CAPSTONE PROJECT

Machine Learning Engineer Nanodegree
Kaggle Competition - Planet: Understanding the Amazon from Space

Jordan Carson
July 6th, 2018

DEFINITION

PROJECT OVERVIEW

Climate change is drastically affecting the habitat in the Amazon and there is an on-going concern that the Amazon will be caught up in a set of "feedback loops" that could drastically speed up the pace of forest lost and degradation, leading to a point of no return.

According to Kaggle, “Every minute, the world loses an area of forest the size of 48 football fields. And deforestation in the Amazon Basin accounts for the largest share, contributing to reduced biodiversity, habitat loss, climate change, and other devastating effects. But better data about the location of deforestation and human encroachment on forests can help governments and local stakeholders respond more quickly and effectively.”[1]

In this project, I will be creating a classification system to label satellite image chips with atmospheric conditions and various classes of land cover/land use. This is related to the Kaggle competition titled ‘Planet: Understanding the Amazon from

[1] This is found under the Competition homepage n on the Kaggle competition page Planet: Understanding the Amazon From Space.

Space'. The inspiration for this project came from reading about how companies are leveraging alternative data to get an 'edge' within the financial markets.[2]

Problem Statement

The goal of this project is to train and build a deep learning model that is able to accurately classify images from a noisy dataset with ambiguous features. The following steps are:

1. Download the given dataset of satellite images of the Amazon.
2. Preprocess the training data using image augmentation techniques described in later sections.
3. Develop an algorithm to properly classify each image with its associated atmospheric/land/weather tag.

The solution will be geared towards implementing image augmentation techniques to assist a model to properly classify images based on probability. We essentially want to determine from an unknown labelled image what the contents of the image contain based on a set of tags. There are 17 possible tags: agriculture, artisinal_mine, bare_ground, blooming, blow_down, clear, cloudy, conventional_mine, cultivation, habitation, haze, partly_cloudy, primary, road, selective_logging, slash_burn, water.

Evaluation Metrics

The deep learning model will be evaluated based on the mean F2 score. The F score, commonly used in information retrieval, measures accuracy using the precision

[2] Articles such as this from the Financial Times: https://www.ft.com/content/2f454550-02c8-11e8-9650-9c0ad2d7c5b5

p and recall r. Essentially, F2 score is a way of combining precision and recall[3] into a single score – like the F1[4] score, but with recall weighted higher than precision. Thus, we needed not only to train our models to predict label probabilities, but also to select the optimum thresholds to determine whether or not to select a label gives its probability. We used a helper function to obtain our optimal thresholds. See the below function as the evaluation metric.

$$(1+\beta^2)\frac{pr}{\beta^2 p+r}\ where\ p=\frac{tp}{tp+fp}, r=\frac{tp}{tp+fn}, \beta=2$$

Although the final solution and accuracy will be based on the F2 score above, I also use log-loss as another measure of accuracy because it brings in probability confidence[5]. Essentially, it is the cross entropy between the distribution of the true labels and the model's predictions. Cross-entropy incorporates the deterioration of the distribution, plus the unpredictability when one assumes a different distribution than the true distribution (Kingma, Ba). Therefore, log-loss is a measure to gauge the noise that comes from using a predictor as opposed to the true labels. We must minimize the cross-entropy, or the log-loss as they are essentially the same, to maximize the accuracy of our proposed classifier and obtain a passable F2 score.

[3] Precision is the ratio of true positives (tp) to all predicted positives (tp + fp). Recall is the ratio of true positives to all actual positives (tp + fp).

[4] F1 score is computed as $f_1 = 2\frac{p\cdot r}{p+r}$

[5] When training our model, we minimize the 'binary cross-entropy' which is essentially the log-loss function converted to a binary representation.

ANALYSIS

DATA EXPLORATION

Fig 1: The datasets for this project are all located in Kaggle[6]. According to Kaggle, "The chips for this competition were derived from Planet's full-frame analytic scene products using our 4-band satellites in sun-synchronous orbit (SSO) and International Space Station (ISS) orbit.

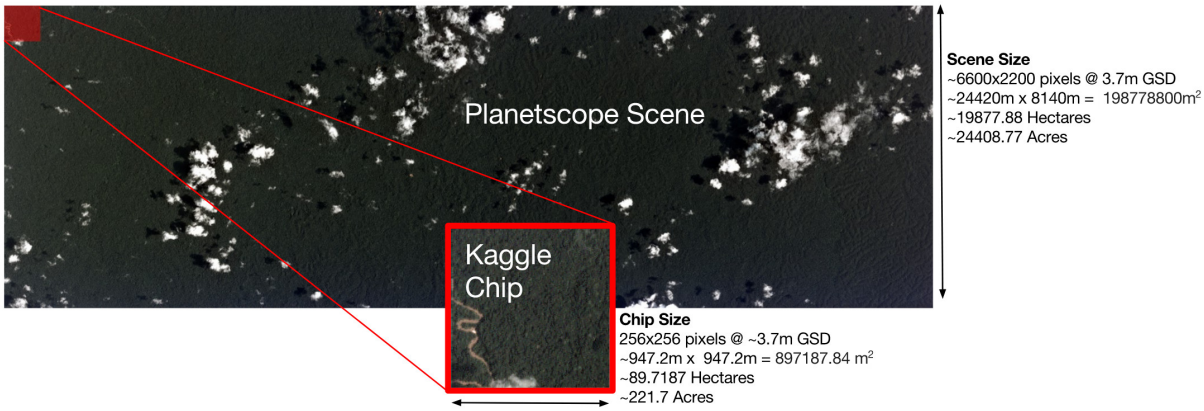


Fig 2: The set of chips for this competition use the GeoTiff format and each contain four bands of data: red, green, blue, and near infrared. The specific spectral response of the satellites can be found in the Planet documentation. Each of these channels is in 16-bit digital number format and meets the specification of the Planet four band analytic ortho scene product."[7]

[6] Kaggle is a data science website that hosts data science competitions and house open-sourced data.
[7] This is found under the Data Description on the Kaggle competition page Planet: Understanding the Amazon From Space.

This Kaggle competition contains over 40,000 training images, and each training image could contain a label with multiple tags. The 17 possible tags/labels can broadly be broken into three groups: atmospheric conditions, common land cover/land use phenomena, and rare land cover/land use phenomena. Each chip – a chip size is 256x256 pixels – will have at least one and the potential to have more than one atmospheric label and zero or more common and rare labels. (See above for the sample chips and their labels as provided by Planet's impact team.)

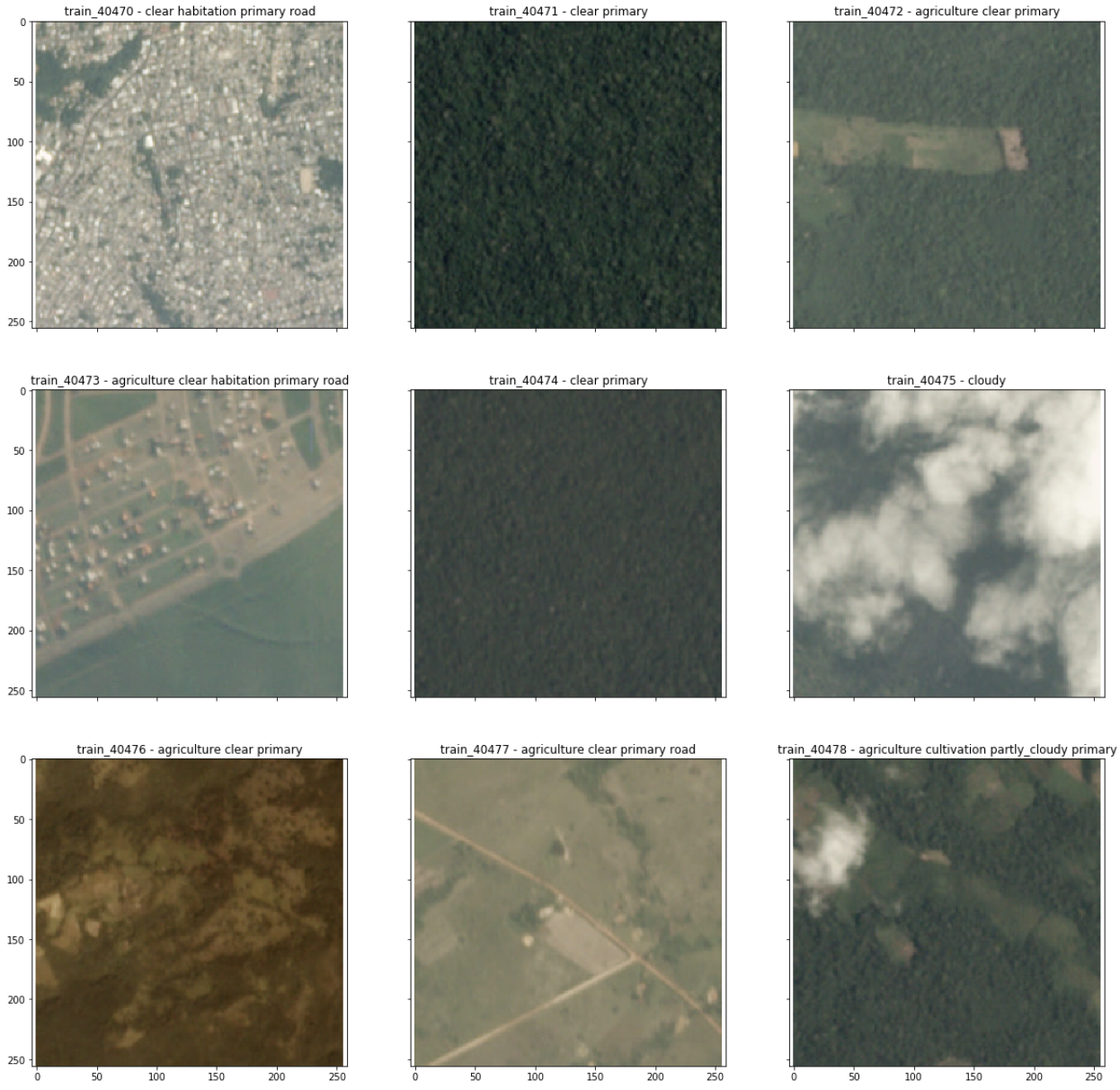
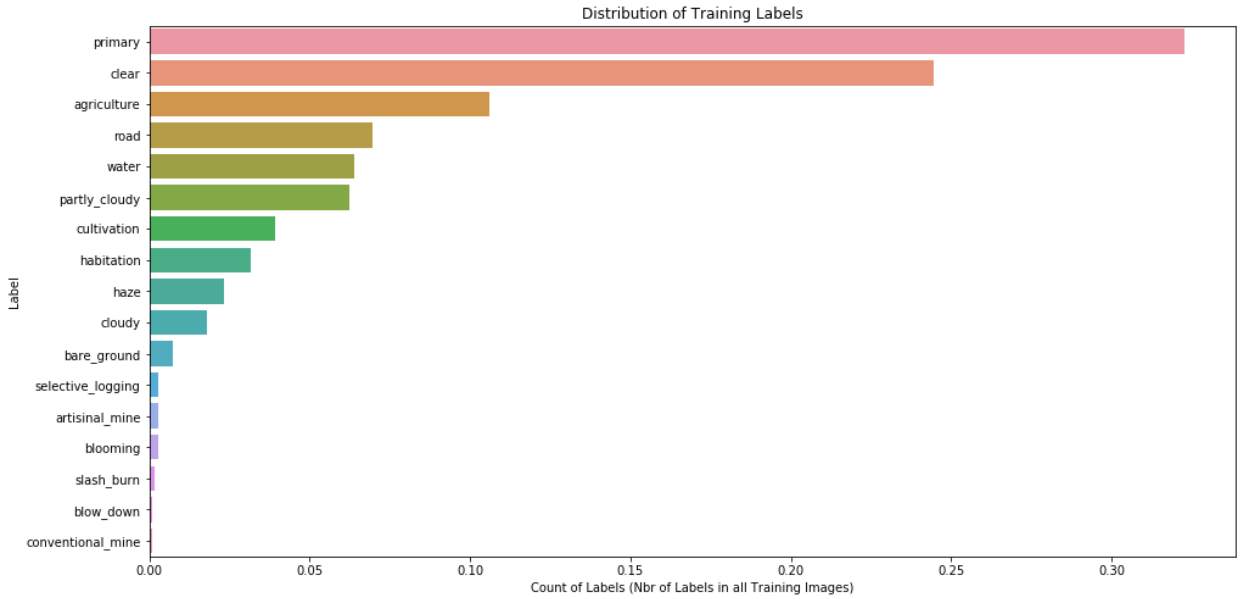


