Lung Cancer Detection

Suhail Shaikh 52

ADS:- 8

Aim: Illustrate data science lifecycle for selected case study.

Theory:

The data science project life cycle is a methodology that outlines the stages of a data science project, from planning to deployment. This methodology guides data scientists through a structured process that enables them to develop data-driven solutions that address specific business problems. The project life cycle provides a framework that helps data scientists to manage projects effectively and efficiently.

Step 1: Problem Identification and Planning

The first step in the data science project life cycle is to identify the problem that needs to be solved. This involves understanding the business requirements and the goals of the project. Once the problem has been identified, the data science team will plan the project by determining the data sources, the data collection process, and the analytical methods that will be used.

Case Study:
The objective is to create a Computer-Aided Diagnosis (CAD) system that can analyze an image and determine whether the lung CT scan depicted are normal or cancerous. Developing a machine learning model to categorize lung CT scan as normal or malignant exemplifies the application of deep neural networks in computer vision.

Step 2: Data Collection

The second step in the data science project life cycle is data collection. This involves collecting the data that will be used in the analysis. The data science team must ensure that the data is accurate, complete, and relevant to the problem being solved.

Case Study:
The dataset which we will use here has been taken from:
https://wiki.cancerimagingarchive.net/display/Public/SPIE-AAPM+Lung+CT+Challenge#19039197210adedc7a9841f2ad27255e18a2dafc

This dataset includes 3094 images for two classes of lung conditions:

- Normal Class
- Malignant Class

These images for each class contain 1547 images, hence both classes are balanced. That is why we won't be using Data Augmentation further on these images.

Step 3: Data Preparation

The third step in the data science project life cycle is data preparation. This involves cleaning and transforming the data to make it suitable for analysis. The data science team will remove any duplicates, missing values, or irrelevant data from the dataset. They will also transform the data into a format that is suitable for analysis.

Case Study:
In this section, we will convert the given images into NumPy arrays of their pixels after resizing them because training a Deep Neural Network on large-size images is highly inefficient in terms of computational cost and time.

For this purpose, we will use the OpenCV library and Numpy library of python to serve the purpose. Also, after all the images are converted into the desired format, we will split them into training and validation data so, that we can evaluate the performance of our model.

Step 4: Data Analysis

The fourth step in the data science project life cycle is data analysis. This involves applying analytical methods to the data to extract insights and patterns. The data science team may use techniques such as regression analysis, clustering, or machine learning algorithms to analyze the data.

Case study:
For lung cancer detection, we are going to use a CNN model for this Python-based data science project. A convolutional layer and pooling layers are the two pillars of a CNN model. The reason behind the success of CNN for image classification problems is its compatibility with grid-structured data.

Step 5: Model Building

The fifth step in the data science project life cycle is model building. This involves building a predictive model that can be used to make predictions based on the data analysis. The data science team will use the insights and patterns from the data analysis to build a model that can predict future outcomes.

Case Study:
From this step onward we will use the TensorFlow library to build our CNN model. Keras framework of the tensor flow library contains all the functionalities that one may need to define the architecture of a Convolutional Neural Network and train it on the data.

Model Architecture
We will implement a Sequential model which will contain the following parts:

- Three Convolutional Layers followed by MaxPooling Layers.
- The Flatten layer to flatten the output of the convolutional layer.
- Then we will have two fully connected layers followed by the output of the flattened layer.

- We have included some BatchNormalization layers to enable stable and fast training and a Dropout layer before the final layer to avoid any possibility of overfitting.
- The final layer is the output layer which outputs soft probabilities for the three classes.

Step 6: Model Evaluation

The sixth step in the data science project life cycle is model evaluation. This involves evaluating the performance of the predictive model to ensure that it is accurate and reliable. The data science team will test the model using a validation dataset to determine its accuracy and performance.

Case Study:
Now as we have our model ready let's evaluate its performance on the validation data using different metrics. For this purpose, we will first predict the class for the validation data using this model and then compare the output with the true labels.

Step 7: Model Deployment

The final step in the data science project life cycle is model deployment. This involves deploying the predictive model into production so that it can be used to make predictions in real-world scenarios. The deployment process involves integrating the model into the existing business processes and systems to ensure that it can be used effectively.

