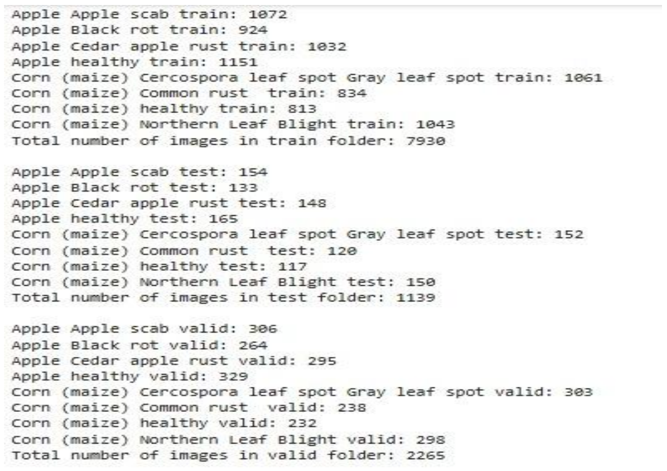
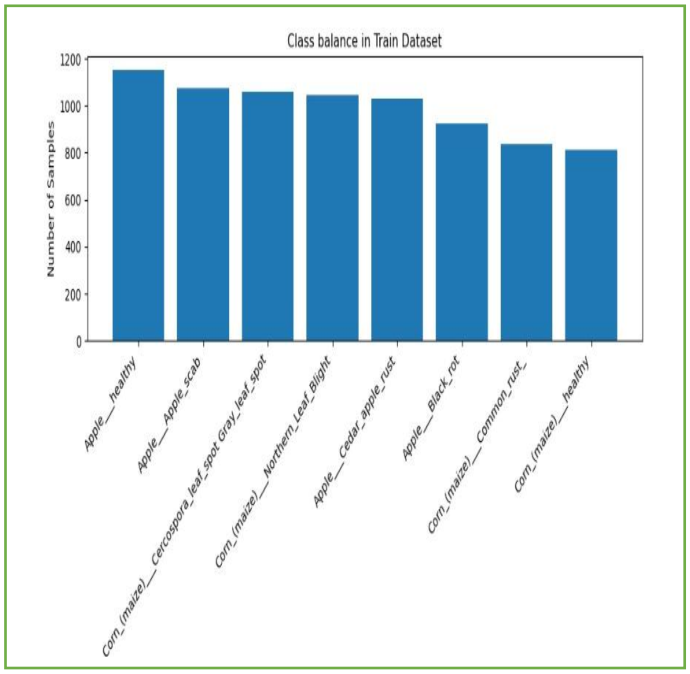
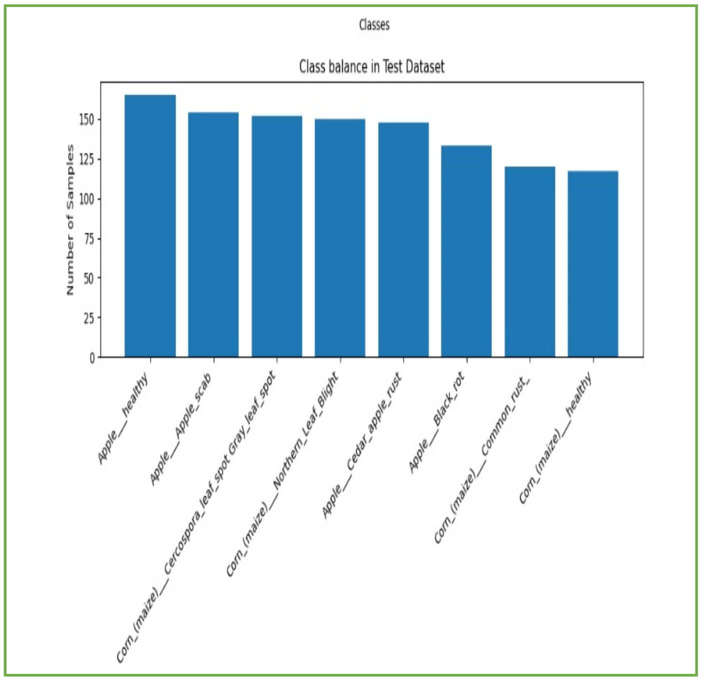
Visualizing Crop Disease Detection: Exploring ResNet and Custom CNN Models with XGradCAM for Enhanced Interpretability

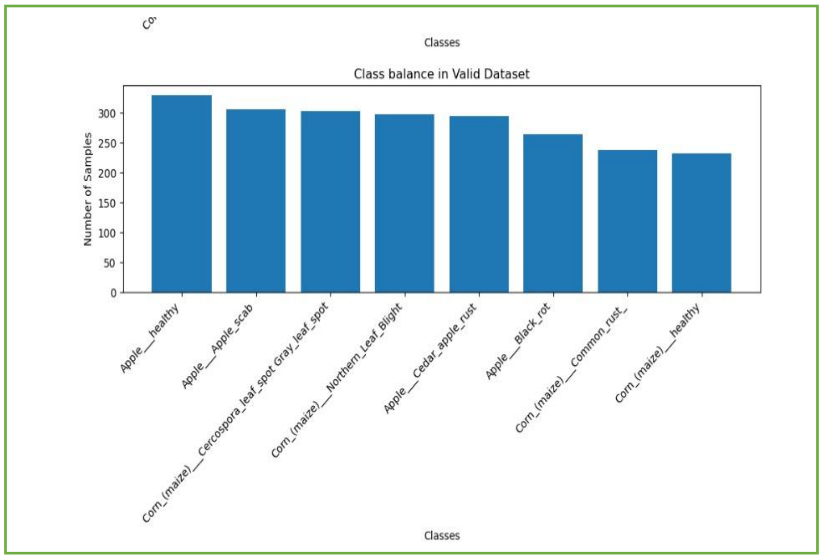
Dataset:



```
Apple Apple scab train: 1072
Apple Black rot train: 924
Apple Cedar apple rust train: 1032
Apple healthy train: 1151
Corn (maize) Cercospora leaf spot Gray leaf spot train: 1061
Corn (maize) Common rust  train: 834
Corn (maize) healthy train: 813
Corn (maize) Northern Leaf Blight train: 1043
Total number of images in train folder: 7930

Apple Apple scab test: 154
Apple Black rot test: 133
Apple Cedar apple rust test: 148
Apple healthy test: 165
Corn (maize) Cercospora leaf spot Gray leaf spot test: 152
Corn (maize) Common rust  test: 120
Corn (maize) healthy test: 117
Corn (maize) Northern Leaf Blight test: 150
Total number of images in test folder: 1139

Apple Apple scab valid: 306
Apple Black rot valid: 264
Apple Cedar apple rust valid: 295
Apple healthy valid: 329
Corn (maize) Cercospora leaf spot Gray leaf spot valid: 303
Corn (maize) Common rust  valid: 238
Corn (maize) healthy valid: 232
Corn (maize) Northern Leaf Blight valid: 298
Total number of images in valid folder: 2265
```

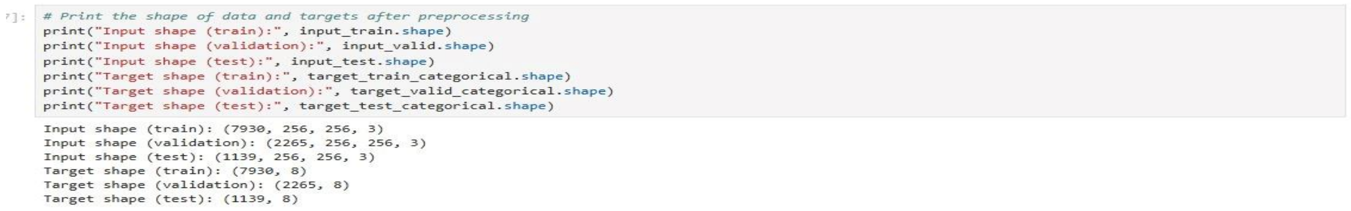




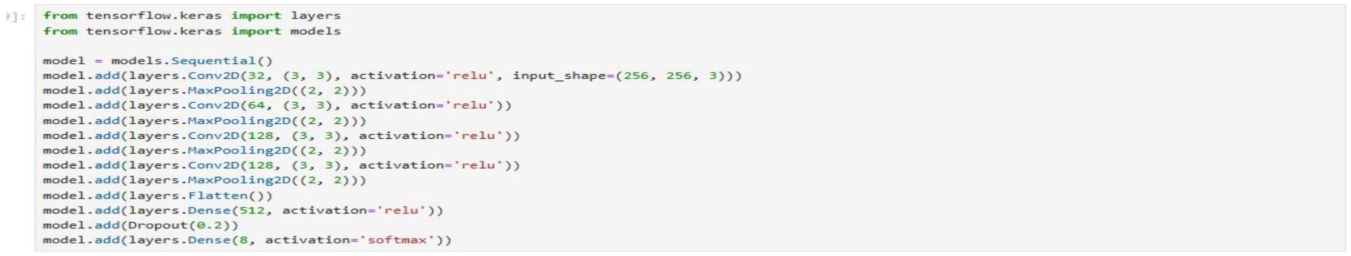


Custom CNN Models

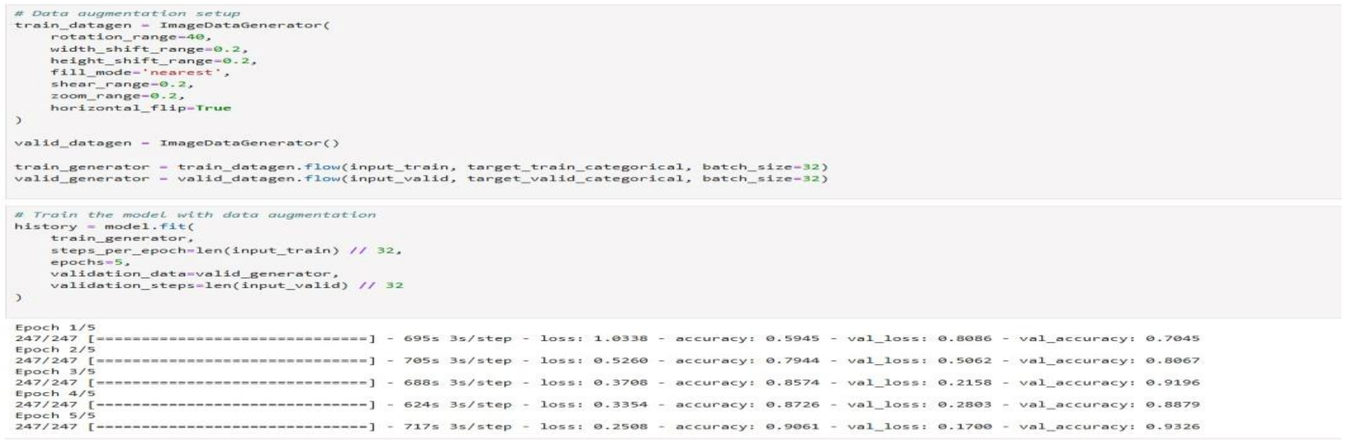
Model Version-6



```
7]: # Print the shape of data and targets after preprocessing
    print("Input shape (train):", input_train.shape)
    print("Input shape (validation):", input_valid.shape)
    print("Input shape (test):", input_test.shape)
    print("Target shape (train):", target_train_categorical.shape)
    print("Target shape (validation):", target_valid_categorical.shape)
    print("Target shape (test):", target_test_categorical.shape)

    Input shape (train): (7930, 256, 256, 3)
    Input shape (validation): (2265, 256, 256, 3)
    Input shape (test): (1139, 256, 256, 3)
    Target shape (train): (7930, 8)
    Target shape (validation): (2265, 8)
    Target shape (test): (1139, 8)
```

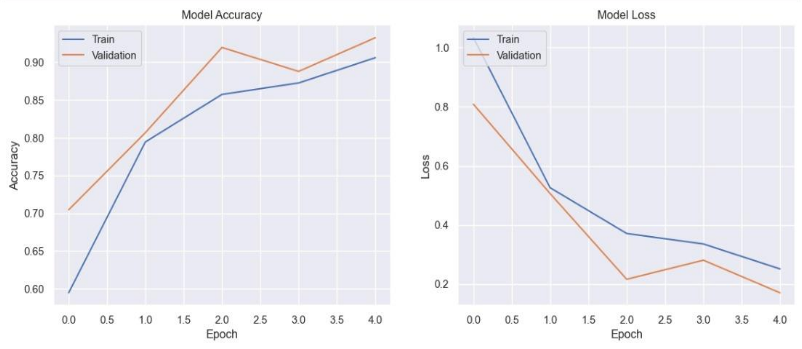


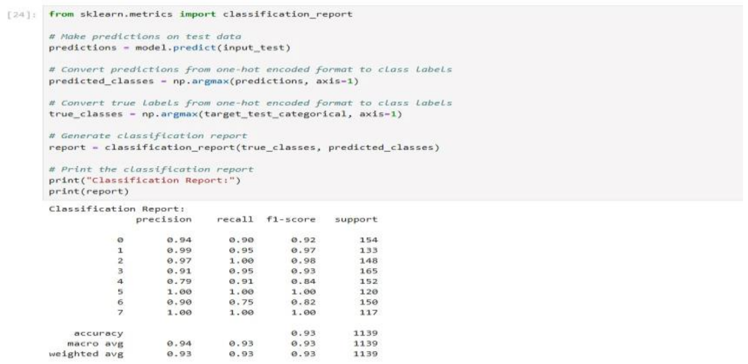
```
8]: from tensorflow.keras import layers
    from tensorflow.keras import models

    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(layers.Dense(8, activation='softmax'))
```



```
# Data augmentation setup
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    fill_mode='nearest',
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

valid_datagen = ImageDataGenerator()

train_generator = train_datagen.flow(input_train, target_train_categorical, batch_size=32)
valid_generator = valid_datagen.flow(input_valid, target_valid_categorical, batch_size=32)
```

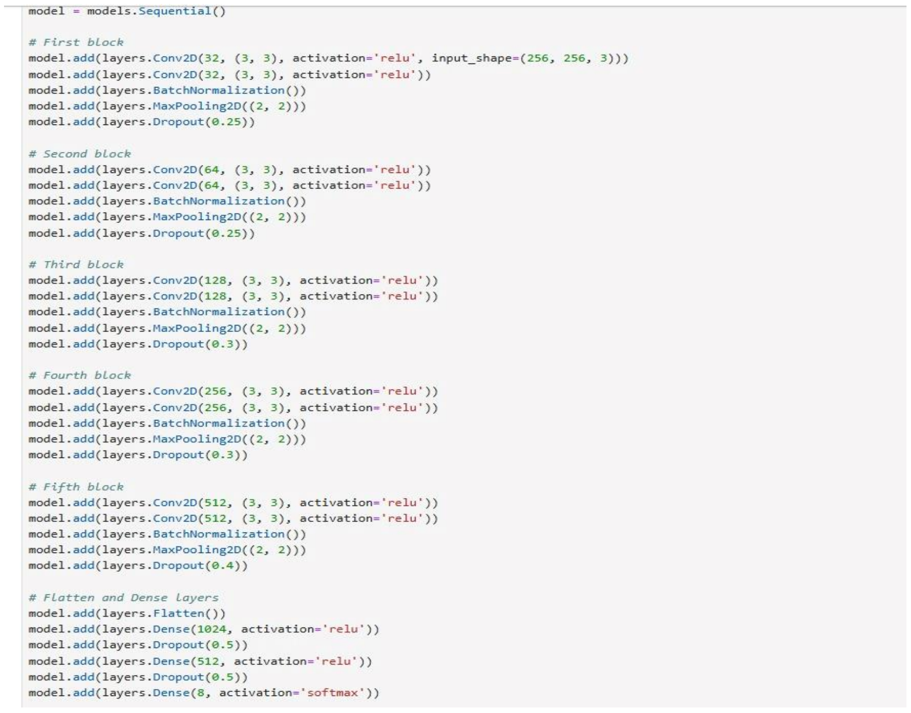
```
# Train the model with data augmentation
history = model.fit(
    train_generator,
    steps_per_epoch=len(input_train) // 32,
    epochs=5,
    validation_data=valid_generator,
    validation_steps=len(input_valid) // 32
)

Epoch 1/5
247/247 [==============================] - 695s 3s/step - loss: 1.0338 - accuracy: 0.5945 - val_loss: 0.8086 - val_accuracy: 0.7045
Epoch 2/5
247/247 [==============================] - 705s 3s/step - loss: 0.5260 - accuracy: 0.7944 - val_loss: 0.5062 - val_accuracy: 0.8067
Epoch 3/5
247/247 [==============================] - 688s 3s/step - loss: 0.3708 - accuracy: 0.8574 - val_loss: 0.2158 - val_accuracy: 0.9196
Epoch 4/5
247/247 [==============================] - 624s 3s/step - loss: 0.3354 - accuracy: 0.8726 - val_loss: 0.2803 - val_accuracy: 0.8879
Epoch 5/5
247/247 [==============================] - 717s 3s/step - loss: 0.2508 - accuracy: 0.9061 - val_loss: 0.1700 - val_accuracy: 0.9326
```



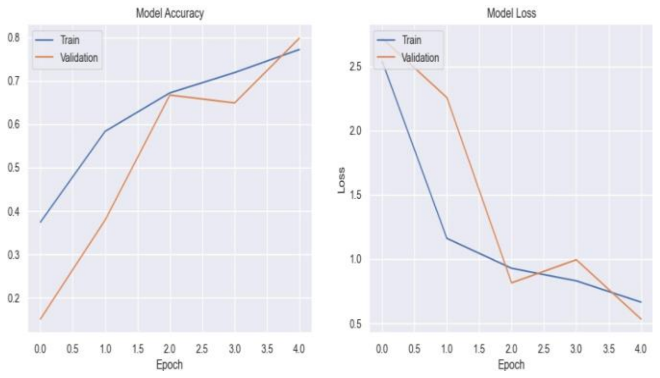


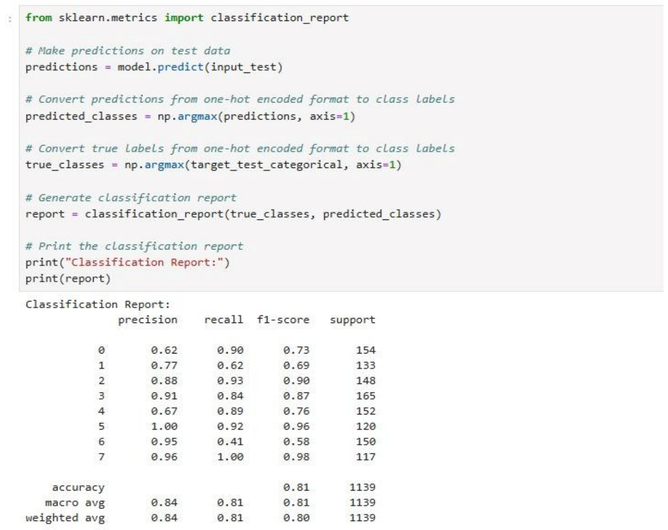
```
[24]: from sklearn.metrics import classification_report

      # Make predictions on test data
      predictions = model.predict(input_test)

      # Convert predictions from one-hot encoded format to class labels
      predicted_classes = np.argmax(predictions, axis=1)

      # Convert true labels from one-hot encoded format to class labels
      true_classes = np.argmax(target_test_categorical, axis=1)

      # Generate classification report
      report = classification_report(true_classes, predicted_classes)

      # Print the classification report
      print("Classification Report:")
      print(report)

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.94      0.90      0.92       154
                 1       0.99      0.95      0.97       133
                 2       0.97      1.00      0.98       148
                 3       0.91      0.95      0.93       165
                 4       0.79      0.91      0.84       152
                 5       1.00      1.00      1.00       120
                 6       0.90      0.75      0.82       150
                 7       1.00      1.00      1.00       117

          accuracy                           0.93      1139
         macro avg       0.94      0.93      0.93      1139
      weighted avg       0.93      0.93      0.93      1139
```

Model Version6.1



```python
model = models.Sequential()

# First block
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

# Second block
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

# Third block
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# Fourth block
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

# Fifth block
model.add(layers.Conv2D(512, (3, 3), activation='relu'))
model.add(layers.Conv2D(512, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

# Flatten and Dense layers
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(8, activation='softmax'))
```





```python
from sklearn.metrics import classification_report

# Make predictions on test data
predictions = model.predict(input_test)

# Convert predictions from one-hot encoded format to class labels
predicted_classes = np.argmax(predictions, axis=1)

# Convert true labels from one-hot encoded format to class labels
true_classes = np.argmax(target_test_categorical, axis=1)

# Generate classification report
report = classification_report(true_classes, predicted_classes)

# Print the classification report
print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.90      0.73       154
           1       0.77      0.62      0.69       133
           2       0.88      0.93      0.90       148
           3       0.91      0.84      0.87       165
           4       0.67      0.89      0.76       152
           5       1.00      0.92      0.96       120
           6       0.95      0.41      0.58       150
           7       0.96      1.00      0.98       117

    accuracy                           0.81      1139
   macro avg       0.84      0.81      0.81      1139
weighted avg       0.84      0.81      0.80      1139
```

Explainable AI (XAI) refers to a set of processes and methods that make the output of artificial intelligence (AI) systems understandable to humans. The primary goal of XAI is to provide clear and interpretable explanations of how AI models make decisions, enabling users to understand, trust, and effectively manage these systems.

Key Objectives of Explainable AI

Clarity: XAI aims to shed light on how AI systems reach their conclusions, making their processes and logic accessible.
Comprehensibility: The explanations provided should be understandable to various stakeholders, including developers, end-users, and regulatory bodies.
Reliability: When users can see and understand the reasoning behind AI decisions, they are more likely to trust the system.
Error Detection: By understanding the decision-making process, developers can identify and rectify errors or biases in the model.
Optimization: Insight into feature importance and model behavior facilitates continuous improvement of AI systems methods of Explainable AI

Method:

Feature Importance: Techniques like permutation feature importance or feature attribution highlight the significance of each input feature in the model's decision.
Local Explanations: Methods like LIME (Local Interpretable Model-agnostic Explanations) create simpler surrogate models around specific predictions to approximate and explain complex model behavior.
Global Explanations: SHAP (SHapley Additive exPlanations) provides a comprehensive view of feature importance based on cooperative game theory, offering insights across all predictions.
Visual Explanations: **Saliency maps, GradCam** and other visualization tools illustrate which parts of the input data (such as image regions) are most influential in the model's predictions.

Is the Gradcam is an explainable AI?

Yes, Grad-CAM (Gradient-weighted Class Activation Mapping) is considered a method within the realm of Explainable AI (XAI). Grad-CAM is specifically designed to provide visual explanations for the decisions made by convolutional neural networks (CNNs), particularly in image classification tasks. By highlighting the regions of an input image that are most influential in the model's prediction for a given class, Grad-CAM makes the model's decision-making process more transparent and interpretable.

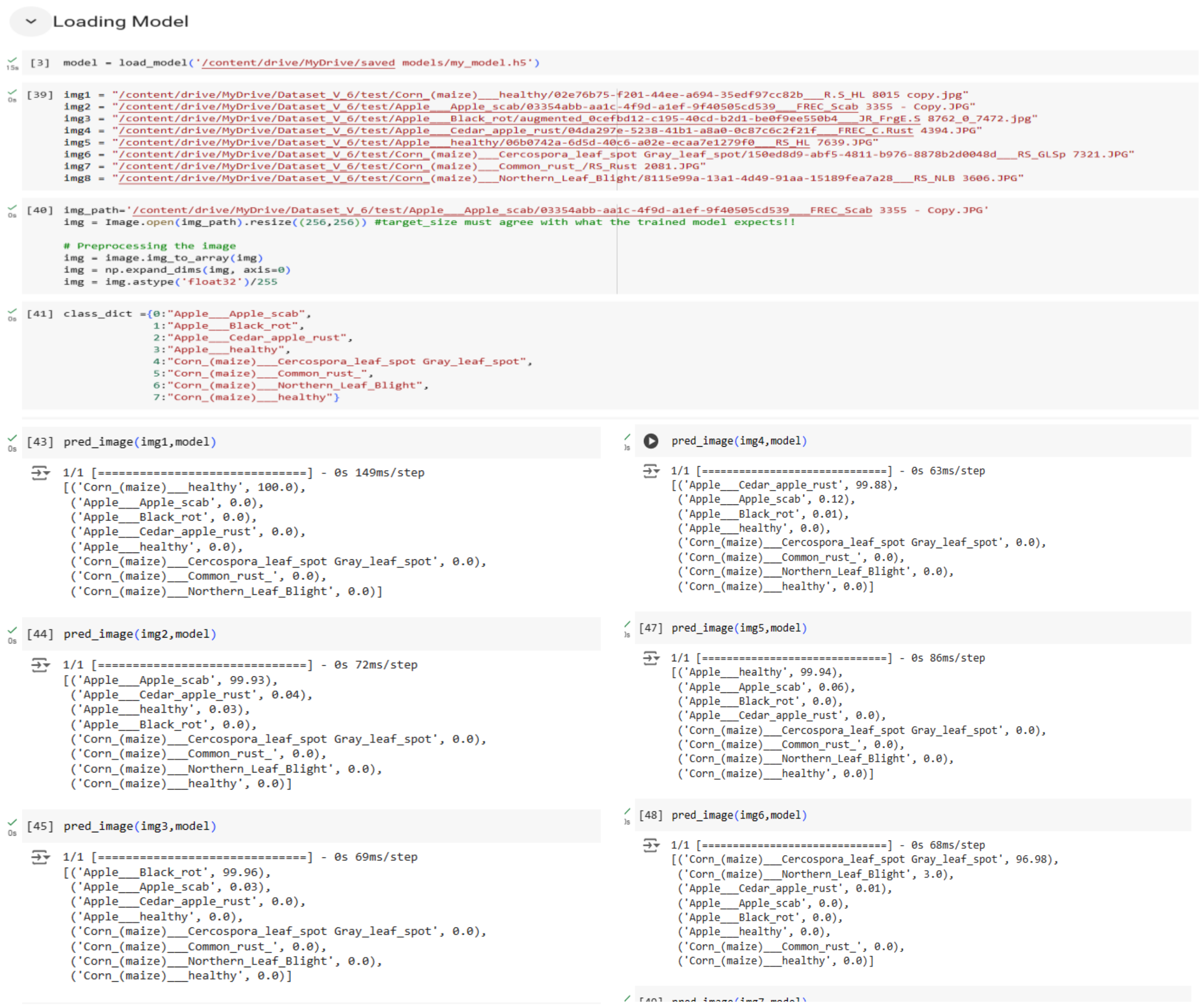
How Grad-CAM Contributes to Explainable AI?

Visual Explanations: Grad-CAM produces heatmaps that show which parts of an image were most influential in the model's prediction. These visual explanations help users understand what the model "sees" when making a decision.
Transparency: By revealing the areas of the input that contributed most to the prediction, Grad-CAM increases the transparency of CNNs, which are often considered "black boxes" due to their complex and non-intuitive decision-making processes.
Trust and Accountability: Visual explanations from Grad-CAM can help build trust in AI systems by providing insights into how they arrive at their conclusions. This is especially important in high-stakes applications such as medical imaging, where understanding the model's reasoning can be crucial.
Bias Detection: Grad-CAM can be used to identify biases in the model by showing which features or regions are being used for predictions. If a model is focusing on irrelevant or biased features, this can be detected and addressed.
Model Debugging and Improvement: Developers can use Grad-CAM to understand failure cases better. By seeing where the model is focusing its attention, they can diagnose and fix problems such as overfitting to background noise or misinterpreting certain features.

Example:

Imagine an AI model trained to identify different species of animals in images. When presented with an image of a tiger, Grad-CAM can generate a heatmap that highlights the stripes and face of the tiger, showing that these features were most important in the model's classification decision. This helps users understand and verify that the model is correctly focusing on the characteristic features of a tiger.

<u>GradCam Implementation in our model:</u>



Loading Model

```
[3] model = load_model('/content/drive/MyDrive/saved models/my_model.h5')
```

```
[39] img1 = "/content/drive/MyDrive/Dataset_V_6/test/Corn_(maize)___healthy/02e76b75-f201-44ee-a694-35edf97cc82b___R.S_HL 8015 copy.jpg"
     img2 = "/content/drive/MyDrive/Dataset_V_6/test/Apple___Apple_scab/03354abb-aa1c-4f9d-a1ef-9f40505cd539___FREC_Scab 3355 - Copy.JPG"
     img3 = "/content/drive/MyDrive/Dataset_V_6/test/Apple___Black_rot/augmented_0cefbd12-c195-40cd-b2d1-be0f9ee550b4___JR_FrgE.S 8762_0_7472.jpg"
     img4 = "/content/drive/MyDrive/Dataset_V_6/test/Apple___Cedar_apple_rust/04da297e-5238-41b1-a8a0-0c87c6c2f21f___FREC_C.Rust 4394.JPG"
     img5 = "/content/drive/MyDrive/Dataset_V_6/test/Apple___healthy/06b0742a-6d5d-40c6-a02e-ecaa7e1279f0___RS_HL 7639.JPG"
     img6 = "/content/drive/MyDrive/Dataset_V_6/test/Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot/150ed8d9-abf5-4811-b976-8878b2d0048d___RS_GLSp 7321.JPG"
     img7 = "/content/drive/MyDrive/Dataset_V_6/test/Corn_(maize)___Common_rust_/RS_Rust 2081.JPG"
     img8 = "/content/drive/MyDrive/Dataset_V_6/test/Corn_(maize)___Northern_Leaf_Blight/8115e99a-13a1-4d49-91aa-15189fea7a28___RS_NLB 3606.JPG"
```

```
[40] img_path='/content/drive/MyDrive/Dataset_V_6/test/Apple___Apple_scab/03354abb-aa1c-4f9d-a1ef-9f40505cd539___FREC_Scab 3355 - Copy.JPG'
     img = Image.open(img_path).resize((256,256)) #target_size must agree with what the trained model expects!!

     # Preprocessing the image
     img = image.img_to_array(img)
     img = np.expand_dims(img, axis=0)
     img = img.astype('float32')/255
```

```
[41] class_dict ={0:"Apple___Apple_scab",
                  1:"Apple___Black_rot",
                  2:"Apple___Cedar_apple_rust",
                  3:"Apple___healthy",
                  4:"Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot",
                  5:"Corn_(maize)___Common_rust_",
                  6:"Corn_(maize)___Northern_Leaf_Blight",
                  7:"Corn_(maize)___healthy"}
```

```
[43] pred_image(img1,model)

1/1 [==============================] - 0s 149ms/step
[('Corn_(maize)___healthy', 100.0),
 ('Apple___Apple_scab', 0.0),
 ('Apple___Black_rot', 0.0),
 ('Apple___Cedar_apple_rust', 0.0),
 ('Apple___healthy', 0.0),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___Northern_Leaf_Blight', 0.0)]
```

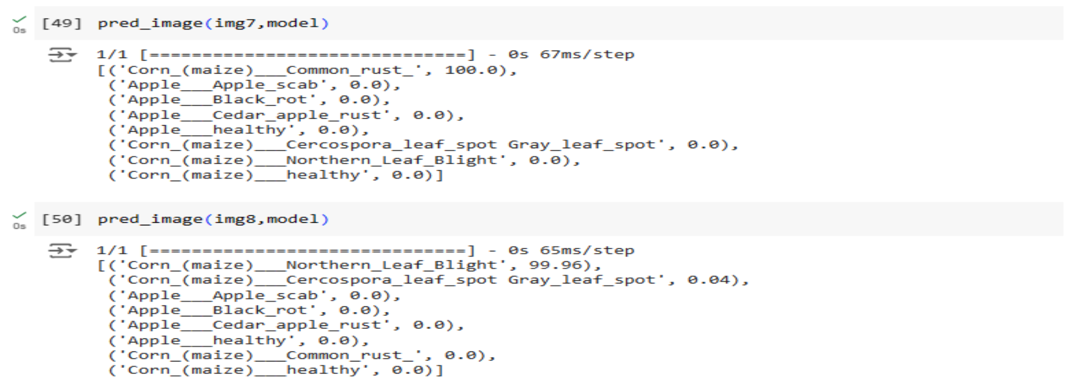
```
[44] pred_image(img2,model)

1/1 [==============================] - 0s 72ms/step
[('Apple___Apple_scab', 99.93),
 ('Apple___Cedar_apple_rust', 0.04),
 ('Apple___healthy', 0.03),
 ('Apple___Black_rot', 0.0),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___Northern_Leaf_Blight', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```

```
[45] pred_image(img3,model)

1/1 [==============================] - 0s 69ms/step
[('Apple___Black_rot', 99.96),
 ('Apple___Apple_scab', 0.03),
 ('Apple___Cedar_apple_rust', 0.0),
 ('Apple___healthy', 0.0),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___Northern_Leaf_Blight', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```

```
pred_image(img4,model)

1/1 [==============================] - 0s 63ms/step
[('Apple___Cedar_apple_rust', 99.88),
 ('Apple___Apple_scab', 0.12),
 ('Apple___Black_rot', 0.01),
 ('Apple___healthy', 0.0),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___Northern_Leaf_Blight', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```

```
[47] pred_image(img5,model)

1/1 [==============================] - 0s 86ms/step
[('Apple___healthy', 99.94),
 ('Apple___Apple_scab', 0.06),
 ('Apple___Black_rot', 0.0),
 ('Apple___Cedar_apple_rust', 0.0),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___Northern_Leaf_Blight', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```

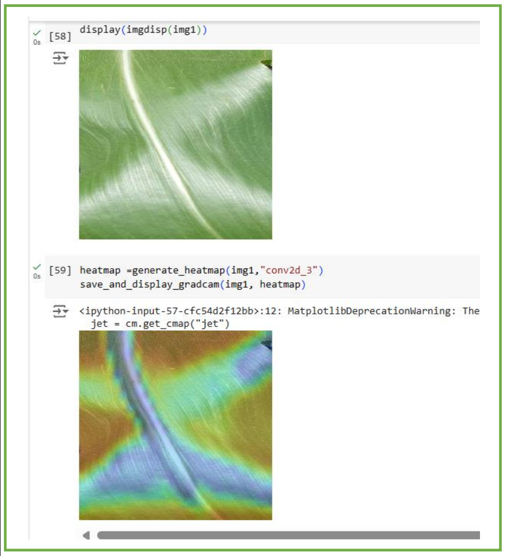
```
[48] pred_image(img6,model)

1/1 [==============================] - 0s 68ms/step
[('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 96.98),
 ('Corn_(maize)___Northern_Leaf_Blight', 3.0),
 ('Apple___Cedar_apple_rust', 0.01),
 ('Apple___Apple_scab', 0.0),
 ('Apple___Black_rot', 0.0),
 ('Apple___healthy', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```



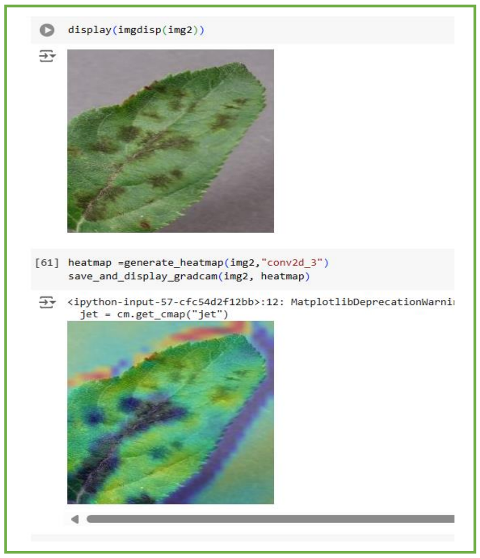
```
[49] pred_image(img7,model)

1/1 [==============================] - 0s 67ms/step
[('Corn_(maize)___Common_rust_', 100.0),
 ('Apple___Apple_scab', 0.0),
 ('Apple___Black_rot', 0.0),
 ('Apple___Cedar_apple_rust', 0.0),
 ('Apple___healthy', 0.0),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.0),
 ('Corn_(maize)___Northern_Leaf_Blight', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```

```
[50] pred_image(img8,model)

1/1 [==============================] - 0s 65ms/step
[('Corn_(maize)___Northern_Leaf_Blight', 99.96),
 ('Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 0.04),
 ('Apple___Apple_scab', 0.0),
 ('Apple___Black_rot', 0.0),
 ('Apple___Cedar_apple_rust', 0.0),
 ('Apple___healthy', 0.0),
 ('Corn_(maize)___Common_rust_', 0.0),
 ('Corn_(maize)___healthy', 0.0)]
```

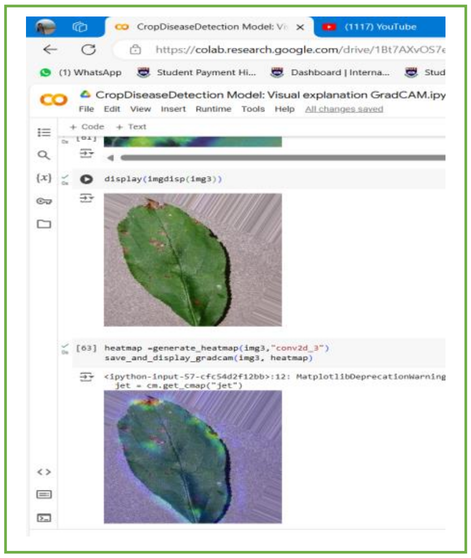
Generate Heatmap:



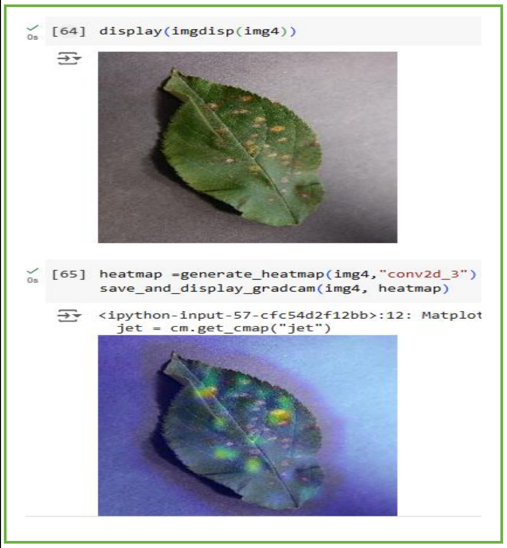
```
[58] display(imgdisp(img1))

[59] heatmap =generate_heatmap(img1,"conv2d_3")
     save_and_display_gradcam(img1, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecationWarning: The
  jet = cm.get_cmap("jet")
```



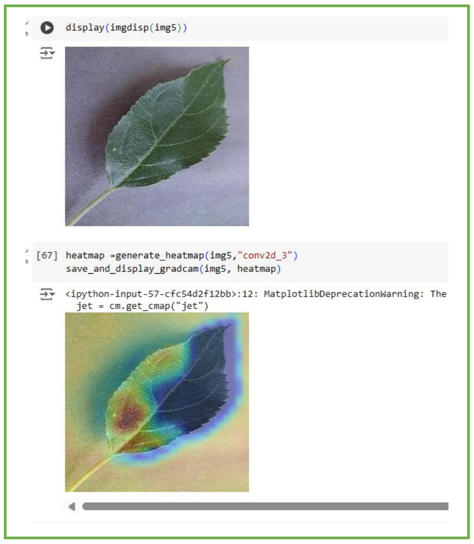
```
display(imgdisp(img2))

[61] heatmap =generate_heatmap(img2,"conv2d_3")
     save_and_display_gradcam(img2, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecationWarni
  jet = cm.get_cmap("jet")
```


CropDiseaseDetection Model: Vi
(1117) YouTube
https://colab.research.google.com/drive/1Bt7AXvOS7
(1) WhatsApp
Student Payment Hi...
Dashboard | Interna...
Stud
CropDiseaseDetection Model: Visual explanation GradCAM.ipy
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

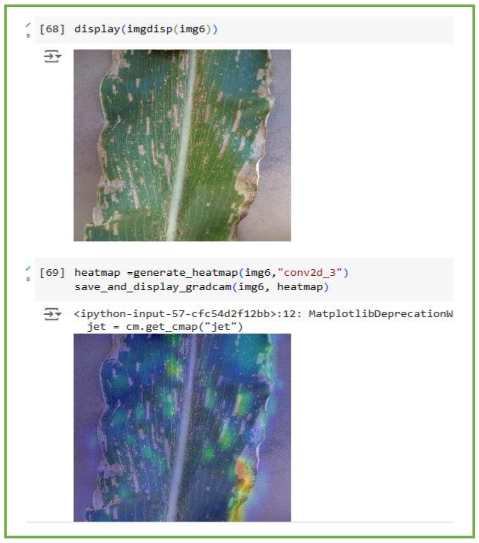
```
display(imgdisp(img3))

[63] heatmap =generate_heatmap(img3,"conv2d_3")
     save_and_display_gradcam(img3, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecationWarning
  jet = cm.get_cmap("jet")
```



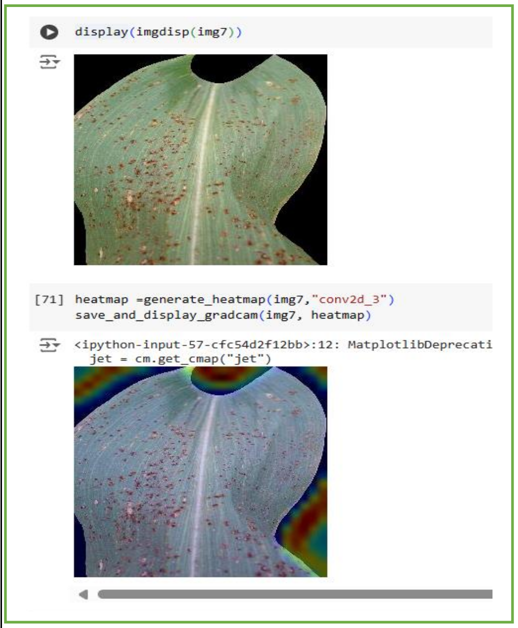
```
[64] display(imgdisp(img4))

[65] heatmap =generate_heatmap(img4,"conv2d_3")
     save_and_display_gradcam(img4, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: Matplot
  jet = cm.get_cmap("jet")
```



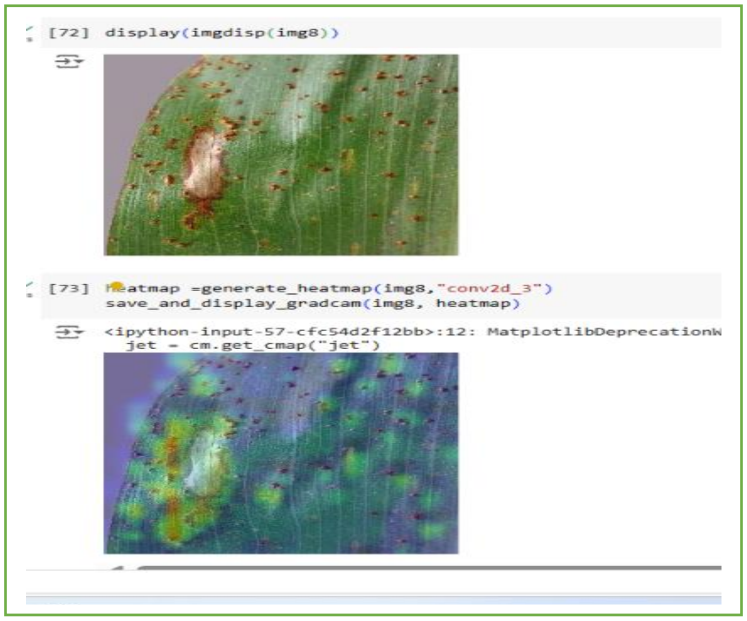
```
display(imgdisp(img5))

[67] heatmap =generate_heatmap(img5,"conv2d_3")
     save_and_display_gradcam(img5, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecationWarning: The
  jet = cm.get_cmap("jet")
```



```
[68] display(imgdisp(img6))

[69] heatmap =generate_heatmap(img6,"conv2d_3")
     save_and_display_gradcam(img6, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecationW
  jet = cm.get_cmap("jet")
```



```
display(imgdisp(img7))

[71] heatmap =generate_heatmap(img7,"conv2d_3")
     save_and_display_gradcam(img7, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecati
  jet = cm.get_cmap("jet")
```



```
[72] display(imgdisp(img8))

[73] heatmap =generate_heatmap(img8,"conv2d_3")
     save_and_display_gradcam(img8, heatmap)

<ipython-input-57-cfc54d2f12bb>:12: MatplotlibDeprecationW
  jet = cm.get_cmap("jet")
```

Grad-CAM generates a heatmap that highlights the crucial regions of an image by analyzing the gradients flowing into the last convolutional layer of the CNN.