MACHINE LEARNING

CS-6301

TITLE : SOIL CLASSIFICATION AND PLANT DISEASE IDENTIFICATION USING CNN

MINI PROJECT 100% implementation documentation

Team : 20

Team members : Aakash K – 2018103502

Ashwath Narayan K S - 2018103517

MODULES IMPLEMENTED

MODEL 1 : SOIL CLASSIFICATION

1 Soil Data set Pre-processing

2 Building CNN MODEL for Soil classification

3 Train ,Test ,Evaluate and save Model

4 SOIL Classification

MODEL 2 : PLANT DISEASE IDENTIFICATION

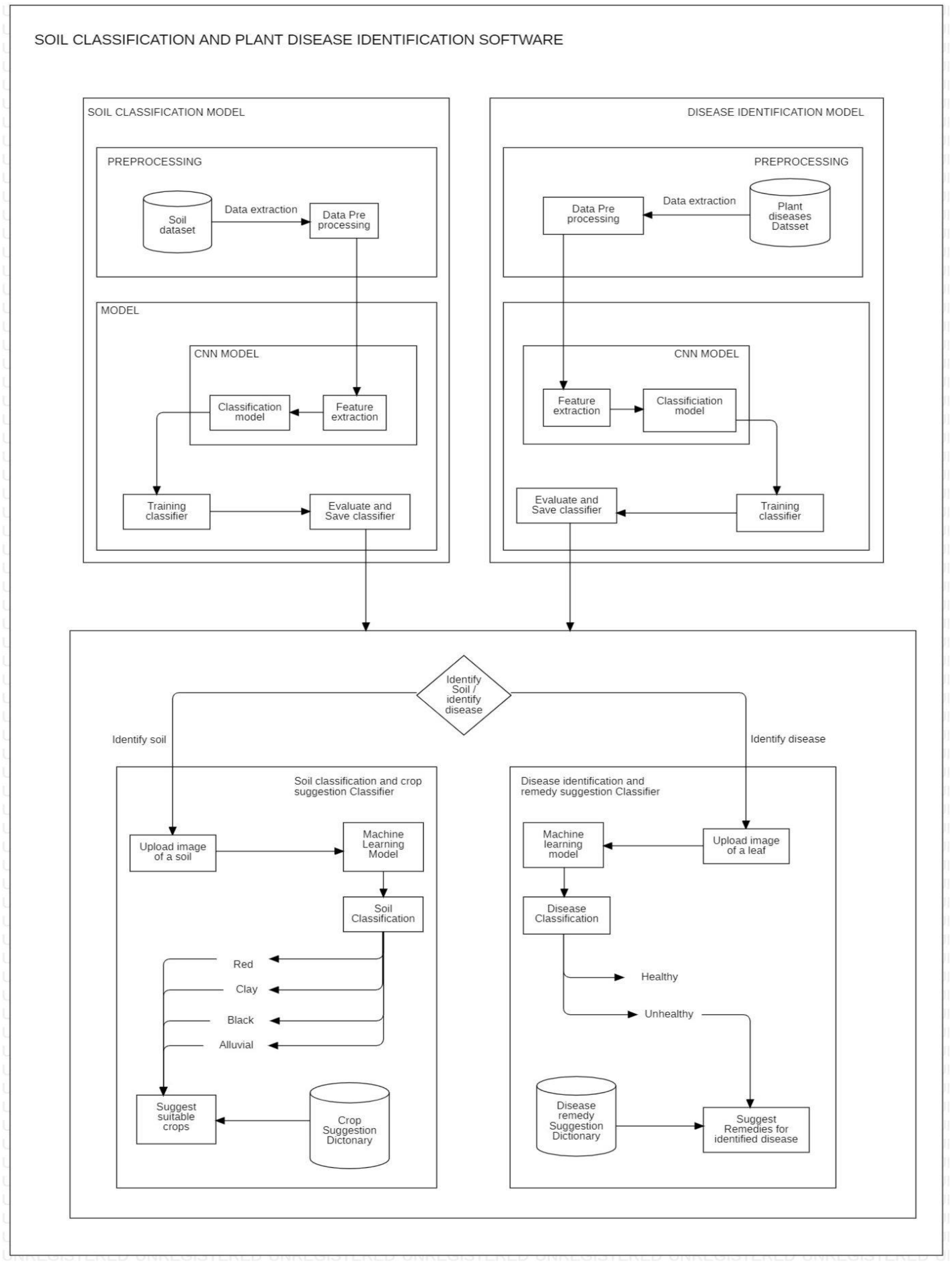
5 Plant disease Data set Pre-processing

6 Building CNN MODEL for Plant disease classification

7 Train ,Test ,Evaluate and save Model

8 Plant disease identification

SYSTEM ARCHITECTURE OF PROPOSED SYSTEM



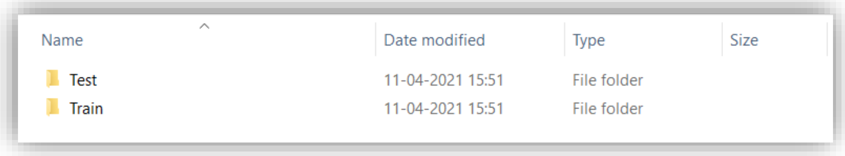
DATASET DESCRIPTION

SOIL DATA-SET

This data-set is created for Soil Type Classification from Image. There are collected 903 RGB images . The main classifications of the data-set are :

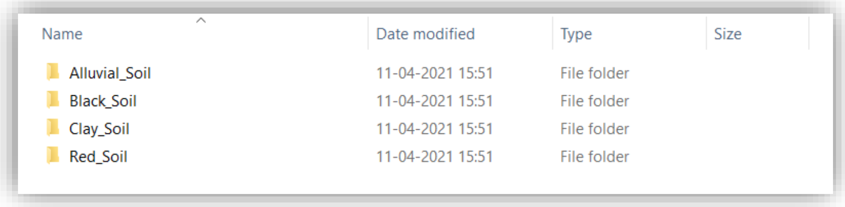
"Alluvial Soil", "Red Soil", "Clay Soil" and "Black Soil".

The dataset has 2 folder : Test and Train



Name	Date modified	Type	Size
Test	11-04-2021 15:51	File folder	
Train	11-04-2021 15:51	File folder	

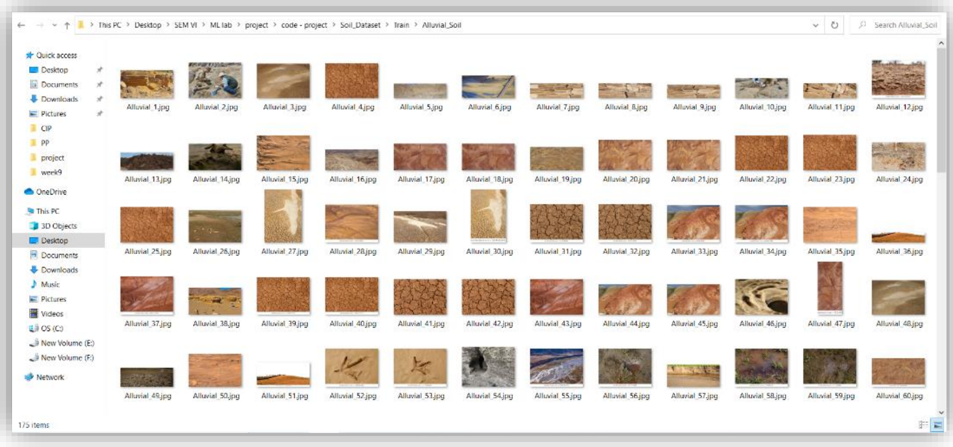
Train and test , both have 4 main folders representing each classification



Name	Date modified	Type	Size
Alluvial_Soil	11-04-2021 15:51	File folder	
Black_Soil	11-04-2021 15:51	File folder	
Clay_Soil	11-04-2021 15:51	File folder	
Red_Soil	11-04-2021 15:51	File folder	

Under train , each soil has 180 images and under train each class has 48 images.

For Alluvial soil which is located under train , has 180 images of Alluvial soil which are collected from various parts of the globe.



MODULES 1 IMPLEMENTATION

In this module,

A. We have divided images into 4 folders so that each folder represents a different soil type. The shape of data is (n,x,y,z) which means that there are n images of size x*y pixels and z means the data contains colored images.

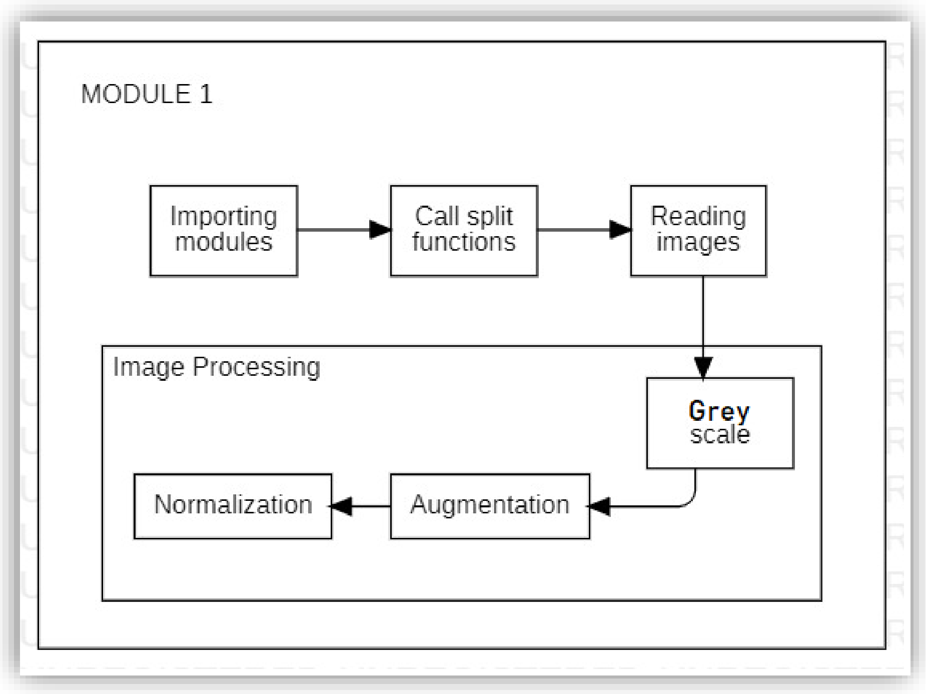
B. With the split folders package, we use the ratio() function for splitting data arrays into subsets (for training data and for validating data and using ImageDataGenerator() method

C. We then start to augment and normalize the images to reduce noise , resize and grey scale them

INPUT : Image files from the dataset.

OUTPUT : Data generated normalised images split into training and testing dataset.

ARCHITECTURE :



Soil Classification

Dataset before splitting

Dataset:

- Train(180img in each class)
  1. Alluvial soil
  2. Black Soil
  3. Clay soil
  4. Red Soil

- Test(48img in each class)
  1. Alluvial soil
  2. Black Soil
  3. Clay soil
  4. Red Soil

Dataset after splitting

Dataset:

- Data

  A Train(144img in each class)

  1. Alluvial soil
  2. Black Soil
  3. Clay soil
  4. Red Soil

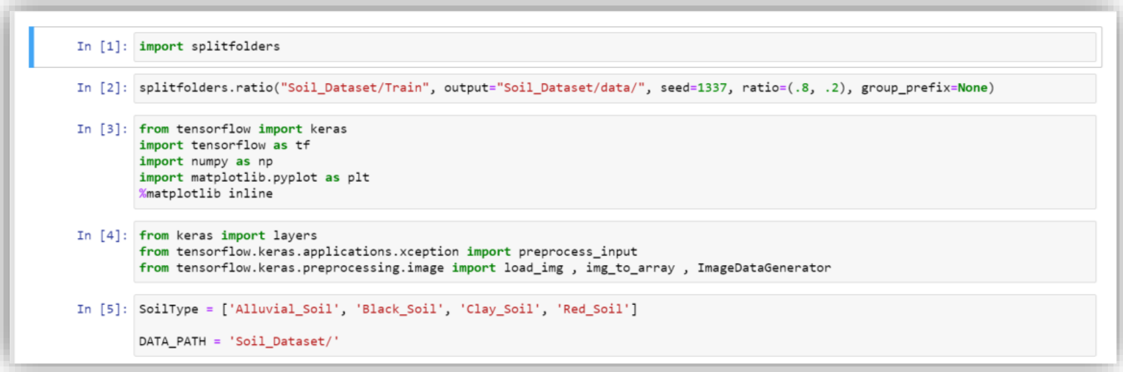
  B Val(36img in each class)

  1. Alluvial soil
  2. Black Soil
  3. Clay soil
  4. Red Soil

- Train(180img in each class)
  1. Alluvial soil
  2. Black Soil
  3. Clay soil
  4. Red Soil

- Test(48img in each class)
  1. Alluvial soil
  2. Black Soil
  3. Clay soil
  4. Red Soil

CODE :



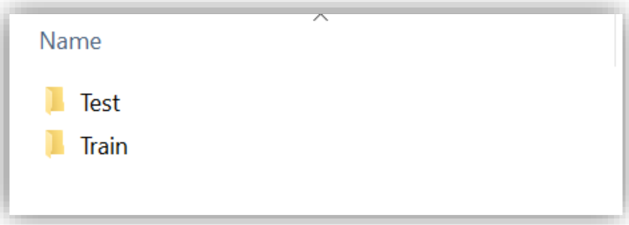
```
In [1]: import splitfolders

In [2]: splitfolders.ratio("Soil_Dataset/Train", output="Soil_Dataset/data/", seed=1337, ratio=(.8, .2), group_prefix=None)

In [3]: from tensorflow import keras
        import tensorflow as tf
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

In [4]: from keras import layers
        from tensorflow.keras.applications.xception import preprocess_input
        from tensorflow.keras.preprocessing.image import load_img , img_to_array , ImageDataGenerator

In [5]: SoilType = ['Alluvial_Soil', 'Black_Soil', 'Clay_Soil', 'Red_Soil']

        DATA_PATH = 'Soil_Dataset/'
```

Data is split to train and validation.

The required modules and packages are imported.

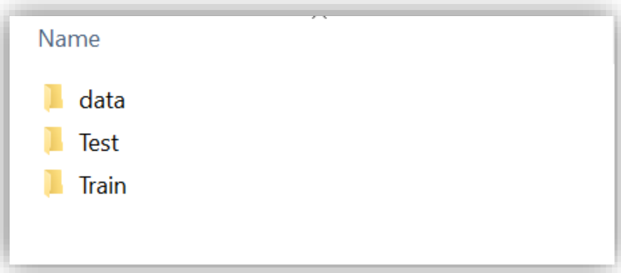
Class type is labelled.

INPUT :



```
Name

Test
Train
```

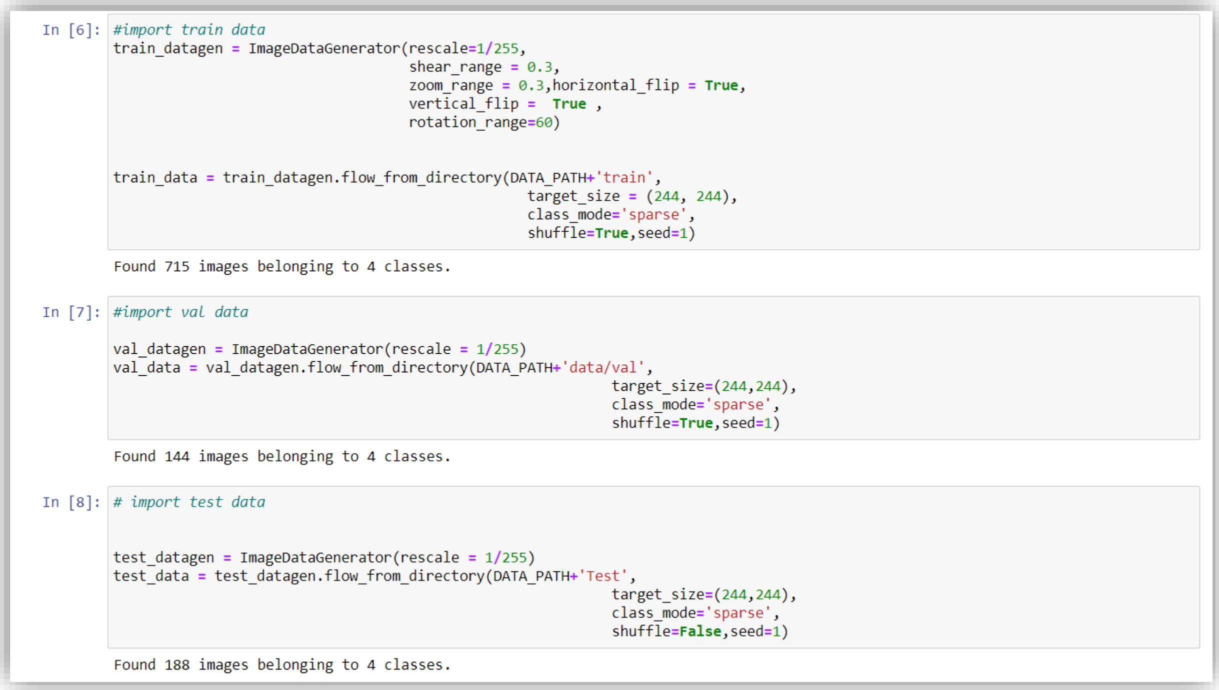
OUTPUT :



```
Name

data
Test
Train
```

Data Pre-processing

RESCALING AND RESIZING



```
In [6]: #import train data
        train_datagen = ImageDataGenerator(rescale=1/255,
                                           shear_range = 0.3,
                                           zoom_range = 0.3,horizontal_flip = True,
                                           vertical_flip =  True ,
                                           rotation_range=60)


        train_data = train_datagen.flow_from_directory(DATA_PATH+'train',
                                                       target_size = (244, 244),
                                                       class_mode='sparse',
                                                       shuffle=True,seed=1)

        Found 715 images belonging to 4 classes.

In [7]: #import val data

        val_datagen = ImageDataGenerator(rescale = 1/255)
        val_data = val_datagen.flow_from_directory(DATA_PATH+'data/val',
                                                         target_size=(244,244),
                                                         class_mode='sparse',
                                                         shuffle=True,seed=1)

        Found 144 images belonging to 4 classes.

In [8]: # import test data


        test_datagen = ImageDataGenerator(rescale = 1/255)
        test_data = test_datagen.flow_from_directory(DATA_PATH+'Test',
                                                         target_size=(244,244),
                                                         class_mode='sparse',
                                                         shuffle=False,seed=1)

        Found 188 images belonging to 4 classes.
```

Train data, validation data and test data are imported, rescaled, resized, shuffled and converted to greyscale which are stored in different labels.

Test Case for Module 1

TEST CASE	**DESCRIPTION**	**DATA INPUT**	**EXPECTED OUTPUT**
T01	Path of the given soil data set is correct	Soil_data with folders Test and Train	Train is copied to Data and is split into Train , Val
T02	Path of the given soil data set is incorrect	No data set folder	Error for not having Soil_data

MODULES 2 IMPLEMENTATION

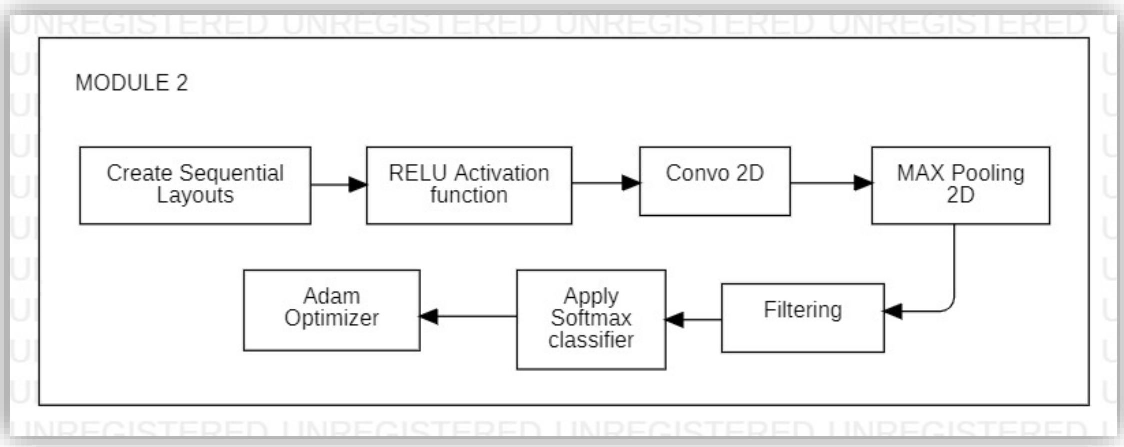
To classify the Soil, we build a CNN (Convolutional Neural Networks) model

A. We use a sequential model. So that, the layers in the network will be added in sequence. We'll use a feed forward network with 6 convolutional layers followed by a fully connected hidden layer. We'll also use dropout layers in between.
B. Dropout regularizes the networks, i.e. it prevents the network from overfitting. All our layers will have relu activations except the output layer. Output layer uses softmax activation as it has to output the probability for each of the classes.
C. Sequential is a keras container for linear stack of layers. Each of the layers in the model needs to know the input shape it should expect, but it is enough to specify input_shape for the first layer of the Sequential model. Rest of the layers do automatic shape inference.
D. To attach a fully connected layer (aka dense layer) to a convolutional layer, we will have to reshape/flatten the output of the conv layer. This is achieved by Flatten layer

INPUT : Parameters for the CNN model

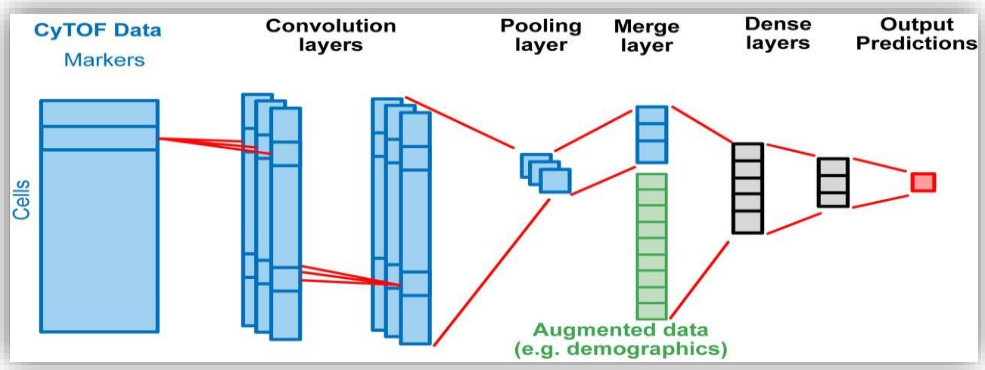
OUTPUT : CNN model.

ARCHITECTURE :

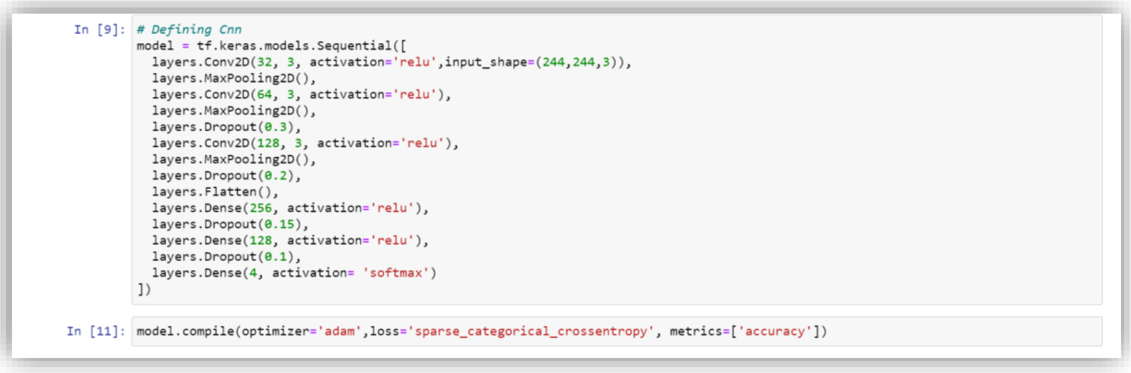


Building CNN model

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.



```
In [9]: # Defining Cnn
        model = tf.keras.models.Sequential([
          layers.Conv2D(32, 3, activation='relu',input_shape=(244,244,3)),
          layers.MaxPooling2D(),
          layers.Conv2D(64, 3, activation='relu'),
          layers.MaxPooling2D(),
          layers.Dropout(0.3),
          layers.Conv2D(128, 3, activation='relu'),
          layers.MaxPooling2D(),
          layers.Dropout(0.2),
          layers.Flatten(),
          layers.Dense(256, activation='relu'),
          layers.Dropout(0.15),
          layers.Dense(128, activation='relu'),
          layers.Dropout(0.1),
          layers.Dense(4, activation= 'softmax')
        ])

In [11]: model.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Here, Feature extraction is done and CNN model by using various layers like convolution, relu, pooling and fully connected in a order.

Finally, The marginal loss in model is optimized using adam optimizer.

MODULES 3 IMPLEMENTATION

In this module

A. During the training, our model will iterate over batches of training sets, each of size batch_size. For each batch, gradients will be computed and updates will be made to the weights of the network automatically.

B. One iteration over all the training set is referred to as an epoch. After building the model architecture, we then train the model using model.fit().

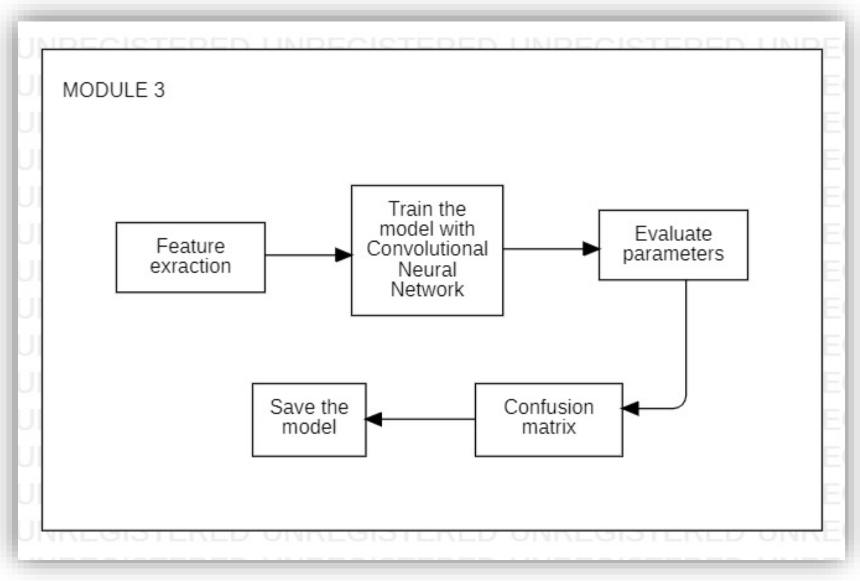
C. The dataset contains a test folder, it has the details related to the image path and their respective class labels. From there, we extract the image path and labels using pandas.

D. Using the confusion matrix, we get the best model with high accuracy.

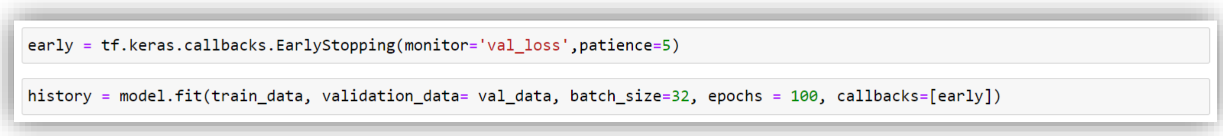
Input : CNN model, Testing and Training data.

Output : Soil Classifier.

ARCHITECTURE :



Training and Validating



```
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=5)

history = model.fit(train_data, validation_data= val_data, batch_size=32, epochs = 100, callbacks=[early])
```

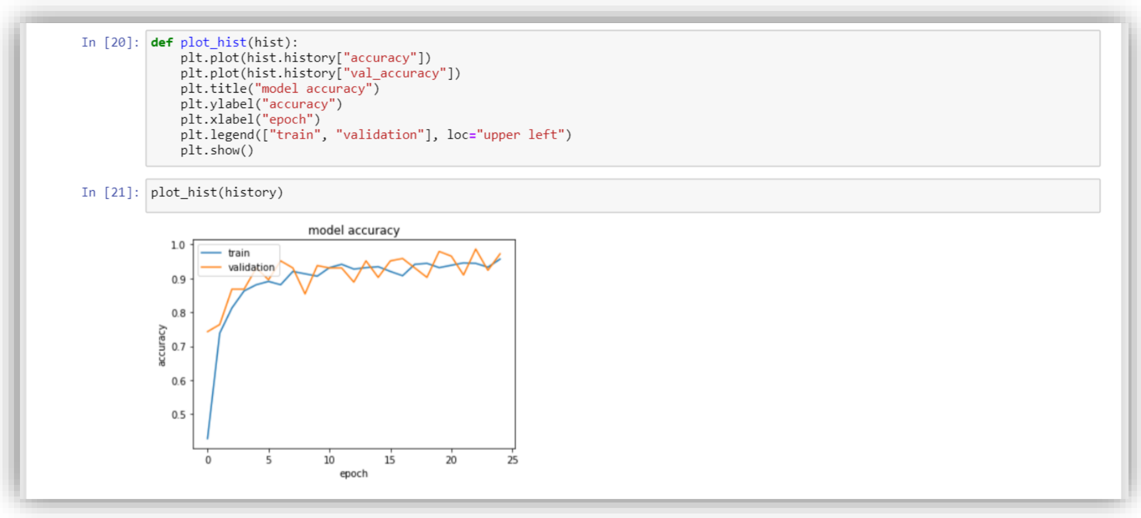
Here, In this section the model is trained with train data set and is validated with validation set. The epochs and batch size are set to be 100 and 32.

The model will be trained and validated continuously within given epochs till stable accuracy and loss is attained which is done using callback attribute.

Here there can be 3 possible test cases : epoch with early callback , epoch set to 50 without call back and epoch set to 75.

Evaluating Trained model (for EPOCH 25)

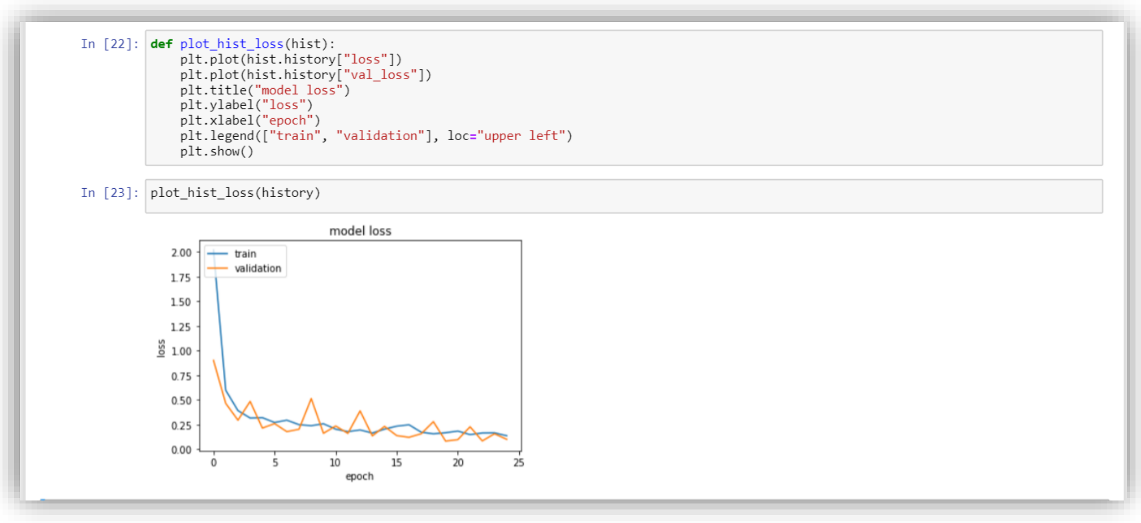
Accuracy Vs Epoch



```
In [20]: def plot_hist(hist):
             plt.plot(hist.history["accuracy"])
             plt.plot(hist.history["val_accuracy"])
             plt.title("model accuracy")
             plt.ylabel("accuracy")
             plt.xlabel("epoch")
             plt.legend(["train", "validation"], loc="upper left")
             plt.show()

In [21]: plot_hist(history)
```

A graph between accuracy and epoch is plotted.The accuracy of the model raises and remains almost unchanged after particular epoch. The accuracy of the model during training and validating comes out to be 95% and 97%.

Loss Vs Epoch

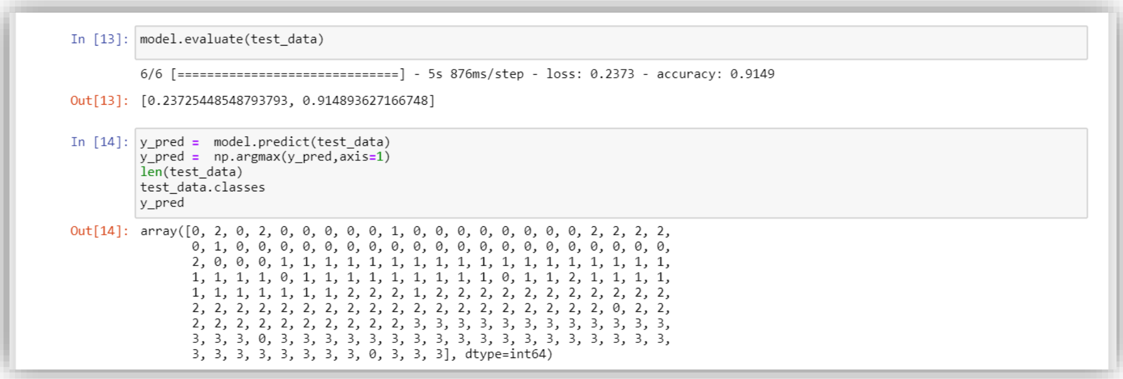


```
In [22]: def plot_hist_loss(hist):
             plt.plot(hist.history["loss"])
             plt.plot(hist.history["val_loss"])
             plt.title("model loss")
             plt.ylabel("loss")
             plt.xlabel("epoch")
             plt.legend(["train", "validation"], loc="upper left")
             plt.show()

In [23]: plot_hist_loss(history)
```

A graph between Loss and epoch is plotted. The Loss in the model lowers and remains almost unchanged after particular epoch. The Loss in the model is lowered and an exceptional loss less than 10% is achieved.

Test Evaluate and Save model

Testing

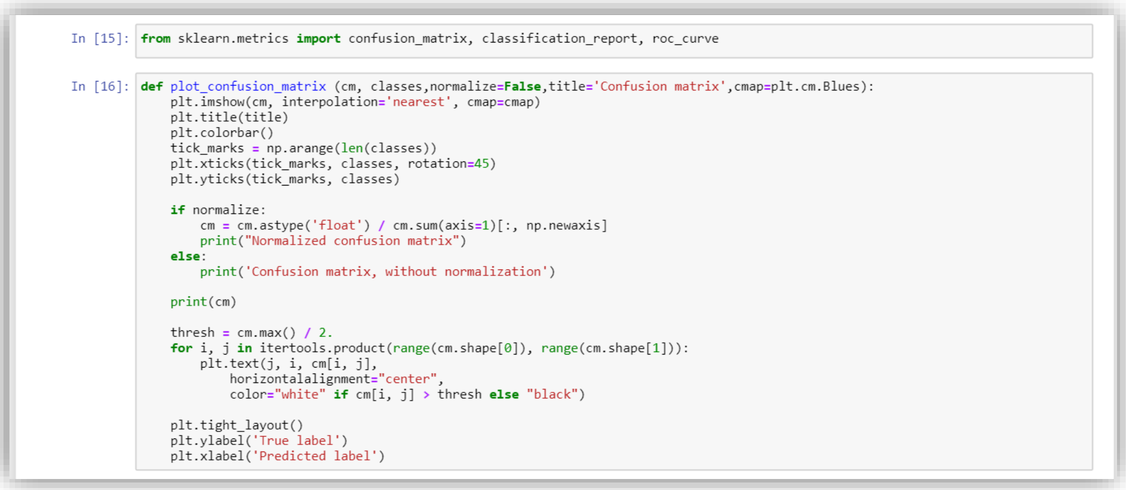


```
In [13]: model.evaluate(test_data)

         6/6 [==============================] - 5s 876ms/step - loss: 0.2373 - accuracy: 0.9149

Out[13]: [0.23725448548793793, 0.914893627166748]

In [14]: y_pred =  model.predict(test_data)
         y_pred =  np.argmax(y_pred,axis=1)
         len(test_data)
         test_data.classes
         y_pred

Out[14]: array([0, 2, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2,
                0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                2, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 2, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3], dtype=int64)
```

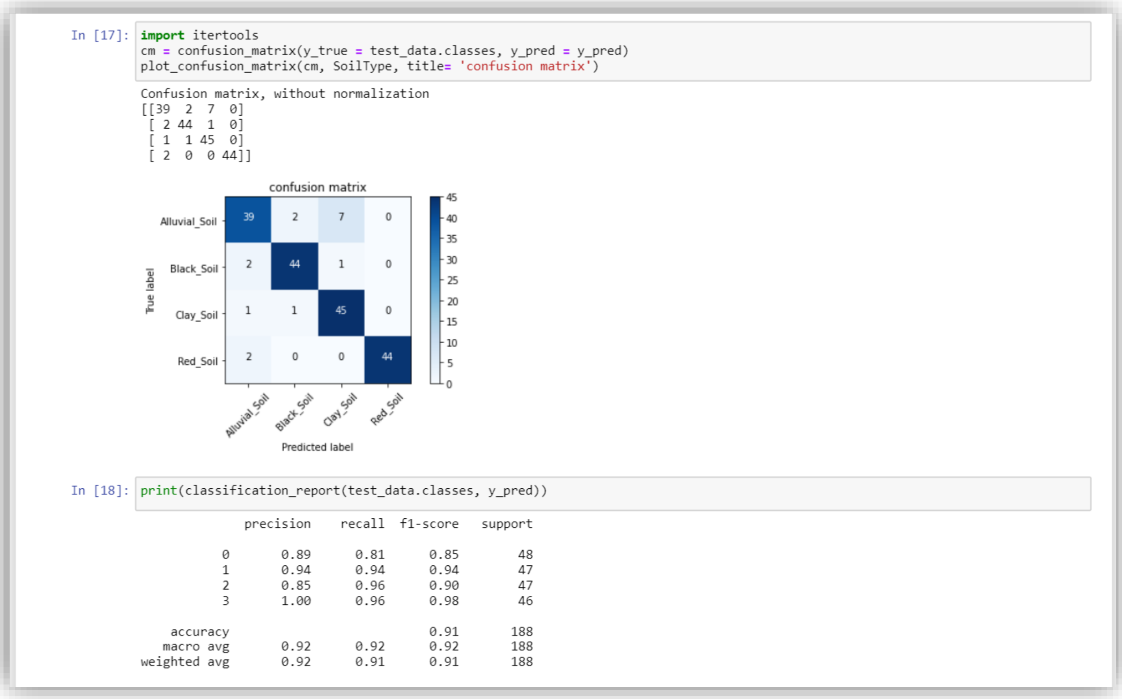
The model is tested with test data and we get an accuracy of 92%.

The model is then used to predict test data set and the necessary classification on different classes is obtained.

Evaluation with confusion Matrix

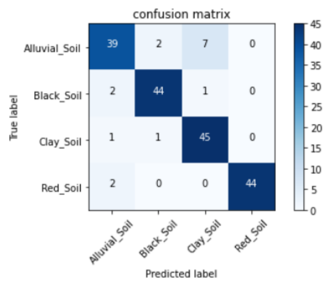


```
In [15]: from sklearn.metrics import confusion_matrix, classification_report, roc_curve

In [16]: def plot_confusion_matrix (cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)

             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             print(cm)

             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

             plt.tight_layout()
             plt.ylabel('True label')
             plt.xlabel('Predicted label')
```



```
In [17]: import itertools
         cm = confusion_matrix(y_true = test_data.classes, y_pred = y_pred)
         plot_confusion_matrix(cm, SoilType, title= 'confusion matrix')
```

```
Confusion matrix, without normalization
[[39  2  7  0]
 [ 2 44  1  0]
 [ 1  1 45  0]
 [ 2  0  0 44]]
```



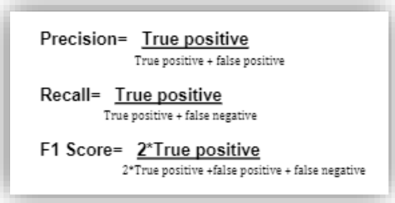
```
In [18]: print(classification_report(test_data.classes, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.81      0.85        48
           1       0.94      0.94      0.94        47
           2       0.85      0.96      0.90        47
           3       1.00      0.96      0.98        46

    accuracy                           0.91       188
   macro avg       0.92      0.92      0.92       188
weighted avg       0.92      0.91      0.91       188
```

The function for confusion matrix construction is written.

The confusion matrix is constructed with test data set and the model’s prediction.

Using Confusion matrix,the evaluation parameters like precision, recall, f1-score and support is obtained.



$$\text{Precision}= \frac{\text{True positive}}{\text{True positive + false positive}}$$

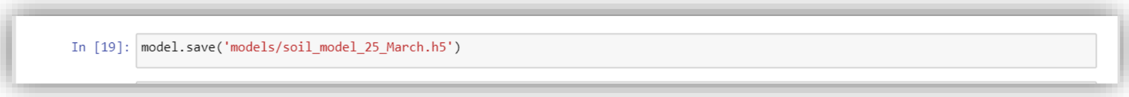
$$\text{Recall}= \frac{\text{True positive}}{\text{True positive + false negative}}$$

$$\text{F1 Score}= \frac{\text{2*True positive}}{\text{2*True positive +false positive + false negative}}$$

Precision , recall and f1-score : 0.92,0.91,0.91

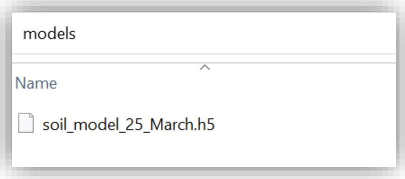
Macro Average : 0.92

Saving model in separate directory



```
In [19]: model.save('models/soil_model_25_March.h5')
```

The final model is saved in a directory called models, inside the directory in which we are currently working.



```
models
Name
soil_model_25_March.h5
```

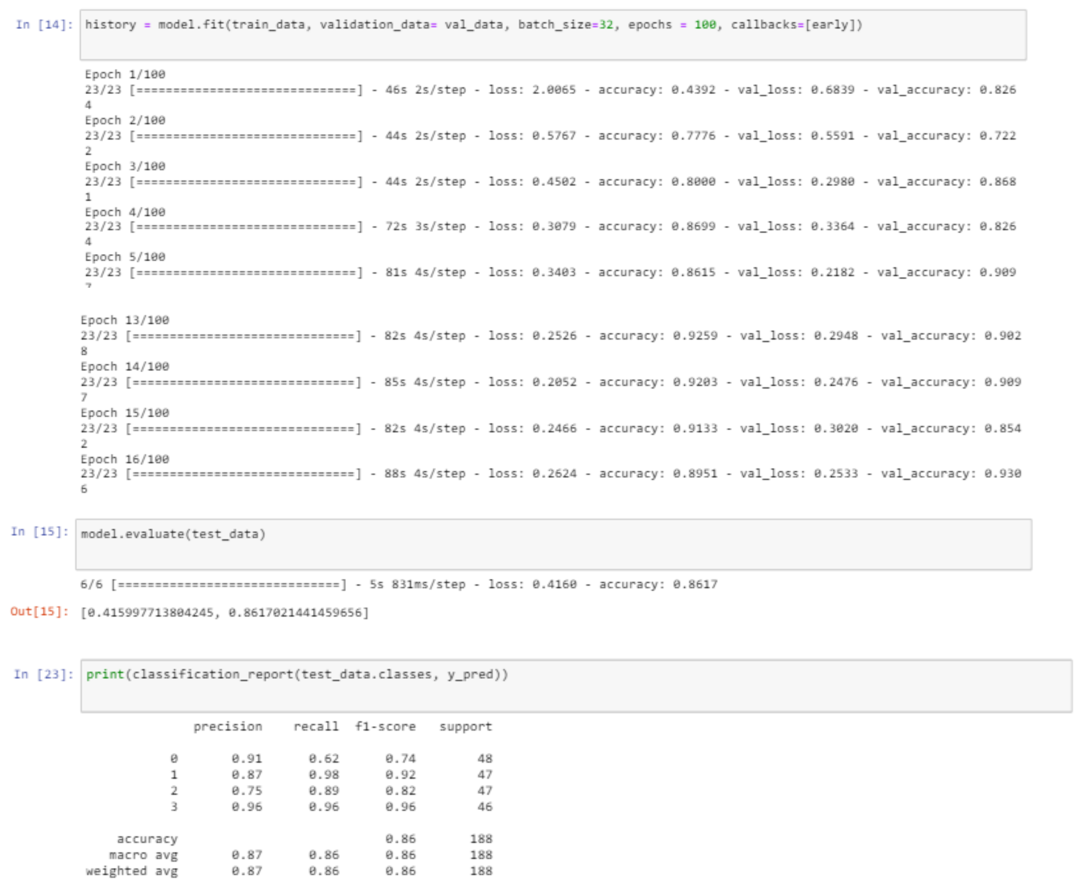
This shows the saved model in the models directory.

TEST CASES for MODULE 3

Each test case along with it’s outcomes are shown in consecutive pages. The model is run with various EPOCH values and it’s outcomes are noted.

Test case 1 :

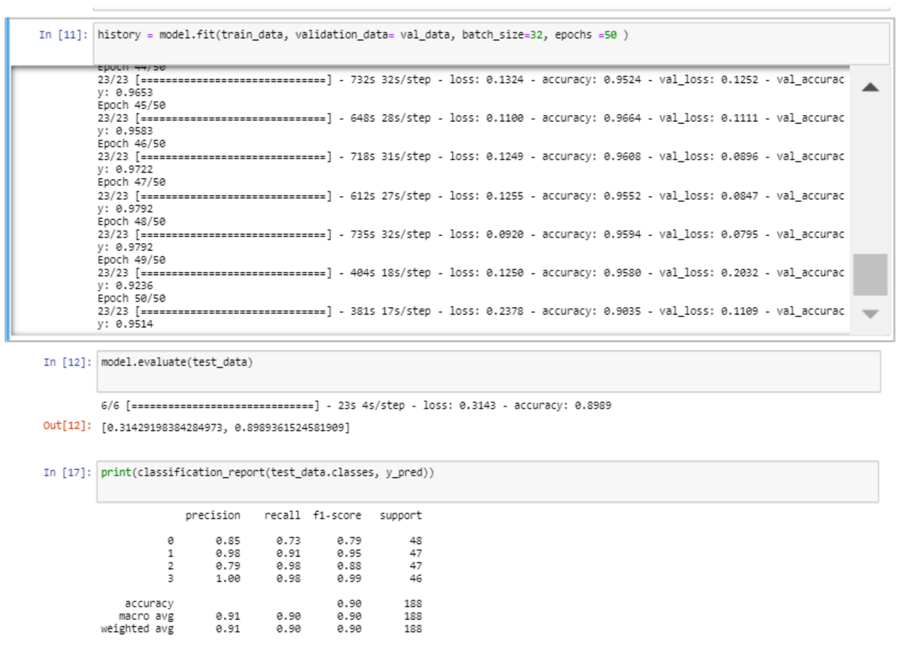
Epoch : 25 epoch



```
In [14]: history = model.fit(train_data, validation_data= val_data, batch_size=32, epochs = 100, callbacks=[early])

Epoch 1/100
23/23 [==============================] - 46s 2s/step - loss: 2.0065 - accuracy: 0.4392 - val_loss: 0.6839 - val_accuracy: 0.826
4
Epoch 2/100
23/23 [==============================] - 44s 2s/step - loss: 0.5767 - accuracy: 0.7776 - val_loss: 0.5591 - val_accuracy: 0.722
2
Epoch 3/100
23/23 [==============================] - 44s 2s/step - loss: 0.4502 - accuracy: 0.8000 - val_loss: 0.2980 - val_accuracy: 0.868
1
Epoch 4/100
23/23 [==============================] - 72s 3s/step - loss: 0.3079 - accuracy: 0.8699 - val_loss: 0.3364 - val_accuracy: 0.826
4
Epoch 5/100
23/23 [==============================] - 81s 4s/step - loss: 0.3403 - accuracy: 0.8615 - val_loss: 0.2182 - val_accuracy: 0.909
7

Epoch 13/100
23/23 [==============================] - 82s 4s/step - loss: 0.2526 - accuracy: 0.9259 - val_loss: 0.2948 - val_accuracy: 0.902
8
Epoch 14/100
23/23 [==============================] - 85s 4s/step - loss: 0.2052 - accuracy: 0.9203 - val_loss: 0.2476 - val_accuracy: 0.909
7
Epoch 15/100
23/23 [==============================] - 82s 4s/step - loss: 0.2466 - accuracy: 0.9133 - val_loss: 0.3020 - val_accuracy: 0.854
2
Epoch 16/100
23/23 [==============================] - 88s 4s/step - loss: 0.2624 - accuracy: 0.8951 - val_loss: 0.2533 - val_accuracy: 0.930
6

In [15]: model.evaluate(test_data)

6/6 [==============================] - 5s 831ms/step - loss: 0.4160 - accuracy: 0.8617

Out[15]: [0.415997713804245, 0.8617021441459656]

In [23]: print(classification_report(test_data.classes, y_pred))

              precision    recall  f1-score   support

           0       0.91      0.62      0.74        48
           1       0.87      0.98      0.92        47
           2       0.75      0.89      0.82        47
           3       0.96      0.96      0.96        46

    accuracy                           0.86       188
   macro avg       0.87      0.86      0.86       188
weighted avg       0.87      0.86      0.86       188
```

Test case 2 :

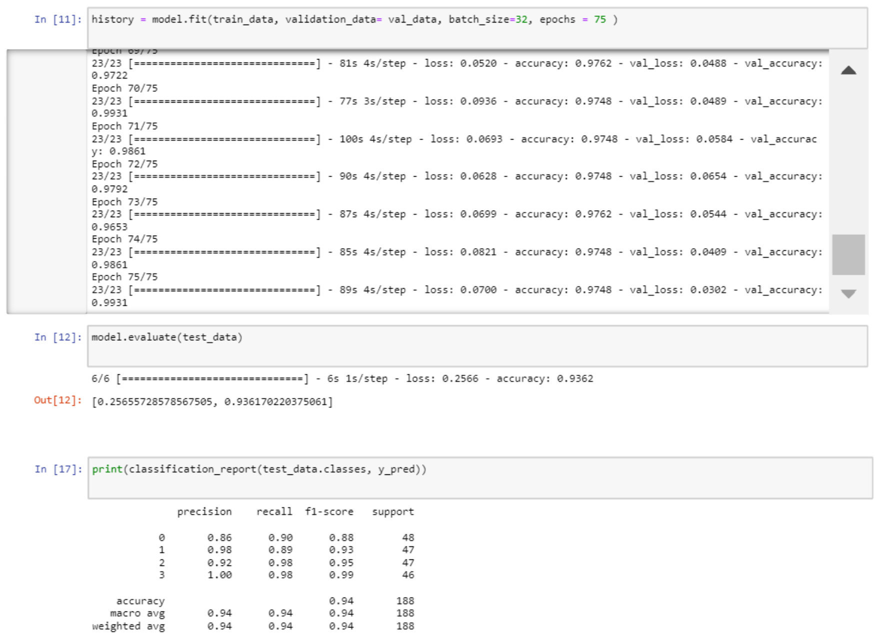
Epoch : 50



```
In [11]: history = model.fit(train_data, validation_data= val_data, batch_size=32, epochs =50 )

23/23 [==============================] - 732s 32s/step - loss: 0.1324 - accuracy: 0.9524 - val_loss: 0.1252 - val_accurac
y: 0.9653
Epoch 45/50
23/23 [==============================] - 648s 28s/step - loss: 0.1100 - accuracy: 0.9664 - val_loss: 0.1111 - val_accurac
y: 0.9583
Epoch 46/50
23/23 [==============================] - 718s 31s/step - loss: 0.1249 - accuracy: 0.9608 - val_loss: 0.0896 - val_accurac
y: 0.9722
Epoch 47/50
23/23 [==============================] - 612s 27s/step - loss: 0.1255 - accuracy: 0.9552 - val_loss: 0.0847 - val_accurac
y: 0.9792
Epoch 48/50
23/23 [==============================] - 735s 32s/step - loss: 0.0920 - accuracy: 0.9594 - val_loss: 0.0795 - val_accurac
y: 0.9792
Epoch 49/50
23/23 [==============================] - 404s 18s/step - loss: 0.1250 - accuracy: 0.9580 - val_loss: 0.2032 - val_accurac
y: 0.9236
Epoch 50/50
23/23 [==============================] - 381s 17s/step - loss: 0.2378 - accuracy: 0.9035 - val_loss: 0.1109 - val_accurac
y: 0.9514

In [12]: model.evaluate(test_data)

