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

The motivation for picking up this project is mainly getting started with Kaggle competitions.

What is Kaggle?

Kaggle is an online community of data scientists and machine learners, owned by Google LLC. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

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.

Solution Statement:

1. Download and preprocess the training images.
2. Check for data imbalance and unbalanced classes.
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 Metrics(Validation accuracy, loss and area under the ROC curve).
5. Visualize results.
6. Predict the classes and compare the results based on the AUC score of the benchmark model.

Metrics

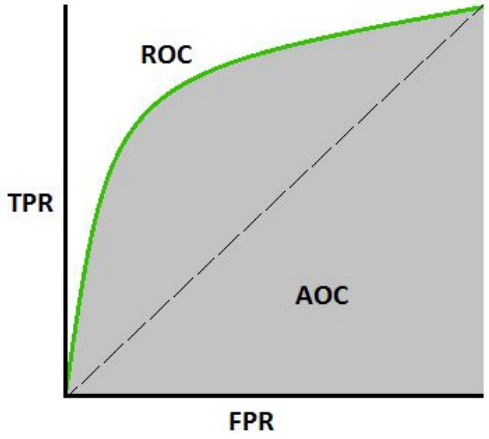
Submissions are evaluated on area under the ROC curve or AUC between the predicted probability and the observed target.

We will also be calculating accuracy as our secondary metric.

What is AUC - ROC Curve?

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. [Source]

The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.



Defining terms used in AUC and ROC Curve:

TPR (True Positive Rate) / Recall /Sensitivity

$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN}$$

Specificity

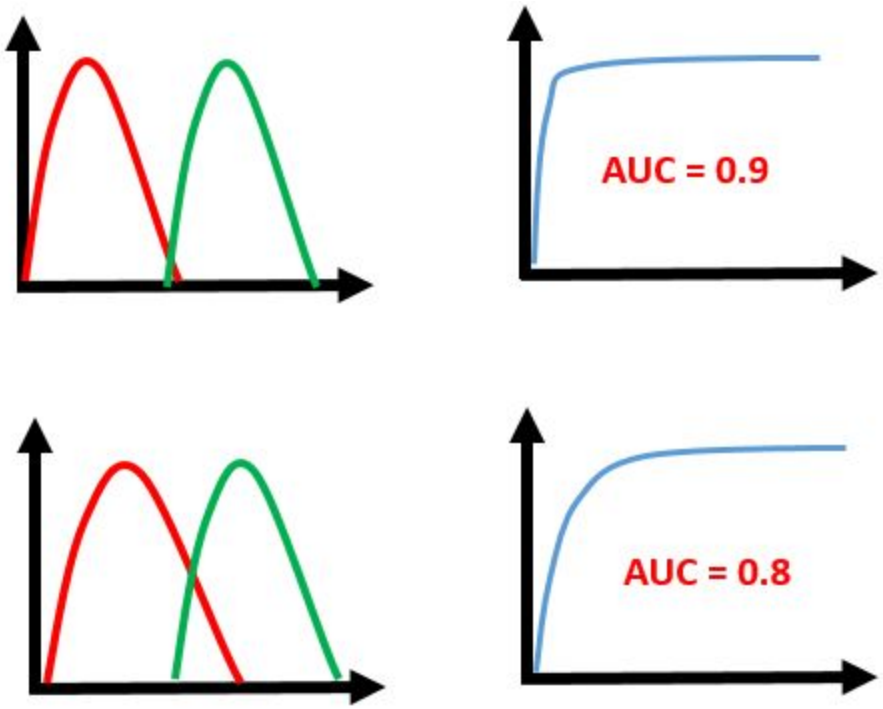
$$\text{Specificity} = \frac{TN}{TN + FP}$$

FPR(False Positive Rate):

$$FPR = 1 - \text{Specificity}$$

$$= \frac{FP}{TN + FP}$$

Let us take a few examples:



