In [1]:
```
!pip install imutils
```

```
Requirement already satisfied: imutils in c:\users\lenovo\anaconda3\lib\site-packages (0.5.4)
```

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
import os
from os import listdir
```

In [4]:
```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

In [5]:
```
import matplotlib.pyplot as plt
```

In [6]:
```
%matplotlib inline
```

In [7]:
```
import numpy as np
import sys
```

In [8]:
```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\lenovo\anaconda3\lib\site-packages (4.5.4.58)
Requirement already satisfied: numpy>=1.19.3 in c:\users\lenovo\anaconda3\lib\site-packages (from opencv-python) (1.20.
3)
Note: you may need to restart the kernel to use updated packages.
```

In [9]:
```
import cv2
import imutils
import itertools
```

In [10]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [14]:
```
from tensorflow.keras.models import Model,load_model
from tensorflow.keras.layers import Conv2D,Input,ZeroPadding2D,BatchNormalization,Flatten,Activation,Dense,MaxPooling2D
```

In [15]:
```
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

In [16]:
```
images="C:\\Users\\Lenovo\\Desktop\\Project\\Data"
```

In [18]:
```
os.makedirs('C:\\Users\\Lenovo\\Desktop\\Project\\Output')
```

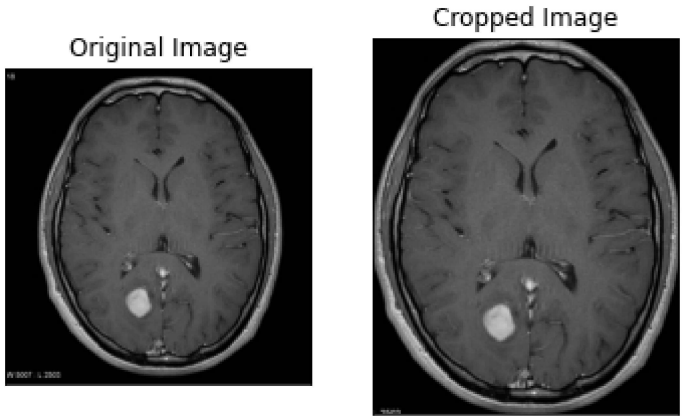
In [19]:
```
os.makedirs('C:\\Users\\Lenovo\\Desktop\\Project\\Output\\Yes')
os.makedirs('C:\\Users\\Lenovo\\Desktop\\Project\\Output\\No')
```

In [20]:
```
def augment_data(file_dir,n_generated_samples,save_to_dir):
    data_gen=ImageDataGenerator(rotation_range=10,width_shift_range=0.1,height_shift_range=0.1,shear_range=0.1,
                                brightness_range=(0.3,1.0),horizontal_flip=True,vertical_flip=True,fill_mode='nearest')
    for filename in listdir(file_dir):
        image=cv2.imread(file_dir+'/'+filename)
        image=image.reshape((1,)+image.shape)
        save_prefix='aug_'+filename[:-4]
        i=0
        for batch in data_gen.flow(x=image,batch_size=1,save_to_dir=save_to_dir,save_prefix=save_prefix,save_format='jpg
                i+=1
                if i>n_generated_samples:
                    break
```

In [22]:
```
augmented_data_path='C:\\Users\\Lenovo\\Desktop\\Project\\Output'
augment_data(file_dir=images+'yes',n_generated_samples=6,save_to_dir=augmented_data_path+'yes')
augment_data(file_dir=images+'no',n_generated_samples=9,save_to_dir=augmented_data_path+'no')
```

In [26]:
```
def crop_brain_contour(image,plot=False):
    gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    gray=cv2.GaussianBlur(gray,(5,5),0)
    thresh=cv2.threshold(gray,45,255,cv2.THRESH_BINARY)[1]
    thresh=cv2.erode(thresh,None,iterations=2)
    thresh=cv2.dilate(thresh,None,iterations=2)
    cnts=cv2.findContours(thresh.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts=imutils.grab_contours(cnts)
    c=max(cnts,key=cv2.contourArea)
    extLeft=tuple(c[c[:, :, 0].argmin()][0])
    extRight=tuple(c[c[:, :, 0].argmax()][0])
    extTop=tuple(c[c[:, :, 1].argmin()][0])
    extBot=tuple(c[c[:, :, 1].argmax()][0])
    new_image=image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]
    if plot:
        plt.figure()
        plt.subplot(1, 2, 1)
        plt.imshow(image)
        plt.tick_params(axis='both',which='both',top=False,bottom=False,left=False,right=False,labelbottom=False,labelto
