lyearprojectlungcancerdetection-4

April 26, 2024

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
%cd /content/drive/MyDrive/1_FinalYearProjectLungCancerDetection
```

```
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection
```

[ ]:
```
#!unzip IQ-OTHNCCD_dataset.zip -d dataset
```

[ ]:
```
import os
import imageio
import numpy as np
import shutil
import pandas as pd
import cv2

import keras
import matplotlib.pyplot as plt # for plotting
import os # provides a way of using operating system dependent functionality
import cv2 #Image handling library
import numpy as np
```

[ ]:
```
dir = '/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/The␣
↪IQ-OTHNCCD lung cancer dataset'
categories = sorted(os.listdir(dir))
print(categories)
len(categories)
```

```
['Bengin cases', 'Malignant cases', 'Normal cases']
```

[ ]:
```
3
```

[ ]:
```
categories = sorted(os.listdir(dir))
print(categories)
len(categories)
```

```
['Bengin cases', 'Malignant cases', 'Normal cases']
```

[ ]:
```
3
```

[ ]:
```
'''
def train_test_split():
    print("############ Train Test Script started ############")
    root_dir = '/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/
 ↪Dataset/splitted_dataset'
    classes_dir = ['Bengin cases', 'Malignant cases', 'Normal cases']
    processed_dir = '/content/drive/MyDrive/
 ↪1_FinalYearProjectLungCancerDetection/Dataset/The IQ-OTHNCCD lung cancer␣
 ↪dataset'
    test_ratio = 0.10

    for cls in classes_dir:
        # Creating partitions of the data after shuffeling
        print("$$$ Class Name " + cls + " $$$")
        src = processed_dir +"/" + cls  # Folder to copy images from

        allFileNames = os.listdir(src)
        np.random.shuffle(allFileNames)
        train_FileNames, test_FileNames = np.split(np.array(allFileNames),␣
 ↪[int(len(allFileNames) * (1 - test_ratio)) ])

        train_FileNames = [src + '/' + name for name in train_FileNames.
 ↪tolist()]
        test_FileNames = [src + '/' + name for name in test_FileNames.tolist()]

        print('Total images: '+ str(len(allFileNames)))
        print('Training: '+ str(len(train_FileNames)))
        print('Testing: '+ str(len(test_FileNames)))

        # # Creating Train Test folders (One time use)
        os.makedirs(root_dir + '/train/' + cls)
        os.makedirs(root_dir + '/test/' + cls)

        # Copy-pasting images
        for name in train_FileNames:
            shutil.copy(name, root_dir + '/train/' + cls)

        for name in test_FileNames:
            shutil.copy(name, root_dir + '/test/' + cls)

    print("############ Train Test Script Ended ############")

train_test_split()
```

```
'''
```

```
############ Train Test Script started ############
$$$ Class Name Bengin cases $$$
Total images: 120
Training: 108
Testing: 12
$$$ Class Name Malignant cases $$$
Total images: 561
Training: 504
Testing: 57
$$$ Class Name Normal cases $$$
Total images: 416
Training: 374
Testing: 42
############ Train Test Script Ended ############
```

[ ]:
```
dir1 = '/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/
 ↪splitted_dataset'
cat1 = sorted(os.listdir(dir1))
print(cat1)
```

```
['test', 'train']
```

[ ]:
```
data_path = '/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/
 ↪Dataset/splitted_dataset/train'
categories  = sorted(os.listdir(data_path))
print(categories )
```

```
['Bengin cases', 'Malignant cases', 'Normal cases']
```

[ ]:
```
size_data = {}
for i in categories:
    path = os.path.join(data_path, i)
    print(path)
    class_num = categories.index(i)
    print(class_num)
    temp_dict = {}
    for file in os.listdir(path):
        filepath = os.path.join(path, file)
        height, width, channels = imageio.imread(filepath).shape
        if str(height) + ' x ' + str(width) in temp_dict:
            temp_dict[str(height) + ' x ' + str(width)] += 1
        else:
            temp_dict[str(height) + ' x ' + str(width)] = 1

    size_data[i] = temp_dict
```

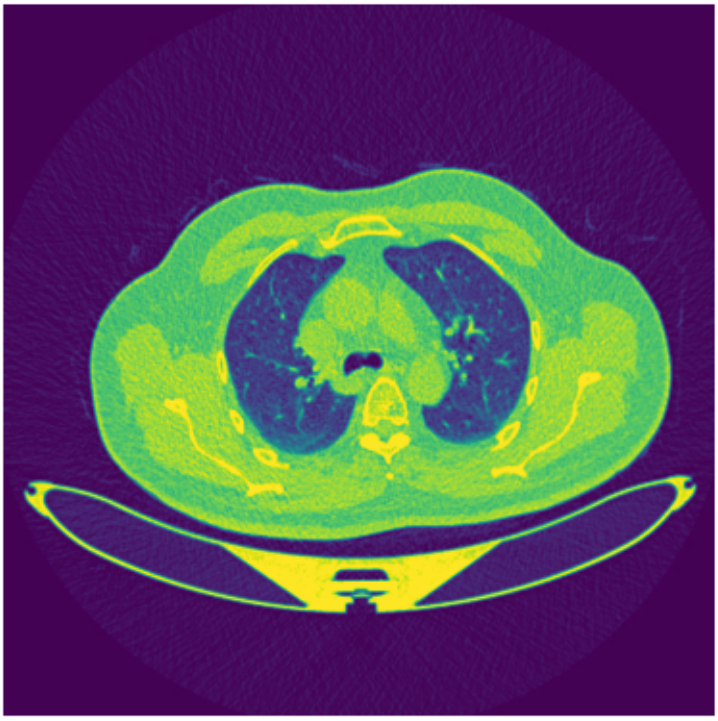
```
size_data
```

```
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/train/Bengin cases
0
<ipython-input-8-371b1566016e>:10: DeprecationWarning: Starting with ImageIO v3
the behavior of this function will switch to that of iio.v3.imread. To keep the
current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  height, width, channels = imageio.imread(filepath).shape
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/train/Malignant cases
1
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/train/Normal cases
2
```

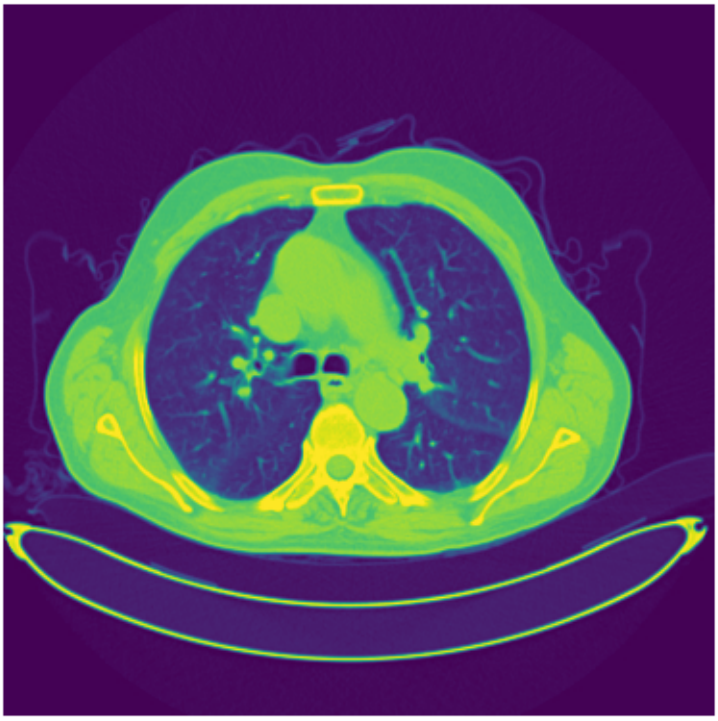
[ ]:
```
{'Bengin cases': {'512 x 512': 108},
 'Malignant cases': {'512 x 512': 455,
  '512 x 623': 26,
  '512 x 801': 22,
  '404 x 511': 1},
 'Normal cases': {'512 x 512': 373, '331 x 506': 1}}
```

[ ]:
```
for i in categories:
    path = os.path.join(data_path, i)
    class_num = categories.index(i)
    for file in os.listdir(path):
        filepath = os.path.join(path, file)
        print(i)
        img = cv2.imread(filepath, 0)
        plt.imshow(img)
        plt.axis('off')
        plt.show()
        break
```

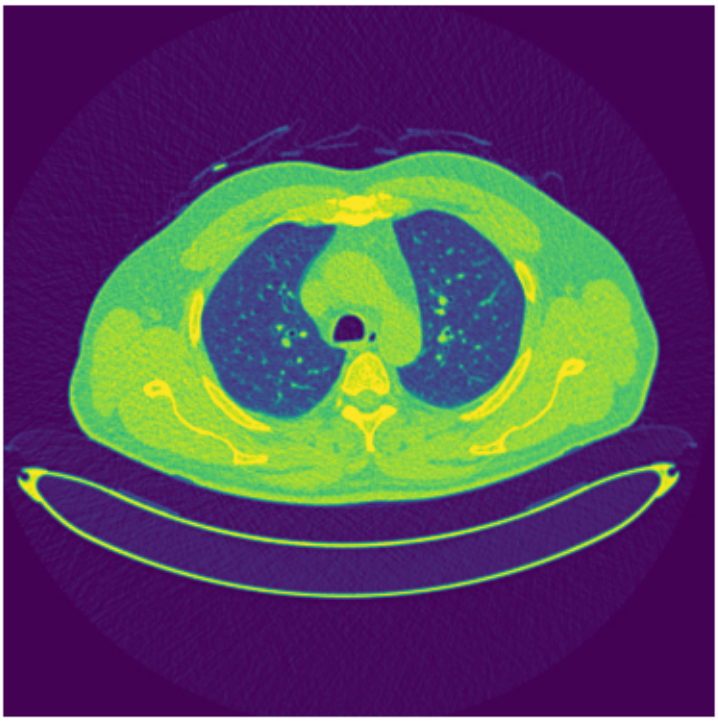
```
Bengin cases
```



Malignant cases



Normal cases



[ ]:
```
# Loading the images and their class(0 - 2)
image_data = []
img_size=256
for i in categories:
    path = os.path.join(data_path, i)
    print(path)
    class_num = categories.index(i)
    print(class_num)
    for file in os.listdir(path):
        filepath = os.path.join(path, file)
        img_arr = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)
        # preprocess here
        img = cv2.resize(img_arr, (img_size, img_size))
        image_data.append([img, class_num])
```

```
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/train/Bengin cases
0
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/train/Malignant cases
```

```
1
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/train/Normal cases
2
```

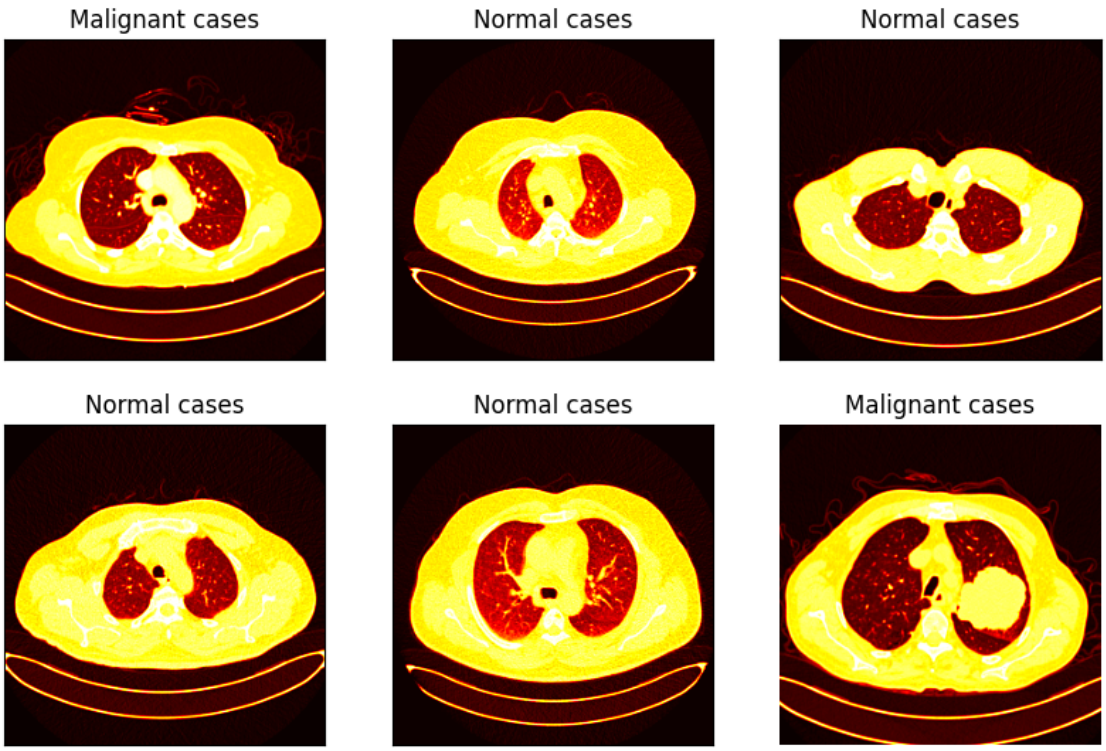
[ ]:
```
# shuffle the input data
import random
random.shuffle(image_data)
```

[ ]:
```
input_data = []
label = []
for X, y in image_data:
    input_data.append(X)
    label.append(y)
```

[ ]:
```
label[:7]
```

[ ]:
```
[1, 1, 2, 2, 2, 2, 1]
```

[ ]:
```
plt.figure(1, figsize=(10,10))
for i in range(1,7):
    plt.subplot(3,3,i)
    plt.imshow(image_data[i][0], cmap='hot')
    plt.xticks([])
    plt.yticks([])
    plt.title(categories[label[i]])
plt.axis('off')
plt.show()
```



[ ]:
```
print( "Image in form of array:\n" , input_data[0])
print( "Image label: " , label[0])
```

```
Image in form of array:
 [[38 38 38 … 38 38 38]
 [38 38 38 … 38 38 38]
 [38 38 38 … 38 38 38]
 …
 [38 38 38 … 38 38 38]
 [38 38 38 … 38 38 38]
 [38 38 38 … 38 38 38]]
Image label:  1
```

[ ]:
```
# Normalizing the data
input_data = np.array(input_data)
label = np.array(label)
input_data = input_data/255.0
input_data.shape
```

[ ]:
```
(986, 256, 256)
```

[ ]:
```
# one hot encoding
from tensorflow.keras.utils import to_categorical
```

```
label = to_categorical(label, num_classes=3,dtype='i1')
label[0]
```

[ ]:
```
array([0, 1, 0], dtype=int8)
```

[ ]:
```
# reshaping the data
input_data.shape = (-1, img_size,img_size, 1)
```

[ ]:
```
input_data.shape
```

[ ]:
```
(986, 256, 256, 1)
```

[ ]:
```
# splitting the input_data to train and test data
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(input_data, label,␣
 ↪random_state=10, test_size = 0.2)

print(len(X_train), X_train.shape)
print(len(X_valid), X_valid.shape)
```

```
788 (788, 256, 256, 1)
198 (198, 256, 256, 1)
```

[ ]:
```
print("Found {} training images belonging to 3 classes".format(X_train.
 ↪shape[0]))
print("Found {} testing images belonging to 3 classes".format(X_valid.shape[0]))
```

```
Found 788 training images belonging to 3 classes
Found 198 testing images belonging to 3 classes
```

[ ]:
```
input_foldr = '/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/
 ↪Dataset/splitted_dataset/train'
files_per_class = []
for folder in os.listdir(input_foldr):
    if not os.path.isfile(folder):
            files_per_class.append(len(os.listdir(input_foldr + '/' + folder)))

files_per_class
```

[ ]:
```
[108, 504, 374]
```

[ ]:
```
total_files = sum(files_per_class)
print(total_files)
class_weights = {}
for i in range(len(files_per_class)):
    class_weights[i] = 1 - (float(files_per_class[i]) / total_files)
print(class_weights)
```

```
986
{0: 0.8904665314401623, 1: 0.4888438133874239, 2: 0.6206896551724138}
```

[ ]:
```
#Building Model
```

[ ]:
```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dropout,Flatten,Dense,Activation
from tensorflow.keras.optimizers import SGD
from keras import losses
```

[57]:
```
# Step 1 - Building the CNN

# Initializing the CNN
model = Sequential()

# First convolution layer and pooling
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256,256, 1)))
model.add(MaxPooling2D((2, 2)))

# Second convolution layer and pooling
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Third convolution layer
model.add(Conv2D(128, (3, 3), activation='relu'))

# input_shape is going to be the pooled feature maps from the previous␣
 ↪convolution layer
model.add(MaxPooling2D((2, 2)))
# Flattening the layers
model.add(Flatten())

# Adding a fully connected layer
model.add(Dense(256, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

[ ]:
```
# Included libraries for PSO and CNN model setup
import numpy as np
import random
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation
from tensorflow.keras.optimizers import Adam

# PSO parameters and helper functions
num_particles = 10
num_iterations = 20
num_dimensions = 3
```

```
bounds = [(16, 64), (32, 128), (0.0001, 0.01)]

positions = np.zeros((num_particles, num_dimensions))
velocities = np.zeros_like(positions)
personal_best_positions = np.zeros_like(positions)
personal_best_scores = np.full(num_particles, np.inf)
global_best_position = None
global_best_score = np.inf

def evaluate(position):
    model = build_model(int(position[0]), int(position[1]), position[2])
    return train_and_evaluate(model)

def update_bests():
    global global_best_position, global_best_score
    for i in range(num_particles):
        score = evaluate(positions[i])
        if score < personal_best_scores[i]:
            personal_best_scores[i] = score
            personal_best_positions[i] = positions[i].copy()
        if score < global_best_score:
            global_best_score = score
            global_best_position = positions[i].copy()

def build_model(filters1, filters2, learning_rate):
    model = Sequential()
    model.add(Conv2D(filters1, (3, 3), activation='relu', input_shape=(256,256,␣
 ↪1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(filters2, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,␣
 ↪metrics=['accuracy'])
    return model

# Initialize PSO
for i in range(num_particles):
    positions[i] = np.array([random.uniform(*bounds[j]) for j in␣
 ↪range(num_dimensions)])
    velocities[i] = np.array([random.uniform(-(bounds[j][1] - bounds[j][0]),␣
 ↪(bounds[j][1] - bounds[j][0])) for j in range(num_dimensions)])
```

```
    personal_best_positions[i] = positions[i].copy()
    personal_best_scores[i] = evaluate(positions[i])
global_best_position = personal_best_positions[np.argmin(personal_best_scores)]
global_best_score = min(personal_best_scores)

# PSO loop
for _ in range(num_iterations):
    for i in range(num_particles):
        velocities[i] = 0.5 * velocities[i] + 0.5 * (personal_best_positions[i]␣
 ↪- positions[i]) + 0.5 * (global_best_position - positions[i])
        positions[i] += velocities[i]
        positions[i] = np.clip(positions[i], [b[0] for b in bounds], [b[1] for␣
 ↪b in bounds])
    update_bests()

print('Best Position:', global_best_position)
print('Best Score:', global_best_score)
```

Best Hyperparameters Identified:

Filters in Conv1 Layer: Approximately 23 filters

Filters in Conv2 Layer: Approximately 115 filters

Learning Rate: Approximately 0.0028

[58]:
```
model.compile(loss='categorical_crossentropy',
              optimizer = 'Adam',
              metrics = ['accuracy'])
```

