LUMEN Data Science

2022.

GeoGuessr AI Agent

Project documentation

Submission date: *May* 5, 2022

Contents

1. Business understanding

1.1 Challenge description

Goal of the presented challenge is to create a system which accepts an arbitrary number of images representing a particular geographical location in Croatia and then predicts and outputs the predicted coordinates. The idea for the given challenge originates from a famous online game - GeoGuessr. Premise of the game is simple - the game puts the user at a random location in the world on Google Maps' StreetView and the user is prompted to put a marker on the minimap where they think that the location is.

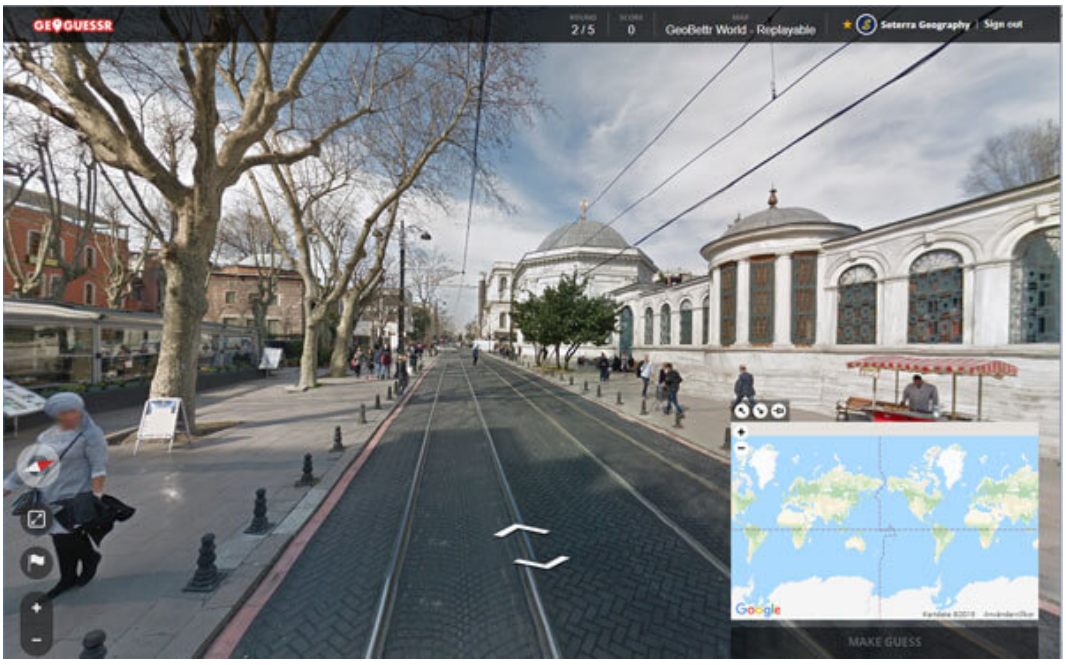


Figure 1.1: Example of a GeoGuessr game

1.2 Problem description

The most interesting part about this challenge is that an average human is not likely to perform very well on this task. Unlike a lot of classification problems

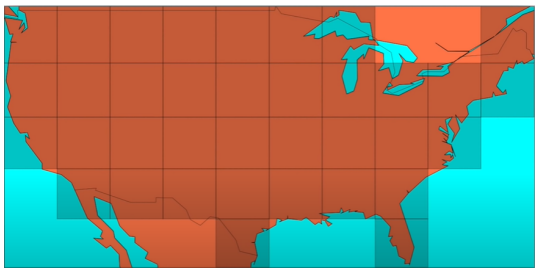
where humans excel, such as dog versus cat classification or digit classification, this challenge poses a problem that does not have a simple cause-consequence solutions. There are many factors that a human should take into account before making a decision regarding where the location is. Therefore, a solution that could perform with accuracy better than an average Croatian citizen would be deemed as very acceptable.

1.3 Potential benefits

If proven successful, GeoGuessr AI Agent could be useful to the whole GeoGuessr community, which could use it to generate coordinate predictions on their given location, although it may be deemed as cheating by some. Practical use outside of the online game domain could be in detecting where the person who submitted the images is located and may prove useful to the police in narrowing down the location of the perpetrator in specific cases of kidnapping and ransom where the perpetrator may send the video of victim to the authorities.

