# Exploring CNN-RNN Architectures for Multilabel Classification of the Amazon

Cristian Zanoci
Stanford University
czanoci@stanford.edu

Jim Andress
Stanford University
jandress@stanford.edu

## Abstract

*Despite the fact that most of the research into classification via deep learning has focused on single-label classification tasks, there are many important problems which require a set of labels to be output for each example. In this paper, we explore one such scenario, in which satellite images of the Amazon rainforest need to be tagged with labels describing the land use and atmospheric conditions. We focus on architectures which take a two-stage approach. First, we adapt a Convolutional Neural Network (CNN) architecture to encode the input image into a high-level representation. Then, a Recurrent Neural Network (RNN) decoder produces a sequence of labels from this representation. Overall, we find that the CNN-RNN architectures perform well at this multi-label classification technique, although slightly worse than pre-trained CNN models on their own.*

## 1. Introduction

Deforestation of the Amazon rainforest accelerated significantly over the past 20 years. Today, the Amazon rainforest covers only 80% of area it used to cover in 1970, and the annual forest loss is around $6,000\ km^2$. This deforestation results in reduced biodiversity, habitat loss, and climate change. The main causes of deforestation are cattle pasture, excessive and unsustainable farming, and natural disasters, such as forest fires. Luckily, with the recent increase in available satellite images of the Amazon Basin, we have more available data to study the extent of and possible solutions to deforestation in the Amazon.

Recently, Planet has uploaded to Kaggle a dataset [17] of satellite images of the Amazon Basin and challenged its members to help classify different patches of the images according to their atmospheric conditions and type of terrain. As a result of labeling these different parts of the Amazon, we can gain a better understanding of where and why the effects of deforestation are especially severe. By automating the process of identifying damaged areas of the rainforest, we help local authorities to respond faster and more efficiently to these problems.

This task is interesting from both a technical and a societal perspective. First, it provides an interesting twist on the standard image recognition task, as the labels are not mutually exclusive. More importantly, though, success on the task has important implications for sustainability efforts taking place in South America.

In this paper, we first describe a modification to existing CNN models which adapts them for the multi-label classification task. Over the past few years, CNNs have proven to be very successful at image classification and recognition tasks when trained on large-scale datasets, such as ImageNet. However, the relatively small size of our dataset, as well as its non-uniform distribution of labels, makes it difficult to fully train a modern deep neural network, such as ResNet [6] or DenseNet [8]. To overcome this challenge, we use a combination of transfer learning and data augmentation techniques to fine-tune a pre-trained CNN. Next, we propose three architectures that combine CNNs and RNNs to predict the set of labels associated with each satellite image. By incorporating RNNs into the model, we hope to capture the inter-dependencies between labels.

## 2. Background

### 2.1. Remote Sensing

Interpretation of satellite images has a wide variety of applications, from urban planning to modeling climate change. Since these images became available in the 1950s and 60s, the majority of the early work on labeling and classifying them was performed by human experts. However, this process is very tedious and often impractical.

The first attempts to automate this process using neural network models date back to 1989 at the latest [3]. More recently, the increase in the quantity and quality of satellite images has allowed for the application of large-scale machine learning algorithms, which resulted in automated classifiers and detectors with a remarkable degree of accuracy. For instance, Mnih and Hinton successfully applied Restricted Boltzmann Machines to the problem of detecting patterns, such as roads, in satellite images [12, 13].

Similarly, Basu et al. [1] introduced a framework that extracts statistical features from satellite images and feeds them into a Deep Belief Network with Restricted Boltzmann Machines for classification. Although this approach was very successful on the SAT-4 and SAT-6 datasets, it relies on carefully selected features and a large corpus of training images. Our dataset, however, is much smaller and has high inter-class variability, which make the approaches described above less suitable for our project.

Penatti et al. [15] showed that CNNs generalize well to remote sensing images, even though they were trained on a considerably different dataset. Moreover, they show that CNN architectures outperform a wide range of other visual descriptors tuned specifically to recognizing patterns in aerial and satellite images.

Training a neural network from scratch is preferable since it learns to extract specific visual features for the target dataset. This strategy also gives full control over the network architecture and parameters, which often results in a more robust design. However, this approach requires a significant amount of data in order to be effective and avoid over-fitting. As an alternative, one can use networks that were pre-trained on large datasets, such as ImageNet, and fine-tune their parameters using the dataset of interest. The assumption here is that the first layers in a CNN encode generic image features that are relevant to all datasets, while the later stages are application-specific and can be adjusted using our data. Recently, Nogueira et al. [14] performed a comparative study of modern deep neural net architectures which showed that fine-tuned CNNs perform better than fully-trained CNNs when applied to three satellite image datasets. Similarly, Xie et al. [22] showed that a fine-tuned VGG network can successfully learn high-level feature representations of satellite images and predict poverty regions on the map.

Given the success of these approaches, several methods have been proposed to improve the performance of standalone fine-tuned CNN architectures. In [19], the authors introduced a method combining Principal Component Analysis (PCA) and CNNs. The PCA layer before the CNN is meant to synthesize spatial information of remote sensing images in each spectral channel. This design reduces the differences in scale and color between the target images and the images in the original dataset used for pre-training the CNN.

### 2.2. Multi-Label Classification

Although classification is the quintessential machine learning problem, the majority of research into classification algorithms has focused on scenarios in which the classes are mutually exclusive. However, in many important tasks this restriction does not hold, meaning that data points lie in several classes simultaneously. In the decade or so that this problem has been studied, models have been proposed based on both classic machine learning techniques and deep learning.

Amongst the non-neural techniques, Boutell et al. investigate various training and scoring paradigms for multi-label classification in general [2]. For image labelling in particular, a common non-neural technique is a nearest neighbor based approach computed using one of several distance metrics on various statistical features of the images [5, 10]. Although these techniques did achieve some success in some limited situations, their results no longer hold up against deep learning-based approaches.

One of the earliest applications of deep learning to multi-label classification was the work done by Zhang et al., which applied a simple feed forward network to the functional genomics problem in computational biology [23]. In 2013, Gong et al. [4] explored how the application of various multi-label classification loss functions influenced the training of an AlexNet-like CNN architecture [9]. While these results were promising, in our case the techniques explored are less feasible since they train the network from scratch.

Even more recently, Wei et al. put forth a model which combines the labels from a series of proposal regions, each of which is classified using a single-label classification network [21]. By using single-label classification as a subcomponent, Wei's architecture is capable of leveraging the latest advances in CNN models. However, because it involves predicting a label for each of the proposal regions, this method tends to be significantly slower than techniques which predict a set of labels all at once.

Finally, last year Wang et al. produced a paper documenting their efforts to use RNNs in conjunction with CNNs for multi-label classification [20]. Wang's team hoped that by using an RNN to explicitly model conditional dependencies between the labels, their model would be able to exploit the full image context better than previous techniques.

## 3. Dataset

Our data comes from the Kaggle competition "Planet: Understanding the Amazon from Space." The competition provides 40,479 training images and 61,192 testing images, each of which is $256 \times 256$ pixels stored in JPEG format. These images have relatively high spatial resolution: each pixel represents a 3m $\times$ 3m land area. The images are each tagged with some subset of the following labels: 4 atmospheric condition labels ("clear," "partly_cloudy," "haze," "cloudy") and 13 land condition labels ("primary," "agriculture," "road," "water," "cultivation," "habitation," "bare_ground," "selective_logging," "artisinal_mine," "blooming," "slash_burn," "blow_down," "conventional_mine"). Several examples of the images are

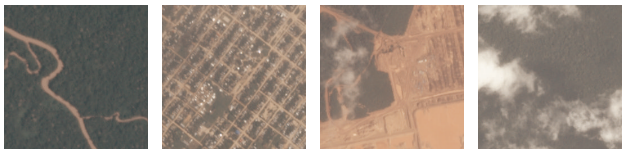


Figure 1: Several example images taken from the dataset.

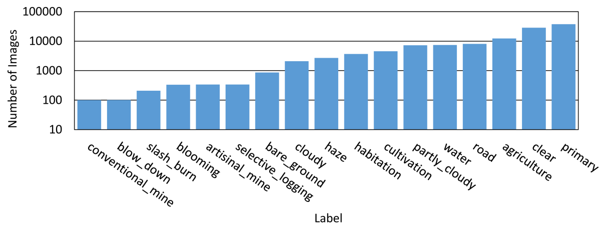


Figure 2: The number of training images with each of the 17 different labels.

shown in Figure 1.

The distribution of labels in the training data is shown in Figure 2. As can be seen, the distribution is highly skewed, ranging from 92.7% of the images having the "primary" label to only 0.25% of the images having the "conventional_mine" label. In an attempt to compensate for the data skew, we augmented the dataset by including many rotated and flipped copies of images with rare labels. We made sure to have at least 1000 example for each label in our augmented dataset. However, this approach did not work as we had hoped. Figure 3 displays the co-occurrence matrix, which indicates what percentage of images with the label on the X-axis also have the label on the Y-axis. Notice that the label "primary" is present with almost all the other labels, while the label "cloudy" very rarely co-occurs with any other label. As evidenced by this data, we found that the images with very rare labels almost always have several extremely common labels as well, meaning that data augmentation did not help much in evening the label distribution. It also did not impact the final scores in any significant way and only added to training times. Therefore our experiments did not use this type of data augmentation.

Nonetheless, we still performed random transformations on our data. Whenever we drew an image from our dataset, we randomly applied transpositions, horizontal and vertical flips, and affine transformations with a small rotation angle and translation to it. Then we randomly cropped the image to $224 \times 224$ pixels to adjust its size to the dimension of the first layer of our CNN. These transformations made our models more robust and prevented overfitting by introducing slight variations in the training data at each epoch.

It is worth mentioning that the Kaggle competition also includes TIFF versions of the dataset in which each image

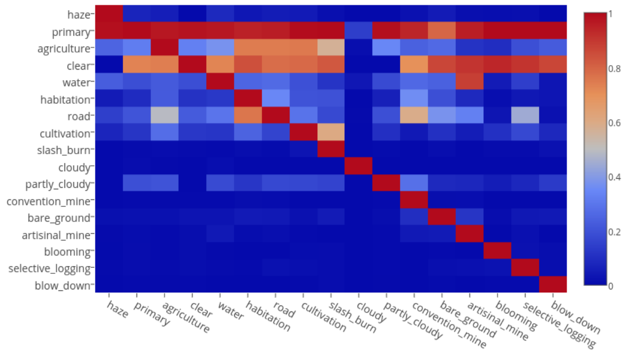


Figure 3: The co-occurrence matrix for the 17 labels (probability of $Y$ given $X$).

includes a fourth, near-infrared (NIR) channel. Unfortunately, based on our own exploration and results from other Kagglers, we chose to only work with the JPEG data. Although in theory the JPEGs and TIFFs should cover the same area, it turns out this is not the case. Many of the TIFF images are shifted, and the labels seem to have been produced by looking at the JPEGs. This discrepancy means that many of the TIFF images disagree with their gold-standard labels, making training on this data difficult or even impossible. This unforeseen feature of the dataset is quite disappointing, as we had initially hoped to use the NIR channel to generate useful features such as the Normalized Vegetation and Water Indices [16, 11].

## 4. Methods

### 4.1. Transfer Learning

Our first approach was to train a Convolutional Neural Network on our data. This choice is justified by the great performance of CNNs on computer vision tasks, which is due to the ability of the convolutional layers to extract spatial and translation invariant features from images. As our main architecture, we use a residual network (ResNet) introduced by He at al. [6]. The original model has 152 layers and relies on a residual learning layers which are easier to train and optimize than conventional feed-forward layers. The residual learning framework also addresses the degradation of training accuracy due to the increase in complexity of the optimization problem that is usually associated with deep networks [6]. ResNet has proven to be very successful by winning the 1st place in all five tracks of the ILSVRC and COCO 2015 competitions, including the image classification and detection challenge. Thus, ResNet is a natural choice for our project.

Although the original architecture consisted of 152 layers, smaller versions of the same network (with fewer repeating blocks) with 18, 34, 50, and 101 layers are also available. As our first network, we chose ResNet-50 because it has significantly fewer parameters than the full 152-layer network, thus making it easier to train on our small dataset while still achieving a good accuracy. We use the PyTorch implementation of ResNet [1] and substitute its last fully connected layer with one of size 17 to accommodate for the number of classes in our problem. For our loss function we use the multilabel soft-margin loss given by

$$L(s,y) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log\left(\frac{e^{s_i}}{1+e^{s_i}}\right) + (1-y_i)\cdot \log\left(\frac{1}{1+e^{s_i}}\right)\right], \quad (1)$$

where $s_i$ is the score for label $i$ as given by the fully connected layer, $y_i$ indicates whether label $i$ is indeed associated with this image, and $N$ is the number of classes ($N = 17$ for our problem). When predicting the labels for an image, we take the scores $s$ from the fully connected layer and map them to the range $[0, 1]$ via a sigmoid function. Then we use a set of per-label thresholds to determine whether a label should be associated with this image. A detailed description of how we determine these thresholds is presented in Section 5.

Unfortunately, as we will show in our experiments section, the model trained exclusively on our dataset underperforms when compared to the other entries on the Kaggle leader board. We believe this is because the size of our dataset, even after augmentation, does not allow the weights to converge to their optimal values when training from scratch.

Therefore, to address this problem, we use CNN models that were previously trained on the ImageNet dataset. The idea is to leverage the low-level image features, such as edges and shapes, that the CNN has learned from the large ImageNet corpus and use them as a starting point for our model. Just as in the case of the CNN trained from scratch, we begin by substituting the final fully connected layer with one that has the correct output dimension for our problem. Since this layer is new, while the rest of the network is pretrained, we only train the fully connected layer for a few (typically 5) epochs until we see the loss plateau. At this point, the final layer is in sync with the rest of the pretrained network and we can start training it as a whole. We use a smaller learning rate over a larger number of epochs to fine tune the parameters of both the CNN and the fully connected layer.

For performance comparison, we also study a different deep CNN architecture, namely DenseNet [8]. Proposed in late 2016, this model has 169 layers and represents one of the newest trends in building CNNs. As opposed to its predecessors, in which each layer is connected to its two immediate neighbors, DenseNet takes the output of each layer and feeds it into each of the subsequent ones. This design reduces the vanishing-gradient problem, strengthens feature propagation, and encourages feature reuse at later stages [8]. DenseNet achieves comparable and even slightly better results than ResNet on both the ImageNet benchmark and our Amazon dataset. However, since DenseNet has about three times fewer parameters than ResNet, we prefer to use it as the backbone of our RNN-CNN architectures below to allow for faster training.

## 4.2. RNN-based Approaches

After having attempted a pure transfer-learning approach, we next sought to better model the interconnections between various labels. As demonstrated by Figure 3, there are strong conditional dependencies between the labels, and we were concerned that the transfer learning approach treated each class too independently. To address this shortcoming, we began to consider the transfer learning CNN as only the encoding phase in a larger architecture. For the decoders, we turned to RNNs, both for their ability to predict variable length sequences of labels as well as their explicit modeling of conditional relationships between predictions at different time-steps[2]. The three RNN decoding architectures we explored are shown in Figure 4.

## 4.3. Label-Only RNN Model

This model draws its inspiration from Wang et al.'s CNN-RNN model [20]. In this model, the CNN and RNN components are completely independent. The CNN is used to produce a high-level representation of the image, and the RNN is used solely to model conditional dependencies between the various labels.

We start by representing each of the labels as a one-hot vector $e_k$, which is all zeros except for a single 1 in the $k$th spot. We can then obtain a distributed representation $w_k$ of the label by multiplying the one-hot vector by the embedding matrix $U_\ell$:

$$w_k = U_\ell e_k \quad (2)$$

For our RNN, we used Long-Short Term Memory (LSTM) cells, which have been shown to avoid some of the vanishing gradient issues common in other RNN implementations [7]. Here, the LSTM takes as input for time-step $t$ the embedding of a label and produces a hidden state $h_t$ as output using the standard LSTM equations. The output at time step $t$ is then a vector

$$s_t = W \cdot ReLU(W_I \cdot I + W_h \cdot h + b_h) + b \quad (3)$$

where $I$ is the output of the transfer learning CNN encoder, $W_I$ and $W_h$ are the projection matrices for the image and

[1]http://pytorch.org/docs/torchvision/models.html

[2]We referenced the code located at https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning for a PyTorch RNN example.

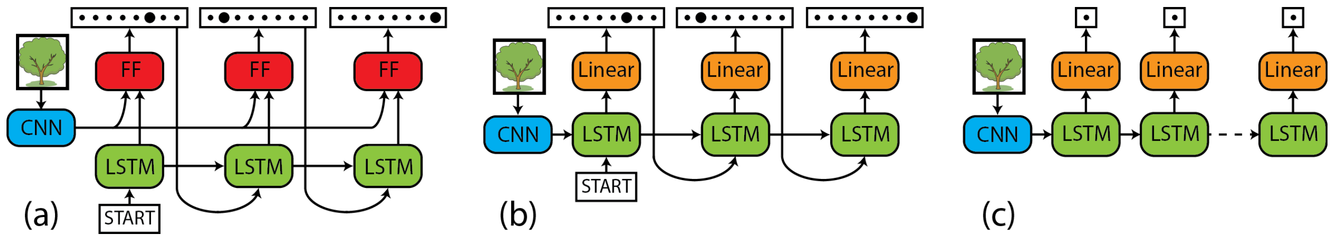


Figure 4: Architectures of the various CNN-RNN models: **(a)** label-only model **(b)** captioning model **(c)** binary decoder model. These decoders are composed of a transfer learned CNN encoder, a sequence of LSTM cells, and either a small feed-forward network or linear layer to get the final output at each time-step.

hidden state respectively, $W$ is a projection matrix with an output of size 18, and $b$ and $b_h$ are bias terms. The score vector $s_t$ is converted to a probability distribution over labels using softmax normalization.

In this framework, the behavior of the model differs at training and test time. Let $L$ be the vector of ground truth labels for an image, sorted from the most common label to the least. During training, at time step $t$ we feed in $L_{t-1}$ as input, and the loss the model incurs is the cross entropy loss between the score vector $s_t$ and the true next label $L_t$. This formulation has edge cases for the input at the first time step and the output at the last time step. To begin, we feed the LSTM the embedding of a special START label, and we force the LSTM to predict a special END label at the end of the sequence (explaining why the $W$ matrix has output size 18 even though there are only 17 true labels).

During testing time, we take a simple greedy approach to predict the set of labels for an image. We start by feeding in the START label to the LSTM, getting an initial score vector $s_1$ as output. The argmax index of this vector is the label which has the highest probability, so this label is fed as input to the next iteration of the LSTM. This process repeats until we have made 17 predictions or until we sample the END label, whichever occurs first. This procedure can be seen pictorially represented in Figure 4(a). In this diagram, the network first predicts label 6, this is then fed as input to the second step which predicts label 2, and when that is fed to the third step we sample the final END label, stopping the process. Although a beam search approach to decoding might have produced slightly better results by avoiding bad early decisions, we chose this greedy approach for its simplicity and speed.

The final potential point of concern is the decision to force the RNN to predict labels in order of decreasing frequency. Although in theory we could have trained the model to predict the labels in any order, we chose this specific order for ease of training. Our simple cross-entropy loss function punishes the model not only if predicts a label which does not apply to the image, but also if it predicts a true label in the wrong order of the sequence. By ordering the labels as we have, the label sequences for the images look as alike as possible for as long as possible.

### 4.4. Captioning RNN Model

This model was based off of the neural image captioning model of Vinyals et al. [18], as well as our third CS231N assignment. This model is similar to the label-only model in that the input to the RNN is the previously predicted label and the output at each time step is a score vector over the 17 different true labels and the END label. However, whereas the previous model combined the output of the CNN with the output of the LSTM using a small feed-forward neural net, in this model the image features from the CNN are fed as the initial hidden state for the LSTM.

### 4.5. Binary Decoder RNN Model

This model combines aspects of the simple transfer learning model and the neural captioning model. Like the captioning model, the image features from the CNN are still fed as the starting hidden state of the LSTM. However, whereas the previous models output variable length sequences, this model runs for exactly 17 time-steps, each time producing a single scalar which is the score for one particular class. Thus, while training we use the same loss function given in Equation 1, and while testing we use the same procedure of predicting any labels whose score is greater than the corresponding threshold.

Because this model has to make an explicit prediction for each of the 17 labels, we expect it to have better performance than the models which output a sequence, since these models will terminate as soon as they predict an END label. However, we also hope that the addition of the RNN component will enable this model to outperform the CNN, since it now explicitly handles connections between labels.

## 5. Experiments

Since there was no validation set provided by the organizers of the competition, we designated 10% of our training data as validation. It was important to sample this data intelligently, since the label distribution was so skewed. If we were to simply take a random selection of 10% of the

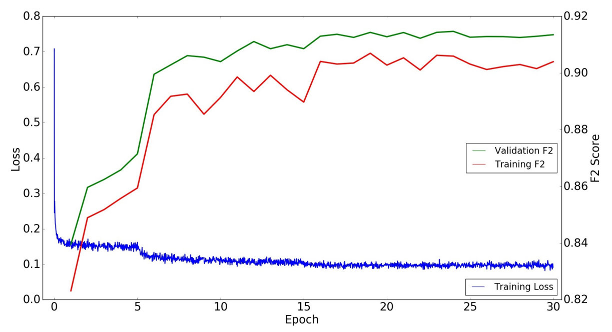


Figure 5: Training loss (blue, left Y-axis) as well as training and validation F2 scores (red and green, right Y-axis) as a function of training epoch.

training data, it is quite possible that our sample would contain no examples with several of the labels. Therefore, we made sure to sample roughly 10% of the examples from each of the labels.

Our experiments were run using the Adam optimizer with a batch size of 32. The majority of our models involved a pre-trained CNN module with either a new linear layer or RNN at the end. Therefore, in order to train these models, we first trained only our custom layers for around 15 epochs. Once those new layers had converged, we began to train the entire network as a whole for another 20 epochs. The learning rate during the first round of training was larger than during the second (they typically started at around $10^{-3}$ and $10^{-4}$, respectively), and we implemented an annealing learning rate which decreased by a factor of around 1.5 after each epoch.

Figure 5 shows the evolution of training loss and F2 scores during the course of 30 training epochs for the ResNet-50 model. We see that decreases in the training loss correspond to increases in the F2 scores, even though we don't optimize directly for the F2 score during training. The slightly higher validation F2 values suggest that our models don't suffer from overfitting.

In the testing phase of both the CNN-only and binary decoder RNN models, the predicted labels are found by selecting all scores which lie above some designated threshold for that class. Because each threshold only impacts the predictions for a single label, each threshold can be computed independently. Thus, at the end of training, in order to compute the thresholds we simply generated scores for each of the examples in the training set, and then for each label found the threshold value which gives the best F2 score. We found that the threshold were vastly different for each class, ranging from 0.08 for the "road" label to 0.39 for "conventional_mine". This can be interpreted as follows: labels with low thresholds, such as "road", are easy to identify and occur rather often in our dataset. Therefore we are very confident in our prediction for these classes. On the other hand, labels with high thresholds, such as "conventional_mine", are harder to distinguish and occur very rarely in our dataset. Hence we are not so confident in these prediction and only assign these labels when we have an extremely high score for them.

### 5.1. Evaluation Metric

To evaluate our results, we are using the average F2 metric, which is the metric being used in the official Kaggle competition. For each example, we compute the F2 score via the equation

$$F_2 = (1 + \beta^2)\frac{pr}{\beta^2 p + r} \tag{4}$$

where $p$ is the precision of the predicted set of labels, $r$ is the recall, and $\beta = 2$. Note that based on this metric, it is better to predict a label an incorrect label (a false positive) than it is to not predict a correct label (a false negative). The final score is simply the average across all testing examples.

### 5.2. Results

The results from each of our explored models are shown in Table 1. We create an ensemble from our best-performing models, where the labels for each image are determined by majority vote from the labels predicted by each model. That is, if a label is predicted by at least half the models in our ensemble, then that label is predicted for the image.

We found that the binary RNN model had the best performance of the three RNN models explored, although it was still worse than any of the pretrained CNN models. Notice that the ResNet-50 model that was trained exclusively on our dataset performed significantly worse than its counterpart that was pre-trained on the ImageNet dataset. This performance test convinced us to switch entirely to using pre-trained CNNs. We also experimented with the number of layers for the ResNet architecture. Table 1 shows that the F2 scores become progressively higher as we increase the depth and number of parameters in our model. This suggests that indeed deeper architectures perform slightly better on this task. Also, the performance of DenseNet-169 is on par with the rest of the models, even though it has fewer parameters. This justifies our use of DenseNet as the main convolutional network for all the RNN-CNN architectures.

Figure 6 shows a few example images on which our models try to predict labels. In the leftmost image, we correctly identify all the labels: primary, agriculture, clear, cultivation, habitation, and road. In the next image, our predictions also match the ground truth, except that we also predict the existence of a road in this picture. However, if we look more carefully, we can see that there is indeed a road which was not included as a ground truth. This shows that the labels are not always correct. Lastly, we get the wrong

Figure 6: Several examples on which we generate predictions.

labels on the rightmost image by predicting habitation, conventional_mine, and bare_ground. It shows that our model has difficulties predicting the more rare labels.

| Model | Val F2 | Test F2 |
|---|---|---|
| ResNet-18 | 0.91104 | 0.92229 |
| ResNet-50 (not pre-trained) | 0.88916 | 0.90269 |
| ResNet-50 | 0.91462 | 0.92335 |
| ResNet-152 | 0.91387 | 0.92401 |
| DenseNet-169 | 0.92755 | 0.92276 |
| Label-Only RNN | 0.91184 | 0.90624 |
| Caption RNN | 0.90765 | 0.90467 |
| Binary RNN | 0.91168 | 0.91127 |
| Ensemble | - | 0.92555 |
| Current Kaggle Leader | - | 0.93334 |

Table 1: A table showing the F2 scores for each model on both our validation and test sets.

Figure 7 shows the precision and recall for each of the labels as generated by (a) a CNN-only model and (b) the label-only RNN-based model. These models had fairly similar scores on the test set (the difference in F2 score was around 0.015), but by examining the data we can see that they took very different routes to achieve this high score. Across the board, the CNN-based model had a much higher recall score, meaning that when there was a label that it was unsure of, the model tended to predict the label rather than leaving it out. The fact that this model had higher recall scores is unsurprising: as previously discussed, we chose thresholds which specifically optimized for F2 score, and the F2 score weights recall over precision.

On the other hand, the RNN-based model was more conservative and tended to have a higher precision score, meaning that it would only predict labels that it was extremely confident of. Again, we can trace this behavior back to features of the model. The models which generate sequences stop as soon as they predict an END label, which means that the model can avoid making a judgment call on some of the harder, more rare labels. This is opposed to the models which use thresholds, as they are forced to produce a score for each label.

This explanation is supported by the other two RNN models, caption and binary. Although we have not included the graphs, the corresponding image for the caption RNN model looks very similar to the label-only model, and the binary RNN model looks similar to the CNN. Thus, we see that the two models which output sequences tend to have higher precision, and the two models which generate a score for each label tend to have higher recall.

## 6. Conclusion

In conclusion, we have demonstrated several successful models for multi-label classification on satellite images. In particular, we have explored three different RNN-based models. Interestingly, although each of the three models performed well, we have shown that they perform well for different reasons. While models that output a score for each label tend to take risks and have high recall, those which output sequences tend to be more risk-adverse and have higher precision.

In the future, we would like to extend our work by exploring different loss functions and prediction strategies for the RNN models which produce sequences. Again, while in theory the sequence model should be able to learn to output the labels in our forced ordering, it is likely that this strict requirement causes the model to focus on getting the labels in the correct order rather than focusing on learning the true labels to predict. We would also likely to explore further data augmentation techniques, as it is likely that without more examples of the rare labels it will be impossible to achieve much higher F2 scores.

## References

[1] S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. Nemani. Deepsat: a learning framework for satellite imagery. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 37. ACM, 2015.

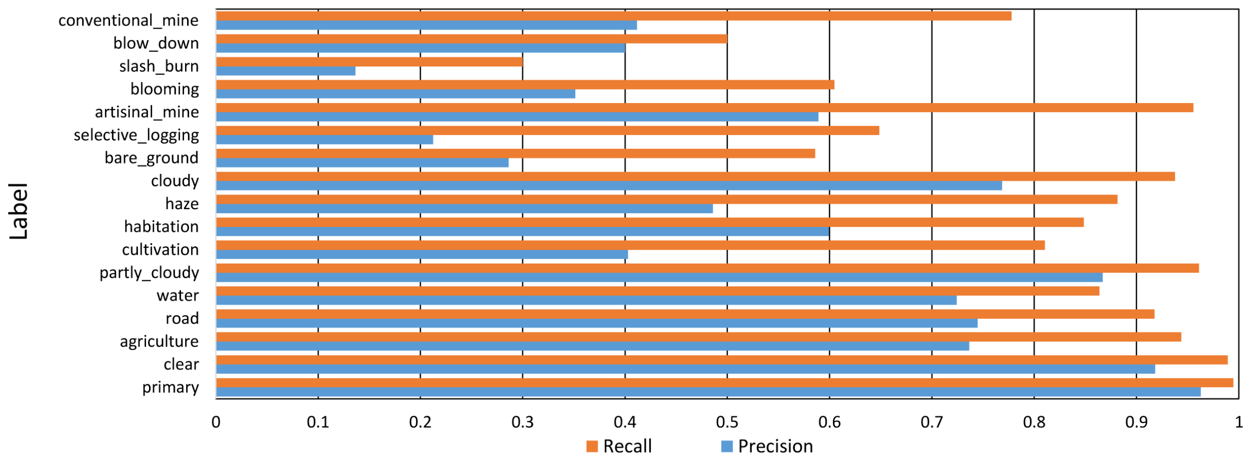
[2] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757 – 1771, 2004.

[3] S. E. Decatur. Application of neural networks to terrain classification. In *Proc. IJCNN*, pages 283–288, 1989.

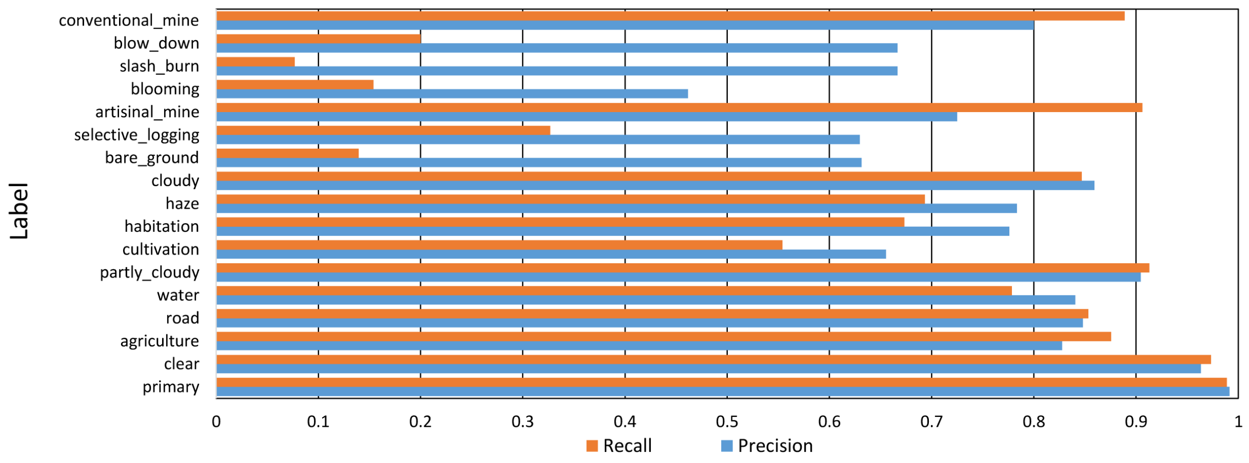
[4] Y. Gong, Y. Jia, T. Leung, A. Toshev, and S. Ioffe. Deep convolutional ranking for multilabel image annotation. *arXiv preprint arXiv:1312.4894*, 2013.

[5] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 309–316. IEEE, 2009.

[6] 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.



**(a)** CNN-only model.



**(b)** Label-only RNN model

Figure 7: The precision and recall for each of the classes as generated by a two different types of models.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[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] A. Makadia, V. Pavlovic, and S. Kumar. A new baseline for image annotation. *Computer Vision–ECCV 2008*, pages 316–329, 2008.

[11] S. K. McFeeters. The use of the normalized difference water index (ndwi) in the delineation of open water features. *International journal of remote sensing*, 17(7):1425–1432, 1996.

[12] V. Mnih and G. Hinton. Learning to detect roads in high-resolution aerial images. *Computer Vision–ECCV 2010*, pages 210–223, 2010.

[13] V. Mnih and G. E. Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 567–574, 2012.

[14] K. Nogueira, O. A. Penatti, and J. A. dos Santos. Towards better exploiting convolutional neural networks for remote sensing scene classification. *Pattern Recognition*, 61:539–556, 2017.

[15] O. A. Penatti, K. Nogueira, and J. A. dos Santos. Do deep features generalize from everyday objects to remote sensing and aerial scenes domains? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 44–51, 2015.

[16] J. Rouse Jr, R. Haas, J. Schell, and D. Deering. Monitoring vegetation systems in the great plains with erts. *Goddard Space Flight Center 3d ERTS-1 Symp.*, 1974.

[17] `https://www.kaggle.com/c/planet-understanding-the-amazon-from-space`.

[18] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[19] J. Wang, C. Luo, H. Huang, H. Zhao, and S. Wang. Transferring pre-trained deep cnns for remote scene classification with general features learned from linear pca network. *Remote Sensing*, 9(3):225, 2017.

[20] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2285–2294, 2016.

[21] Y. Wei, W. Xia, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan. Cnn: Single-label to multi-label. *arXiv preprint arXiv:1406.5726*, 2014.

[22] M. Xie, N. Jean, M. Burke, D. Lobell, and S. Ermon. Transfer learning from deep features for remote sensing and poverty mapping. *arXiv preprint arXiv:1510.00098*, 2015.

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