Lost & Found: Predicting Locations from Images

Team name: ExampleTeam

Group Members
Linus Schlumberger
Lukas Stöckli
Yutaro Sigrist

Table of content

Business Understanding

This project explores the development of an Image Classification model, focusing on simple street-view images grouped by countries to predict the country where an image was taken. Given limited prior experience with Image Classification, this initiative aims to enhance understanding and skills in this domain. The first objective is to create a model capable of identifying the country from a given image. Building upon this, a second model will be developed to predict the exact district of the image, providing a more precise location than just the country.

The main goal is to develop a robust Image Classification model that can serve as a foundational tool for various applications. This overarching objective supports the specific sub-goals of predicting the country

and coordinates of an image. This leads to the question: for what main purposes could an image classifier for countries or coordinates be valuable? By exploring potential applications, the project aims to demonstrate the broader utility of the developed models in real-world scenarios.

- **Helping find missing persons**: Our solution can help find where missing people might be by analyzing pictures shared publicly. The emotional impact of helping reunite families or providing important clues is huge. Especially when the model will be used in addition to the search process for the police. For missing people, every second counts after a kidnapping, especially when the search is international.
- **Rediscovering memories and family history**: Have you ever come across an old image of someone close to you? Maybe of a deceased family member or someone who may just not remember where it was taken. Our model can try to predict the rough location to help you rediscover your past.
- **Supporting humanitarian action:** In disaster situations, it could help to quickly identify the most affected areas by analyzing current images from social media or aid organizations. This would improve the coordination of rescue and relief efforts and offer hope and support to those impacted.
- **Discovering new travel destinations:** Have you ever encountered stunning images of places on Instagram or other social media platforms and wondered where they were taken? Our image classifier can help you with that. By analyzing the image, our classifier can identify the location and provide you with the information you need to plan your next visit to this amazing place. This way, you can discover new and exciting travel destinations that you may have never known about before.
- **Classification as a service**: With this service, we will help other companies or data science projects label their data. Sometimes companies want to block, permit, or deploy individual versions of their applications in different countries. Some countries have more restrictions for deploying applications, therefore the image predictor can help the companies have the right version on the right devices for these countries.

Literature review

Common approaches

Follows...

Common architectures

Follows...

Possible Sources

EfficientNet Rethinking Model Scaling for CNNs
MobileNetV2: Inverted Residuals and Linear Bottlenecks
Deep Residual Learning for Image Recognition / ResNet
Efficient ResNets: Residual Network Design
Pigeon for calculating loss

Data Augmentation and overfitting
Data Augmentation in Training CNNs

Requirements (2-3 pages with a minimum of 7 sources):

1. **Foundation of Knowledge:** A literature review establishes the theoretical foundation and current state of research in the field. By requiring a minimum number of sources, students are encouraged to engage deeply with existing literature, ensuring a comprehensive understanding of the subject.
2. **Critical Thinking and Contextualization:** Analyzing and synthesizing various sources enhances critical thinking skills. It allows students to understand different perspectives and place their work within the broader context of the field.

Data collection

Data source

When it comes to relatively uniform street imagery, there are not many sources. Google Street View <-LINK> being by far the biggest. But instead of sourcing our images directly from Google, we wanted to have a more representative distribution, as well as a more interactive demonstration.

For this reason we instead opted for the online Geography game called Geoguessr <-LINK>. This has the advantage of not manually having to source where there is coverage, at what density and decide on a distribution. The game revolves around being "dropped" into a random location on Google Street View, and having to guess where it is located.

<-POTENTIALLY INSERT GEOGUESSR PICTURE>

Originally the player is allowed to move around, but there are modified modes to create harder difficulties which prevent the moving or even the panning of the camera, which is what we'll be opting for. This will also allow it to generalize more to other static pictures than if we were using the 360° spheres.

<-POTENTIALLY INSERT SAMPLE PICTURES>

Because different countries are of different sizes, but also have different amounts of Google Street View coverage, deciding on a representative distribution for generalization would be very difficult. Instead, we opted to play the Geoguessr multiplayer game mode called "Battle Royale: Countries" <-LINK>. This game mode revolves around trying to guess the country of a location before the opponents do. It has a much more even distribution of countries, while still taking into account the densities of different places.

<-INSERT MULTIPLAYER GRAPH>

Unfortunately, data collection using a multiplayer game mode is quite slow, as even though we do not need to guess and can spectate the rest of the game, we still need to wait for the other players to guess every round. The number of concurrent games was also be limited by the number of currently active players. Additionally, while spectating it is not easily possible to get the exact coordinates of a location, restricting us to only predicting the correct countries. Lastly, we were detected by their anti-cheating software as the automation environment is injecting scripts into the website.

Instead, we chose to collect data through the most popular singleplayer game mode called "World" ("Classic Maps"), by putting in arbitrary guesses and playing a lot of rounds. This allowed us to collect data a lot quicker, as well as also collecting the coordinates, however, it came at the cost of a very skewed distribution.

<-INSERT SINGLEPLAYER GRAPH>

To remedy this, we instead use the country distribution of our multiplayer games and apply it to our collected singleplayer data. This leaves a lot of data unused and forces us to remove very rare countries, but it allows us to get the required amount of data a lot quicker.

<-POTENTIALLY INSERT MAPPED SINGLEPLAYER GRAPH>

Web scraping

To collect this data we built our own scraper, utilizing the testing and browser automation framework "Playwright" <-LINK>. We then deployed 5 parallel instances of this script to a server and periodically retrieved the newly collect data.

Our script starts off by logging and and storing the cookies for further sessions, it then accepts the cookie conditions and attempts to start a game. We do this by navigating the page using the text, as there are no stable identifiers. For multiplayer it additionally checks for rate-limiting or if it joined the same game as another instance of the script, it those cases it waits for a certain amount of time and attempts the same again.

After a game started it will wait for a round to start, wait for the image to load, hide all other elements on the page and move the mouse cursor out of the way and take a screenshot. For singleplayer it then guesses a random location while in multiplayer it waits for the round to end, spectating the rest of the game afterwards. At the end of each round the coordinates or in the case of multiplayer the country are read from the page and saved to a file. Both these files are named after the "game id" we extract from the URL, preventing duplicates. This is the repeated until the script is stopped.

<-POTENTIALLY INSERT SCRAPING CONTROL FLOW GRAPH>

Initially we had a lot of issues with stability, especially with our parallelized workers. After we got rid of hardware bottlenecks we also looked to eliminate as many fixed waits as possible, replacing them wait dynamic ones to avoid timing issues. Finally, we made sure to enable auto-restarting and added a lot of other measures to completely restart after our environment stops working, which can happen during extended scraping sessions. We then let this script run in parallel, non-stop for multiple weeks, collecting <-INSERT FIGURE> multiplayer datapoints and <-INSERT FIGURE> singleplayer datapoints.

To make sure our data is collected correctly, we manually inspected it periodically. Any faults we noticed in the images like black screens and blurring, we would address later in our filtering. However, we also had to inspect whether the coordinates and countries were accurate.

(After an initial run of our singleplayer script, we noticed that the way we collected coordinates in multiplayer did no longer work and had been collecting incorrect coordinates for tens of thousands of images. To address this, we built an additional script looking up the correct coordinates using the "game id", this was a lot quicker than the collection of new data, allowing us to correct the mistake quite quickly. We also then used this new way of looking up coordinates for our collection script.)

No Requirements, it is good to show what we did to achieve our goal with data. Maybe also ask Umberto if we should include it.

Data processing

Resizing of the images

We can't train the classifier using images in a high resolution, because our resources are limited, and also often images (like from missing persons) are also very low quality. So we decided to reduce the resolution, at the beginning of the processing, about the 1/4 of the original resolution of 1280p x 720p. This also helps to move the images for learning to the server or also between us and also loading takes lot less time for future processing steps.

To talk about:

Enriching (Singleplayer coordinates, Multiplayer names)

(Issues with reverse geocoding, country name matching)

Regions Enriching (Source, Mapping)

Mapping to a distribution

As mentioned in the previous section (Web scraping), our singleplayer data is skewed towards a few countries, with some countries only appearing very rarely. To address this, we are mapping our singleplayer data to the country distribution of our multiplayer data. This allows us to have a better distribution while still not having every country appear with the same frequency to account for size and coverage differences. It, however, comes with the downside of not being able to use all of our data, although some tests showed that using all of our data unmapped performed worse <-CHECK AND MENTION RESULTS>.

Unfortunately, this also doesn't allow us to include all countries as some of them do not appear often enough and would reduce the number of images we are allowed to use for other countries as well. To achieve a mapping including enough files while including as many countries as possible, we set a minimum threshold of how often a country has to appear within the singleplayer data (<-INSERT FIGURE>). Because this included too few countries, we added a slack factor (<-INSERT FIGURE>), allowing countries that could almost meet the distribution to be included as well.

Finally, we saved this as a list of file names using our "data-loader", and commit it to our repository, making our runs reproducible. We created a few different variants of the mapped list, sometimes including more countries and other times more files per country, until we found a good balance.

<-POTENTIALLY INSERT MAPPED SINGLEPLAYER GRAPH>

Filtering of data

To address issues with our scraping's inherently unstable nature, as well as the big variety of Google Street View images, we had to do some automated filtering of unsuitable data. This consisted of both filtering our images, but also the corresponding data. After filtering we again saved this as a list of file names using our "data-loader", and commit it to our repository.

To filter images we started by setting a minimum threshold of the biggest variance of color between the pictures of an image, meaning either red, green or blue has to vary by some amount. This easily filters out black screens and dark images, like the ones indoor or inside tunnels. Additionally, we added a threshold

for the variance after the laplacian kernel was applied, allowing us to filter some blurry and low quality images. We set our thresholds after doing manual sampling and some test runs.

<-INSERT SAMPLE PICTURES WITH VARIANCE>

Additionally, we realized that some rounds were in the exact same locations, so we decided to filter out duplicates by comparing the coordinates, only keeping the first image. This, as well as the image filtering, comes with the added benefit of filtering corrupted data, which would otherwise have to be handled in our training code.

Requirements:

1. **Core Competency in Data Science:** Data processing is a fundamental step in any data science project. Demonstrating this process shows the student's ability to handle and prepare data for analysis, which is a critical skill in the field.
2. **Transparency and Reproducibility:** Detailing the data processing steps ensures transparency and aids in the reproducibility of the results, which are key aspects of scientific research.

Methods

Follows...

To talk about:

Basic method (Cross-entropy, ...)

DIfferent pre-trained models

(Coordinates attempt)

Data augmentation

Regions with custom loss

Requirements:

1. **Understanding and Application:** This section allows students to demonstrate their understanding of various methodologies and their ability to apply appropriate techniques to their specific project.
2. **Rationale and Justification:** Discussing the methods used provides insight into the student's decision-making process and the rationale behind choosing specific approaches.

Model Validation

Common approaches for model evaluation

Follows...

Human baseline performance

Collection of baseline scores

To compare our model to the performance of a human classifier, we would first have to measure the performance of a similar human. To calculate this, we built a small interactive application using "Gradio" <-LINK>. It loads a random image in our downscaled resolution, though not quite as low as most of our models are trained on, and asks the user to type in the 5 most likely countries. This then allows us to calculate a reasonable Top-1, Top-3 and Top-5 accuracy for comparison with our model.

Follows...

Model performance on other datasets

Follows...

Requirements:

1. Ensuring Model Reliability: Model validation is crucial for assessing the accuracy and reliability of the model. This section shows how the student evaluates the performance and generalizability of their model.
2. Critical Evaluation: It encourages students to critically evaluate their model's performance, understand its limitations, and discuss potential improvements.

Machine Learning Operations (MLOps)

Project structure

As we did for our last project ("DSPRO1"), we are using a "monorepo" setup with a pipeline-style setup consisting of numbered folder and subfolders, each representing different stages and sub-stages of our dataflow, from data collection to model training. Every stage consists of at least one Jupyter Notebook, with more helpers and reused python code dispersed throughout the project. Each notebook saves the generated data in its current folder, making the flow obvious. Within each sub-step, the notebooks can be run in arbitrary order because they are not inter-dependent.

Handling a lot of files

Differing from our last project, however, is the amount of data. With our scraping generating hundreds of thousands of images, we could not store them in our git repository. Instead, we opted for storing them in our server we had used for scraping, although in a scaled and already enriched format, making it quicker to get our training and repository up and running on a new machine. This server is public to allow for our results to be reproduced.

Using a server for storage made storing the files easy, but it came with the added challenge of reproducibility. Ideally, we would want to store all of our data on the server but only pull the required ones for a particular training, ensuring that they were always the same ones.

(To quickly return a list of all files present without overloading the web server we use to serve the files, we wrote a small PHP script returning the files names as a list of links, which can be easily parsed.)

To solve this came up a custom set of helpers called "data-loader". This would get the list of files from our server, filter them by criteria, sort, optionally shuffle or limit them, and output the full paths to the files that should be used for this processing step or training. Note that each data point consists of both an image file and a JSON file, the "data-loader" treats them as pairs and has stable shuffling and limiting behavior, no matter where or how the files are stored.

Behind the scenes, it writes a text file ("data-list") to the repository listing all of the files used. This file is meant to be committed to the repository and ensures that all future runs of this setup will get the exact same files, otherwise throw an error. If some files were still missing locally, they are automatically downloaded before returning the paths.

Once we had this running, we could easily deploy this on persistent cloud environments like HSLU's GPUHub, however, we also wanted to be able to deploy it on Google Colab <-LINK>. which does not have persistent storage. To address this, we wrote a shell script automatically clone our git repository from GitLab <-LINK>, install dependencies using "Poetry" <-LINK>, convert the training notbooking to plain python and run it.

(Even with the script, setup was still slow because hundreds of thousands of files had to be downloaded from our server first. To solve this, we mounted a Google Drive <-LINK> and stored our files there. However, since the drive adapter is slow and seizes to work with a lot of files, we had to take a couple of measures to address this.

Firstly, we stored our downloaded files in nested directories, containing the first and second characters in the "game ids" of the files. Secondly, we store a list of all files present in the Google Drive, preventing a slow file listing, and lastly, we store the files in a zip file, copy the entire file and uncompress them on the local storage of the runner. This allowed us to quickly deploy our model training to Google Colab, which gave us the chance to rain on more powerful GPUs.)

To speed up training in other environments, especially when using a lot of transformations for data augmentation, we cache the prepared dataset using pytorch right before training. The dataset is saved to a file named after the preprocessing parameters, as well as a hash of all file names to ensure consistency. A file only containing the test data after the split is also saved to make calculating the metrics quicker.

For monitoring and deploying we log and push all of our run data to "Weights and Biases" <-LINK>, which allows us to plot and compare many runs, as well as automatically do hyperparameter-tuning. After each training we also push the model weights as well as the test data, if it has not been saved before, otherwise a link to it. This allows us to deploy a model and calculate the final metrics in seconds.

To talk about:

Creating the demo for the geoguessr wizard and how we are deploying the model in this real-world scenario

Requirements:

1. Practical Application: This section emphasizes the practical aspect of machine learning. It’s not just about building models but also about deploying them effectively in real-world scenarios.
2. Bridging Theory and Practice: It allows students to demonstrate their ability to translate theoretical knowledge into practical applications, showcasing their readiness for industry challenges.
