Recognizing Cities from Street View Images

David Hershey and Blake Wulfe
Stanford University
450 Serra Mall, Stanford, CA 94305
dshersh@stanford.edu, wulfebw@stanford.edu

Abstract

In this research, we explore the ability of a computer to determine the location of a photo from its pixel data. GPS generally enables the geotagging of most images; however, for photos without GPS data, it can be difficult to estimate location. This problem can be difficult, as many cities have similar visual characteristics and locations, and photos can be arbitrarily uninformative. Despite these challenges, can a computer correlate the potentially numerous visual cues in an image with that image's city of origin? We train a convolutional neural network, called LittlePlaNet, to answer this question. We show that our approach achieves super-human accuracy on a dataset of photos take from Google Street View in 10 different cities. Furthermore, we also explore the limitations of our dataset by testing our model on photos from Flickr, which contains geo-tagged images of everything from landmarks to people.

1. Introduction

Since the advent of GPS technology, geo-tagged photos have become ubiquitous. Most pictures taken with smartphones have coordinate data included in their metadata. This information has a wide range of uses, from simply giving a bit more information to friends and family to enabling services like Google Earth to display photos at many locations worldwide. This technology is popular for its ability to connect people more intimately to photos taken around the world.

Photos taken without GPS cannot be easily geo-tagged. Traditionally, these photos must be manually labeled in order to provide location data. This is a difficult task, as many photos do not contain obvious distinguishing landmarks or features. Humans achieve some degree of success despite these challenges by looking for cues that reflect the location of origin of the depicted scene. This project aims to assess the capability of convolutional neural networks (CNNs) to automatically learn to identify these features, and, by extension, the originating city of images.

In our formulation of this task we present the computer with street-level images taken in 10 different cities, and ask it to output the city in which the image was taken. As we have formulated this as a classification problem, we exclusively test for correct classification, so simply guessing a nearby city gives no credit.

The final trained neural network, which we have named LittlePlaNet, achieves super-human accuracy when tested on our dataset. We employ multiple feature-visualization techniques to attempt to extract information about how LittlePlaNet makes its classification decisions. These techniques have allowed us to identify some unique attributes of each of the cities considered.

We analyze the performance of the network further by introducing a secondary dataset for testing. LittlePlaNet maintains its accuracy on similar street-level photographs from outside of the original dataset. As expected, the network does not generalize well to photos unlike those on which it was trained, for example, wide angle landscapes or portraits.

The rest of this paper is organized as follows: section 2 covers related work, section 3 our methods, section 4 the datasets we use, section 5 our results, and section 6 feature visualizations.

2. Related Work

Historical attempts at solving the geo-location problem have focused largely on feature retrieval. The most notable application of this method was Im2GPS [3] which uses image features to compare an image to similar images from a dataset of millions of Flickr photos. Using similarity scores, this method can achieve reasonable geo-location accuracy, classifying roughly 25% of photos to country-level accuracy.

In an attempt to solve the problem of sparse data in rural areas, [1] did work utilizing aerial photos for geolocalization. This work uses a Siamese CNN to attempt to match ground photos to aerial photographs with known locations.

Until recently, relatively little prior research has applied CNNs to this task. In a 2015 project also for CS231n, Hong and Peddada attempted this task with a subset of the CRCV dataset [9] as well as geo-tagged Flickr photos. The researchers achieved super human-level performance in coarsely labeling cities in the CRCV dataset with a CNN trained from scratch [4].

A recent paper from Google used a CNN for a similar geo-localization task [8]. They developed a platform named PlaNet (hence the name of our network), which classifies photographs into variably sized regions across the globe. They used a dataset from Flickr which contained geo-tagged photos with a wide variety of subject matter. Because of this, they achieve very high accuracy on images with landmarks or cultural subject matter, which is more likely to be the subject of a photo on Flickr. They attain state of the art accuracy in global geo-localization, managing to classify pictures to city-level accuracy 25% of the time.

3. Methods

We formulate the geo-location task as a classification problem. For this, we have selected 10 cities from culturally and geographically diverse locations on the globe. The input to our CNN is the raw pixel data from an image taken in one of these cities. The output is a probability distribution across the 10 cities, which is then translated into a single prediction city. Other formulations of this problem are possible, including regression onto coordinates, though classification is the most natural fit for our dataset.

3.1. CNN Architecture

We have elected to use a GoogLeNet model [7] pre-trained on the Places205 dataset [11] and the neural network library caffe [5] for the implementation. We used a model pre-trained on a scene-classification dataset because we believed this data would most closely resemble the images in our dataset, and we selected the GoogLeNet model because it achieved the highest available accuracy on the Places205 dataset.

This network is composed of a series of ”Inception Modules”, which are consist of parallel convolutions of different breadth. The intuition behind convolutional filters is that they capture spatial relationships within their breadth. Therefore the key idea behind the inception module is that by using simultaneous filters of different sizes, you can capture different resolutions of information at each stage of the network. This allows you to detect and operate on both fine features and features involving larger parts of the image. These features are then concatenated into a single layer output. There are also occasional max pooling layers which serve to intermittently reduce the dimensionality of the data flowing through the network.

The network ends with a single fully connected layer which operates on the features extracted from the final inception module. This layer computes scores, $\mathbf{f}$, for each class, which correspond to how activated the network is for each class. These class scores are then fed into a softmax classifier according to the cross-entropy loss:

Figure 1: The GoogLeNet CNN Architecture [7]

$$\mathbf{L_i} = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \tag{1}$$

Where $\mathbf{L_i}$ is the loss of the i^{th} example and f_j is the score of the j^{th} class. This serves as both the loss function for training, and also intuitively as the probability of each class at test time.

3.2. CNN Training

The network was trained using Caffe [5] on a NVIDIA GRID K520 GPU. This straightforward interface allows for rapid training and easy network and weight modification.

To speed training, we used transfer learning for our network. This method involves using the weights of a CNN trained on a different dataset in order to avoid having to train an entire network from scratch. As mentioned above, by selecting a network trained on the MIT Places205 dataset [11], which contains photos of natural scenes, we hope to utilize some of the scene recognition abilities of the pre-trained layers.

One of the significant decisions in transfer learning is determining which layers to retrain. As a baseline, we first only retrained the fully connected layer of the architecture. This fine-tuned network was able to achieve 52% validation accuracy after 1 epoch. We then replaced weights in the last inception module and trained for 4 epochs on the dataset, reducing the learning rate when validation accuracy stagnated. For optimization, we used stochastic gradient descent with initial learning rate of 1e-5 (multiplied by a constant factor for later network layers) and momentum of .9, training in batches of four images (chosen as the maximum that would fit on the K520), and a small L2 regularization penalty.

4. Dataset and Features

We obtained 100,000 street-level images using the Google Street View API [2] [1]. Data points were generated by requesting Street View images from random locations within 25 kilometers of the city center of the following cities: Barcelona, Washington DC, Detroit, London, Moscow, New York, Paris, Rio de Janiero, San Francisco, Sydney. The training set has approximately 90,000 of these images selected at random, the validation set has 5,000 images, and the test set has 5,000 images.

The images were downloaded as 256x256 images, at the API's minimum aspect ratio. This guarantees that the image contains a large deal of data from the scene; however, at this zoom and resolution it is difficult to make out fine details such as text. It is possible the features present in high-resolution images could contribute to higher performance.

At training and test time, the network subtracts the mean train-set image from each image, randomly mirrors some images, and crops the images to 227x227. This effectively helps to artificially increase the size of the training set to further prevent overfitting. Only this cropped pixel data is used as input to the network, and during testing time a random crop from the image is used to make a prediction for that image.

5. Results

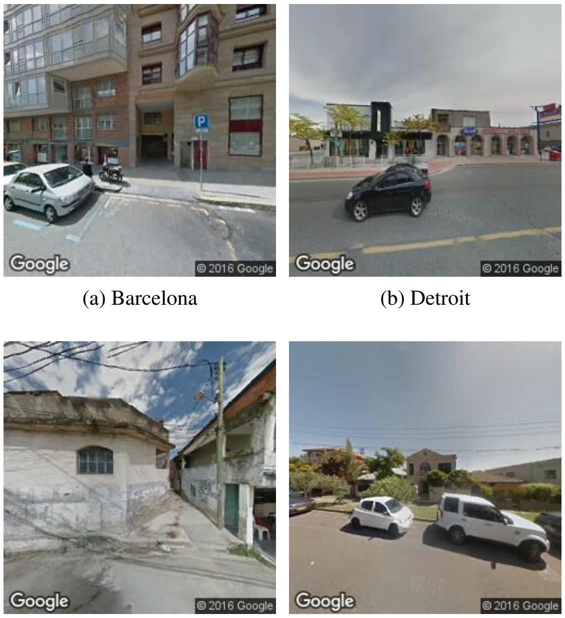
5.1. Test Accuracy

Table 1 gives the classification accuracy for LittlePlaNet on the Street View dataset. LittlePlaNet achieves superhuman accuracy on this dataset, at a surprisingly high 75%.

Table 1: LittlePlaNet Classification Accuracy

Method	Classification Accuracy
Random	.1
Human	.272
LittlePlaNet	.753

Table 2 shows the accuracy of LittlePlaNet on each city in the Street View dataset. Notably, some cities display significantly different classification accuracies. There are a number of reasons that could account for these differences, such as similarity between cities, or noisy data. Both of these explanations are explored below in our discussion of TSNE visualization.



(a) Barcelona

(b) Detroit

(c) Rio de Janeiro

(d) Sydney

Figure 2: Randomized Street View Images from Four Cities

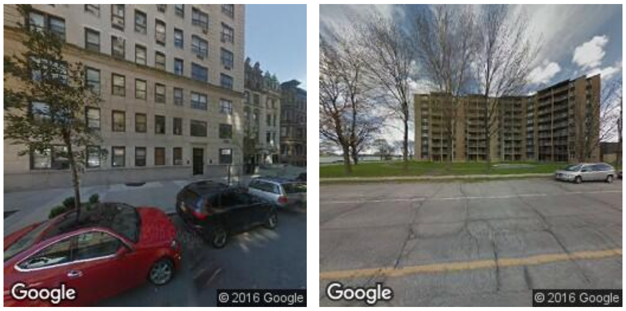
Table 2: LittlePlaNet City-by-City Accuracy

City	Classification Accuracy
Barcelona	.785
Washington DC	.820
Detroit	.875
London	.735
Moscow	.840
New York	.405
Paris	.479
Rio de Janeiro	.640
San Francisco	.615
Sydney	.785

Figure 3 shows example images that are correctly classified by LittlePlaNet. These two photos, although not easily distinguishable by humans, are the type of photo that LittlePlaNet generally classifies correctly. The New York City photo contains distinctive architecture. The Detroit photo has a chracteristic type of road with very few cars and a unique building. Across the dataset, photographs with clearly visible cars, buildings, and roads are generally correctly classified.

Figure 4 shows images that were incorrectly classified by LittlePlaNet. Analyzing these photos, we can see the types

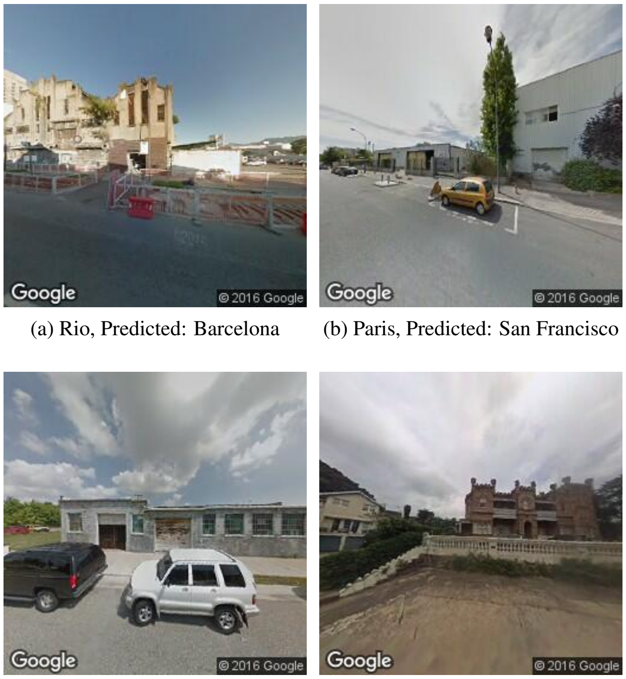
[1] Project code can be found at https://github.com/dmakian/LittlePlaNet



(a) New York

(b) Detroit

Figure 3: Images Correctly Classified by LittlePlaNet



(a) Rio, Predicted: Barcelona

(b) Paris, Predicted: San Francisco

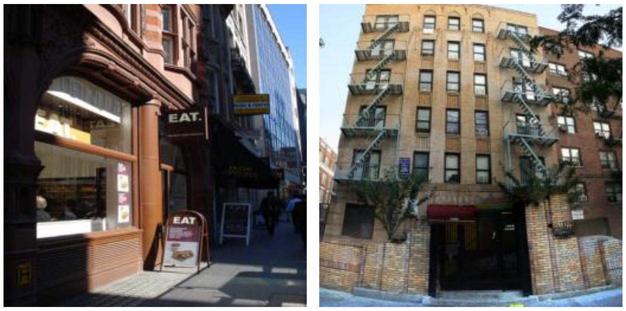
(c) Detroit, Predicted: Washington DC

(d) Sydney, Predicted: Paris

Figure 4: Images Incorrectly Classified by LittlePlaNet

of images that the network struggles to classify correctly. Image (b) shows a photo from Paris that is not in the architectural style of Paris. Alternatively, image (d) shows an image of Sydney that has architectural elements very similar to those of Paris! These types of images are particularly difficult for both humans and LittlePlaNet to classify correctly.

5.2. Performance on Flickr Dataset



(a) London

(b) New York

Figure 5: Correctly Classified Flickr Images

In order to test the robustness of our model, we've introduced a secondary dataset composed of 800 geo-tagged images from Flickr. These images were taken in one of four of the cities included in our initial dataset. Notably, these images are not just street-level images. They also include a wide range of completely unrelated photographs such as portraits or artwork. We have tested our network on this dataset without filtering the dataset in any way.

Table 3: Accuracy on Flickr Dataset

City	Accuracy
Paris	.125
New York	.14
London	.235
Sydney	.18
Overall	.17

As one would expect, our network does not perform well on this very different dataset. Notably, the performance of the network is above random, which is likely due to street-level images or images of buildings in the Flickr dataset. Examples of photos that the network correctly classified can be seen in Figure 5.

6. Feature Visualizations

6.1. t-SNE Visualization

t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used to visualize extracted features in a low-dimensional space while maintaining information about the distance between those features in higher dimensions. Using the finetuned network, we extracted features after the final pooling layer from 1000 randomly selected images and visualized the 2-dimensional embeddings in figure 6.

6.1.1 Least Distinct Cities

All of the cities inhabit relatively distinct regions within this space. The city that is least distinct is NYC. This is in part due to the fact that fewer images of this city were

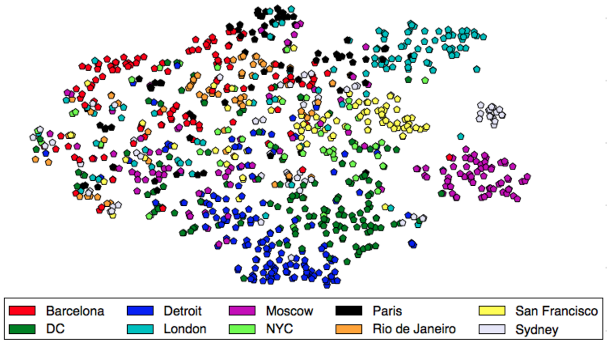


Figure 6: Visualization of features using t-SNE

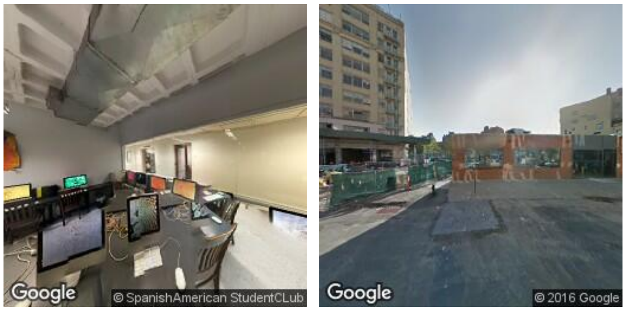
randomly selected, but may also be due to some other underlying cause because NYC also exhibited the worst classification accuracy. Figure 7 compares two images of NYC, one from the densely-populated, center-right region and one from the far left.

The out-of-region photo (a) was taken inside an office building and perhaps partially explains the poor performance of the classifier on NYC images. If indoor locations in NYC are more heavily accounted for within Google Street View than in other cities, then it is likely that our dataset contains a higher fraction of indoor images from NYC than other cities.

Whether or not this phenomenon, if present, adversely impacts classifier performance on images from NYC depends upon the network's ability to recognize indoor images as such. If the network is able to recognize indoor images, then their existence in the dataset would not necessarily diminish performance, because the classifier could learn to predict NYC for these images. However, if the classifier is unable to recognize indoor images (for example, because they vary widely in appearance), then its prediction on such images will likely be close to a random guess. We conjecture that this second situation is more likely to be the case for our classifier due to the relatively few indoor images in the dataset. This potentially explains the poor performance of LittlePlaNet in classifying images from NYC.

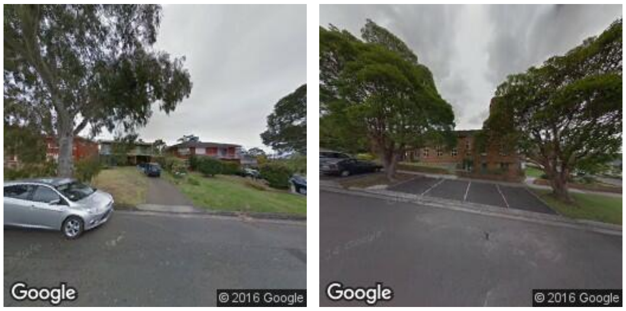
6.1.2 Most Distinct Sub-Cities

Also interesting to note is the relative location of certain cities in this low-dimensional space. The primary groupings of Sydney and Moscow, for example, are particularly distinct. Figures 8 and 9 show images from each city in these far-right regions. These images are quite distinctive, with those from Sydney containing the same tree species and similar houses, and those from Moscow both containing large, white buildings.



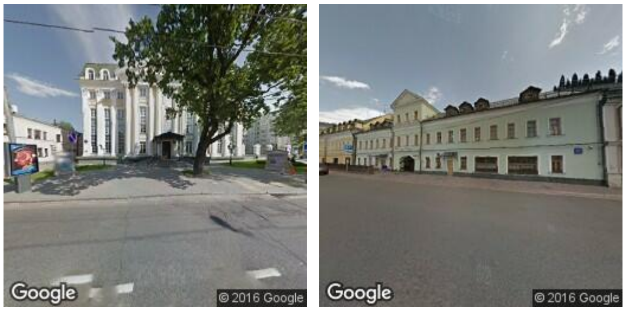
(a) Out-of-region (b) In-region

Figure 7: Images of New York City. (a) An image that exists in the t-SNE graph far away from the cluster of NYC images. (b) An image from inside the t-SNE NYC cluster.



(a) Sydney (b) Sydney

Figure 8: Two images from the primary t-SNE Sydney region.

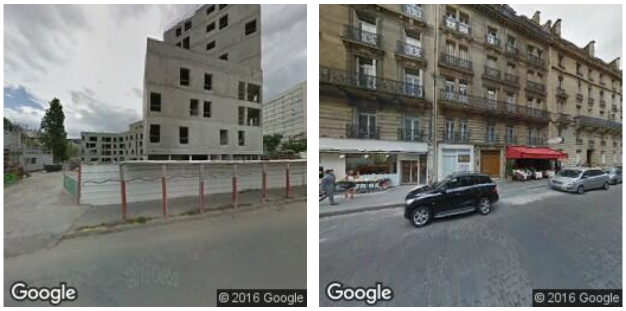


(a) Moscow (b) Moscow

Figure 9: Two images from the primary t-SNE Moscow region.

6.1.3 Multimodal Cities

When plotted individually, some cities exhibit multiple, distinctive regions. This makes sense given that many cities have neighborhoods with different characteristic traits. As shown in figure 10, Paris exhibits two primary ”modes” in



(a) Mode 1

(b) Mode 2

Figure 10: Images representative of the primary subregions of Paris. Image (a) from around the Montreuil Commune in East Paris. Image (b) from around the Levallois-Perret Commune in North Paris.

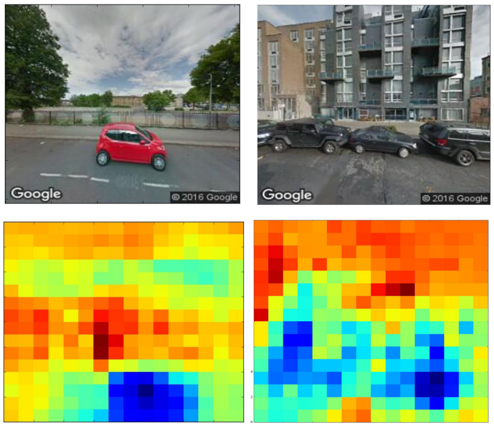


Figure 11: Occlusion heatmaps showing the effect of occluding a region of the image on the class score.
Left: London, Right: New York

the 1000 image sample considered.

6.2. Image Visualizations

The filters beyond the first layer of the CNN can be difficult to visualize as they are not operating directly on pixel data. As such, other methods must be used to extract useful information about how the CNN is making its classifications. One such method is occlusion, used by [10]. This method involves sliding an occlusion box over an image and performing a forward pass of the occluded image through the network. By comparing the output score for the desired label as each region is occluded, we can determine which regions of the image are most important for making each classification. The output of this method is a heatmap which effectively demonstrates the impact of occlusion on each region of the image. An example of this visualization is shown in Figure 11.

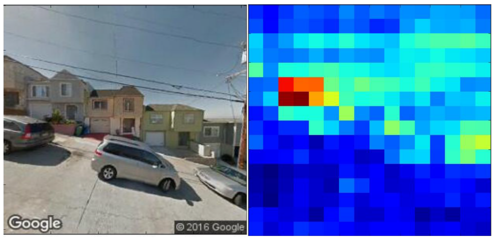


Figure 12: Inclusion heatmap showing the effect of only showing the network a region of the image on the class score

These heatmaps from two different cities indicate that cars are one of the important features used to make classifications in our dataset. This is a logical feature to use, as cars are both common in our Street View dataset and also informative of the city of origin. By exploring more heatmaps, we have also seen that weather patterns such as cloud cover can have an important impact on the output.

Another technique we developed, which was inspired by the occlusion method, is the method of inclusion. In this method, we occlude the entire image except for a small region. We then slide this included region across the image in a method very similar to occlusion and use this to generate a heatmap. This result of this method is displayed in Figure 12

In this image of San Francisco, it is clear that the distinctive houses activate the San Francisco score when they are the feature left in the image. This indicates that our network may make use of architectural patterns while making classifications.

6.3. Saliency Maps

Another popular visualization technique is the use of saliency maps, a technique developed by [6]. In this method, a random noise image is optimized via gradient ascent to maximally activate a class score for a desired class. Patterns often form in the resulting image that resemble image features characteristic of the target class.

In Figure 13, we show the result of this method for the San Francisco label. Although there is clear structure in this image, there is no human-discernible meaning to it. This is unsurprising, as unlike ImageNet labels which involve a single target in the image, many factors contribute to the label of a city. As already shown, cars, architecture, and weather can all impact the output.

7. Conclusion

In this paper we considered the geolocalization task of predicting which city an image originated from using only the pixel data from that image. Despite the significant challenges posed by this problem, we showed that a CNN can learn to correctly predict city of origin with super-human

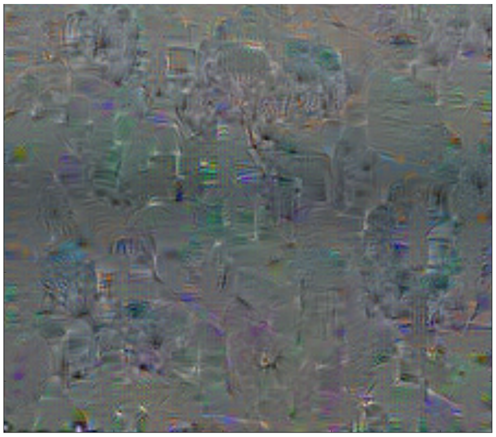


Figure 13: Maximally Activating Image for San Francisco

accuracy.

Part of this success can be attributed to our particular formulation of the problem - i.e., many of our design decisions make this problem easier than it might otherwise be. For example, we focus on relatively few, highly-distinctive cities, allow images taken near one another to be in the training and test sets, and limit the boundaries of cities to a relatively small region. These decisions make the problem easier than it might otherwise be, but, nevertheless, it still remains quite challenging, and the success of our model in this task provides yet another demonstration of the impressive capabilities of CNNs.

We additionally attempted to analyze what the model learned in order to produce such accurate results. In plotting the learned representations in lower-dimensional space, we found that the learned features effectively differentiate images into separate regions, often even within cities. Images in these regions exhibited highly similar traits, often containing nearly identical buildings, trees, or other objects. We also visualized the learned features using occlusion, inclusion, and maximally activating image techniques. While the latter did not produce any discernible patterns, the former two demonstrated the significance of particular objects in making correct predictions

These visualizations and our cross-validation on the Flickr dataset help confirm the ability of the model to learn to differentiate cities based upon reasonable evidence, and not just upon artifacts of the manner in which images were collected or other unreliable correlations.

References

[1] S. Belongie and J. Hays. “Learning Deep Representations for Ground-to-Aerial Geolocalization”. In: *CVPR* (2015).

[2] *Google Street View Image API*. Feb. 2016. URL: `https://developers.google.com/maps/documentation/streetview/`.

[3] J. Hays and A. Efros. “IM2GPS: estimating geographic information from a single image”. In: *CVPR* (2008).

[4] Peddada Hong. “Geo-Location Estimation with Convolutional Neural Networks”. In: (2015).

[5] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093* (2014).

[6] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *CoRR* abs/1312.6034 (2013). URL: `http://arxiv.org/abs/1312.6034`.

[7] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[8] Tobias Weyand, Ilya Kostrikov, and James Philbin. “PlaNet - Photo Geolocation with Convolutional Neural Networks”. In: *CoRR* abs/1602.05314 (2016). URL: `http://arxiv.org/abs/1602.05314`.

[9] A.R. Zamir and M. Shah. “Image Geo-localization Based on Multiple Nearest Neighbor Feature Matching using Generalized Graphs”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PP.99 (2014), pp. 1–1. ISSN: 0162-8828. DOI: `10.1109/TPAMI.2014.2299799`.

[10] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901 (2013). URL: `http://arxiv.org/abs/1311.2901`.

[11] Bolei Zhou et al. “Learning deep features for scene recognition using places database”. In: *Advances in neural information processing systems*. 2014, pp. 487–495.