POTATO DISEASE CLASSIFICATION USING CNN

A

PROJECT REPORT

Submitted to Faculty of CSE/IT

Jaypee University of Information Technology, Waknaghat

In Partial Fulfilment of the Requirement for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

BY

HARSHIT KUMAR NARWAL [181471]

UNDER THE SUPERVISION OF

ASST. PROF. AAYUSH SHARMA

TO

Department of Computer Science & Engineering and Information Technology,

Jaypee University of Information Technology, Waknaghat,

Himachal Pradesh - 173234

TABLE OF CONTENTS

TITLE	PAGE NO.
DECLARATION	4
ACKNOWLEDGEMENT	5
CERTIFICATE	6
ABSTRACT	7
Chapter I INTRODUCTION	8
1.1 Problem Statement	8
1.2 Objectives	8
1.3 Methodology	9
Chapter II LITERATURE SURVEY	10
Chapter III SYSTEM – DEVELOPMENT	13
3.1 Development of Design and Algorithm	13
3.2 Developing the Model	20

3.3 Analytical Development **22**

3.4 Experimental Development **25**

3.5 Mathematical Development **28**

Chapter IV PERFORMANCE - ANALYSIS **33**

Chapter V CONCLUSION **37**

5.1 Future Scope **37**

5.2 Application **37**

REFERENCES **38**

DECLARATION

I, hereby declare that the project entitled, **'Potato Disease Classification Using CNN'** in the partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Information Technology has been done by me under the supervision of **Mr. Aayush Sharma, Assistant Professor**, Jaypee University of Information Technology, Waknaghat, Himachal Pradesh. I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any other degree or diploma.

Supervised by:

Aayush Sharma

Assistant Professor

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat

Submitted by:

Harshit Kumar Narwal (181471)

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat

ACKNOWLEDGEMENT

It is a genuine pleasure to express my deep sense of thanks and gratitude to those who have been helpful in the completion of this project.

First and foremost, I would like to express my deep, sincere gratitude and indebtedness to my mentor, supervisor, and guide ***Mr. Aayush Sharma, Assistant Professor,*** *Department of CSE/IT Jaypee University of Information Technology, Waknaghat who has a good knowledge in the field of Data Science and Machine Learning for allowing me to work under his invaluable guidance, vigilant supervision, and cordial attitude throughout this research. His scientific approach, timely and scholarly advice have helped me to a very great extent to accomplish this task.*

I also express my gratitude to ***Prof. Dr. Vivek Sehgal, HOD,*** *Department of CSE/IT Jaypee University of Information Technology, Waknaghat for his kind cooperation and for providing me the facilities without which the work would not have been completed.*

A special thanks my parents, my friends and the names remained unvoiced, for always standing beside me, listening to me, supporting me, encouraging me, and being a part of my life. Special thanks to various staff individuals, both educating and non-instructing, for taking out their precious time, helping me in completing the project and facilitate my undertaking.

Harshit Kumar Narwal [181471]

CERTIFICATE

This is to certify that the project report entitled **'Potato Disease Classification Using CNN'** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in 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 June 2024 to July 2024 under the supervision of **Mr. Aayush Sharma**, Assistant Professor, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat. It is further certified that this work has not been submitted in part or full for any other degree in this or any other University/Institution for the award of any degree.

Harshit Kumar Narwal [181471]

This is to certify that the above statement made by the candidate is true to the best of my knowledge and is recommended for acceptance for the award of the B.Tech. degree by the Jaypee University of Information Technology, Waknaghat.

Mr. Aayush Sharma	**Prof. Dr. Vivek Sehgal**
(Assistant Professor, CSE/IT)	(Head of Department, CSE/IT)
Jaypee University of Information Technology, Waknaghat	Jaypee University of Information Technology, Waknaghat

ABSTRACT

Potato is one of the widely staple foods worldwide, ranking fourth in terms of importance. During the COVID-19 pandemic, potato consumption surged significantly worldwide. However, potato crops are vulnerable to various diseases, which negatively affect both its yield and quality. Incorrect or delayed identification of these diseases can exacerbate the problem, leading to substantial financial losses for farmers. Identifying and treating these diseases early can help farmers reduce their losses. The condition of potato leaves provide important clues for identifying different plant diseases. In the current project, a Convolutional Neural Network (CNN) was employed to group potato plants according to their leaf health. The model was designed to categorize leaves into three classes i.e., late blight, early blight and healthy. Each leaf image was classified based on the confidence levels of the model. A simple CNN architecture was utilized for training and testing, and the primary goal was to develop a web API that accepts leaf images as input and classifies them according to the trained model.

CHAPTER I
INTRODUCTION

Potato is among of the most widely consumed crops globally. Ensuring stable output requires identifying and addressing potential threats to crop health. Two major pathogens, *Phytophthora infestans* and *Alternaria solani* are responsible for significant potato yield losses, causing the diseases known as Late Blight and Early Blight, respectively. Timely detection of such diseases can assist in implementing cautionary measures, thereby reducing both financial and yield losses. Traditionally, plant disease detection has relied on expert visual inspection, but this method can be impractical due to the shortage of experts in remote farming areas and the time required for manual assessments. Consequently, image processing technologies have emerged as an efficient solution for continuous plant health monitoring, early detection of the disease and to automate potato plant health classification, saving both time and resources. The main goal of this study is to develop a classification system using a Convolutional Neural Network (CNN) to detect potato diseases by analyzing leaf images.

1.1 Problem Statement

The goal of this project is to classify the health status of a potato plant incorporating a leaf image. This is a multi-class categorization issue, where a leaf can be categorized into one of the following classes:

Late Blight: The leaves are in an advanced stage of disease.

Early Blight: The leaves are at the initial stage of disease.

Healthy: The leaves show no signs of disease.

1.2 Objectives

Given the significant role of agriculture in the Indian economy, contributing 18% to the GDP and employing about 60% of the population, the need for advanced technologies to enhance crop production is crucial. Early disease detection allows farmers to apply timely interventions, ultimately boosting crop yield and revenue. Manual monitoring is time-consuming and requires field expertise, making it inefficient.

Hence, the objectives of the study are:

i To utilize CNN for the image classification.

ii To develop a web application that accepts an image as input and evaluates the health of potato leaves based on a pre-trained model.

1.3 Methodology

The project involves solving an image classification problem using a convolutional neural network. CNNs are preferred for their efficient resource utilization and superior accuracy compared to traditional neural networks. The model includes several convolutional layers, max-pooling layers and the activation functions, which are then followed by dense layers to improve accuracy. The ReLU activation function is used, and the Adam optimizer is employed for its dynamic learning rate and ability to reduce noise using exponential weighted averages.

CHAPTER II
LITERATURE SURVEY

Several algorithms have been developed for crop disease classification, including methods utilizing ultrasound, X-ray, and RGB imaging **(Wahabzada *et al.*, 2016)**. A different approach was introduced by **Macedo-Cruz *et al.* (2011)**, who assessed frost damage in oat crops by transforming the RGB colour space to L* a* b* colour space and applied three different thresholding techniques to analyze the data.

Medina *et al.* (2024) developed a mobile application for identifying diseases in potato crops using deep neural networks, based on images sourced from the Plant Village dataset, achieving 98.7% accuracy with the MobileNetv2 architecture. According to them, the offline app that was developed is compatible with Android 4.1 and latest versions, also include details on 27 potato diseases and a gallery of symptoms.

In another study by **Oppenheim and Shani (2017)**, an algorithm for potato disease classification was used through deep learning that utilizes unique visual symptoms and advances in computer vision. In their study, a deep CNN was trained to categorize potatoes into five classes: four disease and one healthy. The image dataset, containing various potato diseases, sizes and shapes, which were labelled manually by experts in their article. They trained the model on various train-test splits to evaluate the data required for effective classification using deep learning.

A significant role of machine vision and image processing in detecting defects in agricultural products, particularly potatoes was found by **Arshaghi *et al.* (2023).** In the article, increasing use of image processing and artificial intelligence was emphasized for detecting and categorizing pests and diseasesin agriculture. The authors applied convolutional neural networks (CNN) to classify five potato disease categories using a collection of 5000 distinct images. In their study, they examined the difference between the performance of the proposed model and other models, finding that their deep learning approach outperformed others, achieving 100% and 99% accuracy in some classes.

The study by **Rozaqi and Sunyoto (2020)** addressed the challenge of identifying disease in the potato leaves, specifically late blight early blight, which significantly affect crop yield. It proposed using deep learning with CNNs to identify these diseases using the leaf images, with data categorized into healthy, early blight, and late blight. They found that the model had an accuracy of 97% and 92% on training and validation data, respectively with a 70:30 train-test split performing better than an 80:20 split.

Sharma *et al.* (2022) in their study focused on improving plant disease diagnosis through deep learning to support sustainable agriculture in India. In their article, a CNN model was proposed to group rice and potato leaf diseases, using datasets of 5932 rice images and 1500 potato images. They found that the model secured an accuracy of 99.58% and 97.66% for the disease of rice and potato leaves, respectively, outperforming traditional methods of machine learning methods.

Another study by **Sharma *et al.* (2021)** developed a detection system for rice disease, utilizing real-time images gathered from the rice fields of Punjab and a deep learning model based on CNN. After preprocessing the dataset and splitting it into a 70:30 ratio, their model achieved 94% accuracy. Their research addressed lack of existing systems for detecting hispa disease, highlighting the need for efficient detection tools in agriculture.

A review by **Lu *et al.* (2021)** concentrated on the usage of deep learning, especially CNN, for the precise categorization of plant diseases, while tackling the shortcomings of conventional techniques such as visual observation and lab testing. They summarized the recent architectures of CNN relevant to the plant leaf disease classification, underlying deep learning principles and the main challenges faced along with their solutions. The study also discussed future directions for improving plant disease detection technologies.

The significant impact of the diseases related to rice leaf like hispa, leaf blast and brown spot on crop yield and quality in India were discussed by **Tejaswini *et al.* (2022).** It explored various approaches for deep and machine learning for identifying these diseases, evaluating their performance based on accuracy, recall, and precision. The results indicated that deep learning models outperform traditional methods, with a 5-layer convolution model achieving 78.2% accuracy, compared to 58.4% for VGG16, ultimately assisting farmers in improving crop health and yield.

Priyadharshini *et al.* (2019) in their study, introduced an adapted LeNet deep CNN architecture for classifying diseases in maize leaves, addressing the challenges of rapid disease identification crucial for food security. They trained the model to distinguish between four groups: one healthy and three other diseases leveraging images from the dataset of Plant

Village. They found that CNN achieved 97.89% accuracy, demonstrating the effectiveness of proposed method in identifying maize leaf diseases efficiently.

The challenge of accurately identifying rice leaf diseases in India, where farmers often lack the knowledge to diagnose these issues manually was taken by **Ghosal *et al.* (2020).** To overcome the scarcity of available datasets, the researchers created a small dataset and utilized Transfer Learning to develop a deep learning model according to the VGG-16. Their model, trained and tested on images collected from rice fields and online sources, achieved 92.46% accuracy, revealing the effectiveness of CNNs in automating disease recognition.

Another research by **Russel and Selvaraj (2022)** introduced a specialized parallel multiscale CNN architecture for plant disease detection, combining customized filters to describe texture patterns with trainable filters for adaptive learning. According to their methodology, the challenges that were addressed focused on disease intensity and image resolution by extracting features at different scales and positions on the leaves. The model achieved 99.17% accuracy for classification of plant species and 98.61% for classification of disease on the Plant Village dataset, with additional experiments on other datasets showing similarly high accuracies.

A study by **K.P and Anitha (2021)** focused on early detection of plant diseases to avoid the loss of crops in Indian agriculture, leveraging visible symptoms on leaves for accurate diagnosis. Traditional methods rely on manual observation by farmers or plant pathologists, which can be slow and subjective. It employed a deep learning approach using CNNs for image-based categorization of the disease of various plants. It compared performance of several models, including pre-trained networks like VGG, ResNet, and DenseNet, with the DenseNet model achieving the highest accuracy in disease classification, thus providing faster and more reliable predictions.

CHAPTER III

SYSTEM - DEVELOPMENT

3.1 Development of Design and Algorithm

3.1.1 Dataset Used

Every supervised machine learning project begins with the data collection phase, which typically involves three key steps:

i Collecting and annotating data independently.

ii Developing web scraping scripts to gather images from the internet.

iii Purchasing data from third-party vendors or utilizing public repositories, such as Kaggle.

For this project, the dataset is sourced from the Kaggle repository, consisting of images categorized into three distinct classes. You can access the dataset using the link https://www.kaggle.com/datasets/arjuntejaswi/plant-village?resource=download.

Plant Village dataset includes 3 kind of images as described in Figure 1.

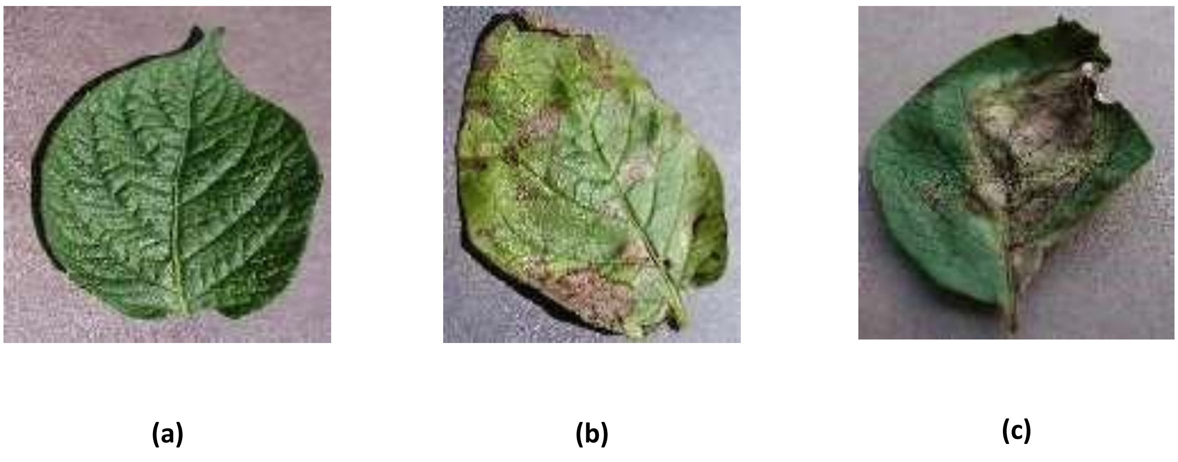


Figure 1: Plant Village Dataset Images of Potato Leaves

(a) Healthy (b) Early Blight (c) Late Blight

3.1.2 Pseudo Code/ Algorithms of the Research Problem

Defining Convolutional Neural Networks (CNNs)

The CNN is a type of neural network design that has always been an important method in computer processing and excels in image classification tasks (Yamashita *et al.*, 2018). Its architecture features additional layers, including pooling and convolutional layers, making them particularly effective for applications in computer vision, including face recognition and object detection.

Typically, CNNs comprise of several convolutional layers, followed by completely connected dense layers similar to traditional neural networks. They leverage the two-dimensional structure of images by employing tied weights and local connections. Max or mean pooling layers are incorporated to identify the most significant features from given dataset. The main advantage of using CNNs over conventional neural networks is their reduced number of parameters, which simplifies training and enhances efficiency.

Overview of Different Operations in CNN

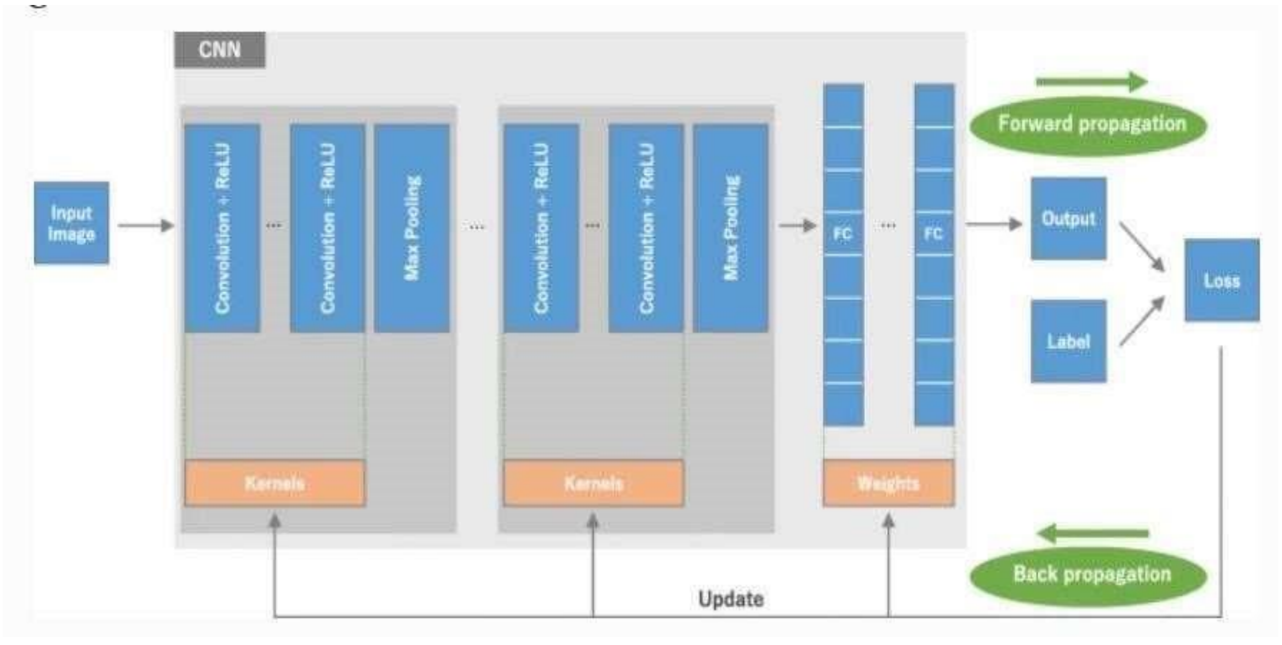


Figure 2: CNN architecture containing sequential operations

The CNN architecture illustrated in Figure 2 incorporates a series of sequential operations. This basic CNN begins with a convolutional layer that applies a kernel or filter to identify horizontal/ vertical edges, followed by an activation function named as ReLU. The subsequent layer is max pooling layer, which captures the most significant features from the image using a kernel, followed by the fully connected dense layers. Accuracy of the model is assessed using a loss function during forward propagation, and in backpropagation, weights, filters, and biases are updated based on the loss value, typically utilizing optimization methods.

Process of Convolution

In this particular technique, small tensors called kernels/filters are applied to the input image. The purpose of this function is to obtain essential features from the image. Consequently, a feature map is generated by conducting element-wise multiplication between the kernel and the image values. The feature map highlights the horizontal and vertical edges, which will be refined in the subsequent step.

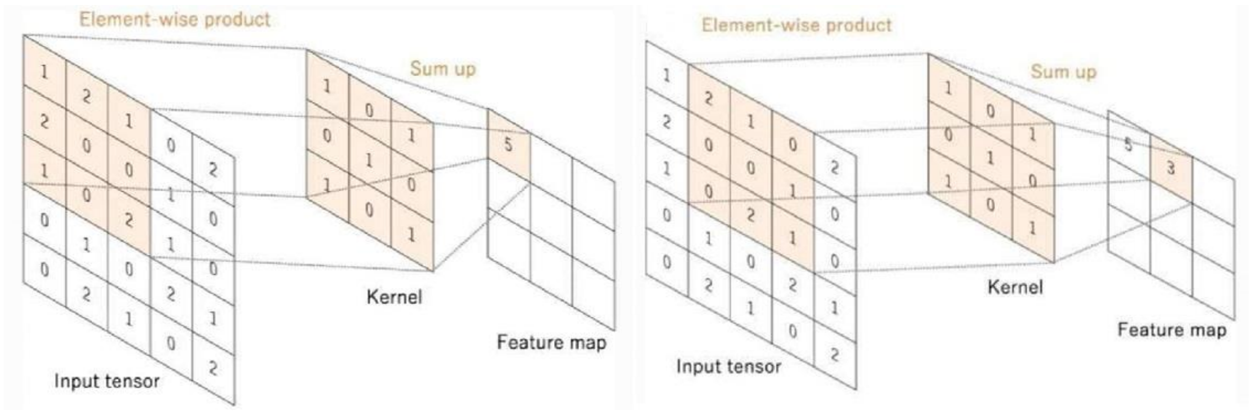


Figure 3: Convolution Operation on a 5x5 Image with No Padding

Figure 3 above illustrates a convolution operation performed on a 5x5 image using a 3x3 filter with no padding and a 1-stride. Without padding, the image size decreases, as shown in the figure. To preserve the original dimensions of the image, padding can be applied, which involves adding extra pixels around the edges.

Calculation of the Size of Image

$$\frac{(\boldsymbol{N}+\boldsymbol{2P}-\boldsymbol{F})}{\boldsymbol{S}+\boldsymbol{1}} \tag{1}$$

Where, N is the image size, P is the padding, F is the size of a kernel and S is stride.

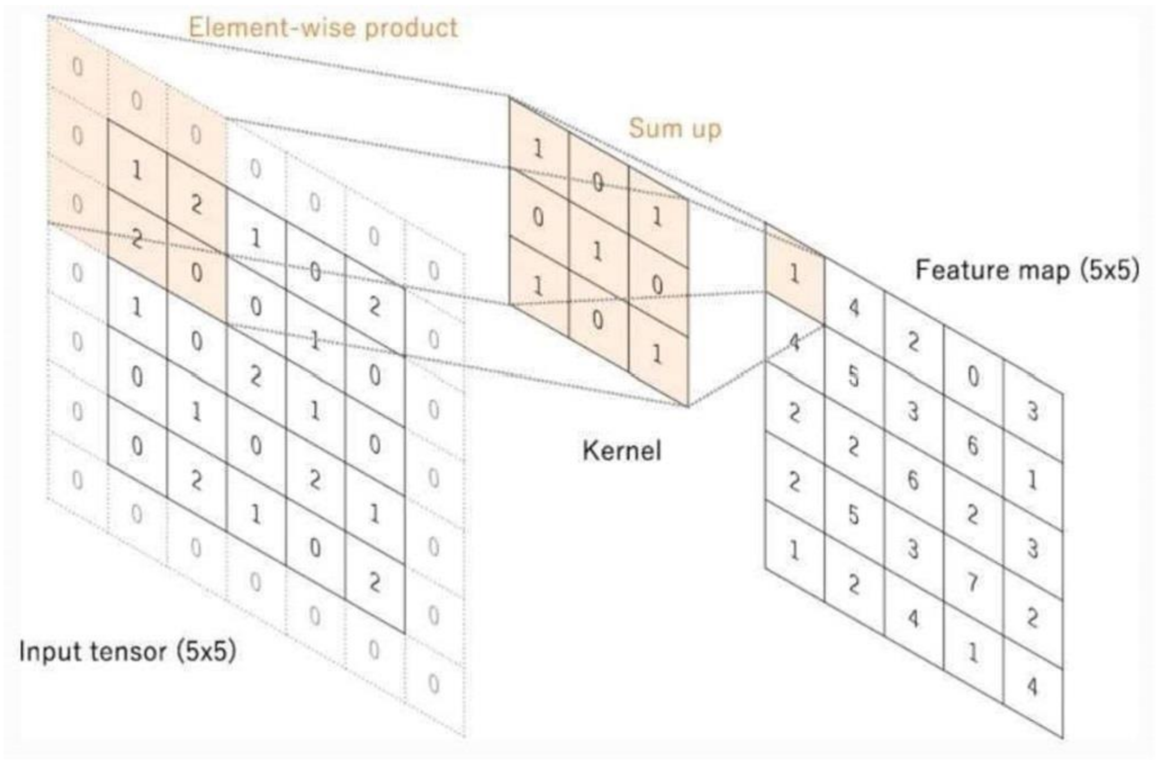


Figure 4: Convolution Operation with Padding

Usage of the Activation Function

The feature maps that are generated are operated on by a nonlinear activation function. These activation functions are non-linear, allowing for the introduction of non-linearity into the network. Sigmoid and tangent functions are typically avoided in the hidden layers due to potential for vanishing gradient issues. The ReLU activation function, which is the most frequently employed, is mathematically defined as the maximum of zero and its input value, i.e., $F(x) = \max(0, x)$.

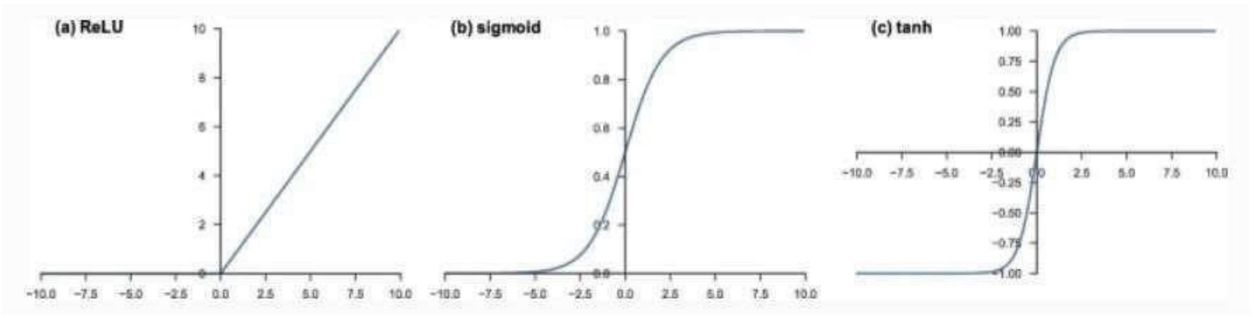


Figure 5: Activation Functions

Function of the Pooling Layer

The layer is used to extract the most important features. Its primary purpose is to diminish the number of trainable parameters. Although it decreases the spatial dimensions of the feature map, it preserves the depth (number of channels) of the input. (Nirthika *et al.*, 2022).

Function of the Max Pooling

A kernel is applied to the image in this layer to capture the highest value from each patch. Generally, a 2x2 size of the filter is utilized with a stride of 2 across the image. Figure 6 shows the function/ operation of max pooling.

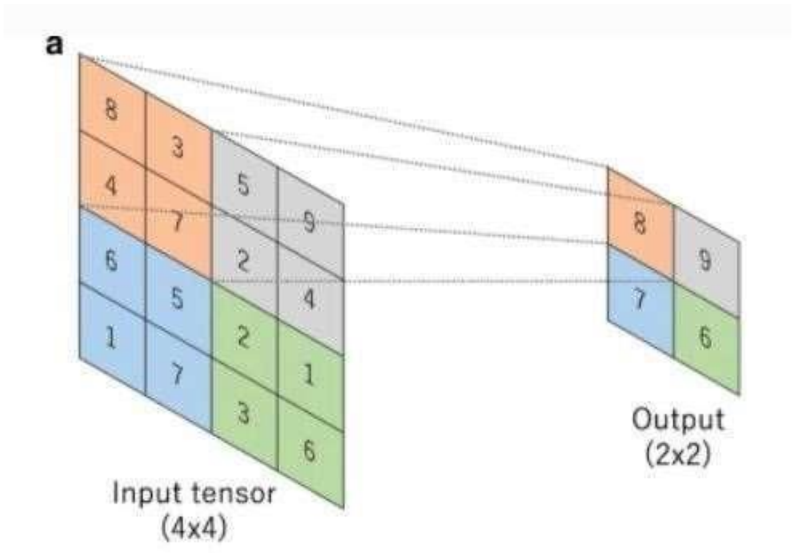


Figure 6: Max Pooling Operations

Function of a Fully Connected Dense Layer

The feature maps extracted from the max pooling layer are then flattened and fed into a fully connected dense layer. Prior to entering the dense layer, the feature maps are converted into a one-dimensional array format. The network may contain multiple dense layers stacked in succession. The final output layer is designed to match the number of target classes. Each hidden dense layer incorporates an activation function, with ReLU being the most prevalent choice.

Activation Function utilized at the Final Output Layer

The activation function employed in the last layer contrasts with those applied in the hidden dense layers, as latter primarily serve to introduce non-linearity. The activation function in the terminal layer is used for classification, providing the probability for each class. When there are two classes, the sigmoid function is typically employed. However, since this project involves more than two categories, the softmax function is used, normalizing the probabilities. Figure 7 illustrates the output.

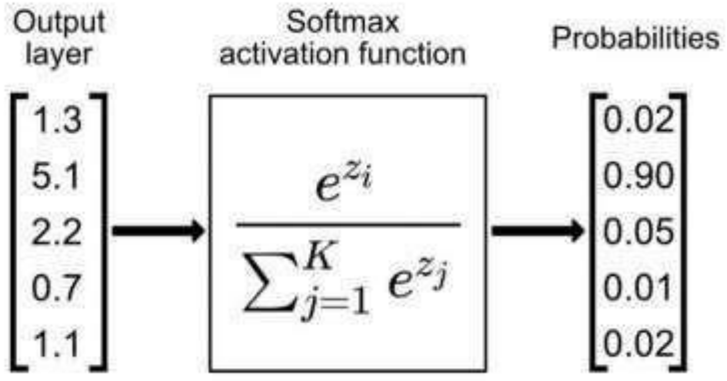

Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Figure 7: Output of a Sigmoid Function

During forward propagation, all weights, filters, and biases are computed. These parameters are then assessed using a cost function that quantifies the error, which is the discrepancy between the predicted and true label. The backpropagation phase involves updating the weights, filters and biases. An optimizer, an algorithm designed to minimize the loss, manages this updating process. While various optimizers are available, the Adam optimizer is utilized in this case.

Loss Function

The loss or cost function is a critical component that evaluates the precision of our forecasts by quantifying the discrepancy between the real and predicted labels. When handling data in batches or assessing all records collectively, the loss function is referred to as the cost function (Rajaraman *et al.*, 2021). Various types of cost functions exist, including multi-class and sparse categorical cross-entropy. Selecting the appropriate loss function is essential, as different functions can produce varying outcomes depending on the specific application.

Binary Cross Entropy

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

(2)

Multi Class Cross Entropy

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

(3)

Defining Optimizers

Optimizers are certain algorithms designed to reduce loss by adjusting the weights and biases. These are various types available and this study employs the Adam optimizer.

