In [1]:
```python
import tensorflow as tf
from tensorflow.keras import models, layers, Sequential
import matplotlib.pyplot as plt
```

In [2]:
```python
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory = "Cotton Plant Disease Dataset",
    shuffle = True,
    label_mode = 'int',
    batch_size = 32,
    image_size = (256,256)
)
```

```
Found 1707 files belonging to 4 classes.
```

In [3]:
```python
class_name = dataset.class_names
class_name
```

Out[3]:
```
['Bacterial Blight', 'Curl Virus', 'Fussarium Wilt', 'Healthy']
```

In [4]:
```python
n_class = len(class_name)
n_class
```

Out[4]:
```
4
```

Print Image of each batch

In [5]:
```python
plt.figure(figsize = (3,3))
for image_batch, label_batch in dataset.take(1):
    # It show tensor of images
    print(image_batch[0])
    # It Show numpy array images
    print(image_batch[0].numpy())
    # it show shape of image
    print(image_batch[0].shape)
    # it show the different class value
    print("class number = ", label_batch[0].numpy())
    # It show the class name of the image
    print("class Name = ", class_name[label_batch[0]])
    # It show the specfic image
    plt.imshow(image_batch[0].numpy().astype('uint8'))
    plt.axis('off')
```

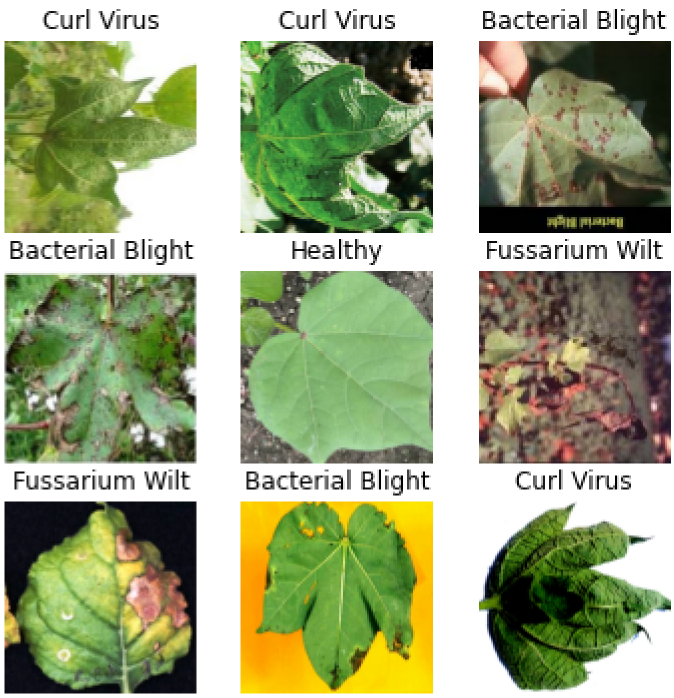
```
tf.Tensor(
[[[202.11133  174.11133  126.111336]
  [185.35191  157.35191  109.35191 ]
  [182.53188  154.53188  106.531876]
  ...
  [192.2875   163.02774  120.088295]
  [179.38672  160.36328  125.59962 ]
  [247.79993  241.34473  218.4209  ]]

 [[203.31021  175.31021  127.31021 ]
  [202.46774  174.46774  126.46774 ]
  [203.82057  175.82057  127.82057 ]
  ...
  [208.03865  178.77888  135.83943 ]
  [197.03322  178.00978  143.24611 ]
  [249.98341  247.95444  225.2614  ]]

 [[213.9313   185.9313   137.9313  ]
  [202.80698  174.80698  126.80698 ]
  [197.74576  169.74576  121.745766]
  ...
```

```
  [211.57106  182.3113   139.37184 ]
  [200.31703  181.2936   146.52992 ]
  [250.23143  247.7337   225.34125 ]]

 ...

 [[216.55318  188.55318  138.55318 ]
  [212.84433  184.84433  134.84433 ]
  [220.67851  192.67851  142.67851 ]
  ...
  [204.14815  174.46846  130.68916 ]
  [200.96875  179.70117  145.68555 ]
  [250.58917  246.15419  226.97841 ]]

 [[215.92937  187.92937  137.92937 ]
  [209.35843  181.35843  131.35843 ]
  [215.69461  187.69461  137.69461 ]
  ...
  [203.70021  174.02052  130.24123 ]
  [201.43262  180.16504  146.14941 ]
  [250.6123   246.04492  226.86914 ]]

 [[218.50238  190.50238  140.50238 ]
  [206.91245  178.91245  128.91245 ]
  [208.70789  180.70789  130.70789 ]
  ...
  [199.50826  169.82857  126.04927 ]
  [199.59343  178.32585  144.31023 ]
  [250.4719   244.37305  225.19727 ]]], shape=(256, 256, 3), dtype=float32)
[[[202.11133  174.11133  126.111336]
  [185.35191  157.35191  109.35191 ]
  [182.53188  154.53188  106.531876]
  ...
  [192.2875   163.02774  120.088295]
  [179.38672  160.36328  125.59962 ]
  [247.79993  241.34473  218.4209  ]]

 [[203.31021  175.31021  127.31021 ]
  [202.46774  174.46774  126.46774 ]
  [203.82057  175.82057  127.82057 ]
  ...
  [208.03865  178.77888  135.83943 ]
  [197.03322  178.00978  143.24611 ]
  [249.98341  247.95444  225.2614  ]]

 [[213.9313   185.9313   137.9313  ]
  [202.80698  174.80698  126.80698 ]
  [197.74576  169.74576  121.745766]
  ...
  [211.57106  182.3113   139.37184 ]
  [200.31703  181.2936   146.52992 ]
  [250.23143  247.7337   225.34125 ]]

 ...

 [[216.55318  188.55318  138.55318 ]
  [212.84433  184.84433  134.84433 ]
  [220.67851  192.67851  142.67851 ]
  ...
  [204.14815  174.46846  130.68916 ]
  [200.96875  179.70117  145.68555 ]
  [250.58917  246.15419  226.97841 ]]

 [[215.92937  187.92937  137.92937 ]
  [209.35843  181.35843  131.35843 ]
  [215.69461  187.69461  137.69461 ]
  ...
  [203.70021  174.02052  130.24123 ]
  [201.43262  180.16504  146.14941 ]
  [250.6123   246.04492  226.86914 ]]

 [[218.50238  190.50238  140.50238 ]
  [206.91245  178.91245  128.91245 ]
  [208.70789  180.70789  130.70789 ]
```

```
 ...
 [199.50826  169.82857  126.04927 ]
 [199.59343  178.32585  144.31023 ]
 [250.4719   244.37305  225.19727 ]]]
(256, 256, 3)
class number =  2
class Name =  Fussarium Wilt
```

In [7]:

```python
plt.figure(figsize = (6,6))

for image_batch, label_batch in dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3,3,i+1)
    plt.imshow(image_batch[i].numpy().astype('uint8'))
    plt.title(class_name[label_batch[i]])
    plt.axis('off')
```



Divide the Dataset into Train, Test, Valid

In [5]:

```python
# The size of train dataset is 80%
# The size of test dataset is 10%
# The size of valid dataset is 10%

train_datasize = int(0.8*len(dataset))
test_datasize = int(0.1*len(dataset))
valid_datasize = int(0.1*len(dataset))

train_datasize, test_datasize, valid_datasize
```

Out[5]:

```
(43, 5, 5)
```

In [6]:

```
Train_dataset = dataset.take(train_datasize)
Test_dataset = dataset.skip(train_datasize).take(test_datasize)
Valid_dataset = dataset.skip(train_datasize+test_datasize)
print(f"Train Dataset lenght = {len(Train_dataset)}\nTest Dataset lenght = {len(Test_
```

```
Train Dataset lenght = 43
Test Dataset lenght = 5
Valid Dataset lenght = 6
```

Through a function we also splite the Dataset

In [8]:

```
def split_dataset(dataset, train_size, test_size, valid_size, shuffle = True, shuffle
  dataset_size = len(dataset)

  if shuffle:
    dataset = dataset.shuffle(shuffle_size, seed = 15)

  train_datasize = int(train_size*dataset_size)
  test_datasize = int(test_size*dataset_size)
  valid_datasize = int(valid_size*dataset_size)

  Train_dataset = dataset.take(train_datasize)
  Test_dataset = dataset.skip(train_datasize).take(test_datasize)
  Valid_dataset = dataset.skip(train_datasize+test_datasize)

  return Train_dataset, Test_dataset, Valid_dataset
```

In [9]:

```
Train_dataset, Test_dataset, Valid_dataset = split_dataset(dataset, 0.8, 0.1, 0.1)
print(f"Train Dataset lenght = {len(Train_dataset)}\nTest Dataset lenght = {len(Test_
```

```
Train Dataset lenght = 43
Test Dataset lenght = 5
Valid Dataset lenght = 6
```

Preprocessing

In [11]:

```
Train_dataset = Train_dataset.cache().shuffle(1000).prefetch(buffer_size = tf.data.AU
Test_dataset = Test_dataset.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTO
Valid_dataset = Valid_dataset.cache().shuffle(1000).prefetch(buffer_size = tf.data.AU
```

In [12]:

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

In [13]:

```
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])
```

Resnet50 Model

In [17]:

```
Resnet50 = tf.keras.applications.ResNet50(
    include_top = False,
    weights="imagenet",
    input_shape = (256,256,3),
    classes = 4
)
```

In [18]:

```
Resnet_Model = Sequential()
Resnet50.trainable = False

Resnet_Model.add(Resnet50)
Resnet_Model.add(layers.Flatten())
Resnet_Model.add(layers.Dense(512, activation = 'relu'))
Resnet_Model.add(layers.Dense(4, activation = 'softmax'))
```

In [19]:

```
Resnet_Model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
resnet50 (Functional)       (None, 8, 8, 2048)        23587712

_________________________________________________________________
flatten (Flatten)           (None, 131072)            0

_________________________________________________________________
dense (Dense)               (None, 512)               67109376

_________________________________________________________________
dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 90,699,140
Trainable params: 67,111,428
Non-trainable params: 23,587,712
_________________________________________________________________
```

In [20]:

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

In [21]:

```
Resnet50_history = Resnet_Model.fit(
    Train_dataset,
    epochs = 4,
    validation_data = Valid_dataset
)
```

```
Epoch 1/4
43/43 [==============================] - 248s 6s/step - loss: 14.0304 - accuracy: 0.8
074 - val_loss: 0.5540 - val_accuracy: 0.9708
Epoch 2/4
43/43 [==============================] - 238s 6s/step - loss: 0.1754 - accuracy: 0.99
11 - val_loss: 0.3438 - val_accuracy: 0.9766
Epoch 3/4
43/43 [==============================] - 224s 5s/step - loss: 0.0656 - accuracy: 0.99
26 - val_loss: 0.0202 - val_accuracy: 0.9942
Epoch 4/4
43/43 [==============================] - 226s 5s/step - loss: 0.0531 - accuracy: 0.99
63 - val_loss: 4.4636e-05 - val_accuracy: 1.0000
```

In [25]:

```
Resnet50_Score = Resnet_Model.evaluate(Test_dataset)
Resnet50_Score
```

```
5/5 [==============================] - 22s 5s/step - loss: 9.4781e-06 - accuracy: 1.0
000
```

Out[25]:

```
[9.47814442042727e-06, 1.0]
```

VGG16 Model

In [26]:

```
VGG16 = tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
```

```
    input_shape=(256,256,3)
)
```

In [27]:

```
VGG16_Model = Sequential()
VGG16.trainable = False

VGG16_Model.add(VGG16)
VGG16_Model.add(layers.Flatten())
VGG16_Model.add(layers.Dense(512, activation = 'relu'))
VGG16_Model.add(layers.Dense(4, activation = 'softmax'))
```

In [28]:

```
VGG16_Model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Functional)           (None, 8, 8, 512)         14714688  
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0         
_________________________________________________________________
dense_2 (Dense)              (None, 512)               16777728  
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 2052      
=================================================================
Total params: 31,494,468
Trainable params: 16,779,780
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [29]:

```
VGG16_Model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']
)
```

In [30]:

```
VGG16_history = VGG16_Model.fit(
    Train_dataset,
    epochs = 4,
    validation_data = Valid_dataset
)
```

```
Epoch 1/4
43/43 [==============================] - 650s 15s/step - loss: 9.9458 - accuracy: 0.8
170 - val_loss: 0.0786 - val_accuracy: 0.9942
Epoch 2/4
43/43 [==============================] - 614s 14s/step - loss: 0.2021 - accuracy: 0.9
838 - val_loss: 0.0826 - val_accuracy: 0.9825
Epoch 3/4
43/43 [==============================] - 597s 14s/step - loss: 0.0759 - accuracy: 0.9
919 - val_loss: 0.2390 - val_accuracy: 0.9766
Epoch 4/4
43/43 [==============================] - 777s 18s/step - loss: 0.0469 - accuracy: 0.9
956 - val_loss: 0.2302 - val_accuracy: 0.9825
```

In [38]:

```
VGG16_Score = VGG16_Model.evaluate(Test_dataset)
VGG16_Score
```

```
5/5 [==============================] - 70s 14s/step - loss: 0.3428 - accuracy: 0.9875
```

Out[38]:

```
[0.34280914068222046, 0.987500011920929]
```

Xception

In [31]:

```
Xception = tf.keras.applications.Xception(
    include_top=False,
    weights="imagenet",
    input_shape=(256,256,3)
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xc
eption/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 1303s 16us/step
83697664/83683744 [==============================] - 1303s 16us/step
```

In [34]:

```
Xception_Model = Sequential()
Xception.trainable = False

Xception_Model.add(Xception)
Xception_Model.add(layers.Flatten())
Xception_Model.add(layers.Dense(512, activation = "relu"))
Xception_Model.add(layers.Dense(4, activation = "softmax"))
```

In [35]:

```
Xception_Model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Functional)        (None, 8, 8, 2048)        20861480
_________________________________________________________________
flatten_2 (Flatten)          (None, 131072)            0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               67109376
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 2052
=================================================================
Total params: 87,972,908
Trainable params: 67,111,428
Non-trainable params: 20,861,480
_________________________________________________________________
```

In [36]:

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

In [37]:

```
Xception_Model_history = VGG16_Model.fit(
    Train_dataset,
    epochs = 4,
    validation_data = Valid_dataset
)
```

```
Epoch 1/4
43/43 [==============================] - 662s 15s/step - loss: 0.0469 - accuracy: 0.9
926 - val_loss: 0.4551 - val_accuracy: 0.9825
Epoch 2/4
43/43 [==============================] - 626s 15s/step - loss: 0.3164 - accuracy: 0.9
867 - val_loss: 0.5010 - val_accuracy: 0.9942
Epoch 3/4
43/43 [==============================] - 616s 14s/step - loss: 0.0244 - accuracy: 0.9
985 - val_loss: 0.5877 - val_accuracy: 0.9883
Epoch 4/4
43/43 [==============================] - 689s 16s/step - loss: 0.0325 - accuracy: 0.9
970 - val_loss: 0.8113 - val_accuracy: 0.9942
```

In [40]:

```
Xception_Score = Xception_Model.evaluate(Test_dataset)
```

```
Xception_Score
```

```
5/5 [==============================] - 24s 5s/step - loss: 13.8336 - accuracy: 0.2313
```

Out[40]:
```
[13.833582878112793, 0.23125000298023224]
```

InceptionV3

In [10]:
```
InceptionV3 = tf.keras.applications.InceptionV3(
    include_top=False,
    weights="imagenet",
    input_shape=(256,256,3)
)
```

```
A local file was found, but it seems to be incomplete or outdated because the auto fi
le hash does not match the original value of bcbd6486424b2319ff4ef7d526e38f63 so we w
ill re-download the data.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/in
ception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 510s 6us/step
87924736/87910968 [==============================] - 510s 6us/step
```

In [11]:
```
InceptionV3_Model = Sequential()
InceptionV3.trainable = False

InceptionV3_Model.add(InceptionV3)
InceptionV3_Model.add(layers.Flatten())
InceptionV3_Model.add(layers.Dense(512, activation = "relu"))
InceptionV3_Model.add(layers.Dense(4, activation = "softmax"))
```

In [13]:
```
InceptionV3_Model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 6, 6, 2048)        21802784
_________________________________________________________________
flatten (Flatten)            (None, 73728)             0
_________________________________________________________________
dense (Dense)                (None, 512)               37749248
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 2052
=================================================================
Total params: 59,554,084
Trainable params: 37,751,300
Non-trainable params: 21,802,784
_________________________________________________________________
```

In [15]:
```
InceptionV3_Model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']
)
```

In [16]:
```
InceptionV3_Model_history = InceptionV3_Model.fit(
    Train_dataset,
    epochs = 4,
    validation_data = Valid_dataset
)
```

```
Epoch 1/4
43/43 [==============================] - 119s 3s/step - loss: 296.0261 - accuracy: 0.
5160 - val_loss: 183.7151 - val_accuracy: 0.5521
Epoch 2/4
43/43 [==============================] - 107s 2s/step - loss: 80.9226 - accuracy: 0.7
```

```
129 - val_loss: 40.2923 - val_accuracy: 0.8490
Epoch 3/4
43/43 [==============================] - 102s 2s/step - loss: 67.9309 - accuracy: 0.7
528 - val_loss: 16.9933 - val_accuracy: 0.8906
Epoch 4/4
43/43 [==============================] - 335s 8s/step - loss: 29.1202 - accuracy: 0.8
365 - val_loss: 44.1854 - val_accuracy: 0.7836
```

In [18]:

```
InceptionV3_Model_Score = InceptionV3_Model.evaluate(Test_dataset)
InceptionV3_Model_Score
```

```
5/5 [==============================] - 10s 1s/step - loss: 46.2853 - accuracy: 0.7875
```

Out[18]: [46.28532409667969, 0.7875000238418579]

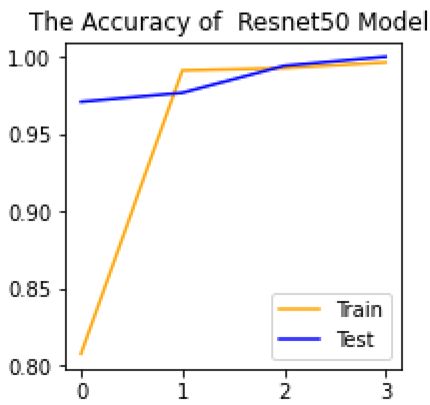
Visualize The Loss and Accuracy of Train and valid data

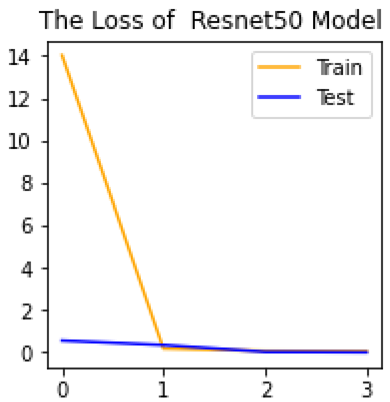
In [ ]:

```
import seaborn as sb
```

In [57]:

```
plt.figure(figsize = (3,3))
plt.plot(range(4),Resnet50_history.history['accuracy'], color = 'orange', label = 'Tr
plt.plot(range(4),Resnet50_history.history['val_accuracy'], color = 'blue', label = '
plt.title('The Accuracy of  Resnet50 Model ')
plt.legend()
plt.show()
plt.figure(figsize = (3,3))
plt.plot(range(4),Resnet50_history.history['loss'], color = 'orange', label = 'Train'
plt.plot(range(4),Resnet50_history.history['val_loss'], color = 'blue', label = 'Test
plt.title('The Loss of  Resnet50 Model ')
plt.legend()
plt.show()
```



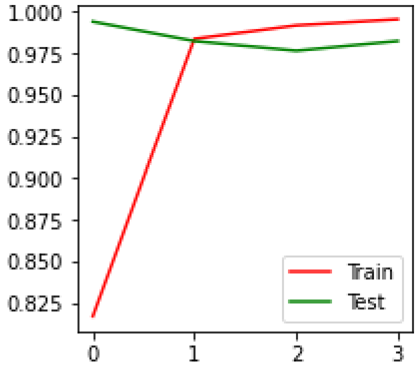


In [58]:

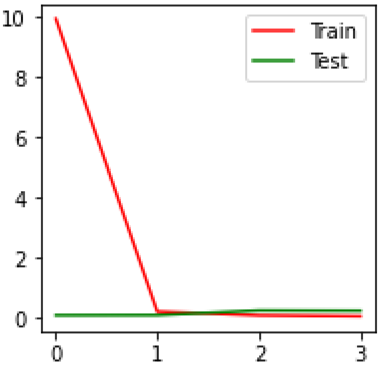
```
plt.figure(figsize = (3,3))
plt.plot(range(4),VGG16_history.history['accuracy'], color = 'red', label = 'Train')
plt.plot(range(4),VGG16_history.history['val_accuracy'], color = 'green', label = 'Te
plt.title('The Accuracy of the VGG16 Model Train and Valid Data ')
plt.legend()
plt.show()
plt.figure(figsize = (3,3))
```

```
plt.plot(range(4),VGG16_history.history['loss'], color = 'red', label = 'Train')
plt.plot(range(4),VGG16_history.history['val_loss'], color = 'green', label = 'Test')
plt.title('The Loss of the VGG16 Model Train and Valid Data ')
plt.legend()
plt.show()
```

The Accuracy of the VGG16 Model Train and Valid Data



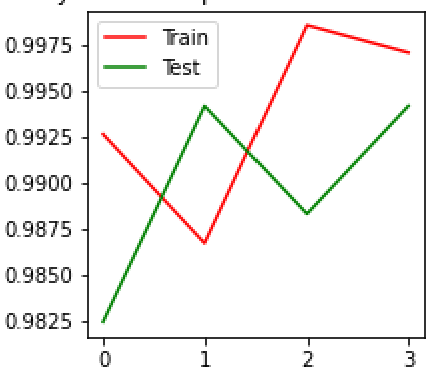
The Loss of the VGG16 Model Train and Valid Data



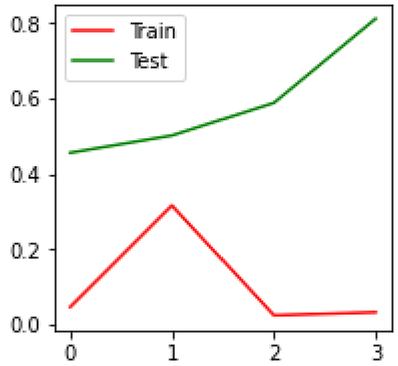
In [59]:

```
plt.figure(figsize = (3,3))
plt.plot(range(4),Xception_Model_history.history['accuracy'], color = 'red', label =
plt.plot(range(4),Xception_Model_history.history['val_accuracy'], color = 'green', la
plt.title('The Accuracy of the Xception Model Train and Valid Data ')
plt.legend()
plt.show()
plt.figure(figsize = (3,3))
plt.plot(range(4),Xception_Model_history.history['loss'], color = 'red', label = 'Tra
plt.plot(range(4),Xception_Model_history.history['val_loss'], color = 'green', label
plt.title('The Loss of the Xception Model Train and Valid Data ')
plt.legend()
plt.show()
```

The Accuracy of the Xception Model Train and Valid Data



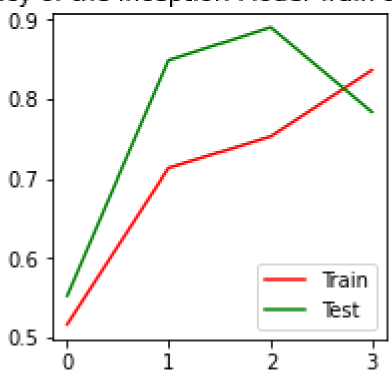
The Loss of the Xception Model Train and Valid Data



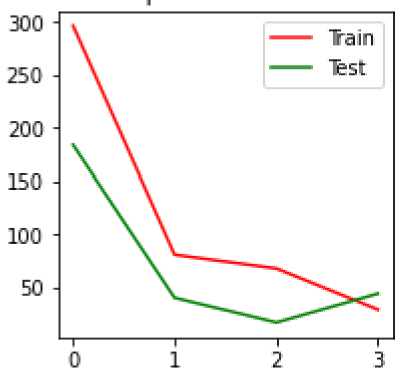
In [19]:

```
plt.figure(figsize = (3,3))
plt.plot(range(4),InceptionV3_Model_history.history['accuracy'], color = 'red', label
plt.plot(range(4),InceptionV3_Model_history.history['val_accuracy'], color = 'green',
plt.title('The Accuracy of the Inception Model Train and Valid Data ')
plt.legend()
plt.show()
plt.figure(figsize = (3,3))
plt.plot(range(4),InceptionV3_Model_history.history['loss'], color = 'red', label = '
plt.plot(range(4),InceptionV3_Model_history.history['val_loss'], color = 'green', lab
plt.title('The Loss of the Inception Model Train and Valid Data ')
plt.legend()
plt.show()
```

The Accuracy of the Inception Model Train and Valid Data



The Loss of the Inception Model Train and Valid Data



In [ ]: