Land Cover Classification in the Amazon

Zach Maurer
zmaurer@stanford.edu

Tanuj Thapliyal
tanuj@stanford.edu

Shloka Desai
shloka@stanford.edu

Abstract

The Amazon rainforest has been subject to accelerating and aggressive deforestation. Human conservation efforts have been stifled by limited human and financial capital as well as inaccessibility of many regions of the Amazon itself - both for data collection and coordinating on-the-ground action. Low orbit satellite imagery may provide an important monitoring tool to conservationists by predictively labeling regions of the amazon with evidence of the human footprint. To help achieve that end, we explore a family of multi-label machine learning classifiers and successfully obtain an F2 score of 89% on a test set of satellite images from the Amazon rainforest.

1. Introduction

In this project, we identify evidence of human activities in the Amazon Rainforest, such as road development or mining, using satellite imagery from the Planet Kaggle Dataset[1]. This problem is most similar to land cover classification where each individual example could have multiple labels (i.e. one or more). The world's natural habitat is being rapidly developed, and tools to measure the rate of environmental destruction fall short. For example, conventional satellite imagery can only resolve to coarse 30m resolution, which makes it difficult to classify what types of activity are going on in the images. Planet Labs has super-fine 2-5m resolution data available for the first time[1], which opens an interesting opportunity to help identify why deforestation occurs, predict where it could occur next, and help identify strategies to combat deforestation. Our plan is to take the training image dataset, divide it into a training and validation sets, preprocess the images so the pixel intensities are normalized, augment the data by introducing transformations (translations, rotations, flips, jitter, etc) and experiment with different neural network architectures (and appropriate loss functions) with different hyperparameters to determine the best architecture for this dataset based on the validation F2-score metric.

2. Background and Related Work

There are several categories of prior work which detail techniques to classify images with multiple labels. State-of-the-art (as of early 2017) is using pairwise rankings to select the top labels to classify an image.[13, 16, 17]

2.1. Multi-Task Learning

One technique is to have a single architecture before a final fully connected layer, and then from that final layer, branch into 17 parallel softmax layers used to do classification per each label.

A similar class of techniques ranks the labels and then selects a subset, either through top-k, thresholding, etc. Li et. al (2017) expand on this by introducing a smoother (non-hinge) loss function that allows a deep net to converge faster, obtaining the best results in the literature on a variety of datasets. We believe this is the most clever technique because they also treat the thresholding per class as a parameter that needs to be optimized (vs. fixing it at .5 for a sigmoid activation layer, for example). This makes sense when considering some class labels to be sparse while others may have pretty high frequency in a training or test dataset.

2.2. Per Label Ensemble

Another technique is to have 17 entirely separate neural networks perform classification per each label. The advantages are each network can learn intermediate representations unique to each label. The disadvantages are that it can be more computationally and memory intensive, and that it may not capture correlations and cross-label patterns (For example, perhaps there are often mines near roads).

3. Dataset and Features

3.1. Description and Distribution

We use the dataset from Kaggle's Planet: Understanding the Amazon from Space challenge [1]. The dataset consists of over 150K 256 x 256 image tiles labelled with at least one of 17 classes. Each tile covers a ground-sample distance of 3.7m. Example classes are primary rainforest, water, habitation, cultivation, cloudy etc. We have around 40K images in the training set and 60K images in the test set

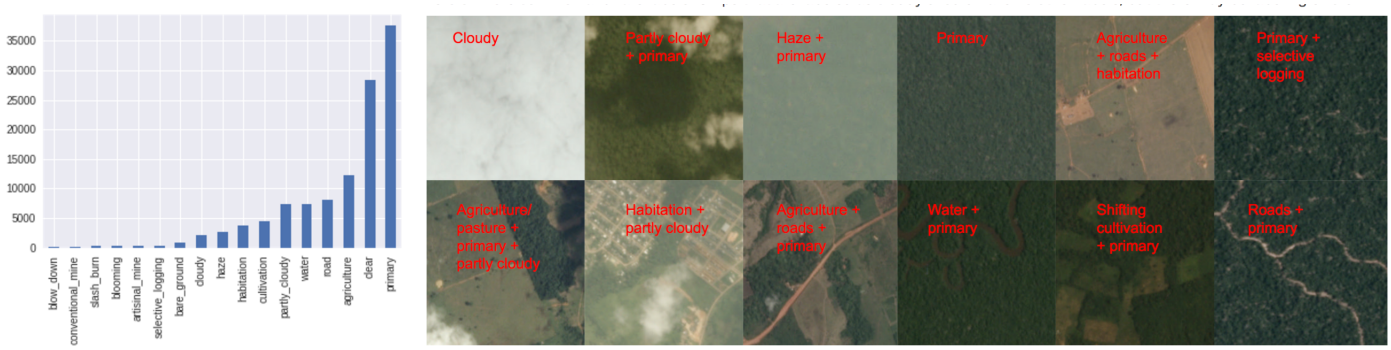

Figure 1. Class label distributions, and example images, from the starter notebook in the Kaggle challenge [1]

(as provided by Kaggle). Each image contains RGB values for each pixel, as well as Near Infrared channels. We use only the RGB information to classify the images. Figure 1 shows class distribution of the difference labels. As can be seen, the most common label is primary rainforest, followed by clear. Images can have only one for four weather labels haze, clear, cloudy, partly cloudy.

The data presented interesting challenges. First, the distribution of class labels was heavily skewed, with 2 classes (primary and clear) occurring in at least 75% of all the training examples, and the majority of remaining classes barely represented in the dataset. For example, blow down and conventional mine only appeared in less than 100 examples out of 40,000. Since each class prediction is considered separately in the calculation of the F2-score (and consequently the loss function), many of our exploratory models tended to predict the minority classes as always absent and majority classes as always present.

It was discovered through the Kaggle community that the higher resolution TIF images provided by the competition organizers were mislabeled. Thus, we only considered the JPEG files for this project.

3.2. Data Preprocessing and Augmentation

Several techniques were used to help preprocess the data and improve performance. First, the pixel values were standard normalized per color channel in order to have cleaner inputs into the neural net. Second, the data was augmented by introducing random jitter, cropping, and linear transforms (rotations and flips). Third, we selectively upsampled training examples which contained underrepresented class labels. This was challenging with 17 classes - how do you choose which examples to upsample while still maintaining equal coverage of the classes? We solved this problem by treating it as a linear optimization problem (constrained least squares). All the training examples were stacked into a matrix, M, as columns. We then defined a weight vector, w, which contained a relative weight of each training example. Lastly, we defined a target vector, b, which was just 17 ones, representing each of the class labels. Thus, $w* = argmin_w||Mw - b||$ represents a linear combination of all the training examples being as close as possible to an equal distribution of the classes. In other words, we are projecting perfect class coverage onto the range of all of our training examples, finding the closest possible weight vector w. We constrained the least squares projection further by not letting any example be less than of the average weight of a training example, assuming uniform probability. Unfortunately, this approach resulted in dramatic overfitting of our data, with F2 training scores in the 90s and poor validation set performance after a single epoch of training. No amount of regularization (dropout, etc) was able to successfully reduce overfitting, and thus we ultimately removed it from our data processing pipeline.

4. Methods

In total, we explored several dozen distinct models, bucketed into 4 major groupings, 6 unique model architectures, and numerous hyperparameter explorations (learning rate, layer sizes, regularization strength, etc). The 4 groupings are - (1) traditional machine learning (boosted trees), (2) simple multilayer convolutional networks, (3) recursive convolutional networks, and (4) transfer learning.

4.1. Traditional Machine Learning

Our first baseline was an XGBoost decision tree[18] trained on aggregate image features. For each color channel (red, green, blue), we looked at 6 extracted features per image: the mean, standard deviation, minimum, maximum, kurtosis, and skewness for each channel, which resulted in a total of 18 features per image. With 17 possible class labels per image, we created 17 trees with independent binary logistic classifiers to determine if each label should be applied to the image, independently. The trees had a max depth of 5 with 100 estimators per label [25].

4.2. Simple Multilayer Convolutional Network

We also implemented a basic CNN neural network architecture using two convolutional layers followed by 2 fully connected layers. Each convolutional layer used a ReLU nonlinearity, batch-norm and max-pooling layers. In addition to the convolutional layers the network contained a linear layer of 2048 units and a final linear layer for the 17 classes. The first convolutional layer had 32 7x 7 filters with stride of 1 and padding of 2. The second convolutional layer had 32 3 x 3 filters with stride of 1 and padding of 2. The flattened vector fed into the first fully connected layer had a dimensionality of 7200.

To train this network we use Multi Label Soft Margin Loss, also known as Sigmoid Cross-entropy loss. It is defined as follows. Below, x represents the calculated scores from the network, and y represents the k-hot encoding of image labels. We minimized this training loss using Adam as the optimizer.

$$\text{loss}(x, y) = -\frac{1}{n}\sum_i^n (y_i \log(\frac{exp(x_i)}{(1+exp(x_i)}) + (1 - y_i) \log(\frac{1}{1+exp(x_i)}))$$

The learning rate, along with other hyperparameters, were tuned using a random search followed by a grid search. The reason we chose random search over grid search is because the loss landscape in deep neural networks with respect to the hyperparameters is often sharp and steep, and a grid search may miss local optima sitting inside these hidden valleys.

4.3. Recursive Convolutional Network

Recursive convolutional networks are a type of deep neural network architecture which have a repeating, or recursive layer-blocks. Essentially, on the downward branch of the recursive net, activations are passed through successive convolutional layers and maxpool blocks. Once the recursion bottoms out at some pre-defined depth, activations are then scaled up through upsampling to match activation dimensions of the previous level. This proceeds in a recursive fashion where lower levels are upsampled and recombined with the convolutional activation immediately above it. Recursive convolutional networks are useful because combining activation maps that are down-sampled and resampled allow the classifier to integrate multiple levels of spatial resolution at different steps in the network. This design has the potential to highlight small details and enhance their prominence in the upscaled activation maps.

(i) U-Net[26]

The model starts by creating successively downsampled representations of the input image by passing it through several CONV-BN-ReLU-MAXPOOL blocks. The images are then upsampled to the previous representations resolution and recursively recombined with each other to form the final prediction activations. The U-net is an attractive architecture choice for two reasons. First, since the U-Net consists heavily of convolutional layers, we reduce the parameter count relative to models consisting of more Dense layers due to weight sharing. This will be detailed more in the Dataset section. Second, U-Nets have been shown to work well on other image segmentation tasks[26], which makes its cross-application to the domain of satellite images attractive, where we dont need to spatially segment images but merely put a binary label if a classification is present. We believe that an architecture that could be adapted to segmentation tasks may have a positive bias towards detecting fine-details that would indicate the presence of different classes in our dataset such as roads and blooming events. Instead of an Adam optimizer, our recursive architectures achieved optimal performance with stochastic gradient descent plus momentum.

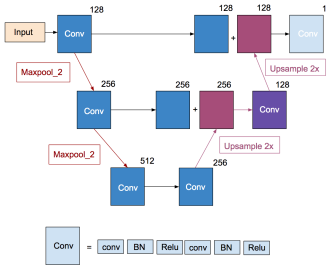

Figure 2. Our implementation of the U-Net architecture.

(ii) Pyramid Net [24]

Pyramid Net is similar to a U-Net, but with the important addition of residual short-circuit connections between recursive levels. The advantage of this is that weights which end up not transforming the input vector into a node have an intuitive value of either 0 or a very small, near-zero number. This behavior is significantly easier for a network to learn since the backpropagating error signals can update these weights more strongly than trying to identify an identity matrix pattern.

4.4. Transfer Learning

In transfer learning, we take pre-trained weights from another model architecture, and then modify and re-train the final layers of the network to make predictions on our dataset. Transfer learning in computer vision applications has analogs with word vectors in natural language processing. In NLP, word embeddings (derived through a semi-supervised classification process) captures the semantic meaning of the word as a list of numbers. In a similar way, pre-trained image networks act as a generalized feature

extractor or encoded that provides high-quality representations which can be fed to other specialized networks.

We tried several transfer learning implementations. First, we stacked our own fully connected layers on top of the pretrained model layers to perform additional linear and nonlinear operations. Second, we can selectively freeze and unfreeze previous layers, so the model continues to learn and update weights, even for the pretrained model architecture. In this project, we experimented with VGG[3] and Resnet pretrained models [4, 19].

5. Experiments, Results and Discussion

We implemented our models in Pytorch [18]. We used the starter notebook from the Kaggle challenge and Python Imaging Library (PIL) [23] to preprocess the data. We also used the XGBoost starter notebook from the Kaggle as a base for our implementation [25].

We primarily use the F2-score (using precision and recall) to evaluate the robustness of our models, since this is a metric that reflects performance on a dataset with unbalanced classes and was a performance indicator for the Kaggle leaderboard.

For the XG Boost baseline, the hyperparameters we settled on were a max tree depth of 5, a learning rate of 0.1, and a classification threshold of 0.5 to convert the logistic output into a binary indicator. We achieved a baseline F2 score of 0.68 on the dev set, which was submitted to the Kaggle leaderboard. After being trained on the full training set, the XGBoost model achieved a test-set F2 score of 88%. Further tuning is likely to improve F2 performance, but we were satisfied with this as a baseline machine learning performance metric to measure our CNN implementations against. Overall, the strong performance of XGBoost given its simplicity makes it a powerful choice for multilabel image classification tasks.

Loss curves and training curves for our first full run of our baseline CNN is shown below. Our best validation F2 score was 0.84077 and was achieved on the fifth epoch, which outperformed the XGBoost baseline. Based on our survey of other participants, we believe that this is on par with a fine-tuned XGBoost implementation. We used a learning rate of 1e-3 and a batch size of 100. The batch-size was chosen to utilize the amount of available memory during training.

One primary performance issue that we have observed so far is that recall is quite high (0.93), while exact match (0.45)is much lower. While this is expected, we hypothesize that our CNN baseline is prone to predicting the absence of a class more frequently, because most examples are only labelled with a couple of classes; thus most of the labels will be false, and the classifier can conservatively reduce its loss by tending to predict false for many labels. This observation is supported by the rapidly plateauing loss curve. Furthermore, there is a significant class imbalance where primary, clear and agriculture are observed far more often than other labels. To have a truly performant classifier it will be necessary to penalize our existing loss function more heavily for mistakes on these minority classes or change our objective function such that the classifier implicitly understands the importance these minority classes.

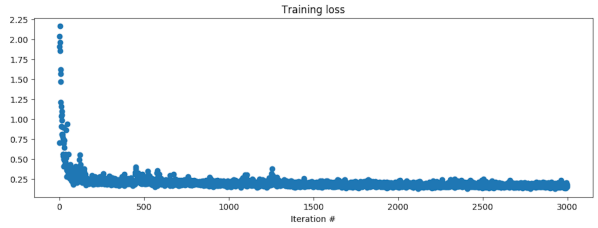

Figure 3. Loss curve

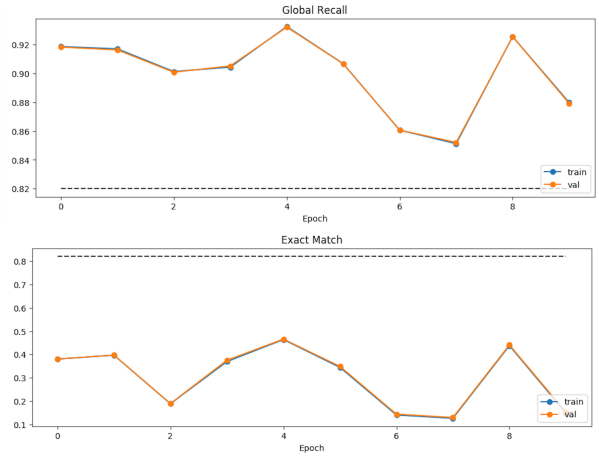

Figure 4. Recall and Exact match curves

The relative model complexity of the approaches we tried is below, measured by the number of parameters. Our simplest model was a boosted tree ensemble, and our most complex model was Resnet (via transfer learning).

Model	Number of Parameters
XGBoost	Depth: 5, Estimators: 100
Baseline CNN	14.8M
U-Net	8.6M
Simple U-Net	13.3M
Pyramid Net (Heng CherKeng)	3.7M
ResNet (retrained Dense)	35K
ResNet (retrain Conv + Dense)	14.9M
ResNet (Conv + Conv + Dense)	22M
VGG16	70K

Figure 5. Model complexity of the different models

The results of our various approaches are detailed below. The best performing models were in the transfer learning

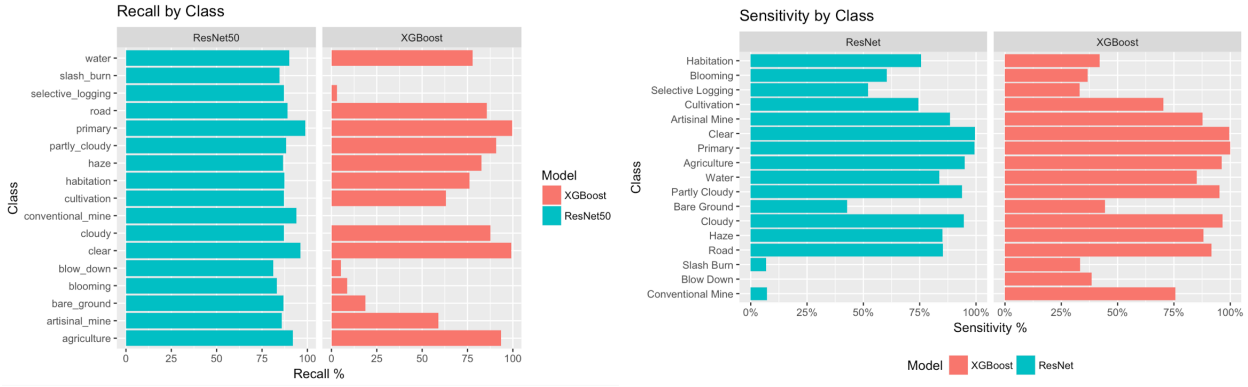

Figure 6. Sensitivity and recall by class

Model	F2-Score	Recall	Exact Match
ResNet (Conv + Conv + Dense)	89.24%	95.39%	58.50%
ResNet (retrain Conv + Dense)	88.48%	95.16%	56.69%
Baseline CNN	85.35%	93.89%	48.74%
ResNet (retrained Dense)	84.81%	93.86%	48.70%
Pyramid Net (Heng CherKeng)	84.08%	94.02%	46.83%
U-Net	82.41%	93.19%	46.13%
VGG16	82.18%	92.97%	42.33%
Simple U-Net	79.61%	91.70%	37.98%
XGBoost	79.40%	92.24%	39.16%
Simple U-Net (Jacc. Loss)	74.40%	90.82%	32.00%

Figure 7. Validation F2-scores, recall and exact match scores for various models

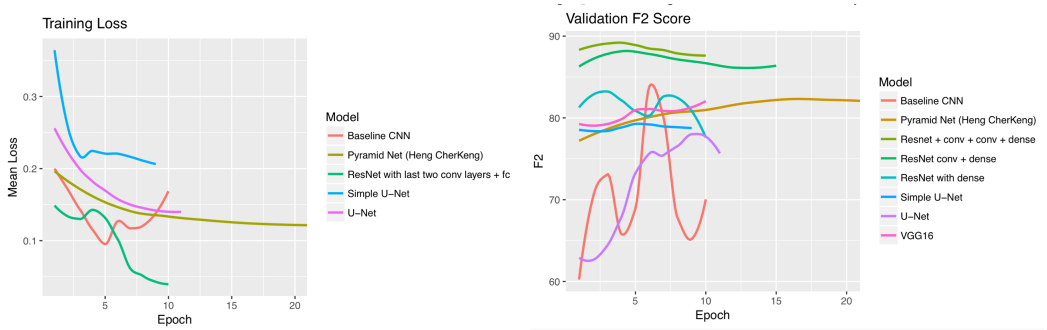

Figure 8. Training loss curves and Val F2 curves for various models

group. More specifically, we obtained the best performance measured by F2 validation score with Resnet 50 when we unfroze the fully connected layer and the last two convolutional blocks. We employed a learning rate schedule, starting with a high learning rate of 1e-3. We did this using an open source Pytorch learning rate scheduler [21]. Every time the loss would flatten out, we lowered the learning rate by an order of magnitude. Using SGD-Momentum(μ = 0.9) was crucial to lowering loss below 0.2. LR Decay with Adam halted progress for early results.

Recursive neural networks seem to perform well on possibly related tasks (i.e. image segmentation). However, a standard U-Net implementation was not as performant as a U-Net with modified residual connections such as Pyramid Net. Residual connections allow for a clean passthrough of the input to the output to have zero or near-zero weights, which is easier to learn than an identity matrix. Unfortunately, our own implementation of a U-Net was not competitive with transfer learning approaches.

Model	F2-Score
ResNet50	89.0%
XGBoost	88.1%

Figure 9. Test F2-scores for ResNet50 and XGBoost

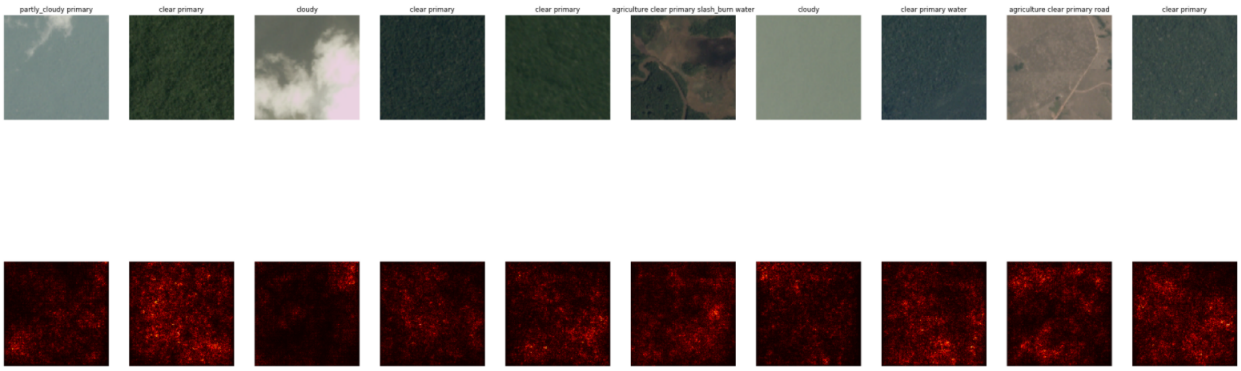

Figure 10. Saliency maps for various classes in the dataset

Additionally, the imbalanced dataset caused per-class sensitivity to vary significantly. Generally, the rarer labels had poorer sensitivity than the commonly occurring labels. Boosted trees did a significantly better job of performing prediction on the rarest labels. Our convolutional networks were better at predicting semi-rare labels, thus leading to the slight increase in test-set F2-score. Some of the features, like a conventional mine lacked spatial, frequency-domain characteristics that could be picked up by a convolution. However, they had unique color content which was very amenable to the statistics features that XGBoost was trained on.

We also experimented with different loss functions. Modifying the loss function to include Jaccard distance [7, 8] seems to have been used effectively in other cases, but our implementation requires more tuning. Jaccard distance is a penalty that measures how dissimilar the a predicted set of labels are from the true labels. Our original hope was that including a Jaccard distance approximation in our loss function would encourage higher sensitivity across minority classes and increase the exact match score of our classifier. This becomes significant when we have a dataset with multiple labels with nonuniform frequencies. We also looked at a hard margin multiclass SVM loss function, which obtained inferior performance to the softmax classifier. Due to the imbalanced nature of our dataset, we believe the hard-margin loss was able to establish a sufficient margin across all classes relatively quickly by predicting false for most minority classes, thus satisfying the constraints of the optimization and effectively terminating the learning process too early.

In general, we observed that this imbalanced dataset results in classifiers who can safely minimize a significant portion of the loss by being overly conservative for minority classes and liberally predict the presence of majority classes. We hypothesize, this results in a particularly flat loss landscape, which results in extremely small gradient error signals that backpropagate through the network. This causes the weights to not change very much from batch to batch. Thus, learning rate schedules and proper optimization are key ingredients in order to get the network to get unstuck and actually learn. We believe SGD with momentum outperformed the Adam optimizer for this reason – SGD was able to continue making progress, when the adaptive function of Adam wouldve reduced the magnitude of subsequent updates.

6. Future Work

We have several planned future experiments. We would like to train multiple sub-networks that each are for a single label type. The drawback of this approach is that

We'd like to condition neural architectures on statistical features. This allows us to capture additional non-spatial information into our convolutional networks that helped XGBoost obtain good validation set performance.

As an experiment in the other direction, we plan to feed Resnet features into XGBoost, treating the flattened outputted vector outside Resnet as analogous to a word embedding in natural language processing.

Wed like to ensemble all high-performing trained models above an F2 score of 90%. Decisions would be made on individual class labels based on a majority vote per class.

7. References

[1] Kaggle Planet Understanding the Amazon from Space Challenge: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3] Very Deep Convolutional Networks for Large-Scale Image Recognition K. Simonyan, A. Zisserman.

arXiv:1409.1556

[4] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

Multitask classification [5] Caruana, Rich. "Multitask learning." Learning to learn. Springer US, 1998. 95 133. http://www.cs.cornell.edu/~caruana/mlj97.pdf

[6] Qi, Kunlun, et al "Multi-Task Joint Sparse and Low-Rank Representation for the Scene Classification of High-Resolution Remote Sensing Image." Remote Sensing 9.1

[7] Jean, Neal, et al. "Combining satellite imagery and machine learning to predict poverty." Science 353.6301 (2016): 790-794. http://science.sciencemag.org/content/353/6301/790

[8] Zhang, Fan, Bo Du, and Liangpei Zhang. "A multi-task convolutional neural network for mega-city analysis using very high resolution satellite imagery and geospatial data." arXiv preprint arXiv:1702.07985 (2017). https://arxiv.org/ftp/arxiv/papers/1702/1702.07985.pdf

[9]Wei, Yunchao, et al. "CNN: Single-label to multi-label." arXiv preprint arXiv:1406.5726 (2014). https://arxiv.org/pdf/1406.5726.pdf

[10] Wang, Jiang, et al. "Cnn-rnn: A unified framework for multi-label image classification." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. https://www.ics.uci.edu/ yyang8/research/cnn-rnn/cnn-rnn-cvpr2016.pdf

[11] You, Shan, et al. "Privileged Multi-label Learning." arXiv preprint arXiv:1701.07194 (2017). https://arxiv.org/abs/1701.07194

[12] Zhu, Zhe, and Curtis E. Woodcock. "Continuous change detection and classification of land cover using all available Landsat data." Remote sensing of Environment 144 (2014): 152-171.

[13] Li, Yuncheng, Yale Song, and Jiebo Luo. "Improving Pairwise Ranking for Multi-label Image Classification." arXiv preprint arXiv:1704.03135 (2017). https://arxiv.org/abs/1704.03135

[14] Frnkranz, Johannes, et al. "Multilabel classification via calibrated label ranking." Machine learning 73.2 (2008): 133-153. https://pdfs.semanticscholar.org/5918/04251e15cfb571bc90c2fab2344f462e1617.pdf

[15] Bucak, Serhat S., et al. "Efficient multi-label ranking for multi-class learning: application to object recognition." Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009. http://www.robots.ox.ac.uk/~vgg/rg/papers/multilabelranking.pdf

[16] Mencia, Eneldo Loza, and Johannes Furnkranz. "Pairwise learning of multilabel classifications with perceptrons." Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. IEEE, 2008. https://www.ke.tu-darmstadt.de/~juffi/publications/ijcnn-08.pdf

[17] Menca, Eneldo Loza, Sang-Hyeun Park, and Johannes Frnkranz. Advances in efficient pairwise multilabel classification. Technical Report TUD-KE-2008-06, TU Darmstadt, Knowledge Engineering Group, available at¡¡ http://www. ke. informatik. tu-darmstadt. de/publications/reports/tud-ke-2008-06. pdf, 2008. http://www.ke.tu-darmstadt.de/publications/reports/tud-ke-2008-06.pdf

[18] Extreme Gradient Boosted Trees http://xgboost.readthedocs.io/en/latest/model.html

[26] O. Ronneberger and P.Fischer and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation" (2015) http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

Code: [18] Pytorch, deep learning framework. https://github.com/pytorch

[19] Pytorch Tranfer learning and finetuning tutorial http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#finetuning-the-convnet

[20] CS231n Assignment 3 code (Network visualization notebook, saliency map generation). cs231n.github.io/assignments2017/assignment3/

[21] Pytorch learning rate scheduler: https://github.com/Jiaming-Liu/pytorch-lr-scheduler

[22] Kaggle Understanding the planer from Space challenge starter notebook: https://www.kaggle.com/robinkraft/getting-started-with-the-data-now-with-docs

[23] Python Imaging Library: http://www.pythonware.com/products/pil/

[24] Hengcherkeng notebook on PyramidNet https://github.com/hengck23-kaggle/kaggle_forest_2017/blob/master/baseline-1/forest-1/net/dataset/kgforest.py#L55-L96,https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/discussion/32402

[25] Kaggle Understanding the planer from Space challenge XGBoost starter notebook: https://www.kaggle.com/opanichev/planet-understanding-the-amazon-from-space/xgb-starter/run/1101066