Stanford ACMLab Fall 2020 Project: Mapping Income Distribution with Machine Learning

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.

This project was created by **Bowen Jing** (ACMLab Co-Director '19).
Any mistakes are due to Jillian Tang and Ethan Chi (ACMLab Co-Directors '19, '20).

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 sub-national 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 would be 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 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 to 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, the problem statement is:

> Build a model that, given a *satellite imagery tile*,
> outputs the *predicted average yearly income* for people residing in that tile.

You will train and test your model on satellite imagery from the **Los Angeles Metropolitan Area**. We will then evaluate your model's ability to generalize the income distribution on unreleased test sets drawn from Southern California and 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 Onboarding Contest Structure

Please form teams of up to **4** people.[1] Don't worry if you don't have a team in mind – we're happy to help match you up into teams. Once you have formed a team or decided you want us to be match you up, please fill out the form in `#acmlab-general`, which you should do by this **Wednesday, October 14**.

[1] If you want to form a team of 5, please talk to an administrator for approval.

3.1 Tracks

There will be two tracks: the **Framework** track and the **Open** track. Participants in the **Framework** track will receive a partial codebase to start off their progress (less than the notebooks we've done in workshops, though), while participants in the **Open** track will be creating all of their infrastructure on their own. We recommend the Framework track if you want to solidify your knowledge of deep learning and would like some guidance through the project, while we recommend the Open track if you feel comfortable with what we've gone over so far and would like a challenge. At any point, you are welcome to switch from Open to Framework, but you cannot switch in the opposite direction.

3.2 Prizes

We are still working on determining final prizes, but tentatively:

- Everyone who submits as part of a project team will receive an ACM-branded **Patagonia jacket** and/or other swag item. They will also be invited to join the **Research Team** for winter/spring's research submissions.
- The members of the first-place team in the Framework track and several teams in the Open track will each additionally receive a **gift card**.

4 Data

All datasets necessary to complete this project have been posted in the `#acmlab-general` channel 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; please notify `@jiltang` and `@echi` beforehand if you wish to do so. However, you **cannot** use pretrained weights (e.g. pretrained ResNet). If you want to pretrain on other data, you will have to do so on your own.

4.1 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 Figure 1.

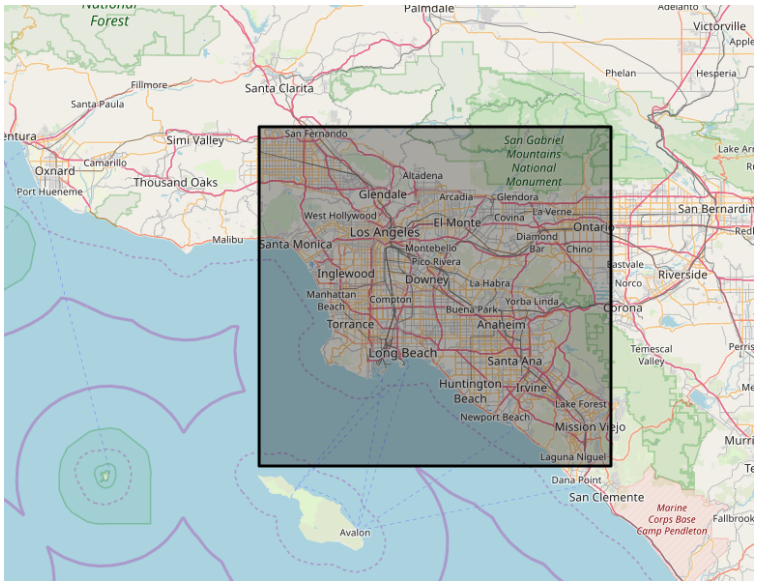

Figure 1: Region from which training data is provided

More precisely, *satellite web map tiles* at zoom level $Z = 14$ are provided for the region. Web map tiles are 256×256 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.

Each tile is 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. Transformations from (latitude, longitude) to (x, y) are provided in `webmercator.py` for your convenience.[2]

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`.

To load a numpy array from an image, this may help:

```
from PIL import Image
filename = "images/something.jpg"
image = Image.open(filename).convert("RGB")
```

4.2 Census data

To compute the labels for each tile, you will use `16zpallnoagi.csv`, a `.csv` file containing tax returns data for the entire United States in 2016. There are many columns 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.

We suggest applying methods we have gone over in workshops to keep only the data points that you need.

To map satellite images to zip codes, you will use `ziplatlon.csv`, which contains a mapping from zip codes (`zip` column) to coordinates (`latitude` and `longitude`) fields. Of course, there are more zip codes than there are tiles; therefore, most zip codes contain multiple tiles. To work out the correspondence from tiles to zip codes (and thus to income values), please perform the following algorithm. 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

Of course, you do not need to consider zip codes whose geometric centers are not inside any of the supplied tiles. Each tile will be weighed **equally** in the final evaluation.

Finally, please note that your model will not be evaluated on **oceanic tiles**, defined as those with elevation 0. Elevation data is provided in **worldelev.zip**; functions to access this data are provided in `util.py`.

[2] Specifically:

$$x = \frac{2^Z}{360}(\text{longitude} + 180^\circ)$$
$$y = 2^Z \left(\pi - \log\tan\left(\frac{\pi}{4} + \frac{\text{latitude}}{2}\right)\right)/(2\pi)$$

5 Implementation

You must implement your model in **PyTorch** using the methods we have described in the workshops. Please use PyTorch 1.0.1 or later. More implementation hints may be found in the Appendix.

6 Evaluation

At the conclusion of the project, you will submit:

- Your **code** as a path to a GitHub repository. 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.
- Your final **model**. Specifically, please provide a `predict` function in your code which accepts a single **string parameter** which is a path to a `.jpg` image 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.
- A written **report** describing your model. 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?
  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?

Please submit these on Gradescope by **6:00PM PST on Wednesday, November 4** (we will send out a class enrollment link later).

The final evaluation criterion will be the **average absolute error** between the prediction and ground truth (as calculated by us) over all the tiles.

6.1 Awards

A **winning team**, or multiple winning teams, will be selected and recognized after the project concludes. The winner(s) will be selected with $\frac{2}{3}$ consideration to the performance of the model and $\frac{1}{3}$ to the quality of the report. Additionally, all teams will be asked to briefly present their model and/or paper to the other teams.

7 Timeline

- **Monday, October 12, 18:00 PDT**: Project specification and data released.
- **Wednesday, October 14, 23:59 PDT**: Team signups close.
- **Sunday, November 1, 18:00 PDT**: Project work day.
- **Wednesday, November 4, 18:00 PST**: Code, model, and report due. Project closes.
- **Sunday, November 8, 18:00 PST**: Presentations and announcement of winners. Test dataset released.

8 Hints

- The size of the dataset is not very large, so if your convolutional network is too large, you may suffer from **overfitting**. To avoid this issue, you should consider performing **data augmentation**, where each of your dataset items are used multiple times (Figure 2). You can do this through the transforms listed in this page, which your dataset could apply with some random probability. This should significantly improve the performance of your model.
- If you're working on Colab, loading the data from Google Drive can be extremely slow. To avoid this issue, when you initialize your dataset, you should convert your images to tensors and store them in a list. Otherwise, training may be extremely slow.

Figure 2: Data Augmentation.

- When you start training, make sure you're **working on CUDA**! Otherwise, training may be up to 50x slower than you expect! Consult Workshop 3 for more details (make sure you call `.to(device='cuda')` wherever you have an input tensor).

9 Getting Help

Please reach out to us with any questions! The entire ACMLab leadership team is available to provide help and guidance. We encourage you to post general questions that may be of interest to all teams in `#acmlab-general`. You can also DM `@jiltang` and `@echi` with specific questions about the project specification, or any administrator of ACMLab with technical questions. Further, we will be doing weekly office hours if you would like to have in-person help outside of the project workday — stay tuned to `#acmlab-general` for updates. Finally,

Have fun!

Acknowledgements

Bowen Jing came up with the idea of the project, designed the problem specificiation, and scraped the satellite tiles. Ethan built the framework used as part of the Framework Track. All three of the authors are responsible for the specification in its current form.

Appendix

9.1 Loading Data

We strongly suggest the use of a **dataset** and **dataloader** to easily load and batch your data. These are commonly used in PyTorch to load data in an easy and convenient way, and will significantly save you time when you're processing data.

In workshop 3, we demonstrated the use of a dataset to load MNIST data. However, this was a prebuilt dataset that was pre-loaded with MNIST data, which obviously won't work in our case. Instead, you'll need to **subclass** `torch.utils.data.Dataset`, as follows:

```
class ImageDataset(torch.utils.data.Dataset):
    def __init__(self, ...arguments...):
        # initialize your data from file and store it in memory

    def __len__(self):
        return None # this should return the number of elements in your dataset

    def __getitem__(self, idx):
        return image, label # this should return a tuple of (image, label)
                            # where `image` is the image as a tensor
                            # and `label` is the ground truth label as a float
```

You can then test your dataset as follows in a Colab/Jupyter notebook:

```
import numpy as np
from PIL import Image

i = 5                                       # we can view the 5th image
image, label = dset[i]                      # image shape: [3, 256, 256]
image = np.moveaxis(image.numpy(), 0, -1)  # image shape: [256, 256, 3]
image = (image * 255).astype(np.uint8)      # to preview, needs to be ints
display(Image.fromarray(x).convert("RGB"))
```

Once you have your dataset, you can build train and test **dataloaders** for this dataset. Here's how we suggest doing it:

```
from torch.utils.data.sampler import SubsetRandomSampler
train_indices, val_indices = # indices of the dataset elements you want to be
                             # in the train and validation sets

train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(val_indices)

train_dataloader = DataLoader(dset, batch_size=32, sampler=train_sampler)
valid_dataloader = DataLoader(dset, batch_size=32, sampler=valid_sampler)

for image_batch, label_batch in train_dataloader:
    # ...
```

9.2 Loss

Previously, we used losses such as `BCELoss` and `CrossEntropyLoss` for classification problems. However, this is a **regression** problem, where we are trying to predict a specific value. Unfortunately, these are harder to optimize, so your choice of loss function can be very important.

Here are some losses you can try:

- `torch.nn.L1Loss`: penalizes the mean absolute error.
- `torch.nn.MSELoss`: penalizes the squared mean absolute error.
- `torch.nn.SmoothL1Loss`: smooth interpolation between the above two.

Of course, you will be evaluated on the **L1 loss** on the test set, so please keep that in mind.

Finally, when we built our models, we had some issues with what is known as *gradient explosion*, where the regression loss causes gradients to become unreasonably high. To avoid this, try:

```
loss.backward()
torch.nn.utils.clip_grad_norm(model.parameters(), 10)
optimizer.step()
```