1.4 Existing solutions

With a quick Google search, we have found that there have been numerous attempts, some successful and some not so much, at creating a solution for a similar problem. One of the solutions that has provided a basis for our proposed solution has been described in this video by a user named *adumb*. The mentioned user has made a GeoGuessr AI Agent for The United States of America. He described how he split the map of the USA into multiple square-shaped zones and then trained the convolutional neural network to predict the probabilities of an image belonging into each one of the zones. The output of the CNN is then used to calculate a weighted sum to then output the actual coordinates of the location.



(a) Square-like zones

Game breakdown

Round 1	**240 km** (149 miles)
Round 2	**1,080 km** (671 miles)
Round 3	**902 km** (560 miles)
Round 4	**270 km** (168 miles)
Round 5	**335 km** (208 miles)
Total	**2,829.3 km** (1,758.1 miles)

(b) Results

Figure 1.2: Described existing solution

1.5 Proposed solution

We propose a solution similar to the one mentioned above with a few key notes. The .csv file will be processed in such a way that multiple columns will be added, including group label, longitude and latitude centers of the group. We will use the ensemble learning approach to train multiple CNN models with different architectures and put them to a vote so that they all have a say in the predicted zone. We will then postprocess the received results in such a manner as to shift the prediction more so towards the actual coordinates. Detailed explanation of our solution will be described in the following chapters.

1.6 Technologies and tools

We will be using Python since it has become a staple programming language for machine learning, deep learning, and artificial intelligence in general because of its numerous external libraries that enable a user-friendly approach regarding data processing, modeling, etc. More precisely, we will be using Tensorflow and Keras for modeling, Scikit-learn for splitting the dataset, and lastly, Numpy and Pandas for array and dataframe creation and manipulation. Additionally, we have used git for version control.

2. Dataset

2.1 Data description

We have been provided with the dataset containing 64,000 images from 16,000 geographical locations. Each location includes 4 images that are taken from the point of geographical location with angle difference of 90 degrees.

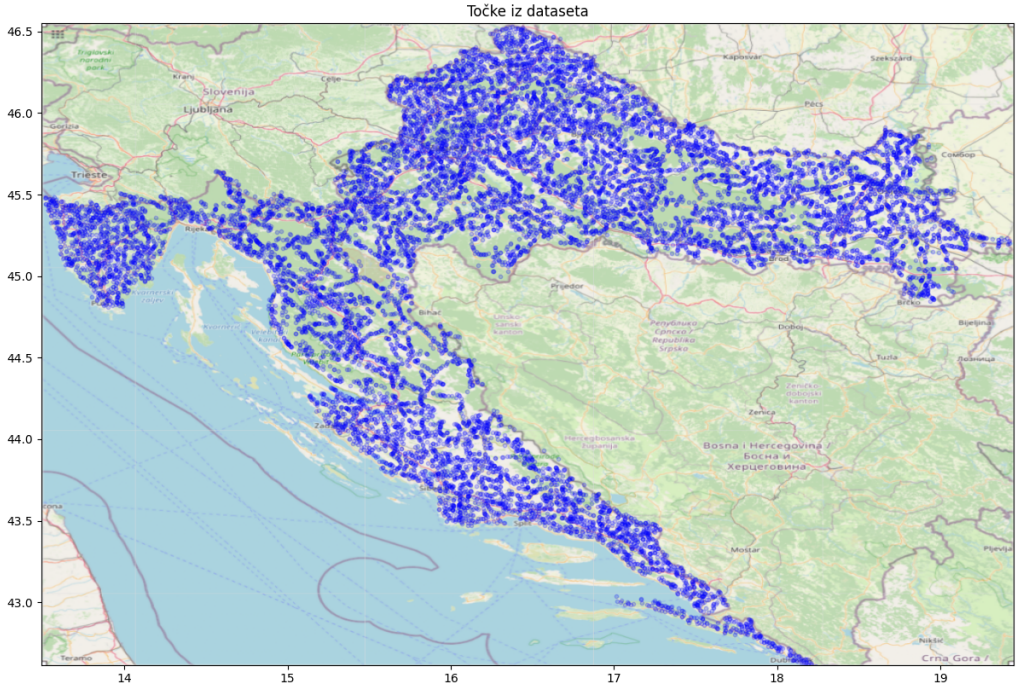


Figure 2.1: Visualization of given data on the map of Croatia

Images are provided in RGB JPG format with dimensions equal to 640×640. The dataset includes image folders and a CSV document which is composed of 3 columns: UUID, longitude, latitude. The image folders each contain 4 images. Connection between locations from CSV document and image subfolders is established by naming them with corresponding UUIDs.

(a) 0° (b) 90° (c) 180° (d) 270°

Figure 2.2: Example images from a random folder

Feature name	**Description**
UUID	location identifier
longitude	distance east or west of the prime meridian
latitude	distance north or south of the equator

Table 2.1: CSV feature descriptions

UUID	**Longitude**	**Latitude**
69387a76-b6f6-4a76-9d82-59367e14cb12	45.55222786237915	18.53839695354916
83fd0354-8781-4325-9139-653ba0ce718f	45.11632629078026	14.82181715265881
5e2f692d-a2e6-45b1-b18b-3cec90b31b64	45.42498633931014	18.76785331863612

Table 2.2: Example rows from the CSV

2.2 Feature engineering

With regards to decision to use the classification approach, data should be adequately preprocessed. We propose creating a new column which would represent the zone/group in which a specific location belongs. The groups can take on integer values which act as identifiers for the given group. Besides adding the group column, we have added two more columns which represent the longitude and latitude center of a particular group, calculated as a mean of all the longitudes and latitudes of the locations residing in that group.

Feature name	Description
group	group identifier in an N×N grid
longitude_center	mean of all the longitudes of the locations residing in the group
latitude	mean of all the latitudes of the locations residing in the grou

Table 2.3: New feature descriptions

2.3 Data cleaning

Going through the provided images by hand, we noticed that a significant amount of them contained interiors, backyards, as well as greatly distorted and warped images. Going by the assumption that GeoGuessr is only using images from the Google StreetView, we decided to remove them from the dataset, even though we knew it would be a time consuming job.

(a) Backyard

(b) Warped

(c) Humans

(d) Interior (?)

(e) Drone

Figure 2.3: Examples of undesirable images for the model

Fortunately for us, the images contained small textual labels in the bottom right corner displaying the information of the image source. Most of the time, images that were not taken from Google were incurring significant loss to the model, therefore we decided to automatically remove them using Optical Character Recognition (OCR).
OCR is an algorithm used for detecting characters in images and turning them into text. Such a task is not always trivial, since for the OCR to work as expected, extra processing to the images had to be applied, such as resizing, grayscaling, dilation and erosion.

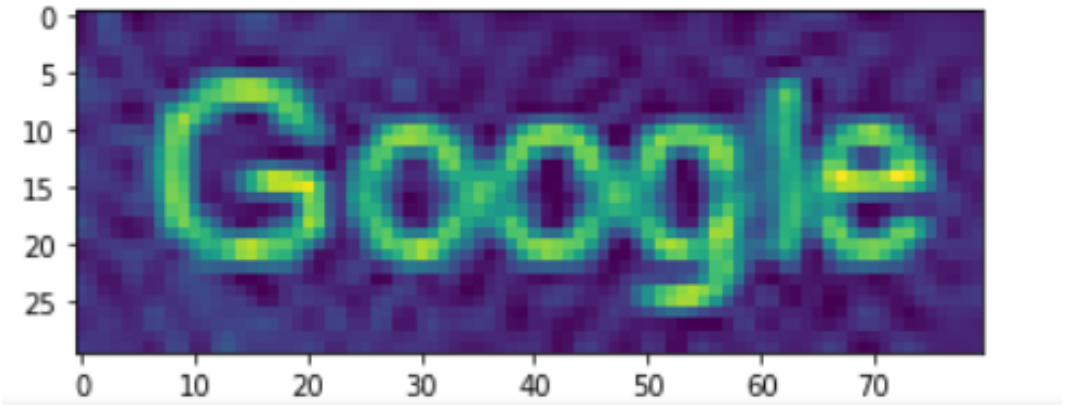


Figure 2.4: Visualization of image transformations preceding OCR

Zooming in on the image data representation on the map of Croatia, we can see that some of the points are located a bit outside of Croatia. We decided not to remove those, since they shouldn't affect the final prediction all that much.

2.4 Balancing the dataset

The data we have prepared so far is still not ready to be used for training the model. In a situation where the data is randomly split on training data and validation/test data, we would have no control over which images have been chosen for each subset. That could pose a problem because the model would then receive an unbalanced number of classes, so it would probably perform very well on images from a certain class, but not so much on the others. To mitigate that, we should balance the dataset in a way that has all the classes equally distributed.

