Large Scale Data Engineering: Deep Locations

Group 03

November 1, 2020

C. Lin
Department of Computer Science, Vrije Universiteit Amsterdam
1081 HV Amsterdam
The Netherlands
c2.lin@student.vu.nl

H. Wang
Department of Computer Science, Vrije Universiteit Amsterdam
1081 HV Amsterdam
The Netherlands
h5.wang@student.vu.nl

K. Ma
Department of Computer Science, Vrije Universiteit Amsterdam
1081 HV Amsterdam
The Netherlands
k.ma@student.vu.nl

ABSTRACT

This paper describes a project development procedure and its result of a five-week-long practical assignment in the Large Scale Data Engineering (LSDE) course at Vrije Universiteit Amsterdam. The assignment tackles a photo geolocation task by using Yahoo Flickr Creative Commons 100 Million (YFCC100M) dataset and the deep learning model. This Paper would further outline how the images in YFCC100M can be efficiently accessed in the Amazon S3 bucket rather than downloading images by Flickr uniform resource locator links. Moreover, more than 410402 images have been inferred by our model; a website application was also created for showing our prediction accuracy, and for users to compare the ground truth coordinate with the model predicted coordinate of the images.

1. INTRODUCTION

Predicting the GPS location of images is an extremely challenging task since many images often contain only a few information or ambiguous cues about their location. For instance, an image of a forest could be taken in many similar places in mountains around the world. Humans can use their worldwide geographical knowledge, the cues like the country flags, the language of street signs, or the human race in the images to infer the location of an image. There is an online website application like Geoguessr[1] for users to guess the location of several street view images. It is not surprising that few of us can perform well while inferring the correct location of images by our geographical knowledge. Even with worldwide geographical knowledge, this guessing task is still challenging for humans. Therefore, predicting the GPS location of images through computer vision methods arouses computer scientists' interest. However, lots of traditional computer vision methods lack this kind of worldwide geographical knowledge, restricted by the features of data during training. In contrast, our goal of this assignment is to localize any type of images by our deep learning model. We consider this task as a classification problem and divide the earth into lots of geographical cells which construct the classes. We then train a residual neural network (ResNet) using geotagged images in the YFCC100M dataset.

In this paper, we will first introduce some related work which gives us enormous inspiration and useful methods to tackle this assignment problem. Afterward, the comprehensive method of achieving our goal will be described. This would be done by mentioning the architecture we used for this assignment, after which the procedure of data analysis and sampling would be noted. Next, the pipeline which consists of four parts; data processing, accessing images through Amazon S3 bucket, training model, and inferring images would be explained. After this, the result of the experiments and the comparison with other similar works would be shown. Last but not least, the function of our visualization website would be introduced, the conclusion and discussion of this task would be given. The appendix offers some additional information and the contribution of the authors to this assignment.

2. RELATED WORK

In this section, we will describe several related works for this project, and focus on two of the research papers that are closely related to our project's objective. These two academic papers did provide us with substantial insights into our assignment.

Some related work had been restricted to specific conditions or environments so that they could only focus on limited subsets of problem, like cities [7, 10], or landmark buildings [12, 1]. However, in this project, we would like to implement a metric that can infer any of the images taken on the earth without any restrictions. After seeking the planet-scale geolocalization works which are more closely related to our project's objective, Im2GPS [2] and PlaNet [9] came into our views. IM2GPS was introduced by Hays and Efros, they match a query image based on global image descriptors to refer to the dataset of six million GPS-tagged images, and pick the closest match. However, this kind of retrieval method may not meet our project's goal and setting. PlaNet was introduced by Weyand et al. [9], and was a deep learning model that could classify the images' geolocation with

[1]Geoguessr: https://www.geoguessr.com

amazing accuracy. They treated the geolocalization task as a classification problem. The surface of the earth was properly subdivided into geographical cells which assign to the specific classes. They then train the Convolutional Neural Networks (CNN) model by using 91 million geotagged images. The training process took them 2.5 months on 200 CPU cores using the DistBelief framework. Indeed, this research paper gave us deeper insight into implementing our project and shed light on some aspects that are relevant to our project. However, the number of resources they used and the time they took may not meet our project setting.

In the meantime, we found out many other similar methods for this task introduced by Eric et al. [11] in 2018, which uses a much less training dataset than PlaNet yet still outperformed at the geolocalization task. They introduced an advanced method based on the PlaNet concept, which exploits the geographical hierarchical knowledge of multiple geo-partitionings and takes the image's scene into consideration, such as urban, indoor, natural. They proved the effectiveness of their method by using a much lower number of training images. The training images of their work are less than 0.47 million images while the PlaNet used about 91 million for the training process. Therefore, we consider the approach introduced by Eric et al. [11] as the most effective way and related work for our project.

To conclude, the related work of tackling the geolocalization problem could be divided into two categories: (1) metrics restricted by certain conditions - landmark building, cities, streets, etc. (2) approaches to classify images in the worldwide range without any restrictions. We focus on the latter category and find out two of the related work (PlaNet[6] and Eric et al. [11] work) which could offer us deep inspiration and implementation knowledge for our project. Moreover, the setting of Eric et al. [11] work is more similar to our project. Therefore, Due to the effectiveness of their approach, we aim to use the geographical hierarchical knowledge of multiple geo-partitionings introduced by them to improve our performance of prediction accuracy in this geolocalization task.

3. RESEARCH QUESTIONS

The following sections will describe the methodology of our DeepLocation project in detail. To better understand the method that we use to realize the project's goal and illustrate the advantages of technologies for Big Data in Deep Learning, we firstly state our main architecture of the system and implement tools. Secondly, we will present the analysis of datasets by sampling some records. Thirdly, the pipeline of the project will be described in detail which contains four parts to aim at requirements. These four sections are as follows: data processing, person removal, model building, and prediction GPS.

To achieve the goal of the project, the following research questions should be answered:

- How could we efficiently obtain the images from Flickr or AWS S3 bucket?
- Which method or deep learning model should we use for pursuing higher performance in prediction?
- After training models, how do we efficiently do inference for images in such a large dataset?

Table 1: Basic configuration of the instances

Instance	vCPUs	Memory	GPU	Networking
m4.2xlarge	8	32 GiB	-	high
i3.xlarge	4	30.5 GiB	-	10 Gbps
g4dn.xlarge	4	16 GiB	NVIDIA T4	25 Gbps

4. ARCHITECTURE

The development of the DeepLocation project mainly is based on Databricks, an open and unified data analytics platform. Databricks is a web-based platform providing developers and scientists with an interface of Spark framework and implement programs on notebooks. It provides a data science workspace, which is a collaborative environment for us to manage and run our scripts. With the basic configure, the platform has the Spark engine both in Python and Scala programming language. Databricks has a friendly user interface for coding in a notebook and worker status viewing in logfiles or the Spark UI.

It is separated from storage and computing clusters both using Amazon Web Service. Amazon Web Service (AWS) is a cloud platform not only providing infrastructure technologies like compute, storage, and databases. The data including metadata and image dataset are stored in Amazon S3, Simple Storage Service. Amazon S3 is an object storage service that offers scalability, data availability, security, and performance. The data we use is mounting from the public S3 bucket to ours and managed in the Databricks File System. All the computations are executed on clusters of the Amazon EC2, Elastic Compute Cloud. Amazon EC2 is a web service that provides resizable compute capacity in the cloud. In the data processing stage and some of the prediction stages, we use the type of m4.2xlarge instance as a driver and i3.xlarge instance as workers.

For the driver, the m4.2xlarge is a balance of computing, memory, and network resources like the web server in the Databricks. For the workers, each i3.xlarge instance provides instances optimized for low latency, very high random I/O performance, high sequential read throughput. Both two instances have very high networking performance. Databricks can dynamically reallocate workers to be suitable for the characteristics of our job. With autoscaling in 16 workers, it allows us to run the scripts maximal on 16 instances, the same as one machine with 64 cores. In the stage of building the model and the stage of DeepLocation, we use a GPU-based node g4dn.xlarge to speed up our deep learning model training task. The instance g4dn.xlarge (beta) can help accelerate deep learning training or inference and graphics-intensive workloads. The basic configuration of the above instances is shown in Table 1.

The main framework adopted in the project is Apache Spark. Spark is a computation engine and distributed cluster computing framework for large scale data processing. It provides high-level APIs in Scala and Python. It also supports a rich set of tools including PySpark library and Spark SQL package for structured data processing. Databricks provides users with a high-level of Spark configuration in drivers and workers. The complex configuring work has already done when creating Databricks cluster both in python interpreter and Java virtual machine for Scala.

Table 2: The description of dataframe

Column name	Datatype	Adopt
media_id	bigint	
user_nsid	string	
user_name	string	
date_taken	string	
date_uploaded	int	
device	string	
title	string	
description	string	
user_tag	string	
machine_tag	string	
longitude	double	✓
latitude	double	✓
geo_accuracy	int	
media_url	string	
download_url	string	✓
license_name	string	
license_url	string	
sever_id	int	
farm_id	int	
secret	string	
secret_ori	string	
extension_ori	string	
media_type	int	✓

5. DATA SAMPLING AND ANALYSIS

In this part, we will investigate the data related to the project. As mentioned in the introduction, DeepLocation is using images to predict the most possible geographic coordinates that indicate the location where images are taken. Therefore, the final objects we process are data of images and their geographic information. There are numbers of public datasets that have images along with their information, like Open Images published by Google and Facebook social media. We use the YFCC100M [8] dataset in the project, which is released by Yahoo Flickr.

The YFCC100M is an abbreviation of the Yahoo Flickr Creative Commons 100 Million Dataset. We notice that YFCC100M is the largest publicly and freely useable multimedia collection, containing the metadata of around 99.2 million images and 0.8 million videos from Flickr, all of which were shared under one of the various Creative Commons licenses. The metadata is originally distributed through AWS and hosted in an S3 data bucket. Mounting the metadata, it is compressed and divided into BZ files total around 15GB. Mounting the multimedia commons data, the images take up approximately 13.5 TB at default pixel resolution. It would be much easier and faster to launch an EC2 instance and process the images directly from the S3 bucket.

Initially, we take one of ten in metadata files and sample a fraction of 0.003 to investigate the original data schema detailly. After parsing, we have a dataframe with 23 columns and 29,988 rows of records which contain the full information about multimedia from Flickr. To the requirement of our project, it is not necessary to utilize every column. Moreover, there are some missing data in important columns. The schema of the dataframe is listed in Table 2. Each media record included in the dataset is represented by its Flickr identifier, the user who created it, the camera that took it,

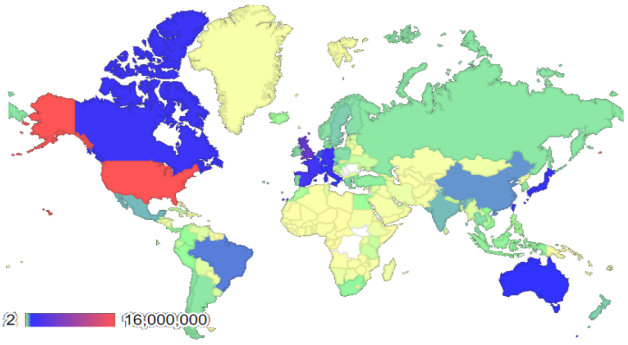

Figure 1: The distribution of images in Flickr

the time it was taken and uploaded, the location where it was taken (if available), and the Creative Commons license under which it was published. The title, description, and tags are also available. Together with viewing and downloading a uniform resource locator, they are convenient to check on the browser. However, it can be missing in the Flickr server or the database. We randomly choose 3,000 image download links. Fortunately, only one image in the S3 bucket cannot be accessed successfully. We access manually in the DBFS to find that the image file is disappeared.

Querying in detail, we find more features of the YFCC100M dataset in category and amount. Totally, the full metadata has 99,959,167 rows. Among these, there are 48,366,323 images and 103,506 videos in the metadata dataframe that have been annotated with geographic coordinates, either manually by the users or automatically through devices recording the latitude and longitude. The original distribution of geography is shown in Figure 1. We only need lines showing images with GPS information and the way of accessing the exact files in Amazon S3. Therefore, we take columns of longitude, latitude, download_url, and media_type to form a new dataframe as the main processing object and save as a comma-separated-value file.

6. PIPELINE

In this subsection of the method, we will demonstrate the pipeline of the whole development in the DeepLocation project which starts from the metadata and ends with giving the image's GPS location. There are four stages of implementations: the first is processing with the metadata and image files, the second is removing person portraits from the dataset, the third is building the core model inputting images then outputting geographic information, and the last stage is predicting the geographic coordinates of images by the hierarchy structure.

6.1 Data processing

In the part of data analysis and sampling, we have illustrated that the data we use is metadata in the comma-separated values file and images in the S3 bucket. Therefore, it is comprehensive that we deal with the metadata and images. What's more, we must find out the connection between a concrete media record and its corresponding image to access the valid and correct one. And delete the missing images and empty files.

6.1.1 Clean the metadata

The previous sampling reveals the fact that we do not have to load and use all the columns in the metadata file. It could not only increase the time of processing but occupy too much space on memory and storage. As analyzed before, we adopt columns of longitude, latitude, downloadable link, and media type mark to create a new dataframe for all over the metadata and save as comma-separated-values files in snappy-parquet. Thanks to Delta Lake, one data lake which provides ACID transactions, scalable metadata handling, and batch data processing, we can have faster access and writing speed.

Next, we must confirm every record has full and valid information in columns. There have been several cases that miss either latitude or longitude or neither to be found. Geographic coordinates are the most important representative data of the project in later modeling. We eliminate such rows that show null value in the arbitrary one column. Then, we have complete geography information of each record about the multimedia.

According to the aim of the DeepLocation, we focus on the images but not videos. It is suggested that records on videos are needless for the project. That is why we keep the media type mark in pre-processed metadata. We can do further to purify our metadata by deleting video rows whose media type mark is integer 1. Thereafter, we drop the column of media type mark because all the rows are images and it can reduce compute resources.

6.1.2 Access images in S3

The ideas of accessing images are in two ways. The first is to using download uniform resource locators showing in the metadata directly. Most groups obtained images in this way before and they took an average of four days to download images. We suppose that it will take much time and not fit our expectations. While sampling, there are indeed some of the images missing or being rejected to access. We take the second way. We use multimedia commons data mounting in the S3 by accessing images via its absolute path in Databricks file system.

Conducting to access images via the absolute path in S3, one obstacle is to obtain the path. We find that the image file name in S3 multimedia commons is an MD5-hashed name. With the basic cognizance, MD5 is used to make the data more anonymous and easier to manage in the file storage system. By testing and trying numbers of times, we notice that the plain code of the MD5-hashed file name is right download uniform resource locators in the columns from our processed metadata. Fortunately, we can use the message digest class in Scala to run MD5 encoding on the dataframe in parallel. We now have the MD5-hashed name of every image in bytes. However, they need some modification before we use them to access images in the S3.

We discover that the original MD5-hashed code maybe not the same as whose corresponding file name in the S3. Actually, the reason is a type of mandatory conversion problem by Amazon when gathering the whole images. The normal file name is a string; however, we have built a bytes-based MD5-hashed code. Regularly, it is obvious to read bytes to a string. There is an issue that byte and string have different allocations in the memory. In the memory, the unit of byte occupies 8 bits while a string unit occupies 16 bits. But Amazon regards as the same in encoding. We raise a converting work, making MD5-hashed file name from bytes to a string. The key point is to set the format string back to only one position intentionally. That is why the image file names in the S3 may lack several "0". They are default values when occurring vacancies of bytes units.

Then, we notice that the file is stored in a double folded directory. It is intuitive to know that the relative directory is separated every first three characters of the image file name. We take a slice of a string of file names to represent the relative directory.

By now, we can get the correct relative path of images plus the file names. we do the above work defining in one function and fit it in Spark to execute on every row of the dataframe in metadata concurrently.

6.1.3 Image existence verification

Although we have been able to locate each image in the S3, it is unknown that the image whether exists or has valid data content. Image files losing is acceptable for various reasons, like unrecoverable read error and storage failure. For example, in the multimedia commons bucket, the files in */data/images/ac8* are all lost because even this directory is missing. It leads to at least 420 images in that directory disappeared.

The rows in the metadata like that are useless in future deep learning model training. On the contrary, they might be harmful to training when having a file not exists exception. As a result, the procedure of training might abort suddenly. For the robustness of building the model, we here deploy an image existence verification to detect whether the file paths shown in the processed metadata can be accessed in the S3. If raising the file does not exist exception, we will delete this row immediately. In this way, we can make sure that all left rows can be accessed normally.

Only doing the image file existence verification is not complete. In the test of sampling and tiny scale of dataset trying, there is a small number of images pass the existence verification, however, the operating system shows that they are empty. We turn to find out such image files. It is unbelievable that the image is truly existed but do not have pixels and shows in empty. The obvious feature of such images is that the file size of them is small, no larger than 1 KB. The rows in the metadata whose images have empty content must be ignored because they would take up classes without any geographic information like landscapes, buildings, or scenes from images.

We implement checking empty images while training and predicting, not in the stage of the data processing stage. Because the possibility of accessing an empty image is tiny, it is a waste of time and computes resources checking every image individually in this stage. We set an empty list for such empty images before feeding to the model. In this way, we reduce the time meanwhile increase the robustness of training and prediction.

6.1.4 Resize images

We have every image valid accessing so far. Due to images are from user uploading to Flickr, the size and resolution of images vary from each other. It is unfriendly to feed these raw images to TensorFlow-based neural networks. When feeding images to networks as the input of deep neural networks for version, the size, resolution, and shape must be in the definitive format. We will take three steps to uniform

each image to a regular input in color, size, and shape as a Tensor.

The first step is to decode the image into the float type. Our function accesses and reads images in the S3. It is decoded into three channels for the RGB color model. Then, the type is converted from image to float, so as to do more calculation to modify the image. The second step is to do a random crop on the image in the training set. For a general model and avoiding overfitting, we set a small range of random crop ratio to modify the image only in the training set. It is a usual measure of training a very large amount and massive classes image classification model. the last step is to resize the image to the definitive shape. According to the input of the Resnet in the neural networks, the shape of an image must be [224, 224, 3]. For each image, we calculate the ratios between 224 and actual height and width. Then, we use resize, a TensorFlow image function, to transform the size of the image to [224, 224] by the ratio in three channels, [224, 224, 3].

Besides, the images in the validation set are different from those in the training set. The resizing step and cropping step exchange their order in the validation set. And the cropping method is a fixed center crop. What is more, we set the ratio of 0.5 more than the definitive [224, 224] and then calculate the offset to crop. They are the same as we do in prediction. The reason is that we can have more consistency of input images for validation and prediction in this way.

6.2 Person removal

DeepLocation aims to provide geographic coordinates from images which have plenty of scene information or representative feature about the background. We do not expect many pixels to display human beings. In other words, the size of one image is definitive, and more people in the image means less information on geography. The issue is critical to the model that inputting images with portraits would have a huge influence on accuracy. The actual features we need are from scenes like scenery, landmark, buildings, and so on in the background. In the sampling, we have experienced many images with people in them. We consider that the images are from the social media website. It is common to have many portraits, selfies, and group photos. When processing images with portraits, detection, and removal work is required.

6.2.1 Face detection

Not all the images have persons in them and not all of them need to be modified. To enhance the efficiency of the person removal, we first carry out the detection of the face in an image. From the previous research [6], the neural network-based upright frontal face detection is a mature method to capture faces in any certain image. There are some public well-trained models of the face. We choose Stump-based 20x20 gentle Ada boost frontal face detector [4] in the Open Source Computer Vision Library as the model of our face detection task.

For each image, we have a simple flow of processing. Using the OpenCV library, the image can be read from the S3 by the path in the metadata. We do the grayscale on the image to reduce the influence of color on detection. Plus, the face detector is also particular in the greying image. Then, the image is input to the detector which can give out the result containing a list of rectangular regions of every face in the image if it has. Next, we conduct a sum-up of the area in the face region list. The sum-up value is the total proportion of faces in the image. As different images may have different sizes, we calculate the ratio of face regions to the image. The ratio can represent the influence of person portraits in images. Deploying this workflow on the metadata, we can have a new column save the ratio of every image.

6.2.2 Removal strategy

We have known a person's influence on each image when training. The measure to reduce such influence mainly is to eliminate the person in the images. We consider two ways to do the elimination. One is to circle out the portrait and fill pixels with noise or the background. The other is to delete such a row from the metadata.

For the first way, we want to circle out full bodies as well as close-up views in the presence of clutter and occlusion. Further, we can use grab cut to remove the portrait and fill with noise to get the purified image which mostly contains geographic information. The grub cut algorithm [5] is designed for foreground extraction with minimal user interaction. However, during the test it needs python OpenCV library to do and takes near 2 seconds for one image. We must give up this way and turn to the second due to too much time consumed.

The second way is to delete the rows regarded much influence. After the face detection, we check a few images with the non-zero ratio. We decide to set a threshold of 3.5%. If the column "ration" of the row is more than 0.035, we regard that it will have much influence on training the model. We remove the person in the image by removing this record of the image from the metadata directly. It is not a rough handing method but has a good result in the validation of the training model. The accuracy of the model using person removal images is 36.9% a little higher than no removing in 35.2%. This is a trade-off between the time consumption and the accuracy of the model.

6.3 Training the model

Since our approach treats the geolocalization task as a classification problem, we should make up specific classes for training. Therefore, we do geo-partitioning which subdivides the earth's surface into plenty of cells. Each cell represents a certain class so that the model can use it for classifying. The fineness of partitioning could be controlled by limiting the number of images in each cell. For instance, if each cell is able to contain more images, the cell would occupy a larger area. On the other hand, we also set a minimum number of images for each cell. If the cells do not contain more than the minimum numbers of images, such images are considered unimportant. We would discard unimportant images since they are often the area like polar regions or oceans which are hard to classify. In order to obtain good performance in the stage of inferring by utilizing the technique of geographical hierarchy, we must create three partitioning files according to different levels of fineness. Figure 2 shows that we obtain three partitioning files – sparse one (Geo Cell-C1), medium one (Geo-Cell C2), and dense one (Geo-Cell C3). Afterward, in the inferring stage, we can utilize these three different levels of partitioning to generate a geographical hierarchy from different spatial resolutions. The details of the setting in each partitioning file are as follows in Table 3.

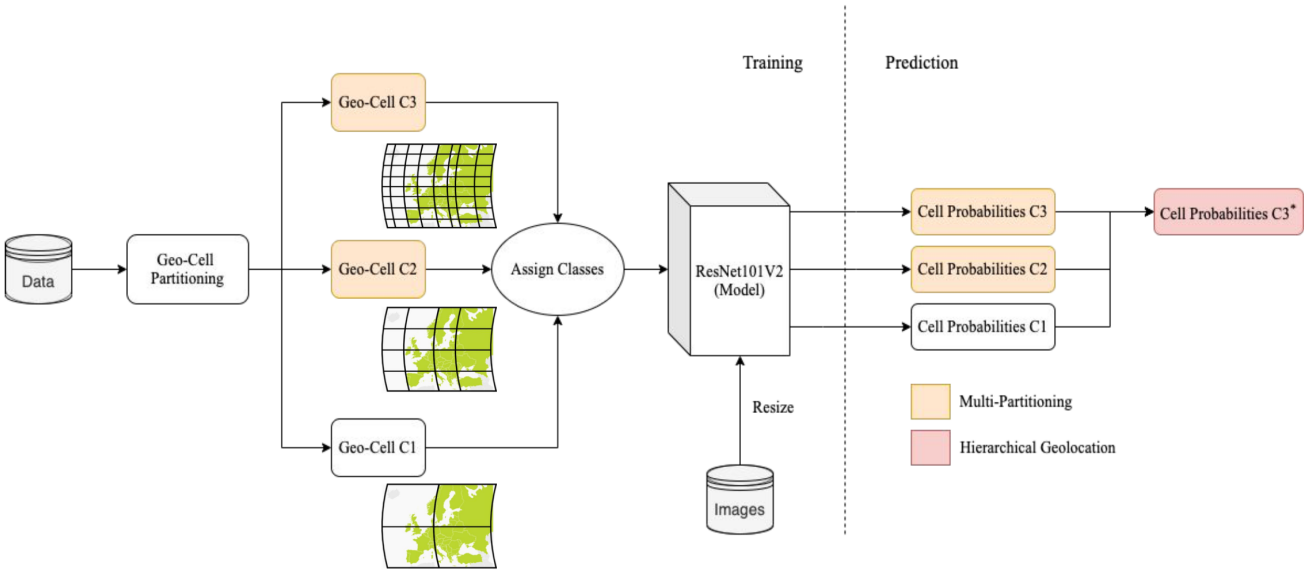

Figure 2: Pipeline of geolocalization task

Table 3: The setting of each partitioning files

Partitioning files	Fineness	Max	Min	Classes
Geo-Cell C3	Dense	100	5	4103
Geo-Cell C2	Medium	200	5	2428
Geo-Cell C1	Sparse	500	5	1024

We use the transfer learning from the residual neural network, Resnet101V2 [3]. The network can be imported and loaded in the beginning to conduct the abstraction of image features. Per-trained model ResNet-101 is a convolutional neural network that is 101 layers deep. Deeper ResNet encoder has produced better results taking performance into consideration when compared to the shallow. In contrast to the only one partitioning file method (like PlaNet), a fully connected layer for all of the cells in three partitioning files was added. Then an activation SoftMax is linked next for classifying. The classification loss of three partitioning files was the mean of loss for each partitioning.

After finishing geo-partitioning for making up the classes, we then divide 154052 images into training data and validation data with a ratio of 8:2. Each epoch in the training process took about 4.5 hours with the checkpoint, and we would choose the model with the lowest loss for inferring. Moreover, we also use another dataset for training which was obtained by our removal strategy mentioned in the section 6.2.2. We will discuss more details about these two models which contain slightly different datasets in the section of result.

6.4 Inferring Process

In this section, we would describe the process of inferring for geolocalization tasks and more details about using the geographical hierarchy for improving accuracy. We first explain the condition of only one partitioning file is used for inferring. In this condition, after training, we would only use one class probability at a single spatial resolution. For instance, when we only use the coarse partitioning file for inferring, the maximum probability of each class label in this partitioning is used for predicting the cell. However, it may not perform well in prediction, so we tried to implement the hierarchical prediction using three partitioning files with different spatial resolution.

The different maximum numbers of images in one cell were applied when making up the classes so that we are able to ensure each class in the finest partitioning file can be connected to a larger cell in both the middle one and coarse one. Hence, we can create a geographical hierarchy from different levels of spatial resolution. We then multiply the respective probabilities at different levels of the hierarchy. As a result, the finest representation of division could be improved by utilizing the information from coarser representations. Furthermore, we will describe more details in the result section, comparing between two different conditions- using only one partitioning file for inferring, or using hierarchical prediction.

To speed up the inferring work, we decide to use Spark to predict each row in the metadata. The reason is that the inference should refer to the information of images in the dataset. Therefore, the task can run in parallel with Spark. The metadata is loaded as one dataframe, and every row can be read as an RDD. The workers of Spark clusters can execute the inferring job in parallel then collect the prediction of geographical coordinates and write to a new dataframe as the result. To be more specific, each method of dataframe is called to implement the parallelized inferring work.

7. RESULT

In this section, we will demonstrate and analyze the results of our data products from the pipeline. The model of classifying images into geographic coordinates will be shown at first. And the performance of DeepLocation, predicted by the model, will be illustrated next.

As detailed in the model building of the pipeline, we train deep learning networks to do a classification task. We use the processed metadata and images to train our model. The

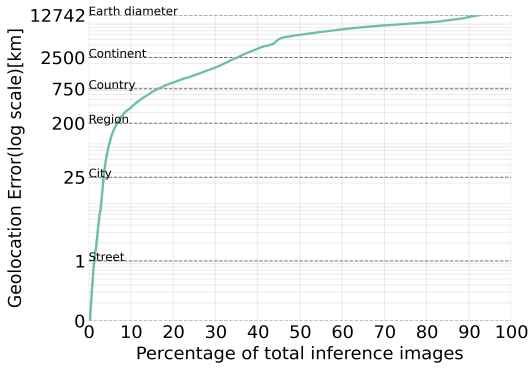

Figure 3: The accuracy of prediction

number of images with geographic coordinates is 154052. The training is deployed in the g4dn.xlarge instance using a GPU to speed up the training procedure. The part of image processing and data loading execute in the CPU. Only the fitting model uses GPU to compute.

The parameters are explained in Section 6.3, the minimum threshold for the adaptive subdivision is 5. The maximum thresholds are set 100, 200, and 500. Therefore, there are 7,555 geo-cells created which are 3.4 times less than PlaNet's. Since the adaptive strategy, DeepLocation uses dynamic cells according to the geographic coordinates of images. It has a more accurate resolution while decreasing useless cells comparing with the fixed number of cells. The resulting number of cells for different partitionings to train are shown in Table 3.

7.1 Comparing models

The result of the geo classification model is the loss in the validation set and direct prediction accuracy of the testing set. As mentioned in the Section 6.2.2 removal strategy, two datasets, which differ in whether has the face ratio of person portraits above 3.5%, are fed to the deep learning networks separately. The distance of the predicted geography location and the ground truth on the map is defined to represent the accuracy. The ranges of distance can use to evaluate the precision. The overall results are shown and compared in Table 4.

With the number of images is reduced slightly after removing, the accuracy of each range is a bit decreased. The result states that person removal is less effective to improve the model. Generally, when the quantity of images after removing a person raises, the result suggests that the original dataset without person removal should be better.

Next, there is some comparison among DeepLocation model and others introduced in the section of related work, GeoEstimation, Im2GPS, and PlaNet.

From the former research [11], GeoEstimation using multiple partitionings has no significant improvement for Im2GPS. However, DeepLocation and GeoEstimation both exploit the hierarchical knowledge at different spatial resolutions the localization accuracy can be indeed further increased. Figure 3 shows the inference of geographic coordinates on the same criterion in DeepLocation. The DeepLocation shows high accurate prediction at the distance between the location of the prediction and the ground truth. Consequently, DeepLocation is to a valid inference of the image's GPS position.

Last, we evaluate to compare the results of DeepLocation to the networks from PlaNet(900K) [9]. Table 5 compares the performance of PlaNet trained with 900,000 images from Flickr. From Table 5, DeepLocation has the same performance as PlaNet on the street and a little lower on the others. However, the model of PlaNet has a problem with a bit overfitting. The performance is better on the dataset with 126M photos with Exif geolocations from Google than on the datasets from INRIA Holidays and Flickr. While DeepLocation has more universality to images than PlaNet in the comparison.

7.2 Prediction with DeepLocation

This subsection will illustrate the result and performance using DeepLocation to make a prediction on the dataset from the S3. The dataset is processed in the pipeline and information of every valid image has been saved in the metadata. The total amount of images used for prediction is 410400. The final accuracy is depicted in Figure 3. The concrete accuracy is illustrated in Table 6.

The first part is how much improvement in geography hierarchy. The hierarchy architecture carried out in the Section 6.4 is generated from the different spatial resolutions. Consequently, the prediction for the densest subdivision can be refined by incorporating the knowledge of sparser representations. The accuracy of each fineness of cells and the hierarchy in the prediction of 35517 images are shown in Table 7.

The percentages inside the table prove that the geography hierarchy has improved the accuracy of prediction. The sparse has the best in the range of continent because the area of each cell is the largest and the scope of 2500 km can be the most tolerant. While the dense has the best in the range of street. It is turned out that the more serried partitioning has a higher precision in a rigorous range. When the range gets small, every fineness of partitionings shows lower accuracy. The accuracy of the sparse declines drastically and the dense one goes down slowly. The hierarchy can have the perfect combination of each partitioning and give a tardy decline in ranges.

8. VISUALIZATION

This section will describe the idea of the visual design and the process of developing a static website. First, considering the needs of users, the visual design will be discussed. After that, the process of developing the web application will be displayed.

8.1 Design

The main goal of the visualization part is to predict the position of the image and display the result and accuracy of the model prediction. In order to provide users with intuitive results and a better user experience, the requirements of the website are as follows:

- Select an image from given images
- Display the predicted location on the world map
- Compare the predicted location with the real location
- Show the distance between two locations
- Display the prediction accuracy of the model

Table 4: Result and Comparison of removal

Dataset	Loss	Continent(2500km)	Country (750km)	Region (200km)	City(25km)	Street (1km)
Removal	20.85%	33.9%	16.8%	6.4%	3.6%	1.3%
Original	20.87%	36.0%	17.2%	7.3%	3.8%	1.5%

Table 5: Result and Comparison of DeepLocation and PlaNet

Method	Continent (2500km)	Country (750km)	Region (200km)	City (25km)	Street (1km)
PlaNet	43.5%	21.6%	7.6%	3.8%	0.4%
DeepLocation	35.3%	16.5%	6.8%	3.6%	1.3%

Table 6: Result of DeepLocation

Continent	Country	Region	City	Street
35.3%	16.5%	6.8%	3.6%	1.3%

Table 7: Accuracy of DeepLocation

Fineness	Continent	Country	Region	City	Street
Sparse	35.2%	16.6%	6.3%	2.7%	0.3%
Medium	35.1%	16.2%	6.2%	3.0%	0.9%
Dense	34.5%	15.8%	6.1%	3.0%	1.1%
Hierarchy	35.0%	16.3%	6.4%	3.2%	1.2%

The website will randomly show 100 images from the dataset (with 410402 images). Users can select any image they want to predict or press the 'choose random' button to select a random image. After pressing the "Predicted Location" button, a model marker with the longitude and latitude information predicted by the model will be displayed on the world map. Compared with the real position on the ground, it can provide users with intuitive coordinate effects. The user can clearly see two markers displayed on the map, and the distance between the two markers will be displayed below the map. In order to let users better understand the effect of the model, the prediction accuracy will be displayed on the web page, including the continent, country, region, city, and street accuracy.

8.2 Static Web Application

To achieve the requirements of the design, the Bootstrap framework was chosen to develop the website and the open-source JavaScript library Leaflet[2] was used for developing the interactive map. By using the Leaflet library, it is easier to zoom-in and zoom-out on the map, which can provide a more dynamic and intuitional effect. We also use CSS to statically decorate the HTML web page, including the layout, fonts, and colors.

The usage of the web application can be seen in Figure 4 and Figure 5. Figure 5 shows the initial interface of the website and Figure 6 is the result interface after the prediction of an image.

9. CONCLUSION AND DISCUSSION

This section will state the conclusions from the result and have a discussion on the project.

At first, we can know useful detail from the dataset, YFCC-100M, and we give the well-processed metadata for future development. After the processing with the metadata, there are 48.386% of the full dataset having geographical coordinates. Besides, 0.047% of the images are lost in the multimedia commons S3 bucket. We provide the absolute path to access valid images in the S3 bucket. Then, it is a pity that we cannot improve the performance and accuracy of the model by person removal. However, if the strategy of filling portraits with noise can be implemented more efficiently to modify the images input when training the model, the result might be better. Next, from the website of DeepLocation, the visualization shows an intuitionistic presentation. It contains the accuracy of five criteria, continent, region, country, city, and street on the distance between the prediction and its ground truth. Users can choose any one of the images we offer randomly from the prediction dataset and click the button to display the result on the interactive map.

Due to the difficulties we meet during the implementation process, we obtain plenty of useful views and insights which will help us when doing similar work efficiently in the future. A high proportion of images from YFCC100M stored in Flickr are lost, so we tried to find a way to access the images stored in the AWS S3 bucket which the loss rate is far lower than Flickr. We successfully converted the Flickr URL into the address of the S3 bucket so that we could access the images as much as possible. Furthermore, we convert all of the Flickr URL links in datasets into the address of the AWS S3 bucket. We then use UDFs to make sure that the paths to access the images in the S3 bucket exist. Finally, we successfully stored the relevant data for 6 out of 10 datasets in 6 CSV files so that it may help the incoming students taking the LSDE course to conveniently access images. However, there are still some problems with the images stored in AWS S3 Buckets. For instance, some paths of images exist but the files are empty, or the size of images is smaller than 1KB which means these certain images are still not useful. Indeed, this kind of data error may occur in various conditions so that we must think carefully and comprehensively when dealing with data afterward.

Moreover, we still think about the further work we could do in the future. Firstly, there still be lots of ways to improve our model's performance, such as scene classification which considers the conditions of environmental settings (e.g. indoor, outdoor, natural, urban). Since we only use three partitioning files for hierarchy prediction, the model may perform better using more partitioning files for hierarchy prediction work. Secondly, Tensorflow.js can be used to run the pre-trained model in the browser. We believe that our project and visualization website could be more comprehensive by running our model in the browser. As a result, the users not only can choose the images from our dataset but also be able to upload their own photos for inferring.

[2]Leaflet: https://leafletjs.com

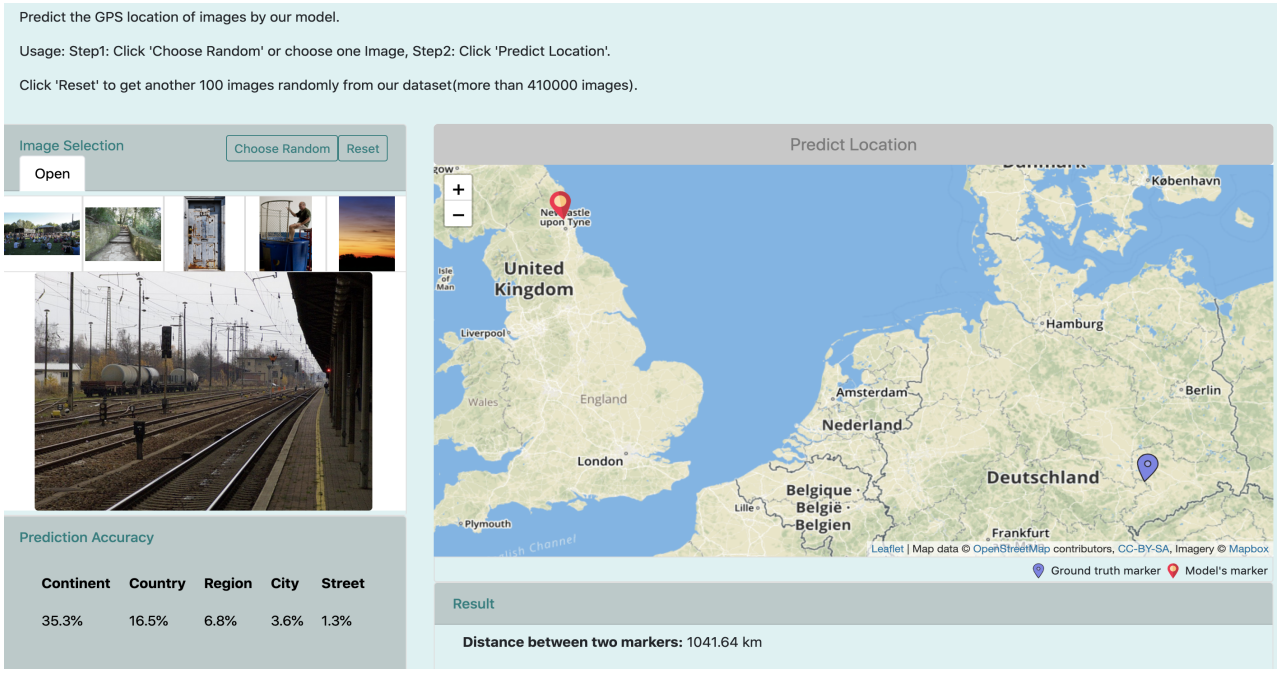

Figure 4: The process of DeepLocation Web Application

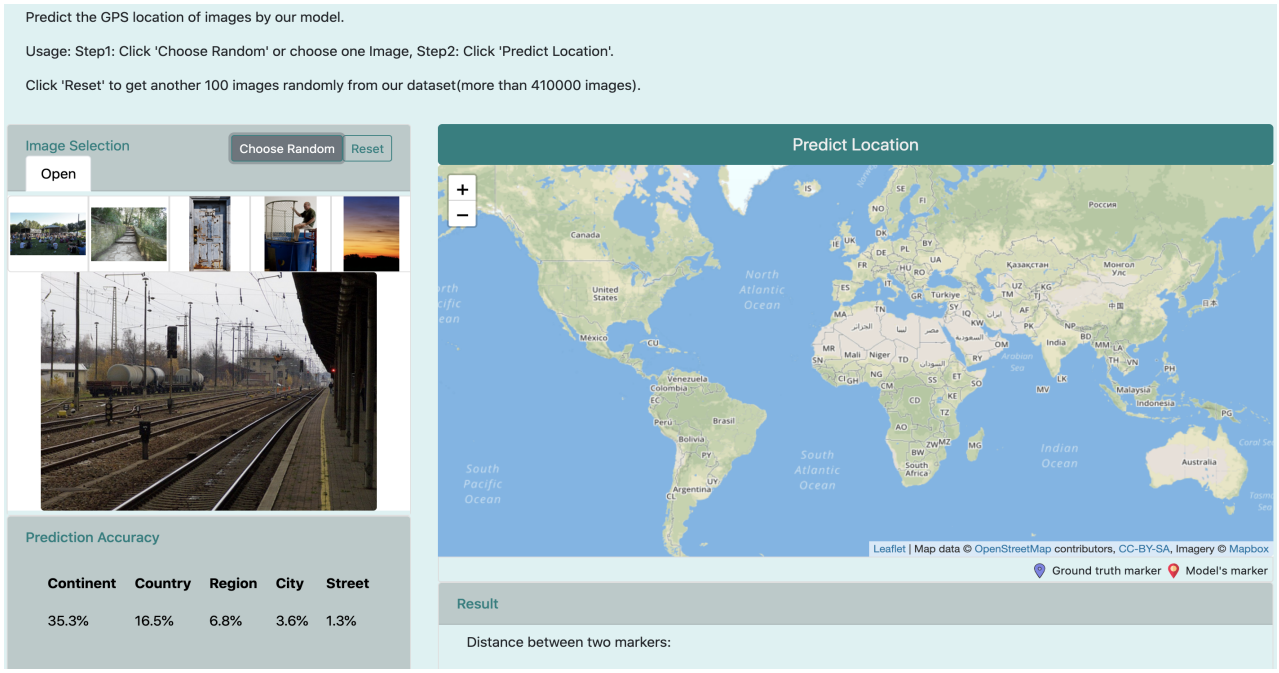

Figure 5: The process of DeepLocation Web Application

10. REFERENCES

[1] Y. Avrithis, Y. Kalantidis, G. Tolias, and E. Spyrou. Retrieving landmark and non-landmark images from community photo collections. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 153–162, 2010.

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

[3] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[4] R. Lienhart. Stump-based 20x20 gentle adaboost frontal face detector. *Intel Corporation*, 2000.

[5] C. Rother, V. Kolmogorov, and A. Blake. " grabcut" interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, 23(3):309–314, 2004.

[6] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[7] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE, 2007.

[8] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[9] T. Weyand, I. Kostrikov, and J. Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[10] A. R. Zamir and M. Shah. Image geo-localization based on multiplenearest neighbor feature matching usinggeneralized graphs. *IEEE transactions on pattern analysis and machine intelligence*, 36(8):1546–1558, 2014.

[11] W. Zhang and C. Xiao. Pcan: 3d attention map learning using contextual information for point cloud based retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12436–12445, 2019.

[12] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven. Tour the world: building a web-scale landmark recognition engine. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1085–1092. IEEE, 2009.

APPENDIX

A. VISUALIZATION AND DATA STORED DIRECTORY ON DBFS

- Visualization website
  In the repository of lsde2020_group3
- 10 csv files with S3 Bucket address to access images
  dbfs:/mnt/group03/dataset_S3

Table 8: Contribution Overview 1: Report paper

Part	Person	Percentage
Abstract	Chih-Chieh Lin	100%
	Haochen Wang	0%
	Kaixi Ma	0%
Introduction	Chih-Chieh Lin	100%
	Haochen Wang	0%
	Kaixi Ma	0%
Related Work	Chih-Chieh Lin	100%
	Haochen Wang	0%
	Kaixi Ma	0%
Method	Chih-Chieh Lin	15%
	Haochen Wang	85%
	Kaixi Ma	0%
Result	Chih-Chieh Lin	0%
	Haochen Wang	100%
	Kaixi Ma	0%
Visualization	Chih-Chieh Lin	0%
	Haochen Wang	0%
	Kaixi Ma	100%
Conclusion and Discussion	Chih-Chieh Lin	65%
	Haochen Wang	35%
	Kaixi Ma	0%
Latex	Chih-Chieh Lin	20%
	Haochen Wang	60%
	Kaixi Ma	20%

Table 9: Contribution Overview 2: Codes

Part	Person	Percentage
Initial Data processing	Chih-Chieh Lin	30%
	Haochen Wang	70%
	Kaixi Ma	0%
Training model	Chih-Chieh Lin	40%
	Haochen Wang	60%
	Kaixi Ma	0%
Inference process	Chih-Chieh Lin	40%
	Haochen Wang	60%
	Kaixi Ma	0%
Visualization	Chih-Chieh Lin	5%
	Haochen Wang	5%
	Kaixi Ma	90%

- 6 csv files with checking that ensures images exist
  dbfs:/mnt/group03/dataset_S3_exist