

Plant Disease classification using Deep Learning

Team #8

Mahesh Mandava

Adhisekhar Raju Uppalapati

Sudheer Nimmagadda

1. Motivation:

We depend on edible plants just as we depend on oxygen. Without crops, there is no food, and without food, there is no life. It's no accident that human civilization began to thrive with the invention of agriculture.

Today, modern technology allows us to grow crops in quantities necessary for a steady food supply for billions of people. But diseases remain a major threat to this supply, and a large fraction of crops are lost each year to diseases. The situation is particularly dire for the 500 million smallholder farmers around the globe, whose livelihoods depend on their crops doing well. In Africa alone, 80% of the agricultural output comes from smallholder farmers.

With billions of smartphones around the globe, wouldn't it be great if the smartphone could be turned into a disease diagnostics tool, recognizing diseases from images it captures with its camera?

2. Project Idea:

Using CNN, Alexnet on a public data set by plant village for plant diseases classification.

First we collect data that is publicly available by plant village.

Second we do image pre processing is used to normalize the images of the dataset.

The most used techniques are image resizing.

Third, we train the data using CNN.

Fourth the model is deployed where the output of the network determines which diseases are present in the plant.

Below are 38 classes of crop disease pairs that the dataset is offering:



Dataset is downloaded from crowdai.org maintained by plant village.

← → ⌛ https://www.crowdai.org/challenges/plantvillage-disease-classification-challenge/dataset_files# Paused

Apps eJoror W: Shamir's Secret S... Realizing secret s... I3-RA-RAxter-Pub... (3) Bitcoin: How C... M: How to Extract W... Analyzing Bitcoin... How Bitcoin Work... CSEE5590/490: P... »

PlantVillage Disease Classification Challenge

PlantVillage is built on the premise that all knowledge that helps people grow food should be openly accessible to anyone on the planet.

By PlantVillage

Completed

38619 1893 36

Views Participants Submissions

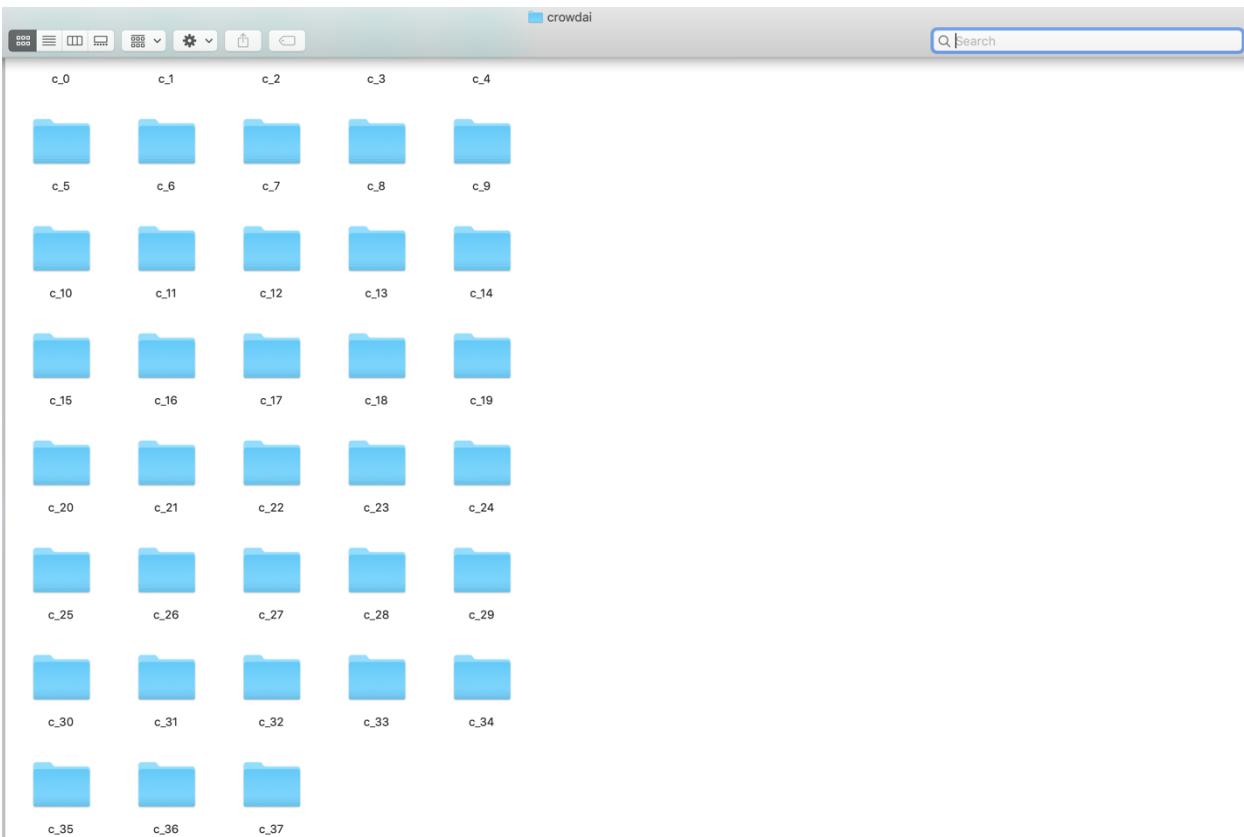
57 UNFOLLOW

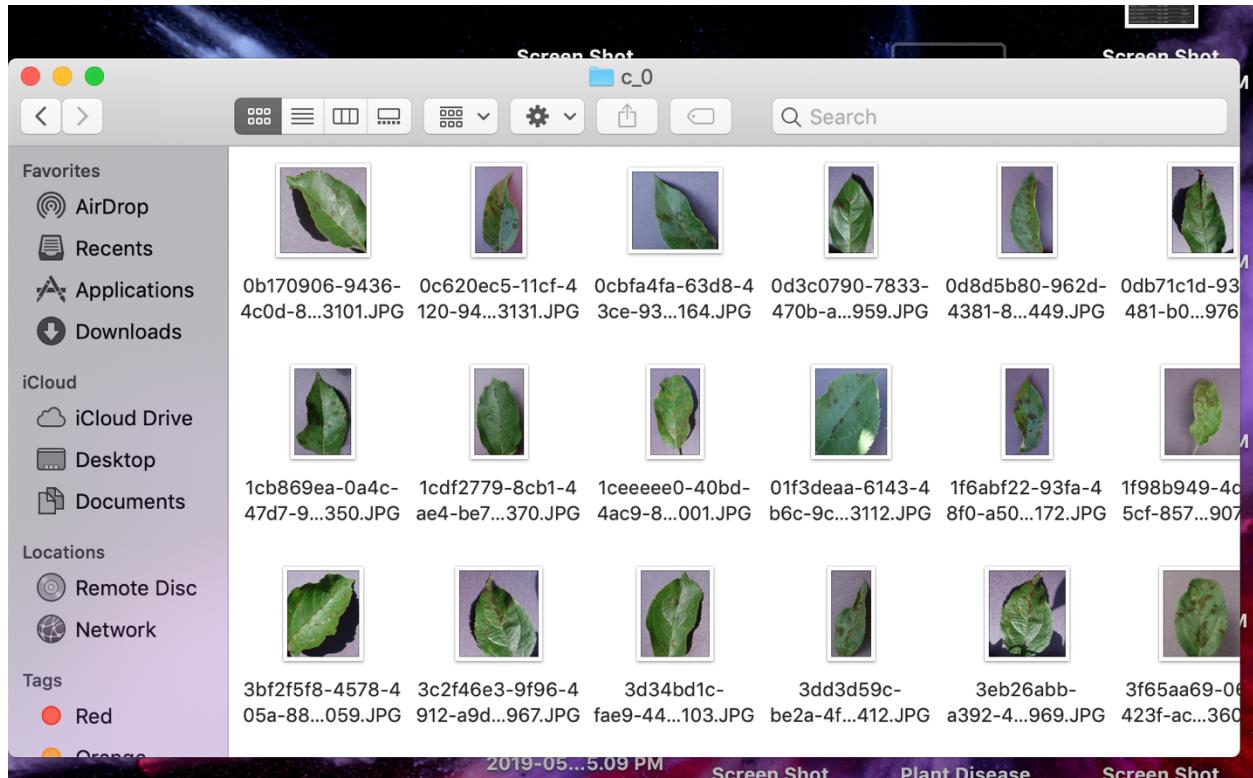
Overview Leaderboard Discussion Dataset Submissions Winners

| Training Data | File Format | File Size |
|---------------|-------------|-----------|
| | TAR | 515 MB |

| Test Data | File Format | File Size |
|-----------|-------------|-----------|
| Test Data | TAR | 762 MB |

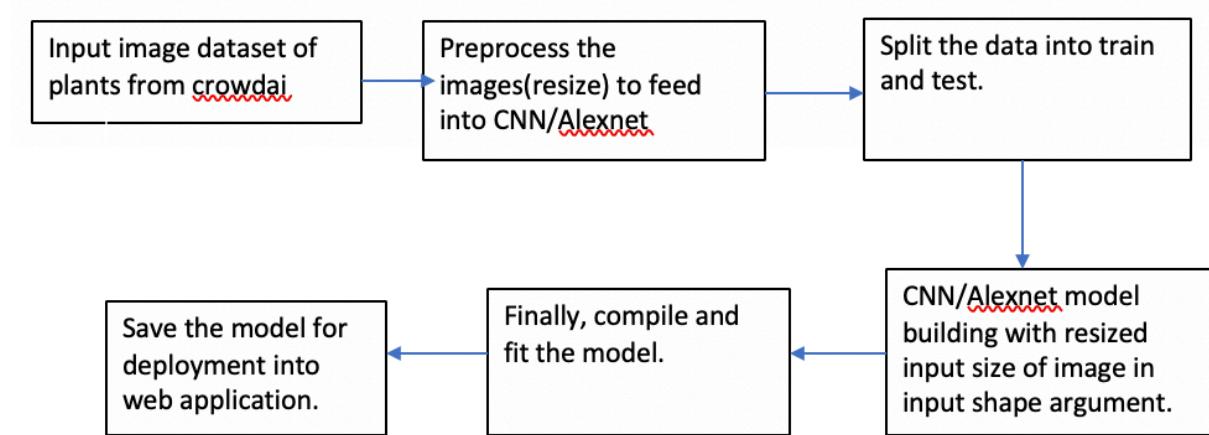
Below are the snapshots of our dataset.





3. System Architecture:

Basic Block Diagram:



CNN architecture in our project:

For CNN the image input shape is given as (28,28) and the below code represents the architecture of CNN.

```

model = Sequential()

model.add(Convolution2D(32, 3, 3, input_shape=(28, 28, 3)))
model.add(Activation('relu'))

model.add(Convolution2D(64, 3, 3))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(16, 3, 3))
model.add(Activation('relu'))

model.add(Flatten())

model.add(Dropout(0.2))
model.add(Dense(num_classes))

model.add(Activation('softmax'))

model.summary()

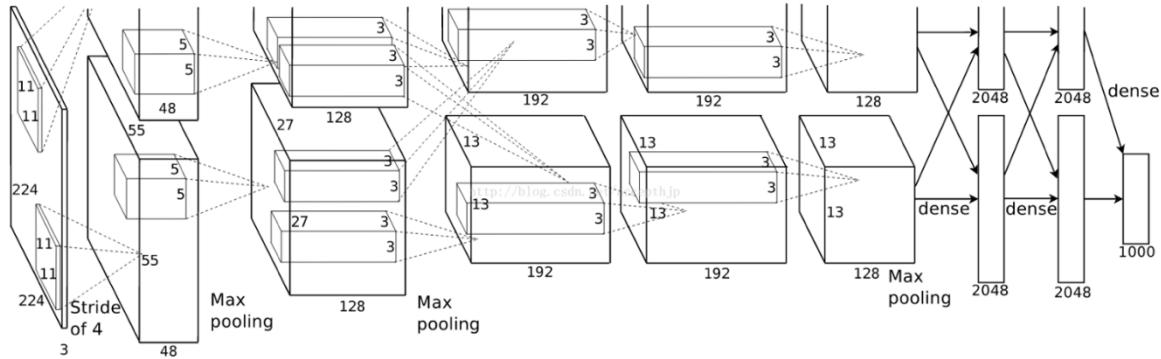
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

```

Below is the summary of CNN:

| Layer (type) | Output Shape | Param # |
|--------------------------------|--------------------|---------|
| <hr/> | | |
| conv2d_1 (Conv2D) | (None, 26, 26, 32) | 896 |
| activation_1 (Activation) | (None, 26, 26, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 24, 24, 64) | 18496 |
| activation_2 (Activation) | (None, 24, 24, 64) | 0 |
| max_pooling2d_1 (MaxPooling2D) | (None, 12, 12, 64) | 0 |
| conv2d_3 (Conv2D) | (None, 10, 10, 16) | 9232 |
| activation_3 (Activation) | (None, 10, 10, 16) | 0 |
| flatten_1 (Flatten) | (None, 1600) | 0 |
| dropout_1 (Dropout) | (None, 1600) | 0 |
| dense_1 (Dense) | (None, 38) | 60838 |
| activation_4 (Activation) | (None, 38) | 0 |
| <hr/> | | |
| Total params: 89,462 | | |
| Trainable params: 89,462 | | |
| Non-trainable params: 0 | | |

Alexnet architecture in our project:



Similar structure to LeNet, AlexNet has more filters per layer, deeper and stacked. There are 5 convolutional layers, 3 fully connected layers and with Relu applied after each of them, and dropout applied before the first and second fully connected layer.

Below is the code for alexnet architecture.

```
# Initializing the CNN
classifier = Sequential()

# Convolution Step 1
classifier.add(Convolution2D(96, 11, strides=(4, 4), padding='valid', input_shape=(224, 224, 3))

# Max Pooling Step 1
classifier.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
classifier.add(BatchNormalization())

# Convolution Step 2
classifier.add(Convolution2D(256, 11, strides=(1, 1), padding='valid', activation='relu'))

# Max Pooling Step 2
classifier.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
classifier.add(BatchNormalization())

# Convolution Step 3
classifier.add(Convolution2D(384, 3, strides=(1, 1), padding='valid', activation='relu'))
classifier.add(BatchNormalization())

# Convolution Step 4
classifier.add(Convolution2D(384, 3, strides=(1, 1), padding='valid', activation='relu'))
classifier.add(BatchNormalization())

# Convolution Step 5
classifier.add(Convolution2D(256, 3, strides=(1, 1), padding='valid', activation='relu'))

# Max Pooling Step 3
classifier.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
classifier.add(BatchNormalization())

# Flattening Step
classifier.add(Flatten())
```

```
# Full Connection Step
classifier.add(Dense(units=4096, activation='relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units=4096, activation='relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units=1000, activation='relu'))
classifier.add(Dropout(0.2))
classifier.add(BatchNormalization())
classifier.add(Dense(units=38, activation='softmax'))
classifier.summary()
```

we chose to train the top 2 conv blocks, i.e. we will freeze the first 8 layers and unfreeze the rest:

The code is as follows:

```
for i, layer in enumerate(classifier.layers[:20]):
    print(i, layer.name)
    layer.trainable = False
classifier.summary()

classifier.compile(optimizer=optimizers.SGD(lr=0.001, momentum=0.9, decay=0.005),
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])
```

Below is the summary of Alexnet:

| Layer (type) | Output Shape | Param # |
|---|---------------------|----------|
| conv2d_1 (Conv2D) | (None, 54, 54, 96) | 34944 |
| max_pooling2d_1 (MaxPooling2D) | (None, 27, 27, 96) | 0 |
| batch_normalization_1 (Batch Normalization) | (None, 27, 27, 96) | 384 |
| conv2d_2 (Conv2D) | (None, 17, 17, 256) | 2973952 |
| max_pooling2d_2 (MaxPooling2D) | (None, 8, 8, 256) | 0 |
| batch_normalization_2 (Batch Normalization) | (None, 8, 8, 256) | 1024 |
| conv2d_3 (Conv2D) | (None, 6, 6, 384) | 885120 |
| batch_normalization_3 (Batch Normalization) | (None, 6, 6, 384) | 1536 |
| conv2d_4 (Conv2D) | (None, 4, 4, 384) | 1327488 |
| batch_normalization_4 (Batch Normalization) | (None, 4, 4, 384) | 1536 |
| conv2d_5 (Conv2D) | (None, 2, 2, 256) | 884992 |
| max_pooling2d_3 (MaxPooling2D) | (None, 1, 1, 256) | 0 |
| batch_normalization_5 (Batch Normalization) | (None, 1, 1, 256) | 1024 |
| flatten_1 (Flatten) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 4096) | 1052672 |
| dropout_1 (Dropout) | (None, 4096) | 0 |
| batch_normalization_6 (Batch Normalization) | (None, 4096) | 16384 |
| dense_2 (Dense) | (None, 4096) | 16781312 |
| dropout_2 (Dropout) | (None, 4096) | 0 |
| batch_normalization_7 (Batch Normalization) | (None, 4096) | 16384 |

| | |
|--|---------------------------|
| <code>batch_normalization_7 (Batch (None, 4096)</code> | <code>16384</code> |
| <code>dense_3 (Dense)</code> | <code>(None, 1000)</code> |
| <code>dropout_3 (Dropout)</code> | <code>(None, 1000)</code> |
| <code>batch_normalization_8 (Batch (None, 1000)</code> | <code>4000</code> |
| <code>dense_4 (Dense)</code> | <code>(None, 38)</code> |
| <hr/> | |
| <code>Total params: 28,117,790</code> | |
| <code>Trainable params: 4,137,038</code> | |
| <code>Non-trainable params: 23,980,752</code> | |
| <hr/> | |
| <code>WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions</code> | |
| <code>Instructions for updating:</code> | |
| <code>Use tf.cast instead.</code> | |
| <code>Train on 13150 samples, validate on 8767 samples</code> | |
| <code>Epoch 1/25</code> | |

This is how our both CNN and alexnet models are built in our project.

4. Technical Stack:

We used keras for neural network model building and tensor board for viewing the accuracy and loss graphs. And we also used scientific packages like scikit-learn, numpy, matplotlib. And finally we deployed our neural network model in the web application developed using flask (a micro web framework written in python). And also we used Pycharm IDE.

5. Implementation:

CNN code snippets:

Below is the snippet for resizing the target image

```

import numpy as np
import keras
from keras.preprocessing import image
from keras.layers import MaxPooling2D, Convolution2D, Dropout, Flatten, Dense, Activation
from keras.models import Sequential, save_model
from keras.utils import np_utils
import os
import cv2
from sklearn.utils import shuffle

path = '/Users/adisekharrajuuppalapati/Downloads/crowdai'

leaf = os.listdir(path)
print(len(leaf), type(leaf))

print(leaf)

print(leaf[0][2:4])

x, y = [], []

for i in leaf:
    images = os.listdir(path+"/"+i)
    for j in images:
        img_path = path+"/"+i+"/"+j
        #Better method then cv2.imread
        img = image.load_img(img_path, target_size=(28, 28))
        img = image.img_to_array(img)
        #print(img.shape)
        #img = img.flatten()
        #img = img.reshape(1, 784)
        print(img.shape)
        img = img.reshape((28, 28))
        img = img/255.0
        x.append(img)
        y.append(int(i[2:4]))

```

Below is the code snippet for converting to binary class matrix using `to_categorical` and also splitting data into train and test.

```

y_data = np_utils.to_categorical(y_data)
print(y_data.shape)
num_classes = y_data.shape[1]
print(num_classes)

x_data, y_data = shuffle(x_data,y_data, random_state=0)

split = int(0.6*(x_data.shape[0]))

x_train = x_data[:split]
x_test = x_data[split:]
y_train = y_data[:split]
y_test = y_data[split:]

```

CNN architecture code snippets are in the 3.System architecture part.

Below is the code snippet of predicting the disease label using model.predict from keras.

```

import numpy as np
from keras.models import load_model
from keras.preprocessing import image
import matplotlib.pyplot as plt

model = load_model('Disease_detector_model.h5')

path = '/Users/adisekharajuuppalapati/PycharmProjects/plant_diseases/static/4e770559-e332-470e-aae3-c7ad0399dd20_FREC_Scab_3513.JPG'

img = plt.imread(path)
img = np.array(img)
img = img/255.0
plt.imshow(img)
plt.show()
img = image.load_img(path, target_size=(28,28))
img = image.img_to_array(img)
img = img/255.0
img = np.array(img)
plt.imshow(img)

img = img.reshape((1,28,28,3))

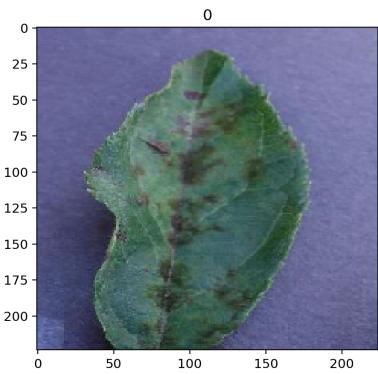
print(img.shape)

ans = model.predict(img).argmax()

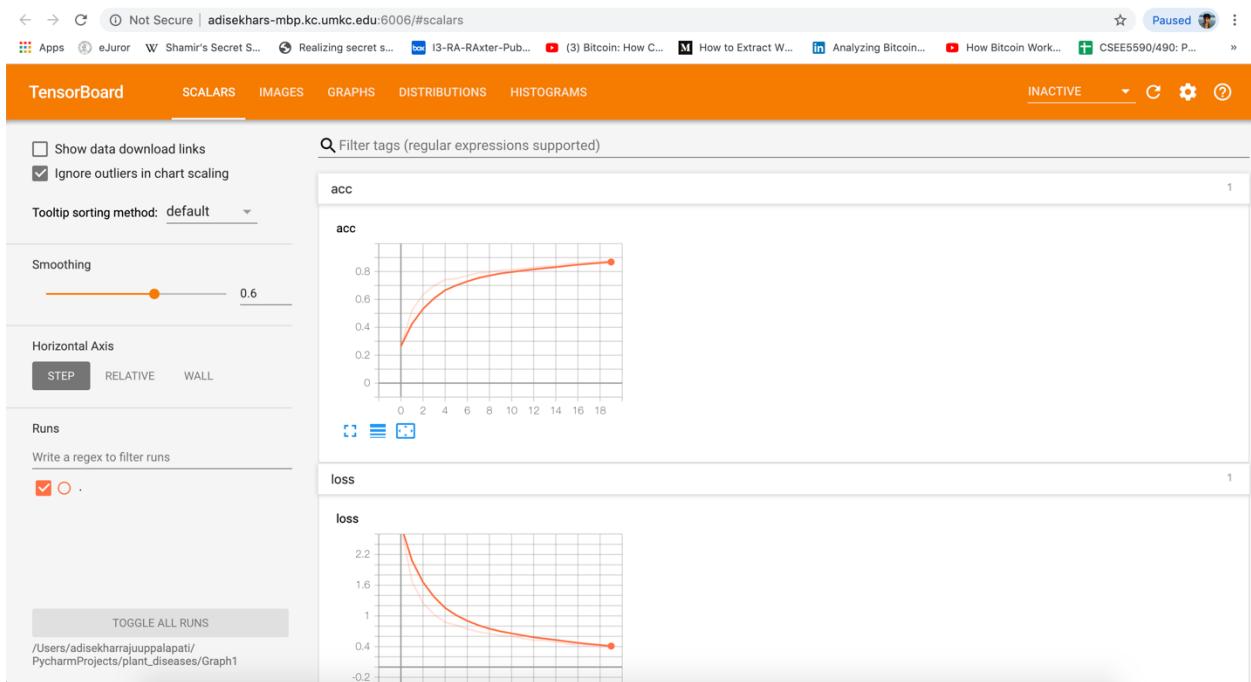
print(ans)
plt.title(ans)
plt.show()

```

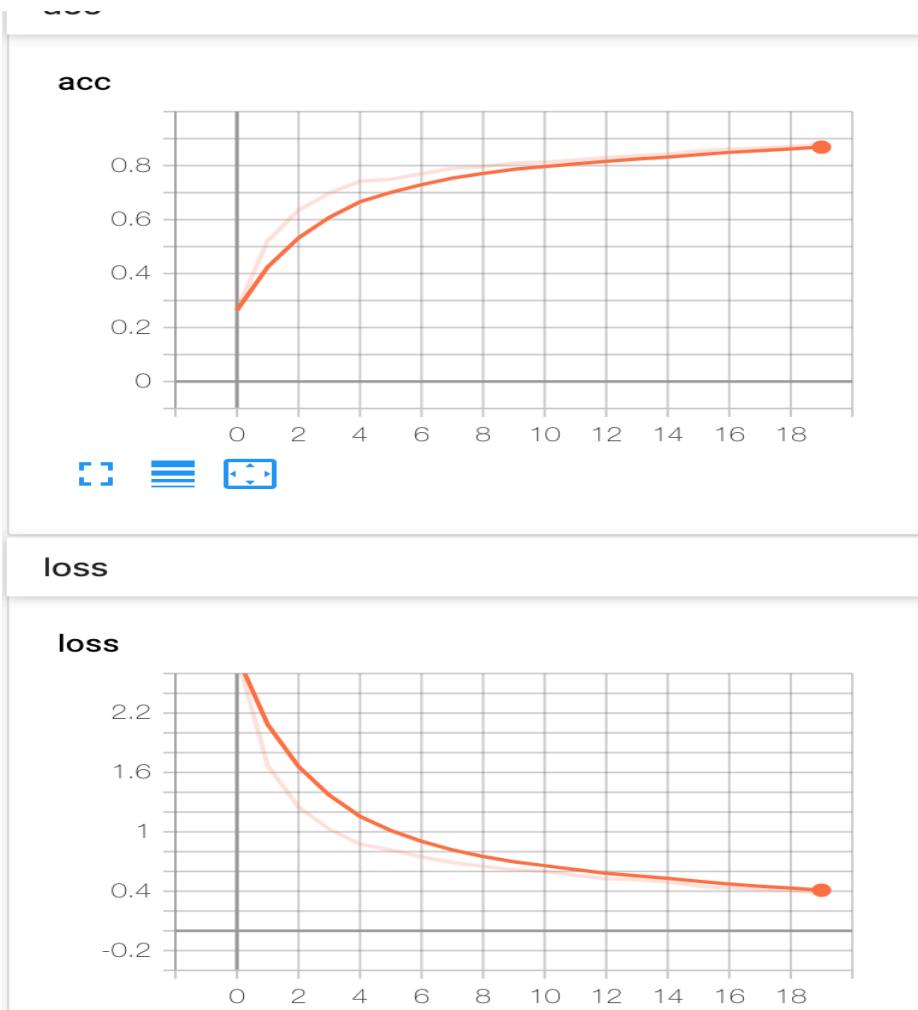
Below is the sample output for the predicted disease label ‘0’.



Below is the Tensorboard loss and accuracy graphs for CNN.

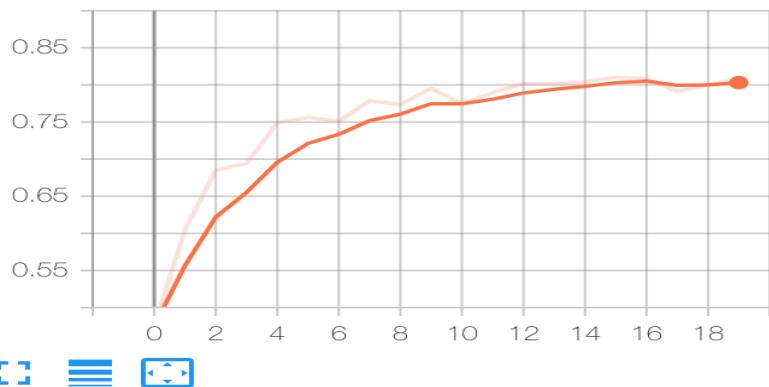


Below is the graph for training accuracy and loss for CNN.



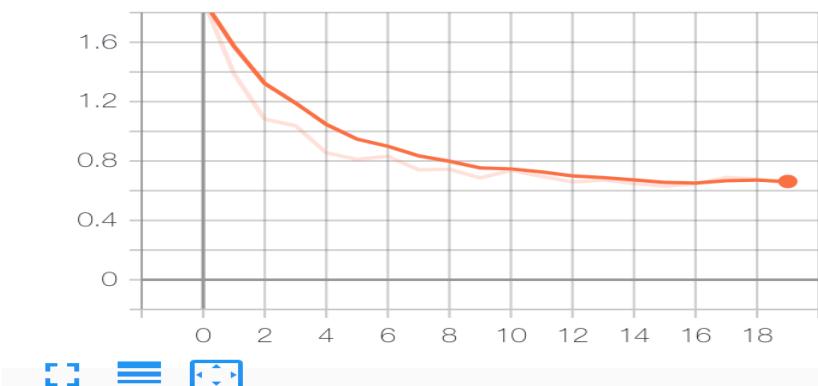
Below is the graph for validation accuracy and loss for CNN.

val_acc



val_loss

val_loss



Alexnet code snippets:

The image preprocessing steps are same for CNN and alexnet. Difference is in the architecture.

Those snippets for alexnet are in the 3.System architecture part.

Below are the code snippets while running alexnet architecture in pycharm.

The screenshot shows the PyCharm IDE interface. The top navigation bar has tabs for 'plant_diseases' and '1.py'. The left sidebar shows a project structure with files like 'plant_diseases.py', 'Graph.py', 'Graph1.py', 'static.py', 'templates.py', '1.py', and 'app.py'. The main code editor contains Python code for a neural network, including imports from tensorflow, definitions of layers (Convolution2D, MaxPooling2D, BatchNormalization, Flatten, Dense, Dropout), and training logic. Below the code is a 'Run' section showing the command 'python 1.py' and its output. The output includes a warning about TensorFlow's use of deprecated functions, the number of non-trainable parameters (23,980,752), and a series of training logs. Each log entry shows a step number (e.g., 128/13150, 256/13150, etc.), followed by 'ETA:', 'loss:', and 'acc:' values. The accuracy values range from 0.8906 to 0.9488.

```

1.py
Project
plant_diseases ~/PycharmProjects/plant_diseases
graph.py
Graph
events.out.tfevents.1559052568.Adicek
events.out.tfevents.1558142576.Adicek
events.out.tfevents.1558143042.Adicek
Graph1
static
templates
1.py
app.py

Run: 1
Non-trainable params: 23,980,752
WARNING:tensorflow:From /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 13150 samples, validate on 8767 samples
Epoch 1/25
128/13150 [=====] - ETA: 5:04 - loss: 0.2335 - acc: 0.8906
256/13150 [=====] - ETA: 5:10 - loss: 0.1700 - acc: 0.9219
384/13150 [=====] - ETA: 5:16 - loss: 0.1514 - acc: 0.9323
512/13150 [=====] - ETA: 4:53 - loss: 0.1669 - acc: 0.9277
640/13150 [=====] - ETA: 4:37 - loss: 0.1521 - acc: 0.9359
768/13150 [=====] - ETA: 4:26 - loss: 0.1778 - acc: 0.9284
896/13150 [=====] - ETA: 4:17 - loss: 0.1789 - acc: 0.9242
1024/13150 [=====] - ETA: 4:04 - loss: 0.1789 - acc: 0.9321
1152/13150 [=====] - ETA: 4:05 - loss: 0.1827 - acc: 0.9323
1280/13150 [=====] - ETA: 4:01 - loss: 0.1760 - acc: 0.9359
1408/13150 [=====] - ETA: 3:56 - loss: 0.1672 - acc: 0.9389
1536/13150 [=====] - ETA: 3:52 - loss: 0.1633 - acc: 0.9488
1664/13150 [=====] - ETA: 3:47 - loss: 0.1592 - acc: 0.9417
1792/13150 [=====] - ETA: 3:43 - loss: 0.1671 - acc: 0.9392
1920/13150 [=====] - ETA: 3:39 - loss: 0.1813 - acc: 0.9365
2048/13150 [=====] - ETA: 3:35 - loss: 0.1824 - acc: 0.9360
2176/13150 [=====] - ETA: 3:33 - loss: 0.1777 - acc: 0.9366
2304/13150 [=====] - ETA: 3:29 - loss: 0.1774 - acc: 0.9388
2432/13150 [=====] - ETA: 3:25 - loss: 0.1737 - acc: 0.9488
2560/13150 [=====] - ETA: 3:22 - loss: 0.1782 - acc: 0.9495
2688/13150 [=====] - ETA: 3:18 - loss: 0.1750 - acc: 0.9495
2816/13150 [=====] - ETA: 3:16 - loss: 0.1755 - acc: 0.9414
2944/13150 [=====] - ETA: 3:12 - loss: 0.1738 - acc: 0.9416
3072/13150 [=====] - ETA: 3:09 - loss: 0.1747 - acc: 0.9427
3200/13150 [=====] - ETA: 3:07 - loss: 0.1747 - acc: 0.9427

Run: 1
Event Log

```

As you can see from below snippet the validation accuracy improved from -inf to 0.97582.

This screenshot shows the PyCharm interface with the same project structure and code as the previous one. The run output shows the final validation accuracy. The validation accuracy (val_acc) improved from -inf to 0.97582, and the model was saved to 'best_weights.hdf5'. The log entries show the validation accuracy increasing from 0.1156 to 0.97582 across the 25 epochs.

```

1.py
Project
plant_diseases ~/PycharmProjects/plant_diseases
graph.py
Graph
events.out.tfevents.1559052568.Adicek
events.out.tfevents.1558142576.Adicek
events.out.tfevents.1558143042.Adicek
Graph1
static
templates
1.py
app.py

Run: 1
Epoch 00001: val_acc improved from -inf to 0.97582, saving model to best_weights.hdf5
Epoch 2/25
128/13150 [=====] - ETA: 3:16 - loss: 0.1156 - acc: 0.9453
256/13150 [=====] - ETA: 3:11 - loss: 0.1056 - acc: 0.9570
384/13150 [=====] - ETA: 3:13 - loss: 0.0937 - acc: 0.9661
512/13150 [=====] - ETA: 3:14 - loss: 0.0954 - acc: 0.9648
640/13150 [=====] - ETA: 3:14 - loss: 0.0970 - acc: 0.9656
768/13150 [=====] - ETA: 3:13 - loss: 0.0938 - acc: 0.9674
896/13150 [=====] - ETA: 3:12 - loss: 0.1069 - acc: 0.9598
1024/13150 [=====] - ETA: 3:10 - loss: 0.1185 - acc: 0.9561
1152/13150 [=====] - ETA: 3:08 - loss: 0.1174 - acc: 0.9566
1280/13150 [=====] - ETA: 3:06 - loss: 0.1133 - acc: 0.9586
1408/13150 [=====] - ETA: 3:03 - loss: 0.1186 - acc: 0.9553
1536/13150 [=====] - ETA: 3:01 - loss: 0.1179 - acc: 0.9555
1664/13150 [=====] - ETA: 2:59 - loss: 0.1159 - acc: 0.9555
1792/13150 [=====] - ETA: 2:50 - loss: 0.1182 - acc: 0.9554
1920/13150 [=====] - ETA: 2:49 - loss: 0.1151 - acc: 0.9562
2048/13150 [=====] - ETA: 2:57 - loss: 0.1147 - acc: 0.9570
2176/13150 [=====] - ETA: 2:55 - loss: 0.1092 - acc: 0.9596
2304/13150 [=====] - ETA: 2:53 - loss: 0.1064 - acc: 0.9605
2432/13150 [=====] - ETA: 2:53 - loss: 0.1045 - acc: 0.9613
2560/13150 [=====] - ETA: 2:51 - loss: 0.1063 - acc: 0.9605
2688/13150 [=====] - ETA: 2:49 - loss: 0.1051 - acc: 0.9613
2816/13150 [=====] - ETA: 2:47 - loss: 0.1035 - acc: 0.9616
2944/13150 [=====] - ETA: 2:45 - loss: 0.1034 - acc: 0.9620
3072/13150 [=====] - ETA: 2:43 - loss: 0.1043 - acc: 0.9622

Run: 1
Event Log

```

Below snippet is the final training accuracy, loss and validation loss, accuracy for alexnet.

This screenshot shows the PyCharm interface with the same project structure and code as the previous ones. The run output shows the final training and validation metrics. The validation accuracy (val_acc) did not improve from 0.97582. The process finished with exit code 0.

```

1.py
Project
plant_diseases ~/PycharmProjects/plant_diseases
graph.py
Graph
events.out.tfevents.1559052568.Adicek
events.out.tfevents.1558142576.Adicek
events.out.tfevents.1558143042.Adicek
Graph1
static
templates
1.py
app.py

Run: 1
Epoch 00025: val_acc did not improve from 0.97582
Process finished with exit code 0

Run: 1
Event Log

```

Below is the code snippet for predicting the disease label using alexnet.

```
import matplotlib.pyplot as plt
import matplotlib.image as im

model = load_model('disease_alexnet_detector.h5')

path = '/Users/adisekharrajuuppalapati/PycharmProjects/plant_diseases/static/9bd4db81-94ef-4a9e-a44a-e335fb372691_Matt.S CG 7786.JPG'

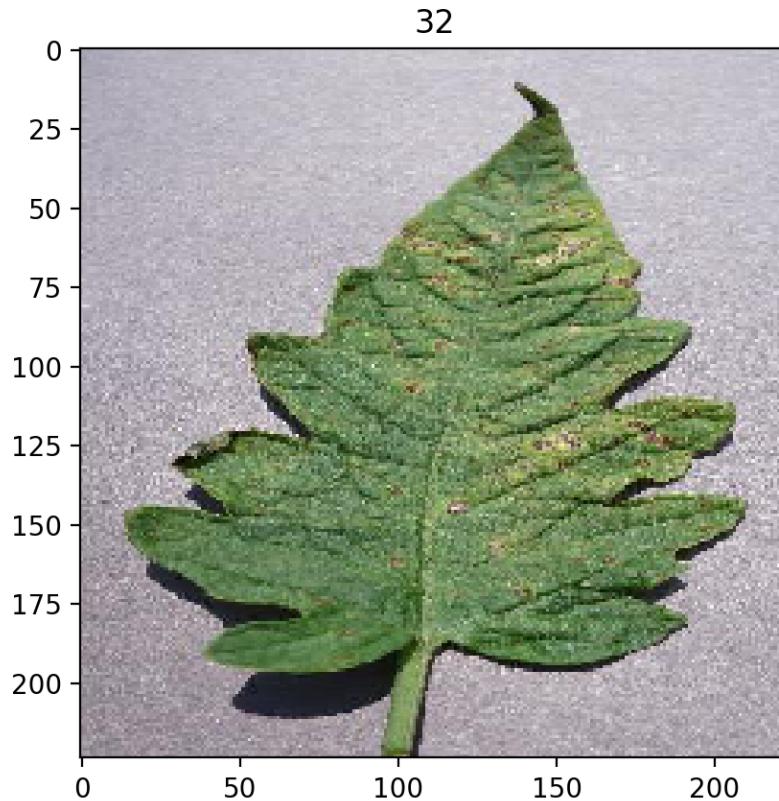
img = plt.imread(path)
img = np.array(img)
img = img / 255.0
plt.imshow(img)
plt.show()

img = image.load_img(path, target_size=(224,224))
img = image.img_to_array(img)
img = img / 255.0
img = np.array(img)
plt.imshow(img)

img = img.reshape((1,224,224,3))
ans = model.predict(img).argmax()
print(ans)

plt.title(ans)
plt.show()
```

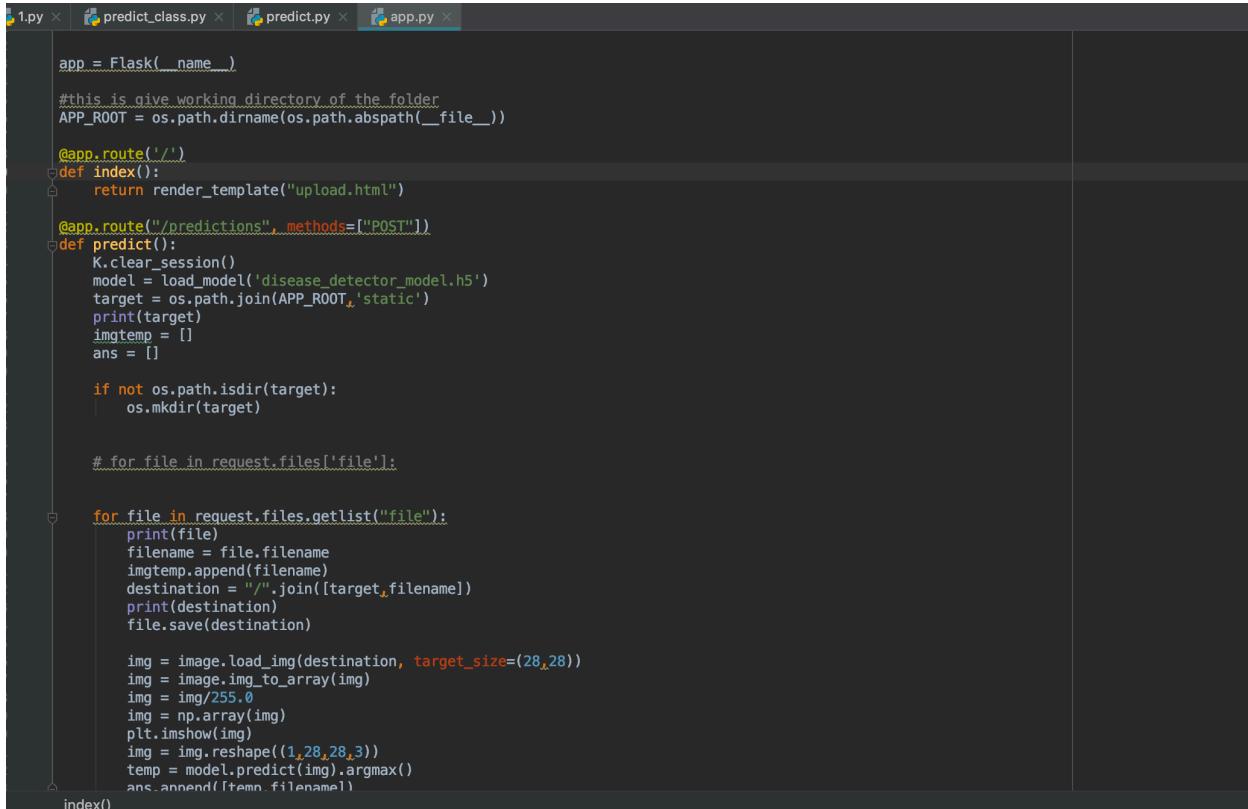
Below is the output for predicted disease label ‘32’.



Finally we develop our web application using flask with the saved model.

The Flask Framework looks for HTML files in a folder called **templates**. You **need to create a templates** folder and put all your HTML files in there.

Next is the creation .py file where we import flask module and creating a flask web server from the flask module. Below in the code snippet you can see **`_name_` means this current file**. In this case, it will be app.py. This current file will represent my web application.



```
app = Flask(__name__)

#this is give working directory of the folder
APP_ROOT = os.path.dirname(os.path.abspath(__file__))

@app.route('/')
def index():
    return render_template("upload.html")

@app.route("/predictions", methods=['POST'])
def predict():
    K.clear_session()
    model = load_model('disease_detector_model.h5')
    target = os.path.join(APP_ROOT, 'static')
    print(target)
    imgtemp = []
    ans = []

    if not os.path.isdir(target):
        os.mkdir(target)

    # for file in request.files['file']:
    for file in request.files.getlist("file"):
        print(file)
        filename = file.filename
        imgtemp.append(filename)
        destination = "/".join([target, filename])
        print(destination)
        file.save(destination)

        img = image.load_img(destination, target_size=(28,28))
        img = image.img_to_array(img)
        img = img/255.0
        img = np.array(img)
        plt.imshow(img)
        img = img.reshape((1,28,28,3))
        temp = model.predict(img).argmax()
        ans.append([temp, filename])

    index()
```

Below snippet is the templates folder.

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying a file tree for a project named 'plant_diseases'. The tree includes 'Graph', 'Graph1', 'static', 'templates' (containing 'layout.html', 'result.html', and 'upload.html'), and several Python files like '1.py', 'app.py', 'best_weights.hdf5', 'disease_alexnet_detector.h5', 'Disease_detector_model.h5', 'plant_disease_detection.py', 'predict.py', and 'predict_class.py'. The 'result.html' file is selected and shown in the main code editor. The code is a Jinja template that extends 'layout.html', contains logic to display predicted classes and images, and loops through image names.

```
{% extends 'layout.html' %}

{% block body %}

<!-- {{ans}} -->
<div class="container" >
<div style="margin-top: 50px">
{% for i in ans %}
<!-- {{i[1]}} -->
<div>

</div>
<div>
<h4 style="color:white;">Predicted class: {{i[0]}}</h4>
</div>

{% endfor %}
</div>
</div>
</div>



{{image_name}}
{% endfor %}
-->

{% endblock %}
```

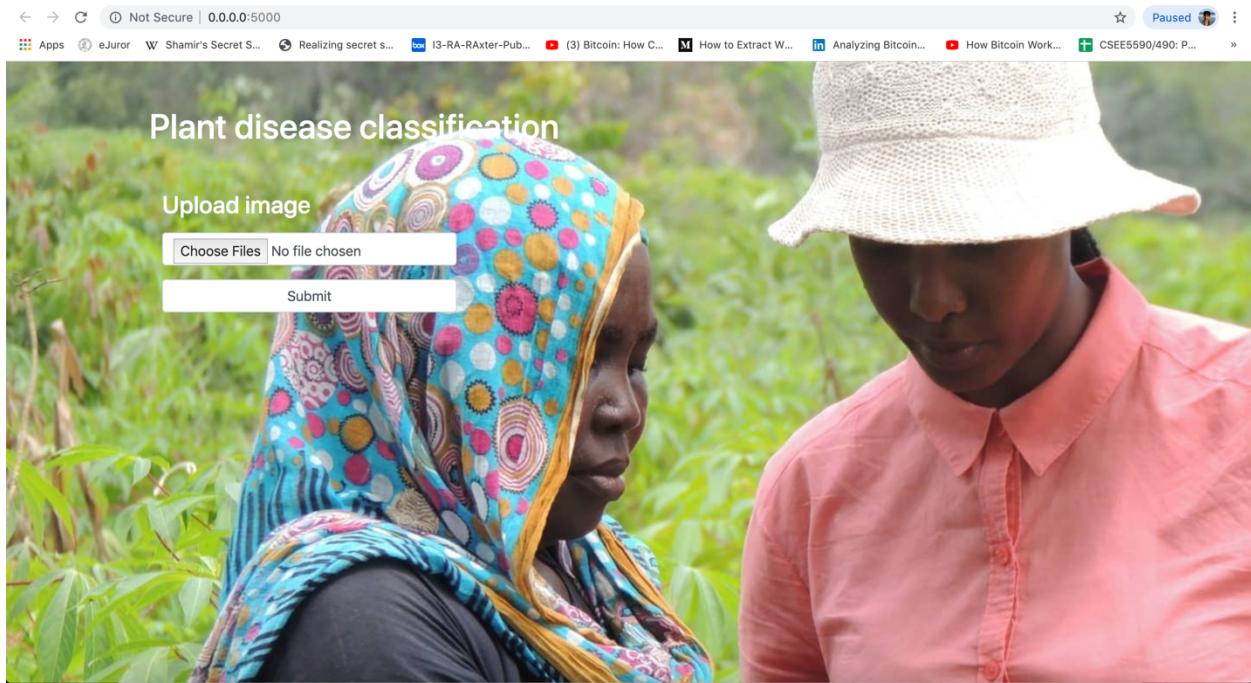
And next we run our flask web application by running app.py file.

Below is the snippet,

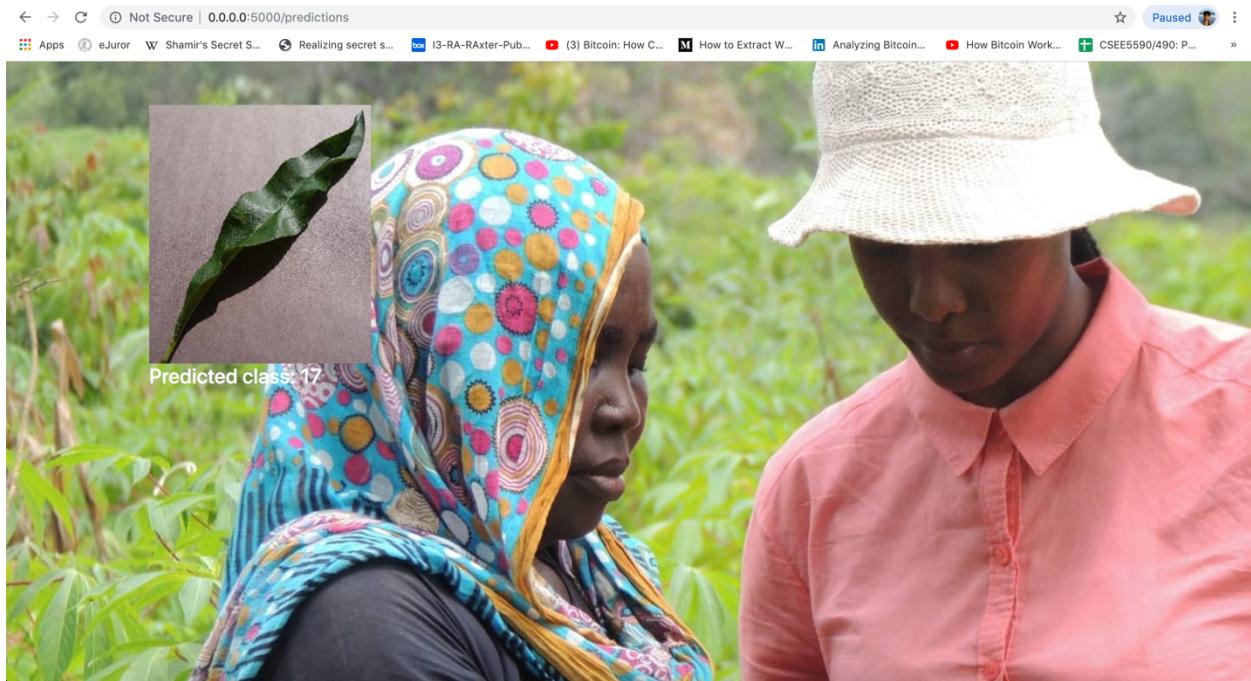
The screenshot shows a terminal window with the output of running the 'app.py' file. It indicates that the application is using the TensorFlow backend, serving on port 5000, and is in production mode. It also shows that a debugger is active with a PIN of 332-576-383.

```
Using TensorFlow backend.
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
Using TensorFlow backend.
* Debugger is active!
* Debugger PIN: 332-576-383
```

Below is the web site running on 0.0.0.0 with port number 5000,



Below is the predicted disease label '17'.



Below is the Github link available for the entire model building and web application:

https://github.com/sekhar1926/CSEE_5590_PYTHON_ICP/tree/master/team_8_project

6. Contribution:

Adhisekhar contribution is on CNN model building and model predictions.

Sudheer contribution is on Alexnet model building, computing best weights, and model predictions.

Mahesh contribution is on deploying the model to the web application using flask.

7. References:

1. <https://medium.com/coinmonks/understand-alexnet-in-just-3-minutes-with-hands-on-code-using-tensorflow-925d1e2e2f82>
2. <https://www.crowdai.org/challenges/plantvillage-disease-classification-challenge>
3. <https://medium.freecodecamp.org/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492>
4. <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
5. <https://www.mydatahack.com/building-alexnet-with-keras/>