Using Deep Learning to solve a Image Classification problem
(Kaggle Competition - Planet: Understanding the Amazon from space)

Nilson Yuki Ivano

August xx, 2017

Abstract

This work tackle a classic image classification problem using deep learning as a main tool to solve this problem. We'll experiment with some models based on CNN VGG architecture and discuss results

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 on solving this Kaggle challenge: *Planet: Understanding the Amazon from Space[1]*

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.

Figure 1 - Satellite image samples of Amazon Kaggle Challenge

2 Data

The data for this competition 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 32,383 training samples and 8,096 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.

2.1 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’, ‘artisanal mine’, ‘selective logging’ and ‘blooming’ are very infrequent and together only account for about 1% of all the labels found in the dataset.

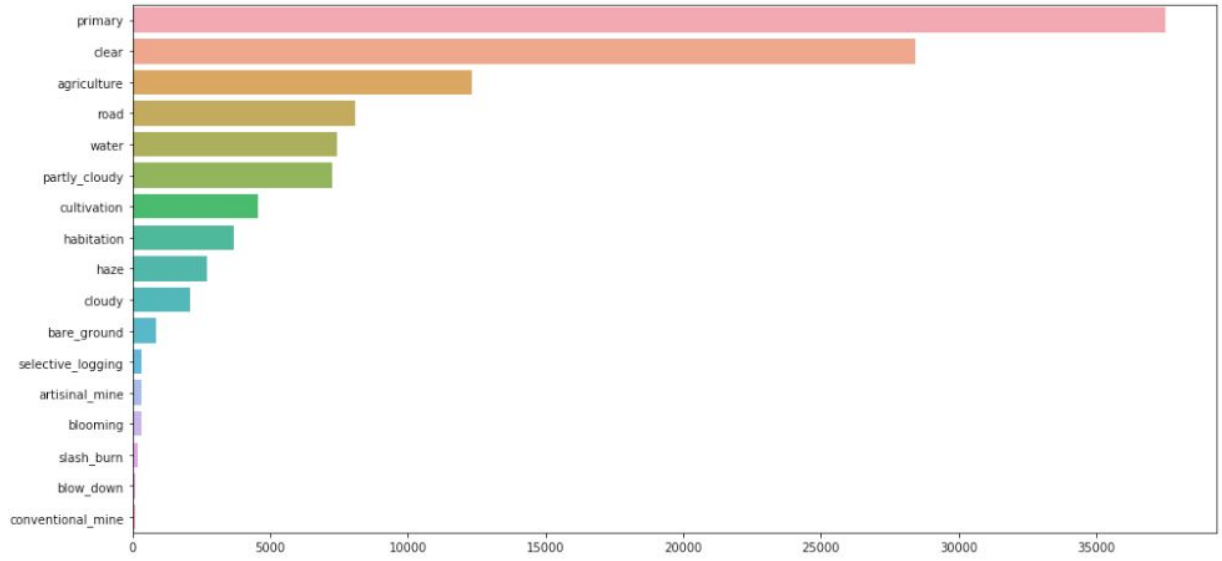


Figure 2 - Dataset histogram image classes distribution

2 Convolutional Neural Networks

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been

successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars and today are the best solution on dealing with complex image classification problems.

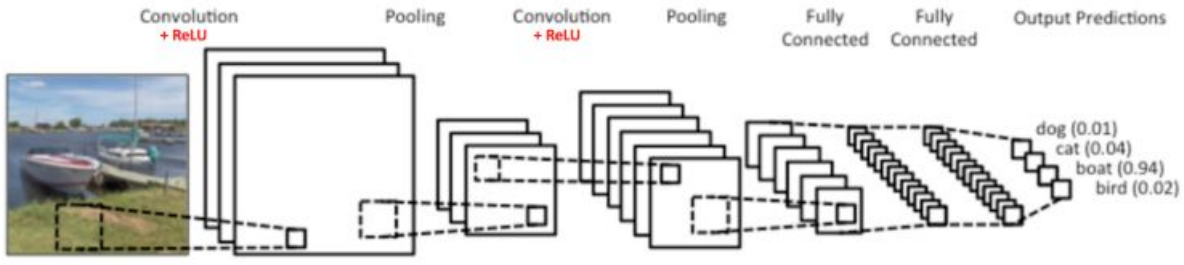


Figure - A simple convnet example

A typical CNN is basically composed by a mix of the following layers:

Convolution Layer

The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Pooling Layer

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

In our case we'll use only Max Pooling layers. We define a spatial neighborhood (for example, a 3×3 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window.

Fully Connected Neural Network Layer

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

Activation Layers (Relu)

After each conv or FCN(Fully Connected Network) layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward.The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).

One of the most used activation layers is the ReLU (Rectified Linear Unit) and is a non-linear operation. Its output is given by:

Output = Max(zero, Input)

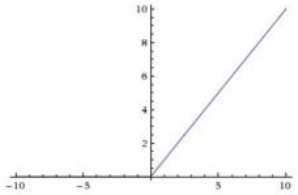


Figure 8: the ReLU operation

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. Other non-linear functions such as tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

3 Methods

3.1 Pre-processing

Although the image dataset has a skewed distribution, there was enough data of each class to train properly a model, each class had at least more than 100 samples of each. So we decided to not use any image augmentation.
The only image pre-processing we made was image resizing due to performance issues and hardware memory limitation. We resized the original image samples (256x256 RGB) to 64x64 RGB image size.

3.2 Loss Function

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

3.3 Model Implementation

The coding and CNN implementation on this project was based on Keras library. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano, it was developed with a focus on enabling fast experimentation. Using Keras library was by far the best decision on choosing how to tackle the problem of implementing CNNs and model evaluation. I consider libraries such as Tensor Flow and Theano to have a steeper learning curve.

The main idea was to experiment with CNN models based on the VGG architecture[11]. This network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above). 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.

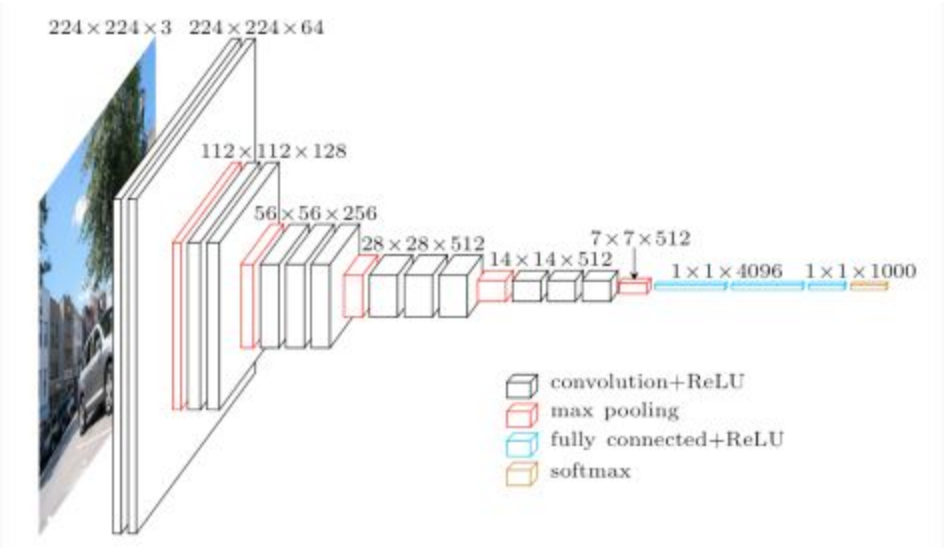


Figure 3 - VGG Architecture

First we started experimenting with a small model, the VGG1 that is described on figure 4 with samples with image size of 64x64 RGB. That model could be evaluated using my computer (a late 2011 Macbook Pro), but it could take up to 2 hours to complete the task. The idea here was to build a solid code base, so later we could optimize our time when experimenting with deeper VGG models using the Amazon EC2 instance specific for Deep Learning purposes.

3.4 Model Refinements

One of objectives of this project was to analyse how VGG architecture would behave on this image classification problem using a wide range of similar structures. Basically we experimented using models with different quantities of conv-conv-pooling layers and different quantities of Fully Connected layers. We trained some VGG16 models with and without dropout, to see if there is a real performance improvement.

VGG1	VGG2	VGG3	VGG16	VGG16 Dropout
conv3-32 conv3-32 Maxpool Dropout 0.25	conv3-64 conv3-64 Maxpool Dropout 0.25	conv3-64 conv3-64 Maxpool Dropout 0.25	conv3-64 conv3-64 Maxpool	conv3-64 conv3-64 Maxpool Dropout 0.25
conv3-64 conv3-64 Maxpool Dropout 0.25	conv3-128 conv3-128 Maxpool Dropout 0.25	conv3-128 conv3-128 Maxpool Dropout 0.25	conv3-128 conv3-128 Maxpool	conv3-128 conv3-128 Maxpool Dropout 0.25
FC128 Dropout FC17	conv3-256 conv3-256 conv3-256 Maxpool Dropout 0.25	conv3-256 conv3-256 conv3-256 Maxpool Dropout 0.25	conv3-256 conv3-256 conv3-256 Maxpool	conv3-256 conv3-256 conv3-256 Maxpool Dropout 0.25
	FC512 Dropout 0.5 FC17	FC512 Dropout 0.5 FC512 Dropout 0.5 FC17	conv3-512 conv3-512 conv3-512 Maxpool	conv3-512 conv3-512 conv3-512 Maxpool Dropout 0.25
			conv3-512 conv3-512	conv3-512 conv3-512

			conv3-512 Maxpool	conv3-512 Maxpool Dropout 0.25
			FC4096 FC4096 FC17	FC4096 Dropout 0.5 FC4096 Dropout 0.5 FC17

Figure 4 - Description of VGG models evaluated in this project

3.4.1 Optimizers

We experimented with 3 types of optimizers that Keras has under his library, the RMSProp (Root Mean Square Propagation), Adam(Adaptive Moment Estimation) and SGD (Stochastic Gradient Descend). In the end, we choose Adam optimizer because it showed slightly better results of accuracy than RMSProp and SGD.

3.4.2 Number of epochs

Since we got a reasonable hardware from Amazon instance, we could experiment with more epochs per model.

- VGG1 - 50 epochs
- VGG2 - 50 epochs
- VGG3 - 25 epochs
- VGG16 - 25 epochs
- VGG16 dropout - 25 epochs

The most challenging part of this project was dealing with the Amazon instance setup and connection. Since I had almost no experience on this kind of task, I've found some serious problems along this project. Below are some problems that really:

- It took me a while how to setup the ec2 instance and understand how to run jupyter notebooks remotely;
- Blocked SSH (port 443) connection from my internet provider. Since I couldn't SSH on Amazon instance, I had to use my 3G to connect to it;
- Network Connectivity - sometimes 3G was not reliable and my network connection with the virtual instance failed. I constantly lost some model evaluation results and had to re-run it

3.5 System specifications

This project was processed using an Amazon EC2 instance specific for Deep Learning purposes (https://aws.amazon.com/marketplace/pp/B01M0AXXQB#product-description). We used a p2.xlarge instance that is composed by one NVIDIA GK210 GPU (2496 parallel processing cores) with 12 GiB of memory.

4 Results

4.1 F beta Score

The Kaggle competition is evaluated based on their mean (F_{2}) score. The F score, 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). The (F_{2}) score is given by:

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

The (F_{2}) score weights recall higher than precision. The mean (F_{2}) score is formed by averaging the individual (F_{2}) scores for each row in the test set.

4.2 Benchmark

Our benchmark for this project was the VGG1, our simplest model . The evaluation of this model could be done using my desktop computer (that don't have a strong gpu) and was my first attempt before using a amazon instance with gpu to process deeper networks.

With the configuration below we got a Fbeta Score of **0.642933347871**

- Model - VGG1
- Epochs - 5
- Image sample size - 32x32 RGB

This Fbeta score was our baseline of our project

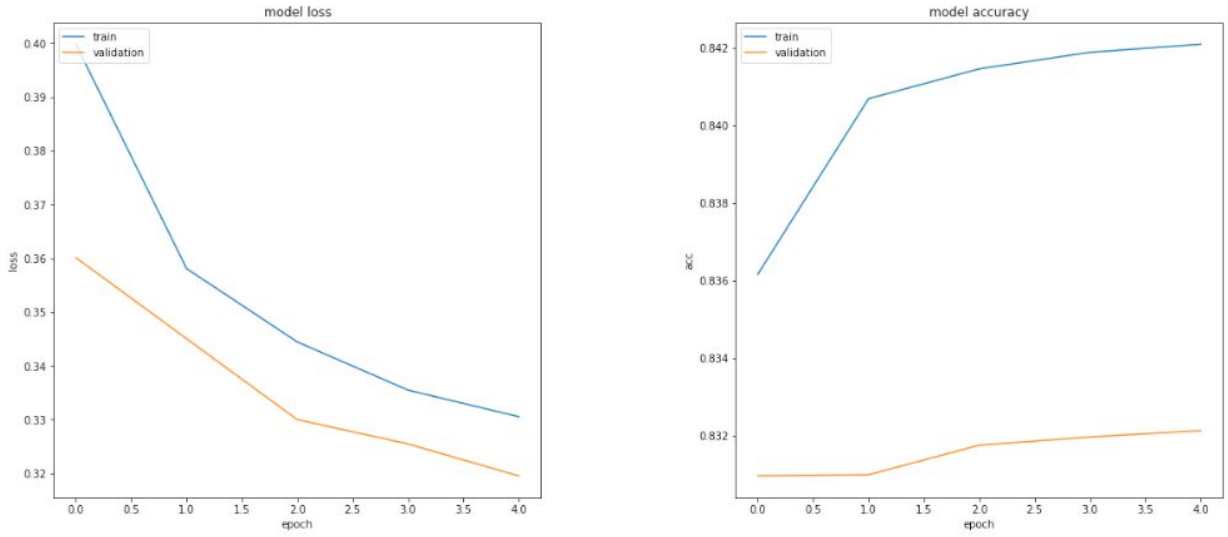


Figure - VGG1 benchmark model performance

4.3 Model results and performance

Our best result was a F Beta score of 0.8179 using the VGG16 model without dropout, this result was expected since the VGG16 was the deepest and most complex model evaluated. Since our initial benchmark Fbeta score was 0.64293334787, we got a 12, 75% of improvement on this score if compared to our VGG16 model. I'm quite happy by this result.

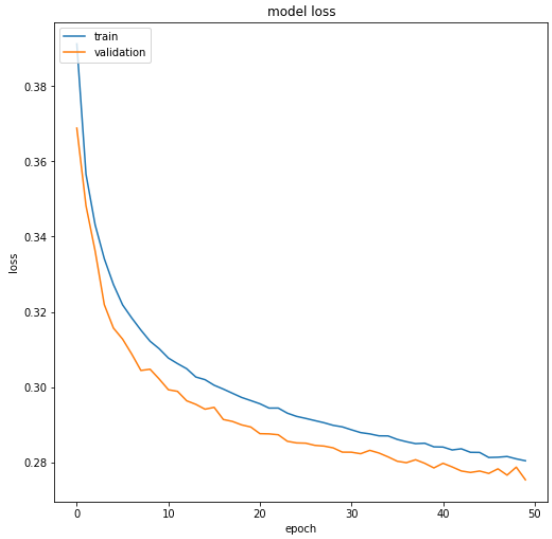
Surprisingly the difference in performance (model loss, accuracy and F beta score) between the simplest model (VGG1) and the most complex (VGG16) was not big at all. In the end, all models finished the evaluation with an accuracy performance inside the 84.5% ~ 85.5% range (train and validation samples) and a model loss inside the 0.22 - 0.28 range (train and validation).

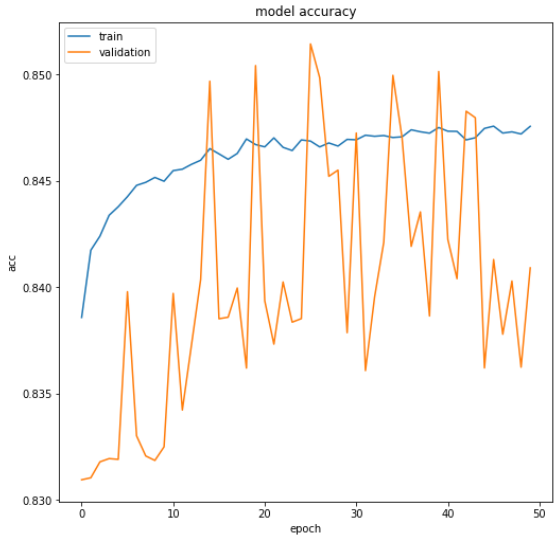
Observing the results of models VGG2 and VGG3, we can see that the adicional Fully Connected Layer in the end played a negative role on performance to model VGG3.

The usage of dropout on VGG16 helped with model overfitting o later epochs of evaluation, but in the end the VGG16 without dropout performed slightly better on F Beta score that the one using dropout.

VGG1

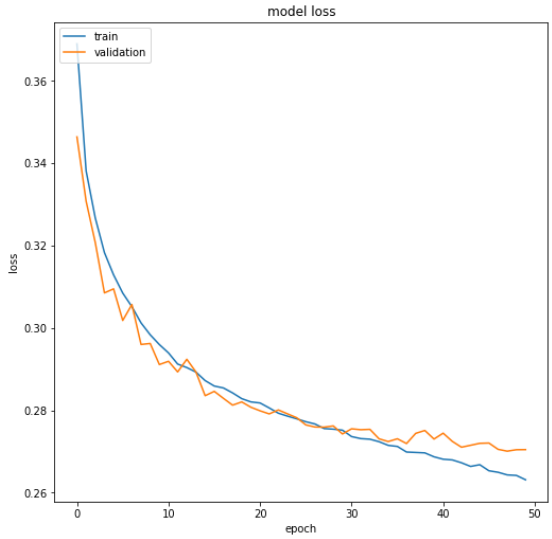
FBeta Score 0.769514291275

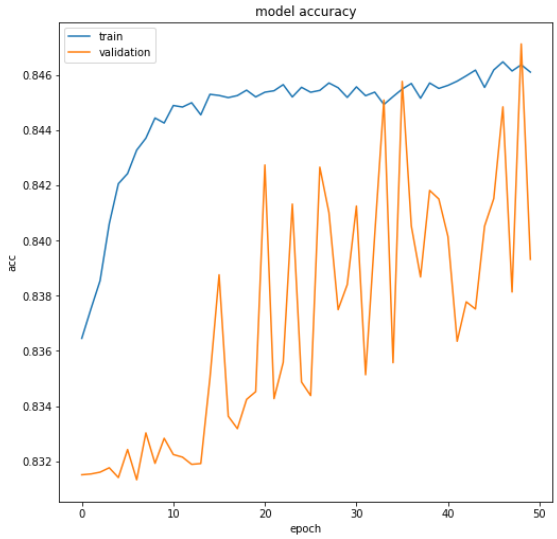




VGG2

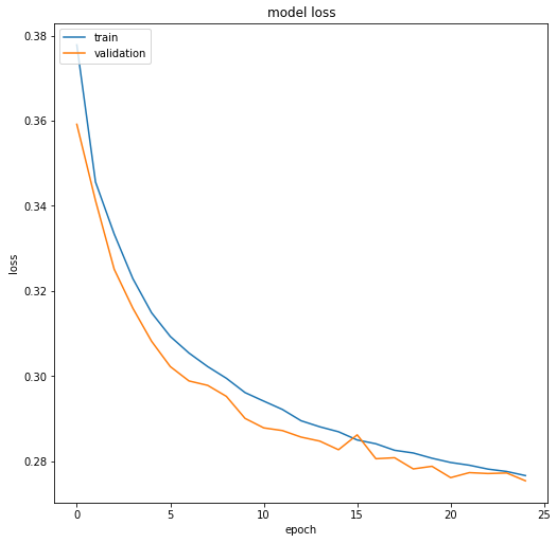
FBeta Score 0.785445340505

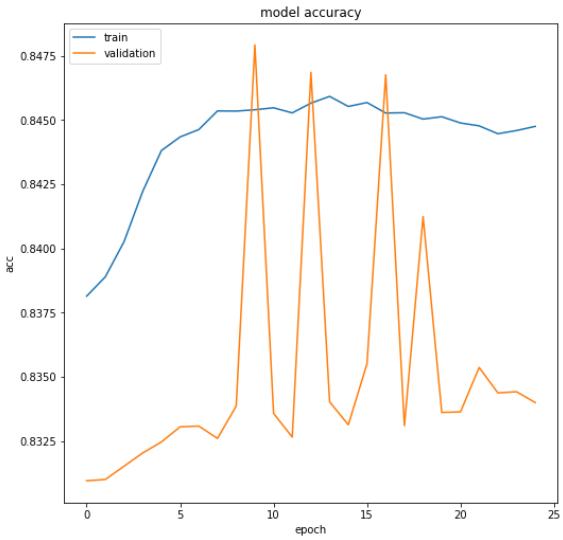




VGG3

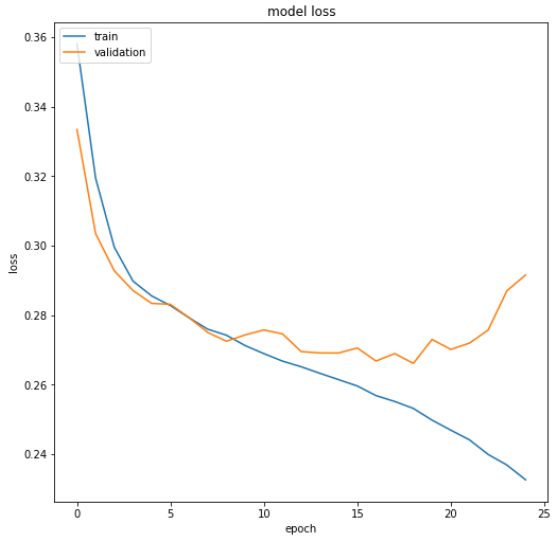
FBeta Score 0.778196431806

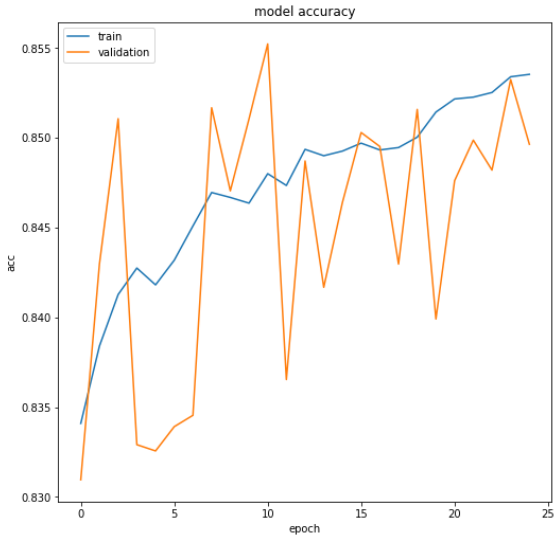




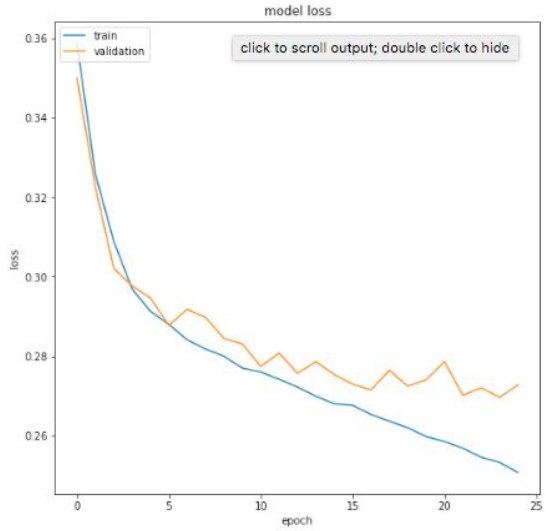
VGG16

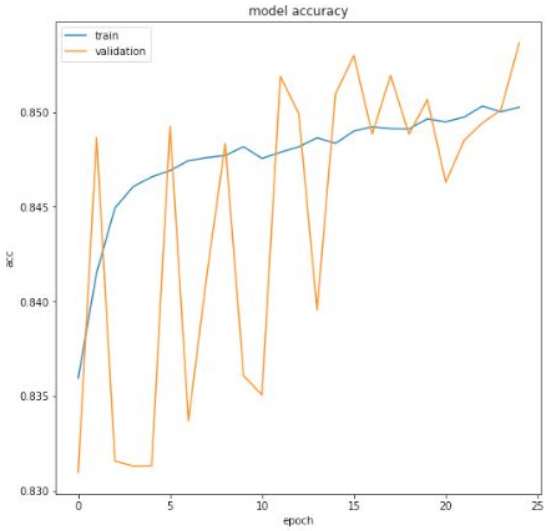
FBeta Score 0.817906235932





VGG16 dropout
FBeta Score 0.812349497364





5 Conclusion

Our final submission got a score of 0.82003, a little bit higher than our calculated F beta score on VGG16 model. That difference between both values is because the Kaggle public score is calculated with approximately 66% of the test data. Since it was a late submission (after the competition deadline) we couldn't get the exact place on leaderboard.

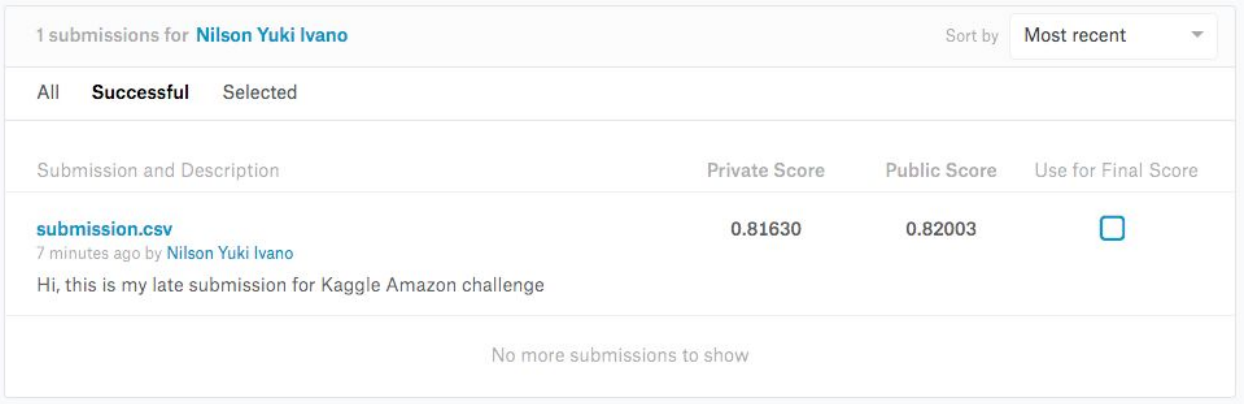


Figure 5 - Kaggle sumission results

The competition leader on public score has a score of 0.93449 so I think for a first time challenge, it was a good result.

At the beginning of this project, the idea was to train models from scratch, but that turned out to be a bad idea at all. After talking with some Kaggle users and reading some kernels, I've figured out that we could easily reach scores higher than 0.92 just using models like ResNet, Inception or VGG with pre-loaded imagenet weights.

So later instead trying to reach high scores on Kaggle leaderboard, I refocused the objective of this work on experimenting with VGG architecture and how it would behave dealing with this image classification problem.

Small models based on VGG architecture can have really good results, even with few convolutional layers. For example, with the VGG1 model reached a fair good score with 20x less parameters than VGG16 model.

VGG1	1,452,337 parameters	Fbeta score 0.769514291275
VGG16	39,958,353 parameters	Fbeta score 0.817906235932

5.1 Improvements

There is considerable room for improvement on this image classification challenge. After reading some information on Kaggle forum, those topics below can reasonably improve the score on this project:

- **Image augmentation** - since we're running many epochs on deep learning model training, we could perform some image pre-processing like random rotation, shifts, shear and flips, dimension reordering. This can improve our model accuracy
- **Use bigger image size** - all models trained with 64x64 fixed image size. Using bigger images like 128x128 or even original image size 256x256 can lead to improvements on accuracy of the model. We didn't trained with images with bigger aspect ratio because of time and hardware limitations
- **Try other network architectures such as Squeeze-net, Inception, Xception, AlexNet and Resnet and use ensemble of models** - a good way to get started with this problem was

to train models with weights already trained on imagenet. This could be a base to find the best performant architecture. Many users got really good results using ensemble of models with different architectures mixing for example VGG, Inception and Resnet architectures

5.2 Reflections

Since I'm hooked to deep learning since the beginning of this Nanodegree, I really wanted to try to implement an end to end solution applying this technology, so I've found and studied a lot about CNN architectures. Later I've chosen to find a problem related to image classification, where I would apply some of CNN knowledge.

My first attempt was to use deep learning to solve the "Where is waldo" puzzle. The idea was to train a CNN to identify if in that image crop there is a waldo or not. So given one "Where is waldo" image, I would use a sliding window to correctly find waldo inside the big puzzle image.

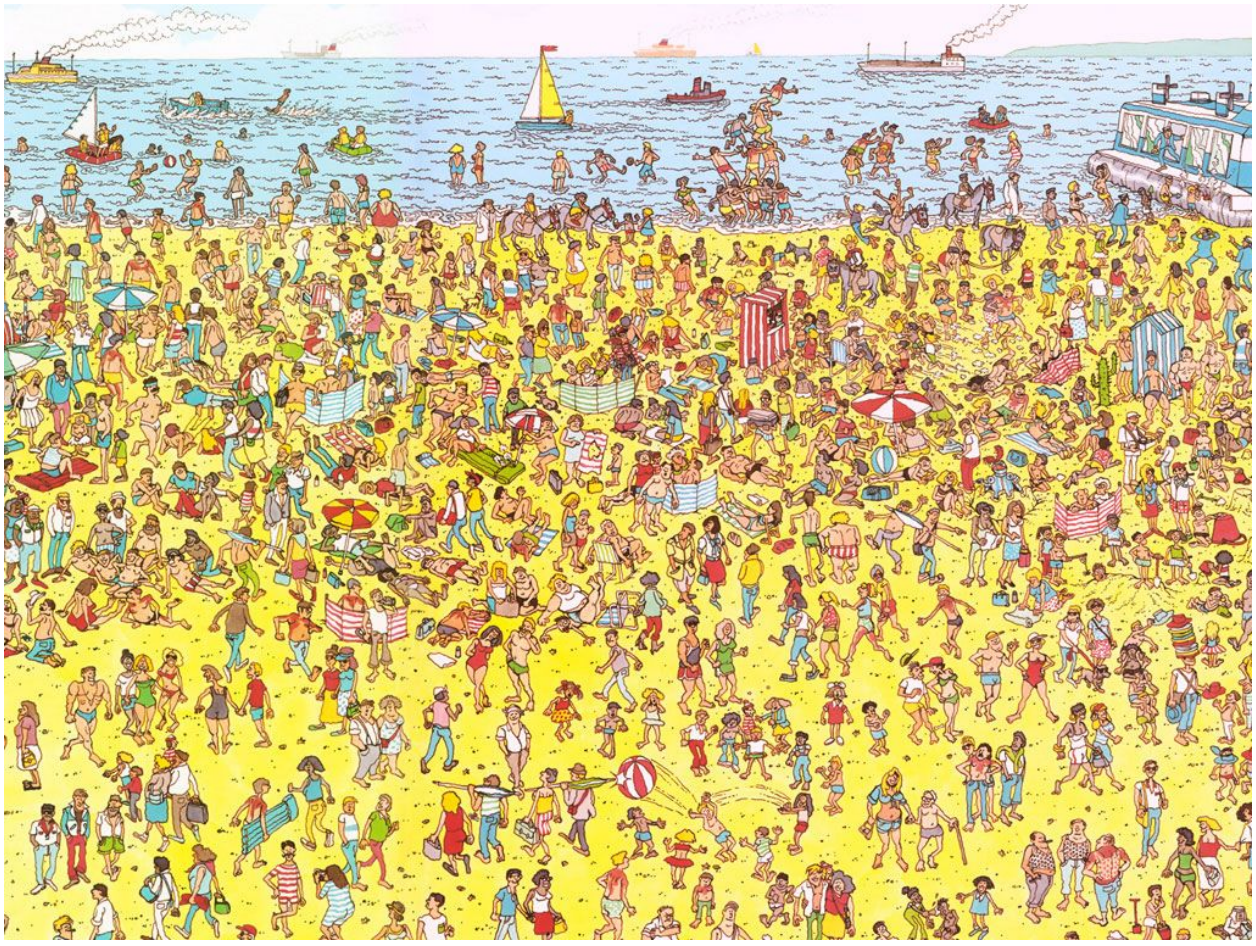


Figure 6 - "Where is waldo" sample image

The main problem on this project was that I had to build the dataset (the datasets that I found on internet where pretty incomplete). So I spend almost 1 month making the dataset and later since it was a problem with unbalanced data (the majority of crops don't have a waldo) I couldn't get. If you want to see my attempt, here is a link to the git https://github.com/nilsonivano/waldo-deep-learning

Failing with my first project, made my way to look for a challenge on Kaggle, where I would find problems with pre existent data sources. Tackling a problem with an already existent dataset made my life much easier and leaded to this capstone completion. Here are some key learn topics when I've finished this project

- Building a proper dataset to use on deep learning model, can take much more time than finding and evaluate the final model that will solve the problem. Most of times data is your main issue;
- If you think you have enough data to approach a problem using deep learning, before attach yourself into a specific architecture (VGG, Resnet, Inception, etc), try to quick experiment with a broader quantity of architectures. This process can help you find the initial best performant architecture;
- Not always, the least and most performance architecture on Imagenet is the best CNN architecture for your image classification solution. Sometimes a VGG or SqueezeNet can perform better than ResNet or Inception

References

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

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

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

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

[5] 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.

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

[7] K.Inc.Planet:Understandingtheamazonfromspace,2017. [Online; accessed 12-June-2017].

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

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

[10] How to Use Metrics for Deep Learning with Keras in Python https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/

[11] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large Scale Image Recognition [https://arxiv.org/pdf/1409.1556.pdf]

[12] D. Kingma, J. Lei Ba. Adam: A Method for Stochastic Optimization [https://arxiv.org/pdf/1412.6980.pdf]