6/6 [==============================] - 23s 4s/step - loss: 0.3143 - accuracy: 0.8989
Out[12]: [0.31429198384284973, 0.8989361524581909]

In [17]: print(classification_report(test_data.classes, y_pred))

              precision    recall  f1-score   support

           0       0.85      0.73      0.79        48
           1       0.98      0.91      0.95        47
           2       0.79      0.98      0.88        47
           3       1.00      0.98      0.99        46

    accuracy                           0.90       188
   macro avg       0.91      0.90      0.90       188
weighted avg       0.91      0.90      0.90       188
```

Test case 3 :

Epoch : 75



```
In [11]: history = model.fit(train_data, validation_data= val_data, batch_size=32, epochs = 75 )

23/23 [==============================] - 81s 4s/step - loss: 0.0520 - accuracy: 0.9762 - val_loss: 0.0488 - val_accuracy:
0.9722
Epoch 70/75
23/23 [==============================] - 77s 3s/step - loss: 0.0936 - accuracy: 0.9748 - val_loss: 0.0489 - val_accuracy:
0.9931
Epoch 71/75
23/23 [==============================] - 100s 4s/step - loss: 0.0693 - accuracy: 0.9748 - val_loss: 0.0584 - val_accurac
y: 0.9861
Epoch 72/75
23/23 [==============================] - 90s 4s/step - loss: 0.0628 - accuracy: 0.9748 - val_loss: 0.0654 - val_accuracy:
0.9792
Epoch 73/75
23/23 [==============================] - 87s 4s/step - loss: 0.0699 - accuracy: 0.9762 - val_loss: 0.0544 - val_accuracy:
0.9653
Epoch 74/75
23/23 [==============================] - 85s 4s/step - loss: 0.0821 - accuracy: 0.9748 - val_loss: 0.0409 - val_accuracy:
0.9861
Epoch 75/75
23/23 [==============================] - 89s 4s/step - loss: 0.0700 - accuracy: 0.9748 - val_loss: 0.0302 - val_accuracy:
0.9931

In [12]: model.evaluate(test_data)

6/6 [==============================] - 6s 1s/step - loss: 0.2566 - accuracy: 0.9362
Out[12]: [0.25655728578567505, 0.936170220375061]

In [17]: print(classification_report(test_data.classes, y_pred))

              precision    recall  f1-score   support

           0       0.86      0.90      0.88        48
           1       0.98      0.89      0.93        47
           2       0.92      0.98      0.95        47
           3       1.00      0.98      0.99        46

    accuracy                           0.94       188
   macro avg       0.94      0.94      0.94       188
weighted avg       0.94      0.94      0.94       188
```

TABLE OF INFERENCE FOR TEST CASES

TESTT CASE	EPOCH	TRAIN ACCURACY	TRAIN LOSS	PERCISION	RECALL	F1 SCORE
T01	25	93%	11.9%	91%	90%	91%
T02	50	95.8%	12.8%	91%	90%	90%
T03	75	97%	4%	94%	94%	94%

MODULES 4 IMPLEMENTATION

In module 4

A. We import necessary modules and label the 4 main soil classification.

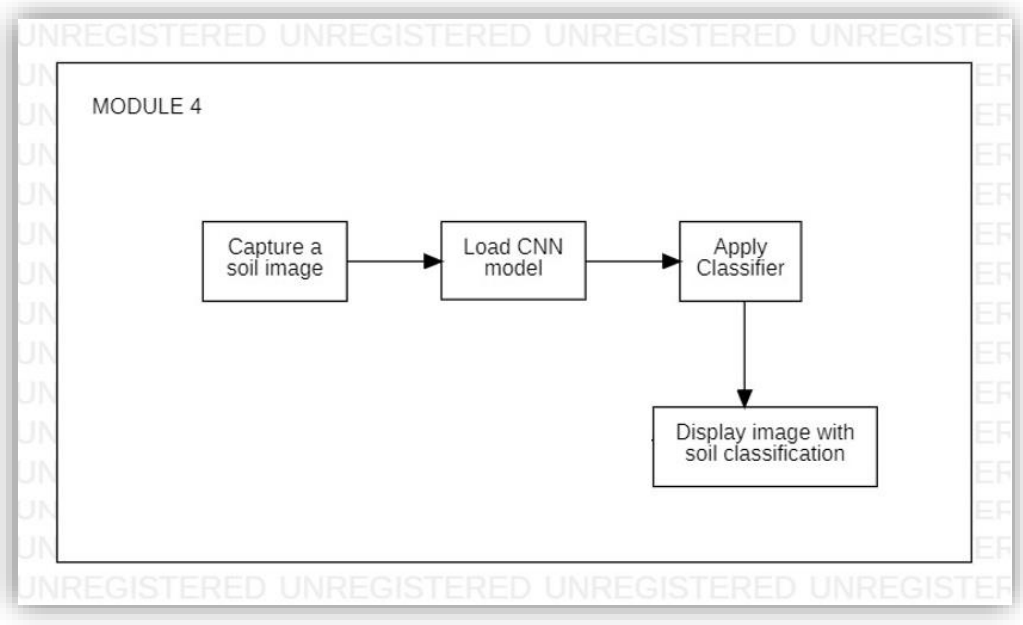
B. The h5 extension model which we saved previously is loaded here.

C. The input image of SOIL is given through path to the model which is loaded and the output is obtained.

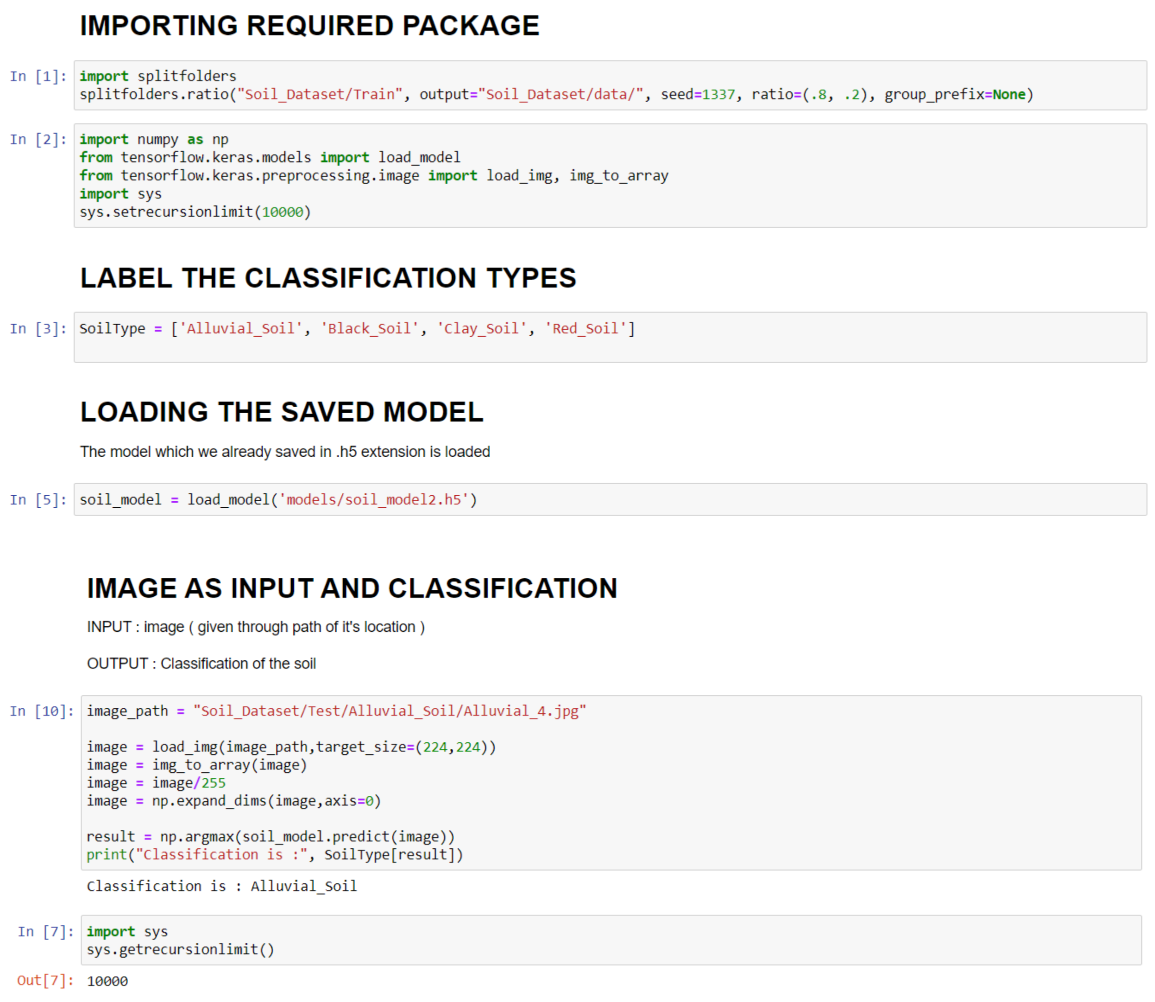
INPUT : Uploaded image of soil.

OUTPUT : Predicted soil type.

ARCHITECTURE :



Classify Soil



IMPORTING REQUIRED PACKAGE

In [1]:
```
import splitfolders
splitfolders.ratio("Soil_Dataset/Train", output="Soil_Dataset/data/", seed=1337, ratio=(.8, .2), group_prefix=None)
```

In [2]:
```
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import sys
sys.setrecursionlimit(10000)
```

LABEL THE CLASSIFICATION TYPES

In [3]:
```
SoilType = ['Alluvial_Soil', 'Black_Soil', 'Clay_Soil', 'Red_Soil']
```

LOADING THE SAVED MODEL

The model which we already saved in .h5 extension is loaded

In [5]:
```
soil_model = load_model('models/soil_model2.h5')
```

IMAGE AS INPUT AND CLASSIFICATION

INPUT : image (given through path of it's location)

OUTPUT : Classification of the soil

In [10]:
```
image_path = "Soil_Dataset/Test/Alluvial_Soil/Alluvial_4.jpg"

image = load_img(image_path,target_size=(224,224))
image = img_to_array(image)
image = image/255
image = np.expand_dims(image,axis=0)

result = np.argmax(soil_model.predict(image))
print("Classification is :", SoilType[result])
```

```
Classification is : Alluvial_Soil
```

In [7]:
```
import sys
sys.getrecursionlimit()
```

Out[7]:
```
10000
```

In this, the necessary modules are imported. The class labels are specified. The saved model is loaded. The image path is specified. The image is preprocessed and predicted with the model. Finally, the Soil is Classified.

TEST CASES for MODULE 4

TEST CASE	TEST CASE OBJECTIVE	TEST CASE DESCRIPTION	INPUT DATA	OUTPUT
T01	Build a CNN model to classify the SOIL from the image uploaded	Make soil prediction : User accesses it after uploading a clear image of soil	Clear soil image uploaded	The built CNN model will classify the soil based on the input
T02	Build a CNN model (result may not be accurate if image uploaded is not clear)	Make soil prediction : User accesses it after uploading a unclear/blurry image of soil	Unclear soil image uploaded	The output may not be precise.
T03	Build a CNN model (result is accurate if image uploaded is soil focused)	Make soil prediction : User accesses it after uploading a image.	Soil along with other things--soil dominant.	Precise Classification
T04	Build a CNN model (result is not accurate if image uploaded is not soil focused)	Make soil prediction : User accesses it after uploading a image.	Soil along with other objects--object dominant.	The output may not be precise.

TEST CASE 1 :

INPUT :

This is a clear image of a black soil which is taken from our data set and is given as input to the CNN model. There are no filters or disturbance in image , hence a precise output is expected

OUTPUT :

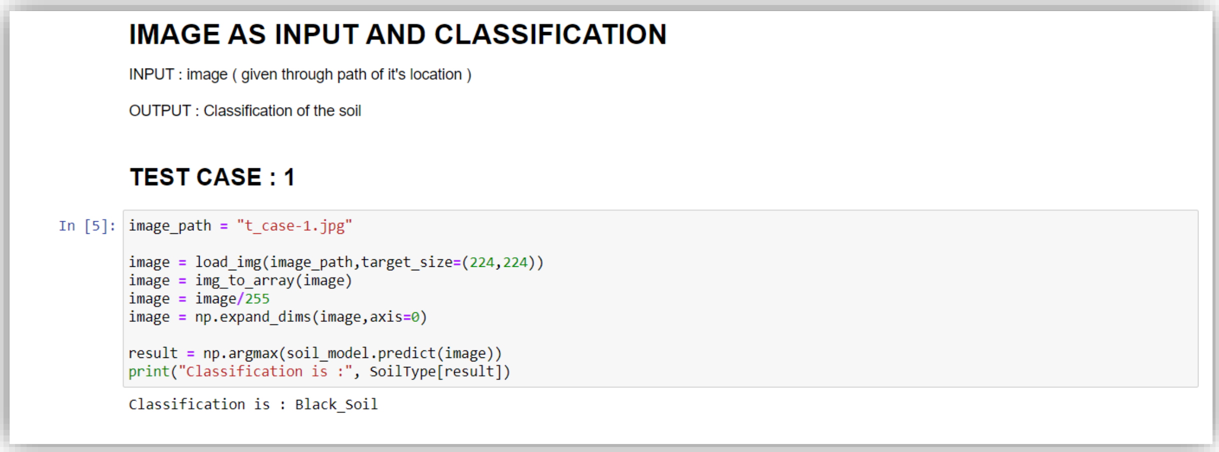


IMAGE AS INPUT AND CLASSIFICATION

INPUT : image (given through path of it's location)

OUTPUT : Classification of the soil

TEST CASE : 1

```
In [5]: image_path = "t_case-1.jpg"

        image = load_img(image_path,target_size=(224,224))
        image = img_to_array(image)
        image = image/255
        image = np.expand_dims(image,axis=0)

        result = np.argmax(soil_model.predict(image))
        print("Classification is :", SoilType[result])

        Classification is : Black_Soil
```

INFERENCE :

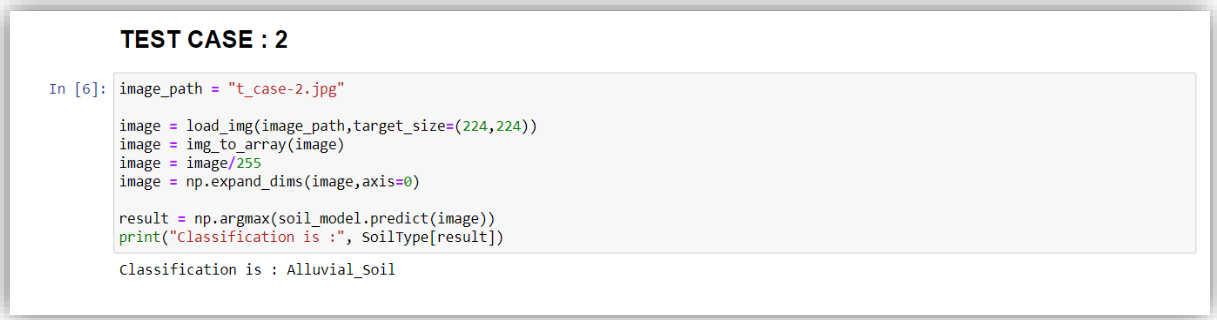
Image uploaded is a sample image of a black soil which is clear and without any filters. We provide this image as a input to our CNN MODEL and the outcome label is BLACK SOIL. Therefore a clear image will give a precise output.

TEST CASE 2 :

INPUT :

This is a image of clay soil taken from the data set. We have added an extra layer of black filter in order to make the image unclear and pass this to the CNN MODEL and test using it. Since the image is not clear and there exists a extra added filter , a misclassification is expected.

OUTPUT :



TEST CASE : 2

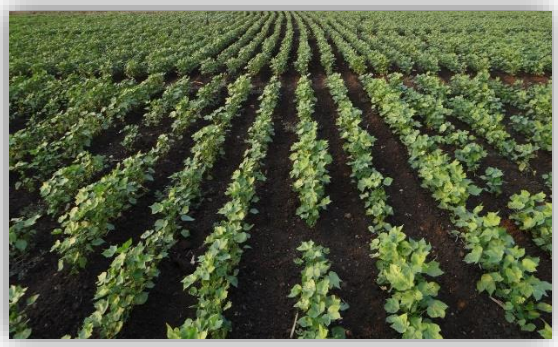
```
In [6]: image_path = "t_case-2.jpg"

        image = load_img(image_path,target_size=(224,224))
        image = img_to_array(image)
        image = image/255
        image = np.expand_dims(image,axis=0)

        result = np.argmax(soil_model.predict(image))
        print("Classification is :", SoilType[result])