Conclusion:

- The data science project lifecycle offers a systematic framework for developing data-driven solutions tailored to business needs. In tackling the challenge of lung cancer detection, our approach leverages machine learning and image processing techniques to accurately classify CT scans as malignant or normal.
- By following the lifecycle steps of problem identification, data collection, preparation, analysis, model building, evaluation, and deployment, we've constructed a CNN model capable of effectively distinguishing between malignant and normal CT scan images.
- This project underscores the potential of data science methodologies in enhancing medical diagnostics and underscores the significance of rigorous lifecycle adherence in developing robust solutions for real-world applications.

In [1]:
```
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')
```

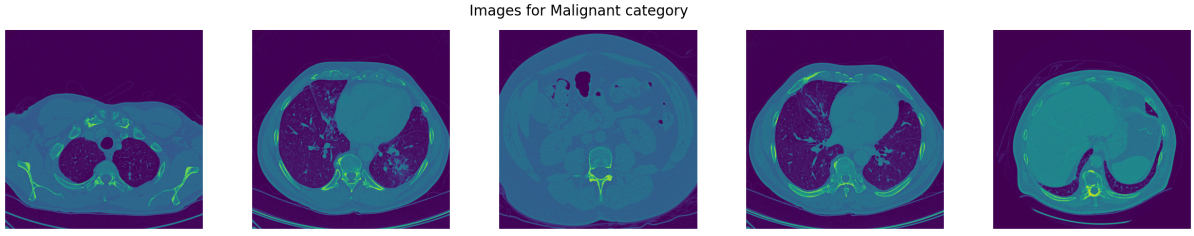
In [2]:
```
from zipfile import ZipFile

data_path = '/content/drive/MyDrive/ADS_Exp_8/Lung_Image_Set.zip'

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

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

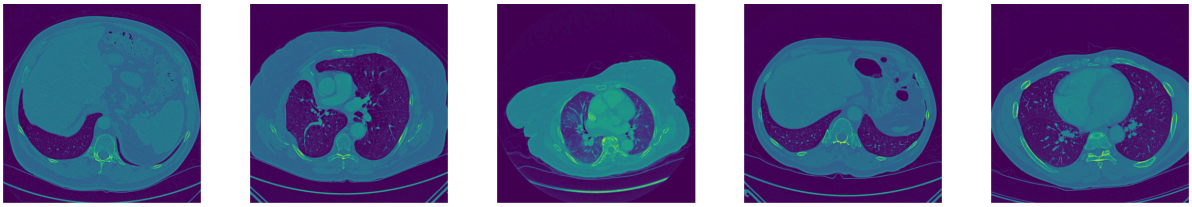
In [3]:
```
path = '/content/Lung_Image_Set'
classes = os.listdir(path)
classes
```

Out[3]:
```
['Malignant', 'Normal']
```

In [4]:
```
path = '/content/Lung_Image_Set'

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

        fig, ax = plt.subplots(1, 5, figsize=(30, 5))
        fig.suptitle(f'Images for {cat} category', fontsize=20)

        for i in range(5):
                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 Normal category



