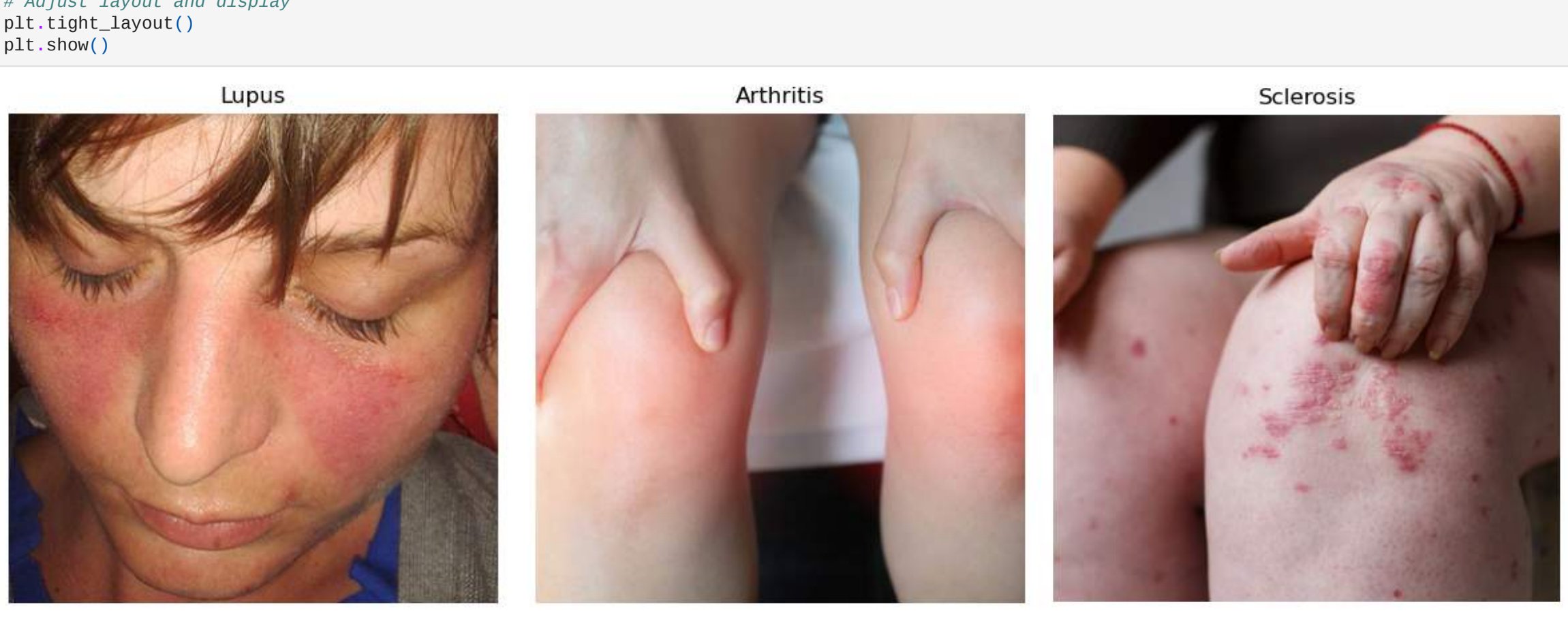
Quantitative Analysis of Histopathological Images for Autoimmune Diseases Diagnosis

Brief about the Autoimmune diseases

In [1]:
```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load images
img_lupus=mpimg.imread('Medical/Disease Images/Lupus.png')
img_arthritis=mpimg.imread('Medical/Disease Images/Arthritis.png')
img_sclerosis=mpimg.imread('Medical/Disease Images/Sclerosis.png')

# Create a figure with a single row and three columns
fig, axis=plt.subplots(1, 3, figsize=(12, 4))

# Display each image
axis[0].imshow(img_lupus)
axis[0].set_title('Lupus')
axis[0].axis('off')

axis[1].imshow(img_arthritis)
axis[1].set_title('Arthritis')
axis[1].axis('off')

axis[2].imshow(img_sclerosis)
axis[2].set_title('Sclerosis')
axis[2].axis('off')

# Adjust layout and display
plt.tight_layout()
plt.show()
```



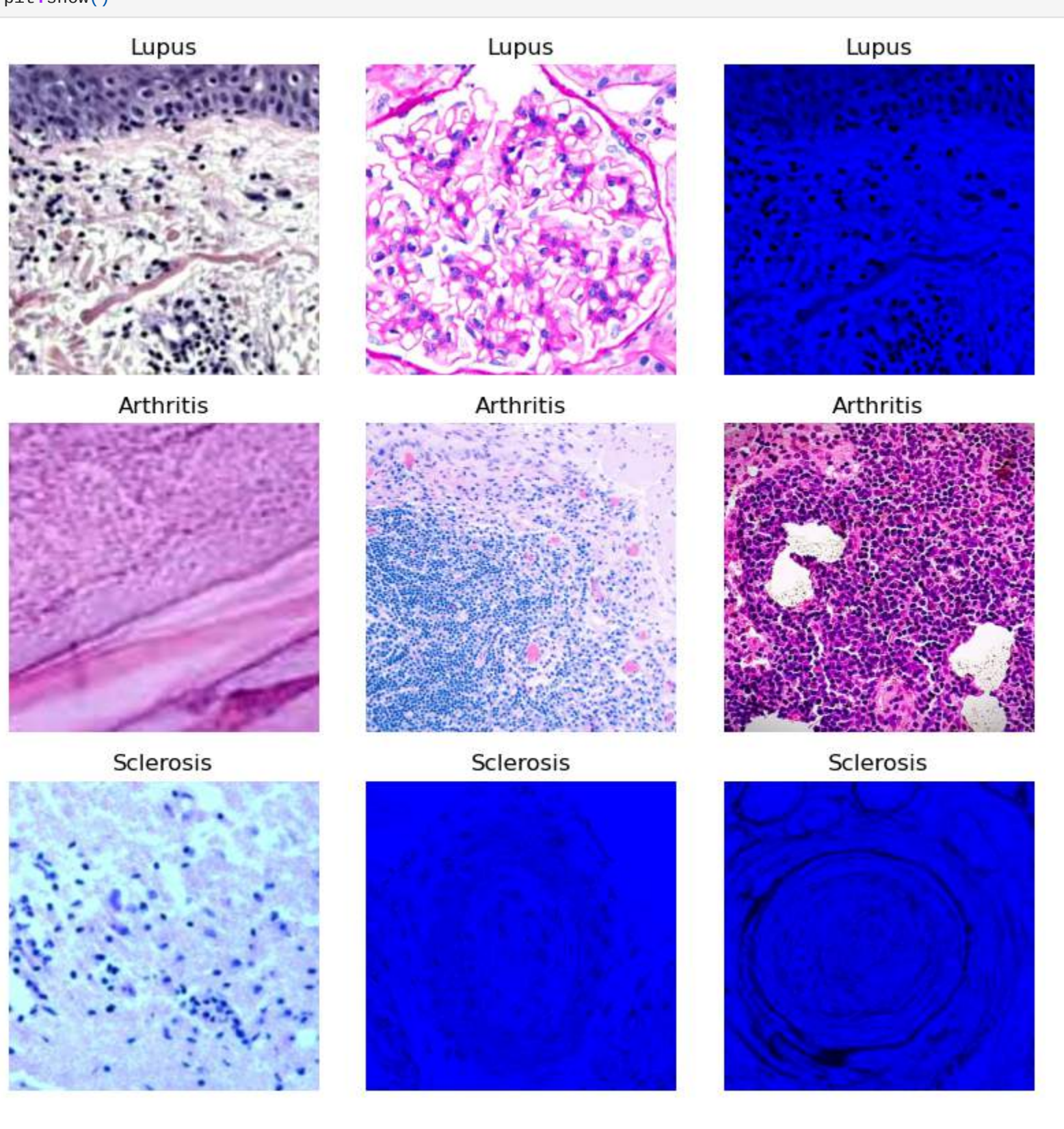
Importing Libraries

In [3]:
```python
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import  GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet import preprocess_input
import seaborn as sns
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

Data Collection

In [4]:
```python
data_dir='Medical'
subdirectories=['Lupus', 'Arthritis', 'Sclerosis']
```

In [5]:
```python
fig,axis=plt.subplots(len(subdirectories),3,figsize=(8, 8))

for i,subdir in enumerate(subdirectories):
    subdir_path = os.path.join(data_dir, subdir)
    for j in range(3):
        filename=os.listdir(subdir_path)[j]
        imagepath=os.path.join(subdir_path,filename)
        image=cv2.imread(imagepath)
        image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        axis[i,j].imshow(image)
        axis[i,j].set_title(subdir)
        axis[i,j].axis('off')

plt.tight_layout()
plt.show()
```



Data Preprocessing

In [6]:
```python
cleaned_images = []

for subdir in subdirectories:
    subdir_path=os.path.join(data_dir, subdir)
    for filename in os.listdir(subdir_path):
        imagepath=os.path.join(subdir_path, filename)
        image=cv2.imread(imagepath)
        if image is not None:
            cleaned_img=cv2.GaussianBlur(image, (5, 5), 0)
            cleaned_images.append((cleaned_img, subdir))
        else:
            print(f"Could not read image: {imagepath}")
```

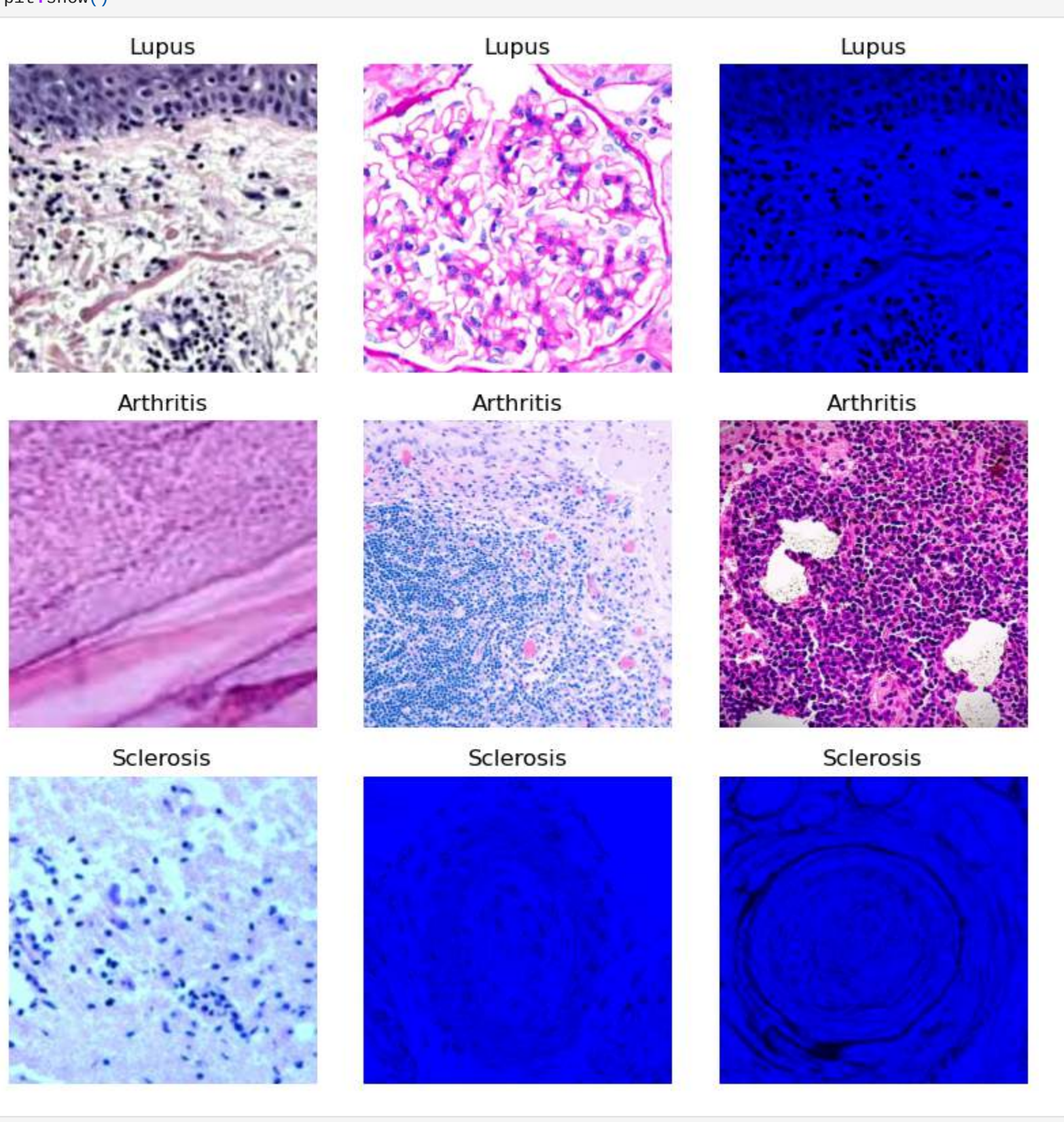
In [7]:
```python
# Preprocessing Function
def preprocess_image(img):
    resized_img=cv2.resize(img, (224, 224))
    return resized_img
```

In [8]:
```python
preprocessed_images=[]
labels=[]
```

In [9]:
```python
for image,label in cleaned_images:
    preprocessed_img=preprocess_image(image)
    preprocessed_images.append(preprocessed_img)
    labels.append(label)
```

Data Visualization

In [10]:
```python
# Display preprocessed images after Gaussian blur
fig,axis=plt.subplots(len(subdirectories),3,figsize=(8, 8))

for i, subdir in enumerate(subdirectories):
    subdir_preprocessed_images=[image for image,label in cleaned_images if label == subdir]
    for j in range(3):
        image=subdir_preprocessed_images[j]
        image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        axis[i,j].imshow(image)
        axis[i,j].set_title(subdir)
        axis[i,j].axis('off')

plt.tight_layout()
plt.show()
```



In [11]:
```python
# Convert lists to numpy arrays
preprocessed_images=np.array(preprocessed_images)
labels=np.array(labels)
```

In [12]:
```python
# Encode labels as integers
label_encoder=LabelEncoder()
encoded_labels=label_encoder.fit_transform(labels)
```

In [13]:
```python
# Convert encoded labels to categorical (one-hot encoding)
categorical_labels=to_categorical(encoded_labels)
```

In [14]:
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test=train_test_split(preprocessed_images, categorical_labels, test_size=0.2, random_state=42)
```

In [15]:
```python
datagen=ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=preprocess_input
)
```

In [16]:
```python
# Apply data augmentation only to training data
train_generator=datagen.flow(X_train, y_train, batch_size=32)
```

In [17]:
```python
# Load VGG16 model without the top layer
base_model=VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.summary()
```

```
WARNING:tensorflow:From D:\New folder\lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From D:\New folder\lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Model: "vgg16"
```

Layer (type)	Output Shape	Param #
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
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
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

```
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
```

Feature Extraction

In [18]:
```python
# Extract features for each preprocessed image using VGG16
def extract_features(data, model):
    features=model.predict(data)
    return features
```

In [19]:
```python
# Extract features for training and testing data
train_features=[]
for image in X_train:
    image=np.expand_dims(image,axis=0)
    image=preprocess_input(image)
    feature=extract_features(image,base_model)
    train_features.append(feature.flatten())
```

```
1/1 [==============================] - 1s 983ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 233ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 139ms/step
```

In [20]:
```python
test_features = []
for image in X_test:
    image=np.expand_dims(image,axis=0)
    image=preprocess_input(image)
    feature=extract_features(image, base_model)
    test_features.append(feature.flatten())
```

```
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 139ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 144ms/step
```

In [21]:
```python
train_features=np.array(train_features)
test_features=np.array(test_features)
```

In [22]:
```python
# Define the model
model=Sequential([
    Dense(512, activation='relu', input_shape=(train_features.shape[1],)),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(len(subdirectories), activation='softmax')
])
```

In [23]:
```python
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

In [24]:
```python
model.summary()
```

```
Model: "sequential"
```

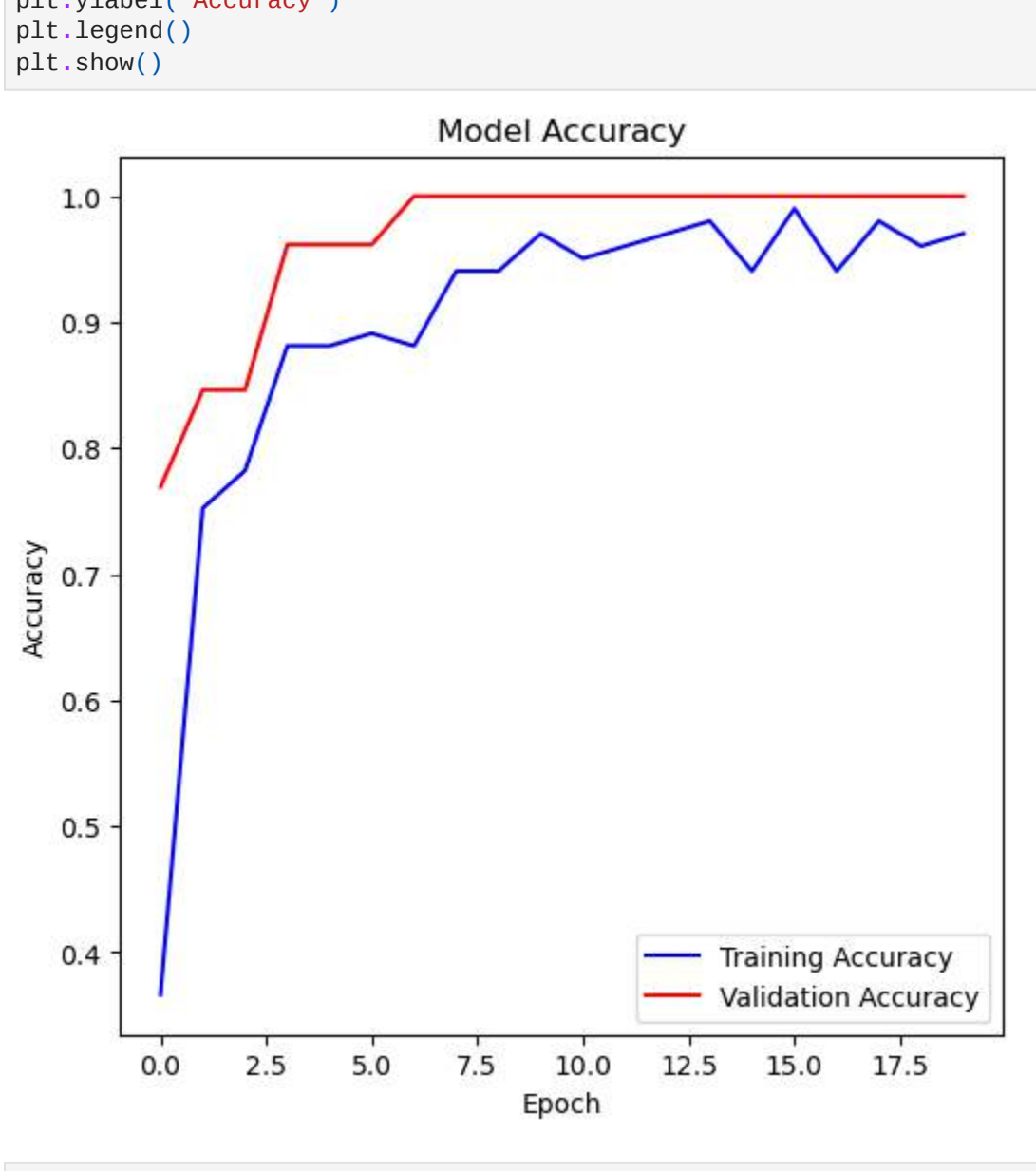
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	12845568
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 3)	771

```
Total params: 12977667 (49.51 MB)
Trainable params: 12977667 (49.51 MB)
Non-trainable params: 0 (0.00 Byte)
```

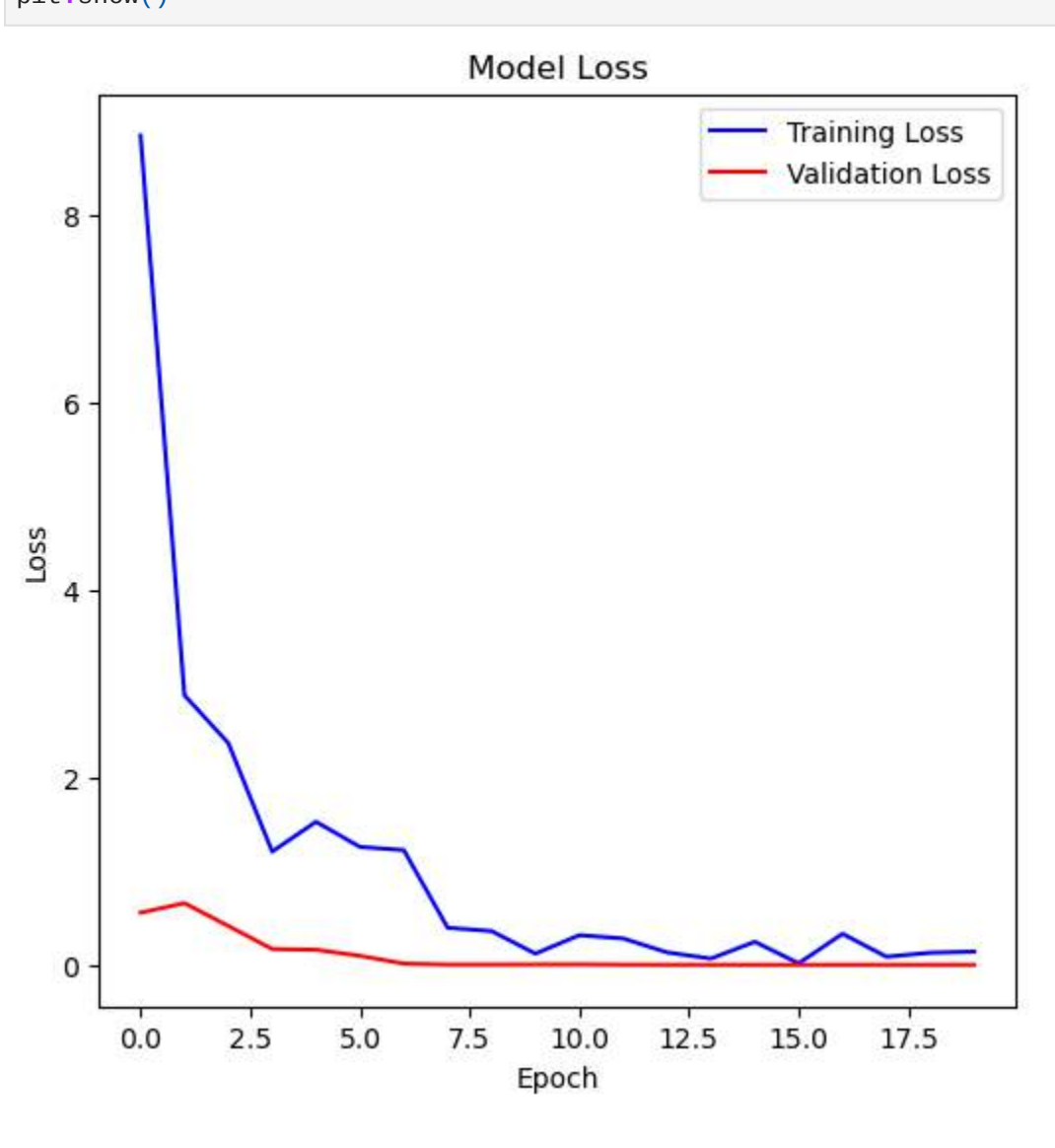
In [25]:
```python
history=model.fit(
    train_features,y_train,
    epochs=20,
    batch_size=30,
    validation_data=(test_features, y_test)
)
```

```
Epoch 1/20
WARNING:tensorflow:From D:\New folder\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From D:\New folder\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

4/4 [==============================] - 3s 259ms/step - loss: 8.8544 - accuracy: 0.3663 - val_loss: 0.5564 - val_accuracy: 0.7692
Epoch 2/20
4/4 [==============================] - 1s 152ms/step - loss: 2.8790 - accuracy: 0.7525 - val_loss: 0.6580 - val_accuracy: 0.8462
Epoch 3/20
4/4 [==============================] - 1s 159ms/step - loss: 2.3687 - accuracy: 0.7822 - val_loss: 0.4154 - val_accuracy: 0.8462
Epoch 4/20
4/4 [==============================] - 1s 150ms/step - loss: 1.2095 - accuracy: 0.8812 - val_loss: 0.1687 - val_accuracy: 0.9615
Epoch 5/20
4/4 [==============================] - 1s 147ms/step - loss: 1.5277 - accuracy: 0.8812 - val_loss: 0.1605 - val_accuracy: 0.9615
Epoch 6/20
4/4 [==============================] - 1s 153ms/step - loss: 1.2587 - accuracy: 0.8911 - val_loss: 0.0958 - val_accuracy: 0.9615
Epoch 7/20
4/4 [==============================] - 1s 154ms/step - loss: 1.2255 - accuracy: 0.8812 - val_loss: 0.0133 - val_accuracy: 1.0000
Epoch 8/20
4/4 [==============================] - 1s 150ms/step - loss: 0.3955 - accuracy: 0.9406 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 9/20
4/4 [==============================] - 1s 146ms/step - loss: 0.3607 - accuracy: 0.9406 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 10/20
4/4 [==============================] - 1s 150ms/step - loss: 0.1196 - accuracy: 0.9703 - val_loss: 0.0039 - val_accuracy: 1.0000
Epoch 11/20
4/4 [==============================] - 1s 146ms/step - loss: 0.3168 - accuracy: 0.9505 - val_loss: 0.0047 - val_accuracy: 1.0000
Epoch 12/20
4/4 [==============================] - 1s 153ms/step - loss: 0.2815 - accuracy: 0.9604 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 13/20
4/4 [==============================] - 1s 150ms/step - loss: 0.1329 - accuracy: 0.9703 - val_loss: 1.6159e-04 - val_accuracy: 1.0000
Epoch 14/20
4/4 [==============================] - 1s 147ms/step - loss: 0.0668 - accuracy: 0.9802 - val_loss: 1.3606e-05 - val_accuracy: 1.0000
Epoch 15/20
4/4 [==============================] - 1s 146ms/step - loss: 0.2463 - accuracy: 0.9406 - val_loss: 8.1153e-07 - val_accuracy: 1.0000
Epoch 16/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0165 - accuracy: 0.9901 - val_loss: 1.1921e-07 - val_accuracy: 1.0000
Epoch 17/20
4/4 [==============================] - 1s 153ms/step - loss: 0.3312 - accuracy: 0.9406 - val_loss: 2.2925e-08 - val_accuracy: 1.0000
Epoch 18/20
4/4 [==============================] - 1s 145ms/step - loss: 0.0866 - accuracy: 0.9802 - val_loss: 1.3755e-08 - val_accuracy: 1.0000
Epoch 19/20
4/4 [==============================] - 1s 148ms/step - loss: 0.1290 - accuracy: 0.9604 - val_loss: 9.1699e-09 - val_accuracy: 1.0000
Epoch 20/20
4/4 [==============================] - 1s 148ms/step - loss: 0.1415 - accuracy: 0.9703 - val_loss: 9.1699e-09 - val_accuracy: 1.0000
```

In [27]:
```python
plt.figure(figsize=(6, 6))
plt.plot(history.history['accuracy'], color='blue', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], color='red', label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



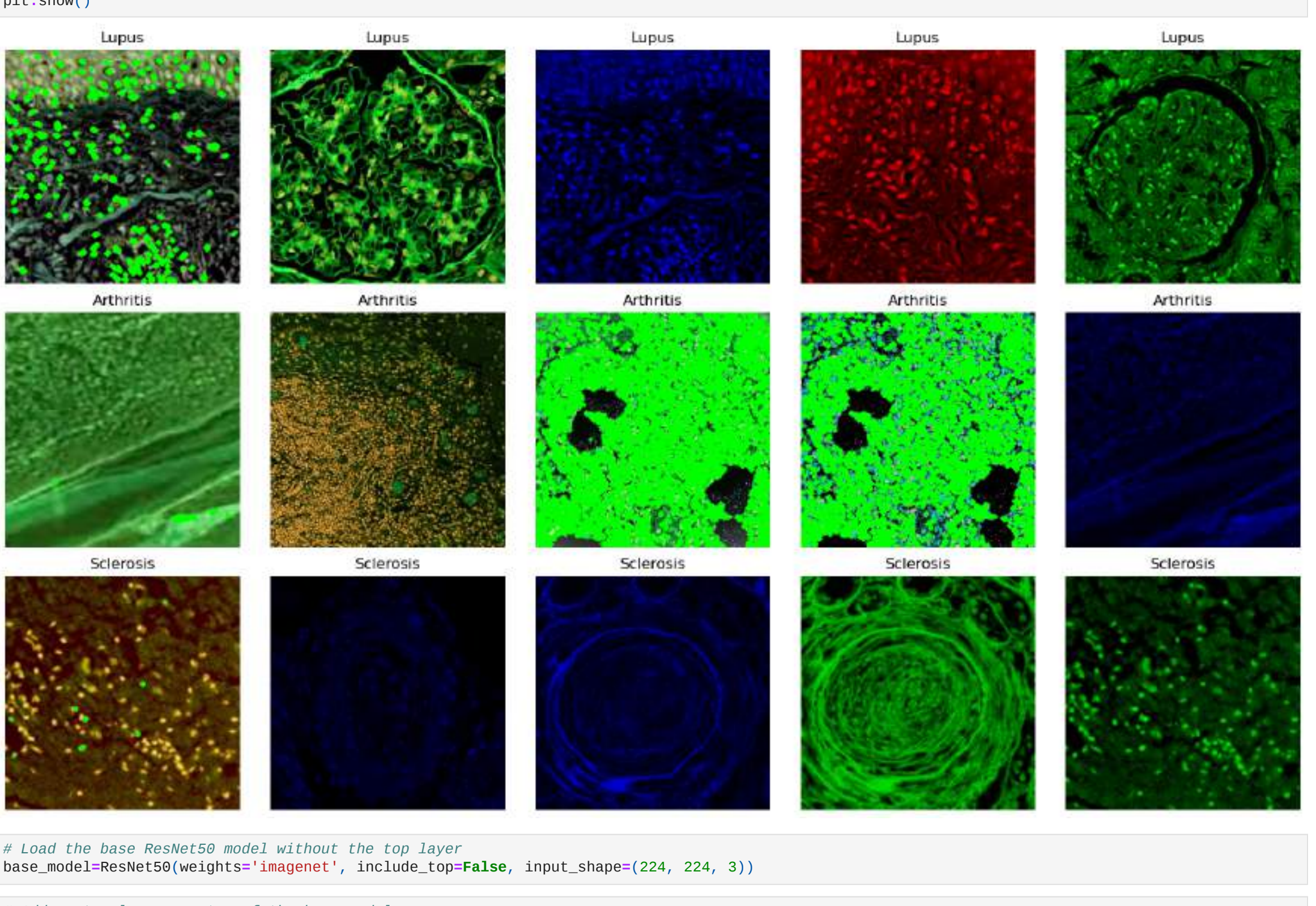
In [28]:
```python
plt.figure(figsize=(6, 6))
plt.plot(history.history['loss'], color='blue', label='Training Loss')
plt.plot(history.history['val_loss'], color='red', label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Analysis Using Patterns

In [30]:
```python
# Function to identify patterns
def identify_patterns(image):
    gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    # Threshold the image
    i,thresh=cv2.threshold(gray,200,255,cv2.THRESH_BINARY)
    # Find contours
    contours, i=cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    pattern_img=cv2.drawContours(image.copy(),contours,-1,(0,255,0),3)
    return pattern_img
```

In [31]:
```python
# Function to display identified patterns
num_images_per_directory=5
# Calculate number of rows and columns for subplots
num_rows=len(subdirectories)
num_cols=num_images_per_directory

fig,axis=plt.subplots(num_rows,num_cols,figsize=(3*num_cols,3*num_rows))

for i, subdir in enumerate(subdirectories):
    subdir_images = preprocessed_images[labels==subdir][:num_images_per_directory]
    for j in range(num_images_per_directory):
        image=subdir_images[j]
        pattern_img=identify_patterns(cv2.cvtColor((image*255).astype(np.uint8), cv2.COLOR_RGB2BGR))

        axis[i,j].imshow(pattern_img, cmap='gray')
        axis[i,j].set_title(f'{subdir}')
        axis[i,j].axis('off')

# Hide empty subplots (not necessary since we ensure the number of images matches the subplots)
for i in range(num_rows):
    for j in range(num_images_per_directory, num_cols):
        axis[i, j].axis('off')

plt.tight_layout()
plt.show()
```



In [32]:
```python
# Load the base ResNet50 model without the top layer
base_model=ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

In [33]:
```python
# Add custom layers on top of the base model
x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(512, activation='relu')(x)
x=Dropout(0.5)(x)
x=Dense(256, activation='relu')(x)
x=Dropout(0.5)(x)
predictions=Dense(len(subdirectories), activation='softmax')(x)
```

Model Evaluation

In [34]:
```python
# Create the complete model
model=Model(inputs=base_model.input,outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

In [35]:
```python
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=20, validation_data=(X_test, y_test))
```

```
Epoch 1/10
6/6 [==============================] - 49s 3s/step - loss: 1.2492 - accuracy: 0.4356 - val_loss: 1.1733 - val_accuracy: 0.3077
Epoch 2/10
6/6 [==============================] - 18s 3s/step - loss: 1.0731 - accuracy: 0.5149 - val_loss: 0.9691 - val_accuracy: 0.4615
Epoch 3/10
6/6 [==============================] - 19s 3s/step - loss: 0.5841 - accuracy: 0.7921 - val_loss: 0.8511 - val_accuracy: 0.6154
Epoch 4/10
6/6 [==============================] - 18s 3s/step - loss: 0.3798 - accuracy: 0.8911 - val_loss: 0.7261 - val_accuracy: 0.5769
Epoch 5/10
6/6 [==============================] - 17s 3s/step - loss: 0.4218 - accuracy: 0.8119 - val_loss: 0.6411 - val_accuracy: 0.6538
Epoch 6/10
6/6 [==============================] - 20s 3s/step - loss: 0.2153 - accuracy: 0.9307 - val_loss: 0.5375 - val_accuracy: 0.8077
Epoch 7/10
6/6 [==============================] - 17s 3s/step - loss: 0.2208 - accuracy: 0.9307 - val_loss: 0.4361 - val_accuracy: 0.8462
Epoch 8/10
6/6 [==============================] - 17s 3s/step - loss: 0.1158 - accuracy: 0.9604 - val_loss: 0.3829 - val_accuracy: 0.8846
Epoch 9/10
6/6 [==============================] - 18s 3s/step - loss: 0.1326 - accuracy: 0.9703 - val_loss: 0.3736 - val_accuracy: 0.9231
Epoch 10/10
6/6 [==============================] - 18s 3s/step - loss: 0.0691 - accuracy: 1.0000 - val_loss: 0.3881 - val_accuracy: 0.8846
```

Out[35]:
```
<keras.src.callbacks.History at 0x1b1b1c8f670>
```

In [37]:
```python
final_train_accuracy=history.history['accuracy'][-1]
final_val_accuracy =history.history['val_accuracy'][-1]
print(f'Final Training Accuracy: {final_train_accuracy:.4f}')
print(f'Final Validation Accuracy: {final_val_accuracy:.4f}')
```

```
Final Training Accuracy: 0.9703
Final Validation Accuracy: 1.0000
```

In [38]:
```python
final_train_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]

print(f"Final Training Loss: {final_train_loss:.4f}")
print(f"Final Validation Loss: {final_val_loss:.4f}")
```

```
Final Training Loss: 0.1415
Final Validation Loss: 0.0000
```

In [39]:
```python
# Prepare to identify correct predictions
correct_images_per_dir={subdir: [] for subdir in subdirectories}
```

In [40]:
```python
# Predict and identify correct images
for subdir in subdirectories:
    subdir_images=[image for image, label in zip(preprocessed_images, labels) if label==subdir][:50]

    if len(subdir_images)==0:
        continue

    subdir_images=np.array(subdir_images)
    predictions=model.predict(subdir_images)

    for image, prediction in zip(subdir_images, predictions):
        predicted_label=label_encoder.inverse_transform([np.argmax(prediction)])[0]
        if predicted_label==subdir:
            correct_images_per_dir[subdir].append(image)
        if len(correct_images_per_dir[subdir])>=5:
            break
```

```
2/2 [==============================] - 4s 664ms/step
2/2 [==============================] - 2s 536ms/step
2/2 [==============================] - 2s 459ms/step
```

In [ ]: