Capstone Project

Aditya Vikram Bhattacharya

Machine Learning Engineer Nanodegree

May16th, 2019

Aerial Cactus Identification

I. Definition

Project Overview

The project is a step to take an initiative towards conserving the environment using AI. Similar to my interest I found a competition on Kaggle with relevant data set. The current project would require image processing for its execution. In this work, we present a Deep Learning approach for columnar cactus recognition, specifically, the Neobuxbaumia tetetzo species specific to a habitat found at Mexico. The current approach is to find Columnar Cactus.

Problem Statement

The initiatives taken by the Researchers in Mexico have created the VIGIA project, which aims to build a system for autonomous surveillance of protected areas. The aim of the algorithm is to find a specific type of cactus in aerial imagery and carry out its surveillance.

1. Download the given data set
2. Train a classifier that can determine if an image contains cactus or not.
3. Use a different pre trained model to make predictions.

Accuracy for the test set Metrics

The evaluation metric for both the models to quantify the performance can be determined using ROC. ROC curve (Receiver Operating Characteristic Curve) is a graph showing the performance of a classification model at all classification thresholds by determining the area under the curve. This curve plots two parameters:

True Positive Rate False Positive Rate True Positive Rate ***(TPR)*** is a synonym for recall and is therefore defined as follows:
TPR = TP / (TP + FN)

False Positive Rate (FPR) is defined as follows:

FPR = FP / (FP + FN)

- ***TP*** *: True Positive*
- ***TN*** *: True Negative*
- ***FN*** *: False Negative*
- ***FP*** *: False Positive*

As per the current scope of the project the aim was detecting the Columnar Cactus in the clips of images provided by Kaggle using deep neural network. The method of graphically comparing the accuracy and loss of model are easier. I have used VGG16 and ResNet50 pre trained networks for the project.
The training accuracy and the validation accuracy of the two models were compared. Similarly, the validation losses and the training losses were compared for both the models. This is a way off determining which models achieved a higher accuracy and gives a lower regression by considering the region under the curve.

II. Analysis

Data Exploration

The source for dataset was from Kaggle. This 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 le name of an image corresponds to its id. The problem uses detection of cactus (Neobuxbaumia tetetzo) in frames. the Datasets and Inputs
frames will be used as the input for the convolutional neural network. The number of images in the training set is 17500. The number of images in the test is 4000.

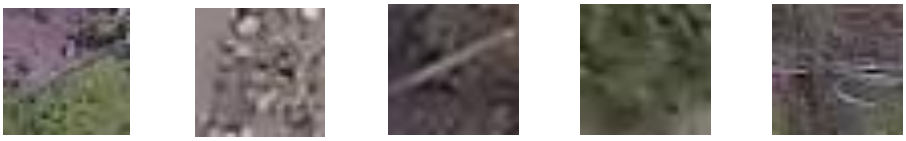
The train.csv data contains the following fields:

- 'Id': This holds the names of the all the files in the training set.
- 'Has_cactus': This field holds the classification values. In train.csv all the fields hold the default value of 1. Later the predictions are assigned to this field.

The fields mentioned above actually hold the value for the prediction.

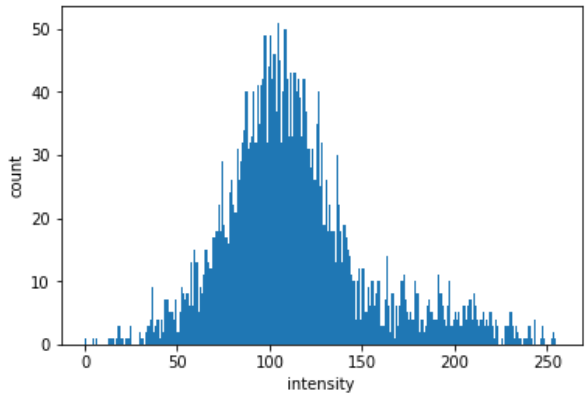
Sample Training Data

Sample Testing Data



Visualisation

The plot given below describes the intensity of white in the image. The basic idea is that most of the images depicting cactus appear white. The higher the value of all the three pixels in rgb (red blue green) the more the pixel appears white. This technique converts the 2D image array to 1D array and the intensity of the pixels are plotted against the length of the array So we can find that the distribution shows the intensity of some similar pixels.



