NORTH SOUTH UNIVERSITY

KrishiSheba

by

Adiba Sharif
1912611642
Al Sadat Rafi
1813118642
Hafiz Al Asad
1731821642
Md. Shamsur Rahman Khan
1911843642
Tashin Chowdhury
1911037642

Junior Project

Faculty Advisory:
Ms. Silvia Ahmed
Senior Lecturer
ECE Department
North South University
Fall 2021

Abstract

Machine Learning is a branch of computer science that focuses on using data and algorithms to imitate the human-like process of learning- there is a gradual increase of accuracy. In this project we were asked to use machine learning and image processing algorithms to find various stages of blight in plants. We used Pepper Bell data-set from Kaggle. Herein lies our final report.

Keywords: Machine Learning, Pattern Recognition, Classification, Supervised learning, Artificial Intelligence.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.

Secondly, to our advisor Ms. Silvia Ahmed for her kind support and advice in our work. She helped us whenever we needed help.

Thirdly, Jon Van Haaren and the whole judging panel of Machine Learning in Sports Analytics Conference 2015. Though our paper not accepted there, all the reviews they gave helped us a lot in our later works.

A special acknowledgement to our beloved institute, North South University for providing us with the opportunity to execute this project.

Finally, our deepest regards to our family and friends for their endless support, love and blessings. We would like to express our deepest gratitude and appreciation to all those who bestowed us the prospect to complete the report.

Table of Contents

Abstract i

Acknowledgment ii

Table of Contents iii

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Introduction . . . 2
- 1.2 Motivation . . . 2
- 1.3 Identifying The Issues . . . 2
- 1.4 Why Krishisheba . . . 3
- 1.5 Objective . . . 3
- 1.6 Related Work . . . 3

2 Related Work 4

3 Design 6
- 3.1 System Features . . . 7
- 3.2 Proposed Solution . . . 7
- 3.3 System Flowchart . . . 7
- 3.4 Front End . . . 8
  - 3.4.1 Initial appearance . . . 8
  - 3.4.2 After Prediction . . . 8
- 3.5 System Overview . . . 9
- 3.6 Software Components . . . 10
- 3.7 The AI . . . 10
  - 3.7.1 Entropy Calculation . . . 11
  - 3.7.2 Information Gain . . . 11
  - 3.7.3 Data Training . . . 11

4 Result and Discussion 12

5 Future Work 16

Bibliography 18
- 5.1 Appendices . . . 19

Activity(Codes) **20**

List of Figures

3.1 The flowchart 7
3.2 Website Interface: Before selecting file 8
3.3 Website Interface: After Prediction 9
3.4 Workflow 11

4.1 Random samples used in training 13
4.2 Graphing training and validation accuracy and loss 14

List of Tables

4.1 Progressing of loss and accuracy values as training progressed 15

Chapter 1

Introduction

1.1 Introduction

Machine learning is a sub-domain of computer science which evolved from the study of pattern recognition in data, and also from the computational learning theory in artificial intelligence. It is the first-class ticket to most interesting careers in data analytics today [3]. As data sources proliferate along with the computing power to process them, going straight to the data is one of the most straightforward ways to quickly gain insights and make predictions.
Machine Learning can be thought of as the study of a list of sub-problems, viz: decision making, clustering, classification, forecasting, deep-learning, inductive logic programming, support vector machines, reinforcement learning, similarity and metric learning, genetic algorithms, sparse dictionary learning, etc. Supervised learning, or classification is the machine learning task of inferring a function from a labeled data [4] . In Supervised learning, we have a training set, and a test set. The training and test set consists of a set of examples consisting of input and output vectors, and the goal of the supervised learning algorithm is to infer a function that maps the input vector to the output vector with minimal error. In an optimal scenario, a model trained on a set of examples will classify an unseen example in a correct fashion, which requires the model to generalize from the training set in a reasonable way. In layman's terms, supervised learning can be termed as the process of concept learning, where a brain is exposed to a set of inputs and result vectors and the brain learns the concept that relates said inputs to outputs. A wide array of supervised machine learning algorithms are available to the machine learning enthusiast, for example Neural Networks, Decision Trees, Support Vector Machines, Random Forest, Naïve Bayes Classifier, Bayes Net, Majority Classifier[4,7,8,9] etc., and they each have their own merits and demerits. There is no single algorithm that works for all cases, as merited by the No free lunch theorem [2]. In this project, we try and find patterns in a data-set, which is a sample of spots on leaves, and attempt to predict if a plant is affected by blight or not.

1.2 Motivation

While selecting the topic for our CSE299: Junior designing course our aim was to focus on solving he major social issues in our country that we face in our everyday lives. So, we wanted to come up with something that would make the life of the rural citizens: mainly the farmers make things a lot easier and more hassle-free. That's when we thought why not look for a solution for one of the most sever issue that farmers face in their daily lives. Using pesticides on plants: more importantly when to apply pesticides. To tackle the problem, we'd first have to come up with a way for the farmers to diagnose if a plant had a certain disease or not. Our aim with the project was to help farmers come to that decision.

1.3 Identifying The Issues

While we were brainstorming about the overuse of pesticides and its impact on the environment, we realized that one of the main reasons of pesticides being an issue is farmers misdiagnosing and overusing pesticides. While conversing on how to reduce

use of pesticides, we figured we could work out a way to properly identify plant diseases to address the situation. Our motto, prevention is better than cure.

1.4 Why Krishisheba

In developing countries like Bangladesh, people do not always have access to vets. Delayed identification and treatment of plant diseases usually cost them dearly. Among the other features that will be discussed later, one of the features of our app is to show Originality in its kind. There are no similar free products in the market that works like our app. The apps available in Bangladesh only allows people to calculate taxes and the like. Our app actually helps people make decisions in case a vet not reachable instantly.

1.5 Objective

The main objective of our project is to help farmers detect if a potato plant is affected by blight. Farmers will take a picture of a plant leaf they suspect of being affected by blight and upload it to our website or application. Using pictures of plant leaves and pre-trained model our website and app will predict the stage of blight a plant is in.

- Healthy: Shows no signs of being affected by blight
- Early Blight: Green with shades of yellow and has some black spots. Is affected by early blight.
- Late Blight: Has a lot of black spots. Is affected by late blight.

1.6 Related Work

Spectroscopy and machine learning have been combined to study late blight in plants [5]. Deep Learning has also been used in Image-Based Plant Disease Detection [1]. Enhanced Field-Based Detection of Potato Blight in Complex Backgrounds Using Deep Learning [6] is another excellent paper to look through while researching.

Chapter 2

Related Work

Spectroscopy and machine learning have been combined to study late blight in plants [5]. Deep Learning has also been used in Image-Based Plant Disease Detection [1]. Enhanced Field-Based Detection of Potato Blight in Complex Backgrounds Using Deep Learning [6] is another excellent paper to look through while researching.

Chapter 3

Design

3.1 System Features

- There will be no sign-ins needed in the website
- Farmers will be able to upload pictures of leaves
- Farmers will be get a label of the stage of Blight
- There is "Confidence" percentage that suggests how accurate the prediction is

3.2 Proposed Solution

We want to build an efficiently working and robust prediction system. Our system will have 2 types of interfaces:

- Website: Farmers can upload pictures to get predictions
- App: Farmers can take live pictures and upload those pictures to get predictions

3.3 System Flowchart

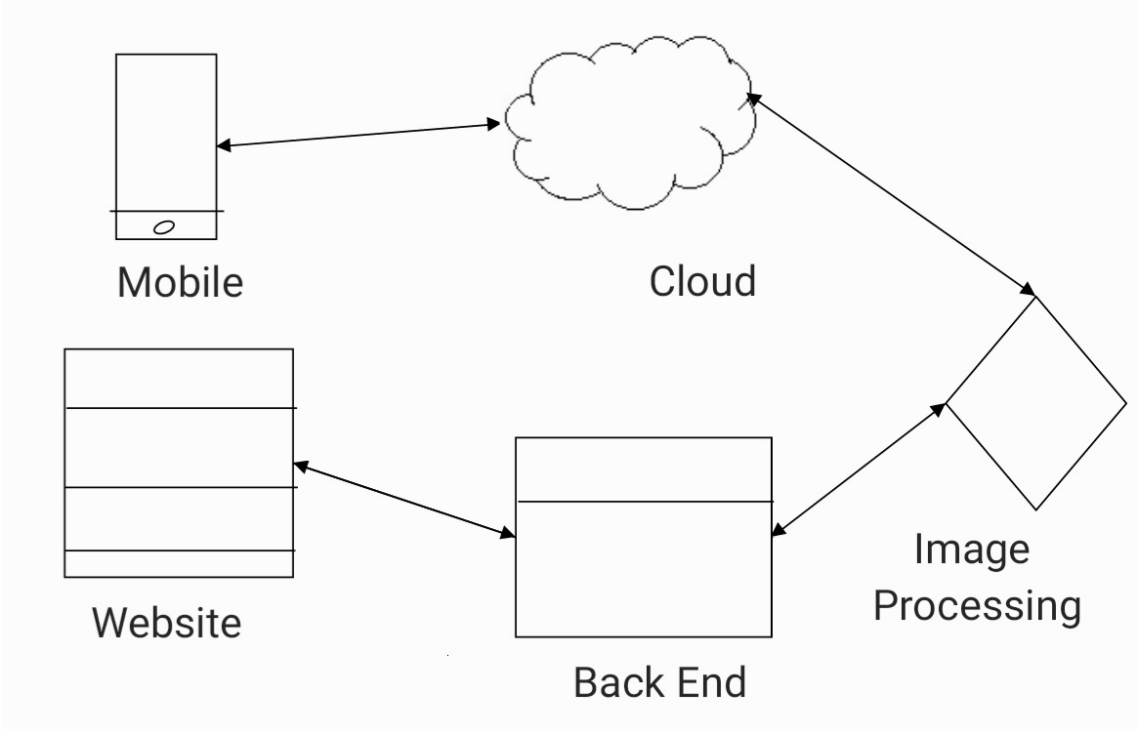

Figure 3.1: The flowchart

3.4 Front End

3.4.1 Initial appearance

The front end of the website is very simple and straightforward. There is no log-in feature. The Label and confidence fields just show some dashes.

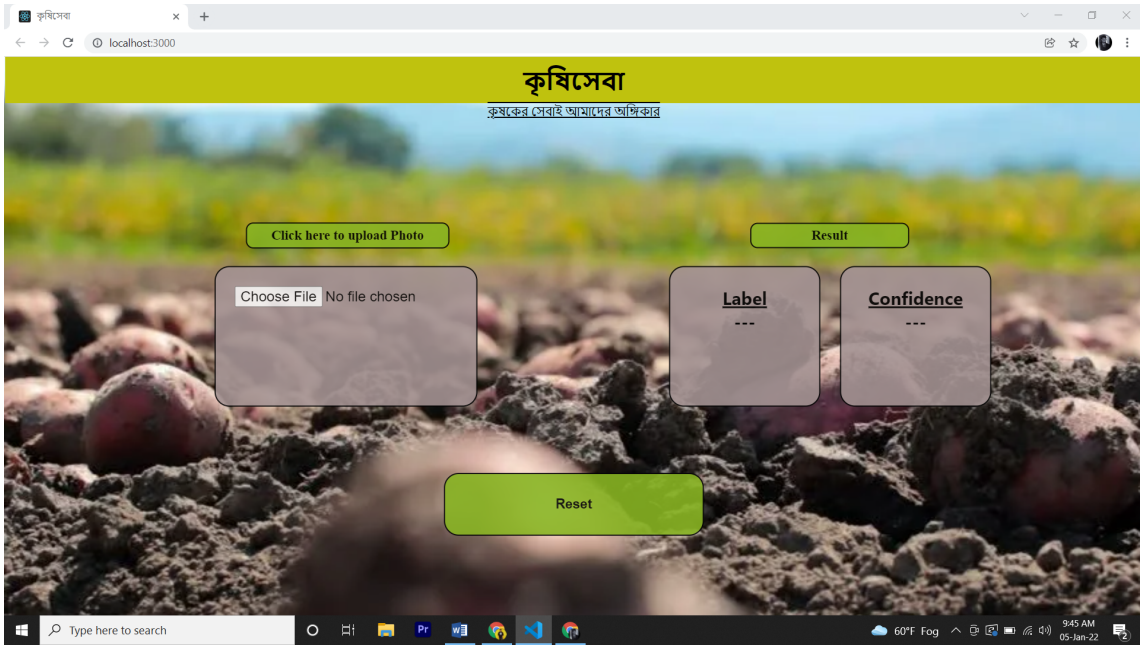

Figure 3.2: Website Interface: Before selecting file

3.4.2 After Prediction

After an image has been uploaded, the Label and confidence fields start showing values. The Label field can have three values, namely:

- Healthy: This represents leaves that are healthy
- Early Blight: These are leaves that show symptoms of early blight
- Late Blight: These leaves are heavily affected by blight

Figure 3.3: Website Interface: After Prediction

3.5 System Overview

Our key goal when building our app was to make the Graphical User Interface (GUI) basic and user-friendly so that the user could comprehend and operate it easily. The implementation of our project was broken down into four parts:-

1. The front end
    - ReactJS
    - HTML
    - CSS
    - JavaScript
2. The back end:
    - FastAPI
    - CORS Middleware
    - Numpy
    - TensorFlow
    - Pillow
    - io
    - TFlite
3. The AI model:
    - Python 3.9.7
    - tensorflow

- matplotlib

4. Documentation
    - LaTeX
    - MS Power Point
    - MS Excel

———-Insert detailed part about the front end and back end here———

3.6 Software Components

1. Visual Basic used as compiler for front end and back end
2. tensorflow
3. fastapi
4. uvicorn
5. python-multipart
6. pillow
7. tensorflow-serving-api
8. matplotlib
9. numpy
10. Anaconda had most of the ML packages built in
11. Overleaf to run LaTex

3.7 The AI

Decision tree algorithm is a very popular way to design a predictive modeling. Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Runs, Wickets and Run-Rate). Leaf node (e.g., Result) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data. Decision tree built on the calculation of Entropy and Information gain.

3.7.1 Entropy Calculation

Entropy is a measure of unpredictability and uncertainty of a data-set. Entropy is generally considered to determine how disordered a data-set is. The higher rate of entropy refers to the uncertainty and more information needed in these cases to improve the predictability. One outcome is very much certain when the entropy is zero.

$$Entropy(S) = \sum_{i=1}^{C} Pi \log_2 Pi \tag{3.1}$$

Where Pi is the proportion of instances in the data set that take the i-th value of target attribute, which has C different values. This probability measure give us the idea of how uncertain we are about the data. We use a log2 measure as this represents how many bit we would need in order to specify what the class is of a random instance.

3.7.2 Information Gain

Now we want quantitative way of splitting the data-set by using a particular attribute. We can use a measure called Information Gain, which calculates the reduction in entropy that would result in split-ting the data on an attribute, A. Information Gain is actually a procedure to select the particular attribute to be a decision node of a decision tree.

$$Gain(S, A) = Entropy(S) - \sum_{v \epsilon A} \frac{S_v}{S} Entropy(S_v) \tag{3.2}$$

where v is a value of A, Sv is the subset of instances of S where A takes the value v and S is the number of instances With the help of this node evaluation technique we can proceed recursively through the subset we create until leaf nodes have been reached throughout and all subsets are pure with zero entropy. This is how a decision tree algorithm works.

3.7.3 Data Training

After collecting the data we converted those data into an attributed relation file format (.arff) and then we have used Weka for classification. After classification using some algorithm we got some result and later we have analyzed those result. Here is the simple work flow chart given.

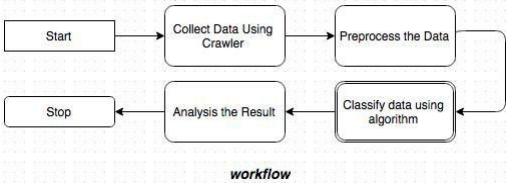

Figure 3.4: Workflow

Chapter 4

Result and Discussion

As we have divided our total model into two segment and we considered first segment for learning the image outcome. We have taken total 2152 for making our model using multiple linear regression and we have merged all the attributes from those images.

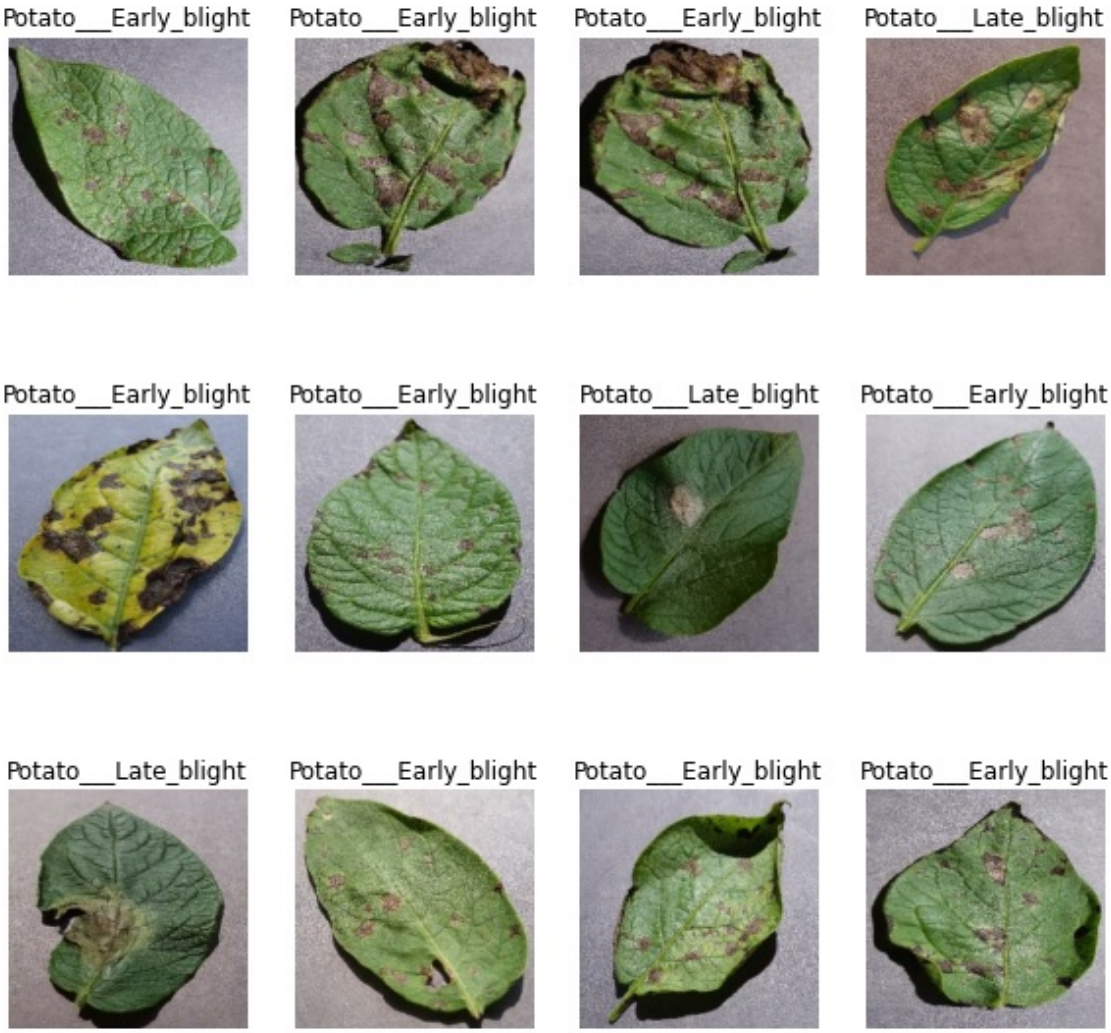

Figure 4.1: Random samples used in training

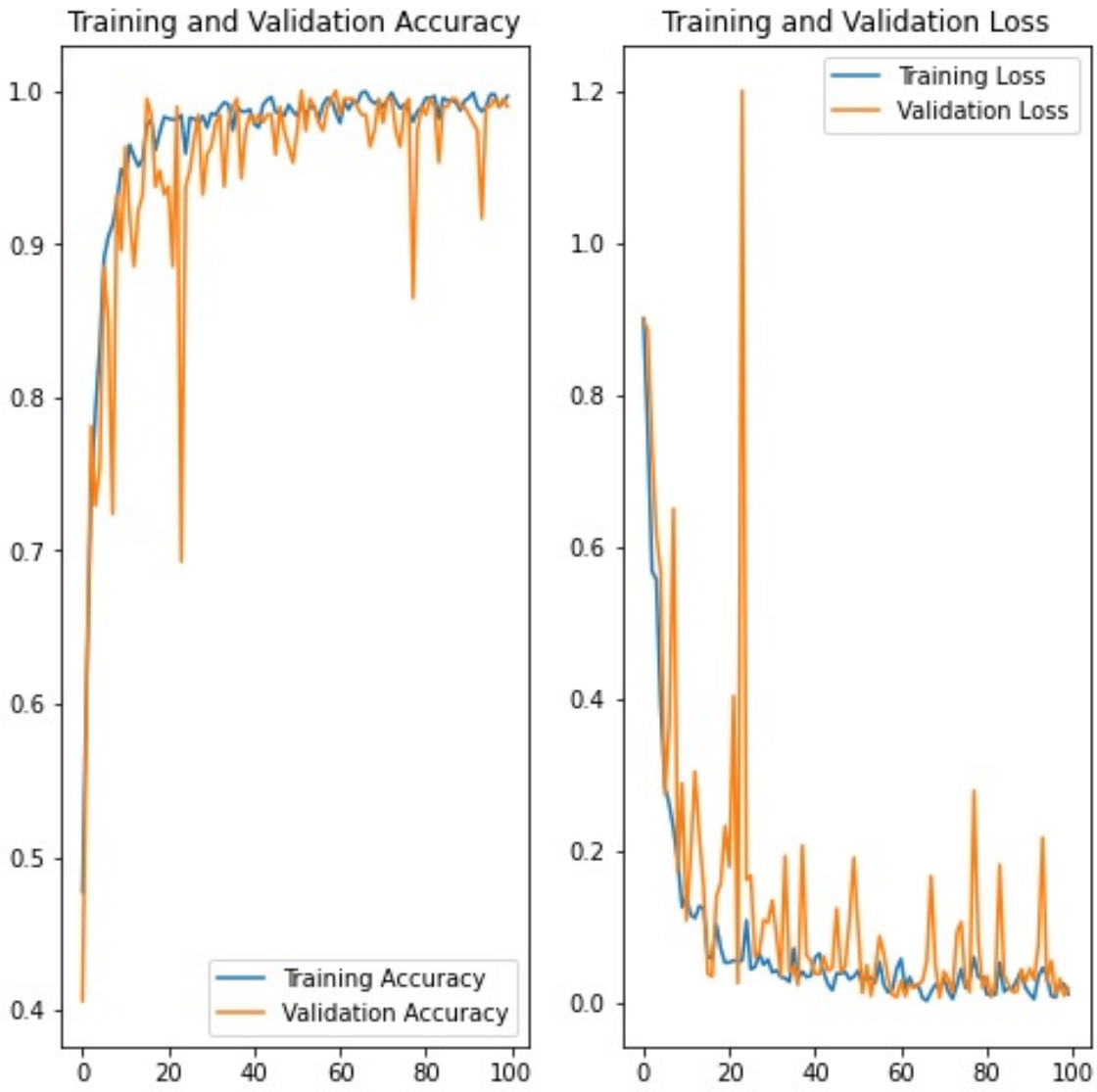

Figure 4.2: Graphing training and validation accuracy and loss

From the figure above we observe that as the number of epochs increase, the training and validation increases and the loss decreases.

Epoch	Accuracy	Loss
1/100	0.9016	0.4771
5/100	0.3928	0.8322
10/100	0.1259	0.9489
20/100	0.0528	0.9759
30/100	0.1259	0.9489
40/100	0.0356	0.9878
50/100	0.0344	0.9865
60/100	0.0451	0.9855
70/100	0.0180	0.9935
80/100	0.0340	0.9894
90/100	0.0197	0.9935
100/100	0.0111	0.9965

Table 4.1: Progressing of loss and accuracy values as training progressed

Chapter 5

Future Work

We highlighted some of the future features that we believe are necessary for our project here, but we haven't been able to execute them owing to a lack of time. The following are the features:

1. Create account options
2. Add paid professional consultancy
3. Cross-platform development is the ability to build and deliver apps that can run across multiple device platforms, such as iOS, Android, and the Universal Windows Platform.
4. Predict varieties of diseases
5. Make predictions about more types of plants
6. Work with more data to make even better predictions

Bibliography

[1] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 1419, 2016.

[2] J.-m. Kwon, K.-H. Jeon, H. M. Kim, *et al.*, "Deep-learning-based risk stratification for mortality of patients with acute myocardial infarction," *PloS one*, vol. 14, no. 10, e0224502, 2019.

[3] S. Chittora, "Machine learning for high school students," 2020.

[4] P. Garg *et al.*, "Multiple organ failure detection using machine learning," 2020.

[5] K. M. Gold, P. A. Townsend, I. Herrmann, and A. J. Gevens, "Investigating potato late blight physiological differences across potato cultivars with spectroscopy and machine learning," *Plant Science*, vol. 295, p. 110 316, 2020.

[6] J. Johnson, G. Sharma, S. Srinivasan, *et al.*, "Enhanced field-based detection of potato blight in complex backgrounds using deep learning," *Plant Phenomics*, vol. 2021, 2021.

5.1 Appendices

app.js

```
import React, { useState, useEffect } from 'react';
import './App.css';
//import {AiOutlinePicture} from 'react-icons/ai';
import { getPredictionData} from './file.js';

function App(){

    const [result, setResult] = useState('');


const uploadImage = async(files) => {

    console.log(files[0])

    const data = await (getPredictionData(files[0]))
   setResult(data);
   console.log(data);
}

const reset = () =>{
    let inputs = document.querySelectorAll('input');
    inputs.forEach(input => input.value = '')
    setResult('')
}

 return(
 <div>
    <div>
        <div className="App">
             <h1 className="header">                                    </h1>
         </div>
    </div>
    <div>
    <div className="motto">

    </div>
    </div>
    <div className='photo'>
        Upload Photo
    </div>
    <div className='result'>
        Result
    </div>
    <div className='label'>
       <div> <u>Label </u> <br/>{result.class==null?'---':(result.class)}
       </div>
       <div className='confidence'>
```

```
      <div> <u> Confidence </u> <br/>{result.confidence==null?'---':parse
    </div>
     <div className="image">

    <input
        type="file"
        className='choose'
        onChange={(e) => {uploadImage(e.target.files)
    }
}
/>
</div>
<div>
    <button onClick={reset} className='clear'> Reset  </button>
 </div>
 </div>


 )

}
export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a functio
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vital
reportWebVitals();
```

main-tf-serving.py

```
from fastapi import FastAPI, File, UploadFile
from fastapi.middleware.cors import CORSMiddleware
import uvicorn
import numpy as np
from io import BytesIO
from PIL import Image
import tensorflow as tf
import requests

app = FastAPI()

origins = [
    "http://localhost",
    "http://localhost:3000",
]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

endpoint = "http://localhost:8501/v1/models/potatoes_model:predict"

CLASS_NAMES = ["Early Blight", "Late Blight", "Healthy"]

@app.get("/ping")
async def ping():
    return "Hello, I am alive"

def read_file_as_image(data) -> np.ndarray:
    image = np.array(Image.open(BytesIO(data)))
    return image

@app.post("/predict")
async def predict(
    file: UploadFile = File(...)
):
    image = read_file_as_image(await file.read())
    img_batch = np.expand_dims(image, 0)

    json_data = {
        "instances": img_batch.tolist()
    }

    response = requests.post(endpoint, json=json_data)
```

```
    prediction = np.array(response.json()["predictions"][0])

    predicted_class = CLASS_NAMES[np.argmax(prediction)]
    confidence = np.max(prediction)

    return {
        "class": predicted_class,
        "confidence": float(confidence)
    }

if __name__ == "__main__":
    uvicorn.run(app, host='localhost', port=8000)
```

main.py

```
from fastapi import FastAPI, File, UploadFile
from fastapi.middleware.cors import CORSMiddleware
import uvicorn
import numpy as np
from io import BytesIO
from PIL import Image
import tensorflow as tf

app = FastAPI()

origins = [
    "http://localhost",
    "http://localhost:3000",
]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

MODEL = tf.keras.models.load_model("E:/CSE 299/project299/cse299.5/saved_

CLASS_NAMES = ["Early Blight", "Late Blight", "Healthy"]

@app.get("/ping")
async def ping():
    return "Hello, I am alive"

def read_file_as_image(data) -> np.ndarray:
    image = np.array(Image.open(BytesIO(data)))
    return image

@app.post("/predict")
async def predict(
    file: UploadFile = File(...)
):
    image = read_file_as_image(await file.read())
    img_batch = np.expand_dims(image, 0)

    predictions = MODEL.predict(img_batch)

    predicted_class = CLASS_NAMES[np.argmax(predictions[0])]
    confidence = np.max(predictions[0])*87.2
    return {
        'class': predicted_class,
```

```
        'confidence': float(confidence)
    }

if __name__ == "__main__":
    uvicorn.run(app, host='localhost', port=8000)
```

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt

IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS = 100

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    seed=123,
    shuffle = True,
    image_size = (IMAGE_SIZE,IMAGE_SIZE),
    batch_size = BATCH_SIZE
)

class_names = dataset.class_names
class_names

for image_batch, labels_batch in dataset.take(1):
    #print(image_batch.shape)
    #print(label_batch.numpy())

#the last 3 in the O/P is the RGB channel
#the 2nd set of number is from Out[6] where early blight is 0, late bligh

    #print(image_batch[0].numpy())                                  #converting t
                                                                    #print 1st im
    #print(image_batch[0].shape)                                    #to get the s
    plt.figure(figsize = (10,10))                                   # to create s
    for image_batch, labels_batch in dataset.take(1):
        for i in range(12):
#to display 12 images of the batch
            ax = plt.subplot(3,4, i+1)
#without this line it will show 1 img, so to show all the 12 images we ar
            plt.imshow(image_batch[i].numpy().astype("uint8"))
#to visualize the image, we are using matplotlib, so when we type "plt.im
                                                                    #the .astype(

            plt.title(class_names[labels_batch[i]])
#to add name of the image we use label_batch[0] but this will show numbe
                                                                    #.title is us
            plt.axis("off")
# it is used to hide the x and y axis
```

```
    train_size = 0.8
len(dataset)*train_size

train_ds = dataset.take(54)
len(train_ds)

test_ds = dataset.skip(54)
len(test_ds)

val_size = 0.1  #validation
len(dataset)*val_size

val_ds = test_ds.take(6)
len(val_ds)

test_ds = test_ds.skip(6)
len(test_ds)

#we just split our dataset into validation dataset and train dataset

def get_dataset_partitions_tf(ds,train_split = 0.8, val_split = 0.1, test
    assert (train_split + test_split +val_split ) == 1
    # this function takes the dataset and split ration that is 80% traini

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed = 12)

    train_size = int(train_split * ds_size) # train size in integer
    val_size = int (val_split * ds_size) #validation size in integer

    train_ds = ds.take(train_size)

    val_ds = ds.skip(train_size).take(val_size)  # validation dataset
    test_ds = ds.skip(train_size).skip(val_size) # test dataset

    return train_ds, val_ds, test_ds


train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

len(train_ds)

len(val_ds)

len(test_ds)
```

```
trian_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTU
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTO
#prefetch() - when cpu is loading batch -2 the gpu willl load batch 1 at


resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE), #
    layers.experimental.preprocessing.Rescaling(1.0/255),   # this will s

])
# this resize_and_rescale layer will go to the model and when we train ou

data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical
    layers.experimental.preprocessing.RandomRotation(0.2),
])
#this data_augmentation layer will help to understand a rotated img

train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE) #NEW

#CNN

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])
```

```
model.build(input_shape = input_shape)

model.summary()

 # in deeplearing we define the neural network architecture first, then c
model.compile(
    optimizer = 'adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
    metrics=['accuracy']
)


history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=100,
)

scores = model.evaluate(test_ds)

scores

history

history.params


history.history.keys()

type(history.history['loss'])

len(history.history['loss'])

history.history['loss'][:5] # show loss for first 5 epochs

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']


#plotting the traing and plotting accuracy

#accuracy chart
plt.figure(figsize=(8, 8))
```

```
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label = 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')
plt.title('Training and Validation Accuracy')

#loss chart
plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label = 'Training Loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')
plt.title('Training and Validation Loss')


#make a prediction
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    #plt.imshow(images_batch[0].numpy().astype('uint8'))
    #print(images_batch[0].numpy().astype('uint8'))
    #print will show as 3D rgb formate
    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("First image to predict")
    plt.imshow(first_image)
    print("First image's actual label: ", class_names[first_label])

    #lets do some prediction since model is complete
    batch_prediction = model.predict(images_batch)
    print("Predicted Label: ",class_names[np.argmax(batch_prediction[0])]


def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy
    img_array = tf.expand_dims(img_array, 0) # create a batch

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


plt.figure(figsize = (15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3, i+1)
```

```
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class

        plt.axis("off")


import os
#model_version = 1
#model.save_weights(f"../models/{model_version}")
#model.save(f"../models/{model_version}")

#model_version=max([int(i) for i in os.listdir("E:/CSE 299/project299/cse
#model.save("E:/CSE 299/project299/cse299.5/models/{model_version}")

# Setup base model and freeze its layers (this will extract features)

model_version=max([int(i) for i in os.listdir("../models") + [0]])+1
model.save(f"../models/{model_version}")


model.save("../potatoes.h5")
```