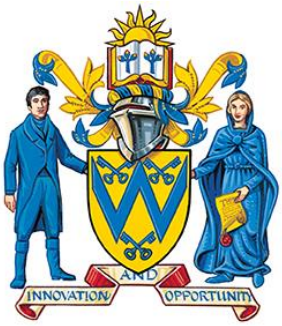


FINAL PROJECT

Deep Machine Learning

Potato Disease Classification

***Submission Date*:** 20 Aug, 2023

Project Start Date and End Date: 26 June to 21 Aug,2023

Submitted by: Mehmet Omer DEMIR

Faculty of Science and Engineering

University of Wolverhampton
7CS082/UZ1:Deep Machine Learning

Instructor: Ahmed Khubaib

Email: M.O.Demir@wlv.ac.uk

Word Count = 3240

Table of Contents

Potato Disease Classification

1.Introduction

In the agricultural sector, potato growers struggle with annual economic losses, especially diseases such as Late Blight and Early Blight, which are the main causes of these losses. Early Blight is triggered by the fungus Alternaria solani, while Late Blight is caused by the microorganism Phytophthora infestans. Our aim in this project is to try to classify diseases in potato plants as "Healthy", " Late Blight " and " Early Blight". We aim to significantly reduce the economic losses of farmers by providing the opportunity to detect diseases early and take effective measures by using deep learning techniques.

2.Problem Statement

Potato farmers face difficult financial situations due to serious economic losses every year. These losses are the result of various diseases affecting potato plants. Especially Late Blight and Early Blight diseases are the most common diseases.

Early blight **(Photo1)** is a potato disease caused by a fungus caled Alternaria solani. It is found wherever potatoes are grown. The disease usually affects leaves and stems, but if not controlled under suitable weather conditions, it can cause significant defoliation and an increased chance of tuber infection.

Late blight**(Photo2)** is the most important disease of potato caused by a microorganism called Phytophthora infestans and can cause harvest losses in a short time if appropriate control measures are not taken.

While early blight is a fungal disease, late blight is caused by certain microorganisms, and if farmers detect the disease at an early stageand apply appropriate treatment, they can both prevent waste and prevent economic losses. Treatment methods for early blight and late blight are slightly different, so it is important to accurately determine the type of disease on the plant.
Our aim is to classify the type of diseases in the potato plant. In the current dataset, there are a total of 2152 leaf images. These images have been categorized into three distinct classes based on their characteristics. The classes are as follows:

- Healthy
- Early Blight
- Late Blight

In this project, we aim to correctly classify the type of diseases on the potato plant using deep learning techniques. In this way, we aim to minimize economic losses by providing farmers with the opportunity to intervene earlier and apply the right treatment.

Photo 1:**Potato Early blight**

Photo 2:**Potato Late blight**

3.Dataset

The dataset used in this study was taken from the "Plant Village" dataset on the Kaggle platform which contains 2152 leaf images divided into three disease classes. The dataset contains a comprehensive data collection to detect and classify plant diseases. Although the dataset includes diseases of different plant species, except for the alkaline data of the potato plant, the data were deleted for our project.

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

Using this code we import TensorFlow library, import models and layers modules from TensorFlow's keras module. It also imports plt module from matplotlib.pyplot library and imports HTML class from IPython.display module to display interactive content in Jupyter Notebook.

```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

BATCH_SIZE = 32: Specifies the size of the batch of images to be fed into the model each time
IMAGE_SIZE = 256: Indicates the target size to resize images. By bringing the dimensions of all Images to this value, the model is enabled to operate more efficiently.
CHANNELS=3: Indicates the number of color channels of the images. (red, green, blue).

EPOCHS=50: Specifies how many times the model will be trained on the entire dataset. An epoch represents an iterative process in which the model processes and updates data. More epochs allow the model to learn more.

Import data into tensorflow dataset

```
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.
```

Next, we specify the path to the "PlantVillage" directory in the dataset variable. This directory represents the home directory where the dataset is located. We assign a seed value to the variable **seed** for random **shuffling**. This is used to repeat the same random shuffle results.In the **IMAGE_SIZE** variable we set the target size to resize the images. In the **BATCH_SIZE** variable we specify how many images will be processed in each batch.We create the dataset using the **tf.keras.preprocesing.image_dataset_from_directory** function. This function loads the dataset according to the specified directory structure and uses the important parameters we specified (mixing, batch size, etc.).

Three classes are listed below.

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

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

Three classes are listed below.Each element in the dataset is a tuple, as can be seen in the example below. The first element is a collection of 32 image elements. A collection of 32 class label elements makes up the second element.

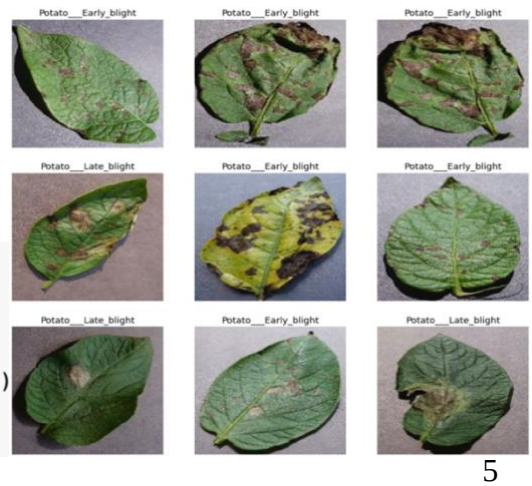
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

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

Visualize some of the images from our dataset

In this way, we visually display the images from the snippet dataset on the screen in a 3x3 grid and print the corresponding label on top of each image.

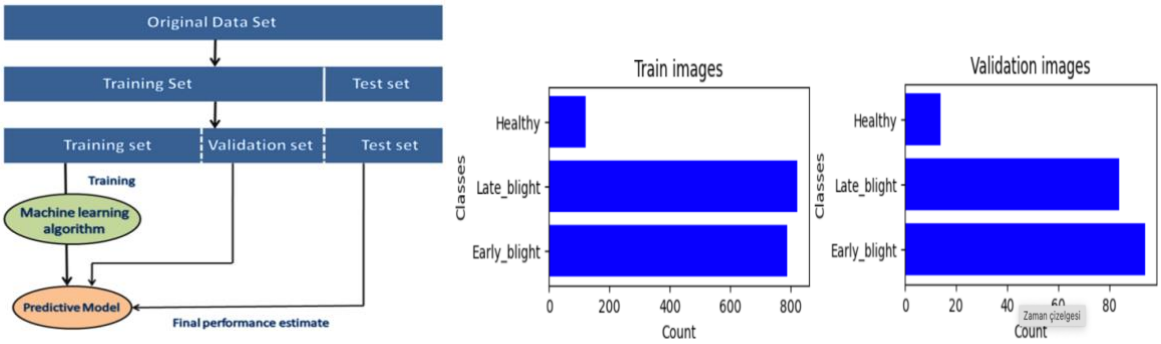
```
plt.figure(figsize=(12, 12))
for image_batch,labels_batch in dataset.take(1):
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Function to Split Dataset

The dataset should be divided into these 3 subsets:

1. Validation: Dataset to be tested against while training.
2. Training: Dataset to be used during training
3. Test: Dataset against which the model's performance will be evaluated.



In this project, we divided the dataset into three different sections to train, validate and test the model. We used 80% of the total dataset as the training set. To monitor the performanciing of the model during training and to check for overfiting, we allocated 10% of it as a validation set. Finally, we allocated 10% of the dataset as a test set to evaluate the real-world performance of the model.

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    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)
```

Cache, Shuffle, and Prefetch the Dataset

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

.cache(): Speeds up recursive access by caching data in memory.
.shuffle(1000): Helps the model generalize better by randomly shuffling the data.
.prefetch(buffer_size=tf.data.AUTOTUNE): Increases processing speed by caching data during loading and automatically selects the optimal cache size.
These processes aim to integrate the datasets into the training of the model more quickly and effectively.

Creating a Layer for Resizing and Normalization

If the input image is not (256x256), a layer using Keras has been defined to resize the image to (256x256) so that training can be carried out. Additionally, the RGB values will be scaled.

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

Data Augmentation

Data augmentation becomes crucial in scenarios with limited data availability. Its purpose is to increase model accuracy by artificially expanding the dataset. This includes creating variations of existing data with techniques such as magnifying, rotating, flipping, and cropping. These augmented examples contribute to a more robust model by providing a variety of training examples, even in situations of data scarcity.

```
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. Model Architecture

What are Convolutional Neural Networks?

A unique variety of neural networks called convolutional neural networks have proven to be very successful at classifying images. Pooling these networks can be used in computer vision applications, and they are very helpful in face recognition and object detection because they provide extra layers like convolution.

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.

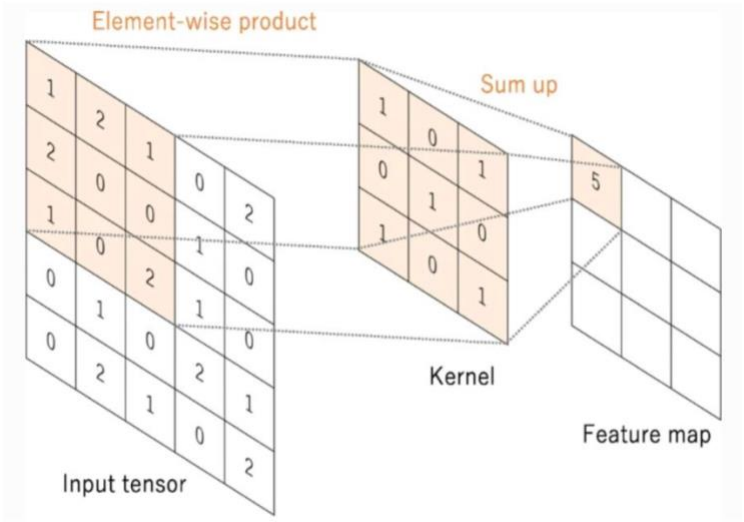
Overview of different operations in CNN:

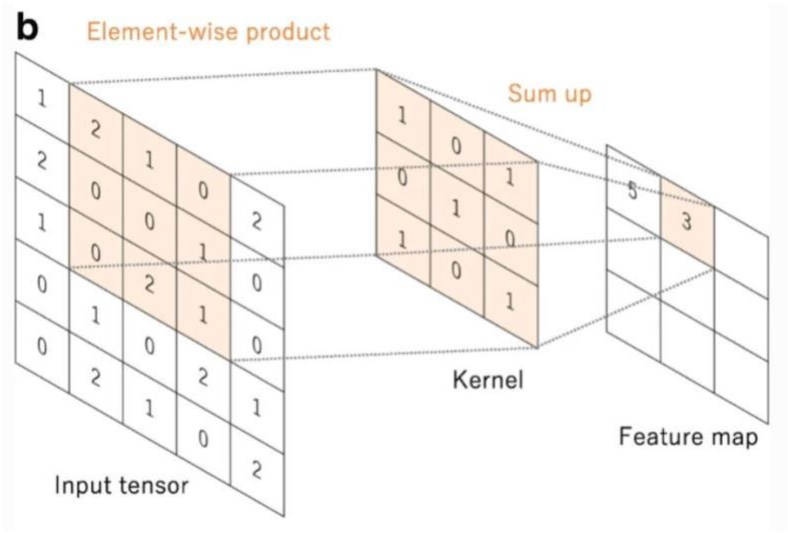
As shown in figure above, it is a CNN architecture which contains some sequential operations. This simple CNN consists of convolutional layer in which a kernel or filter is applied which is used to detect the vertical or horizontal edges and then it is followed by Relu activation function. Next layer 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 propagationand according to the loss value weights, filters and bias are updated inback propagation which may be through gradient descent or Ada Delta or adam

Convolution

In this operation, a tiny tensor known as a kernel or filter is applied to the input image. Its primary responsibility is to extract the crucial details from the image. Following this process, a feature map is created by multiplying each value in the image and kernel element by element. The vertical and horizontal edges on this feature map will be enhanced in the following step.

Figure showing convolution operation





Above figure is showing a convolution operation on a 5x5image. Without padding, the filter is 3x3, and the stride is 1. The figure illustrates how it will shrink in size if there is no padding. Padding is used to maintain the same image size, which will spread out the image's size by adding extra pixels.

Image size calculation formula:

(N+ 2P –F)/S + 1

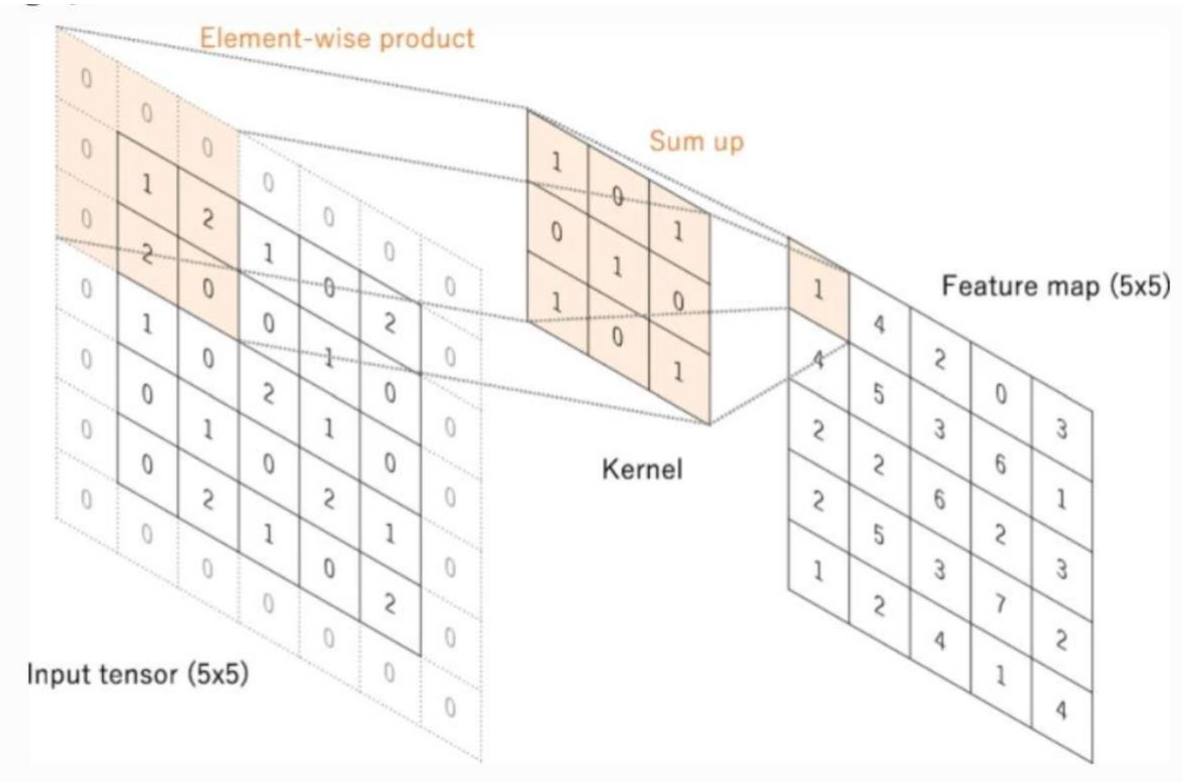
Where N = image size

S = stride

P = padding

F = kernel size

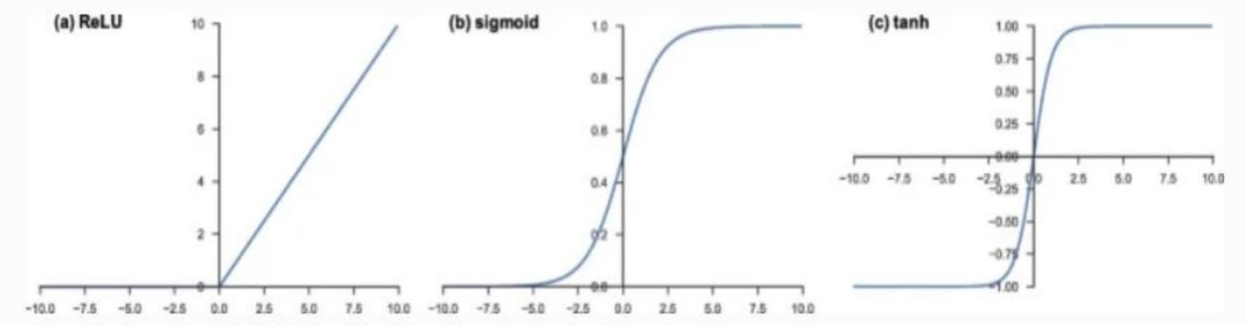
Figure with padding:



Activation Function:

An activation function is applied to the feature maps that result from the application of convolution. These non-linear activation functions introduce non-linearity into the network. In hidden layers, activation functions like sigmoid and tangent are avoided because they may cause a problem with vanishing gradients. Relu function is a popular activation function.

F(x) = max(0,x)

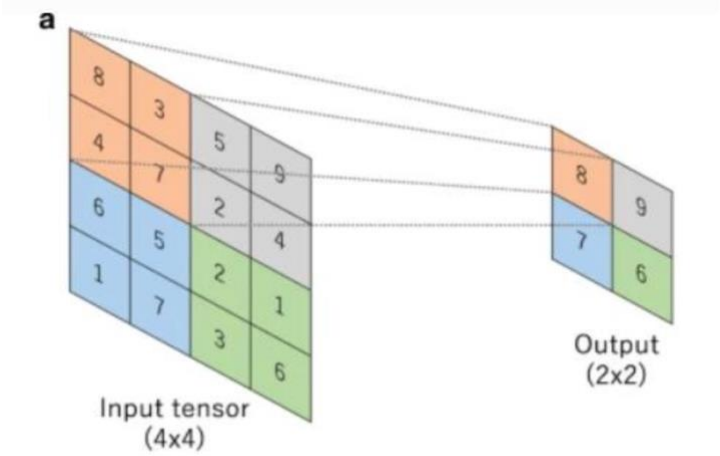


Pooling Layer

The most crucial features are extracted using this layer. Reduce the learnable parameters is its main task. The depth stays the same while the height and width of the feature map are reduced.

Max Pooling

This layer applies a kernel to the image in order to maximise the value of each patch. The most common filter size used over an image is 2x2 with a stride of 2. The figure below depicts the maximum pooling operation.

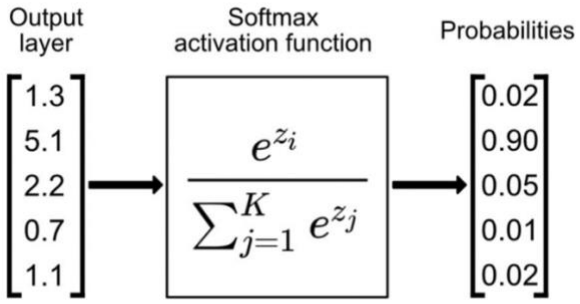


Fully Connected Dense Layer

These feature maps are flattened and then passed to a fully connected dense layer after being received from the max pooling layer. A dense layer receives feature maps that have first been transformed into a one-dimensional array. There may be a lot of thick layers between them.The number of output classes is primarily contained in the final output layer. Each dense hidden layer includes an activation function. Relu is the activation function most frequently used.

Activation Function at Last Layer

As their primary function is to introduce non-linearity, the activation function used at the final output layer differs from that used in the hidden dense layers. The final layer's activation function is used to categorise, and it provides the probabilities for each class. The sigmoid function is used when there are two classes. However, there are more than two categories in this project. Therefore, in this case, the softmax function is used, which normalises so that each probability value falls within the range of 0 to 1, and the final sum is 1.The figure below displays the sigmoid function's output.



All of the weights, filters, and bias are calculated in forward propagation. These are subsequently verified by a cost function that will return an error. The difference between the actual label and the predicted label is the error. Weights, filters, and bias updating take place in back propagation. An algorithm called anoptimizer is used to perform this update. The goal of this task is to determine the weights, filters, and bias values that will minimise loss. There are numerous optimizers on the market. Adam is one that I am utilising.

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.

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.

In this project, we used a Convolutional Neural Network (CNN)(**Photo 3)** architecture coupled with a Softmax activation at the output layer. Additionally, we have integrated the first layers for resizing, normalizing and data augmentation. Below is a description of the model type and a high-level description of its architecture:

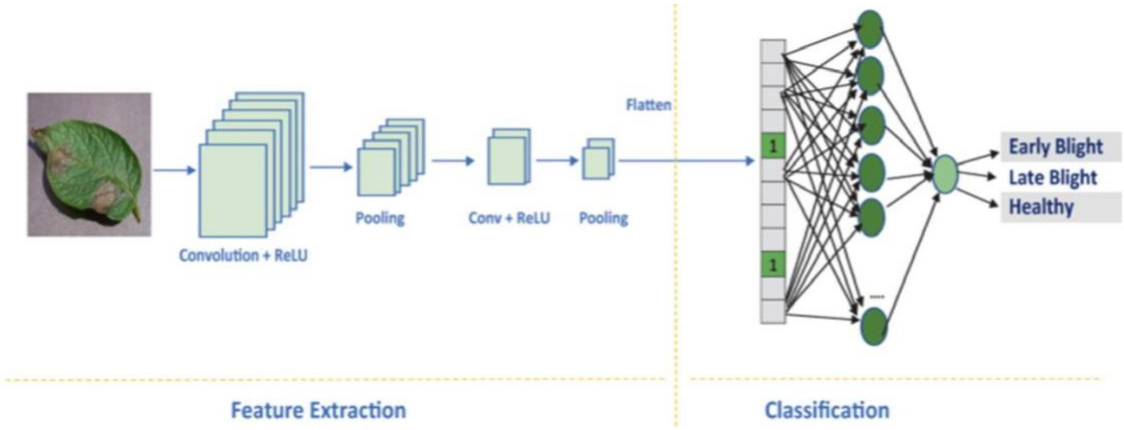


Photo 3:CNN Application process for leaf classification.

Architecture Selection:

The chosen architecture, a CNN combined with Softmax activation, is well suited for image classification tasks such as identifying plant diseases. Convolutional Neural Networks are designed to automaticaly learn and represent hierarchical patterns from image data. Softmax activation in the output layr allows us to obtain probability distributions over different classes.

By including the first layers for resizing, normalization and data augentation, we ensure that the input data is properly preprocesed and augmented, which can improve the models ability to learn and generalize from limited data.

The architecture's combination of convolutonal and fully connected layers allows the model to capture both low_level features (edges,textures) and high_level models (shapes, structures), making it suitable for accurately classifying different types of plant diseases based on visuality. features.

```
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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

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

 max_pooling2d (MaxPooling2  (32, 127, 127, 32)        0
 D)

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

 max_pooling2d_1 (MaxPoolin  (32, 62, 62, 64)          0
 g2D)

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

 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 64)          0
 g2D)

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

 max_pooling2d_3 (MaxPoolin  (32, 14, 14, 64)          0
 g2D)

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

 max_pooling2d_4 (MaxPoolin  (32, 6, 6, 64)            0
 g2D)

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

 max_pooling2d_5 (MaxPoolin  (32, 2, 2, 64)            0
 g2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In the figure above, the total trainable parameters are depicted.Different loss functions come in various forms. This project makes use of a sparse categorical cross-entropy loss function. Using the Adam optimizer, which updates weights and bias, the loss will be reduced.Back propagation loss has been validated using accuracy metrics.

Compiling the Model

Compile Model:

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

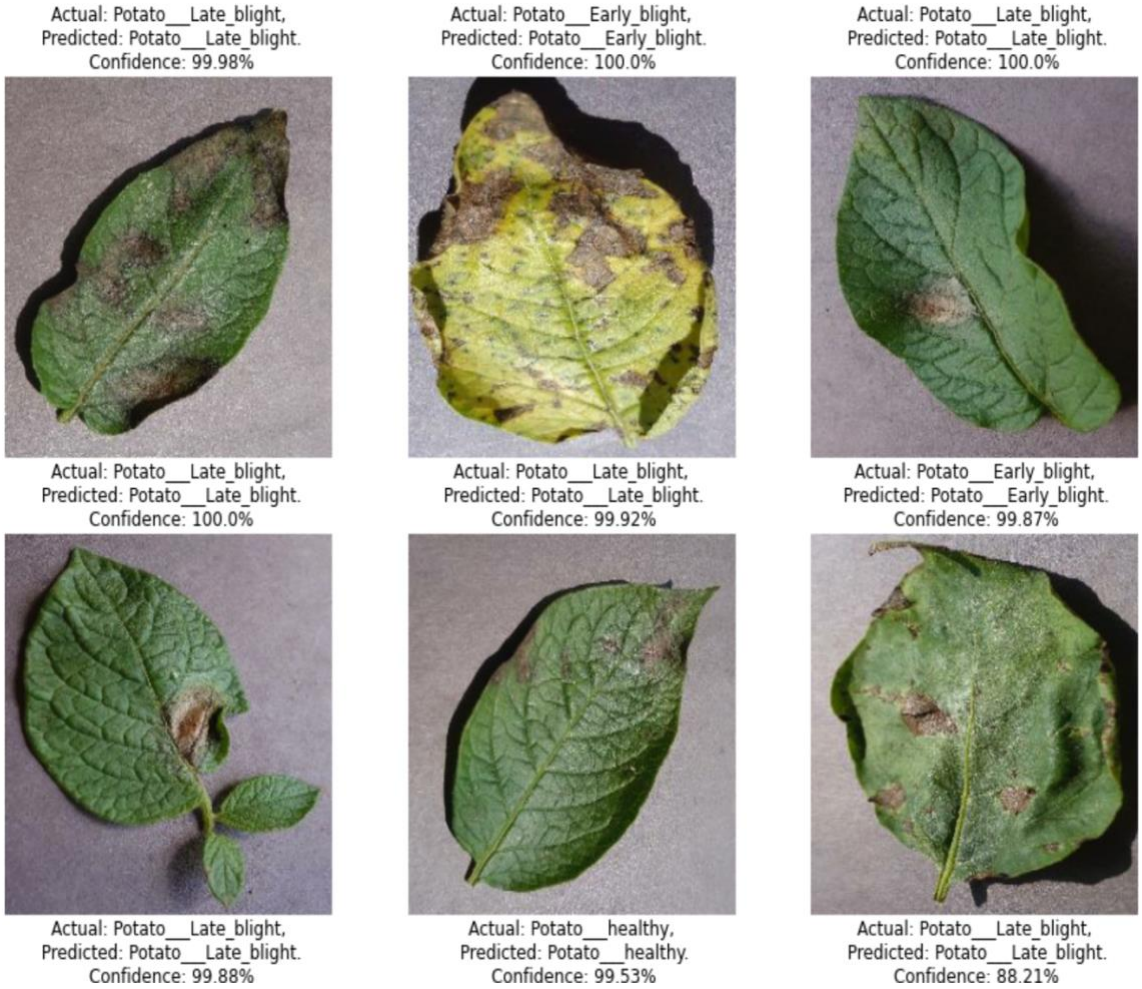
A function has been written for prediction that will accept a model and an input image as input parameters. The model will return an array with the probabilities associated with each class, from which we will choose the probabilities with the highest values. The code is given below:The predictions we obtained after using the predict function are displayed 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)

    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
```

The predictions we obtained after using the predict function are displayed below:



4.Result

Our model was built with the Convolutional Neural Network (CNN) architecture. The image dimensions were resized to a predetermined size, normalized, and the data set was expanded for better results using data augmentation techniques. In this way, we found that the model could better adapt to images from different angles, sizes and positions.

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

Accuracy – It is the capacity to correctly differentiate between real and fake note test cases.
Accuracy = (tp + tn) /(tp + tn + fp + fn)

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

Specificity - It entails having a precise identification of the original note cases.
Specifcity= tn / (tn+ fp)

Precision- The test's accuracy is determined by the classifier's ability to determine whether the notes it has classified as real are actually 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 determinedas counterfeit notes.

False positive = It is the count that contains misidentified as original notes.

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

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

For validation, accuracy matrix hasbeen 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=50,
)
```

click to unscroll output; double click to hide

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

```
54/54 [==============================] - 26s 482ms/step - loss:
0.0360 - accuracy: 0.9913 - val_loss: 0.0045 - val_accuracy: 1.
0000
Epoch 47/50
54/54 [==============================] - 27s 491ms/step - loss:
0.0204 - accuracy: 0.9925 - val_loss: 0.1574 - val_accuracy: 0.
9635
Epoch 48/50
54/54 [==============================] - 26s 488ms/step - loss:
0.0352 - accuracy: 0.9890 - val_loss: 0.0189 - val_accuracy: 0.
9948
Epoch 49/50
54/54 [==============================] - 27s 492ms/step - loss:
0.0279 - accuracy: 0.9907 - val_loss: 0.0079 - val_accuracy: 1.
0000
Epoch 50/50
54/54 [==============================] - 27s 490ms/step - loss:
0.1195 - accuracy: 0.9595 - val_loss: 0.1255 - val_accuracy: 0.
9479
```

When evaluated on a test dataset, model gave 97% accuracy. Below is the code and result.

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

```
8/8 [==============================] - 1s 110ms/step - loss: 0.0792 - accuracy: 0.9766
```

For plotting training and validation accuracy, value at each stepsare 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']
```

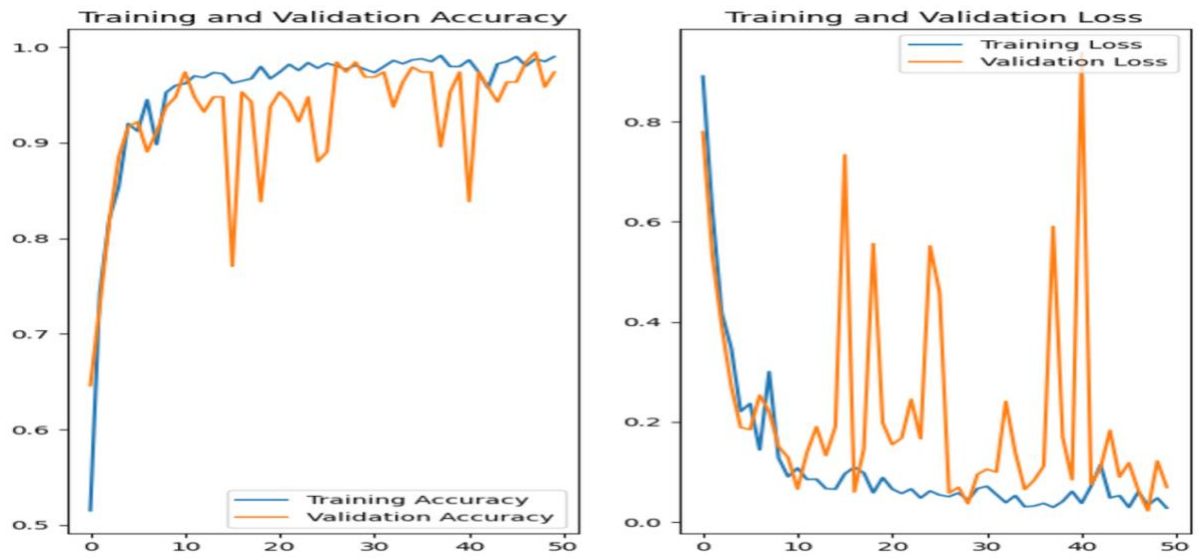
Below is the code and result of a graph between training accuracy vs validation accuracy.

```
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')

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')
plt.show()
```

In order to create our model, we used CNN to solve the issue. To train our model, we used images of more than 2,000 potato leaves. The accuracy of our model, which is around 98%, is shown in the following graph. Prediction based on representative models is shown in the image below.

Our model was built with the Convolutional Neural Network (CNN) architecture. The image dimensions were resized to a predetermined size, normalized, and the data set was expanded for better results using data augmentation techniques. In this way, we found that the model could better adapt to images from different angles, sizes and positions.



As a result, the developed deep learning model has achieved very successful results in identifying and classifying diseases in potato plants. This project provides an example of the use of artificial intelligence technologies in agriculture and can help potato growers make more effective decisions and generate more revenue in disease diagnosis and treatment.

5.User Interface

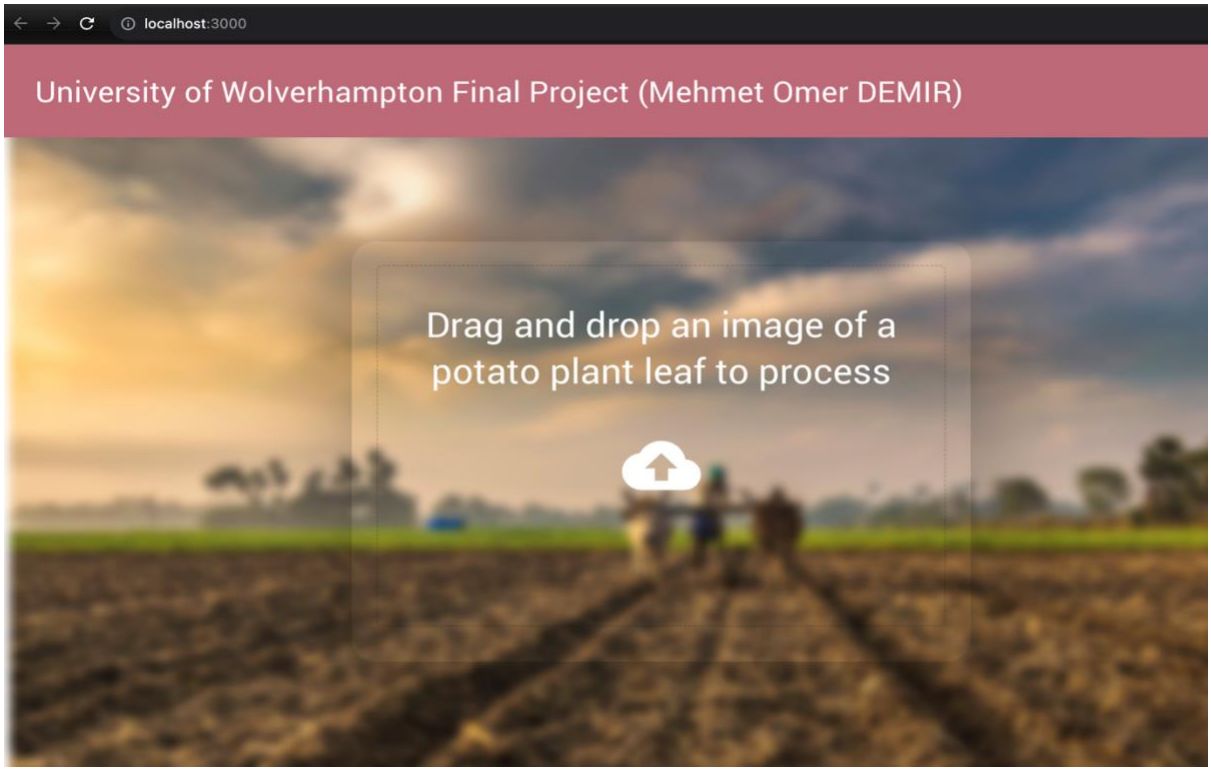


Photo 4: View of the application as a web page

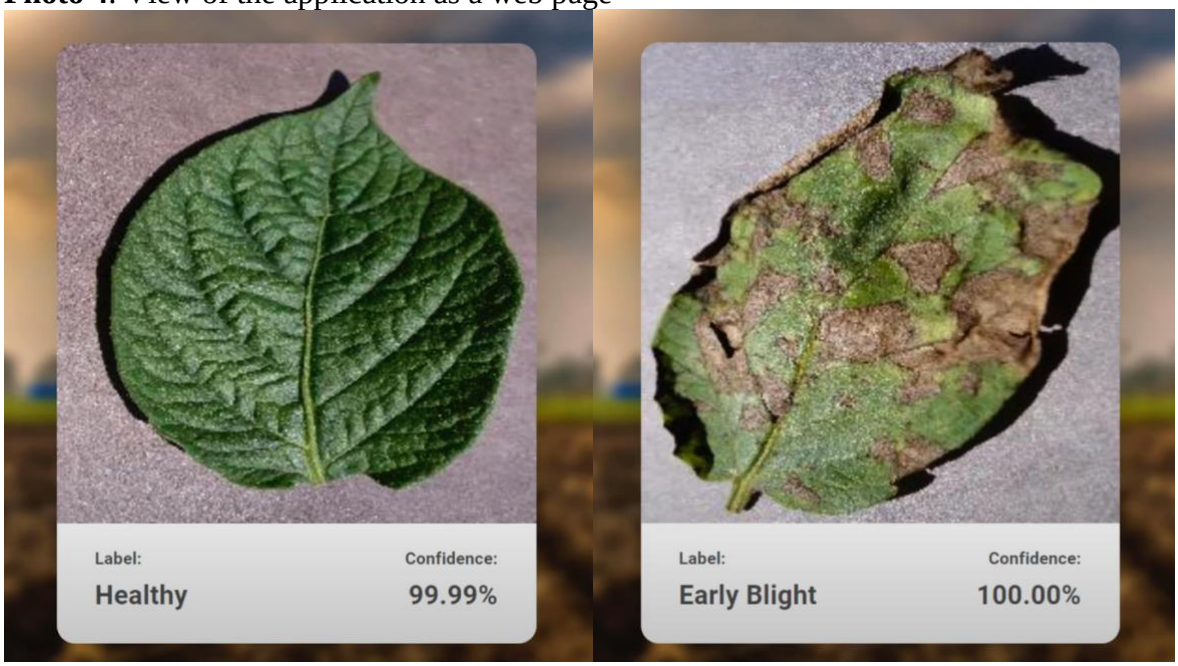


Photo 4: Example result demonstration in practice

6.Conclusion and Future Prospect

In this project, we addressed the critical challenges of potato plant disease classification using deep learning techniques. The agricultural sector often graples with economic losses from diseases such as Early Blight and Late Blight. We aimed to significantly reduce these losses by developing an accurate disease classification model.

Our project focused on identifying diseases in potato plants and classifying them into three different categories: "Healthy", " Late Blight " and " Early Blight." With the use of Convolutional Neural Networks (CNNs), we haressed the power of deep learning to effectively distinguish these types of diseases and got a good 98% result.

The project's dataset was extracted from the "Plant Village" dataset on Kaggle, which contains 2152 leaf images divided into three disease classes. We strategicaly preprocesed the data, which involved resizing and normalizing images, as well as applying data augmentation techniques. These steps increased the accuracy and capacity of our model to process a variety of images with various features.

The selected CNN architecture combined with Softmax activation in the output layer proved suitable for the image classification task. We improved the generalzation capability of the model by optimizing the preprocessing of the input data by including the first layers for resizing, normalization and data augmentation.

We compiled the model with a Adam optimizer and used Sparse Categorical Cross-Entropy as the loss function, with accuracy as our primary criterion. The training process of the model was run with the training data and validated against the validation dataset. The model exhibited suficient accuracy, achieving approximately 98% accuracy on the test dataset.
As a result, our deep learning model has successfully overcome the challenge of potato disease clasification.

Using advanced machine learning techniques, we paved the way for a more efficient and accurate approach to disase diagnosis in agriculture. The results of this project underscore the potential of AI technologies to revolutionize the way we tackle challenges in the farming industry, ultimately leading to greater productivity and better decision-making for growers.

In summary, the success of our project in disease classification has the potential to have a significant impact on potato farming. Looking ahead, a promising path for further progress is the creation of a user-friendly web-based or Android application. This practice can promote early intervention, cost savings and increased food safety by providing farmers with timely disease diagnosis. By leveraging the power of artificial intelligence and deep learning, we aim to improve agricultural practices and contribute to a more resilient and sustainable future for potato growers worldwide.

References

[1] Campos, T., Fuentes, A., Plaza, A., Bagnato, V. S., & Teixeira, L. S. G. (2020). Identification of soybean diseases using convolutional neural networks. Computers and Electronics in Agriculture, 179, 105863.

[2] Dacal-Nieto, A., Vazquez-Fernandez, E., Formella, A., Martin, F., Torres-Guijarro, S., & González-Jorge, H. (2009). A genetic algorithm approach for feature selection in potatoes classification by computer vision. IEEE, Spain, 1955-1960.

[3] Golhani, K., Balasundram, S. K., Vadamalai, G., & Pradhan, B. (2018). A review of neural networks in plant disease detection using hyperspectral data. Information Processing in Agriculture, 5(3), 354–371.

[4] Hocaoğlu, İ., & Yanar, O. (2018). A deep learning model for early diagnosis of crop diseases. Computers in Biology and Medicine, 102, 70-77.

[5] Islam, M., Dinh, A., Wahid, K., & Bhowmik, P. (2017). Detection of potato diseases using image segmentation and multiclass support vector machines. In 2017 IEEE 30th Canadian conference on electrical and computer engineering (CCECE), pp. 1-4. IEEE.

[6] M. S. Prasad Babu and B. Srinivasa Rao. (2007). Leaves Recognition Using Back Propagation Neural Network Advice For Pest and Disease Control On Crops, IndiaKisan.

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

[8] Mattihalli, C., Gedefaye, E., Endalamaw, F., & Necho, A. (2018). Plant leaf diseases detection and auto-medicine. Internet of Things, 1-2, 67–73.

[9] Melike Sardogan, Adem Tuncer, Yunus Ozen. (2018). Plant Leaf Disease Detection and Classification based on CNN with LVQ Algorithm. In 2018 3rd International Conference in Computer Science and Technology, pp. 382-385. IEEE.

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

[11] Mukherjee, A., Maulik, U., & Bandyopadhyay, S. (2007). Plant leaf recognition: a review. Pattern Recognition, 39(4), 609- 625.

[12] Panagiotis Tzionas, Stelios E. Papadakis and Dimitris Manolakis. (2005). Plant leaves classification based on morphological features and fuzzy surface selection technique. In 5th International Conference ON Technology and Automation ICTA05, Thessaloniki, Greece, pp. 365-370.

[13] Pantazi, X. E., Moshou, D., & Tamouridou, A. A. (2019). Automated leaf disease detection in different crop species through image features analysis and One Class Classifiers. Computers and Electronics in Agriculture, 156, 96–104.

[14] Ungsumalee Suttapakti, Aekapop Bunpeng. (2019). Potato Leaf Disease Classification Based on Distinct Color and Texture Feature Extraction. In 2019 19th International Symposium on Communications and Information Technologies (ISCIT), pp. 82-85. IEEE.

[15] Zhang, S., Wu, G., Zhao, Y., & Hu, J. (2019). Plant disease identification using explainable 3D CNN and Wasserstein GAN. Computers in Biology and Medicine, 109, 51-63.