Machine Learning Engineer Nanodegree

Capstone Project

Asset Karazhay

July 2nd, 2019

I. Definition

Project Overview

Tehuacán-Cuicatlán Valley is a semi-arid zone in the south of Mexico. It was inscribed in the World Heritage List by the UNESCO in 2018. This unique area has wide biodiversity including several endemic plants. Unfortunately, human activity is constantly affecting the area. A way to preserve a protected area is to carry out autonomous surveillance of the area. A first step to reach this autonomy is to automatically detect and recognize elements in the area.

An image dataset was generated for this study by VIGIA project research team, containing more than 10,000 image examples. The researchers used this dataset to train a modify LeNet-5 Convolutional Neural Network. Experimental results have shown a high recognition accuracy, 0.95 for the validation set, validating the use of the approach for columnar cactus recognition. Research paper: Efren López-Jiménez, Juan Irving Vasquez-Gomez, Miguel Angel Sanchez-Acevedo, Juan Carlos Herrera-Lozada, Abril Valeria Uriarte-Arcia, Columnar Cactus Recognition in Aerial Images using a Deep Learning Approach. Ecological Informatics. https://doi.org/10.1016/j.ecoinf.2019.05.005

Problem Statement

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. A first step in such an effort is the ability to recognize the vegetation inside the protected areas.

We have pictures of plants from Tehuacán-Cuicatlán Valley. We need to identify which pictures contain columnar cactus (Neobuxbaumia tetetzo). This is a binary image classification problem. We will need to produce a neural network that could look at the images it has never seen before and correctly identify if the image contains columnar cactuses or not.

The following steps will be used:

- To combat imbalanced data the lacking image class (pictures not containing the columnar cactus) will be balanced out by adding augmented pictures (flipping images horizontally, rotating, shifting and zooming on the images)
- 80:10:10 split will be used for training, validation and testing data
- A benchmark image classification model with 4 Convolutional, 1 MaxPooling and 2 Fully Connected layers will be created and its performance will be measured
- Than a project CNN will be modeled. Use VGG16 pre-trained convolutional layers and add fully connected layers manually
- Feed the network with rescaled images. Pixel values rescaled to between 0 and 1 by dividing each pixel by 255
- Final testing of benchmark and transfer learning models will be conducted using a test set of samples that the model haven’t seen before.

Note:

- Evaluation of proposed model through Kaggle submission is impossible. Because this is a kernel only competition. There is no way of saving additional augmented images on Kaggle kernel.
- In Keras methods ‘validation dataset’ and ‘testing dataset’ are used interchangeably. Throughout this project to minimize the confusion data samples used to validate the CNN model on each epoch over and over will be called validation samples/set as it is been called this way by Keras model training progress info. Data samples/set used only once at the end of training to test how model generalize on unseen data will be called test samples/set.

Metrics

Kaggle.com Aerial Cactus Identification competition uses accuracy as the evaluation metric. This project will use accuracy as well. To combat imbalanced data the lacking image class (pictures not containing the columnar cactus) will be balanced out by adding augmented pictures (flipping images horizontally, rotating, shifting and zooming on the images)

Accuracy = (True Positive + True Negative) / (True Positive + False Positive + False Negative + True Negative)

II. Analysis

Data Exploration

The dataset contains 17 500 of 32 x 32 thumbnail aerial photographs of plants. The dataset is imbalanced: 13136 pictures contain a columnar cactus (Neobuxbaumia tetetzo), 4364 pictures contain other vegetation. https://www.kaggle.com/c/aerial-cactus-identification/download/train.zip

Image augmentation will be used to balance out the number of pictures that don’t contain columnar cactus.

Attached is also csv file with image labels. https://www.kaggle.com/c/aerial-cactus-identification/download/train.csv

The dataset comes from Kaggle.com Aerial Cactus Identification competition which is aimed at raising the awareness of human impact on environmental diversity.

Exploratory Visualization

Csv file has 2 columns: image ‘id’ and ‘has_cactus’ binary label:

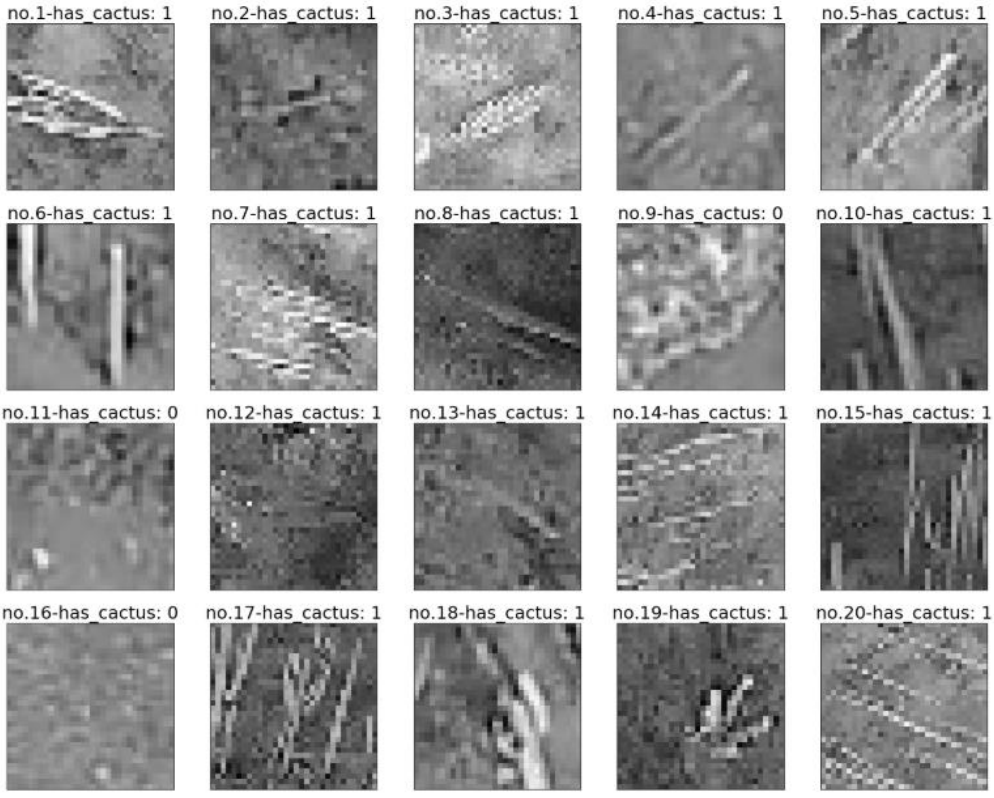
	id	has_cactus
0	0004be2cfeaba1c0361d39e2b000257b.jpg	1
1	000c8a36845c0208e833c79c1bffedd1.jpg	1
2	000d1e9a533f62e55c289303b072733d.jpg	1
3	0011485b40695e9138e92d0b3fb55128.jpg	1
4	0014d7a11e90b62848904c1418fc8cf2.jpg	1
5	0017c3c18ddd57a2ea6f9848c79d83d2.jpg	1
6	002134abf28af54575c18741b89dd2a4.jpg	0
7	0024320f43bdd490562246435af4f90b.jpg	0

Sample images from the dataset with ‘has_cactus’ labels:

Image dimensions are all same: 32x32 pixels, 3 channels- RGB.

For some image classification problems we can convert images in grayscale to reduce the number of channels to 1 and speed up the process of model training. For example, that works well for car make classification problem where we don’t care if the car is red or blue.

Let’s convert same images as above to grayscale and analyze it:



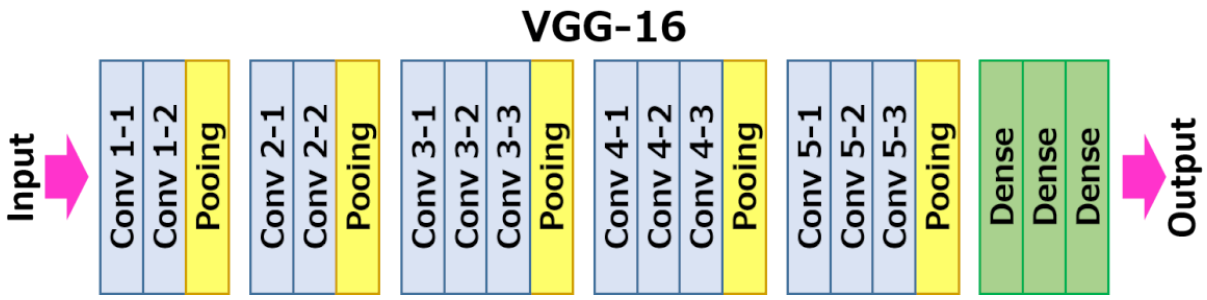
From these examples we can see that images with 3 channels that don't contain cactuses have difference in colors (n.9 and no.11). Some images with 3 channels that do contain cactuses distinguished from their environment not only by their shape but by color as well (no.1 and no.6).

The decision is to feed the model with original color images to prevent losing any information on dataset.

Algorithms and Techniques

The problem at hand is a binary image classification. Common Machine Learning solution for this problem is Convolutional Neural Network model. Instead of training our own model from scratch we will be using transfer learning as a solution. Our choice is the VGG16 Deep Convolutional Neural Network. This network is developed by Karen Simonyan and Andrew Zisserman is trained on 15 million pictures ImageNet dataset.

VGG16 has a following architecture:



Source: neurohive.io

VGG16 is trained on 1666 categories of plants which include almost a million images (source: http://www.image-net.org/about-stats). For the purpose of our problem we won't be training any convolutional layers as the VGG16 network already trained on so many plants.

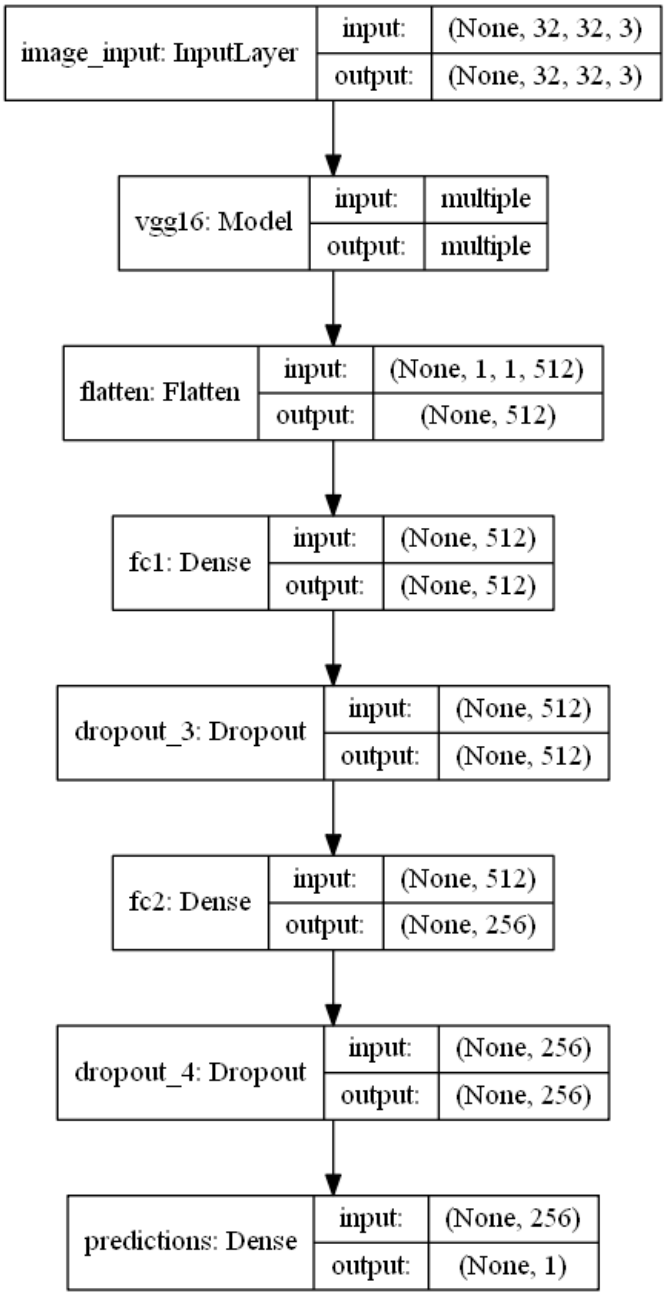
Following architecture will be implemented:

- First is the Input layer with shape of 32x32 pixels and 3 channels
- We will keep all VGG16 pre-trained convolutional layers weights and all fully connected layers will be removed because we are going to concentrate on our binary classification model
- Flattening layer will follow
- Next will be fully connected layer with 512 nodes and ReLu activation followed by 20% Dropout layer
- Next is another fully connected layer with 256 nodes and ReLu activation followed by 20% Dropout layer
- The last one is a Dense prediction layer with Sigmoid activation used, because we have a binary classification

We use ReLu activation in our Dense layers as it rectifies the vanishing gradient problem. Dropout layers are used to avoid model overfitting.

When we compile the model we are choosing:

- Binary Cross-Entropy as our Loss Function because we have binary classification problem.
- Stochastic gradient descent (SGD) was chosen as a robust and fast optimizer.
- Keras high-level neural networks API with TensorFlow backend will be used to achieve this goal.



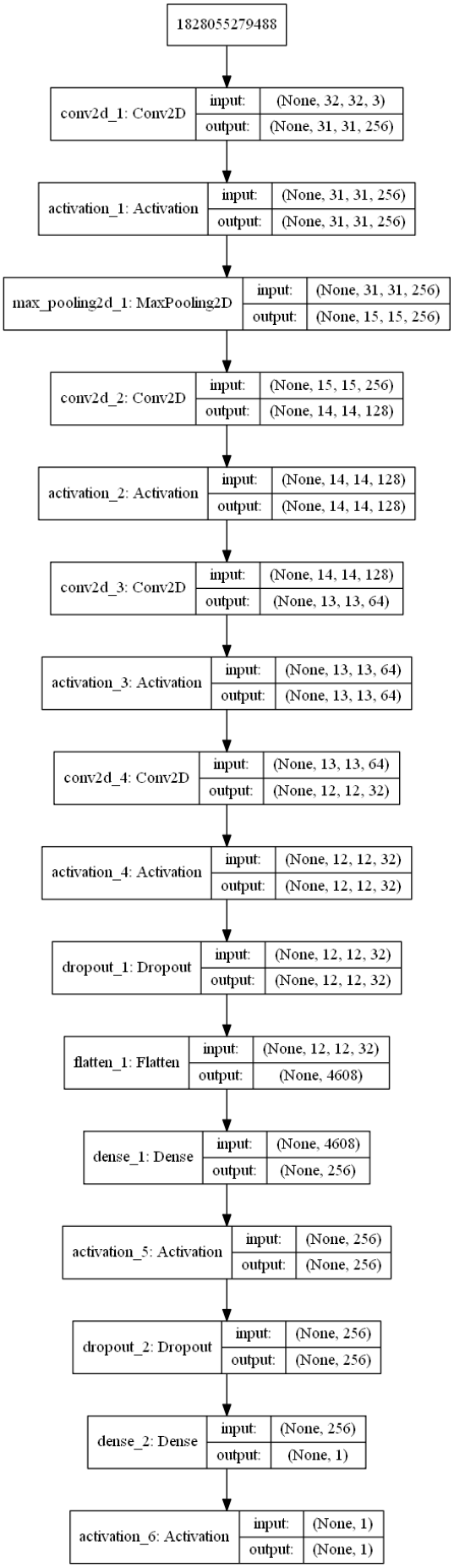
Transfer Learning model architecture

Benchmark

A simple CNN model with following architecture will used: 4 Convolutional, 1 MaxPooling and 2 Fully Connected layers. This model will be used as a bench mark against which our final solution will be measured.

The benchmark model achieved 95% accuracy on test set. We will try to outperform it.

Note: dozens of submissions for this challenge on Kaggle achieved 100% accuracy. Our ultimate goal is to achieve this result as well. Most of the submissions with 100% accuracy are using Fast.ai deep learning library based on PyTorch.



Benchmark model architecture

III. Methodology

Data Preprocessing

The dataset is imbalanced: 13136 pictures contain a columnar cactus (Neobuxbaumia tetetzo), 4364 pictures contain other vegetation. To train the model we're going to upsample the images without cactuses by augmenting and saving it. But because the image augmentation is in essence copying existing images there is a concern that we will have a data leakage from the testing set if we use augmented data for testing. For this reason, we are going to set aside 1313 data samples for testing from each class before starting image augmentation to make sure that our model is tested on data it has never seen before. This is 10% of the data. In the Proposal I've written that the training/validation/testing split is going to be 60:20:20. But if we follow this split it will leave us with only 1736 unaugmented images without cactuses. It will force us to upsample this category by 6 times which makes the possibility of overfitting bigger. This led to a new training/validation/testing split of 80:10:10.

Now we have 11823 images with cactuses and 3051 images without cactuses left to work with. Images without cactuses were upsampled 3.9 times using Keras ImageDataGenerator class. Following augmentation was applied:

- Rotation angle in degrees: 60
- Shift in the x direction: 0.3
- Shift in the y direction: 0.3
- Zoom in the x direction: 0.8
- Zoom in the y direction: 0.8
- Horizontal flip

Sample augmentation visualization:

Augmentation was applied to images without cactuses. Now we have 11823 samples for each class.

Final split is 21280 training images, 2366 validation images and 2626 testing images belonging to 2 classes.

Images are being fed into CNN as NumPy arrays. Each pixel in each channel is represented by a number ranging between 0-255. This might give the model wrong signal about importance of particular pixel. To normalize the data, pixel values will be divided by 255 to have a range between 0-1.

Implementation

As stated above Transfer Learning CNN model was used as solution for this problem. VGG16 Convolutional layers with ImageNet weights but without top Dense layers was used. We build Our 2 Dense layers on top of it. Model summary with total of 15,108,929 trainable parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
image_input (InputLayer)     (None, 32, 32, 3)         0
_________________________________________________________________
vgg16 (Model)                multiple                  14714688
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
fc1 (Dense)                  (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
fc2 (Dense)                  (None, 256)               131328
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
predictions (Dense)          (None, 1)                 257
=================================================================
Total params: 15,108,929
Trainable params: 15,108,929
Non-trainable params: 0
```

ModelCheckpoint function from Keras callbacks class was used to save the best performing weights of the model. This function checks the validation loss and saves the weights with smallest loss.

The training was set up for 50 epochs. The loss stopped improving at epoch no.15. When the model with saved network weights was tested against testing set it showed 100% accuracy.

Refinement

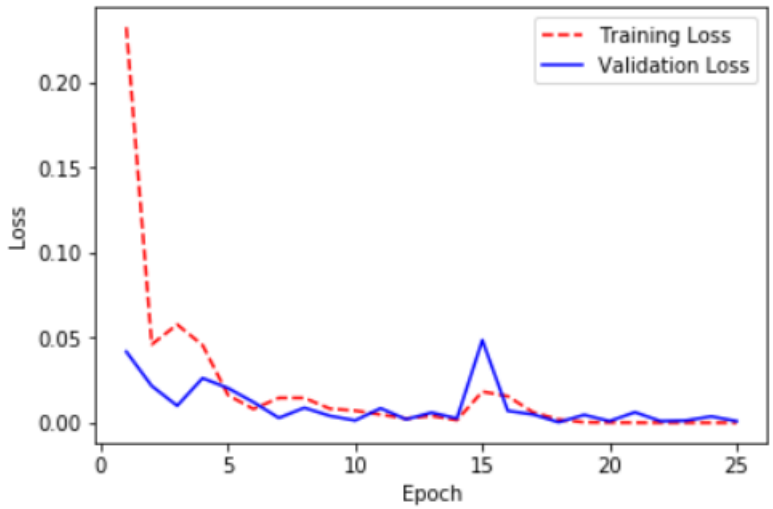
First time training the model I didn't separate the test set at the beginning resulting in test set containing augmented data. Even though dozens of Kaggle competition submissions achieved 100% accuracy, I still looked through my code to see if I can spot any red flags. This is when I noticed that I'm using augmented data as testing set and potentially might have data leakage.

After that code was rerun and testing set was put aside before applying augmentation. This time I set number of epochs to 25. The loss stopped improving at epoch no.18 with validation loss of 0.00038. Testing showed 100% accuracy on the data the model never seen before.

IV. Results

Model Evaluation and Validation

One of the main concerns during model training was extensive use of image augmentation for one of two classes. We had to upsample it by 390% to match the other class dataset. To investigate whether the overfitting occurred during the training period training and validation loss was plotted on 25 epoch timeline:



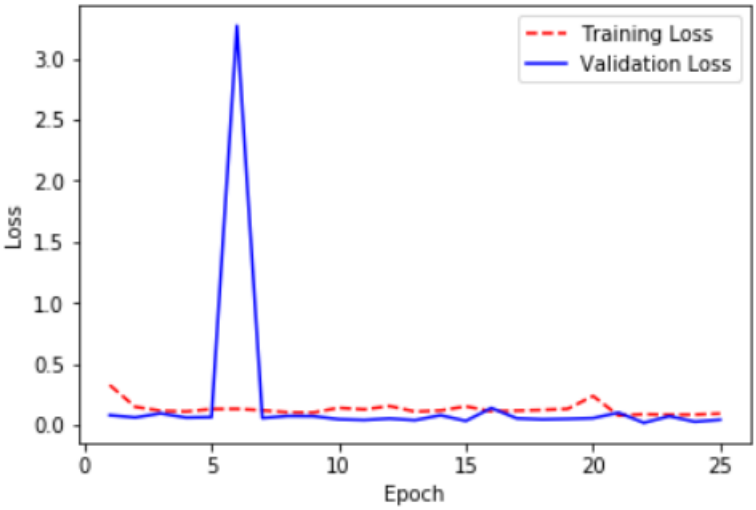
On the graph training and validation losses began at high point but far apart. Then both losses dropped as training progressed. After epoch no.10 the lines converged. There was and anomaly spike for both lines at epoch no.15, but after that convergence continued. This is a textbook example of a model with good fit.

Final test of the model was conducted using a test set consisting of 1313 images for each class never seen by the model. This a good number of samples to perform final test of the model to see if it generalizes well across broad spectrum of data. The results of such testing can be confidently relied on. The final test showed a 100% accuracy of the model reaching state of the art performance and standing at same height as the best submissions on Kaggle challenge.

If the model would've shown any lesser results on the testing set more sophisticated model optimizers could've been tested out. But we were able to reach state of the art with Stochastic gradient descent with default parameters.

Justification

The Benchmark model achieved 95% accuracy. Which is a solid result for image classification model. Benchmark model training and validation loss was plotted on 25 epoch timeline can be found below. There is anomaly of validation loss spike on epoch no.6, other than that it is obvious that the benchmark model has a good fit.

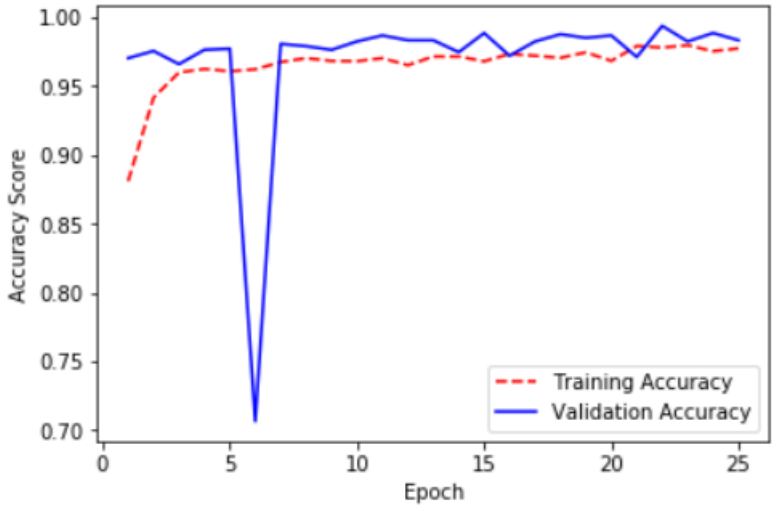


But final Transfer Learning model managed to achieve state of the art which in this case is 100%. This I s a significant improvement and the model can be relied upon extensively as it was tested on large amount of unseen data.

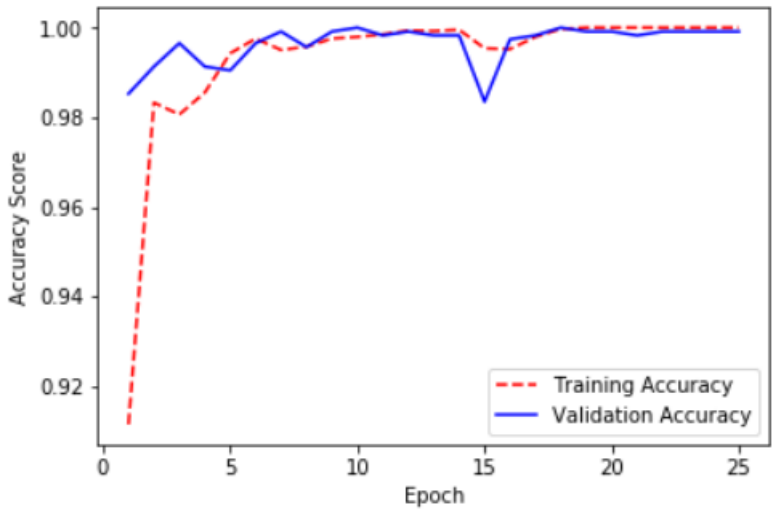
V. Conclusion

Free-Form Visualization

Here we can see training and validation accuracy of Benchmark and Transfer Learning models plotted over 25 epochs timeline:



Benchmark model accuracy plot



Transfer Learning model accuracy plot

These plots are basically loss plots upside down. These plots tell us the same story as loss plots and reassure us that both models have a good fit.

Reflection

The following steps were taken:

- 10% of data was set aside as testing set

- To combat imbalanced data the lacking image class (pictures not containing the columnar cactus) was upsampled by adding augmented pictures (flipping images horizontally, rotating, shifting and zooming on the images)
- Balanced data was further split into training and validation sets by 90% and 10% respectively
- Transfer Learning CNN was modeled using VGG16 pre-trained convolutional layers and fully connected layers were manually added and trained
- The network was fed with rescaled images. Pixel values rescaled to a range between 0 and 1 by dividing each pixel by 255
- As the problem at hand is binary image classification Binary Cross-Entropy was used as Loss Function
- For the starters normal SGD optimizer with default parameters was used with possibility of further fine-tuning in head
- ModelCheckpoint function from Keras callbacks class was used to save the best performing weights of the model
- Model achieved 100% validation accuracy at epoch no.18 out of 25
- Model showed 100% on unseen broad testing samples

Final model showed state of the art results and can be used for general data.

The model showed great results being trained on augmented data with 390% of upsampling. Which once again proved image augmentation as a good practice for data upsampling.

Improvement

This model reached 100% accuracy using Keras with TensorFlow backend. But most of the leading submissions on the Kaggle leaderboard for this challenge are done using Fast.ai with PyTorch backend. Some of the submissions displayed their code and seem to have much less lines of code than solution in this project. I would like to Fast.ai library as well and learn how it was achieved.