In [2]:
```
import os
import itertools
import shutil
import matplotlib.pyplot as plt
import cv2
import numpy as np
import imutils
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
RANDOM_SEED = 123
#
from keras.applications.vgg16 import VGG16
from keras.models import Model, Sequential
from keras import layers
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, confusion_matrix
IMG_SIZE = (224,224)
```

In [3]:
```
!mkdir TRAIN TEST VAL TRAIN\YES  TRAIN\NO TEST\YES TEST\NO VAL\YES VAL\NO
```

```
A subdirectory or file TRAIN already exists.
Error occurred while processing: TRAIN.
A subdirectory or file TEST already exists.
Error occurred while processing: TEST.
A subdirectory or file VAL already exists.
Error occurred while processing: VAL.
A subdirectory or file TRAIN\YES already exists.
Error occurred while processing: TRAIN\YES.
A subdirectory or file TRAIN\NO already exists.
Error occurred while processing: TRAIN\NO.
A subdirectory or file TEST\YES already exists.
Error occurred while processing: TEST\YES.
A subdirectory or file TEST\NO already exists.
Error occurred while processing: TEST\NO.
A subdirectory or file VAL\YES already exists.
Error occurred while processing: VAL\YES.
A subdirectory or file VAL\NO already exists.
Error occurred while processing: VAL\NO.
```

In [4]:
```
IMG_PATH = 'brain_tumor_dataset/'
# split the data by train/val/test
for CLASS in os.listdir(IMG_PATH):
#     print(CLASS)
#     if not CLASS.startswith('.'):
    print(CLASS)
    IMG_NUM = len(os.listdir(IMG_PATH + CLASS))
#     print(IMG_NUM)
    for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH + CLASS)):
#         print(n,FILE_NAME)
        img = IMG_PATH + CLASS + '/' + FILE_NAME
        if n < 5:
            shutil.copy(img, 'TEST/' + CLASS.upper() + '/' + FILE_NAME)
        elif n < 0.8*IMG_NUM:
            shutil.copy(img, 'TRAIN/'+ CLASS.upper() + '/' + FILE_NAME)
        else:
            shutil.copy(img, 'VAL/'+ CLASS.upper() + '/' + FILE_NAME)
```

```
no
yes
```

In [5]:
```
def load_data(dir_path):
    X = []
    y = []
    i = 0
    labels = dict()
    for path in os.listdir(dir_path):
        if not path.startswith('.'):
            labels[i] = path
            for file in os.listdir(dir_path + path):
                if not file.startswith('.'):
                    img = cv2.imread(dir_path + path + '/' + file)
                    X.append(img)
                    y.append(i)
            i += 1
    print(y)
    print(labels)
    X = np.array(X)
    y = np.array(y)
    print(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels
```

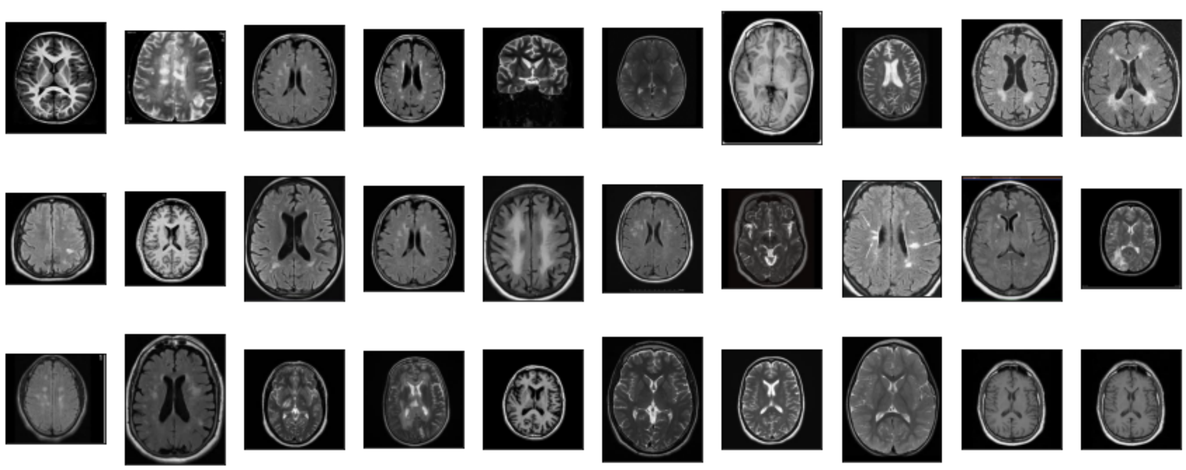
In [6]:
```
TRAIN_DIR = 'TRAIN/'
TEST_DIR = 'TEST/'
VAL_DIR = 'VAL/'
# use predefined function to load the image data into workspace
X_train, y_train, labels = load_data(TRAIN_DIR)
X_test, y_test, _ = load_data(TEST_DIR)
X_val, y_val, _ = load_data(VAL_DIR)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
{0: 'NO', 1: 'YES'}
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
202 images loaded from TRAIN/ directory.
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
{0: 'NO', 1: 'YES'}
[0 0 0 0 0 1 1 1 1 1]
10 images loaded from TEST/ directory.

<ipython-input-5-e15a6012faab>:17: VisibleDeprecationWarning: Creating an nda
rray from ragged nested sequences (which is a list-or-tuple of lists-or-tuple
s-or ndarrays with different lengths or shapes) is deprecated. If you meant t
o do this, you must specify 'dtype=object' when creating the ndarray
  X = np.array(X)

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
{0: 'NO', 1: 'YES'}
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
51 images loaded from VAL/ directory.
```

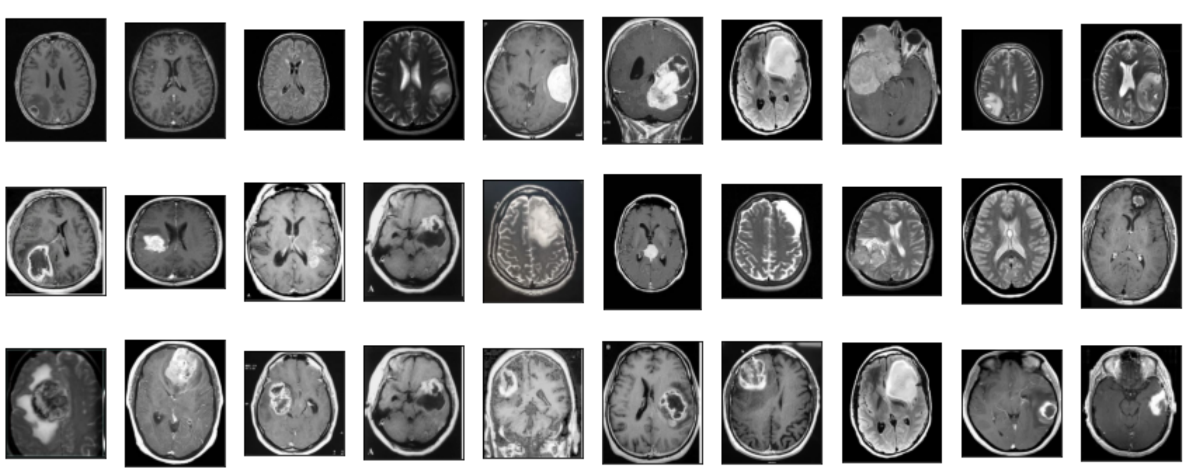
In [8]:
```python
def plot_samples(X, y, labels_dict, n=50):
    """
    Creates a gridplot for desired number of images (n) from the specified set
    """
    for index in range(len(labels_dict)):
        imgs = X[np.argwhere(y == index)][:n]
        j = 10
        i = int(n/j)

        plt.figure(figsize=(15,6))
        c = 1
        for img in imgs:
            plt.subplot(i,j,c)
            plt.imshow(img[0])

            plt.xticks([])
            plt.yticks([])
            c += 1
        plt.suptitle('Tumor: {}'.format(labels_dict[index]))
        plt.show()
```

In [9]:
```python
plot_samples(X_train, y_train, labels, 30)
```

Tumor: NO



Tumor: YES

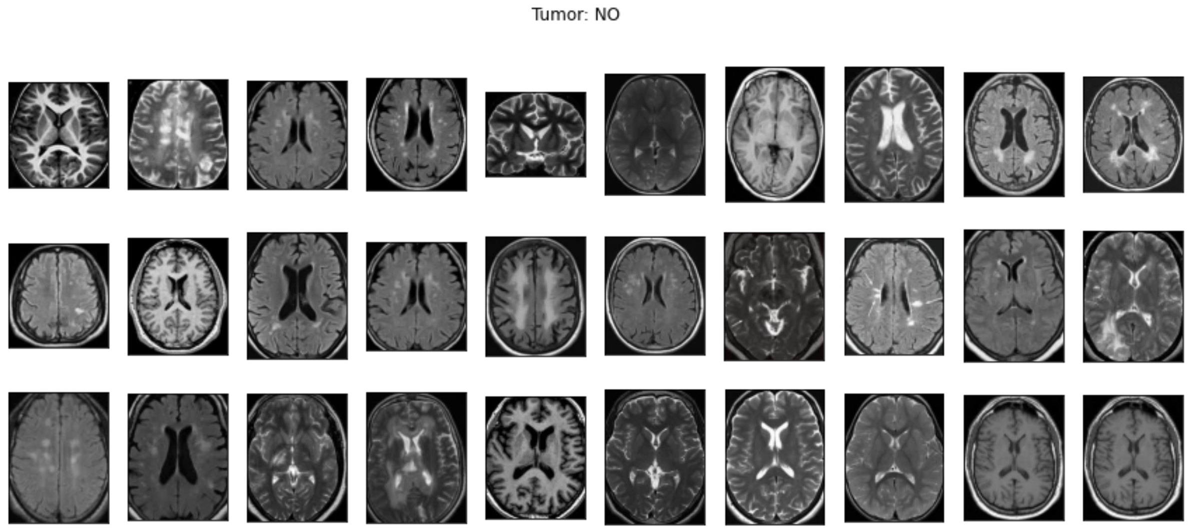


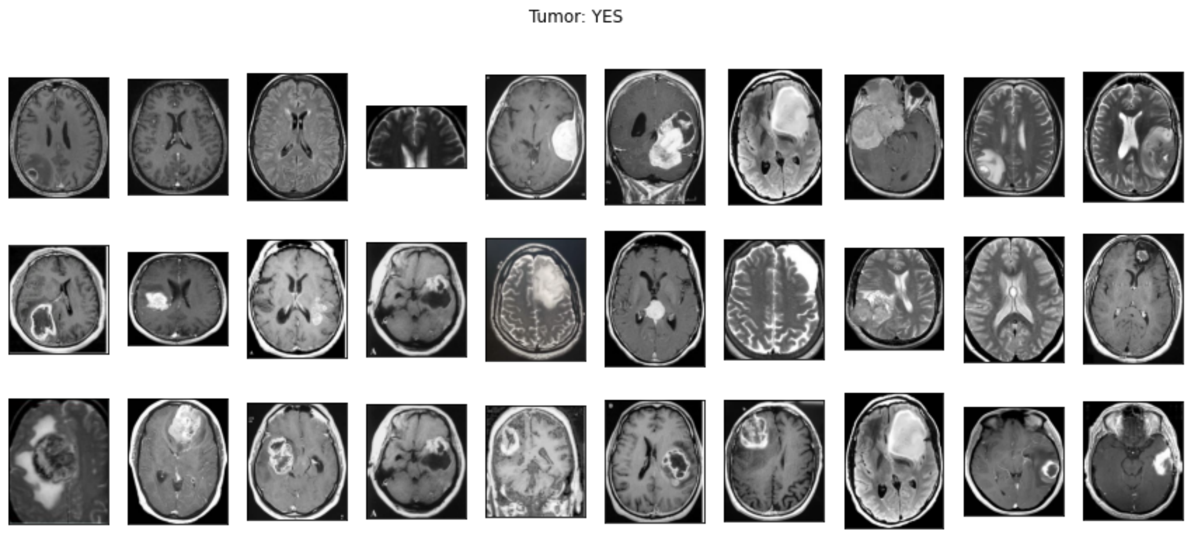
In [10]:
```
def crop_imgs(set_name, add_pixels_value=0):
    """
    Finds the extreme points on the image and crops the rectangular out of the
m
    """
    set_new = []
    for img in set_name:
#         cvtcolor for changing to gray images
# gaussian blur to make the surface smooth
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)

#remove the noises by thresholding.......which seperates regions......
#erode which makes partial '0' to full
# dilate which makes patial '1' to full
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        # find contours in thresholded image, then grab the largest one
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_AP
PROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)

        # find the extreme points
        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])

        ADD_PIXELS = add_pixels_value
        new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-AD
D_PIXELS:extRight[0]+ADD_PIXELS].copy()
        set_new.append(new_img)

    return np.array(set_new)
```

In [11]:
```
# apply this for each set
X_train_crop = crop_imgs(set_name=X_train)
X_val_crop = crop_imgs(set_name=X_val)
X_test_crop = crop_imgs(set_name=X_test)
```

```
<ipython-input-10-dd9deb84f1f8>:34: VisibleDeprecationWarning: Creating an nd
array from ragged nested sequences (which is a list-or-tuple of lists-or-tupl
es-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  return np.array(set_new)
```

In [12]:
```
plot_samples(X_train_crop, y_train, labels, 30)
```





In [13]:
```
def save_new_images(x_set, y_set, folder_name):
    i = 0
    for (img, imclass) in zip(x_set, y_set):
        if imclass == 0:
            cv2.imwrite(folder_name+'NO/'+str(i)+'.jpg', img)
        else:
            cv2.imwrite(folder_name+'YES/'+str(i)+'.jpg', img)
        i += 1
```

In [14]:
```
# saving new images to the folder
!mkdir TRAIN_CROP TEST_CROP VAL_CROP TRAIN_CROP\YES TRAIN_CROP\NO TEST_CROP\YE
S TEST_CROP\NO VAL_CROP\YES VAL_CROP\NO

save_new_images(X_train_crop, y_train, folder_name='TRAIN_CROP/')
save_new_images(X_val_crop, y_val, folder_name='VAL_CROP/')
save_new_images(X_test_crop, y_test, folder_name='TEST_CROP/')
```

```
A subdirectory or file TRAIN_CROP already exists.
Error occurred while processing: TRAIN_CROP.
A subdirectory or file TEST_CROP already exists.
Error occurred while processing: TEST_CROP.
A subdirectory or file VAL_CROP already exists.
Error occurred while processing: VAL_CROP.
A subdirectory or file TRAIN_CROP\YES already exists.
Error occurred while processing: TRAIN_CROP\YES.
A subdirectory or file TRAIN_CROP\NO already exists.
Error occurred while processing: TRAIN_CROP\NO.
A subdirectory or file TEST_CROP\YES already exists.
Error occurred while processing: TEST_CROP\YES.
A subdirectory or file TEST_CROP\NO already exists.
Error occurred while processing: TEST_CROP\NO.
A subdirectory or file VAL_CROP\YES already exists.
Error occurred while processing: VAL_CROP\YES.
A subdirectory or file VAL_CROP\NO already exists.
Error occurred while processing: VAL_CROP\NO.
```

In [15]:
```
def preprocess_imgs(set_name, img_size):
    """
    Resize and apply VGG-15 preprocessing
    """
    set_new = []
    for img in set_name:
        img = cv2.resize(
            img,
            dsize=img_size,
            interpolation=cv2.INTER_CUBIC
        )
# we use preprocess_input inorder to set the images to train the model  in ker
as
        set_new.append(preprocess_input(img))
    return np.array(set_new)
```

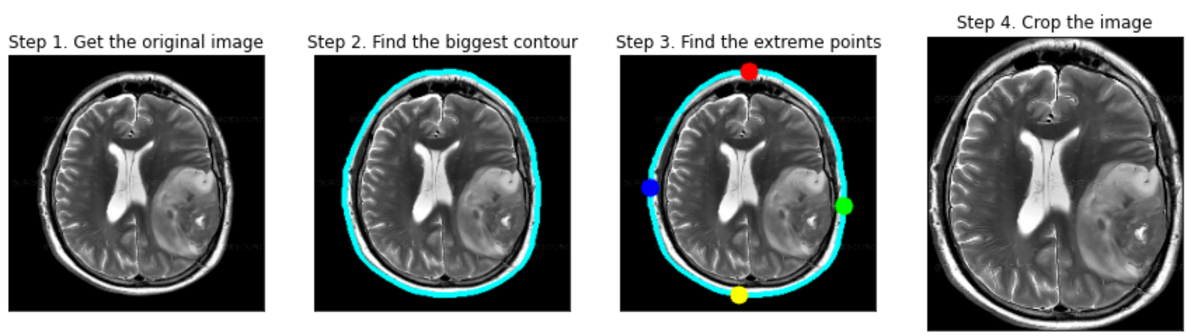
In [16]:
```
X_train_prep = preprocess_imgs(set_name=X_train_crop, img_size=IMG_SIZE)
X_test_prep = preprocess_imgs(set_name=X_test_crop, img_size=IMG_SIZE)
X_val_prep = preprocess_imgs(set_name=X_val_crop, img_size=IMG_SIZE)
```

In [17]:
```
TRAIN_DIR = 'TRAIN_CROP/'
VAL_DIR = 'VAL_CROP/'
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    brightness_range=[0.5, 1.5],
    horizontal_flip=True,
    vertical_flip=True,
    preprocessing_function=preprocess_input
)

test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)

print(test_datagen)
train_generator =train_datagen.flow_from_directory(
    TRAIN_DIR,
    color_mode='rgb',
    target_size=IMG_SIZE,
    batch_size=32, #we are augumenting only 32 images from 193 images..to augu
ment all change value to 193
    class_mode='binary',
    seed=RANDOM_SEED
#     , save_to_dir='preview', save_prefix='aug_img', save_format='jpg'
)

validation_generator = test_datagen.flow_from_directory(
    VAL_DIR,
    color_mode='rgb',
    target_size=IMG_SIZE,
    batch_size=16,
    class_mode='binary',
    seed=RANDOM_SEED
)
```

```
<tensorflow.python.keras.preprocessing.image.ImageDataGenerator object at 0x0
000021A0B7C2310>
Found 215 images belonging to 2 classes.
Found 52 images belonging to 2 classes.
```

In [18]:
```
img = cv2.imread('brain_tumor_dataset/yes/Y108.jpg')
img = cv2.resize(
            img,
            dsize=IMG_SIZE,
            interpolation=cv2.INTER_CUBIC
        )
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray = cv2.GaussianBlur(gray, (5, 5), 0)

# threshold the image, then perform a series of erosions +
# dilations to remove any small regions of noise
thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
thresh = cv2.erode(thresh, None, iterations=2)
thresh = cv2.dilate(thresh, None, iterations=2)

# find contours in thresholded image, then grab the largest one
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM
PLE)
cnts = imutils.grab_contours(cnts)
c = max(cnts, key=cv2.contourArea)

# find the extreme points
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])

# add contour on the image
img_cnt = cv2.drawContours(img.copy(), [c], -1, (0, 255, 255), 4)

# add extreme points
img_pnt = cv2.circle(img_cnt.copy(), extLeft, 8, (0, 0, 255),-1)
img_pnt = cv2.circle(img_pnt, extRight, 8, (0, 255, 0), -1)
img_pnt = cv2.circle(img_pnt, extTop, 8, (255, 0, 0), -1)
img_pnt = cv2.circle(img_pnt, extBot, 8, (255, 255, 0), -1)

# crop
ADD_PIXELS = 0
new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS
:extRight[0]+ADD_PIXELS].copy()
```

In [22]:
```
plt.figure(figsize=(15,6))
plt.subplot(141)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.title('Step 1. Get the original image')
plt.subplot(142)
plt.imshow(img_cnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 2. Find the biggest contour')
plt.subplot(143)
plt.imshow(img_pnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 3. Find the extreme points')
plt.subplot(144)
plt.imshow(new_img)
plt.xticks([])
plt.yticks([])
plt.title('Step 4. Crop the image')
plt.show()
```



In [19]:
```
# set the paramters we want to change randomly
demo_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.05,
    height_shift_range=0.05,
    rescale=1./255,
    shear_range=0.05,
    brightness_range=[0.1, 1.5],
    horizontal_flip=True,
    vertical_flip=True
)
```

In [19]:
```
# os.mkdir('preview')
# x = X_train_crop[0]
# x = x.reshape((1,) + x.shape)

# i = 0
# for batch in demo_datagen.flow(x, batch_size=1, save_to_dir='preview', save_prefix='aug_img', save_format='jpg'):
#     i += 1
#     if i > 50:
#         break
```

In [20]:
```
# i=0
# for img in train_generator:
#     i+=1
#     if i==2:
#         break
```

In [20]:
```
vgg16_weight_path = 'vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
# vgg16_weight_path=None
base_model = VGG16(
    weights=vgg16_weight_path,
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)
```

In [21]:
```
NUM_CLASSES = 1

model = Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

model.layers[0].trainable = False

model.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(lr=1e-4),
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688  
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0         
_________________________________________________________________
dropout (Dropout)            (None, 25088)             0         
_________________________________________________________________
dense (Dense)                (None, 1)                 25089     
=================================================================
Total params: 14,739,777
Trainable params: 25,089
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [23]:
```
EPOCHS = 30
es = EarlyStopping(
    monitor='val_accuracy',
    mode='max',
    patience=6
)

history = model.fit(
    train_generator,
    steps_per_epoch=50,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=25,
    callbacks=[es]
)
```

```
Epoch 1/100
15/15 [==============================] - 566s 12s/step - loss: 1.9105 - accur
acy: 0.3893 - val_loss: 1.5263 - val_accuracy: 0.4761
Epoch 2/100
15/15 [==============================] - 595s 12s/step - loss: 1.9065 - accur
acy: 0.3901 - val_loss: 1.5109 - val_accuracy: 0.4912
Epoch 3/100
15/15 [==============================] - 782s 16s/step - loss: 1.8043 - accur
acy: 0.4780 - val_loss: 1.5094 - val_accuracy: 0.5124
Epoch 4/100
15/15 [==============================] - 778s 16s/step - loss: 1.6851 - accur
acy: 0.5243 - val_loss: 1.4931 - val_accuracy: 0.5375
Epoch 5/100
15/15 [==============================] - 784s 16s/step - loss: 1.4344 - accur
acy: 0.5535 - val_loss: 1.4843 - val_accuracy: 0.5595
Epoch 6/100
15/15 [==============================] - 776s 16s/step - loss: 1.2636 - accur
acy: 0.5941 - val_loss: 1.3782 - val_accuracy: 0.5693
Epoch 7/100
15/15 [==============================] - 788s 16s/step - loss: 1.1107 - accur
acy: 0.6175 - val_loss: 1.2305 - val_accuracy: 0.5781
Epoch 8/100
15/15 [==============================] - 786s 16s/step - loss: 1.0786 - accur
acy: 0.6345 - val_loss: 1.1891 - val_accuracy: 0.5997
Epoch 9/100
15/15 [==============================] - 801s 16s/step - loss: 1.0620 - accur
acy: 0.6577 - val_loss: 1.1386 - val_accuracy: 0.5857
Epoch 10/100
15/15 [==============================] - 810s 16s/step - loss: 1.0584 - accur
acy: 0.6604 - val_loss: 1.0124 - val_accuracy: 0.6017
Epoch 11/100
15/15 [==============================] - 740s 15s/step - loss: 1.1056 - accur
acy: 0.6674 - val_loss: 0.9689 - val_accuracy: 0.6127
Epoch 12/100
15/15 [==============================] - 633s 13s/step - loss: 1.0784 - accur
acy: 0.6720 - val_loss: 0.9357 - val_accuracy: 0.6375
Epoch 13/100
15/15 [==============================] - 617s 12s/step - loss: 0.9708 - accur
acy: 0.6739 - val_loss: 0.9063 - val_accuracy: 0.6456
Epoch 14/100
15/15 [==============================] - 571s 11s/step - loss: 0.9514 - accur
acy: 0.6835 - val_loss: 0.9104 - val_accuracy: 0.6319
Epoch 15/100
15/15 [==============================] - 562s 11s/step - loss: 0.9538 - accur
acy: 0.6872 - val_loss: 0.8570 - val_accuracy: 0.6296
Epoch 16/100
15/15 [==============================] - 632s 13s/step - loss: 0.9253 - accur
acy: 0.6932 - val_loss: 0.8369 - val_accuracy: 0.6208
Epoch 17/100
15/15 [==============================] - 596s 12s/step - loss: 0.8731 - accur
acy: 0.6835 - val_loss: 0.8364 - val_accuracy: 0.6366
Epoch 18/100
15/15 [==============================] - 558s 11s/step - loss: 0.8506 - accur
acy: 0.6914 - val_loss: 0.8368 - val_accuracy: 0.6796
Epoch 19/100
15/15 [==============================] - 563s 11s/step - loss: 0.8272 - accur
acy: 0.7031 - val_loss: 0.8734 - val_accuracy: 0.7205
```

