Predicting Economic Indicators via Machine Learning on Remote Sensing Imagery

Submitted by Shreyas Ravishankar
shreyas.ravishankar@yale.edu

Advised by Professor Dana Angluin
dana.angluin@yale.edu

May 2017
1. Abstract

It is extremely difficult to gather reliable data on economic and social indicators in many parts of the developing world. Efforts at gathering such data through conventional means, like censuses and economic surveys, are hampered by the lack of state presence and infrastructure in these areas. This lack of information in turn constrains efforts to design meaningful policy interventions. In this light, a group at Stanford University sought to use cheaply available remote sensing data from Africa - specifically, Malawi, Nigeria, Rwanda, Tanzania, and Uganda - to estimate economic indicators for villages in the region. The group used a three stage transfer learning approach in this process. First, they started with a convolutional neural net (CNN) that was trained on the ImageNet classification challenge. Second, they used the information from this model to build another neural net to predict the nighttime light intensity, a coarse proxy for poverty outcomes, corresponding to a given daytime satellite image. Finally, they leveraged the output from this neural net to build a ridge regression model to predict asset holdings and consumption outcomes, as measured by the World Bank’s Living Standards Measurement Study (LSMS) and USAID’s Demographic and Health Survey (DHS) respectively. The authors have reported reasonable success from this approach. In this report, I will walk through my attempts at replicating these results and extrapolating their idea to other parts of the world. At the time of writing, my efforts have been unsuccessful.

2. Introduction

One of the biggest obstacles in designing economic policies for the developing world is the lack of data to drive these decisions. Measurements of economic and social indicators shape policy research questions and the subsequent interventions designed by policymakers. However, such data are often unavailable or inaccurate. Conventional door to door censuses are expensive to conduct, especially in nations with a variety of terrains. And political phenomena like insurgencies make it nearly impossible for administrations to send surveyors into the field - assuming, of course, that state presence exists in these areas in the first place.

One approach to getting better data is to use mine data that do not require boots-on-the-ground measurement. Remote sensing imagery is one such (hitherto untapped) data source. Satellite data is ubiquitous, in that satellites capture images from every part of the world; cheap, as many sources of high definition satellite images make them freely available for consumption; and publicly available, unlike other forms of data like telephone records that are proprietary. Moreover, there is an intuitive appeal to seeing satellite images as a source of information on economic outcomes. We can imagine the infrastructure details contained in a given image as being a proxy for economic development in the geographic area the image corresponds to.
This intuition, or at least something like it, led a group at Stanford University to build a machine learning model to predict economic indicators in a particular area given daytime satellite images corresponding to it. In a series of papers, the group developed this idea using a three stage transfer learning approach, and they reported rather encouraging results. These results received considerable attention in the press, with Bill Gates (who now devotes his attention to development efforts at the Bill and Melinda Gates Foundation) particularly effusive in his praise for the concept on Twitter.

In this report, I will discuss my attempts to replicate the group’s work. There are broadly two parts to this document. First, I will explore the Stanford group’s original idea in greater depth. Then I will walk through my own (eventually unsuccessful) attempts at building a similar model.

2.1. Motivation

My motivation behind attempting this project was to expand the group’s idea to another part of the world that I am interested in: north east India. I was hoping to use a machine learning model of this nature to study this region, with the aim of giving policymakers more data to work with.

2.2. Technical Background

The project uses many significant concepts in machine learning. Here, I have compiled a list of resources that might help readers understand the nuances of these models. This list is not by any means exhaustive: there are thousands of resources available on the internet that talk about these concepts.


Transfer Learning: [http://cs231n.github.io/transfer-learning/#tf](http://cs231n.github.io/transfer-learning/#tf)

3. A Deep Dive into the Original Project

3 These are Andrej Karpathy’s lecture notes from the class he teaches at Stanford called CS231: Convolutional Neural Nets for Visual Recognition. I found these extremely helpful when I began this project.
4 Class notes from Stanford’s Statistics 305 class.
5 Class notes from Stanford’s CS231 class.
3.1. The Big Picture

The group at Stanford wanted to build a machine learning model that would predict economic indicators in the African nations of Malawi, Nigeria, Rwanda, Tanzania, and Uganda, using daytime satellite images of areas in these countries. The authors focus on two key economic indicators: asset holdings, as measured by USAID’s Demographic and Health Survey (DHS), and consumption levels, as measured by the World Bank’s Living Standards Measurement Study (LSMS).

3.2. Technical Overview

The group took a three stage transfer learning approach to building this model.

In the first stage, the authors start with a convolutional neural net trained for the purposes of the ImageNet classification challenge.6 (In their earlier paper, the authors mention using the VGG-F net trained by Chatfield et al. as this first stage model, but their more recent work suggests that they have now built a model of their own.)7 This model, they explain, “learns to identify low level image features such as edges and corners that are common to many vision tasks.”8 This is a useful capability that can serve as the foundation for more advanced identification tasks.

It is perhaps most intuitive to think of this stage as equivalent to the following hypothetical function:

\[ \text{Stage 1 ImageNet neural net : Image (such as a picture of a cat or a chair)} \rightarrow \text{Label (cat or chair)} \]

In the second stage, they build a neural net that leverages this edge detection capability to predict nighttime light emission from a particular geographic location, given a daytime satellite image corresponding to it. In effect, the neural net serves as a black box that implements a dimensionality reduction function. As the authors explain, “the model learns to summarize the high dimensional input daytime satellite images as a lower dimensional set of image features that are predictive of variation in nightlights.”9

The intuition here is that nightlights are seen as a proxy for economic conditions, albeit a coarse one. This means that the neural net learns a representation of an input image that helps predict a value that is in turn correlated with the economic outcomes we are ultimately interested in. Crucially, humans do not tell the model what features to look for. This is known as unsupervised learning: the neural net picks up important features

---

6 [http://www.image-net.org/](http://www.image-net.org/). The ImageNet dataset consists of over a million images of everyday things - phones, dogs, books, and so on. The challenge is to build an algorithm that can identify an appropriate label for each of the these images. For example, given an image of a rainbow, the algorithm should be able to identify the image as featuring a rainbow.

7 Xie et al., *Transfer Learning*, 4.

8 Jean et al., *Combining satellite imagery*, 791.

9 Ibid.
entirely by itself, without direction from human analysts. For example, the papers show that different filters in the neural net learn representations for roads, rivers, houses, and the like, which are all features we would reasonably expect to be indicative of economic outcomes.

Again, this stage can be seen as performing the following hypothetical function:

\[ Stage \text{ 2 Neural net} : \text{Daytime satellite image} \rightarrow \text{Nightlights value} \]

In the process of transforming a satellite image into a corresponding nightlights value, every layer in the neural net creates a vector representation of the image, which is then passed onto a subsequent layer for processing. The authors use the vector generated by the seventh convolutional layer of the neural net for a given input image to represent the information encapsulated in the original image itself in the final stage of their model.

On an aside: it is important to understand that a neural net, or indeed any predictive model in the machine learning literature, says nothing about the causal relationship between the features identified and the outcomes being predicted. It may seem intuitive to us that building a road through a village, for example, is likely to lead to better economic outcomes here. However, the fact that a filter in this neural net learns a representation for a road does not constitute proof of a causal relationship of this nature. Unlike traditional econometrics, the field of machine learning is not concerned with causality. No claims about the underlying nature of the relationship between roads and poverty are being made here, beyond a recognition that the two are somehow correlated in the dataset.

Finally, the third stage uses the vector produced in the second stage as input to a ridge regression model, which is trained to predict economic outcomes. More specifically, the ridge regression model accepts as input a principal component representation of a vector that encapsulates the information in a particular image, and outputs some predicted economic indicator value corresponding to the same image.

And once again, this stage can be interpreted as the following hypothetical function:

\[ Stage \text{ 3 Ridge regression} : \text{Vector representation of satellite image} \rightarrow \text{Assets/Consumption indicator} \]

3.3. Data Sources and Codebase

The authors used the following data in their project.

1. Survey data, which are available on request from the respective organizations’ websites.
2. Satellite images, which were procured from Google Maps using the Google Static Maps API. The authors use images of size 400px x 400px at zoom level 14.

3. Nightlights data, which are available from the National Oceanic and Atmospheric Administration (NOAA) website. The NOAA annually “publishes nighttime images of the world with 30 arc-second resolution, or about 1 square kilometer”. These maps are available for free at:
   http://ngdc.noaa.gov/eog/data/web_data/v4composites/F18<year>.v4.tar
   where <year> corresponds to the year for which nightlights data is desired. The authors predominantly use data from 2013 in their model.

The code written by the authors to process data, get image download locations, and create the figures in their Science paper can be found at:
https://github.com/nealjean/predicting-poverty

3.4. Training Procedure

To generate training data for the neural net in Stage 2, the authors said in an email to me that they “sampled daytime satellite images randomly around DHS cluster locations [in the African countries mentioned above], then labeled these images with the corresponding nighttime lights value for that lat/lon”. A cluster refers to a geographical grouping of households that are spatially close to each other: every household surveyed is part of a larger cluster. Hence, the training dataset for their neural net corresponds to:

\((X_{\text{train}}, Y_{\text{train}}) = \text{(satellite images from around DHS survey locations, nightlights values for the latitude and longitude corresponding to each image)}\)

Where \(X_{\text{train}}\) comprises the input data and \(Y_{\text{train}}\) comprises the (ground truth) labels for each data point in the input data.

To generate training data for the ridge regression in Stage 3, they find a vector for every image of interest using the neural net, and map each vector to its corresponding cluster-mean value for the economic indicator of interest. Hence, the training set for the ridge regression model corresponds to:

---

10 Xie et al., Transfer Learning, 3.
\((X_{\text{train}}, Y_{\text{train}}) = \text{(vector representations of satellite images created by Stage 2 neural net, cluster-mean value for consumption data/asset holdings corresponding to each image)}\)

### 3.5. Results and Potential Flags

Although the authors present encouraging results from this transfer learning approach in their Science paper, it is possible to be skeptical of how well their numbers generalize across datasets. In general, they claim that their predictions display significant \(r^2\) values (coefficients of determination) with respect to actual outcomes. But the provided codebase throws up some questions that merit investigation.

The first problem stems from the implementation of the ridge regression model used to create Figure 3, which shows a fairly high coefficient of determination for their predictions on consumption outcomes vis a vis the actual values recorded in LSMS surveys.\(^{11}\)

The purpose of any predictive machine learning algorithm \(r_d(\cdot)\) that is trained on some dataset \((X_{\text{train}}, Y_{\text{train}})\) is to be able to accurately predict a corresponding output label \(Y\) for some \(X\) that is not found in the set \(X_{\text{train}}\). Unfortunately, we have no way of modelling a data generating process to generate these \((X, Y)\) data points, so we cannot perfectly measure predictive ability. The closest we can get is to check how well \(r_d(\cdot)\) performs on some dataset \((X_{\text{test}}, Y_{\text{test}})\) that is generated by the same process that created \((X_{\text{train}}, Y_{\text{train}})\), but where no data points in \(X_{\text{test}}\) are in \(X_{\text{train}}\). We can then conduct some analysis on how closely \(Y_{\text{test}}\) resembles \(Y_{\text{predicted}}\), the outcomes predicted by \(r_d(\cdot)\) for input \(X_{\text{test}}\).

Crucially, for this testing mechanism to be informative, the algorithm \(r_d(\cdot)\) must not bias the predictions \(Y_{\text{predicted}}\) it makes by using information from \(Y_{\text{test}}\). Although in reality we know what the outcomes \(Y_{\text{test}}\) are, we ignore them for the purpose of simulating a situation where we encounter a dataset \(X\) with unknown labels \(Y\).

However, the ridge regression implementation described by the authors does not respect this principle. The code in /figures/fig_utils.py indicates that they implement a ridge regression with both an ‘outer’ and an ‘inner’ set of k-fold cross validations. The ‘outer’ cross validation randomly splits the entire test dataset, \((X_{\text{test}}, Y_{\text{test}})\) into \(k\) folds \((X_{\text{test},1}, Y_{\text{test},1}),(X_{\text{test},2}, Y_{\text{test},2}), \ldots, (X_{\text{test},k}, Y_{\text{test},k})\). Predictions are made on a fold by fold basis: that is, the model uses \(k\) ridge regressions, one for each fold. But in order to make this prediction, they also conduct an ‘inner’ cross validation on every one of these ‘outer’ folds. This splits each \((X_{\text{test},i}, Y_{\text{test},i})\) into \(k\) folds \((X_{\text{test},i,1}, Y_{\text{test},i,1}),(X_{\text{test},i,2}, Y_{\text{test},i,2}), \ldots, (X_{\text{test},i,k}, Y_{\text{test},i,k})\).

In order to find the best regularization parameter, \(\alpha\), to make predictions for the data points in each ‘outer’ fold, the authors build regression models on each of these ‘inner’ folds, using different \(\alpha\) values. Then, for each ‘outer’ fold \((X_{\text{test},i}, Y_{\text{test},i})\), they pick the \(\alpha\)

\(^{11}\) Jean et al., *Combining satellite imagery*, 792.
that maximizes the coefficient of determination of the predicted outcomes vis a vis the known $Y_{test,i,j}$ values across all of the j ‘inner’ folds. But here lies the issue: this prediction algorithm clearly biases results using knowledge of the actual label values of the test dataset.

This mechanism raises questions of how powerful their prediction model actually is. The final $r^2$ values reported in the paper are simply a mean across the $r^2$ values generated by each of these ‘outer’ fold prediction models. Hence, the summary statistics the paper reports are biased by prior knowledge of the true outcomes of the test set. Naturally, this mechanism will not work when the model encounters a dataset X with truly unknown Y labels, that is, with truly unknown outcomes. At the very least, we can say that the figures presented in the paper are misrepresentative of the true predictive power of their model.

There is also potentially a second problem in their results, which stems from the comparison the authors make between their transfer learning model and an alternate model that only uses nightlights data to make predictions. This is certainly a comparison worth investigating. If it is true that nightlights are a proxy for economic outcomes, perhaps we do not need to leverage any information from other models to make accurate predictions. Figure 4 in the Science paper makes a comparison between the two models on the basis of the $r^2$ values each generates after training on datasets of different sizes.\(^4\) The transfer learning approach, they claim, consistently provides more explanatory power than the nightlights approach.

Yet the code shows that the comparison being made here may not be entirely fair. The authors use a ridge regression to run their transfer learning model, as opposed to a standard non regularized ordinary least squares (OLS) model in their nightlights model. However, OLS models are known to cause overfitting: this is the primary reason why regularized regressions are used in the first place. I raised this question in an email to the authors, and they responded by saying “Nightlights is only 1-D so we weren’t worried about overfitting, thus no regularization”. But I am not entirely convinced that this observation justifies their approach: the bias-variance tradeoff exists even in the case of 1-dimensional data, though it is certainly more of a problem in higher dimensional space.

4. My project

As mentioned earlier, I attempted to reproduce the results presented in the paper. I used largely the same data sources as the authors did, and also used the survey data processing code they wrote. However, I wrote my own scripts for everything else (see section below for details on my codebase).

4.1. Picking a machine and setting up Caffe

12 Jean et al., *Combining satellite imagery*, 793.
(This might turn out to be a little more dramatic than you expected)

The authors built their convolutional neural net infrastructure using the Caffe framework, so the first thing I had to do was set up Caffe on a machine to run my code. Caffe needs to be made from source (source code available at https://github.com/BVLC/caffe): unfortunately, there was no easy package manager solution that I could find. The framework is written in C++, but it also offers a python layer over this underlying implementation.

I first tried to link Caffe to a python3.5 installation on my own MacBook Air (1.3GHz dual-core Intel Core i5, 4GB RAM), and this proved extremely messy. After about six hours of manually linking libraries, I had it up and running. I did this to bring myself up to speed with the framework by fiddling around with it.

Next, I tried to link Caffe to an Anaconda distribution of python2.7 on the same laptop. This was also a little painful, but considerably less so than my experience with python3.5.

Of course, the lack of processing power and storage space on my laptop makes it a terrible machine to train machine learning models on. I intended on getting access to the Yale High Performance Cluster (HPC) to run my model on there. After being granted access to the grace and grace2 clusters (the former does not have a GPU core), I began the process of setting up Caffe on there. Unfortunately, a series of access permission issues made this nearly impossible. After about a month of trying to make things work, I gave up on this idea.

Instead, a friend named Henry Li agreed to let me use his gaming computer to train my machine learning model. He had built arguably the best machine on campus for deep learning - he is both a gamer and a machine learning enthusiast - and he kindly agreed to let me share the box’s resource with him (he was training a machine learning model too: I ended up using the box at night while he mostly used it during the day). All my simulations were eventually run in an Anaconda python2.7 environment on this box, which runs Ubuntu 16.0 and features a 1.3GHz dual-core Intel Core i7 processor, 16 GB RAM, and Nvidia GeForce GTX 1080 GPU. To store the large image datasets and high performance databases needed to train the neural network, I mounted a 300 TB disk onto the machine as well.

I also ran into an unexpected difficulty in finals week. As I was training a model, my Caffe environment and some of my other files inexplicably disappeared from disk. This meant I had to go through the process of setting up Caffe all over again. But this iteration was pretty smooth sailing. Having learnt what needed to be done through all my failed attempts earlier, I was able to get everything set up in under an hour!

4.2. Data Sources
I used the same survey data and nightlights data as the Stanford group did. However, I considered other options for satellite images. Mapbox was a particularly interesting option because they make extremely high resolution data available for free. However, after a little research, I concluded that Google Maps was the better option, as Mapbox only offers black and white images of remote parts of Africa at the zoom levels I wanted. I wrote a script that downloaded ~500,000 images using the Google Maps Static API. These images corresponded to the very same locations the authors say they downloaded images for. However, I did not download any images from Rwanda as I was unable to parse survey data from the nation to generate download locations.

### 4.3. Using the weights provided by the authors

The authors uploaded the weights learned by their machine learning model, as well as the structure of their prediction neural net, to their github page. Hence, the first approach I tried was to use their weights in their in neural net to generate appropriate representations for the satellite images in my dataset. However, I was unable to recreate the figures in the papers. While the model did learn vector representations for the images I fed in, the predictive ability of these representations was virtually nil. When I ran the same code the authors used to generate the figures in their paper, I produced $r^2$ values in the range of 0.00 to 0.02.

I began to investigate potential causes for this. Given that my dataset of images should have been exactly the same as the one they used, I could not immediately put my finger on what could have caused the results to be entirely useless. I realized later that there might have been one difference in our datasets. Unlike the sample satellite images in the paper, every one of my satellite images had a Google watermark and a satellite attribution sticker along its bottom edge. Given how localized this signature was across all my images, I was surprised that it had such a significant impact on the end results, but it was the most feasible explanation I could think of.

This meant that I would have to train my own convolutional neural net from scratch - and this took me down the rabbit hole, so to speak.

### 4.4. Model Training

#### 4.4.1. Data Manipulation

I decided to use a high performance database to store my training data images (in the form of preprocessed 3-channel numpy arrays) and their corresponding nightlights labels. I first tried to use an lmdb database to hold this data. Unfortunately, I only discovered after creating this database that it was throwing my model off because of a technicality. The lmdb system cannot store label values as floats, so it was converting all the numeric nightlights data into strings and interpreting them as class labels.
Instead, I decided to use the hdf5 file format to store my data, which worked successfully until Caffe threw me an error. The framework cannot handle hdf5 files that are larger than 2GB, so I had to split up my large database into many smaller ones. This proved tricky because my files were too large to fit into memory. Eventually, I found an hdf5 manipulator on github that could split large hdf5 files by reading lazily into memory from disk: [https://github.com/TomaszGolan/hdf5_manipulator](https://github.com/TomaszGolan/hdf5_manipulator).

I did a few back of the envelope calculations and figured that the upper bound on the number of images (and labels) one database file could contain, given the 2GB size threshold, was about 1025. Hence, I ended up splitting the larger hdf5 files I generated into smaller ones, each containing 1000 images and labels. My attempts with a large training set (more on this below) left me with around 200 such database files; with a smaller training set, I had around 60 such files.

### 4.4.2. Image Preprocessing

Following the image processing steps detailed in `/scripts/extract_features.py` on the group’s github account, I preprocessed every training image before storing it in the database. The preprocessing steps included channel switching and channel-mean subtraction.

### 4.4.3. Model Structure

I built my training neural network using the prediction neural net shared by the authors on their github profile as a guideline. For the purpose of training, I modified the last few layers to include an Accuracy layer and a Loss function. The picture below shows the structure of the neural net I built.

![Image of neural net structure]

Surprisingly, the authors’ prediction model uses a Softmax function in the output layer, which is only used in classification problems, where the objective is to predict class labels. However, the nightlights prediction task struck me as a regression problem, as the objective is to predict continuous values from the input satellite images. The loss function conventionally used for tasks of this nature is Mean Squared Error.

I followed the original approach by using a Softmax Loss layer in my training model, but I suspect that using an Euclidean Loss function may yield interesting results too. I did try training the model once using Euclidean Loss, but unfortunately I made a mistake (that I now understand) in setting it up, so I do not have any results from this attempt.

### 4.4.4. Training Parameters
I used a training batch size of 128, and a validation batch size of 64. I used a stochastic gradient descent optimizer with an initial learning rate was 1e-05: this was dropped an order of magnitude after every 10,000 epochs.

4.5. Training Attempts

Attempt 1

In my first training attempt, I used the entire set of ~500,000 images I downloaded as input data, and I mapped each image to the mean nightlights value over a 100 square kilometer cluster around the area represented. I used an 80/20 test/validation split. Hence, my training set was:

\[(X_{\text{train}}, Y_{\text{train}}) = (80\% \text{ of the satellite images I had downloaded, mean nightlights value corresponding to a } 100\text{km}^2 \text{ cluster around each image})\]

After training this model for 50,000 epochs, I used the weights learned to generate vector representations for images for the next stage in the model.

However, my results were totally uninformative: I achieved \(r^2\) values of nearly 0 for every dataset I tested on. This was true of the \(r^2\) values within every fold as well.

Attempt 2

In my second attempt, I used only images corresponding to DHS survey locations as my \(X_{\text{train}}\), and constructed my \(Y_{\text{train}}\) vector as I did above. This yielded no fruitful results.

Attempt 3

I suspected that the issue might have stemmed from my nightlights labels summarizing too information - so this time, I used the same \(X_{\text{train}}\) as in my second attempt, but reconstructed my \(Y_{\text{train}}\) values to be mean nightlights value corresponding to a 10 square km cluster around each image, instead of 100 square kilometers as earlier. This was more in line with the group’s original specification. However, once again, I got no meaningful results.

Attempt 4

Looking at my training dataset, I realized that around 80% of the images in it were labelled 0. That is, 80% of my training set comprised images of poor areas, that ostensibly had no infrastructural features the model could learn. I suspected this may have been biasing the model towards learning no useful features.
To counter this, I created a training dataset that only had pictures corresponding to areas with non zero nightlights values. I also increased the base learning rate to 0.001. (It was also during this attempt that my data and environment mysteriously vanished.)

Training on this dataset yielded the most bizarre results I had so far. The weights of every layer in the convolutional neural net somehow began showing up as ‘NaN’ values, so the corresponding output was ‘NaN’ as well. I suspected this may have been because the learning rate was too high to begin with, as a result of which there may have non mathematical operations (like division by zero) somewhere in the training process.

**Attempt 5**

I reduced the learning rate back to 1e-05, but the ‘NaN’ results persisted.

**Attempt 6**

I realized that I had been making one mistake all along: I was not initializing the weights in my neural net with the weights learned by a model trained on the ImageNet classification task. I tried to incorporate weights from the VGG-F model into my model. Unfortunately, this did not work either, but I suspect this was because Caffe was parsing the data in the weights file incorrectly.

None of these attempts were successful, and the semester ended before I could delve even further into the problem. However, I am optimistic that with a little more time, I will be able to make something work. Having been down this rabbit hole once, I am aware of the mistakes I made in this process, and I hope this experience will manifest itself in success in the near future.

**4.6. Codebase and Advice**

Although the authors provide some code to parse survey data and extract features from a neural net, I had to write a lot of code to download and manipulate data, and then train the neural net itself. All my code will soon be available at: [https://github.com/shreyastheroadrunner](https://github.com/shreyastheroadrunner), along with an easy to read guide on using it.

Having been through the process of building this model over the semester, I have a few pieces of advice for interested readers.

1. Use keras to build a convolutional neural net. Caffe is annoying. The only reason I stuck with Caffe was because I was in too deep beyond a point. But I am convinced that keras incorporates everything that is needed to make this project work.

2. Bounce your ideas off other people. Most of the advances I made during this project (insofar as they were advances, given that I don’t have results to speak of...) came from when I tried explaining what I was doing to my friends. Rubber duck debugging does work!
3. Strap in for a long ride. Things often break inexplicably, and every solution will raise new problems.

5. Future Work and Reflections

I fully intend on working on the project even after this submission. I recently received a Merriman-Bensinger fellowship stipend from Davenport College to travel to north east India over summer 2017 to continue this work. And I have been in touch with civil society organizations in India who might find work of this nature useful.

This project has piqued my interest in the nascent field of computational social science (and I would extend that interest to the digital humanities as well). Having read these papers and attempted to recreate them, I realize that my technical skills could prove to be of great service in this niche setting. This is certainly an area of inquiry I would like to conduct further research in.

Moreover, I think this project has alerted me to some of the pitfalls in the academic process. The idea at the heart of the Stanford group’s project is undoubtedly brilliant. However, the results they present are extremely difficult to verify, especially without a technical background. And as I discussed earlier, some of the figures they present are a little misleading. This is not to suggest a deliberate attempt at obfuscating information. But I think it is fair to say that, at the very least, the group was not entirely upfront about what their figures represented. Perhaps it is also fair to say that they were writing to a technical audience, and they expected their audience to be able to look through their publicly available code and understand the nature of these results for themselves. But I would like to believe that clear explanations are valued over glitzy figures in the academic world.

6. Conclusion

This report presented my attempts at replicating a transfer learning model created by a group at Stanford that predicts economic outcomes from daytime satellite images corresponding to locations in Africa. I explain the intuitions behind this idea and address a couple of possible pitfalls in the results they present. Despite multiple efforts to train a similar model this semester, I was unable to produce successful results. However, I intend to continue working on this project, and I certainly hope to do further work in the field of computational social science.