```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/content/gdrive/MyDrive/data'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/content/gdrive/MyDrive/data) that gets preserved as output when you create a versio
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/content/gdrive/MyDrive/data/Healthy/Corn_Health (673).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (653).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (660).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (669).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (676).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (686).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (685).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (648).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (662).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (677).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (654).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (65).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (650).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (674).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (644).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (661).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (651).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (672).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (665).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (663).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (649).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (678).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (68).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (668).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (659).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (67).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (652).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (657).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (671).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (646).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (683).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (666).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (66).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (658).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (664).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (670).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (7).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (70).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (706).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (711).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (731).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (724).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (73).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (71).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (703).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (701).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (720).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (710).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (695).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (725).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (722).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (719).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (726).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (702).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (705).JPG
/content/gdrive/MyDrive/data/Healthy/Corn_Health (708).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (721).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (712).jpg
/content/gdrive/MyDrive/data/Healthy/Corn_Health (715).jpg
```

Importing necessary Libraries

```
import os
import numpy as np
import pandas as pd
import tqdm as tqdm
import cv2 as cv
import keras
from keras.models import Sequential
from keras.layers import Conv2D , MaxPooling2D ,GlobalAveragePooling2D ,Flatten , Dense , Dropout , BatchNormalization
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
tf.keras.applications.VGG16
from keras.applications.vgg16 import VGG16
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.models import Model
```

Data Preprocessing

converting categorical labels to numeric codes

```
encoder = OneHotEncoder()     #LabelEncoder encode labels with value between 0 and n_classes-1
encoder.fit([[0],[1],[2],[3]])
```

```
▾ OneHotEncoder
OneHotEncoder()
```

Loading the data

Resizing the image based on input dimension required for the model

```
input_path = []
label = []
```

```
data = []
paths = []
result = []
fpath = []
for r,d,f in os.walk(r"/content/gdrive/MyDrive/data/Blight"):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r,file))
for path in paths:
    img = Image.open(path)
    img = img.resize((224,224))
    img = np.array(img)
    if(img.shape == (224,224,3)):
        data.append(img)
        label.append(0)
        input_path.append(os.path.join("Blight","Blight",path))
        result.append(encoder.transform([[0]]).toarray())
print(len(paths))
```

```
275
```

```
paths = []
for r,d,f in os.walk(r"/content/gdrive/MyDrive/data/Common_Rust"):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r,file))
for path in paths:
    img = Image.open(path)
    img = img.resize((224,224))
    img = np.array(img)
    if(img.shape == (224,224,3)):
        data.append(img)
        label.append(1)
        input_path.append(os.path.join("Common_Rust","Common_Rust",path))
        result.append(encoder.transform([[1]]).toarray())
print(len(paths))
```

```
115
```

```
paths = []
for r,d,f in os.walk(r"/content/gdrive/MyDrive/data/Gray_Leaf_Spot"):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r,file))
for path in paths:
    img = Image.open(path)
    img = img.resize((224,224))
    img = np.array(img)
    if(img.shape == (224,224,3)):
        data.append(img)
        label.append(2)
        input_path.append(os.path.join("Gray_Leaf_Spot","Gray_Leaf_Spot",path))
        result.append(encoder.transform([[2]]).toarray())
print(len(paths))
print(len(input_path))
```

```
130
515
```

```
paths = []
for r,d,f in os.walk(r"/content/gdrive/MyDrive/data/data/Healthy"):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r,file))
for path in paths:
    img = Image.open(path)
    img = img.resize((224,224))
    img = np.array(img)
    if(img.shape == (224,224,3)):
        data.append(img)
        label.append(3)
        input_path.append(os.path.join("Healthy","Healthy",path))
        result.append(encoder.transform([[3]]).toarray())
print(len(paths))
```

```
0
```

```
df = pd.DataFrame()
df['images'] = input_path
df['label'] = label
df=df.sample(frac=1).reset_index(drop=True)
df.head()
```

	images	label
0	/content/gdrive/MyDrive/data/Common_Rust/Corn_...	1
1	/content/gdrive/MyDrive/data/Common_Rust/Corn_...	1
2	/content/gdrive/MyDrive/data/Blight/Corn_Bligh...	0
3	/content/gdrive/MyDrive/data/Common_Rust/Corn_...	1
4	/content/gdrive/MyDrive/data/Blight/Corn_Bligh...	0

```
pd.unique(df['label'])
```

```
array([1, 0, 2])
```

```
import seaborn as sns
sns.countplot(df['label'])
```

```
<Axes: ylabel='count'>
```


```
print("Total no.of images are :",len(result))
```

```
Total no.of images are : 515
```

```
len(data)
```

```
515
```


```
len(label)
```

```
515
```

```
result = np.array(result)
result = result.reshape(-1,4)
```

```
result.shape
```

```
(515, 4)
```

```
data = np.array(data)
data.shape
```

```
(515, 224, 224, 3)
```

Displaying random images

```
%matplotlib inline
plt.figure(figsize=(10, 10))
for i in range(6):
    plt.subplot(1, 6, i+1)
    plt.imshow(data[i], cmap="gray")
    plt.axis('off')
plt.show()
```

Splitting data into training and testing data

```
x_train , x_test , y_train , y_test = train_test_split(data , result , test_size = 0.25 , shuffle = True , random_state = 100)
```

```
print("Shape of an image in x_train : ",x_train[0].shape)
print("Shape of an image in x_test : ",x_test[0].shape)
```

```
Shape of an image in x_train :  (224, 224, 3)
Shape of an image in x_test :  (224, 224, 3)
```

```
x_train = np.array(x_train)
x_test = np.array(x_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

```
print("x_train Shape : ", x_train.shape)
print("x_test Shape : ", x_test.shape)
print("y_train Shape: ", y_train.shape)
print("y_test Shape: ", y_test.shape)
```

```
x_train Shape :  (386, 224, 224, 3)
x_test Shape :  (129, 224, 224, 3)
y_train Shape:  (386, 4)
y_test Shape:  (129, 4)
```

Pre-Trained VGG16 model

```
modelVGG = VGG16(include_top = False,weights = 'imagenet', classifier_activation = 'Softmax',input_shape = (224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n
58889256/58889256 [==============================] - 0s 0us/step
```

```
for layer in modelVGG.layers:
    layer.trainable = False
```

```
num_classes = 4
for (i,layer) in enumerate(modelVGG.layers):
    print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
    def lw(bottom_model,num_classes):
        top_model = bottom_model.output
        top_model = GlobalAveragePooling2D()(top_model)
        top_model = Dense(4096,activation='relu')(top_model)
        top_model = Dropout(0.625, name='dropout_1')(top_model)
        top_model = Dense(4096,activation='relu')(top_model)
        #top_model = Dropout(0.625, name='dropout_2')(top_model)
        top_model = Dense(512,activation='relu')(top_model)
        top_model = Dense(4,activation='softmax')(top_model)
        return top_model
```

```
0 InputLayer False
1 Conv2D False
2 Conv2D False
3 MaxPooling2D False
4 Conv2D False
5 Conv2D False
6 MaxPooling2D False
7 Conv2D False
8 Conv2D False
9 Conv2D False
10 MaxPooling2D False
11 Conv2D False
12 Conv2D False
13 Conv2D False
14 MaxPooling2D False
15 Conv2D False
16 Conv2D False
17 Conv2D False
18 MaxPooling2D False
```

```
num_classes = 4
FC_Head = lw(modelVGG , num_classes)
model = Model(inputs = modelVGG.input,outputs = FC_Head)
```

Summary of the network layers

```
print(model.summary())
```

```
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
```

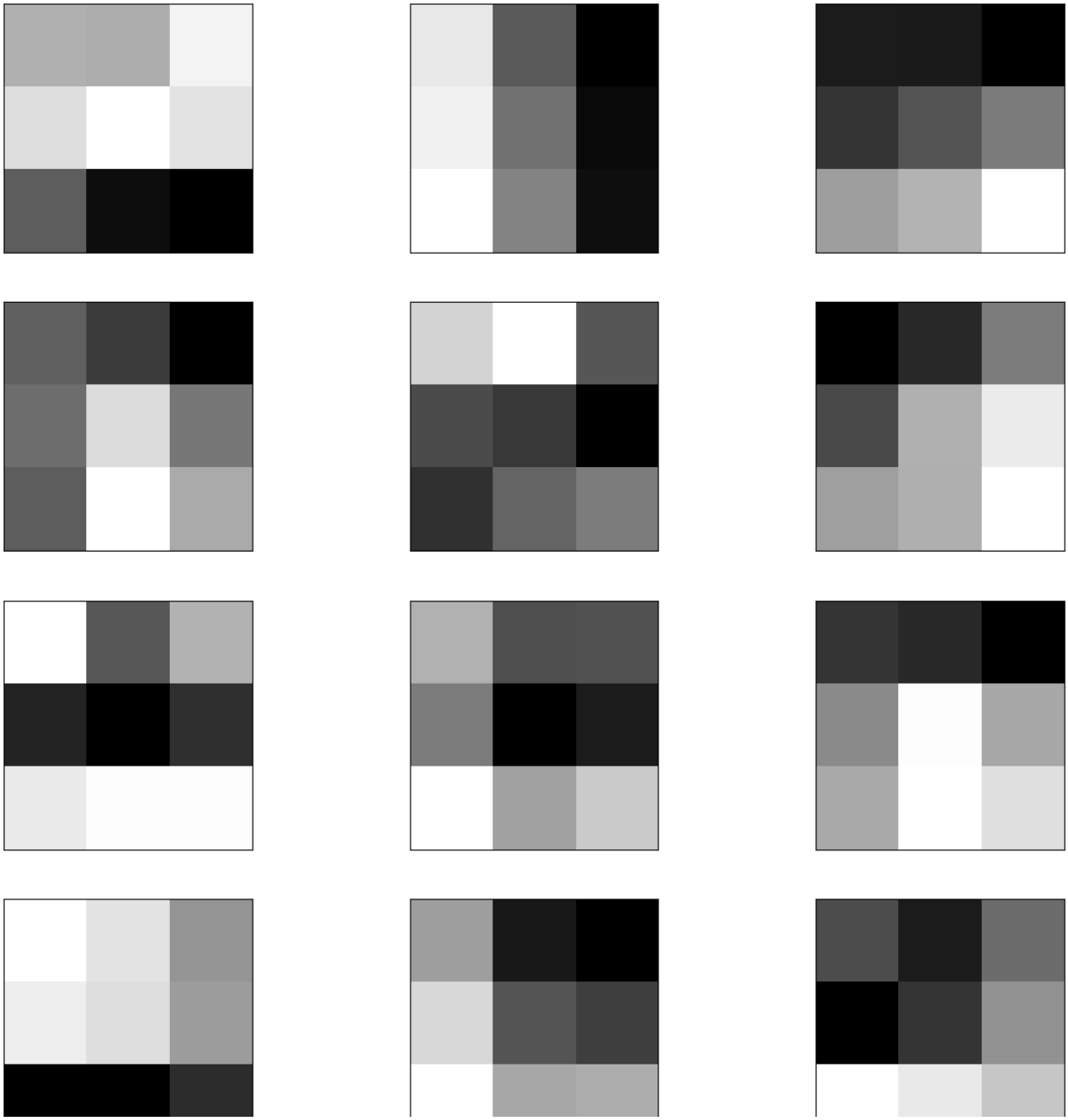
```
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 4096)              2101248

 dropout_1 (Dropout)         (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 512)               2097664

 dense_3 (Dense)             (None, 4)                 2052

=================================================================
Total params: 35696964 (136.17 MB)
Trainable params: 20982276 (80.04 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
None
```

```
# summarize filters in each convolutional layer
#from keras.applications.vgg16 import VGG16
#from matplotlib import pyplot
# load the model
#model = VGG16()
# summarize filter shapes
for layer in model.layers:
# check for convolutional layer
    if 'conv' not in layer.name:
        continue
    # get filter weights
    filters, biases = layer.get_weights()
    print(layer.name, filters.shape)
```

```
block1_conv1 (3, 3, 3, 64)
block1_conv2 (3, 3, 64, 64)
block2_conv1 (3, 3, 64, 128)
block2_conv2 (3, 3, 128, 128)
block3_conv1 (3, 3, 128, 256)
block3_conv2 (3, 3, 256, 256)
block3_conv3 (3, 3, 256, 256)
block4_conv1 (3, 3, 256, 512)
block4_conv2 (3, 3, 512, 512)
block4_conv3 (3, 3, 512, 512)
block5_conv1 (3, 3, 512, 512)
block5_conv2 (3, 3, 512, 512)
block5_conv3 (3, 3, 512, 512)
```

```
# normalize filter values to 0-1 so we can visualize them
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)
```

```
import matplotlib.pyplot as plt
f = plt.figure(figsize=(16,16))
nof_filters, ix = 4, 1
for i in range(nof_filters):
    f = filters[:, :, :, i]
    # plot each channel separately
    for j in range(3):
        ax = plt.subplot(nof_filters, 3, ix)
        ax.set_xticks([])
        ax.set_yticks([])
        # plot filter channel in grayscale
        plt.imshow(f[:, :, j], cmap='gray')
        ix += 1
# show the figure
plt.show()
```



Compiling the VGG16 model

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

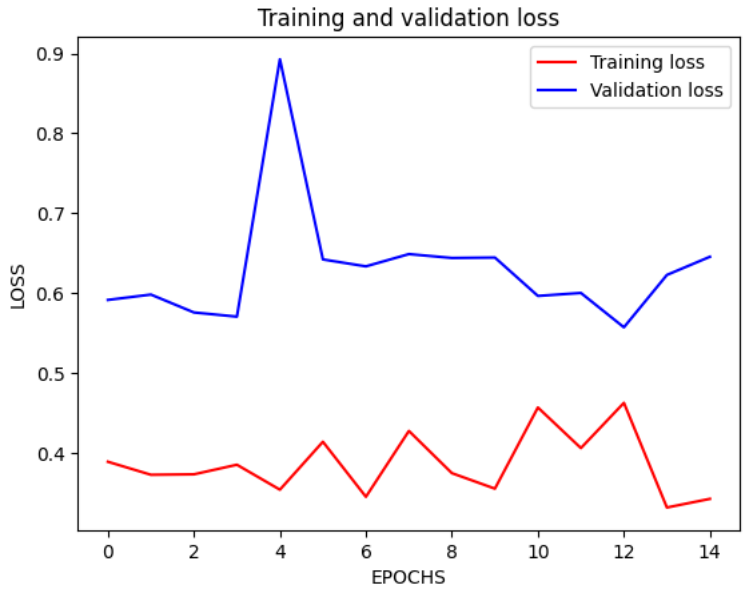
```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=50,
    width_shift_range=0.01,
    height_shift_range=0.01,
    horizontal_flip=False,
    vertical_flip=False)
datagen.fit(x_train)
```

Fitting the Model

```
#history = model.fit(x_train , y_train , epochs = 15, validation_data = (x_test , y_test), verbose = 1)
history = model.fit(datagen.flow(x_train,y_train),epochs=15,validation_data=(x_test,y_test),verbose=1)
```

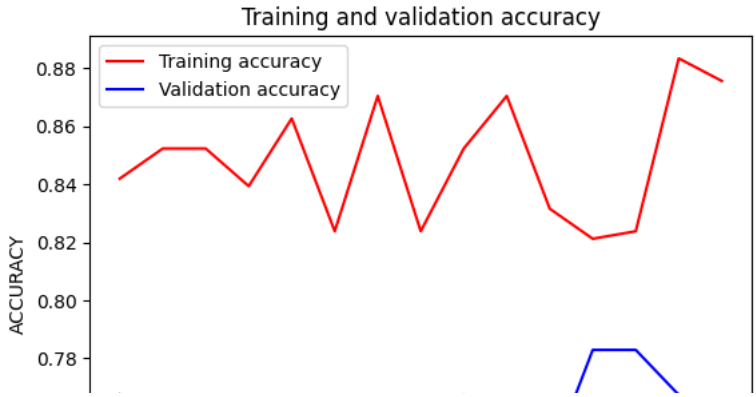
```
Epoch 1/15
13/13 [==============================] - 346s 27s/step - loss: 8.7699 - accuracy: 0.4249 - val_loss: 0.9820 - val_accuracy: 0.5504
Epoch 2/15
13/13 [==============================] - 330s 26s/step - loss: 0.9931 - accuracy: 0.5596 - val_loss: 0.8083 - val_accuracy: 0.6124
Epoch 3/15
13/13 [==============================] - 325s 25s/step - loss: 0.8981 - accuracy: 0.5725 - val_loss: 0.7870 - val_accuracy: 0.6512
Epoch 4/15
13/13 [==============================] - 321s 25s/step - loss: 0.9067 - accuracy: 0.5855 - val_loss: 0.8358 - val_accuracy: 0.6047
Epoch 5/15
13/13 [==============================] - 326s 25s/step - loss: 0.7300 - accuracy: 0.6658 - val_loss: 0.6889 - val_accuracy: 0.6667
Epoch 6/15
13/13 [==============================] - 381s 30s/step - loss: 0.7098 - accuracy: 0.6969 - val_loss: 0.6547 - val_accuracy: 0.7442
Epoch 7/15
13/13 [==============================] - 314s 24s/step - loss: 0.7580 - accuracy: 0.6736 - val_loss: 0.6741 - val_accuracy: 0.7597
Epoch 8/15
11/13 [========================>.....] - ETA: 36s - loss: 0.6385 - accuracy: 0.7143
```

```
%matplotlib inline
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))


plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('EPOCHS')
plt.ylabel('LOSS')
plt.legend(loc=0)
plt.figure()

plt.show()
```




```
<Figure size 640x480 with 0 Axes>
```

```
plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.xlabel('EPOCHS')
plt.ylabel('ACCURACY')

plt.figure()

plt.show()
```




```
print('Test accuracy:', model.evaluate(x_test, y_test))
```

```
5/5 [==============================] - 56s 11s/step - loss: 0.7071 - accuracy: 0.7597
Test accuracy: [0.7071017026901245, 0.7596899271011353]
```

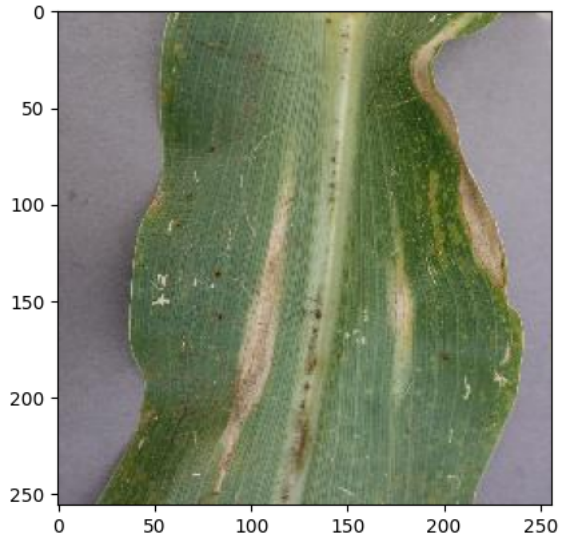
```
def names(number):
    if number==0:
        return "It's a leaf with disease BLIGHT"
    elif number==1:
        return "It's a leaf with disease Common Rust"
    elif number==2:
        return "It's a leaf with disease Gray_Leaf_Spot"
    elif number==3:
        return "It's a Healthy leaf"
```

Testing the model

```
from matplotlib.pyplot import imshow
def Prediction(img):
    #img = Image.open(r"/content/gdrive/MyDrive/data/Blight/Corn_Blight (1010).JPG")
    x = np.array(img.resize((224,224)))
    x = x.reshape(1,224,224,3)
    res = model.predict_on_batch(x)
    classification = np.where(res == np.amax(res))[1][0]
    imshow(img)
    print(str(res[0][classification]*100) + '% Confidence ' + names(classification))
```

```
img = Image.open(r"/content/gdrive/MyDrive/data/Blight/Corn_Blight (1010).JPG")
Prediction(img)
```

```
97.33526110649109% Confidence It's a leaf with disease BLIGHT
```



Testing with other random images

CNN

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

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

```
IMAGE_SIZE = (256,256)
BATCH_SIZE = 32
CHANNELS = 3
EPOCHES= 100
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/gdrive/MyDrive/corn_data/data",
    shuffle=True,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE
)
```

```
Found 4188 files belonging to 4 classes.
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import os
import glob as gb
path = "/content/gdrive/MyDrive/corn_data/data"
size=[]
for folder in os.listdir(path):
    files = gb.glob(pathname=str(path+"//"+folder+"/*.jpg"))
    for file in files:
        image=plt.imread(file)
        size.append(image.shape)
pd.Series(size).value_counts()
```

```
(256, 256, 3)      1323
(768, 1024, 3)        8
(2448, 3264, 3)       7
(1200, 675, 3)        7
(350, 750, 3)         5
                    ...
(378, 482, 3)         1
(329, 432, 3)         1
(679, 432, 3)         1
(1048, 1500, 3)       1
(309, 497, 3)         1
Length: 265, dtype: int64
```

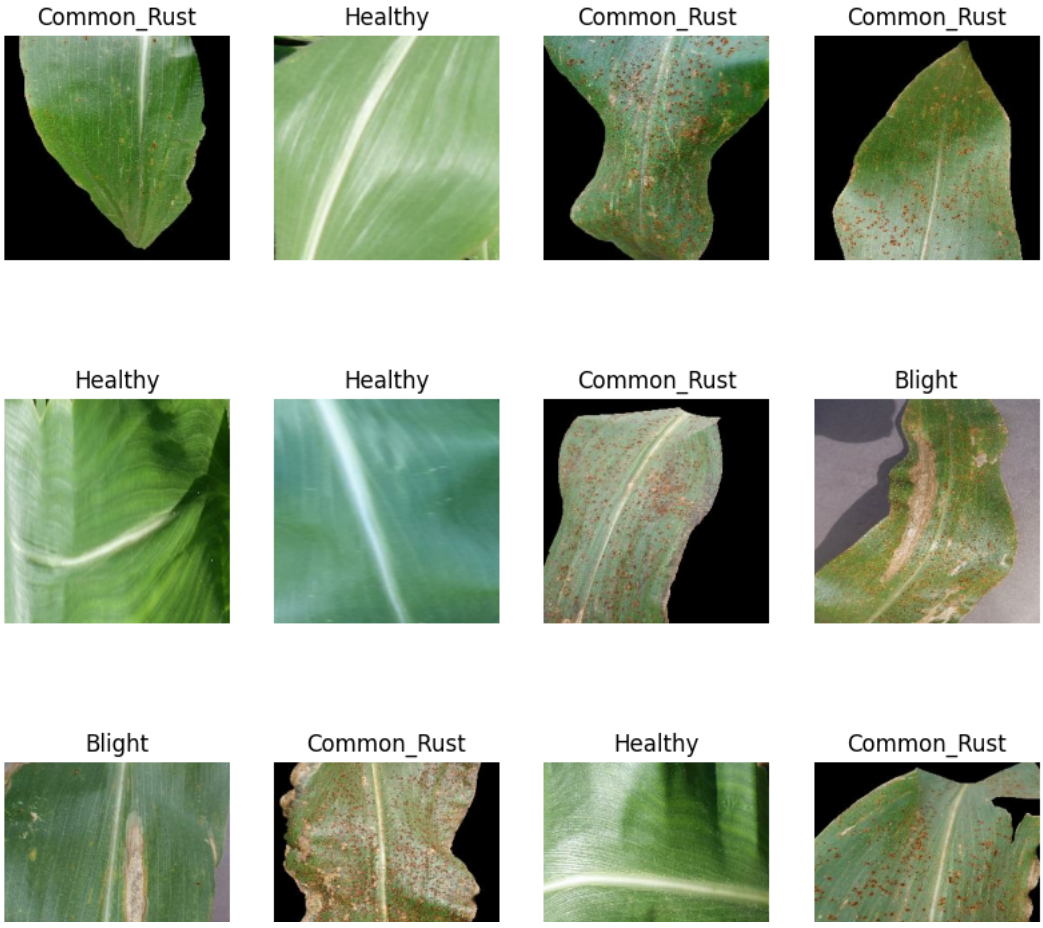
```
class_names = dataset.class_names
class_names
```

```
['Blight', 'Common_Rust', 'Gray_Leaf_Spot', 'Healthy']
```

```
len(dataset)
```

```
131
```

```
plt.figure(figsize=(10,10))
for image_batch , label_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
```




```
def get_dataset(ds,train_split=0.8,val_split=0.1,test_split=0.1,shuffle=True,shuffle_size=10000):
    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size,seed=8)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)

    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds,val_ds,test_ds


train_ds,val_ds,test_ds = get_dataset(dataset)


print("Length of Training Dataset is",len(train_ds))
print("\nLength of Validation Dataset is",len(val_ds))
print("\nLength of Testing Dataset is",len(test_ds))
```

```
Length of Training Dataset is 104

Length of Validation Dataset is 13

Length of Testing Dataset is 14
```

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


resize_and_rescale = tf.keras.Sequential([
layers.experimental.preprocessing.Resizing(256,256),
layers.experimental.preprocessing.Rescaling(1.0/255)
])


data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.3),
])
```

```
n_classes = 4
input_shape = (BATCH_SIZE,256,256,3)
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32,(3,3),activation ='relu',input_shape = input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size = (3,3),activation ='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128,kernel_size = (3,3),activation ='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(256,activation='relu'),
    layers.Dense(128,activation='relu'),
    layers.Dense(64,activation='relu'),
    layers.Dense(n_classes,activation='softmax'),

])
model.build(input_shape=input_shape)


model.summary()
```

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

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

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

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

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

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

 conv2d_2 (Conv2D)           (32, 60, 60, 128)         73856

 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 128)         0
 g2D)

 flatten (Flatten)           (32, 115200)              0

 dense (Dense)               (32, 256)                 29491456

 dense_1 (Dense)             (32, 128)                 32896

 dense_2 (Dense)             (32, 64)                  8256

 dense_3 (Dense)             (32, 4)                   260

=================================================================
Total params: 29626116 (113.01 MB)
Trainable params: 29626116 (113.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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


history = model.fit(
train_ds,
epochs=10,
batch_size=BATCH_SIZE,
verbose=1,
validation_data=val_ds)
```

```
Epoch 1/10
104/104 [==============================] - 562s 5s/step - loss: 0.6875 - accuracy: 0.7262 - val_loss: 0.4715 - val_accuracy: 0.7524
Epoch 2/10
104/104 [==============================] - 528s 5s/step - loss: 0.4613 - accuracy: 0.8099 - val_loss: 0.4119 - val_accuracy: 0.8149
Epoch 3/10
104/104 [==============================] - 530s 5s/step - loss: 0.3873 - accuracy: 0.8285 - val_loss: 0.3550 - val_accuracy: 0.8510
Epoch 4/10
104/104 [==============================] - 528s 5s/step - loss: 0.3590 - accuracy: 0.8400 - val_loss: 0.3129 - val_accuracy: 0.8702
Epoch 5/10
104/104 [==============================] - 532s 5s/step - loss: 0.3517 - accuracy: 0.8553 - val_loss: 0.4366 - val_accuracy: 0.8510
```

```
Epoch 6/10
104/104 [==============================] - 498s 5s/step - loss: 0.3060 - accuracy: 0.8682 - val_loss: 0.3238 - val_accuracy: 0.8510
Epoch 7/10
104/104 [==============================] - 506s 5s/step - loss: 0.3032 - accuracy: 0.8806 - val_loss: 0.2878 - val_accuracy: 0.8750
Epoch 8/10
104/104 [==============================] - 523s 5s/step - loss: 0.3620 - accuracy: 0.8529 - val_loss: 0.3425 - val_accuracy: 0.8389
Epoch 9/10
104/104 [==============================] - 551s 5s/step - loss: 0.2927 - accuracy: 0.8800 - val_loss: 0.2622 - val_accuracy: 0.8894
Epoch 10/10
104/104 [==============================] - 518s 5s/step - loss: 0.2686 - accuracy: 0.8929 - val_loss: 0.2933 - val_accuracy: 0.8750
```

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

```
14/14 [==============================] - 52s 1s/step - loss: 0.2643 - accuracy: 0.9062
```

```
history
```

```
<keras.src.callbacks.History at 0x7e66fbfcf8b0>
```

```
history.params
```

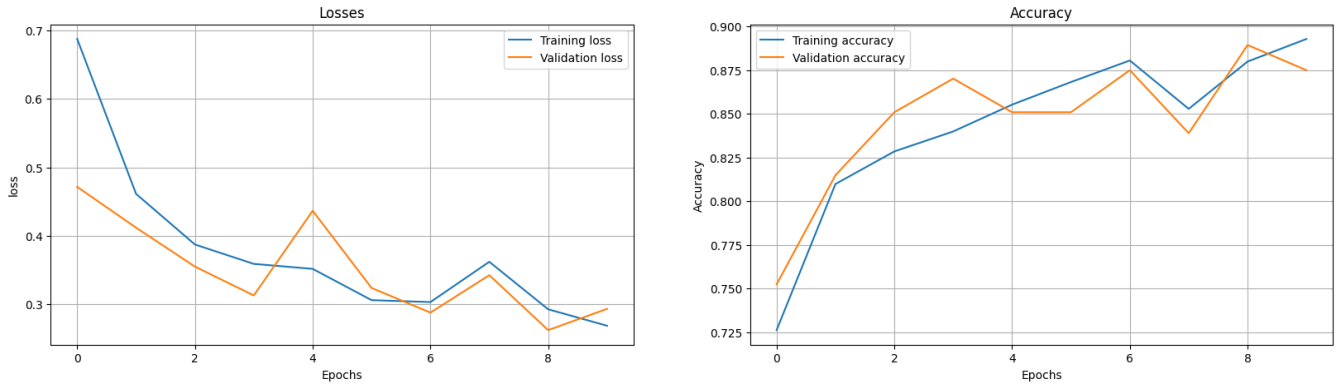
```
{'verbose': 1, 'epochs': 10, 'steps': 104}
```

```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

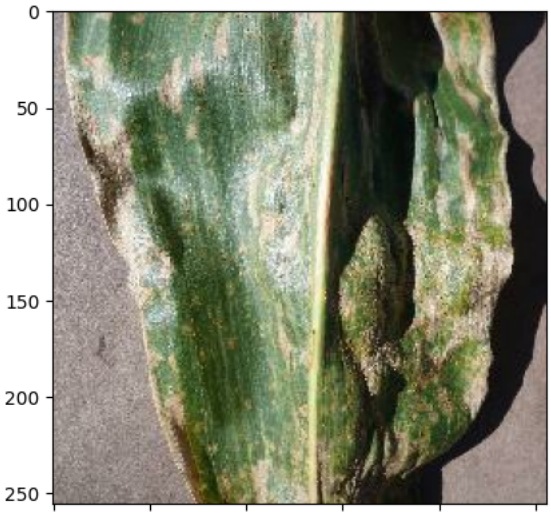
```
his_data = pd.DataFrame(history.history)
plt.figure(figsize=(20,5))

plt.subplot(1,2,1)
plt.plot(his_data.loss, label="Training loss")
plt.plot(his_data.val_loss, label="Validation loss")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.title("Losses")
plt.grid()
plt.legend()

plt.subplot(1,2,2)
plt.plot(his_data.accuracy, label="Training accuracy")
plt.plot(his_data.val_accuracy, label="Validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Accuracy")
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7e66fbe7b3d0>
```



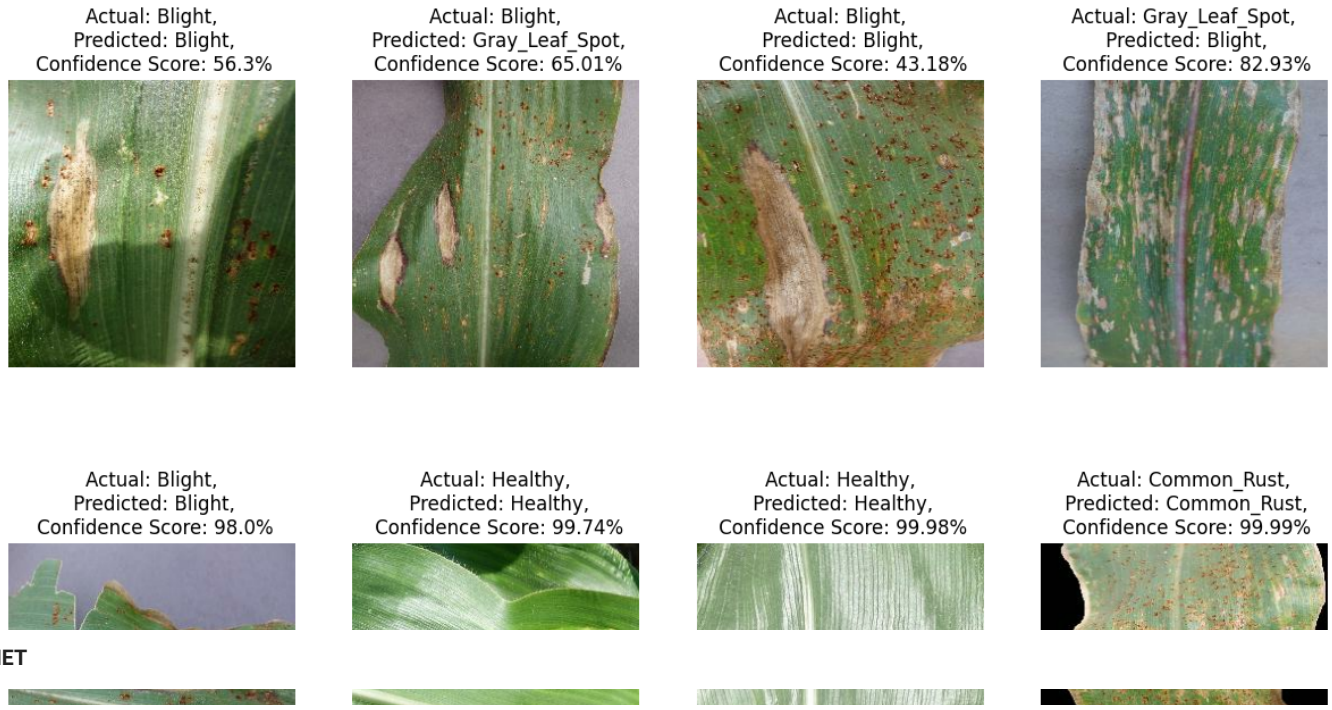

```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):
    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()
    print("First Image to Predict :")
    plt.imshow(first_image)
    print("\nActual label:",class_names[first_label])
    batch_prediction = model. predict (images_batch)
    print("\nPredicted label" ,class_names[np.argmax(batch_prediction[0])])
```

```
First Image to Predict :

Actual label: Gray_Leaf_Spot
1/1 [==============================] - 1s 1s/step

Predicted label Gray_Leaf_Spot
```




```
def predict(model,img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0) # Create a batch
    predictions = model.predict (img_array)
    predicted_class = class_names[np.argmax(predictions[0])]
#     confidence = round(np.max(predictions[0]),2)
    confidence = round(100*(np.max(predictions[0])),2)

    return predicted_class, confidence


plt.figure(figsize=(15,15))
for images,labels in test_ds.take(1):
    for i in range(12):
        ax = plt.subplot(3,4,i+1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class , confidence = predict(model,images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class},\n Confidence Score: {confidence}%")
        plt.axis("off")
```

```
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 82ms/step
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 77ms/step
```



RESNET

```
import os
import time
import shutil
import pathlib
import itertools
# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers
# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
print ('modules loaded')
```

```
modules loaded
modules loaded
```

```
data_dir = '/content/gdrive/MyDrive/corn_data/data'
filepaths = []
labels = []
folds = os.listdir(data_dir)
for fold in folds:
 foldpath = os.path.join(data_dir, fold)
filelist = os.listdir(foldpath)
for file in filelist:
 fpath = os.path.join(foldpath, file)
filepaths.append(fpath)
```

```
labels.append(fold)
# Concatenate data paths with labels into one dataframe
Fseries = pd.Series(filepaths, name= 'filepaths')
Lseries = pd.Series(labels, name='labels')
df = pd.concat([Fseries, Lseries], axis= 1)
```

```
df
```

	filepaths	labels
0	/content/gdrive/MyDrive/corn_data/data/Healthy...	Healthy

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random

import torch
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
from torch import nn
from torch.optim import Adam
from torchvision import models

from sklearn.metrics import classification_report, confusion_matrix
```

```
!pip install split-folders
```

```
Requirement already satisfied: split-folders in /usr/local/lib/python3.10/dist-packages (0.5.1)
```

```
import splitfolders
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
'cpu'
```

```
splitfolders.ratio("/content/gdrive/MyDrive/corn_data/data",
            output="splitted_data",
            seed=42,
            ratio=(.7, .2, .1),
            group_prefix=None,
            move=False)
```

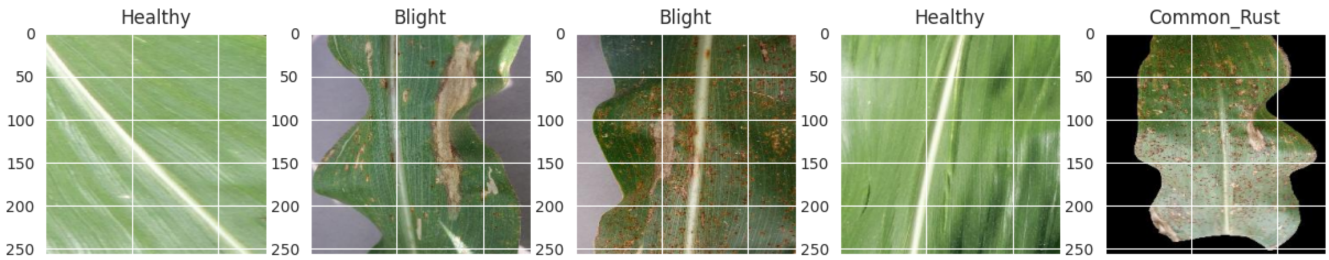
```
Copying files: 4188 files [00:21, 190.53 files/s]
```

```
#----------------------
train_transform = transforms.Compose([
    transforms.Resize(size=(256, 256)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.GaussianBlur(kernel_size=(3, 7), sigma=(0.1, 2)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize(size=(256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize(size=(256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
# -------- Transformation for the Whole Dataset for Visualization Purposes --------
data_transform = transforms.Compose([
    transforms.Resize(size=(256, 256)),
    transforms.ToTensor(),
])
```

```
train = datasets.ImageFolder(root="/content/gdrive/MyDrive/corn_data/data",
                             transform=train_transform)

val = datasets.ImageFolder(root="/content/gdrive/MyDrive/corn_data/data",
                             transform=val_transform)

test = datasets.ImageFolder(root="/content/gdrive/MyDrive/corn_data/data",
                             transform=test_transform)

# -------- Getting the Whole Dataset for Visualization Purposes --------
data = datasets.ImageFolder(root="/content/gdrive/MyDrive/corn_data/data",
                             transform=data_transform)


print("Dataset Labels:\n", train.class_to_idx, "\n")

for name, dataset in zip(["TRAIN", "VALIDATION", "TEST"], [train, val, test]):
    images_per_class = pd.Series(dataset.targets).value_counts()
    print(f"Images per Class in {name}:")
    print(images_per_class, "\n")
```

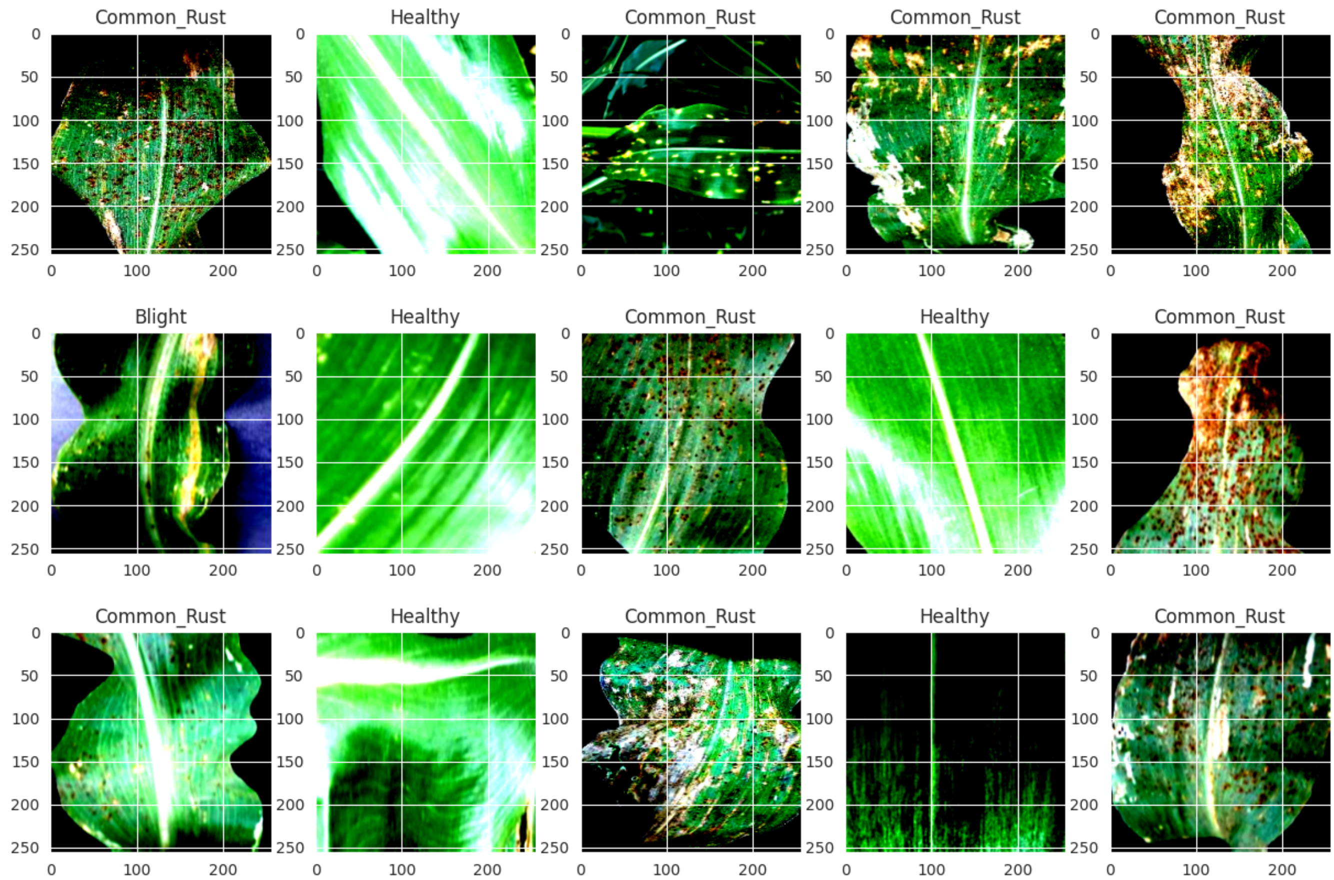
```
Dataset Labels:
 {'Blight': 0, 'Common_Rust': 1, 'Gray_Leaf_Spot': 2, 'Healthy': 3}

Images per Class in TRAIN:
1    1306
3    1162
0    1146
2     574
dtype: int64

Images per Class in VALIDATION:
1    1306
3    1162
0    1146
2     574
dtype: int64

Images per Class in TEST:
1    1306
3    1162
0    1146
2     574
dtype: int64
```

```
labels_for_viz = {v: k for k, v in data.class_to_idx.items()}

fig, ax = plt.subplots(3, 5, figsize=(15, 10))
ax = ax.flatten()
for i in range(15):
    sample = random.randint(0, len(data))
    ax[i].imshow(data[sample][0].permute(1, 2, 0))
    ax[i].title.set_text(labels_for_viz[data[sample][1]])
```




```
fig, ax = plt.subplots(3, 5, figsize=(15, 10))
ax = ax.flatten()
for i in range(15):
    sample = random.randint(0, len(train))
    ax[i].imshow(train[sample][0].permute(1, 2, 0))
    ax[i].title.set_text(labels_for_viz[train[sample][1]])
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
```




```
import os
print(os.listdir("/content/drive/MyDrive/corn_data/data"))
```

```
['Common_Rust', 'Gray_Leaf_Spot', 'Blight', 'Healthy']
```

```
import matplotlib.pyplot as plt
```

```
path = "/content/drive/MyDrive/corn_data/data"
classes = os.listdir(path)
```

```
print('classes: ', classes)

def display_images(random_number):
    for i in classes:
        new_path = path + i
        image_path = new_path + '/' + os.listdir(new_path)[random_number]
        plt.title(i)
        plt.imshow(plt.imread(image_path))
        plt.show()
```

```
classes:  ['Common_Rust', 'Gray_Leaf_Spot', 'Blight', 'Healthy']
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
'''for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
'''
# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Sav
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
'for dirname, _, filenames in os.walk('/kaggle/input'):\n    for filename in filenames:\n        print(os.path.join(dirname, filen
ame))\n'
```

```
import os
import pandas as pd
import numpy as np


import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from keras.models import Sequential,Model
from keras.layers import Dense,Conv2D,MaxPool2D,BatchNormalization,Dropout,Flatten,Activation,GlobalAveragePooling2D
from PIL import Image
import tensorflow as tf
tf.keras.applications.VGG16
from keras.applications.vgg16 import VGG16
from pathlib import Path


os.listdir('/content/drive/MyDrive/corn_data/data')
```

```
['Common_Rust', 'Gray_Leaf_Spot', 'Blight', 'Healthy']
```

```
import matplotlib.pyplot as plt
from matplotlib import image
from PIL import Image


def load_imgs(impath):
    imgs=[]
    label=[]
    l1=os.listdir(impath)
    for i in l1:
        c=0
        l2=os.listdir(impath+'/'+i)
        for j in l2:
            if c<=1000:
                img = Image.open(impath+'/'+i+'/'+j)
                img = img.resize((224,224))
                img = np.array(img)
                if(img.shape == (224,224,3)):
                    imgs.append(img)
                    label.append(i)
                del img
                c=c+1
                '''if(c%1000==0):
            plt.imshow(img)
```

```
            plt.show()
            c=c+1'''
    return np.array(imgs),label


x,y=load_imgs('/content/drive/MyDrive/corn_data/data/')
x.shape,len(y)
```

```
((3573, 224, 224, 3), 3573)
```

```
target=pd.Series(y,dtype='category')
target
```

```
0       Common_Rust
1       Common_Rust
2       Common_Rust
3       Common_Rust
4       Common_Rust
           ...
3568        Healthy
3569        Healthy
3570        Healthy
3571        Healthy
3572        Healthy
Length: 3573, dtype: category
Categories (4, object): ['Blight', 'Common_Rust', 'Gray_Leaf_Spot', 'Healthy']
```

```
target.value_counts()
```

```
Blight            1001
Healthy           1001
Common_Rust        999
Gray_Leaf_Spot     572
dtype: int64
```

```
t=target.cat.codes
t
```

```
0       1
1       1
2       1
3       1
4       1
       ..
3568    3
3569    3
3570    3
3571    3
3572    3
Length: 3573, dtype: int8
```

```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y=train_test_split(x,t,test_size=0.2,shuffle=True)
train_x.shape,test_x.shape,train_y.shape,test_y.shape
```

```
((2858, 224, 224, 3), (715, 224, 224, 3), (2858,), (715,))
```

```
from tensorflow.keras import layers
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D,
from tensorflow.keras.models import Model, load_model


def channel_attention(input_tensor, reduction_ratio=16):
    channels = input_tensor.shape[-1]

    # Global Average Pooling
    x = GlobalAveragePooling2D()(input_tensor)

    # Channel-wise fully connected layers
    x = Dense(channels // reduction_ratio, activation='relu')(x)
    x = Dense(channels, activation='sigmoid')(x)

    # Reshape and multiply attention weights
    x = tf.reshape(x, [-1, 1, 1, channels])
    x = Multiply()([input_tensor, x])

    return x


def spatial_attention(input_tensor):
    # Convolutional layers for spatial attention
    x = Conv2D(filters=1, kernel_size=7, padding='same', activation='sigmoid')(input_tensor)
```

```
        # Multiply attention weights
        x = Multiply()([input_tensor, x])

        return x


    import tensorflow as tf


    def CBAM_ResNet50(input_shape, num_classes, reduction_ratio=16):
        # Input tensor
        input_tensor = Input(shape=input_shape)

        # ResNet50 base model
        base_model = tf.keras.applications.ResNet50(include_top=False, weights='imagenet', input_tensor=input_tensor)
        for layer in base_model.layers:
            layer.trainable = False
        # CBAM implementation
        x = base_model.output
        x = channel_attention(x, reduction_ratio)
        x = spatial_attention(x)

        # Classifier head
        x = GlobalAveragePooling2D()(x)
        x = Dense(1024, activation='relu')(x)
        x = Dense(num_classes, activation='softmax')(x)

        # Create the CBAM ResNet50 model
        model = Model(inputs=base_model.input, outputs=x)

        return model



    from tensorflow.keras.layers import Multiply


    input_shape = (224, 224, 3)
    num_classes = 4  # Change this according to your specific classification task
    reduction_ratio = 16

    model = CBAM_ResNet50(input_shape, num_classes, reduction_ratio)
    model.summary()
```

```
                                                                                  conv2d[0][0]']

 global_average_pooling2d_1  (None, 2048)                0         ['multiply_1[0][0]']
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1024)                2098176   ['global_average_pooling2d_1[0
                                                                    ][0]']

 dense_5 (Dense)             (None, 4)                   4100      ['dense_4[0][0]']

==================================================================================================
Total params: 26316805 (100.39 MB)
Trainable params: 2729093 (10.41 MB)
Non-trainable params: 23587712 (89.98 MB)
__________________________________________________________________________________________________
```

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

```
(2858, 224, 224, 3)
```

```
train_y.shape
```

```
(2858,)
```

```
model.fit(train_x, train_y, epochs = 10, batch_size = 1, shuffle=True)
```

```
Epoch 1/10
2858/2858 [==============================] - 905s 314ms/step - loss: 0.8720 - accuracy: 0.5511
Epoch 2/10
2858/2858 [==============================] - 888s 311ms/step - loss: 0.5263 - accuracy: 0.7491
Epoch 3/10
2858/2858 [==============================] - 890s 311ms/step - loss: 0.3586 - accuracy: 0.8439
Epoch 4/10
2858/2858 [==============================] - 934s 327ms/step - loss: 0.2746 - accuracy: 0.8849
Epoch 5/10
2858/2858 [==============================] - 913s 319ms/step - loss: 0.2236 - accuracy: 0.9122
Epoch 6/10
2858/2858 [==============================] - 903s 316ms/step - loss: 0.2004 - accuracy: 0.9342
Epoch 7/10
2858/2858 [==============================] - 882s 309ms/step - loss: 0.1792 - accuracy: 0.9405
Epoch 8/10
2858/2858 [==============================] - 845s 296ms/step - loss: 0.1098 - accuracy: 0.9619
Epoch 9/10
2858/2858 [==============================] - 835s 292ms/step - loss: 0.1756 - accuracy: 0.9514
Epoch 10/10
2851/2858 [============================>.] - ETA: 2s - loss: 0.0951 - accuracy: 0.9646
```

```
model.evaluate(test_x,test_y)
```