Open in Colab

```
from google.colab import drive
drive.mount('/gdrive')
```

```
    Mounted at /gdrive
```

```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
    '/device:GPU:0'
```

```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,Dropout,BatchNormalization
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from PIL import Image
import warnings
import glob

warnings.filterwarnings("ignore")
```

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
```

Modification of original code starts here

```
from google.colab import drive
import os

drive.mount('/content/drive')

os.chdir('/content/drive/My Drive/Colab Notebooks/BIO 6306/Brain Tumor Detection Convolutional Neural Network Project/Modified/DATASET/yes/')

files = os.listdir()
print("Files in the directory:")
print(files)
```

```
    Mounted at /content/drive
    Files in the directory:
    ['y1267.jpg', 'y1.jpg', 'y1108.jpg', 'y14.jpg', 'y176.jpg', 'y1199.jpg', 'y1040.jpg', 'y1092.jpg', 'y101.jpg', 'y1341.jpg', 'y1026.jpg', 'y1225.jpg', 'y1305.jpg', 'y1087.jpg', 'y1132.jpg', 'y1020.jpg', 'y1412.jpg', 'y1486.jpg', 'y15.jpg', 'y1392.jpg', 'y11.jpg', 'y1269.jpg', 'y1164.jpg', 'y1210.jpg', 'y1140.jpg', 'y1477.jpg', 'y1057.jpg', 'y136.jpg', 'y1363.jpg', 'y1277.jpg', ':
```

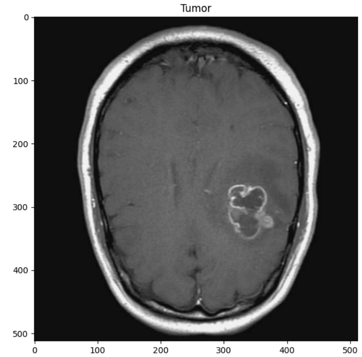
```
from google.colab import drive
import os

drive.mount('/content/drive')


os.chdir('/content/drive/My Drive/Colab Notebooks/BIO 6306/Brain Tumor Detection Convolutional Neural Network Project/Modified/DATASET/no/')

files = os.listdir()
print("Files in the directory:")
print(files)
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
    Files in the directory:
    ['no270.jpg', 'no1470.jpg', 'no1455.jpg', 'no238.jpg', 'no157.jpg', 'no1176.jpg', 'no1424.jpg', 'no1330.jpg', 'no248.jpg', 'no177.jpg', 'no185.jpg', 'no260.jpg', 'no1162.jpg', 'no1489.jpg', 'no1452.jpg', 'no1495.jpg', 'no1372.jpg', 'no1269.jpg', 'no1475.jpg', 'no1146.jpg', 'no1426.jpg', 'no1312.jpg', 'no1086.jpg', 'no1287.jpg', 'no1214.jpg', 'no1342.jpg', 'no1471.jpg', 'no204.j
```

```
import cv2
import glob
import numpy as np

tumor = []
no_tumor = []

for file in glob.iglob('/content/drive/My Drive/Colab Notebooks/BIO 6306/Brain Tumor Detection Convolutional Neural Network Project/Modified/DATASET/yes/*.jpg'):
    img = cv2.imread(file)
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (512, 512))
        tumor.append((img, 1))

for file in glob.iglob('/content/drive/My Drive/Colab Notebooks/BIO 6306/Brain Tumor Detection Convolutional Neural Network Project/Modified/DATASET/no/*.jpg'):
    img = cv2.imread(file)
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (512, 512))
        no_tumor.append((img, 0))

data = tumor + no_tumor
x = np.array([i[0] for i in data])
y = np.array([i[1] for i in data])
```

```
def plot_img(i):
    plt.figure(figsize=(7,7))
    plt.imshow(x[i])
    if y[i]==1:
        plt.title('Tumor')
    if y[i]==0:
        plt.title('No_Tumor')
```

```
import matplotlib.pyplot as plt
plot_img(300)
```



Notice 512 x 512 dimensions

```
from sklearn.utils import shuffle
x,y=shuffle(x,y,random_state=101)
```

```
from sklearn.model_selection import train_test_split
x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)
x_valid, x_test, y_valid, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)
```

```
print(y_train)
```

```
[1 1 1 ... 1 0 1]
```

```
print(x_train.shape)
print(x_temp.shape)
print(x_test.shape)
print(y_train.shape)
print(y_temp.shape)
print(y_test.shape)
```

```
(2100, 512, 512, 3)
(900, 512, 512, 3)
(450, 512, 512, 3)
(2100,)
(900,)
(450,)
```

```
print(x_train)
```

```
[[[[ 24  24  24]
   [ 33  33  33]
   [ 47  47  47]
   ...
   [ 56  56  56]
   [ 55  55  55]
   [ 55  55  55]]

  [[ 39  39  39]
   [ 48  48  48]
   [ 64  64  64]
   ...
   [ 52  52  52]
   [ 53  53  53]
   [ 53  53  53]]

  [[ 50  50  50]
   [ 54  54  54]
   [ 61  61  61]
   ...
   [ 50  50  50]
   [ 51  51  51]
   [ 52  52  52]]

  ...

  [[251 251 251]
   [252 252 252]
   [254 254 254]
   ...
   [ 45  45  45]
   [ 45  45  45]
   [ 46  46  46]]

  [[251 251 251]
   [252 252 252]
   [254 254 254]
   ...
   [ 45  45  45]
   [ 44  44  44]
   [ 43  43  43]]

  [[251 251 251]
   [252 252 252]
   [254 254 254]
   ...
   [ 45  45  45]
   [ 43  43  43]
   [ 43  43  43]]]


 [[[  0   0   0]
   [  0   0   0]
   [  0   0   0]
   ...
   [  0   0   0]
   [  0   0   0]
   [  0   0   0]]
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
tf.keras.backend.clear_session()
```

```
ini_input=keras.Input(shape=(512,512,3),name="image")
x1=layers.Conv2D(64,(22,22),strides=2)(ini_input)
x1=layers.MaxPooling2D((4,4))(x1)
x1=layers.BatchNormalization()(x1)
x2=layers.Conv2D(128,(11,11),strides=2,padding="same")(x1)
x2=layers.MaxPooling2D((2,2))(x2)
x2=layers.BatchNormalization()(x2)
x3=layers.Conv2D(256,(7,7),strides=2,padding="same")(x2)
x3=layers.MaxPooling2D((2,2))(x3)
x3=layers.BatchNormalization()(x3)
x4 = layers.Conv2D(512, (3, 3), strides=2, padding="same")(x3)
x4 = layers.MaxPooling2D((2, 2))(x4)
x4 = layers.BatchNormalization()(x4)
x5 = layers.GlobalAveragePooling2D()(x4)
x5 = layers.Activation("relu")(x5)
x6 = layers.Dense(1024, "relu")(x5)
x6 = layers.BatchNormalization()(x6)
x7 = layers.Dense(512, "relu")(x6)
x7 = layers.BatchNormalization()(x7)
x8 = layers.Dense(256, "relu")(x7)
x8 = layers.BatchNormalization()(x8)
x8 = layers.Dropout(0.2)(x8)
output = layers.Dense(1, activation="sigmoid")(x8)
model = keras.Model(inputs=ini_input, outputs=output)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image (InputLayer)          [(None, 512, 512, 3)]     0

 conv2d (Conv2D)             (None, 246, 246, 64)      92992

 max_pooling2d (MaxPooling2  (None, 61, 61, 64)        0
 D)

 batch_normalization (Batch  (None, 61, 61, 64)        256
 Normalization)

 conv2d_1 (Conv2D)           (None, 31, 31, 128)       991360

 max_pooling2d_1 (MaxPoolin  (None, 15, 15, 128)       0
 g2D)

 batch_normalization_1 (Bat  (None, 15, 15, 128)       512
 chNormalization)

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         1605888

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 batch_normalization_2 (Bat  (None, 4, 4, 256)         1024
 chNormalization)

 conv2d_3 (Conv2D)           (None, 2, 2, 512)         1180160

 max_pooling2d_3 (MaxPoolin  (None, 1, 1, 512)         0
 g2D)

 batch_normalization_3 (Bat  (None, 1, 1, 512)         2048
 chNormalization)

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 activation (Activation)     (None, 512)               0

 dense (Dense)               (None, 1024)              525312

 batch_normalization_4 (Bat  (None, 1024)              4096
 chNormalization)

 dense_1 (Dense)             (None, 512)               524800

 batch_normalization_5 (Bat  (None, 512)               2048
 chNormalization)

 dense_2 (Dense)             (None, 256)               131328

 batch_normalization_6 (Bat  (None, 256)               1024
 chNormalization)
```

```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
r=model.fit(x_train,
        y_train,
        epochs=200,
        batch_size=10,
        verbose=1,
        validation_data=(x_valid,y_valid),
        shuffle=False
        )
```

```
Epoch 1/200
210/210 [==============================] - 34s 93ms/step - loss: 0.6353 - accuracy: 0.7010 - val_loss: 0.5767 - val_accuracy: 0.7067
Epoch 2/200
210/210 [==============================] - 18s 85ms/step - loss: 0.4839 - accuracy: 0.7862 - val_loss: 0.3533 - val_accuracy: 0.8489
Epoch 3/200
210/210 [==============================] - 18s 87ms/step - loss: 0.4081 - accuracy: 0.8176 - val_loss: 0.4791 - val_accuracy: 0.7889
Epoch 4/200
210/210 [==============================] - 20s 94ms/step - loss: 0.3058 - accuracy: 0.8767 - val_loss: 0.3131 - val_accuracy: 0.8667
Epoch 5/200
210/210 [==============================] - 19s 92ms/step - loss: 0.2469 - accuracy: 0.9090 - val_loss: 0.3124 - val_accuracy: 0.8644
Epoch 6/200
210/210 [==============================] - 19s 92ms/step - loss: 0.1898 - accuracy: 0.9286 - val_loss: 0.3167 - val_accuracy: 0.8800
Epoch 7/200
210/210 [==============================] - 19s 91ms/step - loss: 0.1732 - accuracy: 0.9381 - val_loss: 0.4209 - val_accuracy: 0.8689
Epoch 8/200
210/210 [==============================] - 18s 88ms/step - loss: 0.1428 - accuracy: 0.9510 - val_loss: 0.3245 - val_accuracy: 0.8778
Epoch 9/200
210/210 [==============================] - 19s 92ms/step - loss: 0.1492 - accuracy: 0.9448 - val_loss: 0.6624 - val_accuracy: 0.8267
Epoch 10/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0998 - accuracy: 0.9624 - val_loss: 0.5939 - val_accuracy: 0.7756
Epoch 11/200
210/210 [==============================] - 18s 87ms/step - loss: 0.0567 - accuracy: 0.9805 - val_loss: 0.1001 - val_accuracy: 0.9667
Epoch 12/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0857 - accuracy: 0.9714 - val_loss: 0.0981 - val_accuracy: 0.9622
Epoch 13/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0546 - accuracy: 0.9819 - val_loss: 0.0879 - val_accuracy: 0.9711
Epoch 14/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0463 - accuracy: 0.9848 - val_loss: 0.0537 - val_accuracy: 0.9822
Epoch 15/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0269 - accuracy: 0.9905 - val_loss: 0.2627 - val_accuracy: 0.9156
Epoch 16/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0221 - accuracy: 0.9952 - val_loss: 0.0906 - val_accuracy: 0.9733
Epoch 17/200
210/210 [==============================] - 18s 87ms/step - loss: 0.0543 - accuracy: 0.9819 - val_loss: 1.0837 - val_accuracy: 0.9111
Epoch 18/200
210/210 [==============================] - 18s 87ms/step - loss: 0.1121 - accuracy: 0.9605 - val_loss: 0.2204 - val_accuracy: 0.9133
Epoch 19/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0763 - accuracy: 0.9733 - val_loss: 1.0560 - val_accuracy: 0.7489
```

```
Epoch 20/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0245 - accuracy: 0.9929 - val_loss: 0.0966 - val_accuracy: 0.9778
Epoch 21/200
210/210 [==============================] - 18s 87ms/step - loss: 0.0141 - accuracy: 0.9962 - val_loss: 0.0555 - val_accuracy: 0.9800
Epoch 22/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0069 - accuracy: 0.9981 - val_loss: 0.0585 - val_accuracy: 0.9778
Epoch 23/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0081 - accuracy: 0.9986 - val_loss: 0.0635 - val_accuracy: 0.9822
Epoch 24/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0062 - accuracy: 0.9990 - val_loss: 0.1074 - val_accuracy: 0.9733
Epoch 25/200
210/210 [==============================] - 19s 92ms/step - loss: 0.0690 - accuracy: 0.9771 - val_loss: 0.9417 - val_accuracy: 0.7889
Epoch 26/200
210/210 [==============================] - 18s 87ms/step - loss: 0.0583 - accuracy: 0.9786 - val_loss: 2.2755 - val_accuracy: 0.7000
Epoch 27/200
210/210 [==============================] - 18s 88ms/step - loss: 0.0259 - accuracy: 0.9914 - val_loss: 0.0448 - val_accuracy: 0.9800
Epoch 28/200
210/210 [==============================] - 18s 87ms/step - loss: 0.0291 - accuracy: 0.9886 - val_loss: 0.6657 - val_accuracy: 0.8511
Epoch 29/200
```

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(35, 6))
t = f.suptitle('Basic CNN Performance', fontsize=12)
f.subplots_adjust(top=1.85, wspace=0.8)

epoch_list = list(range(0,200))
ax1.plot(epoch_list, r.history['accuracy'], label='Train Accuracy')
ax1.plot(epoch_list, r.history['val_accuracy'], label='Validation Accuracy')
ax1.set_xticks(np.arange(0, 200, 10))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, r.history['loss'], label='Train Loss')
ax2.plot(epoch_list, r.history['val_loss'], label='Validation Loss')
ax2.set_xticks(np.arange(0, 200, 10))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```


```
y_pred=model.predict(x_test)
y_pred
```

```
15/15 [==============================] - 2s 105ms/step
array([[9.99990702e-01],
       [3.31793126e-05],
       [2.49500516e-16],
       [1.60459182e-16],
       [9.99992013e-01],
       [1.62978849e-05],
       [1.67018352e-05],
       [1.00000000e+00],
       [1.87204022e-16],
       [7.36538505e-16],
       [3.78359277e-06],
       [3.27803245e-05],
       [1.24360922e-05],
       [7.18837919e-06],
       [1.00000000e+00],
       [2.01878669e-09],
       [9.99998331e-01],
       [5.66681156e-05],
       [1.00000000e+00],
       [9.99999285e-01],
       [9.99966860e-01],
       [1.46073954e-18],
       [1.02394347e-06],
       [1.00000000e+00],
       [2.57604679e-05],
       [1.79452963e-05],
       [5.90625259e-13],
       [4.24298250e-08],
       [9.99999166e-01],
       [1.00000000e+00],
       [1.25659164e-03],
       [9.99978781e-01],
       [3.73934045e-14],
       [1.20930045e-12],
       [3.96757241e-05],
       [5.47519985e-09],
       [9.99992967e-01],
       [1.27993180e-08],
       [8.43823582e-05],
       [9.96725023e-01],
       [2.11949700e-05],
       [9.99982238e-01],
       [2.00629120e-06],
       [1.00000000e+00],
       [9.99876499e-01],
       [5.68261882e-10],
       [7.88060618e-07],
       [1.54464574e-09],
       [9.99996543e-01],
       [9.99758899e-01],
       [9.99984622e-01],
       [9.99999881e-01],
       [9.76428737e-07],
```

```
           [1.00000000e+00],
           [3.90703374e-21],
           [1.12175831e-05],
```

```
y_pred=model.predict(x_test)
y_pred=np.round(y_pred,0)
y_pred[:5]
```

```
15/15 [==============================] - 1s 102ms/step
array([[1.],
       [0.],
       [0.],
       [0.],
       [1.]], dtype=float32)
```

```
y_pred.shape
```

```
(450, 1)
```

```
y_test.shape
```

```
(450,)
```

```
from sklearn.metrics import confusion_matrix
threshold = 0.5
binary_predictions = (y_pred[:, 0] >= threshold).astype(int)
print(binary_predictions)
```

```
[1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1
 0 0 1 0 1 0 1 1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 0 0 1 1 1
 1 0 0 1 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1
 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 0
 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1
 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 0
 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 0 0 1 0 1 0 0 1
 0 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0
 0 0 0 0 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0
 0 1 0 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 0 1 0 1 0 1 1 0
 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0
 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 1 1 1 1
 1 0 1 1 0 1]
```

```
y_pred=binary_predictions
y_pred
```

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

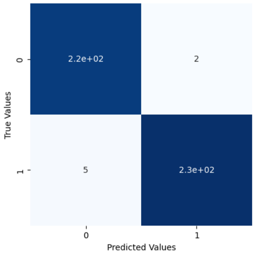
```
y_test
```

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

```
from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix
[[216   2]
 [  5 227]]
```

```
import seaborn as sns
cm=confusion_matrix(y_test, y_pred)
sns.heatmap(cm, square=True, annot=True, cbar=False, cmap=plt.cm.Blues)
plt.xlabel('Predicted Values')
plt.ylabel('True Values');
```




```
from sklearn.metrics import classification_report
target_names = ['No Tumor', 'Tumor']
print('Classification Report:')
print(classification_report(y_test, binary_predictions))
```

```
    Classification Report:
                  precision    recall  f1-score   support

               0       0.98      0.99      0.98       218
               1       0.99      0.98      0.98       232

        accuracy                           0.98       450
       macro avg       0.98      0.98      0.98       450
    weighted avg       0.98      0.98      0.98       450
```

```
import numpy as np
from sklearn import metrics
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred, pos_label=1)
metrics.auc(fpr, tpr)
```

```
    0.9846369819677318
```

```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(fpr, tpr)
ax.plot(np.linspace(0, 1, 100),
         np.linspace(0, 1, 100),
         label='baseline',
         linestyle='--')
plt.title('Receiver Operating Characteristic Curve', fontsize=14)
plt.ylabel('Total Positive Rate', fontsize=12)
plt.xlabel('False Positive Rate', fontsize=12)
plt.legend(fontsize=12);
```