Fig 3: (Image Above) The chip labels can be boisterous due to the labeling process and ambiguity of features, and scenes may omit class labels or have incorrect class labels. This problem and Kaggle challenge is to figure out how to work with noisy data and features that are ambiguous. Below are the last 9 training images in the training data-set – note all labels and images are random and the tag ‘clear’ can be associated with ‘hazy’-like imagery.

EXPLORATORY VISUALIZATION

Fig 4: The plot below shows the distribution of tags by each label name.

Kaggle distributes the image data in both training and test sets in jpg and tiff file extensions. In this project I used the jpg images for both training and testing.



It can be seen that:

- Primary & Clear account for >55% of the training data-set
- The distribution of the training labels shows that the data is not evenly distributed.

Fig 5: Below is a sample of the training labels data-set (See the Data Processing

section to infer how the training data is processed).

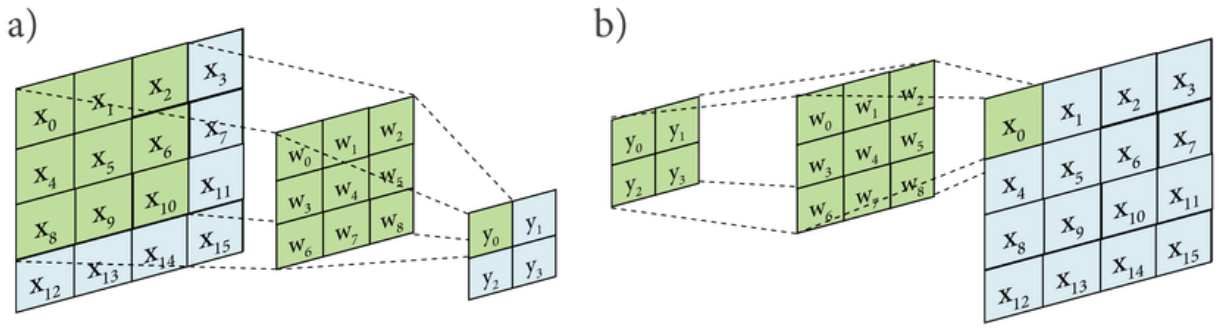
	image_name	tags
0	train_0	haze primary
1	train_1	agriculture clear primary water
2	train_2	clear primary
3	train_3	clear primary
4	train_4	agriculture clear habitation primary road

Algorithms and Techniques

The classifier is a Convolutional Neural Network, which is the state-of-the-art algorithm for most image processing tasks, including classification. The input of a ConvNet is typically a 2D image with RGB (red, green blue ← 3) channels. ConvNet's obtain their name from the **convolution** operation, which is an operation of filtering the input data. These convolution filters are used to process data at different layers to generate features. The filters are defined before training, and during training their coefficients are computed. This is done via backpropagation with gradient descent (Zhou 19).

As we can see below from the example, the output of the kernel is the altered image, which is called a feature map. We then apply an element wise multiplication with the image path and the convolution kernel. After one pixel of the feature map has been computed, the center of the image patch extractor slides one pixel into another direction and repeats this computation. The computation ends when all pixels of the feature map have been computed this way.

Fig 6. Below is an example of a discrete convolution (a) and equivalent transposed convolution operation (b) for a 3x3 filter kernel size applied to a 4x4 feature map. The active regions to compute the output value are shaded green.



In particular, unlike regular neural networks, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth.** Every layer of a convolutional neural network transforms the 2D input to a 2D output of neuron activations. The depth typically refers to the RGB channel.

The following parameters can be [fine]tuned to optimize the classifier:

- Optimal F2 Classification Threshold see Implementation section for more information.
- Training parameters:
  - Training length (number of epochs)
  - Batch size (number of images to look at during one single training step)
  - Weight decay
  - Learning rate (how fast to learn)
  - Size of network architecture (i.e. nbr of Conv2d, Dense, Maxpooling layers)
  - Training/Testing Image input size (e.g. controlling the size of the images throughout the training process).

- Neural Network architecture
    - Number of layers
    - Layer types (convolutional, fully-connected, pooling)
    - Layer parameters
- Preprocessing parameters (see the Data Preprocessing section)

When running our program, all training & validation data-sets are loaded into the computers RAM. After this, random batches are selected to be loaded into the GPU[8] memory for processing. The training is done in batches of 128, meaning this is the number of samples to be propagated through the network at one time.

BENCHMARK

Without using a deep learning model, we hard-coded two sets of binary lists, (to be the supposed output of the model) and manually mapped[9] these lists to the labels and predictions. From this, we were able to obtain a f2 score of ~61%. This will be our target benchmark rate to beat using deep learning methods. Additionally, there have been cases posted to Kaggle kernels of individuals achieving rates in the low ~90%s. This is a more aspirational goal as these individuals used deep learning and image processing techniques to greatly improve their accuracy score.

METHODOLOGY

DATA PREPROCESSING

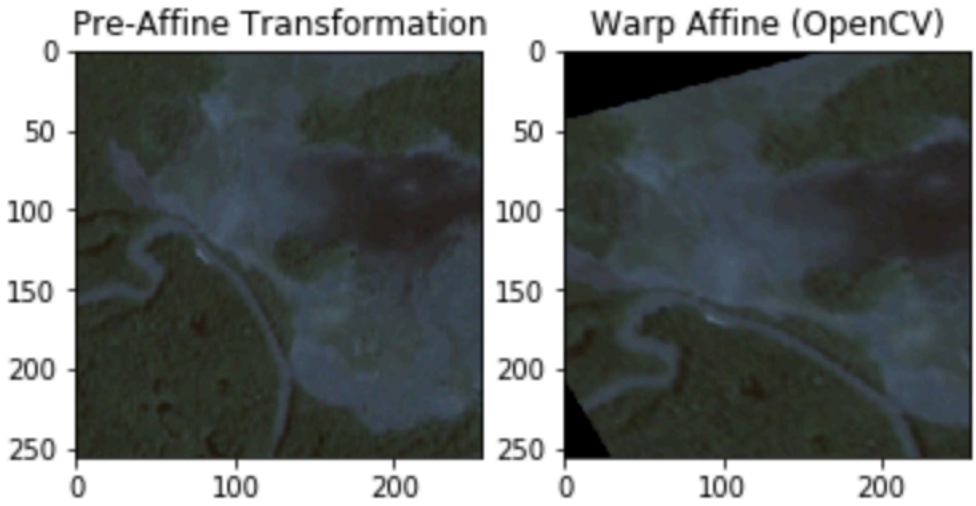
[8] GPU are available in both Kaggle Kernels, Udacity Workspaces, and AWS EC2 instance (described below).
[9] This solution only applies to this data-set.

There are a number of steps done before we are ready to process the images into our deep learning model. We use the OpenCV library for reading and processing jpg images, and the Keras (TensorFlow) machine learning library for building our final deep learning model that is used to create our submission file for this Kaggle competition. Below is the order and steps taken in order to prepare our training data:

- Download the training images (JPG), validation/testing images (JPG), training labels, and submission file via Kaggle API (see appendix for more information)
- Use 7zip or an extracting/archiving tool to unzip the 7z file to a tar, then unzip the tar to obtain the folder of images. Unzip the labels and submission archives.
- Read and shuffle the training labels
- Split each label on the space (i.e. 'haze primary') and create a list of labels for each training image
- Read the images and map them to the labels map based off the training image name (training labels have both image and label)
- Resize the images, to 128 X 128 – here we can resize the images to whatever size we want, however we need to be consistent with what shape we are sending as an input to our deep learning model. Each image has a shape of (rows, cols, channel) -> (256, 256, 3)
- We then flip all images and apply a geometric image transformation called warpAffine, which applies an affine transformation to an image.

Fig 7. The geometric transformation does not change the content of the image, but rather deforms the pixel grip and then remaps the deformed grid back to the destination image. An Affine transformation preserves points, straight lines, planes and keeps the integrity of parallel lines; it is essentially a linear transformation of position vectors followed by a translation (rotation). We apply a warp affine transformation while

flipping all images every 90 degrees.[10] We apply image augmentation techniques to our training/testing data in attempt to increase the number of **relevant features** in our dataset.[11]



IMPLEMENTATION (CNN ARCHITECTURE)

The implementation process can be split into three main stages:

1. Preparing the data
2. Training the classifier
3. Testing the classifier

During the first stage, I used the Kaggle API to download and preprocess the data (refer to Data Processing section for more information). Next, the classifier was trained on the preprocessed training data. This can be divided into the following steps:

1. Load the training images and testing images into memory, preprocess them as described in the previous section

[10] After applying this affine transformation, my model's accuracy increases from 88.3% to 89.8%
[11] Machine learning algorithms work but distinguishing the most obvious features of one class from another. Here we want to increase those obvious features to guide our algorithm to learning what it needs to.

2. Implement helper functions:
    a. _init_attributes(..): Member of BaseClass, which parses all arguments in a dictionary and assigns them as instance variables. (This is designed to mimic a server to pass instance variables to classes)
        i. i.e. batch_size, nfolds, path_to_files...
    b. train (): helper function to preprocess all training data and begin training the classifier
    c. predict(..): helper function to take the classifier and apply it to all the testing images, to return the final prediction dataframe
    d. process_training_data(..): takes all the training data and prepares it for the training classifier, here we implement our geometric image augmentation techniques.
    e. build_training_classifier(..): takes in the processed training data (x_train, y_train), and builds a deep CNN via Keras.
    f. optimal_thresholds(..): takes in the true labels, predicted labels and a number of iterations, and outputs a list of optimal thresholds for each label classification.
3. Define the network architecture and training parameters
4. Define the loss function, here we choose binary cross-entropy as our loss function
5. Train the network (add Log directory for TensorBoard, this will be used to create the network computational graph and to show our training validation loss and accuracy)[12]
6. Keep repeating until we achieve our optimal F2 / log-loss score
7. Optimize the training network architecture and training parameters, Keras callbacks to assist with optimizing training parameters
8. Save and free the trained network, save the best model weights
9. Load the best model weights and apply our classifier to the testing data

[12] After training has completed, run Tensorboard –logdir=[path_to_directory] from the terminal and open up the url the terminal provides (this will provide you with a graph of the logged values)

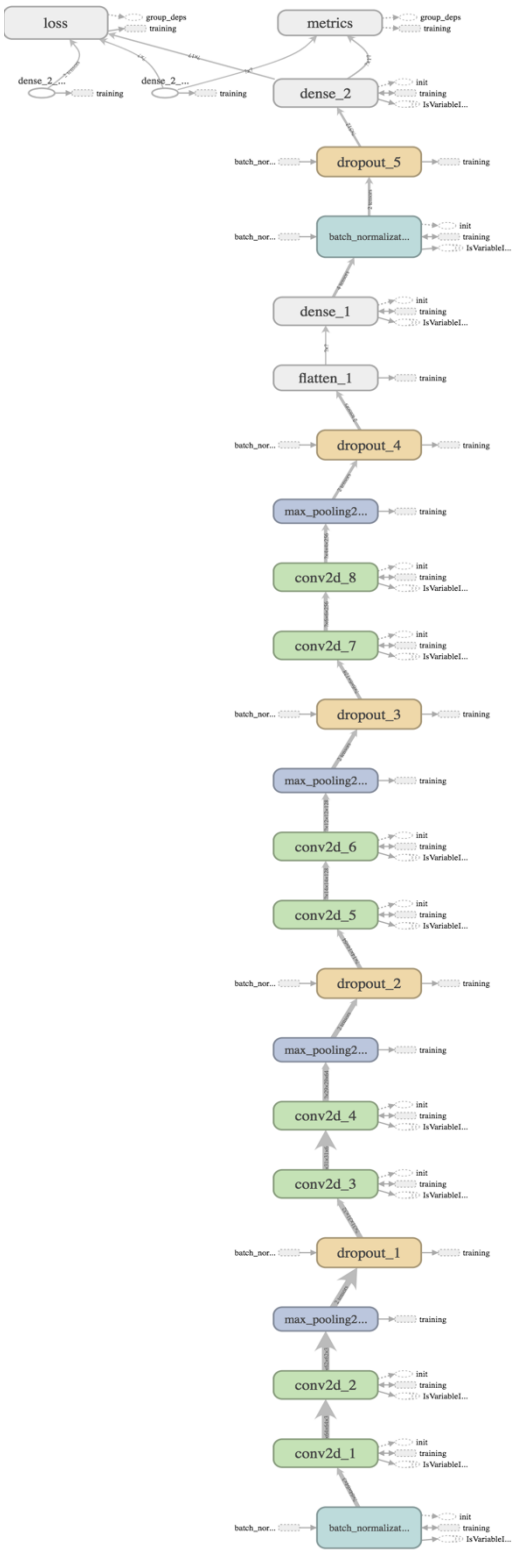


Fig 8. The image on the left shows the computational graph (taken from TensorBoard colored by 'Structure'), which includes the network architecture for each fold and also the variables that are only available during training.

The acronyms can be read as:

Conv2d = Convolutional Layer
Dropout = Dropout Layer
Max Pooling = Max Pooling Layer
Dense = Fully Connected Layer

The **conv2d** layer that feeds the **max pooling** layer contains both the activation operation that is applied to each elements after convolution and subsampling of the data after activation. The **dense** (fully connected) layer will reduce the size of the input data to the size of the classes that the CNN is trained. The **dropout** layer is a regularization layer used to ignore random neurons[13], in attempt to reduce overfitting.

We cannot see from the graph, but our loss function is composed of binary cross entropy and not categorical cross entropy as it normalizes the output before calculating the loss.

[13] During the training phase, a certain set of neurons is chose at random, which are not considered during a particular forward or backward pass via networks. These are either dropped or kept, so that a reduced network is left.

REFINEMENT

As mentioned in the Benchmark section, manually mapping a binary variable to the f2 benchmark score, we receive an accuracy rating of around 80%. Note that although this is a relatively good accuracy score, the binary classification was selected manually which will only give us this score when mapping to our training data set.