Functions of the Adam Optimizer

The Adam optimizer accelerates the gradient descent process. It is particularly useful when working with large datasets. The state of the art algorithm incorporates both the concept of momentum, using an exponentially weighted average, and an adaptively changing learning rate. Adam is known for being both fast and efficient in terms of resource usage as it combines the principles of gradient descent with root mean squared propagation and momentum.

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
α_t = learning rate at time t
∂L = derivative of Loss Function
∂W_t = derivative of weights at time t
β = Moving average parameter (const, 0.9)

(4)

RMS Propagation

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

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

α_t = learning rate at time t

∂L = derivative of Loss Function

∂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})

(5)

Combining equations (4) and (5),

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

(6)

3.2 Developing the Model

Diagram of State Transition

As data progresses through the model, its state changes at each stage. The process starts with data preprocessing, followed by partitioning the data into two subsets: the training set and the test set. The training set is employed to develop and train the model, while the test set is reserved for performance and accuracy assessment. Furthermore, the training set is subdivided into a subset for training and another for validation, utilized during the backpropagation (Mishra *et al.*, 2020). Upon completion of the training phase, accuracy of the model is evaluated using the test set, and then the model is subsequently saved and deployed for integration into an application.

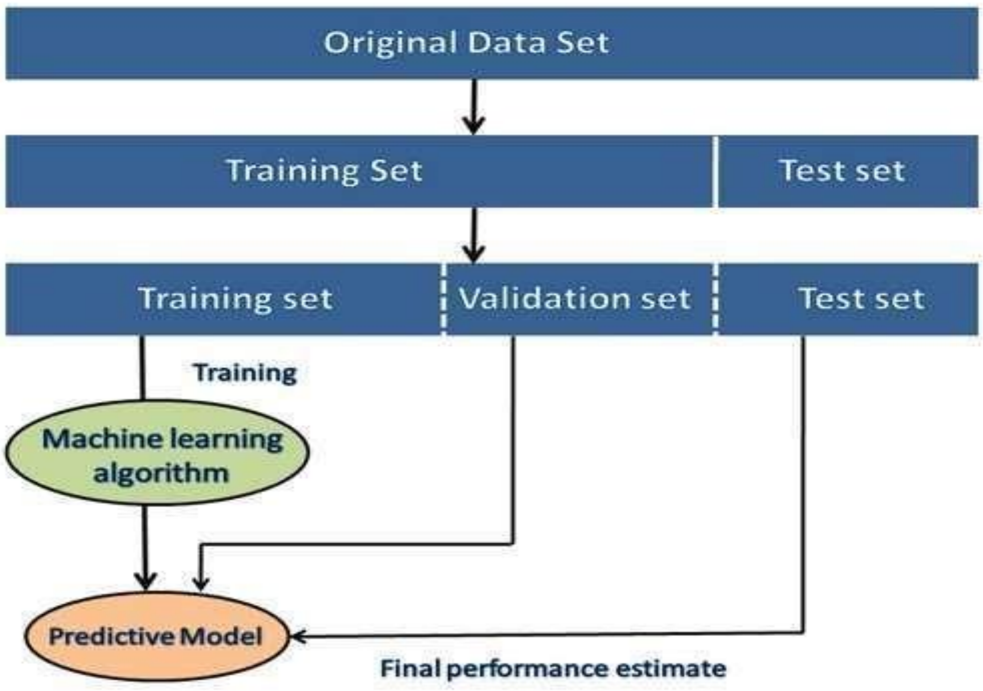


Figure 8: Steps of State Transition

Once the data is collected, the initial step involves data cleaning and preprocessing, utilizing TensorFlow datasets and data augmentation techniques. Utilizing data augmentation methodologies can prove invaluable when the underlying dataset is constrained. Approaches such as rotating imagery or manipulating contrast parameters can generate new images for input. The next phase is constructing the model with a CNN, which is commonly chosen for the image classification because it offers higher accuracy than conventional neural networks. After building the model, it is saved for deployment. Additionally, the backend of the web application is developed using FastAPI. Diagram of the State transition is shown in Figure 9.

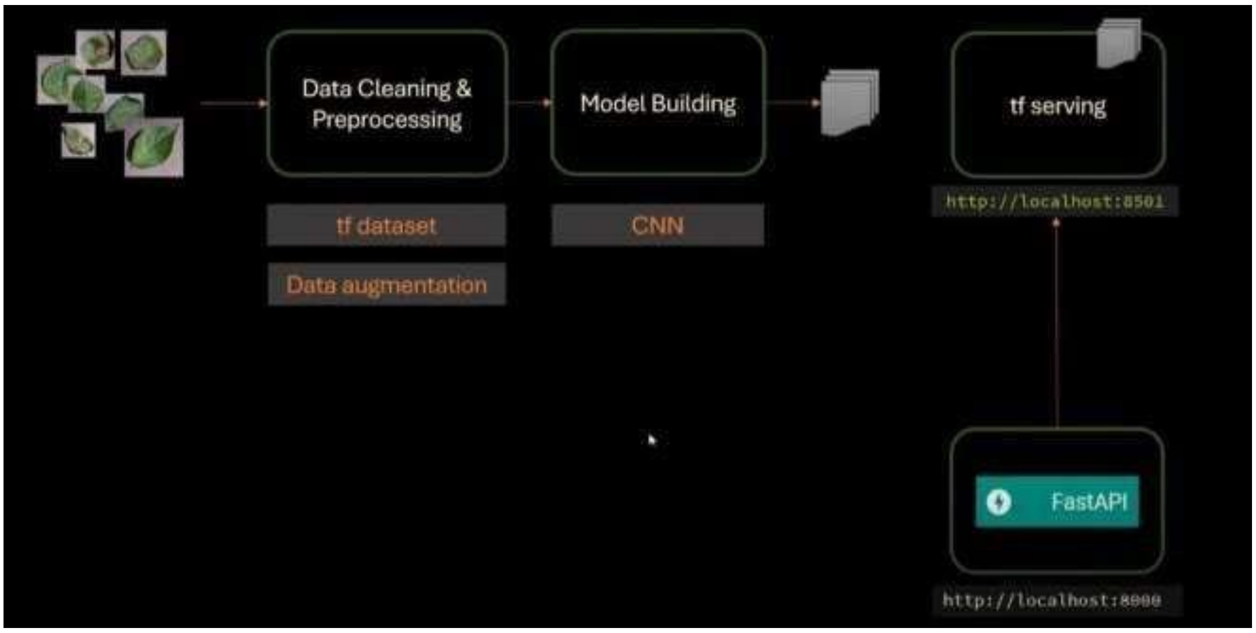
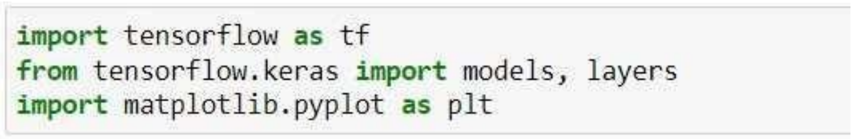


Figure 9: State Transition

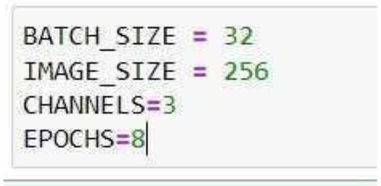
3.3 Analytical Development

Import all the dependencies at start.



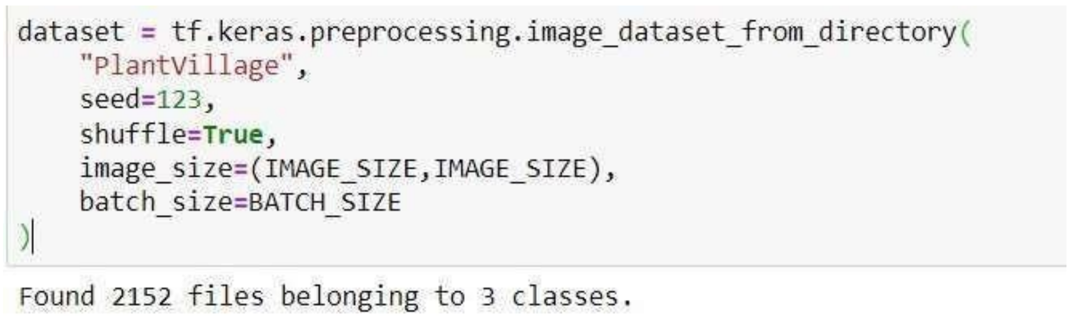
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

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
```

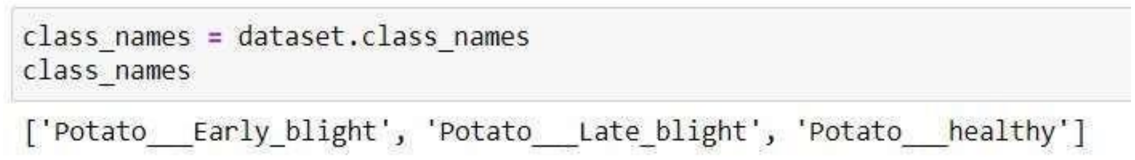
We will use keras to create an input pipeline. In which we have to specify the directory in which images are there. There are three different folders separated according to their classes. Folder name will be considered as the class of each image present in that folder. As all of the images are of (256x256).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

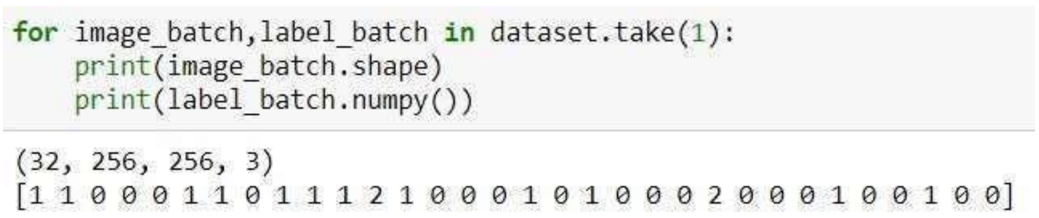
Three classes are shown below.



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

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

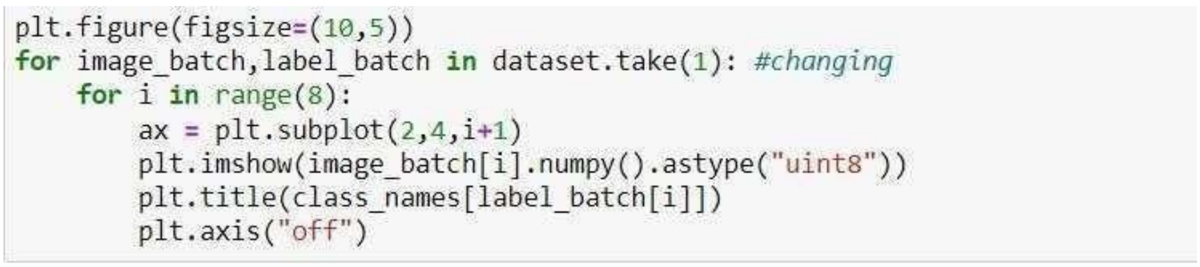
As shown below, each entry in the dataset is a tuple. The first element consists of a batch of 32 image samples and the second element contains a batch of 32 class labels.



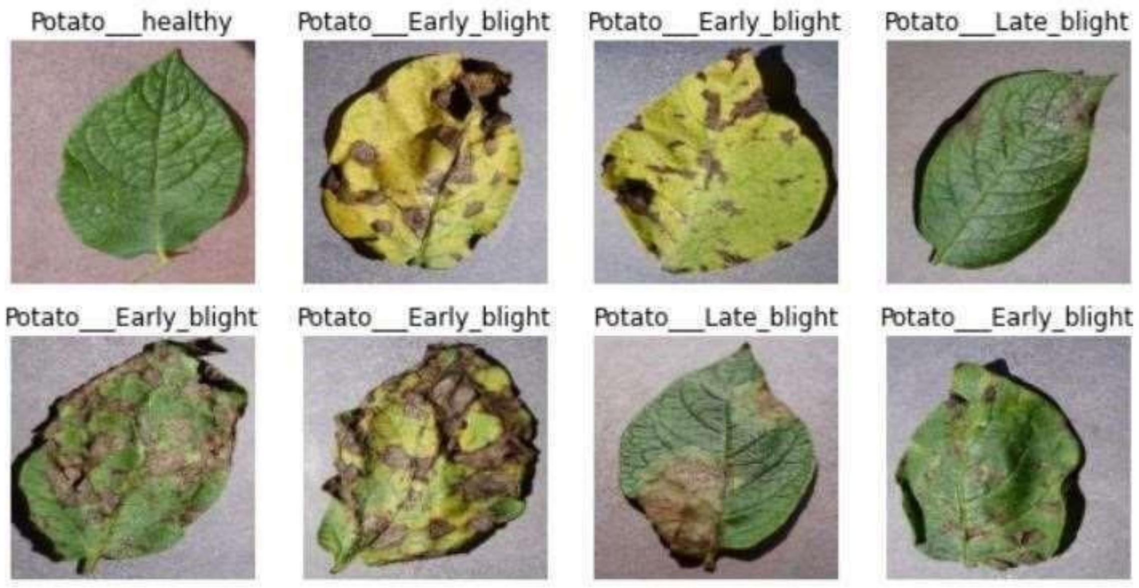
```
for image_batch,label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

```
(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]
```

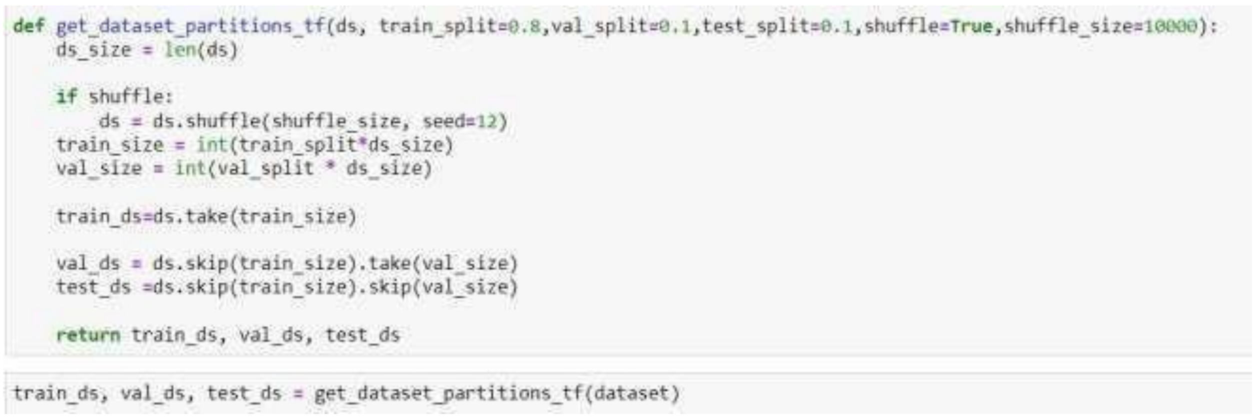
Our dataset is now balanced allowing us to build a model using this data. Plotting the images with their class label using Matplotlib.



```
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")
```



A function is defined to partition the dataset into training, testing, and validation sets, utilizing the dataset as 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)
```

To resize the input image to the required dimensions of 256 x 256, a layer has been defined utilizing the Keras. This layer will adjust the image size if it does not already meet these dimensions, allowing for effective training. Additionally, it will 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),
])
```

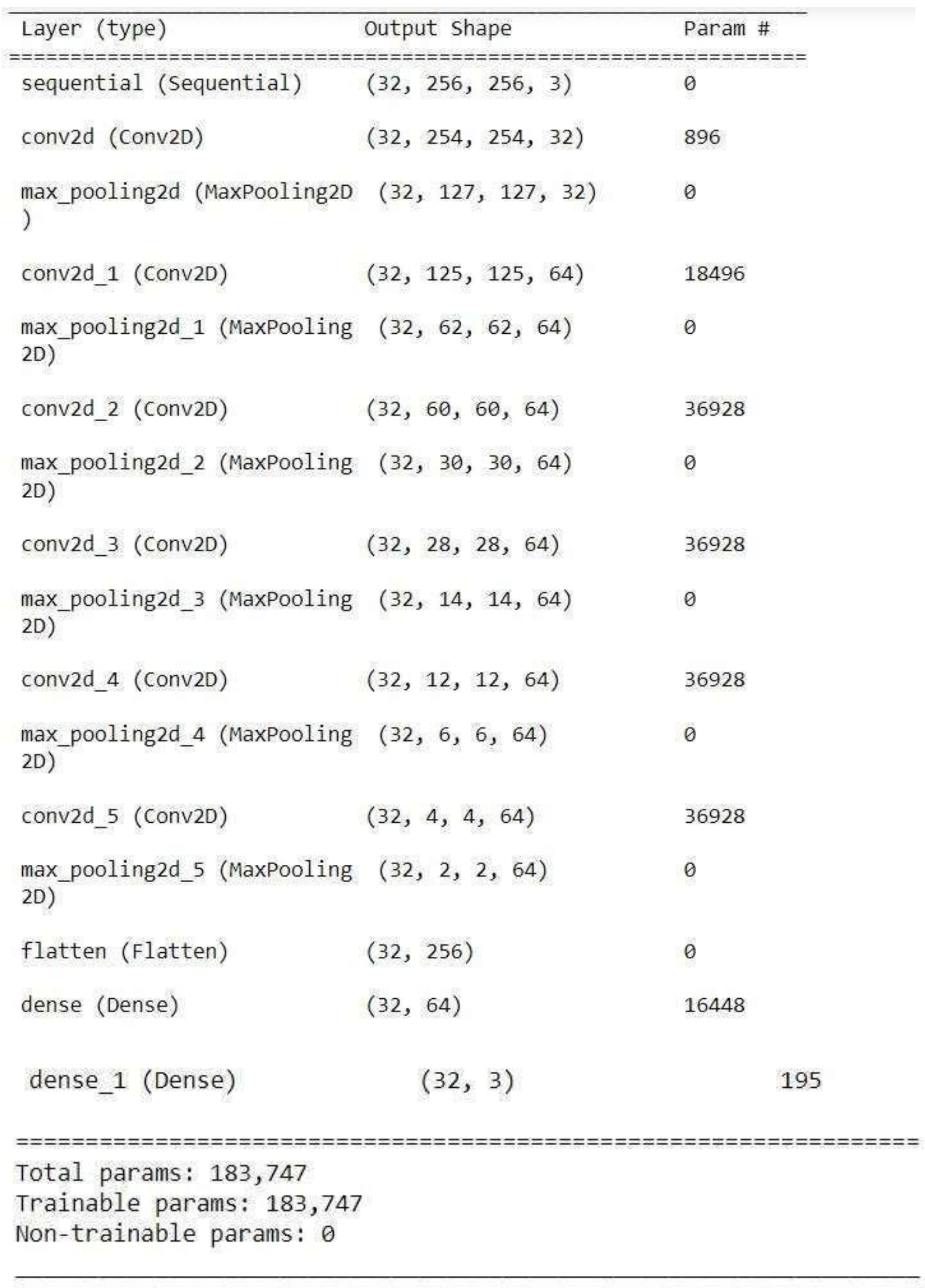
```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

3.4 Experimental Development

After the data preprocessing steps, we trained a simple convolutional neural network. It contains a resizing and rescaling layer followed by a number of max pooling as well as convolution layers, followed by a fully connected layer consisting of 64 neurons. At the end, there is output layer having three neurons as there are three different classes. The ReLU function has been applied in each hidden layer, whereas, softmax function has been employed in the output layer. The code is as following:

```
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 are shown above in the code. There are different kinds of loss functions. The current study employs the sparse categorical cross-entropy loss function and uses the Adam optimizer that 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']
)
```

A function has been created for making predictions, which accepts an input image and the model as parameters. This model will produce an array of probability values for each class, from which the highest probability will be identified. Predictions obtained from the predict function are displayed in Figure 10 and the code is provided below:

