TreeNet: Deep U-Net for Image Segmentation

Andrew Kondrich, Isaac Kasevich[*]
Stanford Computer Science
557 Mayfield Avenue
andrewk1@stanford.edu, isaack97@stanford.edu

Abstract

TreeNet is an image segmentation network that classifies trees in satellite images on the pixel level. Built on top of a U-Net, proposed by Ronneberger et.al in 2014 [1], it uses a unique recurrent-style architecture to enable precise localization of elements. The U-Net was initially proposed as a method to segment biomedical data, particularly cells in microscope imagery. Because satellite imagery contains a fundamentally different distribution of features in comparison to biomedical data, we tested the capabaility of the U-Net architecture to generalize to other types of data. The original paper was also an experiment in using heavy elastic deformation augmentation techniques on a limited labeled data set. Since we also only have 25 labelled images, we did further experimentation by comparing the efficacy of using different data processing methods to understand how they have an impact on accuracy. Our final results showed that using the data processing methods proposed in Ronneberger et.al were outperformed by simple image slicing and preservation of RGB data, with final test Dice coefficient scores of 0.4612 and 0.6312 respectively.

1 Introduction

While a tree classification model has applications in fields like the lumber industry and forestry research, our original motivation was to identify challenges inherent to models for semantic segmentation, or the pixel-by-pixel classification of images, in the scope of satellite imagery. Since semantic segmentation is a key problem in computer vision, it is important to experiment with current models to understand their capacities. In our project, we use the U-net CNN model because it was designed to perform in a space where there exists a small number of labelled imagery. The input to the model is a satellite image, and the output is a predicted black and white mask of the same dimensions where white signifies the presence of trees in that region. For the inputs, we experimented with using 1-channel grey-scaled deformed images of the entire region, and 3-channel RGB non-deformed images sliced into smaller sections.

2 Related work

The state-of-the-art in image segmentation is DeepLab V3, which implements a ResNet model using dilated/atrous convolutions [3]. For our project, we chose to implement the u-net architecture proposed for biomedical image segmentation [1]. We chose to use to use u-net over DeepLab V3 due to it's familiar implementation using concepts learned in class, and because it uses comparably less parameters and trains faster than DCNN models like DeepLab and other pixel classifiers such as the

[*]GitHub: https://github.com/andrewk1/TreeNet

one proposed by Ciresan et.al [2]. This is favorable for us given our general lack of computing power, time to try different implementations and parameter values, and labeled data. We experimented with different data types and loss functions that have been shown to be effective in image segmentation tasks. Particularly, the original u-net paper uses grey-scaled images as is the general format for biomedical imagery. As shown by Geirhos et.al, object recognition CNNs perform significantly worse for grayscale images compared to coloured images, demonstrating a 4.81% drop in performance on average [4]. Since we have access to RGB images, we changed this aspect of the original u-net model. Another aspect of our task that differs from the original u-net paper is that there is a relatively large class imbalance, with the majority of images having sparse tree coverage. Segmentation tasks in this space often benefited from using a Dice coefficient-based loss function, as shown by Sudre et.al [5] and Zhang et.al [6].

3 Dataset and Features

We utilize a dataset from a Kaggle.com deep learning competition that spanned a 9-month period. The original Kaggle contest involved image segmentation of 10 different object types from 1km x 1km satellite images, where outputs are GeoJSON arrays denoting the coordinate outline of target objects. In this project, we perform the single critical operation in this Kaggle competition – generating a black and white pixel mask over the input image.

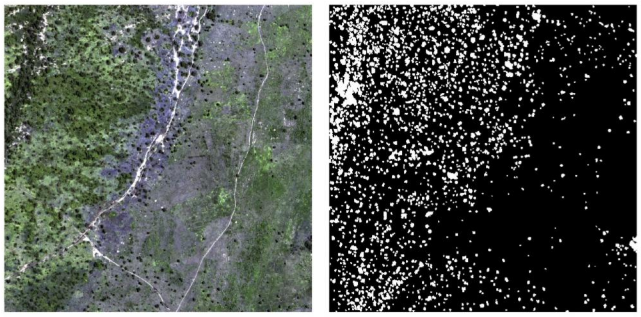
Our dataset consists of 25 3600x3600 pixel images, each taken by satellite of a 1km x 1km region on Earth. We used these images as raw inputs that needed to be rigorously pre-processed and augmented to generate a dataset large enough to train a deep neural network. As a target, we transformed the Geo-JSON coordinates of our outputs into inverted pixel masks, where each mask was a 3600 x 3600 pixel image with black pixels corresponding to non-tree pixels and white pixels corresponding to tree pixels. We then jointly augmented our raw input/pixel mask output images as the input/output pairs in the neural network.

We tried two different tactics for data augmentation and train/dev/test set splits, with varying degrees of success:

Tactic 1: Deformation and Flipping using 1-channel grey-scale images
To establish baseline performance for our model we performed basic data augmentation techniques on the full 3600 x 3600 images. Specifically, we used 1-channel grey-scaled images, applied color distortion and performed image flipping to expand our initial dataset of 25 images to 643 images. We did not perform color shift augmentation on the pixel masks, but did perform identical flipping to that performed on each input image. We split this dataset into 510 training images, 128 validation images, and 5 test images. Before running, we downsampled each image to 480x480 pixels so as to keep or model's training time reasonable.

Tactic 2: Image Slicing
After achieving moderate success with our baseline model data augmentation tactic, we decided to try image slicing. In this tactic, each input was sliced into 40 sub-images, each 160 x 160 pixels in dimension (scaled from 180 x 180). In accord, we sliced each corresponding pixel-mask ground-truth image into 40 sub-images as well. This produced a full dataset of 10,000 images, which we proceeded to split into 7840/1960/200 image train/dev/test sets. In this tactic, we kept the original three-channeled RGB input but did not perform any color augmentation.

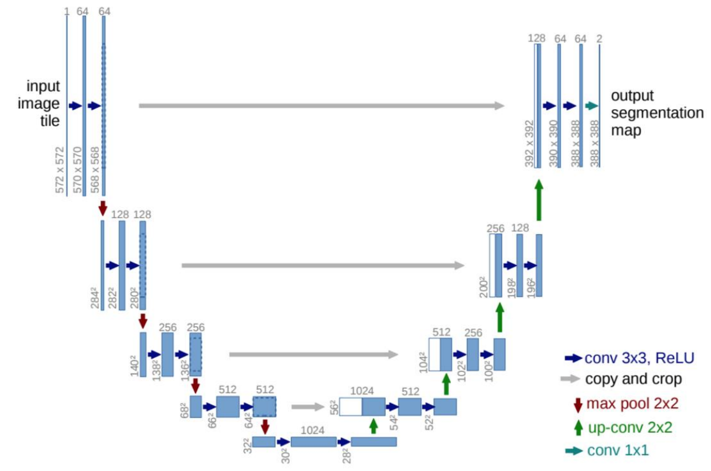
Below is an example raw image from our dataset, along with its corresponding pixel mask:



We sourced our dataset from Kaggle: https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection/data

4 Methods

We initially considered using a Polygon-RNN as suggested by Castrejon et. al. [7] to map directly from input images to output GeoJSON objects (essentially directly predicting the bounding boxes for each tree) but ultimately decided against this approach due to the complexity of the images. We ultimately decided to use a recently developed deep learning model architecture called 'UNet', initially proposed by Ronenberger et. al. [5]. A figure showing the model architecture is shown below:



UNet is an end-to-end encoder-decoder deep convolutional neural network architecture with similarities to recurrent neural networks. It contains two key units. The first, a 'contracting layer', downsamples an input image to one much smaller, applying successive 3x3 same-padded convolutions with ReLU activation before a 2x2 maxpooling layer that halves the image dimension. The second, an 'expanding layer', also applies successive 3x3 same-padded convolutions with ReLU activation, but then applies a 2x2 upsampling convolution, which uses fractional striding to double image dimensions. These units are stacked together in a U shape, with an added skip-connection from each contracting layer to the expanding layer with the same input image shape. In the output layer, we apply a 1x1 convolution. The model does not include any fully-connected layers.

This architecture allows the model to couple low-level, high-resolution features from early layers with more abstract, contextual features from later layers. At the bottom of the 'U', the model has essentially learned an extremely compact image encoding, which it subsequently decodes using the help of skip connections from prior layers.

We compare the output segmented image to the true value image directly using the negative of the dice coefficient, which flattens the resulting output image mask into a vector a, and flattens the ground-truth image mask into a vector b before computing dice loss. Intuitively, dice loss is negative of the amount of space in which the two masks overlap divided by the size of the predicted mask, and is computed as

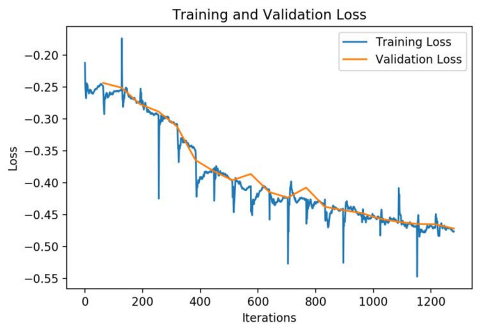
$$d_s = -\frac{2|a \cdot b|}{|a|^2 + |b|^2}$$

While this means the loss values are negative, it still functions as a valid gradient because we are maximizing the overlapping area and minimizing the size of the predicted mask. We initially experimented using simple binary cross-entropy loss, but switched after the model achieved very low loss but outputted nonsensical image masks. Our model saw drastic improvements upon switching to dice loss (analogous to F1 score in natural language processing), which weights precision equally with recall.

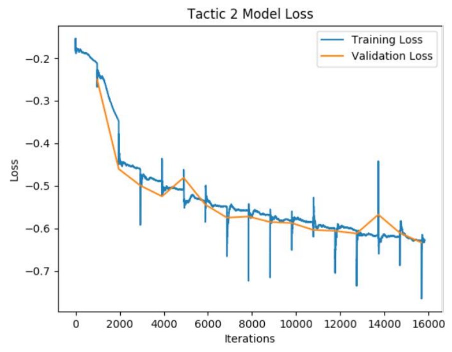
5 Experiments/Results/Discussion

Our primary metric for testing the accuracy was the same as the loss, Dice coefficient (described in above section). This is a common metric used for understanding the accuracy of image segmentation networks [8]. We used a mini-batch size of 8 and $\alpha = .00001$ for both tactics. In the original u-net paper, it was recommended to choose a batch size of a single image. We chose 8 images per batch because it was a good balance between matching this recommendation, and taking some advantage of vectorization to improve speeds. We attempted batch sizes of 16 and 32 as well, but these sometimes caused our AWS instance to run out of memory. We chose a lower learning rate because optimizing runtime was not a priority as it was already taking an acceptable amount of time (≈ 10 hrs), which we could run overnight.

Because image segmentation uses a convolutional layer as the output, it is difficult to use any metric based on classification like precision or recall. In current research, the most commonly used metrics are Dice score and Jaccard similarity [cite]. For our experiment using Tactic 1, the Dice score on 5 held-out images from the original dataset was 0.4612. The value is around equal to our final training and validation accuracies (0.4765 and 0.4722 respectively), so the model did not overfit the training data, as shown in Figure (a) below.

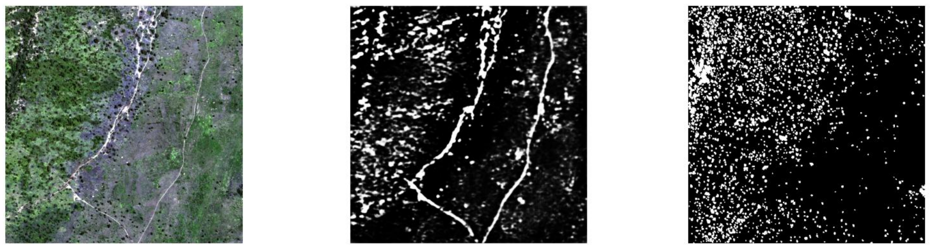


(a) Training/Validation Loss with Tactic 1



(b) Training/Validation Loss with Tactic 2

In Figure 2, we see an example classification with Tactic 1. While it was capable of detecting some general notion of where trees are in the image (grouped towards to the left), the exact classifications are noisy and inaccurate. There is another common misclassification: Roads were all labelled as trees. We believe that some of the issues stem from grey-scaling images: While this choice was capable of preserving features for biomedical data, colors are evidently important features when segmenting satellite imagery. Another noticeable difference is that the predicted masked sections are generally large clusters. This likely resulted from a loss of detail when scaling the entire images to input into the network.



(a) Image (Grey-scaled) (b) Tactic 1 Prediction (c) Ground Truth

Figure 2: Tactic 1 Test Example

(a) Image (b) Tactic 2 Prediction (c) Ground Truth

Figure 3: Tactic 2 Test Example

In Figure 3, we see an example classification with Tactic 2. Because our data was sliced, the prediction tasks were on higher-resolution close-ups of the images, which allowed the network to pick up on distinct shapes of trees. This, along with using 3-channel inputs, allowed for a higher Dice test score of .6312, and faster drop in loss as seen in (b) above.

6 Conclusion/Future Work

In conclusion, we saw that using a modified u-net structure with sliced images and no deformations (Tactic 2) works better for the task of identifying trees in satellite imagery in comparison to the data processing pipeline proposed in Ronnenberger et.al. Due to the scattered nature of trees in the images, Tactic 2 worked better because we could more effectively create a larger set of high-resolution training data, where each image had multiple trees that could be used for training. This is fundamentally different from the environment in which u-net was developed, which had only a handful identifiable objects per class in an image. Furthermore, the inverse of the Dice Coefficient proved to be an effective loss function, and was an interesting experiment in using an unconventional loss function to see relative success.

With more time and more powerful TPU/GPU-powered machines, our team would have experimented with combining deformation with slicing to create a much larger dataset to try training on. We would also try implementing a more complicated model like the ResNet model used in DeepLab V3, which would require more time to understand the general architecture. The data was provided in an esoteric format (GeoJSON and GeoTiff) that we weren't familiar with working in. Now that we are familiar with working in these formats, we can iterate through ideas and implementations much quicker.

7 Contributions

We divided the work evenly in this project, though Andrew focused primarily on data processing and image preparation in the latter stages while Isaac focused on the presentional poster. We pair-programmed the initial model, and split work evenly on the final report. The process was generally collaborative, with us figuring ideas out along the way.

8 References

1. Ronnenberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation.
2. Ciresan, D., Giusti, A., Gambardella, L., Schmidhuber, J.: Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images
3. Chen, L., Papandreou, G., Schroff, F., Adam, H.: Rethinking Atrous Convolution for Semantic Image Segmentation
4. Geirhos, R., Janssen, D., Schutt, H., Rauber, J., Bethge, M., Wichmann, F.: Comparing deep neural networks against humans: object recognition when the signal gets weaker
5. Sudre, C., Li, W., Vercauteren, T., Ourselin, S., Cardoso, M.: Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations
6. Zhang, J., Shen, X., Zhuo, T., Zhou, H.: Brain Tumor Segmentation Based on Refined Fully Convolutional Neural Networks with A Hierarchical Dice Loss
7. Castrejon, L., Kundu, K., Urtasun, R., Fidler, S.: Annotating Object Instances with a Polygon-RNN
8. Taha, Abdel Aziz, and Allan Hanbury. "Metrics for Evaluating 3D Medical Image Segmentation: Analysis, Selection, and Tool." BMC Medical Imaging 15 (2015): 29. PMC. Web. 11 June 2018.
9. Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2018-06-11].
10. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)
11. François Chollet, Keras, GitHub repository, https://github.com/keras-team/keras
12. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org
13. zhixuhao, Implementation of deep learning framework – Unet, using Keras, GitHub repository, https://github.com/zhixuhao/unet
14. jocicmarko, Deep Learning Tutorial for Kaggle Ultrasound Nerve Segmentation competition, using Keras, GitHub repository, https://github.com/jocicmarko/ultrasound-nerve-segmentation
15. Dstl Satellite Imagery Feature Detection, Dataset, https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection