# Potato Disease Classification using CNN

**Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology**

**In**

**Computer Science and Engineering**

**By**

**Ishan Thakur[181302]**

UNDER THE SUPERVISION OF

**Dr. Shubham Goel**

to

**Department of Computer Science & Engineering and Information Technology**

**Jaypee University of Information Technology, Waknaghat,**

**173234, Himachal Pradesh**

# TABLE OF CONTENTS

## DECLARATION

I hereby declare that, this project has been done by me under the supervision of **Dr. Shubham Goel, Assistant Professor (SG),** Jaypee University of Information Technology. I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

**Supervised by:**

**Dr. Shubham Goel**

Assistant Professor

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology

**Submitted by:**

Ishan Thakur

Ishan Thakur (181302)

Computer Science & Engineering Department

Jaypee University of Information Technology

## ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes us possible to complete the project work successfully.

I really grateful and wish my profound my indebtedness to Supervisor **Dr. Shubham Goel, Designation**, Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of **data science and Machine learning** to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Shubham Goel,** Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straight forwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Ishan Thakur [181302]

## CERTIFICATE

## Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Potato Disease classification using CNN"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2015 to December 2015 under the supervision of **Dr. Pradeep Kumar Gupta,** Associate Professor Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ishan Thakur[181302]

This is to certify that the above statement made by the candidate is true to the best of my knowledge

Dr. Shubham Goel
Assistant Professor
Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat.

# 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 api which will take leaf image as input and will classify according to the trained model.

# Chapter 1 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 Objectives

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.

# Chapter 2

# LITERATURE SURVEY

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.

# Chapter 3
# 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.
3) 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_healthy

Potato_Early_blight

Potato_Late_blight

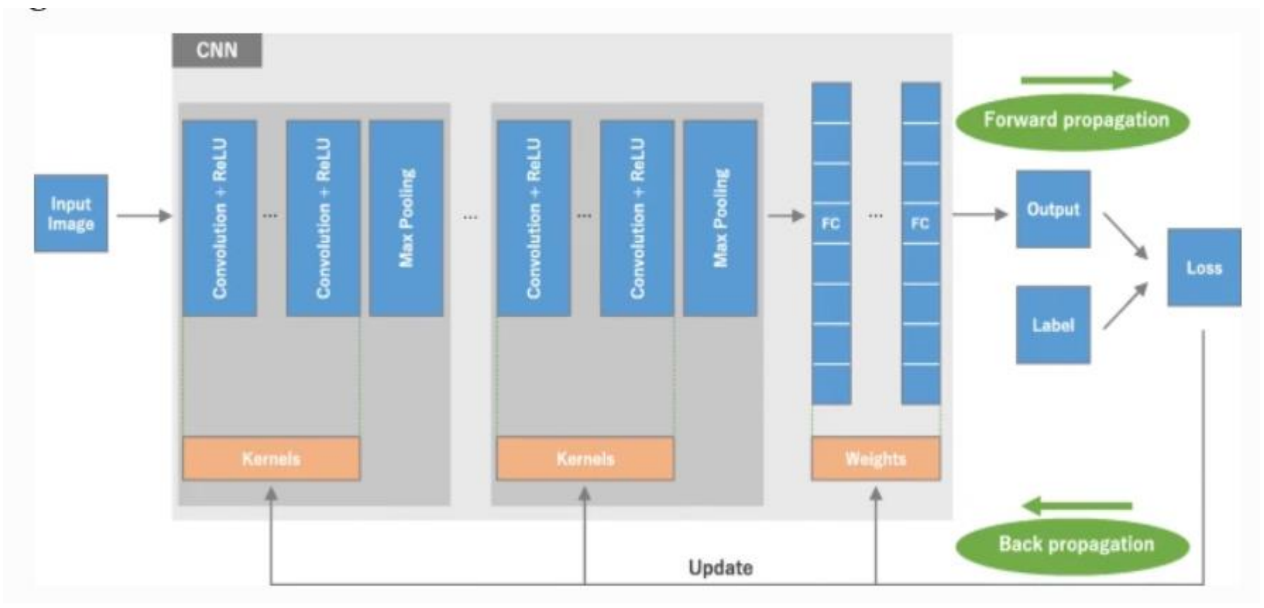
### 3.1.1Algorithms/ Pseudo code of the Project Problem

**What are Convolutional Neural Networks?**

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.

**Overview of different operations in CNN:**

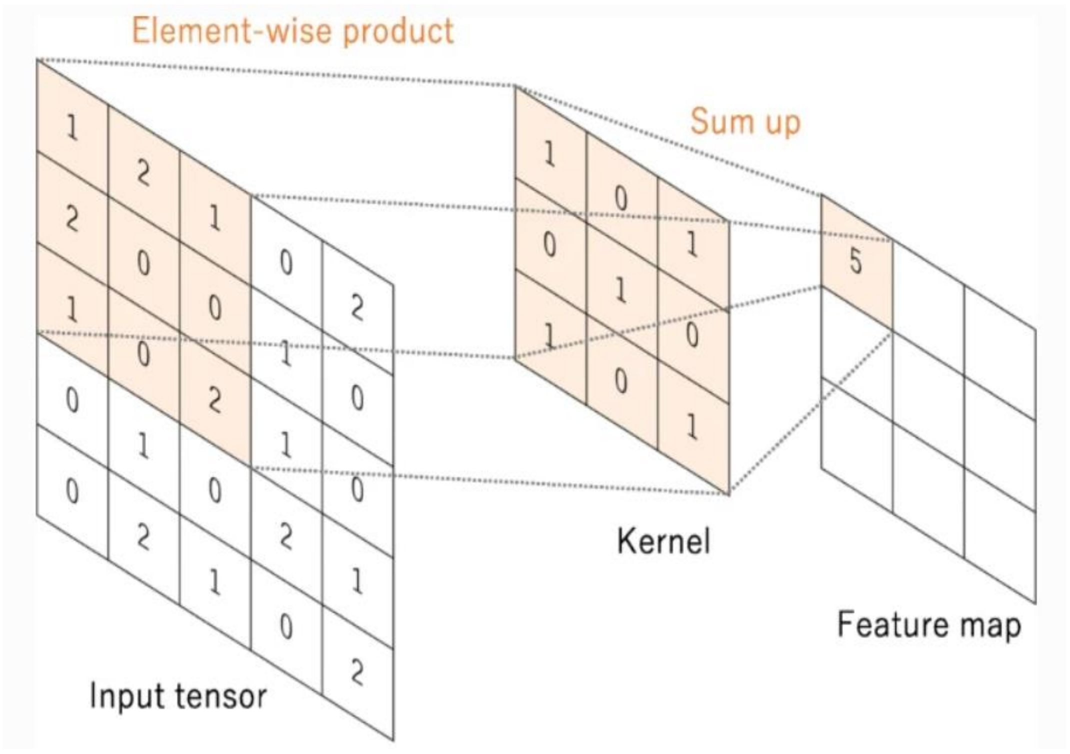


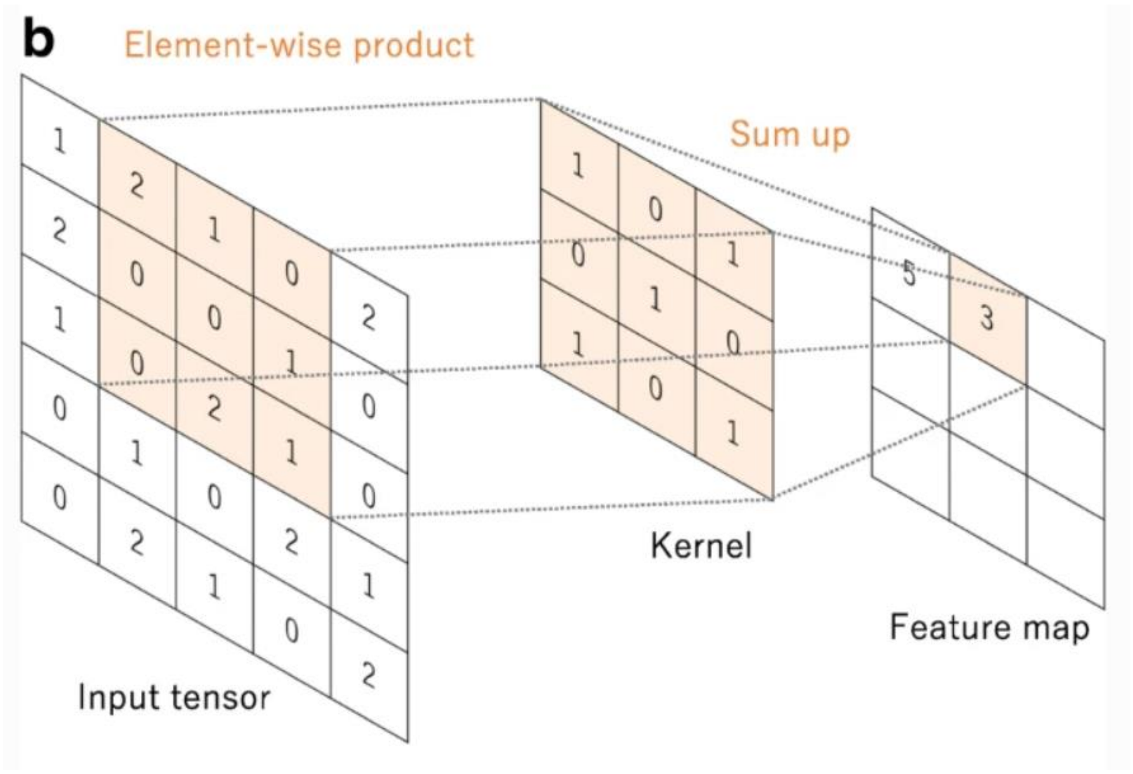
As shown in figure above, it is a CNN architecture which contains some sequential operations. This simple CNN consists of convolutional layer in which a kernel or filter is applied which is used to detect the vertical or horizontal edges and then it is followed by Relu activation function. Next layer 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.

**Convolution**

In this operation, a small tensor is applied across an input image which is called as a kernel or a filter. Its main task is to extract the important features from the image. After this operation a feature map is generated which is computed through element wise multiplication of each value in image and kernel. This feature map will contain the vertical and horizontal edges which are enhanced in next step.

**Figure showing convolution operation**





Above figure is showing a convolution operation on a 5x5 image. Filter size is 3x3 without padding and stride equal to 1. If there is no padding then it will reduce its size as shown in the figure. To maintain same image size padding is used which will add extra pixels across image to maintain its size.

**Formula to calculate image size:**

**(N+ 2P – F)/S + 1**

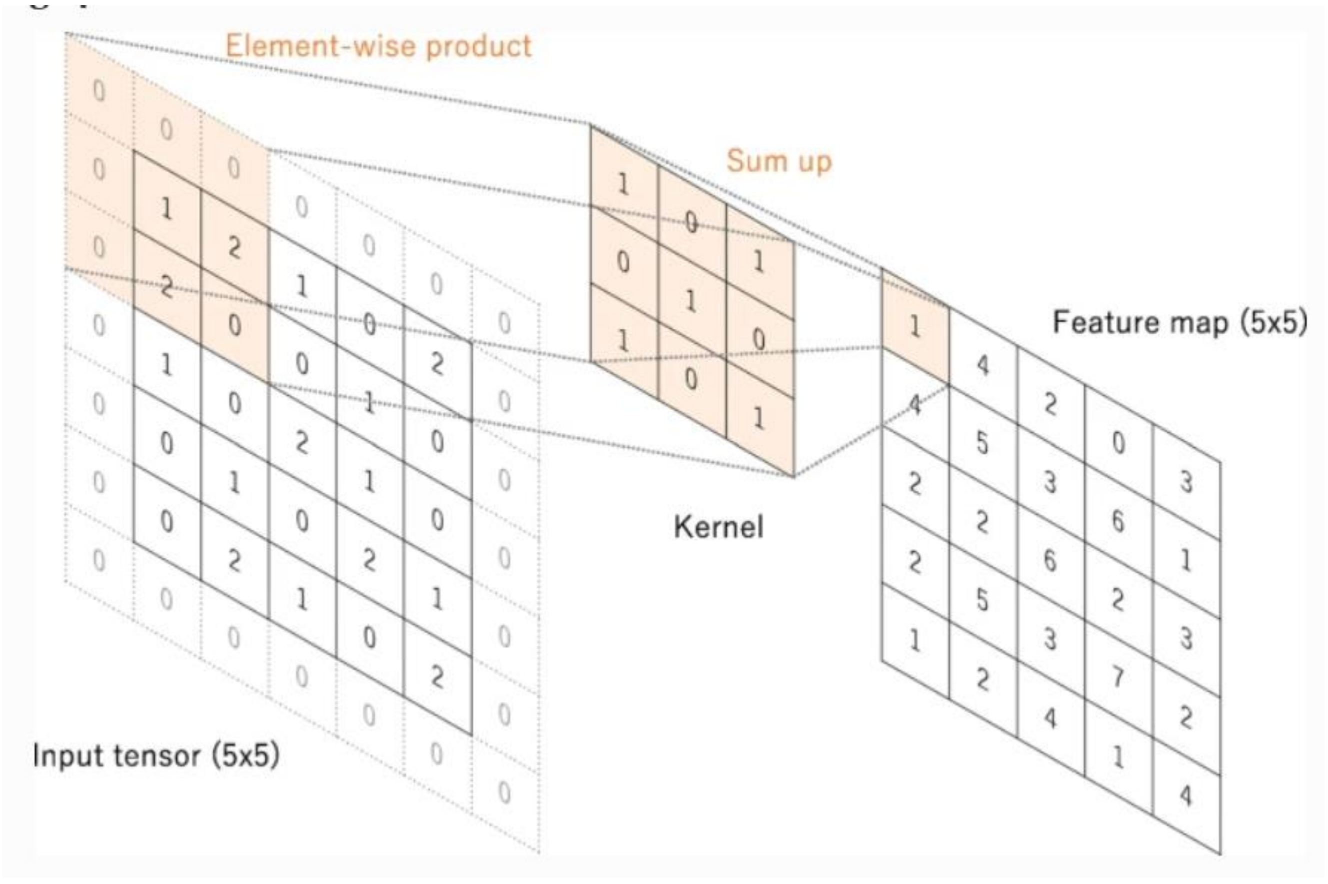
Where N = image size

P = padding

F = kernel size

S = stride

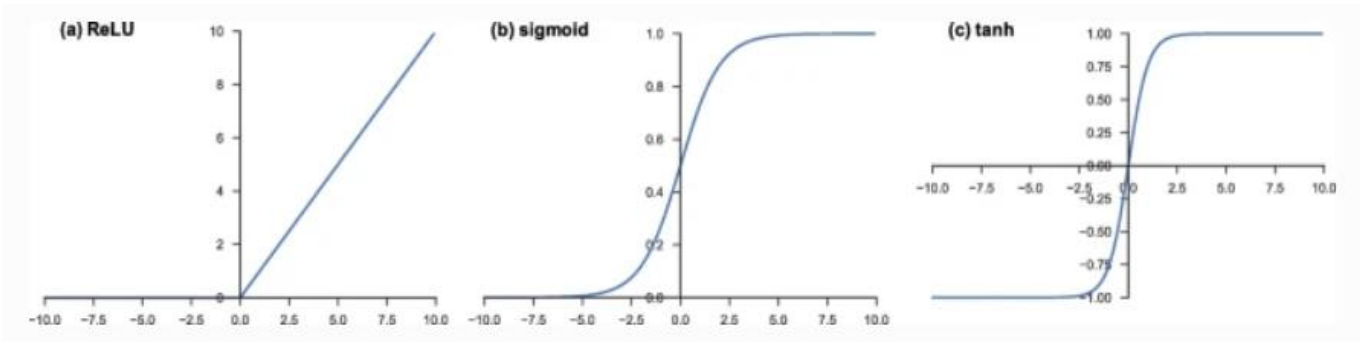
**Figure with padding:**



**Activation Function:**

Feature maps which are obtained after applying convolution are passed through an activation function. These activation functions are non-linear to introduce non-linearity in the network. Activation functions such as sigmoid, tangent are not used in hidden layers as it may result is vanishing gradient problem. Highly used activation function is Relu function.

F(x) = max(0,x)

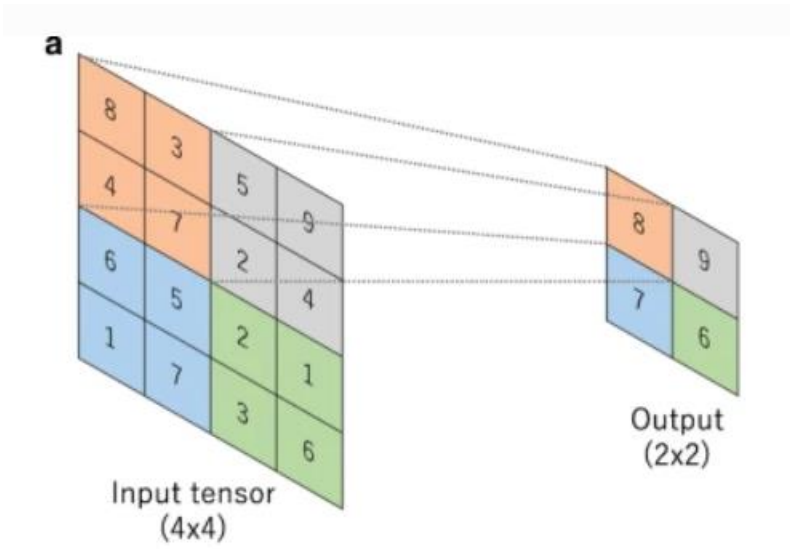


**Pooling Layer**

This layer is used to extract the most important features. Its main task is to reduce the learnable parameters. It reduces the height and width of feature map but depth remains same.

**Max Pooling**

In this layer a kernel is applied across an image which will extract maximum value from each patch. Mainly 2x2 filter size is used with stride equal to 2 over an image. Below is the figure showing max pooling operation.

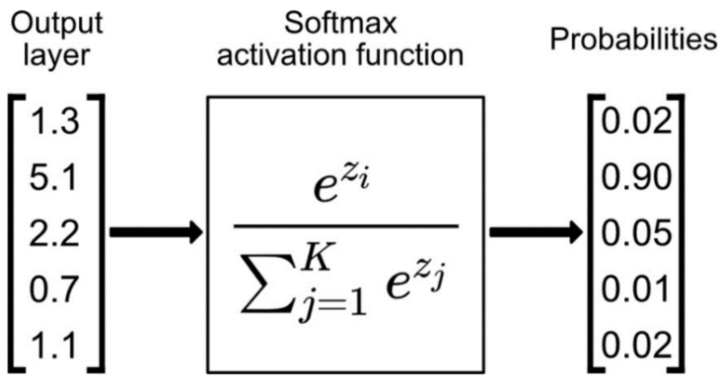


**Fully Connected Dense Layer**

After receiving feature maps from max pooling layer, these feature maps are flattened and passed to a fully connected dense layer. Feature maps are first converted to a one-dimensional array and passed to a dense layer. There can be many dense layers in-between. Final output layer mainly contains the number of output classes. Every hidden dense layer contains an activation function. Most commonly used activation function is Relu.

**Activation Function at Last Layer**

Activation function which is used at last output layer is different from the activation functions that has been used in hidden dense layers as their main task is to introduce non-linearity. Activation function at last layer is used to categorize that means it gives probability of each class. If there are two classes then sigmoid function is used. But in this project, there are more than two categories. So, in this situation softmax function is used which will normalize in such a way that each probability value lies in range of 0-1 and final sum will be 1. Below is the figure showing output of a sigmoid function.



Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

In forward propagation all the weights, filters and bias are calculated. These are then validated by a cost function which will give error. Error is the difference between actual label and predicted label. In back propagation weights, filters and bias updation occurs. This updation is carried out by an optimizer which is an algorithm. Its task is to find values of weights, filters and bias is such a way that it will reduce its loss. There are many optimizers available. One which I am using is adam.

**Loss Function**

Loss function is used to validate our result. It gives the difference between actual label and predicted label. When records are passed in batches or all the records are passed then loss function is termed as cost function. There are many types of cost functions such as multi-class cross entropy and sparse categorical cross entropy. Different loss functions may give different results. So, we need to choose loss functions according to the use case.

**Binary Cross Entropy**

$$\text{Loss} = -\frac{1}{\substack{\text{output}\\ \text{size}}} \sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log\,(1 - \hat{y}_i)$$

**Multi Class Cross Entropy**

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

**Optimizers**

Optimizers are algorithms which are used to minimize the loss by updating weights and bias. Optimizers are of different types. The optimizer which we are using is referred to as adam optimizer.

**Adam Optimizer**

Adam optimizer is state of the art algorithm. It increases the speed of gradient descent. It is very effective when using a data set which is very large in size. It uses both momentum concept i.e., exponential weighted average as well as dynamically changing learning rate. It is very fast and resource effective. It is a combination of both gradient descent with momentum and Root mean squared propagation.

## Momentum Concept

$$w_{t+1} = w_t - \alpha m_t$$

where,

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta w_t}\right]$$

$m_t$ = aggregate of gradients at time t [current] (initially, $m_t$ = 0)

$m_{t-1}$ = aggregate of gradients at time t-1 [previous]

$W_t$ = weights at time t

$W_{t+1}$ = weights at time t+1

$\alpha_t$ = learning rate at time t

∂L = derivative of Loss Function

$\partial W_t$ = derivative of weights at time t

β = Moving average parameter (const, 0.9)

## RMS Prop

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right]$$

where,

$$v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

$W_t$ = weights at time t

$W_{t+1}$ = weights at time t+1

$\alpha_t$ = learning rate at time t

∂L = derivative of Loss Function

$\partial W_t$ = derivative of weights at time t

$V_t$ = sum of square of past gradients. [i.e sum(∂L/∂Wt-1)] (initially, $V_t$ = 0)

β = Moving average parameter (const, 0.9)

ϵ = A small positive constant ($10^{-8}$)

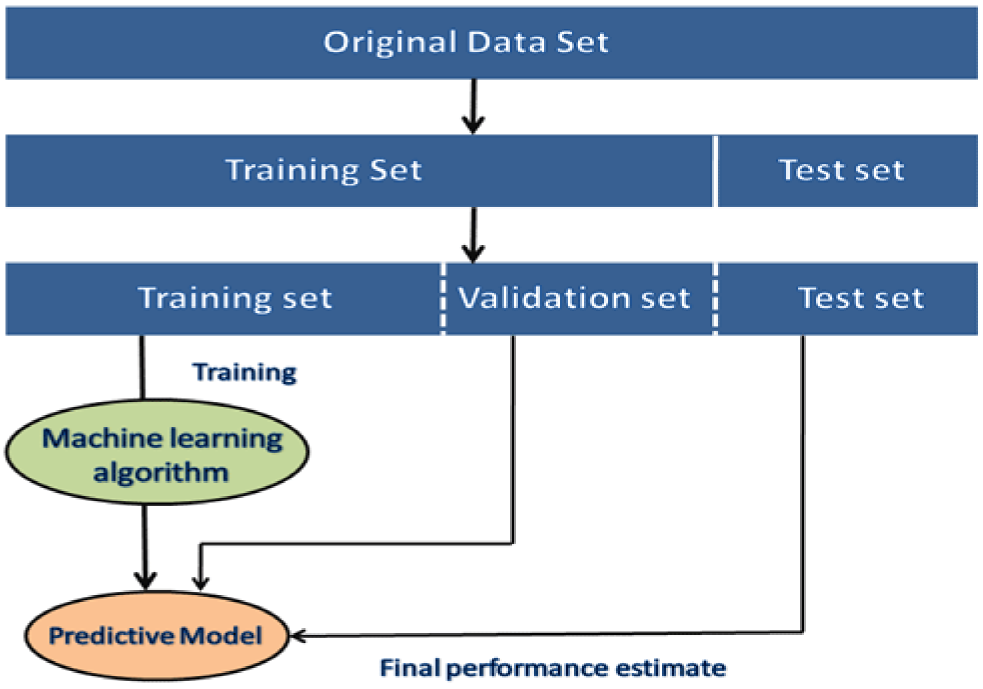
Combining both above equations

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

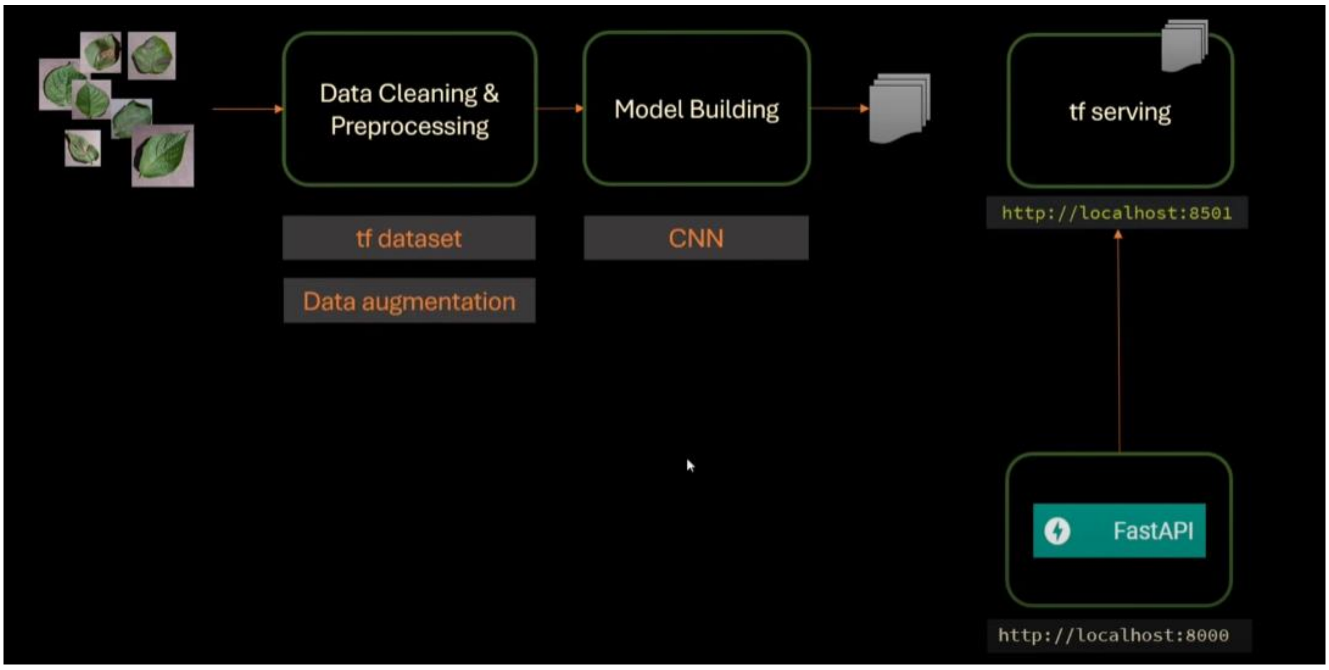
## 3.2 Model Development

### State Transition Diagram of the Major Project

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.

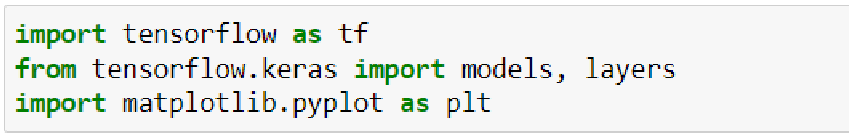


After collecting the data, first step is data cleaning and pre-processing for which I am using tf dataset and data augmentation. Data augmentation used when dataset is not that large then we can just rotate or increase decrease the contrast of the image to create a new image which can be used as an input. 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.



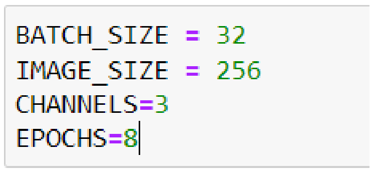
## 3.3 Analytical

First of all, importing all the dependencies



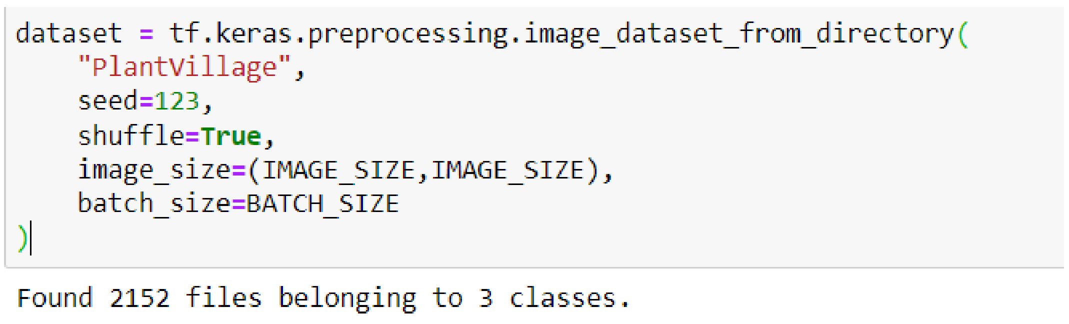
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

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
```

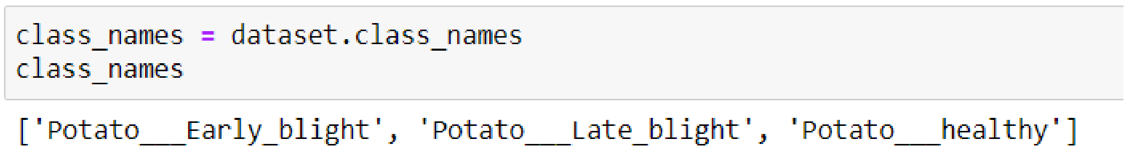
We will use keras to create an input pipeline. In which we have to specify the directory in which images are there. There are three different folders separated according to their classes. Folder name will be considered as the class of each image present in that folder. As all of the images are of (256x256). 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.
```

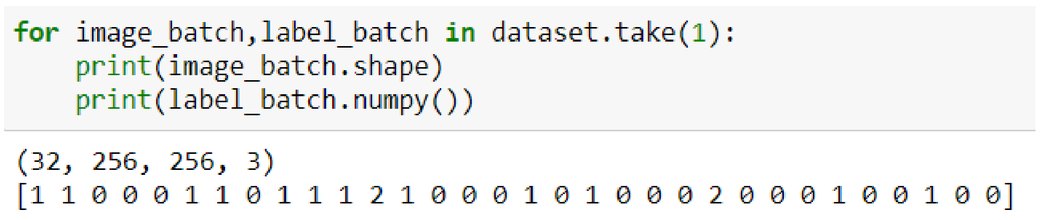
There are three classes as show below



```
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.



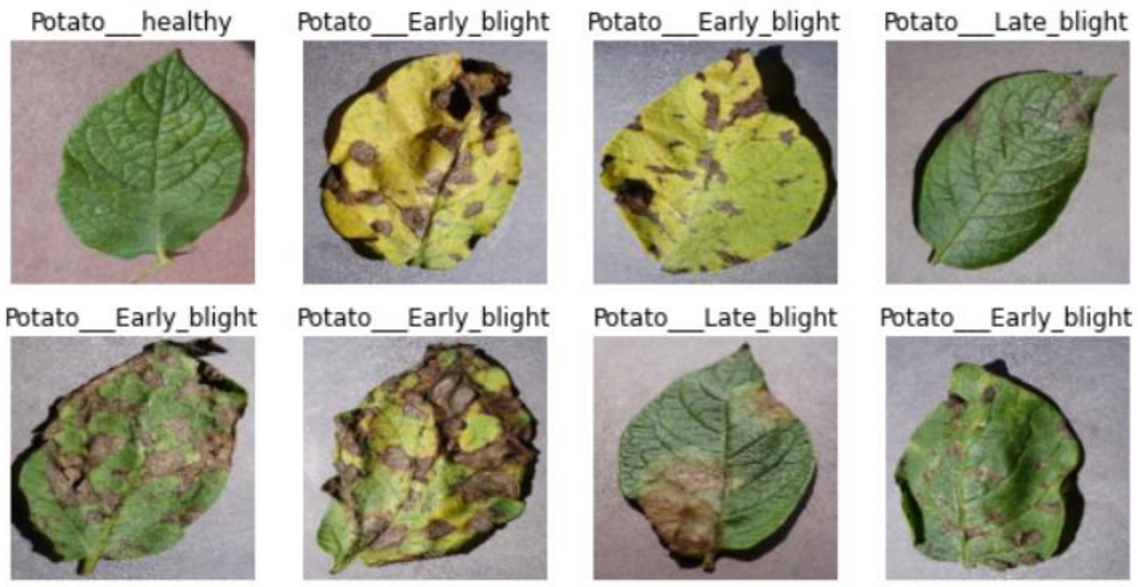
```
for image_batch,label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

```
(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")
```

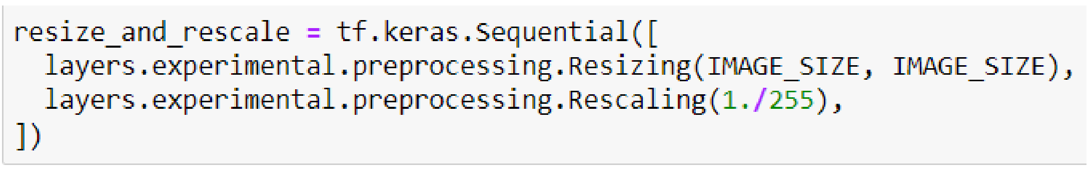


For splitting the data into training, testing and validation set a function is defined which takes dataset as an input. Other parameters have default values for train, validation and test split. There is also a shuffle parameter which will shuffle the images in the dataset to add randomness.

```
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
```

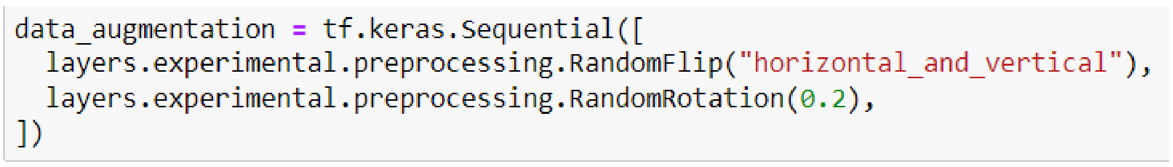
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

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.

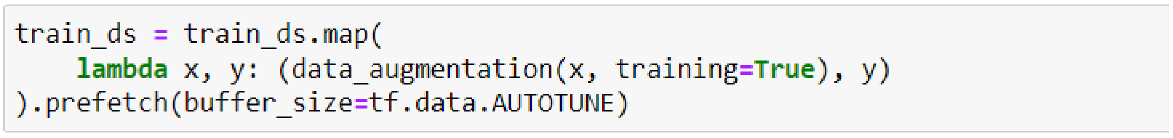


```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

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.



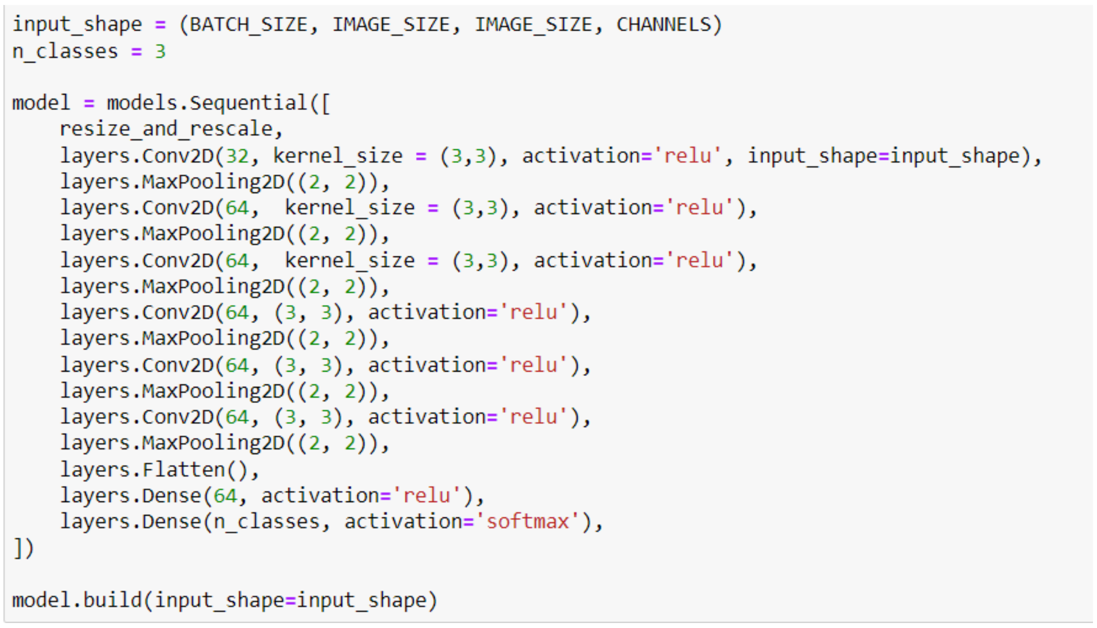
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```



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

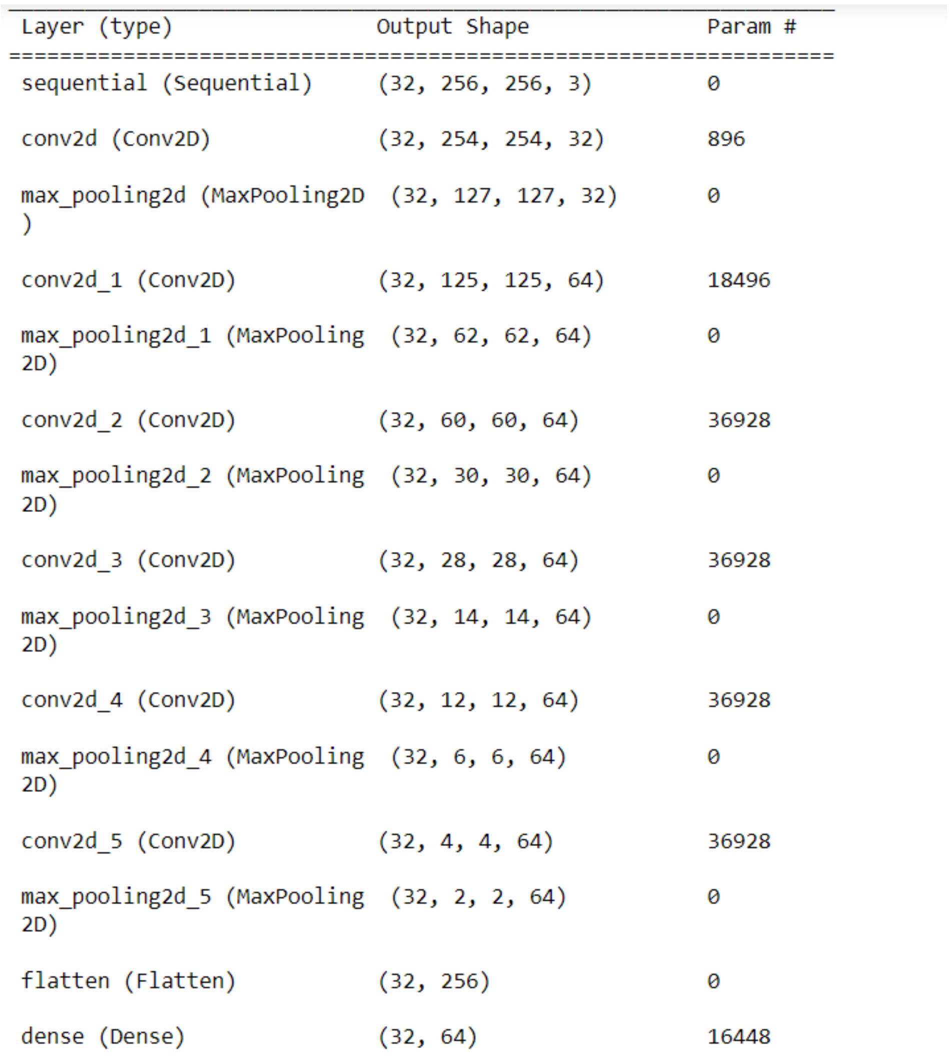
## 3.4 Experimental

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 wieghts and bias.

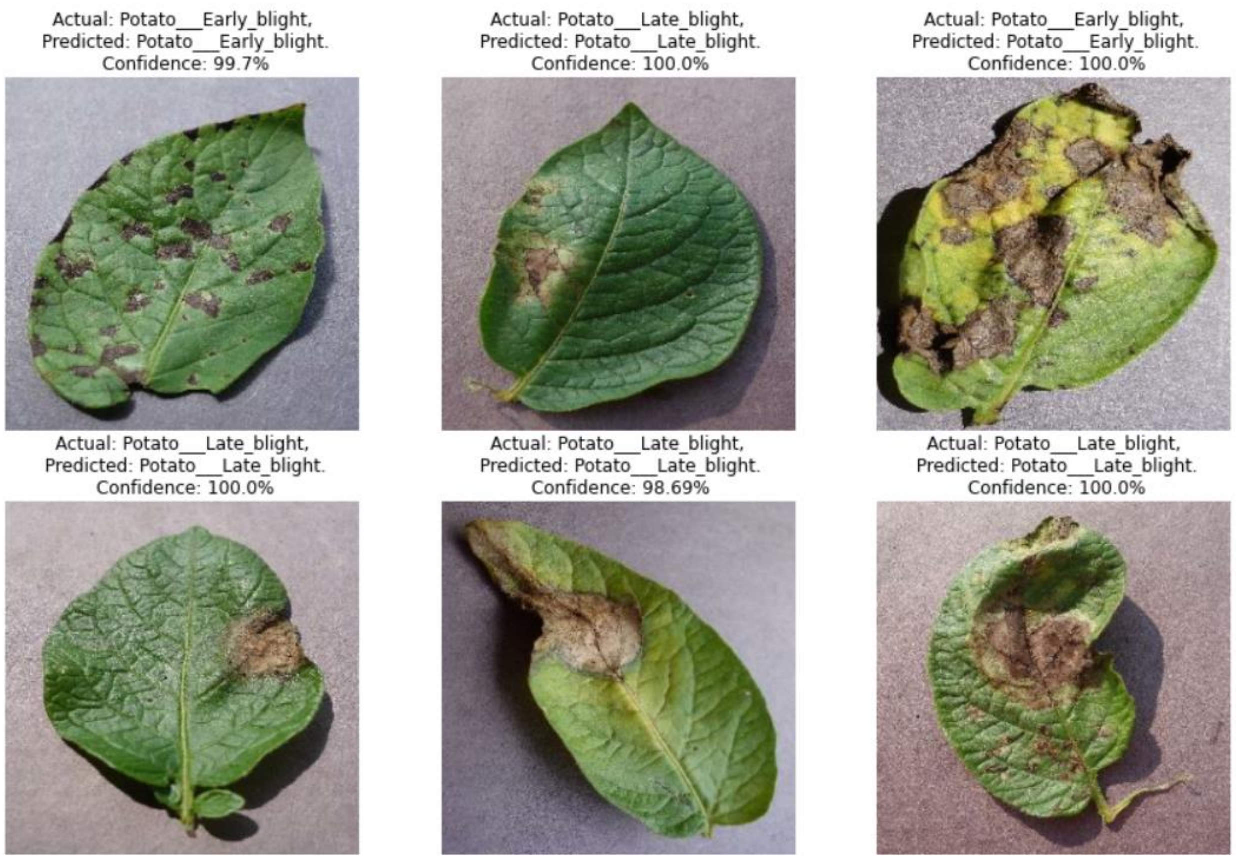
Accuracy metrics has been used for validating the loss in back propagation.

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

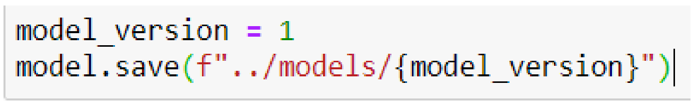
For prediction a function has been written which will take model and an input image as input parameters. Model will return an array which will contain the probability corresponding to each class from which we will select the probability having highest value. Below is the code:

```
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
```

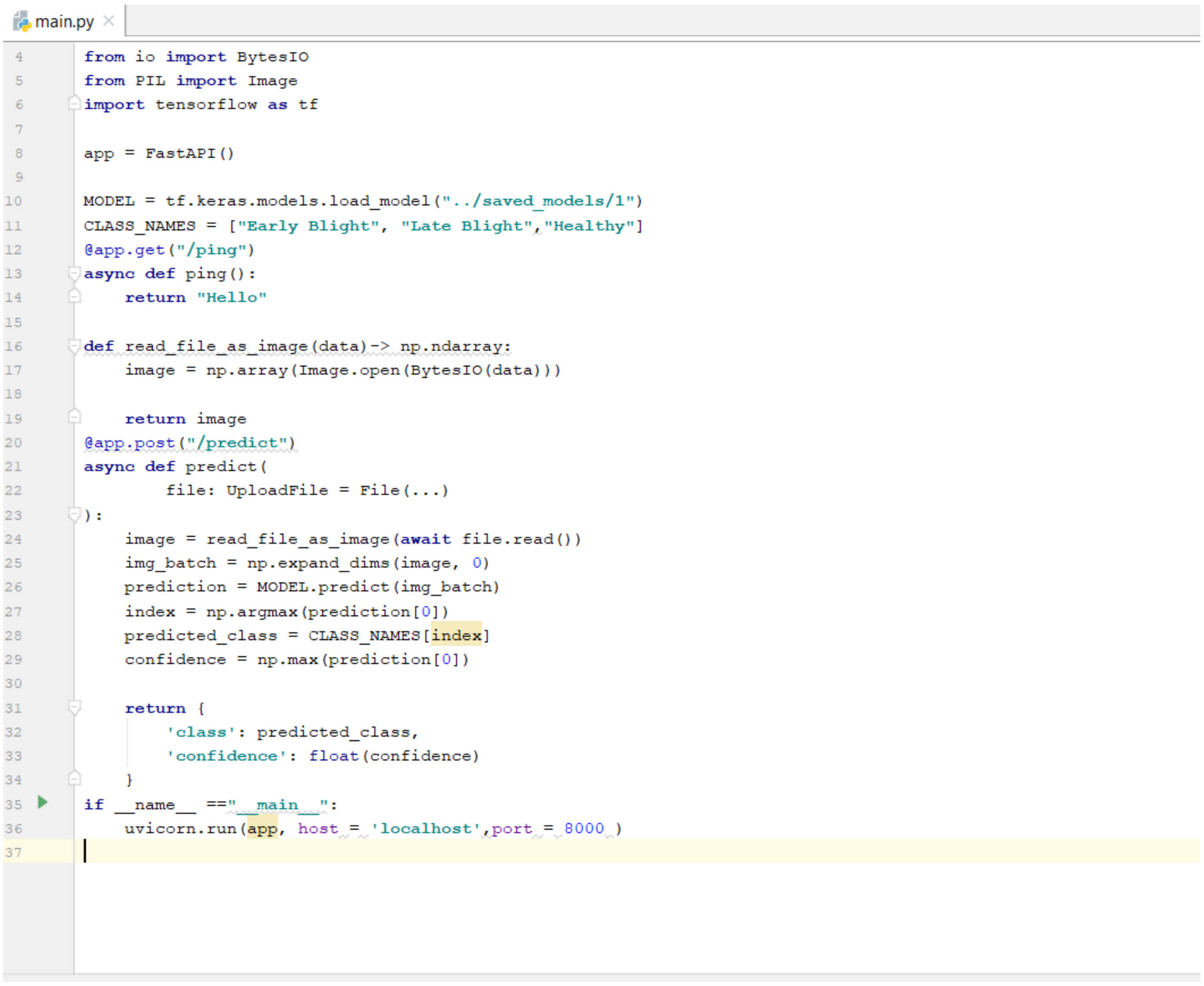
Predictions which we got after applying predict function are shown below:


Actual: Potato__Early_blight,
Predicted: Potato__Early_blight.
Confidence: 99.7%
Actual: Potato__Late_blight,
Predicted: Potato__Late_blight.
Confidence: 100.0%
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: 98.69%
Actual: Potato__Late_blight,
Predicted: Potato__Late_blight.
Confidence: 100.0%

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

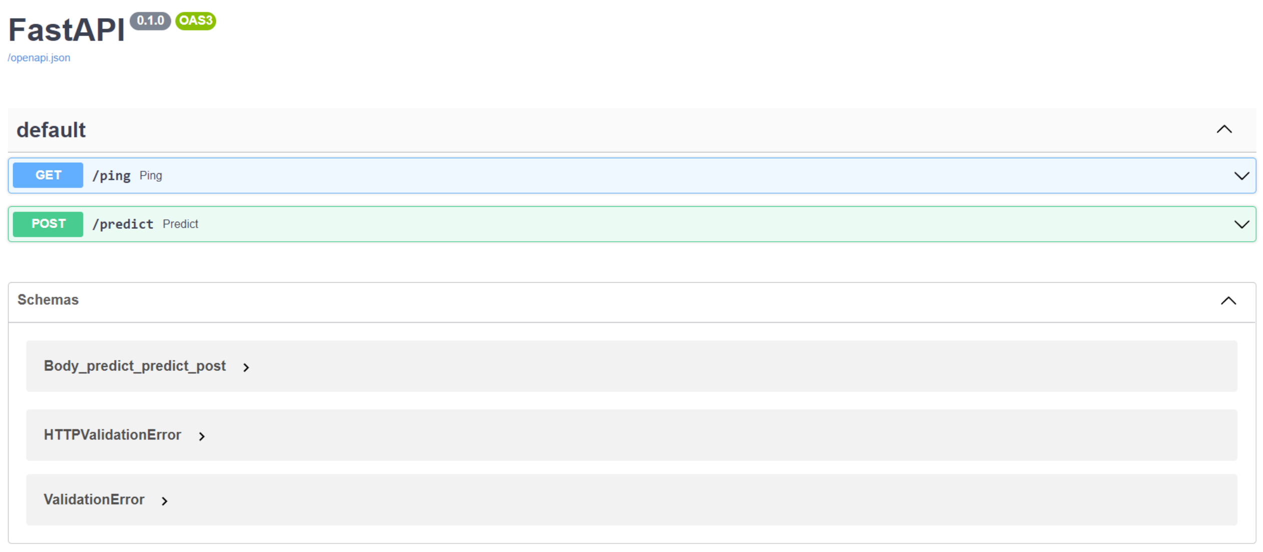

```python
model_version = 1
model.save(f"../models/{model_version}")
```

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:


main.py

