Understanding the Amazon from Space

Vishal Subbiah*
Stanford University
svishal@stanford.edu

Brent Lunghino*
Stanford University
lunghino@stanford.edu

Brian Rohr*
Stanford University
brohr@stanford.edu

Abstract

In this work, we use convolutional neural networks to assign atmospheric conditions labels and land usage labels to satellite images of the Amazon Rainforest. Convolutional Neural Networks have successfully been used previously on various satellite imagery classification. We have trained several neural networks including custom convolutional architectures, ResNets and GoogleNets. Our best result is 0.918 (F2 score) on the test set.

1. Introduction

The Amazon Rainforest is extremely important to preserve due to its unmatched biodiversity and great capacity to uptake carbon dioxide [1], but it is difficult to make a case for the urgency of its preservation without first quantifying the rate of its destruction. Since the Amazon covers a 5.5 million square kilometer area, understanding and quantifying the changes in land use there over the years is a formidable task. With a great deal of effort, humans could classify one set of satellite images of the Amazon, but it is intractable for humans to go through satellite images of the entire rainforest one by one and manually label how the land is being used, and this task would need to be completed many times in order to understand how the land use is changing over time.

For that reason, we have trained deep convolutional neural networks that will be able to rapidly process satellite images of the Amazon and output labels and metrics that quantify how the land is being used. Specifically, the model takes satellite images of the Amazon rain forest as inputs and outputs one or many of seventeen possible labels for each image. This model will be able to be used to classify the remaining hundreds of thousands of images that are not in the training set, and it will be able to be used to classify new images that are taken at future time points. With this machine learning image processing technique, temporal land use data will be easily assembled, providing the necessary data to make a case for protecting the Amazon Rainforest. Furthermore, satellite data of the rain forest can be applied to compute deforestation rates, detect illegal mining/deforestation, and differentiate between natural and human-caused deforestation.

[0]*All authors contributed equally to this work.

2. Related Work

Many previous studies have applied convolutional neural networks to a variety of machine learning problems that use satellite images and other remote sensing data as inputs with some success. Models that predict a single label for each satellite image have achieved a test accuracy of greater than 99%. [28]. Convolutional neural networks have also been used successfully on satellite image segmentation problems. [16] [26] The key advantage of using convolutional neural networks as opposed to other machine learning techniques is that convolutional neural networks are able to learn problem-specific features of the input images that help with the classification or segmentation problem at hand. [14] This has been shown to be true for many applications in remote sensing image labeling from crop identification to poverty mapping. [27] [12] [25]

Previous studies have used a variety of convolutional neural net architectures. These architectures include standard convolutional architectures, similar to VGG-Net[22] as well as newer architectures, similar to ResNet[7] and GoogleNet[23]. In general, in studies applying these techniques to satellite image classification, the newer architectures give better accuracies than the VGG-Net-like architectures.[6] [15] [17] [21]

Previous studies have also employed a variety of training techniques. In this field, it is popular to use transfer learning: to begin with a convolutional neural net that has already been trained on ImageNet,[20] and then fine-tune the network for the new application. Several studies report good results using transfer learning for satellite image classification problems. [24] [17] [3] [21] [8] However, some studies report that for particular architectures, training from scratch gives better results than fine-tuning a pre-trained network.[5] Additionally, previous work has reported that using data augmentation and ensembling im-

proves the accuracy on the test set. [8] [12] Furthermore, importantly, it has been shown that models trained on satellite images of one region can be successfully applied to unseen regions.[25] For example, a neural network trained on images of one set of cities was able to perform well on completely unseen cities. [3] Overall, it has been shown that convolutional neural networks are very useful for automatic satellite image labeling tasks.

3. Methods

Our goal is to train a neural network that gives a high F2 score on the validation set. We have implemented many architectures and used cross-validation to tune hyperparameters and select the best architecture. We implement the methods discussed below using PyTorch [19] and scikit-learn [18] [1].

The F_β score is defined as:

$$F_\beta = (1 + \beta^2)\frac{p \cdot r}{\beta^2 p + r} \tag{1}$$

Where precision is the ratio of true positives to all predicted positives defined as $p = \frac{tp}{fp+tp}$ and recall is the ratio of true positives to predicted positives and incorrectly predicted negatives defined as $r = \frac{tp}{tp+fn}$.

Enhancing training speed

To enhance training speed, we will use the Adam optimizer[11], which generally finds local minima in lesser iterations than stochastic gradient descent. Our optimizer has an adaptive learning rate that scales down when loss decay stagnates [13]. We performed our initial hyperparameter tuning on a subset of the training data set in order to quickly identify a good order of magnitude for each hyperparameter. Also, since the learning rate, one of the most important hyperparameters, decays automatically, it is sufficient to set the learning rate to any reasonable number. Then, the algorithm will attenuate the learning rate over time, allowing the loss to settle to the bottom of the minimum in the high-dimensional loss landscape.