Algorithms and Techniques

The approach used for the project is using Convolutional Neural Network for the processing of image data. This algorithm is most common and also the game changer for image processing tasks, including classification. It requires a large amount of training data.

The Aerial Cactus Identification competition by Kaggle catered for the requirement of the data for the project. The algorithm provides a very good response by eliminating the false positive input from the training set by implementing a threshold. (This helps in increasing the number of false negatives.)

Transfer Learning: Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.

The techniques used for this task contain a few parameters for fine tuning of the model. The following parameters can be tuned to optimize the classifier:

- Classification threshold.
- Training parameters
    1. Training length (number of epochs)
    2. Batch size (how many images to look at once during a single training step)

3. Solver type (what algorithm to use for learning)
4. Learning rate (how fast to learn; this can be dynamic)
5. Weight decay (prevents the model being dominated by a few "neurons")
6. Momentum (takes the previous learning step into account when calculating the next one)

- Convolutional Neural network architecture
    1. Number of layers
    2. Layer types (activation, convolutional, fully-connected, pooling)
    3. Layer parameters
- Pre-processing parameters (see the Data Pre-processing section)

During training, both the training and the validation, sets are loaded into the RAM. After that, random batches are selected to load into the CPU memory for processing. The training is done using the Mini-batch gradient descent algorithm. The resolution required for the ResNet50 to operate is 224 x 224 x 3 but the dimension of the image data was 32 x 32 so input shape to feed into the model was 32 x 32 x 3.

Convolutional Neural networks and Maxpooling

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet **architecture**.

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three colour channels R,G,B.(red green blue).
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the max(0,x) max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Benchmark

To create an initial benchmark for the classifier, VGG16 (a pre rained deep neural network known as Visual Geometry Group 16) was used. The "standard" VGG16 architecture with

changed input shape (32x32x3) achieved the best accuracy, around 0.9593 on local system but the accuracy of this benchmark model was around 0.9707. The mentioned can be achieved with a reduced batch size of batch size of 32. The accuracy of 0.9593 was achieved in the local system and had a batch size of 100 was computed on the local system. The number of epochs for the both the cases were 100. The project's aim was to achieve a higher accuracy with same number of epochs and use a different model which can give a promising result.

The data which was obtained from Kaggle catered for training the model. The data was segregated into training and validation set which were used to determine the reduction in the training and validation loss and the increment in the accuracy respectively. The entire behaviour of the model was represented graphically. The threshold decided for the classification was above 0.75. The following felt to be a suitable threshold as anything below this would have a lower probability of the classification being true.

III. Methodology

Data Pre-processing

Pre-processing helps reducing the abnormalities from the data like removing redundant data, resizing and reshaping the input for better feed to the neural network for the uniform training of the input model. The pre-processing techniques used for the project was converting image file to floating point tensors before being fed to our network. The steps for getting it into our network are roughly:

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

The project used ImageDataGenerator method in Keras for all the pre-processing. This step segregates the input data into training and cross validation sets. The images are shuffled divided into smaller batches for further processing.

Implementation

The approach for the project was to implement a deep neural network for image processing. The Bench mark model used VGG16. The summary for the architecture for the VGG16 is mentioned below.

i. The Benchmark model architecture VGG16(Visual Geometric Group):

The VGG model also known as Visual Geometric Group. The

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

1. After loading the pre trained model the following layers were added:
To obtain the desired output or the target shape.

- Flatten: Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.
- Dense: This is a fully connected layer (256 nodes).
- Activation layer: This layer help in the prediction based on the inputs.
- Drop out layer: This terminates the number of data points which may cause overfitting.

- Dense layer: This layer has a single node to achieve the target output (1 node).
- Activation layer: The final activation layer helps in classification of the data.