                        labelleft=False,labelright=False)
        plt.title('Original Image')
        plt.subplot(1, 2, 2)
        plt.imshow(new_image)
        plt.tick_params(axis='both',which='both',top=False,bottom=False,left=False,right=False,labelbottom=False,labelto
                        labelleft=False,labelright=False)
        plt.title('Cropped Image')
        plt.show()
    return new_image
```

In [28]:
```
ex_img=cv2.imread('C:\\Users\\Lenovo\\Desktop\\Project\\Data\\yes\\Y6.jpg')
ex_crop_img=crop_brain_contour(ex_img,True)
```



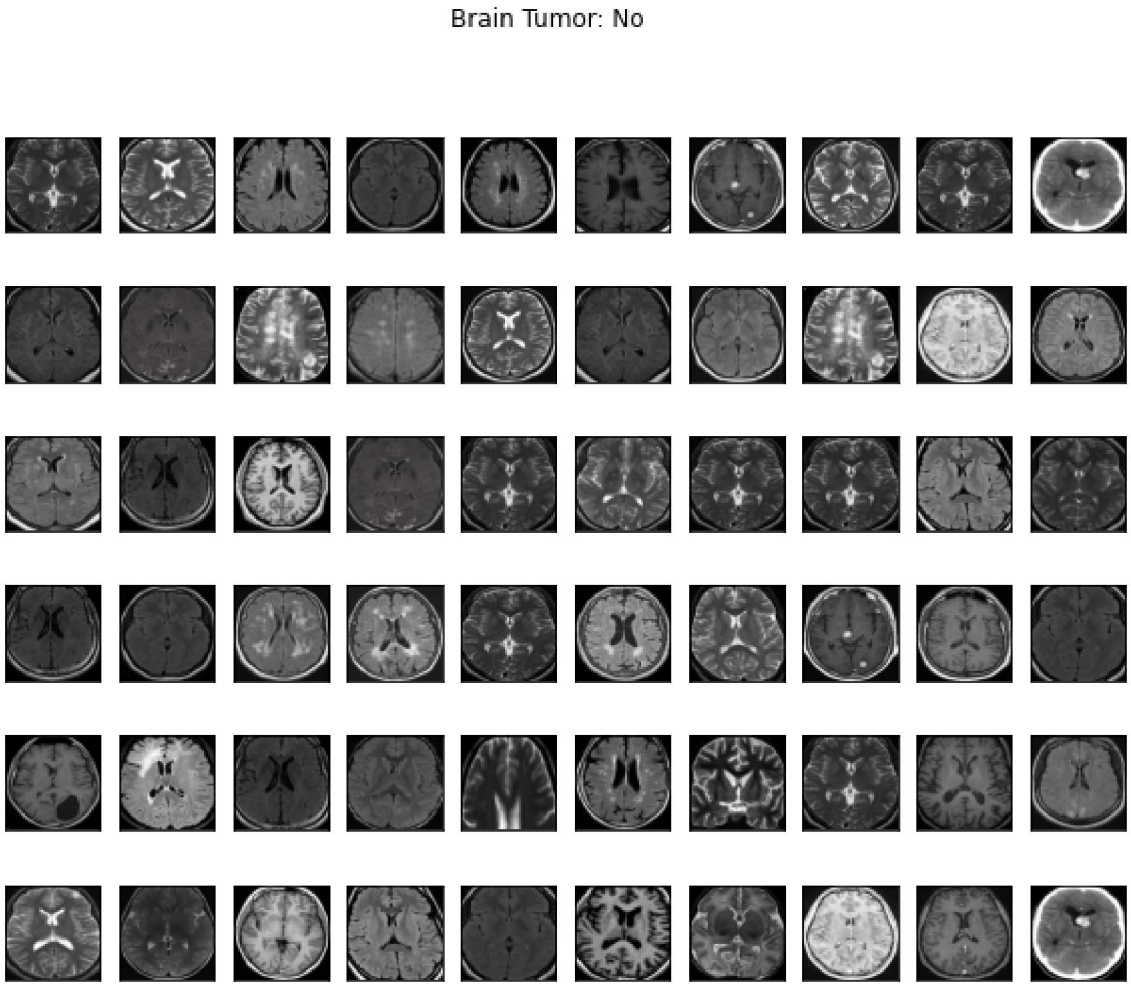
In [29]:
```
def load_data(dir_list,image_size):
    X=[]
    Y=[]
    image_width,image_height=image_size
    for directory in dir_list:
        for filename in listdir(directory):
            image=cv2.imread(directory+'/'+filename)
            image=crop_brain_contour(image,plot=False)
            image=cv2.resize(image, dsize=(image_width,image_height),interpolation=cv2.INTER_CUBIC)
            image=image/255.
            X.append(image)
            if directory[-3:]=='yes':
                Y.append([1])
            else:
                Y.append([0])
    X=np.array(X)
    Y=np.array(Y)
    X,Y=shuffle(X,Y)
    print(f'Number of examples is: {len(X)}')
    print(f'X shape is: {X.shape}')
    print(f'Y shape is: {Y.shape}')
    return X,Y
```

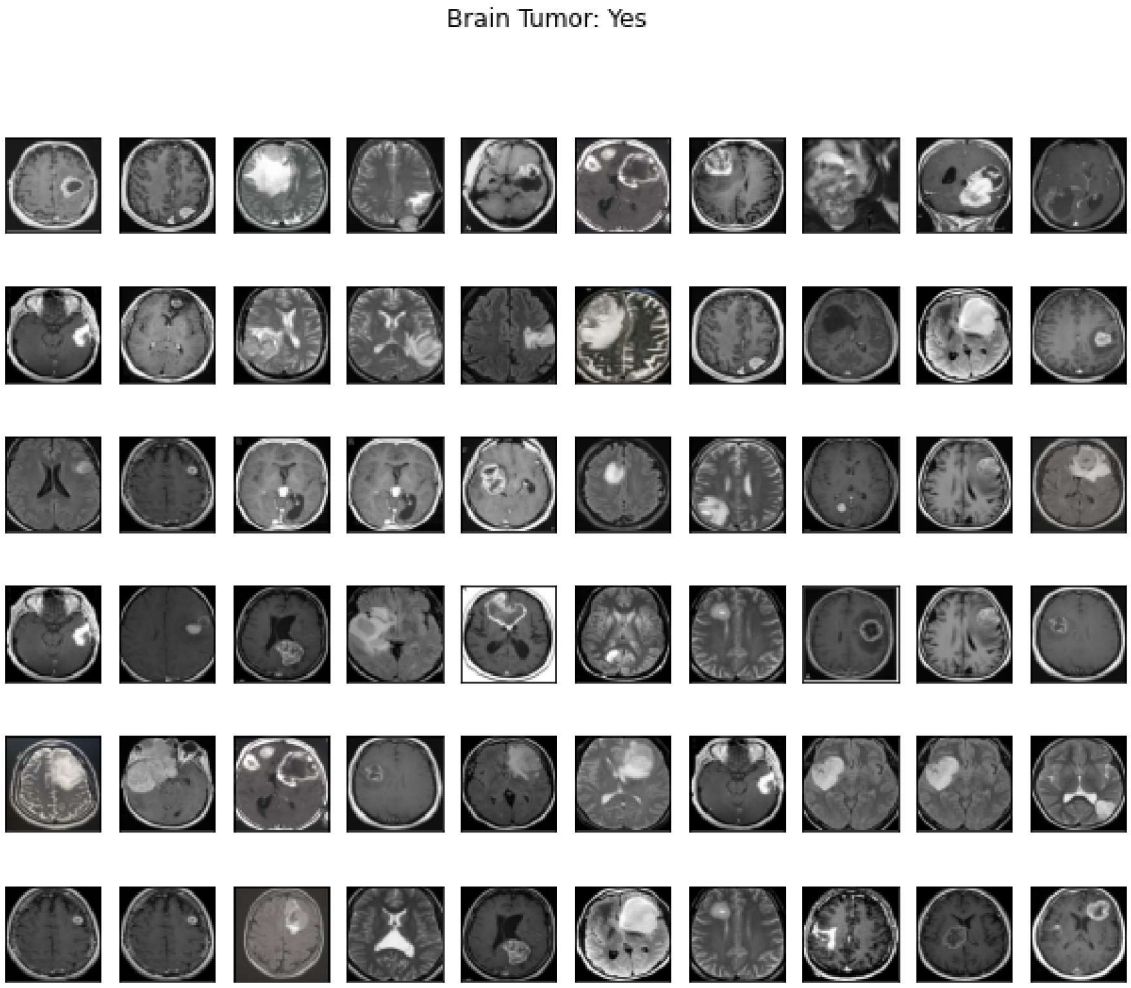
In [30]:
```
augmented_yes=augmented_data_path+'yes'
augmented_no=augmented_data_path+'no'
IMG_WIDTH,IMG_HEIGHT=(240,240)
X,Y=load_data([augmented_yes,augmented_no],(IMG_WIDTH,IMG_HEIGHT))
```

```
Number of examples is: 1518
X shape is: (1518, 240, 240, 3)
Y shape is: (1518, 1)
```

In [42]:
```
def plot_sample_images(X,Y,n=60):
    for label in [0,1]:
        images=X[np.argwhere(Y==label)]
        n_images=images[:n]
        columns_n=10
        rows_n=int(n/columns_n)
        plt.figure(figsize=(10,8))
        i=1
        for image in n_images:
            plt.subplot(rows_n,columns_n,i)
            plt.imshow(image[0])
            plt.tick_params(axis='both',which='both',top=False, bottom=False,left=False,right=False,labelbottom=False,
                            labeltop=False,labelleft=False,labelright=False)
            i += 1
        label_to_str=lambda label: "Yes" if label==1 else "No"
        plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
        plt.show()
```

In [43]:
```
#60 sample input images
plot_sample_images(X,Y)
```





In [45]:
```
def split_data(X,Y,test_size=0.2):
    X_train,X_test_val,Y_train,Y_test_val=train_test_split(X,Y,test_size=test_size)
    X_test,X_val,Y_test,Y_val=train_test_split(X_test_val,Y_test_val,test_size=0.5)
    return X_train,Y_train,X_val,Y_val,X_test,Y_test
```

In [46]:
```
X_train,Y_train,X_val,Y_val,X_test,Y_test=split_data(X,Y,test_size=0.3)
```

In [48]:
```
print("number of training examples = "+str(X_train.shape[0]))
print("number of validation examples = "+str(X_val.shape[0]))
print("number of test examples = "+str(X_test.shape[0]))
```

```
number of training examples = 1062
number of validation examples = 228
number of test examples = 228
```

In [52]:
```
def build_model(input_shape):
    X_input=Input(input_shape)
    X=ZeroPadding2D((2, 2))(X_input)
    X=Conv2D(32,(7,7),strides=(1,1))(X)
    X=BatchNormalization(axis=3,name='bn0')(X)
    X=Activation('relu')(X)
    X=MaxPooling2D((4,4))(X)
    X=MaxPooling2D((4,4))(X)
    X=Flatten()(X)
    X=Dense(1,activation='sigmoid')(X)
    model=Model(inputs=X_input,outputs=X)
    return model
```

In [55]:
```
IMG_SHAPE=(IMG_WIDTH,IMG_HEIGHT,3)
model=build_model(IMG_SHAPE)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 240, 240, 3)]     0

 zero_padding2d_2 (ZeroPaddi  (None, 244, 244, 3)      0
 ng2D)

 conv2d_2 (Conv2D)           (None, 238, 238, 32)      4736

 bn0 (BatchNormalization)    (None, 238, 238, 32)      128

 activation_2 (Activation)   (None, 238, 238, 32)      0

 max_pooling2d_4 (MaxPooling  (None, 59, 59, 32)       0
 2D)

 max_pooling2d_5 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 flatten_2 (Flatten)         (None, 6272)              0

 dense_2 (Dense)             (None, 1)                 6273

=================================================================
Total params: 11,137
Trainable params: 11,073
Non-trainable params: 64
_________________________________________________________________
```

In [56]:
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(x=X_train,y=Y_train,batch_size=32,epochs=22,validation_data=(X_val,Y_val))
```

```
Epoch 1/22
34/34 [==============================] - 117s 3s/step - loss: 0.7653 - accuracy: 0.6846 - val_loss: 0.6562 - val_accura
cy: 0.6096
Epoch 2/22
34/34 [==============================] - 115s 3s/step - loss: 0.3466 - accuracy: 0.8390 - val_loss: 0.5791 - val_accura
cy: 0.8289
Epoch 3/22
34/34 [==============================] - 122s 4s/step - loss: 0.2363 - accuracy: 0.9303 - val_loss: 0.4922 - val_accura
cy: 0.8904
Epoch 4/22
34/34 [==============================] - 118s 3s/step - loss: 0.1548 - accuracy: 0.9661 - val_loss: 0.4583 - val_accura
cy: 0.9649
Epoch 5/22
34/34 [==============================] - 120s 4s/step - loss: 0.1177 - accuracy: 0.9831 - val_loss: 0.3929 - val_accura
cy: 0.9649
Epoch 6/22
34/34 [==============================] - 118s 3s/step - loss: 0.1225 - accuracy: 0.9689 - val_loss: 0.3547 - val_accura
cy: 0.9649
Epoch 7/22
34/34 [==============================] - 126s 4s/step - loss: 0.0674 - accuracy: 0.9906 - val_loss: 0.2810 - val_accura
cy: 0.9781
Epoch 8/22
34/34 [==============================] - 117s 3s/step - loss: 0.0507 - accuracy: 0.9962 - val_loss: 0.2357 - val_accura
cy: 0.9956
Epoch 9/22
34/34 [==============================] - 116s 3s/step - loss: 0.0435 - accuracy: 0.9981 - val_loss: 0.2246 - val_accura
cy: 0.9912
Epoch 10/22
34/34 [==============================] - 124s 4s/step - loss: 0.0470 - accuracy: 0.9962 - val_loss: 0.1426 - val_accura
cy: 0.9956
Epoch 11/22
34/34 [==============================] - 130s 4s/step - loss: 0.0255 - accuracy: 0.9991 - val_loss: 0.1290 - val_accura
cy: 0.9956
Epoch 12/22
34/34 [==============================] - 137s 4s/step - loss: 0.0219 - accuracy: 1.0000 - val_loss: 0.1010 - val_accura
cy: 0.9956
Epoch 13/22
34/34 [==============================] - 191s 6s/step - loss: 0.0169 - accuracy: 1.0000 - val_loss: 0.0683 - val_accura
```

```
cy: 1.0000
Epoch 14/22
34/34 [==============================] - 188s 6s/step - loss: 0.0146 - accuracy: 1.0000 - val_loss: 0.0674 - val_accura
cy: 1.0000
Epoch 15/22
34/34 [==============================] - 186s 5s/step - loss: 0.0128 - accuracy: 1.0000 - val_loss: 0.0373 - val_accura
cy: 1.0000
Epoch 16/22
34/34 [==============================] - 187s 6s/step - loss: 0.0109 - accuracy: 1.0000 - val_loss: 0.0289 - val_accura
cy: 1.0000
Epoch 17/22
34/34 [==============================] - 189s 6s/step - loss: 0.0097 - accuracy: 1.0000 - val_loss: 0.0251 - val_accura
cy: 1.0000
Epoch 18/22
34/34 [==============================] - 189s 6s/step - loss: 0.0088 - accuracy: 1.0000 - val_loss: 0.0184 - val_accura
cy: 1.0000
Epoch 19/22
34/34 [==============================] - 193s 6s/step - loss: 0.0075 - accuracy: 1.0000 - val_loss: 0.0157 - val_accura
cy: 1.0000
Epoch 20/22
34/34 [==============================] - 193s 6s/step - loss: 0.0069 - accuracy: 1.0000 - val_loss: 0.0125 - val_accura
cy: 1.0000
Epoch 21/22
34/34 [==============================] - 193s 6s/step - loss: 0.0061 - accuracy: 1.0000 - val_loss: 0.0098 - val_accura
cy: 1.0000
Epoch 22/22
34/34 [==============================] - 191s 6s/step - loss: 0.0056 - accuracy: 1.0000 - val_loss: 0.0099 - val_accura
cy: 1.0000
```

Out[56]: <keras.callbacks.History at 0x2d175bafca0>

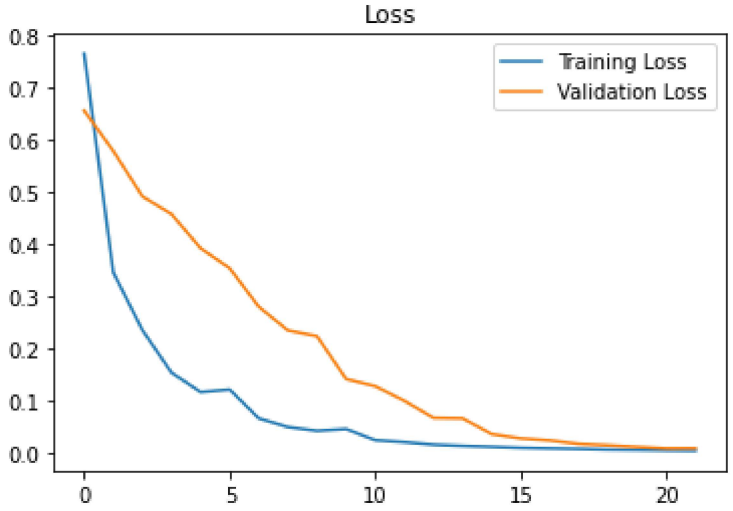
In [57]:
```
history = model.history.history
```

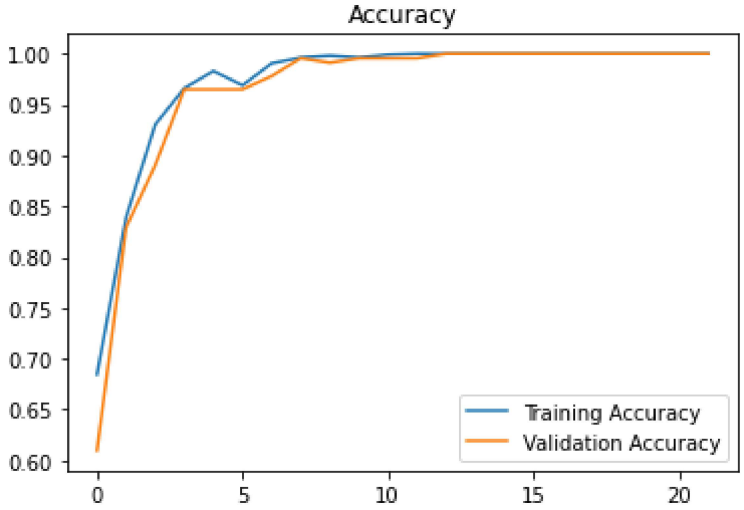
In [58]:
```
def plot_metrics(history):
    train_loss=history['loss']
    val_loss=history['val_loss']
    train_acc=history['accuracy']
    val_acc=history['val_accuracy']

    # Loss Graph
    plt.figure()
    plt.plot(train_loss,label='Training Loss')
    plt.plot(val_loss,label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()

    # Accuracy Graph
    plt.figure()
    plt.plot(train_acc,label='Training Accuracy')
    plt.plot(val_acc,label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

In [59]:
```
plot_metrics(history)
```

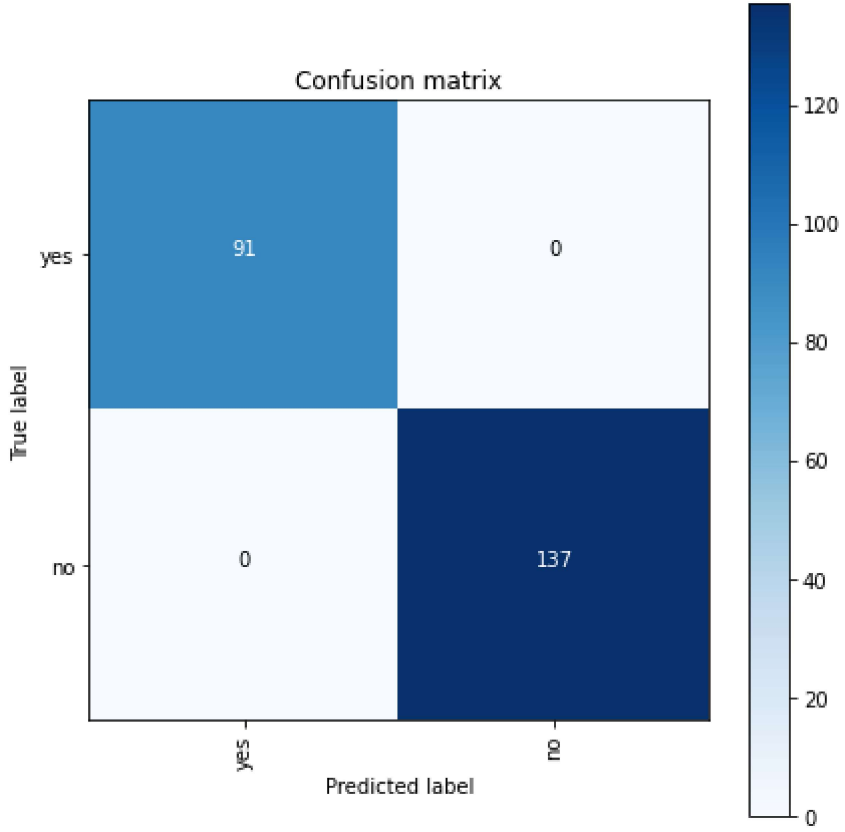




In [60]:
```
def plot_confusion_matrix(cm,classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
    plt.figure(figsize=(6,6))
    plt.imshow(cm,interpolation='nearest',cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks=np.arange(len(classes))
    plt.xticks(tick_marks,classes,rotation=90)
    plt.yticks(tick_marks,classes)
    if normalize:
        cm=cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]
    thresh=cm.max()/2.
    cm=np.round(cm,2)
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
        plt.text(j,i,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')
    plt.show()
```

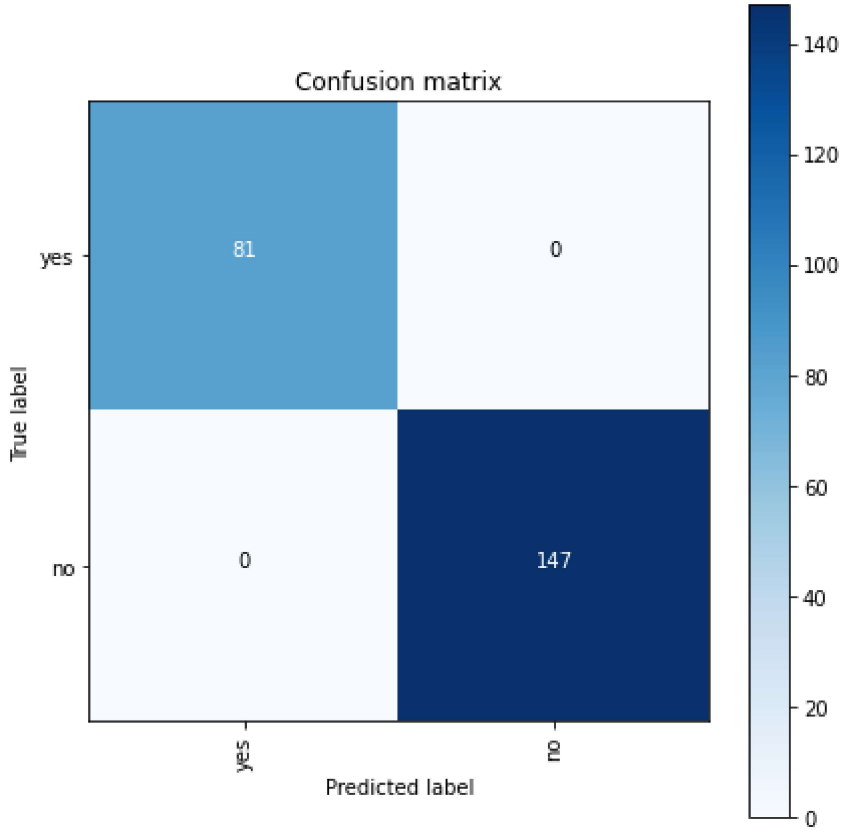
In [61]:
```
labels=['yes','no']
predictions=model.predict(X_val)
predictions=[1 if x>0.5 else 0 for x in predictions]
accuracy=accuracy_score(Y_val,predictions)
print('Val Accuracy = %.2f' % accuracy)
confusion_mtx=confusion_matrix(Y_val,predictions)
cm=plot_confusion_matrix(confusion_mtx,classes=labels,normalize=False)
```

