```python
import datetime
cur_date=datetime.datetime.now()
print("current date and time",cur_date)
```

```
current date and time 2024-02-18 14:02:27.443402
```

```python
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from keras.models import Sequential

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import matplotlib.pyplot as plt
import numpy as np
import random
import cv2
import os

from imutils import paths
imagePaths = sorted(list(paths.list_images("F:\\Paper6_vit+cnn\\
Groundnut_Leaf_dataset\\train1")))
# random shuffle
random.seed(42)
random.shuffle(imagePaths)

data = []
labels = []
image_dims = (224, 224, 3)

for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (image_dims[1], image_dims[0]))
    image = img_to_array(image)
    data.append(image)
    l = label = imagePath.split(os.path.sep)[-2].split("_")
    labels.append(l)

data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
print("{} images ({:.2f}MB)".format(len(imagePaths), data.nbytes /
(1024 * 1000.0)))
```

```
7910 images (9302.16MB)
```

```python
data = np.array(data)
label = np.array(labels)
print(data.shape)
```

```
(7910, 224, 224, 3)
```

```python
mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)
# total 4 labels
print("class labels:")
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i + 1, label))
```

```
class labels:
1. ELS
2. ER
3. HL
4. LLS
5. ND
6. RUST
```

```python
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
def MobileNetV2_model(learning_rate, input_shape,class_number):
    baseModel = MobileNetV2(include_top=False,
input_tensor=Input(shape=input_shape))
    for layer in baseModel.layers[:-4]:
        layer.trainable = False

    model = Sequential()

    model.add(baseModel)
    model.add(Conv2D(32,3,padding="same", activation="relu",
input_shape=(128,128,3)))
    model.add(AveragePooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(512, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(50, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(class_number, activation='softmax'))

    return model
```

```python
bs = 36 # over(32,)
lr = 0.1 #over (0.0001,)
size = (224, 224)
shape = (224,224, 3) #224,224,3
epochs = 30
class_number = 6
```

```
model = MobileNetV2_model(lr,shape,class_number)
model.compile(loss= "categorical_crossentropy", metrics=["accuracy"],
optimizer="adam")
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows`
is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224)
will be loaded as the default.
```

```
trainX, testX, trainY, testY = train_test_split(data, labels,
test_size=0.40) # (0.20)
```

```
print("[INFO] training ...")
H = model.fit(trainX, trainY,
batch_size=42,steps_per_epoch=len(trainX) // 26,
     validation_data=(testX, testY),  validation_steps=len(testX) //
26,   epochs=15)
```

```
[INFO] training ...
Epoch 1/15
182/182 [==============================] - ETA: 0s - loss: 0.7694 -
accuracy: 0.7199WARNING:tensorflow:Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches (in this case,
121 batches). You may need to use the repeat() function when building
your dataset.
182/182 [==============================] - 220s 1s/step - loss: 0.7694
- accuracy: 0.7199 - val_loss: 0.4897 - val_accuracy: 0.8461
Epoch 2/15
182/182 [==============================] - 149s 820ms/step - loss:
0.3098 - accuracy: 0.9019
Epoch 3/15
182/182 [==============================] - 155s 852ms/step - loss:
0.1837 - accuracy: 0.9457
Epoch 4/15
182/182 [==============================] - 163s 894ms/step - loss:
0.1278 - accuracy: 0.9645
Epoch 5/15
182/182 [==============================] - 163s 895ms/step - loss:
0.1266 - accuracy: 0.9682
Epoch 6/15
182/182 [==============================] - 162s 892ms/step - loss:
0.0704 - accuracy: 0.9770
Epoch 7/15
182/182 [==============================] - 164s 901ms/step - loss:
0.0680 - accuracy: 0.9812
Epoch 8/15
182/182 [==============================] - 163s 894ms/step - loss:
0.0324 - accuracy: 0.9922
Epoch 9/15
182/182 [==============================] - 163s 894ms/step - loss:
```

```
0.0334 - accuracy: 0.9895
Epoch 10/15
 57/182 [========>.....................] - ETA: 1:49 - loss: 0.1398 -
accuracy: 0.9724WARNING:tensorflow:Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches (in this case,
2730 batches). You may need to use the repeat() function when building
your dataset.
