# Final Report
## Deep Multi-Label Classification for High Resolution Satellite Imagery of Rainforests

Robert Sun
Stanford University
robsun@stanford.edu

Christian Castellanos
Stanford University
ccastell@stanford.edu

Andrew Nguyen
Stanford University
aknguyen@stanford.edu

## Abstract

*Satellite rainforest imagery is extremely helpful in tracking down deforestation and analyzing behaviors that contribute to wildlife destruction. Constant monitoring of this imagery however is completely infeasible for humans and so convolutional neural networks are the perfect candidate for detecting satellite image features. In this paper, we discuss our strategy for multi-label classification of Amazon satellite imagery. Each image can have multiple tags activated like "clear, slash and burn, habitation, water" and the classifier must be able to detect these wide-scale and small-scale features simultaneously.*

*We applied state-of-the-art image classification architectures and developed an original CNN architecture for solving this problem. Using a binary cross-entropy loss function and our own architecture, we achieved an $F_2$ score of 0.9228 on the validation set. We showcase visualizations of the network on a variety of input images to validate that the network is generalizing to the specific land features for each label.*

## 1. Introduction

Rainforests such as the Amazon are under constant attack by loggers, miners, and human civilization for the land and resources it provides. It is important to know and understand the where, when, why, and how of deforestation in order to decide how best to maintain the remaining rainforest.

High resolution satellite imagery of the Amazon and other rainforests are relatively easy to acquire, but tagging imagery for specific features is often time consuming and intractable for a large landmass. Especially when tracking rainforest changes over months or years, human labeling of satellite imagery is completely impossible. The use of a CNN for continuous labeling of this imagery is extremely helpful in determining changes of rainforest condition.

The key attribute of this dataset and problem is that each image can have multiple labels, with little logical dependence between the labels. Each image can have 17 different tags activated; one example can have tags such as "clear sky, primary rainforest, roads, blooming, water", and another have just "haze". We decided that using standard convolutional feed-forward network to predict the labels of the satellite imagery was the best option.

After many trials of different network topologies, we arrived at a multi-layer CNN that takes in 128x128x3 resized RGB satellite images and outputs a 17-long vector representing label activations. This vector directly comes from a sigmoidal activation layer and is therefore squashed between 0-1. The threshold for whether the label is ON or OFF is determined empirically via cross validation after statistical analysis.

### 1.1. Dataset and Kaggle Competition

The pre-labeled dataset of Amazon rainforest satellite imagery was provided by Planet through Kaggle.

Kaggle, a platform for data science and analytics competitions, allows companies and organizations around the world to create data-based competitions and enlist the help of data analysts and machine learning hobbyists. Our project serves as an entry for the Kaggle competition *Planet: Understanding the Amazon from Space*.[13] Planet produces satellites to image the Earth, and has created a Kaggle competition to have entrants apply labels (e.g. regarding atmospheric conditions, deforestation, agriculture, water sources, etc.) to high-resolution satellite imagery of the Amazon Rainforest. Through the competition, Planet hopes to "help the global community better understand where, how, and why deforestation happens all over the world - and ultimately how to respond."[13].

## 2. Related Work

The problem posed here is fundamentally a single label image classification problem applied to seventeen potential labels. As such, it is important to mention prior work related to image classification, as well as the many of the state

of the art models that the authors leveraged apply multiple labels to a given satellite image.

### 2.1. Image Classification

CNNs have been the go-to for image classification since 2012 as per AlexNet in [9] due to advances in GPU computing. For our project, we apply prior advances in image classification to a multi-label problem with CNNs. The multi-label classification problem relies on applying labels to scenes where the labels themselves are not mutually independent, as researched by Boutell in [2]. They provide general frameworks which can be applied to scene classifications.

The authors evaluated a number of different CNN architectures mentioned below and applied transfer learning [12] to the multi label image classification problem. Transfer learning from these architectures allowed us to use pretrained ImageNet weights with a given architecture, and use them to classify our images on each specific label without retraining all of the weights for the entire architecture.

State-of-the-art CNN architectures for image classification include VGG[15], Inception[16], and ResNet[5]. All of these architectures were constructed with single label image classification in mind, but can be easily generalizable to multi-label classification via unfreezing layers and minor modifications.

The VGG[15] CNN architecture relies on sequential convolutions with 3x3 filters followed by max-pooling layers, repeated for depth. The architecture is relatively simple compared to Resnet and Inception and relies on the principle that visual features in an image are hierarchical.

Inception[16] relies on an entirely different design paradigm than those proposed earlier in [9] and [15], which typically relied on sequential convolutions and max pooling layers. The Inception architecture relies on the 'Inception Module', which consist of convolutions and max pooling layers occurring in parallel, and stacked sequentially. This results in a wider, more accurate network with only moderately more computational complexity[16].

ResNet[5] is an incredibly deep architecture (152 layers), especially compared to Inception's 22 layers and VGG's 19. Resnet utilizes learning residual layers, referencing prior inputs in a sequential fashion.[5] Resnet reported record breaking performance in the ImageNet challenge in 2015, besting all architectures prior.

### 2.2. Similar Problems

Albert in [1] *Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale* uses deep CNNs to label land use in urban satellite images. Albert [1] also utilizes the ResNet and VGG architectures for his research. Some single-label land use classification problems similar to our rainforest classification problem include [3] [10] [17] [19] [20] [11] [21] utilizing different information sources. [3] [19] [20] [11] accomplish the task via CNNs.

### 2.3. Loss Function

The competition evaluation metric is the mean $F_2$ score, mentioned in greater detail below. The $F_2$ score biases towards the ratio of true positives to all actual positives (recall). Standard loss functions for classification problems, e.g. logistic regression and softmax for single label and multi-label identification, are convex and and easily posed as minimization problems. However, the $F_2$ score that we seek to maximize is not a convex function. In [6] a novel method for maximizing $F_\beta$ score via F-Measure loss is proposed, allowing the author to easily bias towards recall. Here, the authors decided to use a binary cross entropy loss function given the challenge of implementation. The authors leave implementation as future work.

### 2.4. Visual Evaluation Metrics

Visualizations were important for evaluating model performance. The authors relied on two approaches implemented in the Keras-Visualization project by Raghavendra in [8]. Saliency maps, presented in [15], visualize how the label scores change with respect to the pixels of an input image. This is done by computing the gradient of the output label with respect to the input image; for the purpose of visualization, the gradients are overlayed on top of an input image (with respect to each potential label). The intensities of pixels are used to highlight the magnitude of the gradient. In essence, we can use saliency maps to tell us which pixels of an input image provide the greatest contribution for a class score. From a performance evaluation standpoint, if we are evaluating whether or not the classifier is paying attention to the right pixels of an image for a ground truth label, we can use the attention maps to do so.

Another set of visualization that are used for evaluation are class activation or heat maps via the Grad-CAM method, presented in [14]. The Keras Visualization project uses Grad-CAM internally, and the heat maps produced tell us which regions in an image are most important for a class. E.g., for the 'road' label and an input image, if the classifier is functioning properly, the heat map will put additional intensity in the regions where roads are.

Saliency maps were particularly useful for evaluating the classifier on less visible features; e.g. when one of our classifiers was functioning properly, the saliency map for the 'blooming' label would show the individual pixels within the image where blooming was occurring. However, for more apparent labels such as roads, sparse pixels in the map would be highlighted, rather than the road. The locality and generality benefit from heat maps were more useful in this situation. The visualization would show that the entire re-

gion around a feature from a true label was the one that had the most impact on the label score. Both of them were used as evaluation metrics.

