Land-Use and Deforestation in the Brazilian Amazon

Using Satellite Imagery and Deep Learning for Environmental Conservation

Cameron Bronstein Feb 22, 2019 · 10 min read

Introduction and Problem Statement

Why satellite imagery?

Coming into the Data Science Immersive course at General Assembly last November, I was excited and assured that there would be many applications of the tools I was going to learn. I barely knew Python, had not touched dataframes, and could not tell you much about neural networks. But I was sure that something exciting was going to come from it.

I have since finished the program and, most proudly, my capstone project that used satellite images from Planet to classify scenes from the Brazilian Amazon based on

weather, land-use, and land-cover.

This project was exciting for many reasons:

- It gave me experience directly applying new technology to a problem of ecological conservation.
- This work presents an opportunity in which I could potentially study any ecological system anywhere on the planet without leaving my home. *(This is not because I'm lazy. And it's not because I do not like to travel. One of my concerns with the field of Ecology is finding ways that we can study the planet without repeatedly traveling by airplane to field sites and conferences around the year after year. This current practice feels ironic and questionable for — something I am still trying to unpack this as I move further along in my career. Maybe it's something we cannot avoid. Or maybe we can move forward with new technology to advance the field in more sustainable ways.)*
- It has opened up a pandora's box of new question, methods, and projects for future scientific exploration!

Why the Amazon?

Many ecology researchers have drawn their focus on tropical and subtropical forests, as the majority of the earth's biodiversity and biomass is concentrated in these regions. These ecosystems are at the center of Earth's ecological-climate system, affecting regional and global climate dynamics through the cycle of water, carbon dioxide, and oxygen.

And of course, these forests are home to vast resources, and are in turn under immense pressure from deforestation and land-use changes — as well as global climate change. With improved satellite technology from companies like Planet, we can find ways to monitor these precious ecosystems remotely. With machine learning models, we can track changes to these ecosystems in real time, alerting local government and conservation organizations of illegal forestry operations that threaten the biodiversity and eco-climate function of our forests.

Gathering the Data

The data for this project is from this 2017 Kaggle competition, gathered and cleaned by Planet.

However, to demonstrate generating my own images with Planet's API, I constructed scripts to filter and download images given an area of interest (AOI). This was an exciting aspect of the project as I thoroughly enjoy data pipeline engineering tasks, and exploring the functionality of new APIs.

Some challenge in generating my own images for a supervised learning classification problem are labeling and formatting. Images do not have inherent labels, so labeling could be a rote and laborious process if I were to gather enough images (potentially thousands) to train a robust model. Also, images must be a standard size to use as inputs for a neural network — writing a program to crop and clip images to a standard format could be a project on its own!

Given time constraints, I decided to move forward with the Kaggle Dataset. However, with increased time and permissions (i.e. access to data outside of Planet's free Open California imagery), Planet's API is an incredible resource for future projects and applications.

Kaggle Dataset

This was my first foray into a more serious big-data project. The dataset included a training dataset (40,000 labeled images) and test dataset — unlabeled images to be submitted and scored on competition website. Overall, this represents over 50 GB of data — far more than the RAM I have on my computer (more on that later).

The images come from Planet's PlanetScope Satellites and are 3m resolution. Included are Analytic Geotiff and Visual JPEG formats — cropped to 256 x 256 pixels. The images contain four color bands (Red, Green, Blue, Near Infrared) while the JPEGs are corrected into a 3 band visual product for EDA.

Exploratory Data Analysis

A large portion of my EDA included investigating satellite image data more broadly. This was my first deep dive into a computer vision project, so this way a considerable task for me. To read more about this, check out my other medium post on the subject of Understanding Image Data and Color Channels in Satellite Imagery.

Before modeling, it was also vital to understand the distributions of varying label classes in the dataset. The labels are separated into two categories — weather and land-cover. Images contain only one weather label, but can contain multiple land-cover labels (this is multi-label classification). The labels are organized below by category:

- **Weather labels:** Clear, Haze, Cloudy, Partly Cloudy
- **Land-use and Land-cover labels:** Primary Forest , Road , Water , Agriculture, Habitation, Bare Ground, Cultivation, Blooming, Artisanal Mine, Selective Logging, Slash and Burn, Conventional Mine, Blow Down

Below, I plot the frequency of the image labels in the dataset.

Frequency of labels in the train dataset. Images can contain multiple labels

It is important to note the severe imbalance of our labels. This is important in model training — the model will have difficulty recognizing the patterns needed to predict the labels that it has not seen with high frequency.

Image Preprocessing

There were varying hurdles to overcome due to the enormity of the dataset. To prepare the images for modeling, images must be read in from the directory to be processed by the neural network. One way to do this through a library called rasterio. Rasterio is a great library as it preserves the geolocation data in the tiffs, in case you are using that information in your project.

I wrote a custom function that achieves this, however, I lacked the RAM needed to achieve this on a single machine and I had to try other tools.

Moving to the Cloud with AWS

Once I grasped the challenges of working on my laptop, I changed gears and moved the project to the cloud using a Deep Learning GPU from Amazon Web Services. This gave me new experience using big data tools offered in the cloud, and required maneuvering of Elastic Block Storage volumes to accommodate for my large dataset.

However, I still faced memory limitations using the AWS servers, so I had to use Image Pre-Processing tools native to KERAS to prepare my dataset for modeling.

Test-Time Augmentation

Neural networks "learn" by recognizing patterns in image features. For each image, a feature is a pixel. However, if a certain weather or land type only appears in a certain region of an image, the neural network will not recognize the exact same feature if it is in a different location in the image.

I was introduced to **ImageDataGenerator** class in Keras Image Preprocessing. This allowed me to apply test-time augmentation (random flipping, rotating, zooming of training images) for a better modeling approach.

Example TTA technique

The Image Data Generator pulls images from their directory in batches, reads in the images, applies TTA, and runs them through the network. This greatly reduces memory load and allowed me to train the neural network and make predictions on over 50 GB of images with just 7.5 GB RAM on my remote server. GPUs are incredible deep learning tools — so incredible that I made a meme about them!

Modeling

I built a modeling pipeline that could accommodate for multiple models and cross validation. I had plans to train a deeper neural network and had functions set up to streamline this process. Unfortunately, bugs in the code prevented me from training the neural network on multiple cross validation folds (see Limitations and Future Steps section below) by the time I had to wrap up my project.

I used a modest-sized convolutional neural network with resized training images — downsized to 3-band, 128 x 128 pixels due to memory and time limitations. Network architecture was as follows:

- Batch Normalization on input
- 32 filter convolutional layer + Max Pooling Layer + Dropout (0.5)

- 64 filter convolution layer + Max Pooling Layers + Dropout (0.5)
- Flatten output — 128 Node Densely Connected Layer + Dropout (0.5)
- 17 Node output layer (representing 17 target label classes).

I trained my model over 25 epochs (first, 20 epochs → weights re-initialized → 5 more epochs).

Results

I tracked Accuracy and F-Beta Score (they follow similar trends), but focused on F-Beta score. F-beta is an average of Precision and Recall (Sensitivity) that includes a weighted penalty on false positives and false negatives, hence its lower scores compared to accuracy. It can be a more stringent classification metric compared to accuracy, especially for multi-label classification.

Final F-Beta score on the validation data was **0.843**. Predictions on the test dataset were submitted to Kaggle and had a similar F-Beta (0.834), showing that my model generalizes well across a broad range of unseen data. The model is slightly under-fit, which could be attributable to overly aggressive TTA or excessive regularization in the network (dropout of 0.5 at multiple layers), or due to over-prediction of the labels on unseen data.

The over-prediction results in relative instability of the F-beta scores on the validation set. For the validation data (of which we could track predictions vs true labels), the model predicted more than 43,000 labels for 10,000 images, of which there were only 28,000 true labels across the data. This would likely contribute to the instability of F-

Beta Scores and much higher accuracy scores. For example, a predictions could have an accuracy of 1 if it predicts all 17 labels for every image (as each correct label is accounted for). But, the F-Beta score would be much lower as there would be many false positives for all the labels that are not actually in the image.

Comparison of True and Predicted Label frequencies after model validation.

Model predictions were also greatly impacted by the imbalanced classes. Model predictions rarely included labels that had under 1,000 initial observations in the training data (see above). While these images likely were of low representation in the test dataset, it is still important that our model could distinguish and predict these features for real-time application of the model.

Future Steps and Limitations

- **Limit number of label predictions per image or per dataset:** To correct the issue of over-predicting, I could limit the total number of possible labels when converting model outputs (probabilities) to better reflect the possible number of labels. What would this look like? First, only consider the top ***n*** probabilities. Of those top probabilities, only accept those above the threshold value.

- **Test F-beta score on predictions using multiple thresholds:** The model outputs 17 *probabilities* representing the *likelihood* that a certain label is in each image. We then set a threshold value and accept only the probabilities above the threshold. These are converted to 1, meaning that feature is in the image. I used one threshold value for all of the labels. However, one way to improve model results could be to more aggressively find optimum thresholds for individual categories. Successfully

implementing k-fold cross validation and training k models would allow for ***k*** searches for optimum thresholds. We could then pick the average thresholds across each fold.

- **Bootstrapping to Balance Label Classes:** I would also bootstrap (create "replicate" samples of infrequent labels) by target images with low sampled labels, augmenting those images and adding the augmented copies back into the dataset. This would likely improve training regardless of initial frequency of labels in the dataset (more data, different images due to augmentation).

- **Invest in powerful computing resources:** The main limitation in my project was RAM. While having access to a remote GPU was vital to reduce training times, I ran into considerable hurdles throughout the project with overloading the RAM (despite the measures I took to reduce batch size). These memory challenges, along with out-of-pocket cost of using cloud-based servers, also made it difficult to experiment with deeper neural networks (I was especially excited about the potential of using ResNet 50 with "ImageNet" weights).

- **Experiment with Model Ensembles:** I learned through research in Kaggle Discussions of the value of ensembling for neural networks. On a basic level, multiple CNNs could be trained and would likely make better predictions on certain labels in the dataset. The predictions from each neural network could then be combined, and, for each output (possible label), a linear regression model would be used to predict the "true value" of the label for each image. The different CNN predictions would be features to predict the target value (either 1 or 0, presence or absence). These new values would still be subject to the same thresholds used to finally land upon presence or absence of a label in each image.

- **Consider Model Effectiveness and Resource Use:** The most effective modeling technique possible would, essentially, require unlimited resources. While a complex, multi-model ensembling approach might yield the best predicting power, stakeholders must consider the cost (money and time) required to construct such an immense modeling system, as well as the reduced speed of classification if the models are to be used in real time. This is an ongoing, case by case trade-off, but one that must be considered in the context of deep learning.

Thanks for reading! Feel free to check out my full project repository on my GitHub. And please share your ideas on how I could improve my approach to this project — I'd love to

hear your thoughts!

-Cameron