**Deep Learning for Satellite Image Analysis: Amazon Kaggle Challenge**

**Capstone Project**

Rafael Odoni

July 22, 2016

**Project Overview**

Image recognition is a big challenge in machine learning that in recent years became a hot subject, this because the several applications that recognize image could be used in many industries however for reach good results is necessary sophisticated algorithms and sometimes a big data set with a big number of images to train those algorithms.

For a while, image feature extractors such as SIFT and HOG were the standard. Recent developments in deep learning research has extended the reach of traditional machine learning models by incorporating automatic feature extraction as base layers [1].

SIFT and HOG went a long way in defining good image features [1]. However, the latest gains in computer vision have come from a very different direction: deep neural network models. The breakthrough happened at the ImageNet challenge in 2012, where a group of researchers from the University of Toronto nearly halved the error rate of the previous year’s winner. They branded their method “deep learning” to emphasize that, unlike previous architecture neural network models, the latest generation contains many layers of neural networks and transformations stacked on top of each other.

In fact, most of those recent development in Deep Learning is due the rising of computational power from GPU’ s that became very popular to resolve big problems due their capabilities to provide a massive number of threads that could fit well with problems like Deep Learning.

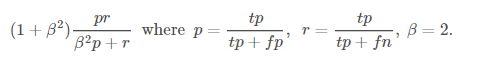
**Problem Statement**

Aware about those great advances in Deep Learning Neural Networks, in this study, we will explore the utilization of Deep Learning to classify Satellite Images from the Amazon Region in South America, this challenge was proposed by Kaggle in a competition named “*Planet: Understanding the Amazon from Space”.* Classify and identify the usage of the Amazon surface is not a trivial problem because there are many combinations of usage and many weather variations that could modify those images from satellite.

For this purpose, we willing to use Convolution Neural Network Convolutional that is rooted in image processing, the structure of one convolutional layer is very convenient to image processing. Furthermore, convolutional layers have found their way into virtually all sub-fields of deep learning, and are very successful for the most part.

**Metrics**

For the measurement of our model we will use the ranking provide by the Kaggle in the competition, for this rank Kaggle used the F score that is one metric commonly used in information retrieval, measures accuracy using the precision p and recall r. 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 + fn)



F-score is a harmonic mean of precision and recall, we use the harmonic mean because it is the appropriate way to average ratios, with this we can evaluate together recall and precision and avoid some problems for example we can have a high precision but low recall, our classifier could be extremely accurate, but it misses a significant number of instances that are difficult to classify, so F-score avoid it making a measurement that take in account both.

**Data Exploration**

For the competition “*Planet: Understanding the Amazon from Space”* were provided several satellite images each one with 256x256 pixels that will be used to train our Neural Network. In fact, the dataset is composed by a collection of images to train and other collection to test the algorithm already trained, furthermore is provide together a csv file that contains all labels for each image in the training collection.



Figure 1 – Example of images provided and their labels.

The dataset provide by Kaggle is formed by 40478 images for the training and 61191 images to test the algorithm, together with it has a csv file with all labels for the 40478 images to train. For each image is possible to has 17 features that could be: *bare ground, conventional mine, blooming, cultivation, artisanal mine, haze, primary, slash burn, habitation, clear, road, selective logging, partly cloudy, agriculture, water and cloudy*. All images on dataset have 256x256 pixels and represent almost 90 hectares over Amazon surface [Figure 2].

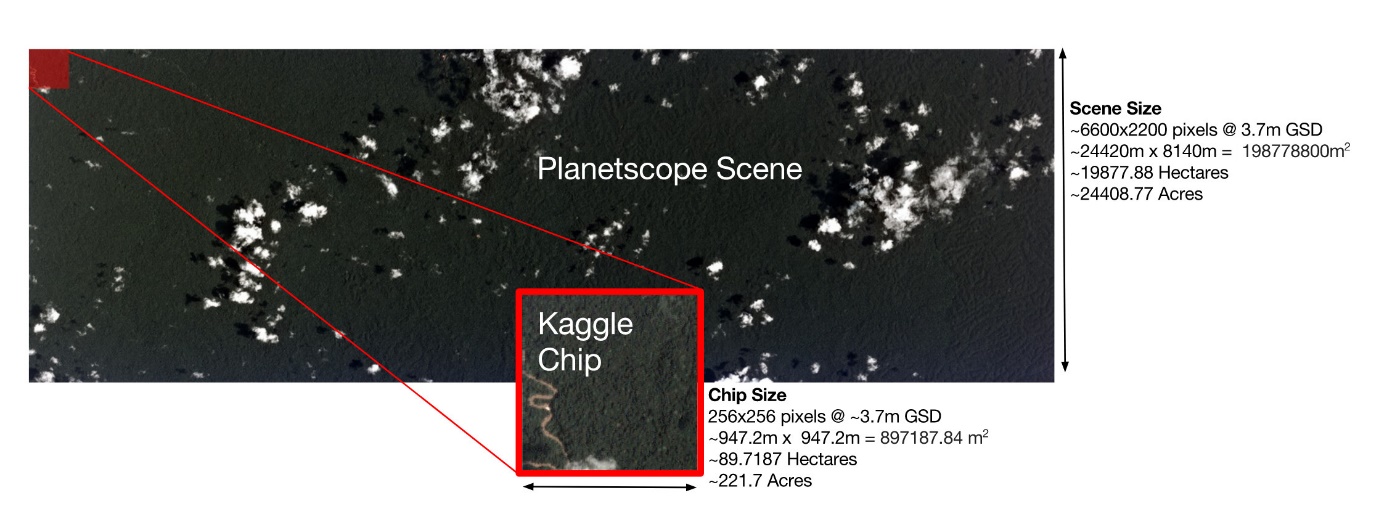


Figure 2 – Representation of one image.

Exploring csv file, most of the data set is labelled as "primary", which is shorthand for primary rainforest, or what is known colloquially as virgin forest. After “primary” we can see the label “clear” as the second most common label in this data set, “clear” is used to represent a “clear sky” above the surface that. “Agriculture” is the third label most common and the one that represents the humankind action in deforestation of rain forest.

Other label is “roads”, that are important for transportation in the Amazon but they also serve as drivers of deforestation. Mainly deforestation often follows new road construction, while smaller logging roads drive selective logging operations. Some rivers look very like smaller logging roads, and consequently there may be some noise in this label. Other important part is “water” that represents rivers, reservoirs, and oxbow lakes.

In data, we had a lot of images labelled as "cloudy", "partly cloudy " and "haze”, those tree types are related with weather conditions. "Cloudy" condition is when the surface could not be recognize showing just clouds, "partly Cloudy" is when some parts of the picture shows something beyond that just clouds and "haze" is defined as any chip where atmospheric clouds are visible but they are not so opaque as to obscure the ground.

The habitation class label was used for chips that appeared to contain human homes or buildings. This includes anything from dense urban centers to rural villages along the banks of rivers. After all there are some labels that are less common in dataset "slash and burn", "bare ground”, "artisanal mine”, " selective logging”, "blooming" and "blow down". Figure 3 shows the distribution of those labels among the dataset.

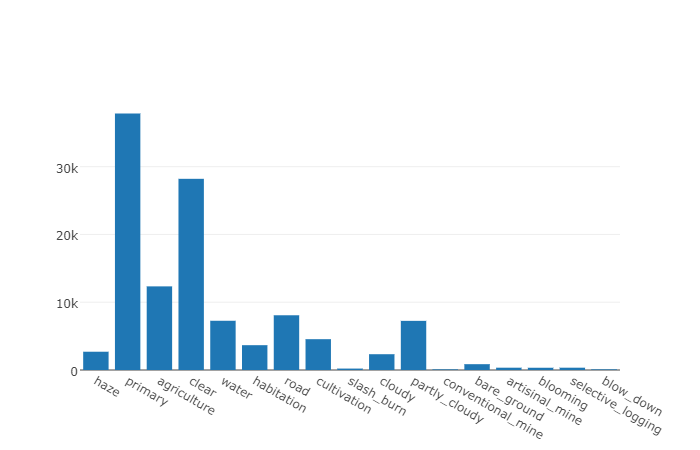


Figure 3 – How many times a label could be find inside of dataset.

**CNN Architecture**

Inspired by how human vision works, layers of a convolutional network have neurons arranged in three dimensions, so layers have a width, height, and depth, as shown in Figure 4. A convolution is defined as a mathematical operation describing a rule for how to merge two sets of information. It is important in both physics and mathematics and defines a bridge between the space/time domain and the frequency domain though the use of Fourier transforms [3]. It takes input, applies a convolution kernel, and gives us a feature map as output.

But in a general way we can separate our architecture in 3 slices: Input Layer, Feature Extraction Layers and Classification Layer.

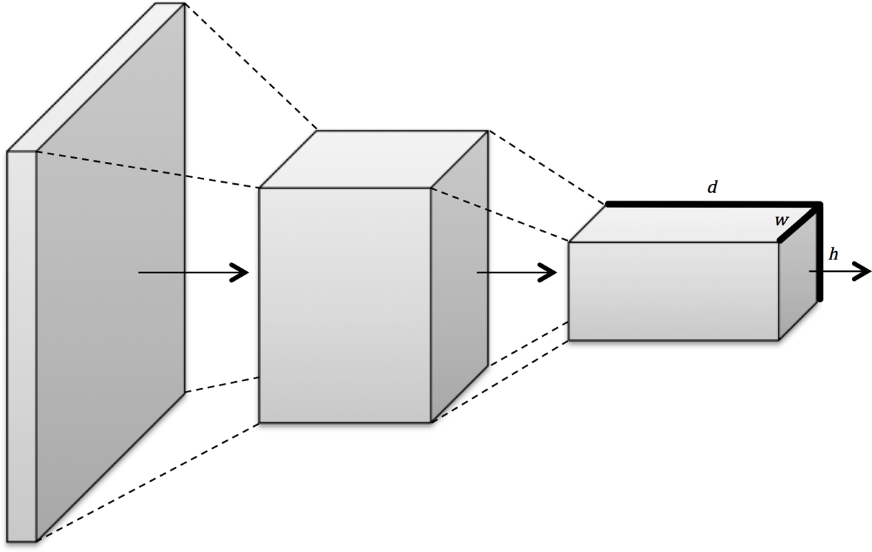


Figure 3 – Convolutional layers arrange neurons in three dimensions, so layers have width, height, and depth [2]

Input Layers where we load and store the raw input data of the image for processing in the network. In our work, we choose to use as input pictures with 64x64 pixels with all RGB channels resulting in 12.288 neurons in the first layer and for get a better result we use Batch Normalization that applies a transformation that keeps the mean activation close to 0.0 while also keeping the activation standard deviation close to 1.0 [3].

After the input layer, we have layers that will learn the features, in our architecture we have four sequences of two Convolutional Layer 2D with *relu* (Rectified Linear Unit) as function activation followed by a Max Pooling Layer with a dropout at the end. Dropout is a regularization method to keep the activation of one neuron with some probability to happen or not avoiding overfitting [4], for example in our architecture we put a value of 0.25 of dropout in other words 25% of probability for one neuron not be activated.

The first pair of Convolutional Layers has a filter of 32 and a kernel size of 3x3 follow by a max polling layer of 2x2 and a dropout of 0.25. For the other 3 sequences of conv Layers and max polling layers the parameters are the same with the varying only the filter size for the convolutional layers, for the second pair of layers the filter is 64, for the third layer 128 and for the fourth layer 256.

Finally, at the end we used a fully connected layer using relu as activation function with 512 neurons followed by other fully connected layer using sigmoid as activation function with 17 neurons, each neuron for each feature class, thus the output will be an array with 17 values. As a optimizer algorithm, our final model used Nesterov Adam [5] that will be better discuss in a Implementation section. All the architecture was written using Keras with TensorFlow, so we use some resources already provided by Keras to create the layers.

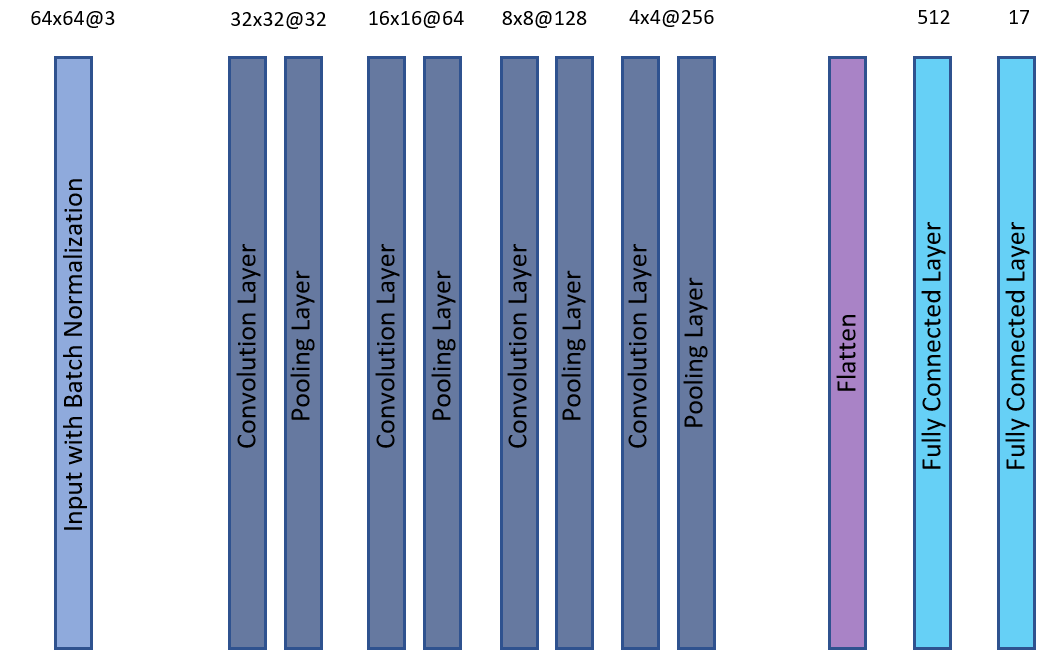


Figure 4 – CNN Architecture, 64x64@3 is a representation of an image with 64x64 pixels and 3 channels.

**Benchmark**

To compare our final architecture, we use as benchmark two basic CNN architectures the first one use as input images with 32x32 pixels with three convolutional layers using Stochastic Gradient Descent as optimizer algorithm. The second model use as input images with 64x64 pixels with four convolutional layers using Stochastic Gradient Descent as optimizer algorithm.

For comparison, the first model got F-score of 0.7952 using 20% of the training data set and the second model reached a F-score of 0.8034 using 20% of the training dataset. Those two models described are almost the same models that we started our development, because that we will use than as a benchmark.

**Data Pre-Processing**

Kaggle provided two types of files, image files and csv files with all labels classifying our images for the training process. The csv files named *train.csv* has 2 columns one with the name of each file to train and other column with an array of labels as we are showing in the table 1.

|  |
| --- |
| *image\_names, tags*  *.*  *.*  *train\_20, agriculture clear primary water*  *train\_21, clear primary road water*  *train\_22, partly\_cloudy primary*  *train\_23, agriculture clear primary road*  *.*  *.* |

Table 1 – Example of csv file format.

In our program, we create a big matrix were each row was a representation of one image and each column was a representation of one tag. For example, the image *train\_20* was set up with number one only in the columns that were a *representation* of *agriculture*, *clear*, *primary* and *water*. In that way, we created a one hot encoding for each image.

Furthermore, we used several technics to improve and fit our dataset with our resources, mainly due memory restrictions. First, we downscale all images to 64x64 pixels after that we transform the RGB image to HSV format with that enhanced the saturation to be clearer all colours and again transform this matrix in a RGB format, those transformations are showed in figure 5.

To reach better results we used a simple technique to augment our dataset, we flip horizontally all images and we double our dataset, during the process it proved to be good technique and was inspired in several examples provided by Kaggle community.

Other pre-processing was mean subtraction, it involves subtracting the mean across every individual *feature* in the data, and has the geometric interpretation of centering the cloud of data around the origin along every dimension. [4] And finally used normalization that refers to normalizing the data dimensions so that they are of approximately the same scale, in our project we divide each dimension by its standard deviation.

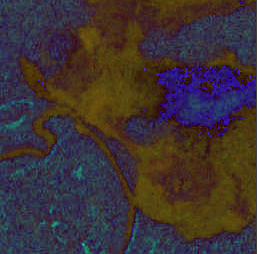


Figure 5 – Original Image, HSV Image and saturated image.

**Implementation**

Our first implementation we thought to separate the problem in two parts, one CNN that could predict only aspects about weather (clear, haze, cloudy and partly cloudy) and other CNN just to predict assets from the Amazon surface. After several tests, our version with only one CNN had reached a better F-Score than our model with two CNN, almost 0.02 of advantage. With those results we decided to use only one CNN to resolve the Kaggle challenge, that changed a little bit our solution proposed.

To check our model performance before send to Kaggle, we used a function F-score provided by *sklearn* library using at least 20% of the training data set to made this metric, sure that for Kaggle the results could be different thus they use the test data set that is provide only the images without their class labels.

In the first version, we decide to downscale all images to 32x32 pixels using only normalization, in our CNN architecture we put only three convolutional layer all layers were set up with ReLu function only the last layer that provide the results we set up sigmoid as activation function. For the training, we put 20 epochs with using Stochastic Gradient Descent with a learning rate of 0.001 and our loss function was configurated with binary cross entropy. We reached a result about 0.7952 in a F Score measurement using our training data set.

With those results in mind we started to investigate many ways to improve our results, first we upscale our data set to 64x64 pixels and with that we create one more layer, only with that we reached a F Score about 0.8103. The next step was improving our input using mean subtraction and trying to manipulate some graphical aspects of each image in the end we observe that rising the saturation value of the images to create more colourful image help the CNN to get better results, however this process to improve our input take a time and we follow some tips explained in an article named *Must Know Tips for Deep Learning* [11], that help us to improve our pre-processing.

After some tests with Stochastic Gradient Descent we did not get better results we decide to try other methods, we changed to Adam that is a Stochastic Method but is based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients [8]. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. Using Adam with the same architecture we reached 0.8801 in the Kaggle F-Score and 0.9052 using our internal F-Score.

However, we were struggling to increase our quantities of epochs in the training sometimes after 20th epoch our model was starting show a variation in the loss and accuracy measurements so to resolve it we started to decrease our learning rate after the 20th epoch, this approach is very good explained in a Stanford course [12], they named it as *annealing learning rate*. So, we decide to increase for 30 epochs to train and for the epochs between 21 and 25 we decreased the learning rate to 0.0001 and between epochs 26 and 30 the decreased the learning rate to 0.00001. With that we teste one more optimizer implementation name Nadam [5], that is an implementation of Adam algorithm but with Nesterov momentum that provide a faster convergence.

Other problem that we found were find a good threshold to interpret the CNN results, we tried to generate the model threshold during the training step, but we failed in find a good balance using those thresholds inside with training data we got good values but when we submitted our results in the Kaggle we got at least a penalty of 10% of our best score. So, after that observing our internal F-Scores and discussing with some competitors we found a value of 0.23 thus every value of one neuron on output layer above 0.23 was considered true.

Finally, in our training process, we did a cross validation to split our data training in 5 steps, in other words the program was performed 5 times the entire process of training choosing 20% of the images on dataset to be used as validation data.

Totally each training cycle was composed by 30 epochs using Nesterov Adam [5] optimizer as a method for a stochastic optimization. The learning rate used were 0.001 for the first 20 epochs followed by 0.0001 for the 5 epochs and finally 0.00001 for the latest 5 epochs. Furthermore, we split our data in batches of 64 samples.

Other refinement in our architecture was initialize the weights of our CNN, we found some discussions about it and we tried several techniques and at the end we configured using a Glorot normal initializer, that provide for us a slight better result.

With all those modifications describe above we reached a F Score about 0.905, after some research we found that augmented our dataset could help a little bit, first we tried to augment only images with classes that were not too common in dataset like *blow down, slash burn, selective logging* and so on. In this first approach, we got bad results and lost at least 2% of our F Score metric, analysing the output from prediction and some statistics from training step we found some overfitting mainly in classes that was not too common. In our second approach, we flip horizontally all images in dataset training and with that we double our dataset and reached 0.916 in a F Score prediction, it proves that for Deep Learning algorithms have a big dataset helps to reach better results.

Finally, we use a K-Fold Cross Validation to split our training dataset, about 20% of our training dataset were used to validation. With K-Fold we slip our dataset in 5 pieces, thus we run 5 times training step but changing our validation and training data, for each loop we saved the weights and the state of our network in files to be recover in the prediction step. So, the results of each fold are sum and we got a mean of then to create the file report to be submitted to Kaggle.

**Results**

Using the CNN architecture as was described above, during the training process we achieve values of F-Score between 0.923 and 0.92 when we predict using only the training dataset. But at the end of competition, *Planet: Understanding the Amazon from Space*, using the entire test dataset provided we achieve 0.91961 and 329th position in private leader board on the Kaggle after we submitted our last results in the public leader board we reached 0.921 and 358th [9]. The difference between those ranks are that the public use 66% of the test data and the private use 34% of the test data.

It was solid performance, to reach this value the program need at least 1 hour and 30 minutes between load all images, training and predict the labels for the test images. In comparison with our benchmark we improved our model quiet well as we can see in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Layers | Learning Algorithm | F-Score |
| 32x32 Pixels | 3 layers CNN | Stochastic Gradient Descendent | 0.7952 |
| 64x64 Pixels | 4 layers CNN | Stochastic Gradient Descendent | 0.8103 |
| 64x64 Pixels+ Augmentation | 4 layers CNN | NADAM+Glorot | 0.9222 |

Table 2 – Benchmark comparison.

We can see that our refinement gave for this model a better performance than use only a simple model, was incredible how much was improved creating more images samples and changing the learning algorithm.

**Conclusion and points to improve**

It was clear that Deep Learning is a powerful tool to classify images, even this problem that could have multiple classes for each image we got a good result at the end. Furthermore, during the implementation were clear the enhances when we provide more data for the CNN, during three times we observed a big jump in our predictions when we upscale the images to 64x64 pixels and when we created more images using an augment dataset technique, flipping horizontally all images and add then in our training dataset doing it we doubled our dataset and finally when we the learning algorithm. Other important thing was cross validation, that really helped to reach a better result in the end of competition.

However other approach that was very helpful was change the learning rate during the training, in figures 5 and 6 we can see clearly how our model start to have difficulties to increase the accuracy and decrease the loss after 20th epoch as we describe in Implementation section in this document. Other thing that we can see in figure 5 and 6 is the variation of results in test data even though there is a trend to converge to a better result sometimes between one result in worse than a previous, it can be observed between epochs 5 and 10 mainly.

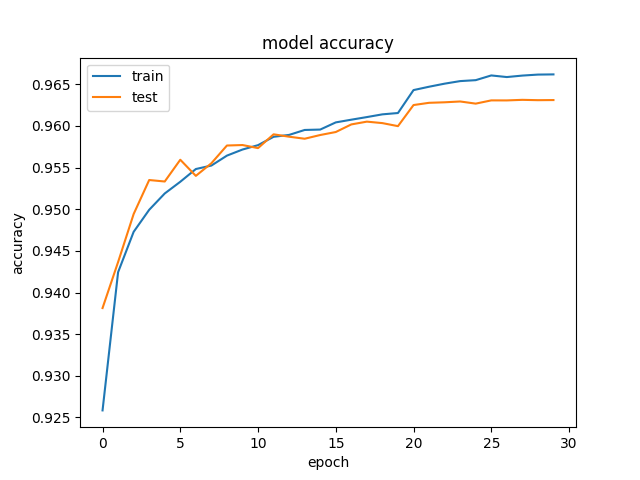


Figure 5 – Accuracy during the training.

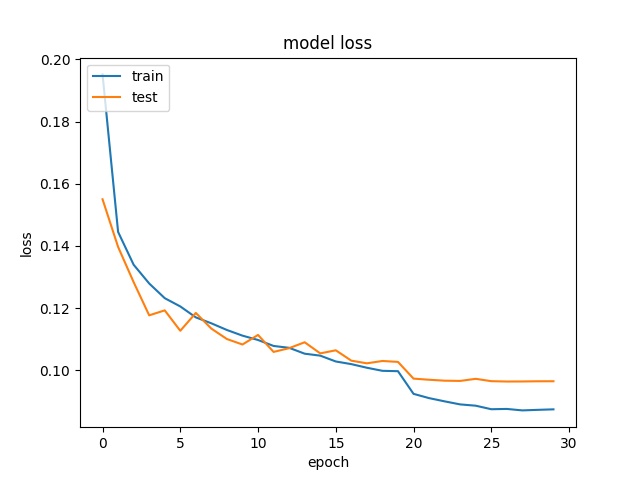


Figure 6 – Loss during the training.

In our solution, we started with a simple model that we used to create a benchmark, after this first result we started to improve this model. First, we modify our input using normalization, modifying the colour intensity and creating more samples to get a better result after that we put more effort in testing several learning algorithms and learning rates to check the behaviour of our architecture, here we found some difficult to evaluate what algorithm was better mainly because our thresholds did not fit well and the Kaggle F-scores were not so reliable because that. But observing the F-scores that we got using the training data set we adjust our threshold to 0.23 and we started to get a better result. It seems clear that not only have a good model is important but how you treat their results is very important to get a good model.

Besides the good values reached, it seems clear that these results could be improved, first thing could be upscale all images to 128x128 pixels and create one more feature layer. Other solution could be enhancing quality of our prediction in rare labels like we saw in figure 3, augmenting these images or creating other CNN our even though using some statistical approach for this part maybe Linear Regression or Clustering.

During the process of evaluation was clear that even datasets with many noise and different quantities of samples of each label like we saw in *Planet: Understanding the Amazon from Space* competition the CNN showed a good performance.

[1] Mastering Feature Engineering, Alice Zeng. O 'Really Books 2017

[2] Fundamentals of Deep Learning, Nikhil Buduma. O 'Really Books 2017

[3] Deep Learning, Adam Gibson and Josh Patterson, O ‘Really Books 2016

[4] http://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

[5] <http://cs229.stanford.edu/proj2015/054_report.pdf>, Nadam Report.

[6] <http://scikit-learn.org>

[7] http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf, Glorot & Bengio, AISTATS 2010

[8] <https://arxiv.org/abs/1412.6980v8>, Adam A Method for Stochastic Optimization

[9] <https://www.kaggle.com/rodoni>

[10] <https://arxiv.org/pdf/1406.5726.pdf>, CNN: Single Layer to Multi-Label

[11] <http://www.kdnuggets.com/2016/03/must-know-tips-deep-learning-part-1.html>, Must Know Tips for Deep Learning

[12] http://cs231n.github.io/neural-networks-3/#anneal