Kaggle Competition: Understanding the Amazon from Space

Sneha Kudli
skudli@stanford.edu

Steven Qian
scqian@stanford.edu

Benjamin Pastel
bpastel@stanford.edu

Abstract

*This paper documents our team's approach to the Kaggle Competition: Understanding the Amazon from Space. The challenge consisted of labeling, as accurately as possible, satellite images of the Amazon rainforest with the atmospheric and geographic conditions shown in each image. Our team employed an ensemble of pre-trained models with customizations along with careful pre- and post-processing to achieve an F_2 score of **0.9285** and a rank of 33rd out of 438 competitors.*

1. Introduction

Deforestation in the Amazon basin is a growing concern due to its devastating impact on biodiversity, habitat loss and climate change. An ongoing competition in Kaggle aims to use the land usage pattern data in the Amazon to better understand how and where deforestation is happening. In this paper, we discuss our approach to solving this Kaggle challenge: *Planet: Understanding the Amazon from Space*. [14]

The objective is to label 256 by 256 satellite image chips from the Amazon with atmospheric conditions and different classes of land cover and use. The atmospheric conditions are either clear, hazy, partly cloudy, or cloudy. Some examples of land cover labels are primary rainforest, cultivation, roads, water, mines etc. Each image can consist of multiple labels.

Our current best model is an ensemble consisting of custom layers trained on top of SqueezeNet [10], Inception [23], ResNet [9], and Xception [6] pretrained models. Our single best performing model was an architecture that modified SqueezeNet to be deeper, and split at the second to last layer between separate networks to predict the weather labels and the ground cover layers.

2. Related Work

There have been previous papers that examine possible approaches to the analysis of satellite imagery. [16] shows how Deep Learning can be applied to the classification and segmentation of satellite city imagery. The paper takes the approach of using CNN's and a per-pixel classification method to categorize the images into the buckets of vegetation, ground, roads, buildings and water.

[25] also takes a CNN based approach to using maps street view data to count number of tress but focuses more on combining satellite views with street views than optimizing just for satellite images.

[3] addresses a similar problem of classifying satellite image data into land use categories. As part of the paper two datasets, SAT-4 and SAT-6 are developed where SAT-6 classifies images into categories: barren land, trees, grassland, roads, buildings and water bodies. The paper first tries a Deep Belief Network (DBN), but an architecture using CNNs easily outperforms that. However, the final architecture presented in the paper includes a feature extractor, followed by some unsupervised pre-training combined with a DBN and this outperforms the CNN. The paper argues that traditional deep learning architectures are good at learning sharp/edge based features but are not useful for satellite imagery because satellite datasets have high intra and inter-class variability. They also have lesser amount of training data compared to the total dataset size including test examples. Also, higher-order texture features are a very important discriminative parameter for various land-cover classes. The best DBN model combined with feature extractor presented in the paper gets an accuracy of around 98% and 94% on SAT-4 and SAT-6. However, the task we are tackling is more involved since the nature of the labels is more intricate as opposed to simple broad classes as in SAT-6. We also have 17 categories instead of 6.

We also derived inspiration from [13] which was another satellite image classification contest on Kaggle. The winner of the contest used sliding windows, ensembling, data augmentation by oversampling rare classes, and post processing to disambiguate easily confused classes. He also trained his network from scratch using the U-NET segmentation network that had been employed in previous Kaggle competitions, which we are considering doing as a next step. [18]

3. Evaluation

The competition is judged on F_2 score, defined as follows. Let N be the total number of test samples, L_i and $\hat{L}_i$ be the true and predicted labels for the i'th test sample, respectively. Then:

$$P_i = \frac{|L_i \cap \hat{L}_i|}{|\hat{L}_i|}$$

$$R_i = \frac{|L_i \cap \hat{L}_i|}{|L_i|}$$

$$F_2 = \frac{5}{N} \sum_{i=1}^{N} \frac{P_i R_i}{4P_i + R_i}$$

Compared to the more common F_1 score, the F_2 penalizes false negatives more heavily than it penalizes false positives. [20]

4. Data

The data for this competition is from [12] and consists of 40,479 training samples and 61,192 test samples from satellite imagery. Each image is of size (256, 256, 3), with the channels representing R, G, B. Each pixel in an image corresponds to a resolution of 3.7 m meters on ground. We split 20% of the training data into a validation set with a fixed random seed, giving us 32,384 training samples and 8,095 validation samples.

The data was also provided in 4-channel TIF format, with the fourth channel being infrared. Top competitors noted severe data quality issues with the TIFs and unclear performance after working around those data quality issues so we decided to limit our attention to the JPGs only. [4] [17] [24] [5]

4.1. Distribution

The dataset has a skewed distribution biased towards the clear weather label and the primary rainforest label. The weather labels are exclusive, i.e. it can only be one of clear, hazy, partly cloudy or cloudy. If the label is cloudy, then the image is too cloudy to identify the land use pattern, hence such an image generally has no land cover label. If the weather label is anything other than cloudy, then any number of land cover labels can be applicable to the image depending on the content. Labels like 'blow_down', 'conventional_mine', 'slash_burn', 'artisinal_mine', 'selective_logging' and 'blooming' are very infrequent and together only account for about 1% of all the labels found in the dataset.

Example images with labeled data are shown in 1. The class distributions for the train/validation datasets is shown in Table 1.

Figure 1. Example Labeled Images

Label Type	Label	Training	Validation
Weather Labels	cloudy	1844	486
	haze	2163	532
	partly_cloudy	5773	1478
	clear	22604	5599
Land Cover Labels	cultivation	3590	887
	primary	30278	7562
	water	5785	1477
	artisinal_mine	267	72
	habitation	2913	749
	bare_ground	678	181
	blow_down	79	19
	agriculture	9840	2498
	selective_logging	263	77
	conventional_mine	86	14
	slash_burn	167	42
	blooming	267	65
	road	6424	1652
Both	**Total**	**93021**	**23390**

Table 1. Class Label Distribution

4.2. Quality Issues

The data was labeled by humans using a 3rd-party crowd-sourcing company, and there are severe data quality issues. [12]

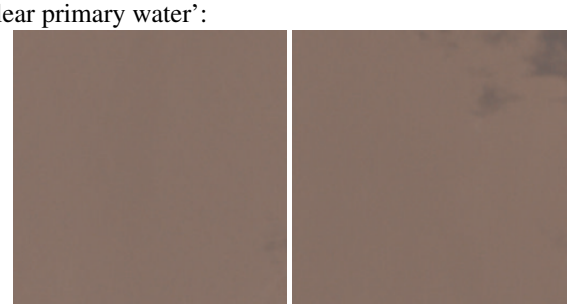
First, the distribution of the labels does not match the descriptions and definitions of the labels. For example, the description of the 'cultivation' label explains:

> Shifting cultivation is a subset of agriculture that is very easy to see from space, and occurs in rural areas where individuals and families maintain farm plots for subsistence.

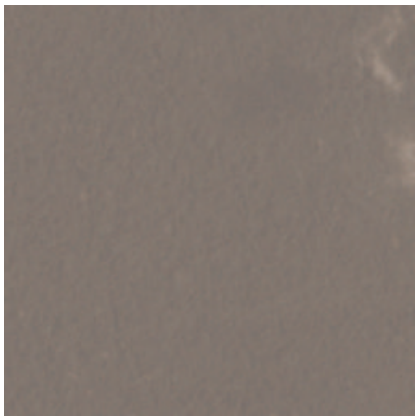
However, in the training data, cultivation is not actually a subset of agriculture; out of 4477 images with the cultivation label, 1100 of them do not include the agriculture label. By eye, we were unable to discern a clear pattern in which of those images also included the agriculture label.

Similarly, the description of 'slash-and-burn' claims it is a subset of cultivation, while in the actual data, 83 out of 209 slash-and-burn images do not include the cultivation label.

Second, the dataset includes many examples that are nearly impossible by eye to determine what is going on. For example, contrast an example of 'clear agriculture' with 'clear primary water':



Or, consider the following image labeled 'haze cultivation primary agriculture', which appears to be missing any agriculture:



We believe the ambiguity inherent in examples like these are the reason why the current best F_2 score of any competitor is only 0.93320.

5. Methods

5.1. Pre-Processing

Since our pretrained models were trained on ImageNet [8], we first normalized by subtracting the per-channel means from the ImageNet dataset. This normalization method is from Keras Framework [7] in TensorFlow [1]. We used numpy low-precision 16-bit floating point values so that we could fit the entire training dataset in memory.

We used two augmentation strategies for different models in our ensemble:

Conservative Augmentation:
In this strategy, before training on any image we randomly applied random horizontal and vertical flips and shifts. The points outside the input in a shift was filled via reflecting the input image. This augmentation strategy improved results on all models and was relatively fast.

Aggressive Augmentation:
In this strategy, before training on any image we performed the same transformations as in Conservative Augmentation above, but also applied a random rotation of up to 180 degrees, a random shear, a random zoom, and a small random perturbation to all the values. This is an aggressive augmentation strategy that significantly slowed training time and improved results only on some models.

We also implemented a flexible sub-sampling and super-sampling framework. We defined a model parameter $\mathbf{\Theta}$ representing the desired super-sampling or sub-sampling of each class. Then during training, we first selected the label of each training sample according to:

$$Pr(\text{label } L \text{ selected for training}) \sim \Theta_L * (\text{Frequency of } L \text{ in training data})$$

And then drew a random row having that label from the training set. Our final configuration sub-sampled the primary class to $\frac{1}{10}$ of its original frequency, super-sampled all rare classes by a factor of 2, and also super-sampled the cloudy class because accurate cloudy labels are particular useful in our post-processing.

5.2. Loss Function

Since there is exactly one true weather label plus one or more non-weather labels, the evaluation metric is partially categorical.

For most models in our ensemble, we ignored this effect and used a sigmoid layer into a binary cross-entropy loss function, and exploited the partially categorical nature in the post-processing instead.

For one version of our SqueezeNet model, we split the architecture near the end, with one branch predicting the four weather labels and the other branch predicting the remaining labels. We used a softmax for the weather labels and a sigmoid for the remaining labels; however, the binary cross-entropy loss for both outputs still outperformed various categorical losses. We weighted the loss on the weather branch $\frac{1}{20}$ the loss of the main branch.

The split architecture models performed slightly better individually, but the ensemble as a whole performed better when both types of models were included. We also tried tweaking the relative weights of loss functions within each class, but were unable to find any performance improvements.

5.3. Post-Processing

The true labels had several logical constraints: exactly one weather label was present at any time, and if the weather label was 'cloudy', then no ground cover labels were present. However, since the F_2 score penalizes false negatives worse than false positives, enforcing these constraints on the predictions reduced performance significantly.

However, we were able to successfully leverage these constraints to gain a small but consistent performance improvement via the following algorithm:

Stage 1: Find the global threshold that optimizes F_2 score. Use this is an initial guess for threshold of each column.

Stage 2: Find the per-column threshold that optimizes F_2 score on the training data, given the best threshold found so far on the other columns.

Stage 3: For each weather label, step through possible thresholds for 'weather exclusivity'; i.e. thresholds at which we are confident enough in this label to remove any other predicted weather labels.

Stage 4: Step through possible thresholds for complete exclusivity of the cloudy label; i.e. thresholds at which we are confident enough in the cloudy label to remove all other predicted labels.

Stage 5: Repeat step 2 once more.

This algorithm generally chose lax thresholds (0.1 to 0.4) for predicting labels, and very strict thresholds (0.8 to 1.0) for applying exclusivity. Stage 5 generally chose a significantly lower threshold for 'primary' than Stage 2, since many of true negatives for 'primary' were accounted for in Stage 4.

Although we developed them independently, stages 1-2 became common knowledge among all competitors after being published in a blog post [2]. To our knowledge, the remaining stages are unique to our group.

5.4. Models

We trained various models available in Keras framework. We did not include the top fully connected layers for any of the pre-trained ImageNet model weights we loaded because the images sizes they were trained on did not necessarily match our image sizes. We included custom Fully Connected layers and Dropout for the top layers. We use a sigmoid activation function at the top so that our model is setup to output one probability for each of the 17 binary class labels.

5.4.1 SqueezeNet

Our current best model uses the SqueezeNet network [10] with weights pretrained on ImageNet [8], accessed through [21]. Dropped the last two dense layers, deepened it by adding three more fire modules plus dropout, replaced the Global Average Pooling with a Global Max Pooling [19] layer, and added two different architectures of output layers. 2

The output layers are either a single FC-2048 followed by dropout and sigmoid-17, or the split architecture discussed in 5.2, with the following branches:

```
branch 1:
FC-1024
    Dropout(0.5)
    Sigmoid-13: ground-cover labels
branch 2:
FC-512
    Dropout(0.5)
    Softmax-4: weather labels
```

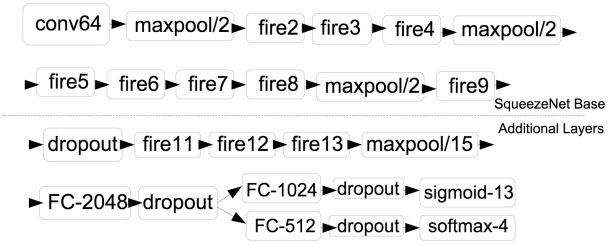


Figure 2. Augmented SqueezeNet Architecture

The split architecture performed better in isolation, but both models contributed to the overall ensemble performance.2

We introduced each of the modifications to the original SqueezeNet one at a time, freezing the weights of the previous iteration of the architecture, running to convergence, and then unfreezing the full model.

Both of these models used the full image size (256, 256, 3), which required training additional parameters in top layers of the original SqueezeNet. We also experimented with resizing the images to the SqueezeNet input size (227, 227, 3) with bicubic interpolation, but were unable to match the results of the full image.

5.4.2 ResNet50

We used the Keras Resnet50 [9] model with weights pretrained on ImageNet [8]. The images used for pre-training are (224, 224, 3) in size. The last layer of Resnet50 without including the top is 7*7 2D AveragePooling. This seems to be optimized for the 224*224 image sizes and we noticed that the training was slow on our images of size 256*256. We changed the 2D AveragePooling layer to 8*8 and added a GlobalAvergaePooling layer along with a couple of fully connected layers and Dropout. The architecture is shown in 2

5.4.3 InceptionV3

We used the Keras InceptionV3 [23] model with weights pre-trained on ImageNet [8]. The images used for pre-training are (299, 299, 3) in size. We include a couple of fully connected layers as shown in 4

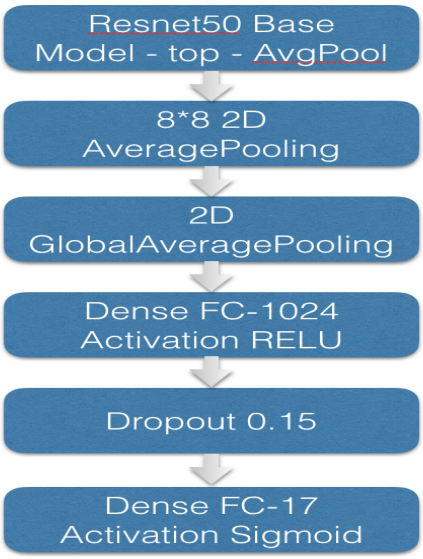


Figure 3. Resnet Architecture

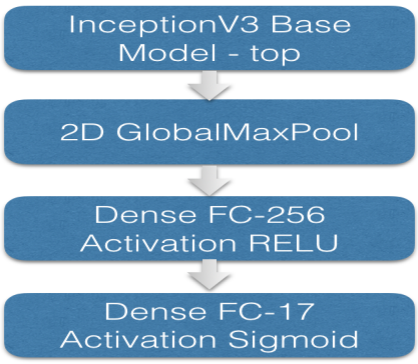


Figure 4. InceptionV3 Architecture

5.4.4 Xception

We used the Keras Xception [6] model with weights pre-trained on ImageNet [8]. The images used for pre-training are (299, 299, 3) in size. We include a couple of fully connected layers as shown in 5

5.4.5 VGG19

We used the Keras VGG19 [22] model with weights pre-trained on ImageNet [8]. The images used for pre-training are (224, 224, 3) in size. We include a couple of fully connected layers as shown in 6

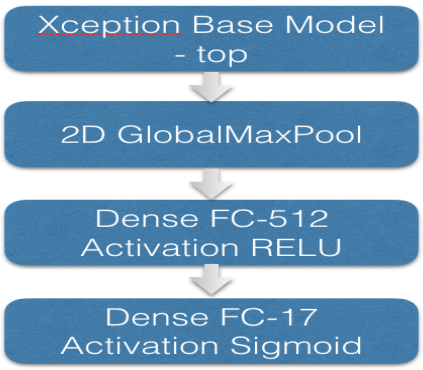


Figure 5. Xception Architecture

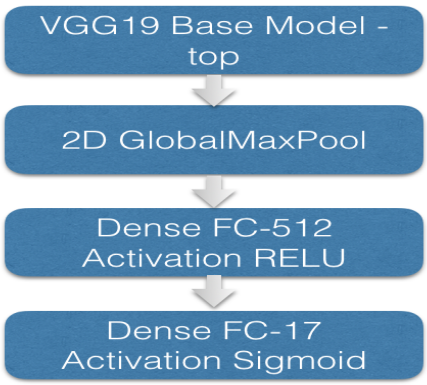


Figure 6. VGG19 Architecture

5.4.6 Ensemble of Models

We use a simple majority voting technique as described in [11]. Each label for each sample was predicted if a majority of models in the ensemble predicted the label on that sample.

5.4.7 Transfer Learning - Training Models

We adopt a transfer learning based approach to training. We start by loading model weights pre-trained on ImageNet and freezing them. We train our added top layers by using a relatively high learning rate. After few epochs, we unfreeze a few of the top layers of the loaded pre-trained model and train for few more epochs with a lower learning rate. We continue the process of unfreezing more layers of the model from top down and training them with subsequently lower learning rates. The size of top layers we added, the number of layers unfrozen every time, the number of epochs after which layers are unfrozen, the learning rate and the rate of learning rate drop are all hyperparameters tuned to each specific model. More details about the training process and graphs are available in appendix A.

Base Model	Augmentation	Top Layers	Train F_2	Val F_2	Test F_2
SqueezeNet	Conservative	Split	0.9352	0.9230	0.9221
SqueezeNet	Conservative	2x FC-2048	0.9360	0.9223	0.9214
Resnet	Aggressive	FC-1024	0.9372	0.9233	-
Resnet	Conservative	FC-1024	0.9307	0.9170	-
Inception	Conservative	FC-256	0.9308	0.9129	-
Xception	Aggressive	FC-512	0.9322	0.9158	-
VGG	Aggressive	FC-256	0.9257	0.9197	-
Ensemble	-	-	-	0.9292	**0.9285**

Table 2. Model Summaries

6. Results and Discussion

Our latest submission on Kaggle scored **0.9285** on the public leaderboard test set, which as of submission on June 12, 2017 placed us 33rd out of 438 competitors. Our best individual model submission was based on SqueezeNet with the split architecture on top, and achieved 0.9230 F_2 on our validation set, and 0.9221 on the public leaderboard test set.

Our ensembling was surprisingly effective. Models trained on different pretrained models turned out to contribute to the ensemble performance even when, like Inception, their individual scores were significantly worse than the other models. By contrast, we had many variations of SqueezeNet with different top layers or meta parameters that scored around 0.92 on the validation F_2, which turned out not to contribute marginal improvements to the ensemble. Presumably they were too similar to the two existing SqueezeNet models in the ensemble.

2 summarizes the performance of each of the individual models along with the ensemble.

7. Future Work

Our approach has paid off with a good leaderboard rank and position [15]. However, the competition has a month remaining, and we need to keep improving to remain competitive.

Although all of our models have some gap between training and validation performance, only Xception is displaying the classic overfitting pattern where the training error continues to decrease while the validation error flattens or increase as epochs increase. For the others, we think we can increase the capacity of our models by increasing the depth or width of our final layers, or adding more layers in the style of the pretrained models. VGG in particular has only a small gap between training and validation and needs an increase in complexity.

In our pre-processing, we plan to carefully incorporate the IR channel, perhaps using some concatenation scheme where the model can easily ignore the channel when it's not useful. We also plan to explore different styles of sliding windows to increase data augmentation, so that we can reduce overfitting in Xception in particular.

In our post-processing, we plan to change our ensembles to take the raw probabilities as input instead of the predictions, and searching for a more sophisticated way to do the exclusive weather post-processing on multiple model probabilities.

Finally, we plan to explore training one model architecture without any pretrained model. In particular, the U-Net architecture is popular among other Kaggle competitors, and is our top candidate for training from scratch. [18]

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Anokas. Better optimisation of f2 threshold. [Online forum post; accessed 1-May-2017].

[3] 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.

[4] H. CherKeng. The jpg and tif files of new test v2 are completely different? [Online forum post; accessed 22-May-2017].

[5] H. CherKeng. Sharing experiment results. [Online forum post; accessed 22-May-2017].

[6] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

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

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[10] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016.

[11] K. Inc. Kaggle ensembling guide, 2015. [Online; accessed 15-May-2017].

[12] K. Inc. Data — planet: Understanding the amazon from space, 2017. [Online; accessed 15-May-2017].

[13] K. Inc. Dstl satellite imagery competition, 1st place winner's interview: Kyle lee, 2017. [Online; accessed 11-Jun-2017].

[14] K. Inc. Planet: Understanding the amazon from space, 2017. [Online; accessed 12-June-2017].

[15] K. Inc. Public leaderboard — planet: Understanding the amazon from space, 2017. [Online; accessed 15-May-2017].

[16] M. A. Martin Lngkvist, Andrey Kiselev and A. Loutfi. Classification and segmentation of satellite orthoimagery using convolutional neural networks. *Remote Sensing*, 2016.

[17] Miku. Artifacts on tiff dataset but not on jpg? [Online forum post; accessed 22-May-2017].

[18] T. B. Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer, LNCS, Vol.9351: 234–241*, 2015.

[19] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localization for free? weakly-supervised learning with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Precision, recall and f-measures — scikit-learn: Machine learning in Python, 2016. [Online; accessed 15-May-2017].

[21] rcmalli. keras-squeezenet. `https://github.com/rcmalli/keras-squeezenet`, 2017.

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

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842 [cs.CV]*, 2014.

[24] Tom57. Tiff versus jpeg - my results so far. [Online forum post; accessed 22-May-2017].

[25] J. D. Wegner, S. Branson, D. Hall, K. Schindler, and P. Perona. Cataloging public objects using aerial and street-level images-urban trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6014–6023, 2016.

Appendix A. Training Graphs

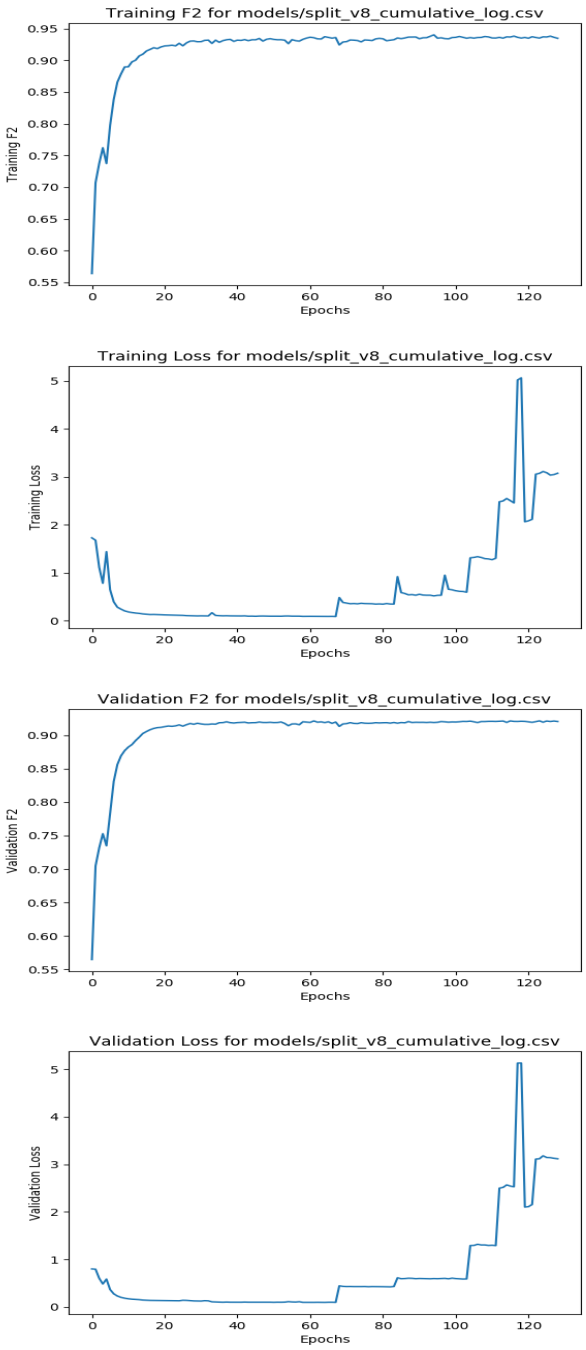
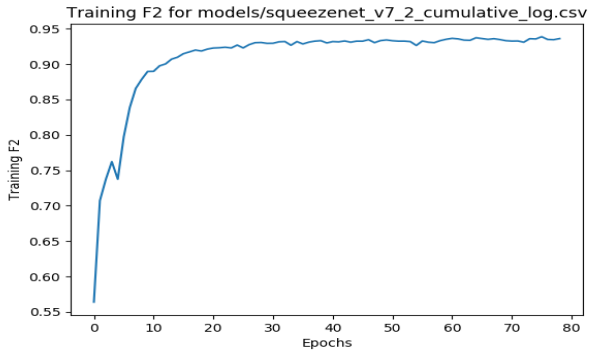
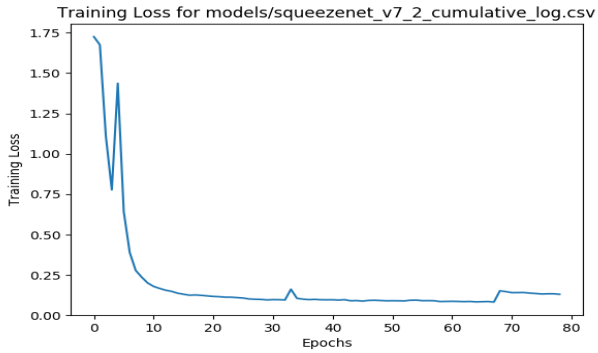
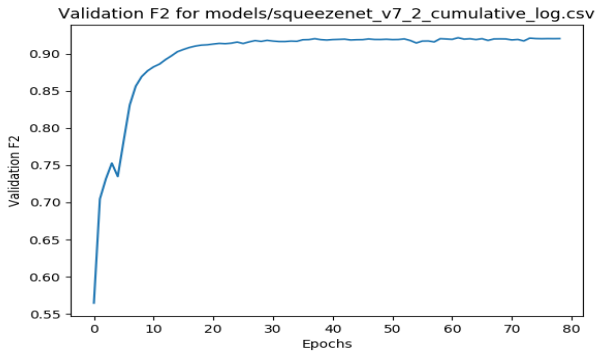


Figure 7. SqueezeNet split-architecture training graphs. The discontinuities near the middle are from changing the architecture of the last few layers; the discontinuities towards the end are from adding the split architecture and tweaking the relative weights of the two loss functions.







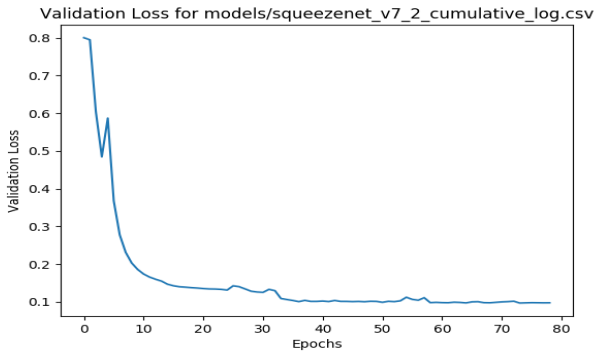
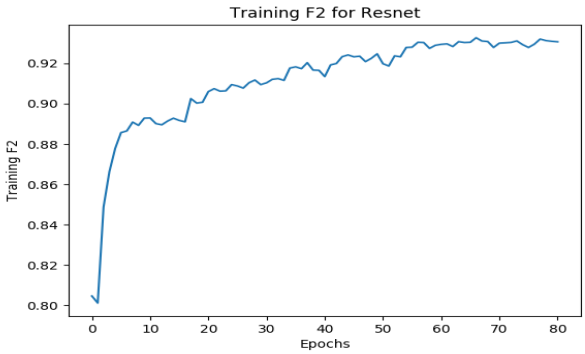
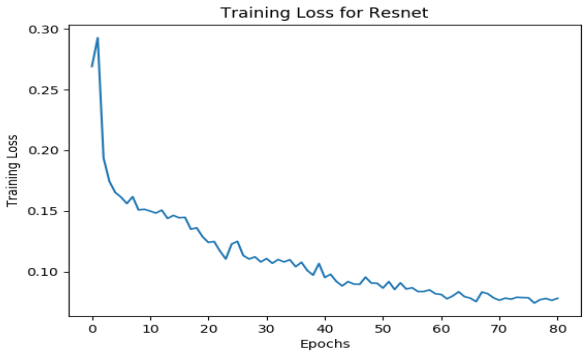
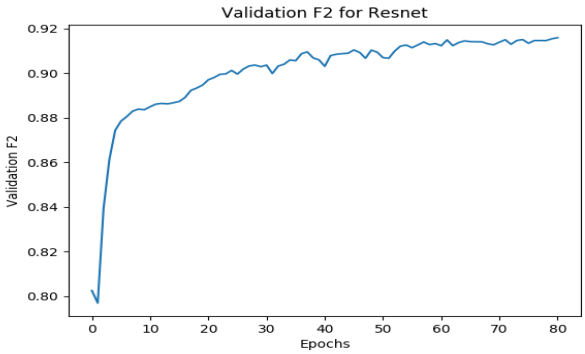


Figure 8. SqueezeNet 2x FC training graphs. The discontinuities are from changing the architecture of the last few layers.







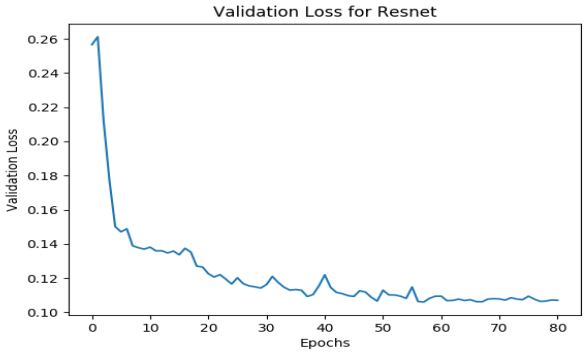
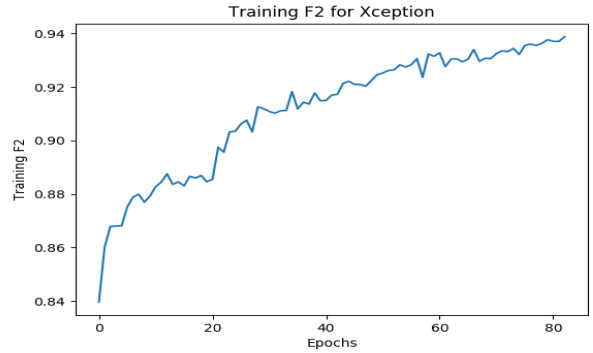
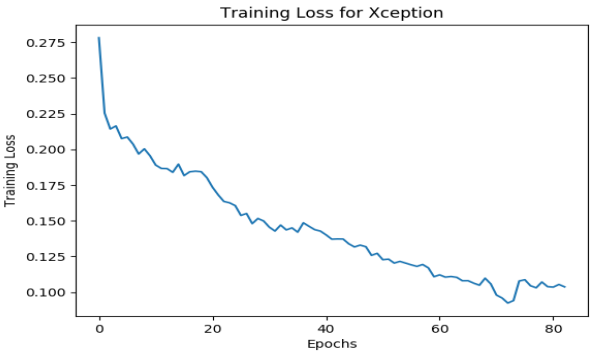
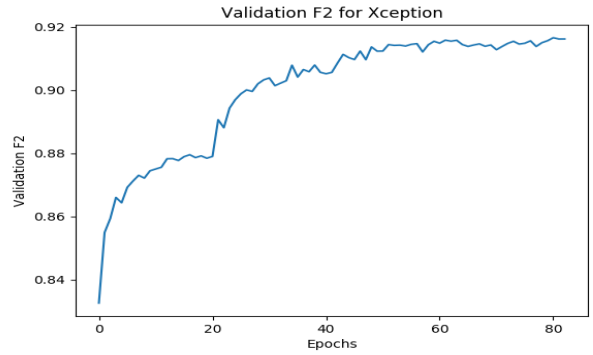


Figure 9. ResNet Training Graphs







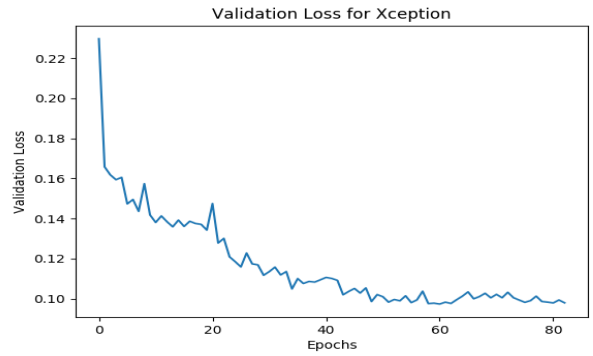
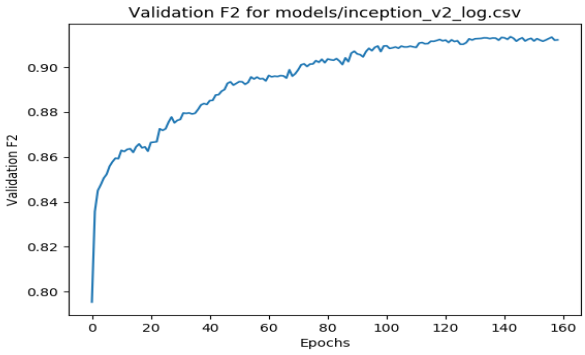


Figure 10. Xception Training Graphs - The sharp edges at epochs 20, 40 and 65 are due to unfreezing of layers. The sharpness reduces at higher epochs since there is lesser dataset specific features to learn at lower layers since these are for learning more general representations.



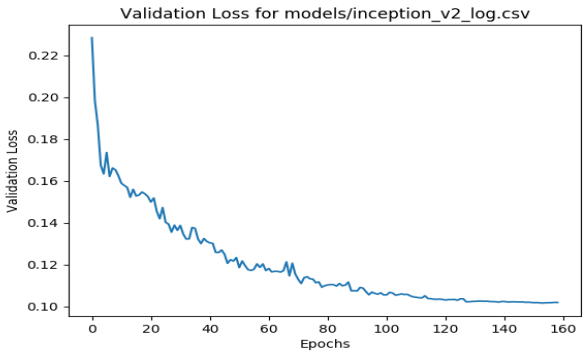


Figure 11. Inception Training Graphs - Inception in general makes steady but very slow progress and in comparison to other models took more epochs to train to similar levels

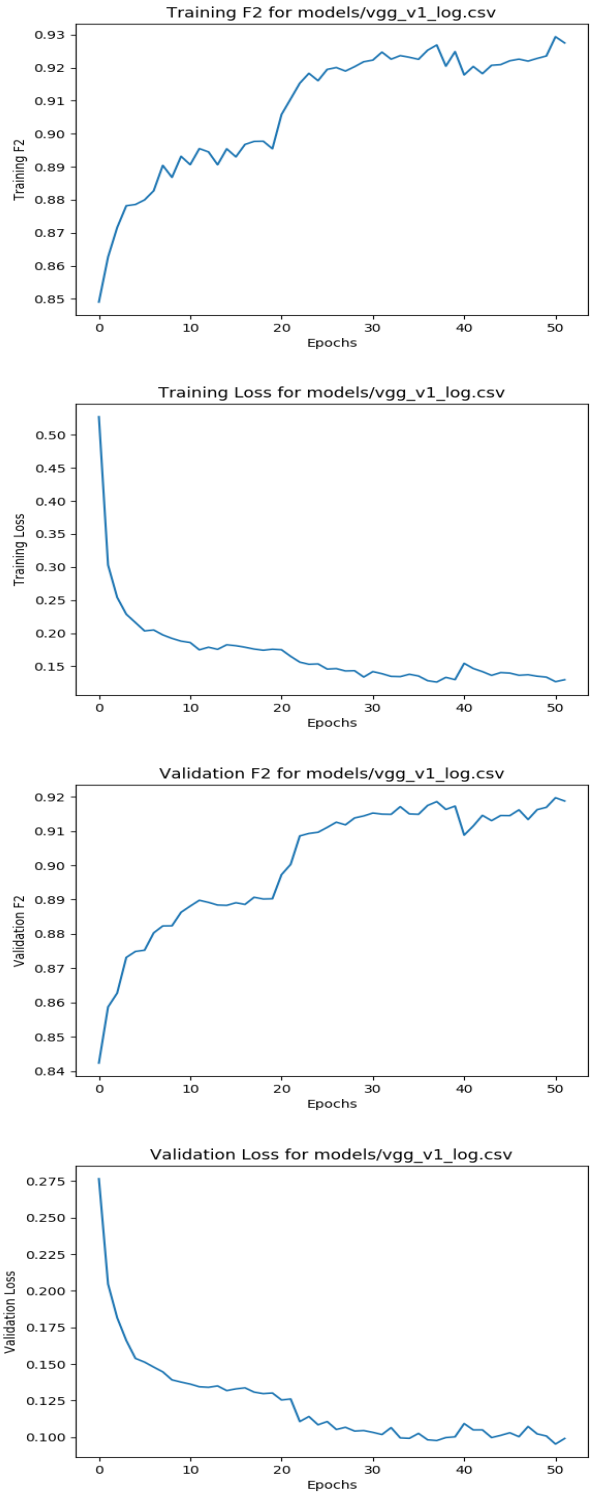


Figure 12. Vgg Training Graphs - The sharp edges seen at epochs 20 and 40 are because we unfreeze layers at these epochs.