Satellite Remote Sensing and U-Net for Predicting Population Density in Kenya

Undergraduate Senior Project – CSEC 491

Danielle J. Daley[1]

[1] *Yale College Computer Science and Economics, 51 Prospect St, New Haven, 06511, CT, US*

4 May, 2023

Introduction

To answer why economic systems move in the way they do, fundamental institutions rely on the work and results of empirical economic research. Creating effective and efficient policy requires significant, robust, applicable research in these fields. As outlined in a paper by Burke et. al., the main bottleneck in the field is the availability and reliability in of (1). In recent years, with the growing availability of datasets from increasingly powerful satellite sensors and This paper is broken down into three main sections:

1. ***Background**: to provide the reader with a knowledge base to interpret view the project in. I'll discuss the neural networks including the U-net architecture used in this project as well as introduce Google Earth Engine (GEE).*
2. ***Project Report**: to walk through the unique workflow as well as the modeling (including model inputs and response, sampling, training, and prediction results), and discussion on experience and limitations.*
3. ***Implications**: in a conclusion of sorts, to further talk about the promise of this sort of data analysis and it's use to economic research.*

Through slight training set and model adjustments, I was able to improve a standard U-net regression model by 72%, leaving still lots of room for further improvement and tuning.

1 Background

1.1 Neural Networks and Image Regression

1.1.1 Basics of Neural Networks and Image Inputs

Before the introduction of image regression and U-Net, it's important to get a good sense for what a neural network is. To start off with, we'll talk about neurons. Neurons can be thought of as nodes, each with their own level of activation represented by a number. Depending on what that neuron's purpose is in the network, it might be a float value, a binary or some other integer value. An input to a neural network is directly mapped to the first layer. For instance, when an image is the input, the first layer is some integer or float representing a pixel value. When an image is on the RGB color-scale, first layer has three times the number of neurons as a black and white image. That's because an RGB image has three image bands: red, green, and blue. The composite of these bands is the RGB image, therefore to map the RGB image to a set of neurons, we need a neuron for each pixel for each band.

Now that we have our first input layer represented in neurons, we can look at what we're trying to get to: the final activation layer – the result. Designing a neural net to do a task means that we're designing it to get this final layer to represent our output. In classification, this layer may be some n neurons, each with a binary activation, only one being activated at any given time, representing the output classification. In image regression, the final layer has a neuron number, in the desired image size (number of bands $\times$ number of pixels per bands). Each of these neurons has some activation representing some band value of a pixel.

We get from that first input layer to the final layer through additional hidden layers. These layers go in between the first and final layer to add a *neural network* of neuron activation to achieve the desired output in the final layer. These layers can effectively represent patterns in images that contribute to the desired outcome. To understand this, we should talk about how these neurons, activate, change, and how the network *learns*. A this point, it is helpful to start using some math notation. First, we'll represent the initial layer as a matrix $\mathbf{a^{(0)}}$. Now to get the activation numbers for the next layer, $\mathbf{a^{(1)}}$, we need to use another matrix of weights, $\mathbf{W}$, representing a weight of every connection from the first layer to the next.

$$\mathbf{a}^{(0)} = \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ \vdots \end{bmatrix} \quad \text{and} \quad \mathbf{W} = \begin{bmatrix} w_{0,0} & w_{0,0} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,0} & \cdots & w_{1,n} \\ w_{2,0} & w_{2,0} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix}$$

Making the first activation layer $\mathbf{a^{(1)}}$ be

$$\mathbf{a}^{(1)} = activation\_function(\mathbf{W} \cdot \mathbf{a}^{(0)} + \mathbf{b}_0)$$

And the final activation layer

$$a^{(f)} = activation\_function(\mathbf{W}^{(f-1)} a^{(f-1)} + \mathbf{b}_f$$

The 2d bias matrix, $\mathbf{b}_i$, helps us "bias" the activation, giving it different thresholds for activation based on what $a_n^{(1)}$ value we're going for. Those familiar with linear regression will see where this is going. In linear regression models, we have the exact value Y is made up of some matrix of inputs, $\mathbf{x}$ and some coefficient matrix β, where $Y = \beta_0 + \beta\mathbf{x}$ and the estimation of Y, $\hat{Y} = \hat{\beta}_0 + \hat{\beta}\mathbf{x} + \epsilon$ where, to make the most accurate $\hat{Y}$ (minimize $|Y - \hat{Y}|$), we want to minimize ϵ. Similarly, machine learning in neural networks is about minimizing loss/cost. The simplest loss function would be subtracting the predicted final activation layer with the one using the current $\mathbf{a}$ and W. Because our purpose is to minimize loss with various inputs, we use something called gradient decent. Essentially, gradient decent is the process of determining which direction to "descend" the loss function in an effort to find a minima. The exact speed and direction for which to descend is determined by the machine learning model optimizer. Stochastic gradient descent (SGD) is a baseline optimizer.

The *activation_function* is some literal function that transforms the otherwise linear activation calculation. Two of the most common activation functions are sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) and rectifier linear unit ($ReLU = max(0, x)$). These have the express purpose of achieving nonlinearity to produce complex and powerful predictors (*2*).

1.1.2 Convolutional Neural Networks

A type of neural network commonly used in image processing is a convolutional neural network (CNN). CNNs have at least one convolutional layer. During this convolutional layer a mathematical... convolution is applied which helps in grabbing local information and reducing the overall complexity of the model.

1.1.3 U-Net

A fairly novel neural network (CNN) now commonly seen in biomedical research is U-net (named after its U-shaped architecture) (*3*). U-net utilizes encoder, decoder, and skip connection that connects corresponding layers in the encoder and decoder. The encoder is designed to extract features from the input image and compress them into a more compressed version in actions called the contacting path. The decoder then uses this compressed version to generate a a pixel-wise label indicating the value of each pixel in the image. This label is called the segmentation map. This part of the U-shape helps determine the "what". All the while, the skip connection allows the decoder to use information from earlier layers in the encoder to improve the quality of the segmentation map. This does a great job at capturing smaller details and pixel context (it's relation to pixels around it) – determining the "where". This CNN has been proven effective for use in satellite image segmentation in numerous recent publications including (*4*).

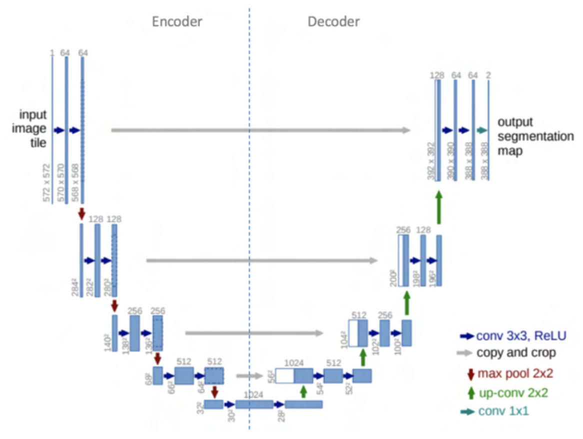


Figure 1: *A figure from (3) demonstrating the U-Net architecture.*

1.2 Google Earth Engine (GEE)

The backbone of project's workflow is Google Earth Engine and Google Cloud Storage. Google Earth Engine API allows you to access Google's Earth Engine, a cloud-based geospatial processing platform. Assets can be stored and shared on your account or in cloud projects. Assets include traditional geospatial asset datatypes based in raster vectors, shapefiles, and more generally GeoTIFF files. These GEE datatypes (ee.*) also have the are compatable with many, . GEE has a web-based IDE based in JavaScript that allows you to view, manage, and create assets, as well as connect to other Google JavaScript API.

2 Project Report

2.1 Workflow

Traditionally, to make these kinds of satellite image predictions, researchers manually create training/testing datasets, make them accessible to a model to train on, manually feed converted prediction input, then
The workflow used in this project takes in an ee.Image/ee.ImageCollection/ee.FeatureCollection asset from GEE and uploads an ee.Image asset prediction all within one Python notebook.

2.2 Modeling

I trained and tweaked multiple U-net TensorFlow Keras models to take in pre-proccessed train and test datasets of Landsat7 satellite imagery with a Kenya 2009 population density raster as the response. Each of these bands (including the population raster as a 'target' band) stacked into an array for actual image sampling and training/evaluation dataset creation. From there, two forms of selective-sampling were used.

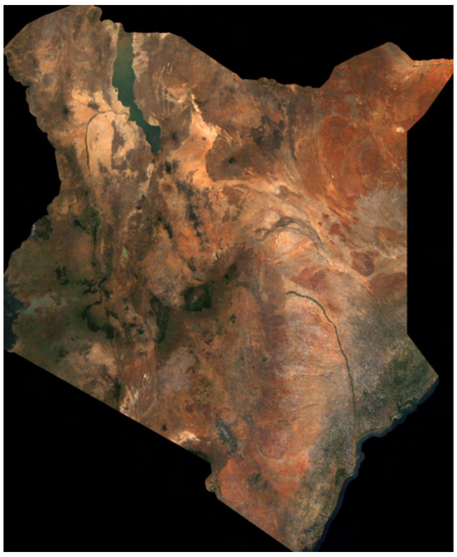


Figure 2: *RGB Bands Image*

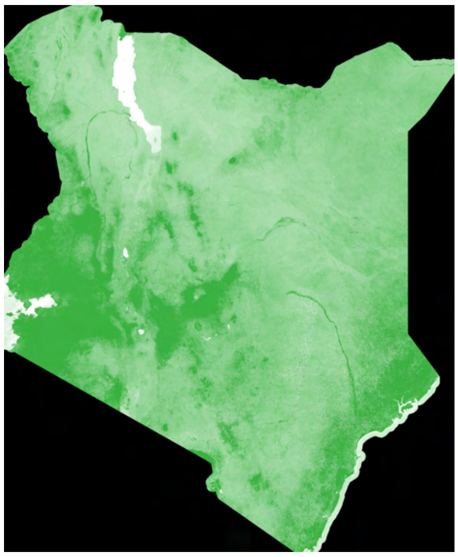


Figure 3: *NDVI Band Image*

2.2.1 Inputs and Response

All models tested use four image bands as inputs from a United States Geological Survey (USGS) data collection: three optical bands (RGB), and normalized difference of the Near Infared (NIR) band and Red band (this is also known as the Landsat Normalized Difference Vegetation Index or NDVI). These inputs were then used to train trained to predict the response in the form of a population density raster.

Landsat7. The specific satellite imagery dataset used was the Landsat 7 Tier 1 Collection 1, adjusted to account for surface reflectance (*LANDSAT/LE07/C01/T1_SR*). The Tier 1 classification means that it considered suitable for time-series analysis. This is a requirement for expansion of this project to include predicting population changes over time (see relevant discussion in Implications Section). The surface reflectance algorithmic adjustments done on the data by USGS further reduce atmospheric noise from our imagery. Beyond this, this project utilized a GEE function for Landsat 7 (*cloudMask457*) that attempts to remove bad pixels that it believes are disrupted by clouds. Figure 2 shows the RGB Bands of the actual input image and Figure 3 shows the NDVI band (stretched to 2σ to show pixel texture of values).

