In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from glob import glob

from sklearn.model_selection import train_test_split
from sklearn import metrics

import cv2
import gc
import os

import tensorflow as tf
from tensorflow import keras
from keras import layers

import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
from zipfile import ZipFile

data_path = 'dataset.zip'

with ZipFile(data_path,'r') as zip:
  zip.extractall()
  print('The data set has been extracted.')
```

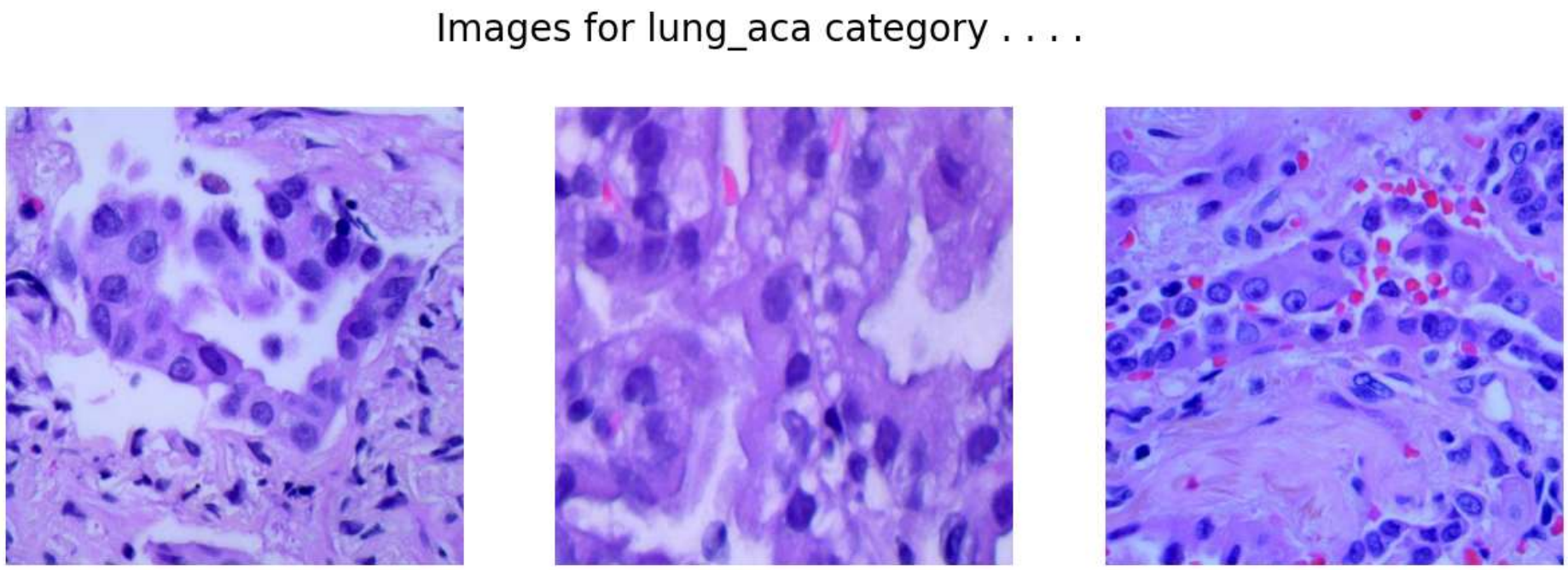
```
The data set has been extracted.
```

In [5]:
```
path = 'lung_colon_image_set/lung_image_sets'
classes = os.listdir(path)
classes
```

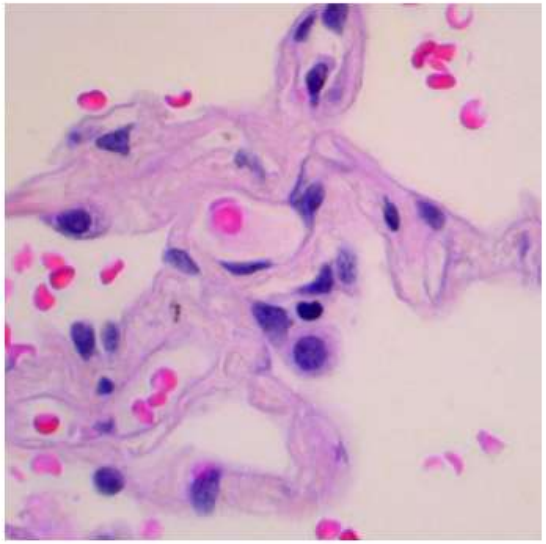
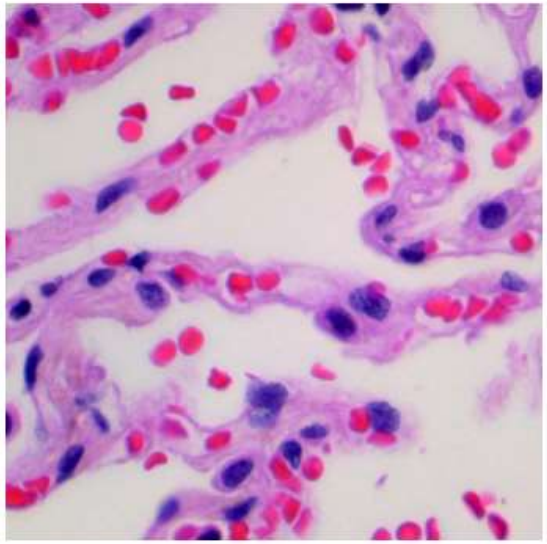
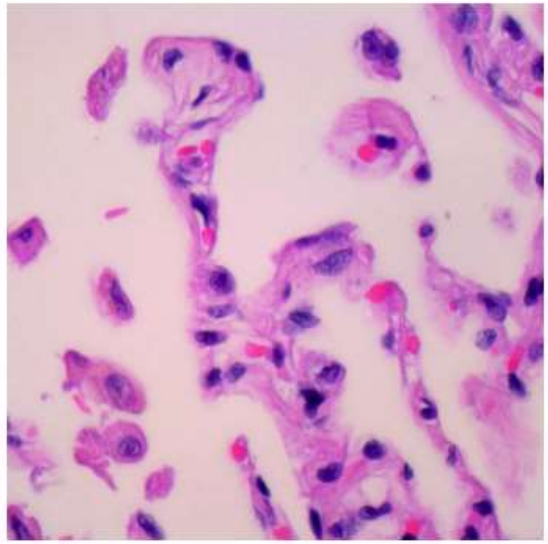
Out[5]:
```
['lung_aca', 'lung_n', 'lung_scc']
```

In [8]:
```
path = 'lung_colon_image_set/lung_image_sets'

for cat in classes:
    image_dir = f'{path}/{cat}'
    images = os.listdir(image_dir)
```

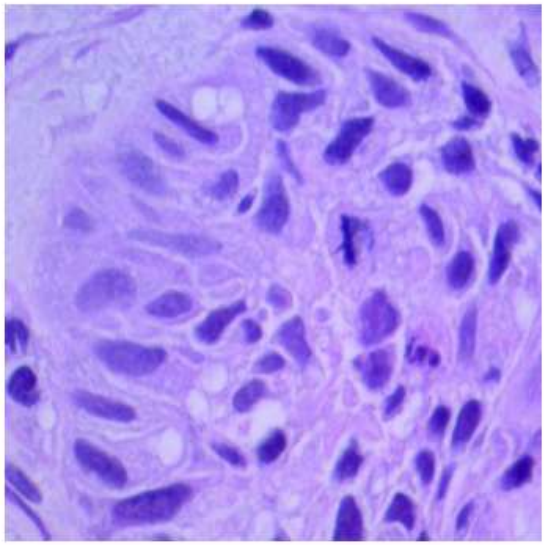
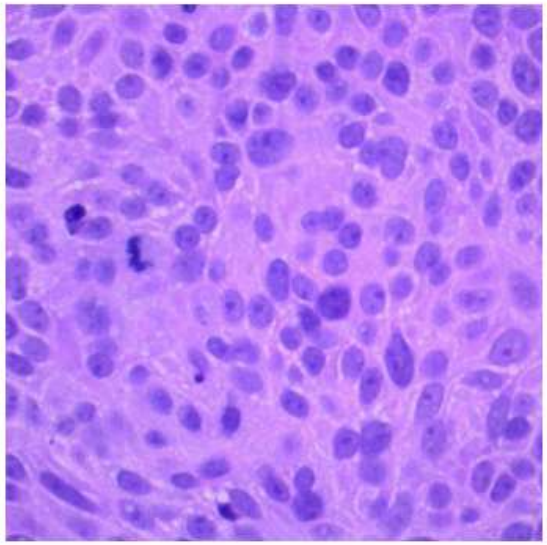
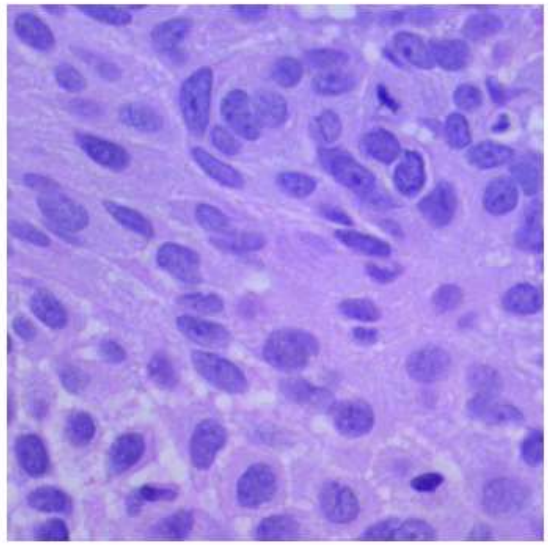
```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle(f'Images for {cat} category . . . .', fontsize=20)

for i in range(3):
    k = np.random.randint(0, len(images))
    img = np.array(Image.open(f'{path}/{cat}/{images[k]}'))
    ax[i].imshow(img)
    ax[i].axis('off')
plt.show()
```



Images for lung_n category . . . .

Images for lung_scc category . . . .

In [9]:
```
IMG_SIZE = 256
SPLIT = 0.2
EPOCHS = 10
BATCH_SIZE = 64
```

In [10]:
```
X = []
Y = []

for i, cat in enumerate(classes):
  images = glob(f'{path}/{cat}/*.jpeg')

  for image in images:
    img = cv2.imread(image)

    X.append(cv2.resize(img, (IMG_SIZE, IMG_SIZE)))
    Y.append(i)

X = np.asarray(X)
one_hot_encoded_Y = pd.get_dummies(Y).values
```

In [11]:
```
X_train, X_val, Y_train, Y_val = train_test_split(X, one_hot_encoded_Y,
                                                  test_size = SPLIT,
                                                  random_state = 2022)
print(X_train.shape, X_val.shape)
```

```
(12000, 256, 256, 3) (3000, 256, 256, 3)
```

In [12]:
```
model = keras.models.Sequential([
    layers.Conv2D(filters=32,
                  kernel_size=(5, 5),
                  activation='relu',
                  input_shape=(IMG_SIZE,
                               IMG_SIZE,
                               3),
                  padding='same'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(filters=64,
                  kernel_size=(3, 3),
                  activation='relu',
                  padding='same'),
    layers.MaxPooling2D(2, 2),
```

```
    layers.Conv2D(filters=128,
                  kernel_size=(3, 3),
                  activation='relu',
                  padding='same'),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(3, activation='softmax')
])
```

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      2432

 max_pooling2d (MaxPooling2  (None, 128, 128, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 32, 32, 128)       0
 g2D)

 flatten (Flatten)           (None, 131072)            0

 dense (Dense)               (None, 256)               33554688

 batch_normalization (Batch  (None, 256)               1024
 Normalization)

 dense_1 (Dense)             (None, 128)               32896

 dropout (Dropout)           (None, 128)               0

 batch_normalization_1 (Bat  (None, 128)               512
 chNormalization)

 dense_2 (Dense)             (None, 3)                 387

=================================================================
Total params: 33684291 (128.50 MB)
Trainable params: 33683523 (128.49 MB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

In [14]:
```
keras.utils.plot_model(
    model,
    show_shapes = True,
```

```
    show_dtype = True,
    show_layer_activations = True
)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot_model to work.
```

In [15]:
```
model.compile(
    optimizer = 'adam',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)
```

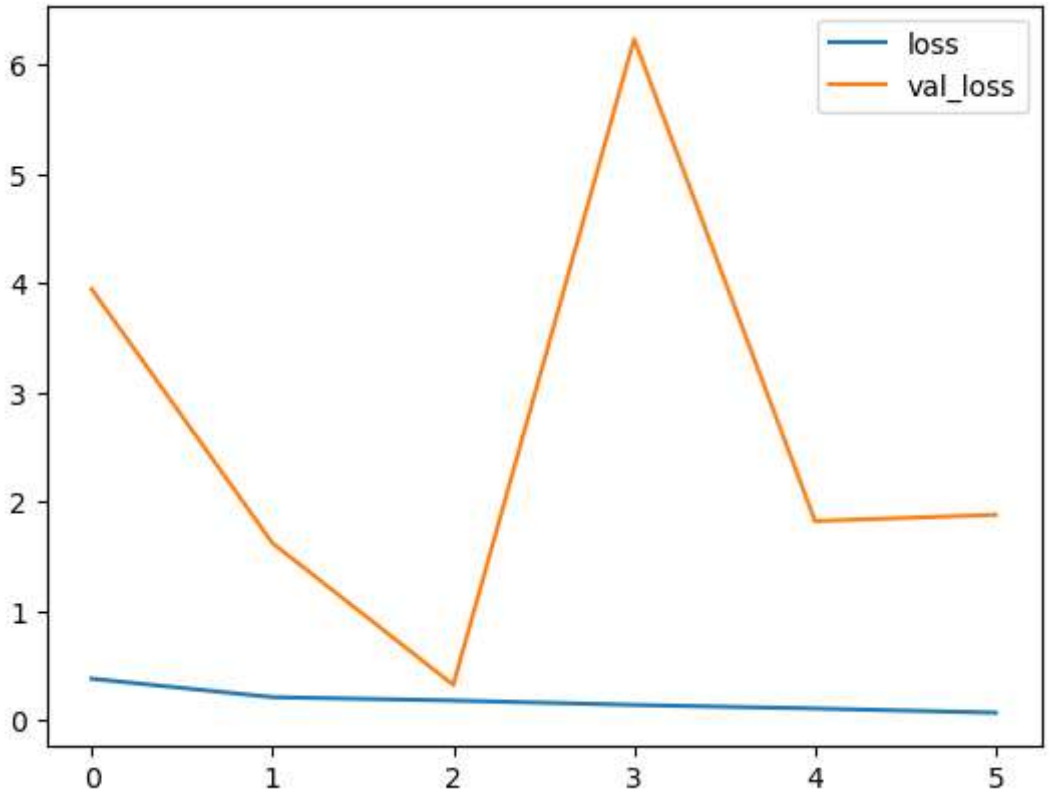
In [16]:
```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau


class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('val_accuracy') > 0.90:
            print('\n Validation accuracy has reached upto \
                      90% so, stopping further training.')
            self.model.stop_training = True


es = EarlyStopping(patience=3,
                   monitor='val_accuracy',
                   restore_best_weights=True)

lr = ReduceLROnPlateau(monitor='val_loss',
                       patience=2,
                       factor=0.5,
                       verbose=1)
```

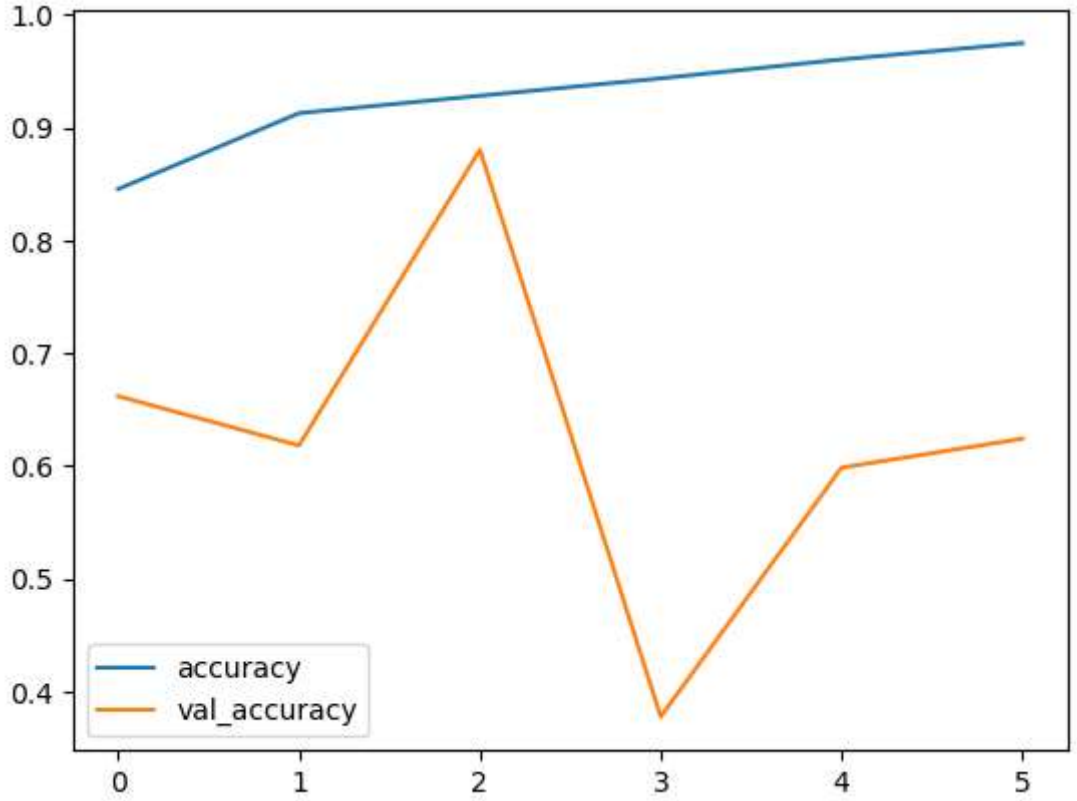
In [17]:
```
history = model.fit(X_train, Y_train,
                    validation_data = (X_val, Y_val),
                    batch_size = BATCH_SIZE,
                    epochs = EPOCHS,
                    verbose = 1,
                    callbacks = [es, lr, myCallback()])
```

```
Epoch 1/10
188/188 [==============================] - 636s 3s/step - loss: 0.3816 - accuracy: 0.8455 - val_loss: 3.9487 - val_accuracy: 0.6
620 - lr: 0.0010
Epoch 2/10
188/188 [==============================] - 630s 3s/step - loss: 0.2126 - accuracy: 0.9128 - val_loss: 1.6230 - val_accuracy: 0.6
183 - lr: 0.0010
Epoch 3/10
188/188 [==============================] - 625s 3s/step - loss: 0.1822 - accuracy: 0.9282 - val_loss: 0.3254 - val_accuracy: 0.8
803 - lr: 0.0010
Epoch 4/10
188/188 [==============================] - 593s 3s/step - loss: 0.1416 - accuracy: 0.9436 - val_loss: 6.2358 - val_accuracy: 0.3
780 - lr: 0.0010
Epoch 5/10
188/188 [==============================] - ETA: 0s - loss: 0.1091 - accuracy: 0.9603
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
188/188 [==============================] - 595s 3s/step - loss: 0.1091 - accuracy: 0.9603 - val_loss: 1.8243 - val_accuracy: 0.5
987 - lr: 0.0010
Epoch 6/10
188/188 [==============================] - 591s 3s/step - loss: 0.0706 - accuracy: 0.9748 - val_loss: 1.8819 - val_accuracy: 0.6
243 - lr: 5.0000e-04
```

In [19]:
```
model.save("model.h5")
```

In [18]:
```
history_df = pd.DataFrame(history.history)
history_df.loc[:,['loss','val_loss']].plot()
history_df.loc[:,['accuracy','val_accuracy']].plot()
plt.show()
```


loss
val_loss
6
5
4
3
2
1
0
0
1
2
3
4
5



In [20]:
```
Y_pred = model.predict(X_val)
Y_val = np.argmax(Y_val, axis=1)
Y_pred = np.argmax(Y_pred, axis=1)
```

```
94/94 [==============================] - 24s 246ms/step
```

In [21]:
```
metrics.confusion_matrix(Y_val, Y_pred)
```

Out[21]:
```
array([[690,  53, 244],
       [  1, 976,   0],
       [ 61,   0, 975]], dtype=int64)
```

In [22]:
```
print(metrics.classification_report(Y_val, Y_pred,
                                    target_names=classes))
```

	precision	recall	f1-score	support
lung_aca	0.92	0.70	0.79	987
lung_n	0.95	1.00	0.97	977
lung_scc	0.80	0.94	0.86	1036
accuracy			0.88	3000
macro avg	0.89	0.88	0.88	3000
weighted avg	0.89	0.88	0.88	3000

In [ ]:

In [ ]:

In [90]:
```
# Now, you can load the model for testing
loaded_model = tf.keras.models.load_model('model.h5')

# Use the loaded model for predictions on new images
# new_image_path = 'lungaca161.jpeg'
# new_image_path = 'lungscc30.jpeg'
# new_image_path = 'lungn29.jpeg'
new_image_path = 'lungscc44.jpeg'




new_image = tf.keras.preprocessing.image.load_img(new_image_path, target_size=(256, 256,3))
new_image = tf.keras.preprocessing.image.img_to_array(new_image)
new_image = tf.expand_dims(new_image, axis=0)
result = loaded_model.predict(new_image)


print(result[0][1])

if result[0][1] >= 1:
    prediction = 'Cancerous'
else:
    prediction = 'Non-Cancerous'

print(f'The prediction for the new image is: {prediction}')
```

```
1/1 [==============================] - 0s 171ms/step
1.0
The prediction for the new image is: Cancerous
```