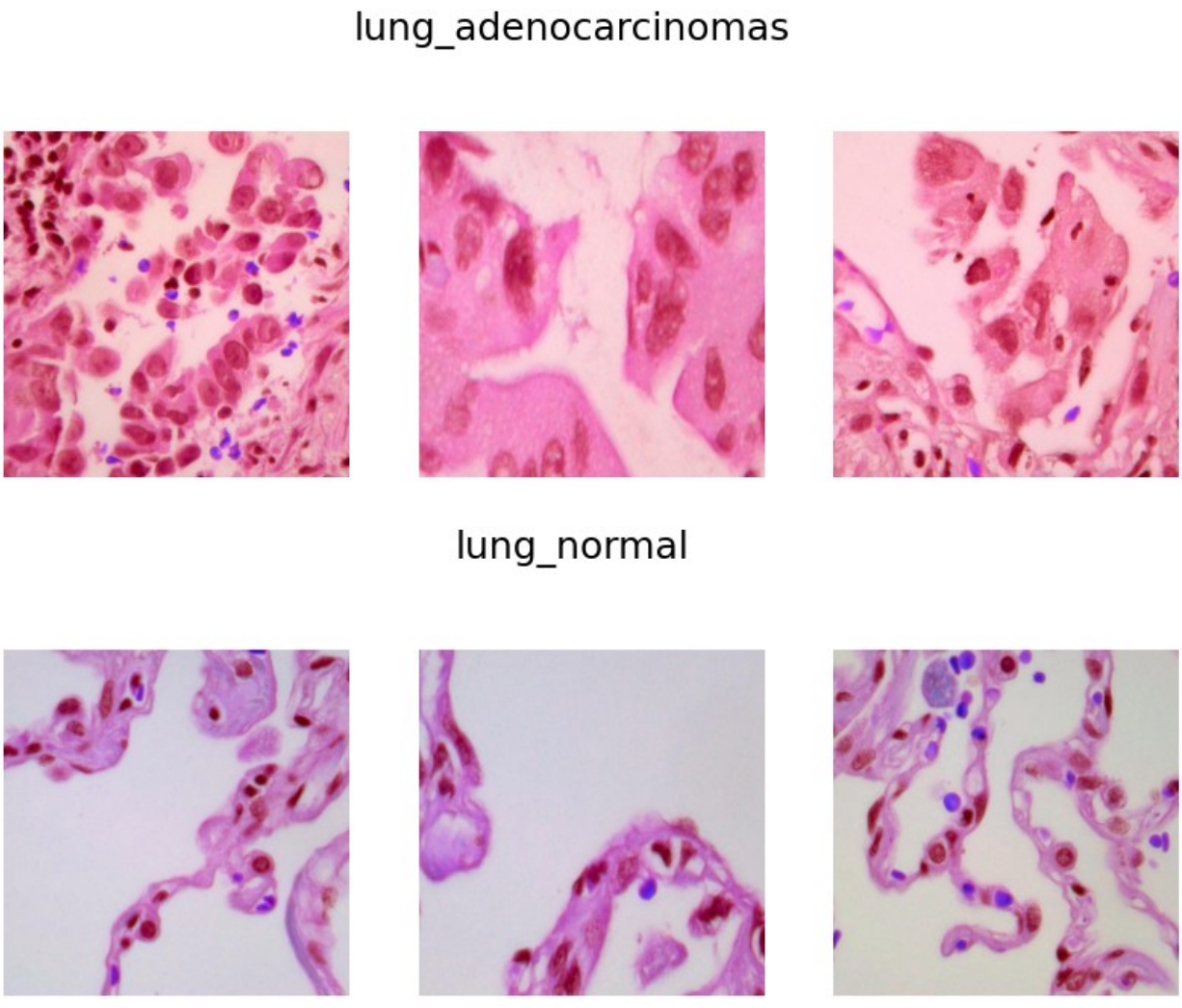
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.applications import vgg16
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tqdm import tqdm
import os
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, TensorBoard, ModelCheckpoint
from sklearn.metrics import classification_report,confusion_matrix
import ipywidgets as widgets
import io
from PIL import Image
from IPython.display import display,clear_output
from warnings import filterwarnings

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
cp: cannot stat 'kaggle.json': No such file or directory
```

```
import zipfile
zip_ref =
zipfile.ZipFile('/content/drive/MyDrive/lung_colon_image_set.zip')
zip_ref.extractall('/content')
zip_ref.close()

DATADIR = '/content/Lung_Cancer'
CATEGORIES = ['lung_adenocarcinomas', 'lung_normal',
'lung_squamous_cell_carcinomas']

for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    images = os.listdir(path)
```

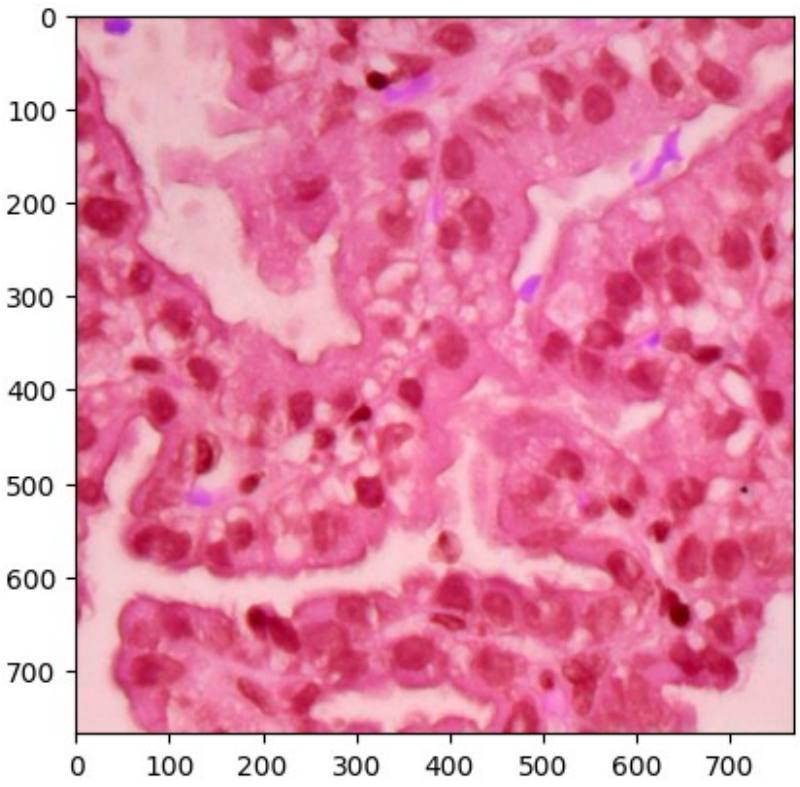
```
    # Initialize a subplot with 1 row and 3 columns
    fig, ax = plt.subplots(1, 3, figsize=(10, 4))
    fig.suptitle(f'{category}', fontsize=18)

    for i in range(3):
        # Randomly select an image
        img_name = images[np.random.randint(0, len(images))]
        img_path = os.path.join(path, img_name)
        img_array = cv2.imread(img_path)

        # Display the image
        ax[i].imshow(img_array)
        ax[i].axis('off')
```




```
img_array.shape
```

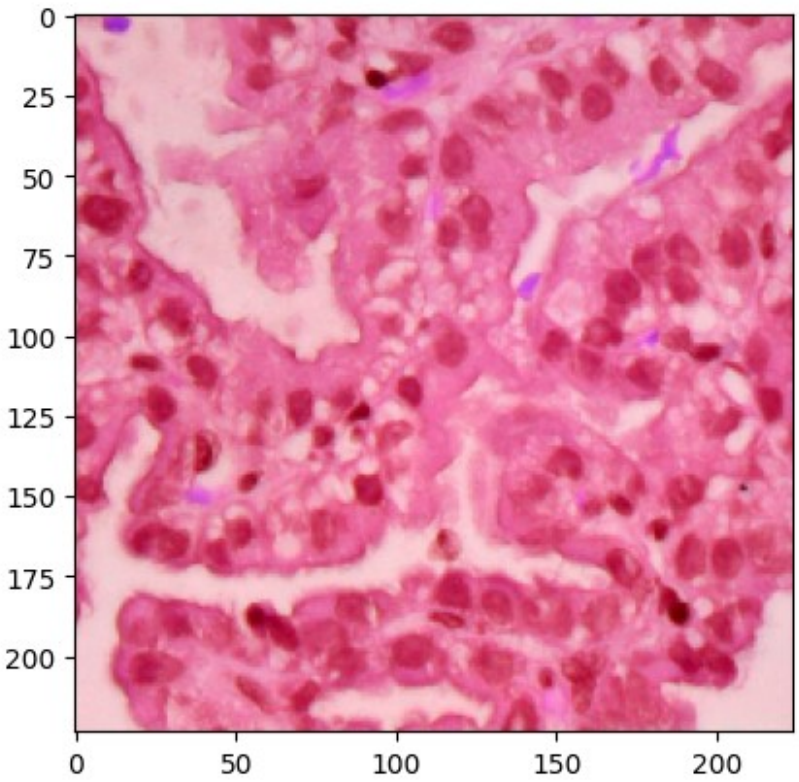
```
(768, 768, 3)
```

```
for category in CATEGORIES:
  path = os.path.join(DATADIR, category)      #path to cats or dogs
directory
  for img in os.listdir(path):
    img_array = cv2.imread(os.path.join(path, img))
    plt.imshow(img_array)
    plt.show()
    break
  break
```




```
IMG_SIZE = 224

new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array)
```

```
<matplotlib.image.AxesImage at 0x7ce014f89310>
```




```
new_array.shape
```

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

```
# creating training data

training_data = []

def create_training_data():
  for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    class_num = CATEGORIES.index(category)
    for img in os.listdir(path):
      try:
        img_array = cv2.imread(os.path.join(path, img))
        new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
        training_data.append([new_array, class_num])
      except Exception as e:
        pass

create_training_data()

len(training_data)
```

```
15000
```

```
X = []
y = []

for features, label in training_data:
  X.append(features)
  y.append(label)

type(X),type(y)
```

```
(list, list)
```

```
# Converting the data type of X and y from list to numpy array
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 3)    # last value '3'
for 'RGB'
y = np.array(y)

type(X),type(y)
```

```
(numpy.ndarray, numpy.ndarray)
```

```
#train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size =
0.2,random_state = 42)

print(f'X_train Length : {X_train.shape[0]}, X_train Image size :
{X_train.shape[1:3]}, Channel Dimension : {X_train.shape[3]}')
print(f'X_test Length : {X_test.shape[0]}, X_test Image size :
{X_test.shape[1:3]}, Channel Dimension : {X_test.shape[3]}')
```

```
X_train Length : 12000, X_train Image size : (224, 224), Channel
Dimension : 3
X_test Length : 3000, X_test Image size : (224, 224), Channel
Dimension : 3
```

```
# vgg model
vgg = vgg16.VGG16(weights = 'imagenet',include_top = False,input_shape
= (IMG_SIZE, IMG_SIZE, 3))

# freezing the bottom (conv) layers
for layer in vgg.layers:
  layer.trainable = False

# building the top (FC) layers
model = keras.Sequential([
    vgg,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dense(1024, activation = 'relu'),
    keras.layers.Dense(512, activation = 'relu'),
    keras.layers.Dense(3, activation = 'softmax'),
])
```

```
#compilation
model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])

model.fit(X_train, y_train, epochs = 5)
```

```
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 65s 161ms/step - accuracy: 0.8909 - loss:
0.7543
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 80s 161ms/step - accuracy: 0.9669 - loss:
0.0810
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 82s 161ms/step - accuracy: 0.9808 - loss:
0.0477
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 82s 161ms/step - accuracy: 0.9883 - loss:
0.0314
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 82s 161ms/step - accuracy: 0.9898 - loss:
0.0299

<keras.src.callbacks.history.History at 0x7ce010559750>
```

```
# evaluation
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Model Accuracy : {accuracy * 100}')
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 16s 161ms/step - accuracy: 0.9709 - loss:
0.0995
Model Accuracy : 97.36666679382324
```

```
# first 5 true labels
y_test[:5]
```

```
array([2, 1, 2, 0, 1])
```

```
pred = np.argmax(model.predict(X_test), axis = -1)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 16s 166ms/step
```

```
# first 5 predicted labels
pred[:5]
```

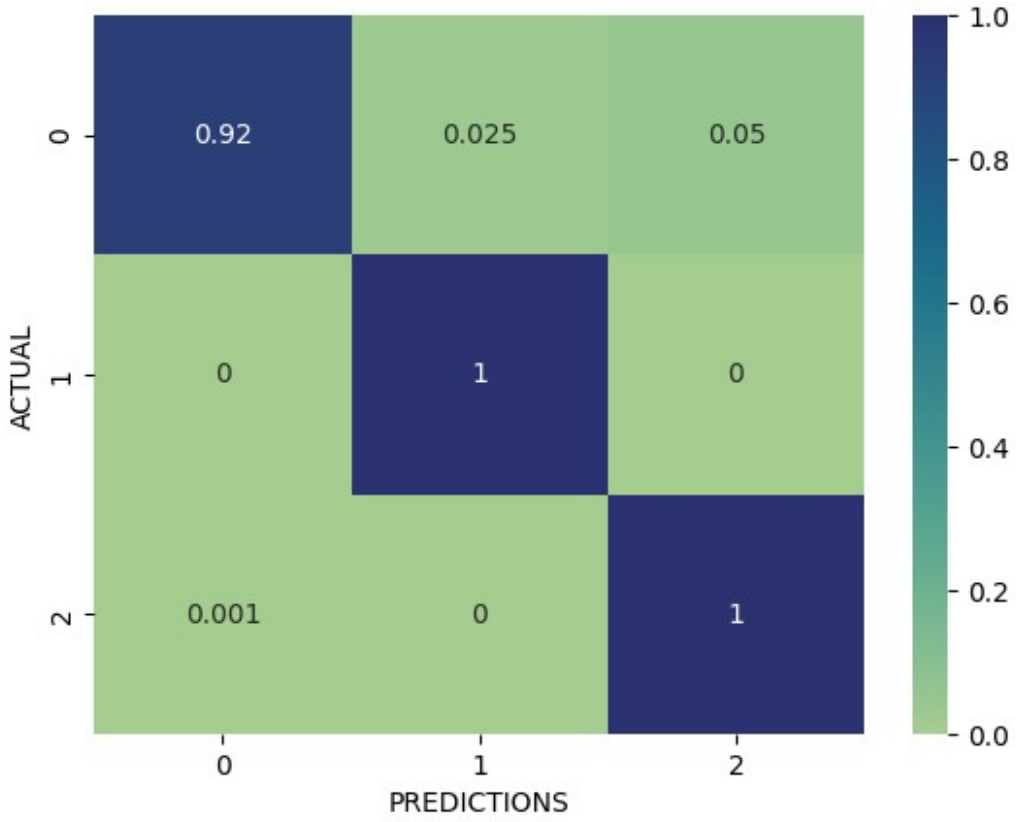
```
array([2, 1, 2, 2, 1])
```

```
# classification report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support
```

```
              0       1.00      0.92      0.96      1037
              1       0.97      1.00      0.99       970
              2       0.95      1.00      0.97       993

    accuracy                            0.97      3000
   macro avg       0.97      0.97      0.97      3000
weighted avg       0.97      0.97      0.97      3000
```

```
#confusion matrix
cf = confusion_matrix(y_test, pred, normalize = 'true')
sns.heatmap(cf, annot = True, cmap = 'crest');
plt.xlabel('PREDICTIONS');
plt.ylabel('ACTUAL');
```



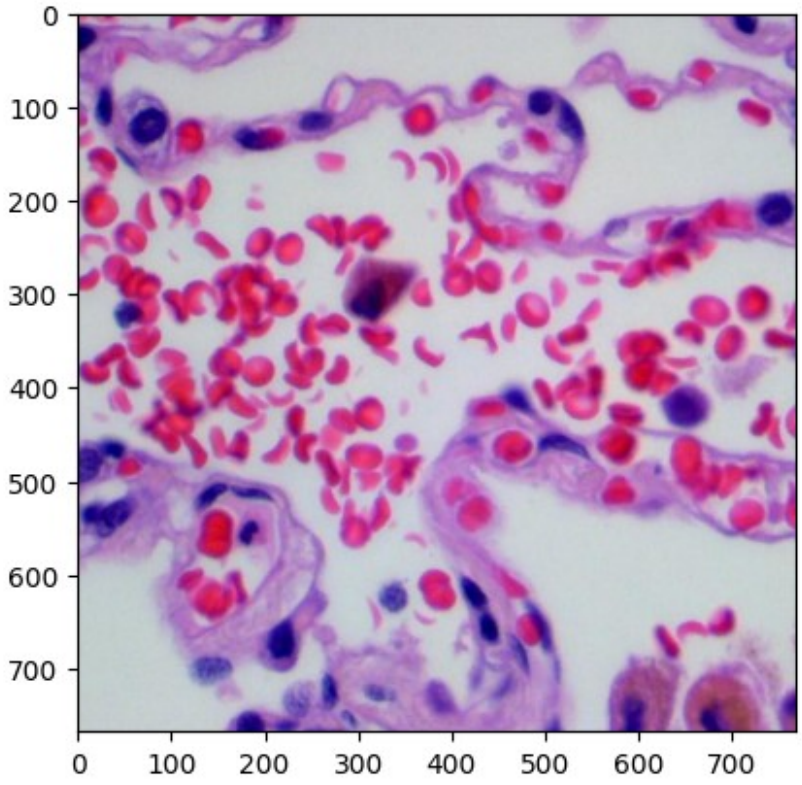

```
import ipywidgets as widgets
uploader = widgets.FileUpload()
display(uploader)
```


```
{"model_id":"24733fdc770f45e5a09c951aa55d1d1b","version_major":2,"vers
ion_minor":0}
```


```
def img_pred(uploader):
    for name, file_info in uploader.value.items():
```

```
        img = Image.open(io.BytesIO(file_info['content']))
    opencvImage = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
    img = cv2.resize(opencvImage,(244,244))
    img = img.reshape(1,244,244,3)
    p = model.predict(img)
    p = np.argmax(p,axis=1)[0]

    if p==0:
        print('\tLung_adenoacrcinomas - Correct classification with
97% accuracy')
    elif p==1:
        print('\tLung_normal - Correct classification with 100%
accuracy')
    elif p==2:
        print('\tLung_squamous_cell_carcinomas - Correct
classification with 99% accuracy')
    else:
        p=='Healthy'

from PIL import Image

for name, file_info in uploader.value.items():
  img3 = Image.open(io.BytesIO(file_info['content']))
  img3.show(title=None)
  plt.imshow(img3)


button = widgets.Button(description='Predict')
out = widgets.Output()
def on_button_clicked(_):
    with out:
        clear_output()
        try:
            img_pred(uploader)

        except:
            print('No Image Uploaded/Invalid Image File')
button.on_click(on_button_clicked)
widgets.VBox([button,out])
```

{"model_id":"e1fbd5a19d05404c85bfd916396e4728","version_major":2,"vers
ion_minor":0}


0
100
200
300
400
500
600
700
0
100
200
300
400
500
600
700