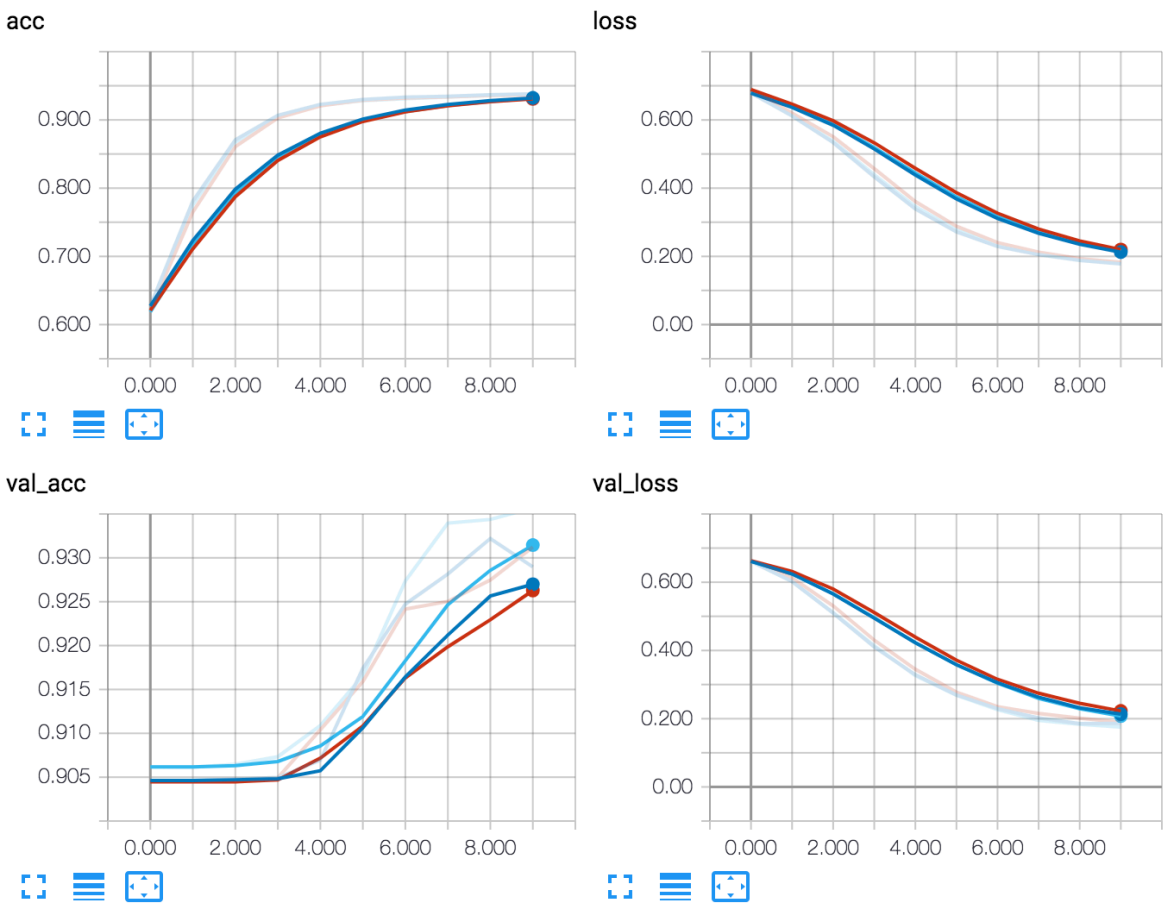
To get our initial result, we built a Keras CNN deep learning model, the result was a true positive (F2 score) of 82.69%. We continued to build and this was dramatically improved upon by using the following techniques:

- We stop training after the monitored quantity has stopped improving, this can be seen in the Keras Callback called EarlyStopping.
- We reduce the learning rate after a metric has stopped improving. According to Keras documentation, “Models often benefit from reducing the learning rate by a factor of 2-10 one learning stagnates. This callback monitors a quantity and if no improvement is seen for a ‘patience’ number of epochs, the learning rate is reduced.” This is known as the Keras callback ReduceLROnPlateau.
- Lastly, we save the model after every epoch. Then selecting only the model with the best weights during evaluation. This is known as the Keras callback ModelCheckpoint.

The final Keras model was derived by training in an iterative fashion, adjusting the model parameters (i.e. learning rate, weight decay ratios) based on plots like the one below. The final model has an accuracy of 93.278%.

The below plot shows the training validation losses and accuracy. Note this is not the F2 score. Our final F2 score across all 17 thresholds is 0.8918.

Fig 9. Below is a chart from TensorBoard showing the models accuracy, and loss. During our models training we set-up Keras callbacks to monitor the val_loss and perform callbacks as denoted in the Refinement section above.



RESULTS

MODEL EVALUATION AND VALIDATION

After the competition was finished, Kaggle released the private validation set into the competitions public data directory. This validation set was used to evaluate the model.

The final architecture and hyperparameters were chosen because they performed the best among the tried combinations.

For a complete description of the final model and the training process, refer to **Figure 6** along with the following list:

- The input shape to the first Batch Normalization layer is 128 x 128, the size of the re-sized images.
- The first convolutional layer learns and outputs 32 filters, the second 64, the third 128 and the fourth 256.
- The activation function for all CNNs in our network is ReLU – rectified linear unit.[14]
- The kernel size to all the convolutional layers is 3 x 3, which specifies the height and width of the 2D convolutional window.
- The max pooling layer has a pool size of 2 x 2, which will have the resolution.
- The training runs for 30 epochs, 3 Kfolds, and to get our optimal f2 thresholds we run our function ~ 10,000 iterations.
- The model is compiled and optimized with Adam, providing an input learning rate of 10^{-4}.
- The first fully connected layer learns and outputs 512 filters, the second learns and outputs 17 filters (note we are given 17 different tags).

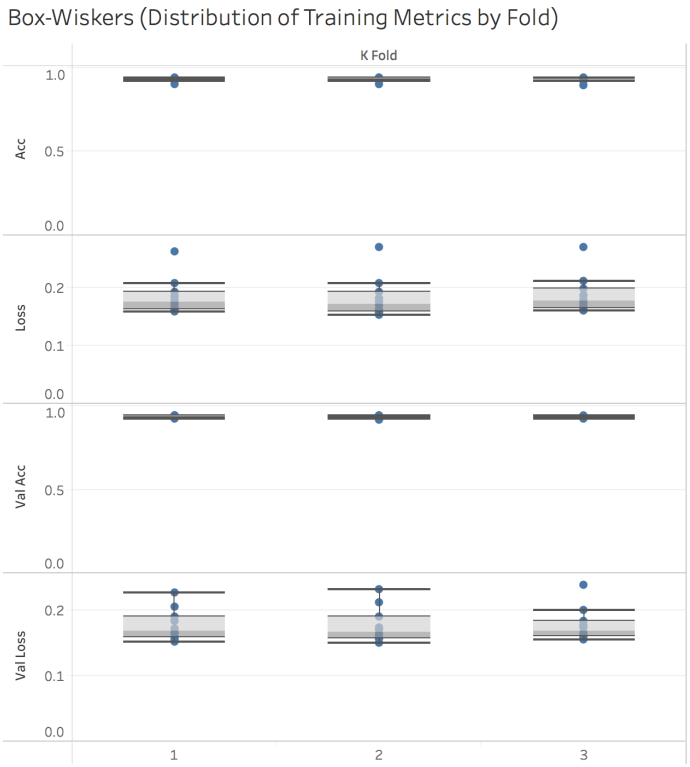
Using the above network architecture, hyper-parameters and pre-processed input data our final model produces an F2 accuracy score of 89% (true positives) after 18 hours of

[14] Activation operation is typically a nonlinear operation that is applied to each elements of convolution output such as max(0, x) a.k.a relu or $\frac{1}{1+e^x}$ where x is the input data. This does not change the size of the input, instead subsampling is applied after activation that will halve the size of the input for each dimension.

training on ubuntu AWS[15] p2.xlarge EC2 instance (GPU capable). See below improvement section for how these results and model could be improved.

We can see from **Fig 9** that over all epochs the models loss and accuracy remained relatively consistent. Thus, we can say without a reasonable doubt that our model fits to the input data, training parameters and image augmentation techniques used. Additionally, the performance o

Fig 10. Below is a chart showing the distribution of training metrics by Kfold. We can see that the data remains consistent throughout each fold.



16

[15] We choose AWS AMI (Amazon Machine Image) to be Deep Learning AMI with Source Code (CUDA 8, Ubuntu) – in the Artificial Intelligence Nanodegree they help setting up AWS using this AMI.
[16] This was built using Tableau Public 2018.2

JUSTIFICATION

The best performing model was the model with the most complex network architecture[17]. We trained over 18 different models all with different training parameters, with consistent data preprocessing. Our initial model has 2 Convolution layers, 2 Maxpooling layers, 2 dropouts, 1 flatten, & 2 fully-connected layers. We activate the convolutional layers using ReLU, while the fully-connected layers use the sigmoid activation function and optimize using adamax. In earlier iterations, we choose to use the following training parameters: *[N] below denotes the model index*

- Learning rate: 10^{-4}
- Batch-Size: 128
- Dropout: [8] layer (0.2, 0.1)
- Maxpooling: pool_size (2x2), strides (2x2)

Fig 11. The table below shows the top three models (in terms of F2 Score) by model type. A simple model would be one fully connected layer, while a complex model will be one of multi-layers with different shape ConvNets, ResNet/DenseNets.

Index	Model Type	Description	F2 Score	Log Loss
1	Simple	Single Layer	0.8081	0.79109
2	Medium	Multi-Layer	0.8269	0.81872
8	Complex	Multi-Layer	0.8627	0.93842
18	Complex	Multi-Layer	0.8918	0.93753

It is important to note that a low log-loss does not necessarily result in a good F2 score, and vice versa - a high log-loss does not necessarily result in a good F2 score.

[17] In my experience, simple architectures with complex data processing techniques tend to outperform more complex architectures.

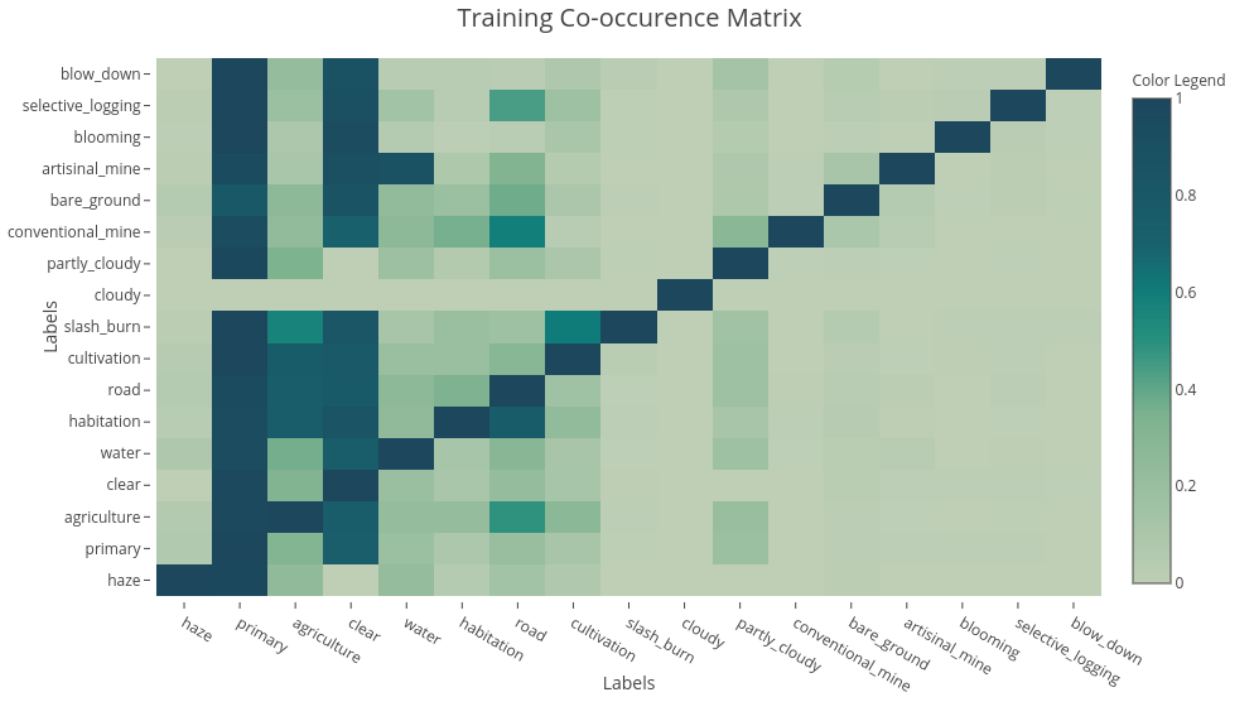
Conclusion

Reflection

The initial challenge for me was setting up my development environment to read and import all images into memory. At first, I used the Kaggle API to download all training and testing images. When loading all 61,000 testing images into memory (image 40,109) always caused an issue when reading via OpenCV. This caused me to work-around this issue by importing all my work into a Kaggle Kernel.

Although we made good use of image augmentation techniques, and experimented with different network architectures, we realized that to greatly improve our true positive score we needed to take advantage of label correlations.

Fig 12. The chart below shows that certain labels coexist together in a specific tag, while others do not. An important insight to note here is the correlation between labels, and finding which labels coexist.



IMPROVEMENT

Kaggle interviewed the first place winner, which can be read at this Kaggle blog post. The winner @bestfitting used a combination of deep learning with ensemble methods. What really struck out to me with @bestfitting's solution was his use of dark channel prior to remove haze from all training/testing images. There are many other image processing techniques such as histogram equalization, elastic transformation, contrast stretching, adaptive equalization, and the list goes on. One way to improve our model is to improve our data preprocessing steps, and to compare which combination yields the greatest accuracy.

Additionally, in order to take advantage of label correlations, @bestfitting, used a ridge-regularized layer to recalibrate each label probability given all the others. And, to predict the final probability, the final model used another ridge regression model which takes in all the deep learning model's predictions of all 17 labels. The important message here is that there exists correlation between labels and certain labels coexist quite frequently, while others do not. See above **Fig 12** for the labels co-occurrence matrix. This means, taking advantage of a correlation structure would greatly improve my models true positive score[18].

Finally, once we have achieved our optimal model we should look in ways we can put this model into production[19]. One way we can do so is by creating a Flask API to accept POST request with an image attached, then use the model we saved to

[18] We designed our network to output the predicted labels given its highest probability.
[19] The solution to the productionizing a machine learning model lies in how one will communicate and pass objects (such as an image) to the classifier.

determine the classification. Another solution would be to use TensorFlow Serving. Google recently released a pre-built rest API that is designed to serve models across any platform. The final step would be to create a client-side tool to classify new images.

APPENDIX

KAGGLE API

Kaggle provides a very simple and easy to use API that can be installed via pip (Python Package Installer).

Take the following steps to configure your API.

- Run 'pip install kaggle' from your terminal
- Once installed, browse to Kaggle.com and go to your account page.
- On your account page, there is a section titled API and button to create a new API token.
- Create a new API token, this will download a Kaggle.json file
- Move this Kaggle.json file to the directory ~/.kaggle/Kaggle.json
- Then, run chmod 600 ~/.kaggle/kaggle.json from your terminal
- You should now be able to run Kaggle commands via terminal (e.g. kaggle competitions list -s health)
- Download the training images & testing image (7z), training labels, and submission file (csv)
- Use 7zip to unpack and retrieve the contents of all compressed directories.

If any errors occur such as 403: Forbidden, refer to the Kaggle community for more information.

SOURCES

1. http://blog.kaggle.com/2017/10/17/planet-understanding-the-amazon-from-space-1st-place-winners-interview/
2. http://www.di.ens.fr/willow/research/cnngeometric/
3. http://search.arxiv.org:8081/paper.jsp?r=1712.00720&qid=1516684733782bas_nCnN_1902725897&qs=convolution+neural+network+and+computer+vision&in=cs
4. https://arxiv.org/pdf/1605.09081.pdf
5. https://www.philadelphiafed.org/-/media/research-and-data/publications/working-papers/2018/wp18-15.pdf
6. http://kaiminghe.com/publications/cvpr09.pdf
7. https://github.com/joyeecheung/dark-channel-prior-dehazing
8. https://keras.io/
9. https://stackoverflow.com/
10. https://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html
11. https://en.wikipedia.org/wiki/Co-occurrence_matrix
12. https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/
13. https://github.com/Kaggle/kaggle-api
14. https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-3.pdf
15. https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-2.pdf
16. https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf
17. https://towardsdatascience.com/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085
18. https://www.cis.rit.edu/class/simg782/lectures/lecture_02/lec782_05_02.pdf
19. https://arxiv.org/abs/1412.6980
20. http://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012028/pdf
    a. Zhou
21. http://timdettmers.com/2015/03/26/convolution-deep-learning/