# Machine Learning Engineer Nanodegree

## Capstone Project

Deepthi

July 23rd, 2019

## I. Definition

### Project Overview

To assess the impact of climate change on Earth's flora and fauna, it is vital to quantify how human activities such as logging, mining, and agriculture are impacting our protected natural areas.

Researchers in Mexico have created the VIGIA project, which aims to build a system for autonomous surveillance of protected areas.

Source: https://jivg.org/research-projects/vigia/

A first step in such an effort is the ability to recognize the vegetation inside the protected areas. In this project, we are tasked with the creation of an algorithm that can identify a specific type of cactus called columnar cactus (Neobuxbaumia tetetzo) in aerial imagery.

source:https://www.kaggle.com/c/aerial-cactus-identification/overview

## Problem Statement

As a part of the Aerial Cactus Identification Kaggle competition we have to find the presence of a columnar cactus in aerial imagery. As it has only two classes i.e has_Cactus and no_cactus it can be identified as a binary classification problem.

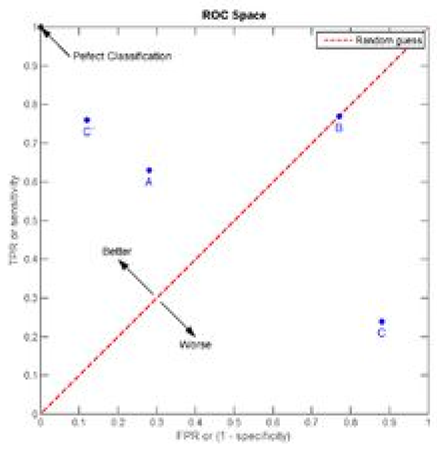
1. Download and preprocess the training images.
2. Check for data imbalance and unbalanced classes. Apply one of the below techniques:
    a. Undersampling
    b. Oversampling
    c. SMOTE - Synthetic minority Over- sampling
3. Use transfer learning from a pre-trained model like VGG16, RESNET50, InceptionV3 or Xception trained on Imagenet data which contains plant data similar to our dataset.
4. Compute Validation accuracy and loss.
5. Predict the classes and compare the results based on the AUC score of the benchmark model.

## Metrics

Submissions are evaluated on area under the ROC curve or AUC between the predicted probability and the observed target.

### What is ROC Curve?

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.



Source: Wikipedia

**Sample submission File**

For each ID in the test set, we must predict a probability for the has_cactus variable. The file should contain a header and have the following format:

id,has_cactus

000940378805c44108d287872b2f04ce.jpg,0.5
0017242f54ececa4512b4d7937d1e21e.jpg,0.5
001ee6d8564003107853118ab87df407.jpg,0.5 etc.

## II. Analysis

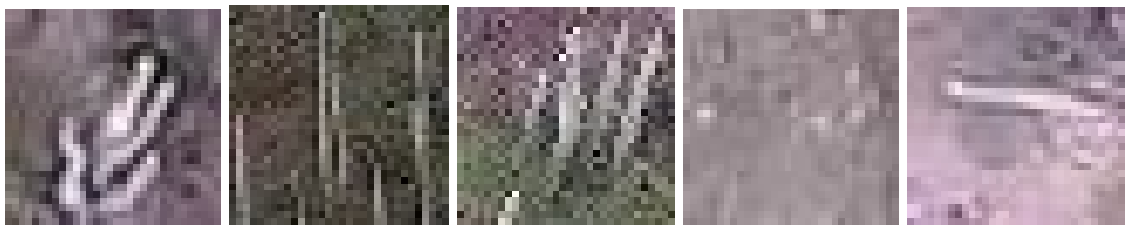
## Data Exploration

The dataset contains a large number of 32 x 32 thumbnail images containing aerial photos of a columnar cactus (Neobuxbaumia tetetzo). Kaggle has resized the images from the original dataset to make them uniform in size. The file name of an image corresponds to its id.

**Files:**

train/ - the training set images

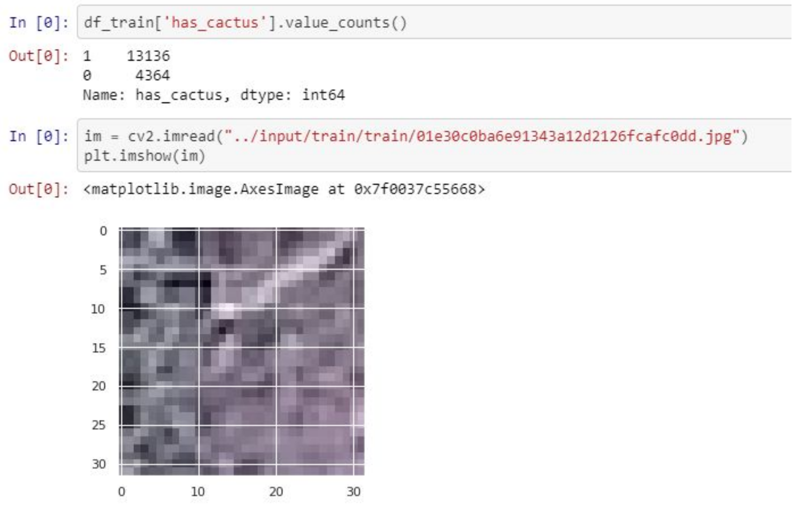
**Image samples from train set:**



test/ - the test set images (labels to be predicted)