```
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
```

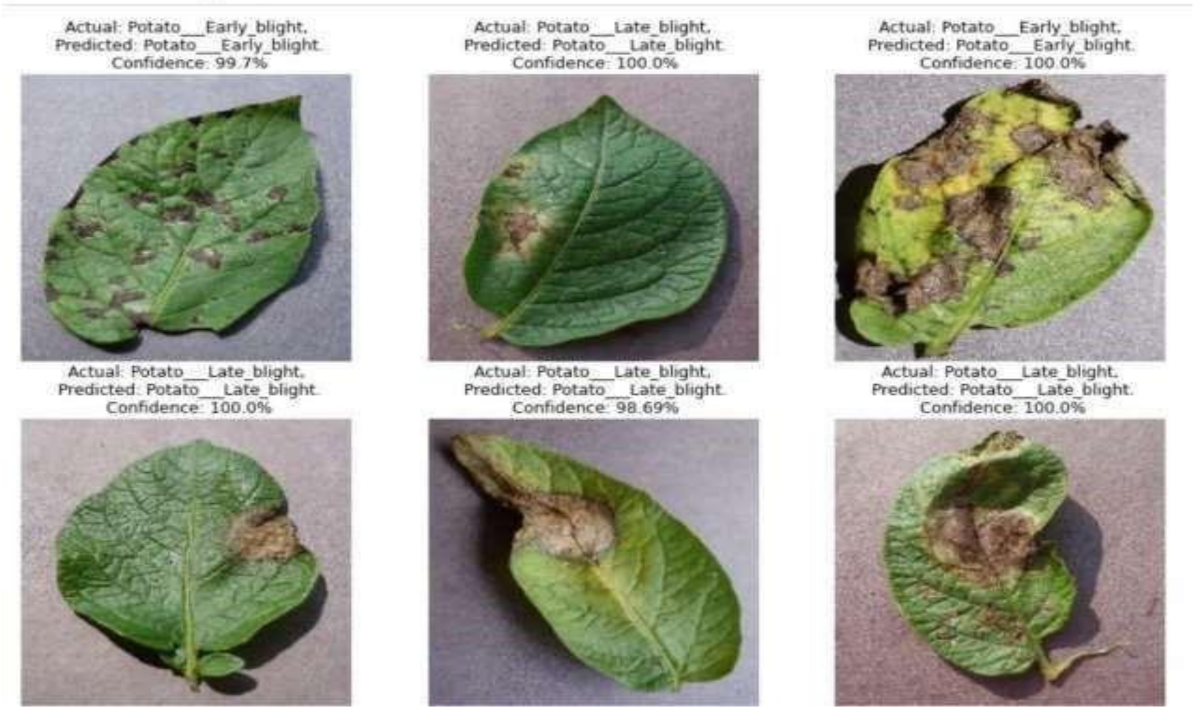

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.7%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
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%

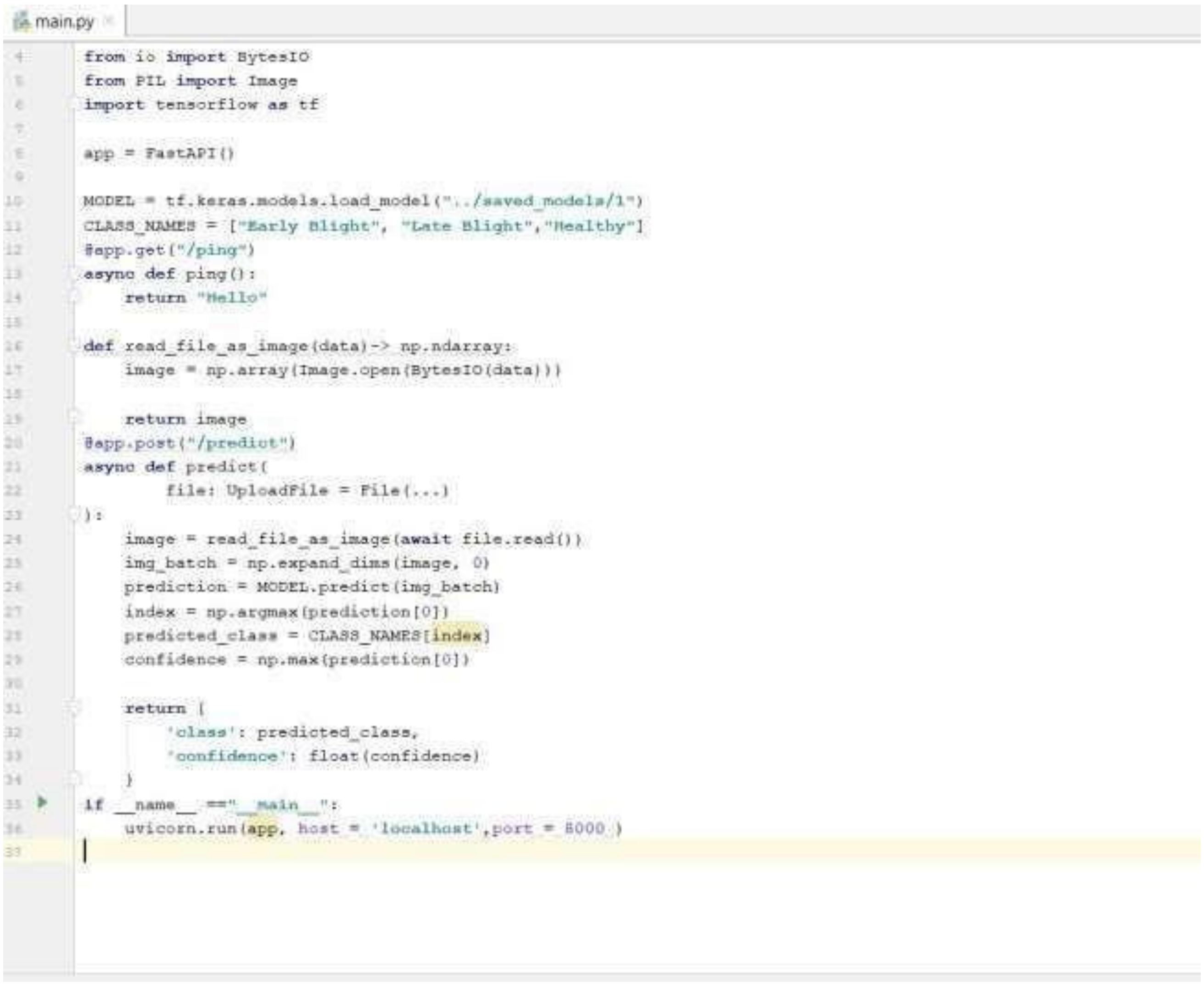
Figure 10: The Predictions

The model was saved later, so that it can be used for prediction in this web application.

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

FastAPI has been used to develop the web application, managing its backend functionality. It is among the emerging python framework which is used to create simple Rest APIs.

Below is the implemented code.



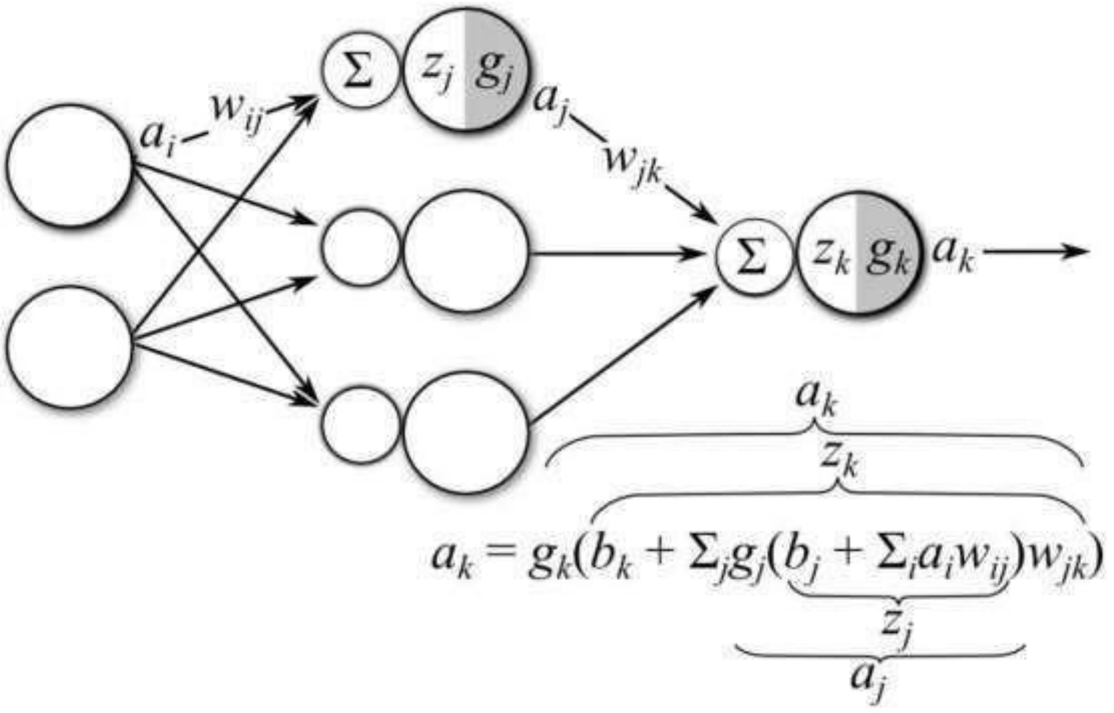
main.py

```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 )
```

3.5 Mathematical Development

The Mathematics Behind Back Propagation

Backpropagation aims to minimize the loss function value by updating the weights and biases (Puig-Arnavat & Carles Bruno, 2015). This process involves efficiently calculating the gradient descent of a particular function, enabling it reach the global minimum more quickly and avoiding getting stuck at local minima or saddle points. It uses a chain of derivatives to update the weights and bias. Updation of parameters in back propagation is mainly carried out by derivatives.


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

Chain Rule

When dealing with a function where its variables are also functions, the chain rule is employed to determine the derivative. For example, if the objective is to differentiate a function C with respect to W, where C is a function of Z and Z is a function of W, the chain rule of differentiation is utilized, as shown below.

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial z}\frac{\partial z}{\partial w} \tag{7}$$

As C depends on multiple variables Z, which are in turn functions of W, the chain rule will become additive, as described below.

$$\frac{\partial C}{\partial w} = \sum_{i=1}^{N} \frac{\partial C}{\partial z_i}\frac{\partial z_i}{\partial w} \tag{8}$$

To train our model using gradient descent, then the derivative of every parameter with respect to the loss function is described as;

$$\frac{\partial C}{\partial W^m} \text{ and } \frac{\partial C}{\partial b^m} \tag{9}$$

Convergence Theorem

$$\mathbf{W_{new} = W_{old} - n * \partial L/\partial W_{old}} \tag{10}$$

$$\mathbf{B_{new} = Bold - n * \partial L/\partial B_{old}} \tag{11}$$

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 and 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. Figure 11 shows a neural network.

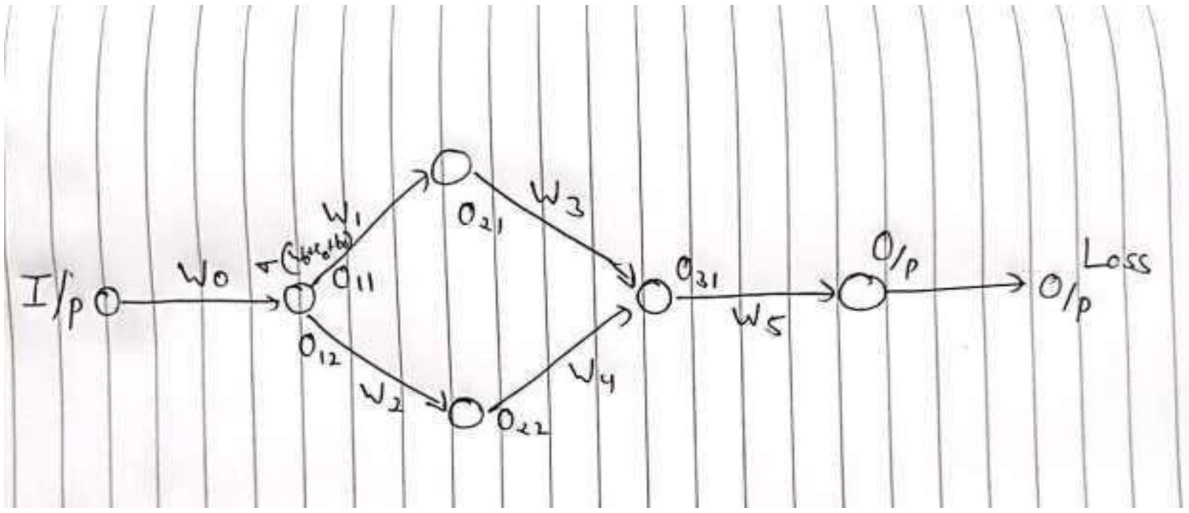


Figure 11: A Neural Network

W0, W1, W2, W3, W4, W5 are the weights 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 passed to the corresponding layers as the input. Suppose, O_{12} is fed into the layer, after which the activation function is applied to $W_2 * O_{12} + B_{12}$ which is giving output as O_{22}. In this way all the outputs are calculated as shown in the Figure 11. 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 equation showing chain rule to calculate $\partial L/\partial W_o$.

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

$$+ \left(\frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{22}} * \frac{\partial O_{22}}{\partial O_{12}} * \frac{\partial O_{12}}{\partial W_{O_{old}}} \right)$$

(12)

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

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

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

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

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

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

(13)

The derivative of the sigmoid function varies between 0 and 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 variants are used. Sigmoid is used in output layers having only two categories. For one or more categories softmax function is used.

After running the code, web application will run at port number 8000. Advantage of FastAPI is that without 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 user interface shown below in Figure 12.

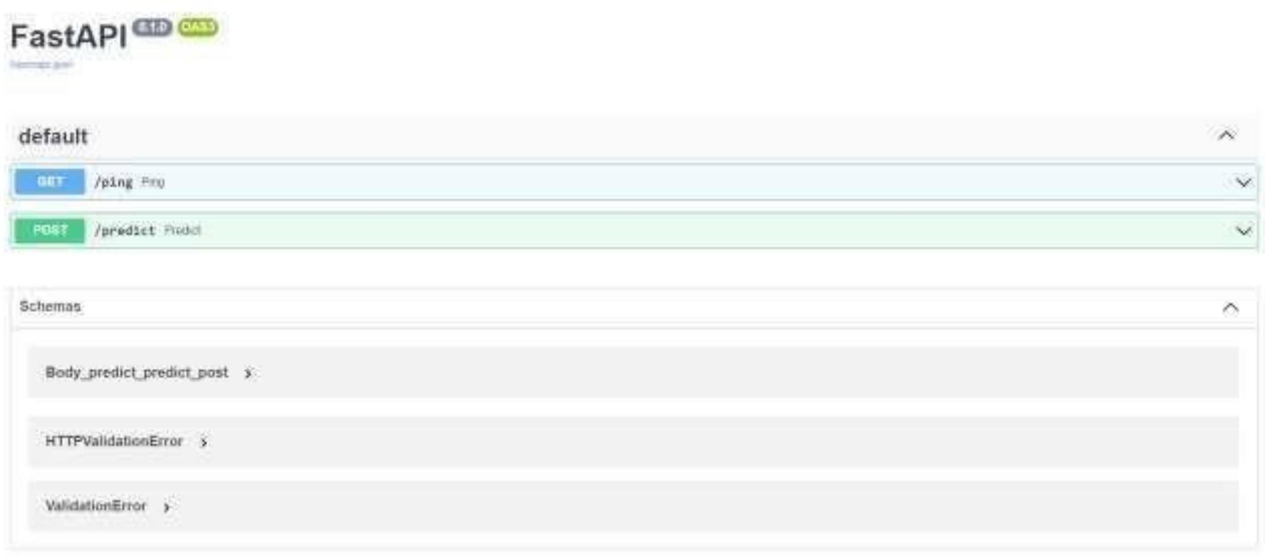
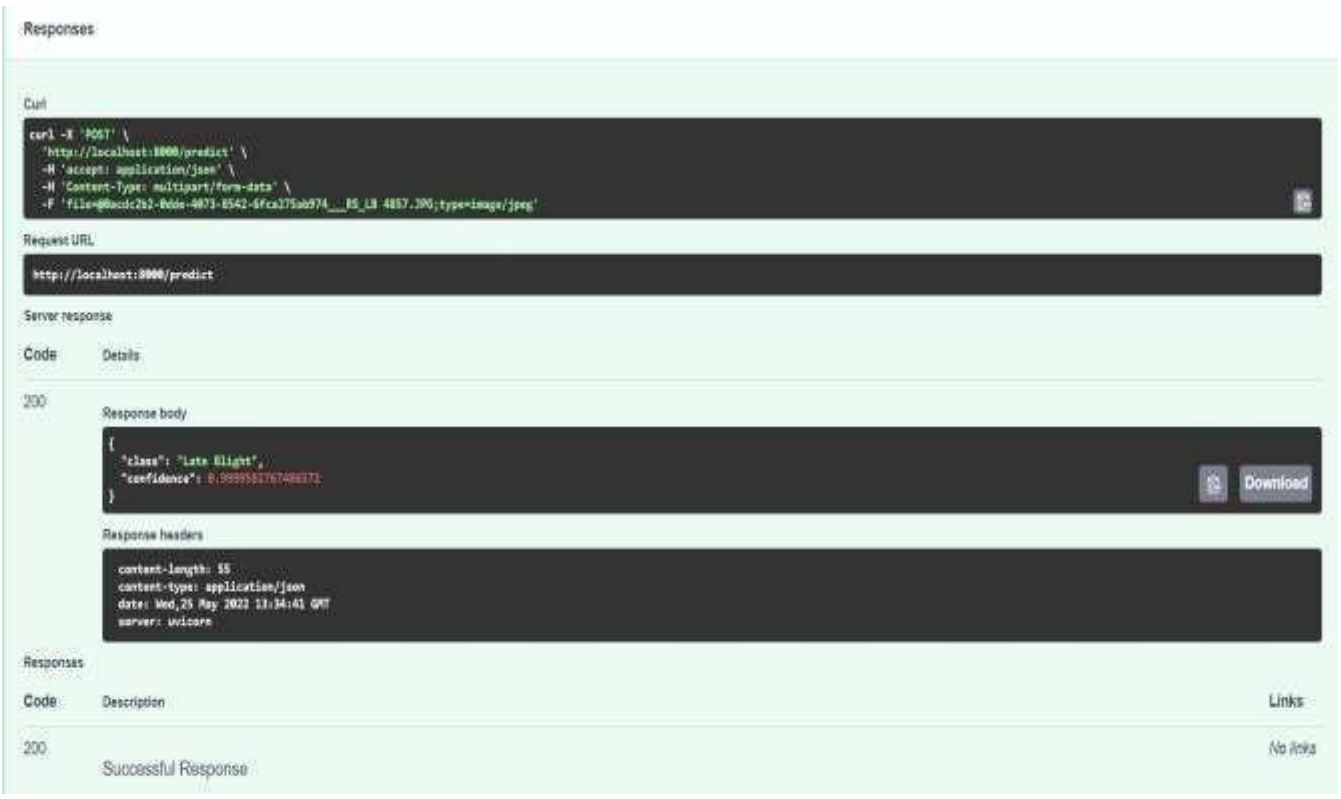


Figure 12: Fast API

In FastAPI, an upload file variable is there which is used to upload a file. This variable has been used to upload an image file and then model will classify image according to the probability. An input image has been classified in Figure 13.



```
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'
```

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

```
{
  "class": "Late Blight",
  "confidence": 0.9999581767486572
}
```

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

Figure 13: Image Classification

Figure 13 has been classified as a Late blight class with 0.99 confidence.

CHAPTER IV

PERFORMANCE - ANALYSIS

To evaluate the efficacy of the deployed models, following metrics were employed:

Accuracy: This metric evaluates the ability to correctly identify genuine versus counterfeit note test cases.

$$\frac{(\boldsymbol{tp}+\boldsymbol{tn})}{(\boldsymbol{tp}+\boldsymbol{tn}+\boldsymbol{fp}+\boldsymbol{fn})} \tag{14}$$

Sensitivity: This metric refers to the capability to correctly identify genuine note cases.

$$\frac{\boldsymbol{tp}}{(\boldsymbol{tp}+\boldsymbol{fn})} \tag{15}$$

Specificity: This metric refers to the test's ability to accurately recognize counterfeit notes.

$$\frac{\boldsymbol{tn}}{(\boldsymbol{tn}+\boldsymbol{fp})} \tag{16}$$

Precision: This measure reflects the accuracy of the test in determining whether the notes classified as genuine are indeed authentic.

$$\frac{\boldsymbol{tp}}{(\boldsymbol{tp}+\boldsymbol{fp})} \tag{17}$$

Where,

True positive represents the count of correctly identified genuine notes.

True negative indicates the count of accurately identified counterfeit notes.

False positive refers to the count of notes incorrectly classified as genuine.

False negative denotes the count of notes mistakenly identified as counterfeit.

We will now compare two algorithms: the Random Forest classifier and K Nearest Neighbors (KNN). The accuracy metric has been utilized for validation, allowing to assess the model’s performance during backpropagation while adjusting the parameters accordingly. The training process has been conducted over 8 epochs. The following code illustrates the model fitting process.

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

The figure below displays the training and validation loss results for each epoch.

