Humanitarian Mapping with Deep Learning

Lars Roemheld
Stanford University
roemheld@stanford.edu

Abstract

This paper uses satellite imagery to support map creation in the developing world. I gather my own dataset by downloading imagery released through the U.S. State Department's MapGive project and using map data provided by the Humanitarian OpenStreetMap Team. Using this data, I train several Convolutional Neural Network models designed after the SegNet architecture to perform semantic image segmentation. The output of these models are map-like images that may in a later step be used to reconstruct map data, accelerating the work of online remote mapping volunteers. This paper details progress made towards this goal; my best model's pixel-average test accuracy of about 69% does not allow production use yet. I conclude on notes for future work.

1. Introduction

When natural or other disasters strike, the most endangered people often live in areas that are not well-mapped: in the developing world, whole regions are lacking detailed information on where exactly people live, and where critical infrastructure is located. Recognizing the need for such data in humanitarian relief missions, the Humanitarian OpenStreetMap Team (HOT)[1] has been founded as a team within the open-source mapping community OpenStreetMap[2]: it uses online volunteer workers around the globe to create missing maps in areas that need it most. Amongst other achievements, HOT has been recognized for its impact in coordinating relief missions after the 2010 earthquake in Haiti.

The importance of humanitarian mapping efforts has been recognized by the U.S. Department of State in the form of the MapGive initiative[3], which makes available U.S.-licensed satellite imagery for humanitarian mapping efforts. This paper aims to support the volunteer efforts within HOT and MapGive by using Convolutional Neural Networks (ConvNets) to accelerate the mapping process and reduce necessary volunteer involvement. To this end, I use MapGive imagery and HOT-created maps in my training and testing steps.

A large part of this project has been integrating with existing HOT technology to collect sufficient training data. The following section will detail the process of data collection and describe the collected dataset. Section 3 presents related work and literature, motivating my methods that I present in section 4. Preliminary results are shown in section 5. Section 6 concludes this paper and discusses future work.

2. Data

To integrate with existing HOT infrastructure and allow potential production use of my results, I closely followed the existing HOT workflows: the team's current interface is the "OSM Tasking Manager," which breaks larger areas into individual map-squares that can be assigned to individual volunteers. For map creation, supporting satellite imagery is overlaid in the project's map editor, allowing tracing of roads, buildings, waterways, etc.

Once a task has been worked on, it can be marked as "done". This lets other volunteers know that the task is ready for validation; once another volunteer has marked the task as "validated," it is considered finished and removed. For my data collection, I crawled the task manager and separately downloaded the satellite imagery and finished map data for "validated" map-squares from MapGive and HOT tile-servers. For easier handling of the data, I also downloaded raw map data (geojson) through OpenStreetMap's Overpass API.

Using this data pipeline, it is possible to (a) learn from future volunteer work and (b) feed ConvNet results back into the existing HOT task manager as "done," prompting human verification before results are released. If a high enough accuracy is achieved, this may significantly speed up the volunteer mapping process.

As my training data, I downloaded two finished HOT projects in full, spanning roughly 150km^2 in Haiti and

[1] https://hotosm.org/
[2] http://www.openstreetmap.org/
[3] http://mapgive.state.gov/

label	% of pixels
nolabel	79.4%
building	11.2%
road	4.0%
farmland	2.6%
wetland	1.6%
forrest	1.0%
river	0.2%

Table 1: Per-pixel distribution of class labels in dataset.

Colombia. At a resolution of 0.6m/pixel,[4] this corresponds to 10403 individual tiles of 256*256 RGB pixels each (squares of 153m side length). I preprocessed the satellite images by subtracting the mean image (i.e. the mean color per pixel location), which had the side-effect of partially washing out watermarking ("Digital Globe," the satellite image provider). For each of the squares, I then downloaded the raw map data and a map image according to latitude/longitude bounding boxes.

From the map image I generated my final labels by replacing colors according to a map legend. This added noise to my image labels, as artifacts such as text-labels, map icons, anti-aliasing, etc. could not be fully recovered from the map image; unfortunately, custom map renderings of the OpenStreetMap data were not available in a straightforward way. This means that labels are noisy, adding to slight label misalignment due to bounding-box inaccuracies and existing mapping errors. Figure 1 shows a sample of my data.

The data suffers from some class imbalance, as shown in table 1: the majority of pixels do not have any label associated with them. This is due to foresty "bush" areas that make up large patches between urban settlements.

I randomly split the dataset into a training set (85%), a validation set (10%) and a test set (5%) and validated that the per-pixel label frequencies are roughly equally distributed amongst all sets.

3. Related work

Xie *et al*. [15] report remarkable results of their ConvNet architecture to predict regional poverty levels based off satellite imagery: using a pre-trained VGG network [13], they first train an image-classification model to predict night-time light intensities from day-time satellite imagery. In a second step, they transfer the learned features into a direct predictor of poverty levels, achieving almost survey-level results. The authors rely heavily on transfer learning to compensate for label sparsity, and they use a resampling method to counteract class imbalance. As their prediction depends more on macro-level features, they use Google Maps[5] satellite imagery at a low resolution of 2.4m/pixel,[6] – this is 16-times less pixels for the same geographic square than used in this paper. In what appears to be a very similar approach, Facebook researchers have recently reported impressive results obtained by significantly larger datasets and networks across a wide variety of geographies [7].

Castelluccio *et al*. [4] present an image classification model that mainly uses the publicly available UCMerced dataset [16]. They employ the CaffeNet[7] and GoogLeNet [14] architectures, pre-trained on ImageNet challenge data [12]. Their strongest result, a version of GoogLeNet, achieves 97% accuracy on the dataset's 21 balanced classes, which range from `dense residential` to specific (and easily recognized) classes such as `baseball diamond`.

Bergman [3] uses the success of image-classification models to produce map images from aerial photos obtained from Google Maps: by classifying the center pixel of an image patch and moving this patch over the input image, the author produces a semantic segmentation which he then post-processes.

Mnih and Hinton [11] specifically address the issue of noisy labels in map segmentation, which they broadly classify into "omission noise" (missing labels in maps) and "registration noise" (misalignment between map and aerial image). They propose an Expectation-Maximization (EM) method to iteratively update model beliefs on these forms of noise, and then use these beliefs to augment the segmentation loss function. The authors report significant performance improvements in their ConvNet segmentation.

Within general image segmentation, the SegNet architecture [1] presents a very general, state-of-the-art core segmentation solution. Originally developed for street scene segmentation during autonomous driving, the fully-convolutional model appends an "inverted" VGG network [13] after a regular VGG network (after removing all non-convolutional layers) to obtain an encoder and a decoder phase of the model. In their architecture, the authors propose an efficient way to maintain spatial information for the decoder step, which enables a very structured network that generates pixel-wise labels (see the following section). In an interesting extension [9], the authors propose a Monte Carlo method to obtain an uncertainty measure of their segmentation results, which may be useful during interpretation.

Finally, Basu *et al*. [2] present a highly specific model for satellite imagery, and compare it to various other classification algorithms. They propose a model that uses a

[4]This corresponds to TMS/Slippy Map zoom level 18.

[5]Due to licensing issues, Google Maps data is unfortunately not available to the Humanitarian OpenStreetMap team.

[6]This corresponds to TMS/Slippy Map zoom level 16.

[7]CaffeNet is a slightly modified version of AlexNet [10], distributed with the caffe package [8]

(a) Satellite image (b) Map image (c) Segmentation label

Figure 1: Samples from the input dataset: a mean-centered satellite image, the corresponding map image, and the derived class labels (colored for easier recognition). Artifacts (noise) created by the text labels and icons are clearly visible. The segmentation task is to create a segmentation, given only a satellite image.

deep belief network on 150 statistical and image-processing features generated from satellite imagery to obtain classification results that are reported to significantly outperform ConvNets models on more complex datasets. Analyzing the statistical properties of their feature space, the authors note that properties beyond the ConvNet-typical edge-detection and color-filters are relevant for aerial images.