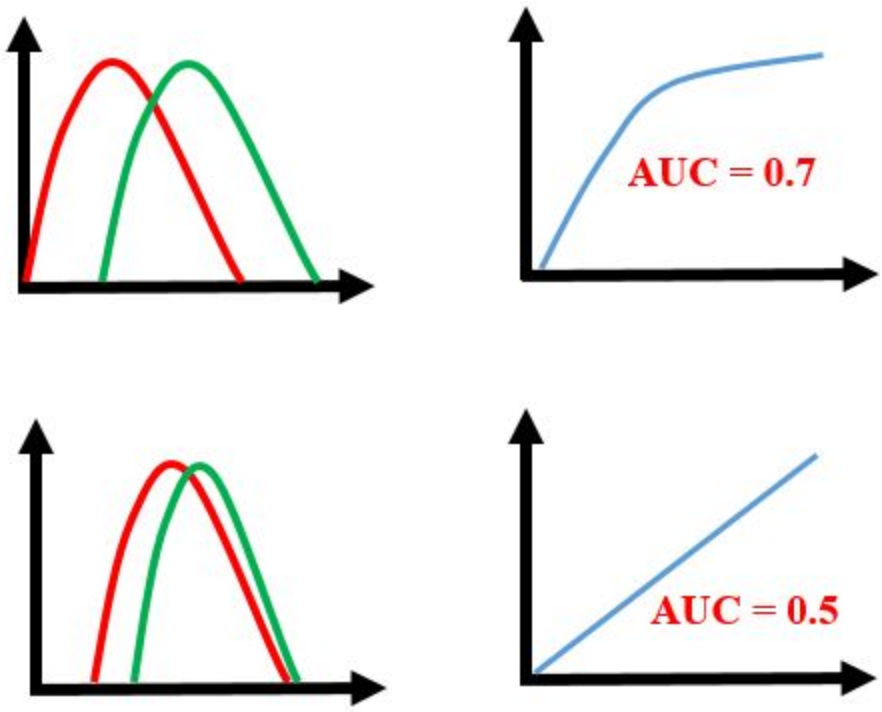


Fig- Source: Greyatom

As we see, the first model does quite a good job of distinguishing the positive and negative values. Therefore, there the AUC score is 0.9 as the area under the ROC curve is large.

Whereas, if we see the last model, predictions are completely overlapping each other and we get the AUC score of 0.5. This means that the model is performing poorly and it is predictions are almost random.

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.[Source]

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

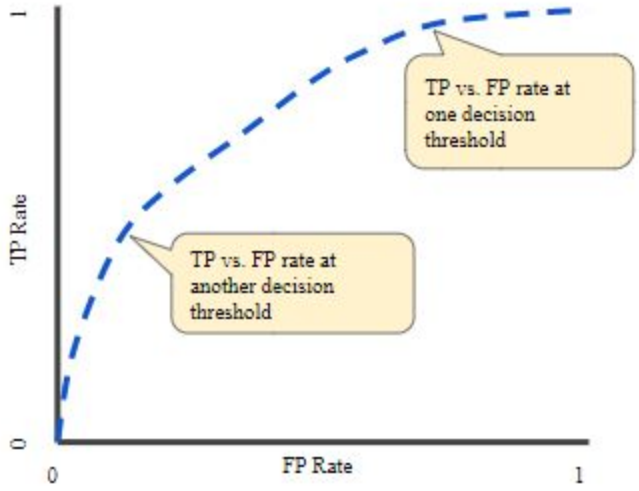


Fig. **TP vs. FP rate at different classification thresholds**(source)

Accuracy:

Classification Accuracy is what we usually mean when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

It works well only if there are an equal number of samples belonging to each class.

For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to class A.

When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the test accuracy would drop down to 60%.Classification Accuracy is great, but gives us the false sense of achieving high accuracy.

The real problem arises, when the cost of misclassification of the minor class samples are very high. If we deal with a rare but fatal disease, the cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests.[source]

Why ROC-AUC metric is the Evaluation Metric?

In our project, Kaggle gave us Area under the ROC as the final evaluation metric. As we could see clearly above from the definition of accuracy metric, classification accuracy can largely vary due to data imbalance. This may be a potential reason for selecting AUC-ROC as the final metric instead of just accuracy. However we will be calculating both to arrive at the best model and observe the variations.

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.

Kaggle calculates AUC on an undisclosed test set and gives public score.Our Benchmark model has obtained an AUC/public score of 0.97 on kaggle public leaderboard. We will be setting a goal to obtain an AUC of more than 0.60 based on our benchmark model's performance.

II. Analysis

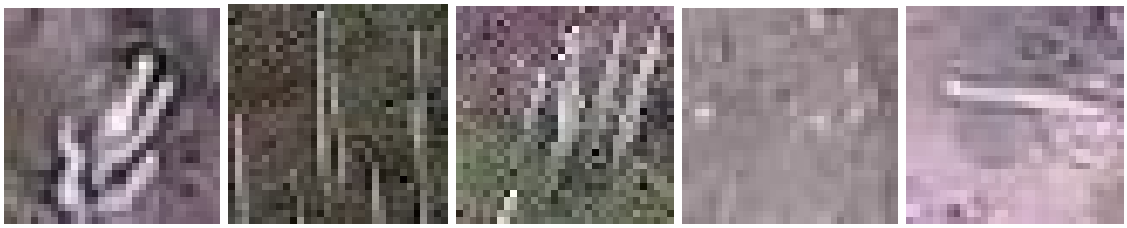
Data Exploration

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:

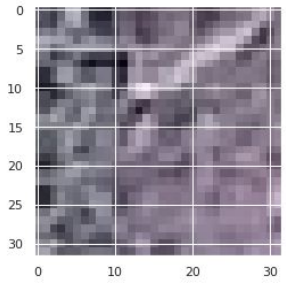


test/ - the test set images (labels to be predicted)

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.

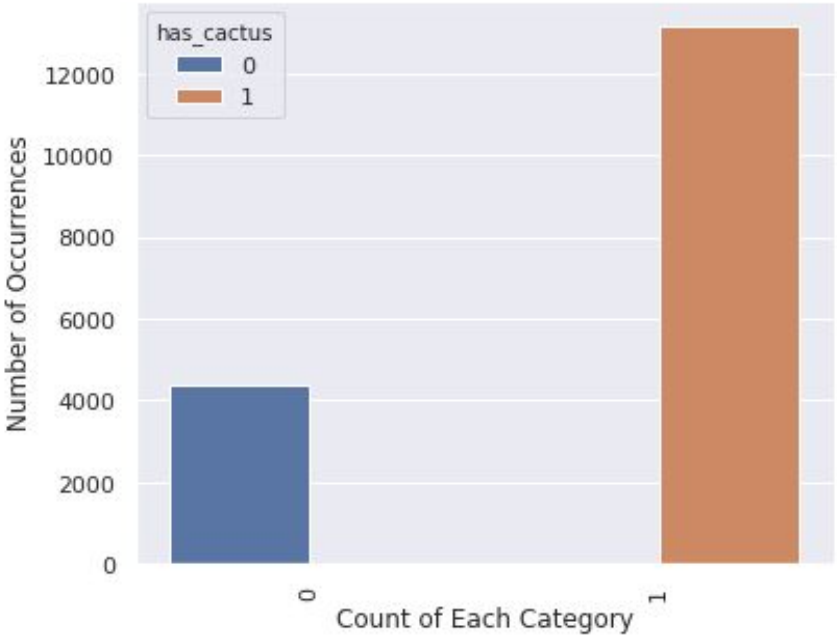
Kaggle Competition Link: https://www.kaggle.com/c/aerial-cactus-identification

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

Exploratory Visualization

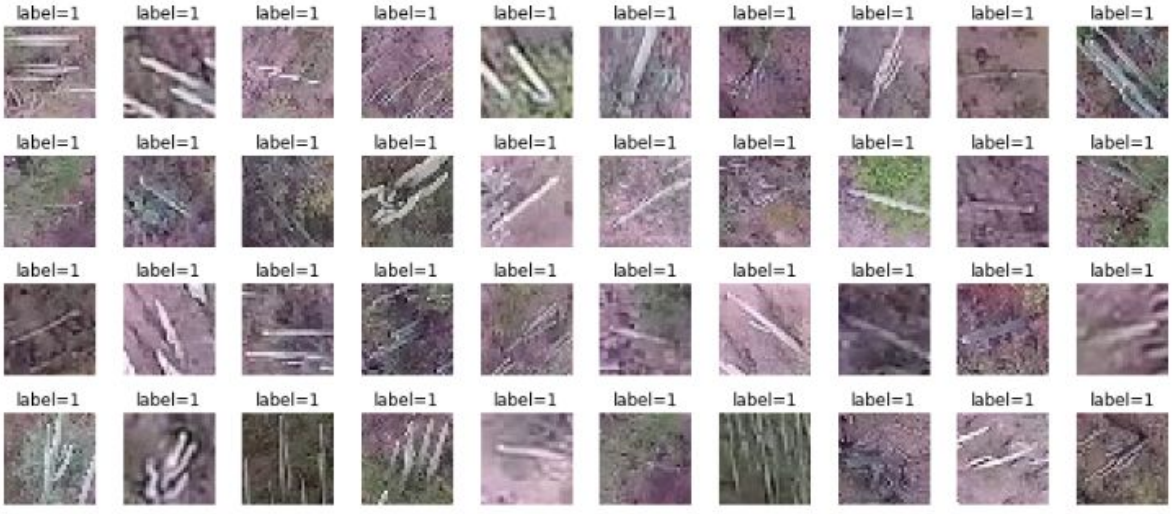
From the Data exploration section, we can clearly see that there is an imbalance in the data. We have a biased dataset. We have has_cactus for more data equal to 1. We used seaborn library to plot a count plot on has_cactus column of train.csv to obtain the below bar chart.

Fig: Count of has_cactus in train.csv

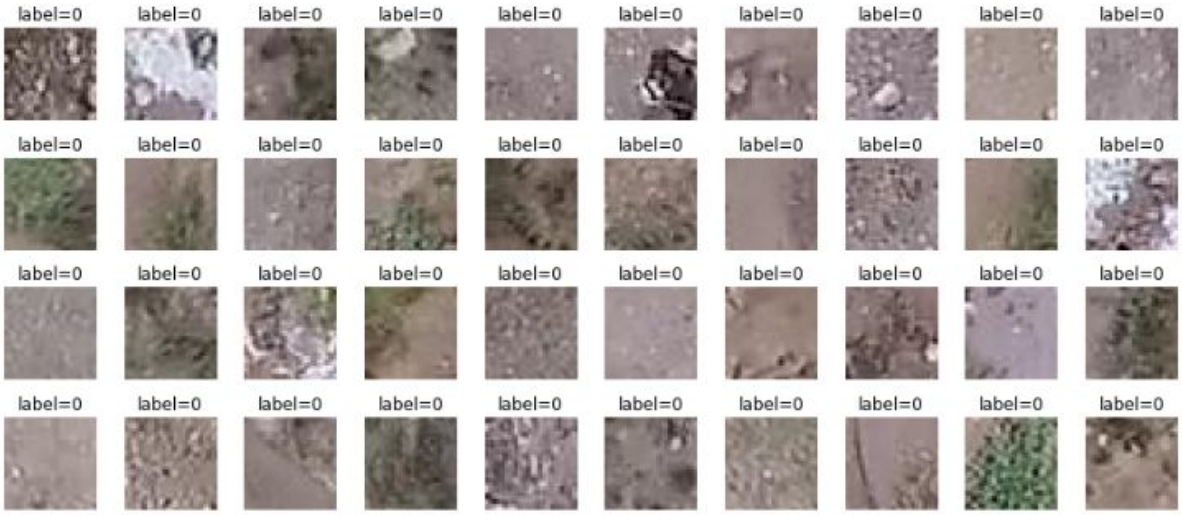


Source: [Project Github Link]

Here are some sample images with label=1:



Some sample images with label=0:

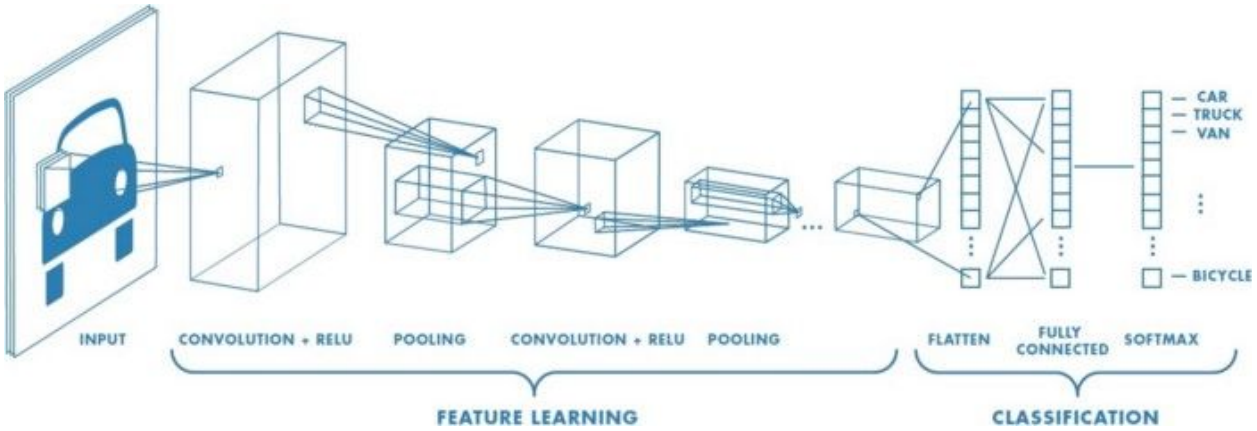


Algorithms and Techniques

For this type of image classification problems, Convolution Neural network(CNN) class of deep learning algorithm is the best solution. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos.

What is a Convolutional Neural Networks(CNN)?

Convolutional neural networks (CNN) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification.



Source : MathWorks (https://goo.gl/zondfq)

The Regular Neural Networks(NN) is not capable of dealing with images. Just imagine each pixel is connected to one neuron and there will thousands of neurons which will be computationally expensive. CNN handles images in different ways, but still it follows the general concept of NN(Neural Network)

Convolution

Let's dive into how convolution layer works. Convolution has got set learn-able filters which will be a matrix(width, height, and depth). We consider an image as a matrix and filter will be sliding through the image matrix as shown below to get the convoluted image which is the filtered image of the actual image.

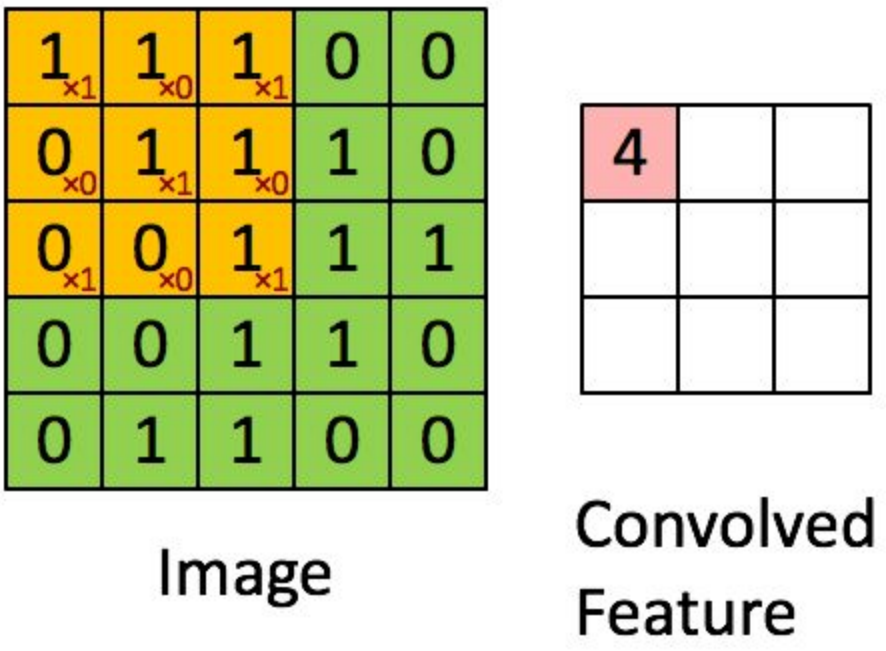


Fig: Source

Depending upon the task, more than one filter is used in the model to cater the different features. Feature might be looking for a cacti, looking for colour etc. Filter matrix value is learned during the training phase of the model.

INPUT IMAGE

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

WEIGHT

1	0	1
0	1	0
1	0	1

429

Source: Udacity deep learning course

Padding is another important factor in convolution. If we apply a filter on the input image, we will be getting output matrix with less size than the original image. Padding comes into play, if we need to get the same size to output as input size.

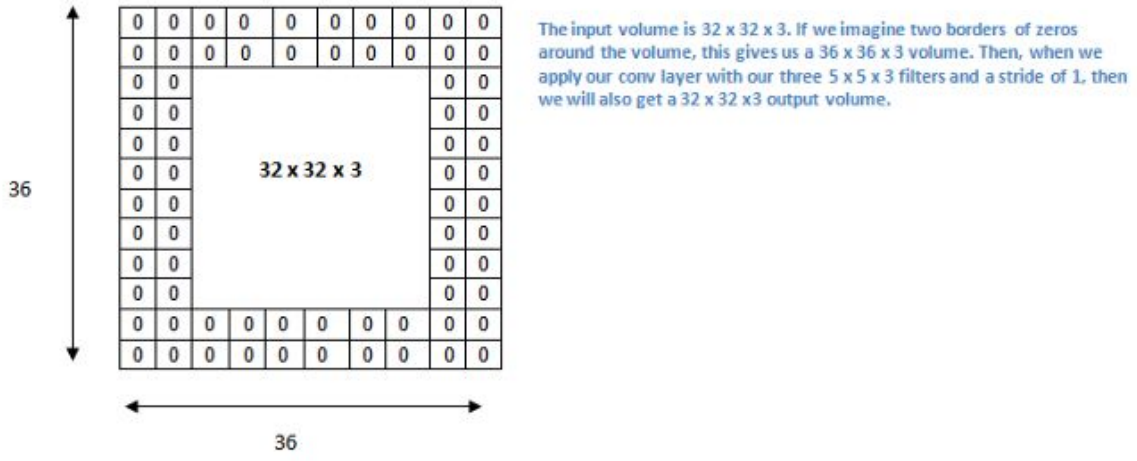


Fig:Source

Let's consider the above example. Apply a zero padding of size 2 to that layer. Zero padding pads the input volume with zeros around the border. If we think about a zero padding of two, then this would result in a 36 x 36 x 3 input volume of 32 x 32 x 3.

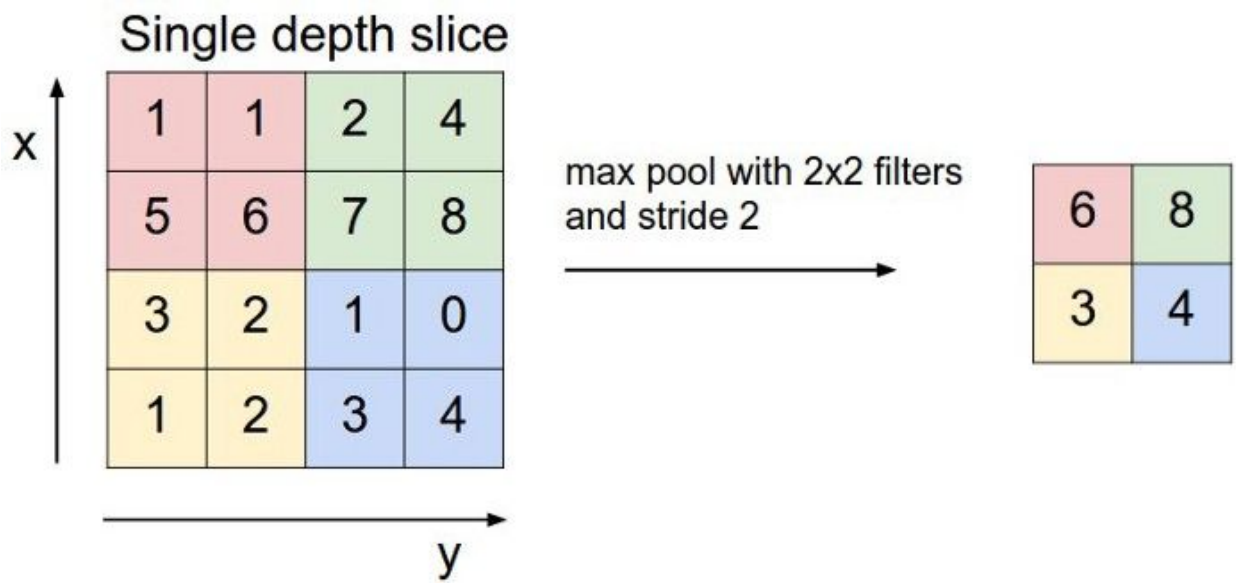
Activation function

Activation functions are functions that decide, given the inputs into the node, what should be the node's output? Because it's the activation function that decides the actual output, we often refer to the outputs of a layer as its "activations". One of the popular activation functions in cnn is ReLu.

References:

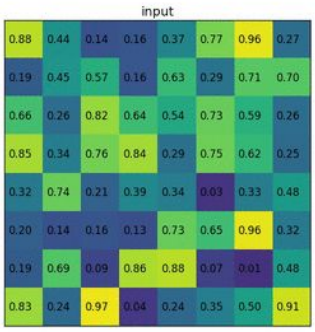
Max pooling

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several nonlinear functions that can do pooling, in which, max pooling is the most common one.



Source: http://cs231n.github.io/convolutional-networks

According to wikipedia, It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features.



Source: Wikipedia

The pooling layer helps to reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network. This also controls the over-fitting.

Fully connected

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.(Wikipedia)

Sigmoid Activation Function

The Sigmoid Function curve looks like a S-shape.

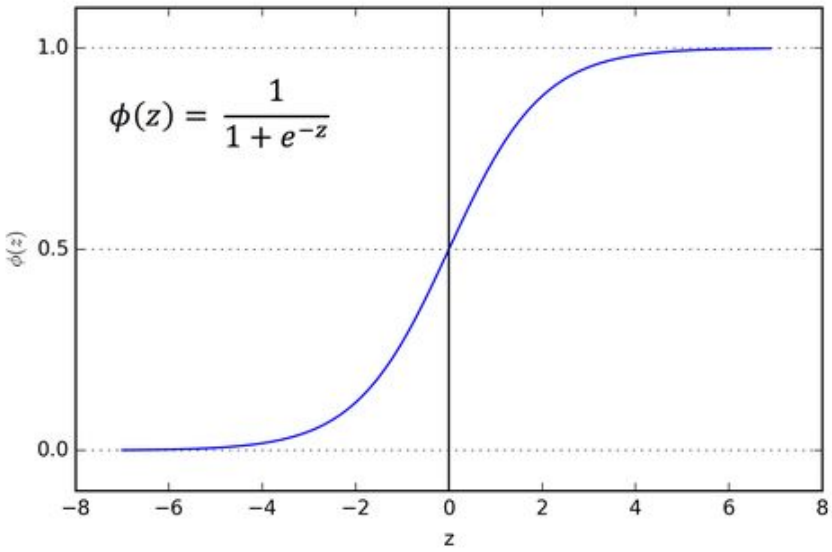


Fig: Sigmoid Function

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models like in our project where we have to predict the

probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. **Refer Other Activation Functions**

For our Aerial Cactus Identification/Image classification problem, we will be generating a CNN using Keras: The Python Deep Learning Library and transfer learning using VGG16.

Why Keras?

- Allows for easy and fast prototyping(through user friendliness, modularity, and extensibility).
- Runs seamlessly on CPU and GPU.

What is transfer learning and VGG16?

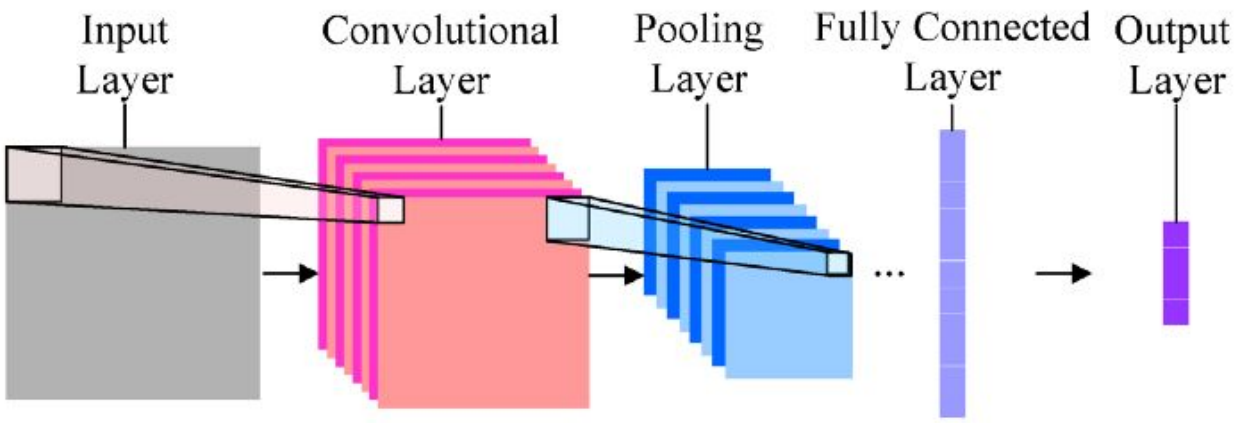


Fig: Source

Though training a CNN from scratch is possible for small projects, most applications require the training of very large CNN's and this takes extremely huge amounts of processed data and computational power. That's where transfer learning comes into play. In transfer learning, we take the pre-trained weights of an already trained model(one that has been trained on millions of images belonging to 1000's of classes, on several high power GPU's for several days) and use these already learned features to predict new classes.

The advantages of transfer learning are that:

1: There is no need of an extremely large training dataset.

2: Not much computational power is required.As we are using pre-trained weights and only have to learn the weights of the last few layers.

Transfer Learning with VGG16 architecture:

We will use the VGG16 architecture, pre-trained on the ImageNet dataset because the ImageNet dataset contains 4486 plant classes, this model will already have learned features that are relevant to our classification problem.

The building of a model is a 3 step process:

- Importing the pre-trained model and adding the dense layers.
- Loading train data into ImageDataGenerators.
- Training and Evaluating model.

We will be importing VGG16 pretrained model in Keras.

Data will 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.

Model Fine Tuning:

The internal parameters of a Model play a very important role in efficiently and effectively training a Model and produce accurate results. This is why we use various Optimization strategies and algorithms to update and calculate appropriate and optimum values of such model's parameters which influence our Model's learning process and the output of a Model. **Now what are the different types of Optimization Algorithms used in Neural Networks ? Refer here**

Why we choose Adam optimizer?

Adam works well in practice and compares favorably to other adaptive learning-method algorithms as it converges very fast and the learning speed of the Model is quiet Fast and efficient and also it rectifies every problem that is faced in other optimization techniques such as vanishing Learning rate , slow convergence or High variance in the parameter updates which leads to fluctuating Loss function.

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. But at the end nothing worked out and the final solution is not using any technique to balance the dataset.

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

The major coding complication that took time was handling data imbalance but at the end implementing CNN with transfer learning and Adam learning rate optimizer resulted in a good AUC of 0.94 on the validation set and kaggle public score of 0.8730.

Here goes our 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.
    a. Other architectures like Xception,RESNET50 and Inceptionv3 have been tried but found VGG16 apt for our solution. Please check the code link for other architectures to understand what did not work.
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).

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.

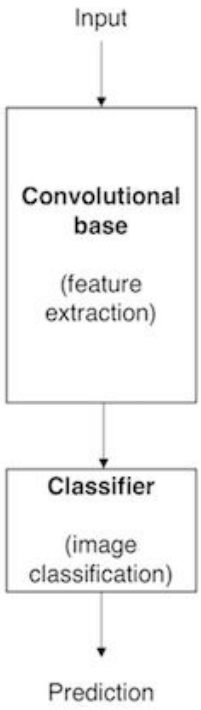


Figure 1. Architecture of a model based on CNN.

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 and Kaggle Kernel

- 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.

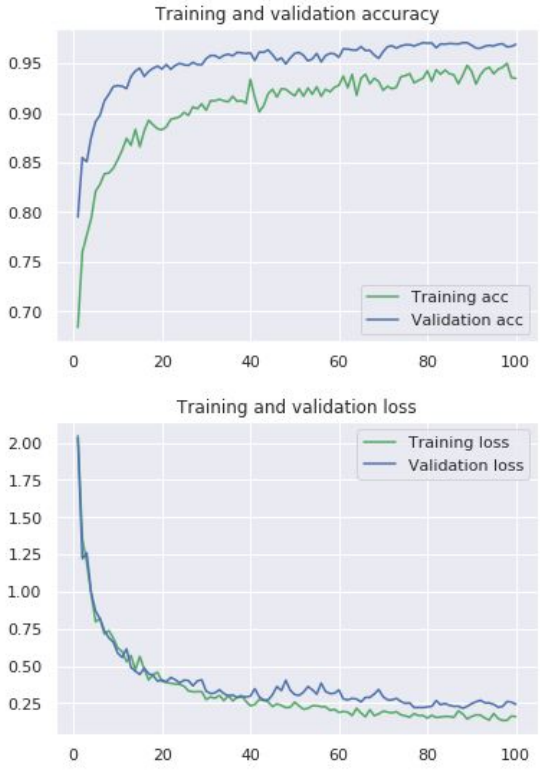
- Arguments of fit_generator: epochs,steps_per_epoch and validation_steps were also tweaked several times to arrive at an optimal solution. The calculations were based largely on the keras documentation of fit_generator

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:



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

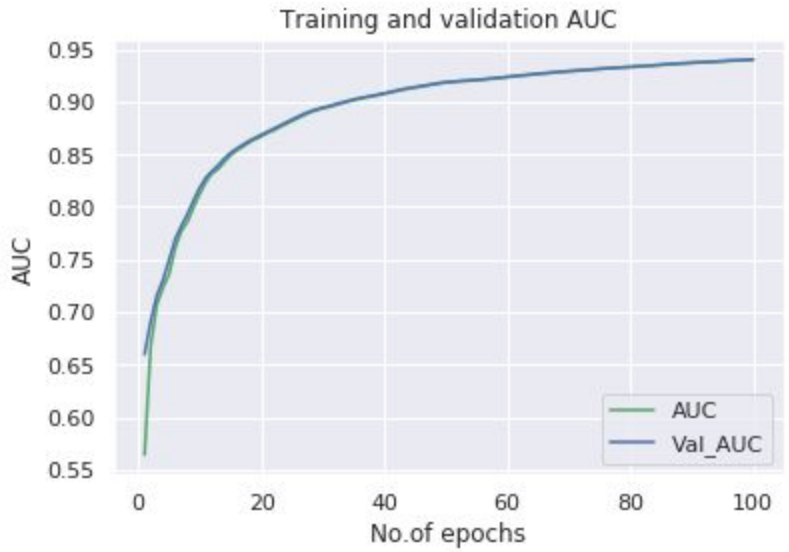
For all participants, the same 50% of predictions from the test set are assigned to the public leaderboard. The score you see on the public leaderboard reflects your model's accuracy on this portion of the test set.

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



Based on the evaluation metric defined in the Metrics section, we computed area under the ROC curve for Training and validation sets over 100 epochs resulting in Training and Validation AUC of 0.94.

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. 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.

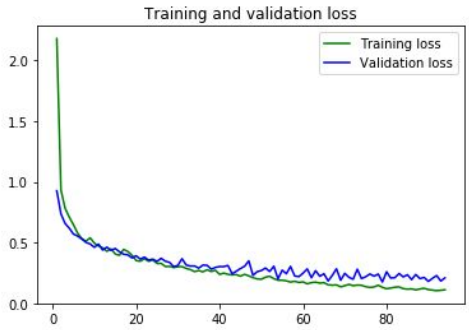


Fig 1. Transfer Learning with Extreme Hyperparameters

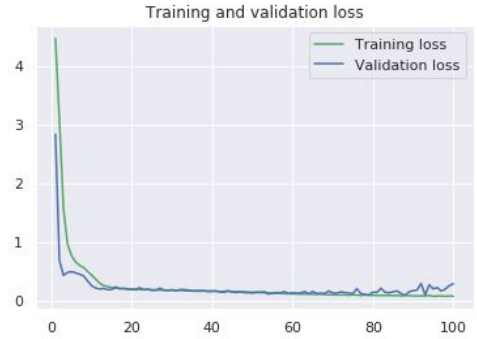


Fig 2. Simple CNN from scratch

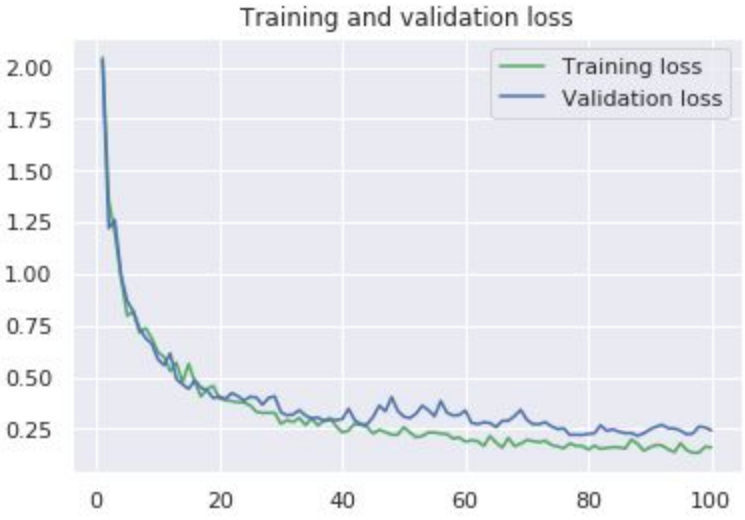


Fig 3. Final Architecture

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*