In [5]:
```
IMG_SIZE = 256
SPLIT = 0.3
EPOCHS = 10
BATCH_SIZE = 256
```

In [6]:
```
X = []
Y = []

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

  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 [7]:
```
X_train, X_val, Y_train, Y_val = train_test_split(X, one_hot_encoded_Y,
                                                  test_size = SPLIT,random_state =
```

In [8]:
```
model = keras.models.Sequential([
        layers.Conv2D(filters=32, kernel_size=(5, 5), activation='relu', input_shap
               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(2, activation='softmax')])
```

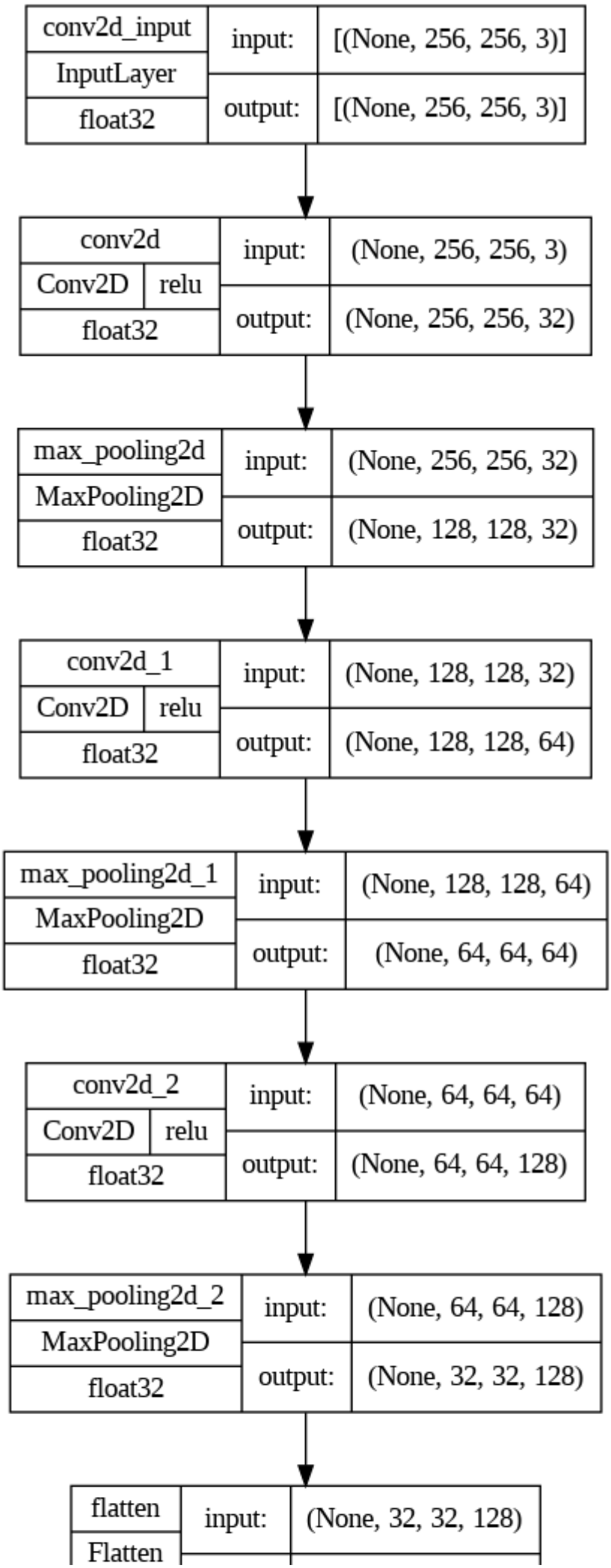
In [9]:
```
model.summary()
```

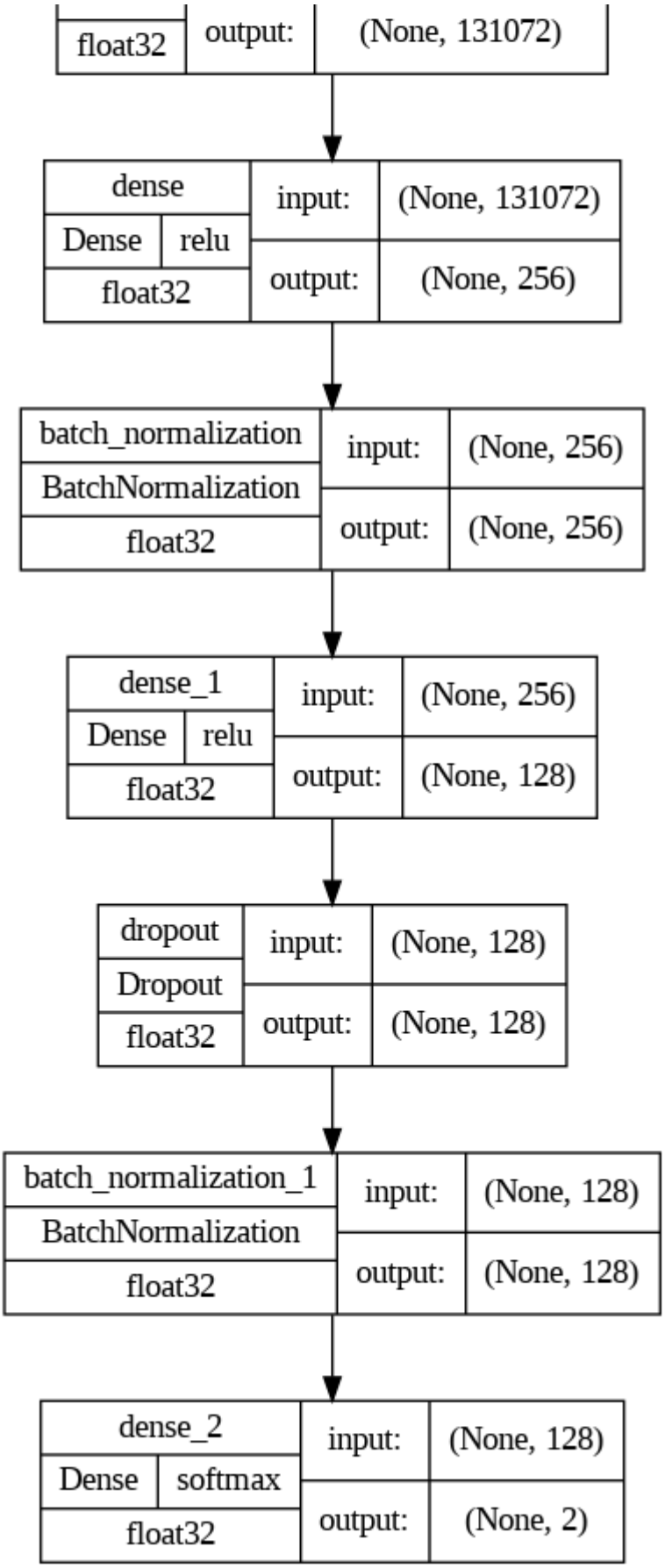
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      2432

 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, 2)                 258

=================================================================
Total params: 33684162 (128.49 MB)
Trainable params: 33683394 (128.49 MB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

In [10]:

```
keras.utils.plot_model(
        model,
        show_shapes = True,
        show_dtype = True,
        show_layer_activations = True
)
```

Out[10]:


conv2d_input
InputLayer
float32
input: [(None, 256, 256, 3)]
output: [(None, 256, 256, 3)]
conv2d
Conv2D relu
float32
input: (None, 256, 256, 3)
output: (None, 256, 256, 32)
max_pooling2d
MaxPooling2D
float32
input: (None, 256, 256, 32)
output: (None, 128, 128, 32)
conv2d_1
Conv2D relu
float32
input: (None, 128, 128, 32)
output: (None, 128, 128, 64)
max_pooling2d_1
MaxPooling2D
float32
input: (None, 128, 128, 64)
output: (None, 64, 64, 64)
conv2d_2
Conv2D relu
float32
input: (None, 64, 64, 64)
output: (None, 64, 64, 128)
max_pooling2d_2
MaxPooling2D
float32
input: (None, 64, 64, 128)
output: (None, 32, 32, 128)
flatten
Flatten
input: (None, 32, 32, 128)



In [11]:

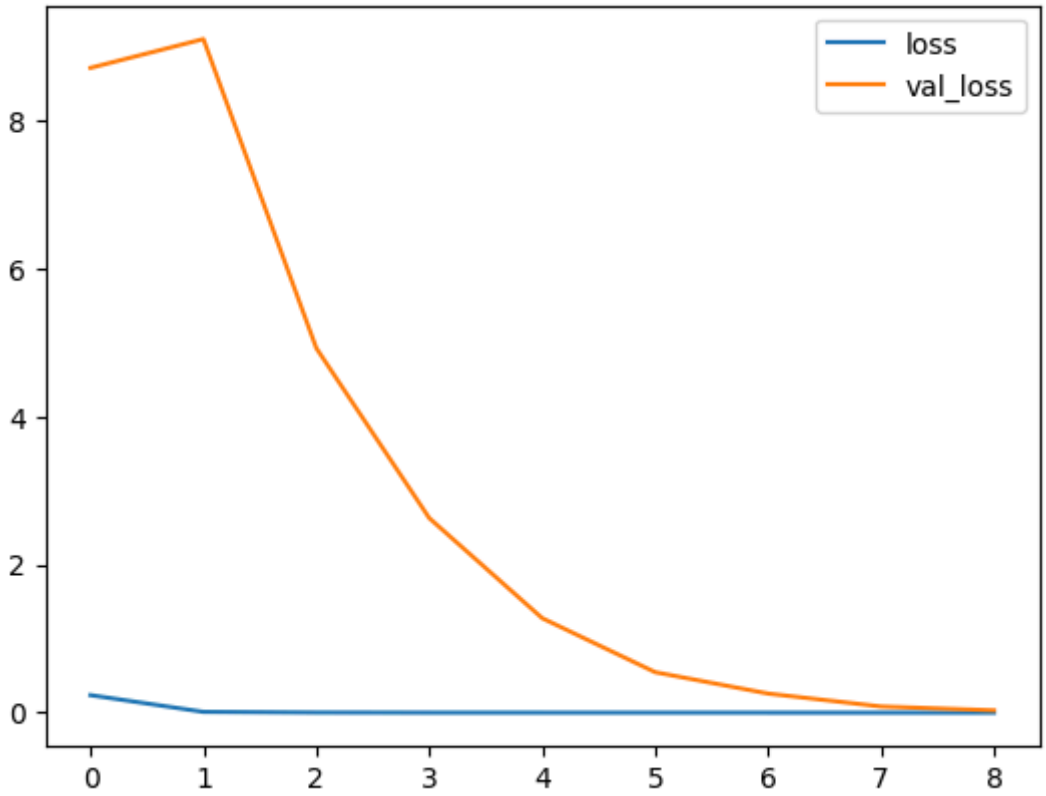
```
model.compile(
        optimizer = 'adam',
        loss = 'categorical_crossentropy',
        metrics = ['accuracy']
)
```

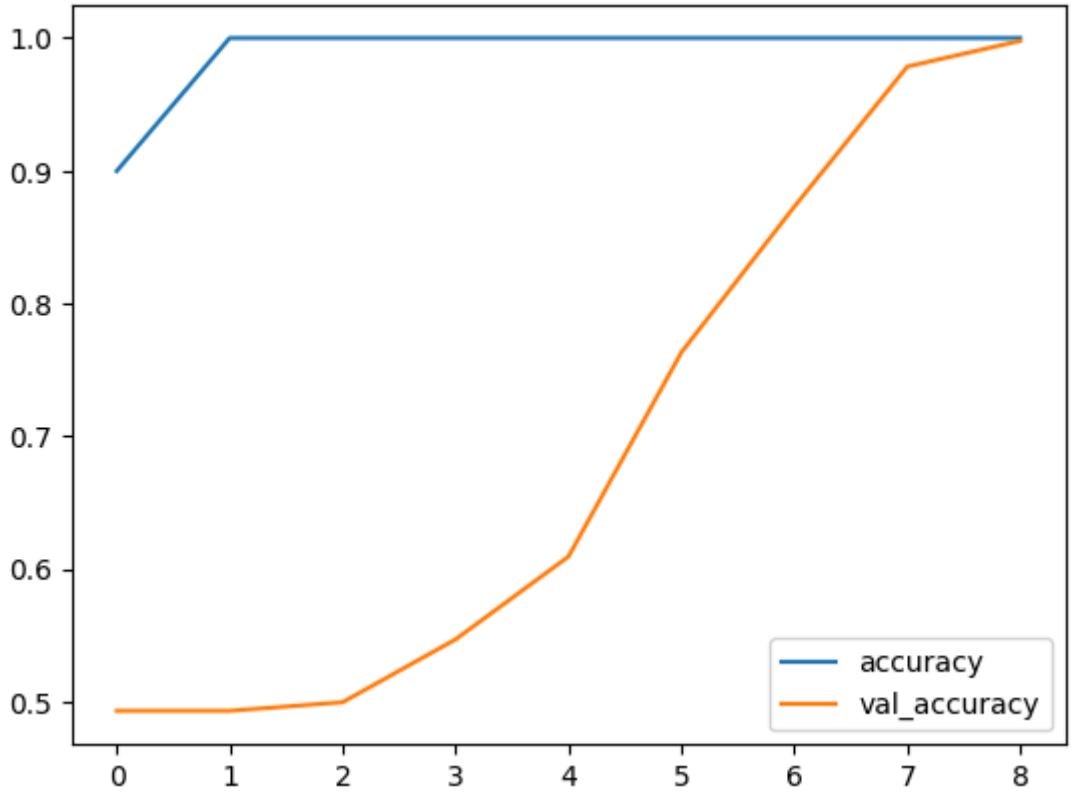
In [12]:
```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

class myCallback(tf.keras.callbacks.Callback):
        def on_epoch_end(self, epoch, logs={}):
                if logs.get('val_accuracy') > 0.98:
                        print('\n Validation accuracy has reached upto \
                                        98% 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 [13]:
```
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
6/9 [==================>.....] - ETA: 1s - loss: 0.3203 - accuracy: 0.8613
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the ba
tch time (batch time: 0.1233s vs `on_train_batch_end` time: 0.2986s). Check your c
allbacks.
9/9 [==============================] - 39s 2s/step - loss: 0.2375 - accuracy: 0.89
98 - val_loss: 8.7082 - val_accuracy: 0.4930 - lr: 0.0010
Epoch 2/10
9/9 [==============================] - 5s 528ms/step - loss: 0.0112 - accuracy: 1.
0000 - val_loss: 9.0971 - val_accuracy: 0.4930 - lr: 0.0010
Epoch 3/10
9/9 [==============================] - 5s 555ms/step - loss: 0.0027 - accuracy: 1.
0000 - val_loss: 4.9232 - val_accuracy: 0.4995 - lr: 0.0010
Epoch 4/10
9/9 [==============================] - 5s 562ms/step - loss: 8.1910e-04 - accurac
y: 1.0000 - val_loss: 2.6281 - val_accuracy: 0.5468 - lr: 0.0010
Epoch 5/10
9/9 [==============================] - 5s 550ms/step - loss: 5.9900e-04 - accurac
y: 1.0000 - val_loss: 1.2766 - val_accuracy: 0.6093 - lr: 0.0010
Epoch 6/10
9/9 [==============================] - 5s 548ms/step - loss: 4.6706e-04 - accurac
y: 1.0000 - val_loss: 0.5472 - val_accuracy: 0.7632 - lr: 0.0010
Epoch 7/10
9/9 [==============================] - 5s 556ms/step - loss: 3.0355e-04 - accurac
y: 1.0000 - val_loss: 0.2583 - val_accuracy: 0.8730 - lr: 0.0010
Epoch 8/10
9/9 [==============================] - 5s 530ms/step - loss: 3.0384e-04 - accurac
y: 1.0000 - val_loss: 0.0839 - val_accuracy: 0.9785 - lr: 0.0010
Epoch 9/10
9/9 [==============================] - ETA: 0s - loss: 2.3818e-04 - accuracy: 1.00
00
 Validation accuracy has reached upto                                    98% so, st
opping further training.
9/9 [==============================] - 6s 700ms/step - loss: 2.3818e-04 - accurac
y: 1.0000 - val_loss: 0.0317 - val_accuracy: 0.9978 - lr: 0.0010
```

In [14]:
```
history_df = pd.DataFrame(history.history)
history_df.loc[:,['loss','val_loss']].plot()
history_df.loc[:,['accuracy','val_accuracy']].plot()
plt.show()
```





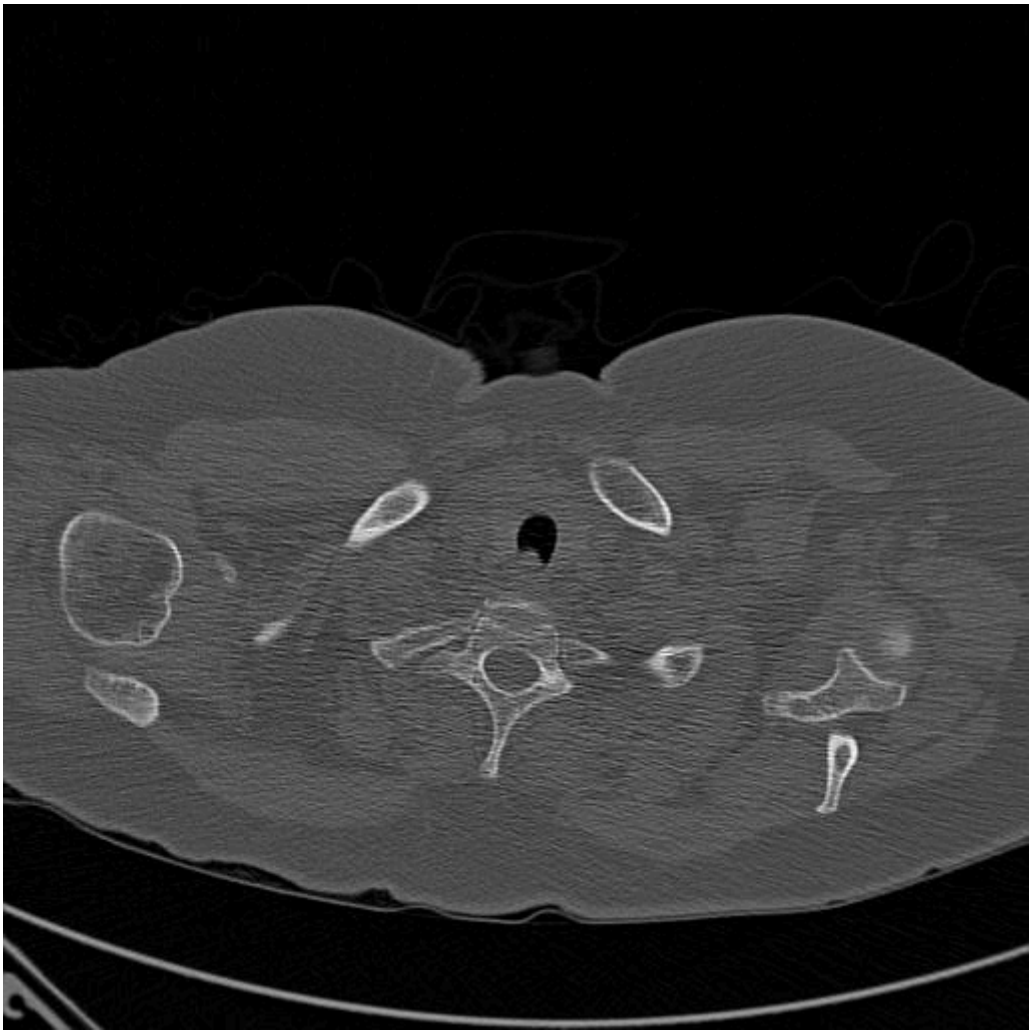
In [15]:
```
Y_pred = model.predict(X_val)
Y_val = np.argmax(Y_val, axis=1)
Y_pred = np.argmax(Y_pred, axis=1)
```

```
30/30 [==============================] - 2s 23ms/step
```

In [16]:
```
metrics.confusion_matrix(Y_val, Y_pred)
```

Out[16]:
```
array([[469,   2],
       [  0, 458]])
```

In [17]:
```
from google.colab.patches import cv2_imshow

user_img = cv2.imread("/content/M1.jpg")
X_img = cv2.resize(user_img, (IMG_SIZE, IMG_SIZE))
X_img = np.asarray(X_img)
X_img = np.reshape(X_img, [1, 256, 256, 3])
cv2_imshow(user_img)
predict_x = model.predict(X_img)
classes_x = np.argmax(predict_x, axis=1)
ans = classes_x[0]
if ans == 0:
    print('Malignant')
else:
    print('Normal')
```



```
1/1 [==============================] - 0s 20ms/step
Malignant
```