Understanding the Amazon from Space

Yiqi Chen Fanming Dong Chuanwei Ruan

Abstract

Kaggle recently released a data challenge which aims to classify various phenomena of interest (atmospheric conditions, land cover phenomena) in the Amazon basin from a provided dataset of satellite images. In this work, we intend to tackle this challenge by experimenting with different convolutional neutral network (CNN) architectures, including VGG, ResNet and DenseNet, as well as various optimization approaches. Our best model achieves 0.93006 F2-score on the test dataset, ranked 22/438 as of 6/12/2017.

1. Introduction

The Amazon rain-forest is the largest tropical rain-forest in the world with the biggest biological diversity. However, the deforestation of Amazon rain-forest has accelerated since 1991[1]. In order to further understand the status quo of Amazon surface, and potentially figure out how to respond to the deforestation, Planet, a company that builds and launches the largest constellation of Earth-imaging satellites in the world, recently released a data challenge over Kaggle[1]. In this challenge, we will classify atmospheric conditions and various land cover phenomena from a dataset of satellite images taken in the Amazon basin, and try to achieve the highest accuracy on the provided test dataset.

We will use convolutional neural network[2] (CNN) models for this competition as they have been demonstrated to be very successful in image classification tasks since the AlexNet model[3] was introduced in the ImageNet competition[4] in 2012. We fine-tuned 3 pre-trained CNN architectures that have achieved great accuracy in the ImageNet competition[4]: the VGG model[5], the ResNet model[6], and the DenseNet model[7]. Among the 3 architectures, VGG network is a simple feed-forward convolutional neutral network, whereas ResNet and DenseNet allow values in lower level layers to directly connect to higher level layers.

Furthermore, we did various optimizations to boost the accuracy of our classification. We preprocessed the image data through normalization and augmentation in order to prevent overfitting. We carefully tuned when to enable weight update, and when to adjust learning rate in order to reduce the loss. Also, we fine-tuned the probability threshold for prediction to better fit our evaluation metric.

[1] `http://rainforests.mongabay.com/amazon/deforestation_calculations.html`

2. Related Work

Previous methods to tackle the classification of satellite images include statistical methods such as decision-trees and SVM. Baker et al.[8] and Otukei et al.[9] applied gradient boost trees and SVM on satellite images in order to classify land coverage. Their studies are similar to ours and they achieve about 85% accuracy on the test datasets. However, their models lack the ability for computers to learn features automatically. These statistical methods rely on hand-crafted features which can be improved by allowing computers to learn the feature representation.

As mentioned in the introduction, we adopted VGGNet model[5], ResNet model[6], and DenseNet model[7]. The VGGNet model consists of several combinations of filters and pooling layers. Compared to AlexNet[3], the VGGNet model is deeper which allows more non-linearities and the filters are smaller but as effective. The ResNet model is a very deep model using residual connections. The residual connection architecture allows ResNet to be able to train very deep without degrading and thus allows the model to have very good results in terms of accuracy. The DenseNet model is a relatively new model. Its architecture is made up of blocks that are connected to every other layers in the model. It alleviates the problem of vanishing gradients and reuses features in the model.

In our study, we also take inspirations from many papers about the tricks to improve training accuracy. As Jacobs pointed out, the learning rates should be able to vary over time in order to ensure faster convergence[10]. Donahue et al.[11] and Razavian et al.[12] pointed out that model weights trained on different tasks are transferable and are able to achieve faster convergence by using pre-trained weights. In Kingma et al.'s paper[13] proposed the Adam optimizer. As they note, the Adam optimizer is computational-efficient, memory-efficient, and suitable for large datasets. Krizhevsky et al. used data augmentation

when they trained the AlexNet and found data augmentation to be helpful in getting more accuracy in results. As Ding et al.[14] also pointed out, data augmentation works great for target recognition in difficult conditions such as random noise, and random missing information. Last but not least, Zhang el al.[15] noted that neural network ensemble models perform very well on scene classification . By creating an ensemble, the result is able to get higher score in accuracy.

There are many other methods to approach satellite image classification. Chen et al.[16] proposed a hybrid deep neural network to do small object detection. It separates the maps of the last convolutional layer and the max-pooling layer of deep neural network to extract features of different scales. Quintano et al.[17] proposed fractional type convolution filtering. It can improve satellite image classification by applying filtering algorithms as pre-classification.

3. Background

In this section, we describe in detail on important information about how we set up the classification problem.

3.1. Dataset

According to description from the Kaggle website[1], the images come from Planet's Flock 2 satellites in both sun-synchronous orbit (SSO) and International Space Station (ISS) orbit. The images were taken from January 1st, 2016 to February 1st, 2017, and all of them come from Amazon basin.

Each raw image is 256×256 in size. Depth wise, different from ordinary JEPG image dataset which contains 3 channel (RGB), the satellite image dataset also uses the GeoTiff format and contains one additional channel: near infrared, which may give us additional information that is not visible. However, we only use the JEPG image format in this project because the GeoTiff format does not show promising results according to the discussion groups on Kaggle[1].

Two sample images in the dataset are shown in Figure 1.

(a) Labels: clear, primary, road, water

(b) Labels: clear, primary, road, agriculture

Figure 1: Two sample satellite images in the training dataset

3.2. Image Labels

From Kaggle[1], the image labels can be broken down into three groups: "atmospheric conditions, common land cover/land use phenomena, and rare land cover/land use phenomena". Labels of each category and their frequencies in the dataset are shown in Figure 2. Each image should have one atmospheric label, and may have any number of land use labels. Also, if the atmospheric label of an image is 'cloudy', then it should not have any land use label.

3.3. Training and Test Set

Kaggle[1] provides two datasets for this project: one training set (40,480 samples) that includes correct labels, and one test set (61,192 samples) whose labels aren't released. We split the provided training set such that 1/10 of the dataset can be used for validation.

3.4. Evaluation

As specified in Kaggle[1], we evaluate the accuracy of the dataset by using average F2-score of each image. Each image has multiple labels and has its own F2-score. F2-score is defined as the following:

$$F_2 = 5 \cdot \frac{\text{precision} \cdot \text{recall}}{4 \cdot \text{precision} + \text{recall}}$$

Note that F2-score weighs recall higher than precision, and we have a dedicated technique to optimize for this weighting which will be discussed in the future sections.

4. Approach

We experimented with multiple state-of-the-art CNN architectures and optimizations that are known to be very helpful in training a neutral network. Section 4.1 talks about the architectures and Section 4.2 talks about our optimization approaches.

4.1. Architectures

4.1.1 VGG

We decided to choose the VGGNet[5] as our starting model given its robust performance and relatively simple architecture. Specifically, we used the VGG-16[18] model. It has 16 layers, in which 13 are convolutional layers, and the rest are fully connected layers.

The VGGNet has achieved great accuracy on ImageNet. However, unlike ImageNet, we only have around 40,000 images in our training data. The size of our data might be too small to train the VGGNet from scratch. We decide to use pre-trained VGGNet and keep the weights of the convolutional layers fixed and modify the last fully connected layer to fit our data. As transfer learning has been proved to be a very successful method in applying CNN

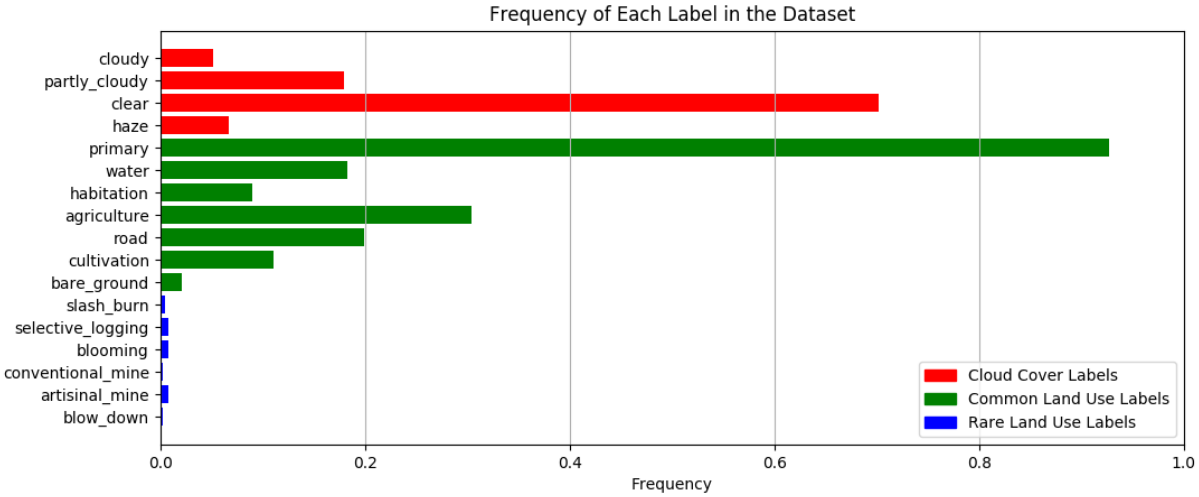

Figure 2: Frequency of Labels

models[14], we believe we can borrow the strength from other large dataset by fine-tunning the VGGNet.

We firstly fixed the convolutional layers and fine-tuned the two fully connected while the weights of the last fully connected layer were not used. The intuition is that the convolutional layers close to the input extract the low-level image features while the fully connected layers extract more tasks specific features. The satellite images used in our project are quite different from the images in ImageNet, which mainly consists of the objects in daily life. Thus, the high-level information in fully connected layers might be irrelevant to our task while the low-level image features might still be very useful. To add more representation ability, we also tried setting the higher-level convolutional layers to be learnable and assigning lower layers with lower learning rates. We hope the high level layers can quickly learn the high-level features representations without disturbing the low-level layers.

The VGG-16 model implementation and the pre-trained weights on ImageNet[3] are downloaded from Tensorflow-Slim[18].

4.1.2 ResNet

We also applied the ResNet-50 model[18]. It has 50 layers and the last layer is a fully connected layer.

We tried two approaches with the ResNet model. One is to use the pre-trained weights and the other one is to train the ResNet model from scratch. Out of the pre-trained weights, we throw out the logits and bias of two layers because the pre-trained weights in the slim model do not have them. However, the result from our pre-trained weight ResNet model does not perform very well. The other approach is to train the ResNet model from scratch. This does not intuitively make too much sense because we only have a limited amount of data. The final result also does not look good either.

The ResNet-50 model implementation and the pre-trained weights on ImageNet[3] are downloaded from Tensorflow-Slim[18]. The ResNet-50 model structure is also referenced from Tensorflow-Slim[18].

4.1.3 DenseNet

In this task, the prediction labels are of very different scales and semantics. We think it would be beneficial to use an architecture that allows final classifiers to easily access information from different layers. For example, features from the early layers might be enough to predict the cloud conditions and primary forest. The DenseNet [7] is an ideal choice for this purpose. The DenseNet consists of stacks of dense blocks. Each dense block is a stack of convolutional layers. In the dense block, the feature map learned in previous layers are concatenated to the channel dimension as shown in Figure 3. This design allows higher layers to access all information from previous layers directly.

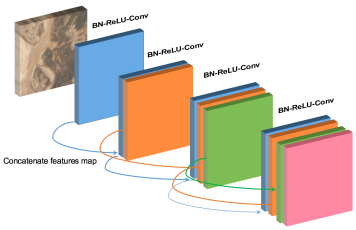

Figure 3: Simple Illustration of a Dense Block

For our project, we first implemented a DenseNet with 40 layers in TensorFlow and trained it from scratch. Because the performance was not ideal we then use a pre-trained DenseNet121-BC model. This model has 121 layers. "BC" means that the model uses Bottleneck layers and has a compression rate larger than 0 as described in [7]. The Bottleneck layer and compression used here are to improve computational efficiency.

The weights and the model implementation of the pre-trained DenseNet121-BC model on ImageNet[3] are downloaded from Github [2] using Keras[19].

4.2. Training Optimizations

4.2.1 Loss Function

In this project, we tried two loss functions:

1. Sigmoid Loss

   In this multi-label classification problem, we treat each label as a binary label and compute its sigmoid cross entropy loss independently. And overall loss of an image is the average loss over all labels. In particular, the following equation is applied to compute the loss:

$$L^{(i)} = \frac{1}{K}\sum_{k=1}^{K} y_k^{(i)} \text{score}_k^{(i)} - \log(1 + e^{\text{score}_k^{(i)}})$$

2. Sigmoid+Softmax Loss

   Even though there may be more than 1 true label among the 17 total labels, there can be 1 and only 1 true cloud cover label out of 4 possible cloud cover labels for each image. In order to optimize for these constraint, we apply the one-hot softmax cross entropy loss on the 4 cloud cover labels, and apply the same sigmoid cross entropy loss on the rest 13 land use labels. Since our evaluation metric, F2-score, weighs each label equally, we put weight of 4/17 on the softmax loss and 13/17 on the sigmoid loss.

4.2.2 Optimizer

We choose the Adam[13] optimizer with learning rate decay for our problem. The Adam optimizer combines the properties of momentum as well as AdaGrad[20]. Adam has been proven to be effective in many situations. Given the limit of time and resources, we use Adam as the default optimizer without experimenting other optimization techniques.

Also, we tuned the decay rate in the Adam optimizer in order to enable the loss to go down even when our model is very deep. The utilization of the decay rate also makes sense because we used pre-trained DenseNet weights rather than training the model from scratch.

[2] `https://github.com/flyyufelix/DenseNet-Keras.git`

4.2.3 Data Augmentation

When we initially built the vanilla transfer learning architecture, we observed heavy overfitting even after applying weight decay and dropout[21]. To mitigate this problem, we applied data augmentation to transform each raw image into 36 training samples. This approach has been heavily used in AlexNet[3].

Specifically, we applied 4 types of image transformation to each image:

1. Resizing: Each image in the Amazon dataset has size 256×256, whereas all ImageNet pre-trained architectures uses input of size 224×224. We simply resize the image to the network input size.

2. Cropping: We take 5 crops for each image - 4 in the corners and 1 in the center.

3. Flipping: We flip each image both vertically and horizontally.

4. Rotating: We rotate each image to 90, 180 and 270 degrees.

Notice that unlike typical photograph images such as what is in the ImageNet, satellite images can preserve the semantic meaning after flipping vertically and rotating. This allows us to do more powerful data augmentation than what was applied in AlextNet[3].

In training, we randomly pick 1 of the 36 transformations for each image in each epoch. In testing, we compute the scores of all transformations and use the average score.

4.2.4 Warm Up and then Fully Train

We tuned the learning rate in two stages. First, we freeze all convolutional layers and only train on the last fully connected layers in the first 2 epochs. Also in the first stage we use relatively large learning rate (0.01 or 0.001). In the second stage, we train on all layers and switch to a smaller learning rate. The intuition behind having two stages of learning rates is that the weights of fully connected layers are randomly re-initialized, hence the gradients given by the first few epochs might be meaningless for the rest of the network. We want the fully connected layers to be quickly adapted to the training data without perturbing the pre-trained convolutional layers too much.

4.2.5 Threshold Tuning

At first, we used a default rate of 0.5 as a threshold. A threshold is the value that when the scores of each label pass this threshold, we will add the corresponding label to the final predicted values. Threshold tuning makes sense in our case because the goal is to optimize F2-score which

has more weight on recall. We used threshold rate from 0.1 to 0.9 and tested our prediction to calculate F2-score in the validation set.

4.2.6 Ensemble

After we got results from test set after tuning the threshold, we made an ensemble model. The intuition behind tuning threshold first and then make an ensemble model is that we want to first optimize individual models and then get a result from these optimized individual models. We create the ensemble model by first looking at the predicted labels from the best epochs of each model. Then, we output the labels that appear more than certain frequencies. We further tuned this threshold and we found the ensemble model is able to give us a little boost in the final F2 score.

5. Experiment

We experimented with VGGNet, ResNet and DenseNet and examined the effectiveness of tricks mentioned in Section 4.2. Section 5.1 and 5.2 discuss the results we got from VGGNet and DenseNet. Section 5.3 compares the best performances among different models.

5.1. VGG

We used the pre-trained weights of VGG-16 model on ImageNet. Table 1 shows the performances of VGG-16 models under different settings. The mini-batch size used for all training is 128 and the input image size is 224 $\times$ 224 $\times$ 3. In the beginning, we only trained the model with very simple setting: train all of the fully connected layers and the last three convolutional layers. No other tricks were used. As we gradually added more tricks including adjusting the learnable layers, designing new loss function, applying data augmentation and assigning different layers with different learning rates, we observe consistent improvements in terms of the F2-score on the validation datasets. The most significant improvement was gained from fine-tuning the threshold for different labels. It consistently provides improvement on F2-score from 0.01 to 0.02. Observing that the last two fully connected layers in the VGG-16 model have too many parameters, we changed number of neurons in the last two hidden layers from 4096 to 1024. This simplified version VGG-16 model gives the best performance compared to the original VGG-16 model.

5.2. DenseNet

We first trained a DenseNet40-B model with 40 layers from scratch to evaluate the potential benefits of using the DenseNet. Although this model doesn't use pre-trained weights, it still gives a validation F2-score comparable to that of the VGG-16 models.

We then focused on fine-tuning the pre-trained DenseNet model. We chose DenseNet121-BC as our model. It has 121 layers but only 7 million trainable parameters. The DenseNet121-BC model consumes a lot of memory during the training, so we use mini-batch size of 16. We first train the full model using Adam with learning rate 5e-4. Then we realized it would be better to warm up the model first as mentioned in Section 4.2.4 where we train only the fully connected layers for a few epochs before training the full model. We then added learning decay to the model. The decay is defined as the following

$$lr := lr \cdot \frac{1}{1 + \text{decay rate} \cdot \text{\# iterations}}$$

We also tried to change the loss function as we did for the VGGNet model. Although it gives better validation F2-score before tuning the threshold, it gives slightly worse result after tuning.

5.3. Models Comparison

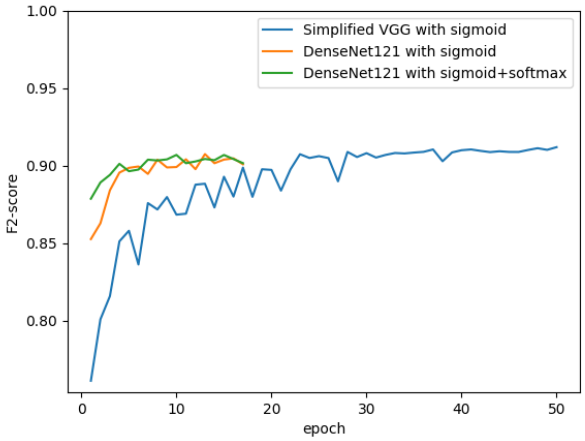

Figure 4: F2 Convergence Curve over Epochs on Best Architectures and Loss Functions

Table 3 lists the best performances we got for different models. The best model is the pre-trained DenseNet121-BC model. To further reduce the variance, we applies data augmentation to the best model at test time and took the average score as mentioned in Section 4.2.3. This helps us get F2-score 0.9288 on the leaderboard. Ideally the test-time data augmentation could be applied to all models, but considering we have 61,192 test images and with test augmentation we need to predict 61,192 $\times$ 36 images which we don't have enough resources to compute. So we only select the best candidate to do test-time data augmentation.

Figure 4 shows the speed of convergence between models. We can see that DenseNet converges much faster than

Model	Learning Rate	Trainable Layers	Data Augmentation	Loss Function	Validation F2
VGG-16	5e-4	FC + last 3 Conv	No	sigmoid	0.880
VGG-16	5e-6	FC + last 3 Conv	Yes	sigmoid	0.899
VGG-16	5e-6	FC + last 3 Conv	Yes	softmax + sigmoid	0.904
VGG-16	5e-6	FC + last 6 Conv	Yes	softmax + sigmoid	0.902
VGG-16	5e-6	All + layer wise learning rate	Yes	softmax + sigmoid	0.8966
Simplified VGG-16	5e-6	All + layer wise learning rate	Yes	softmax + sigmoid	0.9052/0.9167*

Table 1: Experiment Results For VGG. For layer-wise learning rate, we divide learning rate by 10 every 3 layers from fully connected layers to the bottom convolutional layers. * means the model has been applied threshold tunning.

Learning Rate	Warming Up	Learning Rate Decay	Loss Function	Validation F2
5e-4	No	No	sigmoid	0.880
5e-6	Yes	No	sigmoid	0.899
5e-6	Yes	5e-5	sigmoid	0.904/0.9278*
5e-6	Yes	5e-5	softmax + sigmoid	0.9069/0.9205*

Table 2: Experiment Results For DenseNet121-BC. * means the model has been applied threshold tunning. Data augmentation is used during training for all settings.

VGGNet.

5.4. Ensemble of Models

We used the prediction labels from different trained DenseNet121-BC and VGG models. For DenseNet121-BC, we included models trained using different learning rate settings and different loss functions. For VGG models, we used the simplified VGG-16 models. This ensemble of models yields our best F2 performance on the leaderboard, which is 0.9300.

5.5. Detailed Analysis on Best Single Model

Among the architectures we experimented so far, the DenseNet121-BC gives the best performance both in the validation datasets and the leaderboard.

During training, early stopping was applied: The training will stop if the validation loss did not decrease within 3 consecutive epochs. We then select the checkpoint with the highest validation F2-score. Figure 7 shows that the model start to overfit the data after approximately 10 epochs.

The distributions of the F2-scores for each label shown in the Figure 6 seem to be similar to the frequency of labels shown in the Section 3.1. It is reasonable because for some rare labels the classifier tends to only predict negative results. This is also one of the reasons explaining why tunning threshold gives significant performance boosts.

We also plot the co-occurrence matrix for both the true labels and our predication labels on validation set. (see Figure 5. The predication labels are attained after adjusting the threshold. Because the threshold for most labels are lower than 0.5, the model tends to predict more positive results than the ground truth. Hence, the plot on the right is slightly lighter for most of the pairs of labels compared with the plot on the left. From Figure 5, we see the model successfully captures some hierarchical relationships between labels. For example, the blooming will only occur in primary forest.

5.6. Discussion

1. Fully connected layers might not be necessary. Models with very few parameters in fully connected layers (DenseNets) are able to give accurate results. Also, fine-tuning pre-trained model is better than training from scratch considering the size of the training data in this competition is limited.

2. Regularization and model ensemble work really well. Both training and test time data augmentation can help reduce variance, which leads to a better performance. And the ensemble of models trained with different settings is effective for models with many local optimum.

3. Learning rate decay is necessary. Without learning rate decay the model will have a hard time dividing into regions with lower loss. We see a great improvement after adding the learning rate decay.

4. The modified sigmoid+softmax loss function gives worse results compared with simple sigmoid loss function with fine-tuned threshold. One possible reason is that the modified loss function introduces another hyper-parameter, weight of sigmoid loss function compared to softmax loss function. In our experiment we simply set the weight proportional to number of labels in different categories. Tuning this parameter could potentially give us high F2-score.

Model	F2 on training	F2 on validation	F2 on Leaderboard	# Parameters
Pre-trained Simplified VGG	0.9001/0.9234*	0.9052/0.9167*	0.9123*	42.3M
DenseNet40-BC	0.907	0.902	0.9043*	1.15M
Pre-trained DenseNet121-BC	0.9084	0.9053	0.9233*	7M
Pre-trained DenseNet121-BC*	-	-	**0.9288***	-
Pre-trained DenseNet121-Sig+Softmax	0.9115	0.9069	0.9147*	7M
Pre-trained ResNet50	0.87	0.88	-	0.85M
Ensemble	-	-	**0.93006***	-

Table 3: Best Results for Different Models. * means the result was obtained using test-time data augmentation. Due to the limit in time, we did not tune ResNet50 a lot hence did not submit its test time prediction to leaderboard on Kaggle.

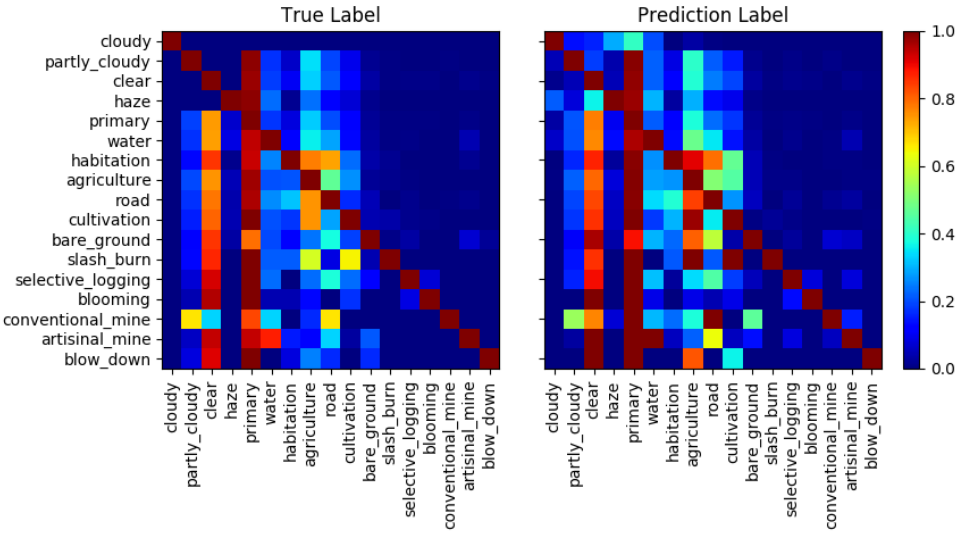

Figure 5: Co-occurrence Matrix on True Labels vs Predicted Labels on the Best Model on Validation Set

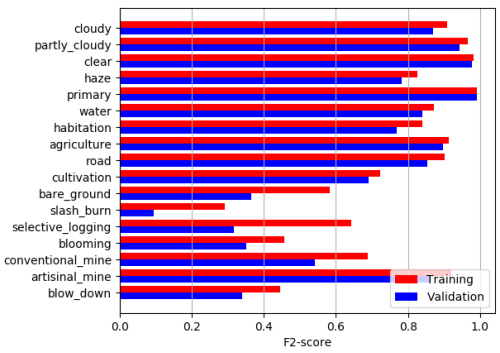

Figure 6: F2-Score of each Label on the Best Model on Validation Set

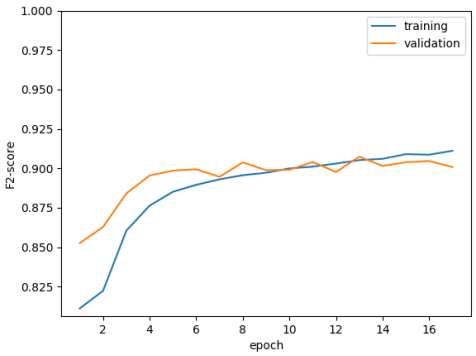

Figure 7: F2 Convergence Curve Comparison between Training set and Validation Set on Best Model

5. Figure 6 shows that F2-scores of rare labels are significantly lower. A potential solution is to train a separate model focus only on these rare labels and apply techniques such as oversampling.

6. Conclusion

The best performing model in our case is the ensemble model which combines the results from three DenseNet121 models and one VGG-16 model. The ensemble threshold for the best performing model is 50%. The single best performing model is the DenseNet121-BC model using pre-trained weights. The best performing single model adopts a two-stage learning rate with the learning rate of 5e-6 and the decay rate of 5e-5 in the second stage. The best performing single model has used test-time data augmentation.

For the ensemble model, the intuition behind why its result is better than the other individual models is that by combining the results from several good models, the variance will decrease and thus boost the overall F2-score. This is what we see in our results as well. The reason why the single best performing model is the above DenseNet model are:

1. The DenseNet architecture allows each block to take information from every other block and this feature matches the characteristics of our dataset.

2. The transferable pre-trained weights gives the model a good starting point and the two-stage learning rate tuning allows the loss to decrease when the model trains deep.

3. Applying data augmentation at test time helps reducing the variance of prediction.

As for the next steps, there are two types of work we want to pursue. First, we want to further tune our current model to have a better test F2-score. This includes trying more hyperparamaters to tune our current best performing model and focusing on the rare labels in order to predict them more accurately. The other type of work requires more changes to our current model. This includes implementing the small object detection technique in Chen et al.'s paper [16] because in our dataset, the labels can correspond to small items in the satellite images. Another possibility is to include some hand-crafted features in the model. It is worth mentioning that from manually designed features such as moments estimations, the XGboost[22] algorithm can give competitive F2-score around 0.88. Including these hand-crafted features may further improve our F2-score.

References

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

[2] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[7] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[8] Corey Baker, Rick Lawrence, Clifford Montagne, and Duncan Patten. Mapping wetlands and riparian areas using landsat etm+ imagery and decision-tree-based models. *Wetlands*, 26(2):465–474, 2006.

[9] John Richard Otukei and Thomas Blaschke. Land cover change assessment using decision trees, support vector machines and maximum likelihood classification algorithms. *International Journal of Applied Earth Observation and Geoinformation*, 12:S27–S31, 2010.

[10] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

[11] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Icml*, volume 32, pages 647–655, 2014.

[12] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Jun Ding, Bo Chen, Hongwei Liu, and Mengyuan Huang. Convolutional neural network with data augmentation for sar target recognition. *IEEE Geoscience and Remote Sensing Letters*, 13(3):364–368, 2016.

[15] Fan Zhang, Bo Du, and Liangpei Zhang. Scene classification via a gradient boosting random convolutional network framework. *IEEE Transactions on Geoscience and Remote Sensing*, 54(3):1793–1802, 2016.

[16] Xueyun Chen, Shiming Xiang, Cheng-Lin Liu, and Chun-Hong Pan. Vehicle detection in satellite images by hybrid deep convolutional neural networks. *IEEE Geoscience and remote sensing letters*, 11(10):1797–1801, 2014.

[17] C Quintano and E Cuesta. Improving satellite image classification by using fractional type convolution filtering. *International Journal of Applied Earth Observation and Geoinformation*, 12(4):298–301, 2010.

[18] Sergio Guadarrama and Nathan Silberman. Tensorflow-slim.

[19] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[22] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.