The steps involved in running the benchmark model are:

2. After the pre trained model was loaded the loss the model was compiled. The parameter sele cted were:

```
model.compile(loss='binary_crossentropy',optimizer=Adam(lr=1e-5),
        metrics=['accuracy'])
```

- Loss: Binary_crossentropy (this is determines the performance loss of the classifi cation model whose output is a probability value between 0 and 1. Cross-entropy loss i ncreases as the predicted probability diverges from the actual label)
- Optimiser: Adam stands for Adaptive moment estimation(lr or learning rate was assign ed as 10^{-5} this gives the rate at which the classifier should about the problem).
- Metric: Accuracy (used to determine the number of correct predictions compared to tot al number of predictions made.)

3. The next step is matching the image files name to the file names present in the csv (coma separated values) file. The above mentioned are then appended to one variable as an i mage file and another variable contains the id or the name of the respective image. Then as mentioned in the pre-processing section the images are converted to floating value and scale d down to values between 1 and 0.

```
X_tr = [ ]
Y_tr = [ ]
imges = train_df['id'].values
for img_id in tqdm_notebook(imges):
   X_tr.append(cv2.imread(train_dir + img_id))
   Y_tr.append(train_df[train_df['id'] == img_id]['has_cactus'].values[0])
X_tr = np.asarray(X_tr)
X_tr = X_tr.astype('float32')
X_tr /= 255
Y_tr = np.asarray(Y_tr)
```

4. After the data is converted and the model is compiled, the model is trained. The model para meter associated are:

- X_tr: This term is used to refer the value of the images converted to float value which was discussed in the previous point.
- Y_tr: Contains the list of id’s for the images.
- Batch_size: The number of inputs that have to be processed in 1 cycle of training a batch.

- Epochs: The total number of times a batch is trained
- Validation_split: The fraction or the part of data used to validate the training set is a validation split.
- Shuffle: The input is randomly shuffled.

```
# Train model
history = model.fit(X_tr, Y_tr,
          batch_size=batch_size,
          epochs=nb_epoch,
          validation_split=0.1,
          shuffle=True,
```

5. After the model is trained the next step is making predictions.
The following code block reads the test images and assigns them to a variable. The appe nded values of the image and the id are converted to float array and then normalised to r ange between 0 and 1.

```
%%time
X_tst = []
Test_imgs = []
for img_id in tqdm_notebook(os.listdir(test)):
   X_tst.append(cv2.imread(test + img_id))
   Test_imgs.append(img_id)
X_tst = np.asarray(X_tst)
X_tst = X_tst.astype('float32')
X_tst /= 255
```

6. The predictions are made based on the model trained. The predictions are appended as dataf rame to the respective Id's with assigned score. Based upon the score of data the predcitions are made. If the score is greater than 0.75 the id is assigned a value of 1 else the Id is assign ed 0. After the process is completed a submission file is generated with prediction with thei r corresponding file names.

```
test_predictions = model.predict(X_tst)

sub_df = pd.DataFrame(test_predictions, columns=['has_cactus'])
sub_df['has_cactus'] = sub_df['has_cactus'].apply(lambda x: 1 if x > 0.75 else 0)
```

```
sub_df['id'] = ''
cols = sub_df.columns.tolist()
cols = cols[-1:] + cols[:-1]
sub_df=sub_df[cols]

sub_df.to_csv('submission.csv',index=False)
```

ii. The Model implemented RESNET50 (Residual Network):