[ ]:
```
# fit model
history = model.fit(X_train, y_train, steps_per_epoch= len(X_train)//64,
validation_data= (X_valid, y_valid), validation_steps= len(X_valid)//64,␣
 ↪epochs=6, verbose=2, shuffle=True,class_weight=class_weights)
```

```
Epoch 1/6
12/12 - 132s - loss: 1.2016 - accuracy: 0.4099 - val_loss: 0.9354 -
val_accuracy: 0.6818 - 132s/epoch - 11s/step
Epoch 2/6
12/12 - 120s - loss: 0.5386 - accuracy: 0.6104 - val_loss: 0.7029 -
val_accuracy: 0.6667 - 120s/epoch - 10s/step
Epoch 3/6
12/12 - 125s - loss: 0.4150 - accuracy: 0.7310 - val_loss: 0.4805 -
val_accuracy: 0.7929 - 125s/epoch - 10s/step
Epoch 4/6
12/12 - 122s - loss: 0.2675 - accuracy: 0.8464 - val_loss: 0.2657 -
val_accuracy: 0.9444 - 122s/epoch - 10s/step
Epoch 5/6
12/12 - 131s - loss: 0.1433 - accuracy: 0.9353 - val_loss: 0.2873 -
```

```
val_accuracy: 0.8737 - 131s/epoch - 11s/step
Epoch 6/6
12/12 - 124s - loss: 0.0945 - accuracy: 0.9543 - val_loss: 0.1644 -
val_accuracy: 0.9545 - 124s/epoch - 10s/step
```

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

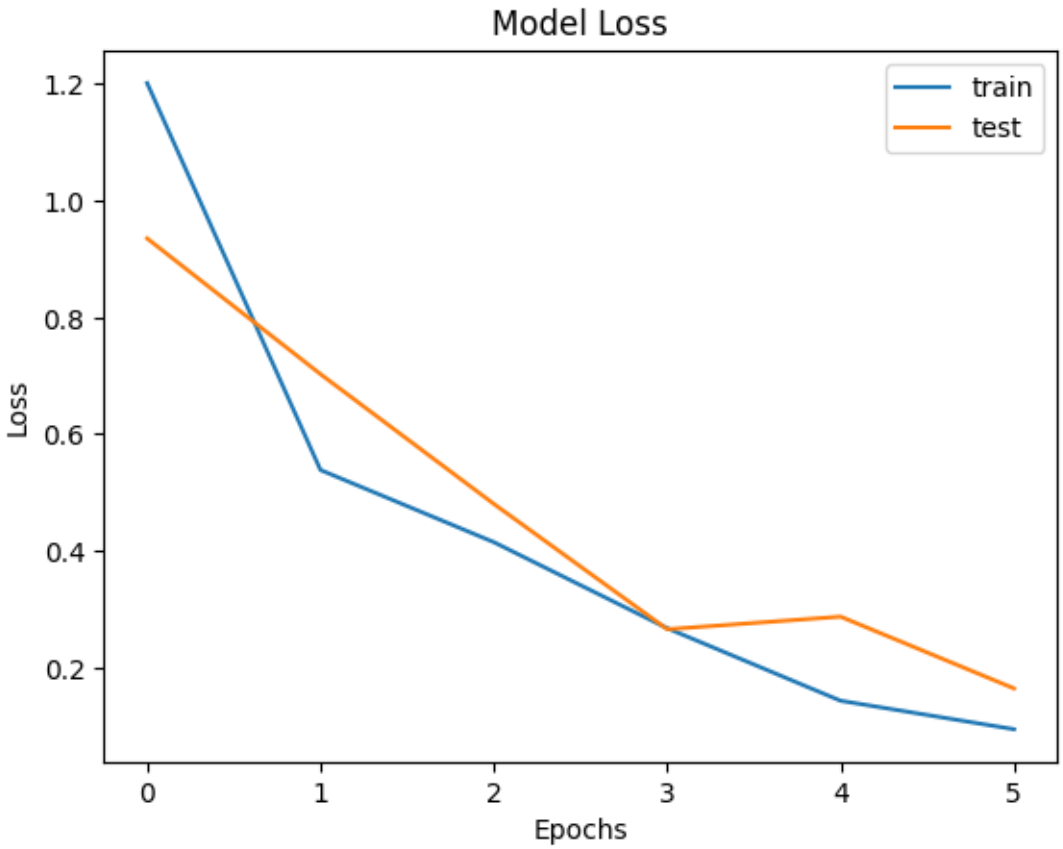
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      320

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 256)               29491456

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 29584899 (112.86 MB)
Trainable params: 29584899 (112.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
model.save('cnn_model1.keras')
```

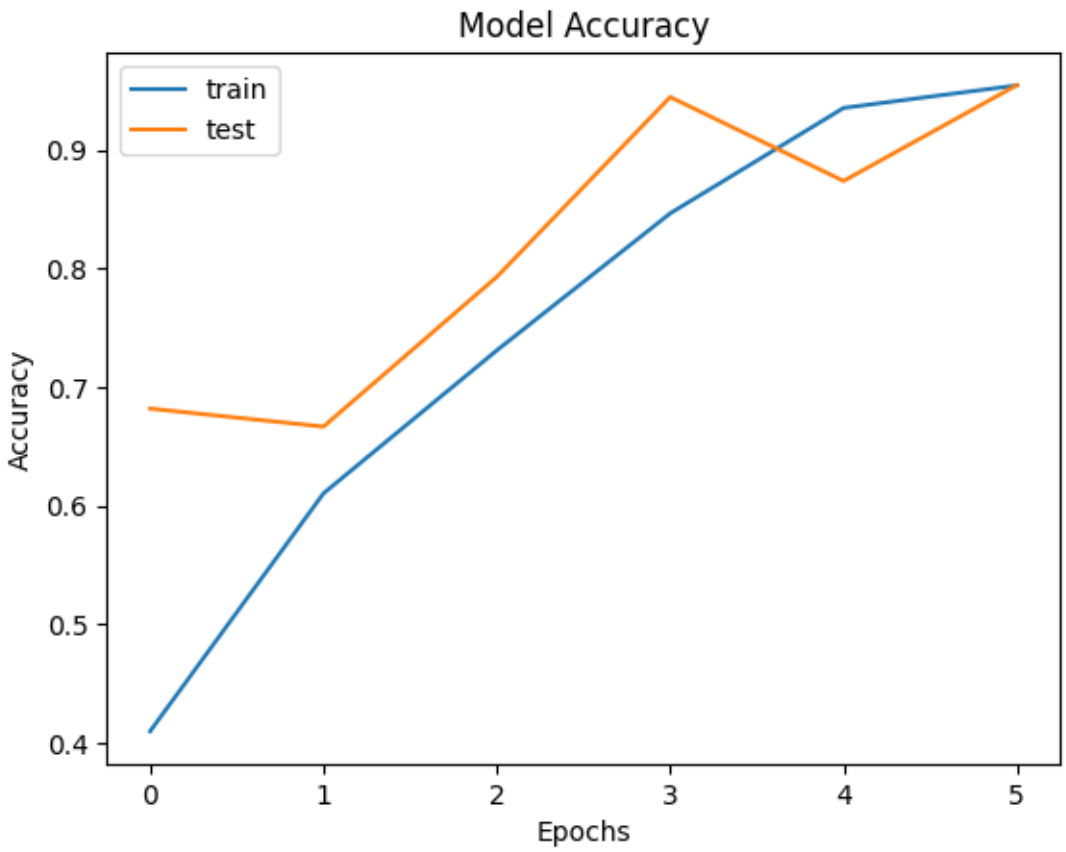
[ ]:
```
model_json = model.to_json()
with open("cnn_model1.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights('cnn_model1_weights.h5')
```

[ ]:
```
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
```

```
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
# save the figure
plt.savefig('loss_plot1.png', dpi=300, bbox_inches='tight')
plt.show()
```



[ ]:
```
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
# save the figure
plt.savefig('accuracy1.png', dpi=300, bbox_inches='tight')
plt.show()
```



[ ]:
```
#calculate loss and accuracy on test data

test_loss, test_accuracy = model.evaluate(X_valid, y_valid)

print('Test accuracy: {:2.2f}%'.format(test_accuracy*100))
```

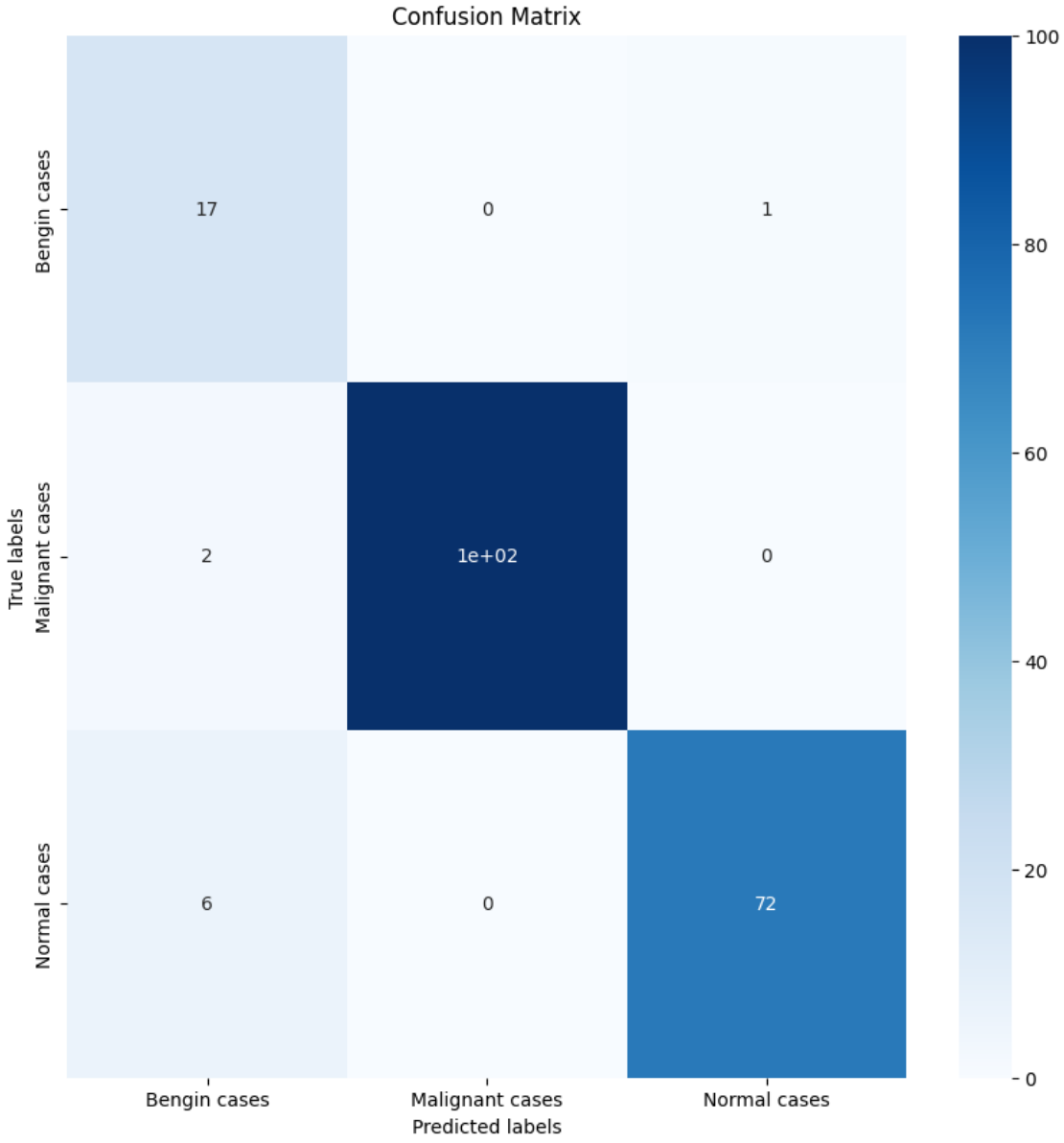
```
7/7 [==============================] - 9s 1s/step - loss: 0.1644 - accuracy:
0.9545
Test accuracy: 95.45%
```

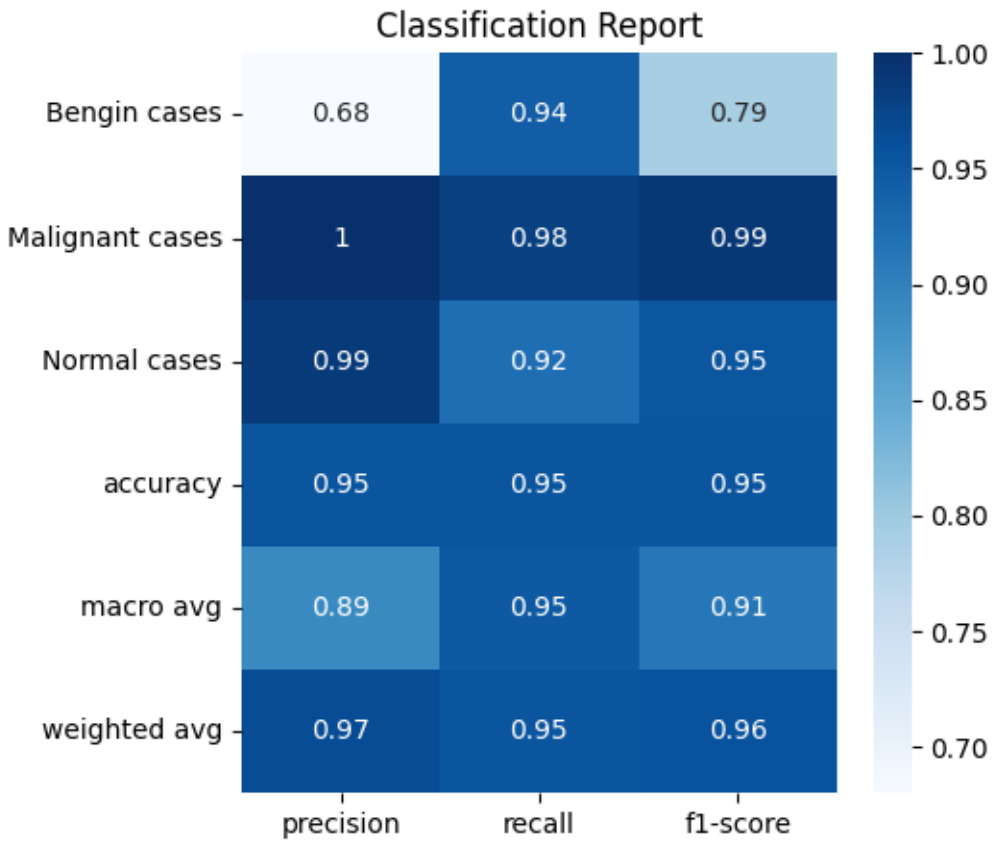
[ ]:
```
#calculate loss and accuracy on train data

train_loss, train_accuracy = model.evaluate(X_train, y_train)

print('Train accuracy: {:2.2f}%'.format(train_accuracy*100))
```

```
25/25 [==============================] - 30s 1s/step - loss: 0.1508 - accuracy:
0.9683
Train accuracy: 96.83%
```

[ ]:
```
from sklearn.metrics import␣
 ↪confusion_matrix,classification_report,accuracy_score
import seaborn as sns

def PerformanceReports(conf_matrix,class_report,labels):
    f, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(conf_matrix, annot=True,ax=ax,cmap="Blues")
    #labels, title and ticks
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix')
    ax.xaxis.set_ticklabels(labels)
    ax.yaxis.set_ticklabels(labels)
    plt.savefig('cmatrix1.png', dpi=300, bbox_inches='tight')
    plt.show()
    f, ax = plt.subplots(figsize=(5,5))
    sns.heatmap(pd.DataFrame(class_report).iloc[:-1, :].T,␣
 ↪annot=True,ax=ax,cmap="Blues")
    ax.set_title('Classification Report')
    plt.savefig('creport1.png', dpi=300, bbox_inches='tight')
    plt.show()
labels=['Bengin cases', 'Malignant cases', 'Normal cases']

test_labels = np.argmax(y_valid, axis=1)
predictions = np.argmax(model.predict(X_valid), axis=1)

cm=confusion_matrix(test_labels,predictions)
cr=classification_report(test_labels, predictions)
class_report=classification_report(test_labels, predictions,
                                   target_names=labels,
                                   output_dict=True)
PerformanceReports(cm,class_report,labels)
```

```
7/7 [==============================] - 9s 1s/step
```

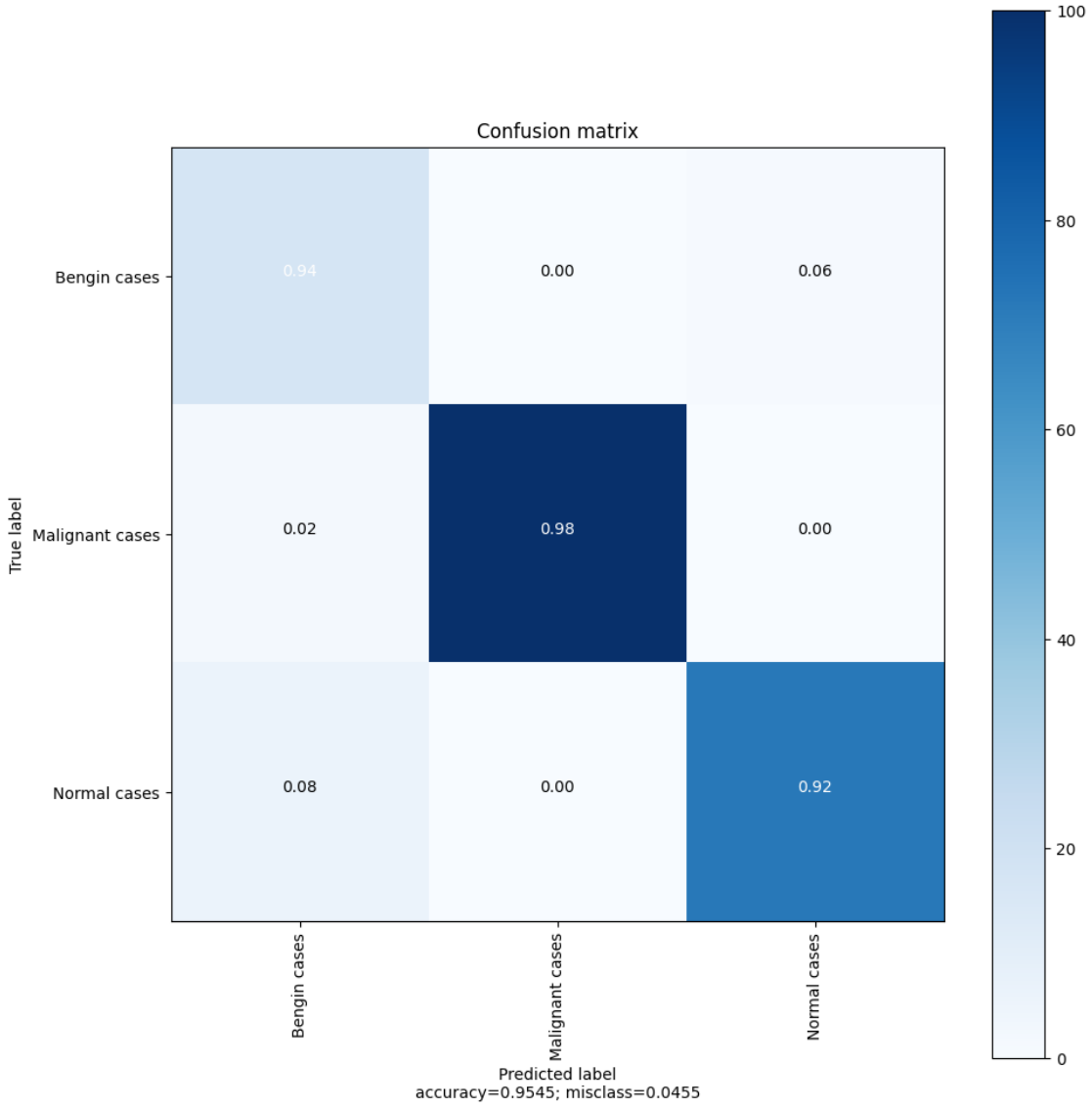

Confusion Matrix
Bengin cases
Malignant cases
Normal cases
True labels
17
0
1
2
1e+02
0
6
0
72
Bengin cases
Malignant cases
Normal cases
Predicted labels
100
80
60
40
20
0



[ ]:
```
import itertools
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):


    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    plt.grid(False)
```

```
    #if target_names is not None:
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=90)
    plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.2f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.
↪format(accuracy, misclass))

    plt.savefig('conmatrix1.png', dpi=300, bbox_inches='tight')
    plt.show()
```

[ ]:
```
#Plotting the confusion matrix
class_names=['Bengin cases', 'Malignant cases', 'Normal cases']
# Plotting non-normalized confusion matrix
plot_confusion_matrix(cm, class_names, title='Confusion matrix')
```



[ ]:
```
from keras.models import load_model
model = load_model('/content/drive/MyDrive/
  ↪1_FinalYearProjectLungCancerDetection/cnn_model1.h5')
```

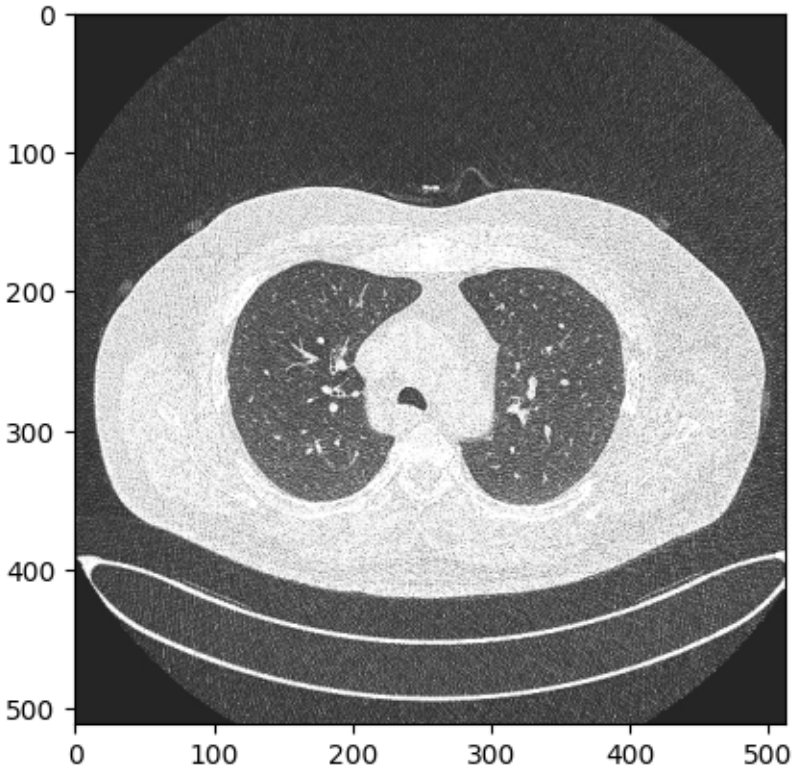
[ ]:
```
import matplotlib.pyplot as plt
!pip install Pillow
```

```
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages
(9.4.0)
```

[ ]:
```
import cv2
import tensorflow as tf
```

```
CATEGORIES = ['Bengin case', 'Malignant case', 'Normal case']
def prepare(filepath):
    IMG_SIZE=256
    img=cv2.imread(filepath)
    plt.imshow(img)
    img_array = img / 255.0
    new_array=cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
    return new_array.reshape(-1,IMG_SIZE,IMG_SIZE,1)


prediction = model.predict(prepare('/content/drive/MyDrive/
 ↪1_FinalYearProjectLungCancerDetection/Dataset/splitted_dataset/test/Bengin␣
 ↪cases/Bengin case (108).jpg'))
print(prediction)
predictions = np.argmax(prediction,-1)
print(predictions[0])
print(CATEGORIES[int(predictions[0])])
```

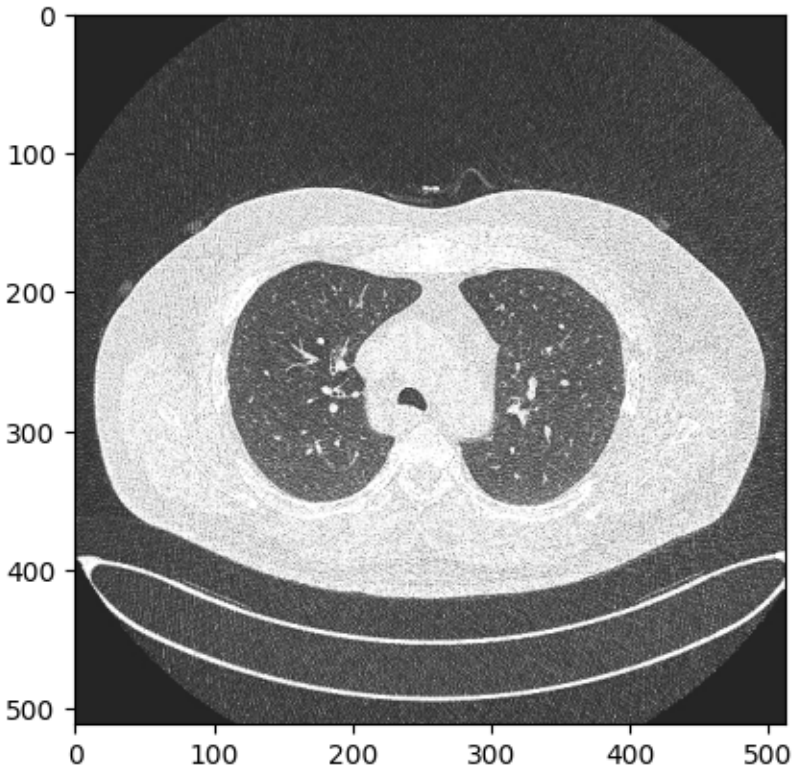
```
1/1 [==============================] - 0s 195ms/step
[[8.5370913e-03 8.6714649e-01 1.2431641e-01]
 [8.9020141e-05 3.8823786e-01 6.1167312e-01]
 [3.2717028e-06 6.6845649e-01 3.3154029e-01]]
1
Malignant case
```



[ ]:
```
import cv2
import tensorflow as tf

CATEGORIES = ['Bengin case', 'Malignant case', 'Normal case']
def prepare(filepath):
    IMG_SIZE=256
    img=cv2.imread(filepath)
    plt.imshow(img)
    img_array = img / 255.0
    new_array=cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
    return new_array.reshape(-1,IMG_SIZE,IMG_SIZE,1)


prediction = model.predict(prepare('/content/drive/MyDrive/
 ↪1_FinalYearProjectLungCancerDetection/Dataset/splitted_dataset/test/Bengin␣
 ↪cases/Bengin case (108).jpg'))
print(prediction)
predictions = np.argmax(prediction,-1)
print(predictions[0])
print(CATEGORIES[int(predictions[0])])
```

```
1/1 [==============================] - 0s 131ms/step
```

```
[[8.5370913e-03 8.6714649e-01 1.2431641e-01]
 [8.9020141e-05 3.8823786e-01 6.1167312e-01]
 [3.2717028e-06 6.6845649e-01 3.3154029e-01]]
1
Malignant case
```



[ ]:
```
# Loading the images and their class(0 - 2)
test_data = []
test_path = "/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/
 ↪Dataset/splitted_dataset/test"
classes = sorted(os.listdir(test_path))

for i in classes:
    path = os.path.join(test_path, i)
    print(path)
    class_num = classes.index(i)
    print(class_num)
    for file in os.listdir(path):
        filepath = os.path.join(path, file)
        test_data.append([filepath, class_num])
```

```
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
```

```
taset/test/Bengin cases
0
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/test/Malignant cases
1
/content/drive/MyDrive/1_FinalYearProjectLungCancerDetection/Dataset/splitted_da
taset/test/Normal cases
2
```

[ ]:
```
test_file = []
label = []
for X, y in test_data:
    test_file.append(X)
    label.append(y)
```

[ ]:
```
CATEGORIES = ['Bengin cases', 'Malignant cases', 'Normal cases']
preds = []
pred_file = []
def prepare(test_path):
    IMG_SIZE=256
    img_array=cv2.imread(test_path,cv2.IMREAD_GRAYSCALE)
    pred_file.append(test_path)

    img_array = img_array / 255.0
    new_array=cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
    return new_array.reshape(-1,IMG_SIZE,IMG_SIZE,1)

for i in test_file:
    prediction = model.predict(prepare(i))
    predictions = np.argmax(prediction,-1)
    preds.append(predictions[0])
```

[ ]:
```
df = pd.DataFrame({
    'File': test_file,
    'Actual' :label,
    'Path':pred_file,
    'Predicted' : preds
})
comparison_column = np.where(df["Actual"] == df["Predicted"], True, False)
df["Equal"] = comparison_column
df.to_csv("predictions1.csv")
df.head()
```

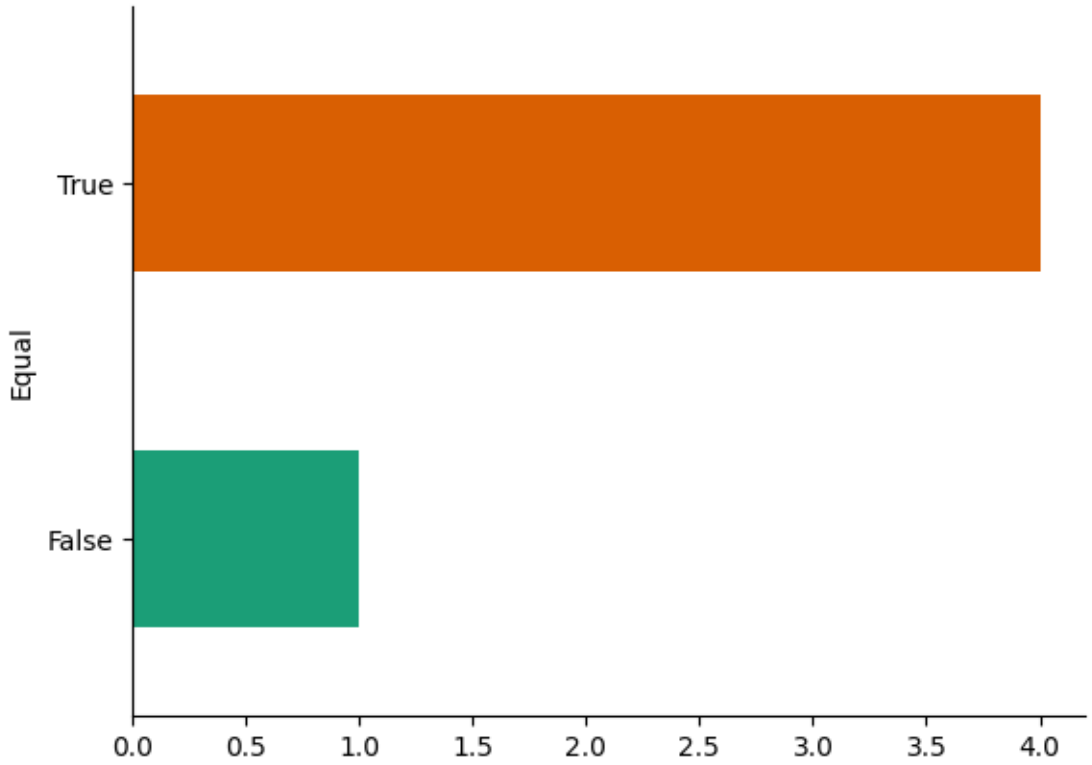
[ ]:

	File	Actual	\
0	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	
1	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	
2	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	

```
3  /content/drive/MyDrive/1_FinalYearProjectLungC…       0
4  /content/drive/MyDrive/1_FinalYearProjectLungC…       0
```

	Path	Predicted	Equal
0	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	True
1	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	True
2	/content/drive/MyDrive/1_FinalYearProjectLungC…	2	False
3	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	True
4	/content/drive/MyDrive/1_FinalYearProjectLungC…	0	True

[ ]:
```
from matplotlib import pyplot as plt
import seaborn as sns
_df_3.groupby('Equal').size().plot(kind='barh', color=sns.palettes.
  ↪mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```



[ ]:
```
df["Equal"].value_counts()
```

[ ]:
```
Equal
True     109
False      2
Name: count, dtype: int64
```