Hani Khaled Kamal

22102516

Detection of Plant Leaf Diseases Using ML

Computing and Digital Technology School

Birmingham City University

Birmingham, United Kingdom

August 2022

Table of Contents

Table of Figures

ABSTRACT

Plant diseases significantly reduce agricultural productivity, economics, quality, and quantity. To avoid such diseases, plants must be watched from the beginning of their life cycle. The conventional form of monitoring is naked eye examination, which takes time, money, and a high level of ability. The illness detection system must be automated to speed up this procedure. Image processing technologies must be used to develop the illness detection system. Many researchers have created systems that make use of various image processing approaches. This study investigates the possibilities of plant leaf disease detecting systems that could aid in agricultural development.

It consists of several stages, including picture capture, image segmentation, feature extraction, and classification.

INTRODUCTION

Agriculture is critical to a country's economic prosperity since it offers the primary source of income and employment for most of the people. It will contribute around 0.5 percent to the UK GDP in 2020. Agriculture produces half of the food we consume, employs almost 500,000 people, and is an important element of the food and drink sector, which contributed £127 billion to the economy in 2019. Farmers and land managers maintained 71% of the UK's land in 2020, and it is through them that we can protect our natural environment and ensure the greatest standards of animal and plant health. This Agriculture in the UK evidence pack compiles existing agricultural statistics to summarise the present situation of the agricultural business.

However, due to varying temperatures and local circumstances, these plants are susceptible to a variety of illnesses. It is also critical to detect and diagnose these illnesses in plants at an early stage since some of these diseases are contagious and can spread over the entire field. Because of the intricacies of cultivated plants and associated Phyto-pathological issues, even expert agronomists and plant pathologists sometimes fail to detect illnesses, resulting in incorrect conclusions and treatment techniques. As a result, this study project was created to address this issue.

BACKGROUND & CHALLENGES

Many researchers conducted study on numerous plants and their illnesses, and they also provided some strategies for identifying such diseases. But in this project, I'm focusing on four plants (Potatoes, Tomatoes, Strawberry, Apples) because as (Q2 2022) and official UK government information source those plats are among the top 10 most planted plants in the United Kingdom. The input date collected from many sources is required for the automation of illness identification.

In this review, I'm identifying and addressing major topics, difficulties on illness, and strategies based on all distinct research publications and counter the Quality image of plant leaves
difficulty that faces some of the researchers.

PROBLEM STATEMENT

As many researchers has discussed in their papers and as Vempaty Prashanthi in specific mentions that the main problem is that It is incredibly difficult to identify plant disease in the agricultural field. When identification is inaccurate, the product's assembly and the market's economic value suffer severe losses. If proper care is not given during this

chamber, it creates major consequences on plants and is influenced by varied product quality, quantity, or productivity.

LITERATURE REVIEW

Fujita et al. employed CNN to evaluate their own sheet data set in the field. Under diverse lighting and photography settings, they achieved an average accuracy of 82.3 percent.

Using the leaf image, M.R developed a simple infection discovery framework for cotton. A photograph of the sickly leaf is taken. Then, utilising image processing and Artificial Neural Network differentiation is adjusted over solid and bad examples. Furthermore, the ANN order has an accuracy of 80%.

M.Bhange and colleagues (2015) By contributing fruit images to the system, a web-based tool for identifying fruit illnesses has been established. Colour, morphology, and CCV (colour coherence vector) parameters were used to extract features. The k means technique was used for clustering. SVM is used to determine whether a person is infected or not. This study has an 82 percent accuracy rate in identifying pomegranate illness.

J.D. Pujari et al. (2015) used a variety of crop types to identify fungal infections on plant leaves, including fruit crops, vegetable crops, cereal crops, and commercial crops.

Several strategies have been employed. For segmentation, the chan-vase technique was used, and local image patterns for extracting texture features were segmented using k-means clustering and the canny edge detector. Color, shape, texture, texture colour, and random transform characteristics were extracted. SVM and closest neighbour classifiers were applied, yielding a total accuracy of 83.72%.

OBJECTIVE

Using image processing and machine learning algorithms such as the k-nearest neighbors (KNN) and convolutional neural network (CNN, or ConvNet), this project proposed a major diagnostic strategy for tomato and potato plant leaf diseases, as well as a graphical layout of preventive actions. To examine the symptoms of diseased leaves, an image processing technique is done on Kaggle dataset (plant_village) of potato, tomato, apple, and strawberry leaves using the operations of data pre-processing, augmentation, and data extraction.

AIM

After reviewing all some of the papers in this area, the aim is to achieve more than 87 percent accuracy rate, and this is the baseline.

MATERIALS AND METHODS

There are numerous approaches for determining the key aspects of plants leaves imaging and classifying them to distinguish between a healthy or having a disease. As a result, the first step is to extract the key characteristics from pictures, a process called feature extraction. After feature extraction, the high dimensionality of the feature vector contributes to computational complexity. As a result, we must choose only the necessary traits while eliminating the rest. This is referred to as feature selection. This is followed by

categorization to differentiate between healthy and infected leaves. Figure 1 depicts the procedure's general framework.

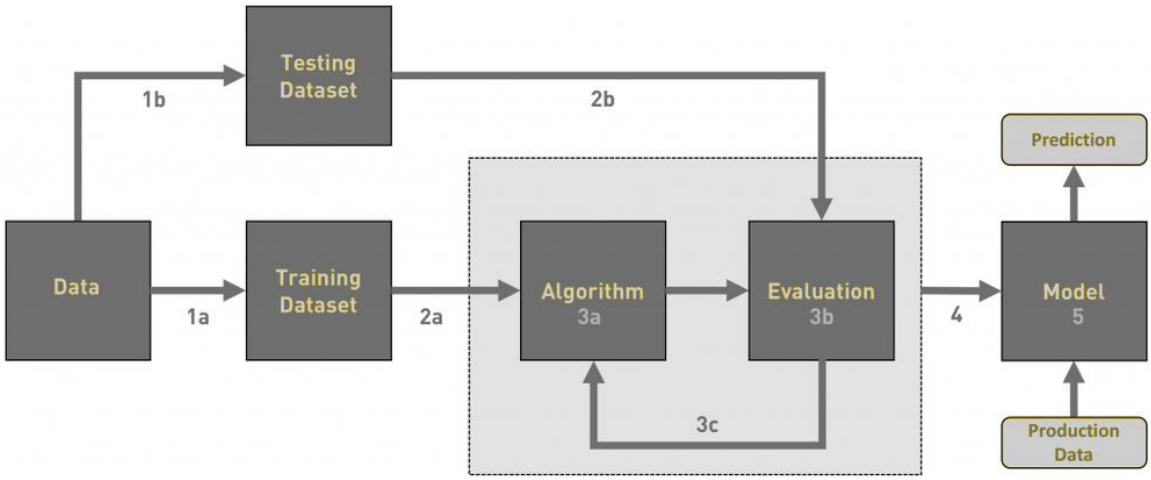


Figure 1: Workflow of the ML project

Data Acquisition

The datasets which are using to build my models of AD classifications taken from **Kaggle datasets** and a three sets model (test, validation, testing) will be used, one of the keys here that I did some manipulation with data to make it more efficient and the reason is to mimic the real scenario of taking pictures using a camera.

