Crop_Prediction_using_Deep Learning_CNN

In [1]:
```
#import libraries
import os
import cv2
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint

#for accuracy and loss graph
import matplotlib.pyplot as plt
```

In [2]:
```
keras.__version__
```

Out[2]:
```
'2.10.0'
```

In [3]:
```
train_data_path=r"C:\Users\user\Desktop\Data_Science _Project\Project _For Exp
#train_data_path = "/content/drive/My Drive/My ML Project /DL Project/CNN/cott
validation_data_path = r"C:\Users\user\Desktop\Data_Science _Project\Project _
```

In [4]:
```
train_data_path.index
```

Out[4]:
```
<function str.index>
```

In [20]:
```
#Show Augmented image
def plotImages(image_arr):
    fig,axes =plt.subplots(1,5,figsize=(20,20))
    axes= axes.flatten()
    for img, ax in zip(image_arr,axes):
        ax.imshow(img)
        plt.tight_layout()
        plt.show()
```

In [6]:
```
# this is the augmentation configuration we will use for training
# It generate more images using below parameters
training_datgen=ImageDataGenerator(rescale=1/255,
                                   rotation_range=40,
                                   height_shift_range=0.2,
                                   width_shift_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
```

In [7]:
```
# this is a generator that will read pictures found in
# at train_data_path, and indefinitely generate
# batches of augmented image data
training_data=training_datgen.flow_from_directory(train_data_path,# this is th
                                                  target_size=(150, 150),# all i
                                                  class_mode='binary',#number of
                                                  batch_size=32)# since we use b
```

```
Found 1951 images belonging to 4 classes.
```

In [8]:
```
training_data.class_indices # check how many diffrent class es are there
```

Out[8]:
```
{'diseased cotton leaf': 0,
 'diseased cotton plant': 1,
 'fresh cotton leaf': 2,
 'fresh cotton plant': 3}
```

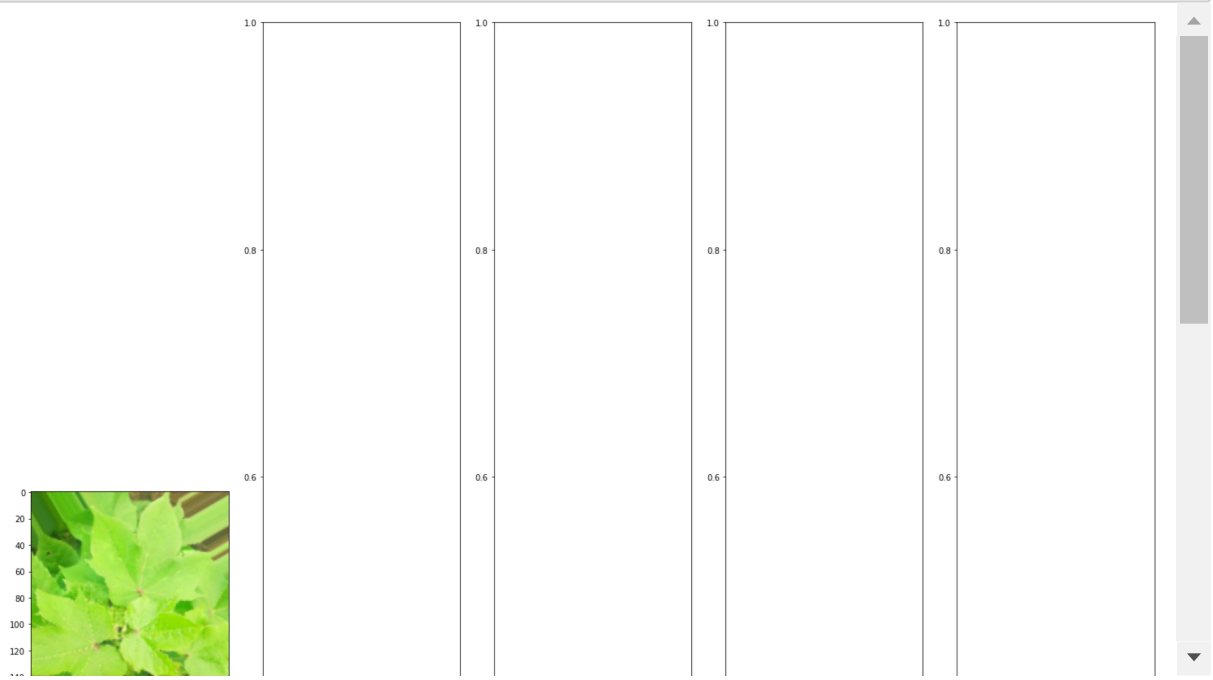
In [9]:
```
# this is the augmentation configuration we will use for validation:
# only rescaling beacuse wea are taking one by one image to check
valid_datagen=ImageDataGenerator(rescale=1/255)
```

In [10]:
```
# this is a similar generator, for validation data
valid_data=valid_datagen.flow_from_directory(validation_data_path,# this is th
                                             target_size=(150, 150),# all i
                                             class_mode='binary',#number of
                                             batch_size=32)# since we use b
```

```
Found 324 images belonging to 4 classes.
```

showing augmented images

In [11]:
```
images = [training_data[0][0][0] for i in range(5)]
plotImages(images)
```



save best model whose validation accuracy is high using vall accuracy

In [12]:
```
modelpath=r"C:\Users\user\Desktop\Data_Science _Project\Project _For Experienc
checkpoint=ModelCheckpoint(modelpath,monitor="val_accuracy",verbose=1, save_be
callbacks_list=[checkpoint]
```

Building CNN model

In [13]:
```
cnn_model= keras.models.Sequential([

                                    keras.layers.Conv2D(filters=32,kernel_size
                                    keras.layers.MaxPooling2D(pool_size=(2,2))
                                    keras.layers.Conv2D(filters=64,kernel_size
                                    keras.layers.MaxPooling2D(pool_size=(2,2))
                                    keras.layers.Conv2D(filters=128,kernel_siz
                                    keras.layers.MaxPooling2D(pool_size=(2,2))
                                    keras.layers.Conv2D(filters=256,kernel_siz
                                    keras.layers.MaxPooling2D(pool_size=(2,2))

                                    keras.layers.Dropout(0.5),
                                    keras.layers.Flatten(), #Building neurl ne
                                    keras.layers.Dense(units=128, activation="
                                    keras.layers.Dropout(0.1),
                                    keras.layers.Dense(units=256, activation="
                                    keras.layers.Dropout(0.25),
                                    keras.layers.Dense(units=4, activation="So
])
```

Compile CNN model

In [14]:
```
cnn_model.compile(optimizer=Adam(learning_rate=0.0001),loss='sparse_categorica
```

In [15]:
```
cnn_model.summary()
```

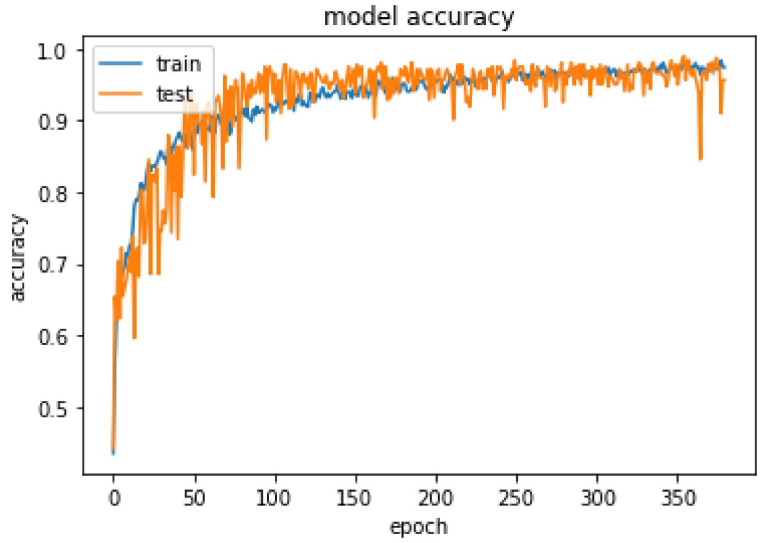
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 256)        0
 2D)

 dropout (Dropout)           (None, 7, 7, 256)         0

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 128)               1605760

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 256)               33024

 dropout_2 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 4)                 1028

=================================================================
Total params: 2,028,228
Trainable params: 2,028,228
Non-trainable params: 0
_________________________________________________________________
```

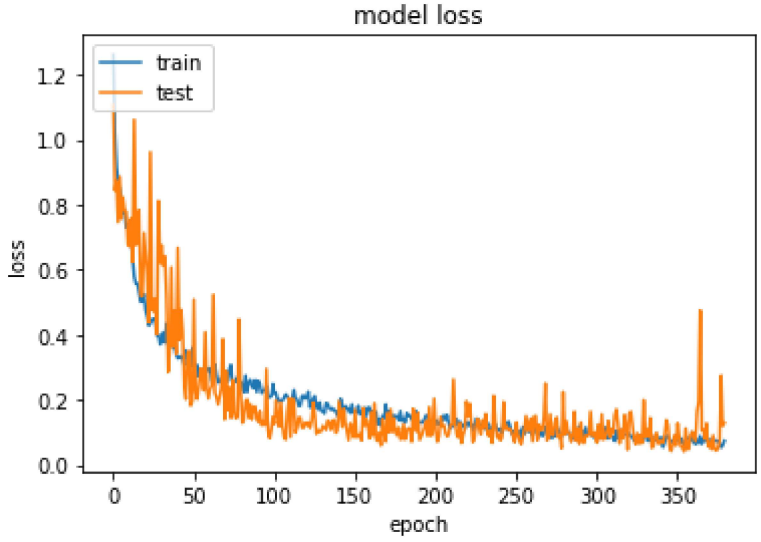
In [16]:
```
# Train cnn model
history=cnn_model.fit(training_data,
                      epochs=380,
                      verbose=1,
                      validation_data=valid_data,
                      callbacks=callbacks_list) # time start 16.06
```

```
Epoch 376: val_accuracy did not improve from 0.99074
61/61 [==============================] - 103s 2s/step - loss: 0.0732 - accu
racy: 0.9754 - val_loss: 0.0511 - val_accuracy: 0.9722
Epoch 377/380
61/61 [==============================] - ETA: 0s - loss: 0.0687 - accuracy:
0.9774
Epoch 377: val_accuracy did not improve from 0.99074
61/61 [==============================] - 90s 1s/step - loss: 0.0687 - accur
acy: 0.9774 - val_loss: 0.0700 - val_accuracy: 0.9753
Epoch 378/380
61/61 [==============================] - ETA: 0s - loss: 0.0536 - accuracy:
0.9846
Epoch 378: val_accuracy did not improve from 0.99074
61/61 [==============================] - 94s 2s/step - loss: 0.0536 - accur
acy: 0.9846 - val_loss: 0.2748 - val_accuracy: 0.9105
Epoch 379/380
61/61 [==============================] - ETA: 0s - loss: 0.0593 - accuracy:
0.9749
Epoch 379: val_accuracy did not improve from 0.99074
```

In [17]:
```
model_path2=r"C:\Users\user\Desktop\Data_Science _Project\Project _For Experie
cnn_model.save(model_path2)
```

In [21]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
```





In [19]:
```
history.history
```

```
0.16010591387748718,
0.18631568551063538,
0.15938708186149597,
0.16472658514976501,
0.16252337396144867,
0.1818152368068695,
0.15474338829517365,
0.16278237104415894,
0.1649567186832428,
0.17495745420455933,
0.1508922427892685,
0.1765608936548233,
0.17586328089237213,
0.15805773437023163,
0.16582506895065308,
0.16056528687477112,
0.16342630982398987,
0.12405578047037125,
0.14094223082065582,
```

In [ ]:

In [ ]:

In [ ]: