passed them through a series of processing layers. The output from the final layer of the model was then transformed into a probability distribution over all of the possible geohashes using a softmax activation function. The geohash with the highest probability was then selected as the prediction and mapped back to the corresponding latitude and longitude coordinates using the geohash to latitude-longitude mapping described in the preprocessing step 2.2. This final prediction represented the predicted geographic location of the input image. While following this straight forward approach, multiple models were tested to determine a benchmark model, that other approaches were measured by. As will be further explained in Section 3.1, the ResNet50 proved to be best suiting as benchmark model.

2.6 Sequential Model

In a bid to optimize our accuracy, we devised a sequential methodology for the model construction. The methodology comprised of two distinct stages. Firstly, a "Head model" was trained on the entire dataset, utilizing the names of the continents as the labeling scheme. Secondly, we constructed seven separate models, each dedicated to a specific continent, that were trained on a subset of the data, where the subset was comprised exclusively of data originating from that specific continent. This approach allowed each model to develop a specialized proficiency in predicting coordinates within its designated region.

The head model would then receive an input image, and after applying a softmax activation function to the final layer, the maximum value would represent the prediction of the continent to which the image belonged. Upon making a prediction, the corresponding continent model would be activated to predict the ground truth geohash, and thus the coordinates.

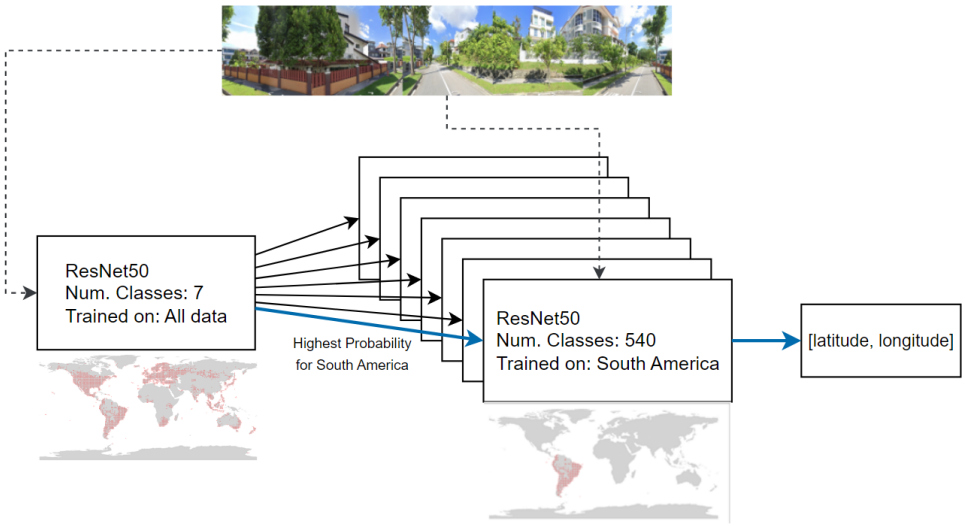

Figure 4: Input image running through the sequential architecture.

2.7 In-Game Testing

To evaluate the performance of our models in an actual in-game setting, we developed a Chrome Extension that can automatically play rounds on GeoGuessr. The extension captures five screenshots of the Google Street View panorama presented in a GeoGuessr round by turning roughly 72 degrees after each screenshot. This way the input to our model will be as close to the data on which it has been trained on. These screenshots are then sent to a locally-running server, where they are cropped to a smaller size to remove unnecessary UI elements and combined into a larger image by concatenating them along the x-axis. After the image processing steps, the image is finally passed as input to the model to make a prediction. The output from the model is then sent back to the extension, which uses it to make a guess. Upon making the guess, the GeoGuessr website provides valuable performance feedback, such as the number of points and distance in kilometers of the guess.

2.8 Haversine Loss

Essentially the loss function provides means to assess the performance of a machine learning model and then adjust the model parameters to reduce the error. In a classification problem Cross Entropy loss is commonly used and often a good place to start because it is well-suited to many binary and multi-class classification tasks. Predicting the right class (true geohash center coordinates) has the highest priority for our model as a correct prediction of the true class will correspond to the minimal distance. Cross Entropy loss only differentiates between true and false. This feature apparently leads to strong convergence to the true class. Usually this is a very positive property as in a classification approach true and false are easily differentiable. In the context of predicting the location of an image "false" is more nuanced. The prediction can be one or two clusters away from the true cluster or it can be on the other side of the globe. Consequently it might be beneficial to give our model information on how wrong i.e. how far away the predictions actually are. In theory this could especially improve all predictions which did not predict the true cluster.

In order to compute distance on the surface of a sphere between two coordinate pairs one should use the Haversine distance formula:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{1}$$

where φ_1, φ_2 are the latitude of point 1 and 2 and λ_1, λ_2 are the longitude of point 1 and 2. We then tested two approaches. Firstly, just Haversine Loss and secondly, a combined version of Cross Entropy Loss and Haversine Loss.

$$L = \frac{1}{B} \cdot \sum_{i=1}^{B} \text{Haversine}(\mathbf{C}, \mathbf{G}_i)\mathbf{Y}_i \tag{2}$$

where $\mathbf{C}$ contains latitude and longitude of all cluster center points, G_i is the ground truth latitude longitude of batch element i , Y_i is the model output ran through a softmax for batch element i and B is the batch size. The derivative for the weights for one particular class j of the output layer boils down to:

$$\nabla L_B(\mathbf{W})_j = \frac{\partial L_B}{\partial \mathbf{W}_j} = \frac{\partial}{\partial \hat{\mathbf{Y}}_j}[YD]\frac{\partial}{\partial \mathbf{s}_j}\sigma(\mathbf{s})_j \frac{\partial}{\partial \mathbf{W}_j}\mathbf{W}_j\mathbf{x}_B = D_j\sigma(\mathbf{s})_j(1-\sigma(\mathbf{s})_j)\mathbf{x}_B \tag{3}$$

with $D = \text{Haversine}(\mathbf{C}, \mathbf{G}_i)$ and σ is the softmax function.
In combination with cross entropy loss we obtain:

$$L = \frac{1}{B} \cdot \sum_{i=1}^{B} \text{Haversine}(\mathbf{C}, \mathbf{G}_i)\mathbf{Y}_i - \gamma \ln(\mathbf{y}_k) \tag{4}$$

And for the derivatives with respect to weights of the final output layer:

$$\nabla L_B(\mathbf{W})_k = \frac{\partial L_B}{\partial \mathbf{W}_k} = D_k\sigma(\mathbf{s})_k(1-\sigma(\mathbf{s})_k)\mathbf{x}_B - \gamma(\sigma(\mathbf{s})_k - 1)\mathbf{x}_B \tag{5}$$

$$\nabla L_B(\mathbf{W})_i = \frac{\partial L_B}{\partial \mathbf{W}_k} = D_k\sigma(\mathbf{s})_i(1-\sigma(\mathbf{s})_i)\mathbf{x}_B + \gamma\sigma(\mathbf{s})_i\mathbf{x}_B \tag{6}$$

where k represents the true class and i one particular wrong class.

3 Results

3.1 Validating architecture

Since the project was constrained by time and computing resources, identifying an architecture that works well but does not require extensive training time has been crucial. Possible Model candidates therefore had to archive good accuracy on the validation set as well as show convergence when trained for 14 epochs. Training models from scratch proved to be inefficient as convergence was very slow compared to using pretrained models from PyTorch (See Table 3.1) As expected, increasing the size resulted in improved performance but with diminishing marginal improvements. A pretrained ResNet50 bridged the gap between training time and accuracy. Consequently ResNet50 has been used as the benchmark model for the experiments in this project.

Table 2: Architecture performance overview

Model	Description	Best validation accuracy
GeoGuessrNet	CNN with 14 layers	6%
ResNet from Scratch	CNN with ResBlocks	12%
Resnet18	Pretrained form PyTorch	24%
Resnet50	Pretrained form PyTorch	30%
Resnet101	Pretrained form PyTorch	31%
ViT B 16	Pretrained form PyTorch	33%

3.2 Sequential

We evaluated the performance of each of the eight models using a validation set. While the Head model was able to classify nine out of ten samples correctly, we observed strong variations regarding the accuracies of the different continent models. Continents with a strong urbanization, such as North America and Europe performed worse than Africa and Oceania.

Table 3: Difference between models

Note that the Antarctica model is left out due to lack of training data.

	Samples trained on	Output cluster	Val. Accuracy	Gini coefficient
Head model	128762	7	91.1%	0.752
North America	39767	779	28%	0.743
Europe	41215	745	25.99%	0.672
South America	11365	540	31.55%	0.759
Asia	25195	734	34.73%	0.759
Africa	5745	183	51.09%	0.638
Oceania	5472	172	43.01%	0.712

Regarding accuracy, overfitting and actual In-Game Score, the sequential approach performed very similar, but did not outperform the conventional End-to-End approach. Due to the end-to end approach having a better trade-off between training time and accuracy, the sequential approach was not further explored.

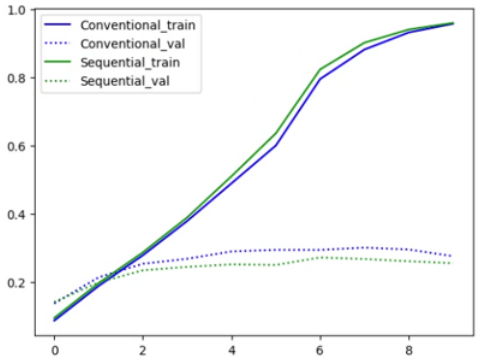

Figure 5: Performance of the sequential approach against the end to end model

3.3 Haversine

In order the determine whether using the haversine modification of cross entropy loss could uncover any value, we applied single batch overfitting to randomly selected batches. For some batches faster convergence time was observed when using the haversine distance in the loss function. Subsequently we performed a hyperparameter search on the full data set with values $1000, 5000, 10000, 15000$ for γ. 10000 showed the most promising result in terms of convergence time and best accuracy.

3.4 In game performance / Real world tests

The most promising models have been tested in the game using a browser extension. Each model played roughly 3000 games, detailed results can be found in Table 3.4. End-to-End Model and Sequential Model performed on equal footing with an average point difference of just 4 points. The transformer model performed slightly worse. Overall model performance could be improved by adding augmentations, using the proposed haversine loss and extending the dataset size with additional pictures from the Google Street View API. Applying all three methods that showed improvements simultaneously resulted in the best model with an average GeoGuessr score of 3600 points.

Table 4: In-game performance overview of the different models.

	ResNet50	Sequential ResNet50	VIT_B_16	Haversine ResNet50	Augment ResNet50	Google Data ResNet50	Haversine Augment Google ResNet50
Score Level							
0-1000	23,0%	22,5%	27,1%	20,7%	20,6%	19,4%	12,1%
1000-2000	6,5%	7,8%	5,4%	5,3%	5,6%	5,3%	5,0%
2000-3000	9,6%	9,7%	8,9%	9,1%	10,2%	8,7%	8,9%
3000-4000	16,4%	15,5%	14,8%	16,3%	15,3%	17,3%	15,2%
4000-5000	44,4%	44,5%	43,8%	48,6%	48,3%	49,4%	58,8%
Average Score	3010	3006	2900	3180	3163	3237	3600

To learn about potential problem areas, that the model was struggling to identify, we visualized the ground truth coordinates of all predictions that lead to a score lower than 1000 in the best performing model.

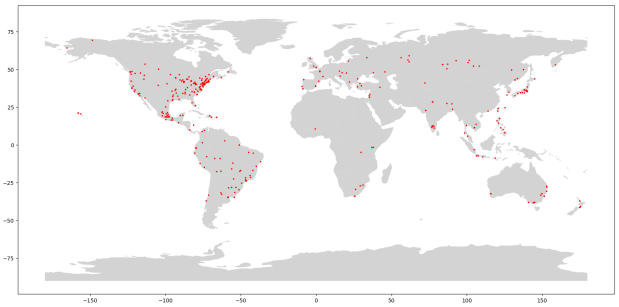

Figure 6: Ground truth coordinates of all guesses that resulted in a score lower than 1000

The plot indicates, that the model is struggling a bit more often in North America than in other areas. Apart from this, the "bad guesses" are spread out over all places that contain Google Street View data.

4 Discussion

4.1 Grad-CAM

To have a deeper understanding on how our final models come to the classifications they make, we use Grad-CAM by Selvaraju et al. [2016], which is a technique that provides visual explanations for deep learning models that use convolutional neural networks. Grad-CAM, or Gradient-weighted Class Activation Mapping, is mainly used to understand and interpret the decisions made by deep learning model for classification tasks, by showing which regions of the input image are most important for a particular classification decision. As humans, we rely on certain visual cues to gather crucial information about the geographic location of an image. For instance, we often focus on elements such as lane markings, traffic signs, vegetation, and other distinctive features to deduce the country of origin. Therefore, we want to explore what aspects the model focuses on in the image and whether they overlap with the aspects that humans tend to focus on.

Grad-CAM works by computing the gradient of the target class with respect to the feature map activations of a convolutional layer in the neural network. This gradient information is then used to generate a heatmap visualization that is overlayed on the original image to highlight the regions of the input image that are most important for identifying the target class. Regions in the image that have a large gradient indicate that these are areas of high influence on the final classification of the network, whereas low gradient areas are not as relevant.

To create the Grad-CAM maps, we use a pretrained ResNet50 architecture and analyze the activation maps from 'layer4'. In Figure 7, we have provided two examples of Grad-CAM visualizations. The top row displays the original input image, while the bottom row showcases the corresponding heatmap representation produced by the Grad-CAM. The image on the left showcases a positive example by the model, as it focuses on key elements in the image such as the street, sidewalk and cars, which provide crucial information about the geographic location of the image. It is noteworthy to mention that the model also puts emphasis on the building and pays attention to the name of the shop or business. This suggests that the model might have some understanding of language, characters, or symbols that are associated with certain locations. The right image on the other hand presents a negative example where the model seems to prioritize the sky rather than more informative features such as the street, cars, or lane markings. This observation suggests that there may be some underlying issues with the model that need to be addressed.

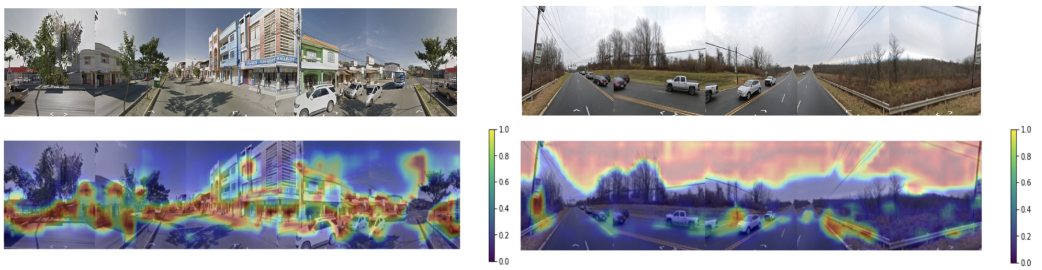

Figure 7: Two examples showing Grad-CAM results. Top: Original image, Bottom: Grad-CAM visualization

4.2 Class Imbalance

As indicated in Table 3, we observed a substantial variation in the performance of the single continent models, which may be attributed to geographic differences, as well as differences in group sizes and class imbalance. To gain insights into the potential impact of the latter on the accuracy, we computed the Gini-coefficient G for each model to measure the class imbalance and compared the results with the accuracy scores. The Gini-coefficient is calculated as:

$$G = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} |p_i - p_j|$$

where N is the number of classes and p_i is the proportion of samples in class i. No correlation between class imbalance and accuracy was observed. On further investigation we took a closer look at the models in game performance. The ground truth coordinates of every round were encoded to geohashes and the number of training samples, that we used for the respective geohash were noted. Then a linear regression was performed, to analyze the relationship between the dependent variable $score$, denoting the in game score and the independent variable $samples$, representing the number of training samples sharing the respective geohash.

The coefficient of the $samples$ variable was 0.4040, indicating that for each additional unit of $samples$, the $score$ variable is expected to increase by 0.4040, holding all other variables constant. The R-squared value of 0.018 indicated that only 1.8% of the variance in the dependent variable could be explained by the independent variable in this model. The F-statistic of 52.29 was significant with a very low p-value ($p < 0.001$), suggesting that the model is statistically significant in predicting the dependent variable.

The regression suggests a weak positive relationship between $samples$ and $score$, but the low R-squared value suggests that among the class imbalance also other variables, such as potentially geo-specific differences play a large role in determining the accuracy. However, gaining more data for underrepresented geohashes could be one way to improve the accuracy.

4.3 Transformer

The in-game result for the transformer did not meet expectations. Despite its impressive 10% lead in validation accuracy during training, the Transformer seemed to struggle with the transfer to the real world and ultimately underperformed in GeoGuessr. The most likely explanation for this outcome is that the Transformer excels at finding the correct cluster, but falls short when it does not hit the bullseye. Interestingly the transformers behavior during training could support this thesis: It managed to over fit the training data in just 7 epochs (Training accuracy of 99%) the second best contender, the ResNet101 was not able to archive this after 14 epochs[5]. One could speculate that the over fitting generally leads to more confident guesses under uncertainty. On the one hand this explains the good results whenever the interpretation of the image is good, on the other hand it might cause terrible results whenever the interpretation of the image fails. Because there is not much research applying vision transformers to geolocation we can not conclude whether this behavior is normal or not. Further research is needed in order to determine the root cause.

4.4 Haversine

Can distance be a useful feedback for a deep learning model that is supposed to geolocate images based on classification? Our results suggest it certainly is beneficial, but the devil is in the details of the implementation. Arguably, our proposed implementation looks like and possibly also performs like a regularization terms that does not use magnitude of parameters but distance instead. Consequently, our proposed term was not able to perform as a standalone loss and had to be combined with plain cross entropy loss and a hyperparameter in order to produce any meaningful results. Initially, the distance term seemed to decrease validation accuracy when given to much weight in relation to the cross entropy signal. Consequently we had to increase the magnitude of the cross entropy loss signal. After a certain threshold the validation accuracy only increased to a level that had already been archived using only cross entropy loss. Finally the effect of haversine loss only became visible using the in-game test and specifically looking at distance. The model trained on haversine loss was able to significantly decrease the error distance for guesses that did not match the true cluster. All in all, one could conclude that the haversine distance does not provide a significant benefit regarding the predictions for the true cluster because the punishments for predicting neighbouring clusters are not that large in magnitude compared to predicting clusters that are located far away. Essentially this means we improve the overall error distance but we do not improve accuracy at the finest level of granularity.

4.5 End-to-End vs. Sequential

From a macro perspective, organizing the model sequentially is a more sophisticated approach. It uses double the amount of parameters and it could theoretically learn to differentiate features on different scales (continent level and region level with in the continent). Despite perceiving these macro factors as advantageous, our results showed that this interpretation simply does not match reality. There is almost no difference in validation accuracy and in-game performance between the two models. In theory this fact suggests that the size of the End-to-End model is sufficient to capture all differentiating features from the training data. This hypothesis could be tested by decreasing model size. We also used an occlusion window in order to visualize differences between the models. In this example we have an image from Europe. The basic idea of this method is to cover up an area of the image, feed it to a model and observe the prediction probability for the true class. To summarize these probabilities a heatmap is the tool of choice. Each square in the heatmap corresponds to exactly one occluding window position. We choose a window size of 100×400 pixels and a step size of 10 pixels.

In some instances both models produce vastly different results on the same image. Nonetheless there are instances of some kind of consensus between models as in our featured example 4.5. Both models consider the bottom left and bottom right areas important whereas the center and the top are considered unimportant. Based on this result, one can infer a list of important features: the cemetery,

[5]These results where obtained using SGD and therefore do not match the results from the comparison End-to-End vs. Sequential as these where obtained using Adam which lead to faster convergence on the training data across the board

the roads, the electricity pylon and possibly the trees on the left and right-hand border of the image. The tree in the center seems to be rather unimportant.

All in all there are a lot of occlusion window results that raise new question marks and deriving practical takeaways for improvement is not straightforward.

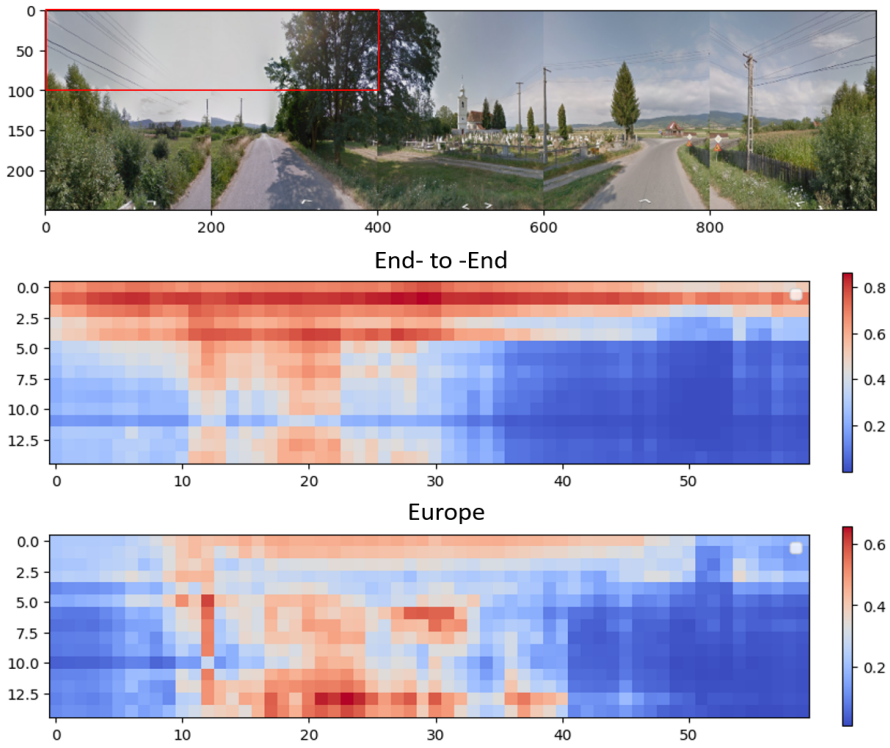

Figure 8: Occlusion window comparison: End-to-End vs. Sequential
(occlusion window size 100×400, step size 10)

Since the main goal of this project is improving the score in GeoGuessr, we did not elaborate the results of this comparison any further.

4.6 Possible extensions

Even though we were able to raise the bar for AI performance in the game of GeoGuessr there is even more untapped potential as we were not able to test all promising approaches from the related work section. The aim of this paragraph is to highlight some of these approaches and illustrate how to apply those to beating GeoGuessr. One possible extension that has also been presented in PlaNet is the use of a LSTM. In the context of GeoGuessr this would naturally be applicable to the game because you are able to walk around at the Street View location you get spawned into. The extension would then take panoramas at different positions and would pass these images to the LSTM to process them sequentially. The only caveat is that one would need a new dataset for this task. In theory one could also separate the existing panorama images and process them in a sequential manner, tough the question remains whether it is possible to extract more distinguishable features in this way than using the whole panorama image. In addition to using a LSTM one could also alter the clustering method. This could have two effects, decrease class imbalance and increase resolution in areas with a lot of samples. Even though our results contain hints that the class imbalance is not a primary factor for poor performance, we can not rule it out beyond all doubt. Therefore reducing class imbalance with the help of alternative clustering methods could shed light on this question and ultimately increase performance. Possible candidates for alternative clustering are S2 Partitioning and advanced k-means sometimes referred to as constraint k-means. The former has already been used in Weyand et al. [2016]. The latter has not been used to our knowledge. Advanced k-means could be advantageous

because it adapts the center point according to the distribution of Street View images in cell, rather than being fixed on one specific point for a certain resolution.

5 Conclusions

We used deep learning techniques in order to explore untapped potential in Geo-image-location. By reducing the scope to image data from Google Street View we found that a specialized approach is superior to a general model like PlaNet. Similar to findings of earlier approaches, treating the problem at hand as a classification proved to be very effective because it allows the model to express its uncertainty with the help of probabilities. Our Model is able to outperform most humans in GeoGuessr and the margin of pro players edge has been reduced further. Organizing the model sequentially did not show any improvements.

References

Geoguessr. https://www.geoguessr.com/. Accessed: 2023-09-30.

Omran Alamayreh, Giovanna Maria Dimitri, Jun Wang, Benedetta Tondi, and Mauro Barni. Which country is this picture from? new data and methods for dnn-based country recognition. *arXiv preprint arXiv:2209.02429*, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

James Hays and Alexei A Efros. Im2gps: estimating geographic information from a single image. In *2008 ieee conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

James Hays and Alexei A Efros. Large-scale image geolocalization. *Multimodal location estimation of videos and images*, pages 41–62, 2015.

Krishna Khadka and Rohit Singh. Polygeohasher: an optimized way to create geohashes — geospatialworld.net. https://www.geospatialworld.net/blogs/polygeohasher-an-optimized-way-to-create-geohashes/. [Accessed 13-Feb-2023].

Eric Muller-Budack, Kader Pustu-Iren, and Ralph Ewerth. Geolocation estimation of photos using a hierarchical model and scene classification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 563–579, 2018.

Salehalwer. Geoguessr-inspired exploration of cnns: Predicting street view image locations, Januar 2023. URL https://medium.com/@salehalwer/geoguessr-inspired-exploration-of-cnns-predicting-street-view-image-locations-e7aaa2dc19f5.

Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

Sudharshan Suresh, Nathaniel Chodosh, and Montiel Abello. Deepgeo: Photo localization with deep neural network, 2018a. URL https://arxiv.org/abs/1810.03077.

Sudharshan Suresh, Nathaniel Chodosh, and Montiel Abello. Deepgeo: Photo localization with deep neural network. *arXiv preprint arXiv:1810.03077*, 2018b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 37–55. Springer, 2016.

A Appendix

A.1 Information on the workload and contributions of each individual in the group

Table 5: Contributions Overview

Group Member	Contribution
Markus Fiedler	Regression Model, Haversine Loss Regression
Valdrin Nimani	Set up for training code, Browser Extension Development, In game model evaluation
Bastian Rothenburger	Set up and modifying training code for different models, Haversine Loss for Classification, supervising training
Timothy Veigel	Data Preprocessing, supervising training, Haversine Loss for Classification

A.2 Duel between human and AI

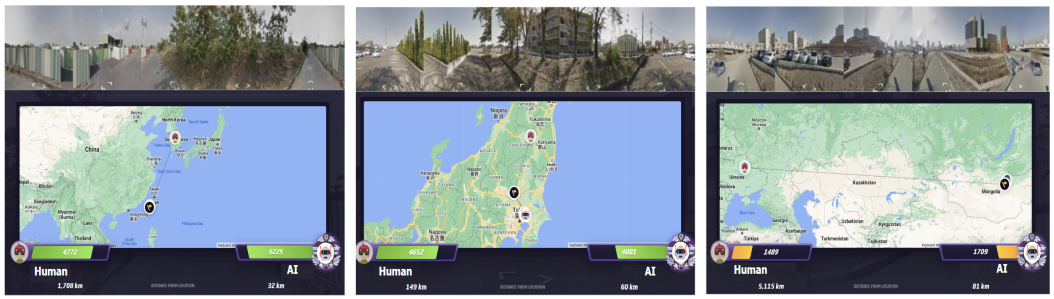

Figure 9: The results of three duel rounds in GeoGuessr between a human and the best performing model, using the chrome extension.