ResNet Based Tree Cover Classification

Jack Fan, Leif Rasmussen, Lucas Zhang, and Romi Asad

UNC Chapel Hill

December 8, 2021

Abstract

Forest outline data is currently one of the most time consuming and expensive types of geographical data to create. It's very useful for a wide variety of applications, which is why the City of Raleigh hires people to manually trace all forests in the city every four years.[1] Work like this could be automated if an accurate enough image classification model was available.

We trained a model that could predict whether the center of an 80*80 meter square image was forest or cleared land. We created training data by merging OpenStreetMap forest outlines with public imagery, and fed it into an implementation of ResNet-50 using Keras and Tensorflow. Our model achieved an accuracy of 94.88 percent on a manually validated test data set. We further showed the performance of this model by running it at every location in an image, and showed the output as an overlay on the image. This allowed us to clearly see how the model classifies data, including how it handles edge cases.

1 Introduction

Classifying landcover is much more difficult than it first appears. The approach we used involves determining whether a given point is wooded using the surrounding area. For a given location, we look at the 80*80 meter image centered at that point, and feed the image into our classifier. The classifier should understand that the center of the image needs to be classified, and that the surrounding area is just for context. The challenge with this approach is that the few pixels in the center are not enough to classify the image, so the model must rely on the entire image to classify just the center. Instead of checking for trees all around, the model must look at the shape that the trees and other features carve out to determine of the center is wooded. The complexity of this classification task is precisely why this problem is interesting to apply machine learning towards. We believe that a deep neural network can learn the fine intricacies needed to greatly reduce the effort and cost involved in creating tree cover data.

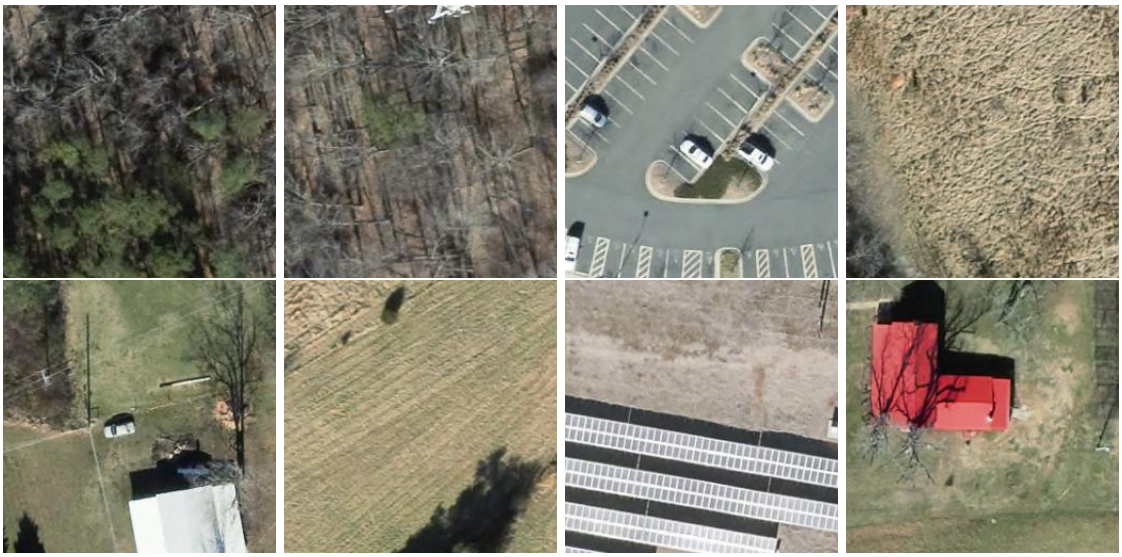

Figure 1: Easy Classifications

Figure 2: Hard Classifications, which make up a significant portion of the real world

2 Materials & Methods

2.1 Creating the Dataset

We had to be assemble data from two existing datasets to create training data for this project. The first source was NCOneMap, which has publicly available leaf off imagery of the entire state, and the second source was hand traced forest outlines from OpenStreetMap. Creating the training data consisted of cleaning up the forest outlines, creating a grid of points over the forest outlines, labeling all points inside of one of the forest outlines, downloading an image at each point, and sorting the images into two folders.

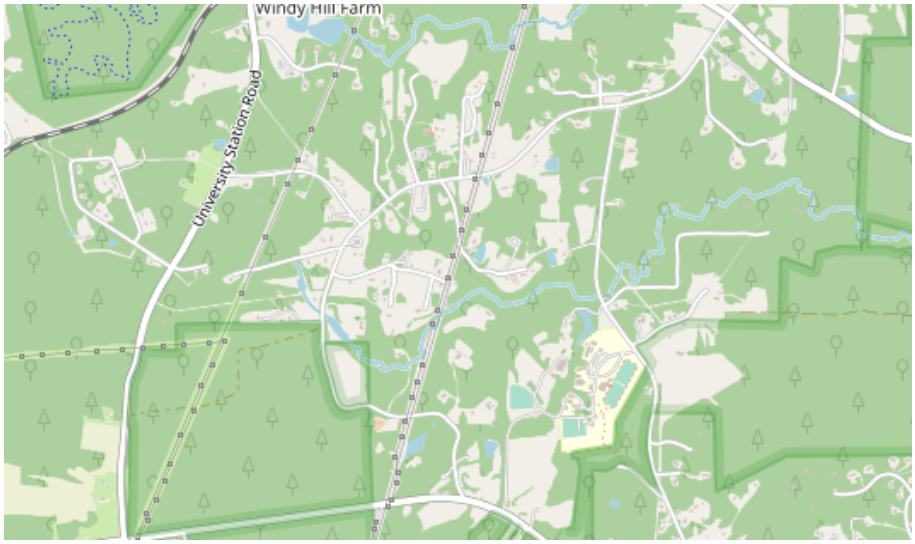

Figure 3: Source data used for labeling images from NCOneMap [2].

Our final dataset contained 107345 not-woods images and 144711 woods images. The ratio between training and validation was 0.8:0.2 respectively. The test dataset was manually procured of size 2911 and 6789 of not-woods and woods images respectively.

2.2 Backbone and Architecture

The main architecture we used to perform our classification task was the ResNet50 backbone by He et. al. [3]. Residual networks are classically known to perform well on classification tasks, addressing the gradient vanishing/explosion problem through additive skip connections. We created ResNet50 backbones using TensorFlow's Keras library. We also tested implementations for ResNet18 and VGG19, but the results from ResNet50 were significantly better, so we will focus solely on ResNet50 results in this paper.

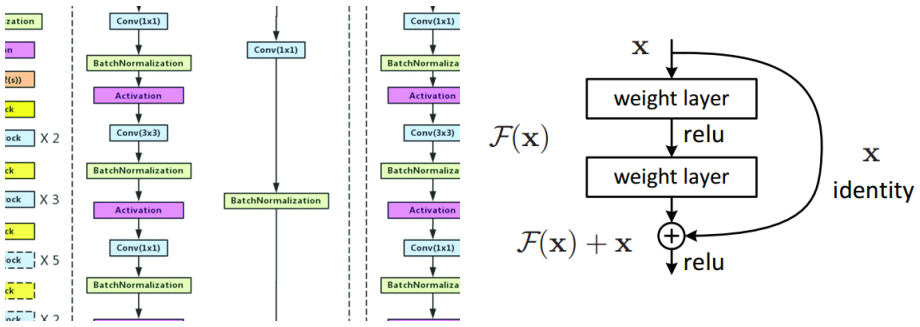

Figure 4: ResNet-50 architecture and Residual Skip Connection

2.3 Training

We trained the model on our dataset over 15 epochs using a Tesla V100-PCIE-32GB GPU. Each epoch took about 8 minutes to train, therefore each model was trained in about two hours.

We decreased the learning rate by a factor of 0.1, monitoring the validation loss with a patience of two epochs. This method improved our results significantly, as our training became more stable, and we were able to converge to the minimum more quickly and reliably.

We kept constant our backbone and dense layer regularizers(l2= 0.0001) for each model. We tested the performance with different layer optimizers (ADAM vs SGD), kernel initializers (Glorot Uniform vs Glorot Normal), and pooling layers (max vs avg). In total, we trained 8 models.

Starting with Glorot Uniform and ADAM, our initial results were very poor, around 80 percent accuracy. We are unsure of the exact effect Kernel Initializers have on performance, but Glorot Normal performed significantly better,about 2 percent increase, for all combinations of model parameters. After adding dynamic learning rate tuning, the accuracy of the models increased all around, bringing all models to similar accuracy. Previously, the difference in performance between SGD and ADAM were drastic, with ADAM optimizer models producing an accuracy around 80 percent compared to 93 percent with SGD. After adding dynamic learning rate scaling, the all versions of the models settled around 94 percent, and models with the ADAM optimizer performed better than models with SGD. This was surprising because classically SGD is preferred for deep neural networks. Given our low number of training epochs, however, its possible that the benefits of momentum in SGD have not been realized in our training process. Average pooling also seems to perform better than max pooling in our application. Initally, we thought that max pooling may be a better fit given the contrast and noise that are common in aerial photos; the bright and stark contrasts may be beneficial information for the model. However, after seeing the increases of performance with average pooling, albeit small, perhaps reducing noise revealed the useful information in the image.

3 Results

Our best performing model, achieving a test accuracy of 94.88 percent, was the ResNet model using Glorot Normal, ADAM, and Average, as its kernel intializer, dense layer optimizer, and pooling layer respectively.

Kernel Initializer	Dense Layer Optimizer	Pooling	Accuracy
Glorot Normal	SGD	Avg	94.6%
Glorot Uniform	SGD	Avg	94.36%
Glorot Normal	SGD	Max	94.2%
Glorot Uniform	SGD	Max	94.14%
Glorot Normal	**ADAM**	**Avg**	**94.88%**
Glorot Uniform	ADAM	Avg	94.86%
Glorot Normal	ADAM	Max	94.81%
Glorot Uniform	ADAM	Max	94.59%

Table 1: Accuracy of the multiple trained models

All models performed relatively well, having less than 1% deviation in accuracy, but the aforementioned model produced the best results. The results from running the model on images centered at each grid square are shown below. The accuracy for obvious cases appears to be well above 99%, with most of 5% of misclassified images appearing to be along the edges of forests, where classifications are less clear cut and expected classification is often arbitrary.

Figure 5: Forest classification in Northeast Raleigh

4 Conclusions

Our model performs surprisingly well, as can be seen in the grid classification example above. It generates outputs that have the same quality that a human would produce. For the purposes of generating landcover data, our model is more than good enough. From looking at how the model performs on the grid example above, it's clear that the model learned that only the center of the image determines the classification of wooded vs cleared, and that the rest should only be used for context. Given an image containing mostly trees with a clearing in the center, the model appears to correctly classify the image as not forested. Given the opposite, however, with a few trees at the center, and none anywhere else, the model seems to fail to classify the center as trees (see the edge of the river in Figure 3). This is likely because our source data, which contains outlines of all forests, didn't contain outlines for small patches of just a couple of trees, so the model learned that those don't count. When using the model to create

landcover data, however, this issue is actually an advantage, since only real forests are included, with insignificant patches of trees ignored.

Next steps include improving performance (high resolution is only need along the edge of forests, not everywhere), finding a way of vectorizing the output (creating polygons around each forest) so it will be easier to work with, and creating a tool that can generate data from start to finish without requiring manual labor.

Acknowledgements

We would like to thank Jorge Silva and Kathryn Kirchoff as the instructors of this course. This project also could not have been done without the generosity of Lenovo Labs, Beijing.

References

[1] City of Raleigh. Similar dataset to our model's output, hand traced every four years. https://data-ral.opendata.arcgis.com/datasets/ral::vegetation-outlines/about.

[2] NC Geographic Information Coordinating Council. Nc onemap. data retrieved with the NC One Map API, https://nconemap.gov.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Microsoft Research*, 2015.