        Classification is : Alluvial_Soil
```

INFERENCE :

The uploaded image here is a image of a Clay soil , but image has added filter , which makes it a unclear input for the model. When the trained CNN model uses this input , it misclassifies it as Alluvial soil. Hance a clear image without any filters should be passed

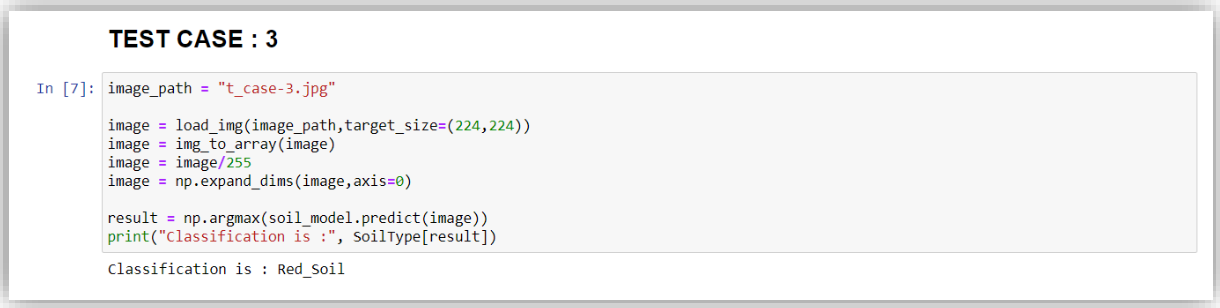
TEST CASE 3 :

INPUT :



Test case 3 and 4 deals with images which contains both soil and plant where one image is soil dominant and another image is plant dominant. This is example of plant dominant. Amount of soil is low when compared to plant in this image. When passed to model , we expect a less precise output.

OUTPUT :



TEST CASE : 3

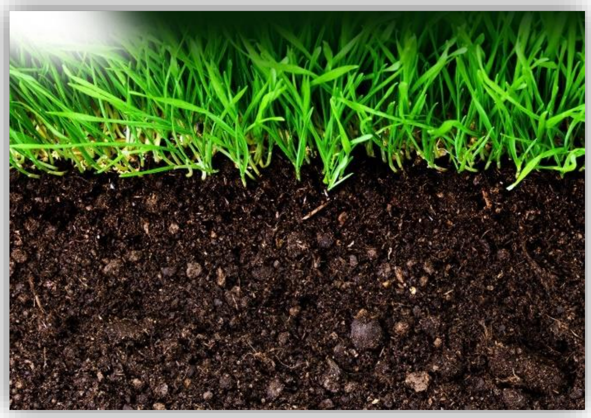
```
In [7]: image_path = "t_case-3.jpg"

        image = load_img(image_path,target_size=(224,224))
        image = img_to_array(image)
        image = image/255
        image = np.expand_dims(image,axis=0)

        result = np.argmax(soil_model.predict(image))
        print("Classification is :", SoilType[result])

        Classification is : Red_Soil
```

INFERENCE :

In this input image , the pic contains both soil and plant but plant is more dominant than the soil which is to be identified. In this case we are able to classify the soil correctly as RED SOIL.

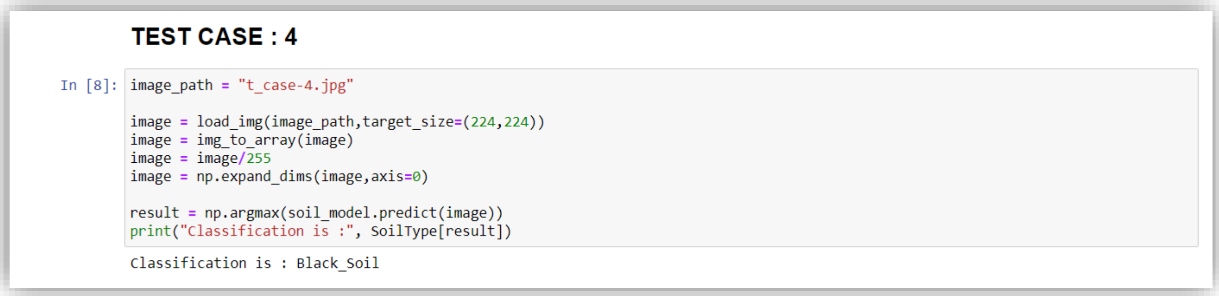
TEST CASE 4 :

INPUT :



This is example for a soil dominant image. There are no added filters and soil is dominant , we expect a precise output when passed to our trained model.

OUTPUT :



TEST CASE : 4

```
In [8]: image_path = "t_case-4.jpg"

        image = load_img(image_path,target_size=(224,224))
        image = img_to_array(image)
        image = image/255
        image = np.expand_dims(image,axis=0)

        result = np.argmax(soil_model.predict(image))
        print("Classification is :", SoilType[result])

        Classification is : Black_Soil
```

INFERENCE :

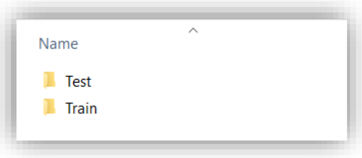
In this test case , soil is predominant than the plant. Here we want to classify the soil , we pass this image to the model and we are getting the output as BLACK SOIL , which is a correct prediction. So if a image contains both soil and plant , our model is able to classify it according to our labels. It may not precise all the time.

DATASET DESCRIPTION

PLANT DISEASE DATA-SET

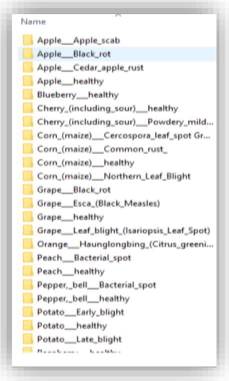
This dataset consists of about 87K RGB images of healthy and diseased crop leaves which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose.

The dataset has 2 folder : Test and Train



```
Name
Test
Train
```

Train and test , both have 38 main folders representing each classification of healthy and un healthy disease.



```
Name
Apple___Apple_scab
Apple___Black_rot
Apple___Cedar_apple_rust
Apple___healthy
Blueberry___healthy
Cherry_(including_sour)___healthy
Cherry_(including_sour)___Powdery_mild...
Corn_(maize)___Cercospora_leaf_spot Gr...
Corn_(maize)___Common_rust_
Corn_(maize)___healthy
Corn_(maize)___Northern_Leaf_Blight
Grape___Black_rot
Grape___Esca_(Black_Measles)
Grape___healthy
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)
Orange___Haunglongbing_(Citrus_greeni...
Peach___Bacterial_spot
Peach___healthy
Pepper,_bell___Bacterial_spot
Pepper,_bell___healthy
Potato___Early_blight
Potato___healthy
Potato___Late_blight
```

For APPLE SCAB disease which is located under train , has images of unhealthy diesease which are collected from various parts of the globe.

MODULES 5 IMPLEMENTATION

In this module,

A. We have divided images into 4 folders so that each folder represents a different soil type. The shape of data is (n,x,y,z) which means that there are n images of size x*y pixels and z means the data contains colored images.

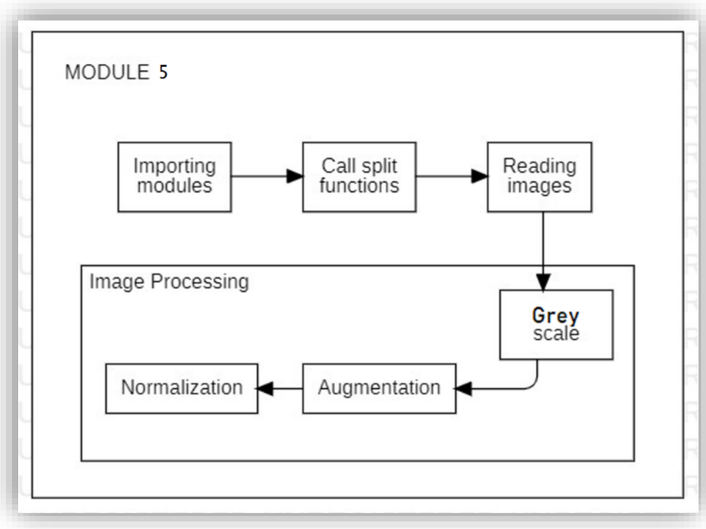
B. With the split folders package, we use the ratio() function for splitting data arrays into subsets (for training data and for validating data and using ImageDataGenerator() method

C. We then start to augment and normalize the images to reduce noise , resize and grey scale them

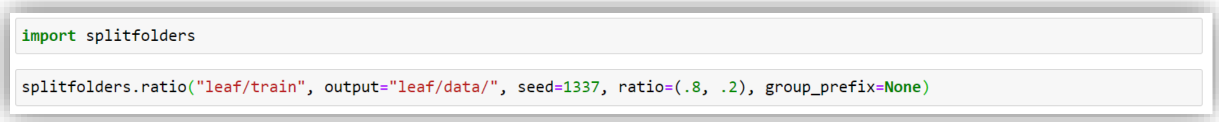
INPUT : Image files from the dataset.

OUTPUT : Data generated normalised images split into training and testing dataset.

ARCHITECTURE :

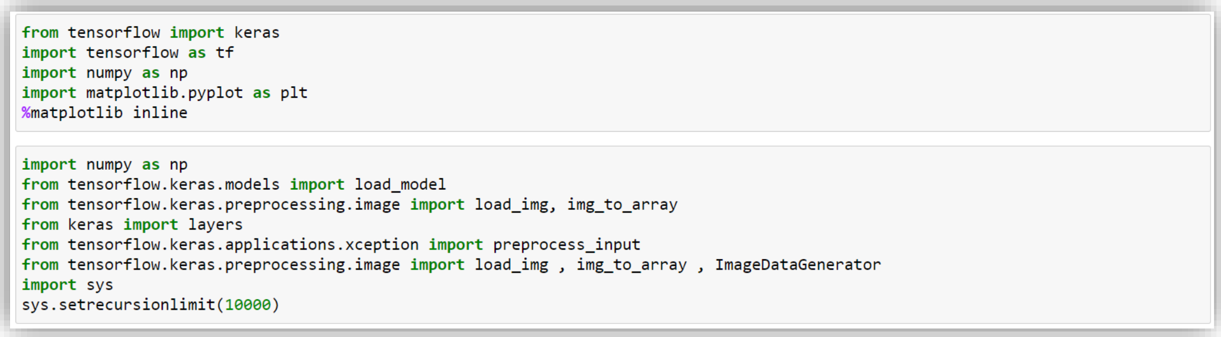


CODE :



```python
import splitfolders

splitfolders.ratio("leaf/train", output="leaf/data/", seed=1337, ratio=(.8, .2), group_prefix=None)
```

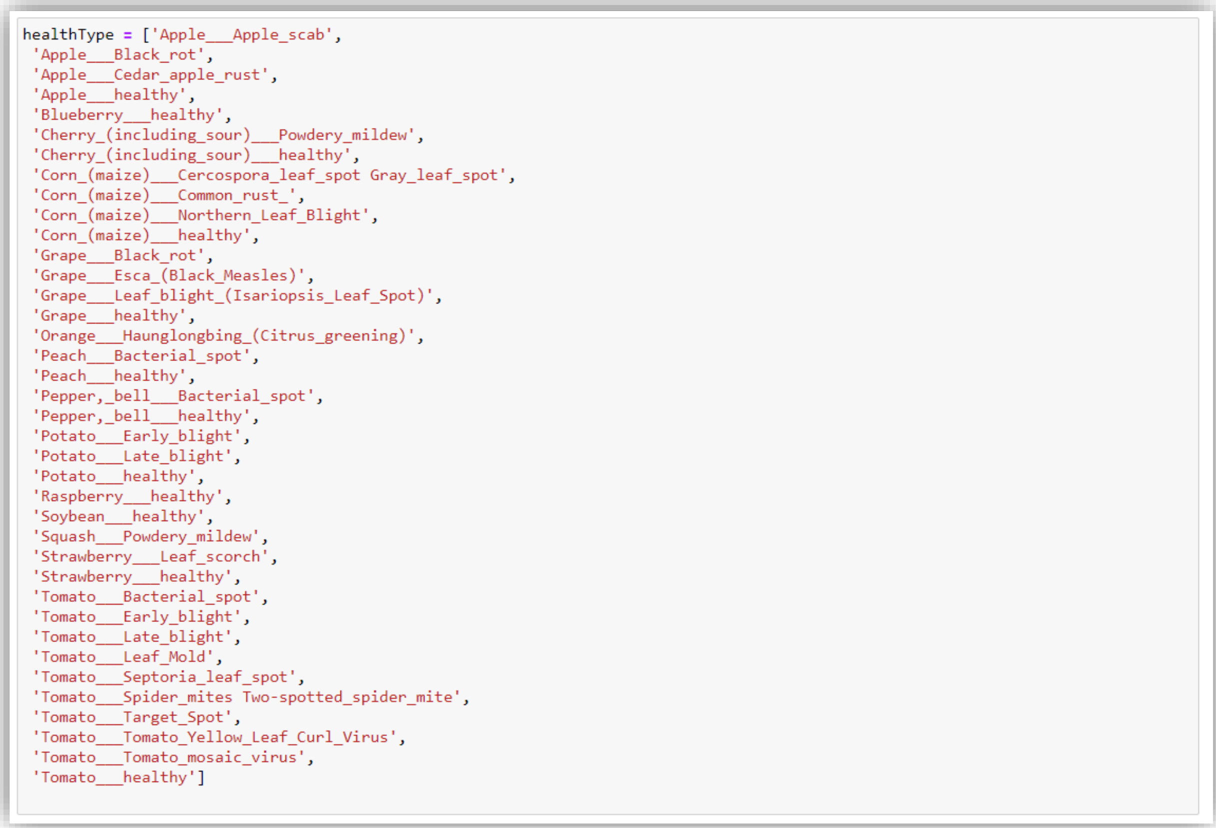
Train data is split to Train and validation in a separate directory named DATA. The Train data is split in the ratio of 8:2 and is saved in new folder DATA. Which is later used for training and validation of the CNN model which will be built.



```python
from tensorflow import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from keras import layers
from tensorflow.keras.applications.xception import preprocess_input
from tensorflow.keras.preprocessing.image import load_img , img_to_array , ImageDataGenerator
import sys
sys.setrecursionlimit(10000)
```

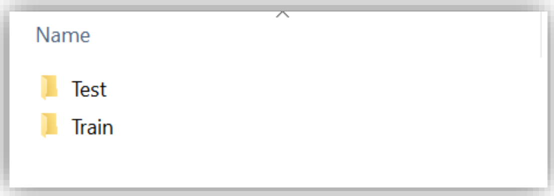
We import KERAS from TENSORFLOW. load_img to load the input image by giving the path. Img_to_array to convert the loaded img to an array of required size. Layers to build our CNN model , which is iported from KERAS. Preprocess_input to preprocess our disease dataset. Finally sys.setrecursionlimit to set the limit for our model to run.

The main class type are labelled first in order to classify the given image.



```
healthType = ['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Blueberry___healthy',
 'Cherry_(including_sour)___Powdery_mildew',
 'Cherry_(including_sour)___healthy',
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 'Corn_(maize)___Common_rust_',
 'Corn_(maize)___Northern_Leaf_Blight',
 'Corn_(maize)___healthy',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Grape___healthy',
 'Orange___Haunglongbing_(Citrus_greening)',
 'Peach___Bacterial_spot',
 'Peach___healthy',
 'Pepper,_bell___Bacterial_spot',
 'Pepper,_bell___healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Raspberry___healthy',
 'Soybean___healthy',
 'Squash___Powdery_mildew',
 'Strawberry___Leaf_scorch',
 'Strawberry___healthy',
 'Tomato___Bacterial_spot',
 'Tomato___Early_blight',
 'Tomato___Late_blight',
 'Tomato___Leaf_Mold',
 'Tomato___Septoria_leaf_spot',
 'Tomato___Spider_mites Two-spotted_spider_mite',
 'Tomato___Target_Spot',
 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 'Tomato___Tomato_mosaic_virus',
 'Tomato___healthy']
```

INPUT :

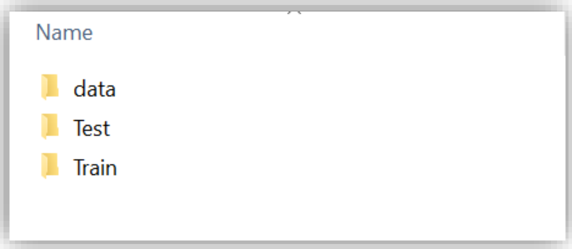


Name

Test

Train

OUTPUT :



Name

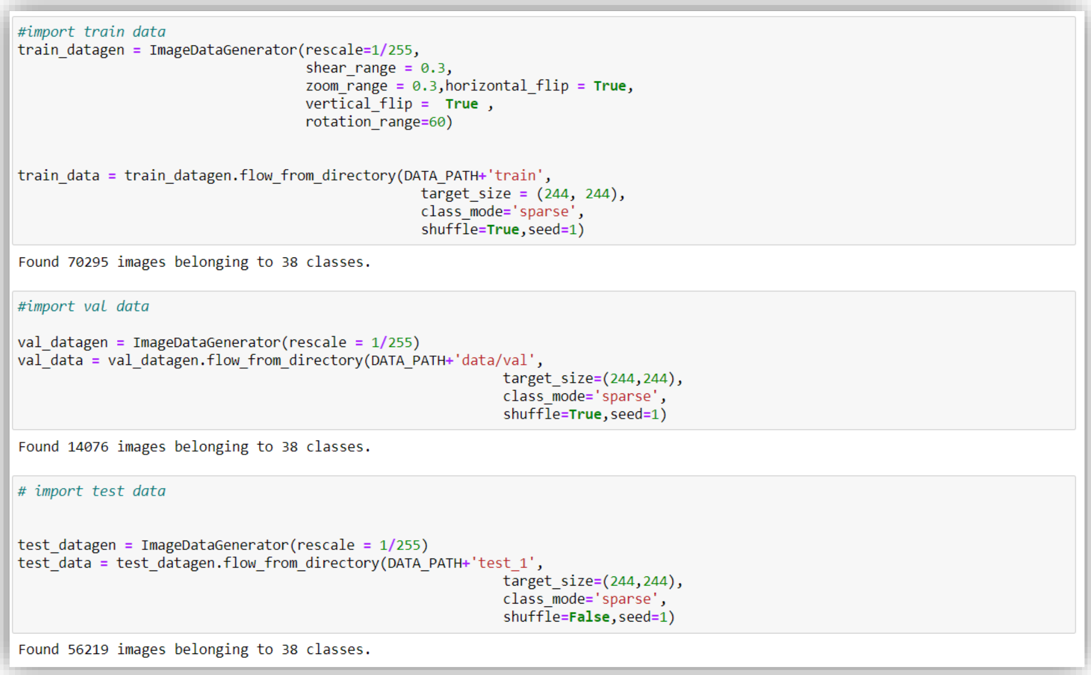
data

Test

Train

Data Pre-processing

RESCALING AND RESIZING



```
#import train data
train_datagen = ImageDataGenerator(rescale=1/255,
                                   shear_range = 0.3,
                                   zoom_range = 0.3,horizontal_flip = True,
                                   vertical_flip =  True ,
                                   rotation_range=60)


train_data = train_datagen.flow_from_directory(DATA_PATH+'train',
                                                 target_size = (244, 244),
                                                 class_mode='sparse',
                                                 shuffle=True,seed=1)
```

```
Found 70295 images belonging to 38 classes.
```

```
#import val data

val_datagen = ImageDataGenerator(rescale = 1/255)
val_data = val_datagen.flow_from_directory(DATA_PATH+'data/val',
                                                           target_size=(244,244),
                                                           class_mode='sparse',
                                                           shuffle=True,seed=1)
```

```
Found 14076 images belonging to 38 classes.
```

```
# import test data


test_datagen = ImageDataGenerator(rescale = 1/255)
test_data = test_datagen.flow_from_directory(DATA_PATH+'test_1',
                                                           target_size=(244,244),
                                                           class_mode='sparse',
                                                           shuffle=False,seed=1)
```

```
Found 56219 images belonging to 38 classes.
```

Train data, validation data and test data are imported, rescaled, resized, shuffled and converted to greyscale which are stored in different labels.

Test Case for Module 5

TEST CASE	**DESCRIPTION**	**DATA INPUT**	**EXPECTED OUTPUT**
T01	Path of the given soil data set is correct	leaf_data with folders Test and Train	Train is copied to Data and is split into Train , Val
T02	Path of the given soil data set is incorrect	No data set folder	Error for not having leaf_data

MODULES 6 IMPLEMENTATION

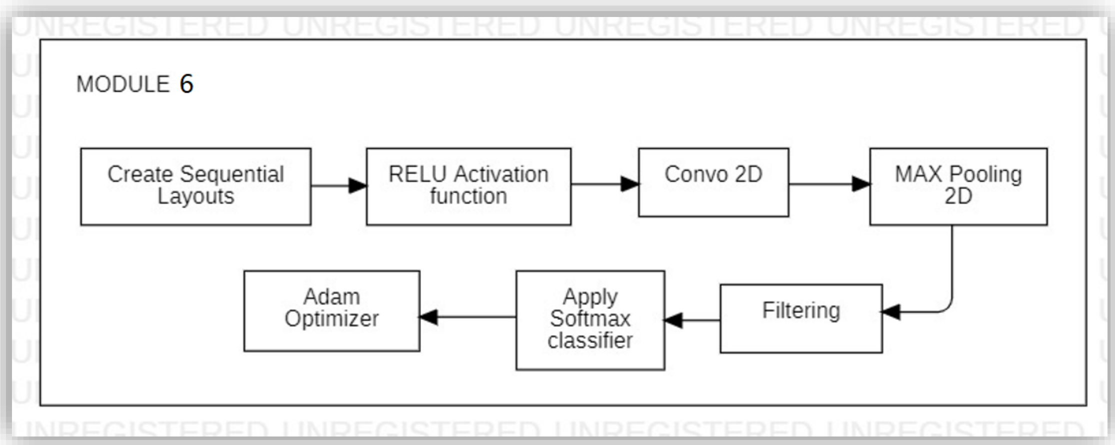
To classify the Soil, we build a CNN (Convolutional Neural Networks) model

A. We use a sequential model. So that, the layers in the network will be added in sequence. We'll use a feed forward network with 6 convolutional layers followed by a fully connected hidden layer. We'll also use dropout layers in between.
B. Dropout regularizes the networks, i.e. it prevents the network from overfitting. All our layers will have relu activations except the output layer. Output layer uses softmax activation as it has to output the probability for each of the classes.
C. Sequential is a keras container for linear stack of layers. Each of the layers in the model needs to know the input shape it should expect, but it is enough to specify input_shape for the first layer of the Sequential model. Rest of the layers do automatic shape inference.
D. To attach a fully connected layer (aka dense layer) to a convolutional layer, we will have to reshape/flatten the output of the conv layer. This is achieved by Flatten layer

INPUT : Parameters for the CNN model

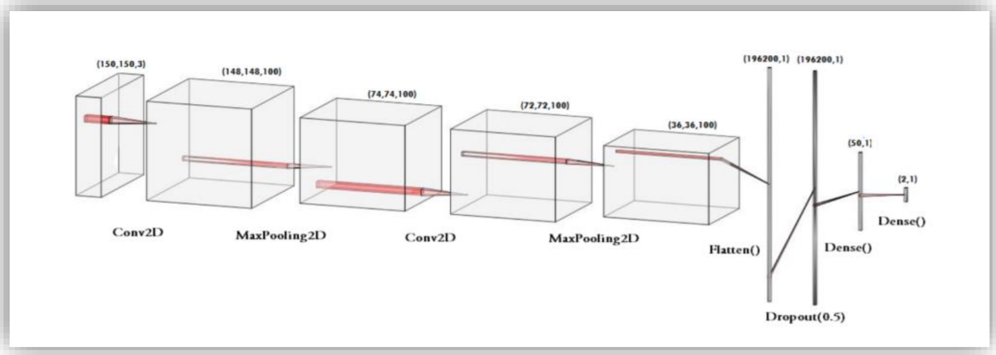
OUTPUT : CNN model.

ARCHITECTURE :

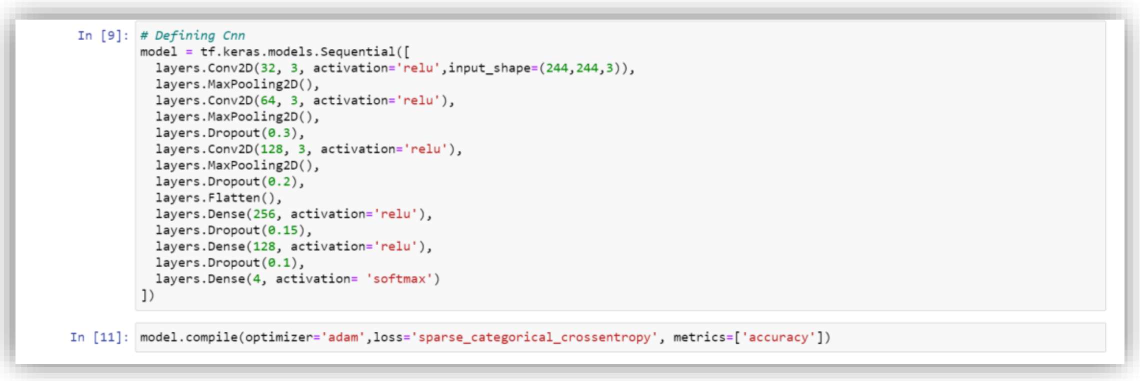


Building CNN model

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.



```
In [9]: # Defining Cnn
        model = tf.keras.models.Sequential([
          layers.Conv2D(32, 3, activation='relu',input_shape=(244,244,3)),
          layers.MaxPooling2D(),
          layers.Conv2D(64, 3, activation='relu'),
          layers.MaxPooling2D(),
          layers.Dropout(0.3),
          layers.Conv2D(128, 3, activation='relu'),
          layers.MaxPooling2D(),
          layers.Dropout(0.2),
          layers.Flatten(),
          layers.Dense(256, activation='relu'),
          layers.Dropout(0.15),
          layers.Dense(128, activation='relu'),
          layers.Dropout(0.1),
          layers.Dense(4, activation= 'softmax')
        ])

In [11]: model.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Here, Feature extraction is done and CNN model by using various layers like convolution, relu, pooling and fully connected in a order. Finally, The marginal loss in model is optimized using adam optimizer.

CNN MODEL and it's PARAMETERS

Layer	**Layer shape details**
Optimizer	Adam
Loss function	Categorical cross_entrophy
Metrics	[accuracy]
Conv2D	64 filter,3 x 3filter size,Relu activation function
Max pooling	2 x 2 kernel size
Dropout	40%
Conv2D	128 filter,3 x 3filter size,Relu activation function
Max pooling	2 x 2 kernel size
Dropout	30%
Dense	64 neurons,Relu
Output	Softmax,4 classes

MODULES 7 IMPLEMENTATION

In this module

A. During the training, our model will iterate over batches of training sets, each of size batch_size. For each batch, gradients will be computed and updates will be made to the weights of the network automatically.

B. One iteration over all the training set is referred to as an epoch. After building the model architecture, we then train the model using model.fit().

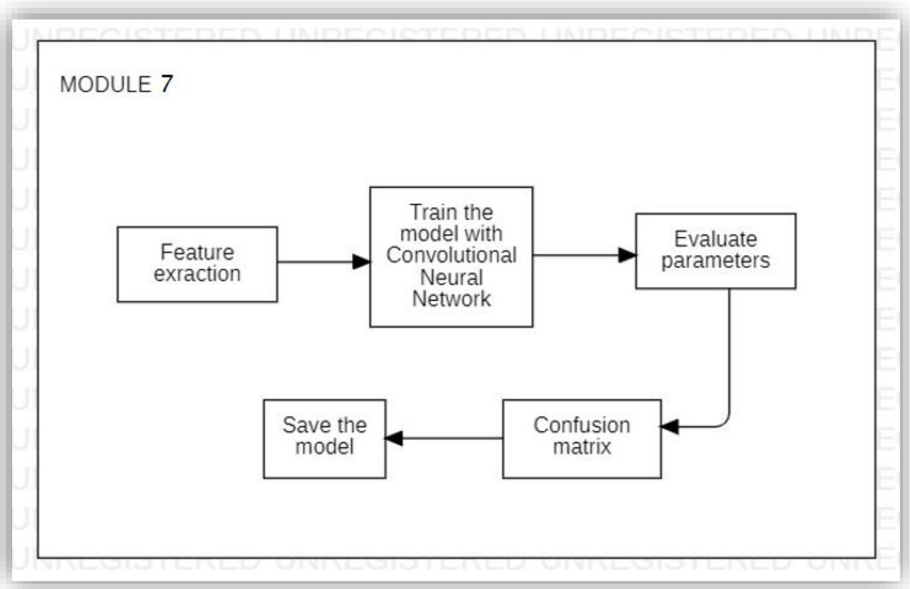
C. The dataset contains a test folder, it has the details related to the image path and their respective class labels. From there, we extract the image path and labels using pandas.

D. Using the confusion matrix, we get the best model with high accuracy.

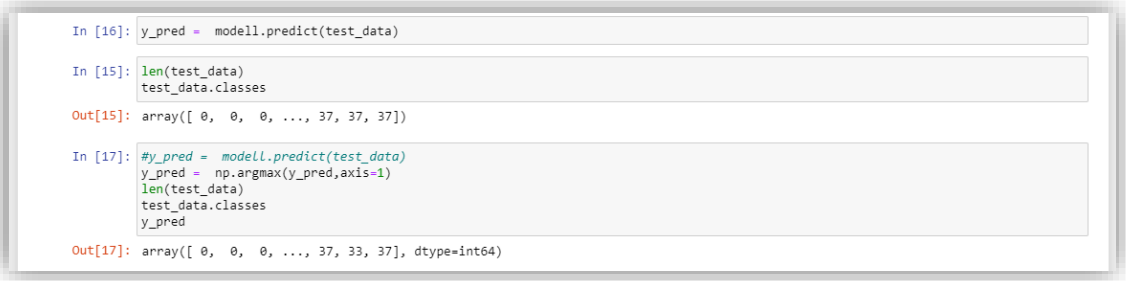
Input : CNN model, Testing and Training data.

Output : Soil Classifier.

ARCHITECTURE :


MODULE 7
Feature exraction
Train the model with Convolutional Neural Network
Evaluate parameters
Confusion matrix
Save the model
UNREGISTERED UNREGISTERED UNREGISTERED

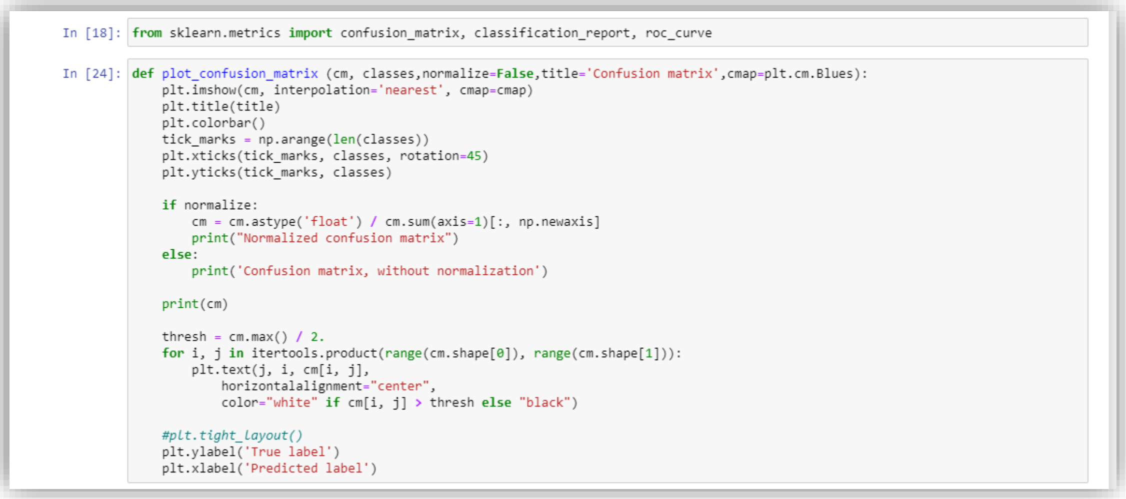
Testing:



```
In [16]: y_pred =  modell.predict(test_data)

In [15]: len(test_data)
         test_data.classes

Out[15]: array([ 0,  0,  0, ..., 37, 37, 37])

In [17]: #y_pred =  modell.predict(test_data)
         y_pred =  np.argmax(y_pred,axis=1)
         len(test_data)
         test_data.classes
         y_pred

Out[17]: array([ 0,  0,  0, ..., 37, 33, 37], dtype=int64)
```

The model is then used to predict test data set and the necessary classification on different classes is obtained.

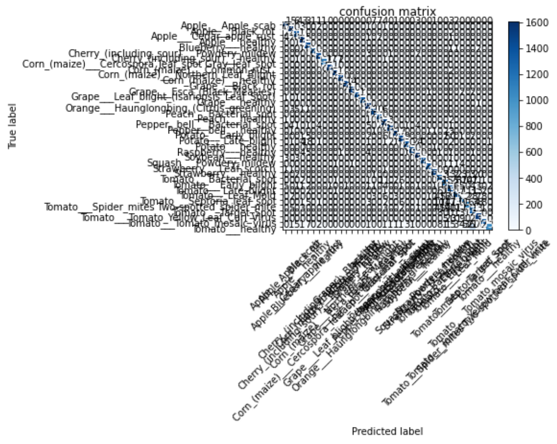
Evaluation with confusion Matrix:



```
In [18]: from sklearn.metrics import confusion_matrix, classification_report, roc_curve

In [24]: def plot_confusion_matrix (cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)

             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             print(cm)

             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

             #plt.tight_layout()
             plt.ylabel('True label')
             plt.xlabel('Predicted label')
```

```
In [25]: import itertools
         cm = confusion_matrix(y_true = test_data.classes, y_pred = y_pred)
         plot_confusion_matrix(cm, healthType, title= 'confusion matrix')
```

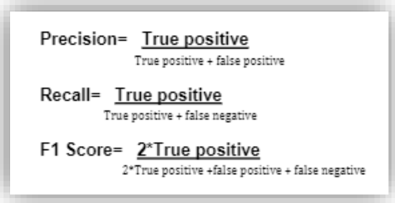
```
Confusion matrix, without normalization
[[1514   15    6 ...    0    0    0]
 [   0 1582    0 ...    0    0    0]
 [   2    1 1402 ...    0    0    0]
 ...
 [   0    0    0 ... 1553    0    0]
 [   0    0    0 ...   11 1354    0]
 [   0    1    5 ...   11    2 1052]]
```



```
In [21]: print(classification_report(test_data.classes, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96      1612
           1       0.98      1.00      0.99      1589
           2       0.98      1.00      0.99      1408
           3       0.96      0.99      0.97      1606
           4       0.96      1.00      0.98      1452
           5       1.00      1.00      1.00      1346
           6       0.99      1.00      0.99      1460
           7       0.99      0.86      0.92      1313
           8       1.00      1.00      1.00      1525
           9       0.89      0.99      0.94      1526
          10       1.00      1.00      1.00      1487
          11       0.96      0.99      0.97      1510
          12       0.99      0.96      0.97      1536
          13       1.00      1.00      1.00      1377
          14       1.00      1.00      1.00      1353
          15       0.98      1.00      0.99      1608
          16       0.99      0.98      0.99      1470
          17       0.99      1.00      0.99      1382
          18       0.92      1.00      0.96      1530
          19       1.00      0.94      0.97      1590
          20       0.99      0.98      0.99      1551
          21       0.96      0.95      0.95      1551
          22       0.99      0.94      0.96      1459
          23       1.00      1.00      1.00      1424
          24       0.98      0.99      0.98      1617
          25       1.00      0.99      1.00      1388
          26       1.00      0.99      1.00      1419
          27       1.00      1.00      1.00      1459
          28       0.90      0.95      0.92      1361
          29       0.91      0.78      0.84      1536
          30       0.73      0.96      0.83      1480
          31       0.92      0.89      0.90      1505
          32       0.91      0.82      0.86      1396
          33       0.92      0.77      0.84      1392
          34       0.67      0.88      0.76      1461
          35       0.92      0.99      0.96      1568
          36       0.98      0.95      0.96      1432
          37       1.00      0.68      0.81      1540

    accuracy                           0.95     56219
   macro avg       0.96      0.95      0.95     56219
weighted avg       0.96      0.95      0.95     56219
```

The function for confusion matrix construction is written. The confusion matrix is constructed with test data set and the model's prediction. Using Confusion matrix, the evaluation parameters like precision, recall, f1-score and support is obtained.



$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{false positive}}$$

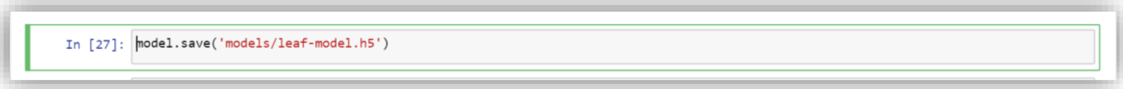
$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{false negative}}$$

$$\text{F1 Score} = \frac{2 * \text{True positive}}{2 * \text{True positive} + \text{false positive} + \text{false negative}}$$

Precision , recall and f1-score : 0.96,0.95,0.95

Macro Average:0.95

Saving model



```
In [27]: model.save('models/leaf-model.h5')
```

The final model is saved in a directory called models, inside the directory in which we are currently working.

MODULES 8 IMPLEMENTATION

In our final module

D. We import necessary modules and label the 38 classification.

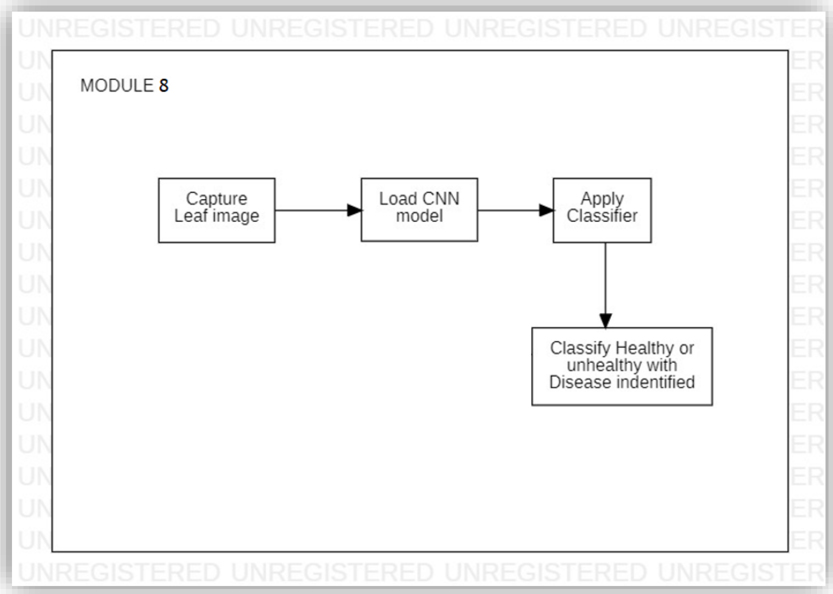
E. The h5 extension model which we saved previously is loaded here.

F. The input image of LEAF is given through path to the model which is loaded and the output is obtained.

INPUT : Uploaded image of Leaf.

OUTPUT : Predicted disease type.

ARCHITECTURE :


UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTER
MODULE 8
Capture Leaf image
Load CNN model
Apply Classifier
Classify Healthy or unhealthy with Disease indentified

```
import splitfolders

splitfolders.ratio("leaf/train", output="leaf/data/", seed=1337, ratio=(.8, .2), group_prefix=None) # default values
```

IMPORTING MODULES

```
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import sys
sys.setrecursionlimit(10000)
```

```
INFO:tensorflow:Enabling eager execution
INFO:tensorflow:Enabling v2 tensorshape
INFO:tensorflow:Enabling resource variables
INFO:tensorflow:Enabling tensor equality
INFO:tensorflow:Enabling control flow v2
```

LABEL THE CLASSIFICATION

```
healthType = ['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Blueberry___healthy',
 'Cherry_(including_sour)___Powdery_mildew',
 'Cherry_(including_sour)___healthy',
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 'Corn_(maize)___Common_rust_',
 'Corn_(maize)___Northern_Leaf_Blight',
 'Corn_(maize)___healthy',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Grape___healthy',
 'Orange___Haunglongbing_(Citrus_greening)',
 'Peach___Bacterial_spot',
 'Peach___healthy',
 'Pepper,_bell___Bacterial_spot',
 'Pepper,_bell___healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Raspberry___healthy',
 'Soybean___healthy',
 'Squash___Powdery_mildew',
 'Strawberry___Leaf_scorch',
 'Strawberry___healthy',
 'Tomato___Bacterial_spot',
 'Tomato___Early_blight',
 'Tomato___Late_blight',
 'Tomato___Leaf_Mold',
 'Tomato___Septoria_leaf_spot',
 'Tomato___Spider_mites Two-spotted_spider_mite',
 'Tomato___Target_Spot',
 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 'Tomato___Tomato_mosaic_virus',
 'Tomato___healthy']
```

LOAD THE SAVED MODEL

```
leaf_model = load_model('models/leaf-model.h5')
```

```
image_path = "leaf/test/0d7d3829-d9be-44c4-8407-ff13b52f5c43___RS_HL 5098_new30degFlipLR.JPG"

image = load_img(image_path,target_size=(224,224))
image = img_to_array(image)
image = image/255
image = np.expand_dims(image,axis=0)

result = np.argmax(leaf_model.predict(image))
print("Classification is :", healthType[result])
```

```
Classification is : Blueberry___healthy
```

```
import sys
sys.getrecursionlimit()
```

```
10000
```

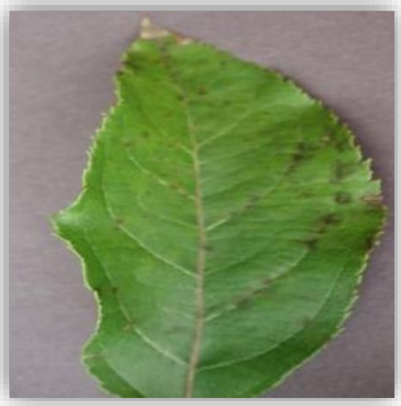
In this, the necessary modules are imported.The class labels are specified. The saved model is loaded.The image path is specified.The image is preprocessed and predicted with the model.Finally, the disease is identified.

TEST CASES for MODULE 8

TEST CASE ID	**TEST CASE OBJECTIVE**	**TEST CASE DESCRIPTION**	**TEST DATA (input)**	**EXPECTED OUTPUT (output)**
T_01	Build a CNN model to classify the LEAF from the image uploaded	Identify Disease : User accesses it after uploading a clear image of plant leaf	Clear leaf image uploaded	The built CNN model will identify any diseases if present based on the input.
T_02	Build a CNN model (result may not be accurate if image uploaded is not clear)	Identify Disease : User accesses it after uploading a unclear/blurry image of plant leaf	Unclear leaf image uploaded	The disease may not be identified correctly.
T_03	Build a CNN model (result may not be accurate if image uploaded is a bunch)	Identify Disease : User accesses it after uploading image with bunch of leaves	Bunch of leaves	The disease may not be identified correctly.

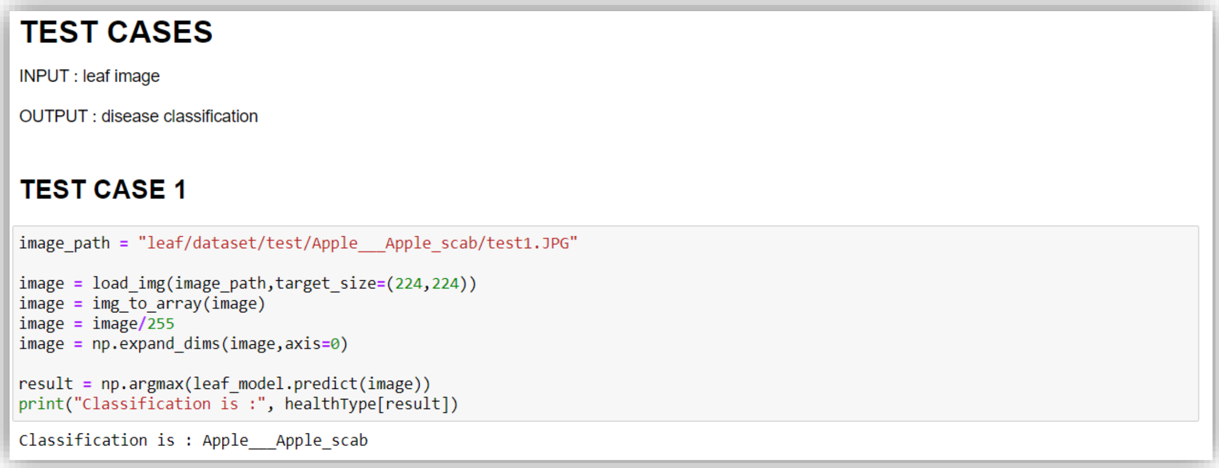
TEST CASE 1 :

INPUT :



In this test case we pass a clear image of a leaf disease , apple scab , taken from our data set. When this is passed to our model we expect a precise output because there are no added filters and image is clear.

OUTPUT :



TEST CASES

INPUT : leaf image

OUTPUT : disease classification

TEST CASE 1

```
image_path = "leaf/dataset/test/Apple___Apple_scab/test1.JPG"

image = load_img(image_path,target_size=(224,224))
image = img_to_array(image)
image = image/255
image = np.expand_dims(image,axis=0)

result = np.argmax(leaf_model.predict(image))
print("Classification is :", healthType[result])
```

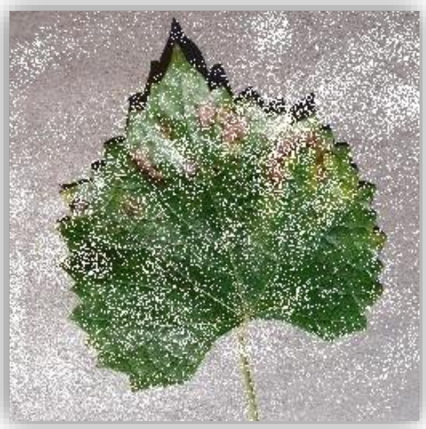
```
Classification is : Apple___Apple_scab
```

INFERENCE :

A clear leaf image without any filters is given as input to the model. The trained model classifies it correctly as Apple Scab. Hence when a clear image is given as input , a precise output is obtained from the model.

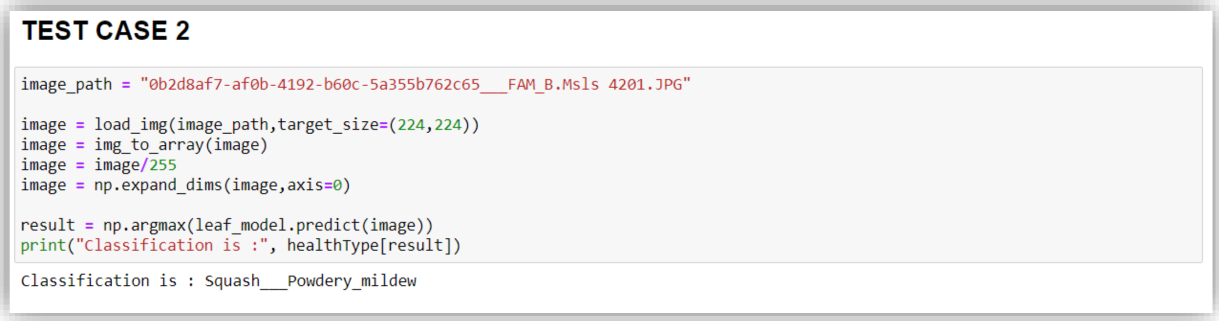
TEST CASE 2 :

INPUT :



This is an example for Late Blight disease. We have added an extra layer of white spots. When this unclear image is passed to our model it may mis-classify the disease because of the un clear image.

OUTPUT :



TEST CASE 2

```
image_path = "0b2d8af7-af0b-4192-b60c-5a355b762c65___FAM_B.Msls 4201.JPG"

image = load_img(image_path,target_size=(224,224))
image = img_to_array(image)
image = image/255
image = np.expand_dims(image,axis=0)

result = np.argmax(leaf_model.predict(image))
print("Classification is :", healthType[result])
```

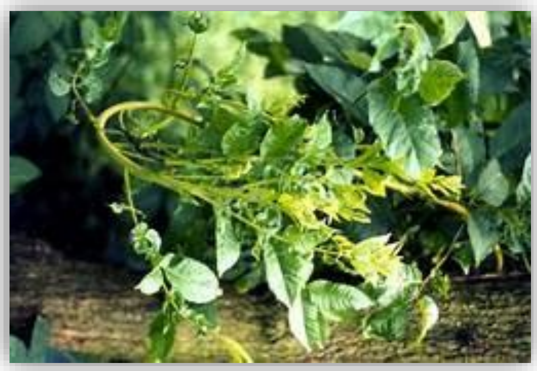
```
Classification is : Squash___Powdery_mildew
```

INFERENCE :

A leaf image with added filter is given. When the trained model takes this as input , it mis-classifies it as Powdery mildew whereas it's actually is Late blight. Thus when a unclear image is fed into model , misclassification occurs.

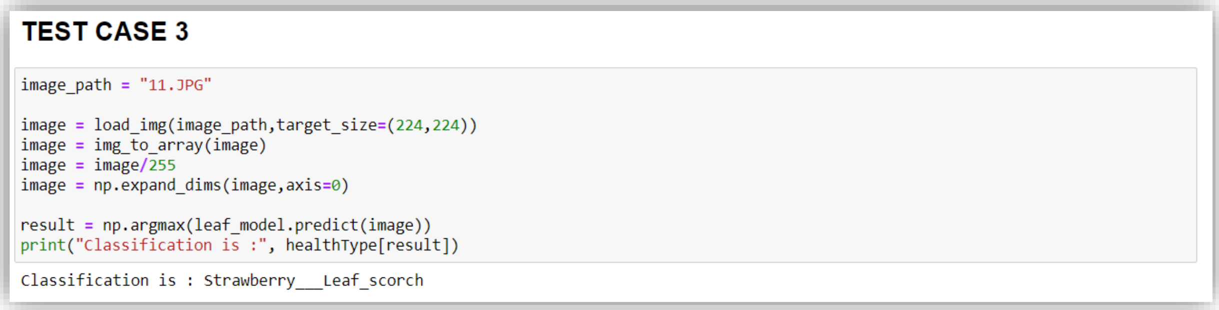
TEST CASE 3 :

INPUT :



Our final test case deals with bunch of diseased leaves. This image is a strawberry – leaf scorch. There are no added filters. When this passed to the model , a precise output is expected.

OUTPUT :



TEST CASE 3

```
image_path = "11.JPG"

image = load_img(image_path,target_size=(224,224))
image = img_to_array(image)
image = image/255
image = np.expand_dims(image,axis=0)

result = np.argmax(leaf_model.predict(image))
print("Classification is :", healthType[result])
```

```
Classification is : Strawberry___Leaf_scorch
```

INFERENCE :

In this case , we tried giving a group of leaves to test whether our model can classify it precisely. The given image is Leaf Scorch of plant Strawberry. The out put from the model is also the same. In this case model has classified it correctly. But chances for misclassification is high when there is added layer or un clear image is uploaded.

IMPLEMENTATION WITH GNN (ALTERNATE ALGORITHM)

ALGORITHM USED : GNN

Graphs have tremendous expressive powers and are therefore gaining a lot of attention in the field of machine learning. Every node has an embedding associated with it that defines the node in the data space. Graph neural networks refer to the neural network architectures that operate on a graph. The aim of a GNN is for each node in the graph to learn an embedding containing information about its neighbourhood (nodes directly connected to the target node via edges). This embedding can then be used for different problems like node labelling, node prediction, edge prediction, etc.

Need for Graph Neural Networks

The need for graph neural networks arose from the fact that a lot of data available to us is in an unstructured format. Unstructured data is data that has not been processed or does not have a pre-defined format which makes it difficult to analyse. To make sense of this data and to derive inferences from it, we need a structure that defines a relationship between these unstructured data points. The existing machine learning architectures and algorithms do not seem to perform well with these kinds of data. The primary advantages of graph neural networks are:

- The graph data structure has proven tremendously successful in the field of computer science while working with unstructured data.

- Graphs are helpful in defining concepts which are abstract, like relationships between entities. Since each node in the graph is defined by its connections and neighbours, graph neural networks can capture the relationships between nodes in an efficient manner.

Thus, developing GNNs for handling data like social network data, which is highly unstructured, is an exciting amalgamation of graphs and machine learning which holds a lot of potential.

SOIL CLASSIFICATION WITH GNN

We have divided images into 4 folders so that each folder represents a different soil type. The shape of data is (n,x,y,z) which means that there are n images of size x*y pixels and z means the data contains colored images. With the split folders package, we use the ratio() function for splitting data arrays into subsets (for training data and for validating data and using ImageDataGenerator() method. We then start to augment and normalize the images to reduce noise , resize and grey scale them.

CODE :

IMPORTING MODULES

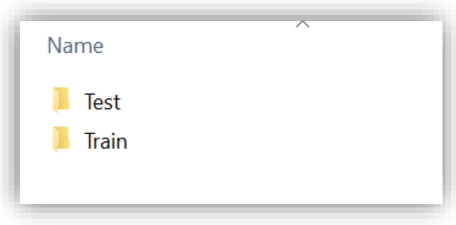
```
import splitfolders
```

```
splitfolders.ratio("Soil_Dataset/Train", output="Soil_Dataset/data/", seed=1337, ratio=(.8, .2), group_prefix=None)
```

```
from tensorflow import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

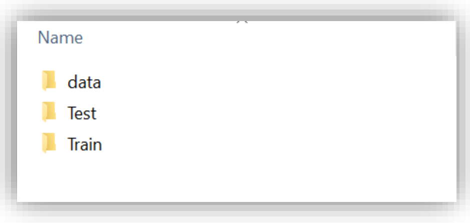
```
from keras import layers
from tensorflow.keras.applications.xception import preprocess_input
from tensorflow.keras.preprocessing.image import load_img , img_to_array , ImageDataGenerator
```

INPUT :



```
Name
Test
Train
```

OUTPUT :



```
Name
data
Test
Train
```

We label the classification of the model :

CLASS LABELS

```
SoilType = ['Alluvial_Soil', 'Black_Soil', 'Clay_Soil', 'Red_Soil']

DATA_PATH = 'Soil_Dataset/'
```

IMAGE PRE PROCESSING

```
#import train data
train_datagen = ImageDataGenerator(rescale=1/255,
                                   shear_range = 0.3,
                                   zoom_range = 0.3,horizontal_flip = True,
                                   vertical_flip =  True ,
                                   rotation_range=60)


train_data = train_datagen.flow_from_directory(DATA_PATH+'train',
                                                 target_size = (244, 244),
                                                 class_mode='sparse',
                                                 shuffle=True,seed=1)
```

```
Found 715 images belonging to 4 classes.
```

```
#import val data

val_datagen = ImageDataGenerator(rescale = 1/255)
val_data = val_datagen.flow_from_directory(DATA_PATH+'data/val',
                                                     target_size=(244,244),
                                                     class_mode='sparse',
                                                     shuffle=True,seed=1)
```

```
Found 144 images belonging to 4 classes.
```

```
# import test data


test_datagen = ImageDataGenerator(rescale = 1/255)
test_data = test_datagen.flow_from_directory(DATA_PATH+'Test',
                                                     target_size=(244,244),
                                                     class_mode='sparse',
                                                     shuffle=False,seed=1)
```

```
Found 188 images belonging to 4 classes.
```

Train data, validation data and test data are imported, rescaled, resized, shuffled and converted to greyscale which are stored in different labels.

BUILDING GNN MODEL

```
# Defining Gnn

class GNN:

    def build(
        self,
        features1,
        features2,
        features3,
        face_feature_size,
        attention_feature_size,
        scene_feature_size,
        hidden_size,
        num_classes,
        num_steps,
        num_face_nodes,
        num_attention_nodes,
        edge_features_length,
        use_bias,
        keep_prob=0.5,
        layer_num=1
    ):

        #Add an extract fully connected layer to shrink the size of features
        self.face_weights = tf.Variable(glorot_init([face_feature_size, hidden_size]), name='face_weights')
        self.face_biases = tf.Variable(np.zeros([hidden_size]).astype(np.float32), name='face_biases')
        self.attention_weights = tf.Variable(glorot_init([attention_feature_size, hidden_size]), name='attention_weights')
        self.attention_biases = tf.Variable(np.zeros([hidden_size]).astype(np.float32), name='attention_biases')
        self.scene_weights = tf.Variable(glorot_init([scene_feature_size, hidden_size]), name='scene_weights')
        self.scene_biases = tf.Variable(np.zeros([hidden_size]).astype(np.float32), name='scene_biases')
        self.face_features = tf.nn.relu(tf.nn.bias_add(tf.matmul(features1, self.face_weights),self.face_biases))
        self.attention_features = tf.nn.relu(tf.nn.bias_add(tf.matmul(features2, self.attention_weights), self.attention_biases)
        self.scene_features = tf.nn.relu(tf.nn.bias_add(tf.matmul(features3, self.scene_weights), self.scene_biases))

        #define LSTM
        with tf.variable_scope("lstm_scope"):
            self.cell = tf.contrib.rnn.GRUCell(hidden_size)
            self.cell = tf.contrib.rnn.DropoutWrapper(self.cell, output_keep_prob=keep_prob)
            #self.mlstm_cell = tf.contrib.rnn.MultiRNNCell([self.cell for _ in range(layer_num)])
        #define edge weights, edge bias, and mask used to take average of edge features.
        self.face_edge_weights = tf.Variable(glorot_init([hidden_size, edge_features_length]), name='face_edge_weights')
        self.face_edge_biases = tf.Variable(np.zeros([edge_features_length]).astype(np.float32),name='face_edge_biases')
        self.attention_edge_weights = tf.Variable(glorot_init([hidden_size, edge_features_length]), name='attention_edge_weights'
        self.attention_edge_biases = tf.Variable(np.zeros([edge_features_length]).astype(np.float32),name='attention_edge_biases'
        self.scene_edge_weights = tf.Variable(glorot_init([hidden_size, edge_features_length]), name='scene_edge_weights')
        self.scene_edge_biases = tf.Variable(np.zeros([edge_features_length]).astype(np.float32),name='scene_edge_biases')

        with tf.variable_scope("lstm_scope") as scope:
            mask = tf.ones(
                       [num_face_nodes+num_attention_nodes + 1, num_face_nodes+num_attention_nodes + 1]
                    ) - tf.diag(tf.ones([num_face_nodes+num_attention_nodes + 1]))
            for step in range(num_steps):
                if step>0:
                    tf.get_variable_scope().reuse_variables()
                else:
                    self.state = tf.cond(
                                    tf.equal(num_face_nodes,0),
                                    lambda:tf.concat([self.attention_features, self.scene_features], axis=0),
                                    lambda:tf.concat([self.face_features, self.attention_features,self.scene_features], axis=0)
                                )

                m_face = tf.matmul(self.state[:num_face_nodes], tf.nn.dropout(self.face_edge_weights, keep_prob=keep_prob))
                m_attention = tf.matmul(
                                  self.state[num_face_nodes:num_face_nodes+num_attention_nodes],
                                  tf.nn.dropout(self.attention_edge_weights, keep_prob=keep_prob)
                              )
                m_scene = tf.matmul(
                              self.state[num_face_nodes+num_attention_nodes:],
                              tf.nn.dropout(self.scene_edge_weights, keep_prob=keep_prob)
                          )

                if use_bias:
                    m_face = tf.nn.bias_add(m_face, self.face_edge_biases)
                    m_face = tf.nn.bias_add(m_attention, self.attention_edge_biases)
                    m_scene = tf.nn.bias_add(m_scene, self.scene_edge_biases)
                m_combine = tf.concat([m_face, m_attention, m_scene], axis=0)
                acts = tf.multiply(tf.matmul(mask, m_combine), 1/(tf.cast(num_face_nodes+num_attention_nodes+1, tf.float32)-1))
                self.rnnoutput, self.state = self.cell(acts, self.state)


        with tf.variable_scope('softmax'):
            W = tf.get_variable('W', [hidden_size, num_classes])
            b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
        self.logits = tf.matmul(self.rnnoutput, W) + b
        self.probs = tf.nn.softmax(self.logits)
        self.data_dict = None

model = GNN()
```

```
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=5)
```

During the training, our model will iterate over batches of training sets, each of size batch_size. For each batch, gradients will be computed and updates will be made to the weights of the network automatically. One iteration over all the training set is referred to as an epoch. After building the model architecture, we then train the model using model.fit(). The dataset contains a test folder, it has the details related to the image path and their respective class labels. From there, we extract the image path and labels using pandas. Using the confusion matrix, we get the best model with high accuracy.

Here, In this section the model is trained with train data set and is validated with validation set. The epochs and batch size are set to be 100 and 32. The model will be trained and validated continuously within given epochs till stable accuracy and loss is attained which is done using callback attribute.

TRAIN , TEST AND EVALUATE MODEL

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(train_data, validation_data= val_data, batch_size=32, epochs = 100, callbacks=[early])
history = model.fit(train_data, validation_data= val_data, batch_size=32, epochs = 100, callbacks=[early])
```

```
Epoch 1/100
23/23 [==============================] - 46s 2s/step - loss: 2.0065 - accuracy: 0.4392 - val_loss: 0.6839 - val_accuracy: 0.826
4
Epoch 2/100
23/23 [==============================] - 44s 2s/step - loss: 0.5767 - accuracy: 0.7776 - val_loss: 0.5591 - val_accuracy: 0.722
2
Epoch 3/100
23/23 [==============================] - 44s 2s/step - loss: 0.4502 - accuracy: 0.8000 - val_loss: 0.2980 - val_accuracy: 0.868
1
Epoch 4/100
23/23 [==============================] - 72s 3s/step - loss: 0.3079 - accuracy: 0.8699 - val_loss: 0.3364 - val_accuracy: 0.826
4
Epoch 5/100
23/23 [==============================] - 81s 4s/step - loss: 0.3403 - accuracy: 0.8615 - val_loss: 0.2182 - val_accuracy: 0.909
7
Epoch 6/100
23/23 [==============================] - 70s 3s/step - loss: 0.2779 - accuracy: 0.8923 - val_loss: 0.2085 - val_accuracy: 0.930
6
Epoch 7/100
23/23 [==============================] - 88s 4s/step - loss: 0.2965 - accuracy: 0.8867 - val_loss: 0.1775 - val_accuracy: 0.944
4
Epoch 8/100
23/23 [==============================] - 86s 4s/step - loss: 0.2611 - accuracy: 0.9063 - val_loss: 0.3432 - val_accuracy: 0.881
9
Epoch 9/100
23/23 [==============================] - 71s 3s/step - loss: 0.3116 - accuracy: 0.8671 - val_loss: 0.2640 - val_accuracy: 0.868
1
Epoch 10/100
23/23 [==============================] - 78s 3s/step - loss: 0.2752 - accuracy: 0.8965 - val_loss: 0.2100 - val_accuracy: 0.916
7
Epoch 11/100
23/23 [==============================] - 89s 4s/step - loss: 0.2239 - accuracy: 0.9119 - val_loss: 0.1390 - val_accuracy: 0.944
4
Epoch 12/100
23/23 [==============================] - 83s 4s/step - loss: 0.2183 - accuracy: 0.9287 - val_loss: 0.1539 - val_accuracy: 0.944
4
Epoch 13/100
23/23 [==============================] - 82s 4s/step - loss: 0.2526 - accuracy: 0.9259 - val_loss: 0.2948 - val_accuracy: 0.902
8
Epoch 14/100
23/23 [==============================] - 85s 4s/step - loss: 0.2052 - accuracy: 0.9203 - val_loss: 0.2476 - val_accuracy: 0.909
7
Epoch 15/100
23/23 [==============================] - 82s 4s/step - loss: 0.2466 - accuracy: 0.9133 - val_loss: 0.3020 - val_accuracy: 0.854
2
Epoch 16/100
23/23 [==============================] - 88s 4s/step - loss: 0.2624 - accuracy: 0.8951 - val_loss: 0.2533 - val_accuracy: 0.930
6
```

```
model.evaluate(test_data)
```

```
6/6 [==============================] - 5s 831ms/step - loss: 0.4160 - accuracy: 0.8617

[0.415997713804245, 0.8617021441459656]
```

The model is tested with test data and we get an accuracy of 86%. The model is then used to predict test data set and the necessary classification on different classes is obtained.

```
y_pred =  model.predict(test_data)
y_pred =  np.argmax(y_pred,axis=1)
len(test_data)
test_data.classes
y_pred
```

```
array([0, 2, 0, 2, 0, 0, 0, 0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2,
       2, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0,
       2, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3], dtype=int64)
```

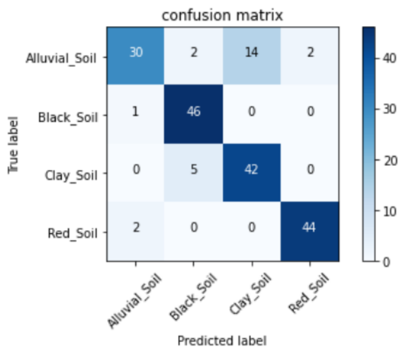
CONFUSION MATRIX AND CLASSIFICATION REPORT

```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve
```

```
def plot_confusion_matrix (cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
import itertools
cm = confusion_matrix(y_true = test_data.classes, y_pred = y_pred)
plot_confusion_matrix(cm, SoilType, title= 'confusion matrix')
```

```
Confusion matrix, without normalization
[[30  2 14  2]
 [ 1 46  0  0]
 [ 0  5 42  0]
 [ 2  0  0 44]]
```



The function for confusion matrix construction is written. The confusion matrix is constructed with test data set and the model's prediction. Using Confusion matrix, the evaluation parameters like precision, recall, f1-score and support is obtained.

```
print(classification_report(test_data.classes, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.62      0.74        48
           1       0.87      0.98      0.92        47
           2       0.75      0.89      0.82        47
           3       0.96      0.96      0.96        46

    accuracy                           0.86       188
   macro avg       0.87      0.86      0.86       188
weighted avg       0.87      0.86      0.86       188
```

SAVE THE GNN MODEL

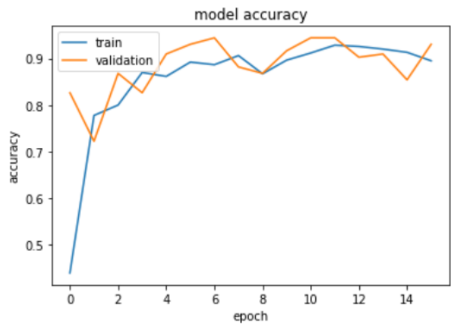
```
model.save('models/soil_model_GNN.h5')
```

The final model is saved in a directory called models, inside the directory in which we are currently working.

ACCURACY vs EPOCH

```
def plot_hist(hist):
    plt.plot(hist.history["accuracy"])
    plt.plot(hist.history["val_accuracy"])
    plt.title("model accuracy")
    plt.ylabel("accuracy")
    plt.xlabel("epoch")
    plt.legend(["train", "validation"], loc="upper left")
    plt.show()
```

```
plot_hist(history)
```

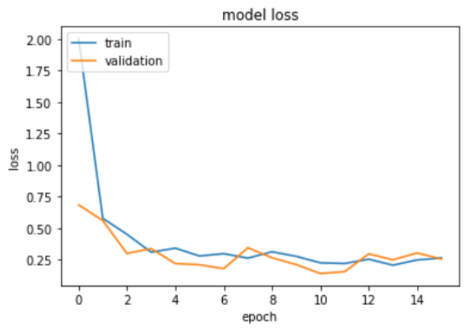


A graph between accuracy and epoch is plotted.The accuracy of the model raises and remains almost unchanged after particular epoch. The accuracy of the model during training and validating comes out to be 86% and 91%.

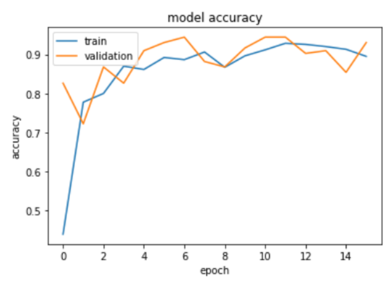
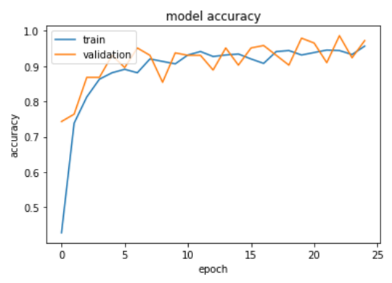
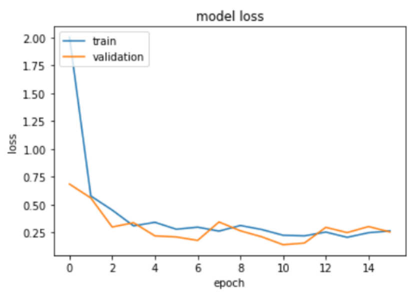
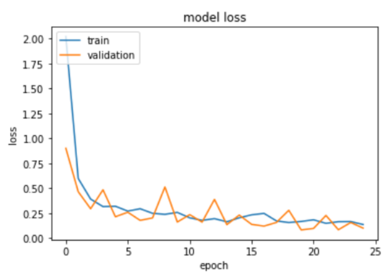
LOSS vs EPOCH

```
def plot_hist_loss(hist):
    plt.plot(hist.history["loss"])
    plt.plot(hist.history["val_loss"])
    plt.title("model loss")
    plt.ylabel("loss")
    plt.xlabel("epoch")
    plt.legend(["train", "validation"], loc="upper left")
    plt.show()
```

```
plot_hist_loss(history)
```



COMPARISION BETWEEN GNN and CNN MODEL

	GNN	**CNN**
TRAIN PERCENTAGE	80%	80%
VALIDATION PERCENTAGE	20%	20%
EPOCH (EARLY CALLBACK)	16	25
VALIDATION ACCURACY	93%	97%
ACCURACY	86%	91%
ACCURACY vs EPOCH		
LOSS vs EPOCH		

CONTRIBUTION BY TEAM MEMBERS

DESCRIPTION

Ashwath (MODULE : 1 , 2 , 5 , 6)

1. The dataset contains 2 main folders , Train and Test. Outcome of first module is to Split Train into Train and Val in the ratio of 8:2 and save it in another folder named Data. Splitfolders module is imported. Source and Target paths are set and the folder Train is passed. After applying the split we are able to see new folder DATA which has Train and Val.

2. The Deep Learning technique used here is CNN. We have built 3 Conv2D layers with each 32 , 64 , 128 features and input shape of 244x244x3. Activation function used here is ReLu function. Two Dropout layers with probability of 0.3 , 0.2 , 0.15 and 0.1 is added. Finally 3 dense layers where former 2 with ReLu activation function and last layer with Softmax is used for getting Classification.

Aakash (MODULE : 3 , 4 , 7 , 8)

3. The CNN model which is built , is tested under 3 test cases : 25 epoch , 50 epoch and 75 epoch. We are using the model which is trained for 25 epoch because 50 and 75 may lead to OVERFITTING of the model. After running the model for 25 epoch , 2 graphs : Accuracy vs Epoch and Loss vs Epoch is made to keep track of our model. We evaluate with test data and generate confusion matrix with test data. Finally we print the Classification Report of our model. We save the trained model in **.h5** extension so that we can load it later and use it.

4. Required modules are imported and soil types based on which we are going to labeled are declared. i.e "Alluvial Soil", "Red Soil", "Clay Soil" and "Black Soil" and in case of plant dataset we have 38 classifications. The model which we saved previously is loaded here using load_model. Input image is fed as path and we convert it into array as we need to pass it to model. Finally the converted image is passed to the model and final output label is obtained which is CLASSIFICATION OF SOIL. We have tested it for 4 different test cases for soil and 3 test cases for plant disease and the outcomes with inference are added.