Predicting Amazon Deforestation with Satellite Images

Eric Xu
Stanford University
ericxu0@stanford.edu

Orien Zeng
Stanford University
ozeng@stanford.edu

Abstract

In this paper, we investigate the impact of using neural networks to analyze satellite images of deforestation in the Amazon. Specifically, we explore various forms of convolutional neural networks, such as the SqueezeNet and VGGNet, in order to identify atmospheric and land labels of the images. We also introduce the Split Classifier model, which first generates an intermediate output of the image and then passes it into separate classifiers that look for atmospheric and land label clues separately. However, after a thorough analysis of the models and saliency maps the neural nets produce, we conclude that atmospheric and land labels may have some correlation, so learning them separately can be suboptimal.

1. Introduction

The forests on Earth are rapidly shrinking due to human urbanization. In fact, the world loses almost fifty football fields of forest area every minute. Deforestation in the Amazon Basin has been particularly noticeable, leading to reduced biodiversity, climate change, and habitat loss. However, satellites have taken thousands of pictures of the forests, and analyzing them could provide significant insight into the causes and effects of deforestation. Consequently, algorithms interpreting satellite images of these locations are necessary to help groups respond to deforestation quickly and effectively.

2. Problem

Planet, a group that provides images of the Earth through satellites, has provided more than 40000 images of the forest scenes from the Amazon basin (which includes Brazil, Peru, Uruguay, Colombia, Venezuela, Guyana, Bolivia, and Ecuador) on Kaggle. The 256×256 images are in Tiff format and contain four channels of data: red, green, blue, and near infrared. There are also a set of JPEG images for reference, which are smaller in storage size and thus may provide less information than the Tiff images. Unfortunately, several sources on Kaggle have reported that the Tiff images do not correctly correspond to the truth labels. Consequently, the models in this paper all use the JPEG images for training, validation, and testing.

The JPEG images are similarly 256×256 in size, and contain three channels: red, green, and blue. Each of the training images is labeled with a subset of seventeen different labels. Those seventeen labels are organized into three groups: atmospheric conditions, common land cover/land use phenomena, and rare land cover/land use phenomena. The atmospheric condition labels are cloudy, partly cloudy, haze, and clear. The common land labels are primary, water, road, agriculture, cultivation, bare ground, and habitation. The rare land labels are artisinal mine, blooming, blow down, conventional mine, selective logging, and slash burn. An image will have at least one atmospheric label and zero or more common/rare labels. Cloudy images should have no other labels. In the labeled dataset given, it is important to note that none of the images have more than one atmospheric label (although one image is incorrectly labeled with no atmospheric label).

Indeed, the labeling of images is very costly and time consuming, so there are labeling errors. However, the signal to noise ratio is still high enough to be sufficient for training. For the final test evaluation, the model predicts the labels of over 60000 JPEG images in a test set. Of the test set, the average score for half of the set is reported, while the average score of the other half is hidden. We use a validation set to choose the best-performing set of weights while training, and use those weights to predict on the test set. The score of the visible test set is recorded, and from the visible test set we pick the highest-scoring models to predict on the entire test data.

The model is evaluated by F_2 score so that recall (ratio of true positives to all actual positives) is weighted higher than precision (ratio of true positives to all predicted positives). The final F_2 score is formed by averaging the individual F_2 scores of each label.

3. Related Work

Satellite image classification is not a new field. Wilkinson, G.G. in *Results and implications of a study of fifteen years of satellite image classification experiments* reviews satellite classification tasks since 1989. Among them, neural network approaches were popular. Benediktsson, J. et. al in *Neural network approaches versus statistical methods in classification of multisource remote sensing data.* compared a three-layer net to statistical approaches such as maximum likelihood for Gaussian data. They revealed similar tradeoffs to using neural networks that we see today, in that the three-layer net could achieve higher accuracy but risk overfitting on non-representative training examples. In 2007, Gamanya et. al in *An automated satellite image classification design using object-oriented segmentation algorithms: A move towards standardization* used spectral and textural feature extraction to perform automated image segmentation, with accuracies around 90%. Other models include a Markov model in *Classification of Multisource Remote Sensing Imagery Using a Genetic Algorithm and Markov Random Fields* by Tso et. al and knowledge-based methods, such as *Implementation of temporal relationships in knowledge based classification of satellite images* by Middelkoop and Janssen. More recently, Jean et. al in *Combining satellite imagery and machine learning to predict poverty* describes an approach that utilizes satellite images of African countries to predict the regions' poverty levels. The model presented relied on transfer learning, in which they took a pretrained convolutional neural net that learned examples from ImageNet and fine-tuned it for this new problem. This modified convolutional neural net then predicts mean cluster-level values from the images as well as features. From these values and image features, they trained ridge regression models that estimated cluster-level expenditures or assets.

Moreover, the Kaggle challenge deals with multi-label classification. Therefore, ordinary softmax loss is not sufficient to make a prediction on the classes. Platt et. al in *Large Margin DAGs for Multiclass Classification* offers a solution by modeling the problem with a binary classifier for each pair of classes and their DAGSVM classifier to combine the binary decisions into a multi-class classifier. Other methods include winner-take-all and max-wins (Zhang and Lenan, *Classification of Fruits Using Computer Vision and a Multiclass Support Vector Machine*). However, these multiclass methods assume the classes are mutually exclusive. In this rainforest labeling task, the labels are neither mutually exclusive nor independent.

For multi-label classification, Tsoumakas et. al in *Multi-label classification: An overview* suggests for transforming small multi-label tasks into multi-class task, where each class represents a subset of the label set. Label set size is a problem; Trohidis et. al avoids this in *Multi-Label Classification of Music into Emotions* by clustering labels into combined labels, e.g. amazed-surprised and happy-pleased. Vens et. al addresses interdependencies between classes in *Decision trees for hierarchical multi-label classification* for labels that can be organized into hierarchical trees or directed acyclic graphs. Zhang and Zhou use a generalized k-nearest neighbor approach in *A k-nearest neighbor based algorithm for multi-label classification* and *A lazy learning approach to multi-label learning* in which nearest neighbors' labels are combined using a maximum a posteriori method to produce the given instance's labels. Zhang and Zhao also experiment with other loss functions in *Multi-Label Neural Networks with Applications to Functional Genomics and Text Categorization*, proposing the following two loss functions.

The first loss function was l_2-loss. In this case, correct labels are given a value of $+1$, and incorrect labels are given a value of -1. Then the loss function, for N examples, becomes

$$E = \frac{1}{2N} \sum_{i=1}^{N} \sum_{l \in Y} (c_l^i - d_l^i)^2$$

where Y is the set of all labels, c^i is the predicted output vector, and d^i is the true label vector.

The second one is given by

$$E = \sum_{i=1}^{N} \frac{1}{|Y_i||\overline{Y}_i|} \sum_{(j,k) \in Y_i \times \overline{Y}_i} \exp(-(c_j^i - c_k^i))$$

where Y_i is the set of correct labels, and $\overline{Y}_i$ is the set of incorrect labels for the ith example. Intuitively, this loss function attempts to stretch the difference between predictions of the correct labels and predictions of the incorrect labels. Thus, they believed that this one performs better than l_2 loss in practice.

In contrast, Huang et. al in *Multi-task deep neural network for multi-label learning* proposes the Multitask Deep Neural Net (MT-DNN) approach, which simply has a binary classifier for each label in the output layer of the network, where the loss is calculated by the sum of cross-entropy losses for each label.

4. Approaches

We used multiple approaches to classify the images. Our initial models were simple convolution neural nets, but they eventually evolved into more complex models that utilized properties of the data.

4.1. Simple Baseline

Our very first model was a vanilla Convolutional Neural Net. We used the JPEG images, so the inputs were of size $256 \times 256 \times 3$. We had two convolutional layers with

thirty-two 3×3 filters, a padding of one, and stride of one. Each convolutional layer was followed by a ReLU layer, batchnorm layer, and a 2×2 pooling layer (no padding) of stride two. As a result, after the convolutions and pooling, we ended up with a tensor of size $64 \times 64 \times 32$.

Following the second pool layer were three dense layers, transforming a flattened input of $64 \times 64 \times 32 = 131072$ units to 1024 units to 256 units and finally to an output layer of 17 units. The first two dense layers had ReLU activations. With the final output vector of size 17, we calculated the loss function with sigmoid cross entropy.

The model was trained on 10 epochs with a batch size of 64 and a learning rate of 2×10^{-5}. To generate the predictions of our model, we used a constant threshold of 0.5 after the sigmoid layer, which translated to a constant threshold of 0 for the output vector before the sigmoid activation. In particular, if the ith element of the output vector was greater than 0, we outputted the ith label in our predictions.

4.2. Complex Baseline

Our second model was very similar to the first, but contained more layers and filters. Inspired by a starter model on Kaggle (https://www.kaggle.com/anokas/simple-keras-starter), our second model took in inputs of size $256 \times 256 \times 3$ (JPEG images) and applied a layer of batch normalization. Next, the following layers were applied several times.

Let f be the number filters in this current block of layers. The model used f filters of 3×3 convolutions with stride of one and padding of one. Following that, it applied another f 3×3 filters but this time with no padding. Both of these convolutions had ReLU activations. After, we applied a max pooling of size 2×2 with stride of two. Finally, we added a dropout with a drop probability of 0.25.

The above block of layers was applied four times in all, with $f = [32, 64, 128, 256]$. The result was a tensor of size $14 \times 14 \times 256$. After flattening this output, we ran it through a dense layer with 512 units with a ReLU activation function. Then we applied another batchnorm layer and a dropout layer with drop probability of 0.5. Finally, we used a dense layer of 17 units to generate the final output.

As before, sigmoid cross entropy loss was used to generate the loss values. In addition, the constant threshold of 0 was again used on the output layer before the sigmoid activation to generate the predictions.

The model was trained on 40 epochs all with batch size 64. The first twenty epochs used a learning rate of 10^{-3}, the next ten used a learning rate of 10^{-4}, and the final ten used a learning rate of 10^{-5}.

4.3. SqueezeNet

Based on memory issues we encountered in larger nets, we decided to try squeezenet in order to maintain efficiency. The initial modifications only included changing the input size to $256 \times 256 \times 3$ and the number of output classes to 17. The original network had fairly low memory usage so we were able to make additions to the original model. We created the initial model using the fire module code in Assignment 3.

The first was that we noticed the network often classified multiple atmospheric labels, e.g. cloudy and partly_cloudy in the same image. This event occured in about every 10-20 images, which would certainly impair the class accuracy. Guessing that this was due to the single dense layer in SqueezeNet, we added intermediate dense layers to allow the network to better learn relationships between the atmospheric labels and between the cloudy label and other labels (a cloudy image does not have ground labels).

Our second optimization was to simply add more depth: we increased the model depth to have 35 fire modules. Based on insights from ResNet, we added residual layers to the network (by concatenating the previous layer to the output of the fire module) to aid gradient flow and ease of training, and used batch normalization after each fire module. Continuing with the precedent in the SqueezeNet paper, we let e11p = e33p = 4 * sp, and sp gradually increased while the image size decreased due to max_pooling: the final fire module uses parameters of sp = 250, e11p = 1000, and e33p = 1000.

4.4. VGGNet

We also tried the VGGNet as a potential classifier. As usual, the input was a JPEG image with dimensions $256 \times 256 \times 3$. The output was then fed into two convolutional layers with 64 filters of size 3×3 with stride of one and padding of one. Then we applied max pooling of size 2×2 of stride 2. After this layer, we added two more convolutional layers of size 3×3, but this time with 128 filters, and one max pooling layer. This was followed by two convolutional layers of size 3×3 with 256 filters and one more max pooling layer.

Next, we added the following layers twice. We added three convolutional layers of size 3×3 with 512 filters, a dropout layer with drop probability of 0.3, and one max pooling layer of size 2×2.

The convolutions did not change the height and width of the input but the pooling layers halved them, so in the end, we are left with a tensor of size $8 \times 8 \times 512$. We then flattened the output and ran it through a dense layer of size 4096 with ReLU activation. The result was passed through a dropout layer with drop probability of 0.3 and another dense layer with 17 units to produce the final prediction vector.

Again, sigmoid cross entropy loss was used to generate the loss values. However, the loss function this time was weighted with a factor of 5 so that recall would be emphasized more over precision. This would help maximize the

F_2 score. In addition, the constant threshold of 0 was again used on the output layer before the sigmoid activation to generate the predictions.

The model was trained on 10 epochs all with batch size 64. The first seven epochs used a learning rate of 2×10^{-5} and the last three used a learning rate of 10^{-6}.

4.5. Split Classifier

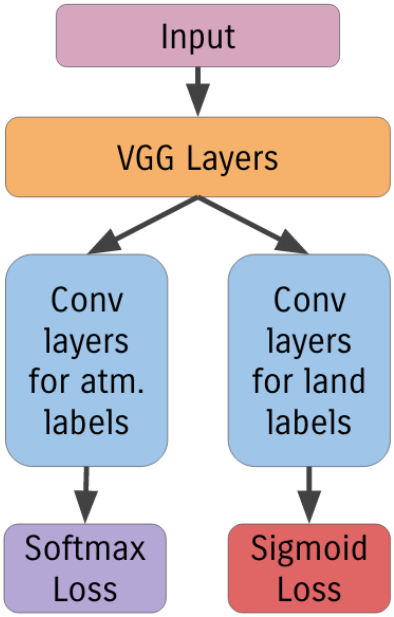


Figure 1. Split Classifier Model

While implementing classic convolutional neural net architectures produced high accuracies, our next step was to build a model that used more information about the data itself. In particular, the labels were split between atmospheric and land descriptions, so we thought that training separate classifiers for atmospheric and land would improve the F_2 score. Furthermore, we knew that each image had at least one atmospheric label. Yet, running a script through the labels, we found that all but one of the images had exactly one atmospheric label (that one image had no labels so that was most likely a labeling error). Consequently, we assumed that images should have exactly one atmospheric label.

Like all of the previous models, the input was still a JPEG image of size $256 \times 256 \times 3$. We then passed this input through several VGGNet-like layers. Let f be the number of filters in a block of layers. First, we applied two convolutional layers of size 3×3 with f filters. Then we used a max pooling layer of size 2×2. This block of layers was implemented three times, with $f = [64, 128, 256]$.

At this point, the model produced an output of size $32 \times 32 \times 256$. Using this tensor as the base input, the model then split into two separate classifiers. Both classifiers ran the tensor through three convolutional layers of size 3×3 with 512 filters, a dropout layer with drop probability 0.3, and a max pooling layer of size 2×2. This gave us two tensors of size $16 \times 16 \times 512$. Next, we flattened the output and applied a dense layer with 1024 units and a ReLU activation, another dropout layer with drop probability of 0.3, and a final dense layer. The first classifier, which predicted atmospheric labels, produced an output vector of size 4, while the other classifier produced an output vector of size 13. These two tensors were then concatenated to produce the final prediction.

Since we believed images to have exactly one atmospheric label, we applied softmax cross entropy on the first four elements (referring to atmospheric predictions). However, sigmoid cross entropy loss (with a weight of 5) was used for the remaining elements. As before, the constant threshold of 0 was again used on the output layer before the sigmoid activation to generate the predictions for land labels, and we predicted the atmospheric label with the largest value.

The model was trained on 10 epochs all with batch size 32. The first seven epochs used a learning rate of 2×10^{-5} and the last three used a learning rate of 10^{-6}.

5. Results and Analysis

Table 1. Results on the dataset using various approaches

Model	Train F_2	Dev F_2	Visible Test F_2
Simple Baseline	0.969	0.827	0.827
Complex Baseline	0.938	0.878	0.876
SqueezeNet	0.928	0.890	0.887
VGG	0.944	0.904	0.903
Split Classifier	0.939	0.885	0.885

(Again, note that Visible Test F_2 is the average of only half of the test set.)

5.1. Simple Baseline

The simple baseline produced a training F_2 score of 0.969, a validation F_2 score of 0.827, and a test F_2 score of 0.827. As a result, the model clearly overfit, and we confirmed this by plotting the training loss versus the validation loss. While the training loss curve decreased, the validation loss curve began to creep up after the 5th epoch. To combat overfitting, we decided to add significant dropout to the next model.

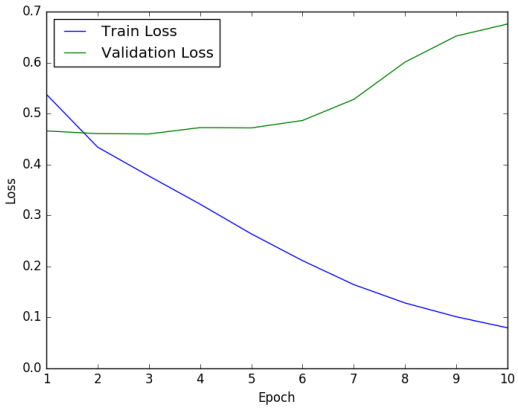


Figure 2. Loss for Simple Baseline

5.2. Complex Baseline

The complex baseline produced a training F_2 score of 0.938, a validation F_2 score of 0.878, and a test F_2 score of 0.876. This was a huge improvement over the simple baseline, suggesting that a greater number of parameters as well as dropout significantly boosted the accuracy. In addition, this new model did a better job of not overfitting, as validation loss matched up closely with training loss up to the first 20 epochs. However, afterwards, the losses diverged, so more dropout or even lower learning rates could help.

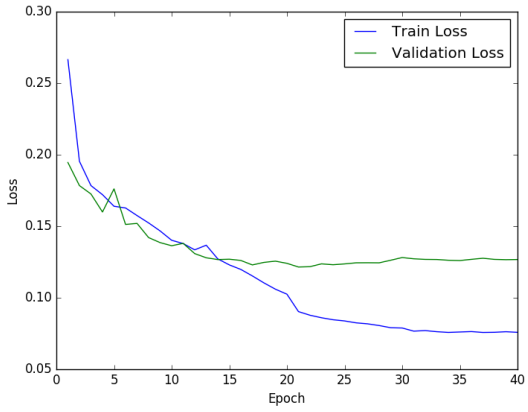


Figure 3. Loss for Complex Baseline

5.3. SqueezeNet

The SqueezeNet produced a training F_2 score of 0.928, a validation F_2 score of 0.890, and a test F_2 score of 0.887. Surprisingly, the model had the lowest train F_2, even compared to the simple baseline, while having the second highest validation F_2 score. The SqueezeNet model used no dropout, either, so the gap between train and validation is smaller than we expected. This may indicate that the model is under-trained, so we could increase the learning rate or add more epochs. At the same time, the SqueezeNet model took a long time to train, as we ran it for many more epochs than the other models. Also, for validation loss, we saw diminishing returns after epoch 10, so we might want to have fewer epochs but with higher learning rate. Towards the end, our learning rate was 10^{-7}, which may have had little impact on the model.

At first, the SqueezeNet model used a single dense layer at the end, and this model commonly predicted multiple incompatible atmospheric labels on the same image, such as cloudy and partly_cloudy. By adding more dense layers, the rate at which this happened dropped heavily, indicating that the dense layers allowed the network to recognize relationships between atmospheric labels. In addition, the model rarely predicts any other class when it predicts cloudy, which is expected behavior.

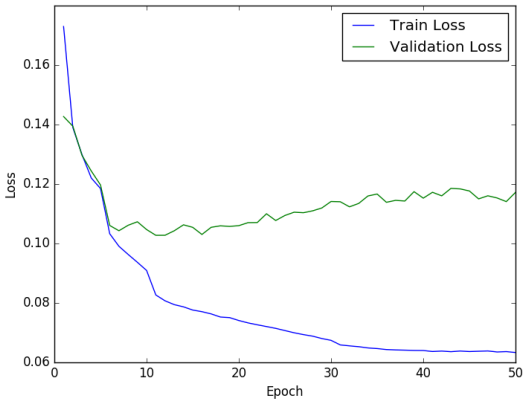


Figure 4. Loss for SqueezeNet

5.4. VGGNet

The VGGNet produced a training F_2 score of 0.944, a validation F_2 score of 0.904, and a test F_2 score of 0.903. This model obtained the best results, partially because of its large number of parameters. Since it used a lot of convolutional layers with small filters (3×3), the network was able to detect small features in the image and better predict the labels. Furthermore, the dropout helped prevent overfitting, as in the plot above we see that there is less of a gap between training and validation loss. However, after epoch 7, the gap widened, suggesting that the model began to overfit at the end. We had lowered the learning rate for those epochs from 2×10^{-5} to 10^{-6}, so decreasing it even more could be beneficial.

Looking at the predictions of this neural network, we

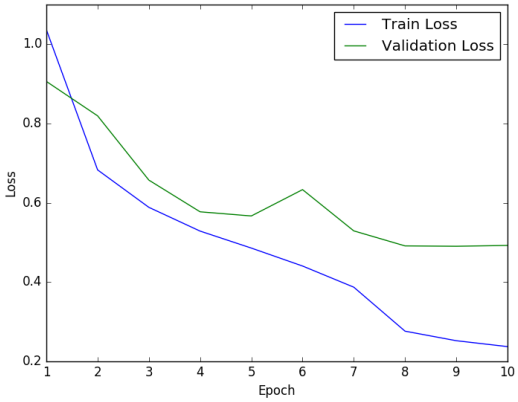


Figure 5. Loss for VGGNet

found that around 1000 images had multiple atmospheric labels. This prompted us to try the split classifier which would only predict one atmospheric label.

5.5. Split Classifier

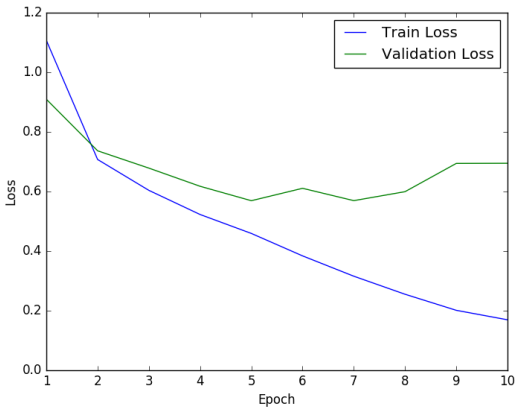


Figure 6. Loss for Split Classifier

The split classifier produced a training F_2 score of 0.939, a validation F_2 score of 0.885, and a test F_2 score of 0.885. Unfortunately, it produced lower scores than the VGGNet, despite only outputting one atmospheric label. This implied that the labels may be correlated with each other across the two categories. For example, a cloudy image has no land labels, but the model does not explicitly know that. Consequently, learning the atmospheric and land labels separately may not be ideal. With regards to the model parameters, we saw that the model began to overfit after 6 epochs, so adding more dropout and lowering the learning rate could help the model obtain higher accuracies. Additionally, we could experiment with using different learning rates for the softmax and sigmoid loss training optimizers.

6. Visualizations

We experimented with class visualizations using gradient ascent for a class score on the pixels of an image, starting from random noise. The resulting images would be labeled to have that class, but were very noisy and appeared similar to random noise.

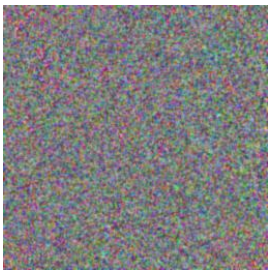


Figure 7. Class visualization for conventional_mine

Here, we see the generated image for the class conventional mine. For comparison, the clear image below is not very different.

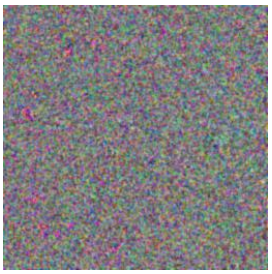


Figure 8. Class visualization for clear

When blurring, jitter and regularization are added, the image appears to deform, as seen below.