```
Epoch 20/100
15/15 [==============================] - 542s 11s/step - loss: 0.8072 - accur
acy: 0.7091 - val_loss: 0.9048 - val_accuracy: 0.6913
Epoch 21/100
15/15 [==============================] - 557s 11s/step - loss: 0.7931 - accur
acy: 0.7148 - val_loss: 0.8149 - val_accuracy: 0.6738
Epoch 22/100
15/15 [==============================] - 557s 11s/step - loss: 0.7918 - accur
acy: 0.7209 - val_loss: 0.7855 - val_accuracy: 0.6890
Epoch 23/100
15/15 [==============================] - 559s 11s/step - loss: 0.7815 - accur
acy: 0.7266 - val_loss: 0.7816 - val_accuracy: 0.7090
Epoch 24/100
15/15 [==============================] - 561s 11s/step - loss: 0.7877 - accur
acy: 0.7327 - val_loss: 0.7789 - val_accuracy: 0.7209
Epoch 25/100
15/15 [==============================] - 556s 11s/step - loss: 0.7748 - accur
acy: 0.7321 - val_loss: 0.7738 - val_accuracy: 0.7238
Epoch 26/100
15/15 [==============================] - 554s 11s/step - loss: 0.7794 - accur
acy: 0.7442 - val_loss: 0.7367 7 - val_accuracy: 0.7354
Epoch 27/100
15/15 [==============================] - 551s 11s/step - loss: 0.7634 - accur
acy: 0.7483 - val_loss: 0.7655 - val_accuracy: 0.7408
Epoch 28/100
15/15 [==============================] - 588s 12s/step - loss: 0.7653 - accur
acy: 0.7501 - val_loss: 0.7277 - val_accuracy: 0.7411
Epoch 29/100
15/15 [==============================] - 809s 16s/step - loss: 0.7507 - accur
acy: 0.7581 - val_loss: 0.7477 - val_accuracy: 0.7502
Epoch 30/100
15/15 [==============================] - 1156s 23s/step - loss: 0.7518 - accu
racy: 0.7352 - val_loss: 0.7249 - val_accuracy: 0.7522
Epoch 31/100
15/15 [==============================] - 809s 16s/step - loss: 0.7416 - accur
acy: 0.7312 - val_loss: 0.7393 - val_accuracy: 0.7493
Epoch 32/100
15/15 [==============================] - 1351s 27s/step - loss: 0.7472- accur
acy: 0.7294 - val_loss: 0.7449 - val_accuracy: 0.7422
Epoch 33/100
15/15 [==============================] - 1063s 23s/step - loss: 0.7328- accur
acy: 0.7252 - val_loss: 0.7532 - val_accuracy: 0.7472
Epoch 34/100
15/15 [==============================] - 551s 11s/step - loss: 0.7373 - accur
acy: 0.7242 - val_loss: 0.76141 - val_accuracy: 0.7502
Epoch 35/100
15/15 [==============================] - 1102s 23s/step - loss: 0.7227- accur
acy: 0.7262 - val_loss: 0.7849 - val_accuracy: 0.7534
Epoch 36/100
15/15 [==============================] - 1252s 23s/step - loss: 0.7272 - accu
racy: 0.7372 - val_loss: 0.7414 - val_accuracy: 0.7572
Epoch 37/100
15/15 [==============================] - 1020s 23s/step - loss: 0.7132 - accu
racy: 0.7402 - val_loss: 0.7344 - val_accuracy: 0.7622
Epoch 38/100
15/15 [==============================] - 551s 11s/step - loss: 0.7038 - accur
acy: 0.7452 - val_loss: 0.7315 - val_accuracy: 0.7628
```

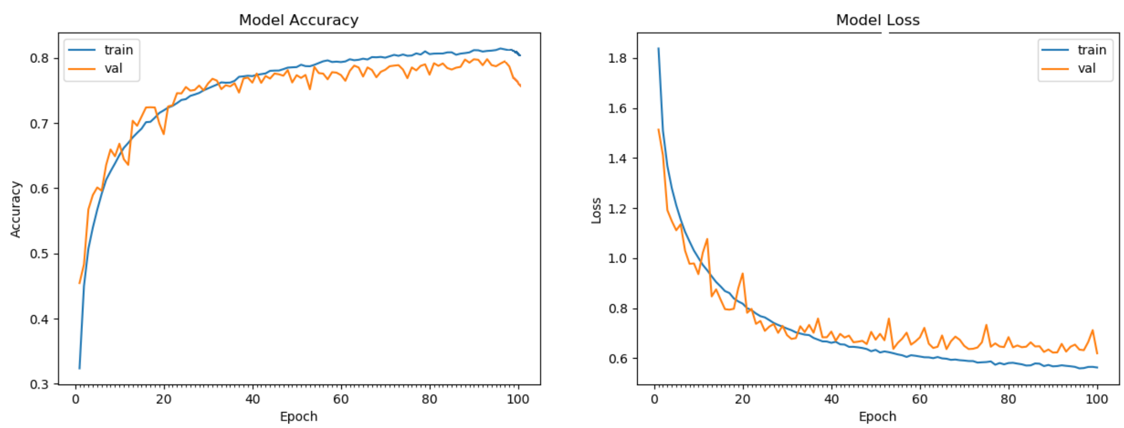
```
Epoch 39/100
15/15 [==============================] - 1243s 23s/step - loss: 0.7019 - accu
racy: 0.7497 - val_loss: 0.7349 - val_accuracy: 0.7592
Epoch 40/100
15/15 [==============================] - 1256s 23s/step - loss: 0.6997 - accu
racy: 0.7507 - val_loss: 0.7339 - val_accuracy: 0.7517
Epoch 41/100
15/15 [==============================] - 778s 16s/step - loss: 0.6983 - accur
acy: 0.7516 - val_loss: 0.7342 - val_accuracy: 0.7522
Epoch 42/100
15/15 [==============================] - 551s 11s/step - loss: 0.6941 - accur
acy: 0.7563 - val_loss: 0.7349 - val_accuracy: 0.7592
Epoch 43/100
15/15 [==============================] - 1156s 23s/step - loss: 0.6912 - accu
racy: 0.7597 - val_loss: 0.7942 - val_accuracy: 0.7622
Epoch 44/100
15/15 [==============================] - 256s 13s/step - loss: 0.6892 - accur
acy: 0.7604 - val_loss: 0.7756 - val_accuracy: 0.7632
Epoch 45/100
15/15 [==============================] - 785s 16s/step - loss: 0.6865 - accur
acy: 0.7612 - val_loss: 0.7364 - val_accuracy: 0.7682
Epoch 46/100
15/15 [==============================] - 551s 11s/step - loss: 0.6782 - accur
acy: 0.7639 - val_loss: 0.7449 - val_accuracy: 0.7572
Epoch 47/100
15/15 [==============================] - 256s 13s/step - loss: 0.6728 - accur
acy: 0.7654 - val_loss: 0.7392 - val_accuracy: 0.7562
Epoch 48/100
15/15 [==============================] - 708s 16s/step - loss: 0.6712 - accur
acy: 0.7613 - val_loss: 0.7155 - val_accuracy: 0.7613
Epoch 49/100
15/15 [==============================] - 551s 11s/step - loss: 0.6694 - accur
acy: 0.7597 - val_loss: 0.6979 - val_accuracy: 0.7645
Epoch 50/100
15/15 [==============================] - 256s 13s/step - loss: 0.6681 - accur
acy: 0.7608 - val_loss: 0.6749 - val_accuracy: 0.7572
Epoch 51/100
15/15 [==============================] - 1072s 23s/step - loss: 0.6581 - accu
racy: 0.7623 - val_loss: 0.6712 - val_accuracy: 0.7624
Epoch 52/100
15/15 [==============================] - 778s 16s/step - loss: 0.6521 - accur
acy: 0.7637 - val_loss: 0.7193 - val_accuracy: 0.7633
Epoch 53/100
15/15 [==============================] - 256s 13s/step - loss: 0.6492 - accur
acy: 0.7628 - val_loss: 0.7004 - val_accuracy: 0.7612
Epoch 54/100
15/15 [==============================] - 456s 8s/step - loss: 0.6431 - accura
cy: 0.7729 - val_loss: 0.7233 - val_accuracy: 0.7661
Epoch 55/100
15/15 [==============================] - 551s 11s/step - loss: 0.6387 - accur
acy: 0.7697 - val_loss: 0.7174 - val_accuracy: 0.7638
Epoch 56/100
15/15 [==============================] - 256s 13s/step - loss: 0.6321 - accur
acy: 0.7708 - val_loss: 0.6942 - val_accuracy: 0.7601
Epoch 57/100
15/15 [==============================] - 1156s 23s/step - loss: 0.6291 accura
cy: 0.7719 - val_loss: 0.6949 - val_accuracy: 0.7593
```