```python
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

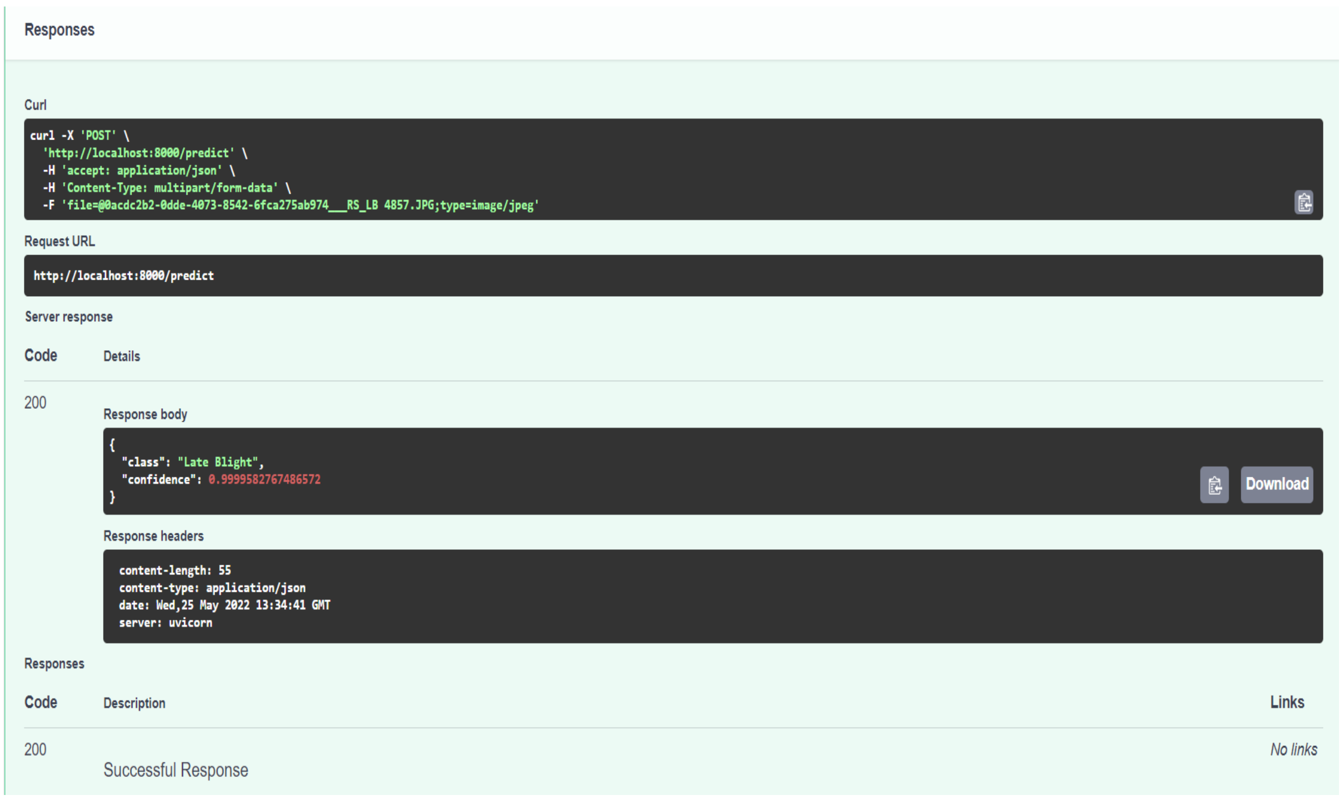
Schemas

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

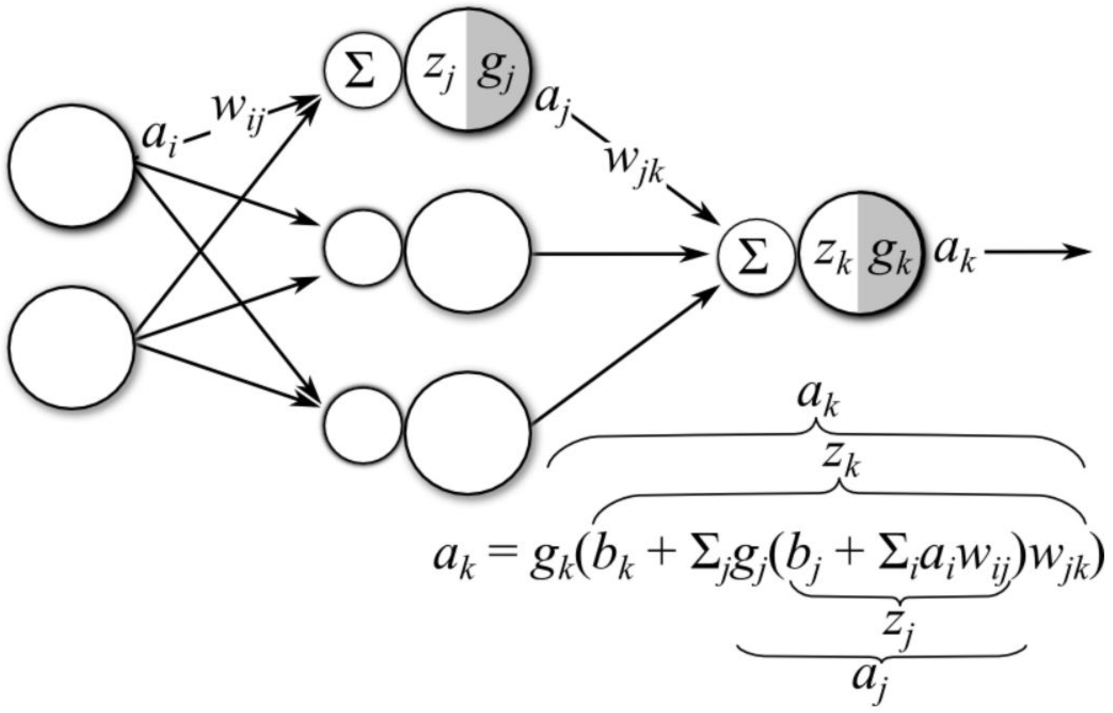
| Code | Description | Links |
|---|---|---|
| 200 | Successful Response | *No links* |

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

## 3.5 Mathematical

### Mathematics involved in Back Propagation

In, back propagation we try to reduce the value of loss function by updating weights and biases. In this we try to find gradient descent of a function in an optimized manner so that it can reach to the global minima early and don't the stick at local minima or a saddle point. It uses a chain of derivatives to update the weights and bias. Updation of parameters in back propagation is mainly carried out by derivatives.


$\Sigma$
$z_j$ $g_j$
$a_i$
$w_{ij}$
$a_j$
$w_{jk}$
$\Sigma$
$z_k$ $g_k$
$a_k$
$$a_k = \overbrace{g_k(\underbrace{b_k + \Sigma_j g_j(\underbrace{b_j + \Sigma_i a_i w_{ij}}_{z_j}) w_{jk}}_{})}^{a_k}$$
$z_k$
$a_j$

### Chain Rule

If there is a function in which its variables are also a function in those cases chain rule is used to calculate the derivative. Let's suppose we want to differentiate a function C with respect to W but C is a function of Z and Z is a function of W then in such cases chain rule of differentiation is used as shown below.

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial z}\frac{\partial z}{\partial w}$$

Suppose if C is a function of number of variables Z. Those variables are function of variable W then in such situation chain rule becomes additive as shown below.

$$\frac{\partial C}{\partial w} = \sum_{i=1}^{N} \frac{\partial C}{\partial z_i} \frac{\partial z_i}{\partial w}$$

If want to train our model using gradient descent then we should know derivative of every parameter with respect to the loss function i.e.

$$\frac{\partial C}{\partial W^m} \text{ and } \frac{\partial C}{\partial b^m}$$

**Convergence Theorem:**

$W_{new} = W_{old} - n * \partial L/\partial W_{old}$

$B_{new}=B_{old}-n*\partial L/\partial B_{old}$

where, $W_{new}$ is the new weight after updation

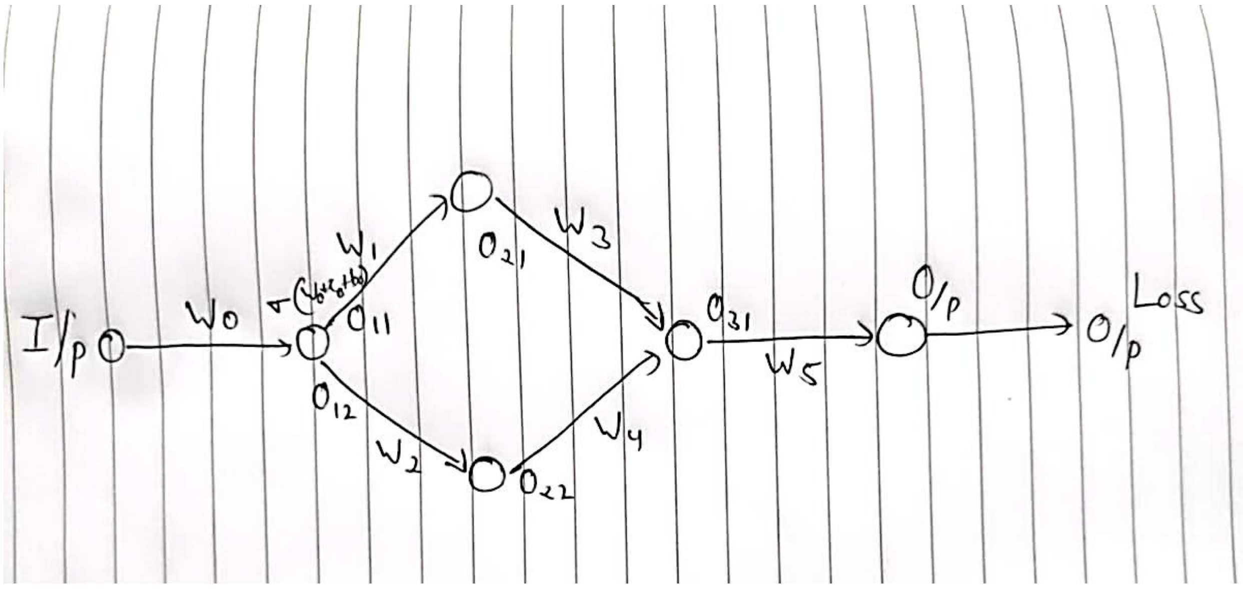
$W_{old}$ is the old weight before any updation

$B_{new}$ is the new bias after updation

$B_{old}$ is the old bias before any updation

n is the learning rate which is taken as very small number. It is a hyperparameter.

Let's calculate gradient of a weight in a very simple neural network. Below is the figure showing a neural network.



$W_0$, $W_1$, $W_2$, $W_3$, $W_4$, $W_5$ are the wieghts assigned to each hidden layer. In input layer let's suppose value x is passed then activation function is applied on $W_0 * X_0 + B_0$. Activation function is represented by sigma. Let's suppose $O_{11}$ and $O_{12}$ are the outputs received after applying activation function. These are then passsed to the corresponding layers as the input. Suppose, $O_{12}$ is passed to the layer then activation function is applied on $W_2 * O_{12} + B_{12}$ which is giving output as $O_{22}$. In this way all the ouputs are calculated as shown above in the figure.

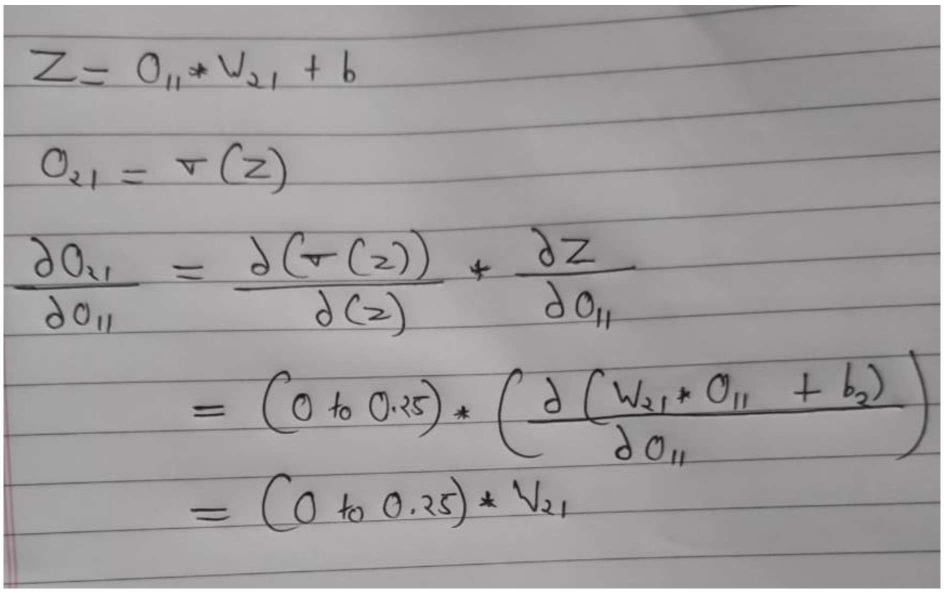
$$W_{new} = W_{old} - n * \partial L/\partial W_{old}$$

To get new weight first we need to compute gradient which will give new parameters in such a way that it will reduce the overall loss.

Below is the figure showing chain rule to calculate $\partial L/\partial W_0$.

$$\frac{\partial L}{\partial W_{0_{old}}} = \left( \frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial O_{11}} * \frac{\partial O_{11}}{\partial W_{0_{old}}} \right) + \left( \frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{22}} * \frac{\partial O_{22}}{\partial O_{12}} * \frac{\partial O_{12}}{\partial W_{0_{old}}} \right)$$

Let’s calculate the value for gradient $\partial O_{21}/\partial O_{11}$. Activation function is sigmoid.



$$Z = O_{11} * W_{21} + b$$

$$O_{21} = \sigma(Z)$$

$$\frac{\partial O_{21}}{\partial O_{11}} = \frac{\partial(\sigma(z))}{\partial(z)} * \frac{\partial Z}{\partial O_{11}}$$

$$= (0 \text{ to } 0.25) * \left(\frac{\partial(W_{21} * O_{11} + b_2)}{\partial O_{11}}\right)$$

$$= (0 \text{ to } 0.25) * W_{21}$$

Derivative of sigmoid function ranges from 0 to 0.25. If value of both derivative and weights or either of one is very less then value of gradient becomes very small due to which weight updation will be very small or negligible. This is termed as vanishing gradient problem. To prevent it, in the hidden layers Relu or its varients are used. Sigmoid is used in ouput layers having only two categories. For one or more categories softmax function is used.

# Chapter 4

## 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.

Specfcity = 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.

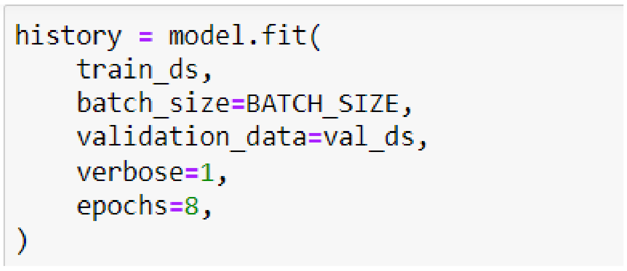
True negative = It is the count that contains correctly determined as counterfeit notes.

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)
```

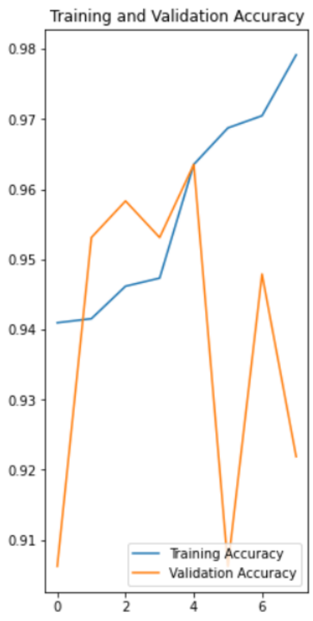
```
8/8 [==============================] - 6s 399ms/step - loss: 0.1160 - accuracy: 0.9414
```

For plotting training and validation accuracy, value at each steps are stored in the variables. Below is the code.

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

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

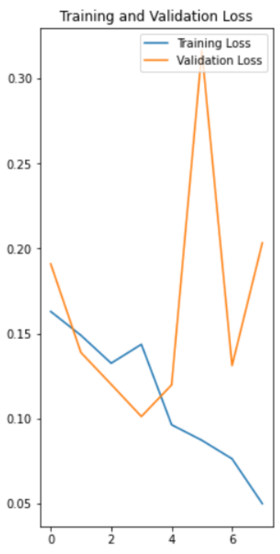
Below is the code and result of a graph between training accuracy vs validation accuracy.

```
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')
```



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

```
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')
```



# Chapter 5

## 5.1 Conclusions

In this project, convolutional neural network architecture is used in the model. So, that it can be used to classify for web application. Image of a potato leaf will be given as an input then it will be able to classify whether plant is healthy or effected by late blight and early blight. With little modifications, this project can prove very beneficial for farmers. Without monitoring on our own, we can easily identify the plant is infected or healthy. It will reduce the production cost as well as preventive measures can be taken early. This project can be modified so that it can be used for disease identification for other species as well. This can be done in real time and in a better way by using computer vision.

## 5.2 Future Scope

We will try out CNN architectures such as VGG, Res-Net, Inception Net and try to improve factors like accuracy.

A mobile application can be created which will make a call to FastAPI using cross origin resource sharing and will give the results.

Deploy this project on Heroku, AWS or various cloud platforms. So that it will be publicly available.

## 5.3 Applications

There are various applications of deep learning algorithms but talking specifically about this project given an image it can identify a plant whether it is healthy or infected. CNN networks can be trained to classify anything. With computer vision combined these architectures can produce amazing results.

After some improvements, this project can be very useful in agriculture sector. Combined with mobile application, it can produce amazing results.

Overall, this project may help not only in the agriculture sector but growth of overall country as we know that crops are one of the neediest things in the country.

## 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.