The number of images and the different aspects improve the model's performance, here are the general details:

Train/Validation Planet Village Dataset (256 x 256)

- The Data is collected Kaggle database title: An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing.
- The Dataset is consisting of **raw JPG images**.
- The Dataset has 4 plant categories
- The Dataset has 19 classes.
- The Dataset is consisting of total 44,270 images divided to:
    - 35416 training images
    - 8854 testing images

Data importing

For the easy use of the data from different users an API and token from Kaggle is used to import data in all ML models by using a username and key as shown in the below Figure 2

```
# {"username":"hanikamal","key":"bb3040d21aad356e7ab7c1bc5548b26c"}
import opendatasets as od
import pandas

od.download(
    "https://www.kaggle.com/datasets/hanikamal/uni-plant-village")
```

Figure 2: Data importing

DATA EXPLORATION AND VISUALIZATION

Becoming one with the data is a critical first step in any machine learning endeavor. This normally entails a lot of visualizing and folder scanning to grasp the data. With that said, let's take a look at the data. Here is a sample from the dataset.

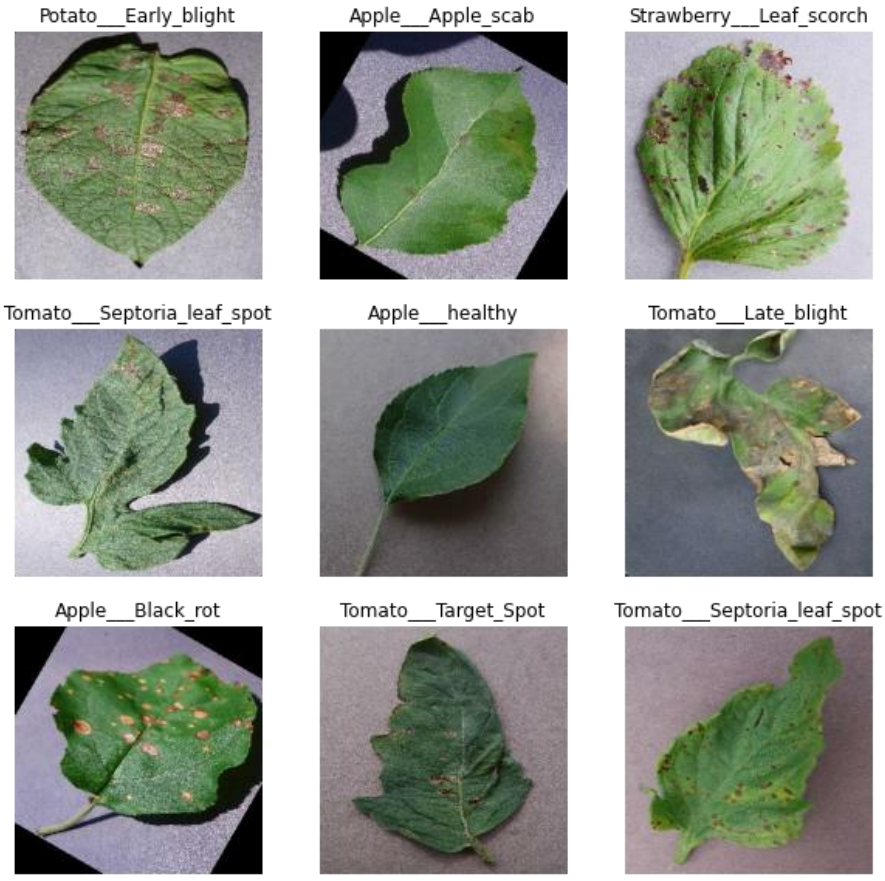


Figure 3: Samples from Data

As mentions before the data set has 4 kinds of plants can be checked with the code in figure .

And for the classes, Figure 4 shows the number of images per class in the training set, and we can see that all the classes have a similar number of images and that's a good aspect and it will help us getting a good accuracy prediction.

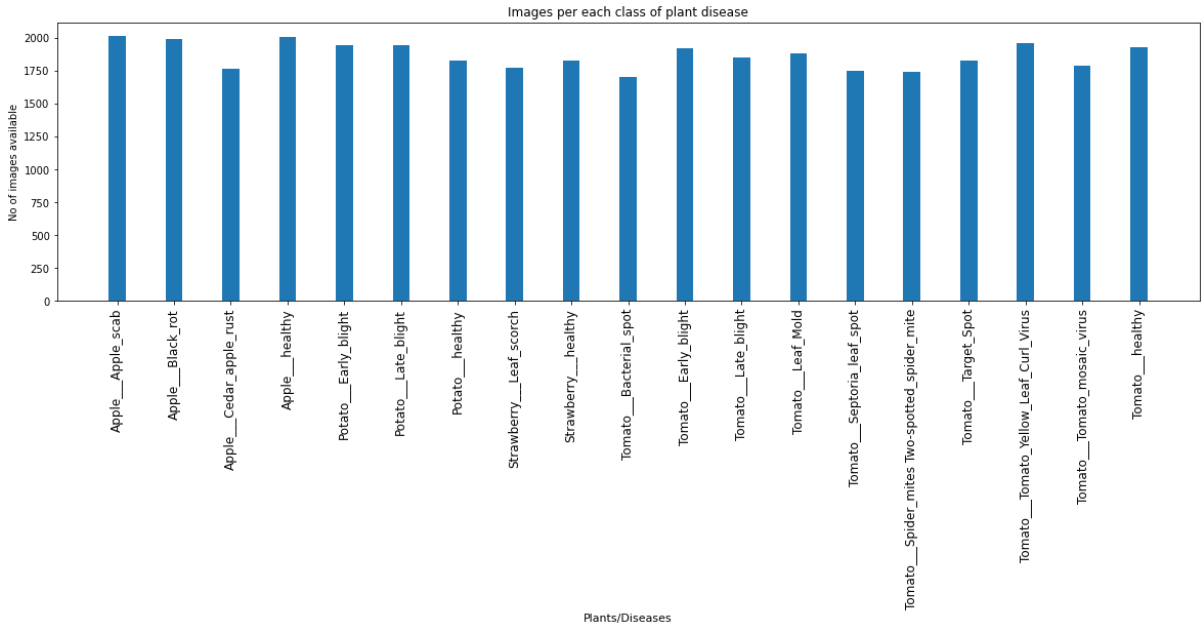


Figure 4: Classes of the diseases

Feature Extraction

is a step in the dimensionality reduction process that divides and reduces an initial collection of raw data to more manageable categories. As a result, processing will be simpler. Feature extraction often involves pattern matching as well as comparing text strings with the names of recognized things. If the underlying knowledge is a basic gazetteer that categorizes these strings, then extraction yields only a categorized collection of extracted strings. However, if we simply utilize one feature vector, this technique is less likely to produce successful results because these species share several features, such as color, and so on. As a result, we must quantify the image by combining numerous feature descriptors to properly define it.

Global Feature Descriptors

Global descriptors are used for picture retrieval, object detection, and classification. convey the visual content of the entire picture as one vector, whereas local features remove the IPs of the image and describe them as a series of vectors. The following are some of the most commonly used global feature descriptors:

- Colour: Colour Histogram
- Shape: Hu Moments
- Texture: Haralick Texture

Hu Moments

To extract Hu Moments characteristics from a picture, we utilise the OpenCV function cv2.HuMoments(). The moments of the picture flattened by cv2.moments() are passed as a

parameter to this function. That is, we compute the image's moments and convert it to a vector using flatten (). Before we do it, we change our colour image to grayscale because moments expect grayscale photos. The result and the code are shown in the Figure 5 and Figure 6 below on a potato leaf that has a disease called (Potato Early Blight)

Figure 5 : Potato Early Blight Image

```python
# import the necessary packages
import cv2
import imutils
from google.colab.patches import cv2_imshow

# load the input image and convert it to grayscale
image = cv2.imread(
    "/content/uni-plant-village/planet_village/test/PotatoEarlyBlight2.JPG")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# find the contours of the three planes in the image
cnts = cv2.findContours(image.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

# loop over each contour
for (i, c) in enumerate(cnts):
    # extract the ROI from the image and compute the Hu Moments feature
    # vector for the ROI
    (x, y, w, h) = cv2.boundingRect(c)
    roi = image[y:y + h, x:x + w]
    moments = cv2.HuMoments(cv2.moments(roi)).flatten()

    # show the moments and ROI
    print("MOMENTS are #{}: {}".format(i + 1, moments))
    cv2_imshow( roi)
    cv2.waitKey(0)
```

Figure 6 : Potato Early Blight Image code

Haralick Textures

We utilise the mahotas library to extract Haralick Texture features from an image. We will use the function mahotas.features.haralick (). First, we transform our colour image to grayscale because the haralick feature descriptor expects photos to be grayscale. The result and the code are shown in the Figure 7 below on a potato leaf that has a disease called (Potato Early Blight)

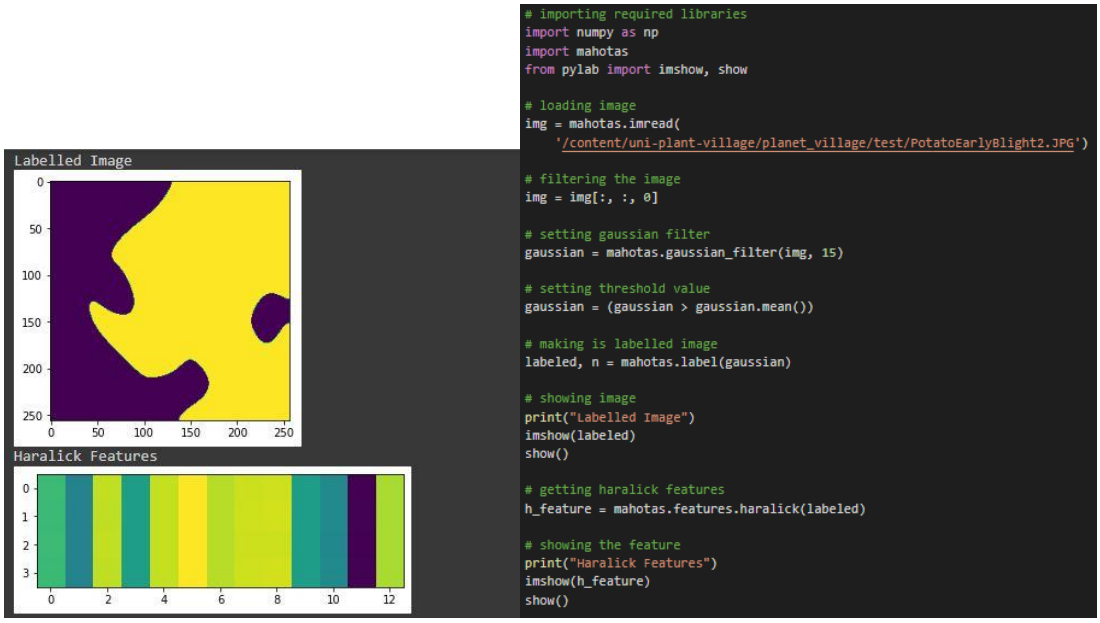


Figure 7: Haralick Textures

Color Histogram

To extract Color Histogram characteristics from an image, we utilise OpenCV's cv2.calcHist() function. The picture, channels, mask, histSize (bins), and ranges for each channel [usually 0-256] are the arguments it expects. We next use OpenCV's normalise () function to normalise the histogram and flatten the resulting normalised matrix (). The result and the code are shown in the Figure 8 and below on a potato leaf that has a disease called (Potato Early Blight)

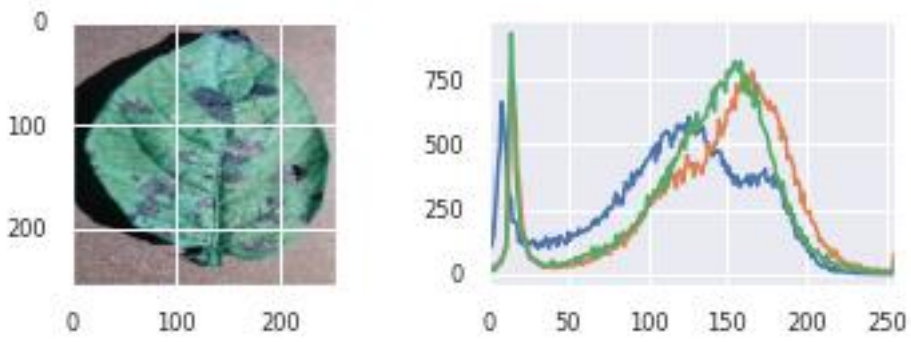


Figure 8: Color Histogram

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('/content/uni-plant-village/planet_village/test/PotatoEarlyBlight2.JPG')

# Calculate histogram without mask
hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
hist2 = cv2.calcHist([img],[1],None,[256],[0,256])
hist3 = cv2.calcHist([img],[2],None,[256],[0,256])

plt.subplot(221), plt.imshow(img)
plt.subplot(222), plt.plot(hist1), plt.plot(hist2),plt.plot(hist3)
plt.xlim([0,256])

plt.show()
```

Data Preparation for models

Data preparation is the procedure of modifying and cleaning up raw data before processing and analysis. Prior to processing, this crucial stage usually entails reformatting input, correcting data, and combining sets of data to improve data.

The Process flow can be seen in the below figure:

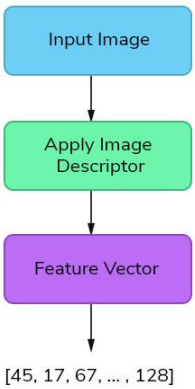


Figure 9 : Image Preparation Flow

We go over the associated folder to obtain all the photographs included within it for each of the training label names. For each picture iteration, we first scale it to a set size. Then, using NumPy's np.hstack() method, we extract the three global features and concatenate them. Using the two lists we built before - labels and data - we maintain track of the feature and its label (global features). You may even consult a dictionary. The code for doing these is provided below.

```
# empty lists to hold feature vectors and labels
data = []
lables = []
# loop over the images in each sub-folder
for image in imagePaths:
    lable = os.path.split(os.path.split(image)[0])[1]
    lables.append(lable)
    # read the image and resize it to a fixed-size
    img = cv2.imread(image)
    img = cv2.resize(img, (32, 32), interpolation=cv2.INTER_AREA)
    # Global Feature extraction
    fv_hu_moments = fd_hu_moments(img )
    fv_haralick   = fd_haralick(img )
    fv_histogram  = fd_histogram(img )
    # Concatenate global features
    img_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])
    # update the list of feature vectors to dataset
    data.append(img_feature)
```

Figure 10 : Image Preparation Function

Normalization and encoding the data

After extracting and concatenating features, we must save this data locally. We utilize LabelEncoder() before saving this data to encode our labels in the right manner. This ensures that the labels are displayed as distinct numbers. Because we employed distinct global features, one feature may outperform the other in terms of value. In such cases, it is preferable to normalize everything within a range (say 0-1). As a result, we use scikit-MinMaxScaler() learn's function to normalize the features. The code for doing these is provided below.

```
# encode the labels as integer
data = np.array(data)
lables = np.array(lables)
class_names = lables
le = LabelEncoder()
lables = le.fit_transform(lables)

# scale features in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
```

Figure 11 : Image Normalization

Vector check and saving the data

The fact that the global feature vector contains 532 columns indicates that when the colour histogram, haralick texture, and hu moments are concatenated, we obtain a single row with 532 columns. As a result, for 35416 photographs, we receive a feature vector of size 35416. (35416, 532). You can also see that the target labels are represented as integer values ranging from 0 to 18, corresponding to the 19 plant disease classifications. The code for accomplishing these tasks is supplied below.

```
dataset_size = data.shape[0]
data = data.reshape(dataset_size, -1)
print(data.shape)
print(lables.shape)
print(dataset_size)
myset = set(lables)
print(myset)

# save the feature vector using HDF5
h5_data          = 'data.h5'
h5_labels        = 'labels.h5'
h5f_data = h5py.File(h5_data, 'w')
h5f_data.create_dataset('dataset_1', data=np.array(data))

h5f_label = h5py.File(h5_labels, 'w')
h5f_label.create_dataset('dataset_1', data=np.array(lables))
```

```
(35416, 532)
(35416,)
35416
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
<HDF5 dataset "dataset_1": shape (35416,), type "<i8">
```

Figure 12 : Saving the Data

MULTICLASS CLASSIFICATION MODELS ARCHITECT

A classification assignment that requires more than two classes; for example, classify a group of photos of fruits that could be oranges, apples, or pears. The assumption in multi-class classification is that each sample is allocated to just one label at a time. We will create and train three models to classify our data:

1. Random forest
2. Simple CNN model
3. Advance CNN model (VGG)

Random forest

A random forest is a combination of tree classifiers that classifies using the tree majority vote. The randomization is due to how each tree is taught. Each node in each tree is divided depending on a random subset of the training data characteristics. To keep the correlation between trees low, the number of variables examined by each node while deciding on a split is often kept much less than the whole feature space for the data set. Furthermore, each tree is trained using a regeneration of the dataset (e.g., 1/3), with the missing samples used to determine the out-of-bag (OOB) error rate.

Creating Training/Test Data

To accomplish this, we will utilise scikit-train test split learn's method to divide our training dataset into train data and test data. We train the models with the train data and test the trained model with the unknown test data in this manner. The test size parameter determines the size of the divide. As shown in the code below.

```
(X_train, X_test, y_train, y_test) = train_test_split(
    data, lables, test_size=0.2, random_state=42, shuffle= True)
```

```
Splitted train and test data...
Train data  : (28332, 532)
Test data   : (28332,)
Train labels: (7084, 532)
Test labels : (7084,)
```

Figure 13 : Dataset Count

Model Optimization

is the process of optimising a model's performance while avoiding overfitting and creating an excessive amount of variance. This is performed in machine learning by picking appropriate "hyperparameters." The hyperparameters of an ML model are its "dials" or "knobs."

We will use and try different numbers of n_estimators and fit the model for each one and plot the accuracy as shown below.

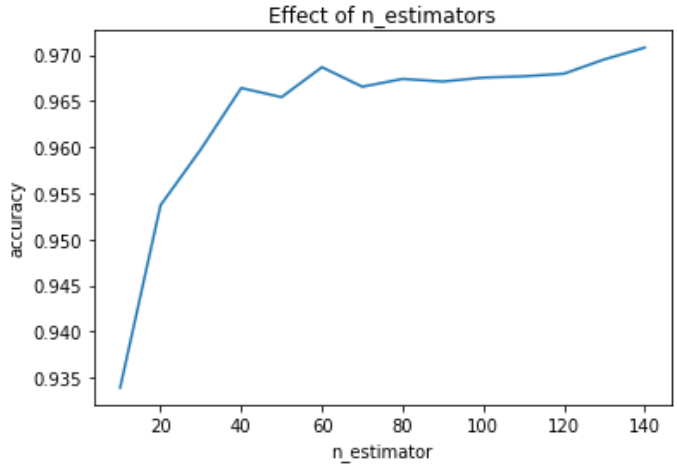


Figure 14 : Effect n_estimators

The result that using number of estimators as 140 will give us the highest accuracy

Train and validate the model

It's time to train our system after extracting, concatenating, and saving global features and labels from our training dataset and we created the sets we want to train. Using sklearn RandomForest algorithm with number of estimators = 140 as we a result from our optimization and K fold validation to accomplish the training and testing. The code is below.

```
'''********************************************************
_________________RandomForest Model_____________
********************************************************'''

from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=140)
model.fit(X_train,y_train)
```

Figure 15 : RandomForst model fit

RandomForest Results

We see validation accuracy after the training , We attain 97% validation accuracy as shown in the below code.

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
y_pred = model.predict(X_test)
accuracy_score(y_pred,y_test)
print(classification_report(y_pred,y_test))
print('Percentage correct: ', round((100*np.sum(y_pred == y_test)/len(y_test)), 3))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97       408
           1       0.99      0.98      0.99       399
           2       1.00      0.99      0.99       356
           3       0.96      0.98      0.97       412
           4       0.99      0.98      0.99       387
           5       0.98      0.96      0.97       393
           6       0.99      1.00      1.00       352
           7       0.99      0.98      0.98       348
           8       1.00      0.97      0.98       372
           9       0.94      0.97      0.95       346
          10       0.96      0.94      0.95       368
          11       0.91      0.96      0.93       354
          12       0.99      0.99      0.99       381
          13       0.91      0.97      0.94       331
          14       0.97      0.94      0.95       361
          15       0.94      0.94      0.94       342
          16       0.96      0.95      0.96       408
          17       1.00      0.97      0.98       359
          18       0.99      1.00      0.99       407

    accuracy                           0.97      7084
   macro avg       0.97      0.97      0.97      7084
weighted avg       0.97      0.97      0.97      7084

Percentage correct:  97.021
```

Figure 16 : Random Forst Acc Result

RandomForest Evaluating Using K-Fold Cross Validation

In this section, I create a random forest classifier and pass it into sklearn's cross validate function. This function is given a model, its training data, a column of target values in an array or dataframe, and the number of folds to cross validate over (the number of models it will train).

The cross-validation object keeps track of each trained model's out-of-sample accuracy. Taking the average of these K out-of-sample scores yields the cross-validation accuracy of the model, which is a low-variance estimate of how the model will perform on unseen data.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X=X_test, y=y_test, cv=10, n_jobs=1)

print('Cross Validation accuracy scores: %s' % scores)

print('Cross Validation accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
Cross Validation accuracy scores: [0.91255289 0.92947814 0.92101551 0.91537377 0.92090395 0.91242938
 0.90960452 0.91525424 0.9039548  0.90677966]
Cross Validation accuracy: 0.915 +/- 0.007
```

Figure 17 : RandomForest K-fold validation

We can see that the accuracy of the evaluation is 91.5% there is a little overfitting but it’s not effecting that much.

RandomForest Evaluating Using Conversion Matrix

With unbalanced classes, it's simple to get high accuracy without producing useful predictions. Thus, accuracy as an assessment metric makes sense only if the class labels are equally distributed. In the situation of unbalanced classes, the confusion-matrix is a helpful tool for summarising the performance of a classification system.

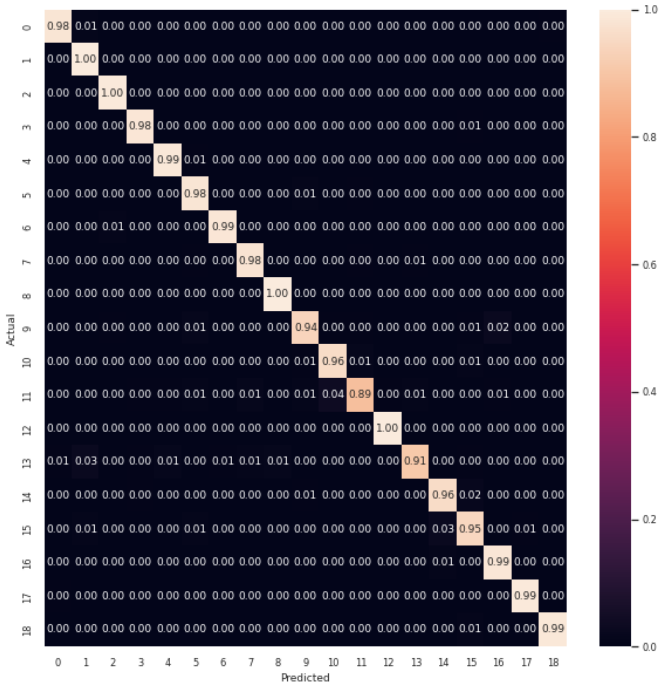


Figure 18 : Conversion Matrix

CNN Models

To begin, the CNN receives an input feature map: a three-dimensional matrix whose first two dimensions correspond to the image's length and breadth in pixels. The third dimension is 3 in length (corresponding to the 3 channels of a colour image: red, green, and blue). The CNN is made up of a series of modules, each of which performs three functions.

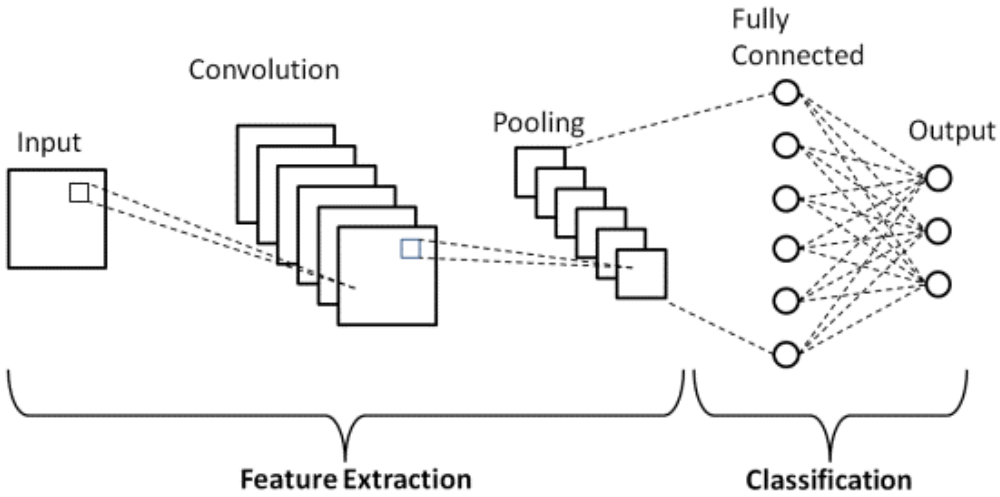


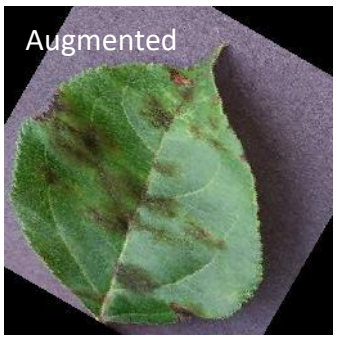
Figure 19 : CNN Design

Data Augmentation

Image data augmentation is a technique for artificially increasing the size of a training dataset by modifying photographs in the dataset.

is the process of changing our training data to make it more diverse, allowing our models to discover more generalizable patterns. Changing the rotation of a picture, flipping it, cropping it, or doing anything similar is an example of altering. The "Keras Image Data Generator" class was used, and the best results were obtained by changing the settings to the following:

- Rotation range: 30
- Shift ranges for width and height = 0.2
- Random zoom = 0.2
- True horizontal flip




```
'''*****************************************************
_________________Data Augmentation for CNN_____________
*****************************************************'''
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

train_ds = train_datagen.flow_from_directory(
    training_dir,
    class_mode='categorical',
    target_size=(124,124),
    batch_size=32,
    shuffle = True

)

valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

valid_ds = valid_datagen.flow_from_directory(
    validation_dir,
    class_mode='categorical',
    target_size=(124,124),
    batch_size=32,
    shuffle = True
)
```

Figure 20 : Data Augmentation

2-layer CNN model

A basic heuristic for computer vision models is the model design that performs best on ImageNet (a large collection of diverse images to benchmark different computer vision models).

.To begin, though, it is best to create a basic model to acquire a baseline result from which to aim for improvement.

Important note: A smaller model in deep learning is one with fewer layers than the state of the art (SOTA). A basic model may have 3-4 layers, but a cutting-edge model, such as ResNet50, may have 50+ layers.

In our situation, we'll take a smaller version of the CNN explanatory model (model 1 from above) and produce a 2-layer CNN model

The structure of the model will be as:

Input -> [Conv + ReLU layers (non-linearities) -> Pooling layer] X 3 -> Fully connected (dense layer)

-> Fully connected (dense layer) as Output as shown in the figures below:

```
'''*******************************************************
_____________Simple CNN model_____________
*******************************************************'''

# Set random seed and number of classes
tf.random.set_seed(42)
num_classes = 19
# create the simple CNN Model
model_cnn= tf.keras.Sequential([
  tf.keras.layers.Conv2D(10, 3, activation='relu',input_shape=(124, 124, 3)),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(10, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(num_classes, activation="softmax")
])
#Compile and Fit the modle
model_cnn.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'])
cnn_history = model_cnn.fit(
  train_ds,
  steps_per_epoch=len(train_ds),
  validation_data=valid_ds,
  validation_steps=len(valid_ds),
  epochs=12,
  verbose=0,
)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 122, 122, 10)      280

 max_pooling2d_5 (MaxPooling  (None, 61, 61, 10)       0
 2D)

 conv2d_6 (Conv2D)           (None, 59, 59, 10)        910

 max_pooling2d_6 (MaxPooling  (None, 29, 29, 10)       0
 2D)

 flatten_2 (Flatten)         (None, 8410)              0

 dense_4 (Dense)             (None, 100)               841100

 dense_5 (Dense)             (None, 19)                1919

=================================================================
Total params: 844,209
Trainable params: 844,209
Non-trainable params: 0
```

2 layers CNN model results

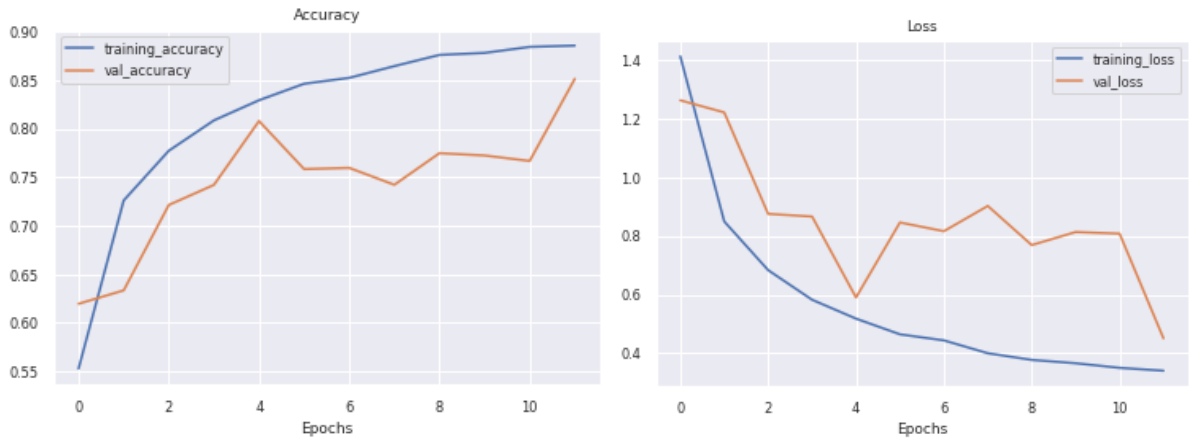
We completed 12 epochs, and this model achieved a progress rate of 85.5% in the train dataset and 83.3% in the approval set that we create as shown in the below code.

```
Epoch 12/12
1107/1107 [==============================] - 175s 158ms/step - loss: 0.3404 - accuracy: 0.8854 - val_loss: 0.4505 - val_accuracy: 0.8511
```

```
# You can also evaluate or predict on a dataset.
print("Evaluate")
result = model_cnn.evaluate(valid_ds)
dict(zip(model_cnn.metrics_names, result))
```

```
Evaluate
277/277 [==============================] - 15s 53ms/step - loss: 0.4505 - accuracy: 0.8511
{'accuracy': 0.8511407375335693, 'loss': 0.4505454897880554}
```

We can plot the loss and accuracy for both train and validation data as shown below. Clearly, we can see that there is overfitting so to reduce that we did another model in the next section



VGG model

VGG (Visual Geometry Group) is a multi-layered deep Convolutional Neural Network (CNN) architecture. The "deep" refers to the number of layers, with VGG-16 and VGG-19 having 16 and 19, respectively, convolutional layers.

The VGG architecture is the building block for cutting-edge visual recognition models. The VGGNet, a deep neural network, exceeds thresholds on a wide range of tasks and datasets other than ImageNet. It is also one of the most often used machine vision frameworks.

The architecture shown below is a scaled-down version of VGG-16, a convolutional neural network that finished second in the 2014 ImageNet classification competition.

After the last CNN model, we add a couple of things to solve the overfiring.

Dropout Regularization

Srivastava et al. suggested Dropout as a regularisation strategy for neural network models in their 2014 study.

Dropout is a method of training where neurons are rejected at random. Randomly, they are "dropped out." This suggests that during the forward pass, their behavioural connection to the downstream stimulation of neurons is abolished and that on the backward trip, no weight modifications are given to the neurons.

The network is much less sensitive to the precise weights of neurons as a result. As a result, the network is more generalizable and less prone to overfit the training data.

Batch Normalization

Batch Norm is a normalisation technique that works on a Neural Network's layers as opposed to the raw data. Instead of doing the complete data gathering, it is done in little portions. Learning is simplified since instruction is expedited and quicker learning rates are used.

There are several theories as to why Batch Norm influences everything. We shall disclose the intuitions of the most crucial causes in this section.

To begin, we can see how standardising the inputs to a comparable range of values might accelerate learning. One easy understanding is that Batch Norm is doing something similar with the values in the network layers, not just the inputs.

Second, Sergey et al. assert in their original study that Batch Norm lowers the network's internal covariate shift. The covariate shift is a shift in the distribution of data.

```
'''*****************************************************
_______________VGG model_____________
*****************************************************'''
# Set random seed and number of classes
tf.random.set_seed(42)
num_classes = 19

from keras.callbacks import EarlyStopping
# Create the TinyVGG Model
model_TinyVGG= tf.keras.Sequential([
  tf.keras.layers.Conv2D(8,(3,3), activation='relu',input_shape=(124, 124, 3),padding='same'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(16,(3,3), activation='relu',padding='same'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(32,(3,3), activation='relu',padding='same'),
  tf.keras.layers.MaxPooling2D((2, 2)),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation="relu"),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(num_classes, activation="softmax")
])
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 124, 124, 8)       224

 max_pooling2d_10 (MaxPoolin  (None, 62, 62, 8)        0
 g2D)

 batch_normalization_6 (Batc  (None, 62, 62, 8)        32
 hNormalization)

 conv2d_11 (Conv2D)          (None, 62, 62, 16)        1168

 max_pooling2d_11 (MaxPoolin  (None, 31, 31, 16)       0
 g2D)

 batch_normalization_7 (Batc  (None, 31, 31, 16)       64
 hNormalization)

 conv2d_12 (Conv2D)          (None, 31, 31, 32)        4640

 max_pooling2d_12 (MaxPoolin  (None, 15, 15, 32)       0
 g2D)

 batch_normalization_8 (Batc  (None, 15, 15, 32)       128
 hNormalization)

 flatten_4 (Flatten)         (None, 7200)              0

 dense_8 (Dense)             (None, 128)               921728

 dropout_2 (Dropout)         (None, 128)               0

 dense_9 (Dense)             (None, 19)                2451

=================================================================
Total params: 930,435
Trainable params: 930,323
Non-trainable params: 112
_________________________________________________________________
```

VGG model results

We completed 12 epochs, and this model achieved a progress rate of 92.5% in the train dataset and 88.8% in the approval set that we created.as shown in the below code.

```
Epoch 20/20
1107/1107 [==============================] - 176s 159ms/step - loss: 0.2060 - accuracy: 0.9325 - val_loss: 0.2346 - val_accuracy: 0.9262
```

```
# You can also evaluate or predict on a dataset.
print("Evaluate")
result_VGG = model_TinyVGG.evaluate(valid_ds)
dict(zip(model_TinyVGG.metrics_names, result_VGG))
```

```
Evaluate
277/277 [==============================] - 15s 53ms/step - loss: 0.2346 - accuracy: 0.9262
{'accuracy': 0.9262480139732361, 'loss': 0.23462514579296112}
```

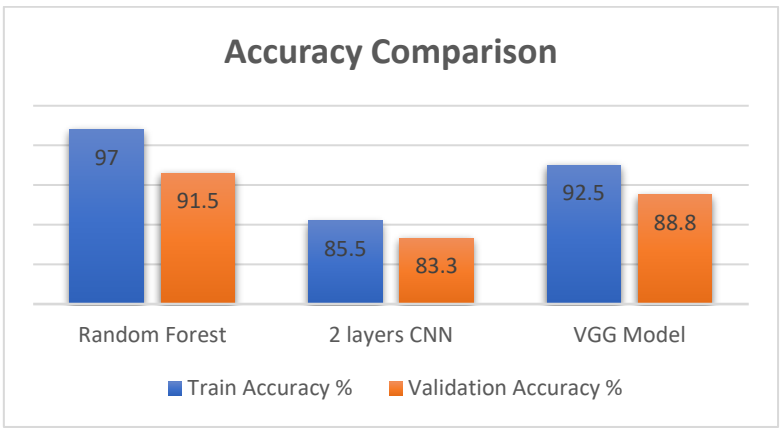
We can plot the loss and accuracy for both train and validation data as shown below. Clearly, we did fix the overfitting and gain a better accuracy

Models Comparison

The three basic plant disease detection procedures are feature extraction, data augmentation and classification. The project compares three illness prediction classification techniques: RandomForest, 2-layer CNN and VGG model. The results of these three classifiers are compared in terms of accuracy.

Accuracy is defined as the number of correctly categorized points divided by the total number of points multiplied by 100.

Figure below compares the accuracy of RandomForest, 2-layer CNN and VGG model classifiers for detecting plant diseases. It has been determined that RandomForest outperforms VGG Model in terms of plant disease identification accuracy.



Results Observations

After training and evaluate all three models here is some observations:

1. The Random Forest Classifier r is a strong model that outperformed the other two models.

2. The VGG has performed close to the Random Forest model, and it would outperform it if it got trained for more epochs.

3. The 2-layer CNN fared the poorest due to overfitting the data.

4. The top algorithms are listed in the following order: Random Forest Regressor > VGG > 2-layer CNN.

5. The Random Forest Classifier beat the VGG (20 Epochs) and 2-layer CNN even after hyperparameter adjustment.

Conclusion

This project analyses and summarises numerous plant disease detection strategies based on image processing that have been deployed by several researchers in recent years. The main techniques used were: Random Forest Classifier, 2-layer CNN, VGG CNN model.

Despite several drawbacks, this study indicates that these disease detection algorithms are efficient and accurate enough to run the system built for detection of leaf diseases. As a result, there is still much that may be done in this sector to improve on previous efforts.

Recommendations and future work

Predicting potato disease detection is a difficult issue. We will create a system that will forecast if the leaf is healthy or sick based on the results of our research. It is a project that involves both software and hardware. So, in the future, we will collaborate with both software and hardware to get better and more real-time results.

Our next efforts will be focused on automatically assessing the severity of the diagnosed ailment.

In the future, a smartphone-assisted system will be created to automate the detecting procedure.

By recognising the sickness, we would be able to provide farmers in the United Kingdom with immediate support and advise on their problems.

References

- Prashanthi, V. (2020). Plant Disease Detection Using Convolutional Neural Networks. International Journal of Advanced Trends in Computer Science and Engineering, 9(3), 2632–2637. https://doi.org/10.30534/IJATCSE/2020/21932020

- Fujita E, Kawasaki Y, Uga H, Kagiwada S, Iyatomi H (2016) Basic investigation on a robust and practical plant diagnostic system. In: Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016, pp 989–992

- M. K. R. Asif, M. A. Rahman and M. H. Hena, "CNN based Disease Detection Approach on Potato Leaves," 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), 2020, pp. 428-432, doi: 10.1109/ICISS49785.2020.9316021.

- Bhange, M., & Hingoliwala, H. A. (2015). Smart Farming: Pomegranate Disease Detection Using Image Processing. Procedia Computer Science, 58, 280–288. https://doi.org/10.1016/J.PROCS.2015.08.022

- Pujari, J. D., Yakkundimath, R., & Byadgi, A. S. (2015). Image Processing Based Detection of Fungal Diseases in Plants. Procedia Computer Science, 46, 1802–1808. https://doi.org/10.1016/J.PROCS.2015.02.137

- GeeksforGeeks. 2022. *OpenCV Python Program to analyze an image using Histogram - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/opencv-python-program-analyze-image-using-histogram/> [Accessed 30 August 2022].

- S. Raina and A. Gupta, "A Study on Various Techniques for Plant Leaf Disease Detection Using Leaf Image," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 900-905, doi: 10.1109/ICAIS50930.2021.9396023.

- V. V. Srinidhi, A. Sahay and K. Deeba, "Plant Pathology Disease Detection in Apple Leaves Using Deep Convolutional Neural Networks : Apple Leaves Disease Detection using EfficientNet and DenseNet," 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), 2021, pp. 1119-1127, doi: 10.1109/ICCMC51019.2021.9418268.

- M. Sardogan, A. Tuncer and Y. Ozen, "Plant Leaf Disease Detection and Classification Based on CNN with LVQ Algorithm," 2018 3rd International Conference on Computer Science and Engineering (UBMK), 2018, pp. 382-385, doi: 10.1109/UBMK.2018.8566635.

- G. K. Sandhu and R. Kaur, "Plant Disease Detection Techniques: A Review," 2019 International Conference on Automation, Computational and Technology Management (ICACTM), 2019, pp. 34-38, doi: 10.1109/ICACTM.2019.8776827.

- M. Islam, Anh Dinh, K. Wahid and P. Bhowmik, "Detection of potato diseases using image segmentation and multiclass support vector machine," 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), 2017, pp. 1-4, doi: 10.1109/CCECE.2017.7946594.

- M. I. Tarik, S. Akter, A. A. Mamun and A. Sattar, "Potato Disease Detection Using Machine Learning," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 800-803, doi: 10.1109/ICICV50876.2021.9388606.

- S. V. Militante, B. D. Gerardo and N. V. Dionisio, "Plant Leaf Detection and Disease Recognition using Deep Learning," 2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE), 2019, pp. 579-582, doi: 10.1109/ECICE47484.2019.8942686.

- P. Patil, N. Yaligar and S. M. Meena, "Comparision of Performance of Classifiers - SVM, RF and ANN in Potato Blight Disease Detection Using Leaf Images," 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2017, pp. 1-5, doi: 10.1109/ICCIC.2017.8524301.

- T. -Y. Lee, J. -Y. Yu, Y. -C. Chang and J. -M. Yang, "Health Detection for Potato Leaf with Convolutional Neural Network," 2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN), 2020, pp. 289-293, doi: 10.1109/Indo-TaiwanICAN48429.2020.9181312.

- H. Ajra, M. K. Nahar, L. Sarkar and M. S. Islam, "Disease Detection of Plant Leaf using Image Processing and CNN with Preventive Measures," 2020 Emerging Technology in Computing, Communication and Electronics (ETCCE), 2020, pp. 1-6, doi: 10.1109/ETCCE51779.2020.9350890.

- K. Karthik, S. Rajaprakash, S. Nazeeb Ahmed, R. Perincheeri and C. R. Alexander, "Tomato And Potato Leaf Disease Prediction With Health Benefits Using Deep Learning Techniques," 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2021, pp. 1-3, doi: 10.1109/I-SMAC52330.2021.9640765.

- GOV.UK. 2022. *Agriculture in the United Kingdom 2021*. [online] Available at: <https://www.gov.uk/government/statistics/agriculture-in-the-united-kingdom-2021> [Accessed 30 August 2022].

- Pavan Kumar, V., Rao, E. G., Anitha, G., & Kumar, G. K. (2021). Plant Disease Detection using Convolutional Neural Networks. Proceedings of the 5th International Conference on Trends in Electronics and Informatics, ICOEI 2021, 1473–1476. https://doi.org/10.1109/ICOEI51242.2021.9453045

- Moid, M. A., & Chaurasia, M. A. (2021). Transfer Learning-based Plant Disease Detection and Diagnosis System using Xception. Proceedings of the 5th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2021, 451–455. https://doi.org/10.1109/I-SMAC52330.2021.9640694

- Marzougui, F., Elleuch, M., & Kherallah, M. (2020). A deep CNN approach for plant disease detection. Proceedings - 2020 21st International Arab Conference on Information Technology, ACIT 2020. https://doi.org/10.1109/ACIT50332.2020.9300072