Data Sci 450 Capstone Project

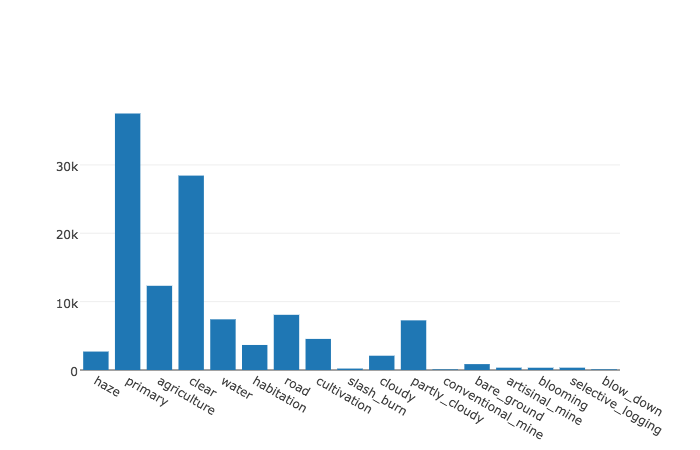
Kaggle Competition: Understanding the Amazon From Space

By: Sam Robinson (Team Megaseed)

**Understanding the Data:**

Understanding the data took longer than I would have liked at the end of the day but that was to be expected having not worked with images before. The first step in understanding the data was reading the kaggle documentation, I went back to the documentation many times to get a better understanding of what I was looking at.

Tags: Here is a histogram of the tags. You can see that some tags have very limited numbers compared to the rest of the data. I felt the need to add quite a few features because of seeing some tags so sparsely populated. That is partially where the concept of using edge detection came from.



Chips: The chip images are in TIFF form. I had not used TIFF images before and took quite a bit of time visualizing them, slicing them up and looking at the underlying data structure. They are great pieces of visual data and I wish I had the chance to work with TIFFs that had each light band represented like there are on some images. With more light bands available there are a platitude of interesting image manipulating techniques you can use to highlight different features like water, buildings, vegetation etc. These TIFF images only had near infrared light (NIR) which gives us some basic highlighting of vegetation and water indexing. (NDVI and NDWI) I found these images fascinating and was eager to do more normalized indexing. However, without more color bands most of the more developed techniques are not usable. No matter! I still was interested in this data. Here are some examples of chips with interesting features on them.



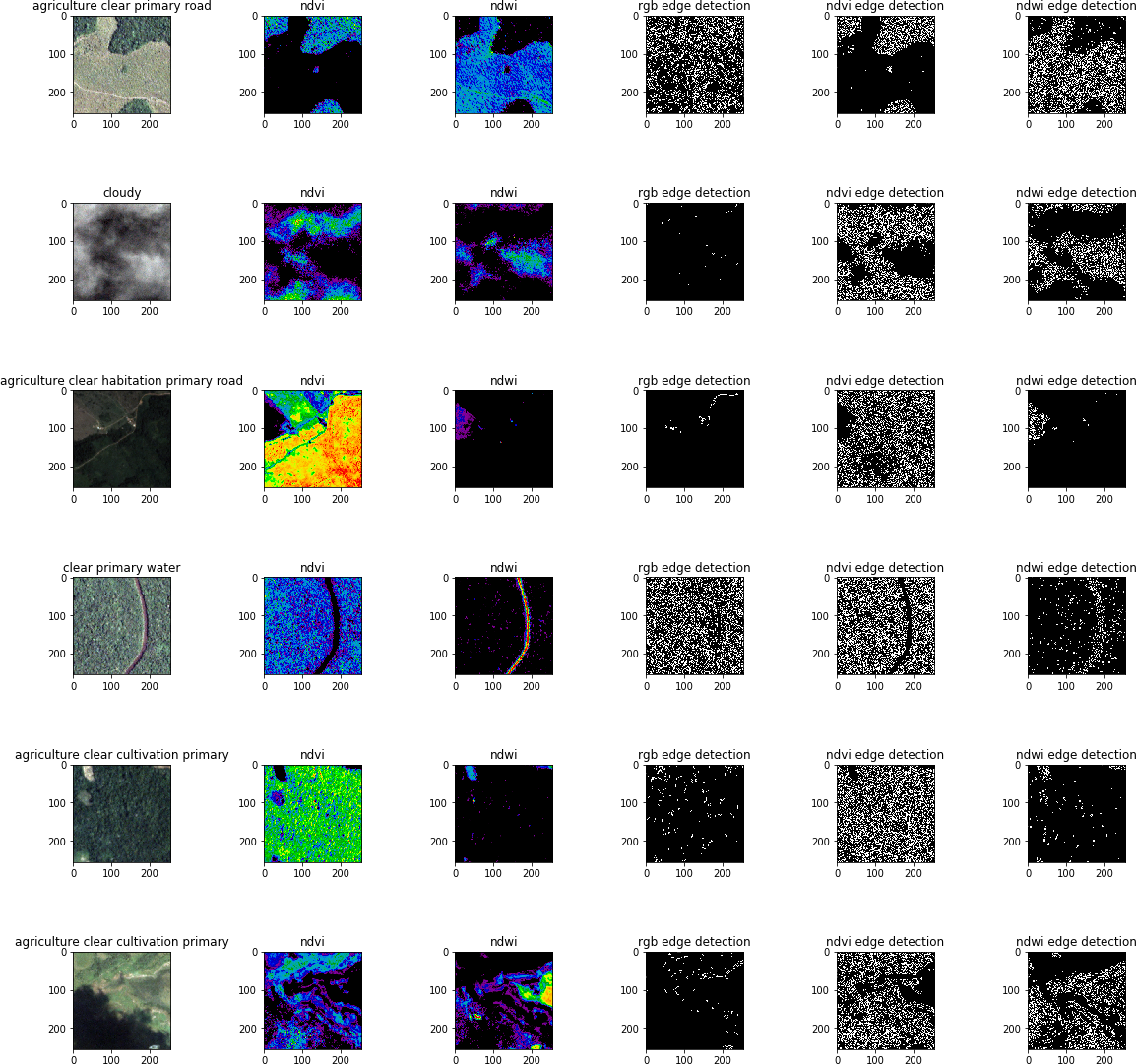
**Feature Engineering**

The first features I tried to create was detecting edges in RGB images. I got interesting results in feature heavy chips and very messy results in plain chips. This felt like a decent feature when all was said and done because there was some fairly clear distinctions in the edge detection between some of the different chips with different tags on them even though the edge detection wasn’t outlining the specific features all the time it still felt usable.

After learning about TIFF images I knew that using NDVI and NDWI was going to be a part of the features I engineered. I had seen some examples of people using these images as overlays on top of the RGB version of the image but I felt that the information was interesting enough to warrant it’s own image. In addition to creating those indexes I also used spectral color mapping to really make the important features popup out of the rest of the terrain.

The final feature I created was edge detection on the NDVI and NDWI images. I felt that highlighting a lot of the features that were popping out of the normalized images was a great way to add further weight to those pieces of information. When all was said and done I ended up with 6 images that I knew I was going to feed into my neural network. Here is an example of them below.

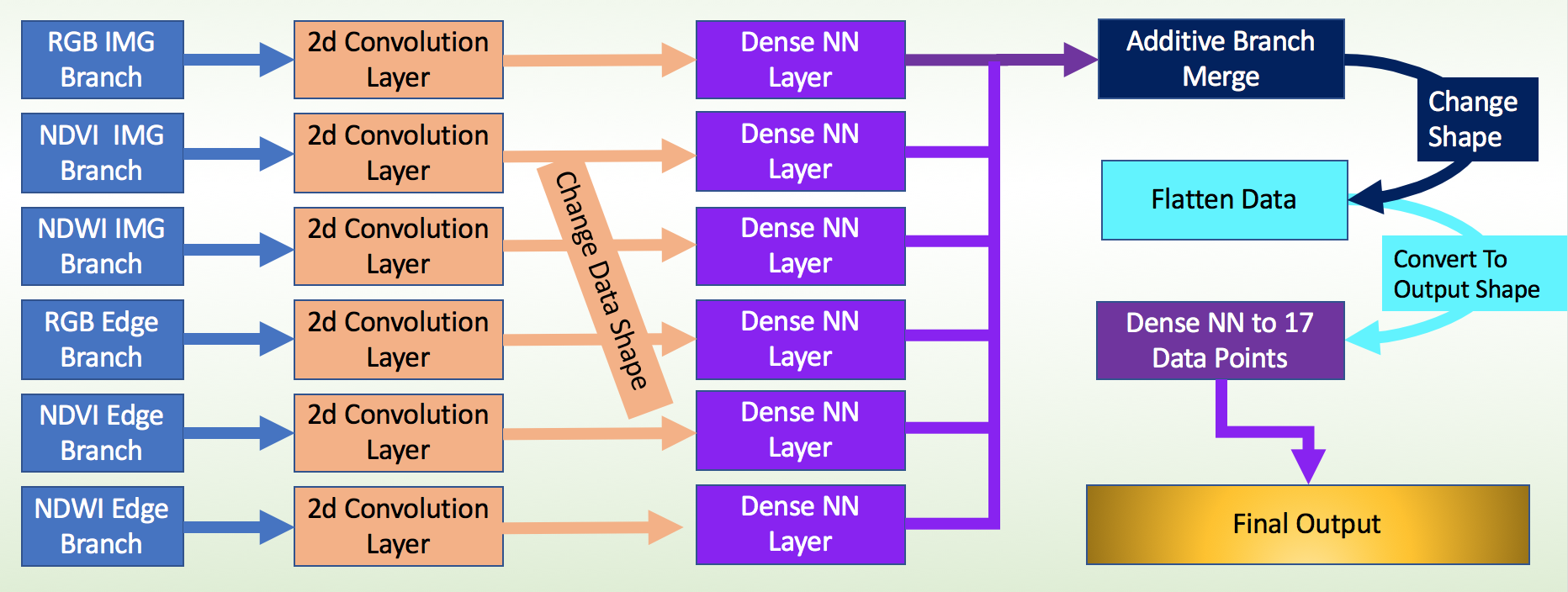
I want to note that I want to take the 3 types of edge detection and assign a specific color to each one. Red for the RGB edges, Green for the NDVI edges and Blue for the NDWI edges. This would allow me to easily represent all 3 types of edges on a single image. Due to time constraints that version of these images won’t be available for the project but I will move forward with that method as I continue to work on the kaggle competition. By reducing these features I can speed up my neural network. I can’t simply turn the edges into arrays of binary information because to merge my neural network layers together they need to have the same shape. This gives me the same shape and the distinct data for each of the different kind of edges represented in the data.



Example of the features generated for the model.

**Tensor Flow Model**:

The first thing I did was review models that other people had already started. This simple Keras Starter (<https://www.kaggle.com/anokas/simple-keras-starter)> was straightforward to understand and I could repeat the output in the notebook. Once I had a basic understanding of the format I went to figure out how to add all of my features to the model. This came in the form of merging layers once there has been some processing done. The merge functionality was interesting to me and right now I’m only exploring a single merge with additive functionality. Given more time I would try an array of different merge types and orders in an effort to improve the model. I would also try more activations on my branches to try and improve the model as well.



**Model Evaluation**

With only a few epoch being able to run in my time constraints my model is not looking very predictive at all! I am using fbeta-score from scikit-learn to evaluate the model. Fbeta-score <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html>. This was used in previous examples and seems like a good indicator of accuracy given the many binary values for tags being used for evaluation.

The real issue with the current model is a prohibitively long run time. 4000+ seconds per epoch means that it needs to run for several days or weeks before I can start getting something interesting out of it. This can be greatly improved on by reducing the 3 image features down to a single image but it will still take a bit of time to run. I probably could use fewer convolution layers for each branch and add an additional 1 after the merge as well. Learning the tensor flow visualization tools and how they can interact with Keras would also help me see how my model is performing without waiting for a long process to finish.

As of turning in this assignment my fbeta-score is only at 0.657506315582 after running 3 epochs. It’s tough to do a real evaluation of how well the model *could* perform under these conditions.

**Whats next**

1. Improve the features I’ve engineered including combining the edge images into a single image and possibly changing values on the edge detection to get less noisy edges.
2. Verify model speeds are improved.
3. Test out different activations for various branches of the model to see if accuracy is improved.
4. Run the model for more epochs
5. Test out different ways of merging the branches together for accuracy performance and possibly for speed performance.
6. Submit my information to the kaggle competition if I can get a more performative model.