182/182 [==============================] - 50s 272ms/step - loss:
0.1398 - accuracy: 0.9724
```

```
model.summary()
```

```
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=16)

# for each image in the testing set we need to find the index of the
label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1),
predIdxs,target_names=mlb.classes_))
```

```
[INFO] evaluating network...
198/198 [==============================] - 60s 298ms/step
              precision    recall  f1-score   support

         ELS       0.92      0.96      0.94       506
          ER       1.00      0.97      0.98       445
          HL       0.85      0.99      0.91       570
         LLS       0.96      0.94      0.95       605
          ND       0.99      0.81      0.89       506
        RUST       0.97      1.00      0.99       532

    accuracy                           0.94      3164
   macro avg       0.95      0.94      0.94      3164
weighted avg       0.95      0.94      0.94      3164
```

```
# from sklearn.metrics import confusion_matrix
# print(confusion_matrix(testY, predIdxs))
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
precision_score(testY.argmax(axis=1), predIdxs, average='micro')
```

```
0.9424778761061947
```

```
recall = recall_score(testY.argmax(axis=1), predIdxs, average='macro')
print('recall=',recall)
```

```
recall= 0.9417894453890064
```

```
from sklearn.metrics import f1_score
f1_score(testY.argmax(axis=1), predIdxs, average='micro')
```

```
0.9424778761061947
```

```
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import balanced_accuracy_score
a=matthews_corrcoef(testY.argmax(axis=1), predIdxs)
b=balanced_accuracy_score(testY.argmax(axis=1), predIdxs)
print(a)
print(b)
```

```
0.9318055621745941
0.9417894453890064
```

```
# N = 14
# plt.style.use("ggplot")
# plt.figure()
# #plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
# #plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
# plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
# plt.plot(np.arange(0, N), H.history["val_accuracy"],
label="val_acc")
# plt.title("Training acc and validation acc")
# plt.xlabel("Epoch #")
# plt.ylabel("Accuracy")
# plt.legend(loc="lower left")
```

```
# N = 10
# plt.style.use("ggplot")
# plt.figure()
# plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
# plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
# #plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
# #plt.plot(np.arange(0, N), H.history["val_accuracy"],
label="val_acc")
# plt.title("Training Loss and validation loss")
# plt.xlabel("Epoch #")
# plt.ylabel("Loss")
# plt.legend(loc="lower left")
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(testY.argmax(axis=1), predIdxs))
```

```
[[485   0   8   8   0   5]
 [  3 430   4   5   0   3]
 [  4   0 562   0   4   0]
 [ 26   2   6 566   0   5]
```

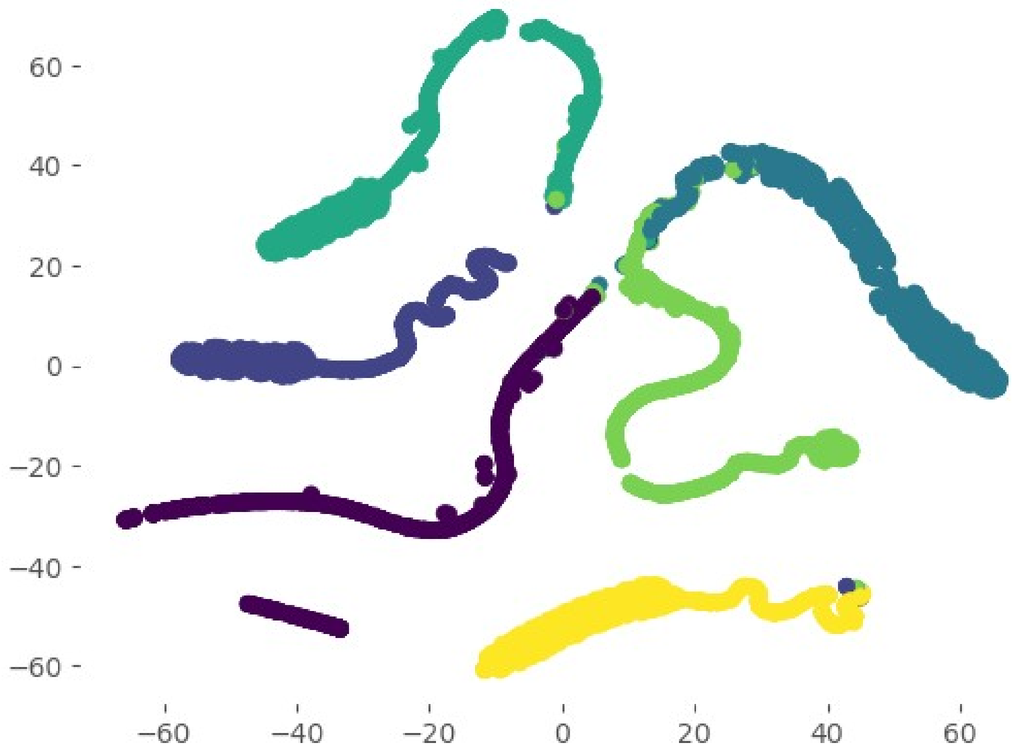
```
 [  8   0  80   9 408   1]
 [  0   0   0   1   0 531]]
```

```
# #Import the necessary libraries
# import numpy as np
# from sklearn.metrics import confusion_matrix
# import seaborn as sns
# import matplotlib.pyplot as plt
# plt.figure(figsize=(4, 4))
# #compute the confusion matrix.
# cm = confusion_matrix(testY.argmax(axis=1), predIdxs)

# #Plot the confusion matrix.
# sns.heatmap(cm/np.sum(cm,axis=1).reshape(-
1,1),cmap='Blues',annot=True,fmt='.2%',xticklabels=['NK','NP','PK'],yt
icklabels=['NK','NP','PK'])
# plt.xticks(fontsize=12)
# plt.yticks(fontsize=12)
# plt.ylabel('Prediction',fontsize=12)
# plt.xlabel('Actual',fontsize=12)
# plt.title('Confusion Matrix',fontsize=12)
#
#plt.savefig('F:/paper_3_code_files_final/results/results1/DECM_CM.jpg
',dpi=600)
# plt.show()
```

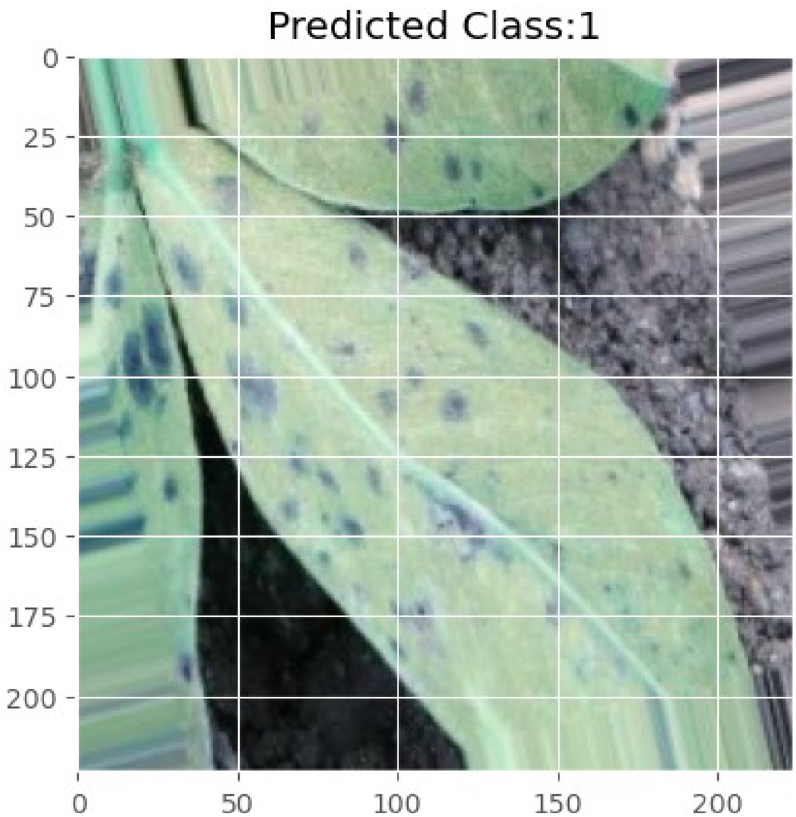
```
from sklearn.manifold import TSNE
# Extract the features from the last layer of the CNN
features = model.predict(trainX)
# Reduce the dimensionality of the features using t-SNE
tsne = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0,
learning_rate=200.0, n_iter=1000, n_iter_without_progress=300,
min_grad_norm=1e-07, metric='euclidean', metric_params=None,
init='pca', verbose=0, random_state=None, method='barnes_hut',
angle=0.5, n_jobs=None)
features_embedded = tsne.fit_transform(features)
```

```
149/149 [==============================] - 92s 620ms/step
```

```
# Plot the results
ax=plt.axes()
ax.set_facecolor("white")
plt.scatter(features_embedded[:, 0], features_embedded[:, 1],
c=trainY.argmax(axis=1))
#plt.axis('off')
plt.savefig('F:/Paper6_vit+cnn/results/updated_results/P6_DECM_TSNE.jp
g',dpi=600)
plt.show()
```




```
import lime
from lime import lime_image

# Choose an image from the test set for explanation
image_index = 5
img = testX[image_index]
img_array = np.expand_dims(img, axis=0)

# Make predictions
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions)

# Display the original image and predictions
plt.imshow(img.reshape(224, 224,3))
plt.title(f'Predicted Class:{predicted_class}')
plt.show()

1/1 [==============================] - 0s 41ms/step
```




```
# Create a LIME explainer for image classification
explainer = lime_image.LimeImageExplainer()

# Explain predictions
explanation = explainer.explain_instance(img_array[0], model.predict,
top_labels=1, hide_color=0, num_samples=1000)
```

{"model_id":"99de1b8cb02140e99c7feaf72e523d30","version_major":2,"vers
ion_minor":0}

```
1/1 [==============================] - 0s 206ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 219ms/step
1/1 [==============================] - 0s 225ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 204ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 229ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 250ms/step
```

```
1/1 [==============================] - 0s 235ms/step
1/1 [==============================] - 0s 224ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 213ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 172ms/step
```

```
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 161ms/step
```

```
# explain predictions
explanation=explainer.explain_instance(img_array[0],model.predict,top_
labels=1,hide_color=0,num_samples=1000)
```

{"model_id":"71b26da6173041e4b57ccfd9a377479f","version_major":2,"vers
ion_minor":0}

```
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 152ms/step
```

```
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 232ms/step
1/1 [==============================] - 0s 216ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 250ms/step
1/1 [==============================] - 0s 251ms/step
1/1 [==============================] - 0s 217ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 200ms/step
1/1 [==============================] - 0s 204ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 213ms/step
1/1 [==============================] - 0s 220ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 171ms/step
```

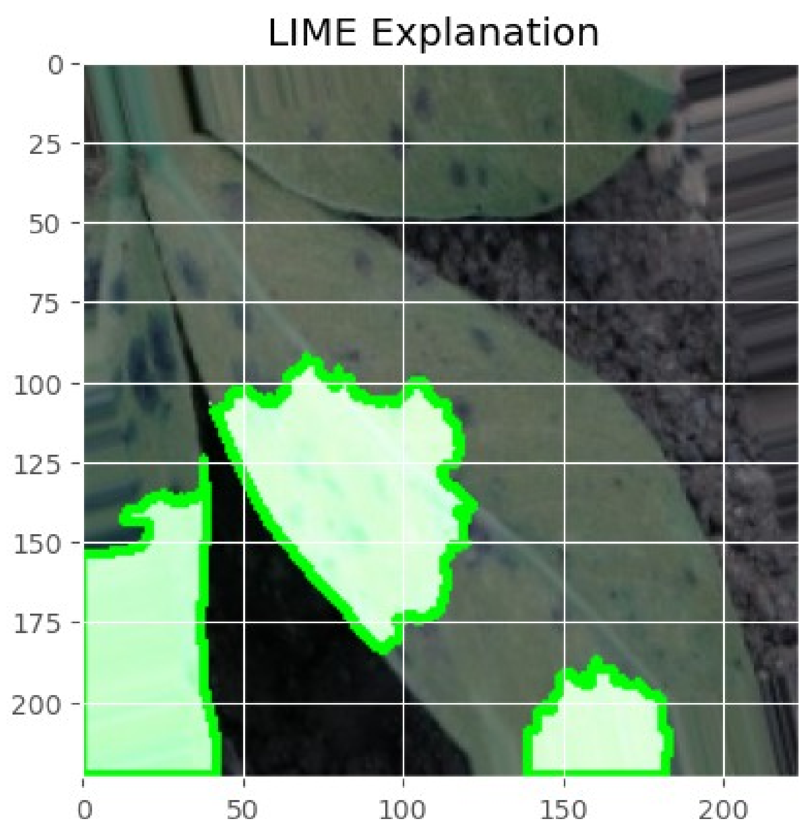
```
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 233ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 206ms/step
1/1 [==============================] - 0s 221ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 155ms/step
```

```python
#visualise LIME explanations with cv2.polylines
temp,mask=explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=False,num_features=5,hide_rest=False)
```

```python
#convert temp to uint8 for cv2.polylines
temp=(temp/2+0.5*mask[:,:,np.newaxis])*255
temp=temp.astype(np.uint8)

#find contours in the mask
contours,_=cv2.findContours(mask.astype(np.uint8),cv2.RETR_EXTERNAL,cv
2.CHAIN_APPROX_SIMPLE)

#Draw contours on the image
cv2.polylines(temp,contours,isClosed=False,color=(0,255,0),thickness=2
)
```

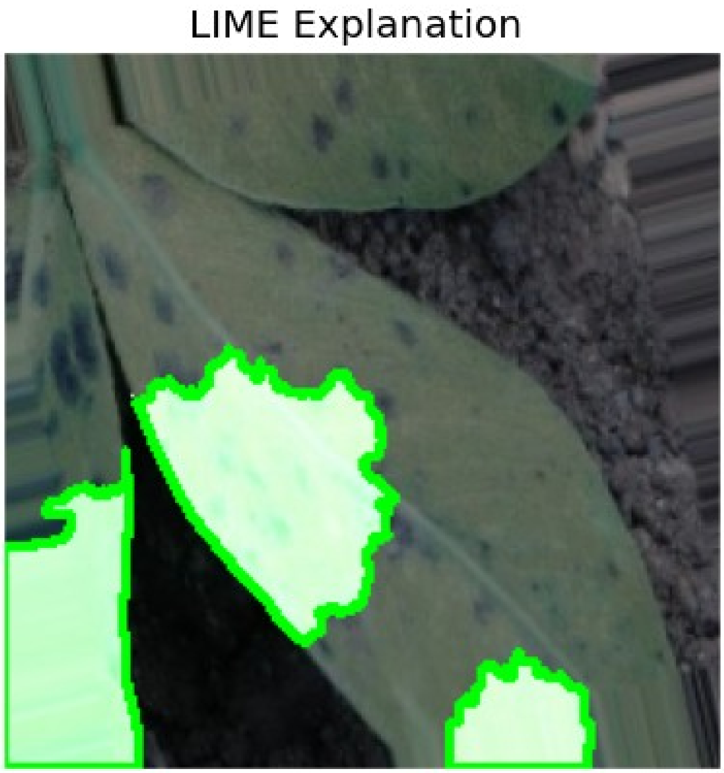
```
array([[[ 47,  53,  47],
        [ 55,  64,  57],
        [ 74,  88,  80],
        ...,
        [ 35,  39,  43],
        [ 35,  38,  43],
        [ 33,  37,  41]],

       [[ 53,  59,  53],
        [ 50,  59,  52],
        [ 73,  86,  78],
        ...,
        [ 37,  40,  44],
        [ 38,  40,  45],
        [ 37,  40,  45]],

       [[ 61,  66,  60],
        [ 46,  53,  47],
        [ 69,  82,  74],
        ...,
        [ 40,  42,  46],
        [ 41,  42,  47],
        [ 41,  42,  47]],

       ...,

       [[  0, 255,   0],
        [  0, 255,   0],
        [180, 254, 198],
        ...,
        [ 25,  27,  30],
        [ 57,  58,  62],
        [ 28,  29,  34]],

       [[  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0],
        ...,
```

```
        [ 26,  29,  31],
        [ 50,  51,  56],
        [ 35,  37,  41]],

       [[  0, 255,   0],
        [  0, 255,   0],
        [  0, 255,   0],
        ...,
        [ 27,  30,  32],
        [ 41,  43,  47],
        [ 42,  43,  48]]], dtype=uint8)

#display the image
plt.imshow(temp)
plt.title('LIME Explanation ')
#plt.savefig('F:/Paper6_vit+cnn/results/P6_proposed_lime1.jpg',dpi=600
)
plt.show()
```




```
# Code for turning off x and y labels
plt.imshow(temp)
plt.title('LIME Explanation ')
plt.xticks([])  # Turn off x labels
plt.yticks([])  # Turn off y labels
```

```
plt.savefig('F:/Paper6_vit+cnn/results/P6_DECM_lime.jpg', dpi=600)
plt.show()
```



```
import datetime
cur_date=datetime.datetime.now()
print("current date and time",cur_date)
```

```
current date and time 2024-02-18 15:06:06.701079
```