Lost & Found: Predicting Locations from Images

Teamname: Example Team

Linus Schlumberger, Lukas Stöckli, Yutaro Sigrist

30-06-2024

Contents

Introduction

Group Members

Linus Schlumberger
Lukas Stöckli
Yutaro Sigrist

Problem description

Nowadays, images are often automatically enriched with various data from different sensors within devices, including location metadata. However, this metadata often gets lost when images are sent through multiple applications or when devices are set not to track locations for privacy reasons. As a result, images may initially have metadata, but it is lost when shared with friends or published online. This raises the question: Is it possible to re-enrich these images with their location after the metadata is lost?

The main goal of this project is to determine if an Image Classification Model can outperform humans in guessing the countries or regions of images based solely on images with low resolution and no additional information.

Project Overview

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 region 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.

Potential Applications

- **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

State of the Art

Recent advancements in deep learning have significantly enhanced the ability to determine the geographical location of an image. DeepGeo, developed by Suresh et al., leverages visual cues such as vegetation and man-made structures like roads and signage to infer locations. This approach aims to replicate the human ability to use environmental indicators and prior knowledge for geolocation (Suresh et al., 2016). The DeepGeo model restricts its scope to the United States, utilizing panoramic viewpoints to classify images based on their state. Each input sample consists of four images taken at

the same location, oriented in cardinal directions, which are then classified into one of 50 state labels (Suresh et al., 2016).

In contrast, PlaNet, developed by Weyand et al., tackles global image geolocation. It employs a deep convolutional neural network based on the Inception architecture, trained on 126 million geotagged photos from Flickr. PlaNet's method involves partitioning the world map into multi-scale geographic cells and classifying test images into these cells. Despite its large dataset and extensive training, PlaNet achieves a country-level accuracy of only 30% on its test set (Weyand et al., 2018). M2GPS, developed by Hays and Efros, is another significant baseline in scalable image geolocation. This model performs data-driven localization by computing the closest match via scene matching with a large corpus of 6 million geotagged Flickr images, utilizing features such as color and geometric information. IM2GPS's approach demonstrates the importance of leveraging large datasets for effective geolocation (Hays & Efros, 2008).

Banerjee's work emphasizes the classification task of predicting image location solely based on pixel data. Their research highlights the use of CNNs and transfer learning to achieve high-accuracy models capable of superhuman performance. CNNs are particularly effective due to their ability to capture low-level and complex spatial patterns (Banerjee, 2023). Dayton et al. explored a similar task by using a ResNet-50 CNN pre-trained on ImageNet for classifying street view images from the game GeoGuessr. Their model utilized transfer learning to refine the pre-trained network on a dataset specifically curated for the task, resizing images to 224x224 pixels for input. By fine-tuning the last layers of ResNet-50, they achieved a test accuracy of over 70% for 20 different countries, highlighting the efficacy of leveraging pre-trained models for geolocation tasks (Dayton et al., 2023).

Another notable model is PIGEON, which combines semantic geocell creation with multi-task contrastive pretraining and a novel loss function. PIGEON is trained on GeoGuessr data and demonstrates the capability to place over 40% of its guesses within 25 kilometers of the target location globally, which is remarkable. This model highlights the importance of using diverse datasets and innovative training techniques to enhance geolocation accuracy (Haas et al., 2024). While these models exhibit high accuracy in controlled conditions, they often rely on high-resolution images, multiple perspectives, and enriched datasets that do not reflect real-world scenarios. For instance, DeepGeo's use of panoramic images and PlaNet's extensive dataset of geotagged Flickr photos introduce biases towards urban areas and well-known landmarks, limiting their effectiveness in arbitrary or rural locations (Suresh et al., 2016). Additionally, these models struggle to generalize to lower-resolution images and more diverse datasets that include unseen locations, as highlighted by the performance discrepancies observed in models like PIGEON when applied to varied datasets (Haas et al., 2024).

Furthermore, Banerjee's research on digital image classification since the 1970s underscores the evolution from using textural and colour features to the current reliance on CNNs. This historical perspective reveals that early models had limited discriminative power and robustness, which were

significantly improved with the advent of SIFT (Scale-invariant feature transform) and visual Bag-of-Words models. However, the transition to CNNs marked a pivotal shift due to their superior ability to capture both low-level and high-level features (Banerjee, 2023). Dayton et al. further illustrate the application of transfer learning in geolocation by refining a pre-trained ResNet-50 model on a specific geolocation task. Their work highlights the importance of data augmentation and hyperparameter tuning in improving model performance, as well as the need for balanced datasets to reduce bias and enhance generalizability (Dayton et al., 2023).

