Predicting Land Use and Atmospheric Conditions from Amazon Rainforest Satellite Imagery

Rohisha Adke *
Stanford University
radke@stanford.edu

Joe Johnson *
Stanford University
jjohnso3@stanford.edu

Abstract

We apply convolutional neural networks to predict land cover and atmospheric conditions from satellite images of the Amazon rainforest. We experiment with CNNs, transfer learning, ensemble models, and a ResNet model. We find that an 8-layer CNN performs the best, and achieves a validation set accuracy of 95.8% and test F_2 score of 0.9077. ResNet and the ensemble model also perform near the level of the CNN.

1. Introduction

Deforestation in the Amazon Basin is happening at an alarming rate - and currently, governments and local stakeholders do not have enough information to diagnose and address the situation. With our project, born out of the "Planet: Understanding the Amazon from Space" Kaggle Competition, we label satellite imagery with atmospheric conditions, type of land cover, and land use occurring. With our algorithm, officials will be able to use satellite imagery to characterize global deforestation.

The input to our algorithm is a satellite image of the Amazon rainforest. We use a CNN to output multiple predicted land cover and atmospheric condition labels for each image.

2. Related Work

In recent years, a number of papers related to deep learning on satellite imagery have emerged. Researchers have created benchmark datasets, feature extraction and data augmentation methods, benchmarked pretrained networks used for transfer learning, and recently, proposed new architectures specifically for satellite imagery.

2.1. Related Large Dataset

[1] introduces two large satellite imagery datasets - the SAT-4 and SAT-6 airborne datasets (DeepSat) - which have been used for research related to our project. Together, the datasets contain about 1 million images encoded as Matlab (.mat) files. Each sample image is 28x28 pixels, with four channels (red, green, blue, near infrared). Image labels are 1x4 and 1x6 (SAT-4 includes barren land, trees, grassland, and other; SAT-6 includes barren land, trees, grassland, roads, buildings, and water bodies), with a single one value in each vector to indicate the correct class label. While the images contained in these datasets are more similar to the images we work with than others, such as the ImageNet dataset, the DeepSAT dataset labels are more simple than those we work with, and the images are smaller than our raw input images.

2.2. Feature Extraction

[2] proposes different techniques for feature extraction from satellite imagery. These include features based on SIFT, co-occurrence kernels, bag-of-visual-words, spatial partitioning via various unsupervised and clustering methods, and color histograms.

2.3. Data Augmentation

[16] introduces a data augmentation method for enhancing remote sensing datasets using flipping, translation, and rotation. We use some of these methods to augment and expand our dataset of Amazon rainforest satellite imagery for use in training the final layers on top of the pretrained network we use for transfer learning.

2.4. Existing CNN architectures for classifying satellite imagery

A number of papers in the last year or two have experimented with using CNN architectures for land use classification on DeepSat [1] [2] [8] [9] [10] [17] and the UC Merced Land Use [15] dataset.

[1] applies a convolutional neural network to classify barren land, trees, grassland, roads, buildings and water coverage from satellite imagery. For our baseline model we take a similar approach, using a CNN to classify an image into multiple potential groups.

[1] Both authors contributed equally to this work.

2.5. Transfer Learning

Recently, transfer learning and pretrained models have become popular tools for land use classification. [11] shows that models pretrained on images of everyday figures, which are very dissimilar from land cover images, can still achieve state of the art results on the UC Merced Land use dataset.

Similarly, [2] evaluates CaffeNet and GoogLeNet [13] CNN architectures on the UC Merced Land Use and Brazilian Coffee Scenes datasets; they find that using a pre-trained GoogLeNet with fine-tuning on the two datasets yields an accuracy of 97.10%.

[9] uses the trained model Overfeat (an improved version of AlexNet) and a custom CNN component to classify images in the UC Merced Land Use dataset with an accuracy of 92.4%. The custom component accepts the derived 2D features and class labels into two convolutional layers, two fully connected layers, and a Softmax classifier trained with SGD with momentum. The custom component also uses max-norm, dropout, weight-decay, and data augmentation.

Instead of using the UC Merced dataset, [8] uses a DCNN based on Inception modules (with inspiration from GoogLeNet), with hyperparameters chosen using a genetic algorithm, to classify DeepSat. The model attains 98.4% accuracy on SAT-4, and 96.0% on SAT-6.

[10] uses the deep learning library of Torch to benchmark a number of existing networks and methods on DeepSat at once: AlexNet [5], AlexNet-small, VGGNet [12] (with horizontal and vertical flips for data augmentation), Deep Belief Networks, Stacked Denoising Autoencoders, and other semi-supervised frameworks. [10] found that Alex-Net and VGGNet achieved classification accuracy rates above 99.9%.

[14] uses transfer learning on a CNN similar to AlexNet with the UC-Merced land use data. The CNN extracts features of the images, then an additional "Extreme Learning Machine" component completes image classification. Their model achieves 95.62% accuracy on the data set. This suggests some benefit to trying out various schemes for fine-tuning neural network models to our data.

2.6. Novel Architectures

[17] proposes SatCNN, an agile CNN architecture specifically for high-spatial resolution remote-sensing images, as opposed to borrowing from natural image scene classification or other CNN architectures. This architecture involves deeper convolutional layers with smaller filters than the previously described approaches, and achieves 99.65% and 99.54% accuracies on DeepSat. Features of the network include data normalization, convolutions, ReLUs, pooling, fully connected layers, dropout, and the cross-entropy loss function optimized by SGD with momentum.

2.7. Ensemble Models

Ensemble models have also proven useful for land cover classification. Early work has used bagging, boosting and AdaBoost methods to classify satellite images. [6] shows that when applying this approach to landsat images with radial basis function neural networks, the ensembles outperform any individual classifier. [7] uses multi-layer perceptrons, CNNs, random forests, and ensembles of CNNs to classify land cover and crop use on satellite images. They find that the ensemble of CNNs performs the best on the dataset, and achieves 85% accuracy on the crop data.

The prior body of work that successfully uses CNNs on land cover data motivates our decision to use neural network models for this task. The success of transfer learning, state-of-the-art models, and ensemble models motivates us to try similar approaches on our dataset.

3. Dataset and Features

Our dataset contains 256x256 pixel images of the Amazon basin, retrieved from Kaggle.com [4]. The original dataset (which was divided into easy and hard to identify labels) contained over 150,000 images. Each image is labeled with an atmospheric condition (clear, partly cloudy, cloudy, and haze), and land cover/use observed (types of rainforest, agriculture, rivers, towns/cities, and roads). There are multiple possible covers/uses for each land area. There are 17 total classes. 3.

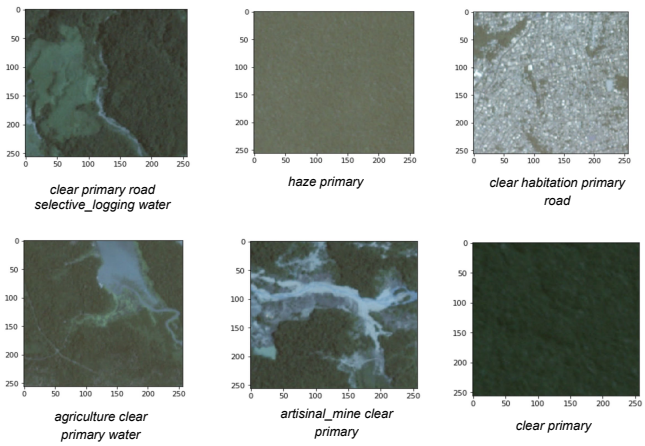

Figure 1. Sample images and their labels.

We divide the dataset into 35,000 training, 5,479 validation, and 61,191 test images.

We augment the data by flipping the images, and by subtracting the per-color mean. For VGGNet transfer learning we crop the images to match the 224x224 initial size of the pretrained model.

For our networks besides those based on transfer learning, we scale the images to 32x32 pixels. Upon scaling,

individual images look like this:

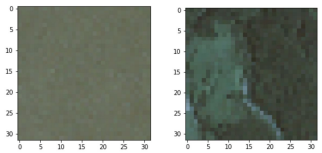

Figure 2. 32x32 scaled dataset images.

An important characteristic of the data is that the class distribution is heavily imbalanced. The most common class is primary forest, which consists of 92% of the sample. The least common class is conventional mine, which is only 0.25% of the sample. The label distribution is displayed in Figure 3.

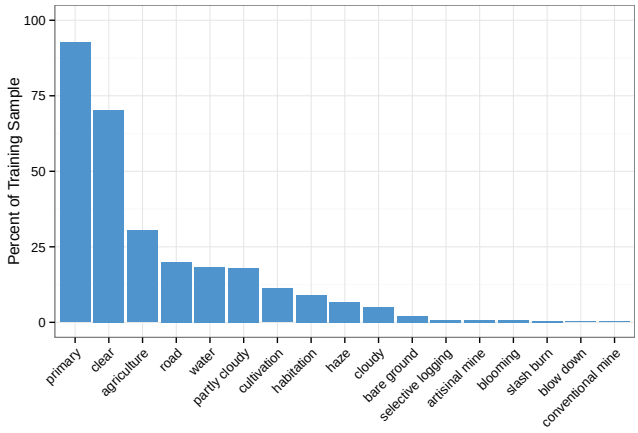

Figure 3. Label Distribution

4. Methods

4.1. Expected results & evaluation

This is a multi-label classification problem, so our expected result is a set of labels for each image indicating atmospheric condition and multiple possible land cover labels. We evaluate these predictions using the same evaluation criterion as the Kaggle competition, which is the F_2 score. The F_2 score is a combined measure of the precision and recall of the classifier, with recall receiving a higher weighting. This implies that we will consider false negatives, where we label a positive class as negative, more heavily than we weight false positives, where we label negative class as positive.

When working with a dataset with an imbalanced label distribution, it makes more sense to use evaluation measures such as recall, precision, and the F_2 Score instead of accuracy (if evaluating on accuracy, your classifier may learn to simply predict the dominant class for all images). The F_2 score is defined as below, and is equal to one if all images are classified correctly.

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

$$p = \frac{tp}{tp+fp}$$

$$r = \frac{tp}{tp+fn}$$

where p is the precision, r is recall, tp is the true positive rate, fp is the false positive rate, and fn is the false negative rate.

Since we are working on a multi-label classification, we use a sigmoid cross entropy loss function for each model. For n observations and K classes, the form of this loss is:

$$E = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K}[p_{n,j}\log\hat{p}_{n,j} + (1-p_{n,j})\log(1-\hat{p}_{n,j})]$$

Where $\hat{p}_{n,j}$ and $p_{n,j}$ are the estimated and true probabilities of class j. This treats each label as a separate classification problem, and, unlike the more commonly used cross-entropy loss, does not normalize the probabilities of each class to be equal to one. If a class probability is above a pre-specified classification threshold, then we label it as belonging to that class. For our CNN and ResNet models, we achieve good performance with a threshold of 0.20. For transfer learning we use a grid search to find the best threshold on the training data.

4.2. Network Architecture

4.2.1 Convolutional Neural Network

We use a standard CNN network as a baseline result. We stack sets of two convolutional layers, followed by max-pool layers, ReLu activation layers, and dropout layers. We repeat this between 2 and 4 times. At the final layer we have two MLP layers and a final sigmoid activation to output predicted probabilities for each class.

4.2.2 ResNet

ResNet is an architecture that supports the training of very deep neural networks [3]. This architecture improves on normal CNNs by using residual learning, where every three convolutional layers there is a residual layer that combines the output of the last layer and the output from three convolutional layers ago. The residual layer is

$$y = F(x, \{W_i\}) + x$$

where y is the layer output, x is the layer input, and $F(x, \{W_i\})$ is the residual mapping.

4.3. Pretrained VGGNet

We also experiment with fine-tuning a pretrained VGGNet model on our data. The VGGNet model was proposed by [12] and is similar to AlexNet, but has more layers and smaller convolutions. In its original application to the 2014 ImageNet challenge it achieved first place results in image localization. To fine-tune the model we replace the fully connected layers with various specifications to fit our problem and train on the new layers and fine-tune the VGGNet's convolutional layers.

4.4. ResNet-CNN Ensemble

We combine our two best performing models, ResNet and CNN-8. We combine these models by taking their predicted class probability estimates and averaging them to produce our predicted label probabilities.

5. Experiments

We begin by implementing three different neural network models: a baseline CNN, an 8-layer CNN, and a ResNet model. For each model we use random hyperparameter search to select the learning rate, batch size, and dropout probabilities if applicable. We use an Adam gradient update rule.

5.1. Baseline Model

As a baseline model, we run a standard CNN model directly on the training set. This consists of two sets of layers as described in the model architecture section, resulting in four convolutional layers.

This model achieves a validation accuracy of 94% and F_2 score of 0.86. While the accuracy is very high, this is partially due to class imbalance in the data, since primary rainforest is by far the most common type of forest cover. This indicates that there is still room to improve on this baseline, and we can improve performance by increasing the number of CNN layers.

5.2. Deeper CNN

We next implement a CNN model with 8 convolutional layers. We increase the number of filters for our model at each set of convolutional layers. We decay the learning rate every 15 epochs and implement early stopping. The hyperparameters we select via random search are 0.0013 learning rate, .0.295 dropout probability on the convolutional layers, 0.562 dropout probability on the fully connected layer, and 128 batch size.

This model substantially improves our performance above the more shallow model. This reaches a validation accuracy of 95.8%, and a validation F_2 score of 0.9072. The progress is displayed in Figure 4.

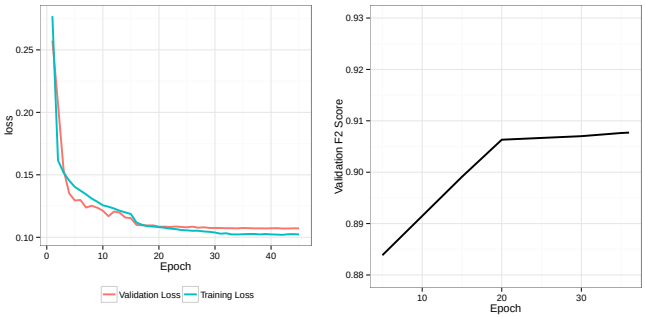

Figure 4. Training for 8 layer CNN

The validation and training loss are fairly close to one another, so while training loss is slightly lower than validation loss, overfitting does not appear to be a major problem.

This is our best performing model, so we also use it on our final test set. On the test set the model achieves a F_2 score of 0.9077, which is approximately the same as our CNN-8 validation result.

To get a better sense of how our model is performing, we break down our performance by class. Table 1 shows the accuracy and F2 score for each class:

Class	Accuracy	F_2-Score
agriculture	0.862	0.8827
artisinal mine	0.995	0.6904
bare ground	0.969	0.3053
blooming	0.993	0.0633
blow down	0.998	0.0
clear	0.939	0.975
cloudy	0.976	0.8705
conventional mine	0.998	0.0
cultivation	0.873	0.6543
habitation	0.901	0.5886
haze	0.951	0.781
partly cloudy	0.962	0.9293
primary	0.964	0.9903
road	0.86	0.7995
selective logging	0.991	0.1269
slash burn	0.993	0.0
water	0.865	0.7144

Table 1. Model performance by class.

The F_2 Score is very low for the very rare classes like blooming and selective logging. Blow down, slash-burn, and conventional mine receive F_2 scores of zero because they are not ever predicted. These classes do still have very high accuracy because they are so rare. The more common classes like primary, clear, and partly cloudy are well-identified. Water and road are two fairly common classes that our model struggles to classify.

Figures 5 and 6 display two images that our model mis-

classifies. 5 is labeled cultivation, haze, primary, and slash burn. Our model labels it as agriculture, clear, cultivation, and primary. Considering that agriculture and cultivation are pretty similar, our model seems to be picking up on some important characteristics. Slash-burn is one of the rare classes so it is not surprising that our model misses this class. Since our images are compressed to be 32x32 it may also be harder for our model to perceive the haze in the image. 6 is agriculture, clear, cultivation, primary, road, and water. Our model sees agriculture, cultivation, habitation, partly cloudy, primary, and road. The model correctly labels primary and road, but misclassifies the atmospheric condition as partly cloudy. This may be because some of the land area is colored white, and the model interprets this area as clouds. The bending shape and darker green color of the forest may also be misinterpreted by our model as a river, causing it to label the image as water.

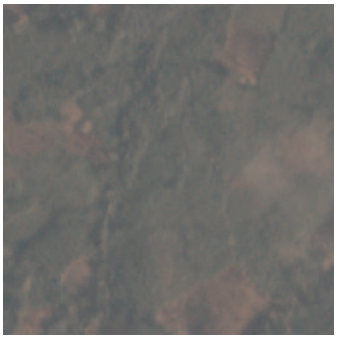

Figure 5. Misclassified Image

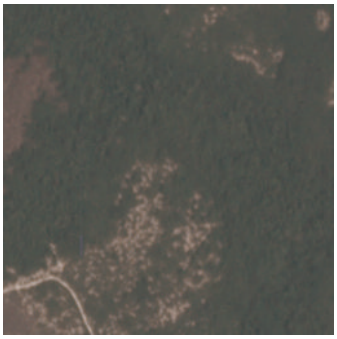

Figure 6. Misclassified Image 2

5.3. ResNet

We implement a ResNet model with 18 layers. We select a learning rate of 0.000085 and a batch size of 128 from random search.

This model does not improve upon the 8-layer CNN. This reaches a validation accuracy of 95.3%, and an F_2 score of 0.89.3. The progress is displayed in Figure 7. The model tends to overfit the data more than the 8-layer CNN.

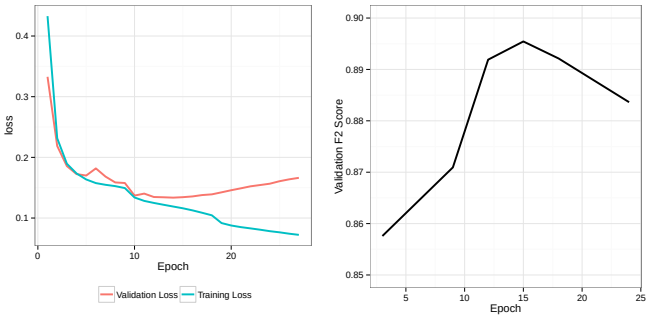

Figure 7. Training for ResNet.

The CNN-8 model has dropout so it is more heavily regularized than our ResNet model which does not have dropout and is more complex, so this result is not surprising. While it is not implemented here, Stochastic Depth may help with this problem. We use early stopping to save and use the model before significant overfitting has occurred, so this should mitigate some of the problem.

5.4. Model Ensemble

The model ensemble performs well, but is slightly below that of the CNN-8 model. It achieves a validation set F_2 score of 0.904.

5.5. VGGNet-16 Transfer Learning

Noting the successful usage of transfer learning in a number of related papers on land use classification, we experiment with several network architectures involving transfer learning from the pre-trained VGGNet-16 (taken from TensorFlow's contrib.framework model zoo). We choose to train on top of VGGNet because VGGNet has been successfully used on land-use classification problems in the past, and the input size to the VGGNet architecture is close to that of our satellite images.

For our architectures involving transfer learning we experimented with various hyperparameters:

- Learning rates on the final layers of the pretrained model or extra layers: 1e-3, 5e-5 - Learning rates for training the entire network (pre-trained and additional components) for a few epochs: 1e-5, 1e-6, 1e-7 - Dropout probabilities for dropout layers: 0.4, 0.5, 0.3, 0.25 - Weight decays: 1e-4, 5e-4

For the most part, with the architectures listed below, we found that the following worked best: - Learning rate on final components: 1e-3 - Learning rate on entire network: 1e-5 - Dropout probability: 0.4 - Weight decays: 1e-4

The five main architectures we experimented with were:

1. Retraining the final fully-connected layer of the VGGNet on our dataset of 17-class labels. After training both stages of the model (final layer and entire network) for 5-10 epochs (where one epoch involves passing the entire

training dataset through the network) each, our network had an F_2 score of around 0.55-0.65, depending upon the iteration. We found that scores jumped around quite a bit during training, depending upon the labels of images that were in a specific batch. Generally, we found that only retraining the final fully-connected layer of the VGGNet was not enough to bridge the gap between the data the model had been pretrained on and our dataset.

2. VGGNet with final two layers replaced by four fully-connected layers of decreasing size. After observing the results of the first transfer learning architecture, we added additional fully-connected layers to the end of the network, with numbers of neurons that decreased to our final 17 classes. We trained the final four layers for 5-10 epochs, then the entire network for a few additional epochs. This architecture yields an improved F_2 score of 0.64 - 0.69. However, we hypothesized that fully-connected layers were not able to learn meaningful features from our data, or there were too many parameters to optimize, and the network was falling into local optima that did not perform well on the validation set.

3. Training the final four layers of the VGGNet on our dataset. We try an architecture that involves retraining the final four layers of the VGGNet on our dataset. We observe a loss function that doesn't bounce around as much as others do during training, but a similarly low F_2 score compared to the previous standalone models.

4. Removing the final two layers of the VGGNet, and adding a 5-convolutional component onto the pretrained network. In order to assist our network in capturing nonlinear patterns amongst pixels in the data, we removed the final two layers of the VGG network, then added a network component with: a fully-connected 4096-unit layer, batch normalization, a convolutional layer with 32 5x5 filters, leaky ReLu activation, max pool, a convolutional layer with 64 5x5 filters and leaky ReLlu, max pool, a fully connected layer, then a final fully connected layer with sigmoid activation. We saw an improvement in our loss function during training with this network (below); the loss function decreased for the first part of training. However, after around 1000 iterations, we stopped seeing much improvement in the loss, indicating that either the capacity of the network to learn was exhausted, or the network had found a local optima in the loss landscape. With this architecture, we saw an F_2 score around 0.66-0.71.

5. Removing the final two layers of the VGGNet, and adding our best-performing CNN-8 as a component onto the end of the pretrained network. After the success of our deeper CNN-8 network, we experimented with adding the CNN-8 as an additional component after the pre-trained VGGNet. With this architecture, we feed the output of the VGGNet's fc6 layer into the following layers: batchnorm, two convolutional layers, max pool, dropout, two convolutional layers, max pool, dropout, fully connected, batchnorm, dropout, fully-connected. The dropout probabilities are 0.25, 0.25, and 0.5, the convolutional layers involve 32 filters and 3x3 kernels with ReLu activation, and the max pool layers have 2x2 kernels. With this architecture, we saw F_2 scores around 0.67. The biggest advantage with this network was that the optimal threshold for our logit to class 0/1 indicator was 0.5, which indicated that the batch normalizations were helping calibrate the network and avoid saturated sigmoids and/or dead ReLus in each layer. Below, we see that the loss function during training is steadily decreasing - perhaps, with more extensive hyperparameter tuning, and more time, the network would reach similar or better performance than the CNN-8 alone.

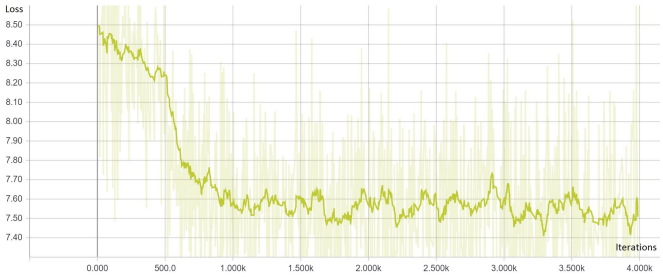

Figure 8. Iteration vs Loss for VGGNet + CNN-5 Unit

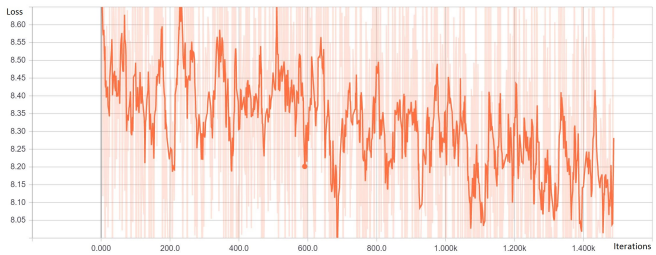

Figure 9. Iteration vs Loss for VGGNet + Best-performing CNN-8

Below, we show a comparison of the training losses for our different transfer learning architectures. We see that the networks involving adding fully connected layers to the end of the network, or training the final four layers on our dataset, have lower starting losses than others (lower red/orange lines), but, lacking flexibility in their fully-connected layers, do not decrease across iterations as compared to the others. Architecture 4, with five layers of CNNs and other layers, has the steepest decrease in loss, but then plateaues (yellow/green middle lines). Finally, architecture 5, with our best performing CNN component added to the pretrained network, steadily decreases in loss, and may work well given enough time to train, or with an increased learning rate.

We hypothesize that transfer learning using VGGNet didn't work as well as standalone CNN and other architectures with our dataset for several different reasons. Firstly,

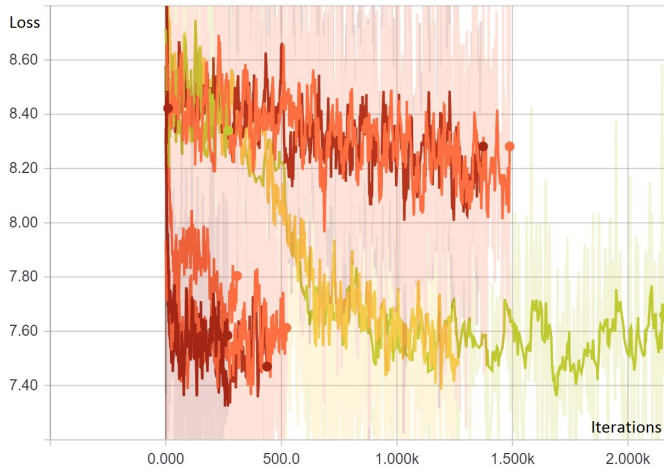

Figure 10. Iteration vs Loss for Transfer Learning Architectures

VGGNets are complex networks with many parameters to train. This means that we may need more time, and concurrent hyperparameter tuning, to tune the network. In fact, existing papers that use VGGNets for transfer learning train for 100 epochs, while we trained for a maximum of 30 epochs. A related concern is that the VGGNetwork is trained on a dataset that is too different from our dataset to transfer well without significant time spent training the base pretrained network component. In this case, pretraining existing architectures on large datasets like DeepSAT may improve performance. It may be the case that using large 224x224 images made it difficult for the network to separate signal from noise (as compared to using 32x32 images in our standalone networks). Perhaps using a different optimizer to avoid the network becoming stuck in local optima, or better hyperparameter tuning, would improve the F_2 score.

6. Conclusions and Future Work

We test several different CNN model architectures, and ultimately a relatively simple 8-layer CNN performs best. It achieves an F2-score of approximately 0.907 on the validation and test sets. A simpler model may perform well here because we only have 17 classes, images are similar to one another, and there are fewer distinct features in the images. Transfer learning may not have worked as well because the ImageNet images it was trained on are very different from our satellite images, and because the complex VGG network requires significant time and tuning to train well.

The ensemble model also seems promising as its performance is very close to the 8-layer CNN. Adding more models and choosing a more principled approach to weighting each model may further improve its performance.

Future work could include trying out a wider variety of pre-trained models. Models trained on the DeepSAT datasets, which are more similar to our dataset than ImageNet, may be better able to distinguish relevant features in our images. Other architectures, like a pretrained ResNet or GoogleNet, may also work better than the VGGNet.

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. 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.

[3] 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.

[4] Kaggle.com. Planet: Understanding the amazon from space, 2017.

[5] 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.

[6] S. Kulkarni and V. Kelkar. Classification of multispectral satellite images using ensemble techniques of bagging, boosting and adaboost. In *Circuits, Systems, Communication and Information Technology Applications (CSCITA), 2014 International Conference on*, pages 253–258. IEEE, 2014.

[7] 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.

[8] Z. Ma, Z. Wang, C. Liu, and X. Liu. Satellite imagery classification based on deep convolution network. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 10(6):1113–1117, 2016.

[9] 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.

[10] M. Papadomanolaki, M. Vakalopoulou, S. Zagoruyko, and K. Karantzalos. Benchmarking deep learning frameworks for the classification of very high resolution satellite multispectral data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 83–88, 2016.

[11] 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.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[14] 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.

[15] 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.

[16] X. Yu, X. Wu, C. Luo, and P. Ren. Deep learning in remote sensing scene classification: a data augmentation enhanced convolutional neural network framework. *GIScience & Remote Sensing*, pages 1–18, 2017.

[17] 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.

7. Base Code References

Data Loading: `https://www.kaggle.com/anokas/simple-keras-starter`

ResNet : `https://github.com/raghakot/keras-resnet`

CNN: `https://github.com/EKami/planet-amazon-deforestation`

Transfer Learning: `https://gist.github.com/omoindrot/dedc857cdc0e680dfb1be99762990c9c`