Data consists of 17500 training files and 4000 test files.

train.csv - the training set labels, indicates whether the image has a cactus(has_cactus = 1)



```
In [0]: df_train['has_cactus'].value_counts()
Out[0]: 1    13136
        0     4364
        Name: has_cactus, dtype: int64

In [0]: im = cv2.imread("../input/train/train/01e30c0ba6e91343a12d2126fcafc0dd.jpg")
        plt.imshow(im)
Out[0]: <matplotlib.image.AxesImage at 0x7f0037c55668>
```

sample_submission.csv - a sample submission file in the correct format.

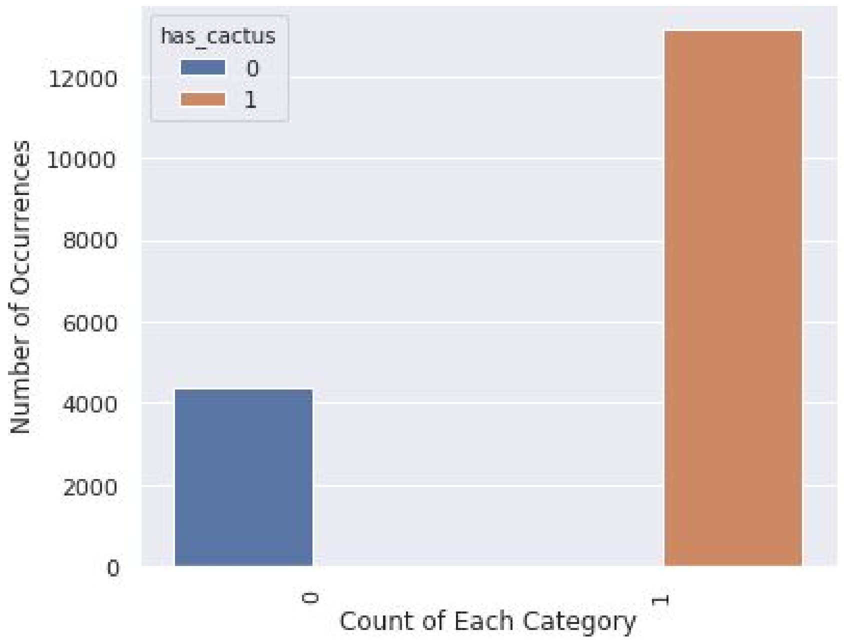
Kaggle Competition Link: https://www.kaggle.com/c/aerial-cactus-identification

Kaggle Datasets Link: https://www.kaggle.com/c/aerial-cactus-identification/data

## Exploratory Visualization

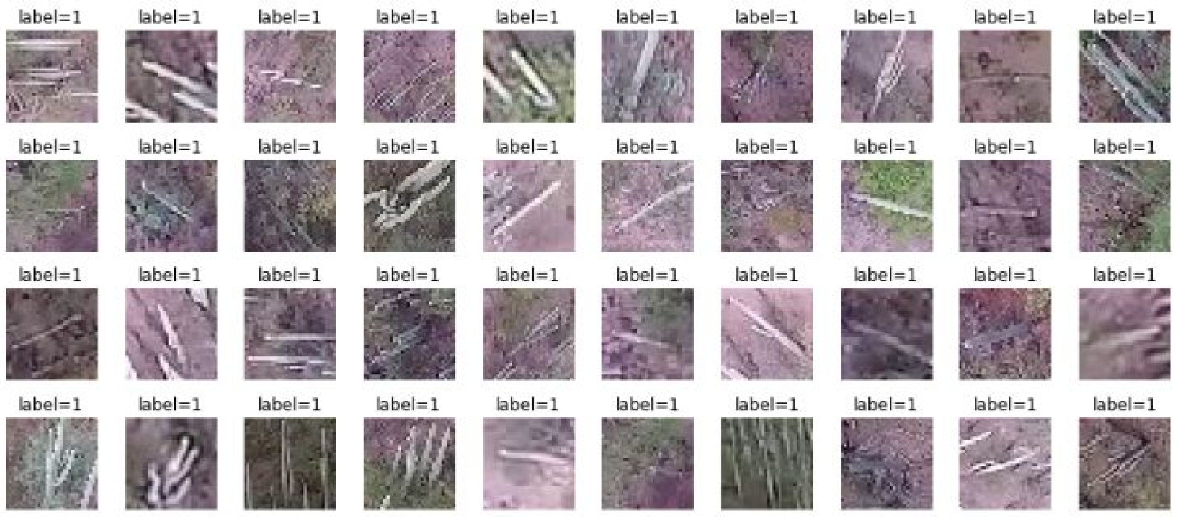
From the Data exploration section, we can clearly see that there is an imbalance in the data. We have a biased dataset. We have has_cactus for more data equal to 1. We used seaborn library to plot a count plot on has_cactus column of train.csv to obtain the below bar chart.