## 3. Dataset

### 3.1. Dataset

The dataset consists of over 40,000 high resolution satellite images, gathered between January 1, 2016 and February 1,2017, over the Amazon basin with a variety of features associated with each image. There are 17 possible labels for each image, and these labels can be separated into three groups: atmospheric conditions, common land cover/land use phenomena, and rare land cover/land use phenomena. The 17 possible tags are: agriculture, artisinal mine, bare ground, blooming, blow down, clear, cloudy, conventional mine, cultivation, habitation, haze, partly cloudy, primary, road, selective logging, slash burn, and water.

According to the data page for the Kaggle competition[13], some of the data may have inherently been labeled incorrectly due to ambiguity of certain features and the nature of crowdsourcing. Additionally, some of the scenes may have been mislabeled. Given the vast size of the dataset, this should be a relative non-issue.

Images in the JPG versions of the dataset provided by Planet come in the form of chips of 256x256 pixels. The class label distribution is shown in bar plot figure 1 out of one of the training sets of over 40000 images. Most images have at least the primary and clear labels, but the bar plot shows the frequency of labels across the training data.

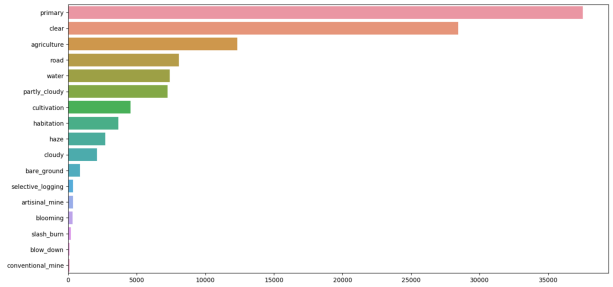


Figure 1. Class Label Distribution over training images, code from [18]

Figure 2 shows example images from each class from the training data, although each image may have more than one label.

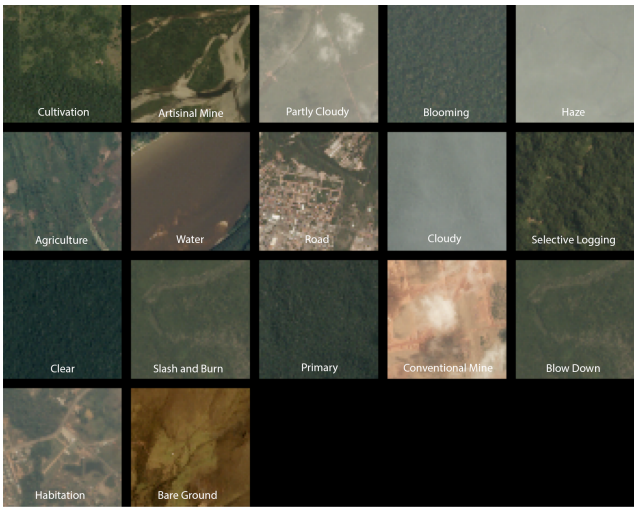


Figure 2. Examples of dataset images

### 3.2. Dataset Split and Test Data

The training set contains 40,479 images and all of the corresponding labels for each image in a .csv file. An 80%/20% split of the training set into training/validation served as a good measure for validation. This leaves us with approximately 32,383 images for the training set and 8,096 for the validation.

In terms of preprocessing our data, we ran our training at various input sizes including 32x32, 64x64, 128x128, and the original image size. Different batch sizes were also tested but we set ours at 128 to achieve a good balance between training time and decent results.

## 4. Methods

### 4.1. Framework and Setup

We are using Keras[4] with a TensorFlow backend to train our networks. Keras is written in Python and provides a friendly and usable framework for instantiating and modifying CNN architectures. Many of the more advanced CNN architectures that we have covered in class have Keras implementations, which allowed us to easily jump in and modify these architectures for our project.

Keras also allow us to easily load images and resize with openCV, all whilst being constrained to a realistic memory limitation. Keras also has an easy way to load and store models while freezing any layers.

The models were trained on two local machines with GTX 1080Tis with 16GB and 6GB of system RAM respectively.

### 4.2. Pre-trained CNN Architectures

The top three image classification models today are classic Deep CNN style (VGG), inception layers (GoogLeNet), and residual layers (ResNet). We choose to trial these three

types of nets on the multi-label classification problem first as they seemed like good options for image classifications, with the use of transfer learning and full random initialization.

Each three of these networks had a 3 Dimensional tensor output (stacked kernels of 2D convolutional output). We took a global 2D average pooling of these CNN feature detectors and added one fully connected layer to get the size of our output. The resulting 17-long vector was evaluated with a binary cross-entropy loss (simple logistic) on each individual element of the vector and the corresponding ground-truth label (T/F). We take the average loss as the loss and validate with labels exceeding 0.5 as T and F otherwise.

$$Loss = \frac{1}{m}\sum L_i = \frac{1}{m}(\sum_{i=1}^{m}(-f_{y_i} + \log\sum_j e^{f_j}))$$

Using ImageNet pre-trained VGG-16, InceptionV3, and ResNet-50 networks, we trained these "bottom" layers separately, freezing the feature network above and assessed their accuracy at convergence after multiple learning rate plateaus.

The results of these networks showed some promise at 85% validation accuracy and on average about 0.75 $F_2$ score. It meant that neural networks were suited for the job but needed a lot more tuning to get accuracy higher. We hoped that tuning the feature detectors (convolutional layers) would increase accuracy. Unfreezing different permutations of the upper layers, we retrained until convergence but there was an insignificant change in performance.

We finally attempted to completely train randomly initialized versions of these networks, as well as deeper versions like VGG-19 and ResNet-112. To our dismay, the randomly initialized ones did converge fairly quickly but produced about the same outputs as the initial experiments we performed. Trying to deduce why these networks failed to produce more accurate results, we ran some visualizations to see how the image pixels activated according to the label activations.

We see in Figure 3 that ResNet is an architecture that looks very broadly in an image. Possibly because of how deep it is and how it is designed for single class identification, it works poorly for detecting all these labels at once and ends up with heat maps that cover large portions of the input image instead of specific features. It is likely the residual architecture does not allow for good separation of the combined detailed and large-scale features. In a single feed-through pass of an image, the network must detect fine-grained features like blooming or roads, but also haze or primary rainforest, features that cover the whole image.

VGG seems to perform a lot better in terms of the areas where the network should be focusing. In Figure 4, the saliency map shows good activation of the bare ground

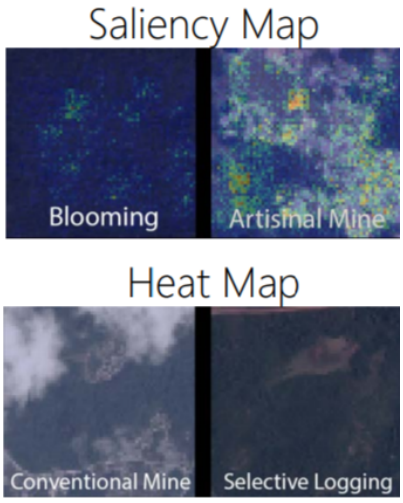

Figure 3. ResNet Visualizations

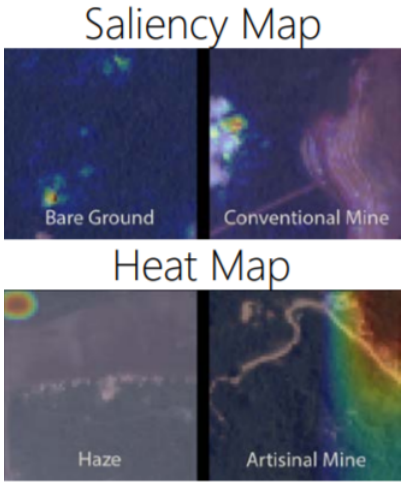

Figure 4. VGG Visualizations

pixels. Still, because of the network depth, possibly fine grained features are lost and suffers the same issue as ResNet. It is possible that these networks were too complex for this application and training in this more complex loss landscape suffered.

### 4.3. Simple Architecture

We sought to fit a simpler architecture and evaluate how it performed. Taking influence from a VGG-style architecture, we created a reduced-depth architecture of two convolutional layers followed by a max pooling layer and two fully-connected layers. Figure 5 shows the architecture layout.

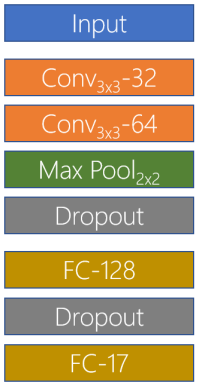


Figure 5. Simple Network

Surprisingly, something as simple as this architecture greatly outperformed the more complex VGG, ResNet, or Inception networks and evaluated to a 92.1% validation accuracy and 0.816 $F_2$ Score. Our hypothesis was correct in that the deeper networks must be getting lost during training. This led us in the direction of simpler custom networks for this problem instead of using the complex ImageNet-winning architectures.

The best score (above) we evaluated with this simple architecture was with an input size of 64x64x3. Even though the dataset had relatively large images, resizing the image might have helped the optimization procedure as there are less trainable weights or added more regularization to the network.

### 4.4. Final Architecture

We cross-validated the increase in network size and found that adding more convolutional layers to the simple network from above worked well to increase accuracy. The resulting architecture stacked blocks of batch normalization, two convolutional layers, max pooling, and dropout on top of each other resulting in fully connected layers. It was a natural extension of the simple network and the successive max pooling helped keep computation down while decreasing the space of our features. The block layout was inspired by other networks with similar goals[18]. With a final 5 blocks of convolutional layers as show in Figure 6, we were able to achieve the best results on the validation set. Anything deeper would have a decrease in F2-score. The batch normalization layers helped increase training efficiency and the dropout served as effective regularizers throughout all networks.

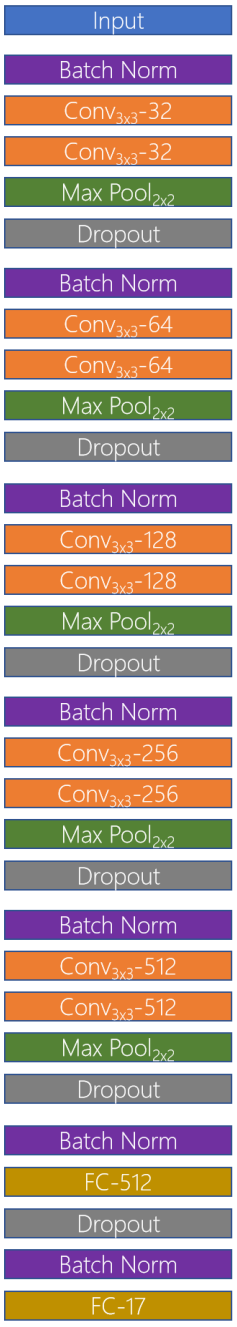


Figure 6. Final Network Architecture

## 5. Experimental Results

### 5.1. Evaluation Metrics

Performance in the Kaggle competition is based on the mean F-Score $F_2$, which measures performance based on precision, $p$, and recall, $r$. The F-Score penalizes a model for false positives, $fp$, and false negatives, $fn$, with false positives being penalized more highly. In the equations below, $tp$ refers to true positives and $tn$ to true negatives. [13]

The equations for recall, precision, and the F-Score $F_2$

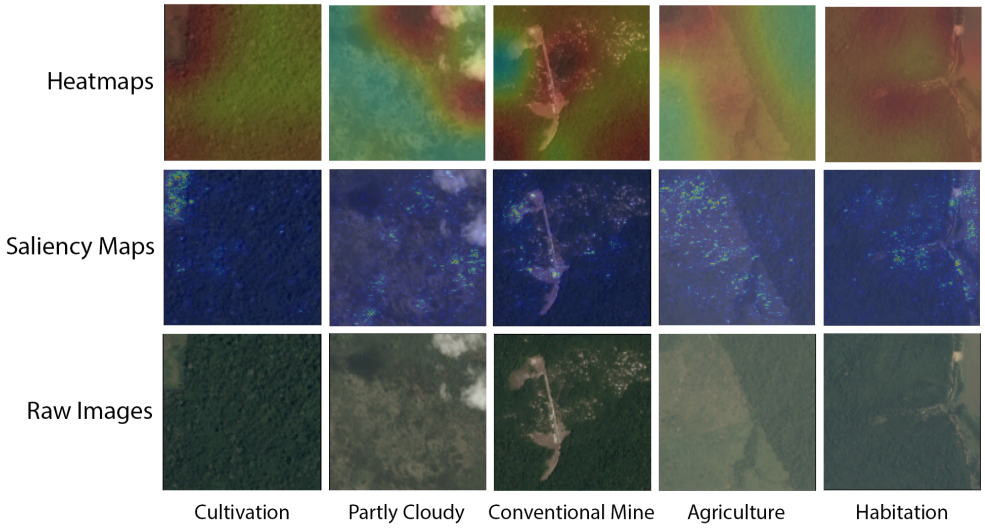


Figure 7. Heatmap and Saliency Map results for best architecture

are given as follows[13]:

$$F_2 = \frac{1 + \beta^2 pr}{\beta^2 p + r}, \text{where}$$

$$r = \frac{tp}{tp + fn}, p = \frac{tp}{tp + fp}, \beta = 2$$

### 5.2. Training Details

Our final architecture uses an Adam[7] optimizer in lieu of SGD. Adam performed reliably and favorably over different update rules in other environments.

For each CNN architecture we trained, we used a learning rate reducer to detect when the loss would plateau and respond by reducing learning rate by a factor of $\sqrt[2]{0.1}$ each epoch. We started learning rate at 0.001 each time and that worked quite well for all networks in conjunction with the learning rate reducer. Training also stopped automatically when the loss would stay the same within a 0.001x margin for 10 epochs.

Each epoch consisted of a full run-through of all training images in batches of 128. The entire validation set was evaluated at the end of each epoch to test for validation accuracy, which is a metric that doesn't fully reveal how well our network performs on the multi-label problem, but serves as an accurate metric for the advancement of training. The $F_2$ score used after training is the final evaluation that matters most.

Image augmentation like scaling or flipping and preprocessing like ZCA whitening seemed unnecessary given the large dataset. After doing a full optimization of our network, we felt that a rework of architecture or loss function principles would bring far better scores than adding image augmentation. The few trials we had with image augmentation only served to increase training time instead of provide generalization to the networks.

### 5.3. Hyperparameter Optimization

After we had determined the optimal architecture, we tried a variety of image-resize sizes in order to further maximize $F_2$ score. Due to the relatively simple architecture, as well as having local machines to train on, iterating over potential image sizes was trivial.We tried $32x32$, $64x64$, $128x128$, and $256x256$, and found that $128x128$ produced the best $F_2$ score on the validation set.

The output of the classifier is a 17-vector of probabilities $[0, 1]$ that represent the probability $p_i, i = 1, \ldots, 17$, where $p_i$ represent the probability that label $i$ applies to an input image. For the purposes of multi-label classification, we instead require a binary 17-vector representing a label being on or off. A threshold value, $t$, is needed as a cutoff for each label $i$, to determine whether the label is active or not for classification purposes. To determine the threshold $t$, we tried two methods: search thresholding (searching for the best $F_2$ score based on varying the thresholds for each label) and per-label sample thresholding.

Per-label sample thresholding had inarguably worse results than the search thresholding. To determine the per-label thresholds, we used the ground truths from the validation set to determine the ratio of 'offs' to 'ons' for a given label. We believed that the classifier's predictions on the validation set would converge to these values, so they would serve as an adequate per-label threshold. They did not, as this resulted in an $F_2$ score of .82 on both the test (evaluated

on Kaggle[13] and validation set, compared to our search thresholding.

Once we settled on a final model, we instead searched over $40$ potential values from $.05$ to $.4$ for each of the labels and find the combination that produced the best $F_2$ score. With thresholds of $(0.134, 0.134, 0.233, 0.219, 0.05, 0.275, 0.078, 0.092, 0.092, 0.458, 0.064, 0.148, 0.148, 0.162, 0.106, 0.458, 0.261)$ for the respective classes: haze, selective logging, bare ground, cultivation, cloudy, clear, partly cloudy, conventional mine, blooming, blow down, artisinal mine, agriculture, habitation, water, road, slash burn, primary.

### 5.4. Final Architecture Results

Our final architecture managed to score $96.5\%$ validation accuracy with a $F_2$ score of $0.9228$ on the validation set. Figure 7 shows several of the resulting heatmaps and saliency maps of some of the classifications which we used to qualitatively verify the effectiveness of our network. Specifically, we can see that the saliency maps and heatmaps for the cultivation label accurately looked at the cleared patch of land in the upper left corner for applying the label. The partly cloudy class feature visualizations are not spot on, but show significantly hotter regions closer to the clouds in their heatmaps. For the conventional mine class, a lot of the superpixels in the saliency map near the bare ground and road features of the image. Likewise, the agriculture class shows similar results with its solid identification of the large patch of cleared ground as farmland in both the heatmap and saliency maps. Finally, the habitation class's features to the left side of the image that pop out from the rest of the forest features are seen as hotter in the heatmap and as superpixels in the saliency map.

Visualizations for different images on the same labels through random sampling shows very similar results. The precision and recall scores on a per-label basis can be seen above in Table 1. Our lackluster performance on labels such as blow down, slash burn, and bare ground hinder our $F_2$ score, as our classifier was not able to correctly predict any true positives, thus setting the precision and recall for those categories to $0$. The ground true positives for those labels were incorrectly predicted as false negatives. Additional focus in the future should be put on enhancing performance on those label scores.

### 5.5. Kaggle Leaderboard

The Kaggle competition provides a set of unlabeled test images. Participants must use their model to evaluate the labels for each of the test images, submit predictions to the Kaggle homepage, and then $F_2$ scores are computed on each set of predicted labels given a test image. The mean of the $F_2$ scores is computed and used as the primary evaluation metric. For the purposes of the Kaggle leaderboard, only a subset of the test $F_2$ scores are evaluated before the end of the competition.

Table 1. Precision and Recall Scores on Validation Set

| Class | Precision | Recall |
|---|---|---|
| haze | 0.554 | 0.867 |
| selective_logging | 0.475 | 0.389 |
| bare_ground | 0.44 | 0.229 |
| cultivation | 0.5 | 0.711 |
| cloudy | 0.603 | 0.965 |
| clear | 0.946 | 0.988 |
| partly_cloudy | 0.855 | 0.966 |
| conventional_mine | 0.368 | 0.318 |
| blooming | 0.32 | 0.397 |
| blow_down | 0 | 0 |
| artisinal_mine | 0.682 | 0.841 |
| agriculture | 0.722 | 0.94 |
| habitation | 0.6 | 0.784 |
| water | 0.62 | 0.854 |
| road | 0.652 | 0.918 |
| slash_burn | 0 | 0 |

Once the network has been fully trained, we can evaluate the mean $F_2$ score for the validation and test sets. The test set consists of 40,479 images, and the labels we compute on the test set are used to evaluate our model's performance on Kaggle's leaderboard via the $F_2$ score. Our final $F_2$ scores of the test set (0.91918) placed us within the top 100 for the competition.

## 6. Conclusions

In the end, the right sized network was the key to our success in this challenge. We learned that the famous architectures like ResNet and VGG were too complex or too deep to train accurately. In testing simple convolutional neural networks and progressively adding convolutional layers we were able to outperform other pretrained architectures.

Looking into the why of how each network worked or failed was extremely insightful in confirming our beliefs of how to make the models perform better as well as give us confidence that a well-trained network was looking in the right spots. The key to qualitatively confirming our tests and results were the visualizations of the labels' heatmaps and saliency maps. While these visualizations guided us in the broad sense toward what network structures worked best, fine tuning the network in search of a better $F_2$ involved diving into a lot of hyperparameter optimization and the exploration of static thresholds and per-class thresholds.

### 6.1. Future Work

Optimizing over the $F_2$ loss for logistic regression, as per [6], is something to pursue in the future given more resources. Implementation is non trivial, but particularly

suited for this project, as well as any other single-label or multi-label classification problems that require maximizing precision or recall with a quantitative bias.

### 6.1.1 Other Architectures

The architecture we used was a relatively simple feed-forward network that tried to predict all labels in one shot. More successful architectures might be to train a full network solely as a binary classifier for each label. This allows the network to solely focus its learnable parameters on one particular feature set and forgo the issues of looking for fine-grained and wide-scale features at the same time.

Another method we could try is to have combine the top layers of these individual architectures and have them split apart into their own branch networks after a couple convolutional layers following the input. Its possible that this achieves similar results to our network if branched too close to the output. The branched networks would simply learn the particular lower-rank representations of our lower layers since the upper-level feature detectors remain the same. An adequate amount of cross-validation would be needed to determine how best to split and combine these unique feature detector networks for each label.

## References


[1] A. Albert, J. Kaur, and M. Gonzalez. Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale. *arXiv preprint arXiv:1704.02965*, 2017.

[2] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004.

[3] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *arXiv preprint arXiv:1508.00092*, 2015.

[4] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[6] M. Jansche. Maximum expected f-measure training of logistic regression models. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 692–699. Association for Computational Linguistics, 2005.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[8] R. Kotikalapudi. Keras visualization toolkit. `https://github.com/raghakot/keras-vis`, 2017.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] M. Lenormand, M. Picornell, O. G. Cantú-Ros, T. Louail, R. Herranz, M. Barthelemy, E. Frías-Martínez, M. San Miguel, and J. J. Ramasco. Comparing and modelling land use organization in cities. *Royal Society open science*, 2(12):150449, 2015.

[11] Q. Liu, R. Hang, H. Song, and Z. Li. Learning multi-scale deep features for high-resolution satellite image classification. *arXiv preprint arXiv:1611.03591*, 2016.

[12] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[13] Planet.

[14] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? *arXiv preprint arXiv:1611.07450*, 2016.

[15] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[17] J. L. Toole, M. Ulm, M. C. González, and D. Bauer. Inferring land use from mobile phone activity. In *Proceedings of the ACM SIGKDD international workshop on urban computing*, pages 1–8. ACM, 2012.

[18] G. Tuatini. planet-amazon-deforestation. `https://github.com/EKami/planet-amazon-deforestation`, 2017.

[19] N. K. Uba. *Land Use and Land Cover Classification Using Deep Learning Techniques*. PhD thesis, Arizona State University, 2016.

[20] M. Vakalopoulou, K. Karantzalos, N. Komodakis, and N. Paragios. Building detection in very high resolution multispectral data with deep learning features. In *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, pages 1873–1876. IEEE, 2015.

[21] Y. Yang and S. Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pages 270–279. ACM, 2010.