Each image from each group is duplicated until that group has reached a predetermined target value representing the maximum number of images that a group can hold.

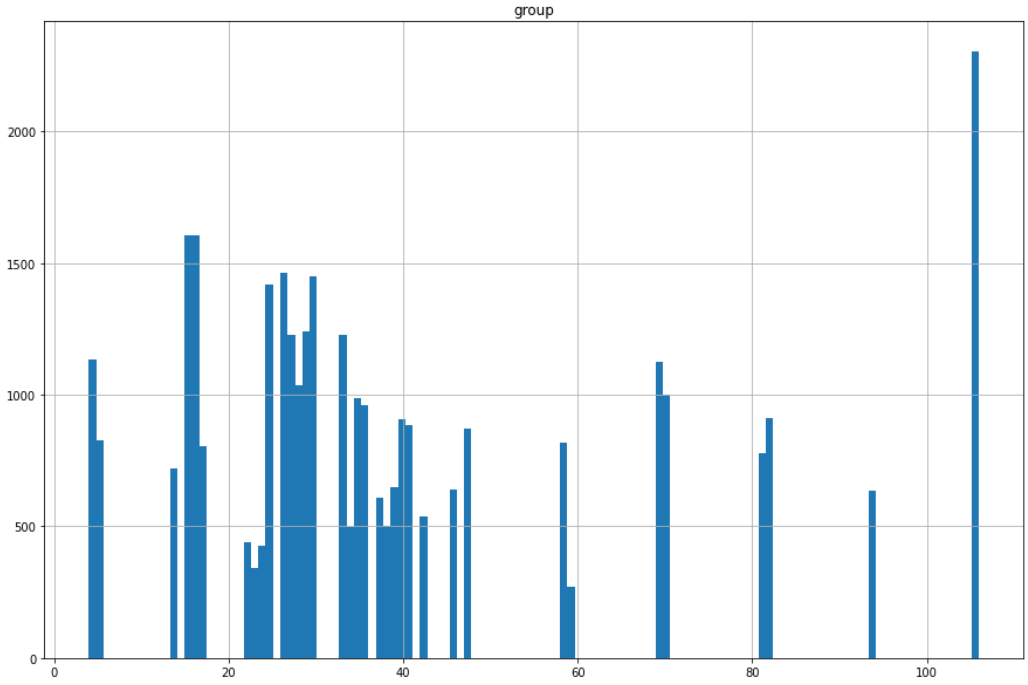


Figure 2.5: Before balancing

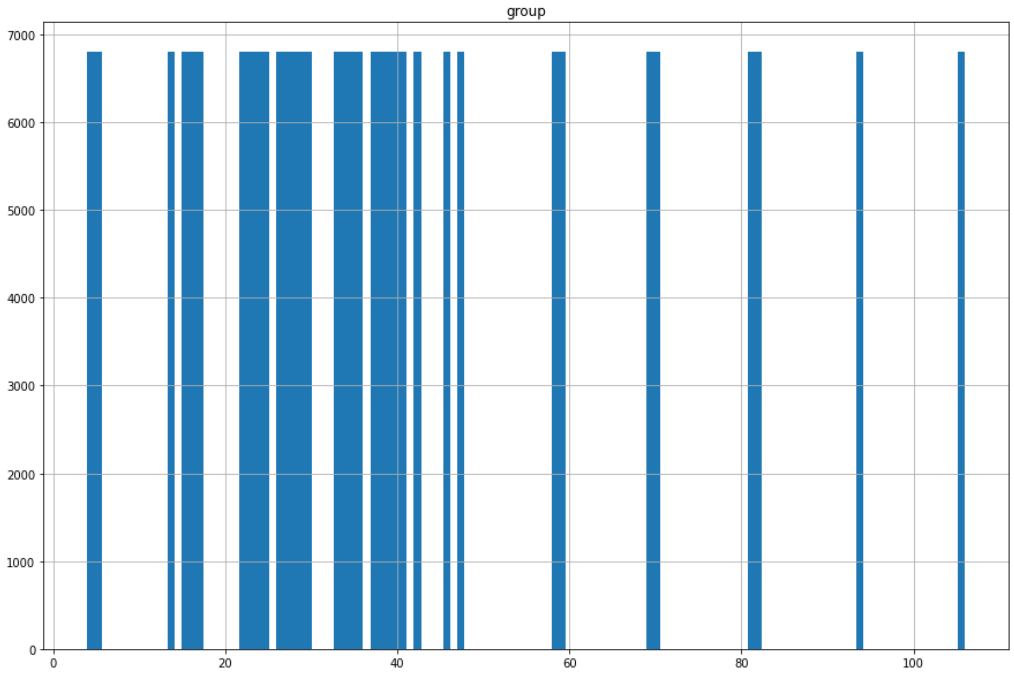


Figure 2.6: After balancing

The question arises - won't the model overfit because of the newly generated duplicate images? Yes, but more about that in the next section.

2.5 Image augmentation

It goes without saying that the more data you have for model training, the better it can possibly be. Arguably, 16,000 locations, translated into 64,000 images, isn't all that much, so we used image augmentation to apply multiple geometric transformations to the images in the training data. Some of the mentioned transformations that we have used are: zooming, flipping, rotation, shear, height and width shifting.

Figure 2.7: Original image

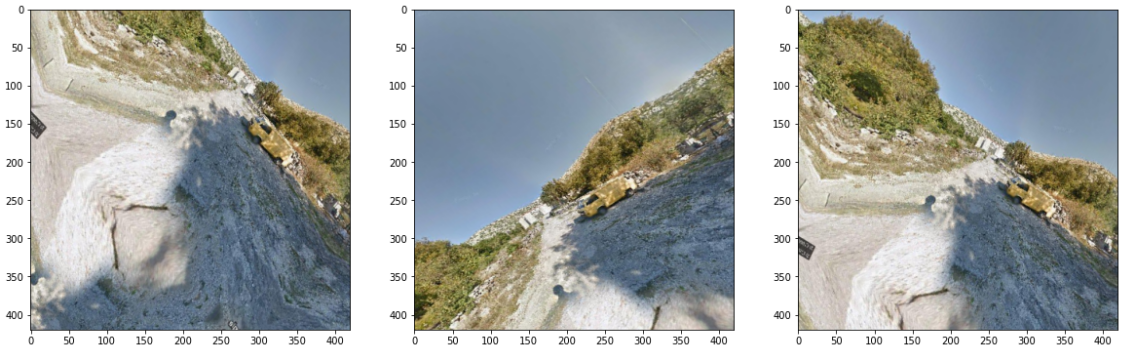


Figure 2.8: Multiple variants of the image after augmentation

Such randomly applied transformations make sure that the model never sees the same image twice during training, since it will always be a bit different than any it has ever received. That is also the solution the the overfitting problem posed

in the section above - even though every image is duplicated, they will always be different because of the applied transformations.

3. Model design

3.1 Convolutional Neural Networks

We will be using Convolutional Neural Networks (CNN) to try to successfully predict the corresponding group of the input images. We will be testing three different CNN architectures - EfficientNetB0, EfficientNetB2 and MobileNetV3Large. Each one of the mentioned CNNs has been pretrained on ImageNet. Additionally, EfficientNetB0 and MobileNetV3Large need the dimensions of images to be scaled to 224x224, while EfficientNetB2 accepts images with the dimensions as 260x260. This fact allows us to practically halve the size of the original images, allowing for faster training and larger batch size. Regarding the fully-connected layers, the original top layer has not been included because we will be using a different number of output classes to classify images into and also because of the incompatibility between using pretrained weights and a custom number of outputs.

3.2 Splitting the dataset

To somehow be able to determine models' performance on the given data, we will be splitting the dataset into training, validation and test data, which will have 56.75%, 25% and 18.75% respectively.

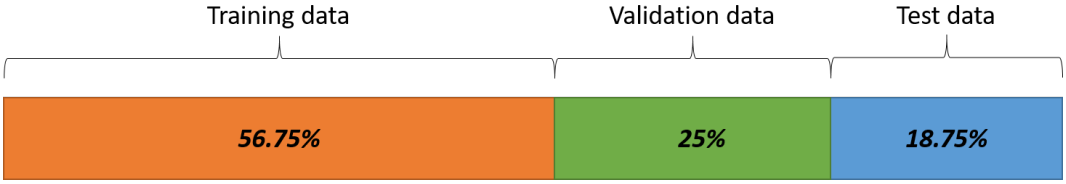


Figure 3.1: Splitting the dataset into train, validation and test data

By splitting the original dataset into 75%/25% for training and validation data, respectively, we then proceed to split the remaining training data into 75%/25% to

achieve train-validation-test split.

3.3 Ensemble Transfer Learning

Using multiple models to obtain better predictive performance is a concept summarized within Ensemble Learning. Even though one model may prove to be quite successful on the given task, using different and well-trained models might in some cases improve the overall performance of the solution. More specifically, since we will consider using EfficientNets pretrained on ImageNet as two of the models incorporated in the ensemble, we should label the final algorithm as Ensemble Transfer Learning. What we will be doing is ”fine-tuning” the EfficientNets on the given dataset to therefore increase their predictive power on the task at hand.

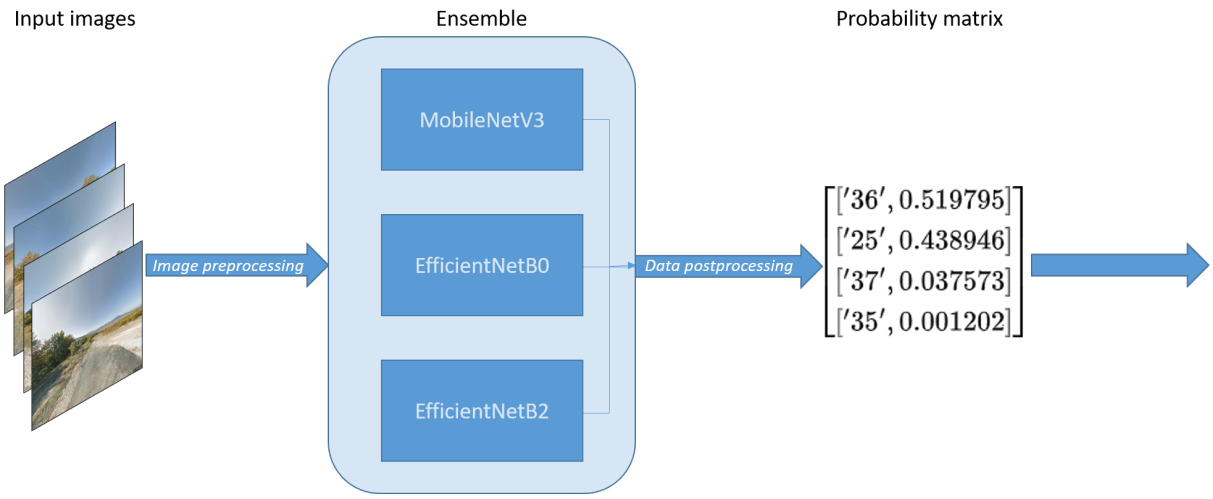


Figure 3.2: Simplified diagram of ensemble inference in the proposed solution

3.4 Training

Lack of computational resources has made us look into alternative ways of training the models. One such way is by using cloud services such as Google Colab, Kaggle and Genesis Cloud, which we have used as a primary training environment the whole duration of this competition. Most of the cloud services offer free training credits when a user signs up on their website, giving us the opportunity to train the model even on low-end computers.

With the exception of Genesis Cloud - Google Colab and Kaggle shut down the users' kernel after some time of inactivity. Therefore, we have not been able to train all of the models for the same number of epochs, but nevertheless every model has reached a point where it starts overfitting. Using the early stopping callback, we have managed to save the best model for each of the mentioned CNNs, and they as such will be used in the ensemble and consequentially in the inference process.

To prevent fast overfitting, we have implemented L2-regularization for our used CNNs. It has shown some improvement in pushing the validation accuracy further before reaching the ceiling point. Categorical cross-entropy loss function has been used since we have been posed with a classification problem.

3.5 Evaluation

Metric that is being monitored to assess the predictive power of the models is accuracy. Main problem with the said metric is the inability to conclude the actual accuracy of the model, which is measured as mean haversine - the mean distance calculated for each pair of data points on a sphere given their longitudes and latitudes.

While it is not the primary evaluation metric, we can still utilize it to see how often the model predicts the correct group. For example, a model with accuracy of 50% would put a given geographical data point in the correct group half the time. Unfortunately, it doesn't say much more than that. Two models with the same accuracy on this problem may have vastly different predictive powers. Where one model predicts incorrectly a group that is 300km away, the other one may predict incorrectly a group just 60km distant from the correct one.

Detailed explanation of how the data from the model is being postprocessed to then represent geographical coordinates will be described in the next chapter.
Before moving on, we will take a look at some of the data collected during the training and plot it.

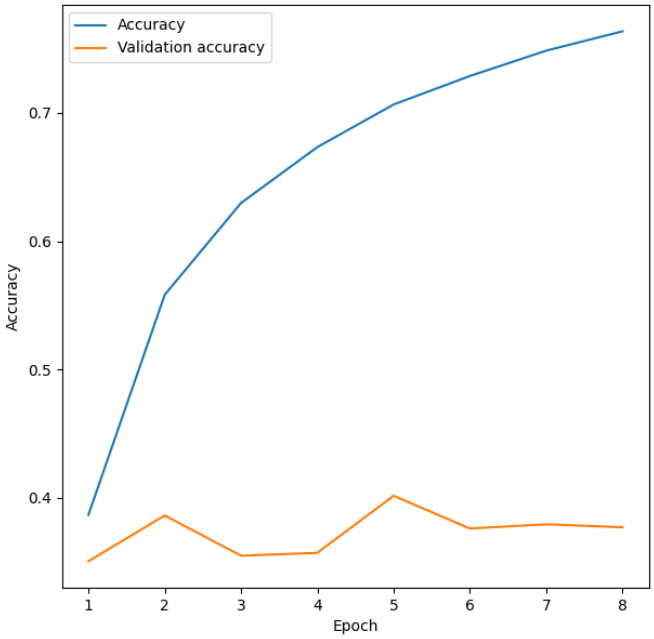


Figure 3.3: EfficientNetB0

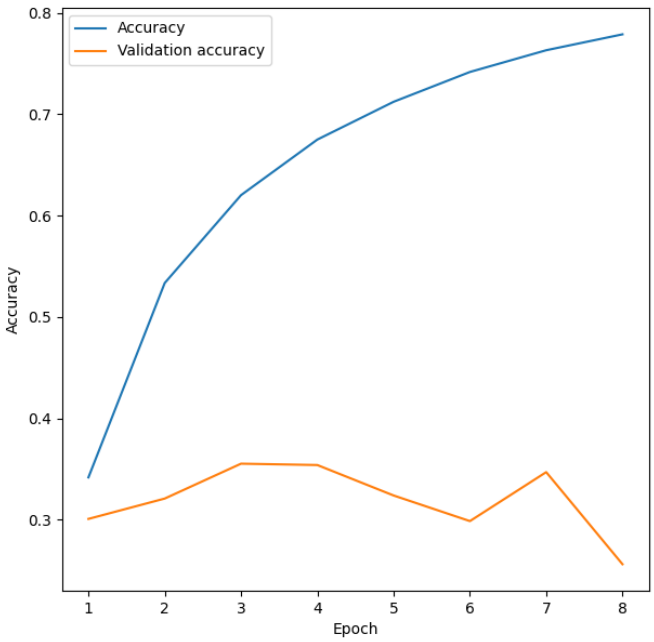


Figure 3.4: MobileNetV3Large

We can see that in both models the validation accuracy has not increased above a certain point. For the EfficientNetB0, the best accuracy has been reached by the end of the epoch 5 and amounts to 40.16%, while MobileNetV3Large has reached best accuracy after epoch 3 and amounts to 35.52%. Both models show signs of overfitting, especially MobileNet which by the end of epoch 8 plummets down to 25.59%. With all our efforts, we have not been able to totally mitigate overfitting, although it has substantially decreased after applying regularization. It should also be noted that low generalization power could originate from the task itself, since it is a very challenging one, even for the computers.
Training models in the cloud has given us a hard time efficiently collecting the training data for all of the used models, so we haven't been able to display the training progress for EfficientNetB2. Future improvements on the project may allow us to add more data to the documentation.

4. Data post-processing

4.1 Aggregation of model results

After the models output their predictions, several things are done to get as accurate geographical predictions as possible. Firstly, we aggregate the model outputs by calculating the mean probability for each group. It is possible to make a prediction based just on one image, but it's assumed that the input consists of 4 images (this number is variable so it can take on any value) and the results are further aggregated based on those different images from the same location.

4.2 Calculating the weighted sum

Each group has it's own pair of coordinates that represent it's center. To get the prediction we multiply the center coordinates of each group with the corresponding probability, but to avoid skewing the prediction, we only take into account those groups that are geographically the closest to the group with highest probability (by default, this number is 4).

4.3 Using Google Street View API

Since GeoGuessr is a game based on Google Street View, we can safely assume that every inputed image will be from the location that can be found on GSV. Since our model will, more often than not, predict locations that are not on GSV and sometimes not even on a road, we can further improve it's performance by utilising GSV API which allows us to input predicted coordinates and as a response get the closest corresponding coordinates that are reachable on GSV. Example result of this process is shown in the figure below.

(a) Prediction before using GSV API (b) Prediction after using GSV API

Figure 4.1: Example images of adjusted prediction

5. Future improvements

Due to time limitations, we haven't been able to try and test out every possible improvement for the AI Agent. Therefore, we have compiled a list of ideas and concepts that would be worth a shot if the project development is to be continued.

5.1 ArcFace

Additive Angular Margin Loss (ArcFace) is a loss function that has recently been used more often in face recognition tasks. It effectively replaces softmax to further improve the discriminative power of face recognition models and stabilize the training process.
We have first found about it from the Google Landmark Recognition 2020 competition on Kaggle.
In the future, it would be interesting to replace the softmax layer with ArcFace layer to then see how it would affect the model's predictive power. There are already multiple existing implementations for Tensorflow/Keras so it shouldn't be difficult to set it up.

5.2 Training more models

The idea behind using Ensemble learning is sound, but it might lead to some improvements in the mean haversine if there were more models in the ensemble. Moreover, models with different architectures may prove to be better at classifying some images in their correct group.
Common technique we have also seen is training multiple variants of the EfficientNet, so the images are passed through the similar architecture, but with different dimensions. It has been proven useful in some cases such as the competition mentioned above, although it leads to extended computational resource uses which is the primary reason why we haven't been able to train more models.

5.3 Multioptimizer

One thing that has generally been proven useful when doing Transfer learning is setting different learning rates for different layers. Simple way to set it up is by using MultiOptimizer, which is also available inside the Tensorflow package. Having said that, each optimizer will optimize only the weights associated with its paired layer.
It is a feature we would definitely consider implementing in the future, as multiple articles have already talked about its benefits in their research.

6. Conclusion

Tackling this given problem is not by any means a small feat. It incorporates doing all the necessary steps a data scientist should take into consideration when given such a task. And even though the success is a very subjective and loosely defined concept in data science, we conclude that this project has been very successful. Our AI Agent definitely outperforms us, the developers, at the same task, which was a measure of success that we have defined in the beginning of the project. Retrospectively, this project could be characterized as one of the best learning experiences a future data scientist (us) could have. We wanted to prove our knowledge on this task, and in doing so we have researched and learned many things that we wouldn't nearly have been able to tackle in our standard college classes. In the beginning, we had a wrong approach. We worried too much about the modeling to the point that we forgot the main thing - data. After things have been set on the right course, many offline and online all-nighters have followed. Be it at someone's place, or via our own Discord server, we loved working with each other and contemplating about different solutions for the challenges that continuously appeared before us. Maybe one of the most important things that we have learned is how to work in a team and how to work as a team. We grasped quickly that the adequate division of tasks is absolutely necessary along with regular updates to the team about the progress of the task assigned to an individual. All of the things mentioned above have taught us how to successfully develop a project of this magnitude from start to finish, and that is an experience that will absolutely be useful to us - your future data scientists.

Resources

1. GeoGuessr, https://www.geoguessr.com/
2. AI Learns to Play GeoGuessr, adumb, https://www.youtube.com/watch?v=mM_dC1HVAQ4
3. Tensorflow documentation, https://www.tensorflow.org/api_docs
4. Keras documentation, https://keras.io/api/
5. Scikit-learn documentation, https://scikit-learn.org/stable/modules/classes.html
6. An Improvement of Data Classification Using Random Multimodel Deep Learning (RMDL), M. Heidarysafa, et. al, https://arxiv.org/ftp/arxiv/papers/1808/1808.08121.pdf
7. Latex documentation, https://www.latex-project.org/help/documentation/
8. Google Landmark Recognition 2020 competition, https://www.kaggle.com/competitions/landmark-recognition-2020/overview
9. Stackoverflow, https://stackoverflow.com/
10. Machine Learning Mastery, J. Brownlee, https://machinelearningmastery.com/
11. Kaggle, https://www.kaggle.com/
12. Genesis Cloud, https://www.genesiscloud.com/
13. Google Colab, https://colab.research.google.com/