AN INTERNSHIP PROJECT

ON

POTATO DISEASE CLASSIFICATION

Team Members :-

Rishi Shankar
Shambhavi
Chukka S V Pavani
Vajrala Saveera Reddy

BRAIN O VISION

Brain O Vision Solutions India Pvt. Ltd...

ACKNOWLEDGMENT

We would like to express our sincere gratitude to **Mr. Ganesh Nagu Doddi (Founder and CEO, Brain O Vision solutions India Pvt Ltd.)** for providing me an opportunity to do my internship for this semester. We would also like to thank my mentor **Mr. Nagoor Shaik (Technical Lead, Brain O Vision Solutions India Pvt Ltd.)** for his constant support throughout this project journey and also like to thank my team member for their support during the period of our project work.

ABSTRACT

Potato is one of the most extensively consumed staple foods, ranking fourth on the global food pyramid. Furthermore, due to the global pandemic coronavirus, global potato consumption is expanding dramatically. Potato diseases, on the other hand, are the primary cause of harvest quality and quantity decline. Plant conditions will be dramatically worsened by incorrect disease classification and late identification due to which Potato farmers are experiencing significant financial losses due to numerous diseases. If farmers can identify these diseases early and treat them appropriately, they can save a lot of money. Leaf conditions can help identify various illnesses in potato plants. In this project, with the help of convolution neural network we tried to classify a potato plant by its leaf whether it is healthy or not. Mainly there are three classes Late Blight, Early Blight and Healthy. According to the confidence each potato leaf has been classified. A simple CNN architecture is used to train and test the model. Main objective was to create a web which will take leaf image as input and will classify according to the trained model.

TABLE OF CONTENT

INTRODUCTION

1.1 Introduction

As we know, potato is one of the most extensively consumed crops. If we want that the crop production continues well, then it is very important to identify the reasons which may cause problems in crop production. Phytophthora infestans and Alternaria solani are two pathogens that cause significant production loss in potato crop, causing diseases named late blight and early blight respectively. If somehow, we can detect these diseases early, then it will allow us for the implementation of preventive measures as well as the reduction of financial and yield losses. The most common method for detecting and identifying plant diseases in recent decades has been expert naked eye monitoring. However, in many circumstances, this strategy is impractical due to scarcity of experts on farms in remote places and long processing times. As a result, the use of image processing technologies has proven to be an excellent way for continuously checking plant health and early disease diagnosis. Hence, the objective of this project is to create classification methodology using simple convolutional neural network to detect disease by giving input as potato leaf.

1.2 Problem Statement

Project statement is to classify whether a plant is healthy or not by using an image of a potato leaf. It is a multi-class classification problem.
Following are the three classes:

- Late Blight- Potato leaves are more deteriorated.
- Early Blight- Potato leaves are in the early stage of disease.
- Healthy- Leaves are healthy.

1.3 Objective

As we know that Indian economy has a huge dependency on agriculture sector. Agriculture sector contributes 18% to India's GDP and about 60% of Indian population work in this sector. So, there is need of new technologies to help in growth of crop production. If we can detect diseases early, we can apply various methods to prevent disease and ultimately production of crops will increase as well as revenue will also increase. If we try to monitor with our naked eyes then it will be very time consuming and it also requires expertise in the field. So, we can apply image processing techniques to correctly classify plant whether it is healthy or not which will save time as well as resources. So, the objective of this project is to use CNN for image classification and create a web application which will take an image as an input and will classify it according to the saved model.

1.4 Methodology

This problem is an image classification problem. We are using convolutional neural network to train our model as it provides better resource utilization and better accuracy as compare to normal neural networks. It normally contains sequentially number of convolution layers, max pooling layers, activation function followed by normal dense layers. These extra layers provide better accuracy as compare to normal dense layers. ReLU activation function has been used and also adam optimizer has been used as it provides both dynamically changing learning rate and exponential weighted average concept to reduce noise.

<u>LITERATURE REVIEW</u>

Various algorithms have been developed to classify a crop disease such as ultrasound, X-ray and RGB imaging [2]. Another method was proposed by Macedo-Cruz [3]. In this method he evaluated damage caused by frost in oat crops. He computed L* a* b* color space from RGB color space. They also applied three distinct thresholding techniques. Another technique was proposed by Yao in [4]. Their objective was to classify whether rice is healthy or infected. In this approach they used segmentation. They separated infected parts using segmentation. Various characteristics were taken such as intensity, color, structure. These features were given to SVM which predicted their classes. Another method was presented in [5]. This method distinguished two diseases that impact rice crops. Firstly, they computed HSI color space from image and applied segmentation which was based on entropy-based thresholding. They applied kernel or filters on the image. After this they detected particular infected points. Another approach was used to classify 26 diseases in 14 crops which used CNN architecture [6]. My work is based on convolutional neural network.

SYSTEM DEVELOPMENT

3.1 Analysis/Design/Development/Algorithm

Data Set Used in the Major Project

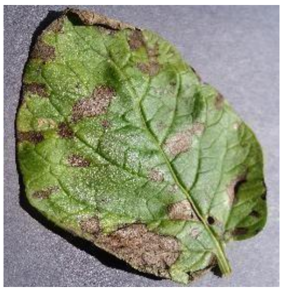
Any supervised machine learning project starts with data collection process There are basically three steps to collect dataset:

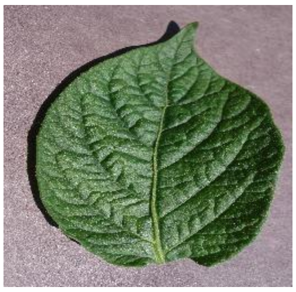
1) Collect and annotate data on your own.
2) Write web scrapping scripts to collect images from internet.

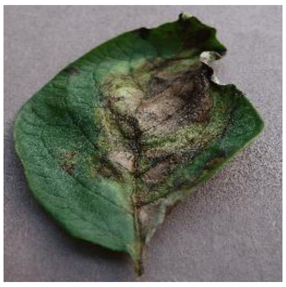
Buy data from third party vendors or use public repositories such as Kaggle. Dataset is taken from Kaggle repository. Dataset consist of images belonging to three different classes. Below is the link for dataset:

https://www.kaggle.com/arjuntejaswi/plant-village

Dataset contains three types of images:


Potato Early Blight

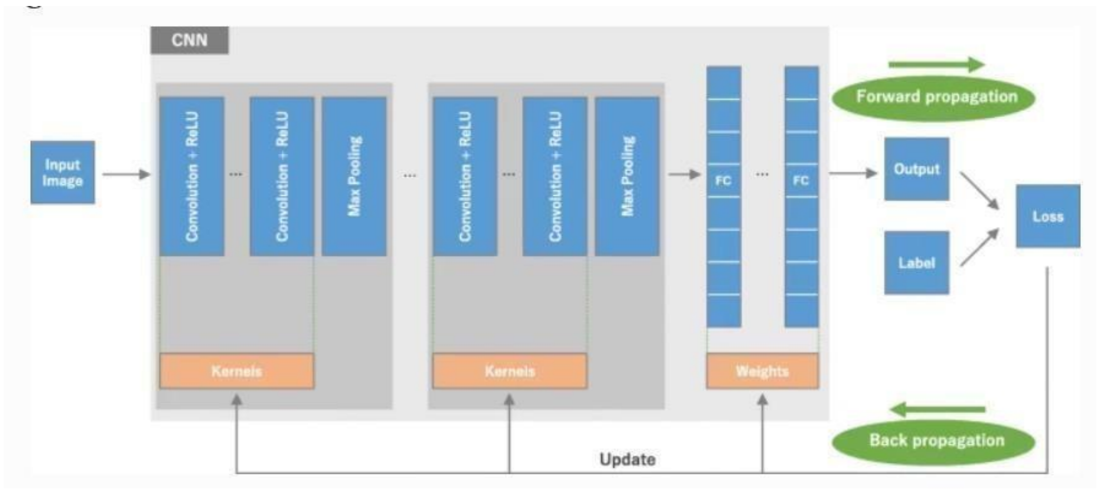

Potato Healthy


Potato Late Blight

3.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks are a special type of Neural Networks that has shown to be extremely effective in image categorization. As they provide extra layers such as convolution, pooling these networks can be used in computer vision applications and they are very useful in face recognition, object detection. Generally, this special type of networks consists of more than one convolutional layer. In the end layers they are followed by dense layers which are fully connected such as a simple neural network. CNN is made in such a way it takes the advantage of Two-dimensional structure of an image. In this, tied weights and local connections has been used. Then they are followed by max or mean pooling layers which are used to extract the most prominent features. Main advantage CNNs have over normal neural networks are that instead of having same number of hidden layers, they will have less parameters to train as compared to neural networks. They are easier to train because of having less parameters.

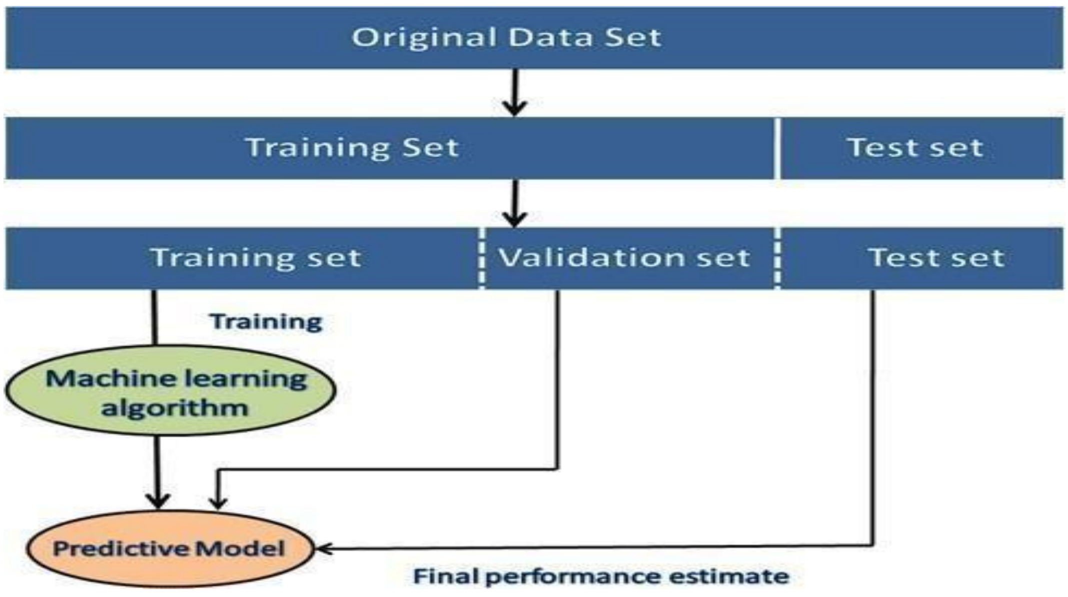
3.3 Overview of Different Operations in CNN



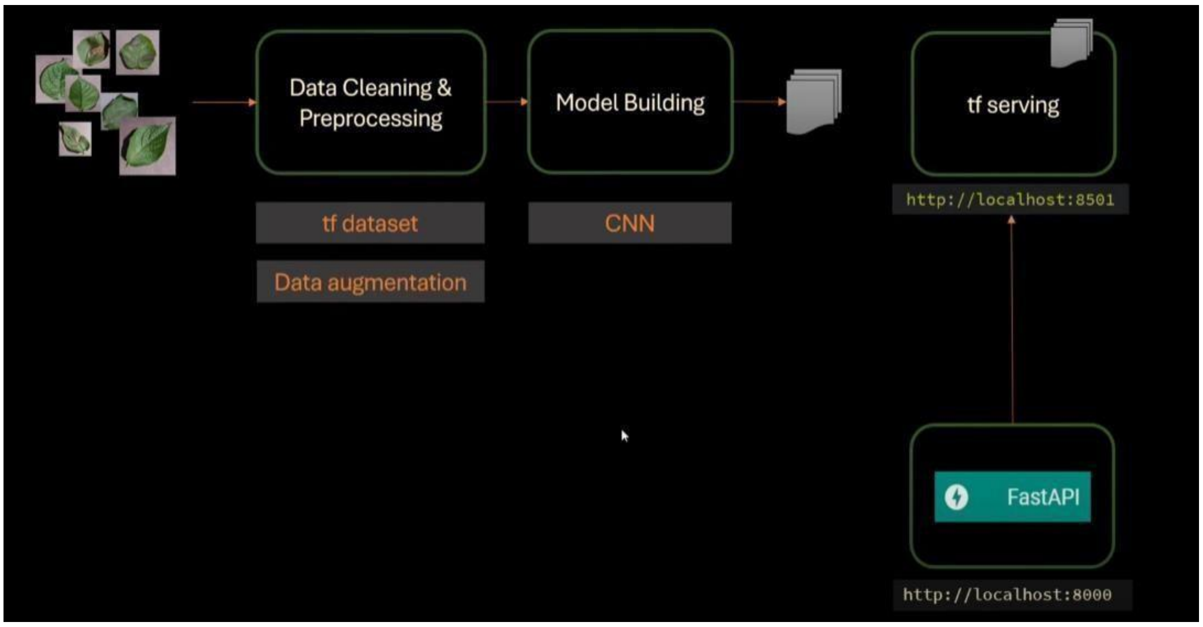
As shown in figure above, it is a CNN architecture which contains some sequential operations. This simple CNN consists of convolutional layer in which a kernel or filter is applied which is used to detect the vertical or horizontal edges and then it is followed by Relu activation function. Next layer is max pooling layer in which a kernel is applied to extract the most prominent features from the image. Then followed by fully connected dense layers. Accuracy of the model is determined by the loss function in forward propagation and according to the loss value weights, filters and bias are updated in back propagation which may be through gradient descent or AdaDelta or adam.

3.4 Model Development

State of the data changes every time it passes through a model. First, step is data pre-processing. Then data is splitted into two sets training set and test set. Training set is used for model building and training. Test set is used for model testing and checking the accuracy. Training set is further divided into a set which will be used for actual training and a validation set which is used for validation at back propagation. After model training, accuracy of the model is validated by using it on testing set. Then model is saved and deployed over an application.

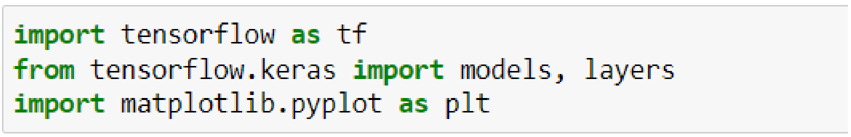


After collecting the data, first step is data cleaning and pre-processing for which I am using tf dataset and data augmentation. Data augmentation used when dataset is not that large then we can just rotate or increase decrease the contrast of the image to create a new image which can be used as an input. Next step is model building using convolutional neural network. CNN is a standard way for image classification as it provides better accuracy as compare to normal neural network. Then saving the model for deployment. Then backend of the web application is designed by FastApi. Below is the figure showing state transition diagram.



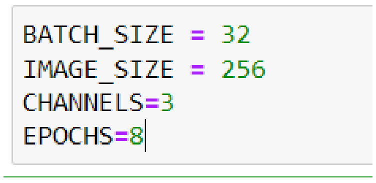
3.5 Analytical

First of all, importing all the dependencies



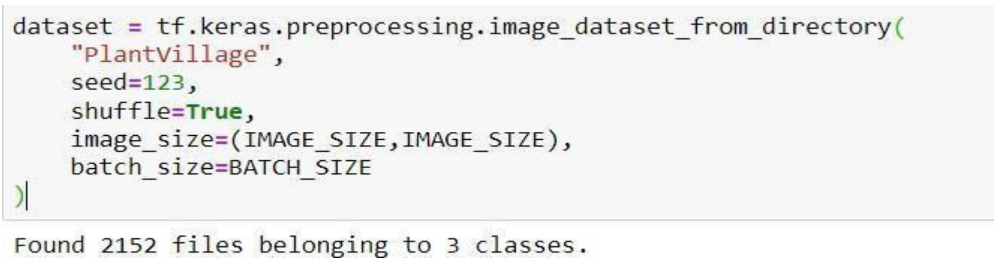
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

Specifying batch size in which each image will be processed. Image size is 256x256. Image is in RGB format. So, there are 3 channels. Epochs specify that how many times forward and back propagation will occur.



```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=8
```

We will use keras to create an input pipeline. In which we have to specify the directory in which images are there. There are three different folders separated according to their classes. Folder name will be considered as the class of each image present in that folder. As all of the images are of (256x256). So, image size is 256. We also need to specify batch size and it will process the images in batches. Benefit of processing in batches will be that it will utilize the resources effectively. If there is a huge dataset that is greater in size as compare to ram of the system then system may crash.



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

```
Found 2152 files belonging to 3 classes.
```

There are three classes as shown below

```
class_names = dataset.class_names
class_names
```

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

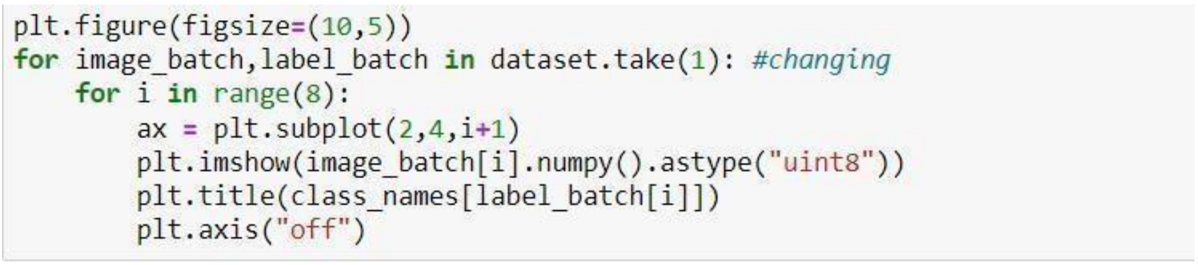
As shown below, each element in the dataset is a tuple. First element is a batch of 32 elements of images. Second element is a batch of 32 elements of class labels.

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

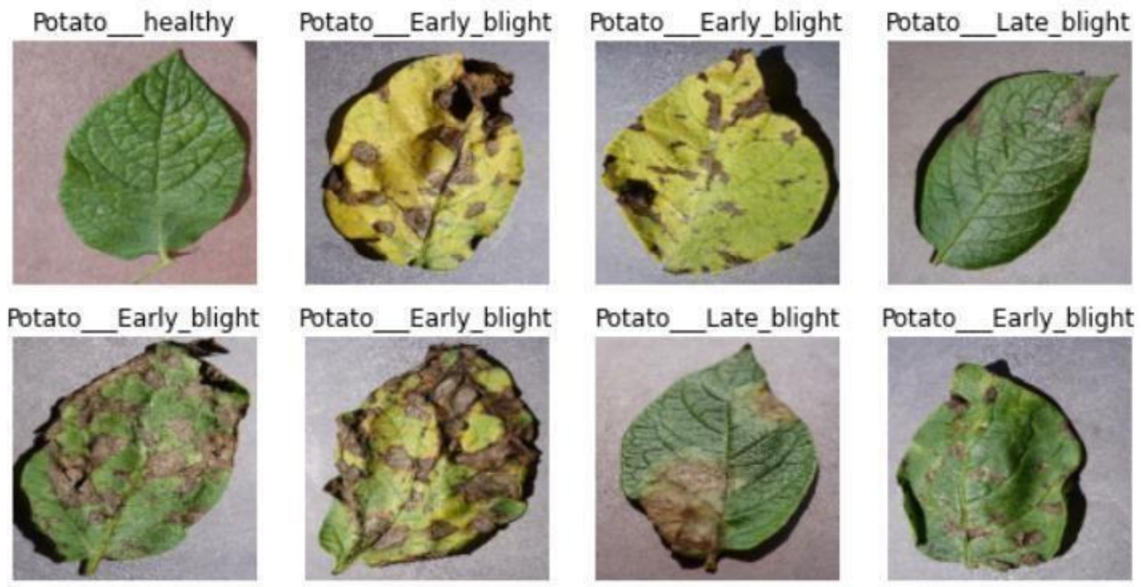
```
(32, 256, 256, 3)
[1 1 0 0 0 1 1 0 1 1 1 2 1 0 0 0 1 0 1 0 0 0 2 0 0 0 1 0 0 1 0 0]
```

Our dataset looks balanced now. We can create a model on top of this data.

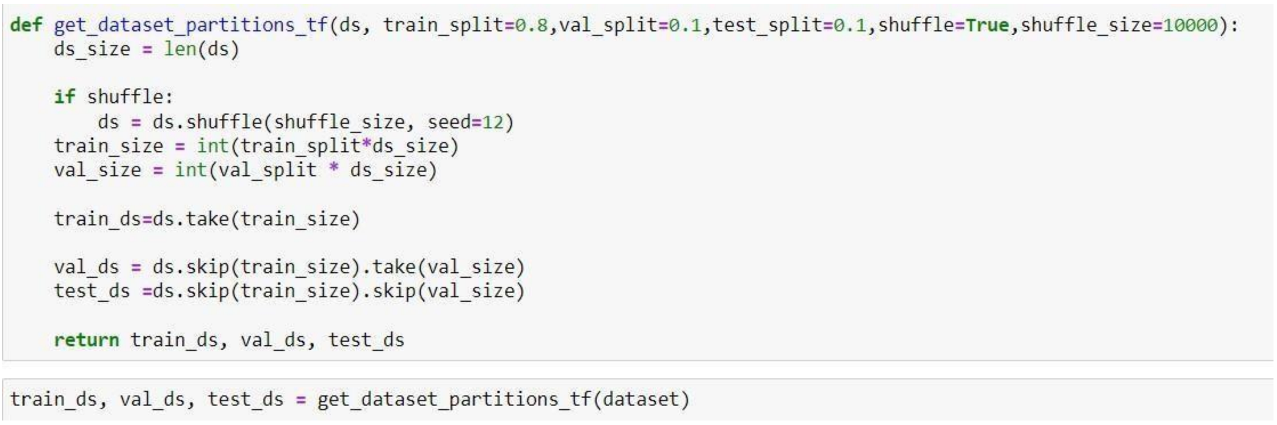
Plotting the images with their class label using matplotlib.



```
plt.figure(figsize=(10,5))
for image_batch,label_batch in dataset.take(1): #changing
    for i in range(8):
        ax = plt.subplot(2,4,i+1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[label_batch[i]])
        plt.axis("off")
```



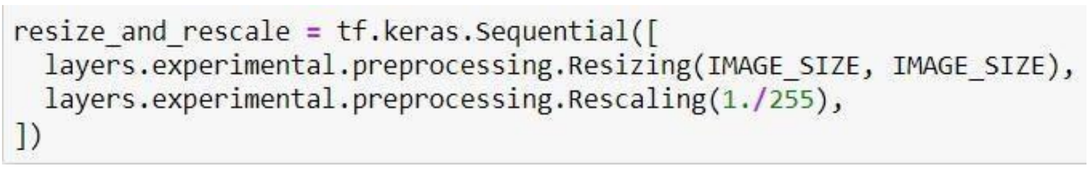
For splitting the data into training, testing and validation set a function is defined which takes dataset as an input. Other parameters have default values for train, validation and test split. There is also a shuffle parameter which will shuffle the images in the dataset to add randomness.



```
def get_dataset_partitions_tf(ds, train_split=0.8,val_split=0.1,test_split=0.1,shuffle=True,shuffle_size=10000):
    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)
    train_size = int(train_split*ds_size)
    val_size = int(val_split * ds_size)

    train_ds=ds.take(train_size)

    val_ds = ds.skip(train_size).take(val_size)
    test_ds =ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

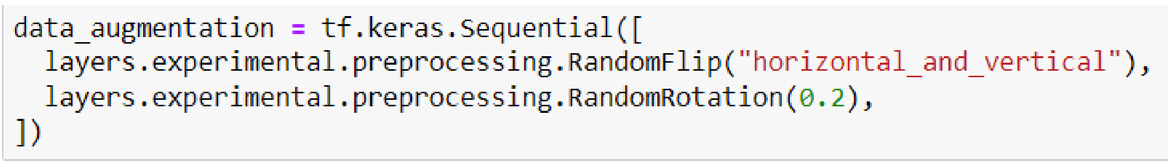
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

For resizing the input image if it is not of size (256x256) a layer has been defined using keras which will resize the image to (256x256) so that training can be done. It will also rescale the RGB values.

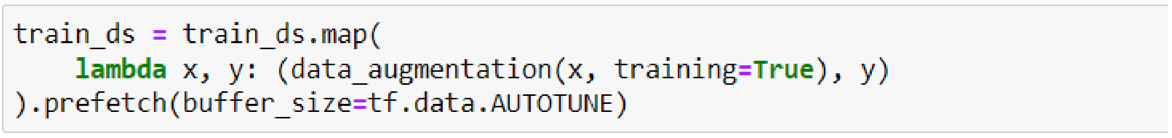


```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

To reduce the chances of underfitting data augmentation has been done which will increase data. It will flip the pictures vertically or horizontally and will also change their contrast so that new input images could be created.



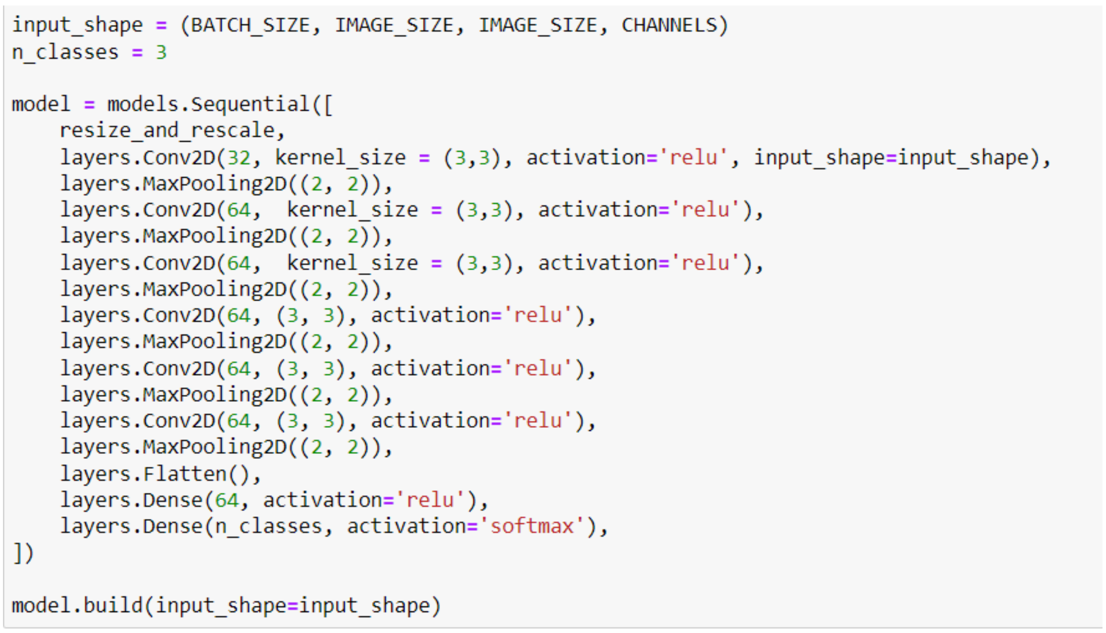
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```



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

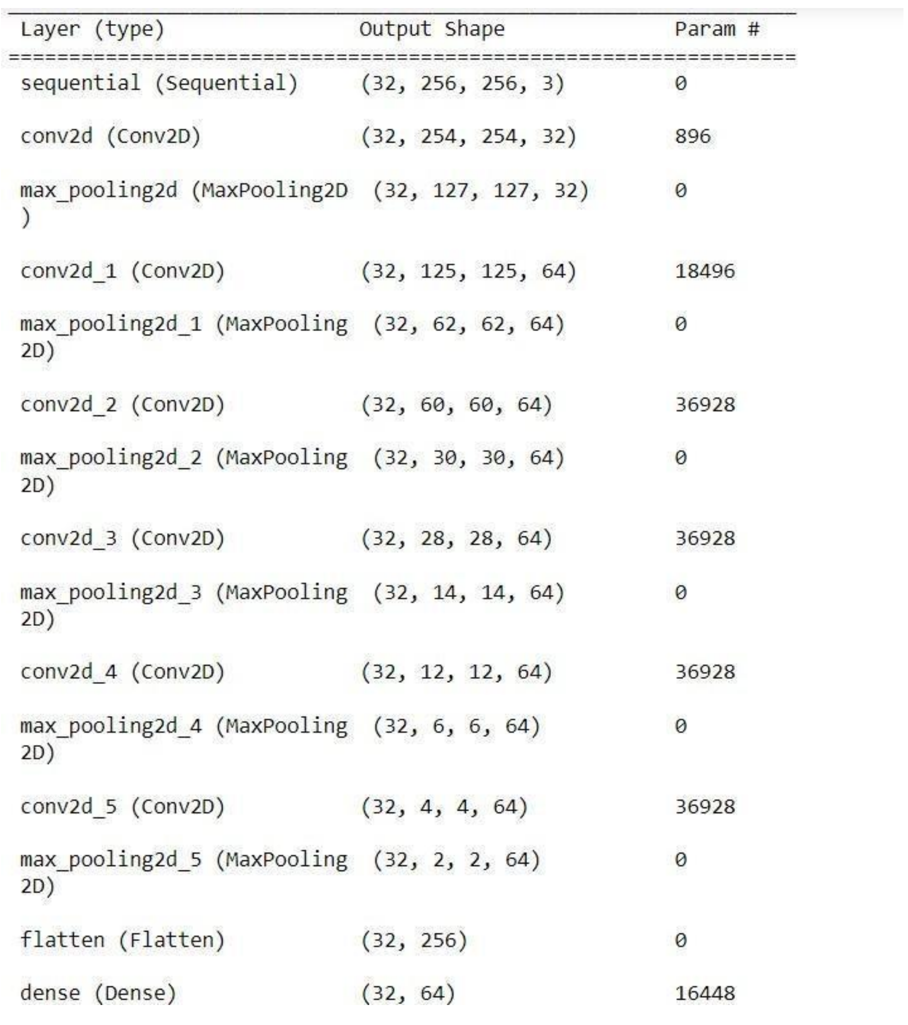
3.6 Experimental

After the data preprocessing steps, we trained a simple convolutional neural network. It contains a resizing and rescaling layer followed by number of convolution and max pooling layers. After these layers a fully connected layer is there with 64 neurons. At the end, there is output layer having three neurons as there are three different classes. In every hidden layer Relu activation function has been used. In output layer softmax function has been used. Below is the code:

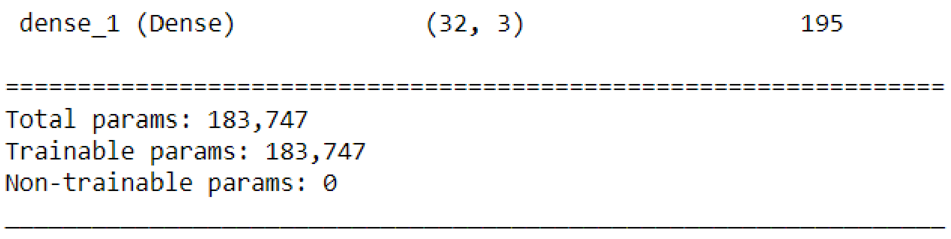


```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=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)
```

Detailed model architecture is shown below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0
 2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448
```

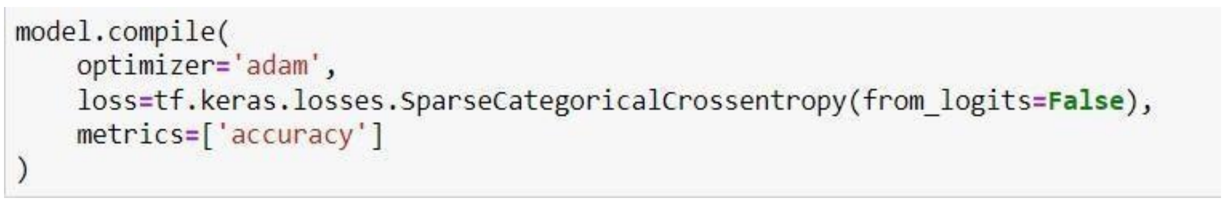


```
 dense_1 (Dense)              (32, 3)                   195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

Total trainable parameters is shown above in the figure.

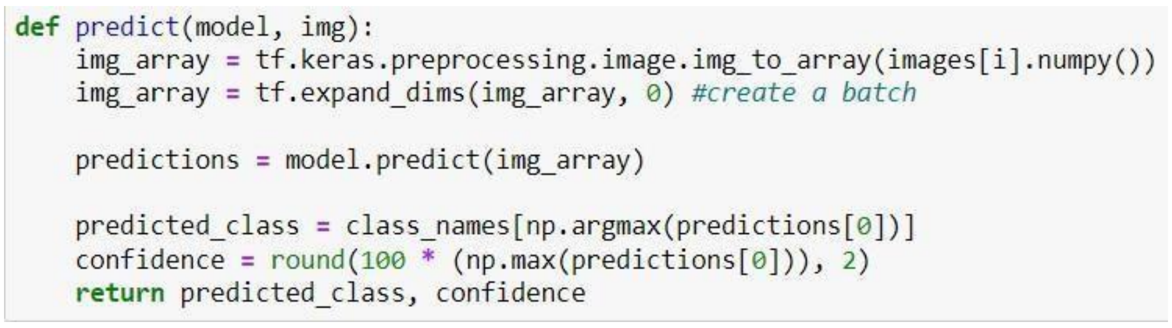
There are different type of loss functions. In this project sparse categorical cross entropy loss function has been used. Adam optimizer has been used which will reduce the loss by updating weights and bias.

Accuracy metrics has been used for validating the loss in back propagation.



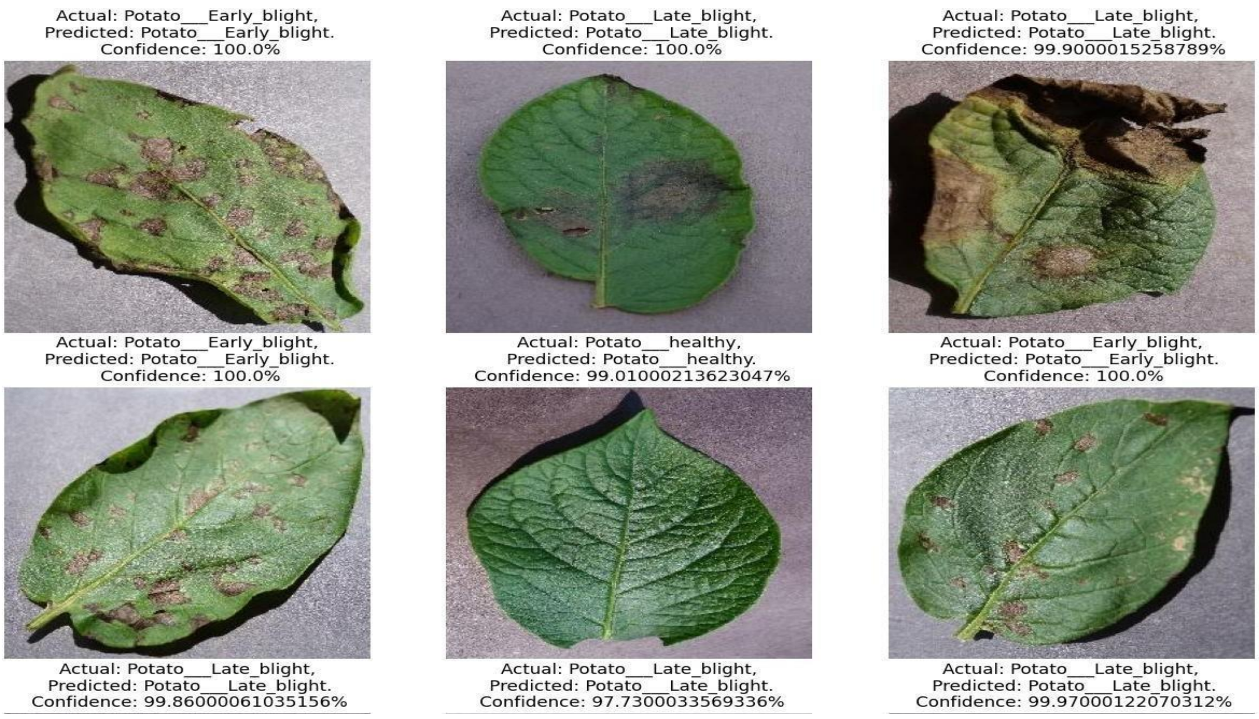
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

For prediction a function has been written which will take model and an input image as input parameters. Model will return an array which will contain the probability corresponding to each class from which we will select the probability having highest value. Below is the code:

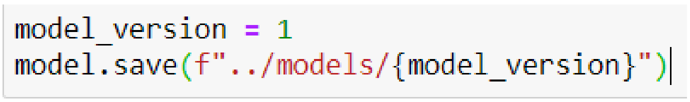


```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0) #create a batch

    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
```

Predictions which we got after applying predict function are shown below:

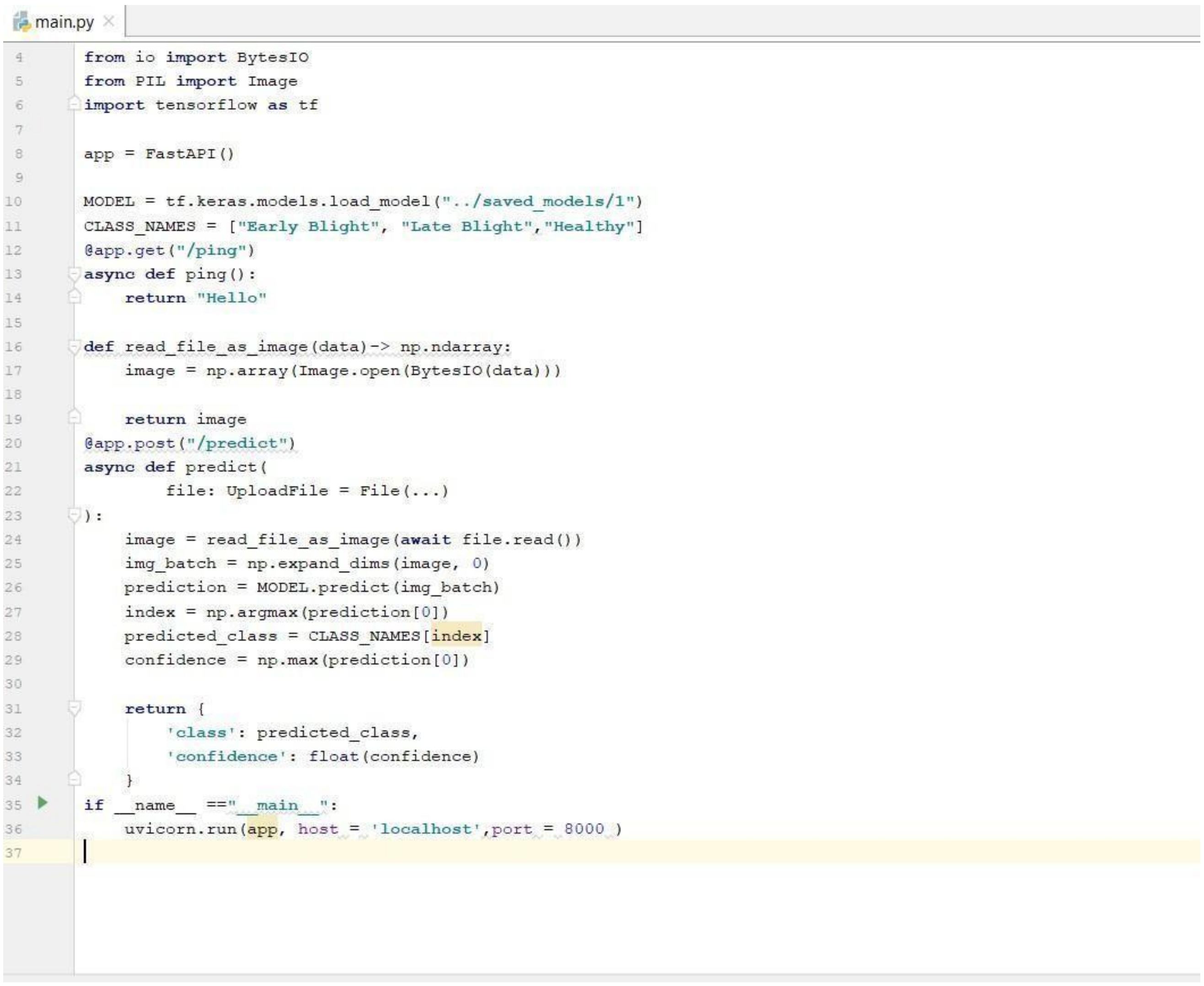

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.9000015258789%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%
Actual: Potato___healthy,
Predicted: Potato___healthy.
Confidence: 99.01000213623047%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.86000061035156%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 97.73000335693336%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.97000122070312%

After that model was saved. So that it can be used for prediction in our web application.



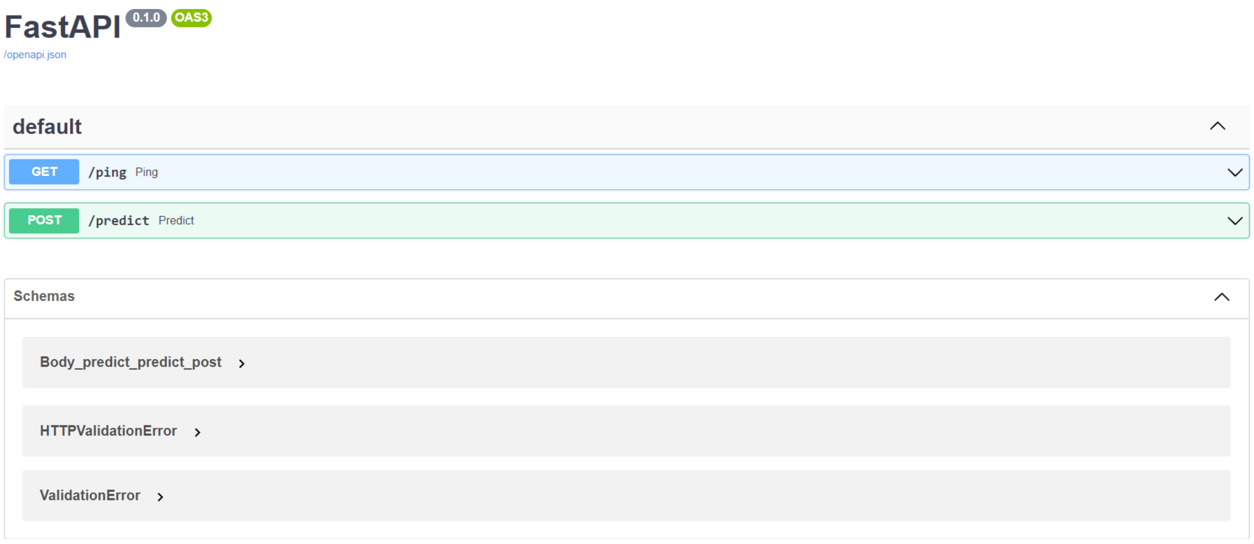
```
model_version = 1
model.save(f"../models/{model_version}")
```

For creating the web application FastAPI has been used which will manage backend of the web applicaion. It is one of the emerging python framework which is used to create simple Rest APIs. Below is the implemented code:



```
from io import BytesIO
from PIL import Image
import tensorflow as tf

app = FastAPI()

MODEL = tf.keras.models.load_model("../saved_models/1")
CLASS_NAMES = ["Early Blight", "Late Blight","Healthy"]
@app.get("/ping")
async def ping():
    return "Hello"

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)
    prediction = MODEL.predict(img_batch)
    index = np.argmax(prediction[0])
    predicted_class = CLASS_NAMES[index]
    confidence = np.max(prediction[0])

    return {
        'class': predicted_class,
        'confidence': float(confidence)
    }
if __name__ =="__main__":
    uvicorn.run(app, host = 'localhost',port = 8000 )
```

After running the code, web application will run at port number 8000. Advantage of FastAPI is that wihout having front end we can check the working of our model in FastAPI itself. There are various softwares which returns output of a web application in which there is no front end. For that we have to go to docs url. After this it will generate ui shown below:


FastAPI 0.1.0 OAS3

/openapi.json

default

GET /ping Ping

POST /predict Predict

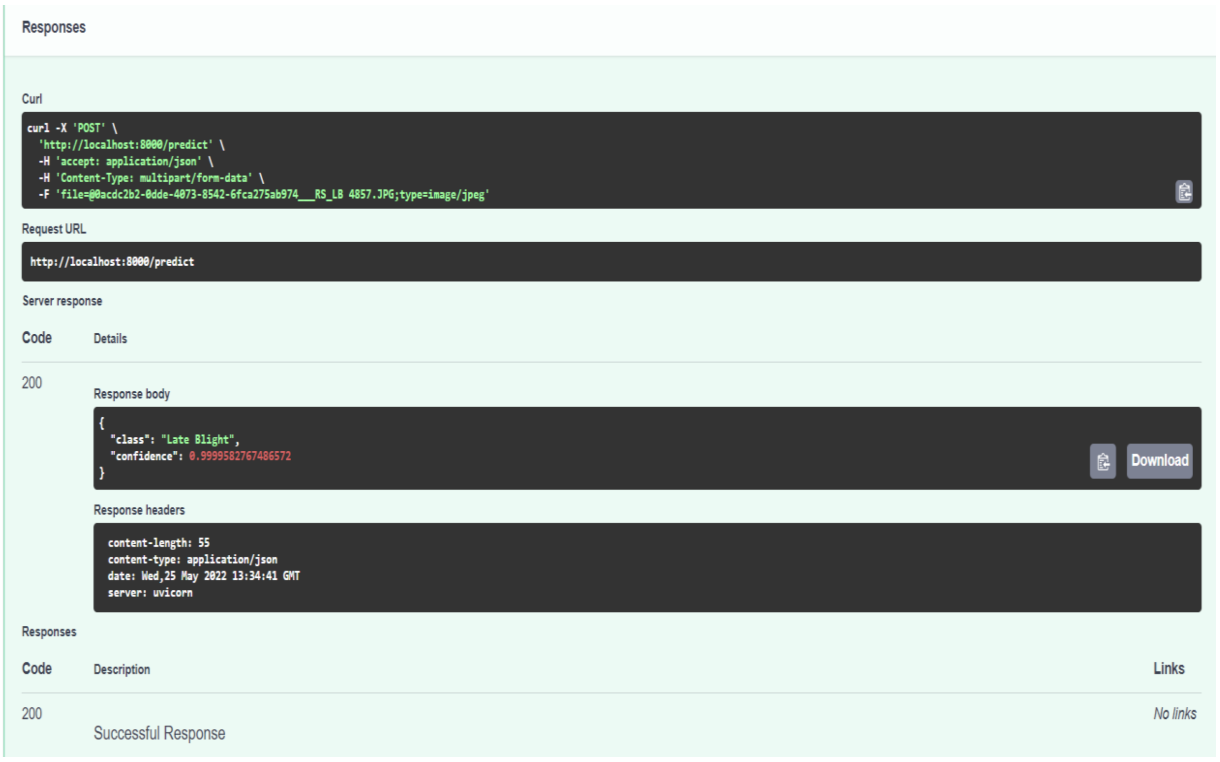
Schemas

Body_predict_predict_post

HTTPValidationError

ValidationError

In FastAPI, an upload file variable is there which is used to upload an file. This variable has been used to upload an image file and then model will classify image according to the probability. Below is the figure in which an input image has been classified


Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@0acdc2b2-0dde-4073-8542-6fca275ab974___RS_LB 4857.JPG;type=image/jpeg'
```

Request URL

```
http://localhost:8000/predict
```

Server response

Code	Details
200	Response body

```
{
  "class": "Late Blight",
  "confidence": 0.9999582767486572
}
```

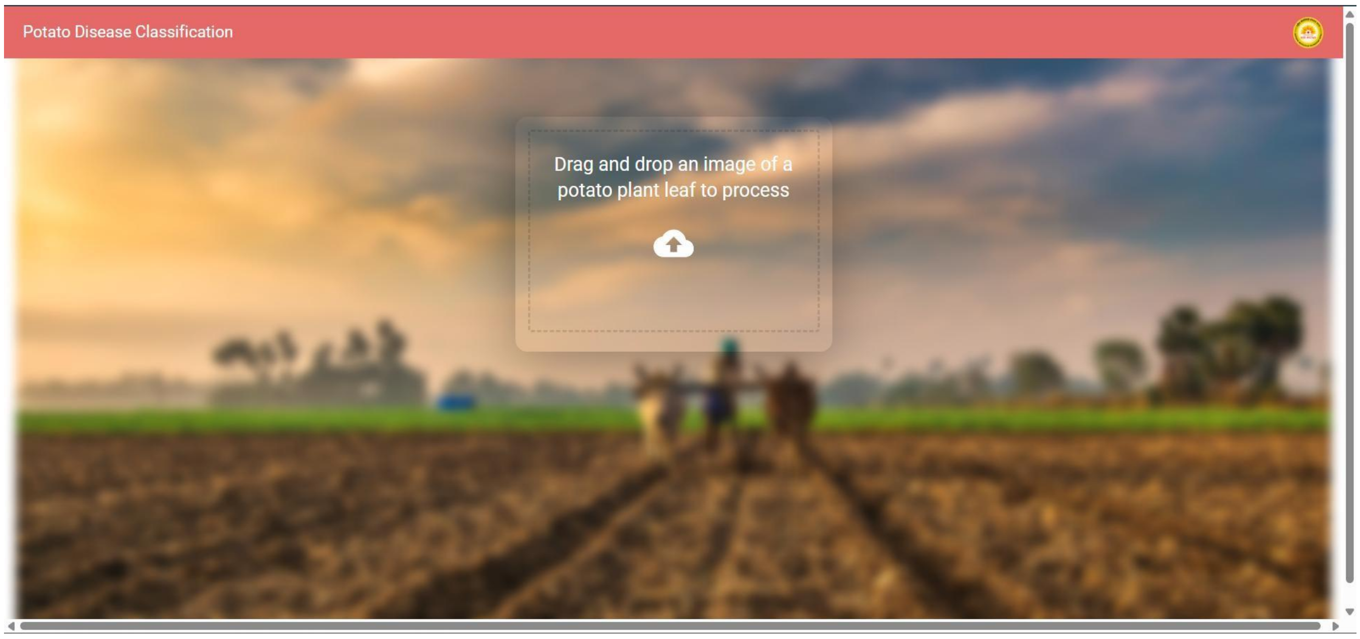
Response headers

```
 content-length: 55
 content-type: application/json
 date: Wed,25 May 2022 13:34:41 GMT
 server: uvicorn
```

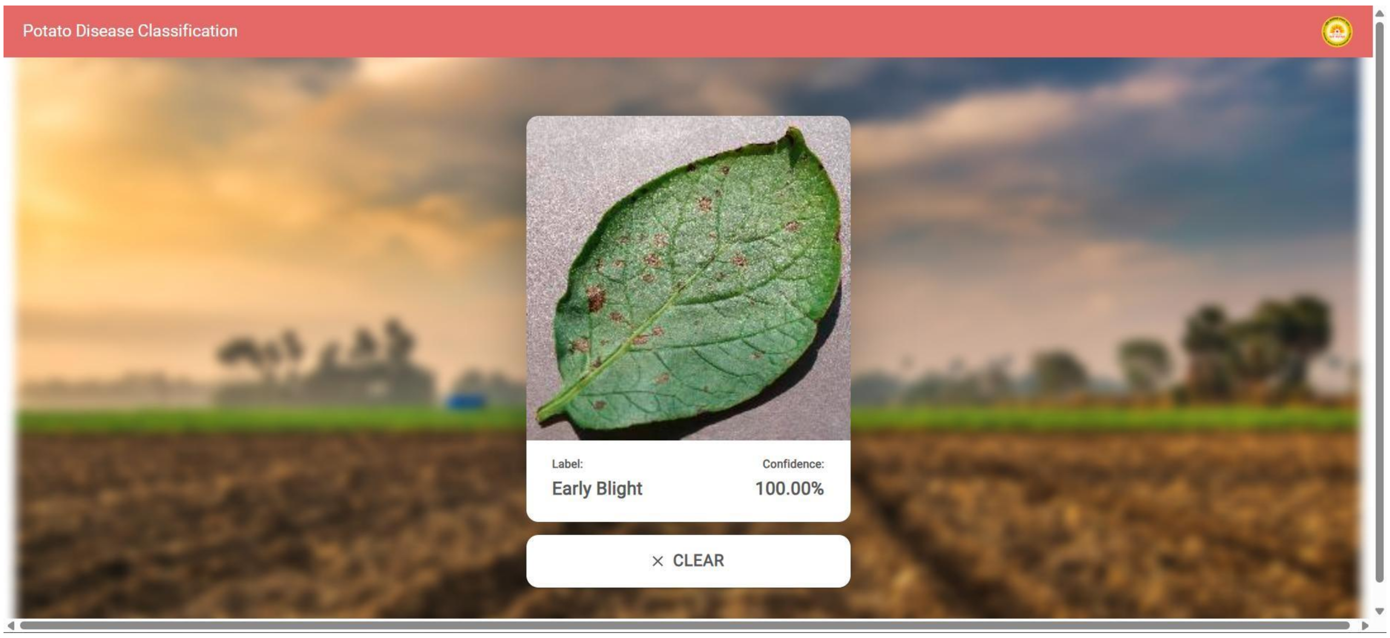
Responses

Code	Description	Links
200	Successful Response	No links

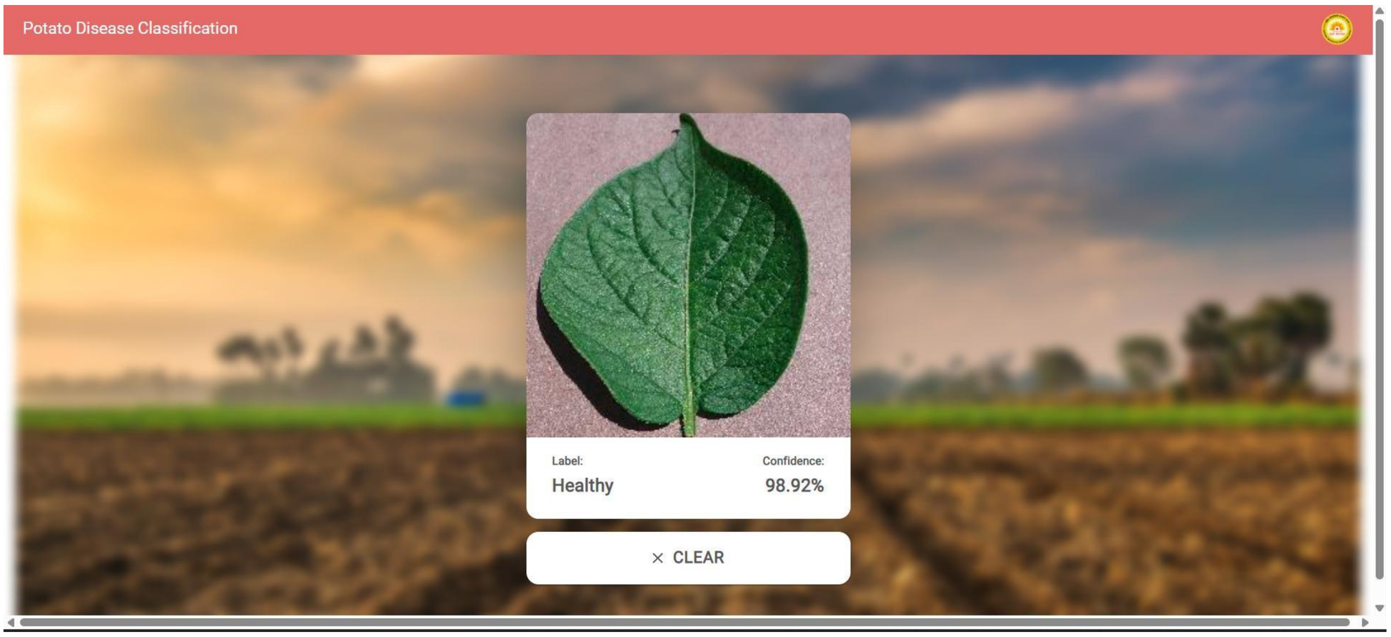
Given image has been classified as an Late_blight class with 0.99 confidence.


Potato Disease Classification
Drag and drop an image of a potato plant leaf to process

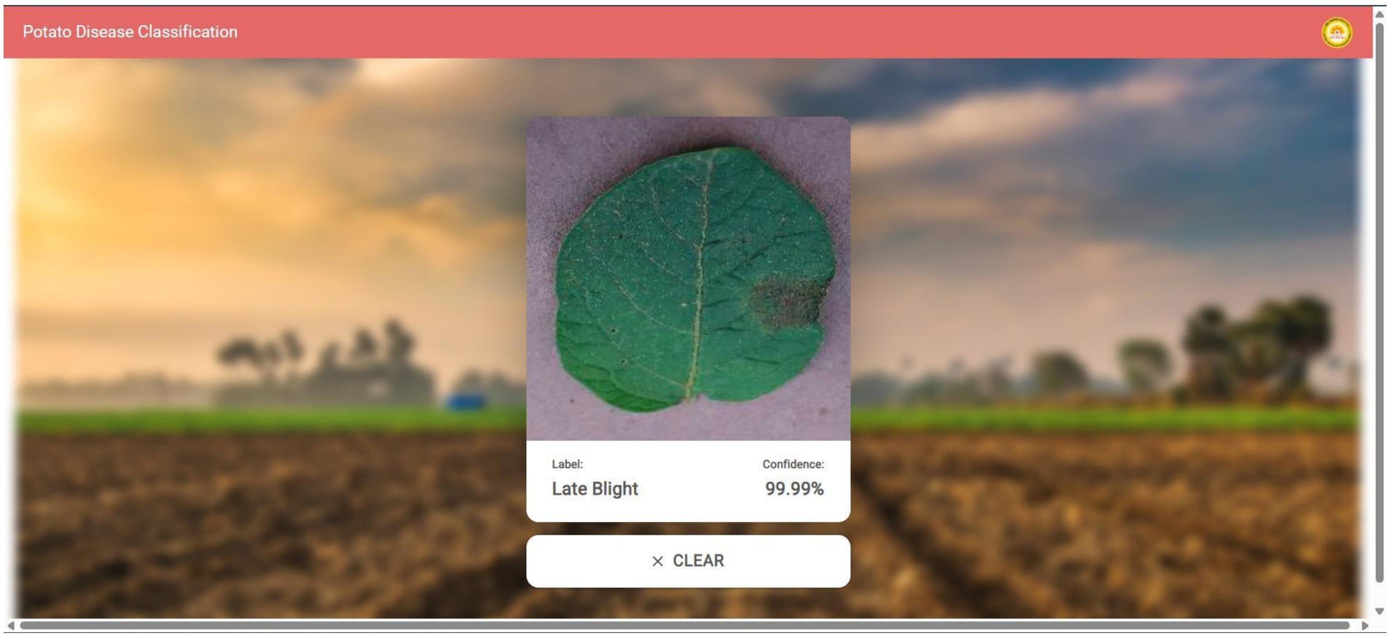
The landing page of the Potato Disease Classification website features a clean and intuitive interface. Users are greeted with a clear instruction to 'Drag and drop an image of a potato plant leaf to process,' accompanied by a cloud icon symbolizing file upload. The background displays a blurred image of a potato field, subtly hinting at the application's purpose. This design encourages users to easily upload an image for disease detection.



Upon uploading an image of a potato leaf exhibiting symptoms, the website processes it and displays the predicted disease. In this instance, the system has identified 'Early Blight' with a confidence score of 100.00%.



In this image, the system has analyzed an image of a healthy potato leaf. The website accurately identifies the leaf as 'Healthy' with a high confidence level of 98.92%. This demonstrates the model's ability to distinguish between healthy and diseased foliage.



Finally, this image illustrates the website's ability to detect 'Late Blight'. The system has confidently identified the disease with a score of 99.99%. This result, consistent with the previous examples, is clearly displayed to the user, reinforcing the application's capability to accurately classify different potato leaf conditions.

PERFORMANCE ANALYSIS

The following measures were used to measure the performance of the implemented models:

Accuracy – It is the ability to accurately distinguish between real and counterfeit note test cases. Accuracy = (tp+tn) / (tp+tn+fp+fn)

Sensitivity - It is the ability to accurately identify the original note cases.
Sensitivity = tp / (tp+fn)

Specificity - The uniqueness of the test is the ability to accurately identify cases of counterfeit notes.
Specifcity = tn / (tn+fp)

Precision - The accuracy of the test is the ability of the classifier to decide whether the notes categorized as real are real.
Precison = tp / (tp+fp)

Where,
True positive = It is the count that contains correctly determined original notes.

True negative = It is the count that contains correctly determined as counterfeit notes. False positive = It is the count that contains misidentified as original notes.

False negative = It is the count that contains misidentified as counterfeit notes.

We will compare two algorithms Random Forest classifier and K Nearest Neighborhood.

For validation, accuracy matrix has been used which will check its accuracy in back propagation and update the parameters according to that. Model has been trained for 8 epochs. Below is the code of model fitting.

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=8,
)
```

The results of training and validation loss in each epoch are shown below in the figure.

```
Epoch 1/8
54/54 [==============================] - 134s 2s/step - loss: 0.1629 - accuracy: 0.9410 - val_loss: 0.1910 - val_accuracy: 0.90
62
Epoch 2/8
54/54 [==============================] - 140s 3s/step - loss: 0.1489 - accuracy: 0.9416 - val_loss: 0.1388 - val_accuracy: 0.95
31
Epoch 3/8
54/54 [==============================] - 135s 2s/step - loss: 0.1325 - accuracy: 0.9462 - val_loss: 0.1201 - val_accuracy: 0.95
83
Epoch 4/8
54/54 [==============================] - 136s 3s/step - loss: 0.1436 - accuracy: 0.9473 - val_loss: 0.1011 - val_accuracy: 0.95
31
Epoch 5/8
54/54 [==============================] - 130s 2s/step - loss: 0.0963 - accuracy: 0.9635 - val_loss: 0.1198 - val_accuracy: 0.96
35
Epoch 6/8
54/54 [==============================] - 133s 2s/step - loss: 0.0872 - accuracy: 0.9688 - val_loss: 0.3165 - val_accuracy: 0.90
62
Epoch 7/8
54/54 [==============================] - 131s 2s/step - loss: 0.0763 - accuracy: 0.9705 - val_loss: 0.1311 - val_accuracy: 0.94
79
Epoch 8/8
54/54 [==============================] - 128s 2s/step - loss: 0.0499 - accuracy: 0.9792 - val_loss: 0.2033 - val_accuracy: 0.92
19
```

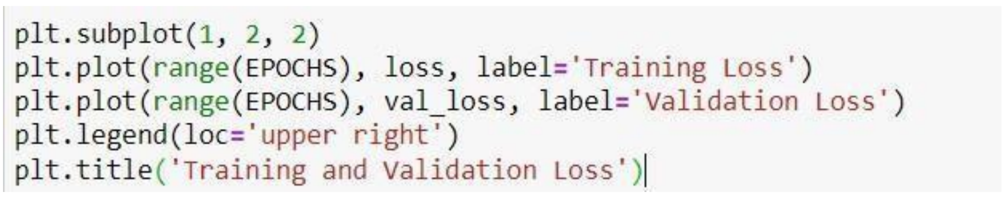
When evaluated on a test dataset, model gave 94% accuracy. Below is the code and result.

```
scores = model.evaluate(test_ds)
```

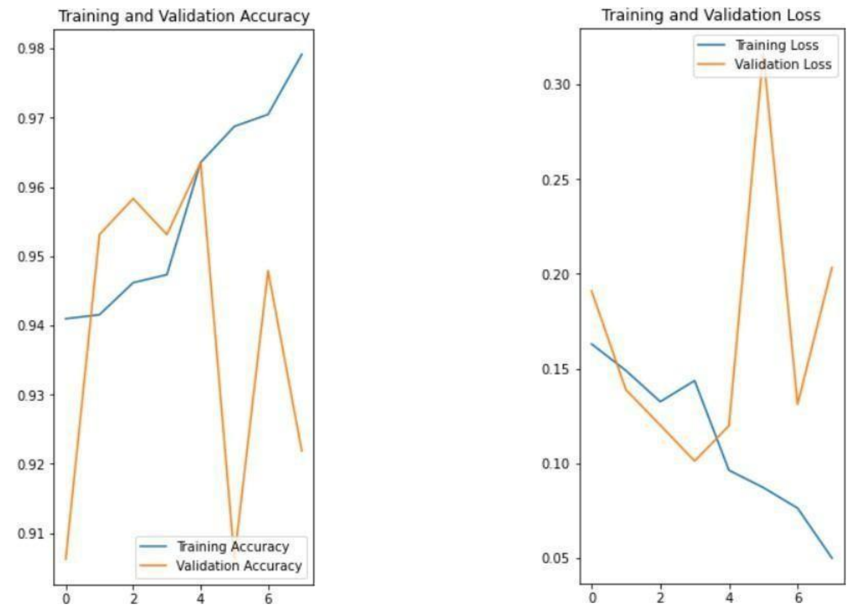
```
8/8 [==============================] - 6s 399ms/step - loss: 0.1160 - accuracy: 0.9414
```

Below is the code and result of a graph between training accuracy vs validation accuracy and training loss vs validation loss.

```
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')
```



```
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')
```



CONCLUSION

This project demonstrates the effectiveness of deep learning and computer vision in automating potato disease detection. By leveraging a Convolutional Neural Network (CNN), the system successfully classifies potato leaves into Healthy, Early Blight, and Late Blight categories, offering a reliable alternative to manual disease detection. The use of TensorFlow and Keras enables efficient model training, while FastAPI facilitates web-based deployment for real-time image classification.

Despite achieving promising results, the project faces challenges such as dataset limitations, computational constraints, and real-world variability in lighting and background conditions. Techniques like data augmentation, transfer learning, and model optimization help improve accuracy and performance. However, further enhancements, such as deploying a mobile application or integrating the model with IoT-based farming solutions, can make this system even more practical for farmers.

By addressing these challenges and expanding the scope of the model to detect other crop diseases, this project has the potential to revolutionize precision agriculture. Future improvements will focus on enhancing real-time detection, scalability, and accessibility, ensuring that AI-powered disease classification becomes a widely adopted tool in modern farming.

REFERENCES

[1]. AurélienGéron (2017), Hands–On ML with Scikit–Learn and TensorFlow (2nd Edition),Publisher:
O'Reilly Media

[2]. Mirwaes Wahabzada, Anne-Katrin Mahlein, Christian Bauckhage, Ulrike Steiner, Erich Christian Oerke, Kristian Kersting, "'Plant Phenotyping using Probabilistic Topic Models: Uncovering the Hyperspectral Language of Plants," Scientific Reports 6, Nature, Article number: 22482, 2016.

[3]. Macedo-Cruz A, Pajares G, Santos M, Villegas-Romero I., "Digital image sensor-based assessment of the status of oat (Avena sativa L.) crops after frost damage," Sensors 11(6), 2011, pp. 6015–6036.

[4]. Yao Q, Guan Z, Zhou Y, Tang J, Hu Y, Yang B, "Application of support vector machine for detecting rice diseases using shape and color texture features," 2009 International Conference on Engineering Computation, IEEE, Hong Kong, 2009, pp. 79–83.

[5]. Phadikar S, Sil J, "'Rice disease identification using pattern recognition techniques," IEEE, Khulna, 2008, pp. 420_423.

[6]. Sharada Prasanna Mohanty, David Hughes, Marcel Salathe, "Using Deep Learning for Image- Based Plant Disease Detection," Computer Vision and Pattern Recognition, to be published.