

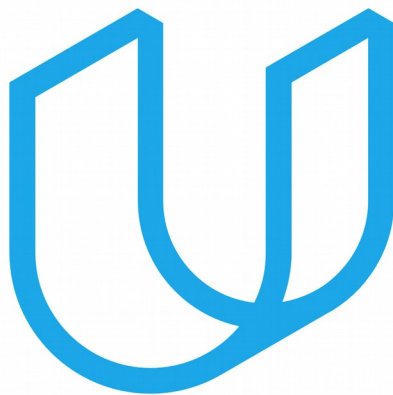
1 Machine Learning Engineer Nanodegree

CAPSTONE PROJECT

Tathagat Malviya

Classification of Plant Diseases On the Android App using
Deep Learning

August 2019



UDACITY

1 Machine Learning Engineer Nanodegree

1 Definition

1.1 Project Overview:

Today, India ranks second worldwide in farm output. Agriculture and allied sectors like forestry and fisheries accounted for 13.7% of the GDP (gross domestic product) in 2013 about 50% of the workforce. The economic contribution of agriculture to India's GDP is steadily declining with the country's broad-based economic growth. Still, agriculture is demographically the broadest economic sector and plays a significant role in the overall socio-economic fabric of India. Indian agricultural/horticultural and processed foods are exported to more than 120 countries.

A very large population of India is working in fields, they don't have access to labs to check for diseases in the plants that can cause huge losses in the crops. To help those farmers we need a solution which can be used to easily detect crop diseases and which is affordable and can be easily accessible. Doing so will help farmers to improve their productivity per acre, also this solution helps them to increase their income.

The solution will be an android application which can be used to deploy our CNN image classifier model on this app using TensorFlow lite. The android app is a simple app with basic UI. The model will be a pre-trained model to reduce the training time. For this app, I will use mobile net architecture as it is lightweight and optimized for mobile application.

1 Machine Learning Engineer Nanodegree

To integrate this model on android application I will use TensorFlow lite library.

1.2 Problem Statement

Major problems faced by farmers

- * Instability
- * Cropping Pattern
- * Excessive use of fertilizers in the soil.
- * Inadequate use of manures and fertilizers
- * Soil degradation
- * Crop Rotation
- * Dependence on monsoon.

I'll try to solve the above problems by implementing an application.

I will see this problem as an image classification problem and implement the resulting model on an android device which can be directly used by farmers.

1.3 Metrics

According to crowdAI:-

Predictions will be evaluated using a Multi Class Log Loss evaluation function, which are defined as :

Mean F1 score

The F1 score is computed separately for all classes by using:

1 Machine Learning Engineer Nanodegree

$$F_1 = 2 \frac{pr}{p + r}$$

$$p = \frac{tp}{tp + fp}$$

$$r = \frac{tp}{tp + fn}$$

- p refers to the precision
- r refers to the recall
- tp refers to the number of True Positives,
- fp refers to the number of False Positives
- fn refers to the number of False Negatives

Then finally the Mean of all the F1 scores across all the classes is used for come up with the combined Mean F1 score.

Mean Log Loss

$$L = -\frac{1}{N} \sum_i^N \sum_j^M y_{ij} \cdot \ln(p_{ij})$$

- N is the total number of examples in the test set
 - M is the total number of class labels (38 for this challenge)
 - y_{ij} is a boolean value representing if the i -th instance in the test set belongs to the j -th label.
 - p_{ij} is the probability according to your submission that the i -th instance may belong to the j -th label.
- \ln is the natural logarithmic function.

1 Machine Learning Engineer Nanodegree

2 Analysis

2.1 Data Exploration and Visualisation

2.1.1 Plant Village Dataset

The dataset is freely available on Kaggle website under the name as plant village dataset.

The datasets include 15 types of diseases, each disease folder include 500 – 1500 images each.

Link:-<https://www.kaggle.com/emmarex/plantdisease> .

The images are already preprocessed and are in various orientations. Each Image is 256x256 in dimension. Then I have to split the images, 20% on the testing set from each class and remaining on the training set, and here the data is ready for the raining.

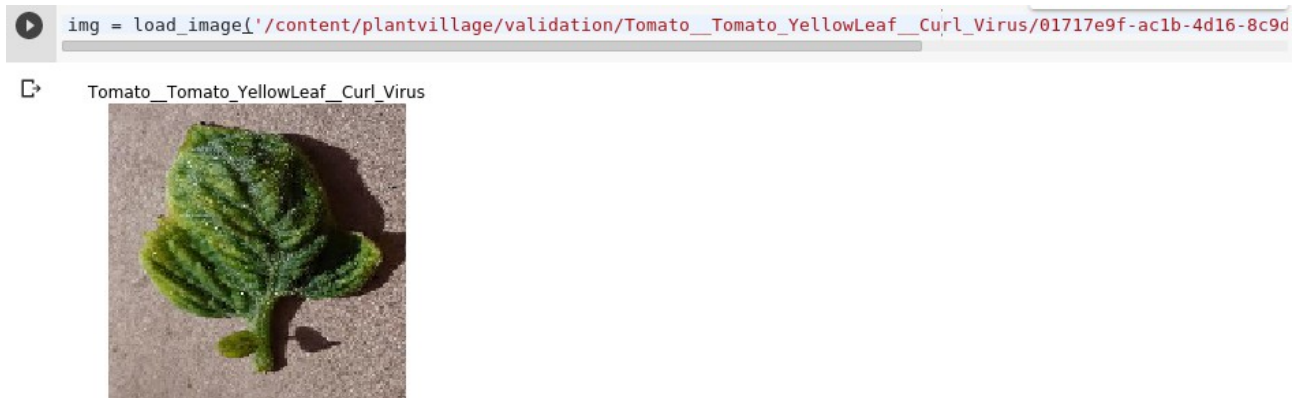
The input to the trained CNN is via mobile camera, the user has to take a picture of the leaf of a plant, then the picture is passed through a CNN and predicted label is shown to the user.

2.1.2 Analysing Image data

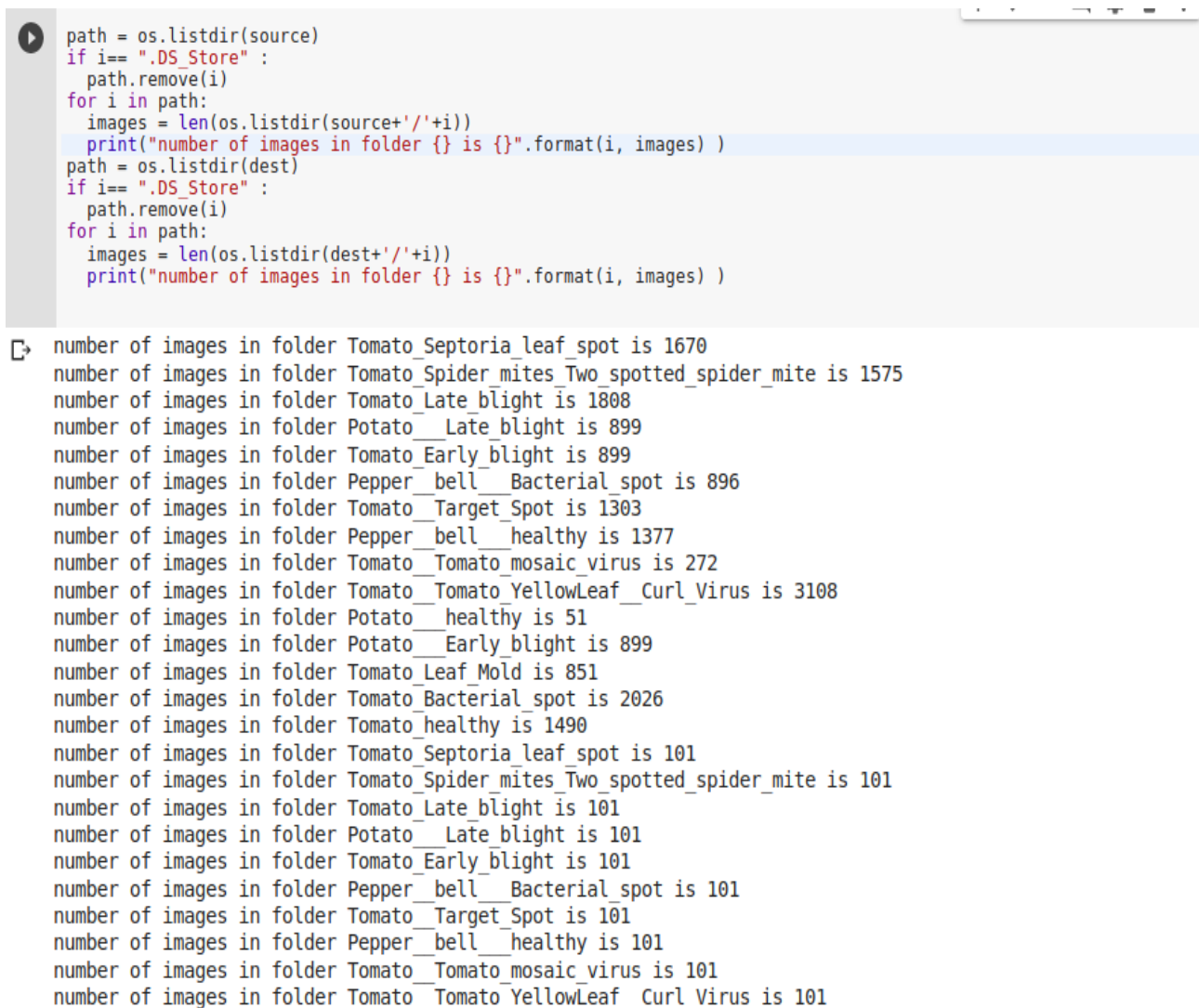
We can use simple matplotlib to see the image data in our notebook.

```
def load_image(img_path, show=False):  
    img = image.load_img(img_path, target_size=(150, 150))  
    img_tensor = image.img_to_array(img) # (height, width, channels)  
    img_tensor = np.expand_dims(img_tensor, axis=0) # (1, height, width, channels), add a dimension beca  
    img_tensor /= 255. # imshow expects values in the range [0, 1]  
  
    if show:  
        plt.imshow(img_tensor[0])  
        name = str(os.path.dirname(os.path.normpath(img_path)))  
        name = name[name.rfind('/')+1:]  
  
        plt.title(name)  
        plt.axis('off')  
        plt.show()  
  
    return img_tensor
```

1 Machine Learning Engineer Nanodegree



The number of images are not equal for every classes.



1 Machine Learning Engineer Nanodegree

The labels in the dataset are 15, Each having different number of images. The folder itself is the label of the image inside it.

```
path = os.listdir(source)
for i in path:
    if i == ".DS_Store" :
        path.remove(i)
print(*path, sep = '\n')
```

```
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato_Late_blight
Potato_Late_blight
Tomato_Early_blight
Pepper_bell_Bacterial_spot
Tomato_Target_Spot
Pepper_bell_healthy
Tomato_Tomato_mosaic_virus
Tomato_Tomato_YellowLeaf_Curl_Virus
Potato_healthy
Potato_Early_blight
Tomato_Leaf_Mold
Tomato_Bacterial_spot
Tomato_healthy
```

2.2 Algorithms and Techniques

The solution will be an android application which can be used to deploy our CNN image classifier model on this app using TensorFlow lite. The android app is a simple app with basic UI. The model will be a pre-trained model to reduce the training time. For this app, I will use mobile net architecture as it is lightweight and optimized for mobile application.

To integrate this model on android application I will use TensorFlow lite library.

1 Machine Learning Engineer Nanodegree

2.3 Benchmark Model

I can use a single layer simple CNN to compare the performance of Mobilenet CNN.

We can train our benchmark model on the complete dataset

We can run the Multi-Class Log function over single-layer CNN and then we can compare our model with the mobile net CNN.

```
step_size_train=train_generator.n//train_generator.batch_size
history = benchmark_model.fit_generator(generator=train_generator,
                                       validation_data=validation_generator,
                                       validation_steps=12,
                                       steps_per_epoch=step_size_train,
                                       epochs=10)
```



```
Epoch 1/10
149/149 [=====] - 61s 408ms/step - loss: 1.5893 - acc: 0.9002 - val_loss: 2.0482 - val_acc: 0.8719
Epoch 2/10
149/149 [=====] - 54s 364ms/step - loss: 1.6435 - acc: 0.8974 - val_loss: 1.7482 - val_acc: 0.8909
Epoch 3/10
149/149 [=====] - 54s 364ms/step - loss: 1.6511 - acc: 0.8969 - val_loss: 1.9458 - val_acc: 0.8785
Epoch 4/10
149/149 [=====] - 54s 363ms/step - loss: 1.6279 - acc: 0.8983 - val_loss: 1.6251 - val_acc: 0.8984
Epoch 5/10
149/149 [=====] - 54s 362ms/step - loss: 1.5768 - acc: 0.9015 - val_loss: 2.0717 - val_acc: 0.8707
Epoch 6/10
149/149 [=====] - 54s 363ms/step - loss: 1.6590 - acc: 0.8964 - val_loss: 1.6080 - val_acc: 0.8997
Epoch 7/10
149/149 [=====] - 54s 362ms/step - loss: 1.5880 - acc: 0.9009 - val_loss: 1.9286 - val_acc: 0.8797
Epoch 8/10
149/149 [=====] - 54s 363ms/step - loss: 1.5495 - acc: 0.9033 - val_loss: 1.5156 - val_acc: 0.9054
Epoch 9/10
149/149 [=====] - 54s 363ms/step - loss: 1.5659 - acc: 0.9023 - val_loss: 2.0984 - val_acc: 0.8691
Epoch 10/10
149/149 [=====] - 54s 363ms/step - loss: 1.5778 - acc: 0.9015 - val_loss: 1.5498 - val_acc: 0.9033
```

Clearly the benchmark model is overfittig and the model is also not learning because loss function is not improving with epochs.

1 Machine Learning Engineer Nanodegree

3 Methodology

3.1.1 Data Preprocessing and Data Splitting

The data is already pre-processed since all the images are in same sizes, also images are in various orientation so there is no need for data augmentation. First, I will split the images into training(80%) and testing set(20%). Now the data is ready for training.

Then we can save our training data in validation folder and training data on plantvillage folder.

A screenshot of a code editor showing Python code. The code is color-coded: 'import' is purple, 'os', 'shutil', and 'cv2' are blue, and the file paths are red. The code is as follows:

```
import os
import shutil
import cv2

source = '/content/plantvillage/PlantVillage'
dest = '/content/plantvillage/validation'
```

```
import os
import shutil
import cv2

source = '/content/plantvillage/PlantVillage'
dest = '/content/plantvillage/validation'
```

1 Machine Learning Engineer Nanodegree

```
for i in path:
    if i== ".DS_Store" :

        path.remove(i)
    for i in path:
        os.mkdir(dest+'/'+i)
        images = os.listdir(source+'/'+i)
        for j in images:
            if j== ".DS_Store" :

                images.remove(j)
        k = 0

    for m in images:
        shutil.move(source+'/'+i+'/'+m,dest+'/'+i+'/'+m)
        k+=1
    #print(m)
    if k >100:
        break
```

3.1.2 Convert the data and labels

Mobilenet cnn in keras can be only be implemented by data generators, So we use keras data generators for creating validation and training dataset to feed in our cnn.

Validation data

```
[77] testing_folder = "/content/plantvillage/validation"
# Image size (set up the image size used for training)
img_size = 256
# Batch size
batch_size = 64

val_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)
validation_generator = val_datagen.flow_from_directory(
    testing_folder,
    target_size=(256, 256),
    color_mode='rgb',
    batch_size=batch_size,
    shuffle=False,
    class_mode='categorical')
```

Found 1515 images belonging to 15 classes.

Training Data

1 Machine Learning Engineer Nanodegree

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input) #included in our dependencies
train_generator=train_datagen.flow_from_directory('/content/plantvillage/PlantVillage',
                                                  target_size=(256,256),
                                                  color_mode='rgb',
                                                  batch_size=128,
                                                  class_mode='categorical',
                                                  shuffle=True)
```

Found 19123 images belonging to 15 classes.

3.1.3 Split the dataset

Dataset is already split and is ready to be fitted into our model.

3.2 Implementation

We will use mobilenet cnn architecture as our model and we will train this network from scratch. In this model we will define custom hidden layers for our purpose to avoid overfitting due to large size of hidden layers in the original network.

```
base_model=MobileNet(include_top=False) #imports the mobilenet model and discards the last 1000 neuron layer.
x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #dense layer 1
x=Dense(512,activation='relu')(x) #dense layer 2
preds=Dense(15,activation='softmax')(x) #final layer with softmax activation
```

/usr/local/lib/python3.6/dist-packages/keras_applications/mobilenet.py:207: UserWarning: 'input_shape' is undefined or non-square, or 'rows' is not in [12
warnings.warn('input_shape' is undefined or non-square, '

We applied 2 hidden fully connected layers with size of 1024 and 512 respectively with relu activation.

We added final layer of size 15 as our output layer.


1 Machine Learning Engineer Nanodegree

```
[79] model=Model(inputs=base_model.input,outputs=preds)
```

```
[80] for layer in model.layers[:]:  
      layer.trainable=True
```

Now our model is ready to compile and train.

```
[93] model.summary()
```



Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, None, None, 3)	0
conv1_pad (ZeroPadding2D)	(None, None, None, 3)	0
conv1 (Conv2D)	(None, None, None, 32)	864
conv1_bn (BatchNormalization)	(None, None, None, 32)	128
conv1_relu (ReLU)	(None, None, None, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, None, None, 32)	288
conv_dw_1_bn (BatchNormaliza)	(None, None, None, 32)	128
conv_dw_1_relu (ReLU)	(None, None, None, 32)	0
conv_pw_1 (Conv2D)	(None, None, None, 64)	2048
conv_pw_1_bn (BatchNormaliza)	(None, None, None, 64)	256
conv_pw_1_relu (ReLU)	(None, None, None, 64)	0

The android application stage can be split into 2 steps :-

1. Copy and compile the android tensorflow example.
2. Copy the Cnn saved file with extension of .pb in assets folder and then deploy the application.

1 Machine Learning Engineer Nanodegree

conv_dw_13_bn (BatchNormaliz	(None, None, None, 1024)	4096
conv_dw_13_relu (ReLU)	(None, None, None, 1024)	0
conv_pw_13 (Conv2D)	(None, None, None, 1024)	1048576
conv_pw_13_bn (BatchNormaliz	(None, None, None, 1024)	4096
conv_pw_13_relu (ReLU)	(None, None, None, 1024)	0
global_average_pooling2d_2 ((None, 1024)	0
dense_6 (Dense)	(None, 1024)	1049600
dense_7 (Dense)	(None, 512)	524800
dense_8 (Dense)	(None, 15)	7695
=====		
Total params: 4,810,959		
Trainable params: 4,789,071		
Non-trainable params: 21,888		

3.2.2 Compiling the model

```
[82] model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

For compiling our model, we will use the following three parameters:

- Loss function - we will use `categorical_crossentropy`. This is the most common choice for classification. A lower score indicates that the model is performing better.
- Metrics - we will use the accuracy metric which will allow us to view the accuracy score on the validation data when we train the model.
- Optimizer - here we will use `adam` which is a generally good optimizer for many use cases.

1 Machine Learning Engineer Nanodegree

3.2.3 Training

Here we will train the model.

We will start with 10 epochs which is the number of times the model will cycle through the data.

The model will improve on each cycle until it reaches a certain point.

We will also start with a low batch size, as having a large batch size can reduce the generalisation ability of the model.

```
[83] step_size_train=train_generator.n//train_generator.batch_size
     history = model.fit_generator(generator=train_generator,
                                   validation_data=validation_generator,
                                   validation_steps=12,
                                   steps_per_epoch=step_size_train,
                                   epochs=10)
```

```
Epoch 1/10
149/149 [=====] - 165s 1s/step - loss: 0.2641 - acc: 0.9199 - val_loss: 1.2354 - val_acc: 0.7227
Epoch 2/10
149/149 [=====] - 157s 1s/step - loss: 0.0648 - acc: 0.9807 - val_loss: 0.7415 - val_acc: 0.8139
Epoch 3/10
149/149 [=====] - 157s 1s/step - loss: 0.0633 - acc: 0.9812 - val_loss: 0.4494 - val_acc: 0.8750
Epoch 4/10
149/149 [=====] - 157s 1s/step - loss: 0.0399 - acc: 0.9886 - val_loss: 0.4882 - val_acc: 0.8956
Epoch 5/10
149/149 [=====] - 157s 1s/step - loss: 0.0326 - acc: 0.9901 - val_loss: 0.2355 - val_acc: 0.9297
Epoch 6/10
149/149 [=====] - 157s 1s/step - loss: 0.0429 - acc: 0.9867 - val_loss: 0.5280 - val_acc: 0.9076
Epoch 7/10
149/149 [=====] - 157s 1s/step - loss: 0.0340 - acc: 0.9901 - val_loss: 0.3717 - val_acc: 0.9036
Epoch 8/10
149/149 [=====] - 157s 1s/step - loss: 0.0354 - acc: 0.9904 - val_loss: 0.3194 - val_acc: 0.9344
Epoch 9/10
149/149 [=====] - 157s 1s/step - loss: 0.0302 - acc: 0.9905 - val_loss: 1.4653 - val_acc: 0.7565
Epoch 10/10
149/149 [=====] - 157s 1s/step - loss: 0.0291 - acc: 0.9907 - val_loss: 0.4340 - val_acc: 0.9183
```

1 Machine Learning Engineer Nanodegree

3.2.4 Test the model

```
[89] #val_datagen=validation_data()
      steps = 24
      predictions = model.predict_generator(validation_generator, steps=steps)
      val_trues = validation_generator.classes
      print(len(predictions), len(val_trues))
```

1515 1515

```
[90] from sklearn import metrics
      val_preds = np.argmax(predictions, axis=-1)
      val_trues = validation_generator.classes
      cm = metrics.confusion_matrix(val_trues, val_preds)
```

```
[91] labels = validation_generator.class_indices.keys()
      precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(val_trues, val_preds)
```

```
[92] print(np.mean(f1_score).sum())
```

0.9211611432493871

Here as discussed previously we will use mean f1 score of all the classes of the dataset to judge our model, clearly 0.92 is a very good f1 score for a classifier, hence we can say that our model is working well under this situation.

3.2.5 Predictions

We can predict any image on google images of leaves through our mobile application or using the notebook.

1 Machine Learning Engineer Nanodegree

```
img = load_image('/content/plantvillage/validation/Tomato__Tomato_YellowLeaf__Curl_Virus/01717e9f-ac1b-4d16-8c9d
```



```
pred = model.predict(img)  
print(pred.argmax())
```

12

Here we can see that model correctly predicted the label of the pic. We can experiment with different picture, untill now all the pictures tested by me is correctly classified.

3.3 Refinement

In the previous attempt we achieve the f1 score of 0.92, this is a very good model but tensorflow lite only uses .pb file which can only be generated by tensorflow session. I tried to convert the h5 model of keras with many methods and also by using stack overflow but failed miserably. Now we will train the fresh model of mobilenet using tensorflow with the help of tensorflow for poets 2 using shell commands and see if this works.

1 Machine Learning Engineer Nanodegree

```
!python -m scripts.retrain \
  --bottleneck_dir=tf_files/bottlenecks \
  --how_many_training_steps=500 \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/"mobilenet_0.50_224" \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --architecture="mobilenet_0.50_224" \
  --image_dir=/content/plantvillage/PlantVillage
```

```
I0817 17:52:03.993009 140329730905984 retrain.py:1084] 2019-08-17 17:52:03.992986: Step 350: Cross entropy = 0.1
I0817 17:52:04.027185 140329730905984 retrain.py:1100] 2019-08-17 17:52:04.027128: Step 350: Validation accuracy
I0817 17:52:04.390117 140329730905984 retrain.py:1082] 2019-08-17 17:52:04.390035: Step 360: Train accuracy = 92.0%
I0817 17:52:04.390325 140329730905984 retrain.py:1084] 2019-08-17 17:52:04.390301: Step 360: Cross entropy = 0.1
I0817 17:52:04.428948 140329730905984 retrain.py:1100] 2019-08-17 17:52:04.428856: Step 360: Validation accuracy
I0817 17:52:04.794064 140329730905984 retrain.py:1082] 2019-08-17 17:52:04.793985: Step 370: Train accuracy = 96.0%
I0817 17:52:04.794229 140329730905984 retrain.py:1084] 2019-08-17 17:52:04.794212: Step 370: Cross entropy = 0.1
I0817 17:52:04.829442 140329730905984 retrain.py:1100] 2019-08-17 17:52:04.829388: Step 370: Validation accuracy
I0817 17:52:05.188602 140329730905984 retrain.py:1082] 2019-08-17 17:52:05.188537: Step 380: Train accuracy = 98.0%
I0817 17:52:05.188760 140329730905984 retrain.py:1084] 2019-08-17 17:52:05.188743: Step 380: Cross entropy = 0.1
I0817 17:52:05.223917 140329730905984 retrain.py:1100] 2019-08-17 17:52:05.223862: Step 380: Validation accuracy
I0817 17:52:05.589964 140329730905984 retrain.py:1082] 2019-08-17 17:52:05.589885: Step 390: Train accuracy = 92.0%
I0817 17:52:05.590164 140329730905984 retrain.py:1084] 2019-08-17 17:52:05.590132: Step 390: Cross entropy = 0.1
```

```
I0817 17:52:08.360704 140329730905984 retrain.py:1082] 2019-08-17 17:52:08.360641: Step 460: Train accuracy = 93.0%
I0817 17:52:08.360862 140329730905984 retrain.py:1084] 2019-08-17 17:52:08.360846: Step 460: Cross entropy = 0.260766
I0817 17:52:08.395399 140329730905984 retrain.py:1100] 2019-08-17 17:52:08.395345: Step 460: Validation accuracy = 81.0% (N=100)
I0817 17:52:08.762638 140329730905984 retrain.py:1082] 2019-08-17 17:52:08.762563: Step 470: Train accuracy = 92.0%
I0817 17:52:08.762809 140329730905984 retrain.py:1084] 2019-08-17 17:52:08.762792: Step 470: Cross entropy = 0.266232
I0817 17:52:08.799453 140329730905984 retrain.py:1100] 2019-08-17 17:52:08.799397: Step 470: Validation accuracy = 89.0% (N=100)
I0817 17:52:09.156688 140329730905984 retrain.py:1082] 2019-08-17 17:52:09.156621: Step 480: Train accuracy = 94.0%
I0817 17:52:09.156844 140329730905984 retrain.py:1084] 2019-08-17 17:52:09.156828: Step 480: Cross entropy = 0.221930
I0817 17:52:09.190648 140329730905984 retrain.py:1100] 2019-08-17 17:52:09.190592: Step 480: Validation accuracy = 86.0% (N=100)
I0817 17:52:09.556972 140329730905984 retrain.py:1082] 2019-08-17 17:52:09.556901: Step 490: Train accuracy = 94.0%
I0817 17:52:09.557148 140329730905984 retrain.py:1084] 2019-08-17 17:52:09.557130: Step 490: Cross entropy = 0.248198
I0817 17:52:09.591201 140329730905984 retrain.py:1100] 2019-08-17 17:52:09.591147: Step 490: Validation accuracy = 84.0% (N=100)
I0817 17:52:09.914844 140329730905984 retrain.py:1082] 2019-08-17 17:52:09.914777: Step 499: Train accuracy = 94.0%
I0817 17:52:09.915009 140329730905984 retrain.py:1084] 2019-08-17 17:52:09.914991: Step 499: Cross entropy = 0.196337
I0817 17:52:09.949752 140329730905984 retrain.py:1100] 2019-08-17 17:52:09.949699: Step 499: Validation accuracy = 81.0% (N=100)
I0817 17:52:10.740290 140329730905984 retrain.py:1126] Final test accuracy = 89.5% (N=1892)
W0817 17:52:10.751311 140329730905984 deprecation.py:323] From /content/tensorflow-for-poets-2/scripts/retrain.py:827: convert_variables_to_constants (f
Instructions for updating:
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
W0817 17:52:10.751562 140329730905984 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/graph_util_impl.py:270
Instructions for updating:
```

We successfully implemented a tensorflow graph with the accuracy of 94% and validation accuracy of 81.9%

1 Machine Learning Engineer Nanodegree

4 Results

4.1 Model Evaluation and Validation

During the model development phase the validation data was used to evaluate the model. The final model architecture and hyperparameters were chosen because they performed the best among the tried combinations. This architecture is described in detail in section 3.

To verify the final application we can perform specified tests:-

1. Application performed well on separate images obtained from internet.
2. There are fluctuations in the predicted labels by the classifier.
3. Majority of the diseases can be predicted by the app.

4.2 Justification

Using samsung j7 max I got the following results:-

1. The classifier is able to classify images with good fps.
2. The application is running smoothly.
3. Classifier is able to detect correct label (even from the google images) with high accuracy (usually $> 50\%$).

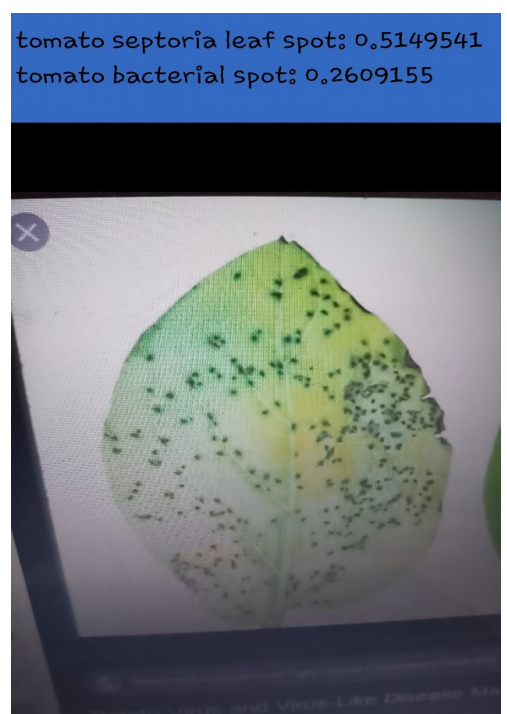
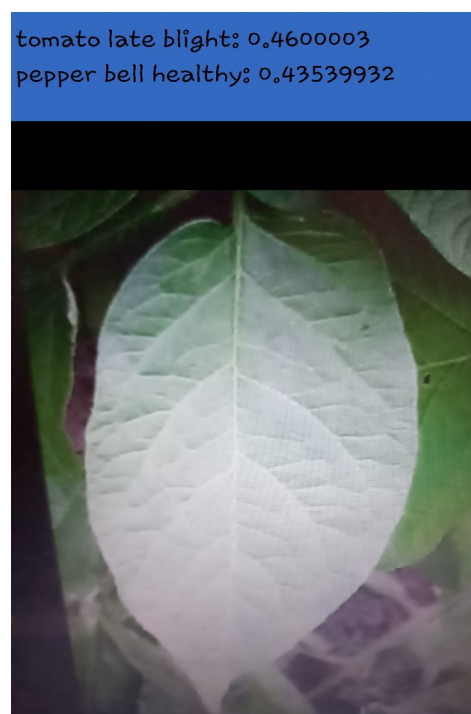
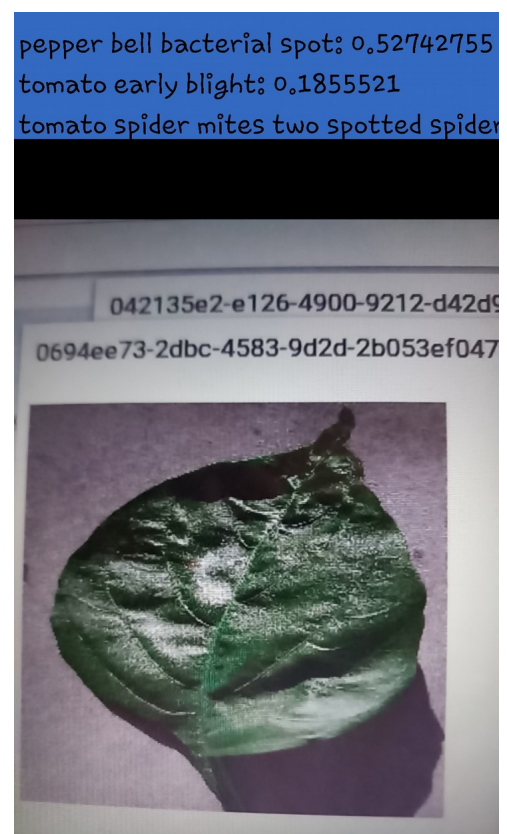
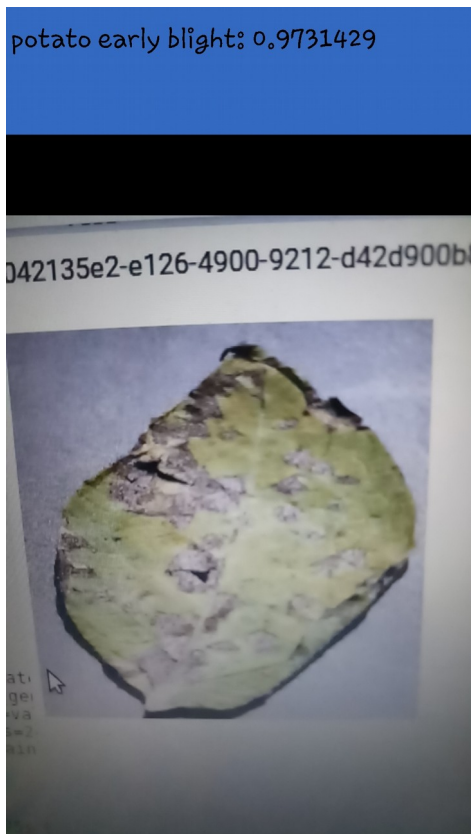
In summary the application is useful in limited domain, to further improve this application we can use more balanced data for training.

1 Machine Learning Engineer Nanodegree

5 Conclusion

5.1 Free Form Visualization

Examples of the application detecting images are:-



1 Machine Learning Engineer Nanodegree

Clearly the identified labels are true and all the images are from google images. Some issues are there which are:-

1. Classifier depend upon the quality of the camera and the person, suppose if you shake the mobile then the label will also fluctuates.
2. Some images are hard to classify for the classifier on the internet.
3. To classify an image classifier need only one image of the leaf, otherwise it will give false results.

5.2 Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public datasets were found
2. The data was downloaded and preprocessed (segmented)
3. A benchmark was created for the classifier
4. The classifier was trained using the data (multiple times, until a good set of parameters were found)
5. The TensorFlow Android demo was adapted to run the classifier

I found steps 4 and 5 the most difficult, as I had to familiarize myself with the files of the TensorFlow Android demo, which uses Bazel and the Android NDK, both of which were technologies that I was not familiar with before the project.

1 Machine Learning Engineer Nanodegree

5.3 Improvement

This application can be improved by introducing more data. To achieve optimal user experience more effective application can be build.

The accuracy of the classifier can be increased by more number of epochs.

The classifier accuracy can also be improved by using imagenet or alexanet cnn.

1 Machine Learning Engineer Nanodegree

5.5 References:

<https://codelabs.developers.google.com/codelabs/tensorflow-for-poets>

<https://www.kaggle.com/emmarex/plantdisease>