Activation Functions

In this work, the activation function is the rectified linear activation unit (ReLU), which has many advantages and some disadvantages. The principal advantage is that it is very simple and computationally efficient, and it provides the necessary non-linearity that allows the model to capture more complex relationships within the data. One downside of the ReLU activation function is that it has zero slope over half of its domain. Therefore, during backpropagation, it can pass zero derivatives backwards, causing all parameters directly downstream from that activation function to receive no information about the gradient of the loss upstream from that activation function.

[1] We used CS 231N assignment 2 code as a starter for our project

Overfitting/Underfitting (Bias/Variance Analysis)

It is important to keep track of whether the model is overfit or underfit. To check how overfit or underfit our model is, we will monitor the loss, validation accuracy, and training accuracy as a function of epoch for each set of hyperparameters that we use. If the validation accuracy begins to suffer while the training accuracy continues to increase, we know that our model has overfit. However, if the loss converges quickly and the training accuracy and validation accuracy are similar, we will know that the model is underfit. Some methods that we can use to fight against overfitting or underfitting are regularization, data augmentation, and model complexity. If the model is overfit, we could increase regularization strength, decrease the model's complexity, or add data augmentation. If the model is underfit, we could decrease regularization or increase the model's complexity by adding more layers or more filters in the convolutional layers.

Sigmoid Cutoff Optimization

Each of our models outputs a score for each label representing how strongly the model believes that label should be assigned to the input image. The scores are then passed through a sigmoid function so that they are between zero and one. Labels are independently assigned a True (the label should be assigned to the image) or False (the label should not be assigned to the image) depending whether the sigmoid of the score for that class is above a certain cutoff. After training a model and outputting the sigmoid of the scores, we run a brute force optimization on the values of these cutoffs for each class to maximize the F2 score on the validation set.

Ensemble Weights Optimization

Some of our architectures involve training multiple convolutional neural networks in parallel and combining the results with a weighted sum. For these architectures, we express the F2 score as a function of the weights and determine the optimal weights by using Particle Swarm Optimization [10] maximize the F2 score.

Batch Balancing

Some labels are very common and others are very rare. To ensure that our model learns about the rare labels as well as the common ones, we have implemented batch balancing functionality. This functionality, when turned on, ensures that the model sees all labels equally frequently despite the fact that the training set is very unbalanced.

Architectures:

6-layer CNN

The first deep convolutional neural net presented in this work is a 6 layer net built from scratch. The model takes in the 256 x 256 x 4 images and passes them through a convolution layer with 16 filters with size 3x3, stride 1, and pad 1. We use a ReLU activation function on the result followed by a batch normalization layer. After that, a max pooling layer is applied with kernel size 2, stride 2, pad 0. This conv - ReLU - batch normalization - max pool block is applied 6 times, and the result is flattened and passed through two fully connected layers with 1024 hidden nodes and 17 outputs, respectively.

This network design takes capitalizes on the core principle of convolutional layers: During training, the filters are modified to look for the for the features most relevant to our labeling task. However, this design does not benefit from any of the advantages of the more recently developed architectures. These advantages are described in the subsequent sections.

GoogleNet

This model takes in 256x256x3 RGB images as the input. The first layer is a convolutional layer with 32 filters of kernel size 3, stride 1, pad 1 so the output image height and width is the same as the input image. Next, the model has two inception modules, inspired by GoogleNet.[23] Each inception module has 16 1x1 filters, 16 3x3 filters, and 16 5x5 filters. The output of the inception modules is flattened and passed through a final affine layer, giving scores the scores for each of the 17 classes.

This model uses convolutional layers of different kernel sizes, and the output is concatenated. The advantage of this approach is that the model is able to look for features of different sizes. One downside is that the model is much slower to train.

ResNets

This model takes in 224x224x3 RGB images as the input. The first layer is a standard convolutional layer with 64 filters with kernel size 7, stride 1, pad 3, so the output image height and width is maintained and now has depth 64. The next layer is a max pooling layer with kernel size 2, stride 2, pad 0, so the output image is half the height and half the width of the input image. Next, there are any number of "residual blocks." A residual block takes the input and passes it through 2 convolutional layers (with a ReLU activation layer in between them) with 64 filters with kernel size 3x3, stride 1, pad 1, so the output size is the same as the input size. The distinguishing feature of this architecture is that the output of these convolutional layers is added to the input, and this sum is passed on to the next layer.

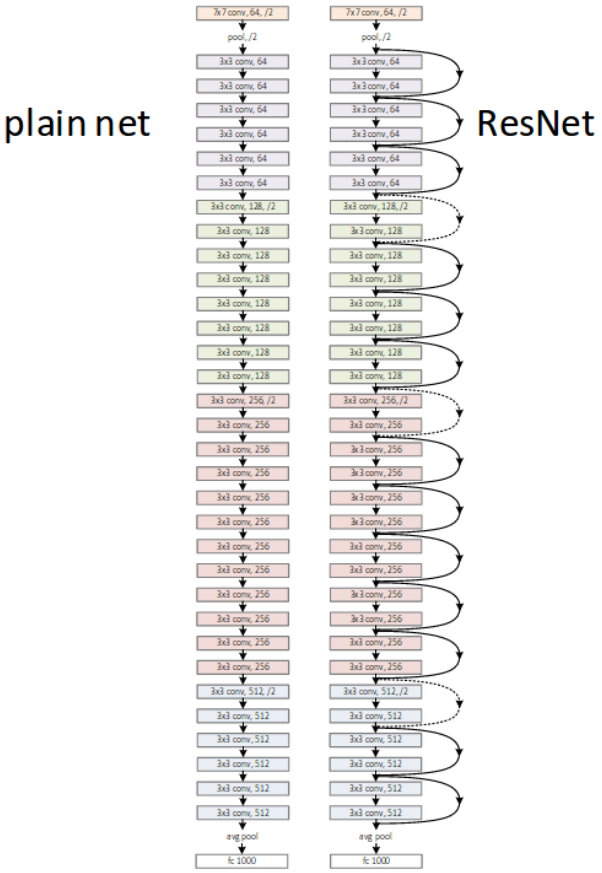
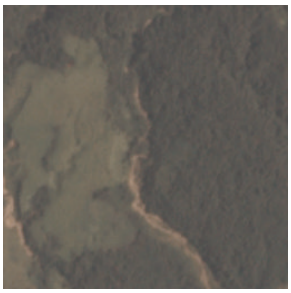


Figure 1: Schematic of ResNet architecture compared to traditional neural network [7]

In this work, we present models with 18, 50, 101 residual blocks. Finally, the output is passed through an average pooling layer and an affine layer to yield the scores for the 17 classes.

This architecture has many advantages. Most importantly, during backpropagation, the derivative of the loss function is passed directly to each residual block. This helps greatly to prevent very deep architectures from suffering from vanishing gradients. Furthermore, the pair of 3x3 convolutional layers in each residual block has the same receptive field as a single 5x5 convolutional layer, but it requires fewer parameters. Also, due to the ReLU activation function between the two convolutional layers in each residual block, the block is able to identify more non-linearities than the single 5x5 filter.

This model gives our best results, and it is the one that has been used for the majority of our experimentation. A diagram of a ResNet-34 is included above.

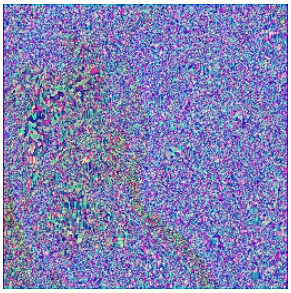


(a) jpg of Sample 1

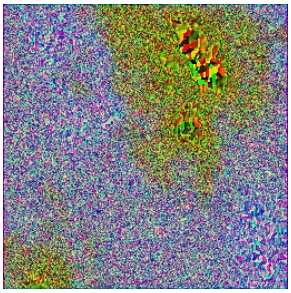
(b) jpg of Sample 2

(c) tif of Sample 1 with infrared

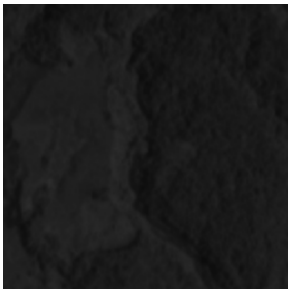
(d) tif of Sample 2 with infrared



(e) gradient image of Sample 1



(f) gradient image of Sample 2



(g) Infrared image of Sample 1

(h) Infrared image of Sample 2

Figure 2: Sample 1 is classified as agriculture, clear, primary and water while Sample 2 is classified as partly cloudy and primary

4. Problem Statement, Data Set, and Features

Data Set Details

The goal is to train a model that can take a satellite image of the Amazon Rainforest as the input and output a 1 or a 0 for each of 17 possible labels. The labels are cloudy,

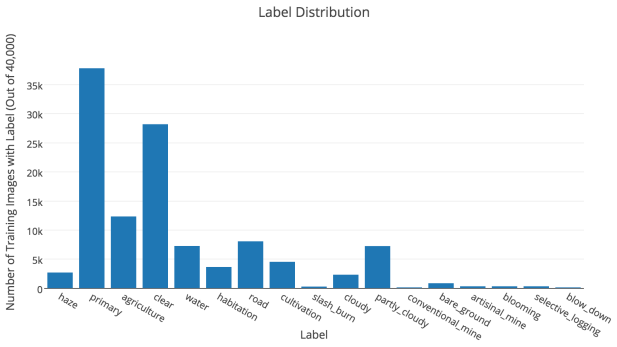


Figure 3: Label Distribution [4]

partly cloudy, hazy, primary rainforest, water, habitation, agriculture, road, cultivation, bare ground, slash and burn, selective lodging, blooming, conventional mining, artisinal mining, and blow down. Each image could one or many labels. For example, an image may have roads, water, habitation, and primary rainforest, and another image could have just primary rainforest. The only constraint is that each image must have at least one label. The data is provided by Planet for a Kaggle competition[2]. The training data set is 40,000 images with an additional 60,000 image test set. During each training epoch, 10% of the training data was used for validation, and the validation set is chosen at random. Each image is 256x256 pixels with 4 bands of color (RGB + near-infrared). Each image captures a land area of 947 meters x 947 meters, giving a resolution of 3.7 meters per pixel. The dataset covers 30 million hectares of Amazon Rainforest. Each of the 40,000 training examples has already been manually labeled as either having or not having each of the 17 possible labels.

The class distribution among the labels is not at all balanced. For example, the labels ”primary forest” and ”clear” (weather) each appear in more than 25,000 of the 40,000 training images, while ”conventional mine” and ”blow-down” appear extremely infrequently [4].

Preprocessing

Some of the models presented were pretrained on ImageNet.[20] In these cases, the images are normalized based on the mean and standard deviation of the images in the ImageNet dataset. When training models from scratch, the images were normalized according to the mean and standard deviation of the entire training set (per channel).

For some models, additional preprocessing steps are taken. The original data consists of 3 channel RGB data from atmospherically corrected .jpg images and 4 channel RGB + near-infrared (NIR) data from uncorrected radiance stored as .tif files. These can be seen in Figure 2 a - d. Other unique color channels were constructed from this data. We

mapped the NIR channel to 3 color bands, shown in 2 g-h. We also generated the gradients along X and Y for the gray scale image, shown in 2 e - f. The gradient images may look noisy at first glance, but they actually contain unique information. For example, the cloud can be easily seen in the top right of figure f. After training separate networks in parallel on data sets that had been pre-processed differently then combining the results, we actually achieved a higher F2 score on the F2 set than any of the individual models.

Data Augmentation

Data augmentation is a crucial strategy for fighting overfitting. When training the models presented, the input images were randomly rotated and flipped then randomly cropped from 256x256 to 224x224. Using these two methods of data augmentation, the model likely never sees the exact same training example twice during training.

5. Experiments, Results, and Discussion

We have explored a variety of models to maximize our F2 score accuracy. A summary of the results for our models is presented in Table 1. The best model is the triple ResNet18 which achieved an F2 score of 0.918 on the test dataset. Figure 4 shows examples of images for which our best model assigned all correct labels, partially correct labels, and completely incorrect labels. The model successfully classifies common labels, such as primary forest and clear skies but struggles to classify less common labels, such as agriculture. The model also struggles to predict features that only occupy a small portion of the image. Each model that we developed is described below along with an assessment of its performance and its contribution to the development process.

6-layer CNN

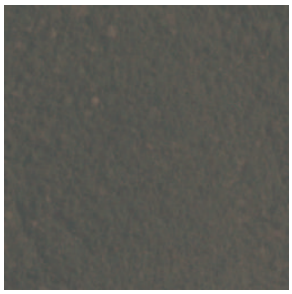
This was a simple architecture that was implemented first as a baseline from which to build on. After training for 10 epochs, the validation accuracy plateaus at 0.83, as seen in Figure 5. Since, at the end of 10 epochs of training, the training accuracy is still increasing, but the validation accuracy has plateaued, this model is beginning to overfit the training data, and further training would likely not result an increase in F2 score on the validation or test set. Since other network architectures, like GoogleNet and Resnet, have performed better in the ImageNet competition,[20] we quickly moved on to these state-of-the-art architectures.

Model Name	Validation F2	Test F2
Basic Conv 6-layer	0.83	0.831
GoogleNet	0.86	
Basic ResNet 18	0.887	
Basic ResNet 101	0.88	
Triple ResNet 18	0.89	0.918
Triple ResNet 18 RinfB	0.85	0.893
ResNet 50 Balancing	0.8384	0.906
ResNet 18 from scratch	0.89	0.912

Table 1: Summary of trained models and results

(a) Correct: clear, primary; Predicted:clear, primary

(b) Correct:agriculture, haze, primary, water; Predicted:cloudy



(c) Correct: agriculture, cultivation, partly cloudy, primary Predicted:agriculture, clear, cultivation, primary, road

Figure 4: Qualitative results: example images for which the model assigned all labels correctly 4a, no labels correctly 4b, and some labels correctly 4c

GoogleNet

The GoogleNet architecture outperformed the simple 6-layer CNN with an F2 score of 0.86 on the validation set. This model took much longer to train, however. Since the training and validation accuracy for this model were nearly identical, it is likely that the model is underfit, and a more complex model would give better results. It would be a reasonable next step to try to train a GoogleNet with more inception modules or more filters in each inception module or additional affine-ReLU layers at the end, however, this option was not pursued. Instead, we explored the ResNet architecture in more depth because the time required to train an epoch was much lower in the ResNet case, and the ResNet architecture performed slightly better in previous image classification tasks. [21] [20]

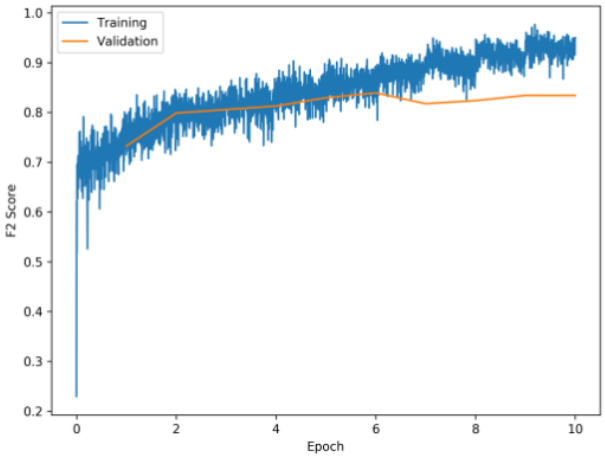


Figure 5: Accuracy (F2 score) for the training set and the validation set as a function of epoch. The training accuracy was plotted at the end of each batch of data, and the validation accuracy was plotted at the end of each epoch.

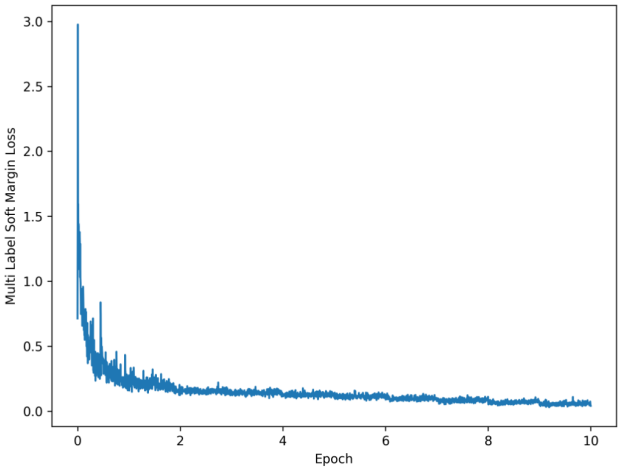


Figure 6: Training loss (PyTorch multilabel soft margin loss) as a function of epoch. It is clear that learning rate decay was introduced near the start of the second epoch, and after 10 epochs the loss is approximately converged.

Pretrained ResNet18

One strategy we explored was fine-tuning pretrained models [2]. Our first effort with this strategy was to fine-tune a ResNet18 initialized with weights from training on the Imagenet dataset. Before fine-tuning, we resized the final fully connected layer to output 17 scores instead of the 1000 used for Imagenet classification. The final fully connected layer was trained individually for 4 epochs at a high learning rate (1e-3). After the loss had stopped decreasing

[2]Pretrained model code was based off the example of [9]

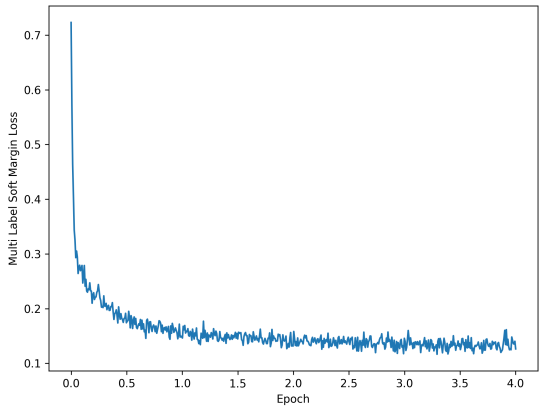


Figure 7: Pretrained ResNet18, loss during training of final fully connected layer

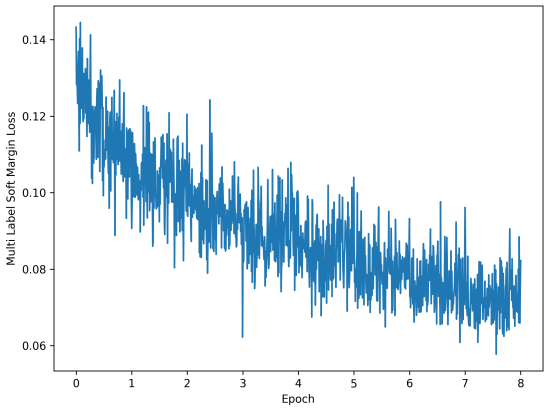


Figure 8: Pretrained ResNet18, loss during fine-tuning of all model weights

by training only the final fully connected layer, (Figure 7) we dropped the learning rate (to 1e-5) and began updating the weights for all layers of the model. In this fine-tuning stage, the loss is able to decrease further (Figure 8). The batch size was 256 images per batch. This model performed quite well considering how quickly it trained and how simple the model was. The F2 score for this model on the validation set was 0.887, which is only about 0.04 less than that of our top-performing models. Since the training and validation accuracy were very similar for this model, it is likely that the model is underfit, and the quantity of data we have warrants a more complex, deeper model.

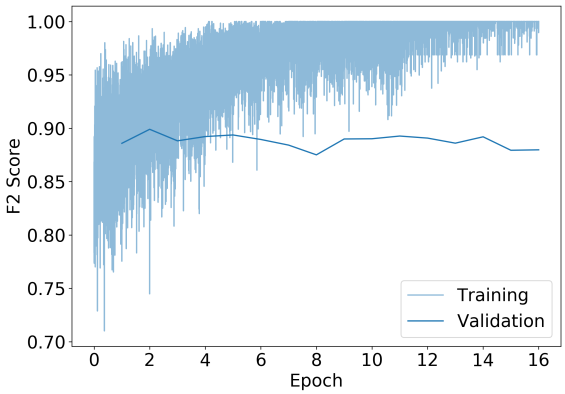


Figure 9: Pretrained ResNet101, training and validation accuracy during fine-tuning of all model weights

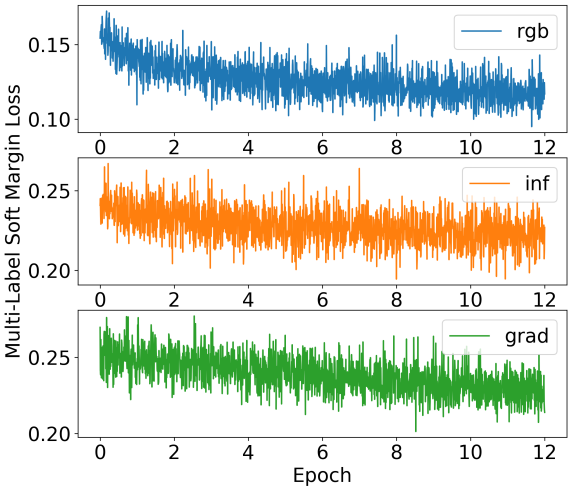


Figure 10: Loss for the Triple ResNet18 individual models as a function of time.

Pretrained ResNet101

Our next strategy was to pursue the same approach described above, but applied to a deeper model: ResNet101. We fine-tuned a pretrained ResNet101 using the same training hyperparameters developed for ResNet18 but with a smaller batch size of 32 instead of 256. The reason for the smaller batch size was simply to decrease memory usage: the deeper model has many more parameters per image, so it can only handle a smaller number of images at a time. Figure 9 shows the evolution of the F2 score accuracy during training. This model shows clear over-fitting: the training F2 score increases to nearly 1.0, while the validation F2 score hovers slightly below 0.90. To counteract the over-fitting, we adopted data augmentation and L2 regularization as described in the Methods section. The remaining models described in this report use these techniques by default. As the subsequent sections show, these two strategies immensely helped to solve our over-fitting problem.

Pretrained Triple ResNet18

Another strategy we pursued for improving on our initial results was to leverage the multiple data sets available for training in this challenge (data from .jpg and .tif versions of the files) and create an ensemble of separately-trained ResNets. To do so, we individually trained three ResNet-18's on different data sets and combined their scores to make our predictions. One ResNet was trained on the RGB channels ('rgb'), another on the near-infrared channel mapped to 3 color bands ('inf'), and a third on grayscale, ∇_x, and ∇_y ('grad'). We elected to train a separate model for each, rather than combine all the channels into the input for a single model so that we could continue to work with pretrained ResNets rather than training from scratch.

Figure 10 shows the loss during training for the three individual models. Note that loss is much higher for the 'inf' and 'grad' datasets than for 'rgb'. This is because the training labels were created by humans using the 'rgb' data and so the 'rgb' data corresponds more faithfully to the labels identified in the images. Figure 11 shows the accuracy for the three individual models during training. All three models do not overfit the training data. As expected, the accuracy is much higher for the 'rgb' model than the 'inf' and 'grad' models.

As summarized in Table 1, the Triple ResNet18 model gives the strongest performance on the test set of any of our models: test F2 of 0.918. To achieve this, we maximize the F2 score by optimizing the weights used when combining the scores from the three models as described in the Methods section. We also optimize the cutoffs for the sigmoid function after combining the scores from the three models as described in the Methods section. Interestingly, the score of the combined model is higher than the score any of its individual components.

ResNet18 from scratch

Having explored a variety of approaches using pretrained ResNets, and since some studies find that training from randomly initialized weights gives better results than fine-tuning pretrained models,[5] we next investigated the performance of a ResNet18 trained from randomly initialized weights. The ResNet was trained on the 'rgb' dataset for 30 epochs using a weight regularization of 1e-3 and a learning rate initialized at 1e-3, which adaptively decreases when loss stagnates. Figure 12 shows the loss during training for this model. The role of the adaptive learning rate can be

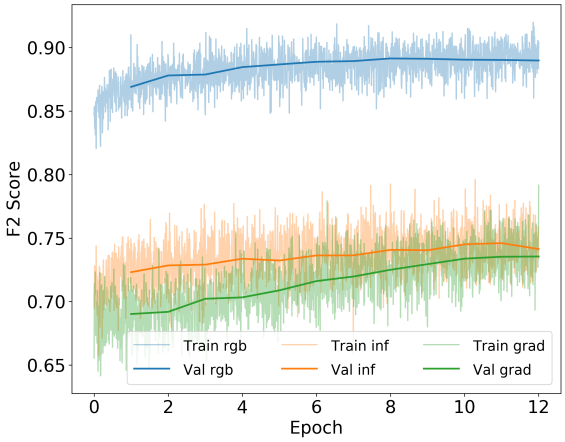


Figure 11: Accuracy (F2 score) results for the Triple ResNet18 individual models on the training set and the validation set as a function of epoch. The training accuracy was plotted at the end of each batch of data, and the validation accuracy was plotted at the end of each epoch.

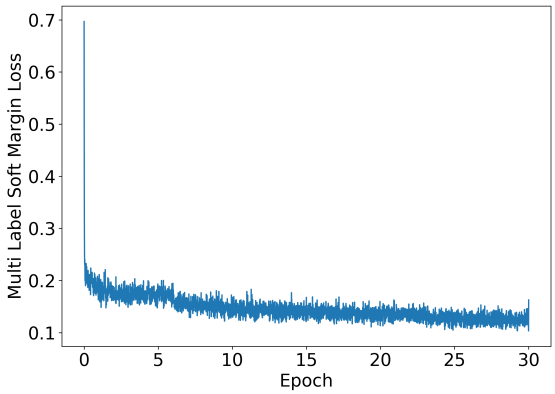


Figure 12: ResNet18 from scratch, loss during training

seen in the sharp jumps down in loss that occur at the end of stagnant periods (for example, at the end of epoch 6). Figure 13 shows the evolution of the F2 scores on the training and validation data. The model does not overfit the training data, at the end of 30 epochs the validation accuracy is still roughly the same as the training accuracy. There are several instances where the validation accuracy sharply decreases and then recovers (for example, at epoch 5). These features do not appear when fine-tuning the pretrained models. This is likely because when the pretrained models are fine-tuned, they are already near a local minimum in the new loss landscape. However, the models trained from randomly initialized weights must traverse the high-dimensional loss landscape from a random point to the best minimum that the optimizer can find. Along that path, there are some points that have very high generalization error, but in the end, the model converges to a minimum with low generalization error.

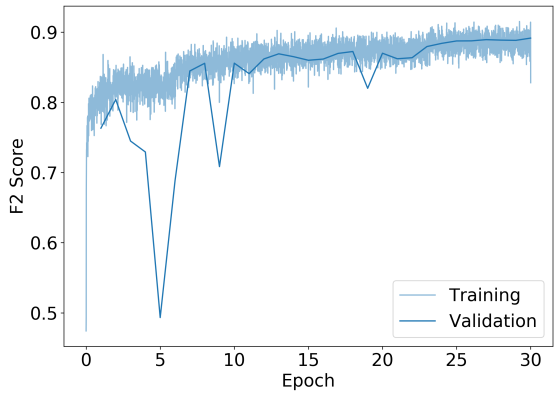


Figure 13: ResNet18 from scratch, accuracy (F2 score) results for the training and validation datasets

6. Conclusions and Future Work

In this project, we trained convolutional neural networks to perform multilabel classification of land usage from satellite images of the Amazon Rainforest. We have trained a variety of networks including custom designed convolutional architectures, ResNets, and GoogleNets. Our highest performing model to date is an averaging of three ResNet18s, trained using transfer learning. This model achieves an F2 score of 0.918 on the test set. In our work so far, we largely focused on fine-tuning pretrained ResNets. In our future work, we intend to explore training the ResNet models from scratch. We intend to use new input data combining all seven of the unique channels available between the .jpg and .tif datasets. This should improve results as the network can learn weights to combine all of the available input data, whereas in our current highest scoring model the different bands of data are grouped into sets of three and their predictions are only combined after training. We also want to further explore model ensembles, which will help the overall model find a global minimum by averaging several models that are converged to nearby local minima in the loss landscape. The leaders of the Kaggle competition have an F2 score only about 0.015 than our current best model, so we hope that the proposed future work will be able to bring our model closer to the state-of-the-art.

References

[1] About the amazon. *World Wildlife Fund*, 2017. http://wwf.panda.org/what_we_do/where_we_work/amazon/about_the_amazon/.

[2] Understanding the amazon from space. *Kaggle*, 2017. https://www.kaggle.com/c/planet-understanding-the-amazon-from-space.

[3] A. Albert, J. Kaur, and M. Gonzalez. Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale. *CoRR*, abs/1704.02965, 2017.

[4] M. Bober-Irizar. Data exploration analysis. https://www.kaggle.com/anokas/data-exploration-analysis, 2017.

[5] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *CoRR*, abs/1508.00092, 2015.

[6] G. Fu, C. Liu, R. Zhou, T. Sun, and Q. Zhang. Classification for high resolution remote sensing imagery using a fully convolutional network. *Remote Sensing*, 10(5):498, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] F. Hu, G.-S. Xia, J. Hu, and L. Zhang. Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. *Remote Sensing*, 7(11):14680–14707, 2015.

[9] J. Johnson. pytorch-finetune. https://gist.github.com/jcjohnson/6e41e8512c17eae5da50aebef3378a4c, 2017.

[10] J. Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.

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

[12] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov. Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, 14(5):778–782, 2017.

[13] J. Liu. pytorch-lr-scheduler. https://github.com/Jiaming-Liu/pytorch-lr-scheduler, 2017.

[14] F. P. S. Luus, B. P. Salmon, F. van den Bergh, and B. T. J. Maharaj. Multiview deep learning for land-use classification. *IEEE Geoscience and Remote Sensing Letters*, 12(12):2448 – 2452, 2015.

[15] Z. Ma, Z. Wang, C. Liu, and X. Liu. Satellite imagery classification based on deep convolution network. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 10(6):1155–1159, 2016.

[16] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. Convolutional neural networks for large-scale remote-sensing image classification. *IEEE Geoscience and Remote Sensing Letters*, 55(2):645 – 657, 2016.

[17] D. Marmanis, M. Datcu, T. Esch, and U. Stilla. Deep learning earth observation classification using imagenet pretrained networks. *IEEE Geoscience and Remote Sensing Letters*, 13(1):105 – 109, 2016.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] PyTorch. Pytorch. https://github.com/pytorch/pytorch, 2017.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[21] G. J. Scott, M. R. England, W. A. Starms, R. A. Marcum, and C. H. Davis. Training deep convolutional neural networks for landcover classification of high-resolution imagery. *IEEE Geoscience and Remote Sensing Letters*, 14(4):549–553, 2017.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[24] Q. Weng, Z. Mao, J. Lin, and W. Guo. Land-use classification via extreme learning classifier based on deep convolutional features. *IEEE Geoscience and Remote Sensing Letters*, 14(5):704–708, 2017.

[25] M. Xie, N. Jean, M. Burke, D. Lobell, and S. Ermon. Transfer learning from deep features for remote sensing and poverty mapping. *CoRR*, abs/1510.00098, 2015.

[26] J. Yue, W. Zhao, S. Mao, and H. Liu. Spectralspatial classification of hyperspectral images using deep convolutional neural networks. *Remote Sensing Letters*, 6(6):468–477, 2015.

[27] W. Zhao, Z. Guo, J. Yue, X. Zhang, and L. Luo. On combining multiscale deep learning features for the classification of hyperspectral remote sensing imagery. *International Journal of Remote Sensing*, 36(13):3368–3379, 2015.

[28] Y. Zhong, F. Fei, Y. Liu, B. Zhao, H. Jiao, and L. Zhang. Satcnn: satellite image dataset classification using agile convolutional neural networks. *Remote Sensing Letters*, 8(2):136–145, 2017.