DEFINITION

Project Overview

In today's world, technology solves problems much quicker than we think. We do not only use the technology in our workspace or home, we have started using it in our backyard. I used one of the phone applications to learn the name of the plant that grows in my backyard.

I was inspired by the application I used, and I thought I could develop something similar to it to help people in agriculture. In this project, I developed a deep learning model that would distinguish between healthy bean leaf and disease bean leaf. If the leaf is diagnosed as diseased leaf, the program will tell us what kind of disease the plant has.

My model will be using AIRlab Bean dataset. This data set has been annotated by experts from the National Crops Resources Research Institute (NACRRI) in Uganda and collected by the Makerere AI research lab.

Problem Statement

According to my research, a farmer cannot identify the symptom at its earlier stage of development. When the diseased plant is identified, the disease covers all over the plant and it requires time, effort, and chemical control. Even controlling the disease at certain point may affect the total harvesting. However, if we are able to diagnose the disease at its early stage, we can prevent these extra costs, time and effort.

My goal is to develop a phone app by using my machine learning model that will distinguish the healthy and two disease classes, including Angular Leaf Spot and Bean Rust diseases.

My model will have the followings:

1. Download and explore the data.

2. Process the data.

3. Develop a model.

4. Evaluate the result.

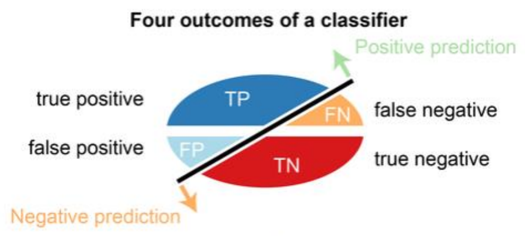
5. Deploy on Android through TensorFlow Lite.

Metrics

For the model evaluation, the following metrics will be used to measure how well the model is.

Validation Accuracy and Validation Loss: My goal is to minimize the loss and increase the accuracy on the validation data set. I will be using “Sparse Categorical Cross Entropy”, this is the best choice for the multiclass classification problems. Cross entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for all classes in the problem.

Confusion Matrix: Confusion matrix is a performance measurement for the classification problems. Confusion matrix indicates how many True positives, False positives and True negatives, False negatives are in the matrix. In this way, we can see where the misclassification occurs.



Classification of a test dataset produces four outcomes – true positive, false positive, true negative, and false negative.

ANALYSIS

Data Exploration

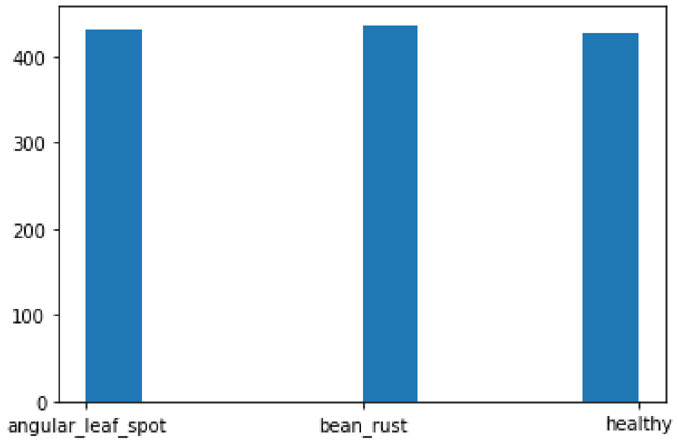
This dataset was collected in different regions by the Makerere AI research lab and annotated by experts from National Crops Resources Institute (NaCRRI) in Uganda. This dataset consists of three classes: the healthy leaves, and two disease classes (Angular Leaf Spot and Bean Rust diseases).

This is comparatively a small data set and it was already classified as Train, Validation and Test data set. Each set has 3 categories: Angular Leaf Spot, Bean Rust and Healthy. I reated a function that calls “extraxt_filenames” that list all the files under each directory, so I can obtain the name of the folders and count of the images.

```
#To extract file names from each directory
def extract_filenames(dir):
  path= os.path.join(root, dir)
  filenames=list()
  labels =list()
  for dir in os.listdir(path):
    for filename in os.listdir(os.path.join(path,dir)):
      filenames.append(os.path.join(path, dir, filename))
      labels.append(dir)
  return filenames, labels
```

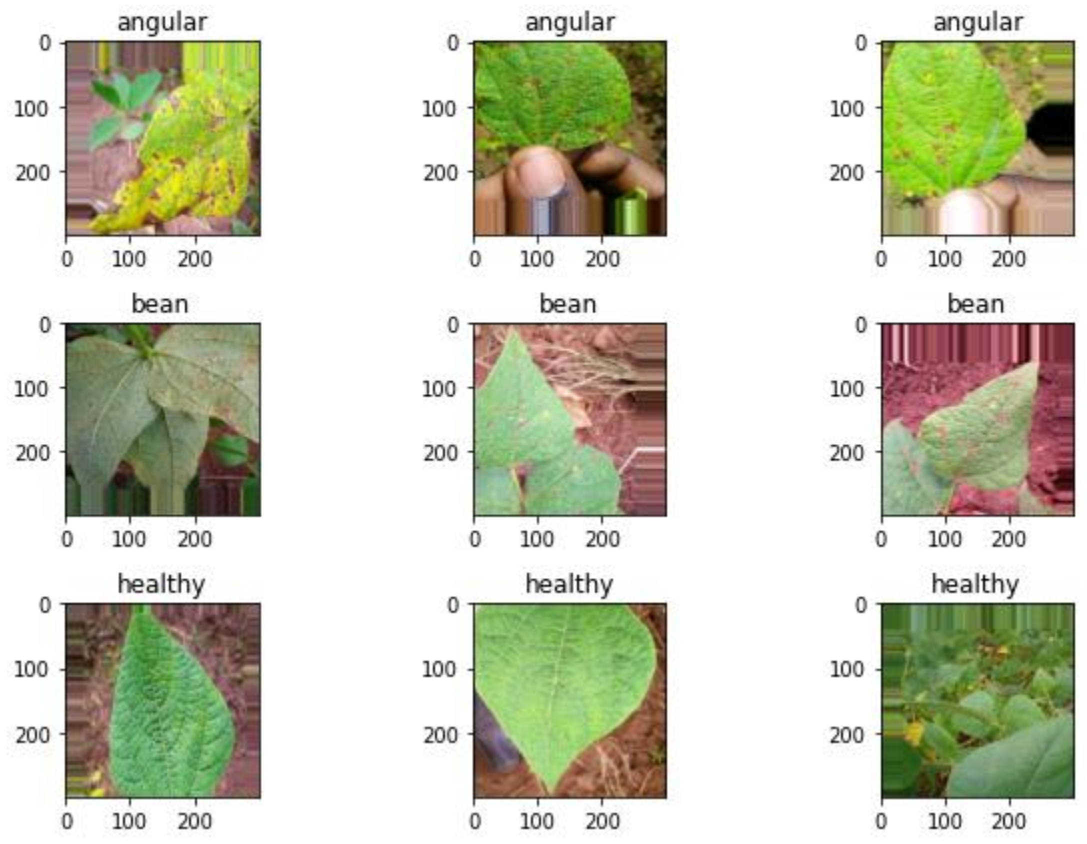
The total data as follows with its classes and counts:

```
Counter({'bean_rust': 436, 'angular_leaf_spot': 432, 'healthy': 428})
```



Exploratory Visualization

The following images are the sample data from the Bean data set. This pictures are taken with Smart phones from the field in Uganda.



Algorithms and Techniques

Distinguishing bean leaves is a difficult task for a classification problem. Convolutional Neural Network (CNN) approach could be the best option, because of its high accuracy. In this study, I used the TensorFlow framework to implement CNN architecture on my model and then I implemented Transfer Learning and I compared the results.

Data preparation (Data augmentation): The TensorFlow ImageDataGenerator method allows me to apply Data Augmentation methods. Data augmentation is a simple but powerful tool to prevent overfitting. Especially works well, if we have limited data. Data augmentation amends images on-the-fly while training. So, it is a kind of way to expand any limited data. Data augmentation also helps us to create more similar images on the test data. For example, If there is no bean leaf on the right side in the training data but there is on the test data, flipping the images might be helpful.

I applied the ImageDataGenerator as follows:

```
train_im_gen = ImageDataGenerator(rescale= 1./255,
                                   zoom_range= 0.2,
                                   rotation_range= 0.4,
                                   width_shift_range= 0.2,
                                   height_shift_range=0.2,
                                   horizontal_flip=True,
                                   validation_split= 0.0)
```

After I applied ImageDataGenerator, I loaded the data by using the TensorFlow flow_from_directory method. I set below parameters train data:

```
train_datagen= train_im_gen.flow_from_directory(train_dir,
                               target_size= (300,300),
                               batch_size=128,
                               shuffle=True,
                               class_mode='sparse' )
```

CNN structure as follows:

```
model= tf.keras.Sequential([

tf.keras.layers.Conv2D(32,(3,3),padding='same', input_shape=(300,300,3), activ
ation ='relu'),
tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(128,(3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(512, (3,3),padding='same', activation='relu'),
tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.4),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128,activation='relu'),
tf.keras.layers.Dense(3, activation='softmax')    ])
```

Model compile and Fit: I applied RMSProp and Adam optimizer which works well on image classification and sparse categorical cross entropy because I have 3 classes here and used accuracy for metrics.

Model training (Model Fit): I used 50 epochs to train the model on both my model and transfer learning.

Model Evaluation (Model Evaluate): I evaluated my model over test data. Because the test data gives me how well my model works on the real data.

Benchmark

The goal is to build a robust machine learning model that can distinguish between diseases in the Bean Plants and as I mentioned earlier, CNN works well on image classification problems. However, due to having limited data, I am going to implement Transfer Learning by using Inception V3. Inception V3 has been pre trained over 1.4 million images in 1000 different classes. Inception V3 has 42-layer deep learning network and has reached at least 78.8% accuracy on image classification tasks.

Methodology

Data Preprocessing

I downloaded the dataset from AI-Lab-Makarere GitHub website. The data have already been split into train, validation, and test groups. After I created path for them, I applied ImageDataGenerator to create more images and to prevent overfitting. I also chose the batch size 128 and shuffled them.

Model Implementation:

I implemented two different CNN models, first I developed my own model from scratch and then I implemented Transfer Learning with Inceptive V3 and I compared the results.

CNN Model:

In my first model, I applied Earlystopping function (patience=3) with Adam optimizer and my model achieved an accuracy score of 72.98% on validation set after 9 epochs.

In my second model, I applied Earlystopping function (patience=3) with RMSprop optimizer (learning rate=0.01), and my second model achieved an accuracy score of 68.48% on validation set after 12 epochs..

In my other models, I removed the Early stopping due its low accuracy rate, I also implemented Callback function by setting the accuracy rate to 84%. I also play with the number of convolutional nods in each layer but it doesn't change the accuracy rate dramatically.

Finally, I got the best prediction with Dropout Layer(0.4) which achieved an accuracy score of 85% on the validation data and 77% on the test data after 50 epochs.

```
Epoch 50/50
9/9 [==============================] - 30s 3s/step - loss: 0.4383 -
accuracy: 0.8020 - val_loss: 0.3311 - val_accuracy: 0.8571

1/1 [==============================] - 1s 1s/step - loss: 0.5143 -
accuracy: 0.7734
Accuracy on test dataset: 0.7734375
```

Transfer Learning:

In this model, I implemented the Transfer Learning by using the Inception_V3 model. I chose not to include the top Fully connected layer and directly start with the Convolutional layers. And I froze the weights in the feature layers, therefore, all the pre-trained convolutional weights will not change during training. So, I will use the fixed feature extractor for all images.

At the end, I replaced the last linear dense layer with a layer that outputs 3 classes to adopt the Inception_V3 model to my own model. I also used the callback function and set the accuracy rate to 98%.

```
x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense  (3, activation='softmax')(x)
```

Validation Data set:

```
Epoch 50/50
9/9 - 24s - loss: 0.1176 - accuracy: 0.9565 - val_loss: 0.3610 -
val_accuracy: 0.8906
```

Test data set:

```
1/1 [==============================] - 1s 1s/step - loss: 0.2327 -
accuracy: 0.9141
Accuracy on test dataset: 0.9140625
```

Refinement

In this study, my goal was at least to reach 98% accuracy. The simple model reached the highest accuracy on %77 even though I implemented different strategies like data Augmentation, Dropout layer and different Nodes on each layer, but it has not changed the accuracy rate dramatically. I did not add more layers because I thought it may cause the overfitting since my data set was very small.

Nevertheless, Transfer learning reached the goal after the 9th epoch. Inception_V3 was trained over million images on ImageNet dataset and provides high accuracy on image recognition tasks.

Deployment:

I used TensorFlow Lite to deploy my model on android programs. I found Introduction Tensorflow Lite Course at Udacity very helpful.

First, I saved my model, and I applied TFLliteConverter on my saved model. After post training integer quantization, I converted my model and downloaded the labels and converted_model.tflite.

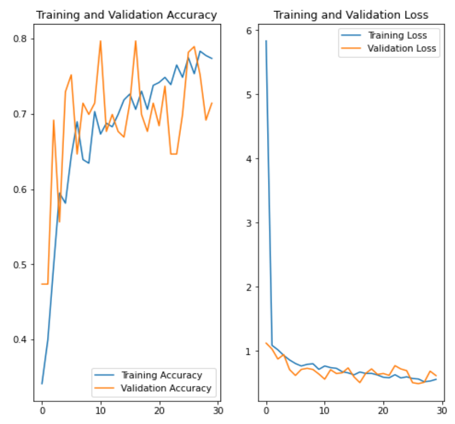
```
#Save Model
BEAN_MODEL= "exp_saved_model"
tf.saved_model.save(model,BEAN_MODEL)

#TFLite Model Converter
converter = tf.lite.TFLiteConverter.from_saved_model(BEAN_MODEL)

#Post Training Quantization
converter.optimizations =[tf.lite.Optimize.DEFAULT]

#Post Training Integer Quantization
def representative_data_gen():
  for input_value in tf.data.Dataset.from_tensor_slices(test_images).batch
(1).take(100):
    yield [input_value]

#Convert the Model
tflite_model = converter.convert()
tflite_model_file = 'converted_model.tflite'

with open(tflite_model_file, "wb") as f:
  f.write(tflite_model)
```

Results

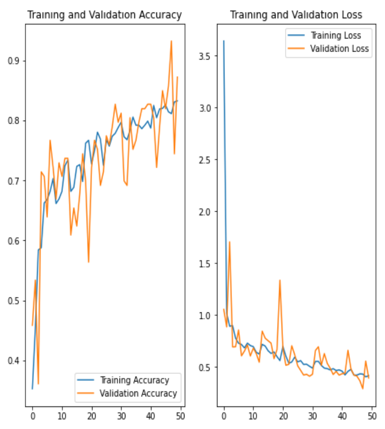
Model Evaluation and Validation

As I mentioned earlier, I used a confusion matrix and loss function to evaluate my models. We can see how up, and downs diminish after the dropout layer is applied on the loss function. Similarly, Validation accuracy also changes dramatically. It follows training accuracy and makes some maximums with a dropout layer and makes minimums without a dropout layer.

Loss Function with Dropout Layer

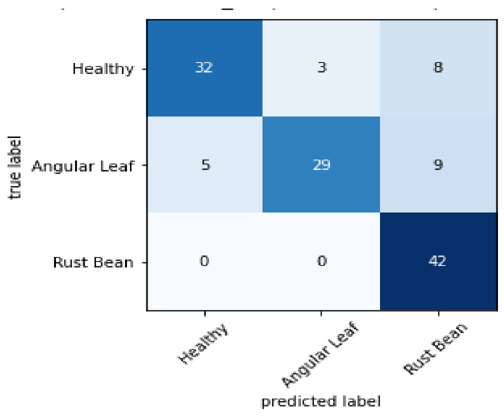


Loss Function without Dropout Layer

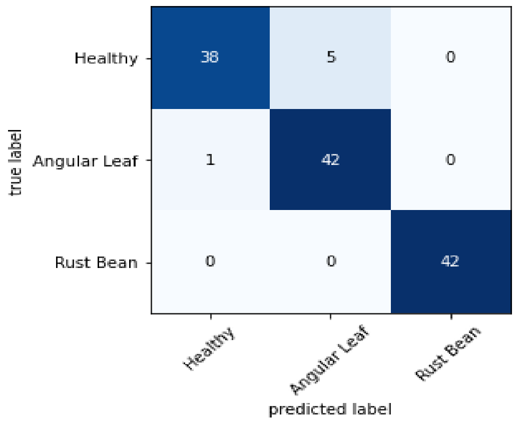


Confusion matrix also gives us a clear picture. Predicted model under Transfer learning has increased dramatically.

Confusion Matrix with Simple CNN Model **VS.** **Confusion Matrix with Transfer Learning**



true label \ predicted label	Healthy	Angular Leaf	Rust Bean
Healthy	32	3	8
Angular Leaf	5	29	9
Rust Bean	0	0	42

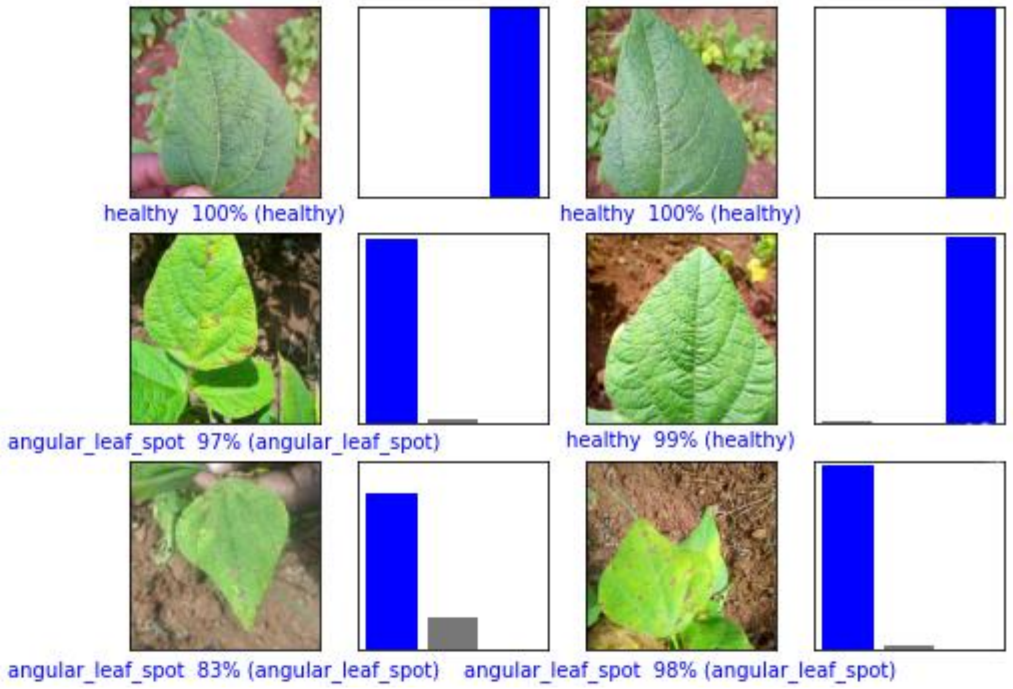


true label \ predicted label	Healthy	Angular Leaf	Rust Bean
Healthy	38	5	0
Angular Leaf	1	42	0
Rust Bean	0	0	42

Justification

Comparison of these two models prove to me that Transfer Learning is a very effective tool in the Image Classification problems. I set the callback to 98% accuracy and the Inception_v3 model predicted 98% accuracy rate after the 9th epoch and stopped the training; however, the CNN model was able to predict 77 % accuracy rate after 50 epochs.

I would like to see the outcome with the prediction rate, I set the blue for correct prediction and red for the misprediction. Most of the images were correctly classified with Inception_V3 model. Few examples are shown below.


healthy 100% (healthy)
healthy 100% (healthy)
angular_leaf_spot 97% (angular_leaf_spot)
healthy 99% (healthy)
angular_leaf_spot 83% (angular_leaf_spot)
angular_leaf_spot 98% (angular_leaf_spot)