Capturing Yale's Campus: Geolocation Based On Visual Features

Linh Pham, Eric Yoon

Abstract

Geolocation data is invaluable for researchers and photographers alike, but many images come without geolocation data due to messaging and social media apps removing the information for privacy purposes. While previous research has been conducted on re-constructing geolocation data from an image's pixels, our approach focuses on a specific use-case of determining whether an image was taken at a prespecified location. To demonstrate this, we created a model that classifies whether an image was taken on Yale campus or not. To address the sparse and unlabeled nature of training photos taken on Yale campus, we propose three methods of data collection: (1) using public Instagram accounts, (2) using the Google Maps Places API, and (3) using our personal photo libraries. We then used transfer learning to develop a binary classification model, the best of which had a final test accuracy of 92%. We observed that the model is generally able to recognize Yale-like scenery and architecture, but fails for some specific types of images.

Background

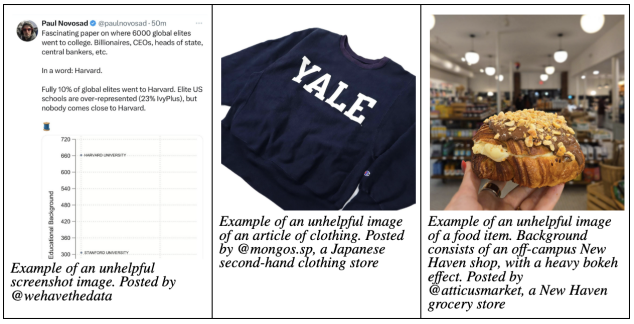
Geolocation metadata is particularly useful in images and is used in a variety of applications. Researchers rely on geolocation information to create specialized datasets; social media apps use geo-data for location-based features and recommendation algorithms; and everyday photographers rely on geo-data to organize their libraries. While most modern smartphone cameras automatically capture this data in machine-readable format, there are many cases in which this data can be lost. For example, when photos are uploaded to social media or messaging apps, EXIF and geolocation data are often stripped for privacy reasons. As such, recreating geolocation data from an image's pixels alone is a task worth researching.

There have been previous approaches to recreating geolocation data, including using scene-matching[1] or deep neural networks[2]. However, these papers focused on geolocation on a global scale, attempting to identify an image taken anywhere in the world. Our approach attempts to trade versatility for accuracy—instead of asking a general question of "where is this image taken," we attempt to see if a specialized model could be trained for a specific location that a data scientist has in mind.

Our project demonstrates how a binary classification model can be trained using publicly-available data to tell whether an image was taken in a specific region. To demonstrate this, we constructed a model that aims to tell whether an image was taken on Yale University campus. This project is as much about training a model as it is about finding an adequate volume of data. We present (a) three ways to find image data associated with a location, like a campus; and (b) an approach to training a binary classification model using transfer learning.

Data Collection

Yale Campus has been widely photographed, with many photos of campus taken and published online. However, there is no central repository of such photos; such a problem is true for other locations of interest of a similar nature.



Example of an unhelpful screenshot image. Posted by @wehavethedata

Example of an unhelpful image of an article of clothing. Posted by @mongos.sp, a Japanese second-hand clothing store

Example of an unhelpful image of a food item. Background consists of an off-campus New Haven shop, with a heavy bokeh effect. Posted by @atticusmarket, a New Haven grocery store

Figure 1: Unhelpful Yale Images

Figure 2: Examples of images downloaded from Instagram account $@yale\_vibes$

While many photos of Yale are published with associated keywords or hashtags, photos of architecture and landscapes are hard to distinguish from photos that do not depict the campus. An Instagram search for the hashtag '#yale' gives plenty of examples of unhelpful images:

As such, keyword or hashtag searches by themselves cannot provide a clean repository of campus images. We instead detail three methods we used to obtain 1000+ images of Yale campus.

Instagram

We discovered an Instagram account, '@yale_vibes', that solely posts pictures of Yale campus. Using a Chrome browser extension called "Mass Download for Instagram,"[3] we were able to download each photo.

While this dataset contained around 500 photos of Yale campus, there were some traits that could decrease the robustness of our model. First, the account is run by three students, making it likely that many photos were taken with the same model of camera. This could result in our model paying attention to the cameras' unique traits and artifacts instead of the features of the campus. Additionally, these photos appear to be enhanced and edited, which could also result in our model classifying any photos edited in a similar fashion as Yale campus images. As such, we decided to augment our dataset with more sources of images.

Google Maps

Google Maps provides a Places API [4] that allows developers to retrieve data, including user-contributed images,

Figure 3: Examples of images taken from Google Maps

of places on Google Maps. Our first approach was to use this Places API to fetch images of the location titled "Yale University." However, the API only allows 10 images to be fetched per place, so this approach wouldn't allow us to get more than 10 images.

We were able to obtain a list of on-campus buildings published by Yale [5]. Using the Places API text search endpoint, we found the Google Maps listing for each of these building names. Then, we fetched the first 10 images of each building. This method resulted in a total of 700 images of Yale campus, taken by Google Maps users.

Since these images were crowdsourced by Google Maps, they represented a wide range of camera models and editing styles, unlike our Instagram dataset.

Personal Photo Libraries

As Yale University students ourselves, we had many photos in our personal photo libraries that were taken on Yale campus. We decided to use these images to augment our dataset.

To facilitate the filtering process of photo libraries hosted in Google Photos, we created a program that could filter out images taken on Yale campus. The owner of a photo library is able to create a Google Takeout export of their Google Photos library, which includes .json files with metadata including geolocation latitude/longitude. Using a Python script, we were able to read this metadata and determine which photos were taken on Yale campus. Additional manual filtering was necessary to select for only photos of landscapes and architecture (as opposed to photos of people), but the filtering program greatly reduced the human workload necessary.

Out of 6122 photos in Eric's Google Photos library, 339 were taken on Yale Campus. Of these, 76 were deemed usable.

Out of 2,452 photos in Linh's Apple Photos library, 42 were deemed usable. (Most of her images were of her cat on campus, and very few of the actual campus itself.) Linh further collected 88 images of Yale campus from friends.

We decided to use these images from personal photo libraries to test the model under real-world conditions, rather than including it in our train/test/validate dataset.

Non Yale Photos

To train our binary classifier, we needed photos of landscapes and architecture that were NOT taken on Yale campus. We used the publicly-available Places365 [6] dataset to provide negative examples. Due to the small number of Yale images we had, we only needed a similar small number of non-Yale images; as such, we decided to use the Places365 validation set rather than the full set. We further filtered by label, only choosing photos that were similar to the ones we would see on Yale campus; examples include dorm room, office building, and dining hall.

Figure 4: Examples of images taken from personal photo libraries

Figure 5: Examples of images from the Places365 dataset

Classification Training

Training without Transfer Learning

We first attempted to train the model without using transfer learning. Both VGG and ResNet architectures failed to converge. When trained on an equal number of Yale and non-Yale images, our accuracy stayed at almost exactly 50%, even after 10 epochs of training. When we changed the ratio of Yale to non-Yale images, we noticed the accuracy also changed to match the ratio, indicating that the model was learning to always classify as non-Yale. As such, we decided to change our approach to take advantage of transfer learning.

Transfer Learning

Training a model from scratch is a time consuming process, and in the age of modern technology, can be considered redundant. This is where Transfer learning comes in. Transfer learning refers to the process of utilizing pre-trained models and modifying it slightly, in order to achieve the desired result. There are different methods of transfer learning, which depend on both the size and similarity of the dataset. This dataset was about moderately sized, but we considered the similarity to be high, as the differentiation between Yale architecture and other architecture is nuanced and complex. We primarily utilized a TensorFlow tutorial to help with code development [7]. More detail on the code can be found

in the tutorial, but we provide a quick summary of our process.

— Part 1 —

- Data download and augmentation

— Part 2 —

- Base model - either VGG or MobileNetV2
  - Freeze convolutional base
  - Feature extraction

— Part 3 —

- Fine Tuning: Where as before we only trained the few layers we added at the top of the base model, we are now updating the weights throughout the first few layers of the pretrained model.
  - Unfreeze and train from layer 100 and onwards

Part 1 We downloaded 1200 images from Place365 for our "non-Yale" classification. We applied an initial resize transformation of 500 x 500 as well as a conversion to a Tensor. We then applied data augmentation, specifically horizontal flips and rotation scaling in order to create more training images. This also helps in feature extraction.

Part 2 Then, we loaded in our base model (either VGG or MobileNetV2). We froze the convolutional base to prevent training of the transferred initial layers. Afterwards, we added a classification layer and a pooling layer to help classify images. We then trained for 10 epochs on this first part.

Part 3 After 10 epochs, we then started fine tuning the model. We unfroze the transferred initial layers, allowing them to further train.

Analysis

MobileNetV2

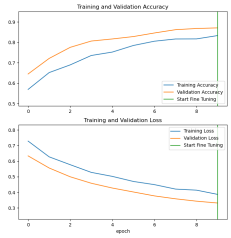


Figure 6: Accuracy and loss before fine-tuning

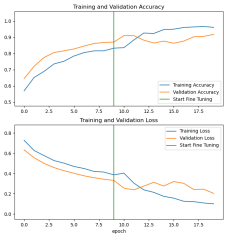


Figure 7: Accuracy and loss for first part after fine-tuning

Figure 6 shows the accuracy and loss for the first part of training, when we only added a classification task to the pretrained model. We can see that accuracy increases for both the training and validation set. This is an indicator of convergence, showing that our transfer learning approach is learning the data. After doing some research online, we found that the training accuracy is likely higher than validation accuracy as a result of using dropout [8].

In Figure 7, we can see that the training accuracy starts to surpass validation accuracy. The validation loss also rapidly increases and then remains above the training loss. This indicates some level of overfitting, with the model memorizing the training data rather than learning to identify features. However, the gap is decreasing with the amount of training; thus, it is plausible that after more epochs, the model would start to generalize better.

VGG16

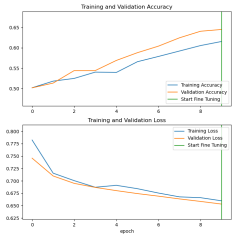


Figure 8: Accuracy and loss before fine-tuning

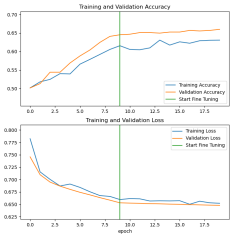


Figure 9: Accuracy and loss for first part after fine-tuning

In Figure 8 we can see that the training accuracy and validation accuracy remain relatively close in the beginning. Though these are not smooth curves, it indicates the model is learning well.

In Figure 9 we can see that training and validation accuracy starts to divide a little more. Both accuracies flatline, indicating an upper bound to the accuracy with respect to amount of training. Unlike VGG16, the validation accuracy remains above the training accuracy, and the validation loss remains below the training loss.

Results

Our final test accuracy for MobileNetV2 was 92%. While this seems like a success, the results from our loss and accuracy suggest overfitting. There are two ways to resolve this problem. We can add even more images, or go through our data again to make identifiable features of Yale. We acknowledge that our classification task is quite a complex one. While a classifier that differentiates, for example, architecture from scenery may have a high accuracy, our classifier differentiating Yale architecture from non-Yale architecture is a significantly harder task. Additionally, Yale has varied architecture, so the question is not as simple as classifying Gothic-style architecture as "Yale" and others as "non-Yale." For such a complex task, we believe adding more layers, or more importantly, adding more training data would allow our model to perform better.

The test accuracy for VGG16 was 65%. MobileNetV2 is a model meant for mobile applications, thus it is lighter and faster, but focuses less on actual accuracy. MobileNetV2 has about 4 million parameters and 314 million FLOPS, but in comparison, the VGG16 model has about 138 million parameters and 15 billion FLOPS. While VGG16 focuses on

Figure 10: True Positives (actual Yale, predicted Yale)

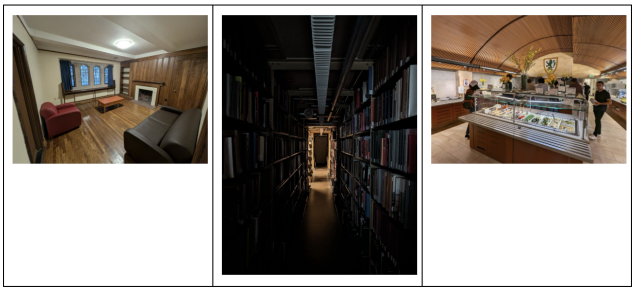


Figure 11: False Negatives (actual Yale, predicted non-Yale)

accuracy and training, MobileNetV2 chooses to sacrifice accuracy for speed. VGG16 is able to maintain high validation accuracy [Figure 4], indicating that it does not overfit. This is important as both VGG16 and MobileNetV2 trained on the same amount of images. Thus, the issue is not the lack of data, but on model construction. One way to improve the accuracy for VGG16 would probably be to train for longer. While 20 epochs was relatively fine for MobileNetV2, 20 epochs was most likely low for VGG16 due to the increased complexity of the model.

Real World Testing

Using our saved model checkpoints, we were able to test our model on images that were not included anywhere in our train/test/validate dataset. We observed a high sensitivity but a low specificity. We manually chose a classification boundary of 0.9 based on our own observations, although we acknowledge that this could have been chosen more empirically.

In a collection of 236 images that were taken in 2022 from Eric's photo library (before Eric entered Yale), 118 were predicted to be Yale and 118 were predicted to be non-Yale.

In a collection of 76 hand-picked images of Yale scenery from Eric's photo library, 63 were predicted to be Yale and 13 non-Yale.

Here are some examples of true positives, true negatives, false positives, and false negatives.

After manually inspecting all false negatives, we determined that these were "understandable" errors, as most of the false negative images were (a) nighttime photos, (b) photos with people in the foreground, or (c) photos of interiors (rather than exteriors).

Figure 12: True Negatives (actual non-Yale, predicted non-Yale)

Figure 13: False Positives (actual non-Yale, predicted Yale)

After manually inspecting false positives, we also understood why the model classified these as Yale images. Most of the false positives were landscapes; this indicates that the model may have learned to identify landscapes rather than Yale architecture.

Conclusion

We conclude that due to the complex nature of this task, and the limited availability of public data, creating a Yale vs. non-Yale classifier is extremely difficult. However, the task seems possible, as our model was able to classify images with a significant accuracy percentage. Our techniques to obtain data proved to be useful, and we stand by our choice to use transfer learning.

Additional Resources

You can view our code in our public GitHub repository.

Please install all requisite packages using pip3 before using our code.

- `data_scraping/google_maps/maps.ipynb`: Interactive notebook to get data from the Places API.
- `data_scraping/google_photos/photos.ipynb`: Interactive notebook to extract Yale images from a Google Takeout export.
- `data_processing.py`: Download non-Yale Places365 images; format images into the correct file structure and dimensions.
- `interactive_classifier.py`: Run our trained model on an image of your choice.
- `train_imagenet.py`: Train the MobileNet model from scratch.
- `train_vgg.py`: Train the VGG model from scratch.

References

[1] Hays, James, and Alexei Efros. *IM2GPS: Estimating Geographic Information from a Single Image*. http://graphics.cs.cmu.edu/projects/im2gps/im2gps.pdf.

[2] Weyand, Tobias, et al. "PlaNet - Photo Geolocation with Convolutional Neural Networks." *Lecture Notes in Computer Science*, Springer Science+Business Media, Jan. 2016, pp. 37–55, https://doi.org/10.1007/978-3-319-46484-8_3. Accessed 14 Dec. 2024.

[3] "Download Instagram Stories - Mass Downloader for Instagram." *Mass-Downloader.com*, 2024, https://mass-downloader.com/. Accessed 14 Dec. 2024.

[4] "Places API Overview." *Google for Developers*, 2024, https://developers.google.com/maps/documentation/places/web-service/overview. Accessed 14 Dec. 2024.

[5] "Building Abbreviations ¡ Yale University." *Yale University*, 2024, https://catalog.yale.edu/ycps/building-abbreviations/. Accessed 14 Dec. 2024.

[6] "CSAILVision/Places365: The Places365-CNNs for Scene Classification." *CSAIL Vision*, 2024, https://github.com/CSAILVision/places365. Accessed 14 Dec. 2024.

[7] "Transfer Learning and Fine-Tuning." *TensorFlow*, 2024, https://www.tensorflow.org/tutorials/images/transfer_learning. Accessed 14 Dec. 2024.

[8] "Higher Validation Accuracy, than Training Accuracy Using TensorFlow and Keras." *@yhenon on StackOverflow*, 15 May 2017, https://stackoverflow.com/questions/43979449/higher-validation-accuracy-than-training-accuracy-using-tensorflow-and-keras/43982656#43982656. Accessed 14 Dec. 2024.