# Stanford ACMLab Fall 2019 Project
# Mapping Income Distribution with Machine Learning

October 17, 2019

## 1 Introduction

In the ACMLab Fall Project, you will sharpen your deep learning skills by engaging with a high-impact, unsolved real-world problem. In doing so, you will learn about infrastructure, data cleaning, implementation, model evaluation, and other critical tools of a machine learning practitioner. At the end of the project, at the very least you will have gained valuable skills and expertise, and at best will have made meaningful contributions to an active field of research.

## 2 Problem statement

This year's Fall Project is in the field of **developmental economics**. A critical problem in developmental economics is mapping the high disparities in developmental indicators, such as GDP per capita, on a subnational or local level. Such knowledge is critical for understanding key issues in developmental aid, climate change, sustainable development, infrastructure, market access, conflict, and myriad other issues. Unfortunately, it is frequently the case that nations for which such data is particularly valuable are also those least capable of collecting such data internally through censuses and economic records.

Previous work sponsored by the Bill and Melinda Gates Foundation (see `https://www.worldpop.org/resources/docs/pdf/Poverty-mapping-report.pdf`) attempted to infer high-resolution development maps based on covariance analysis with several dozen hand-selected data sets. However, their approach is limited to regions for which those specific datasets are available and is labor-intensive the generalize. We would like to use **computer vision** to automatically learn features from satellite imagery to predict the spatial distribution of income within a nation or subnational division.

For this Project, you will tackle a miniature version of this problem by attempting to learn and predict the spatial distribution of income in the urban areas of California. You will leverage the rich and high-resolution economic statistics available for these regions to work towards a proof-of-concept for the model, with the hope that its learning ability can generalize to other archetypes of geographic development in developing nations.

Specifically, you will train and test your model on satellite imagery from the **Los Angeles Metropolitan Area**, and will evaluate the model's ability to predict the income distribution in other urban areas of California. This is exactly analogous to using nations where high-resolution data is available to generalize to nations where such data is unavailable — you will train on Los Angeles and deploy to other urban areas.

# 3 Data

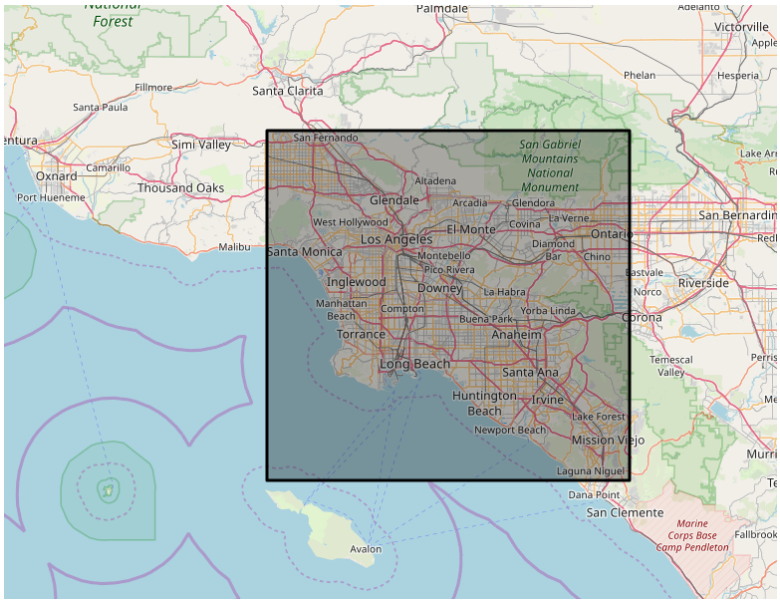
All datasets necessary to complete this project have been prepared and are provided for your convenience. The bulk of the training data is in the form of satellite imagery of the Los Angeles metropolitan area. The training labels will be derived from zip code-level data from the Internal Revenue Service, as well as geospatial data for each zip code.

You are welcome to use additional data, as long as you notify `@bjing` beforehand and it is **restricted to the Los Angeles Metropolitan Area**. However, at evaluation time your model must be able to formulate a prediction based on satellite imagery alone, and will not be provided any additional data.

In addition to the satellite and census data described in detail below, you may also find the elevation data in `worldelev.zip` useful. Functions to access this data are provided in `util.py`.

## 3.1 Satellite data

Satellite data for the Los Angeles Metropolitan Area is provided for the region roughly bounded by 33.5°N, 34.3°N, 118.6°W, and 117.6°W, shown in the rectangle below.



More precisely, *satellite web map tiles* at zoom level $Z = 14$ are provided for the region. Web map tiles are $256 \times 256$ square tiles of the earth's surface under the Web Mercator projection. A tile at zoom level $Z = 0$ covers the entire globe, and each increase in $Z$ corresponds to a twofold increase in resolution. Therefore $Z = 14$ roughly corresponds to 10 meters per pixel at the latitude of Los Angeles. The tiles are identified by an $(x, y)$ integer pair representing the column and row number of the tile; $x$ is zero-indexed from the left and $y$ is zero indexed from the top. (The top is not the North Pole, but rather the latitude which makes the $Z = 0$ tile exactly square.) If one extrapolates these integer indices to a coordinate system, one obtains

$$x = \frac{2^Z}{360}(\text{longitutde} + 180^\circ)$$

$$y = 2^Z \left(\pi - \log\tan\left(\frac{\pi}{4} + \frac{\text{latitude}}{2}\right)\right)/(2\pi)$$

where the longitude must be in degrees and the latitude in radians. These transformations, as well as their inverses, are provided in `webmercator.py`.

The bounding box corresponds $x$ indices in the range $2794 \leq x < 2839$ and $y$ indices in the range $6528 \leq y < 6572$ for a total of $N = 1980$ tiles. These are provided in `imagery.zip` and are named according to the convention `14_{x}_{y}.jpg`. Utilities for converting `.jpg` to `.npy` are provided in `util.py`.

### 3.2 Census data

You will use income and population data at the zip code level to compute training labels. The data is provided for the entire United States in `16zpallnoagi.csv`. There are many fields in this dataset, but you only need to be concerned with `ZIPCODE`, which is the zip code, `N1`, which is the number of income returns, and `A02650`, which is the sum of all household incomes in that zip code. Please disregard other income metrics.

You will also need to map zip codes to satellite images, and you will do so by means of the `ziplatlon.csv` dataset. The `zip` field contains the zip code, and the `latitude` and `longitude` fields contain the location of the geometric center of that zip code. You will use this information to associate tiles with zip codes as described in the section below.

## 4 Implementation

You are free to implement the model based on your discretion. You are not required to use PyTorch or TensorFlow, or even Python, but if you choose to use some other deep learning library please let `@bjing` know. **If you do use Python, you must use Python3**.

For evaluative purposes we will more precisely define a notion of the target metric. That is, given a set of zip codes $\mathcal{Z}$ and a set of tiles $T$, we generate the ground truth labels $Y(t)$ for each $t \in T$ as follows:

**For each** $z \in \mathcal{Z}$ :
  **Assign** $z$ to the tile which contains the center of $z$
**For each** $t \in \mathcal{T}$ which does not have a zip code assigned:
  **If** the center of $t$ is over land:
    **Assign** the zip code $z$ whose center is closest to the center of $t$ to $t$
**For each** $z \in \mathcal{Z}$ :
  **Distribute** the population and income of $z$ evenly across all tiles $t$ to which it is assigned
**For each** $t \in \mathcal{T}$:
  **Assign** $Y(t) \leftarrow$ the income of $t$ divided by the population of $t$

Following this procedure, we see that the ground truth labels $Y(t)$ are in the form of a **weighted average** of the household incomes of the contributing zip codes. (Consider population and number of income returns to be equivalent.) This is the label your model should output. You do not need to consider zip codes whose geometric centers are not inside any of the supplied tiles.

# 5 Evaluation

At the conclusion of the project, you will submit your **code**, your final **model**, and a written **report**. The code will not be directly evaluated, but will be used to run the model on a comprehensive **test set** which you will not be able to access until the project concludes. To do so, you must **clearly provide** a `predict` function in your code which accepts a single string parameter which is a path to a $3 \times 256 \times 256$ `.npy` array and returns a single **real number**. You may assume that the test arrays will be of the exact same resolution, form, and range as the input images in **.npy** format.

Because we would like to predict the correct output for the largest population as possible, we will **weigh** the evaluation for each tile by the population of the tile. You may assume that errors are penalized proportional to the squared error in the label times the population of the tile. (It follows that your model will not be evaluated on oceanic tiles.)

A **winning team** will be selected and recognized after the project concludes. The winner will be selected with $\frac{2}{3}$ consideration to the performance of the model and $\frac{1}{3}$ to the quality of the report. All teams may be asked to present their model and/or paper to the other teams, but this will not factor into the team evaluation.

The **report** does not need to be comprehensive (i.e., with background information and references), and should only contain the following sections:

1. **Architecture**. What model class did you choose, and why? Did you experiment with different architectures? How did they perform?

2. **Training**. How did you train your model? What losses did you use and why? What hyperparameters did you tune and what results did you see? How can you explain your choice of hyperparameters? How much CPU-time did you use?

3. **Results**. Did you validate or test your model? Explain your choice of validation and test set. Report and describe strategies to reduce overfitting, if present. Did you conduct any error analysis? Can you hypothesize an explanation for the errors?

# 6 Timeline

- **Wednesday, October 16, 19:30 PDT**: Project specification and data released.
- **Saturday, October 19, 23:59 PDT**: Team signups close.
- **Wednesday, October 30, 19:30 PDT**: Project work day.
- **Wednesday, November 13, 18:00 PST**: Code and report due. Project closes.
- **Wednesday, November 13, 19:30 PST**: Test data released.
- **Wednesday, November 20, 19:30 PST**: Wrap-up, announcements, presentations.

# 7 Getting Help

Please reach out to us with any questions! All administrators of the ACMLab and ACM are available to provide help and guidance. We encourage you to post general questions that may be of interest to all teams in `#mlab-projects`. You can also DM `@bjing` with specific questions about the project specification, or any administrator of ACMLab and ACM with technical questions. Further, we will try to host office hours if you would like to have in-person help outside of the project workday — stayed tuned to `#mlab-projects` for updates. Finally,

Have fun!