Machine Learning Engineer Nanodegree

Capstone Report

Amitabha Chakravarty
21 July 2017

Kaggle Competition - **Planet: Understanding the Amazon from Space**

Definition

Project Overview

This project is all about a Kaggle competition named 'Planet: Understanding the Amazon from Space'. Our planet is loosing valuable resources every minute. Every minute an area of forest the size of 48 football field is disappearing from Amazon basin due to deforestation. Deforestation has caused a large scale of devastating effects – reduction of biodiversity, habitat loss, climate change to name a few. To counter the effect of loss of such forest area we need data that will pin point areas of human encroachment. This will help local governments and local stakeholders to respond quickly and effectively.

For this purpose, daily imagery from Planet, designer and builder of world's largest constellation of Earth-imaging satellites, will be used along with its Brazilian partner SCCON. The goal for this competition is to label the satellite image chips with atmospheric conditions and various classes of land cover and land use. The wining algorithms will help the global communities understand better where and how deforestation is happening and ultimately how to deal with it.

Problem Statement

Image (chip) data is collected from Planet's full-frame analytic scene products using 4 band satellites in sun-synchronous orbit. The data is labeled using Crowd Flower platform and a mixture of crowd-sourced labor. There are class labels like 'Cloudy', 'Partly cloudy + Primary', 'Shifting cultivation + primary' and so on. There are a significant number of cloudy scenes with complete to partial cloud coverage and also hazy conditions. There are also clearly shown images of 'Primary Rain Forest', 'Water (River and Lakes)' and 'Habitation' etc. The job is to come up with a data analysis model and use these labeled data for training purpose. There will be a set of images with no label and the model has to be tested against the unlabeled data.

The goal of this project would be to develop and algorithm to detect the type of scene represented by the testing images with varying atmospheric conditions and various classes of land cover/land use. There are several approaches to analyzing an image and classify the scene based on prior knowledge. One such approach would be finding PCA (Principal Component

Analysis) components and classify using SVM (Support Vector Machine). However, with recent success of deep learning, Convolution Neural Network (CNN) is another very effective technique to analyze image and classify its content based on prior learning. I decided to go with CNN approach here for its high success rate.
CNN would be used to train and detect the correct label for the ground scenario represented by the amazon image chip. CNN outperforms many image recognition algorithms in ImageNet dataset, The CNN algorithm consists of many convolution operations followed by pooling operation to feed finally to fully connected layer. Finally, the prediction comes out of the classification layer connected next to the fully connected layer.

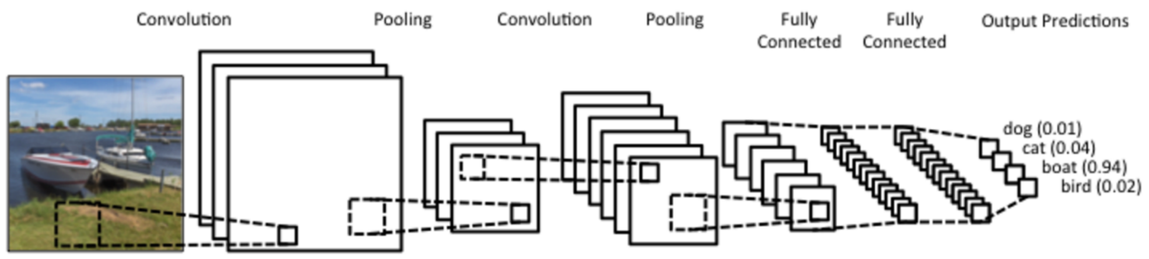


Image courtesy - http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Metrics

In case of multi-label image classification, it is well known that 'accuracy' is not particularly helpful when there are varied distribution of labels among the images. 'F1 score' which measures how well the algorithm is doing among different classification groups is a better choice. However, I have used both of these metrics during validation but given more importance to 'F1 score' while evaluating the model.

'F1 Score' is defined as follows –

F1 = 2 * (precision * recall) / (precision + recall)

Precision: This is proportion of all positive predictions that are correctly predicted. Precision is a measure of how many positive predictions were actual positive observations.

Precision = TP / (TP + FP)
= (positive predicted correctly) / (all positive predictions)

where TP and FP are defined as follows -

True Positive (TP) = positive observations that are correctly predicted as positive

False Positive (FP) = negative observations that are incorrectly predicted as positive

Recall: This is proportion of all real positive observations that are correctly predicted out of all positive observations.

Recall = TP / (TP + FN)

where TP is True Positive as defined above and FN is False Negative that is defined below –

False Negative (FN) = Positive observations that are falsely predicted as negative.

True Negative (TN) = Negative observations that are correctly predicted as negative.

In our implementation I am using F-beta score which skews the F1-score towards precision or recall based on the value of beta.

$$F\beta = (1 + \beta\text{^}2)\ \text{Precision} \cdot \text{Recall} / (\beta\text{^}2 \cdot (\text{Precision} + \text{Recall})$$

The beta influences the score in following way. If beta < 1, assigning correct label is more important whereas beta > 1 means the metric is weighted towards penalizing incorrect class assignment. In our implementation it is not always guaranteed to detect all the correct labels for the images but I can try to suppress using incorrect labels. I have followed this principle and used beta =2 to penalize incorrect label assignments.

'Accuracy' is defined as the ratio of correct predictions to total number of predictions.

Accuracy = (TP + TN) / (TP + TN + FP + FN) where TP, TN, FP and FN are as defined above.
In our implementation it is a case of multi-label accuracy which returns subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0. In our case the output of classification layer is decimal number between 0 and 1 for each label. I have used a threshold to convert the output value of ConvNet to 0 or 1. The threshold is fixed according to a value which gives highest F-beta score.

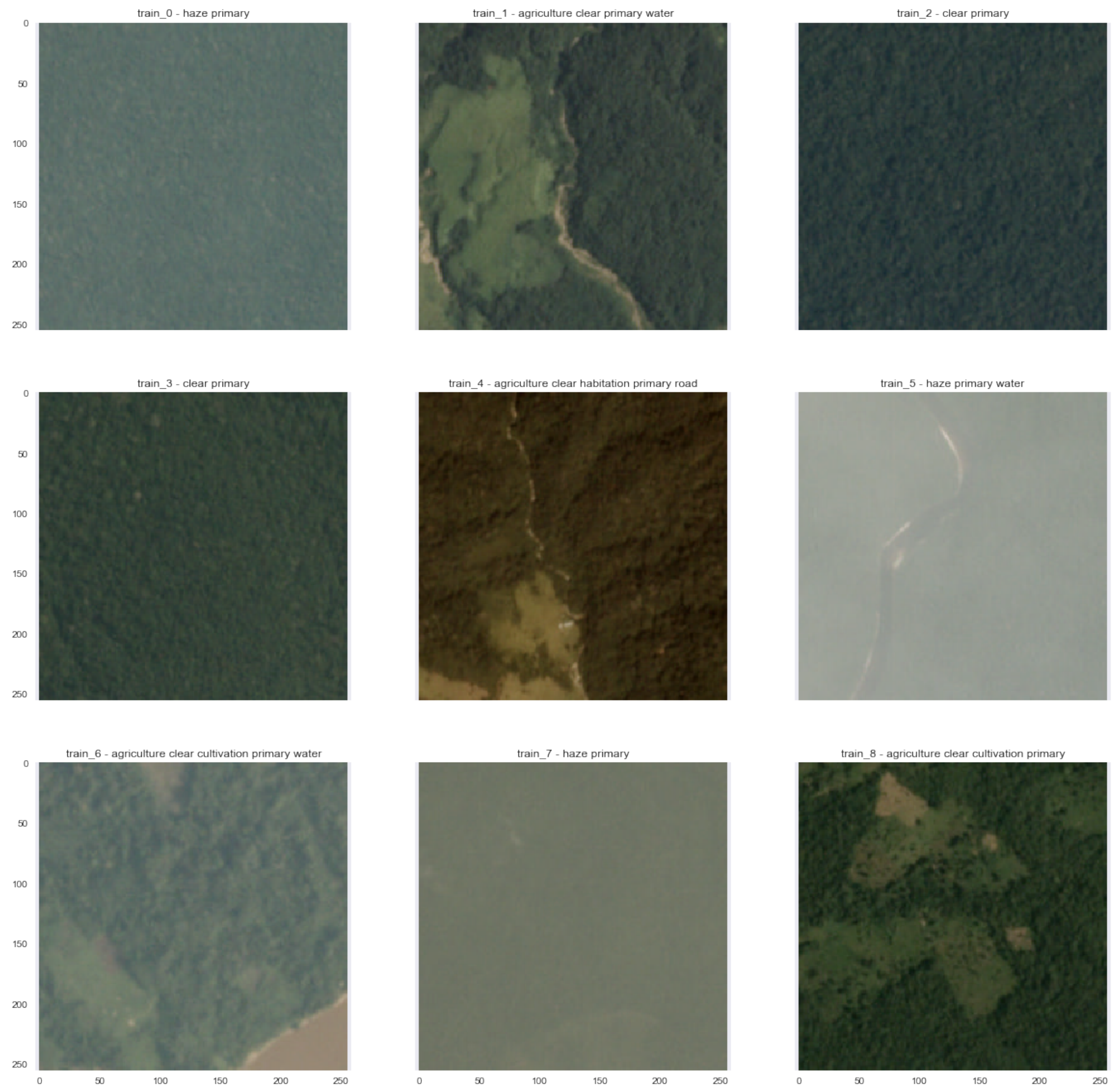
Analysis

Data Exploration

We have around 40000 images for training and 40000 images for testing. There are both jpg and 4-band tiff images for training and testing. Each image is a 256x256 pixels corresponding to a real area of 221.7 Acres. For training the corresponding labels are presented whereas for testing we need to predict the labels.

The majority of the data set is labeled as "primary", which is shorthand for primary rainforest, or what is known colloquially as virgin forest. There are also areas representing ‘Water’, ‘Agriculture’, ‘Road’, ‘Cultivation’ and so on.
The labels are presented as applicable to the ground reality with multiple keywords like – ‘Agriculture/pasture + primary + partly cloudy’.
There are two kinds of images - a "hard" and an "easy" set. The easy set contained scenes that are easier-to-identify labels like primary rainforest, agriculture, habitation, roads, water, and cloud conditions. The harder set of data was derived from scenes to represent shifting cultivation, slash and burn agriculture, blow down, mining, and other phenomenon. A disclaimer is mentioned on the competition page that some training labels could be wrong as there could be labeling error. Overcoming the inaccuracy would be a challenge for this particular completion. There is also significant cloud covering for some of the images. Detecting scenario on ground in presence of cloud would be particularly challenging.

Below are some of the images from the training set. One observation can be made that these images represent scenes that can tolerate horizontal or vertical flips. This can be one method of augmenting the training data set to overcome generalization.


train_0 - haze primary
train_1 - agriculture clear primary water
train_2 - clear primary
train_3 - clear primary
train_4 - agriculture clear habitation primary road
train_5 - haze primary water
train_6 - agriculture clear cultivation primary water
train_7 - haze primary
train_8 - agriculture clear cultivation primary
Image courtesy - Kaggle Kernel - https://www.kaggle.com/anokas/data-exploration-analysis

Exploratory Visualization

I have analyzed the input training labels and found that there are following unique 17 levels.

0: 'agriculture',
1: 'artisinal_mine',
2: 'bare_ground',

3: 'blooming',
4: 'blow_down',
5: 'clear',
6: 'cloudy',
7: 'conventional_mine',
8: 'cultivation',
9: 'habitation',
10: 'haze',
11: 'partly_cloudy',
12: 'primary',
13: 'road',
14: 'selective_logging',
15: 'slash_burn',
16: 'water'

Next I have plotted the relative frequency of occurrence of each label as a bar plot as shown below.

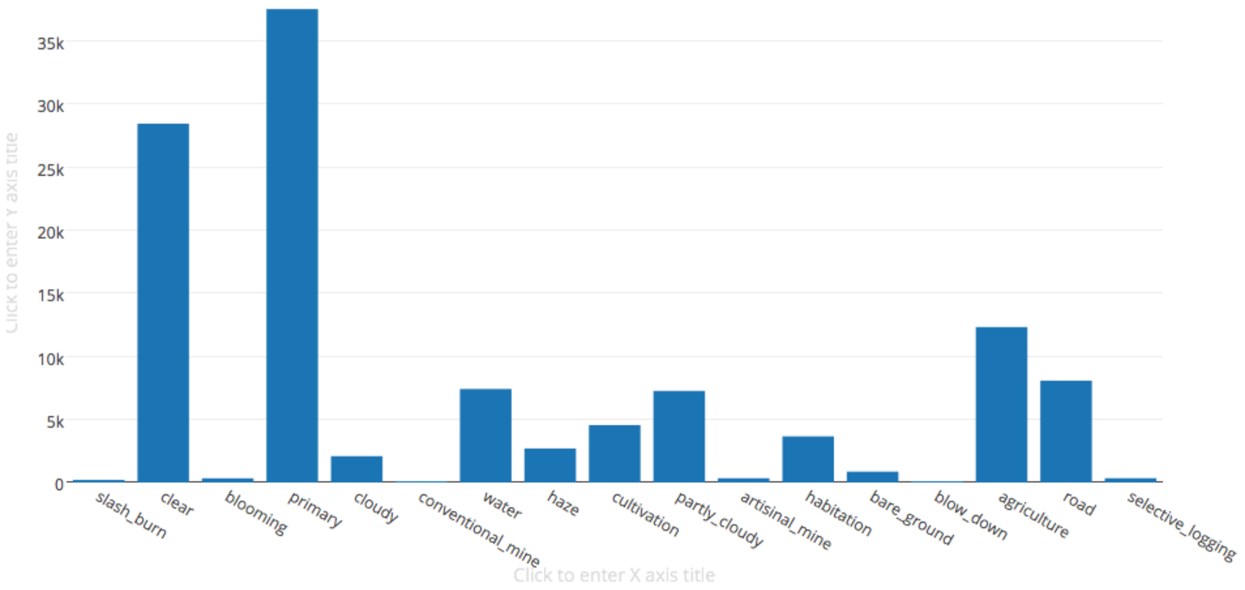
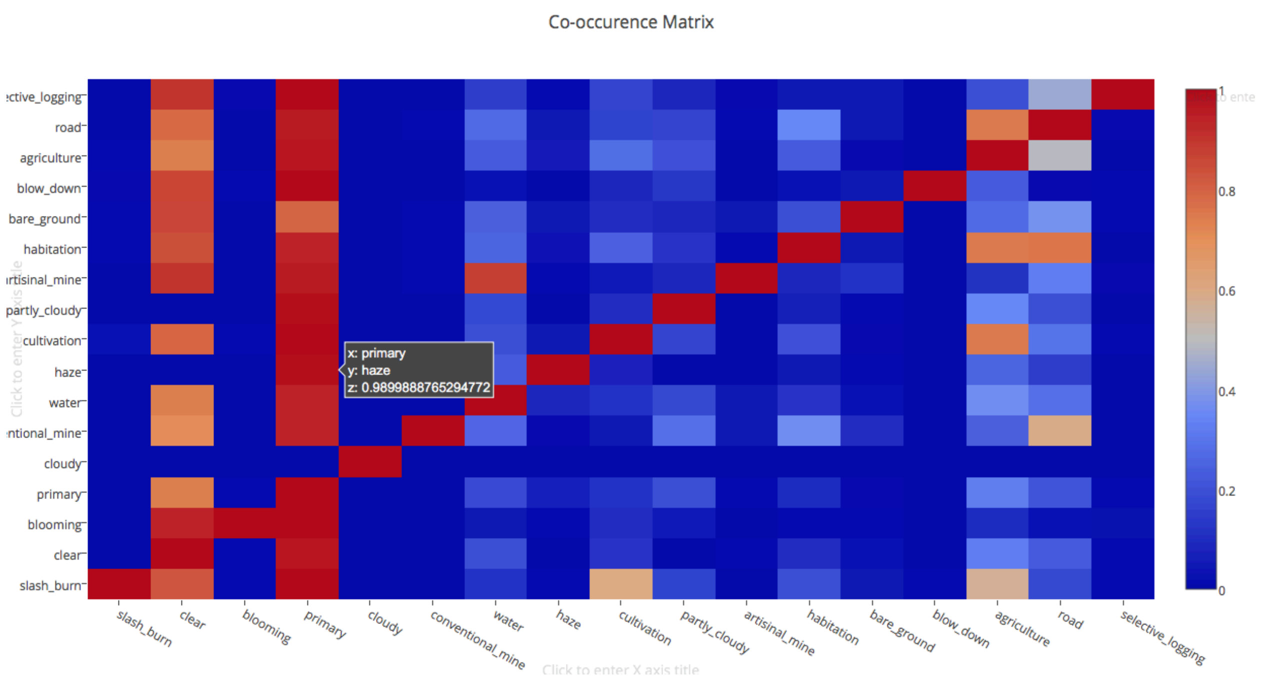


Image courtesy - Kaggle Kernel - https://www.kaggle.com/anokas/data-exploration-analysis

One more observation is made about the relative presence of one label in presence of another. This is done by calculating the co-occurrence matrix of all labels as shown in the heat-map below. This basically shows what percentage label A is present when there is already label B. One can see from following heat-map that 'primary' is present with almost all other labels except when it is 'cloudy'. Similarly, we see presence of most of other labels when label 'clear' is present, which also makes sense. We are seeing also a moderate relation between 'cultivation' or 'agriculture' with 'slash burn' and that is a tale-tell proof of deforestation.



Algorithms and Techniques

This project is all about image detection and classification. I have used Convolutional Neural Networks (CNNs / ConvNets) for processing the images in this project.

Convolutional Neural Networks are neural networks that are specifically designed to process image. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. They have a loss function on the last (fully-connected) layer and algorithms like stochastic gradient descent etc. for learning regular Neural Networks still apply.

Structure:
A ConvNet is made of Layers where every layer has its own structure. Here I have utilized following layers –

- INPUT [64x64x3] will hold the raw pixel values of the image, in this case an image of width 64, height 64, and with three color channels R, G, B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [64x64x32] if Idecided to use 32 filters.
- RELU (rectified linear unit) layer will apply an element wise activation function, such as the f(x) = max (0, x) with threshold at zero. As the problem we are solving has non-linear

nature, we need a non-linear unit in our processing pipeline. This layer introduces the desired non-linearity in the structure. This leaves the size of the volume unchanged ([64x64x32]).

- POOL layer will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as [32x32x32].
- FC (fully-connected) layer will compute the class scores, resulting in volume of size [1x1x17]. Each neuron in this layer will be connected to all the numbers in the previous volume.
- CL (Classification Layer) This is the final layer of the CNN that converts the output of FC to probability of each object being in a certain class. I have chosen 'sigmoid' as the final activation. Each unit will be followed by a 'sigmoid' activation which will produce an output label score where each of the 17 numbers correspond to an image label.

Benchmark

Each Kaggle competition has a leaderboard where competitors' scores are arranged in descending order. I looked at the range of results obtained by the leading competitors and arrived at a benchmark score for the algorithm.

The benchmark score was published when the proposal was made for this project. When I published the project proposal I had following observations.

After looking at the leaderboard for current competition at https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/leaderboard I saw that about 700 teams were competing and here were some of the score ranges –

~400 of them scored 90% or more
~350 scored 91% or more
~215 scored 92% or more
~56 scored 93% or more – TOP Tier with highest score being 93.385%

I chose a benchmark score of 91% as half of competitors have crossed that mark. I needed to better that score to consider our model as successful.

The leaderboard has changed since I arrived at this benchmark. Now there are about 900 teams and around 450 teams are above score 0.91. The highest score still remains 0.93. So, our assumption of crossing at least 0.91 to be in the upper 50th percentile still remains valid.

Methodology

Data Preprocessing

Our Convolutional Neural Network will process images through multiple layers and finally produce a classification label. However, it is not a single class. From the training data it is seen that multiple labels are applied against each image. Here are some samples –

```
image_name                                           tags
0    train_0                                haze primary
1    train_1              agriculture clear primary water
2    train_2                               clear primary
3    train_3                               clear primary
4    train_4  agriculture clear habitation primary road
```

So, I have to predict more than one class instead of just one. I have adopted something similar to one-hot-encoding for the labels. I have converted the tags to a fixed length sparse vector where only the corresponding places of labels we have to insert 1, otherwise there is 0 by default.

For example – we can have following vector for train_0 image –

```
[ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   1.   0.   0.   0.   0.]
```

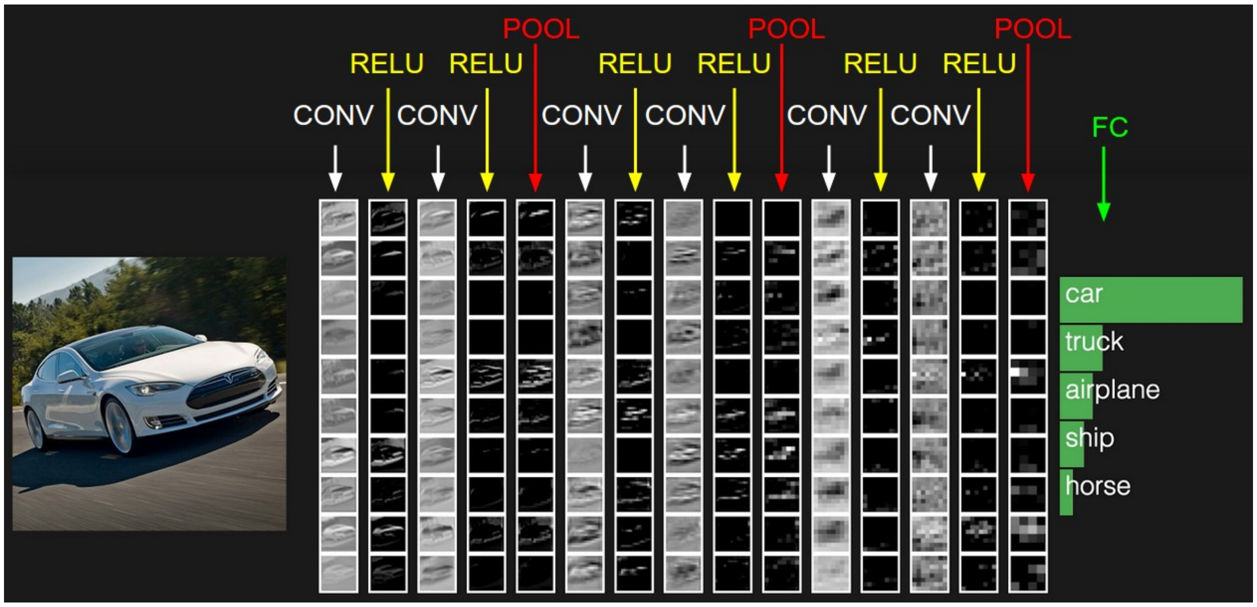
where 1 in 11th place and 13th place indicate presence of labels 'haze' and 'primary'.

Resizing and Normalization:

Before we train out model with input images we need to resize and normalize our data. For images that we received it is already 256x256 with 3 channel RGB format. However, our GPU has only 6GB memory and it would be tough to feed image size of that dimension. So, I resized the images to size 64x64 with 3 RGB channels. For normalization, I divided the pixel values by 255 to contain them between 0 and 1.

Implementation

The RGB 3 channel input is applied to each layer consisting of a convolution, max-pooling and RELU activation. This is followed by a dropout layer. This comprises a typical processing layer for a fixed output size. I have applied output size of 32, 64, 128 and 256 being followed by a fully connected layer of size 512. The output layer consists of as many output as there are number of labels (17). Each neuron's coefficients are computed during training using the training data. Basically, training is done through back propagation algorithm with gradient descent. Following figure depicts the multiple layer of processing that takes place.



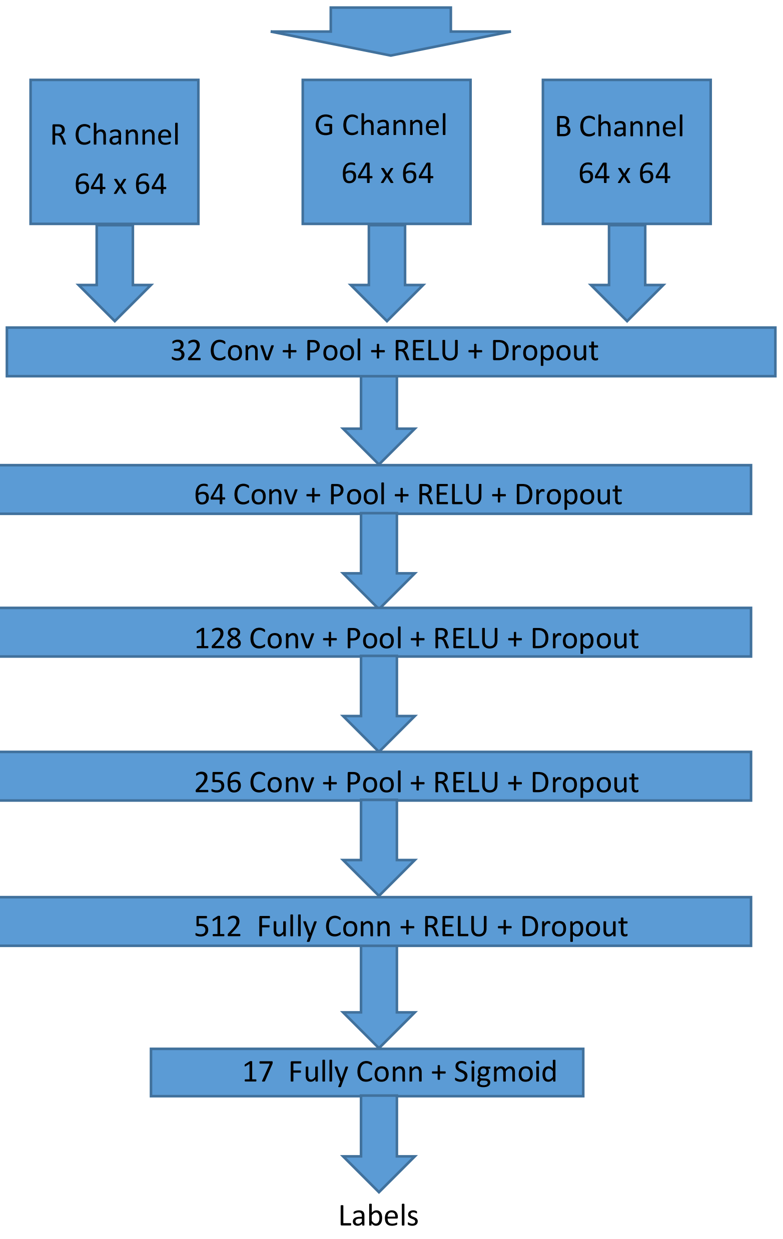
Convolutional Networks (ConvNet) uses mainly three type of layers – CONV (convolutional) +POOL (pooling), FC (fully-connected) and Classification layer. Image goes from Input layer through these processing layers finally to output or classification layers.

Input: At the very start of processing I have images with sizes as 256x256. Since our computation capability is limited, I have resized the images to 64x64. The pixel values are normalized between 0 and 1 (by dividing with 255). There are 3 channels of input data for R, G and B.

Convolutional + RELU+ Pool + Dropout Layer: This is the filtering layer where we apply 2D convolution of dimension 3x3 over the image area where each filter will compute a dot product between their weights and a small region they are connected to in the input. I have Convolutional layers using small filters (3x3) and stride S = 1 and 'same' padding that results in padding the input such that the output has the same length as the original input. This results in volume of output [64x64x32] as there are 32 filter to begin with. The output of each element is applied RELU (Rectifier Linear Unit) activation function. POOL layer will perform a down-sampling operation along the spatial dimensions (width, height). Random dropout is applied with a fixed retain-rate to aid regularization. We have a repetition here of this (Convolution + RELU + Pool + Dropout) layer with varying number of filters starting with 32 and then 64, 128 and 256.

Fully Connected (FC) Layer: This layer will reduce the size of input data to the size of classes that the CNN is trained for by combining output of CNV layer with different weights. Each neuron at the output of the CNV layer will be connected to all other neurons after weighted properly, Similar to CNV layer, weight of these taps in FC layer is found though back propagation algorithm.

Classification Layer (CL): This is the final layer of the CNN that converts the output of FC to probability of each object being in a certain class. I have chosen ‘sigmoid’ as the final activation.


R Channel
64 x 64
G Channel
64 x 64
B Channel
64 x 64
32 Conv + Pool + RELU + Dropout
64 Conv + Pool + RELU + Dropout
128 Conv + Pool + RELU + Dropout
256 Conv + Pool + RELU + Dropout
512 Fully Conn + RELU + Dropout
17 Fully Conn + Sigmoid
Labels

For implementation Ihave used Keras library (https://keras.io) and TensorFlow (https://www.tensorflow.org/) backend to implement each layer. For computation I have used GPU NVidia GeForce GTX 1060 installed as a graphics card on a Linux desktop.

Training:

In the training phase I train the neural units with input images and labels and try to minimize the error or the loss (actually I compute binary cross-entropy as defined by TensorFlow). This reduces the problem to an optimization task. To solve such problem, I take advantage of a well known algorithms like stochastic gradient descent (I actually use a variation called Adam optimizer offered by TensorFlow). This algorithm depends on the rate or gradient of the loss function which then is used to adjust weights of each layer. This backward flow of gradient is called back-propagation.

One of the important parameters of gradient descent is the learning rate that helps gradient descent to converge slower or faster but with different qualities of weights achieved. I start with an initial learning rate and then decrease it over several iterations. The number of full cycles or 'epochs' and the 'learning rate' are among the important parameters that I tune.

Validation:

I use a part of training data and validate the model based on the loss (Tensor flow option 'val_loss') produced on validation samples. I actually use a technique called 'KFold cross validation' to run cross validation over the whole range of training images. I checkpoint the model weights based on better validation loss. I also compute the F1 score and accuracy of the whole training set to arrive at the best model and the parameters.

Prediction:

The test images are applied at the input of the model with best weights loaded. The output levels produced at the final fully connected layer (FC) as subjected to a 'sigmoid' activation is again applied a threshold value for predicting labels. This threshold is obtained by trial and based on the principle to maximize the F1 score. The output labels are then visualized to conform their distribution to be in line with the training images.

Refinement

Threshold Calculation: The output labels are not 1 or 0 but output of sigmoid function which can range from 0 to 1. To arrive at a label, I have to compare the output to a predefined threshold and arrive at a decision about presence of a label. I have adopted an algorithm that will choose the threshold which will maximize the F1 Score computed with that threshold. I have found a threshold of 0.2 (for all labels) provides maximum F1 score (0.93274).

Results

Model Evaluation and Validation

Trusted Model:
I have adopted ConvNet which is a proven technique for image detection and experimented with its layers and parameters.

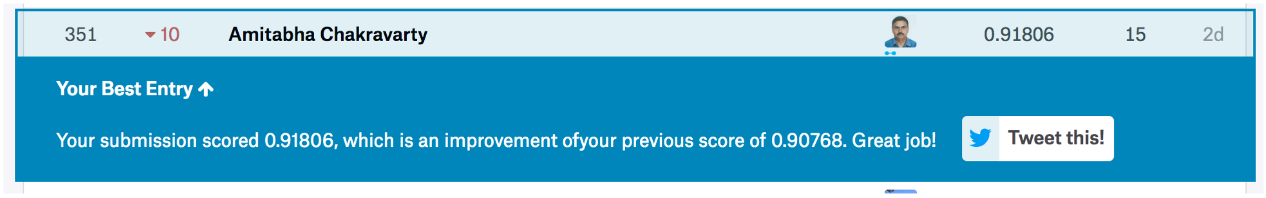
For the learning I have optimized the cross-entropy of the predicted vs actual label-vector. Data is analyzed in batches and best models is check-pointed. I have used K-Fold technique of split and validation to separate training set from validation set and rotate this process multiple times over the whole training set.

I have experimented with parameters like 'learning-rate', 'pooling frequency', convolution strides and k-value, number of layers etc. to arrive at the optimal model as shown above.

I have seen the model converge to optimum weights with validation loss continuously going down and accuracy improving.

I have obtained F1 Score of 0.93274 and accuracy of 0.57313 while iterating the model over the whole training set.

The testing dataset was used to predict final score for the derived model and I have achieved 91.8%



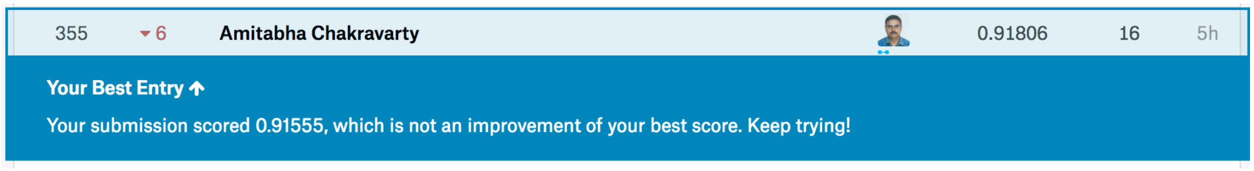
351 ▾10 Amitabha Chakravarty 0.91806 15 2d

Your Best Entry ↑

Your submission scored 0.91806, which is an improvement ofyour previous score of 0.90768. Great job!

Model reliability:
The output labels were plotted for their distribution and it displayed similar proportion as found in the training images.

Model Robustness:
I have subjected the input data to transformation like 'vertical-flip', 'horizontal-flip', height and width 'shift' etc. and trained the model from scratch again. I subjected the model to same test data as before and obtained a score of 91.5 % (see below).
So it proves that small perturbations (changes) in training data or the input space does not affect the results significantly.


355 ▾6 Amitabha Chakravarty 0.91806 16 5h

Your Best Entry ↑

Your submission scored 0.91555, which is not an improvement of your best score. Keep trying!

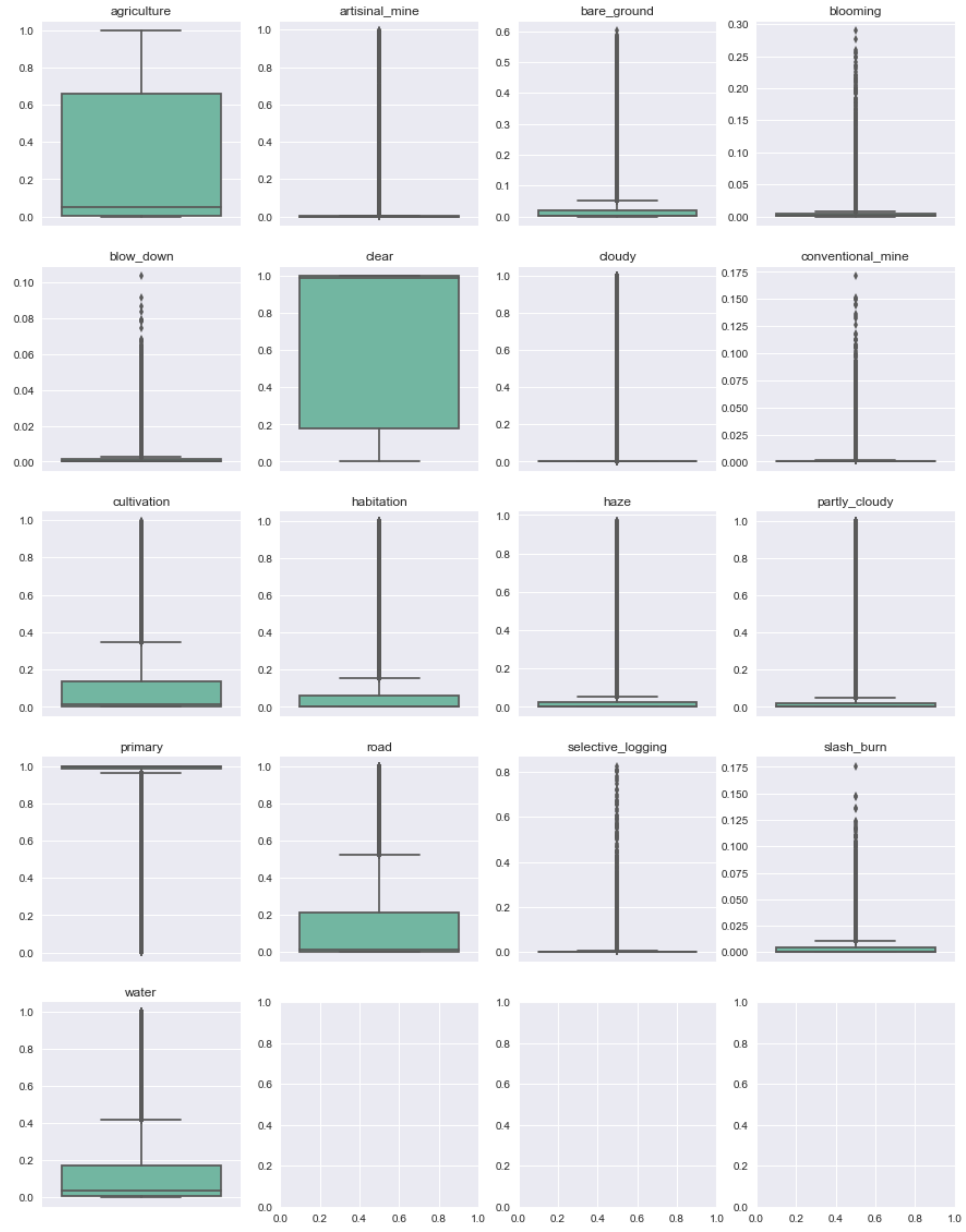
Justification:

Since the benchmark score goal was set as 91% our result of 91.8 % turned out to be stronger. However, there is still room for improvement but that depends on computing resource limitations. One improvement I could try is to increasing the size of input images to 256x256. However, this would require more memory for the GPU. Other option could have been adopting other trusted models like 'Inception' which also would have faced resource limitation issues.

Conclusion

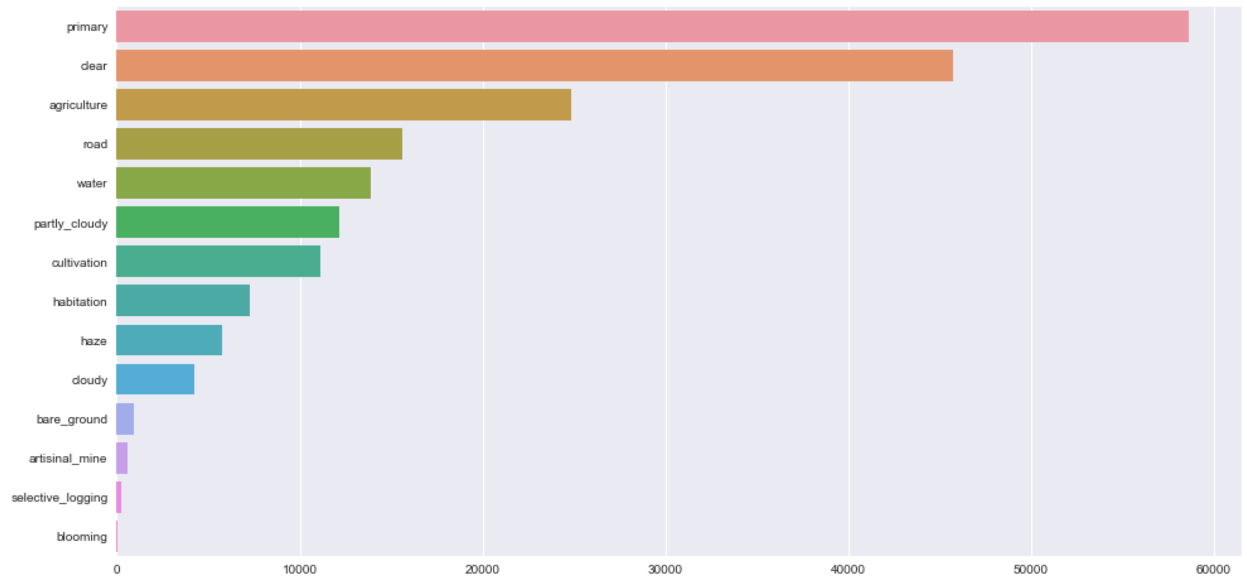
Free-Form Visualization

The output level for each prediction is between 0 and 1 and following box plot represents relative frequency of each label with median and inter-quartile range clearly indicated. For example, 'primary' is always present and it is hovering around 1.0. So is 'clear' label. 'Cultivation', 'road' and 'water' have moderate presence (0.1 to 0.2). These predictions are all in expected line.



agriculture
artisinal_mine
bare_ground
blooming
blow_down
clear
cloudy
conventional_mine
cultivation
habitation
haze
partly_cloudy
primary
road
selective_logging
slash_burn
water

The distribution of predicted labels is also evident from bar plot that is drawn below. Here the maximum length is occupied by 'primary' followed closely by 'clear'. Then we have moderate presence of 'agriculture', 'road' and 'water'. Important entries are seen at the bottom of the chart. The 'selective logging' and 'blooming' are detected and are also in line with expectation with their lower spread. This is also seen in the bar plot obtained from the input training data.



Reflection

This project was all about image classification. With the goal in mind to achieve a good classification score, I have adopted a trusted model like ConvNet which is a proven technique for image detection and classification. For learning I have used 64x64 size of images and optimized the cross-entropy of the predicted vs actual label-vector. I have used K-Fold technique of split and validation to separate training set from validation set and rotate this process multiple times over the whole training set. I have experimented with parameters like 'learning-rate', 'pooling frequency', convolution strides and k-value, number of layers etc. to arrive at the optimal model. Finally, I produced the labels from the testing images and plotted the distribution which matched statistics obtained from the input training data. However, the final model was arrived after a few iterations and my own learning.

Initially I started out with TensorFlow library alone without any wrapper like Keras. The difficulty was to maintain the code as I was continuously modifying layers and changing parameters. Then I switched on to Keras and modifying the code became simpler.

Initially I ran the training with small batches of training data. I was apprehensive about using the whole batch of training data in one go as I feared about out-of-memory scenario. I got reasonable score of around 0.86. I was adjusting the batch lengths and also the filter numbers. However, the score was not improving much. Then I switched to KFold way of cross-validation and used the whole training set in one go. This dramatically improved the score – 0.918.

After achieving a good score of 0.918 I was trying to surpass that by changing the image size to 128x128 which straight-way gave memory error in GPU.

Next I tried augmenting training set by using ImageDataGenerator in Keras that offers functionality like 'vertical/horizontal flip', 'height/width shift' and so on. Initially I was generating such images 'inline' while training. This process was so slow that I had to stop training altogether. Then I generated a whole batch of training data offline and used it in training the model. This gave a good score of 0.915 but this did not surpass the score of 0.918.

Than I got this idea of mixing original and augmented images. I used the best weight for the original training (that gave 0.918 score) and after loading this weight I tried to train with offline augmented data. This was like doing the training twice, with original set and then the augmented set. This gave a score of 0.917, still less than the original score of 0.918. Here I stopped changing the model.

Here was my learning – It was evident that ConvNet fitted the image data fairly well and with a choice of good number of layers and with KFold corss validation the output classification turned out to be pretty accurate. However, there was always the contradiction on how much accuracy one could achieve with the given limitation of computing power. Interestingly, the TensorFlow and Keras library worked smoothly when size of input was within permissible limits. With 6GB GPU card and 16GB desktop memory I was successful in processing 64x64 size of images but it failed miserably with Out of Memory error when image size was increased to 128x128.

Finally, it was nice to see the output plot of labels which resembled similar distribution as found in the training images. Even the presence of labels like 'blooming' and 'selective logging' proved that the model served its real purpose of this project of locating the deforestation areas.

Improvement

There is still scope of improvement as one could increase the image size with better GPU card. I could also try running inline generation of image data using ImageDataGenerator on a more powerful machine. One could even think about using a better model like 'Inception' for this project which of course would take more processing power. With the current limitation of computing power and the given data size the current ConvNet model turned out to be a good compromise with effective score of 91.8% on Kaggle Leaderboard.

References

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

- Convolutional Neural Networks (CNNs / ConvNets) - by Andrej Karpathy (karpathy@cs.stanford.edu) - http://cs231n.github.io/convolutional-networks/

- https://en.wikipedia.org/wiki/Gradient_descent

- https://en.wikipedia.org/wiki/Stochastic_gradient_descent

- https://en.wikipedia.org/wiki/Backpropagation

- https://www.tensorflow.org/

- https://en.wikipedia.org/wiki/Precision_and_recall

- http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

- https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html