Fig: Count of has_cactus in train.csv

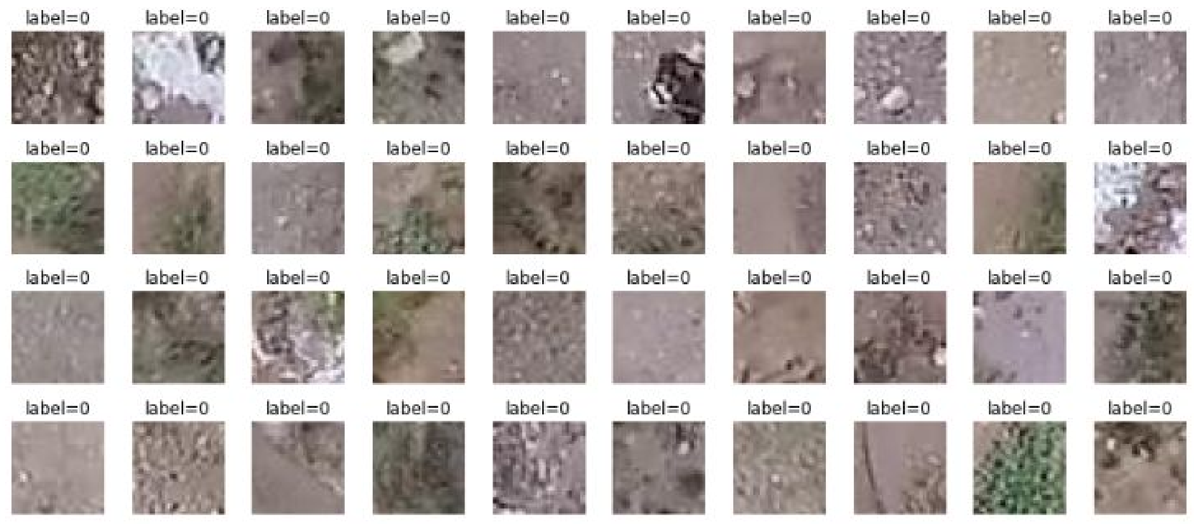


Source: [Project Github Link]

Here are some sample images with label=1:



Some sample images with label=0:



## Algorithms and Techniques

From the description and problem statement it can be inferred that computer vision can be used to arrive at a solution. CNN class of deep learning algorithm can be employed for this problem. Initially data exploration will be carried out to understand possible labels, range of values for the image data and order of labels.

Data should be processed into appropriately pre-processed floating-point tensors before being fed to our network. So, the steps for getting it into our network are roughly

1. Read the picture files
2. Decode JPEG content to RGB pixels
3. Convert this into floating tensors
4. Rescale pixel values (between 0 to 255) to [0,1] interval.

We will make use of ImageDataGenerator method available in keras to do all the preprocessing. This will help preprocess the data and can end up with better predictions.

Now, after preprocessing is done with our data, we will split our dataset to training and validation for training our model and validating the result respectively.Transfer Learning with a pre trained CNN architecture VGG16 will be implemented in Tensorflow/Keras.

Finally, necessary predictions on the test data will be carried out and AUC will be evaluated between the predicted probability and the observed target.

## Benchmark

The model with the Public Leaderboard AUC score of 0.9703 will be used as a benchmark model. Attempt will be made so that score (Area under the ROC curve) AUC obtained will be among the top 50% of the Public Leaderboard submissions.

The architecture of the benchmark model is explained below: Keras with Data Augmentation-

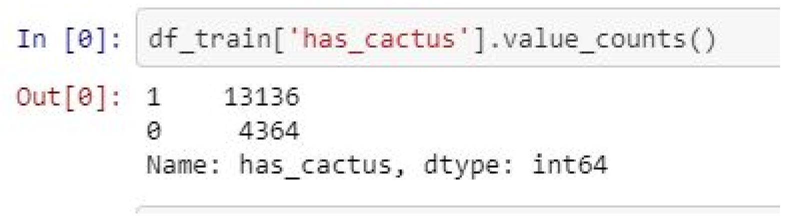
- Connected layers of Conv2D for extracting features from a small 3x3 kernel.
- The number of filters has been increased as the model goes deep to extract more features.
- MaxPooling and Dropout used.
- Densely connected layer.
- The activation function is ReLU in all Convolutional layers.
- Since we want to predict the probabilities last layer used is SoftMax Layer.
- We have a binary classification problem hence the loss function used is binary_crossentropy.
- Adam optimizer and accuracy metrics have been used.

## III. Methodology

## Data Preprocessing

We clearly observed data imbalance during exploratory visualization phase. There are

13136 has_cactus(1) images and 4364 No_cactus(0) images in the train folder.



```
In [0]: df_train['has_cactus'].value_counts()

Out[0]: 1    13136
        0     4364
        Name: has_cactus, dtype: int64
```

For handling imbalanced data in supervised learning, there are several alternatives.2 of them are: Oversampling and SMOTE discussed in paper “Dealing with Bias via Data

Augmentation in Supervised Learning Scenarios” written by Vasileios Iosifidis and Eirini Ntoutsi. In our project we will implement Oversampling i.e."augment" training data via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better by reducing bias.

In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class.

This class allows us to:

1. Configure random transformations and normalization operations to be done on image data during training.We applied data augmentation only to training dataset leaving out validation dataset.
2. Instantiate generators of augmented image batches (and their labels) via . .flow_from_dataframe.
3. .flow_from_dataframe takes the dataframe and the path to a directory and generates batches of augmented/normalized data.
4. We split 15000 ids as training and 2500 as validation data set respectively.
5. We pass y_col=’has_cactus’ as target data to the .flow_from_dataframe generator.This generator is then used with the Keras model methods that accept data generators as inputs, fit_generator and predict_generator.

sklearn.utils.class_weight.**compute_sample_weight**(*class_weight*, *y*, *indices=None*) [source] was also tried to estimate sample weights by class for fixing biased data unbalanced dataset but this led to overfitting. Code

## Implementation

1. After applying required data augmentation, transfer learning from VGG16 pre trained model has been implemented.
2. We start by importing VGG16 pre trained model with imagenet weights and removing the (original classifier) or top layer of VGG16 and adding a new classifier that fits our purpose of identifying columnar cactus.
3. Classifier is built with 1 full connected dense layer, ‘ReLu’ activation, dropout of 0.5 and a dense layer with single node followed by a sigmoid activation layer.
4. Sigmoid activation layer is used because we have to classify between has_cactus(1) and no cactus(0).In Keras, to perform binary classification,sigmoid

activation and ‘binary_crossentropy’ are applied.This is based on the paper(Chollet 2017).

Why Transfer Learning is implemented?

Transfer learning is a popular method in computer vision because it allows us to **build accurate models in a timesaving way (Rawat & Wang 2017)**. With transfer learning, instead of starting the learning process from scratch, we start from patterns that have been learned when solving a different problem. This way we leverage previous learnings and avoid starting from scratch. [source]

In computer vision, transfer learning is usually expressed through the use of pre-trained models. A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published literature (e.g. VGG, Inception, MobileNet). A comprehensive review of pre-trained models’ performance on computer vision problems using data from the ImageNet (Deng et al. 2009) challenge is presented by Canziani et al. (2016).

A typical CNN has two parts:

- Convolutional base, which is composed of a stack of convolutional and pooling layers. The main goal of the convolutional base is to generate features from the image. **In our problem we have taken the convolutional base of VGG16 with Imagenet weights.Imagenet consists of 4486 plant,flora,plant life subset of images which is related to our columnar cactus dataset at high level.**
- Classifier, which is usually composed by fully connected layers. The main goal of the classifier is to classify the image based on the detected features of has_cactus or no_cactus defined in the problem statement. A fully connected layer is a layer whose neurons have full connections to all activation in the previous layer.

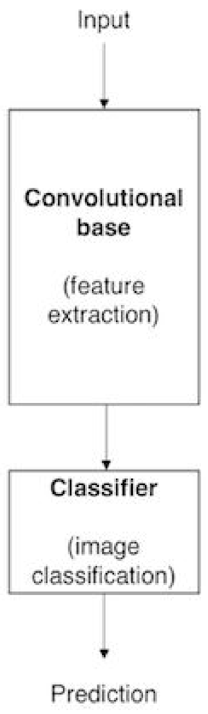


Figure 1. Architecture of a model based on CNN.

## Refinement

An Initial solution using data augmentation and CNN from scratch has been implemented.

The architecture of the model was:

1. Six convolution layers with Max Pooling in between them. Dropout is also added after Max Pool layer in order to prevent overfitting. Dropout is basically a regularization technique. The number of feature maps in each of these layers are 32,64,64,128,128 and 256 with 'relu' as activation function.
2. After that the matrix is flattened to a single row vector, and passed to 1 Dense layer with 512 nodes and  'relu' as activation function.
3. For the last layer, Dense function has 1 (node) for our binary classification problem consisting of 0 and 1 class labels. Sigmoid is used to get predicted probabilities of 0 or 1.

Github Code Link

But this model gave us an AUC score of 0.5 only on kaggle public leaderboard. Hence we chose transfer learning with CNN over CNN built from scratch.

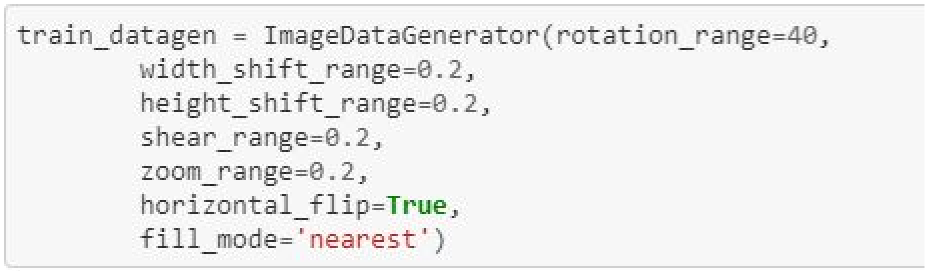
Before arriving at our final architecture tried several architectures with keras applications like RESNET50,Xception and InceptionV3 but found VGG16 to be most efficient with our current task.

RESNET50 architecture Code Link with only 75% accuracy

## Final architecture transfer learning with VGG16 and hyperparameters:

Code - Github Link

- Data Augmentation of training data with below specific augmentation parameters:



```
train_datagen = ImageDataGenerator(rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

- Batch size = 32
- Default size of keras input is 224X224 we took input_size of (150,150,3) based on VGG16 keras application weights pre-trained on ImageNet.
- Dropout of 0.5 has been added in our dense layer to avoid overfitting.
- Model Checkpointing - ‘val_acc’ has been monitored with mode=’max’ and only best weights have been saved to a filepath = "best_model.hdf5"
    - The Keras library provides a checkpointing capability by a callback API.
    - The ModelCheckpoint callback class allows us to define where to checkpoint the model weights, how the file should be named and under what circumstances to make a checkpoint of the model.
- Adam optimizer has been used with a learning rate of 1e-5.
    - The learning rate hyperparameter controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated, such as at the end of each batch of training examples.
    - Several times due to bad learning rate training took longer or didn’t start, after trying several optimizers and Adaptive Learning Rate Methods like :
        - *SGD(lr=0.01,decay=1e-6,momentum=0.9,nesterov=True)*
        - *Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)*

        - *RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)*
        - *Adadelta(lr=1.0, rho=0.95, epsilon=1e-08, decay=0.0)*

    Adam(lr=1e-5) gave best results.

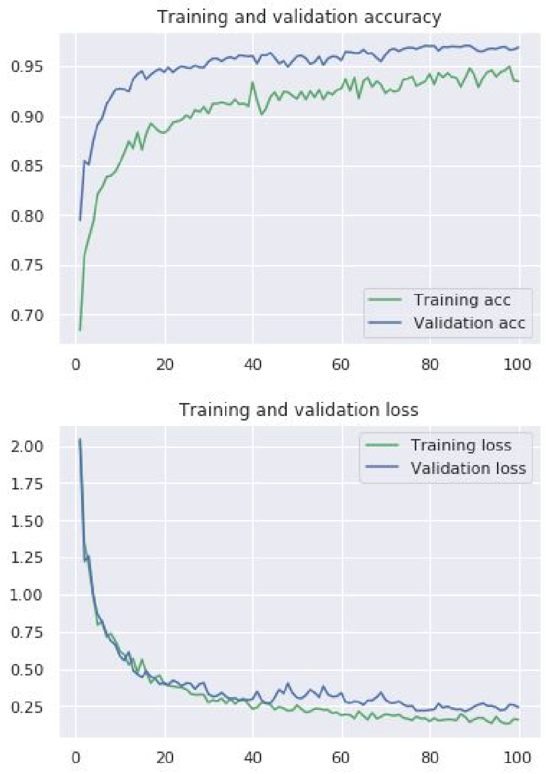
- Arguments of fit_generator: epochs,steps_per_epoch and validation_steps were also tweaked several times to arrive at an optimal solution. The calculations were based largely on the keras documentation of fit_generator

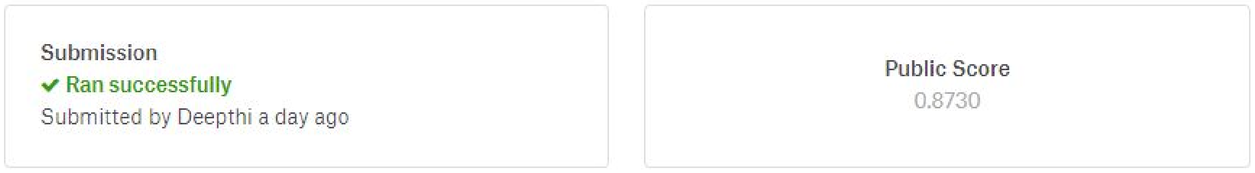
## IV. Results

### Model Evaluation and Validation

Final architecture gave a kaggle public score of 0.87 as compared to the CNN from scratch with public score of 0.5 on the undisclosed test set used by kaggle to generate public score.The final loss value is 0.24 with 97% accuracy. Kaggle kernel

**Final Architecture:**




Submission
✔ Ran successfully
Submitted by Deepthi a day ago

Public Score
0.8730

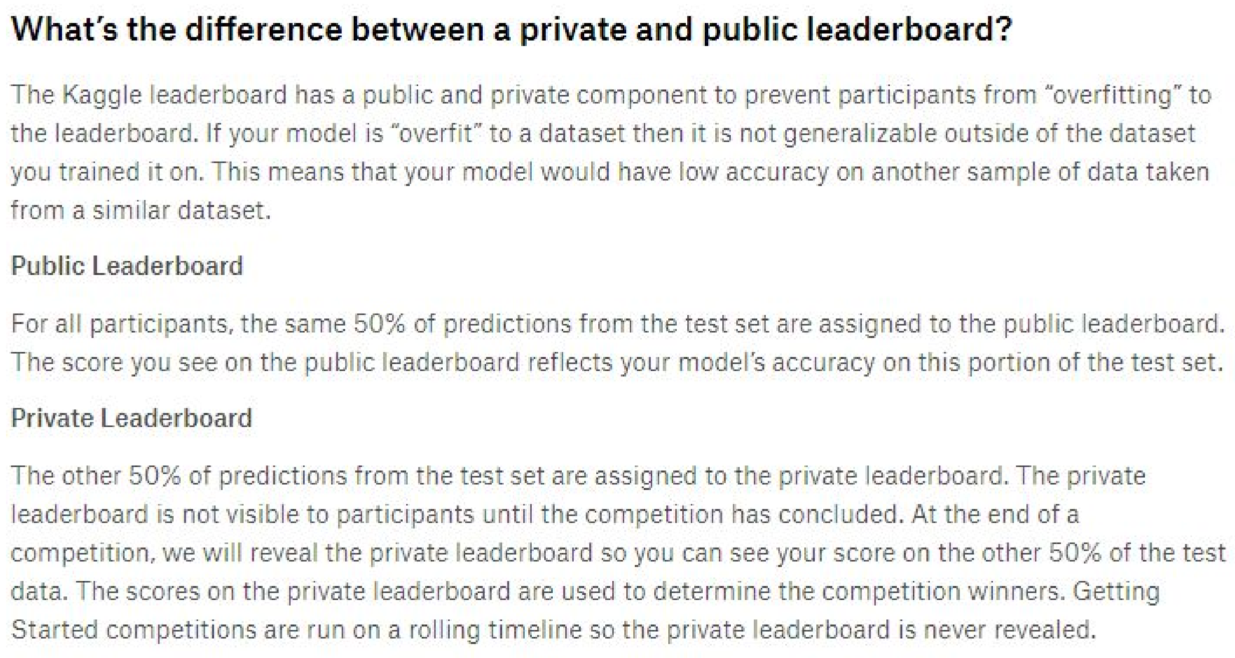
## Justification

Submissions are evaluated on AUC - area under the ROC curve between the predicted probability and the observed target as stated in the evaluation section of Aerial Cactus Identification Kaggle Competition.

The benchmark result reported has achieved kaggle public score of 0.9703. Kaggle public score is calculated on 50% of undisclosed new test set by kaggle.

While our solution could achieve only a public score of 0.8730 after working on several submissions ranging from a public score of 0.5-0.8730 which is less than the benchmark model, it was clearly mentioned in the getting started Titanic competition FAQ on how Kaggle computes its public and private score leaderboard. This clearly signifies that our model with  97% accuracy generalizes pretty well on a new dataset.

Private Leaderboard score is out of scope for our project as the Aerial Cactus Identification Kaggle Competition has concluded long before.


**What's the difference between a private and public leaderboard?**

The Kaggle leaderboard has a public and private component to prevent participants from "overfitting" to the leaderboard. If your model is "overfit" to a dataset then it is not generalizable outside of the dataset you trained it on. This means that your model would have low accuracy on another sample of data taken from a similar dataset.

**Public Leaderboard**

For all participants, the same 50% of predictions from the test set are assigned to the public leaderboard. The score you see on the public leaderboard reflects your model's accuracy on this portion of the test set.

**Private Leaderboard**

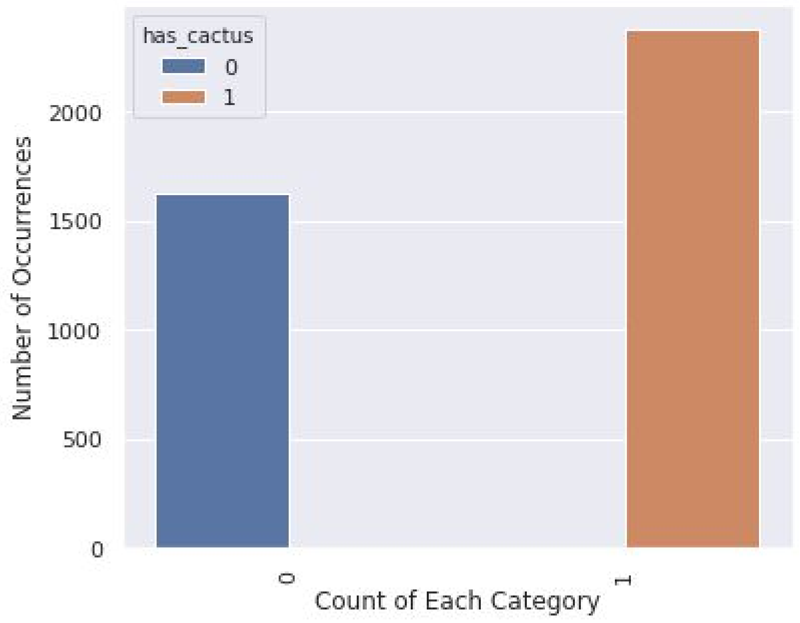
The other 50% of predictions from the test set are assigned to the private leaderboard. The private leaderboard is not visible to participants until the competition has concluded. At the end of a competition, we will reveal the private leaderboard so you can see your score on the other 50% of the test data. The scores on the private leaderboard are used to determine the competition winners. Getting Started competitions are run on a rolling timeline so the private leaderboard is never revealed.

## V. Conclusion

### Free-Form Visualization

Finally a count plot of the predictions (submission.csv) has been visualized to get an understanding of the results. Below plot clearly shows that we could solve the problem statement defined i.e. classifying given images into has_cactus and No_cactus.

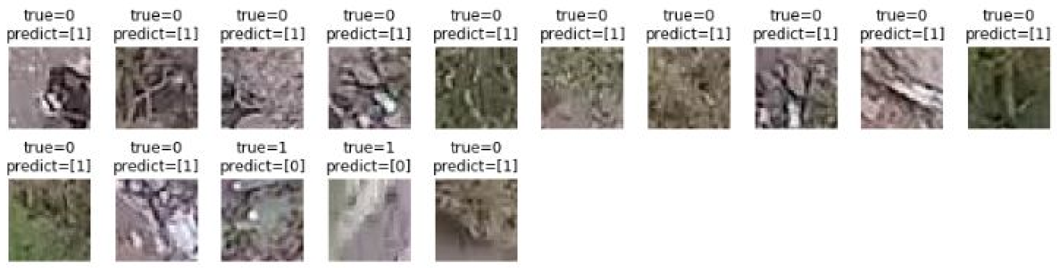
Fig: Count of predictions of has_cactus and No_cactus on the test set.



Error Analysis was also performed to understand the wrongly classified labels from the validation data as we don't have access to the Kaggle test data:

1. Error analysis on the predictions for Validation Data was done.
2. We observed the labels wrongly classified to identify any pattern.

```
plt.figure(figsize=(15,8))
for i in range(len(error_list)):
    plt.subplot(4, 10, i+1)
    plt.imshow(load_img(train_path+train_df.iloc[error_list[i]]['id']))
    plt.title("true={}\npredict={}".format(train_Y[error_list[i]],
                                          pred_dev[error_list[i]]), y=1)
    plt.axis('off')
plt.subplots_adjust(wspace=0.3, hspace=-0.1)
plt.show()
```



These are those in the validation set that were classified incorrectly. We could clearly see that there was some noise created due to the columnar dark patterns in the images which were confused with the columnar cactus images.Hence,wrongly classified.

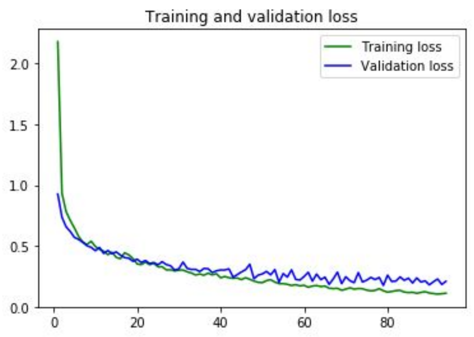
## Reflection

We started with biased sample data of columnar cactus images(has_cactus). The count of images with no_cactus were far less than has_cactus. To handle imbalance in data we tried implementing SMOTE,sklearn class_weights and oversampling. Though SMOTE and sklearn class_weights are more effective measures than oversampling with data augmentation results achieved were poor due to limited knowledge with these 2 methods. An attempt to implement class_weights has been attached here. This resulted in a public score of 0.5 accuracy of 97% and validation loss of 0.21.

Our final architecture is summarized below:

1. Data augmentation, transfer learning using VGG16 imagenet weights CNN architecture removing the top layer and connecting with dense layers followed by Sigmoid activation function.
2. Due to the nature of our problem being binary classification between has_cactus and no_cactus categories,loss function used is ‘binary_crossentropy’
3. Dropout of 0.5 has been added to prevent overfitting.

4. Metrics='accuracy' and after experimenting with several optimizers and learning rates like SGD with Nesterov and Momentum,Adadelta and RMSProp, based on the blog, Adam optimizer with a learning rate of 1e-5 has been baselined.
5. Model checkpoint has been used to monitor 'val_acc' mode='max' and best weights have been saved.
6. ReduceROnPlateu and Early stopping were also tried during experiments with 'patience=15',mode='min' for monitor='val_loss' and added to callbacks list along with modelcheckpoint for passing to .fit_generator [code]. These were not used in the final architecture as they did not add much value to the accuracy.
7. Learning curves and validation loss were monitored across several experiments to arrive at the best model.



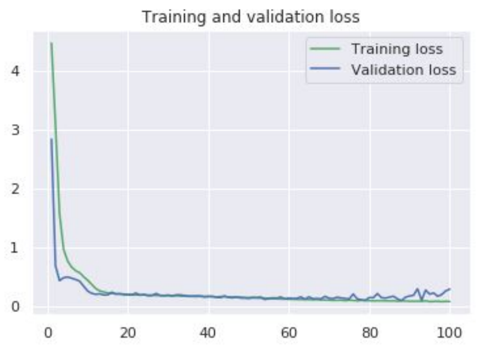


Fig 1. Transfer Learning with Extreme Hyperparameters

Fig 2. Simple CNN from scratch

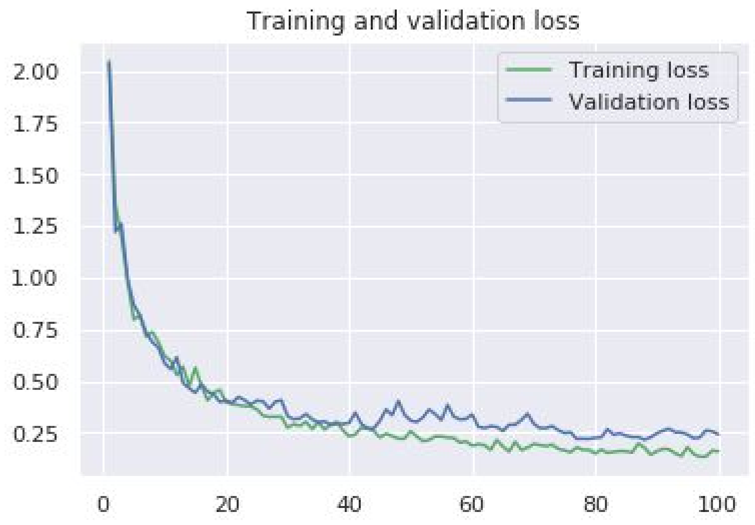


**Fig 3. Final Architecture**

The Final architecture successfully classified with 97% accuracy,validation loss of 0.24 and a Kaggle public score of 0.87. Yes, this approach is largely used across image classification tasks with hyperparameter tuning specific to the problem domain.

## Improvement

After training several architectures using the above techniques results can be merged together to obtain an even better solution. This is known as Model Ensemble and this is one of the widely popular techniques. But  this approach is very computationally expensive. Snapshot Ensembling is one another technique used in Kaggle #1 Winning Approach for Image Classification Challenge

The above methods I researched but did not know how to implement and would definitely consider implementing if i knew, to achieve a higher Kaggle leaderboard score and more generalized model.

---

*References:*

1. *http://machinethink.net/blog/compressing-deep-neural-nets/*
2. *Batch Normalization -- https://r2rt.com/implementing-batch-normalization-in-tensorflow.html*
3. *Relu activation function - https://arxiv.org/abs/1511.07289*
4. *initializers -https://keras.io/initializers/*
5. *tqdm -https://github.com/bstriner/keras-tqdm*
6. *http://image-net.org/challenges/LSVRC/2014/browse-synsets*
7. *http://www.coldvision.io/2016/07/29/image-classification-deep-learning-cnn-caffe-opencv-3-x-cuda/*
8. *https://keon.io/deep-q-learning/*
9. *https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/*
10. *steps-per-epoch = https://github.com/udacity/aind2-cnn/blob/master/cifar10-augmentation/cifar10_augmentation.ipynb*
11. *transfer learning tutorial = https://alexisbcook.github.io/2017/using-transfer-learning-to-classify-images-with-keras/*

12. *dividing training and testing data into test labels-bottleneck features - https://craigmartinson.com/post/keras-transfer-learning/*
13. *https://www.kaggle.com/kmader/transfer-learning-with-inceptionv3*
14. *https://medium.com/neuralspace/kaggle-1-winning-approach-for-image-classification-challenge-9c1188157a86*
15. *SMOTE- https://medium.com/analytics-vidhya/balance-your-data-using-smote-98e4d79fcddb*
16. *Learning rate - https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/*
17. *Adadelta,Adam,RMSProp = https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1*
18. *comparing graph model accuracy - https://github.com/sukilau/Ziff-deep-learning/blob/master/3-CIFAR10-lrate/CIFAR10-lrate.ipynb?source=post_page---------------------------*
19. *Model checkpoints - https://machinelearningmastery.com/check-point-deep-learning-models-keras/*
20. *Best pre trained model for image classification tasks - https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751*
21. *optimizing Gradient Descent - http://ruder.io/optimizing-gradient-descent/*
22. *learning curves for machine learning performance -https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/*
23. *early stopping - https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/*
24. *ROC Curve - https://github.com/Tony607/ROC-Keras/blob/master/ROC-Keras.ipynb*