The quantification of markers of economic development from time-series satellite imagery using deep learning

Eric Peshkin

A thesis submitted to the Department of Mathematics for honors

Duke University
Durham, North Carolina

2018

Abstract

Archival satellite imagery contains massive quantities of largely untapped, objective data documenting the development of nations over time. A key obstacle to leveraging these images for the purposes of advancing population science research has been the lack of systematic methods for quantifying the visually observable changes in a manner that scales efficiently. This paper succeeds in quantifying economically relevant features pertaining to building development, such as square footage and geographical location, from satellite images spanning the years 2005-2009 with an F1-score of 0.8081 in a particularly challenging classification setting. We implement a principled image preprocessing pipeline and a version of the SegNet convolutional neural network architecture described by Badrinarayanan et al. (2016).

1 Introduction

Satellite images contain a wealth of data about the natural and built environment as well as animal and human populations, and have the potential to revolutionize studies of change in all of these dimensions across the entire globe. Moreover, the reach of imagery is unparalleled: it covers areas that have been extremely difficult to monitor, including conflict zones and areas ravaged by natural or man-made disaster, and extends back several decades. This paper leverages mathematical tools to automate a process which extracts reliable information from satellite imagery so that the information can be profitably exploited to advance science.

Aceh, located on the northern tip of the island of Sumatra, Indonesia, is an area for which the application of this methodology is immediately valuable. In 2004, the Sumatra-Andaman earthquake spawned a tsunami which killed over 230,000 people (about 5 percent of the population) along the coast of Aceh. In some areas, the natural and built environments were devastated by the tsunami whereas other coastal areas were left untouched due to the wave direction and local topography. This disaster was followed by one of the largest post-disaster reconstruction efforts in any low/middle income country, and yet, apart from historical satellite imagery, there is no systematic data regarding where and when reconstruction has taken place, or linking that reconstruction to changes in the economic prosperity, health, and well-being of the Acehnese population. Enabling the reliable extraction of markers of reconstruction in satellite imagery of Aceh, and integrating those changes with data from population-representative surveys, would substantially improve our understanding of economic disaster and recovery. This research establishes the feasibility of using image processing in combination with advanced machine learning technologies to extract reliable data on building reconstruction in Aceh, and lays the foundation for the extraction a wider variety of object-level features (road square area and location, agricultural plot square area, variety, and location, etc.) in future work.

Neural networks require a body of ground-truthed imagery data from which they can learn to distinguish visual features that uniquely identify members of each pertinent class. We systematically created a dataset which includes over 9 million manually labeled building pixels and captures urban, rural, beach-side, and inland regions of Aceh between 2005 and 2009. The diversity of the imagery data contributes to the difficulty of the classification problem, but also exposes the network to a wider variety of relevant examples, reducing systematic error and increasing the generalizability of the model. Then, we leveraged several mathematical tools in order to construct a rigorous image preprocessing pipeline which augments the total available information in our imagery data while preparing it for interpretation by the neural network. Doing so required the application of Gram-Schmidt pansharpening, image splitting, normalization, and the application of affine transformations. After processing the images, we implemented a neural network architecture ubiquitous in the computer vision and machine learning literature, SegNet, in order to perform the pixel-wise classification task (Badrinarayanan et al., 2016).

Ultimately, this paper describes a procedure which enables the linkage of algorithmically generated measures of infrastructural and agricultural change to population representative survey data collected in Aceh. Specifically, the Study of the Tsunami Aftermath and Recovery (STAR), is a large-scale longitudinal survey that has tracked the lives of tsunami survivors over the last 15 years and began 10 months prior to the tsunami. There are over 30,000 respondents in the survey with over 97 percent having been interviewed at least once post tsunami. In future work, associating algorithmically extracted measures of economic change over the last two decades with population-representative survey data from that period will allow us to supplement STAR survey data with broader indicators of development contained in time-series satellite imagery but otherwise impossible to isolate via survey or administrative methods. In achieving this goal, we will gain greater insight into the effects of redevelopment on the physical and mental well-being of the Indonesian population, contributing to the population health literature.

2 Review of Literature

For several decades, satellite imagery has been leveraged to measure environmental change through variation in land cover, albeit on a relatively small scale (Lillesand, Kiefer and Chipman, 2014; Foody 2002). Recent research has exploited night light imagery on a large scale to proxy economic outcomes at a fixed point in time Henderson et al., 2012; Ghosh et al., 2013, for example; but see Gillespie et al., 2015, for a more skeptical view based on changes over time). In a recent article in Science, Jean et al. (2016) use transfer learning and the VGG-F neural network model to extract information on night lights, infrastructure and agriculture from satellite imagery for the purpose

of predicting local area poverty levels in five African countries. Their procedure directly associates abstract features extracted by the neural network with quantitative economic data aggregated into community-level clusters. A similar paper published on Cornell arXiv by Suraj et al. (2017) utilizes transfer learning and a modification of the VGG CNN-S convolutional neural network architecture to construct a predictive model that correlates socioeconomic indicators with abstract features generated by the network from satellite images of communities taken across India.

Our methodology complements these approaches by enabling the extraction of exact, easily interpretable features (building square footage or the precise location of buildings, for example) at the level of individual objects, facilitating a broader range of meaningful economic analyses, especially with respect to changes over time. Methods which directly associate abstract features with quantitative measures of economic well-being, even if they perform well in the cross-section, do not capture several object-level features necessary to most comprehensively chart economic development. One such feature is the spatial relationship between objects across time. While abstract methodologies are capable of, for example, accurately associating imagery containing buildings of varying textures and sizes with economically relevant metrics of well-being, they are effectively blind as to whether or not objects have changed location. For example, when such a model is applied to multiple images of the same geographical location taken several years apart, if every image contains structures (i.e. buildings, roads, agricultural plots) of similar quality and in roughly equal proportion, the abstract models will regard them as roughly equal in economic well-being. Effectively, nothing will be detected. However, if the images represent, for example, a community pre-disaster and post-rebuilding, such that the buildings within later images are in entirely different locations, this lack of sensitivity to changes in spatial arrangements across time proves detrimental, as the destruction and reconstruction pass undetected. Since high-quality satellite imagery for specific geographic locations at specific, historically relevant points in time often does not exist, the ability to detect even highly subtle markers of economic change is paramount. Because our methodology measures the location of specific objects in satellite imagery across time, it enables the detection of changes at a greater level of detail than the abstract methodologies, which will improve the quality of the economic analyses which depend on the extracted information. Furthermore, if necessary, object-level features can be aggregated to produce community or population-level features while the opposite is not true, and so capturing features at the greatest possible level of detail enables valuable economic analyses at every other level of detail.

Moreover, Jean et al. utilizes nightlight data in an intermediate training step to encourage the network to learn other features associated with variations in nightlight luminosity, such as the material from which a rooftop is constructed or distance from urban areas, which are better overall predictors of economic outcomes in their regions of interest. Our targeted, pixel-wise image

segmentation approach improves on this procedure by enabling the direct isolation of the desired end feature. For example, given the appropriate training dataset buildings of a specific class or category can be segmented directly and distances between urban and rural regions can be calculated explicitly, without reliance on the controversial nightlights proxy.

More generally, researchers have made attempts at solving the object-level infrastructure segmentation problem since the late 1980's (Huertas and Nevatia, 1988). Two primary schools of thought have formed regarding the appropriate methodology for doing so. Many, following the example of Huertas and Nevatia, believe the solution lies in image processing, and utilize methods such as parallel edge detection, line-linking, and measures of regional pixel variance to isolate buildings hypotheses (Saeedi and Zwick, 2008). Others believe that recent developments in machine learning algorithms and the science of computer vision offer a more effective solution. Ghaffarian and Ghaffarian (2014) utilize a parallelepiped classifier and achieve a 0.779 pixel-wise average F1 score, 0.884 average precision and 0.717 average recall across a primarily urban dataset. Our methodology is competitive with these results despite the unique challenges associated with building classification in Aceh, including the presence of dense forest canopy which which eclipses building edges in rural regions, tropical agricultural features which share qualitative characteristics (color, shape) with building rooftops, and very noisy post-disaster 2005 imagery taken at a suboptimal angle. Because imagery of historical disaster is limited to the fixed body of images which happened to be recorded at the time, methods robust to potential impurities are essential in capturing its value. Figure 1 below displays one such difficult image in our training data set.

With access to additional satellite information, especially LIDAR, classification results can be dramatically improved (Du et al., 2016). However, as stated, any study of historical change will require methods which are suited to whatever photographic information was captured in the past. As such, developing a procedure capable of highly accurate classification utilizing only the information available in historical satellite imagery has the potential to substantially advance the literature.

3 Data and Preprocessing

Our raw data consists of six QuickBird satellite images provided as part of a grant from the DigitalGlobe Foundation. The dataset is split into three pairs of images spanning a five year period, two from January 22, 2005 (one month post-tsunami), two from July 30, 2007, and two from Feb 23, 2009, ensuring a wide variety of the relevant deconstruction and reconstruction markers were captured across time. Within each pair, one image was captured using a high-spatial-resolution, grayscale panchromatic sensor, while the other was captured using a 4-Band,

Figure 1: Notice the scattered destruction and non-perpendicular angle at which the image was taken. These factors make classification more difficult, especially relative to the usual urban, developed country segmentation problems common in the literature.

RGB + Infrared multispectral sensor. The images were stored as 11-bit integer .tiff files and georeferenced appropriately. An example of one such pair of images is displayed in figure 2.

3.1 Gram-Schmidt Pansharpening

Using the Environmental Systems Research Institute (Esri) ArcMap software, we applied the Gram-Schmidt pansharpening transformation as described by Laben and Brower (1998) to each pair of images in order to combine the panchromatic and multispectral information in such a way that both the high spectral resolution of the multispectral image and high spatial resolution of the panchromatic images could be leveraged in our analysis. We selected this approach over other common pansharpening approaches including Brovey, Zhang, and IHS, because empirical work with similar satellite imagery suggests Gram-Schmidt to be most effective at improving the spatial resolution of the multispectral images while preserving the spectral and radiometric resolutions of the original image (Belfiore et al., 2006; see Maglione et al., 2016 for a description of the Brovey, Zhang, and IHS methods).

The procedure begins by computing a simulated, grayscale, low resolution panchromatic band as a linear combination of our four multispectral bands, $R, G, B,$ and NIR. Each band is encoded

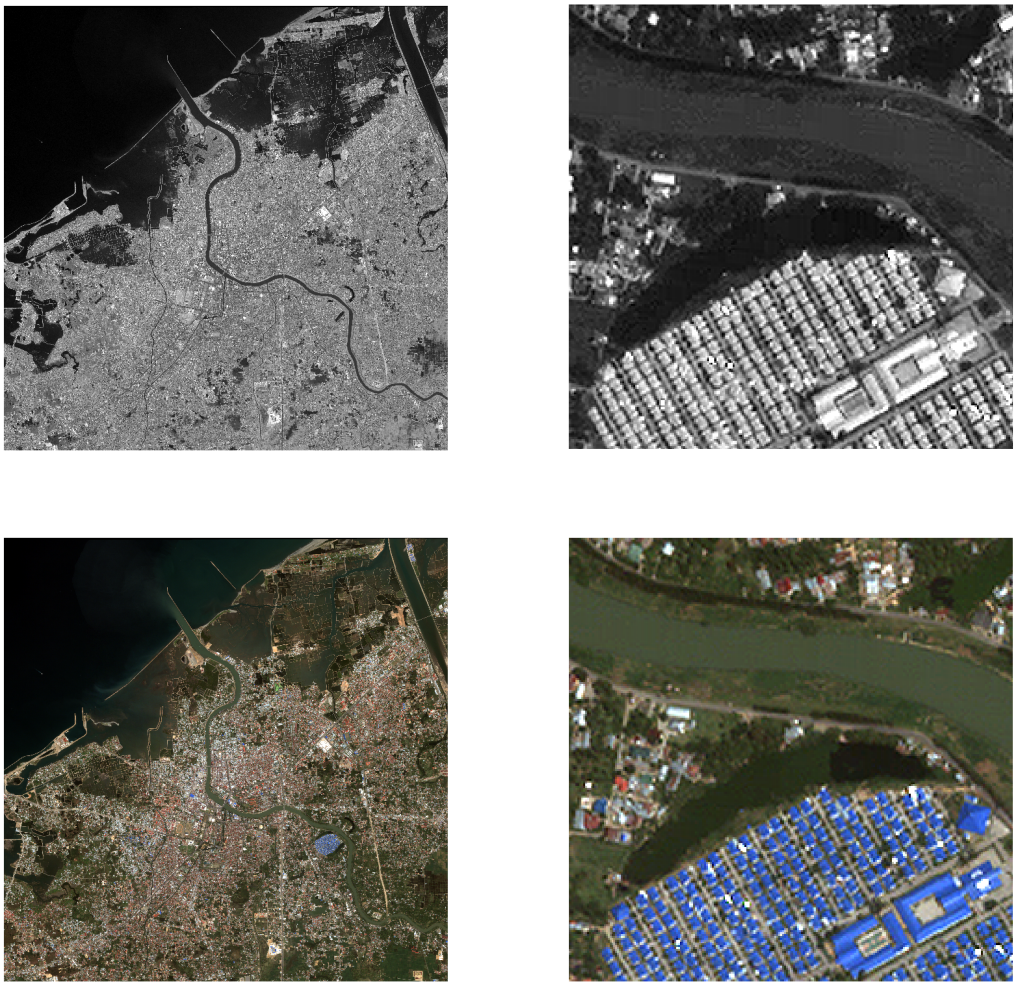

Figure 2: From top left, clockwise: A panchromatic image, the panchromatic image close-up, a multispectral image, the multispectral image close-up. Notice the higher spatial resolution of the panchromatic image at the cost of its color properties. These tiles are a subset of a larger satellite image taken February 23, 2009

as an MxN matrix for which the i^{th}, j^{th} value represents the Red, Green, Blue, or Near-Infrared value of the pixel at the i^{th}, j^{th} position in the image. Because the images are represented by 11-bit integers, these values range from 0 to 2048, where values closer to zero are less intense (more black in the case of RGB, less heat-emitting in the case of NIR). Creating the simulated low-resolution panchromatic image requires determining the coefficients by which we weight each matrix in calculating the linear combination. These coefficients are dictated by the physical properties of the sensor used to capture the images, and are computed by the following formulae, where OT_k represents the optical transmittance for band k, SR_k represents the spectral response of band k, SR_p represents the spectral response of the original panchromatic image, and λ is the wavelength of incoming light recorded by the sensor:

$$R_{wt} = \int_{0.6}^{0.7} OT_R(\lambda) * SR_R(\lambda) * SR_p(\lambda)\ d\lambda$$

$$G_{wt} = \int_{0.5}^{0.6} OT_G(\lambda) * SR_G(\lambda) * SR_p(\lambda)\ d\lambda$$

$$B_{wt} = \int_{0.4}^{0.5} OT_B(\lambda) * SR_B(\lambda) * SR_p(\lambda)\ d\lambda$$

$$NIR_{wt} = \int_{0.7}^{0.9} OT_{NIR}(\lambda) * SR_{NIR}(\lambda) * SR_p(\lambda)\ d\lambda$$

According to Esri, for the optical transmittance and spectral response of the QuickBird satellite's hardware, $R_{wt} \approx 0.85, G_{wt} \approx 0.70, B_{wt} \approx 0.35,$ and $NIR_{wt} \approx 1.00$ (Esri, 2016). Once the coefficient weights are determined, we generate the simulated panchromatic band by straightforward linear combination:

$$Pan_{Sim} = R * R_{wt} + G * G_{wt} + B * B_{wt} + NIR * NIR_{wt}$$

After the simulated panchromatic band is determined, we treat each band as a high-dimensional (M*N dimensional for an MxN image) vector, and starting with Pan_{Sim} as the first vector, v_1 apply Gram-Schmidt orthogonalization. Given arbitrary, linearly independent vectors $v_1 ... v_k$, the standard Gram-Schmidt orthogonalization procedure returns a sequence of orthogonalized vectors represented by $u_1 ... u_k$ (Shifrin, 2011):

$$u_k = v_k - \sum_{x=1}^{k-1} \phi(v_k, u_x), \text{ where } \phi(v_k, u_x) = \frac{< u_x, v_k >}{< u_x, u_x >} u_x$$

In the modified process used for pansharpening, the mean value for each band is subtracted from each pixel in that band before performing orthogonalization, converting the original Gram-Schmidt inner products into covariances (Maurer, 2013). Therefore, the k^{th} newly transformed, orthogonal band constructed from the previous $k-1$ bands and representing the transformed version of band v_k, is given by:

$$u_k = (v_k - \mu_k) - \sum_{x=1}^{k-1} \phi(v_k, u_x) * u_x,$$

where μ_k, is the mean value of band v_k, and where:

$$\phi_k(v_k, u_x) = \frac{\sigma(v_k, u_x)}{\sigma^2(u_x, u_x)}$$

The value $\sigma(v_k, u_x)$ represents covariance between each given Gram Schmidt band u_x and an original band v_k as determined by (for an MxN image):

$$\sigma(v_k, u_x) = \frac{\sum_{j=i}^{MN}(u_{x,i} - \bar{u}_x)(v_{k,i} - \bar{v}_k)}{MN - 1}$$

And for which $\bar{u}_x$ is the mean of band u_x and $\bar{v}_k$ is the mean of band v_k (equivalent to μ_k). The correlation between u_x and v_k is divided by covariance of the Gram-Schmidt band u_x with itself, which is equal to its variance. Following this procedure, the transformed Gram-Schmidt bands $u_1...u_k$ are decorrelated.

Once the data has been decorrelated, we apply a reverse transform to return it to the multispectral band space. First, we construct a gain and bias adjusted version of the original high resolution panchromatic band. This is achieved by stretching the high resolution panchromatic image, P, such that the mean and standard deviation of its pixel elements match those of the first element in our transform procedure, the simulated panchromatic band Pan_{Sim}.

This stretching ensures the preservation of the spectral characteristics of the original multispectral data (Laben and Brower, 1998). Then, up-sampling of the multispectral bands is performed to ensure their dimension matches that of the original panchromatic image. Finally, using the gain and bias adjusted panchromatic band in place of Pan_{Sim} as u_1, the process is reversed by applying an inverse transform to generate the newly pansharpened bands $v_1...v_k$:

$$v_t = (u_t + \mu_t) + \sum_{x=1}^{k-1} \phi(v_k, u_x) * u_x$$

Discarding the gain and bias adjusted panchromatic band, the result is an image in the original multispectral colorspace with both high spatial and spectral resolution. The Gram-Schmidt pansharpened versions of the images in figure 2 are displayed in figure 3.

3.2 Data Ground-Truthing

After pansharpening is performed, several subsections of each satellite image were manually labeled with ground truth information in order to generate the training data set, or the subset of the data from which the neural network algorithm will learn to distinguish the details corresponding to the different classes we have defined. In each of these subimages, the outlines of all building polygons were labeled by our research team. An example of the result of this procedure is provided in figure 4. Once the labels were complete, all area within each polygon was assigned to the appropriate class: building vs. non-building. The result was a one-band MxN integer table associated with each MxN subimage containing ground truth information for each pixel. The total number of building labels vs. other labels per year are given in table 1. Due to the lack of NIR information

Figure 3: Left: The full pansharpened version of the tile seen in figure 2. Right: The pansharpened subset of the tile seen in figure 2. Note the combination of the spectral resolution of the multispectral image and the spatial resolution of the panchromatic image.

in the pre-trained VGG-16 weights which are initialized in our SegNet model, we stripped away the NIR band from each image, sacrificing some information for increased computational efficiency and access to the powerful VGG-16 weights described in section 4.3.

Figure 4: All building pixels in this subsample of the February 2009 image have been labeled.

Label	Building	Other	Yearly Total
2005	2,686,268	5,313,732	8,000,000
2007	3,540,384	19,459,616	23,000,000
2009	3,188,978	20,811,022	24,000,000
Class Total	9,415,630	45,584,370	55,000,000

Table 1: The number of manually labeled pixels per category, per year.

3.3 Neural Network Preprocessing

Once sufficient ground-truth information had been established, we sliced, normalized, and applied affine transformations to the images to prepare them for processing by the neural network and to maximize their value. First, we sliced each training subimage (and its associated integer truth table) from its original size into 100x100 pixel subslices. This was to decreases the amount of memory required to operate on each minibatch during the training procedure. We randomly set aside 30 percent of our training images for post-algorithm validation in order to determine the performance of our algorithm using novel samples on which it was not explicitly trained. The remaining 70 percent of the images constitute the training data set.

During its optimization phase, or training phase, the convolutional neural network applies a set of fixed hyperparameters, including learning rate α, momentum ψ, and $L2$ weight decay λ, all of which are described in section 4. In applying these values across a wide variety of calculations spread through many diverse layers, better results are usually associated with training data that is normalized so that no matter the range of values associated with a particular calculation, the hyperparamters remain meaningful relative to that range. Therefore, we zero-center normalize the 100x100 images in our training set. This is accomplished by subtracting the pixel-wise mean image from every image in the training data set.

Finally, we apply affine transformations to artificially augment the dataset. This allows us to capture patterns in imagery which may exist in the real world but for which we do not have specific examples in our training data, resulting in a neural network model far more robust to novel information. Our affine transformations include translations, rotations,and reflections. As such, each image produces a derivative image which is translated between -8 and 8 pixels in the horizontal or vertical direction, rotated a random angle between 0 and 360 degrees, and then reflected over its central vertical or horizontal axis with 50 percent probability per axis. These transformations seemed natural, as most geographical features visible on a map could foreseeably also exist in flipped, rotated, or translated states. The result is a massive increase in training data size and a reduction in the likelihood of overfitting, an issue which occurs when a neural network learns a specific dataset too precisely at the expense of recognizing more generalizable, broadly applicable patterns which would enable accurate classification on novel data. Naturally, when an

affine transformation is applied to an element of the training set, it is also applied to the associated ground-truth integer table.

In this experiment, the final result of our preprocessing pipeline was a training set of 100x100 properly pansharpened images associated with ground truth information pertaining to the buildings we hope to detect. Seventy percent of these images were directly used in training, while 30 percent were set aside for validation post-training to determine the accuracy of our network. Those which were used for training had been normalized and augmented with various affine transformations in order to increase the robustness of our model and reduce overfitting.

4 Neural Network Methodology and Implementation

SegNet is a deep convolutional encoder-decoder neural network model commonly applied to image segmentation tasks due to its superior accuracy and computational efficiency in training (Garcia-Garcia et al., 2016). In order to maximize its value given our training data, we trained one SegNet network to perform the desired classifications for each image year. Each network classified pixels as building vs. other for each of the three image years, 2005, 2007, and 2009. We split the algorithm into discrete times because images taken in 2005 are drastically qualitatively distinct from those taken in 2007 which are themselves distinct from those of 2009, and so the neural network will likely prefer vastly different features for identifying a particular class from 2005 imagery relative to those unique to that class in 2009 imagery. Furthermore, SegNet trains efficiently relative to fully-connected neural network models and so training additional networks is not computationally expensive. Our SegNet implementation was constructed in MatLab 2017B and trained using a GTX1080 GPU with 3gb on-board RAM.

4.1 SegNet Architecture Overview

Our implementation of the Badrinarayanan et al. (2016) SegNet model propagates data through ninety-one layers. The first layer is the 100x100x3 size image input layer which essentially holds our 100x100x3 RGB input images before passing them through the network. The next 44 layers represent the encoder part of the network which performs the mathematical operations described in section 4.2 to extract features which can differentiate between classes while constantly applying max pooling (a form of downsampling) in order to increase memory efficiency and training speed. The following 44 layers are the decoder layers, which extract relevant features in a similar manner while constantly applying max unpooling (a form of upsampling) in order to translate the encoded data into an interpretable form. Both the encoder and decoder sections of the network utilize convolutional, batch normalization, and ReLU layers in order to extract abstract image features

and learn image patterns effectively. The ninetieth layer applies a softmax function to the data to convert weight information to probabilities, and the ninety-first and final layer calculates a weighted cross-entropy cost function across the classification categories, determining the network's error rate and whether or not adjustments made during training have improved or diminished network performance. Table 6 in Appendix A contains the exact network architecture information, including the precise ordering of the 91 layers. Training utilized stochastic gradient descent with hyperparameters: momentum ψ, initial learning rate α, and $L2$ weight decay λ. These hyperparamters, the number of elements chosen in each mini-batch and the number of epochs for which training persists are unique to each network, and are selected empirically via a randomized grid search as described in section 4.6. The mini-batch size is the number of images in the training set fed through the neural network during a single forward and backpropogation, while the number of epochs is the number of times the entire dataset is passed through the network. The training data was randomly permuted before each epoch to ensure the neural network wasn't improving classification accuracy by learning image order. The role of the hyperparameters in training is discussed in section 4.4.

4.2 Layer Operations

Both the encoder and decoder portions of the network utilize convolutional layers in order to extract meaningful patterns in the data which are used to distinguish elements in an image. The convolutional layers pass matrices, or filters, across each image to extract mathematically pertinent identifying information from each region via the convolutional operation described below. As described in (Karpathy and Li, 2015a), three parameters determine the operation applied by each convolutional layer: the number of filters n, the size of each filter M, and the step-size (or stride) t when passing the filter across the image. Each of the n filters is an MxMxQ matrix, where Q is the depth of the inputs from the previous layer of the network, on which convolution is being performed. For every convolutional layer in this SegNet implementation, $M = 3$ and $t = 1$. So, for example, when passing from our 100x100x3 input layer to the first convolutional layer, we apply $n = 64$ filters of size 3x3x3 with stride length $t = 1$ to ensure each filter performs the convolution operation at every pixel in the input image without jumping over any. Before convolution proceeds, the boundaries of the image are padded with zeros in order to ensure the convolution operation is always well-defined. The values contained in the filters, or the weights of the filter, and the bias term associated with each filter are initialized as explained in section 4.3. Updates to these weights and biases are what enable the network to learn during training. In general, the elements which the network updates to increase classification accuracy during the training phase are called parameters.

As explained in (Karpathy and Li, 2015a), we perform the convolution operation by passing each 3x3xQ-dimensional filter across the image, calculating the dot product between the contents of the filter (its weights) and the input at each of the Q levels, and then summing the results and adding the bias term. As such, the output for each filter is a two-dimensional matrix which contains the scalar output of each of these procedures. The 2-dimensional matrices representing this operation performed for each of the k filters are concatenated along the third dimension to produce the output. Therefore, in aggregate, the convolution process for each 3x3xQ subregion of our input image, the 3x3xQ matrix, $I_{i,j}$, centered at the i^{th}, j^{th} element of the input, is described by the formula below. Ultimately, the output from each convolutional layer is a three dimensional matrix whose i^{th}, j^{th}, k^{th} value is determined by this formula. Note that the operation $\bullet$ does not represent standard matrix multiplication, rather it represents the scalar dot product between the elements of two matrices flattened into vectors.

$$\text{Convolutional Output}_{i,j,k} = \sum_{x=0}^{Q} (F_{k,x} \bullet I_{i,j,x}) + b_k \text{ for } k = 1...n$$

Where $F_{k,x}$ and $I_{i,j,x}$ represent the k^{th} filter of weights and subsample matrix of the input data centered at i, j, respectively. One such convolutional output is calculated for each element of the mini-batch passed through the network to the next layer.

Both encoder and decoder layers apply batch normalization layers between convolutional and ReLU layers in order to reduce the tendency for small changes to weights early in the network to dramatically alter the value of weights down the pipeline. This phenomena, termed internal covariate shift by Ioffe and Szegedy (2015), makes later weights less likely to converge to stable values and can force them to inhabit values beyond the effective range of the network hyperparameters. Normalizing layer inputs addresses these issues and allows the use of higher learning rates while reducing the influence of initialized weight values on the network's ultimate success (Ioffe and Szegedy, 2015). Much like the weights of convolutional layers, γ and β, two parameters fundamental to the batch normalization procedure, are learned during training. For a mini-batch of size m, the batch normalization layer receives m of the PxRxk matrix outputs from the preceding convolutional layer.

As described in Ioffe and Szegedy (2015), for each depth slice k we calculate the normalization mean μ_k and normalization standard deviation σ_k with respect to all PxR elements in the two-dimensional depth slice at level k across all m elements of the mini-batch. So, for x_i representing one of the m*P*R elements at depth k within one of the m matrices of the mini-batch:

$$\mu_k = \frac{1}{mPR} \sum_{i=1}^{mPR} x_i$$

$$\sigma_k^2 = \frac{1}{mPR} \sum_{i=1}^{mPR} (x_i - \mu_k)^2$$

Using these quantities, we calculate $\hat{x}_{m,k}$, the normalized version of the k^{th} depth slice of the PxRxk input matrix representing the m^{th} element of the mini-batch:

$$\hat{x}_{m,k} = \frac{x_{m,k} - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}$$

The small constant ϵ is equal to 0.00005 and numerically stabilizes the function in the event σ_k is very small without affecting the calculation otherwise. Finally, $y_{m,k}$ is the scaled and shifted version of each normalized observation $\hat{x}_{m,k}$:

$$y_{m,k} = \gamma \hat{x}_{m,k} + \beta$$

In aggregate, the batch normalization layer performs a normalization procedure which ensures different elements of the same feature map at different locations, or elements originating within different mini-batches but residing in the same k-depth layer at different spatial locations, are normalized in the same way (Ioffe and Szegedy, 2015). Because the same convolutional filter weights act on every element across a matrix at a given depth layer, normalization procedures which disproportionately modify particular regions of each depth slice reduce the effectiveness of the convolutional neural network model by shifting the values in those regions beyond the range in which the learned weights are meaningful, artificially extending and hindering the training process. Thus, this method of batch normalization enforces a more uniform distribution of values across the network, increasing training efficacy while respecting the unique properties of the convolutional neural network model. The normalized depth-slices are also multiplied by a factor of γ and offset by a factor of β, allowing the network to learn many distinct additional transformations which may be beneficial, including the identity in the event transformations prove only detrimental. The normalized mini-batch is passed to a ReLU layer.

ReLU layers are used by both the encoder and decoder layers to delinearize the neural network and allow it to identify more complicated, nonlinear relationships. A ReLU layer applies the ReLU activation function to every element of its input. The ReLU activation function, as defined by Nair and Hinton (2010), is:

$$f(x) = \max(0, x)$$

Max-pooling layers are applied by the encoder layer to reduce the amount of data to process while preserving critical information in order to improve network efficiency and enable the network to extract more abstract features. Max pooling layers are determined by a stride parameter, t and MxM window size. In our implementation, for each max pooling layer, $M = 2$ and $t = 2$. In

max pooling, the 2x2 matrix passes across the image and assigns the maximum value within its boundaries to represent that region in the output. The stride length of two means these matrices are non-overlapping, unlike the convolutional filters. In our algorithm, the result of max pooling on an $NxNxQ$ input image is an $\frac{N}{2}x\frac{N}{2}xQ$ mathematically precise abstraction of the original input. In SegNet, during max pooling the index, or location of the maximum feature value relative to each 2x2 pooling matrix, is stored for use by the decoder (Badrinarayanan et al., 2016). The max-pooled matrix is passed to the next layer.

Each max-pooling layer in the encoder corresponds to a max un-pooling layer in the decoder which performs non-linear upsampling to reverse the data compression performed by the encoder after many of the computationally-expensive operations have been completed and abstract features have been learned. The decoder upsamples its input using the indexes recorded during its corresponding max-pooling layer in order to deposit the value being upsampled in the appropriate location in the new upsampled matrix. For example, if the original max-pooling layer determined the maximum value in the 2x2 matrix it sought to downsample was in the bottom right corner, the element being upsampled is placed in the bottom right corner of the new 2x2 matrix which replaces it during upsampling. Other elements of the upsample matrices are set to zero. This procedure improves boundary delineation and reduces the number of weights which must be updated during training in comparison with other segmentation networks and relative to other variations on upsampling (Badrinarayanan et al., 2016). The decoded output is then passed to the next layer.

The softmax layer is the penultimate layer in the neural network in which a softmax function is applied to its input, the final feature maps generated for a minibatch by the cumulative action of the network. Each feature map is 100x100xk where k is the number of classes we hope to distinguish, such that the i^{th}, j^{th} entry in the feature map matrix corresponds to the i^{th}, j^{th} pixel in the original input image. The elements in each of the k depth dimensions of the matrix are features whose values are dependent on the parameters (weights, biases, γ and β) of every previous layer, and which the softmax classifier will use as features associated with the i^{th}, j^{th} pixel in determining to which of the k categories the i^{th}, j^{th} pixel belongs. The softmax function maps each of the k features across the depth layers corresponding to index i, j with the probability pixel (i, j) is in a particular class l. The function is represented by:

$$f(x_l) = \frac{e^{x_l}}{\sum_{z=1}^{k} e^{x_z}} \text{for l} = 1...\text{k}$$

where x_l is the element of the k-dimensional depth vector associated with the likelihood pixel (i, j) is a member of class l and k is the total number of classes (Karpathy and Li, 2015b). So, for each pixel at location (i, j), the softmax function maps its real-valued features, the k depth values at

position (i, j), to the interval [0,1] so that they can be interpreted as probabilities. Elements are considered classified as members of the class for which their softmax probability score is highest. The softmax classifier feeds the three-dimensional 100x100xk matrices, one for each image in the mini-batch, to the final cross-entropy layer. This layer will be fundamental when we perform the backpropogation step as described in section 4.4 and update the network weights in order to fine-tune the network and improve classification accuracy.

During backpropagation, weights are adjusted in order to minimize the cross-entropy loss function, defined by:

$$\text{Cross-Entropy} = -\sum_{l=1}^{k} \rho_l(\hat{y}_{ml})(\log(y_{ml}))$$

where $\hat{y}_{ml}$ is 1 if observation m is an element of predetermined class l and 0 otherwise, and y_{ml} is the predicted probability observation m is an element of class l via softmax output (Bishop, 2006). This implies cross-entropy will be lowest when observations are given a high probability of belonging to their ground-truthed class. The weight ρ_l is calculated via median frequency balancing as proposed by (Eigen and Fergus, 2015), and more heavily weights classes which are less prevalent in the training set in order to ensure the network considers them when minimizing cross-entropy. Otherwise, the network may seek to optimize around only the most prevalent classes, accepting any small increases in cross-entropy which would arise from misclassifying examples from sparse classes. For a given class l, ρ_l is given by:

$$\rho_l = \frac{medfreq}{freq(l)}$$

where $freq(l)$ is the number of pixels of class l divided by the total number of pixels in images where at least one pixel of class l is present, calculated across the entire training data set, and $medfreq$ is the median value of the set $freq(l_1)...freq(l_k)$, or the median of the set of values returned by the $freq$ operation with respect to all k classes. In this way, sparse classes return a lower $freq$ value which increases the applied weight ρ_l and ensures increasing classification accuracy for sparse classes is more strongly rewarded. Therefore, by minimizing cross-entropy, we reward the network for adjusting its weights such that the probability an observation is assigned to its proper ground truth class increases, and account for class imbalances in our training set. In the SegNet model, the total cross-entropy cost function J for a given minibatch is taken as the sum of the cross-entropy loss functions (per pixel) across every element of the minibatch.

4.3 Weight Initialization and Forward Propagation

The set of convolutional layers in the encoder portion of the SegNet network is topologically identical to the set of convolutional layers within the VGG-16 neural network (Badrinarayanan

et al., 2016). VGG-16 is an extremely powerful neural network model for image classification but also requires immense quantities of training data to perform well on a specific classification task. However, we leverage the VGG-16's more general image classification power by initializing the convolutional weights of the encoder portion of our network with the weights from a VGG-16 model pre-trained on the ImageNet database which contains over one million training images split among 1000 classes (Deng et al., 2009). Because of this procedure, our network initializes filters capable of extracting useful information and accurately identifying objects in imagery in a general sense without requiring such immense stores of training data pertaining to our classification task in specific (Russakovsky et al., 2015). Because these weights are valuable, they are not adjusted in training. In order to tailor the network to our specific classification task, and so it can adjust its parameters to generate filters responsive to our chosen classes, we must initialize and then modify the parameters belonging to the layers within the decoder portion of the network: the convolutional weights, and the γ and β elements of the batch normalization layers. The decoder's convolutional weights are initialized via the MSRA method described in He et al. (2015), in which they are pulled from a zero mean Gaussian distribution with standard deviation $\sqrt{\frac{2}{n_l}}$, where $n_l = (k_l^2) * (d_{l-1})$, k_l is the convolutional filter size in the currently initializing layer and d_{l-1} is the number of filters in the previous convolutional layer. Additionally, the batch normalization layers' γ and β are initialized to one and zero.

Training the network occurs in two main steps, forward propagation and back propagation. In our forward propagation step, we pass information through the layers of SegNet in the order described in Appendix A and by performing the operations described in section 4.1. This generates a cross-entropy value. Back-propagation is the process of updating the weights in order to reduce this cross-entropy value. Each mini-batch undergoes a forward and backpropogation step, until the entire dataset has been exhausted, constituting one epoch. The network then continues iteratively training for the number of selected epochs, updating the weights in response to changes in cross-entropy.

4.4 Backpropogation

Backpropogation is the process by which network weights and biases are updated during training in order to minimize our cross-entropy cost function J. Our SegNet implementation uses mini-batch gradient descent with momentum and $L2$ regularization. Its foundation is in batch gradient descent, in which we consider the partial derivatives of the cross-entropy cost function as calculated across every element of the training dataset with respect to each of the parameters being adjusted by the network. This allows us to determine how the cost function varies with adjustments in each parameter, so by adjusting the parameters accordingly we can minimize total cross-entropy. In

this procedure, each newly updated parameter W'_m is calculated by:

$$W'_m = W_m - \alpha \frac{\partial}{\partial W_m} J(W)$$

where W_m is the original parameter and α is the learning rate hyperparameter (Ng, 2012). The subtraction term of the equation dictates that when cross-entropy is locally decreasing as a parameter increases, that parameter is increased, while when cross-entropy is locally increasing as a parameter increases, that parameter is decreased. Performing this operation for every parameter in the network progressively minimizes cross-entropy. The learning rate hyperparameter α determines how rapidly the network proceeds towards an optimum by modifying the magnitude of each update. Too large a rate can result in a jump size which passes over an optimum completely, and too low a learning rate may prevent the algorithm from reaching an optimum at all, or trap it in a local optimum. Thus, fine tuning of this hyperparameter is required. As for all the hyperparameters discussed in this section, the value for alpha was chosen by randomized grid search as discussion section 4.6.

Mini-batch stochastic gradient descent (SGD) is a computational improvement on the batch gradient descent model. SGD computes the required partial derivatives based on the elements of a mini-batch rather than the entire training set. This is useful for two reasons. First, it vastly reduces the amount of data required to make each step towards an optima which reduces training time, and second, calculating the gradient with random samples can introduce enough randomness into the results to knock the algorithm out of a local minimum in which it would otherwise be stuck and towards a global one (Bottou, 1991). At the base level, SGD for the parameters contained in vector W is dictated by the following equation in vector notation:

$$W' = W - \alpha \nabla_W J(W; x^{1 \cdots n})$$

where W' represents the vector containing updated parameters and $x^{1 \cdots n}$ are the n elements of the mini-batch across which J is calculated (Ruder, 2017).

Our SGD operation also applies momentum and $L2$ regularization. Momentum adds a fraction of the previous weight update to the current weight update, so weights which are being updated in a consistent direction are updated in that direction more rapidly. Primarily this facilitates the gradient descent algorithm's skipping over local optima in favor of global optima, as were an adjustment to network parameters going to trap cross-entropy in a local optima, where gradient is zero, the momentum factor would still force the parameters to update in the direction of their previous update, dislodging cross-entropy from the local minimum and towards the global one. The momentum factor also smooths the convergence of the parameters such that they cannot oscillate

erratically in directions not along the dominant gradient, as an adjustment in one direction followed by an immediate adjustment in the opposite direction is met by a momentum term of opposite sign. This forces the network to update its parameters in response to the predominant gradient more effectively (Murphy, 2012). Adding momentum to our gradient descent algorithm generates the equation:

$$W' = W - \alpha \nabla_W J(W; x^{1\cdots n}) + \psi(W - W^o)$$

where ψ is the momentum hyperparameter and W^o is the vector containing the parameters from the previous backpropogation step.

Adding the $L2$ regularization term to our cross-entropy cost function discourages model overfitting by reducing variance (Murphy, 2012). Our L_2 regularized cross-entropy J_2 is given by:

$$J_2 = J + \frac{\lambda}{2}(w \cdot w)$$

where w is a vector containing only the convolutional weights, λ is the weight decay hyperparameter, and $\cdot$ represents the vector dot product. This serves to increase cross-entropy error when the weights become large, unless such weights considerably improve cross-entropy. Therefore, the network is incentivized to utilize a broader network of smaller weights, preventing a small network of weights with large values from dominating the model which would result in massively increased variance.

4.5 Segmenting a Test Image

When an image is submitted for classification by the trained network, normal forward propagation is performed and each pixel in the image is classified by the softmax layer in response to the features generated by forward propagation. The only distinction between the training forward propagation step and the final image segmentation procedure is that the batch normalization layers use the mean and variance calculated relative to the entire training set during training, rather than an estimate determined during any of the minibatching stages of training or derived from the testing image or images (Ioffe and Szegedy, 2015). Figure 5 displays a novel segmented 2007 image as an example output from this procedure.

4.6 Empirically Determining Hyperparameters

We apply a randomized grid search as described in Bergstra and Bengio (2012) in order to determine nearly-optimal values of learning rate α, momentum ψ, L2-regularization term λ, mini-batch size, and number of epochs with respect to each of our 2005, 2007, and 2009 networks. We sample 60 random points from the five-dimensional space spanning α = [0.1,0.001], ψ = [0.75,0.95], λ =

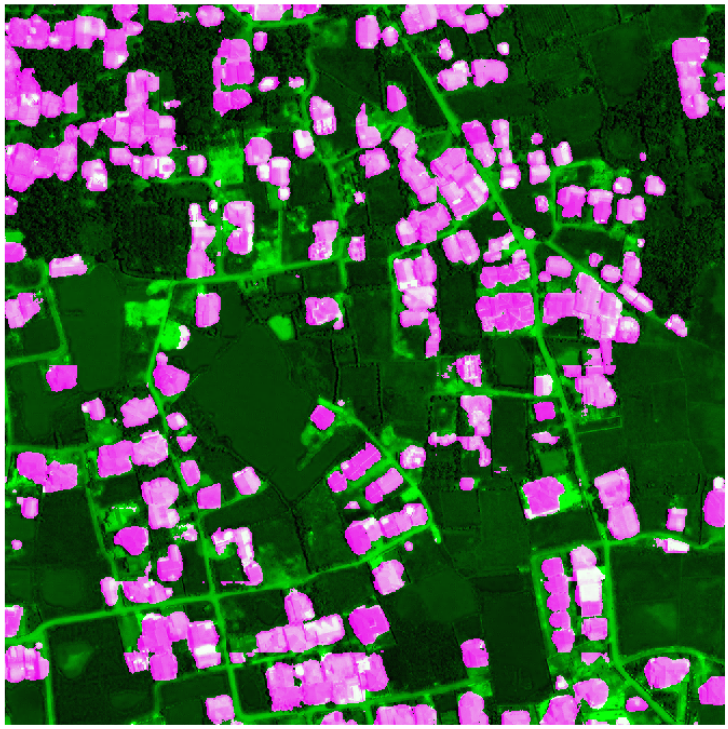

Figure 5: In this image, pixels classified as buildings are given a pink translucent overlay and pixels classified as other are given a green translucent overlay. Pixels which ought to be classified as buildings but are classified as other are false negatives, while pixels which are classified as buildings but ought to be classified as other are false positives.

[0.001,0.00000001], mini-batch size = [4,16], and number of epochs = [40,120]. We then test the network at each of the 60 points and compare the results. This procedure has the advantage over systematic brute-force grid search in that it is far more computationally efficient, requiring only 60 evaluations to provide a 95 percent probability that at least one of the points is within the 5 percent interval around the global maximum. This is true probabilistically, if $1 - 0.05$ is the probability of missing the 5 percent region, $(1 - 0.05)^p$ is the probability of missing it p times, and for $p = 60$, $(1 - 0.05)^{60} = 0.0461$. So, the probability of hitting the 5 percent region is $1 - 0.0461 = 0.954$, or 95.4 percent (Anderson, 2017). Similarly, the probability of hitting the region within 10 percent of the optimum at least once is 99.8 percent. Furthermore, a uniform grid may systematically pass over the region containing the global maximum, while the randomized grid search will not. The resulting hyperparameters determined by this procedure are displayed in table 2 below. The "best" combination of hyperparameters for the purposes of this paper had a high F1 score, as defined below, but also balanced high precision with high recall, and so F1 score was not strictly optimized. In future applications of this procedure to chart markers of economic development for specific purposes, randomized grid search can be used to optimize precision, recall, or any other accuracy metric pertinent to the specific nature and purpose of the classification task.

Net	Learn Rate	Momentum	L2 Regularization	Mini-Batch Size	Num Epochs
2005	0.0842	0.782	0.000431	16	115
2007	0.0314	0.802	0.000897	10	99
2009	0.0205	0.834	0.000869	12	81

Table 2: Results of the randomized grid search algorithm for each hyperparameter and every network.

4.7 Results

After training the 2005, 2007, and 2009 networks with the selected hyperparameters, we applied each network to segment the 30 percent of held-out imagery corresponding to its year and computed a confusion matrix representing the results. The results for each year are summarized by precision, recall, and F1 scores as determined by:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where TP is the number of true positive classifications, FP is the number of false positive classifications, and FN is the number of false negative classifications (Powers, 2011). Higher precision implies that when an object is deemed a building it is very likely to be one, while higher recall implies more pixels are accurately classified as buildings, independent of the quality of those classifications. Since both precision and recall are important, the F1 score, the harmonic mean of precision and recall, enables us to capture the effects of both metrics in one quantity.

2005

		True: Building	True: Other
Predicted	Building	676,481	242,951
	Other	132,986	1,347,582

F1:	0.7826
Precision:	0.7358
Recall:	0.8357

Table 3: Left: Confusion matrix for the 2005 segmentation. Right: 2005 precision, recall, and F1 score.

2007

		True	
		Building	Other
Predicted	Building	929,838	237,680
	Other	154,247	5,578,235

F1:	0.8259
Precision:	0.7964
Recall:	0.8577

Table 4: Left: Confusion matrix for the 2007 segmentation. Right: 2007 precision, recall, and F1 score.

2009

		True	
		Building	Other
Predicted	Building	841,097	263,899
	Other	130,133	5,964,871

F1:	0.8102
Precision:	0.7612
Recall:	0.8660

Table 5: Left: Confusion matrix for the 2009 segmentation. Right: Precision, recall, and F1 score for this procedure.

The results across all three years are represented by the micro-averaged precision, recall, and F1 scores, which take TP, FP, and FN into account across all three classifiers. Micro-averaged precision is calculated by summing all true positives across all three years and dividing that quantity by the sum of all true and false positives from all three years. Micro-averaged recall is calculated similarly, by summing all true positives and dividing by the sum of all true positives and false negatives across all three years. The micro-averaged F1 score is calculated by taking the harmonic mean of the micro-averaged precision and recall (Scott and Matwin, 1999; Wang et al., 2015). Micro-averaging gives each individual pixel classification equal weight in determining the final metrics which is useful because our per-year datasets do not have exactly the same size. Performing this calculation, our micro-averaged precision is 0.7667, our micro-averaged recall is 0.8543, and our micro-averaged F1-score is 0.8081. While the network exhibits superior performance on the qualitatively cleaner 2007 and 2009 imagery, its performance on the 2005 post-disaster imagery is also strong. This indicates its ability to accurately track economic reconstruction beginning at the very moment of crisis and even when completely reliant upon more challenging imagery data.

5 Conclusion

This paper outlines a principled preprocessing and image segmentation pipeline which attained a micro-averaged F1-score of 0.8081 in automatically extracting scientifically useful features from particularly challenging aerial imagery. Because this methodology enables the objective measurement of change across vast landscapes and through time, information which is often difficult or impossible to measure by survey, supplementing the population science literature and survey data

with these extracted metrics will enable population scientists to more completely understand the effects of economic deconstruction and reconstruction on populations, incorporating data previously unavailable in strengthening their arguments. By adjusting the training data, future implementations of this methodology can be utilized to extract other economically valuable features, including markers of road or agricultural development.

6 Acknowledgements

I would like to thank Dr. Paul Bendich, Dr. Duncan Thomas, and Dr. Elizabeth Frankenberg for their guidance and support without which a project of this breadth would not have been possible.

I would also like to thank Keenan Karrigan, Ralph Lawton, Sheila Evans, and Laura Guidera for their assistance producing the labeled training datasets and Peter Katz for technical support.

Finally, I would like to thank the DigitalGlobe Foundation for providing the satellite imagery upon which this project depended.

References

[1] Anderson, Brooks. (2017). Tuning Machine Learning Models. Arlington, Virginia: RiskSpan Inc.

[2] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2016). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. https://arxiv.org/pdf/1511.00561.pdf, Oct 2016

[3] Belfiore, O., Meneghini, C., Parente, C., and Santamaria, R. (2006). Application of Different Pan-Sharpening Methods on WorldView-3 Images. ARPN Journal of Engineering and Applied Sciences, 13, 281-305.

[4] Bergstra, J., and Bengio, Y. (2012). Random Search for Hyperparameter Optimization. Journal of Machine Learning Research, 13.

[5] Bishop, C. (2006). Pattern Recognition and Machine Learning. New York, New York: Springer.

[6] Bottou, L. (1991). Stochastic Gradient Learning in Neural Networks. Proceedings of Neuro-Nîmes 91.

[7] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. Computer Vision and Pattern Recognition, CVPR 2009.

[8] Du, S., Zhang, Y., Quin, R., Yang, Z., Zou, Z., Tang, Y., and Fan, C. (2016). Building Change Detection Using Old Aerial Images and New LiDAR Data. Remote Sensing, 8(12), 1030. DOI:10.3390/rs8121030.

[9] Eigen, D., and Fergus, R. (2015). Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. https://arxiv.org/pdf/1411.4734.pdf, Dec 2015

[10] Esri. (2016). Fundamentals of Panchromatic Sharpening. Retrieved March 3, 2018 from http://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/fundamentals-of-panchromatic-sharpening.htm

[11] Foody, G. (2002). Status of Land Cover Classification Accuracy Assessment. Remote Sensing of Environment, 80, 185-201. DOI:10.1016/S0034-4257(01)00295-4.

[12] Garcia-Garcia, A., Orts-Escolano, S.,Oprea, S.O., Villena-Martinez, V., and Garcia-Rodriguez, J. (2016). A Review on Deep Learning Techniques Applied to Semantic Segmentation. https://arxiv.org/pdf/1704.06857.pdf, April 2016

[13] Ghaffarian, S., and Ghaffarian, S. (2014). Automatic Building Detection Based on Purposive FastICA (PFICA) Algorithm Using Monocular High Resolution Google Earth Images. ISPRS Journal of Photogrammetry and Remote Sensing, 97, 152–159. DOI:10.1016/j.isprsjprs.2014.08.017.

[14] Ghosh, T., Anderson, S., Elvidge, C., and Sutton, P. (2013). Using Nighttime Satellite Imagery as a Proxy Measure of Human Well-Being. Sustainability, 5(12), 4988–5019. DOI:10.3390/su5124988.

[15] Gillespie, T., Frankenberg E., Chum, K., and Thomas, D. (2015). Nighttime lights time series of tsunami damage, recovery, and economic metrics in Sumatra, Indonesia. Remote Sensing Letters, 5(3), 286-294.

[16] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. https://arxiv.org/pdf/1502.01852.pdf, Feb 2015

[17] Henderson, J., Storeygard, A., and Weil, D. (2012). Measuring Economic Growth from Outer Space. American Economic Review, 102(2), 994–1028. DOI:10.1257/aer.102.2.994.

[18] Huertas, A., and Nevatia, R. (1988). Detecting Buildings in Aerial Images. Computer Vision, Graphics, and Image Processing, 41(2), 131–152. DOI:10.1016/0734-189x(88)90016-3.

[19] Ioffe, S., and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/pdf/1502.03167.pdf, March 2015

[20] Jean, N., Burke, M., Xie, M., Davis, M., Lobell, D., and Ermon, S. (2016). Combining Satellite Imagery and Machine Learning to Predict Poverty. Science, 353(6301), 790–794. DOI:10.1126/science.aaf7894.

[21] Karpathy, A., and Li, F. (2015a). CS231n Convolutional Neural Networks for Visual Recognition: Convolutional Neural Networks (CNNs / ConvNets). Stanford University. http://cs231n.github.io/convolutional-networks/

[22] Karpathy, A., and Li, F. (2015b). CS231n Convolutional Neural Networks for Visual Recognition: Linear Classification. Stanford University. http://cs231n.github.io/linear-classify/

[23] Laben, C., and Brower B. (1998). Process for Enhancing the Spatial Resolution of Multispectral Imagery Using Pan-Sharpnening. U.S. Patent 6,011,875.

[24] Lillesand, T., Kiefer, R., and Chipman, J. (2014). Remote Sensing and Image Interpretation. Hoboken, New Jersey: John Wiley.

[25] Maglione, P., Claudio, P., and Andrea, V. (2016). Pan-sharpening WorldView-2: IHS, Brovey and Zhang methods in comparison. International Journal of Engineering and Technology, 8, 673-679.

[26] Maurer, T. (2013). How to Pan-Sharpen Images Using the Gram-Schmidt Pan Sharpen Method – a Recipe. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XL-1/W1, 239-244.

[27] Murphy, K. (2012). Machine Learning: A Probabilistic Perspective. Cambridge, Massachusetts: The MIT Press.

[28] Nair, V., and Hinton, G. (2010). Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning (ICML-10), 807–814,

[29] Ng, A. (2012). CS229 Lecture notes: Supervised Learning. Stanford University. http://cs229.stanford.edu/notes/cs229-notes1.pdf

[30] Powers, D. (2011). Evaluation: From Precision, Recall, and F-Measure to ROC, Informedness, Markedness, and Correlation. Journal of Machine Learning Technologies, 2(1), 37-63.

[31] Ruder, S. (2017). An overview of gradient descent optimization algorithms. https://arxiv.org/pdf/1609.04747.pdf, June 2017

[32] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(4), 211–252.

[33] Saeedi, P., and Zwick, H. (2008). Automatic Building Detection in Aerial and Satellite Images. 10th International Conference on Control, Automation, Robotics and Vision. DOI:10.1109/icarcv.2008.4795590.

[34] Scott, S., and Matwin, S. (1999). Feature Engineering for Text Classification. Proceedings of ICML-99, 16th International Conference on Machine Learning, 379-388.

[35] Shifrin, T., and Adams, M. (2011). Linear Algebra a Geometric Approach. New York, New York: W. H. Freeman and Company.

[36] Suraj, P., Gupta, A., Sharma, M, Paul, S., and Banerjee, S. (2017). On Monitoring Development Using High Resolution Satellite Images. https://arxiv.org/pdf/1712.02282.pdf, Dec 2017

[37] Wang, J., Wang, H., Chen Y., and Liu, C. (2015). A constructive algorithm for unsupervised learning with incremental neural network. Journal of Applied Research and Technology, 12(2).

Appendices

A Neural Network Structure

The following table lists each computational layer of the neural network in order with a brief description of the operation performed. The precise operation of each layer is described in section 4.2.

Layer #	**Layer Type**	**Layer Description**
1	Image Input	100x100x3 images with zerocenter normalization
2	Convolution	64 3x3x3 convolutions with stride [1 1] and padding [1 1 1 1]
3	Batch Normalization	Batch normalization
4	ReLU	ReLU
5	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
6	Batch Normalization	Batch normalization
7	ReLU	ReLU
8	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
9	Convolution	128 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
10	Batch Normalization	Batch normalization
11	ReLU	ReLU
12	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
13	Batch Normalization	Batch normalization
14	ReLU	ReLU
15	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
16	Convolution	256 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
17	Batch Normalization	Batch normalization
18	ReLU	ReLU
19	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]

20	Batch Normalization	Batch normalization
21	ReLU	ReLU
22	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
23	Batch Normalization	Batch normalization
24	ReLU	ReLU
25	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
26	Convolution	512 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
27	Batch Normalization	Batch normalization
28	ReLU	ReLU
29	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
30	Batch Normalization	Batch normalization
31	ReLU	ReLU
32	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
33	Batch Normalization	Batch normalization
34	ReLU	ReLU
35	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
36	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
37	Batch Normalization	Batch normalization
38	ReLU	ReLU
39	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
40	Batch Normalization	Batch normalization
41	ReLU	ReLU
42	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
43	Batch Normalization	Batch normalization
44	ReLU	ReLU

45	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
46	Max Unpooling	Max Unpooling
47	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
48	Batch Normalization	Batch normalization
49	ReLU	ReLU
50	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
51	Batch Normalization	Batch normalization
52	ReLU	ReLU
53	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
54	Batch Normalization	Batch normalization
55	ReLU	ReLU
56	Max Unpooling	Max Unpooling
57	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
58	Batch Normalization	Batch normalization
59	ReLU	ReLU
60	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
61	Batch Normalization	Batch normalization
62	ReLU	ReLU
63	Convolution	256 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
64	Batch Normalization	Batch normalization
65	ReLU	ReLU
66	Max Unpooling	Max Unpooling
67	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
68	Batch Normalization	Batch normalization
69	ReLU	ReLU
70	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]

71	Batch Normalization	Batch normalization
72	ReLU	ReLU
73	Convolution	128 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
74	Batch Normalization	Batch normalization
75	ReLU	ReLU
76	Max Unpooling	Max Unpooling
77	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
78	Batch Normalization	Batch normalization
79	ReLU	ReLU
80	Convolution	64 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
81	Batch Normalization	Batch normalization
82	ReLU	ReLU
83	Max Unpooling	Max Unpooling
84	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
85	Batch Normalization	Batch normalization
86	ReLU	ReLU
87	Convolution	2 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
88	Batch Normalization	Batch normalization
89	ReLU	ReLU
90	Softmax	softmax
91	Pixel Classification Layer	cross-entropy loss

Table 6: Descriptions of the type and function of each neural network layer.