```
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
```

The model achieved around 94% accuracy when assessed on a test dataset. The code and its results are provided below.

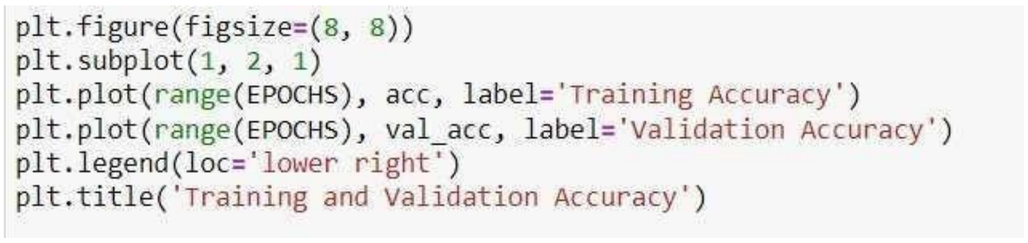
```
scores = model.evaluate(test_ds)
```

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

To plot the training and validation accuracy, values at every step are stored in the variables. The code for this process is shown below.

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

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

The code and the resulting graph comparing training accuracy to validation accuracy are presented in Figure 14.



```
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')
```

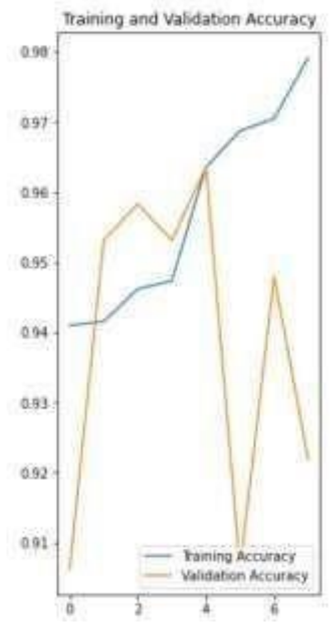
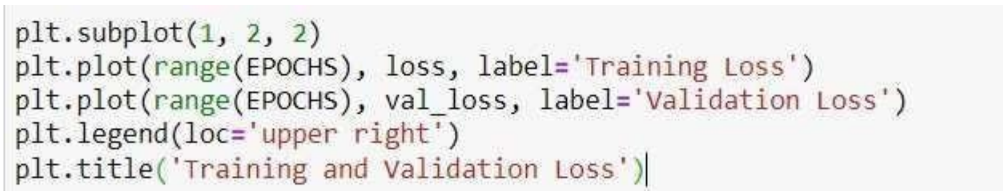


Figure 14: Training Accuracy and Validation Accuracy Plot

The code and the resulting graph illustrating training loss versus validation loss are shown in Figure 15.



```
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')
```

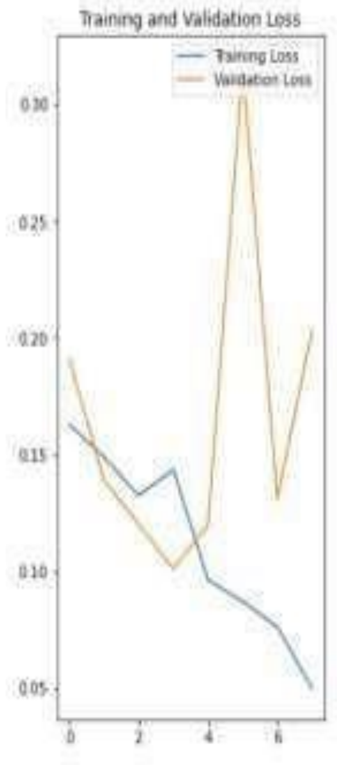


Figure 15: Training Loss and Validation Loss Plot

CHAPTER V
CONCLUSION

The current study utilizes a Convolutional Neural Network (CNN) architecture for facilitating classification in a web application. By inputting an image of a potato leaf, the model can determine whether the plant is healthy or affected by late blight or early blight. With minor adjustments, this project could greatly benefit farmers by allowing for easy identification of plant health without manual monitoring. This approach can lower production costs and enable early preventive measures. Additionally, the system could be adapted for disease identification in other plant species, leveraging computer vision for real-time analysis.

5.1 Future Scope

We will experiment with CNN architectures like VGG, ResNet, and InceptionNet to optimize accuracy. A mobile application can be developed to interact with FastAPI using cross-origin resource sharing (CORS) to provide results. The project can then be deployed on platforms such as Heroku, AWS, or other cloud services, making it publicly accessible.

5.2 Application

Deep learning algorithms have numerous applications, and in this project, they are utilized to identify whether a plant is healthy or infected based on an image. CNN networks can be trained for a various categorization tasks and when allowed to integrate with computer vision, these models yield impressive results. With further enhancements, this project could greatly benefit the agricultural sector. When paired with a mobile application, it can greatly enhance the monitoring of crop health. Ultimately, this study can contribute to both the agricultural industry and overall national growth, given the importance of crops to the economy.

REFERENCES

A. J. Rozaqi and A. Sunyoto. Identification of Disease in Potato Leaves Using Convolutional Neural Network (CNN) Algorithm, 2020 3rd International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2020, pp. 72-76. https://doi.org/10.1109/ICOIACT50329.2020.9332037.

A. KP and J. Anitha. Plant disease classification using deep learning, 2021 3rd International Conference on Signal Processing and Communication (ICPSC), Coimbatore, India, 2021, pp. 407-411. https://doi.org/10.1109/ICSPC51351.2021.9451696

Ahila Priyadharshini, R., Arivazhagan, S., Arun, M. *et al.* Maize leaf disease classification using deep convolutional neural networks. Neural Comput & Applic 31, 8887–8895 (2019). https://doi.org/10.1007/s00521-019-04228-3

Arshaghi, A., Ashourian, M. & Ghabeli, L. Potato diseases detection and classification using deep learning methods. Multimed Tools Appl 82, 5725–5742 (2023). https://doi.org/10.1007/s11042-022-13390-1

Kaggle. Plant Village. 2019. [Online]. Available: https://www.kaggle.com/datasets/arjuntejaswi/plant-village?resource=download

Lu. J, Tan. L, Jiang. H. Review on Convolutional Neural Network (CNN) Applied to Plant Leaf Disease Classification. Agriculture. 2021; 11(8): 707. https://doi.org/10.3390/agriculture11080707

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. 2011; 11(6):6015-6036. https://doi.org/10.3390/s110606015

Mishra P, Biancolillo A, Michel Roger J, Marini F, Rutledge D.N. New data preprocessing trends based on ensemble of multiple preprocessing techniques. Trends in Analytical Chemistry. 2020; 132(116045): ISSN 0165-9936. https://doi.org/10.1016/j.trac.2020.116045

Nirthika, R., Manivannan, S., Ramanan, A. et al. Pooling in convolutional neural networks for medical image analysis: a survey and an empirical study. Neural Comput & Applic 34, 5321–5347 (2022). https://doi.org/10.1007/s00521-022-06953-8

Oppenheim. D, Shani. G. Potato Disease Classification Using Convolution Neural Networks. Advances in Animal Biosciences. 2017; 8(2): 244-249. https://doi.org/10.1017/S2040470017001376

P. Tejaswini, P. Singh, M. Ramchandani, Y. K. Rathore, R. R. Janghel. Rice Leaf Disease Classification Using CNN. IOP Conf. Ser.: Earth Environ. Sci. 2022; 1032. https://doi.org/10.1088/1755-1315/1032/1/012017

Pineda Medina, D.; Miranda Cabrera, I.; de la Cruz, R.A.; Guerra Arzuaga, L.; Cuello Portal, S.; Bianchini, M. A Mobile App for Detecting Potato Crop Diseases. J. Imaging 2024, 10, 47. https://doi.org/10.3390/jimaging10020047

Puig-Arnavat M, Carles Bruno J. Artificial Neural Networks for Thermochemical Conversion of Biomass. Editor(s): Ashok Pandey, Thallada Bhaskar, Michael Stöcker, Rajeev K. Sukumaran; Recent Advances in Thermo-Chemical Conversion of Biomass; Elsevier. 2015; 133-156. https://doi.org/10.1016/B978-0-444-63289-0.00005-3

R. Sharma, V. Kukreja and V. Kadyan. Hispa Rice Disease Classification using Convolutional Neural Network, 2021 3rd International Conference on Signal Processing and Communication (ICPSC), Coimbatore, India, 2021, pp. 377-381. https://doi.org/10.1109/ICSPC51351.2021.9451800.

Rajaraman S, Zamzmi G, Antani SK. Novel loss functions for ensemble-based medical image classification. PLoS One. 2021; 16(12): e0261307. https://doi.org/10.1371/journal.pone.0261307

Russel, N.S., Selvaraj, A. Leaf species and disease classification using multiscale parallel deep CNN architecture. Neural Comput & Applic 34, 19217–19237 (2022). https://doi.org/10.1007/s00521-022-07521-w

S. Ghosal and K. Sarkar, "Rice Leaf Diseases Classification Using CNN With Transfer Learning," 2020 IEEE Calcutta Conference (CALCON), Kolkata, India, 2020, pp. 230-236. https://doi.org/10.1109/CALCON49167.2020.9106423.

Sharma R, Singh A, Kavita, Jhanjhi NZ, Masud M, Jaha ES, *et al.* Plant disease diagnosis and image classification using deep learning. Comput Mater Contin. 2022; 71(2):2125-2140 https://doi.org/10.32604/cmc.2022.020017

Wahabzada, M., Mahlein, AK., Bauckhage, C. *et al.* Plant Phenotyping using Probabilistic Topic Models: Uncovering the Hyperspectral Language of Plants. Sci Rep 6, 22482 (2016). https://doi.org/10.1038/srep22482

Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9