Population Raster. Next, we need something that the model will train to predict – the response. For this, the project used an image of Kenya in a raster data format, each pixel/cell representing the reported population density per 50km^2 derived in individual constituencies' sub-locations. This data was then projected onto a matching shapefile (vector data format for geospatial analysis). Raster creation was done outside the scope of this project and created primarily as part of an ongoing project at SPIRES Lab at Yale University, 2022. The result is a nearly 1 trillion pixel raster to be used for population mapping (Figure 4).

This census raster could easily be put into a logarithmic representation (Figure 5). This allows us to estimate population *magnitude*. You can see a visual representation of why this is important in the histogram charts below. The first shows a linear scale of the population density frequency and the other shows a logarithmic scale (using same pixel range.)

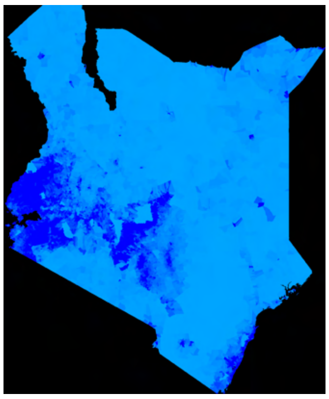


Figure 4: *Population Density (per 50km^2)*

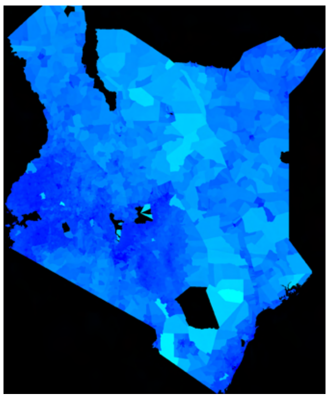
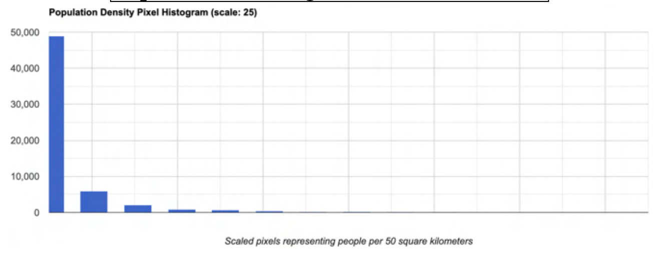
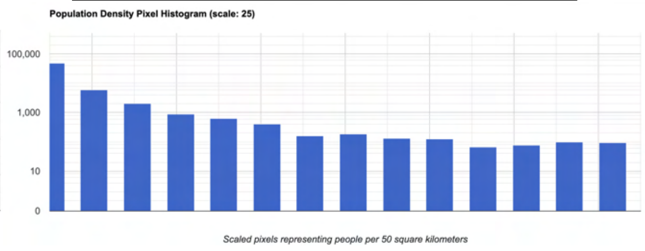


Figure 5: *Logarithmic Population Density*

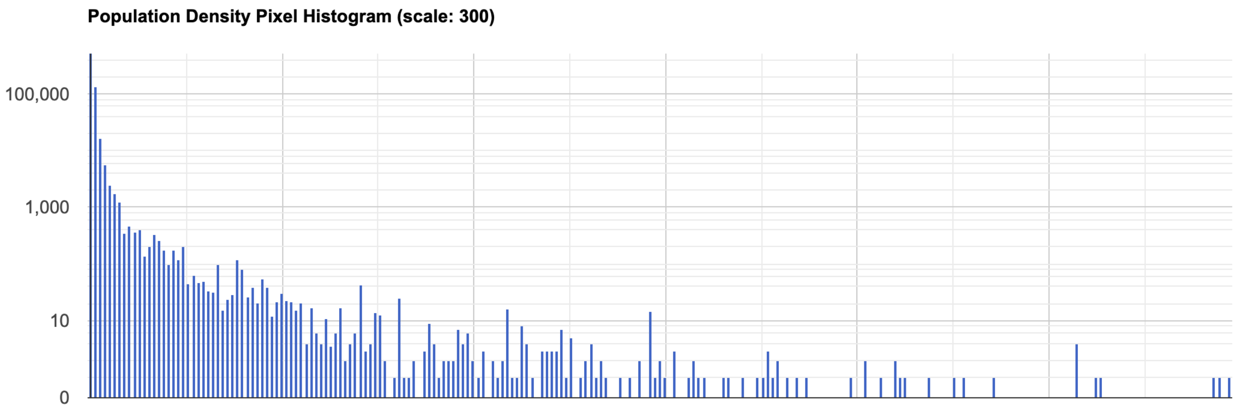
Population Histogram on Linear Scale



Population Histogram on Logarithmic Scale



The above histograms are only looking at the first 7,500 pixel bucket values. With a logarithmic scale, we can see over 15 times that range:



Testing models on this logarithmic raster means we'll be testing on a set of data with a tighter normalized standard deviation which might be easier for the network to predict based on the given inputs. Again, this will also allow us to directly train on and predict population magnitude.

Census Population Density Raster Statistics

Stat	**Linear Raster**	**Log Raster**	**Linear Pred Region**	**Log Pred Region**
Min	0	-6.077	0	-2.807
Max	118993.773	11.6867	118993.773	11.687
Mean	67.028	2.286	250.158	4.296
Median	38.963	2.188	110.466	4.402
SD (σ)	448.787	1.978	1489.346	1.551

The 'Pred Region' is the reduced predictive region zone explained in 2.2.4

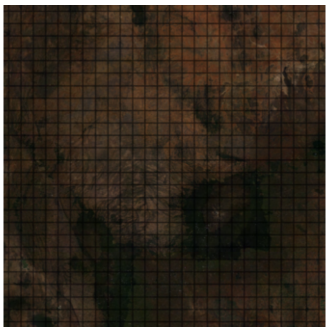


Figure 7: *Closer view of tiling over input image*

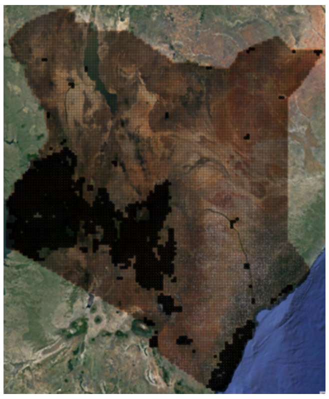


Figure 8: *Centroids based on average tile population density restriction*

2.2.2 Sampling

The greatest limitation to a CNN's performance important part of a model's performance isn't the model itself, but the training/evaluation data. This is especially true in satellite-based input models (*1*). When training on such a large dataset (entire RGB band and spacial raster values), it's too expensive and inefficient to train on the entire image with any significant resolution. By only selecting certain parts of the image to sample, we can both improve the quality and size of the training/evaluation data. The approach used in this project is to sample mostly from high-populated areas, the naive cutoff being an average of 50 people per 50km^2. Sample points were randomly shuffled once determined. This random shuffling remained the same for all models using the same sample points. That is, we have random sampling within the selected sample points, but that randomness remained the same for all of our model training for consistency (similar to setting a seed for random sampling).

Tiling. In the context of image sampling, our sample points here are centroids of whatever patch size we're using for training. In most of my models, a patch size of 256x256 pixels were used. So we want to sample each patch where the average population density is greater than 50 people (per 50km^2) i.e. where the average value in our raster is 50 for a 256x256 patch on the input image. To do this without grabbing overlapping patches, we need to use a tiling table for the country. Using another asset produced at SPIRES, patch tiling can be viewed in Figures 6 and 7.

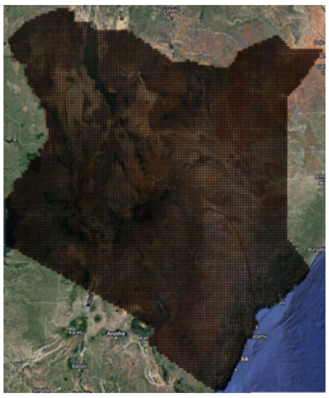


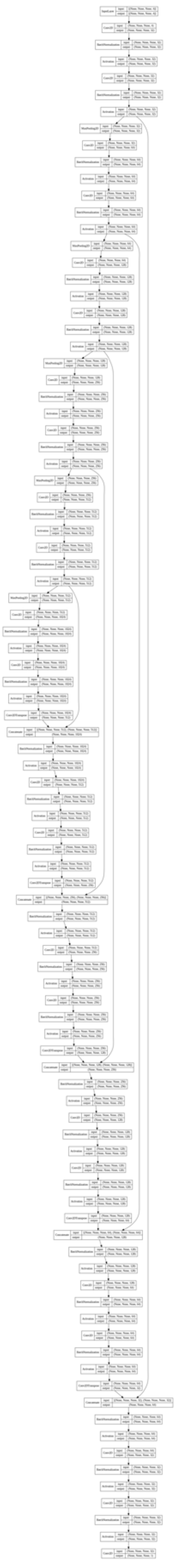
Figure 6: *256x256 tiling over input image*

From there, we can grab the points at the center of the tiles which meet our tile population density requirement to get the centroids shown in Figure 8.

Stratification. As will be shown in Prediction Evolution, there was an issue that arose with this approach, highlighting the value of proper sampling. As can be seen in Figure 8, some selected sampling tiles (high population tiles) encapsulate non-selected tiles (low population tiles). As such, if trained on high population tiles alone, the U-net model will not have trained on the context of these higher population areas surrounding very low ones. Often these are low population tiles cover some less-hospitable terrain or uninhabitable area like lakes. This brings down their density even if near more high-density land.

To improve predictions slightly, a random sampling of 10% of all low-population tiles was added to the training/evaluation set in an effort to allow the model to get a greater context for low populated areas.

2.2.3 Compilation/Training – The Model Itself



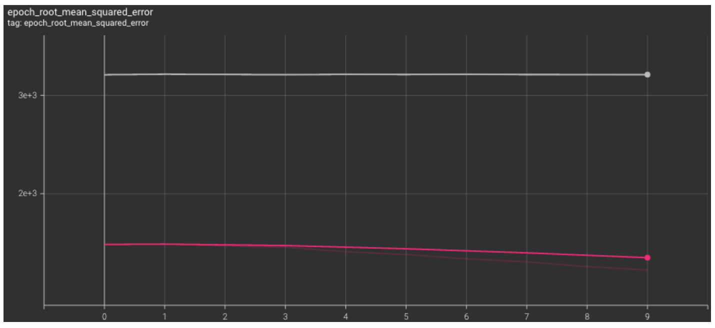
The actual U-Net model layer specifications used were taken from the Google UNET Regression Demo described in 2.1. To the left is an image produced from a *tf.keras.utils.plot_model* method on the model to show you can physically see the "U" shape. It starts by encoding reduce the patches, keeping them grouped in pools, then decoding by upsampling to the proper resolution. Through each encoder and decoder block lay normal a series of 2d convolutional layer followed by concatenation layer, normalization layer, then finally the activation layer, matching the U-Net architecture in Figure 1.

Activation Function. The model uses activation layers with the ReLU activation function along with a final ReLU activation. Sigmoid, another very common activation function, returns a value from 0 to 1. Because we want a real-value prediction, we'll choose ReLU. If we look back to Section 1.1.1, this is our $activation\_function$.

Compilation Parameters. Keras model compilation completes the untrained model creation by specifying parameters to dictate how the model interprets training. This is where the loss function and optimizer are included.

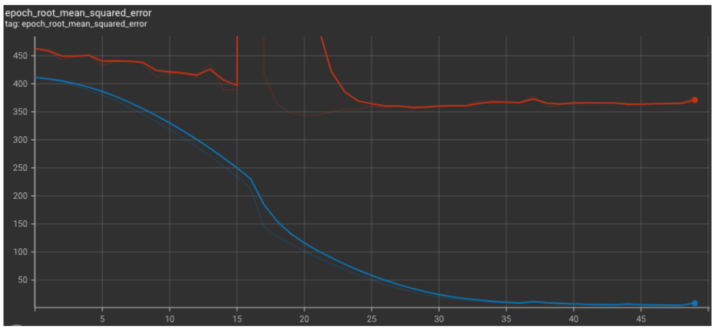
Loss. Because we're running an image regression, mean standard error (MSE) was the chosen loss function. MSE calculates the average squared difference between the predicted value and the actual response value. If we were doing some sort of image classification (response and prediction values would be in some discrete integer classification value list), we could have used some Cross entropy loss function.

Optimizer. The classical stochastic gradient decent has begun to make a comeback in recent years, with many demonstrating that it is typically better at generalization than adaptive moment estimation (Adam), the most popular optimizer. Reminder from 1.1.1: an optimizer's job is find the minima of the losses given the $\mathbf{a}$ activation neurons and $\mathbf{W}$ weights. Adam then uses a different algorithm to adjust weights to activation neurons. Though standard Adam has been shown to generalize worse than SDG, it trains faster and similarly when tuned correctly (*5*). Therefore, Adam was used for *most* models.



The chart above shows a callback comparison between the training root MSE (RMSE) of an SGD optimized vs an Adam optimized model, all else equal. The Adam model is shown smoothed slightly in pink (actual values dark purple below). The SGD model is shown in grey, the smoothed version having little visible effect on the curve.

Training. From changes in model specifications to simple re-runs, over 20 models were trained, 4 representative model predictions viewable in 2.2.4. The model was fit using a training and evaluation dataset of exported feature stacks of the inputs/responses in *tf.data.Dataset* formatting. These feature stacks were split roughly 2/3 to training, 1/3 to evaluation (actual split 0.66). Epochs varied over model training sessions, but steps per epoch remained at the size of the training set divided by a batch size of 16. For almost every model, over-fitting began to occur at around 20-25 epochs. Each epoch signifies one pass through the entire training dataset. This means that the MSE loss on the training set continued to decrease with continuing epochs, but the loss on the validation set fluctuated around the same number.



The above chart from a 50-epoch model callback. It's smoothed with the dark red representing actual validation RMSE and blue representing actual/smoothed training RMSE. Ignoring the outlier spike at epoch 15 (this will be discussed in 2.3), at around epoch 20, the validation RMSE stalls improvement.

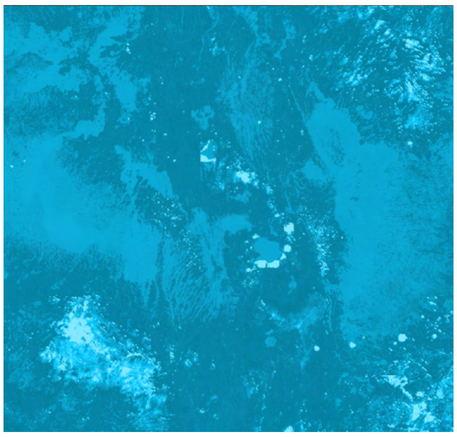
2.2.4 Prediction Evolution

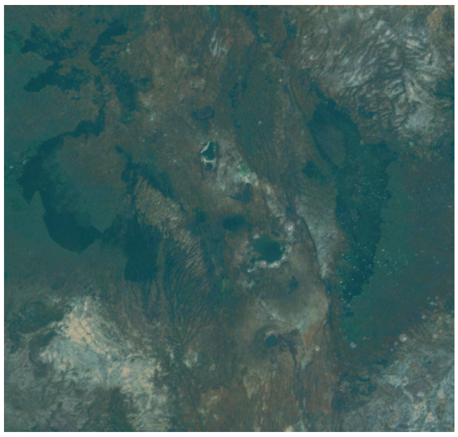
To export prediction images on the entire country takes about 14 hours total (this includes exporting, predicting, and uploading to GEE). As such, most prediction images are of a selected a polygon containing interesting features and a high response variance ($\sigma^2 \approx 2217121$ vs the total country's $\sigma^2 \approx 201410$). By selecting a higher variance sample area with equivalent ranges of [0,11899.773] (bottom right of region includes Nairobi), we should be able to see a lot of the predictive power of the model without running computations on the entire Kenya geometry.

Through tuning, the RMSE for the validation set decreased from 578.90 to 162.53, 71.92% decrease.

Note*: The workflow produces a prediction Google Earth Engine asset image (ee.Image). All prediction images below will be displayed on a stretch of 3σ to see image texture and a color palette of ['004C5F', '80e6ff'].*

Prediction Image 1





This prediction was used as a baseline for future modeling. It used sampling only from the tiles with an average population density of 50 per 50km^2, utilized a patch size of 128x128, used final ReLU activation layer, and ran on 10 epochs, and used the standard linear raster for training/evaluation. Interestingly, this model used the Keras 'BinaryCrossentropy' loss function instead of MSE.

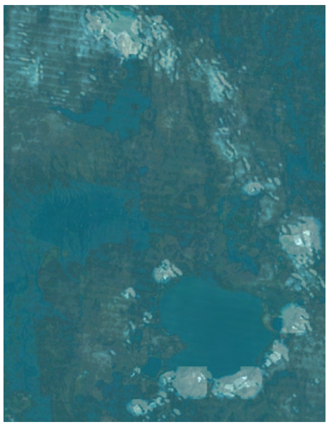
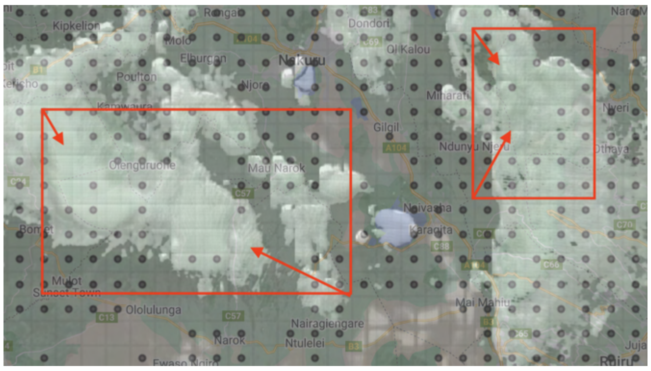


Figure 9: *Prediction capturing estimating high population when RGB bands closer to white/brighter colors*

We can see in the resulting image that the model did a good job at normalizing the RGB bands to see which weights *classified* features that would predict population. In , It actually selects a some buildings as having a high population density (see Fig. 9). Despite the interesting results, it's using a loss function that is much less effective at capturing the proper regression prediction.

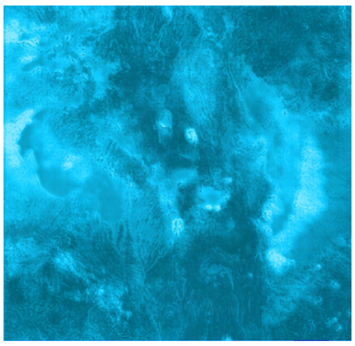
In all models following used the MSE loss function, and shortly after this prediction, the patch size was changed to 256x256 and models were trained on epochs of 10 or greater. The next model, a MSE, 256 patch size, 10 epoch model, produced a final validation set **RMSE of 578.90**. This is our baseline and what we'll use to evaluate futher progress.

Sampling Switch. As stated in 2.2.2, sampling was switched to a naive binary stratification approach. The image below demonstrates why:

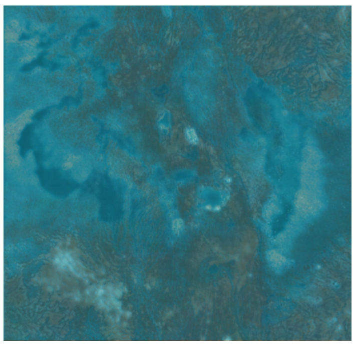


Within the red boxes are pockets of low density tiles in high density tiles (the points represent the centroids of high density tiles, and the white represents predictions of high population). The U-net model was never given the chance to try to understand this context because those tiles were not part of the training or evaluation data set. This encourage the addition of 10% of the low-density tiles to be added into the sampling set, increasing the total number of sample points from 1819 to 2650.

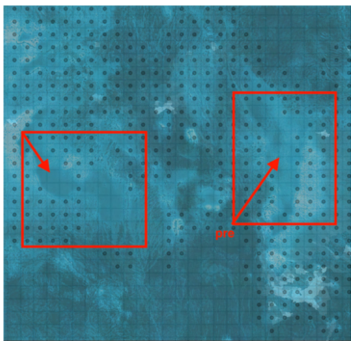
Prediction Image 2



Prediction



Prediction over RGB input

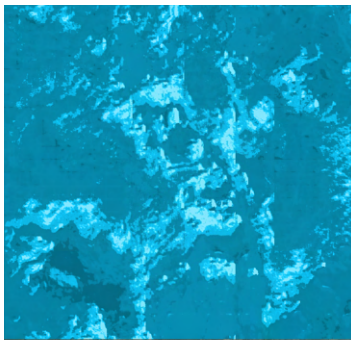


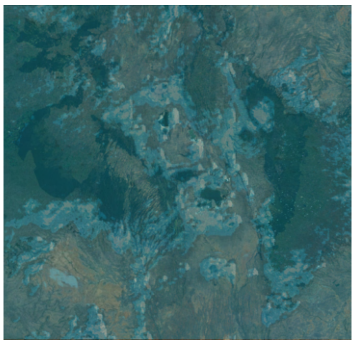
Prediction, response, centroids

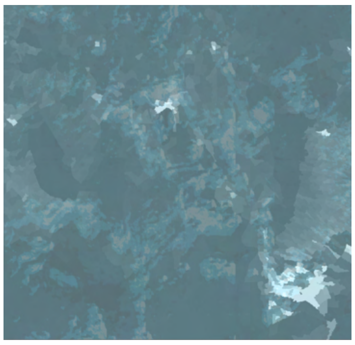
These images show the improvement from adding those low density tiles. Third includes the 256x256 tiles and high density centroids overlaid. The low-density tiles have now been adjusted to be predict a smaller population density than their surrounding tiles. This is a great demonstration on why the *inputs* of a model make a huge difference in the model's actual predictive performance. Good sampling creates a stronger training and validation set, and strong training and validation set make for a more predictive model

The model is improving, but still not incredibly strong. In this region, there are still visibly density disparities near and around Nairobi – the extra white seen in the bottom right of the the third image shows this.

Prediction Image 3

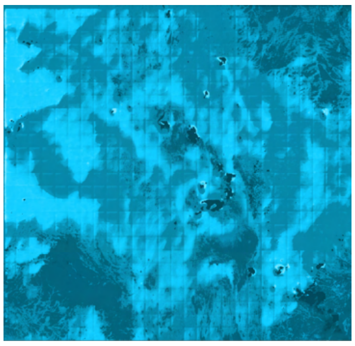

Prediction

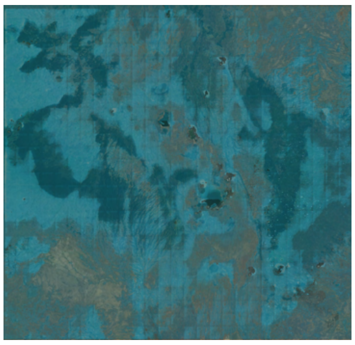

Prediction over RGB input

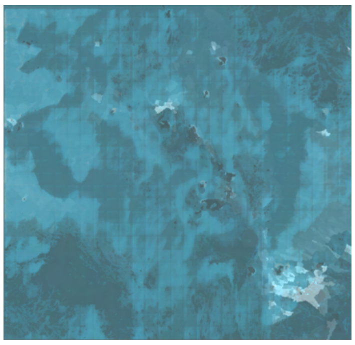

Prediction over response

This prediction was run with the new sampling, 20 epochs, MSE loss, and 256x256 patches. The model evaluated to a total 340.56 RMSE over the entire country's sample points (including the 10% low population tiles) and 440.59 over the test dataset. That means that for all of the 256x256 patches sampled, the average RMSE of the values was only 340.56 – not bad considering our dataset.

Prediction Image 4


Prediction


Prediction over RGB input


Prediction over response (log)

This prediction used the logarithmically transformed response raster. We can noticeably see less texture here. Notably, this model ran for 50 epochs. Prediction images produced at these high epoch ranges are visibly grainy like this one. Though they're not as nice to look at, they produce strong results. This model had a 0.89 RMSE over the entire dataset and a 2.47 RMSE over the validation set. Another 50 epoch model on the linear response raster produced a 355 RMSE on the validation set, granting a total set validation RMSE of 162.53.

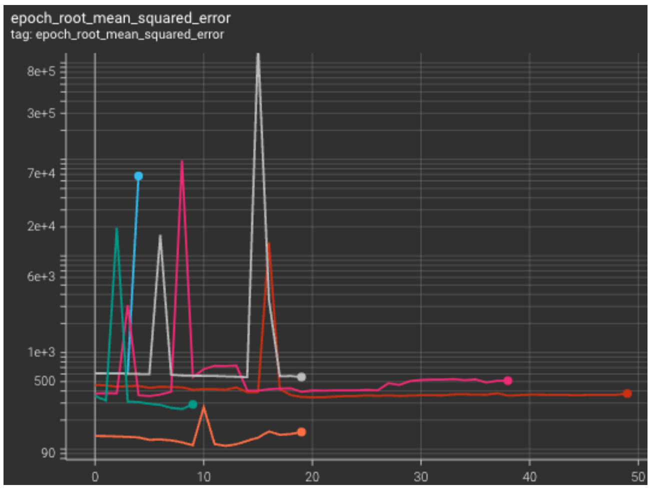
2.3 Further Discussion

2.3.1 Needs More Investigation

There are a few points of discussion which didn't fit too neatly into the model/methods description but arose from implementing the work described in 2.2. They are discussed below.

Pixelation One of the first things I became concerned about was the dramatic difference in appearance between some prediction images (like 1 and 2) and others (like 3 and 4). The vast majority of produced images looked like or "worse" than 3 and 4 – they were grainy and at a larger scale, it's easy to see where the tile borders are (even at this small scale, prediction image 4 has this visible *tiling*). I didn't look too deeply into this matter because "grainy" issue really only affects the visualization – the ability to numerically predict population in areas is what matters i.e. the actual prediction values. Nevertheless, my initial thought was that it was a corruption issue in the way that images were exported for prediction, that the *input image* itself was grainy, so the model produced these types of images. I also noticed that the nicest photos seem to be from lower-epoch models. It's likely that the smoother-looking images are just signs that the CNN didn't make many adjustments yet. However, some 10-epoch models still didn't create a smooth image.

The Mysterious Loss Jump: Across at least 8 training sessions, the validation RMSE spiked to non-intuitive values. This wouldn't happen every training session but approx 50% of the time, the odds of it happening increasing as more epochs were run. Given the same compilation and training parameters and datasets, one model might run into this problem and another might not.



For instance, in the graph above, the green spike shows that model's valuation RMSE at epoch two being 19621, making the MSE 384,983,641. Considering the highest value in the linear raster is less than 11,900, even if the model predicted 0, the MSE would only be 141,586,201, less than half of this perceived spike. Because the raster is in floats, it could possibly be a float overflow. The training/evaluation points did not change from model to model (assuming the same sample points), so the issue definitely lies with a specific member or members of the validation set.

2.3.2 Future Improvements

There are some elements of the project that I wish I had more time to test/evaluate.

Response/Input tuning: The more I tested and attempted to trace sources of faulty, the greater aware I became of some of the flaws in the census raster. For instance, there are large swaths of land with a zero population density. In some areas that's simply because of issues with the census, however in other areas it's because of preservation land. In summary, there are some exogenous variables that are not being evaluated in this model, leading to endogeneity in our predictions. As stated multiple times in this paper, the quality of the feature sets (training data) fed into the model is a huge determinant in the actual predictive power of the model. If we're trying to observe a casual effect, we need to include. Burke et al. (*1*) talks about the immense difference using handcrafted features can make in their article on using satellite imagery and machine learning to promote social science research.

Landsat7 Tier1 SR was used for this project, different imagery datasets exist. I didn't test these models using Sentinel (very new, doesn't go back to 2009) or Nightlights (would probably be a great band for population estimation). Part of the issue is a comparatively low revisit rate of these newer sensors over Kenya and African countries in general. That doesn't mean they wouldn't be useful, but it's a major reason why Landsat was used for this project.

Sampling improvements. The small sampling adjustment made in this project demonstrates the weight that sampling has on a model's predictive power. An ideal sampling would lean further into this, normalizing the distribution of pixel sample densities into bins, and sampling from these bins in roughly equal amounts. Again, this would require more investigation of our response dataset.

Efficiency: Most of the time for this project went into learning about machine learning, convolutional neural networks, image sampling, Google Earth Engine API, and Google Cloud (for the workflow). I had to restrict the scope slightly to fully understand the work I was doing. Something I wish I had more time for was trying more unconventional models. I ran some with SGD, but not enough to see if it could actually generalize better given the somewhat noisy response roster (as previously described). I also didn't really touch the activation layers, though could have tested some slightly more expensive activation functions. Things like the dying ReLU problem were also unaddressed.

Overall the training features and model has a lot of room to become more robust, accurate (predictions closer to actual), and precise (inverse variance).

3 General Implications

This project showed a basic demonstration of an improved workflow and potential predictive power of satellite remote sensing with the U-net architecture. As discussed, much can still be improved upon, but that is the point – this is only a first step and it shows much promise and ability to *be improved*. Moving beyond the scope of this small experiment, this kind of data prediction has widespread applications.

One of the difficult roadblocks in empirical economics is gaining sufficient and relevant data. Natural limitations lie in methods and opportunity of collection and measurement. Data that researchers want is often expensive, time-consuming, elaborate, or often outright unfeasible to acquire. Data availability is extremely valuable for estimating clausal inference related to some of the biggest, most complex to some of the simplest, most approachable questions in economics research.

Government surveys are some of the most powerful data sets due to their versatility, size, and scope. They both directly and indirectly inform vital policy, being instrumental in nearly all branches of academic research. Unfortunately, time between these surveys can often impose harsh limits to the strength and scope of research using the data. Depending on country and type of survey, the time between often surpasses a decade. This limits the power of these extensive surveys, especially when doing more limited time-series studies. An inherit difficulty lies in estimating change between two surveys, potentially decades apart.

Availability of satellite imagery and robust machine learning techniques and technologies have allowed powerful tools into the hands of many researchers. Increasing in the last decade, researchers have expanded testing the use of remote sensing via using these tools to try to produce survey-like data. This has been especially useful in environmental impact research, but recent projects have begun to increasingly delve into general income and population estimation. Examples of this include a paper by (*6*), estimating income and population data in the US and the working paper by (*7*) doing similar to estimate anti-poverty programs in the US.

This data and this project specifically will be useful towards future work on using satellite data to make predictions specifically on Kenyan demographics, similar to work by Burgess et al. (*8*).

Bibliography

1. M. Burke, A. Driscoll, D. B. Lobell, S. Ermon, *Science* **371**, eabe8628, eprint: https://www.science.org/doi/pdf/10.1126/science.abe8628, (https://www.science.org/doi/abs/10.1126/science.abe8628) (2021).
2. D. Danciu, in pp. 331–357, ISBN: 978-1-59904-996-0.
3. O. Ronneberger, P. Fischer, T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015, (https://arxiv.org/abs/1505.04597).
4. P. Ulmas, I. Liiv, *CoRR* **abs/2003.02899**, arXiv: 2003.02899, (https://arxiv.org/abs/2003.02899) (2020).
5. P. Zhou *et al.*, *CoRR* **abs/2010.05627**, arXiv: 2010.05627, (https://arxiv.org/abs/2010.05627) (2020).
6. A. Khachiyan *et al.*, "Using Neural Networks to Predict Micro-Spatial Economic Growth", Working Paper 29569 (National Bureau of Economic Research, 2021), (http://www.nber.org/papers/w29569).
7. L. Y. Huang, S. M. Hsiang, M. Gonzalez-Navarro, "Using Satellite Imagery and Deep Learning to Evaluate the Impact of Anti-Poverty Programs", Working Paper 29105 (National Bureau of Economic Research, 2021), (http://www.nber.org/papers/w29105).
8. R. Burgess, R. Jedwab, E. Miguel, A. Morjaria, G. Padró i Miquel, *American Economic Review* **105**, 1817–51, (https://www.aeaweb.org/articles?id=10.1257/aer.20131031) (2015).