```
Val Accuracy = 1.00
```

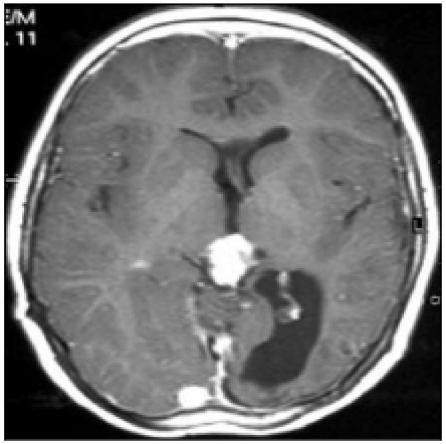


In [64]:
```
predictions=model.predict(X_test)
predictions=[1 if x>0.5 else 0 for x in predictions]
accuracy=accuracy_score(Y_test,predictions)
print('Val Accuracy = %.2f' % accuracy)
confusion_mtx=confusion_matrix(Y_test,predictions)
cm=plot_confusion_matrix(confusion_mtx,classes=labels,normalize=False)
```

```
Val Accuracy = 1.00
```



In [66]:
```
for i in range(10):
    plt.figure()
    plt.imshow(X_test[i])
    plt.xticks([])
    plt.yticks([])
    plt.title(f'Actual class: {Y_test[i]}\nPredicted class: {predictions[i]}')
    plt.show()
```



In [ ]: