Generating City Views from Geographic Coordinates

Thomas Arrizza tarrizza@umich.edu
Sinan Karabocuoglu skara@umich.edu
Misael Ortiz misaelo@umich.edu
Gunwoo Yong gwyong@umich.edu

1. Introduction

Street level images have been an increasingly popular element in our daily lives in recent years as their applications range from navigation to games like Geogusser. The main purpose of our project is to develop an effective GAN, GeoGAN, that will accept a pair of longitude and latitude geographical coordinates in order to obtain realistic images aligned to nearby big city landscapes. There are three main tasks to this project. The first task is creating training and validation datasets that include thousands of street level images that has been taken by individuals from worlds' biggest cities and has been made available via public geospatial data APIs. The second task is building a coordinates-based conditional GAN model for constructing realistic street-level city images. The third and final tasks are training a traditional CNN for scene recognition and using it as an objective metric of how realistic are the street level city images generated by GeoGAN. There are two primary takeaways that we are looking forward to in this project: (i) exploring newer and more advance GAN models than the introductory version introduced in the EECS 504 curriculum, (ii) gain insights on the performance of a CNN to localize GAN-generated fake images.

2. Related Work

The process of generating images based on coordinates can be thought of as a subset to the generation of images based on text. Reed et al. [8] generated images from text descriptions through GAN. Their generator first receives a noise vector concatenated with the projected text embedding, and their discriminator computes the score of discriminating power. They deployed this model for not only image generation, but also image synthesis. Zhang et al. [15] also provided variations given an image with the style from a text description. However, they introduced the conditioning augmentation, bringing more training pairs from a small number of image-text pairs, for robustness to slight changes from the conditioning manifold. Compared to the above two studies, Xu et al. [12] generated images from text using the attention mechanism, while more focusing on semantic features in text descriptions. Additionally, another work incorporates GAN with satelite image for predicting land usage, here results were able to display projected land cover[4]. Witnessing the impact of non-image features for image generation, this study aims to generate an image using geotagging information.

The data required to train a coordinates to image model is widely available in data collection platforms like Google Maps, but there are few compiled datasets that cover a sufficiently wide range of urban environments. The Google Street View Dataset[14] is compiled from a subset of the Google Maps Database and contains 62,058 high resolution images from Pittsburgh, Orlando, and Manhattan. These images are accompanied by Geo coordinates, camera orientation, and an RGB histogram. Similarly the StreetLearn Dataset[6] contains 143,000 panorama images of the same cities with latitude, longitude, altitude, camera orientations, date, and connected members. These datasets are thorough representations of their covered cities, but do not span the a very wide area. Other datasets like the SUN Database[11] and the Places Database[16] are large datasets of global scenes, but label images by class rather than coordinates.

Some prior CNNs for street view image localization separate scenes into classes based on various criteria and train to predict the class of new images[10, 1]. This has the advantage of being able to rank matching locations to an image, but also means the maximum localization is limited to the space of each class. Another common technique involves identifying important landmarks in an image and matching them to a nearest neighbor[13, 9]. Our technique is a classless approach that exploits the similarity between MSE loss and geographic distance.

3. Method

3.1. Dataset

While there are existing datasets for the purpose Geolocation learning, none are freely available and precompiled uniformly cover the coordinate space of all urban environments. Using the open source platform Mapillary, we compiled three datasets for training and testing: one 10k dataset, one 85k dataset, and one 10k dataset covering just the city of Detroit. For the 10k and 85k datasets, we used the Map-

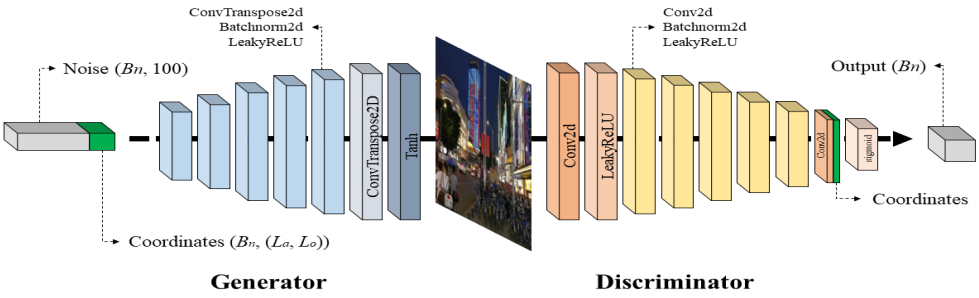

Figure 1. The proposed GeoGAN's architecture.

Figure 2. Dataset Sample

illary python SDK[1] to gather an equal number of images from each city with a population of at least one million[2]. Each image is re-scaled to 256x256 pixels and named using the convention 'latitude_longitude_imageID', with the image ID being a unique code for each image for the case that two images share the same Geocoordinates. The images are then sorted into folder based on their city of origin. Exact duplicates and near duplicates from each city are removed using pythons D-hashing algorithm[4] with a hash size of 8. A sample of these dataset are shown in Figure 2. The single city dataset is processed in the same way, except the result is contains no subfolders for subdividing the city.

The open source nature of the Mapillary platform comes with some advantages and disadvantages with regards to the integrity of the data that impacts the training of machine learning models. Images are collected by individual contributors instead of a single sponsor, such as the case with Google Maps. This means that the type of image can vary greatly, from sidewalk cellphone images to dash cam images. This is good for machine learning algorithms because it gives more diverse data and helps to mitigate over-fitting to a particular type of image. However, this also results in little control of the quality of the images. Each dataset is made noisy by a certain amount of images in each city being out of focus, partially blocked by obstacles, or biased by a small number of contributors. The bias created by this platform needs to be taken into account when reviewing the results of models trained on these datasets. Future revisions of the dataset should gather more images and from more sources to overcome this bias.

[1]https://github.com/mapillary/mapillary-python-sdk
[2]List of cities gathered from: https://geojson.io/#map=0.73/-24.7/0

3.2. GAN Image Generation

There are two networks constituting GeoGAN, that is a generator and a discriminator. The generator creates fake street images from noise vectors and coordinates. The discriminator measures the score for discriminating fake street images from real street images; the higher the score is, the better the discriminator classifies. Figure 1 displays the overall architecture of GeoGAN.

To make a fake street image, this study not only randomly samples a noise vector $z \in \mathbb{R} \sim \mathcal{N}(0, 1)$, but also makes coordinates as a one-dimension coordinate-vector c with the length 2. This study normalizes each component of the coordinate-vector c and concatenate the noise vector z with the coordinate-vector c. This concatenated vector passes five deconvolution blocks, including the batch normalization and the leaky-ReLU, and one deconvolution layer followed by the tanh. Finally, a fake street image with the 256x256 shape is generated through this generator network.

In contrast, there are seven convolution layers in the discriminator. Apart from the first layer and the last layer, the batch normalization and the leaky-ReLU are applied; the first and the last layer do not deploy the batch normalization, and the last layer's leaky-ReLU is replaced with the sigmoid. Furthermore, the coordinate-vector c is duplicated and is fused with the output from the layer prior to the last layer. This discriminator network returns a score representing confidence that an image is real or fake.

3.3. CNN Evaluation Model

Instead of having a single objective with characteristic details, like a dog photo or a human portrait, our street level images have multiple objectives with smaller and not necessarily characteristic details. Therefore, our priority on this CNN architecture was to capture as many small details as possible from all around the image with the hopes of identifying patterns of "random" details.

Our starting point was to identify existing popular CNN models that are commonly used in scene recognition and text related Computer Vision tasks. From a comprehen-

sive summary of deep learning techniques used in Image Captioning[3] pointed out two well-known models: AlexNet and VGGNet. After analyzing the architectures of both models separately, we decided to merge them at the middle where their channel sizes are both 256. Considering the big sizes of these models and our computation constraints, we decided to pick VGG-16 instead of VGG-19, and also eliminated some parts of these models that does not change the channel count. Moreover, the VGG-16 model includes more dense layers, which is useful for capturing the impact of the small feature changes. Thus our model architecture starts with AlexNet and switch to VGG-16 at the midway point. Finally, based on our results from Problem Set 5, we incorporated BatchNorm operation into the VGG-16 portion with the hopes of improving performance. The resulting architecture parameters can be seen Figure 3.

```
================================================================
Total params: 84,381,870
Trainable params: 84,381,870
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 190.36
Params size (MB): 321.89
Estimated Total Size (MB): 512.84
```

Figure 3. GeoCNN Architecture

4. Experiments

4.1. GAN Training

We trained the GAN architecture by drawing from the work and code of Image to Image Synthesis[12] and Generative Adversarial Text to Image Synthesis[7]. The training scheme inputs images and coordinates into the discriminator and used Pytorch's Adam optimizer on the results, from here the generator inputs noise and coordinates, these are then passed into the optimizer. We originally trained using batches of images, their corresponding coordinates, and images generated from noise. These resulted in the images seen on the left in Figure 4. From here we decided to more incorporate training elements from Text to Image Synthesis[7] and implemented generation of wrong coordinates to input to the discriminator and generation steps. The corresponding results can be seen on the right in Figure 4. Implementation of cross-entropy label smoothing was also attempted in order to reduce the discriminator confidence, but generated image results did not see improvement. Increasing epochs or number of images to 7000 also did not have noticeable effects on image quality. However, the images always return to a state with distinct skyline and grey lower section as seen in Figure 5; thus, adding images and epochs in training were no longer conducted.

Figure 4. Without Fake Coordinate in Loss(Left) and With Fake Coordinates in Loss (Right)

Figure 5. General Final GAN Image State Independent of Epochs

4.2. CNN Training

The CNN model was trained in sets of batches of 32 image/coordinate pairs with image shapes 3x227x227 and their corresponding geographical coordinates tuple in the form of $[latitude, longtitude]$. We used PyTorch's Adam optimizer and MSE loss as it is roughly analogous to the geographical distance between predictions and ground truths. After roughly 6 epochs the validation set began to converge and training stopped after 25 epochs.

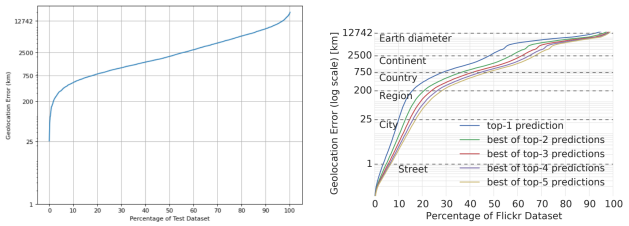

Figure 6. GeoCNN Accuracy (Left) and PlaNet Reference Accuracy (Right)

Exact physical distance in kilometers was calculated using the python Geopy library and recorded for the final batch of validation images. Figure 6 shows the sorted distance error values for the validation test set which gives what fraction of the dataset was localized to a given distance error. Approximate geographical scales are provided for comparison with the PlaNet model CNN training results[10] (shown in Figure 6). Approximately 1% of images were localized to within the city, 17% to within the country, and 55% to within the continent. This is slightly better performance than PlaNet at the continent level (48%), but significantly worse at city level (10.1%) and at country level (28.4%). The model was also validated against a subset of our relatively small dataset as apposed to the 2.3M images used to train PlaNet, so it is likely there is some overfitting to our dataset.

While the results are less impressive than existing models, the purpose of the CNN is to validate whether an image

belongs to a set of coordinates in a way that is believable to human beings. To make this comparison, we used statistics from the GeoGuessr game, where players are dropped into google street view and tasked with identifying where in the world they are[3]. Summing sets of five error distances scaled by the Geoguessr scoring algorithm (approximately[4] $5000 * 0.998^{error}$) gives a GeoGuessr. The average human player scores 6,998 while our algorithm got an average score of 2761 over 50 attempts. This places our model at the level of a novice Geo-guesser. For evaluation purposes, this is sufficient.

4.3. An Experimental CNN Model

Besides our main CNN model that relies on the credibility of the two underlying proven CNN models, we wanted to introduce some scene-recognition specific techniques with the hopes of improving performance.

The first technique we have implemented is the Spatial Unstructured Layer[5]. This process divides the feature map into "blocks" and then shifts them around with neighbouring blocks, altering the scene and the feature map as a result. As the number of blocks increases, the sizes of the blocks decreases. This results in more local alteration in the scene rather than a general alteration. This adds randomness to our street level images and introduces new image scenarios about a place that doesn't already exist in our train dataset(e.g a image that shows pedestrains crossing the street from left to right can be altered to crossing from right to left).

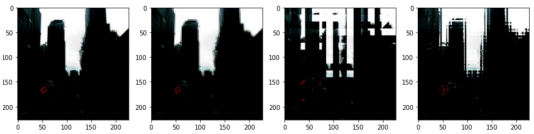

Figure 7. Spatial Unstructured Layer Examples with blocksizes = $[2^0, 2^{16}, 2^8, 2^{12}]$

The second technique we have implemented is the Spatial Pooling Layer[2]. Instead of applying a single max pooling with a fixed kernel and stride size, we apply three different size max poolings to the same feature map, and then concatenate the flattened results and pass them as a "single" flattened feature map into the dense layer. Our kernel and stride sizes are calculated based on the formulas $kernelSize = ceil(inputSize/outputSize)$ and $strideSize = floor(inputSize/outputSize)$, where we used $inputSize = 13$ and $outputSizes = [1, 2, 4]$. Our main motivation for introducing this technique is to capture different details on the feature map that might have been eliminated in the case of a different size max pooling. We chose to replace specifically the max pooling layer right before the dense layer starts as the dense layers are the most important part of the model for relating the details on the feature maps to predictions.

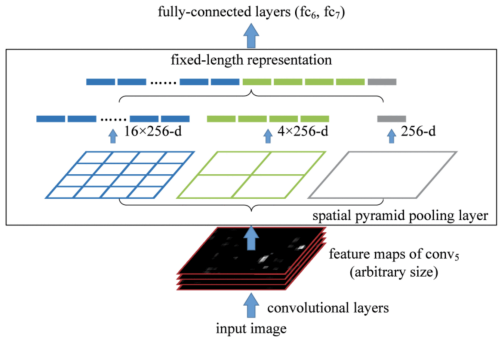

Figure 8. Spatial Pyramid Pooling[2]

Unfortunately, due to time constraints, we have preliminary results from this model's training but didn't have enough time to train it until it converged. Therefore, we don't have the final results to compare it with our original CNN model. However, the preliminary results didn't show a major improvement compared to original CNN results either. The most likely reason for this is that the spatial unstructured layer never really let the model train on the original images, so when it came to validation, our model was only trained for "unstructured" images. One potential solution to this would be to decide whether to pass a training image from the spatial unstructured layer by a Bernoulli random variable during every run, so our model will get a chance to train with original images more.

5. Conclusions

We presented the feasibility of the geotagging to image adversarial network and a functional CNN that localizes real and generated images. While it is difficult to localize images to geocoordinates using a simple feature regression model, it is more difficult to train an adversarial network given the same, relatively small, dataset. Typically Geolocating models leverage some other type of context, such as camera orientation or a histogram of notable landscapes, or GPS images. In the future, we would increase the size of the dataset and introduce additional modalities beyond simple images. For image localization, our biggest advantage over the prior implementations is that it doesn't limit its estimate by pre-defined location groups. This means that our model can localize to arbitrarily fine points, similar to the way a human would pick a point on a map. With additional improvements to the spatial unstructured layer mentioned in section 4.3 and complete training, we could expand on our CNN model's potential.

[3]GeoGuessr statistics can be found on their website: 5ab367f018399e275831ab1d

[4]According to the creator's comment, the exact error formula is subject to change during updates

References

[1] Chen D. et al. City-scale landmark identification on mobile devices. *CVPR*, pages 737–744. 1

[2] Kaiming He. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 4

[3] MD Zakir Hossain. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys*, 51(6):1–36, 2019. 3

[4] Manjusha L. Detect /remove duplicate images from a dataset for deep learning. *Journal of Positive School Psychology*, 6(2):606–609, 2022. 1, 2

[5] M. Bennamoun M. Hayat, S. H. Khan and S. An. A spatial layout and scale invariant feature representation for indoor scene classification. *IEEE Transactions on Image Processing*, 25(10):4829–4841, 2016. 4

[6] Mirowski P. et al. The steetlearn environment and dataset, 2019. DeepMind. 1

[7] Tinghui Zhou Alexei A. Efros Phillip Isola, Jun-Yan Zhu. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 3

[8] Xinchen Yan Lajanugen Logeswaran Bernt Schiele Honglak Lee Scott Reed, Zeynep Akata. Generative adversarial text to image synthesis. pages 1060–1069, 2016. International conference on machine learning. 1

[9] Zheng Y. T. et al. Tour the world: Building a web-scale landmark recognition engine. IEEE Conference on Computer Vision and Pattern Recognition. 1

[10] Philbin J. Weyand T., Kostrikov I. Planet - photo geolocation with convolutional neural networks. *European Conference on Computer Vision*, 9912:37–55. 1, 3

[11] Oliva A. Torralba A. Xiao J., Ehinger K. Sun database: Large-scale scene recognition from abbey to zoo. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3485–3492, 2010. 1

[12] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 1, 3

[13] Avrithis Y. et al. Retrieving landmark and non-landmark images from community photo collections. Proceedings of the 18th ACM International Conference on Multimedia. 1

[14] Shah M. Zamir A. et al. Image geo-localization based on multiple nearest neighbor feature matching using generalized graphs. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 36(8):1546–1558, 2014. 1

[15] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1

[16] Lapedriza A. Khosla A. Torralba A. Zhou B., Oliva A. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 1