The ResNet also known as Residual Network. The given below is the summary of the architect ure. The steps employed on running the model are mentioned below. The following are:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 32, 32, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 38, 38, 3)	0	input_1[0][0]
conv1 (Conv2D)	(None, 16, 16, 64)	9472	conv1_pad[0][0]
bn_conv1 (BatchNormalization)	(None, 16, 16, 64)	256	conv1[0][0]
activation_1 (Activation)	(None, 16, 16, 64)	0	bn_conv1[0][0]
pool1_pad (ZeroPadding2D)	(None, 18, 18, 64)	0	activation_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0	pool1_pad[0][0]
res2a_branch2a (Conv2D)	(None, 8, 8, 64)	4160	max_pooling2d_1[0][0]
bn2a_branch2a (BatchNormalizati	(None, 8, 8, 64)	256	res2a_branch2a[0][0]
activation_2 (Activation)	(None, 8, 8, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 8, 8, 64)	36928	activation_2[0][0]
bn2a_branch2b (BatchNormalizati	(None, 8, 8, 64)	256	res2a_branch2b[0][0]
activation_3 (Activation)	(None, 8, 8, 64)	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, 8, 8, 256)	16640	activation_3[0][0]
res2a_branch1 (Conv2D)	(None, 8, 8, 256)	16640	max_pooling2d_1[0][0]
bn2a_branch2c (BatchNormalizati	(None, 8, 8, 256)	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, 8, 8, 256)	1024	res2a_branch1[0][0]
add_1 (Add)	(None, 8, 8, 256)	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
activation_4 (Activation)	(None, 8, 8, 256)	0	add_1[0][0]
res2b_branch2a (Conv2D)	(None, 8, 8, 64)	16448	activation_4[0][0]
bn2b_branch2a (BatchNormalizati	(None, 8, 8, 64)	256	res2b_branch2a[0][0]
activation_5 (Activation)	(None, 8, 8, 64)	0	bn2b_branch2a[0][0]
res2b_branch2b (Conv2D)	(None, 8, 8, 64)	36928	activation_5[0][0]
bn2b_branch2b (BatchNormalizati	(None, 8, 8, 64)	256	res2b_branch2b[0][0]
activation_6 (Activation)	(None, 8, 8, 64)	0	bn2b_branch2b[0][0]
res2b_branch2c (Conv2D)	(None, 8, 8, 256)	16640	activation_6[0][0]
bn2b_branch2c (BatchNormalizati	(None, 8, 8, 256)	1024	res2b_branch2c[0][0]
add_2 (Add)	(None, 8, 8, 256)	0	bn2b_branch2c[0][0] activation_4[0][0]

```
activation_7 (Activation)        (None, 8, 8, 256)     0           add_2[0][0]
res2c_branch2a (Conv2D)          (None, 8, 8, 64)      16448       activation_7[0][0]
bn2c_branch2a (BatchNormalizati (None, 8, 8, 64)      256         res2c_branch2a[0][0]
activation_8 (Activation)        (None, 8, 8, 64)      0           bn2c_branch2a[0][0]
res2c_branch2b (Conv2D)          (None, 8, 8, 64)      36928       activation_8[0][0]
bn2c_branch2b (BatchNormalizati (None, 8, 8, 64)      256         res2c_branch2b[0][0]
activation_9 (Activation)        (None, 8, 8, 64)      0           bn2c_branch2b[0][0]
res2c_branch2c (Conv2D)          (None, 8, 8, 256)     16640       activation_9[0][0]
bn2c_branch2c (BatchNormalizati (None, 8, 8, 256)     1024        res2c_branch2c[0][0]
add_3 (Add)                      (None, 8, 8, 256)     0           bn2c_branch2c[0][0]
                                                                   activation_7[0][0]
```

Add Layer from 3-15 Activation layers from …….10-45

```
activation_46 (Activation)       (None, 1, 1, 2048)    0           add_15[0][0]
res5c_branch2a (Conv2D)          (None, 1, 1, 512)     1049088     activation_46[0][0]
bn5c_branch2a (BatchNormalizati (None, 1, 1, 512)     2048        res5c_branch2a[0][0]
activation_47 (Activation)       (None, 1, 1, 512)     0           bn5c_branch2a[0][0]
res5c_branch2b (Conv2D)          (None, 1, 1, 512)     2359808     activation_47[0][0]
bn5c_branch2b (BatchNormalizati (None, 1, 1, 512)     2048        res5c_branch2b[0][0]
activation_48 (Activation)       (None, 1, 1, 512)     0           bn5c_branch2b[0][0]
res5c_branch2c (Conv2D)          (None, 1, 1, 2048)    1050624     activation_48[0][0]
bn5c_branch2c (BatchNormalizati (None, 1, 1, 2048)    8192        res5c_branch2c[0][0]
add_16 (Add)                     (None, 1, 1, 2048)    0           bn5c_branch2c[0][0]
                                                                   activation_46[0][0]
activation_49 (Activation)       (None, 1, 1, 2048)    0           add_16[0][0]
flatten_1 (Flatten)              (None, 2048)          0           activation_49[0][0]
dense_1 (Dense)                  (None, 512)           1049088     flatten_1[0][0]
dense_2 (Dense)                  (None, 1)             513         dense_1[0][0]
==================================================================================================
Total params: 24,637,313
Trainable params: 24,584,193
Non-trainable params: 53,120
```

The functions involved in the step by step execution of the solution are:

1. The pre-processing for the solution was done using ImageDataGenerator a library in Keras which is useful in generating train of images. The parameters associated with the ImageDa taGenerator are:
    - Rescale:Converting the float values of the array into the range of 0 to 1.
    - Validation_split: A part of the training data set is converted to validation set.
    - Rotation_range: Degree range of random rotations
    - Shear_range: Shear intensity. Shears angle in counter clockwise direction in degr ee.
    - Horizontal_flip: Randomly flips data horizontally. Takes a Boolean input.
    - Verticle_flip: Randomly flips data vertically. Takes a Boolean input.
    - Zoom_range: Range of random zooms.

Image Data Generator

```
train_datagen=ImageDataGenerator(rescale=1./255,
                                 validation_split=0.1,
                                 rotation_range=30,
                                 shear_range=0.2,
                                 horizontal_flip=True,
                                 vertical_flip=True,
                                 zoom_range=0.2)
```

The following code depicts the function generating the train split for the model training also the process of creating the cross-validation split is similar. This step uses the function flow_from_d ataframe foe performing the operation. Batch size for training has been taken as 10 for faster co mputation. The parameters involved are:

- Dataframe: input data frame
- X_col and y_col: Columns of the data frame
- Batch_size: Decides the number of input at ones.
- Shuffle: Randomly arrange the input.
- Class_mode: Type of class binary, string, etc.
- Target_size: Expected output dimensions.
- Subset: Name of the subset whether its training subset or its validation subset.

Train split

```
train_generator=train_datagen.flow_from_dataframe(
   dataframe=train_df,
   directory="./Cactus/train/train/",
   x_col="id",
   y_col="has_cactus",
   batch_size=10,
   shuffle=True,
   class_mode="binary",
   target_size=(32,32),
   subset='training')
```

Validation Split

```
validation_generator=train_datagen.flow_from_dataframe(
   dataframe=train_df,
   directory="./Cactus/train/train/",
   x_col="id",
```

```
y_col="has_cactus",
batch_size=10,
shuffle=True,
class_mode="binary",
target_size=(32,32),
subset='validation')
```

2. After loading the pre trained network a few more layers were added to the network to get a desired output. For adding the new layers, a function was created. The name of the function is “add_new_layer”. The input parameter for the function was the base model or the pre-trained residual model. This function added a few more layers to the model. The layers are:
    - Flatten
    - Dense Layer with the activation function Relu (Rectifier Linear unit).
    - The next dense layer is used for predictions using sigmoid and the regularisation parameter is 0.05.
    - This function returns the base model and the output layers that we use combined.

```
def add_new_layer(base_model):
    x=base_model.output
    x=Flatten()(x)
    x=Dense(512,activation="relu")(x)
    predictions = Dense(1, activation='sigmoid',activity_regularizer=regularizers.l1(0.05))(x)
    model = Model(input=base_model.input, output=predictions)
    return model
```

3. The model defined in the above section is compiled using the function tranfer_learn. This function uses base model and model as their input parameters. Post that the layers of the base_model (ResNet50) layers are compiled using this function. To know further about the parameters involved refer i.2.

```
def transfer_learn(model, base_model):
    for layer in base_model.layers:
        layer.trainable = True
    model.compile(optimizer=Adam(lr=1e-5), loss='binary_crossentropy',
                  metrics=['accuracy'])
```

4. After this the model is trained using the code mentioned below. The batch size in this case is 10 and the number of epochs were taken as 100.
    - Generator: The parameter was assigned the array value generated by the image data generator for the training set.
    - Validation_data: The parameter was assigned the array value generated by the image data generator for the validation set.
    - Validation_steps: The value for number of steps based on the normalised input shape.
    - Steps_per _epoch: This is same as the batch size which we mentioned earlier.
    - Epochs: The total number of cycles the entire implementation works.
    - Verbose: The type of training display.

```
%%time
history_2=model_res.fit_generator(generator=train_generator,
                    validation_data=validation_generator,
                    validation_steps=int(train_df.shape[0]/32),
                    steps_per_epoch=int(train_df.shape[0]/32),
                    epochs=100,
                    verbose=2)
```

5. After this step the prediction algorithm was applied, the steps to be followed are mentioned in the previous section (i.6.).

Refinement

The process of developing this project was enriching. The start of this project was with the VGG16 model. Which gave a very promising result. The first output that I obtained was very promising. The initial batch size for the model was quite 32 with 1000 epochs to get an accuracy of 0.99 approximately. But the number was quite high and it would have taken a lot of time for the model to train so the number of epochs were dropped down to 100 and this was giving an accuracy of about 0.9672. The next approach was reducing the batch size to 10 the result for this was that the accuracy dropped from 0.9672 to 0.9529. So further the assumption was made that on reducing the batch size the accuracy dropped so may be increasing the batch size might help. The next batch size was 100. But the accuracy still dropped but by a lesser number. It was 0.9593 this gave a clarity that the batch size has to be an optimum number. The next plan was to use a value closer to the batch size taken earlier it was taken 40 and the result improved. But wasn't giving an accuracy as per the original batch size. The optimum number decided for the bench mark was 32.

The next model that used was ResNet50.The first problem that we encountered was for the shape of the input layer. As ResNet takes the input shape for training as 224 x 224 x 3 although

it was rectified easily by making the top layer inclusion of the layer as false. When the output layers used were same as the previous model. The accuracy was below 0.92. The output layers needed tweaks like removing the drop out layers. Increase the number of nodes in the layer. The number of nodes were increased from 256 to 512. The data also had to be regularised. After the change there was sudden increase in the accuracy of the training data with better results. The accuracy started touching 0.9653 also reached an accuracy of 0.9811 at one instance. The observation was made that the current output layer was working fine but could further be reached to an accuracy of 0.99. Then the next approach was tweaking the batch size for training. The starting batch size was 32. The next batch size was 64 but the accuracy suffered further. Then the next batch size was 16 and the accuracy sore close to 0.99 it was 0.9853. Then the batch size was further reduced to 10 which gave us a recurring frequency of 0.99 for last 20 iterations and stopped at 0.9913.

IV. Results

Model Evaluation and Validation

The evaluation of model at the time is done using Validation set. It is a part of the training set which is separated to validate or act as a cross checking method for the model. The final architecture and hyperparameters were chosen because they performed the best among the tried combinations. The final description for the model is given below.

- The total parameters in the model: 24,637,313
- Trainable parameters: 24,584,193
- Non-trainable parameters: 53,120
- The layers added for the output layer, summary is given below:

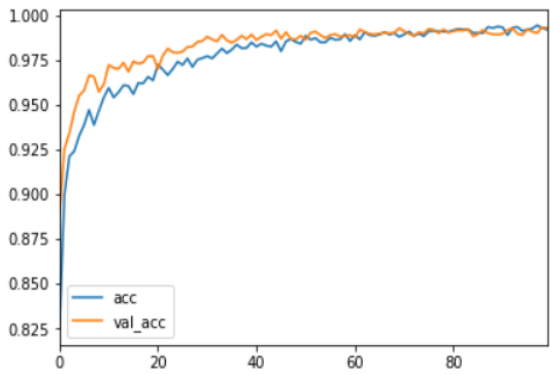
```
activation_49 (Activation)      (None, 1, 1, 2048)   0           add_16[0][0]

flatten_1 (Flatten)             (None, 2048)         0           activation_49[0][0]
_____________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          1049088     flatten_1[0][0]
_____________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            513         dense_1[0][0]
=====================================================================================
```

- The compilation of the model used the ADAM optimiser (Adaptive Model Estimator).
- It used binary cross entropy.
- The metric used is accuracy. The next section has the Visualisations created by using the same metrics.
- The training parameters like batch size and epoch can affect the accuracy and loss. The detailed explanation can be found in the refinement section.
- Total number of images used are 17500 for training and validation combined. The number of test sets used for prediction are 4000.
- The validation split for the final model is 1750 and the training split used is 15750.
- The model can be trusted well because the validation and training accuracy show a good response.
- The predictions made on the test set are up to the mark as we can find the prediction in the conclusion section.

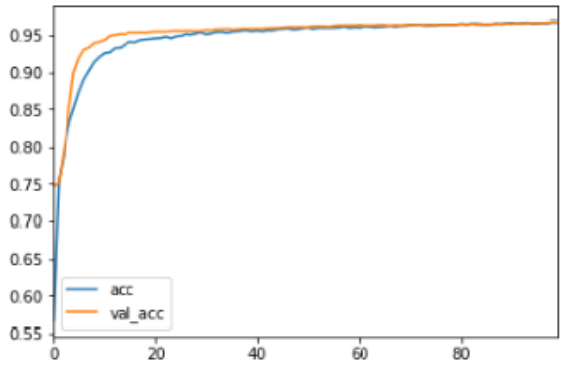
Justification

The graphical representation of the training and validation data are depicted below. The benchmark model and the Solution model will be compared on the basis of the accuracy and the loss metric calculated at the time of training the dataset.

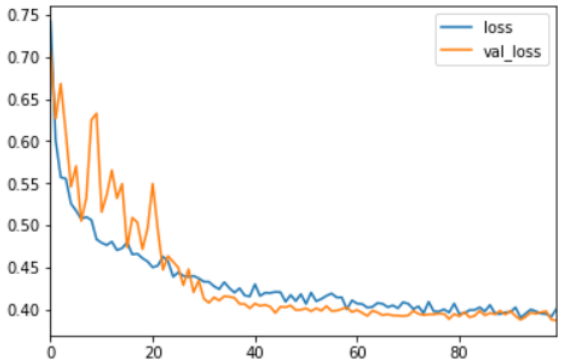
Resnet50 Training Accuracy (0.9919)
Validation Accuracy (0.9930)



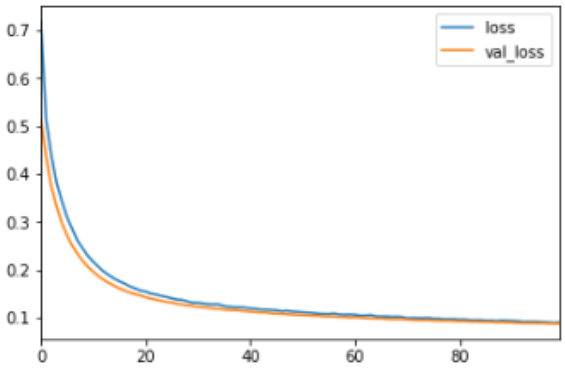
VGG16 Training Accuracy (0.9672)
Validation Accuracy (0.9669)



Resnet50 Training Loss (0.4008)
Validation Loss (0.3872)



VGG16 Training Loss (0.0904)
Validation Loss (0.0876)



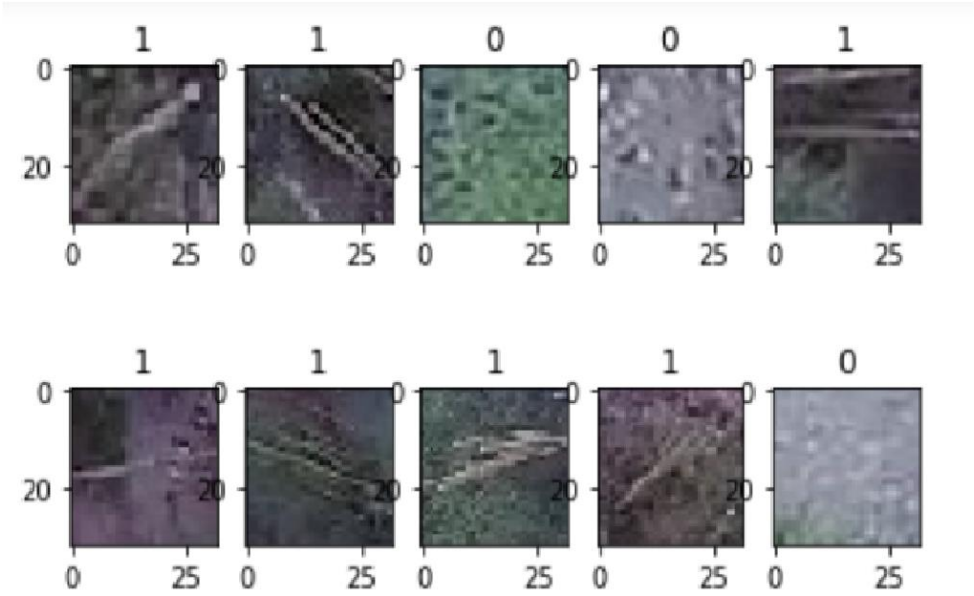
The final model is reasonable as it provides a higher accuracy compared to the bench mark model however the training and validation loses are comparatively higher as well. This might have incurred due to the selection of a smaller batch size as that of the benchmark. The curve for the VGG16 is similar due to that very reason of larger batch size. The increase in the size of the batch makes it more generalised but the training speed also suffers due to an increased batch size so keeping that in account the batch sized was decreased. Larger batch size also made the accuracy of the model reduced. Keeping such factors in mind the final model was decided for the project.

V. Conclusion

Free-Form Visualization

The aim of the competition was detecting images. This was achieved by generating a csv file which contains the classified image's id along with information that the image frame has a cactus or not. The model has been tested over 4000 image files provided by Kaggle. A few test result samples for the following model validation are depicted below:

The prediction made by the Model



The images depicted above are actually the result for the test data. The images have been classified with the aim of detecting cactus in the given test set. The images with title as 1 are the one which contain cactus and the one with the 0 doesn't have cactus in the image. The predictions generated gives a clarity that the pre diction made by the model are just fine and the prediction are quite accurate.

Reflection

The entire project and the process involved can be summarised in the following given step:

1. The initial problem statement and the data set was found in the competition Aerial Cactus Identification by Kaggle.
2. The size of the images was almost equal (138kB approximately). The image data was converted to a float array and converted in the range of 0 to 1 before training them.
3. The benchmark was created for the classifier.
4. The classifier was trained using the training data. This process was repeated several times in order to reach an ideal parameter configuration in order to achieve a satisfactory solution.
5. The compilation was done in the local system.
6. The trained model was then used to classify images and make predictions.

The most difficult part of the project was step 4 and 5. The layers of the neural network to be used and the parameters to be decided required several attempts. The process of training the model required a lot of time. The output layers of the bench mark helped for a few layers although the final architecture selected was different from the benchmark model. The predictions made by the model are up to the mark.

Improvement

In the current project the training and validation accuracy increased compared to the benchmark model but the losses increased. This means that the training and validation losses can be improved and we can get a smaller loss while maintaining the same accuracy or may be a higher accuracy.

The algorithm which I want to learn about is Fast.ai Model. We used the approach of Transfer Learning in this project. Fast.ai is synonymous to transfer learning and achieving great results in a short amount of time. There are better results using Fast.ai to perform image classification. As the number of neural network connections are denser
.
Considering the fact that if the current solution is used as a benchmark the accuracy can be go a notch higher than the current result.