An introduction:

Creating an AI web application that detects diseases in plants using FastAi which built on the top of Facebook's deep learning platform: PyTorch. According to the Food and Agriculture Organization of the United Nations (UN), transboundary plant pests and diseases affect food crops, causing significant losses to farmers and threatening food security.

Dataset used:

I used the "PlanVillage" dataset by S.P. Mohanty. This dataset contains an open access repository of images on plant health to enable the development of mobile disease diagnostics. The dataset contains 54, 309 images. The images span 14 crop species: Apple, Blueberry, Cherry, Grape, Orange, Peach, Bell Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, and Tomato. It contains images of 17 fundal diseases, 4 bacterial diseases, 2 moulds (oomycete) diseases, 2 viral diseases, and 1 disease caused by a mite. 12 crop species also have images of healthy leaves that are not visibly affected by a disease.
#We can further expand this dataset by the help of different **Agricultural Universities**
For training and validation:
I have used fastai which is built on top of Pytorch. Dataset consists of 38 disease classes from PlantVillage dataset and 1 background class from Stanford's open dataset of background images DAGS. 80% of the dataset is used for training and 20% for validation.
The code:

We can use the pre-trained resnet34 model or we will train the model.

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

(these are for the jupyter notebook.The first two lines will ensure automatic reload whenever you make any changes in the library. And the third line to show charts and graphs in the notebook.)

Import all the import libraries:

```
from fastai import *
from fastai.vision import *
from fastai.metrics import error_rate, accuracy
```

We will be adding path of the dataset (in my case it is in the root directory):

```
PATH_IMG = Path('PlantVillage/')
```

Batch size means we will feed x images at once to update parameters of our deep learning model. Set batch size to 64, if smaller GPU use 16 or 32 instead of 64.

bs = 64

ImageDataBunch

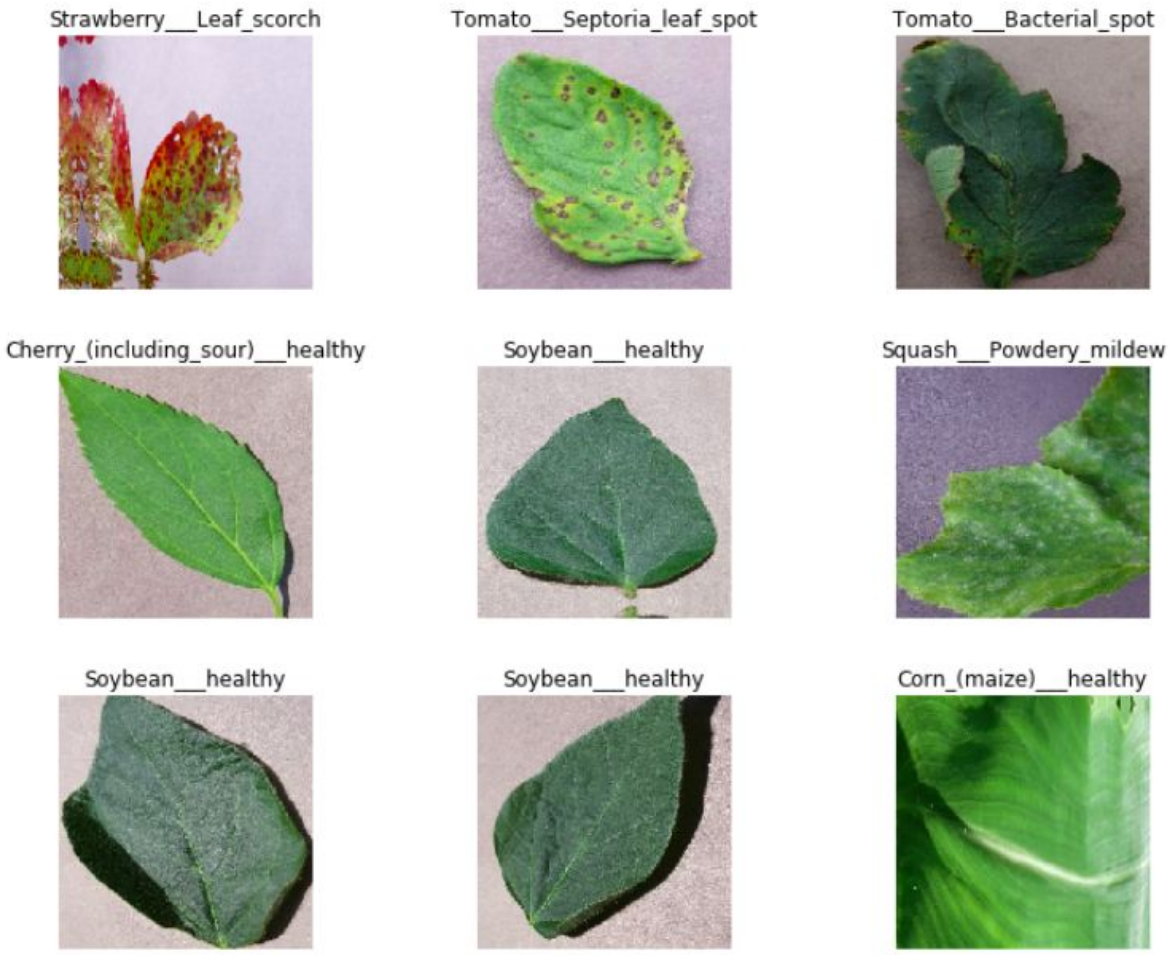
It is used to do classification based on images.
ImageDataBunch.from_folder gets the label names from the folder name automatically. fastai library has documentation to navigate through their library functions with live examples on how to use them. Once the data is loaded, we can also normalize the data by using .normalize to ImageNet parameters.

img_data = ImageDataBunch.from_folder(path=PATH_IMG, train='train', valid='val', ds_tfms=get_transforms(), size=224, bs=bs)
img_data.normalize(imagenet_stats)

- **path** The path of the images directory.
- **ds_tfms** the transformations that are needed for the image. This includes centering , cropping and zooming of the images.
- **size** the size to which the image is to be resized. This is usually a square image.This is done because of the limitation in the GPU that the GPU performs faster only when it has to do similar computations (such as matrix multiplication, addition and so on) on all the images.

To look at a random sample of images, we can use .show_batch() function ImageDataBunch class.

img_data.show_batch(rows=3, figsize=(10,8))



Print all the data classes present in the data. In total, we have images in 39 classes as mentioned above!

```
img_data.classes
'''
Output of img_data.classes:
['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Blueberry___healthy',
 'Cherry_(including_sour)___Powdery_mildew',
 'Cherry_(including_sour)___healthy',
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
```

'Corn_(maize)___Common_rust_',
'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy',
'Grape___Black_rot',
'Grape___Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)',
'Peach___Bacterial_spot',
'Peach___healthy',
'Pepper,_bell___Bacterial_spot',
'Pepper,_bell___healthy',
'Potato___Early_blight',
'Potato___Late_blight',
'Potato___healthy',
'Raspberry___healthy',
'Soybean___healthy',
'Squash___Powdery_mildew',
'Strawberry___Leaf_scorch',
'Strawberry___healthy',
'Tomato___Bacterial_spot',
'Tomato___Early_blight',
'Tomato___Late_blight',
'Tomato___Leaf_Mold',
'Tomato___Septoria_leaf_spot',
'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Target_Spot',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Tomato_mosaic_virus',
'Tomato___healthy',
'background']
'''

To create the transfer learning model we will need to use the function cnn_learner that takes the data, network and the metrics . The metrics is just used to print out how the training is performing.

```
model = cnn_learner(img_data, models.resnet34, metrics=[accuracy, error_rate])
```

We will train for 5 epochs.

model.fit_one_cycle(5)

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.275081	0.137157	0.956834	0.043166	03:43
1	0.146267	0.064265	0.978372	0.021628	03:28
2	0.097072	0.037028	0.987368	0.012632	03:28
3	0.060917	0.027360	0.990549	0.009451	03:27
4	0.042702	0.025219	0.991094	0.008906	03:28

As we can see above by just running five epochs with the default setting our accuracy for this fine-grained classification task is around ~99.10%.

Let's train it for two more epochs.

model.fit_one_cycle(2)

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.067196	0.033519	0.989004	0.010996	03:27
1	0.041814	0.024897	0.992003	0.007997	03:28

This time 99.2% Accuracy!

Now save the model with .save().

```
model.save('train_7_cycles')
```

We can also plot the confusion matrix and interpret the top four losses

```
interpret = ClassificationInterpretation.from_learner(model)
```

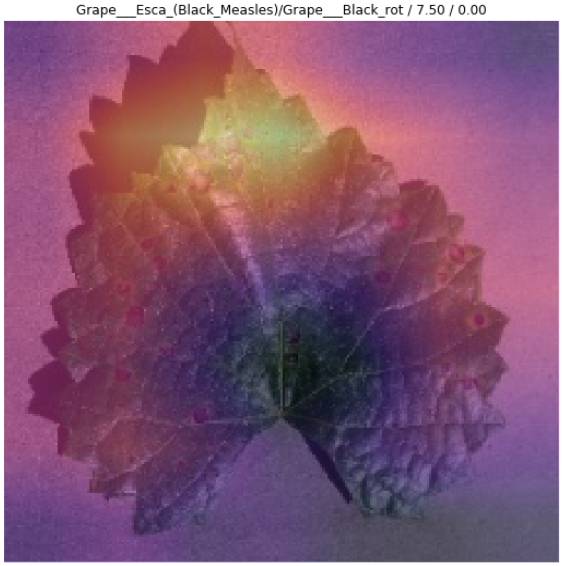
```
interpret.plot_top_losses(4, figsize=(20, 25))
```

prediction/actual/loss/probability

Tomato___Early_blight/Tomato___Late_blight / 9.26 / 0.00

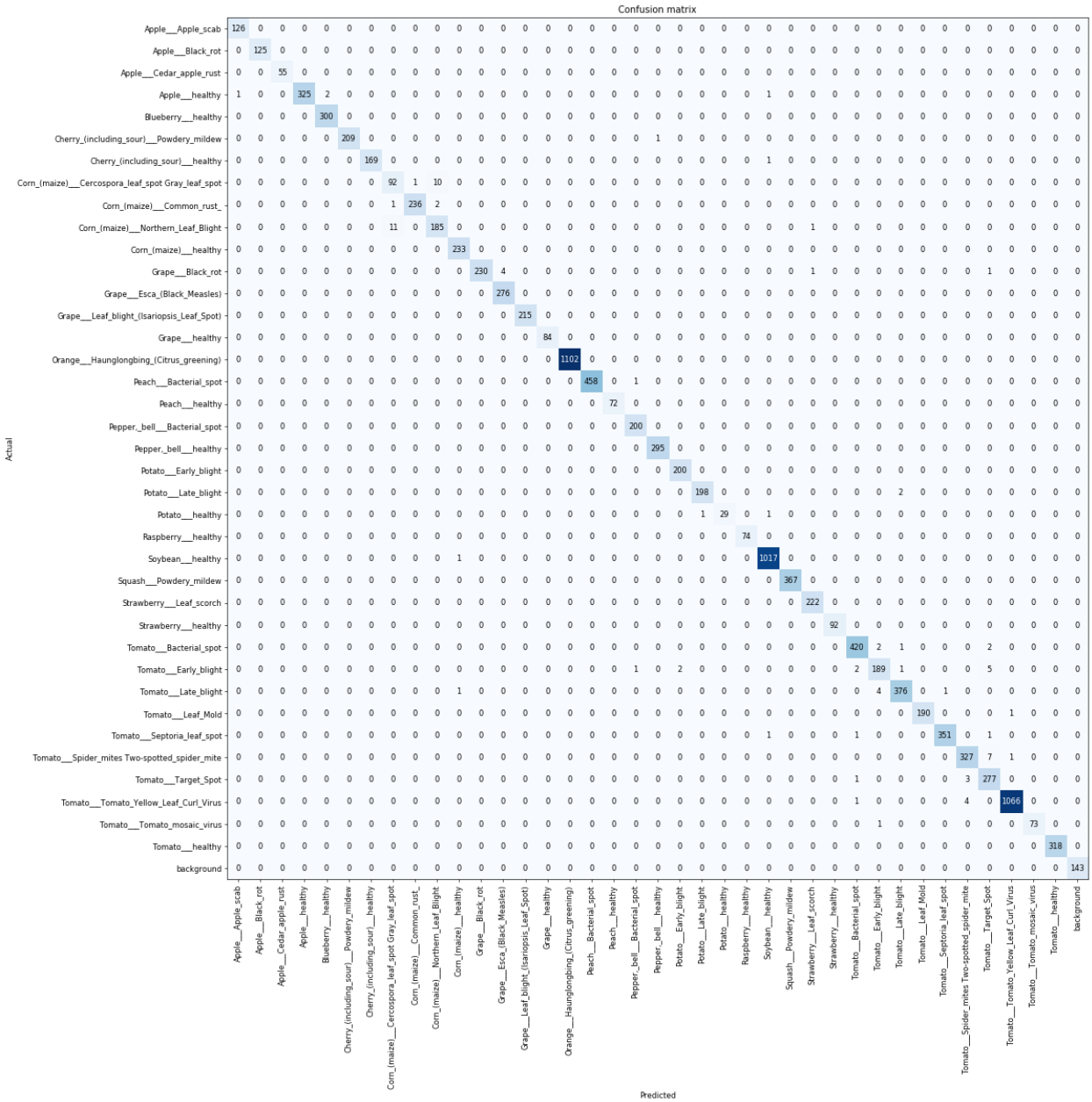
Soybean___healthy/Cherry_(including_sour)___healthy / 8.99 / 0.00

Grape___Esca_(Black_Measles)/Grape___Black_rot / 7.50 / 0.00



Tomato___Target_Spot/Tomato___Early_blight / 6.76 / 0.00

interpret.plot_confusion_matrix(figsize=(20,20), dpi=60)

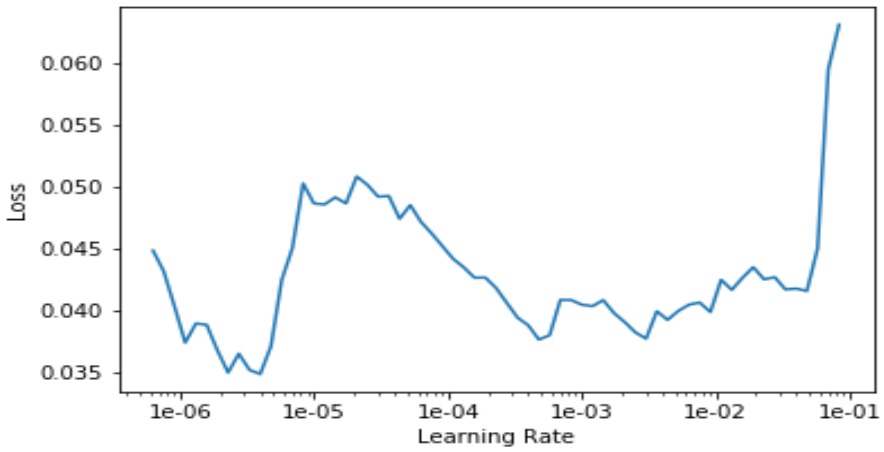


We use the lr_find method to find the optimum learning rate. Learning Rate is an important hyper-parameter to look for. We traditionally use α α to denote this parameter. If the Learning rate is too slow, we take more time to reach the most accurate result. If it is too high, we might not even end up reaching the accurate result. Learning rate Finder was the idea of automatically getting the magic number (which is near perfect), to get this optimum learning rate. This was introduced in last year's Fast AI course and continues to be useful.

```
model.lr_find()
```

After we run the finder, we plot the graph between loss and learning rate. We see a graph and typically choose a higher learning rate for which the loss is minimal. The higher learning rate makes sure that the machine ends up learning faster.

```
model.recorder.plot()
```



The *slice* is used to provide the learning rate wherein, we just provide the range of learning rates (its min and max). The learning rate is set gradually higher as we move from the earlier layer to the latest layers.

Let's unfreeze all the layers so that we can train the entire model using unfreeze() function.

After we run the finder, we plot the graph between loss and learning rate. We see a graph and typically choose a higher learning rate for which the loss is minimal. The higher learning rate makes sure that the machine ends up learning faster.

In [41]:
```
model.unfreeze()
model.fit_one_cycle(3, max_lr=slice(1e-03, 1e-02))
```

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.347306	0.843974	0.800527	0.199473	04:36
1	0.131069	0.062115	0.978644	0.021356	04:37
2	0.038473	0.020997	0.993457	0.006543	04:37

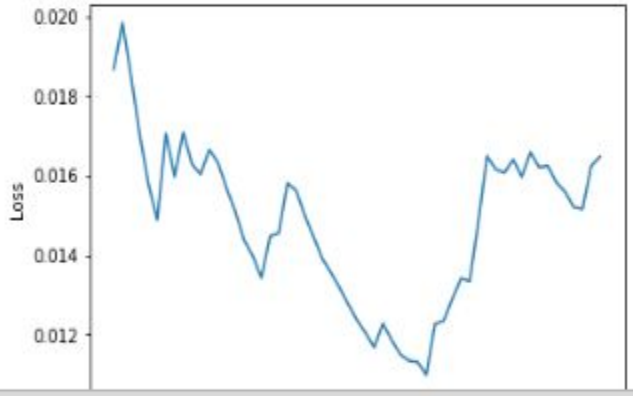
In [43]:
```
model.fit_one_cycle(5, max_lr=slice(1e-03, 1e-02))
```

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	0.124815	0.142209	0.957197	0.042803	04:37
1	0.144097	0.150233	0.959924	0.040076	04:37
2	0.093149	0.058316	0.981461	0.018539	04:36
3	0.041538	0.019202	0.995093	0.004907	04:36
4	0.018872	0.013773	0.996365	0.003635	04:36

In [44]:
```
model.save('train_lr_8_cycles')
```

In [45]:
```
model.freeze()
model.lr_find()
model.recorder.plot()
```

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the
graph.
```



Considering the fact that we are using a pre-trained model of *Resnet34*, we know for sure that our previous layers of this neural network would learn to detect the edges and the later layers would learn complicated shapes. We don't want to ruin out the earlier layers which presumably does a good job of detecting the edges. But would like to improve the model in narrowing down classifying the images.

So, we will set a lower learning rate for earlier layers and a higher one for the last layers.

The *slice* is used to provide the learning rate wherein, we just provide the range of learning rates (its min and max). The learning rate is set gradually higher as we move from the earlier layer to the latest layers.

Let's unfreeze all the layers so that we can train the entire model using unfreeze() function.

```
model.unfreeze()
```

```
model.fit_one_cycle(3, max_lr=slice(1e-03, 1e-02))
```

Accuracy close to 99.63%. Save the Model!

model.save('train_lr_8_cycles')

Freeze the model, find the learning rate, and fine-tune the model:

Save and load the model

```
In [49]: model.save('train_final5_cycles')

In [50]: model.load('train_final5_cycles')
```

Expot the model and use it for the app.

model.export('export_resnet34_model.pkl')

The Front end is made on HTML and CSS
The Backend is made on JavaScript and Flask

Don't put your models inside the models folder beforehand as these files can be large in size, and we want to keep our cloud deployment as light as possible and ask the app to download whatever file it needs.

The Nural Network Algorithm use and flowchat can be found in PPT.

The program was mainly inspired from
https://spj.sciencemag.org/plantphenomics/2019/9237136/

Frontend

The frontend uses basic knowledge of HTML and CSS

- **We have created button and applied an animation for the button**
- **We created and applied icon for the page**
- **We applied CSS and interface of the form.**
- **Created the basic interface of the page using HTML**

Backend

The backend was created using basic knowledge of Javascript and Flask

- **We created the synchronization between Flask (which used the model to provide output) , Javascript (which supported the interface between frontend and backend).**
- **Created form's backend using javascript.**
- **Provided the incoming of image and display of result after going through the model which was trained previously.**
- **Upload of the image as file.**

Working of the web-application

1. **Setting up the Virtual environment conda**
2. **Deploying the page on local host.**

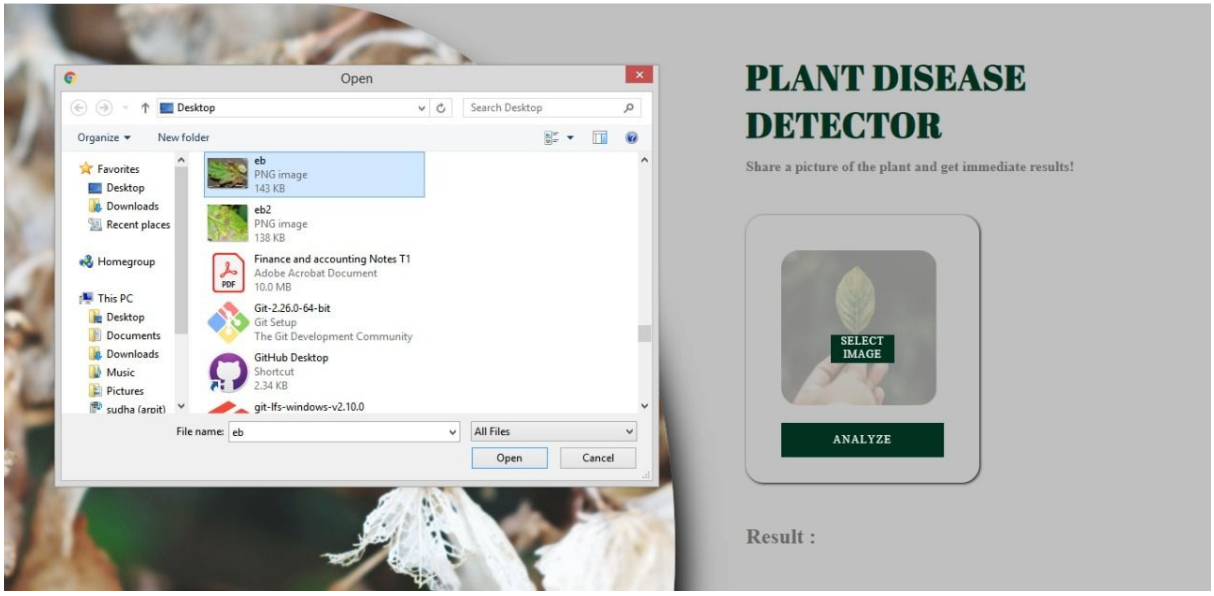
Local:

- It is recommended to set up the project inside a virtual environment to keep the dependencies separated.
  - Python
  - Conda
- Activate your virtual environment.
- Install dependencies by running `pip install -r requirements.txt`.
- Start up the server by running `python app/server.py serve`.
- Visit http://localhost:8080/ to explore and test.

3. **Open the page on local host**

4. Click on Select Image to upload the image.

5. Choose the image . The image will be sent to model and predicted.



6. Click on Analyze button

7. And Vollia !! You have result(Disease Found)