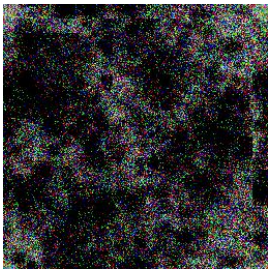
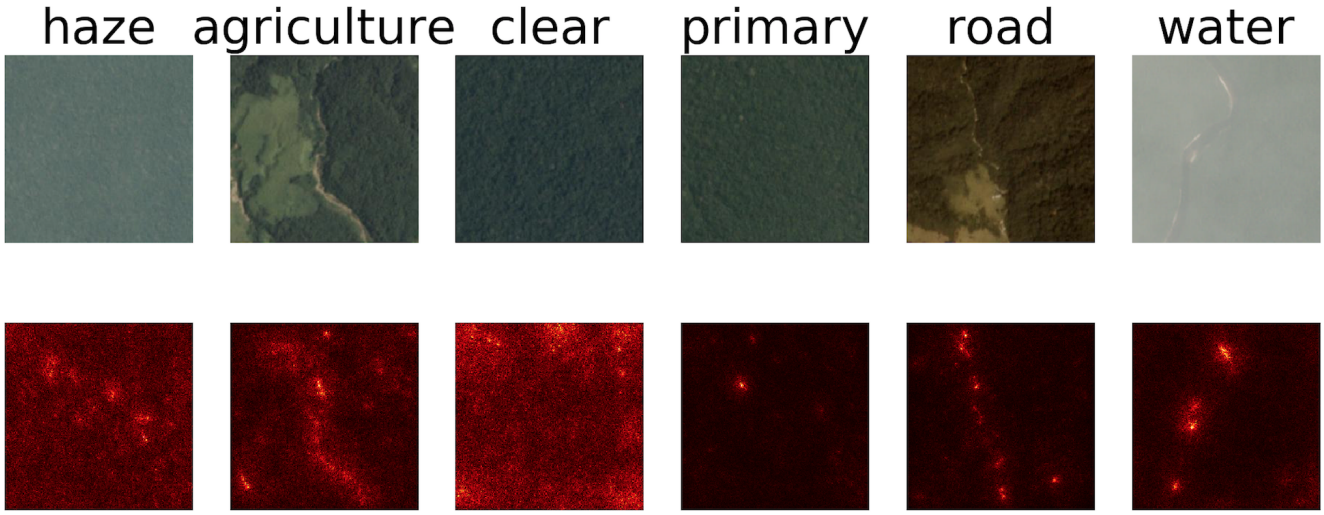


Figure 9. Class visualization for blooming with high regularization

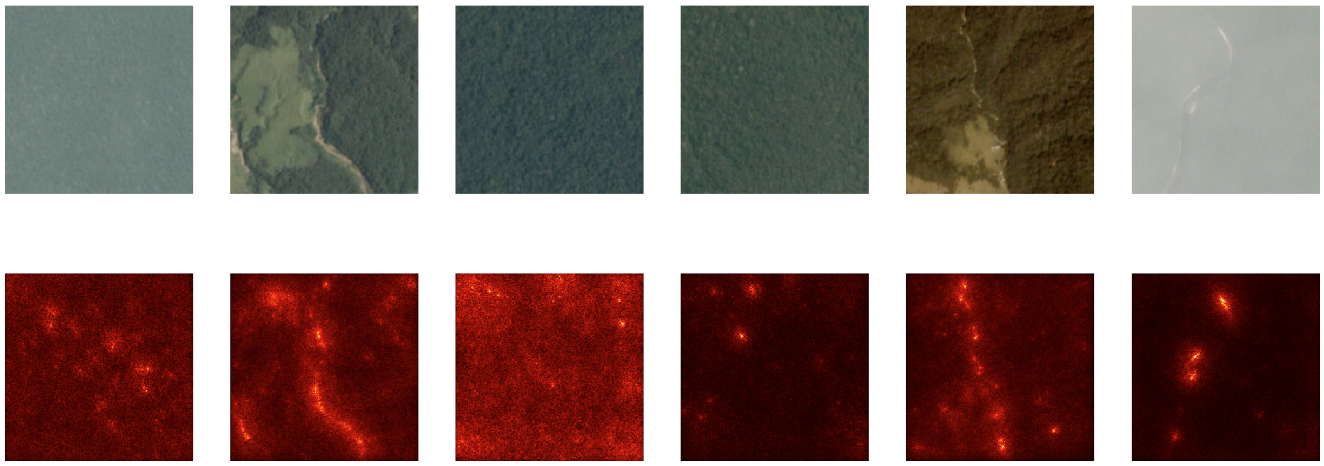
The reasons why the resulting visualizations are noisy and patchy are unknown. The gradient ascent on a single class score did not train very well; we experimented with different learning rates and learning rate decay and various levels of regularization. In the beginning, part of the issue was that the gradient was much higher on certain pixels, which would cause those pixels to go out of bounds while other pixels didn't train – in the above image, much of the black region has pixel values of 0. To address this, we stepped in the direction of the gradient, but class scores still barely increased past the threshold. One hypothesis is that the forest classes don't always form clear shapes in the same way that ImageNet does; clouds don't have any high-level features that may just appear in a visualization.

7. Saliency Maps

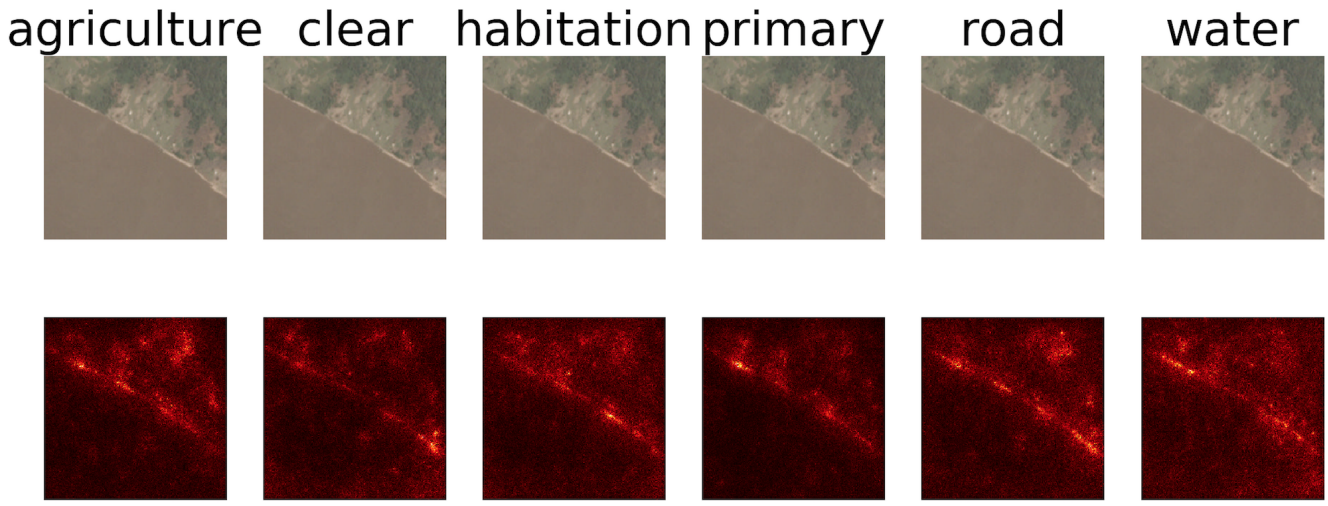
After working on visualization of target classes, we created saliency maps for a few training examples. In the figure below, we chosen one correct class to inspect for each image; most of these images have multiple correct labels, and we simply choose one to inspect. We see that haze has gradients across the entire region, along with clear, although both have hotspots of interest. These may correpond to some of the less homogenous areas of the image. For road, we clearly see the saliency map trace out the road with low noise away far from the road, which is expected.



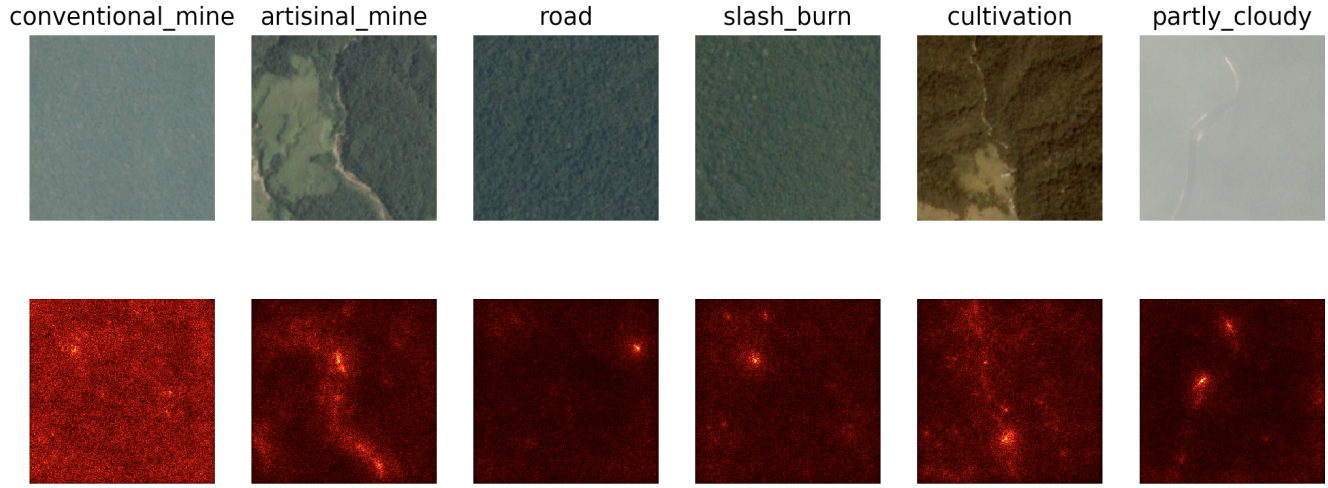
Below, we have the sum of the saliency maps for all the correct classes of the image. Note that the images are similar to the single-class saliency maps, shown below; the model appears to be highly concerned with edge detection, even for classes that don't contain large edges such as agriculture or water. This makes sense as water can be recognized in part by the shoreline.



To inspect the differences between different class visualizations further, we visualize saliency maps on all correct classes of the same image below. We see that the labels all pay attention to the shoreline across the image, but water has gradients in the water while the road class is salient in the gray area of the image. Agriculture is salient in a similar region, where there is a border between the green and gray land masses.



Finally, we visualize what the model looks for in incorrect classes for images. In the hazy image and the class conventional_mine, we generally see noise. Slash_burn and road seems to be salient in small dots in their corresponding images; maybe the slash_burn class appears to show up in the small circular regions at the top-right of the image. Also, the white area in the road may appear to be a cultivated area and the white area of the road could be interpreted as cloudier than the surrounding haze.



8. Conclusion and Future Plans

Ultimately, the VGGNet performs the best in analyzing the satellite images. Furthermore, the results from the saliency maps and F_2 score of the Split Classifier reveal that classifying atmospheric labels and land labels separately is suboptimal. This suggest that there is a correlation between the two types of labels, and it is better for the neural net to learn them by processing them together.

Nevertheless, for the future, we plan to use advanced architectures implemented in GoogLeNet and ResNet to improve our performance. Of these, pre-trained models are strong candidates for training, and we will add additional layers for fine-tuning. Using transfer learning would make the training process faster and would provide highly optimized starting points for training. Lastly, we believe that there may be underlying hierarchies in the labels, and getting a handle on those relationships can greatly improve the accuracy.

References

[1] Anokas. "Simple Keras Starter." Kaggle. ¡https://www.kaggle.com/anokas/simple-keras-starter¿. We used this publicly available code for our complex baseline classifier.

[2] Benediktsson, J. A., Swain, P. H., & Ersoy, O. K. (1990). Neural network approaches versus statistical methods in classification of multisource remote sensing data.

[3] Gamanya, R., De Maeyer, P., & De Dapper, M. (2007). An automated satellite image classification design using object-oriented segmentation algorithms: A move towards standardization. Expert Systems with Applications, 32(2), 616-624.

[4] He, Rich. "XGB Starter." Kaggle.¡https://www.kaggle.com/greenmtn/xgb-starter-lb-0-88232¿. In early networks we tested, we used a feature extraction function based off of the code here. However, due to providing minimal benefits to the scores, this function was not used in the final models.

[5] Huang, Y., Wang, W., Wang, L., & Tan, T. (2013, September). Multi-task deep neural network for multi-label learning. In Image Processing (ICIP), 2013 20th IEEE International Conference on (pp. 2897-2900). IEEE.

[6] Iandola, F., Han, S., Moskewicz, M., Ashraf, K., Dally, W., and Keutzer, K. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size". arXiv:1602.07360. ¡https://arxiv.org/abs/1602.07360¿This paper introduces the SqueezeNet architecture. Squeezenet code for creating fire modules was partly based off of Assignment 3 starter code.

[7] Jean, N., Burke, M., Xie, M., Davis, W. M., Lobell, D. B., & Ermon, S. (2016). Combining satellite imagery and machine learning to predict poverty. Science, 353(6301), 790-794.

[8] Middelkoop, H., & Janssen, L. L. (1991). Implementation of temporal relationships in knowledge based classification of satellite images. Photogrammetric engineering and remote sensing, 57(7), 937-945.

[9] Platt, J. C., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In Advances in neural information processing systems (pp. 547-553).

[10] Simonyan, K. and Zisserman, A. "Very Deep Convolutional Networks for Large-scale Image Recognition". arXiv:1409.1556v6. ¡https://arxiv.org/abs/1409.1556v6¿ This paper introduces the VGG architecture.

[11] Wilkinson, G. G. (2005). Results and implications of a study of fifteen years of satellite image classification experiments. IEEE Transactions on Geoscience and remote sensing, 43(3), 433-440.

[12] Trohidis, K., Tsoumakas, G., Kalliris, G., & Vlahavas, I. P. (2008, September). Multi-Label Classification of Music into Emotions. In ISMIR (Vol. 8, pp. 325-330).

[13] Tso, B. C., & Mather, P. M. (1999). Classification of multisource remote sensing imagery using a genetic algorithm and Markov random fields. IEEE Transactions on Geoscience and Remote Sensing, 37(3), 1255-1260.

[14] Tsoumakas, G., & Katakis, I. (2006). Multi-label classification: An overview. International Journal of Data Warehousing and Mining, 3(3).

[15] Vens, C., Struyf, J., Schietgat, L., Deroski, S., & Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. Machine learning, 73(2), 185-214.

[16] Zhang, M. L., & Zhou, Z. H. (2005, July). A k-nearest neighbor based algorithm for multi-label classification. In Granular Computing, 2005 IEEE International Conference on (Vol. 2, pp. 718-721). IEEE.

[17] Zhang, M. L., & Zhou, Z. H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. IEEE transactions on Knowledge and Data Engineering, 18(10), 1338-1351.

[18] Zhang, M. L., & Zhou, Z. H. (2007). ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition, 40(7), 2038-2048.

[19] Zhang, Y., & Wu, L. (2012). Classification of fruits using computer vision and a multiclass support vector machine. Sensors, 12(9), 12489-12505.