GeoGuessr AI Agent Technical Documentation

Product Name: GeoGuessr AI Agent
Product Version: 0.1 alpha

Product Phase: Testing
Current Date: 08-05-2022

Product Overview 👀

GeoGuessr AI Agent is an app that allows a user to input an arbitrary amount of images from a specific location and then outputs geographical coordinates of the predicted location. The idea behind development was given to us as a competition task, and is an interesting way to limit-test the AI possibilities on this convoluted (no pun intended) problem.

The usage of the product is composed of 5 main steps:

- Data preprocessing
- Modeling
- Training
- Inference
- Evaluation

We will concisely describe the methods and processes used for each of the named features.

State of the product 🎯

The product in its current state should primarily be used on images taken directly from Google's StreetView API. We haven't tested it enough on arbitrary sources to confirm that it will achieve the same level of performance.

Table of Contents ⚓

TABLE OF CONTENTS

Data preprocessing 📋

CSV preprocessing

- The data to be preprocessed for model training should be provided in a format that is a `.csv` file with at least 3 features:
    - `uuid` - the identifier of the location, set as a string of characters
    - `longitude` - geographical longitude of the location
    - `latitude` - geographical latitude of the location

  If your `.csv` file meets the above criteria, it is ready to be sent through the preprocessing pipeline.

- To create classes for the classification problem, a new feature named `group` will be added in the dataframe, representing square-shaped groups aranged in a grid. The coordinates for the borders of the grid are predefined to the size of Croatia in `LATITUDE_MIN`, `LATITUDE_MAX`, `LONGITUDE_MIN` and `LONGITUDE_MAX` variables. Each entry in the dataframe is run against multiple conditions to determine to which group it belongs according to the entry's coordinates.

- Two more features will be added, `center_longitude` and `center_latitude`, which will represent the mean of the longitudes and latitudes, respectively, in a certain group.

- In addition to preprocessing the input `.csv` file, a new `group_reference.csv` file is being created and contains 3 features:
  - `center_longitude`
  - `center_latitude`
  - `group`

  This allows for easier iteration through groups and their feature retrieval.

The end result is a preprocessed input `.csv`, and a `group_reference.csv`

OCR image removal

- Detection of images in the dataset that have not been taken via Google's StreetView has been solved by using OCR (Optical Character Recognition). More specifically, using the `pytesseract` package

- The user should input a path to the folder containing subfolders with images, where the subfolders are named as the `uuid` column in the preprocessed `.csv` .

- Using `OpenCV`, the image is being cropped to the bottom right corner and converted to grayscale. Dilation and erosion are then applied to the image to enhance the OCR's text detection power.

- Results are conveniently appended to an empty dataframe with two features:
  - `image` - path to the image
  - `read` - text generated by the OCR

- Taking into account the possibility that for the same text on a different background the OCR may not give the same results, the dataframe is passed on to the `replace_with_similar` function which calculates the Levenshtein distance between the two words. If they are more than 75% similar, the second input word is being replaced with the first one.

- The dataframe is then being iterated through and all the images that do not have the `read` feature equal to `"Google"` are being removed from the dataset.

The end result is a cleaned up dataset with no warped user-uploaded images.

Merging the groups

- Groups with entry count less than 100 are being merged with their closest group into one. The threshold is set as a parameter `n` .

- Function `replace_groups` checks if the group count is less than predefined 100, and if so, it calls the function `find_nearest_group` to determine which group it should be merged with. The resulting dataframe will have less groups than the original, but between-group variance will now have dropped because of the more balanced dataset.

Train-validation-test split

- Splittin the dataset is done within `sklearn.model_selection.train_test_split` function. The data is split once into train and validation, and then train is split once again into train and test data, which leaves us with 3 dataframes.

Expanding the dataset

- Current dataframe contains in the `uuid` column the name of the subfolder where the images of a specific location are contained. To make our lives easier, the dataset is expanded with the method `expand_dataset` so that for every `uuid` , four entries with image names appended to the `uuid` are being created in a new dataframe.

- Both the train and test, and also validation set are being expanded.

Balancing the dataset

- Dataset balancing is done inside the `balance_data` method which takes in a dataframe and a target size. The whole point of it being to have the same amount of images for each group.

- To balance the dataset, the images from each group are being duplicated until the `target_size` has been reached. If the `target_size` would be exceeded in any iteration, only the leftover number of images are being duplicated, starting from the first in the dataset.

- Balancing should be done on train, test, and validation datasets.

After balancing, the dataset has a class variance of 0, since each class is being represented with the same amount of examples.

Image generators

- Using the `ImageDataGenerator` from Tensorflow we get two advantages:
  - The whole dataset doesn't have to be loaded in the memory at the beginning of the training process
  - We are able to apply multiple image transformations to the training data

- Image transformations applied are:
  - Rotation: +/- 45°
  - Width shift: +/- 0.2
  - Height shift: +/- 0.2
  - Shear: +/- 0.2
  - Horizontal flip
  - Target size: depending on the model to be used

Modeling 🤖

Getting the pretrained model

- Function `get_model` returns a pretrained model from Keras library. It is a simple and efficient way to pass which model you want via an argument as a string and have it delivered.

- All the models have the `include_top=False` to allow us to build our own top in the next step.

- For the EfficientNets, the custom noisy student pretrained weights are loaded through the `weights` argument, and the MobileNet has the pretrained weights set by default to `'imagenet'` .

Building the custom model

- Custom model takes the pretrained model as a base and then processes its output through:
  - `GlobalAveragePooling2D()`
  - `Dense(1024, activation = 'relu')`
  - `Dropout(0.5)`

- Final layer is achieved as a Dense layer with 35 neurons and a softmax activation function, a standard for classification tasks.

- Function `init_model` takes in the same arguments as `get_model` , since it is being called to get the base model.

Regularization

- Function `add_regularization` adds L2 regularization to the pretrained model to help fight overfitting

- It works by looping over all of the layers of our base model, checks if it supports regularization, and if it does, it adds it

Model compilation

- Categorical cross entropy is set to be our loss function, which is a standard loss function in classification tasks, along with ADAM optimizer which will take care of setting up the learning rate for us.

- Metric that will be measured in the training process is accuracy so that we may be able to track the model's predictive power to put the image in a correct class.

Training 🏋️

Three of the models are being trained with their corresponding train and validation data generators. Furthermore, callback that is being passed to the `fit` method as an argument is `ModelCheckpoint` which allows us to save every model with improved validation accuracy and ignore the others. This is done to get the best model for the inference process.

Inference 📈

Getting the predictions

- Two test data generators are being created, one for the `EfficientNetB0` and `MobileNetV3Large`, and one for the `EfficientNetB2`, since the input shape of the images is different.

- Because of the difference in image shapes, we have found out by testing that for the B0 and MobileNet, increasing the `batch_size` to 64 was possible and didn't create any issues.

- The generated predictions for all three models are then being summed and divided by 3 (getting the mean of the predictions).

Evaluation 💡

- To evaluate the performance of our models, we use our `great_circle` function which, as its name suggests, uses a mathematical formula for calculating Great-circle distance between two points

- This function is called for every prediction made and mean Great-circle error is calculated

---

Deployment 🏭

- The trained ensemble is embedded into the backend written in Flask, and ready to be deployed online.

- With a simple `curl` command while the server is up and running, user can get their predictions almost instantaneously. One such command example is: `curl -X POST -F 'files=@/home/user/image.jpg' localhost:5000/predict` . Users can also upload multiple images, but it should be noted that they should all be contained in the `files` variable.

- The results are output as a JSON file with predicted longitude and latitude values.

Additional Resources 👩🏼‍💻

Tensorflow: https://www.tensorflow.org/api_docs/python/tf
Keras: https://keras.io/api/
Flask: https://flask.palletsprojects.com/en/2.1.x/
Scikit-learn: https://scikit-learn.org/stable/modules/classes.html
OpenCV: https://docs.opencv.org/
Numpy: https://numpy.org/doc/stable/
Python: https://docs.python.org/3/
Pandas: https://pandas.pydata.org/docs/reference/index.html
pytesseract: https://pypi.org/project/pytesseract/
Levenshtein: https://pypi.org/project/Levenshtein/
Matplotlib: https://matplotlib.org/stable/index.html
Seaborn: https://seaborn.pydata.org/api.html