To develop more robust and universally applicable geolocation models, it is essential to focus on creating systems that can operate effectively with lower-resolution images and without the need for panoramic views or extensive enriched datasets. This involves training models on diverse, real-world datasets that include a variety of image types, from urban streets to rural landscapes, captured under different conditions and perspectives. By doing so, the models can better mimic the conditions under which humans typically use images for geolocation, such as in social media posts, emergency situations, or historical photo analysis. For instance, PIGEOTTO, an evolution of PIGEON, takes a single image per location and is trained on a larger, highly diverse dataset of over 4 million photos from Flickr and Wikipedia, excluding Street View data. This approach demonstrates the model's ability to generalize to unseen places and perform well in realistic scenarios without the need for multiple images per location (Haas et al., 2024).

Recent Breakthroughs in CNN Architectures

ResNet: Addressing the Degradation Problem in Deep Networks The introduction of deep residual learning by He et al. (2015) marked a significant milestone in the development of convolutional neural networks (CNNs). Their work addressed the degradation problem in deep neural networks by proposing a residual learning framework that allows layers to learn residual functions with reference to the layer inputs. This architecture, known as ResNet, employs shortcut connections that perform identity mapping, which are then added to the outputs of the stacked layers (He et al., 2015). This innovative approach not only mitigates the vanishing gradient problem but also enables the training of extremely deep networks with more than 100 layers, achieving superior performance in image classification tasks.

MobileNetV2: Inverted Residuals and Linear Bottlenecks Designed specifically for mobile and resource-constrained environments, MobileNetV2 introduced by Sandler et al. (2019) represents a significant advancement in efficient CNN architectures. The core innovation of MobileNetV2 is the inverted residual with linear bottleneck layer module, which significantly reduces the memory footprint and computational cost during inference without sacrificing accuracy (Sandler et al., 2019). This

is achieved through a low-dimensional compressed representation that is expanded and then filtered with a lightweight depthwise convolution before being projected back to a low-dimensional representation.

EfficientNet: Rethinking Model Scaling for CNN's EfficientNet, proposed by Tan and Le (2020), introduces a new model scaling method that uniformly scales all dimensions of depth, width, and resolution using a simple yet highly effective compound scaling method. This balanced scaling approach enables EfficientNet to achieve superior performance while being much smaller and faster than previous models (Tan & Le, 2020). For instance, EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, being 8.4x smaller and 6.1x faster on inference compared to traditional CNN architectures (Tan & Le, 2020).

Contributions

This paper has four main contributions. Firstly, we address the limitations of current geolocation models by developing a novel approach that leverages low-resolution images, enabling accurate geolocation in more realistic and diverse scenarios. Secondly, we enhance the dataset by expanding it to include more countries, ensuring a balanced and distributed representation, which is crucial for mitigating biases present in state-of-the-art models. Thirdly, we tackle hardware limitations by optimizing image sizes, making the model more accessible and efficient for deployment on various hardware platforms. Finally, we propose a new methodology for training and fine-tuning our model, incorporating the latest advancements in transfer learning and data augmentation techniques, which significantly improve the model's performance and generalizability across different real-world applications.

Our contributions aim to advance the field of image geolocation, making it more practical and effective for a wide range of applications, from aiding in disaster response to rediscovering family histories and beyond. By addressing these key challenges, we believe our work will pave the way for the development of more robust and universally applicable geolocation technologies.

Methods (REPRODUCIBILIY is the main goal)

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.

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.

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.

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.)

Also write about the ToS we searched and that is not mentioned about web scrapping and therefore it is allowed if it is not exactly permitted from the ToS.

Data analysis?? (We do not have this section now. . .)

As mentioned before under “Data source” and described in detail below under “Mapping to a distribution”, we map our data according to the occurrences of countries in the multiplayer data. Because of this, we started looking at our country distributions quite early.

<-POTENTIALLY INSERT SINGLEPLAYER GRAPH AGAIN>

The singleplayer data country distribution is heavily skewed, consisting mostly of pictures of the United States and a few other overrepresented countries. We knew that this would hamper performance and that a lot of the smallest countries would not have sufficient data for training. However, we also saw a chance to be able to predict a lot of mid-sized countries and smaller countries. Interestingly, it also includes a lot of locations that would normally not be expected to have Google Street View coverage, like very small island nations and even North Korea. This is due to the data not only including official Street View imagery, but also user-submitted photospheres.

<-POTENTIALLY INSERT SAMPLE PICTURES>

After sampling a couple of images we not only realized that predicting the country would be a lot harder than we initially anticipated, wondering ourselves about which labels belonged to which images. We also realized that some images are very dark, low resolution, or blurry, especially the aforementioned photospheres, which we decided to filter before training.

Maybe write more?

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.

Country Enriching (Singleplayer coordinates, Multiplayer names)

In our project, we focused on collecting images from Geoguessr in Singleplayer mode to ensure we obtained real images with precise coordinates. In contrast, images collected in Multiplayer mode only provide the actual country name without exact coordinates. Initially, we faced a ban from Multiplayer mode due to unauthorized webpage injections, which violated Geoguessr's policies. Additionally, we discovered that images from Multiplayer mode often had incorrect coordinates, leading us to concentrate our efforts on Singleplayer mode for accurate data collection.

Our objective is to explore different approaches using low-resolution images to evaluate their effectiveness in predicting coordinates, regions, and countries. For this purpose, we require all three pieces of information for each image, enabling us to train models for accurate predictions. Throughout the process, we encountered various challenges. Reverse geocoding allowed us to derive country information

from coordinates, but the `pycountry` module did not always provide comprehensive location data. For example, Kosovo is not officially recognized by `pycountry`, necessitating its manual addition to our country list. This adjustment was crucial since Geoguessr included information about Kosovo, which was not covered by the `reverse_geocoder` and `pycountry` modules.

We also faced difficulties mapping the correct country to the provided data when names were derived from Multiplayer mode. To address this, we implemented fuzzy matching to find the best match for sample based on the given data from the JSON files, ensuring accurate country assignment.

To enhance the efficiency of processing our large dataset, we employed multiple workers to handle the data concurrently. This approach significantly improved the processing speed by batching the dataset and utilizing parallel processing.

(Issues with reverse geocoding, country name matching) - What exactly issues? Did I mention all of them?

Region Enriching (Source, Mapping)

So to predict the Region of the image, we first searched for a list of regions around the world. And decide to use the geojson file from Natural Earth. Since for each region we had a list of coordinates, which marks the border of the region, we had to get the middle point of each one. Where the python library “geopands” comes in handy. This library has the advantage to be able to work with geojson files and to have an integrated middle point calculation function. In addition, we add a unique region_name for each region using the name of the region + country name + id. This is needed since some region names have similar or the same name. After this preparation, we used the middle point to get the region for each image using their coordinates using k-nearest neighbor method.

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.

Prediction approaches

Predicting Coordinates: Mean Squared Error

The initial approach we took was to predict the exact coordinates of a location. We initially believed this would be easier since coordinate prediction is a basic method. However, after a few training iterations, we realized this approach was more challenging than predicting countries or regions due to several issues.

The first problem was the distribution of our dataset compared to the actual distribution of locations on Earth. From Geoguessr, we collected 81,505 mapped images, with a fairly even distribution after

filtering, and a total of 332,786 images without checked distribution. Despite our efforts to ensure even distribution, the dataset was not uniformly spread across the globe. This imbalance meant that incorrect model predictions often resulted in large geographical errors, increasing the loss significantly.

We observed that while the network performed well on the training set for the first 10 epochs, the validation accuracy for distance was poor. The model struggled to predict the correct continent in the validation set, resulting in high mean loss. We used Mean Squared Error (MSE) and the Haversine distance to calculate the loss, as explained in the "Regions with Custom Loss" subsection. Although we had a mean loss error, it was unclear where the model was making incorrect predictions.

To address this issue, we shifted our focus to the other two approaches: predicting countries and predicting regions. Predicting countries is a classification problem, making it more straightforward. By predicting regions, we can map countries to regions, allowing for a more detailed comparison of results. This fine-grained approach helps us understand which country and region the model believes the image was taken in, making it easier to identify where and why the model makes incorrect predictions and how far off these predictions are from the actual locations.

Predicting Countries: Categorical Cross-Entropy

For predicting the countries, we use torch.nn.CrossEntropyLoss to calculate the loss. This method is well-suited for our classification problem, as it measures how well a predicted probability distribution matches the actual distribution (or ground truth) of class labels.

First, we create a unique mapping list for each dataset, identifying countries by their indices. This mapping allows us to convert country names to indices and vice versa. We then map the countries from the labels to their actual indices in all test splits. Additionally, we calculate the loss using the raw logits from the model without applying softmax, as CrossEntropyLoss internally handles the softmax operation.

We save this information to Weights and Biases (wandb) to ensure the mappings are correctly and centrally stored for each run. The loss for the predictions is calculated using the following formula:

$$L = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{x_{i,y_i}}}{\sum_j e^{x_{i,j}}}\right)$$

where:

- L is the categorical cross-entropy loss.
- N is the number of samples.
- $x_{i,j}$ is the predicted score/logit for class j for sample i.
- y_i is the true class label for the sample i.

By implementing this method, we ensure that our model's predictions are evaluated accurately, providing us with a reliable metric to improve our classification models.

Predicting Regions: Custom haversine smooth loss

For the region-prediction we use a custom loss function. Which, in short text, is a loss function not only look if the correct region is predicted, it considers also the distance to the correct coordinates. Which means if the predicted region is only slightly off then the loss is not that big like if it is far off. There is the paper "PIGEON: Predicting Image Geolocations" from Stanford University, which comes in handy for this task. They're using the haversine smooth loss function. (Haas et al., 2024).

The steps of the custom loss function The haversine distance is a measure of the shortest distance between two points on the surface of a sphere, given their longitudes and latitudes. It is calculated using the following formula:

$$\text{Hav}(\mathbf{p_1}, \mathbf{p_2}) = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

where:

- r is the radius of the Earth (6371 km in this implementation),
- $\mathbf{p_1}, \mathbf{p_2}$ are the 2 points with longitude λ and latitude ϕ

The smoothed labels are calculated using the following formula:

$$y_{n,i} = \exp\left(-\frac{\text{Hav}(\mathbf{g_i}, \mathbf{x_n}) - \text{Hav}(\mathbf{g_n}, \mathbf{x_n})}{\tau}\right)$$

where

- $\mathbf{g_i}$ are the centroid coordinates of the geocell polygon of cell $\mathbf{i}$
- $\mathbf{g_n}$ are the centroid coordinates of the true geocell.
- $\mathbf{x_n}$ are the true coordinates of the example.
- τ is a temperature parameter.

Finally, the cross-entropy loss is calculated between the model outputs and the smoothed labels.

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 architectures

To develop a robust and efficient image classification model for predicting the geographical origin of low-resolution images, we employ several state-of-the-art convolutional neural network (CNN) architectures. These architectures, known for their advanced design and high performance in image recognition tasks, are utilized in our research to ensure optimal results, as detailed in the Literature Review above. We specifically focus on ResNet, MobileNetV2, and EfficientNet architectures, each offering unique advantages for our project.

Addressing the Degradation Problem in Deep Networks. ResNet's deep residual learning framework effectively addresses the degradation problem in deep neural networks by allowing layers to learn residual functions. This approach mitigates the vanishing gradient problem and enables the training of extremely deep networks. ResNet's robustness and generalizability are evident in its performance on the ImageNet dataset, where the 152-layer ResNet achieved a top-5 error rate of 4.49% (He et al., 2015). These characteristics make ResNet an ideal baseline model for our research, aiming to predict the country of origin for low-resolution images.

Efficient Architectures for Mobile and Resource-Constrained Environments. Designed for mobile and resource-constrained environments, MobileNetV2 represents a significant advancement in efficient CNN architectures. The core innovation of MobileNetV2 is the inverted residual with linear bottleneck layer module, which reduces memory footprint and computational cost during inference without sacrificing accuracy (Sandler et al., 2019). This efficiency makes MobileNetV2 particularly suited for our project, where we aim to develop a model that can run efficiently on various hardware platforms while maintaining high accuracy.

Balancing Performance and Efficiency in Model Scaling. EfficientNet introduces a model scaling method that uniformly scales depth, width, and resolution using a simple yet effective compound scaling method. This balanced scaling approach enables EfficientNet to achieve superior performance while being much smaller and faster than previous models (Tan & Le, 2020). EfficientNet's combination of high accuracy and low computational requirements makes it an excellent choice for our project, where hardware efficiency is critical.

By leveraging these advanced CNN architectures and fine-tuning them on our custom dataset, we aim to develop a high-performing, efficient model capable of accurately predicting the geographical origin of images with minimal hardware resources.

Training and Fine-Tuning

For our project, we use the following models from selected architectures:

- **ResNet**: ResNet18, ResNet34, ResNet50, ResNet101, ResNet152
- **MobileNet**: MobileNetV2, MobileNetV3 Small, MobileNetV3 Large
- **EfficientNet**: EfficientNet-B1, EfficientNet-B3, EfficientNet-B4, EfficientNet-B7

To be resource-efficient and enable training the CNN architectures with affordable hardware, we decided to use pre-trained weights and replace the last classification layer with a custom classification layer that matches the classes for the countries or regions in our datasets. For the training, we reduce the learning rate for the layers with the pre-trained weights by a factor of 10, allowing the network to focus more on training the new classification layer with randomly initialized weights while fine-tuning the existing layers with pre-trained weights. The pre-trained weights we use are the default weights from the torch models, IMAGENET1K_V1 and IMAGENET1K_V2, which we loaded and used throughout our project. Our goal is to allow the new classification layer to learn more effectively while merely fine-tuning the existing layers.

Additionally, we considered integrating all possible countries and regions into the models for the final classification layer to make the models more adaptable for multiple tasks and other datasets. This approach could facilitate further training with more data, including more countries and regions, at a later stage. However, we decided against this to help the network perform better with the existing classes in our dataset. Including too many classes would introduce additional complexity and potential issues due to class imbalance and insufficient representation in the training data.

Data augmentation

Data augmentation is crucial for computer vision tasks, especially when using CNN networks. While CNNs are very powerful in identifying objects and patterns, they struggle with variations in rotation, perspective, or view. To address this, it is essential to augment the training data to make the model more robust to different image transformations. This prepares the model to recognize identical objects from different angles or if they are stretched or squeezed. For our Geoguessr images, data augmentation is particularly important. The images are consistently taken by a Google car with the same height and camera settings. This consistency helps the model learn from a uniform perspective, improving

accuracy. However, it also means the model might generalize poorly to other datasets not taken from Google Street View.

To evaluate the impact of data augmentation, we will train our model with and without augmentation, comparing performance on different datasets to see how well the models generalize. This comparison will be insightful in understanding the effectiveness of data augmentation. For our specific case with Google Street View images, we applied the following augmentations to the training data:

- **Random Resized Crop**: This augmentation randomly crops the image and resizes it to the original size. This helps the model learn to recognize objects in different parts of the image and at different scales, addressing the issue of objects appearing in varied locations within the frame.
- **Random Rotation**: This augmentation randomly rotates the image by up to 10 degrees. It helps the model become invariant to slight rotations, which is important since objects in Google Street View images can appear slightly rotated due to changes in the car's movement or camera alignment.
- **Color Jitter**: This augmentation randomly changes the brightness, contrast, saturation, and hue of the image. For example, varying the brightness can simulate different times of day (e.g., night vs. bright daylight), while adjusting the hue and saturation can account for different weather conditions or camera sensor variations. This makes the model more robust to lighting and color changes that are common in real-world scenarios.

»> Add here a picture of the training augmentation to see what it visually »>

These augmentation techniques are essential for making our model robust and capable of generalizing to different images beyond the specific conditions of Google Street View. By simulating various real-world conditions, we aim to improve the model's ability to handle diverse and unseen environments.

Hyperparameter tuning

Another method we used in this student project is hyperparameter tuning. It is a crucial part of machine learning and helps to find the optimal settings for the model to learn and perform at its best. During hyperparameter tuning, all parameters were saved to Weights and Biases (wandb), which also tracked and saved all the metrics. This platform allowed us to organize training schedules, manage the entire training process, and keep everything centralized. Wandb's filtering capabilities made it easy to retrieve specific runs and compare different accuracies for each country and region. Initially, we set some static parameters that we did not change during hyperparameter tuning. For the optimizer, we used the AdamW optimizer for all runs, which handles weight decay internally. We also used a scheduler to decrease the learning rate after a certain number of epochs to prevent overshooting the learned parameters, with the learning rate being decreased every 10 steps. Although these parameters could

be adjusted during hyperparameter tuning, we chose not to tune them due to time constraints and their minimal impact on performance.

For our hyperparameter tuning, we focused on two different parameters: learning rate and weight decay. We trained the models on five different learning rates: 1e-1, 1e-2, 1e-3, 1e-4, and 1e-5. The learning rate significantly impacts how well the model can learn. After initial experiments with a broader range of learning rates, these five were the most promising during training.

Additionally, we applied three different weight decay values: 1e-1, 1e-2, and 1e-3. Weight decay helps penalize large weights in the network, leading to several benefits: reducing overfitting, improving model stability, promoting feature sharing, and enhancing generalization in over-parameterized models. These three weight decay values helped achieve higher performance compared to not using weight decay. We did not need L2 regularization because the AdamW optimizer handles it internally.

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…

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. **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.

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.

Model performance on other datasets

Follows…

Experiments and Results (and also discussions)

Predicting coordinates

Follows…

Table 1.1. Best runs for predicing coordinates

Size dataset	Augmented	Train distance (km)	Val distance (km)	Test distance (km)
$79'000$	No	837	970	xx
$81'505$	No	877	230	xx
$81'505$	Yes	25	415	xx
$332'786$	No	144	23656	xx
$332'786$	Yes	300	556	xx

Predicting country

Follows… DO different tables to see the differences for training validation? Show also the models… the different we tried…

Table 1.1. Best runs for predicing country

Size dataset	Augmented	Train distance (km)	Val distance (km)	Test distance (km)
$79'000$	No	837	970	xx
$81'505$	No	877	230	xx
$81'505$	Yes	25	415	xx
$332'786$	No	144	23656	xx
$332'786$	Yes	300	556	xx

Predicting region

Follows…

Table 1.1. Best runs for predicing region

Size dataset	Augmented	Train distance (km)	Val distance (km)	Test distance (km)
$79'000$	No	837	970	xx
$81'505$	No	877	230	xx
$81'505$	Yes	25	415	xx
$332'786$	No	144	23656	xx
$332'786$	Yes	300	556	xx

General

Write also that we found out that a bigger data size matters more than bigger images for all of the prediction models, which is a really nice catch and learning.

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.

List here all results in plots, confusion matrix and so on. The goal of these sections is to present our result and explain how they help in solving the problem we are working on, and to answer the research questions we are trying to answer.

Our Hypothesis: The main goal of this student project is to determine if an Image Classification Model can outperform humans in guessing the countries or regions of images based solely on the image itself, without additional information.

Conclusions and Future Work

Follows…

Here we should discuss the implications of our results, our limitations, and possible further research possibilities. We should be very honest especially about limitations.

References

Banerjee, A. (2023). *Image geolocation with computer vision*. http://www.arshbanerjee.com/uploads/paper/3bda9_20230723185809.pdf

Dayton, F., Heo, J., & Werner, E. (2023). *CNN plays geoguessr: Transfer learning on ResNet50 for classifying street view images*. https://www.finndayton.com/CS229_Final_Report.pdf

Haas, L., Skreta, M., Alberti, S., & Finn, C. (2024). PIGEON: Predicting image geolocations. *arXiv*. https://arxiv.org/abs/2307.05845

Hays, J., & Efros, A. A. (2008). IM2GPS: Estimating geographic information from a single image. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. http://graphics.cs.cmu.edu/projects/im2gps/

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv*. https://arxiv.org/abs/1512.03385

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). MobileNetV2: Inverted residuals and linear bottlenecks. *arXiv*. https://arxiv.org/abs/1801.04381

Suresh, S., Chodosh, N., & Abello, M. (2016). DeepGeo: Photo localization with deep neural network. *arXiv*. https://arxiv.org/abs/1810.03077

Tan, M., & Le, Q. V. (2020). EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv*. https://arxiv.org/abs/1905.11946

Weyand, T., Kostrikov, I., & Philbin, J. (2018). PlaNet - photo geolocation with convolutional neural networks. *arXiv*. https://arxiv.org/abs/1602.05314