4. Methods & Models

I used a number of different models in architectures that are all inspired by SegNet [1], visualized in figure 2. Conceptually, SegNet uses a series of convolutional, batchnorm, ReLU and max-pool layers to encode an input image into a spatially smaller, deep representation. It then reverts this process using a spatial upsampling technique and more convolutional layers to arrive at a layer of the same spatial extent as the input image, with per-pixel classification scores (in my model, this is 256x256x7). The following paragraphs describe notable specifics in the architecture.

Since the encoding layers of SegNet are identical to the convolutional layers of VGG, it is possible to apply transfer learning to initialize the model weights with pre-trained filters. For this, I used the original VGG filters from [13]. The decoder layers are initialized randomly.

The encoder phase reduces the spatial extend of the input image by applying several steps of max-pooling, during which a window of typically 2x2 pixels is reduced to only the largest value within that window. This method reduces computational complexity and, crucially, increases the receptive field of deeper units in the network. In the decoding phase, the deep representation has to be "upsampled" to the original input dimension again. As a way to preserve spatial information through this process, SegNet stores the position of the maximum element seen during max-pooling and then restores the calculated value to the original position during upsampling, leaving the remaining (typically 3) positions at 0. As the upsampling layer is followed by a convolution, this allows the decoder to convolve over encodings that originated from spatially close positions in the encoder.

For the neural network's loss function, I used a pixel-average maximum entropy loss on softmax class probabilities, as given by

$$\text{Loss}(i) = \frac{1}{256^2} \sum_{x=1}^{256} \sum_{y=1}^{256} -w_{\text{label}} \log\left(\frac{e^{\text{score}_{xy=\text{label}}}}{\sum_{c \in \text{classes}} e^{\text{score}_{xy=c}}}\right)$$

where $i \in \mathbb{R}^{256*256*3}$ is the input image and and $\text{score}_{xy=\text{label}}$ is shorthand for "the classification score of class `label` at position (x, y)". In the naive version, $w_c = 1 \forall c \in$ classes. Label predictions are straightforward: the model will choose pixel-wise classes as $\hat{c}_{xy} = \arg\max_c \text{score}_{xy=c}$.

Given the imbalanced nature of my dataset, this choice of loss function biases the network to predict the prevalent `nolabel` class "with high confidence," i.e. towards $\text{score}_{xy=\texttt{nolabel}} \gg \text{score}_{xy=c} \forall c \neq \texttt{nolabel}$. To balance the loss function, I therefore used median class balancing [6]; this sets

$$w_c = \frac{\text{median}_i(\text{class frequency}(i))}{\text{class frequency}(c)}$$

where class frequency(c) is defined as the total number of pixels of class c divided by the number of images with c-pixels. This framework allows more general tuning of the model, as it allows to put more emphasis on classification errors for specific classes; I used this in later experiments to put more emphasis on the `building` class, as this class is most directly linked to population density, which is often the most pressing issue for humanitarian missions and policy.

I mainly experimented with three networks: after SegNet-basic as proposed by [1] failed to learn within reasonable computation time, I devised two significantly smaller networks, which I initialized from scratch: the first, dubbed "SmallNet" here, only uses two convolutional layers for encoding and decoding, respectively. The second, "LargerNet," like SegNet-basic, has eight convolutional layers in total, but uses significantly less parameters through less, and smaller, filters. In LargerNet, I also experimented with nonlinearities (ReLU) in the decoder phase. The specific design of each network can be found in table 2.[8]

To train each network, I used the AdaGrad algorithm [5], a version of stochastic gradient descent (SGD). This algorithm scales the iterative parameter updates to parameter p at time t by a factor of

$$\alpha_{p,t} = \sqrt{\sum_{i=0}^{t} (\nabla_{\mathrm{p}}^{i} \mathrm{Loss}(p, \dots))^2 + \epsilon}^{-1}$$

where $\nabla_{\mathrm{p}}^{i} \mathrm{Loss}(p, \dots)$ is the gradient of the loss with respect to p at iteration step i, and ϵ is a small constant for numerical stability. To further help learning, I reduced the global learning rate over time (stepwise, halved every training epoch). As a means of regularization, I used a miniscule weight-decay, which over time "pulls" learned weights back to 0.

All networks were trained using AWS GPU instances, using the caffe framework [8]. For upsampling, I used the SegNet version of caffe distributed with [1]. Minor changes to the caffe core code were required to handle the sparse classes in my dataset: as no single map square contains all classes (and many only contain `nolabel`), per-pixel-averages would produce division-by-0 issues. To avoid this, I edited caffe's accuracy layer and "test" module to calculate per-class accuracy only on those images that actually contain pixels of the given class. I also modified caffe's accuracy layer to include the intersection-over-union metric (see below).

Given the sparse dataset, constraints on GPU memory were especially problematic: when training LargerNet, only 15 input images would fit into GPU memory (4GB), making for heavily imbalanced batches even when randomly shuffling each batch. This made per-batch loss virtually impossible to interpret and may have contributed to inconsistent gradients during backpropagation. Caffe's little-documented `iter size` parameter allowed accumulating loss and gradients over a number of batches, which greatly helped training consistency.

[8] Additionally, all code is also available to `https://github.com/larsroemheld/OSM-HOT-ConvNet`.

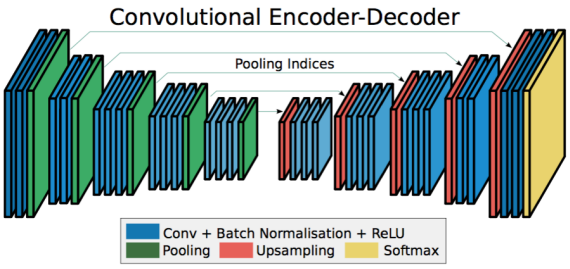


Figure 2: Full SegNet architecture, comprised of the 13 convolutional layers of VGG-16 in the encoder phase, an "inverted" VGG-16 with 13 convolutional layers in the decoder phase, a convolutional classification layer, and a pixel-wise softmax loss function. Visualization from [1]

.

5. Results & Discussion

As mentioned above, Segnet-basic failed to train given my limited computation resources and relatively small and noisy dataset. I therefore discuss only results of the two smaller nets, SmallNet and LargerNet.

LargerNet was trained specifically to detect buildings, as this was a main goal set by HOT: this was achieved by using a higher loss-weight for buildings than the relative class frequency would have warranted. With this setting, the best result achieved after 20 epochs of training (roughly 16 hours of computation) was a global pixel-average accuracy of 68.9% on the test set. Even with an emphasis on detecting buildings, the class accuracy of `building` was only 27% of predicted pixels; the prediction for `nolabel` was correct for 82.7% of pixels.

For the purposes of the project, this result is slightly, but not significantly, better than the naive baseline of "classify all pixels as `nolabel`": this naive baseline would outperform LargerNet on the global accuracy metric (as 79% of pixels are in the `nolabel` class), but it would obviously never find any buildings.

SmallNet was trained on neutral median-frequency loss weights. This results in a significantly higher training accuracy on `farmland`, and lower accuracy on `buildings`.

The real weakness of these results is revealed in an analysis of intersection-over-union (IU), a metric that takes into account false-negative prediction: for each class c, IU is defined as

$$\mathrm{IU}_c = \frac{|\{(x,y)|c = \hat{c}_{xy} = \mathrm{label}_{xy}\}|}{|\{(x,y)|c = \hat{c}_{xy}\} \cup \{(x,y)|c = \mathrm{label}_{xy}\}|}$$

That is, IU_c describes the size of areas classified as c relative to the joined area where c was the correct class and where c was predicted. If $\mathrm{IU}_c = 1$, then all pixels of class c have been recognized as belonging to c, and every pixel

SegNet-basic	SmallNet	LargerNet
conv7 64		conv3 64
BN		BN
ReLU		ReLU
MaxPool2		MaxPool4
conv7 128		conv 3 64
BN		ReLU
ReLU		
MaxPool2		
conv7 256	conv7 64	conv3 128
BN	BN	BN
ReLU	ReLU	ReLU
MaxPool2	MaxPool4	MaxPool2
conv7 512	conv7 128	conv3 256
BN	BN	ReLU
ReLU	ReLU	MaxPool2
MaxPool2	MaxPool4	
(encoded)	(encoded)	(encoded)
upsample2	upsample4	upsample2
conv7 256	conv7 64	conv3 128
BN	BN	ReLU
upsample2	upsample4	upsample2
conv7 128	conv7 64	conv3 128
BN	BN	ReLU
upsample2	conv3 7	conv3 64
conv7 64		BN
BN		ReLU
upsample2		upsample4
conv7 64		conv3 64
BN		BN
		ReLU
conv1 7		conv1 7

Table 2: Detailed architecture of the 3 main networks used in this paper. `convX Y` denotes a convolutional layer with Y filters of size X*X. `MaxPoolX` denotes a max-pooling layer with stride X and size X*X. `upsampleX` denotes upsampling with size X*X as described here. `ReLU` denotes a rectified linear unit, point-wise $y = \max(0, x)$. `BN` denotes Batch Normalization with learnable shift and scale parameters.

classified as c was really a c. For all classes except the dominant `nolabel`, this metric is almost 0, indicating that while LargerNet does have some accuracy in its prediction of buildings, it misses many areas where it should have detected buildings (on the test set, $\text{IU}_{\texttt{nolabel}} = 62.6\%$ and $\text{IU}_{\texttt{river}} = 2.6\%$ are the two highest IU values.)

Table 3 summarizes the most relevant model accuracies. In the remainder of this section, I will provide some qualitative analysis of the bad model performance.

As is evident by comparing training and test accuracies, the networks did not overfit at all. Given the very low regularization rates, this is likely due to significant noise in the training data: the data obtained through MapGive and OSM includes clouds, glare, and significant false-negatives in the map labels.

	LargerNet		SmallNet	
	training	test	training	test
`average`	68.8%	68.9%	60.6%	61%
`nolabel`	83.0%	82.7%	72.2%	72.5%
`building`	25.2%	27.0%	11.6%	13.6%
`farmland`	2.5%	2.5%	23.9%	19.9%
`river`	24.8%	23.9%	28.5%	26.8%

Table 3: Per-pixel average and class average accuracies of SmallNet and LargerNet. Validation accuracies have been ommitted, as they are almost identical to training and test accuracies.

Given "real world data," noise was to be expected – more computation time (more epochs) and more training data may have helped overcome some data problems. Another, more subtle, problem is that with the more shallow nets I used (as opposed to a full 26-layer SegNet), even LargeNet did not have a particularly expressive receptive field, as much of it was created through big max-pooling layers. This may help explain why LargeNet is missing larger structures in images. Figure 3 gives a visual overview of model performance.

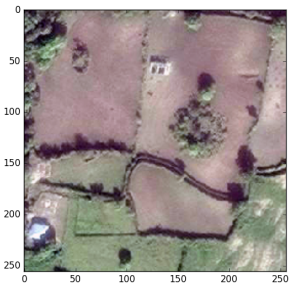
6. Conclusion

The task attempted in this project was ambitious: satellite data is known to be a challenging image recognition task [4], and even more so on a segmentation problem without a pre-trained model. Through the data pipeline created for this project, a large amount of additional training data is available – I was lacking computational power to use significantly more data, but recent work on massive aerial imagery datasets seems promising [7]. If successful, this project would ultimately help HOT and may offer relief to some of the world's most disadvantaged people. I hope to continue the project; maybe others will follow my data.

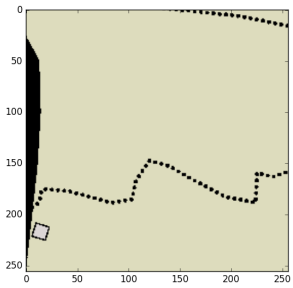
Until now, significant work went into establishing the project outline and the data pipeline; unfortunately, I did not achieve actionable results yet. Besides more in-depth class balancing, for example through a re-sampling technique during training, it would be interesting for future work to see how expectation-maximization models may help model label noise and facilitate learning. It may also be interesting to explore feature generation through more sophisticated image preprocessing, and to explore different ConvNet architectures for aerial imagery in a more structured way.

References

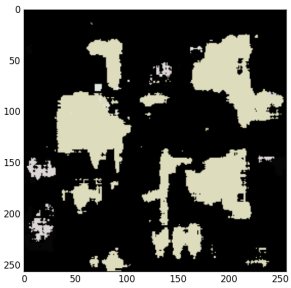
[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image



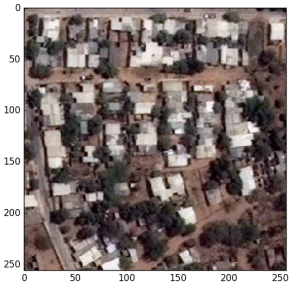
(a) Satellite image. Note the hut towards the top-center.



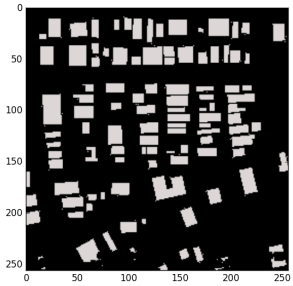
(b) Ground-truth map image. The hut is missing from the map.



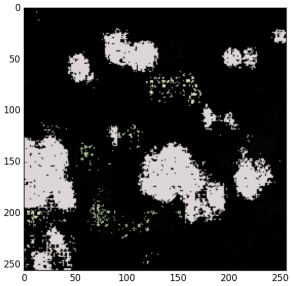
(c) Segmentation result. Noisy, but picked up on the missing building.



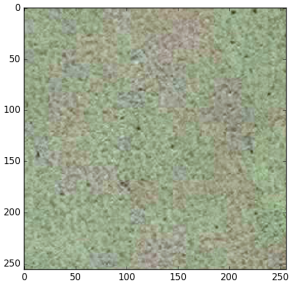
(d) Satellite image of residential area.



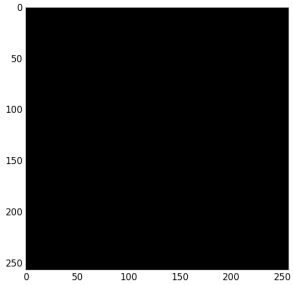
(e) Ground-truth map image.



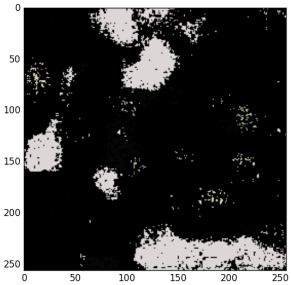
(f) Segmentation picked up some buildings, but failed to get overall structure.



(g) Satellite image of bushland.



(h) Ground truth: nothing to see here.



(i) Segmentation misinterprets evenly spaced structure.

Figure 3: Segmentation examples from LargerNet. The first row shows an example of noisy labels, penalizing the network for a correct prediction of `building`. The second row shows an example where the network is missing overall structure. The last row shows an example of false-positives caused by higher weighting of `building` in the network.

segmentation. *CoRR*, abs/1511.00561, 2015.

[2] S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. R. Nemani. Deepsat - A learning framework for satellite imagery. *CoRR*, abs/1509.03602, 2015.

[3] I. Bergman. Extracting map features from satellite images. *CS231n Winter Final Report*, 2015.

[4] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *CoRR*, abs/1508.00092, 2015.

[5] J. C. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[6] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *CoRR*, abs/1411.4734, 2014.

[7] A. Gros and T. Tiecke. Connecting the world with better maps. *facebook researcb*, 2016.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Gir-

shick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[9] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *CoRR*, abs/1511.02680, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] V. Mnih and G. Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th Annual International Conference on Machine Learning (ICML 2012)*, June 2012.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS)*, 2010.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[15] M. Xie, N. Jean, M. Burke, D. Lobell, and S. Ermon. Transfer learning from deep features for remote sensing and poverty mapping. *CoRR*, abs/1510.00098, 2015.

[16] Y. Yang and S. Newsam. Bag-of-visual-words and spatial extensions for land-use classification. *CoRR*, abs/1409.1556, 2014.