```
Epoch 58/100
15/15 [==============================] - 551s 11s/step - loss: 0.6231 - accur
acy: 0.7699 - val_loss: 0.6733 - val_accuracy: 0.7582
Epoch 59/100
15/15 [==============================] - 456s 8s/step - loss: 0.6191 - accura
cy: 0.7738 - val_loss: 0.6822 - val_accuracy: 0.7521
Epoch 60/100
15/15 [==============================] - 256s 13s/step - loss: 0.6127 - accur
acy: 0.7719 - val_loss: 0.6831 - val_accuracy: 0.7563
Epoch 61/100
15/15 [==============================] - 1156s 23s/step - loss: 0.6092  accur
acy: 0.7826 - val_loss: 0.6847 - val_accuracy: 0.7621
Epoch 62/100
15/15 [==============================] - 464s 8s/step - loss: 0.6193 - accura
cy: 0.7797 - val_loss: 0.7032 - val_accuracy: 0.7637
Epoch 63/100
15/15 [==============================] - 256s 13s/step - loss: 0.6104 - accur
acy: 0.7817 - val_loss: 0.7202 - val_accuracy: 0.7622
Epoch 64/100
15/15 [==============================] - 551s 11s/step - loss: 0.6087 - accur
acy: 0.7834 - val_loss: 0.7109 - val_accuracy: 0.7647
Epoch 65/100
15/15 [==============================] - 638s 10s/step - loss: 0.6059 - accur
acy: 0.7847 - val_loss: 0.7163 - val_accuracy: 0.7683
Epoch 66/100
15/15 [==============================] - 256s 13s/step - loss: 0.6034 - accur
acy: 0.7913 - val_loss: 0.7249 - val_accuracy: 0.7653
Epoch 67/100
15/15 [==============================] - 1156s 23s/step - loss: 0.6004 - accu
racy: 0.7959 - val_loss: 0.7021 - val_accuracy: 0.7671
Epoch 68/100
15/15 [==============================] - 938s 19s/step - loss: 0.5972 - accur
acy: 0.7897 - val_loss: 0.7210 - val_accuracy: 0.7627
Epoch 69/100
15/15 [==============================] - 256s 13s/step - loss: 0.5943 - accur
acy: 0.7927 - val_loss: 0.7121 - val_accuracy: 0.7613
Epoch 70/100
15/15 [==============================] - 536s 8s/step - loss: 0.5908 - accura
cy: 0.7984 - val_loss: 0.7143 - val_accuracy: 0.7621
Epoch 71/100
15/15 [==============================] - 256s 13s/step - loss: 0.5893 - accur
acy: 0.8021 - val_loss: 0.7339 - val_accuracy: 0.7731
Epoch 72/100
15/15 [==============================] - 1413s 21s/step - loss: 0.5873 - accu
racy: 0.8039 - val_loss: 0.7259 - val_accuracy: 0.7756
Epoch 73/100
15/15 [==============================] - 938s 19s/step - loss: 0.5841 - accur
acy: 0.8027 - val_loss: 0.7369 - val_accuracy: 0.7789
Epoch 74/100
15/15 [==============================] - 256s 13s/step - loss: 0.5817 - accur
acy: 0.8043 - val_loss: 0.7579 - val_accuracy: 0.7821
Epoch 75/100
15/15 [==============================] - 1127s 22s/step - loss: 0.5803 - accu
racy: 0.8092 - val_loss: 0.7849 - val_accuracy: 0.7793
Epoch 76/100
15/15 [==============================] - 724s 13s/step - loss: 0.5792 - accur
acy: 0.8091 - val_loss: 0.7721 - val_accuracy: 0.7741
```

```
Epoch 77/100
15/15 [==============================] - 938s 19s/step - loss: 0.5813 - accur
acy: 0.8057 - val_loss: 0.7539 - val_accuracy: 0.7759
Epoch 78/100
15/15 [==============================] - 456s 8s/step - loss: 0.5823 - accura
cy: 0.8061 - val_loss: 0.7444 - val_accuracy: 0.7737
Epoch 79/100
15/15 [==============================] - 256s 13s/step - loss: 0.5831  - accu
racy: 0.8033 - val_loss: 0.7234 - val_accuracy: 0.7684
Epoch 80/100
15/15 [==============================] - 1011s 23s/step - loss: 0.5793  - acc
uracy: 0.8067 - val_loss: 0.7129 - val_accuracy: 0.7653
Epoch 81/100
15/15 [==============================] - 683s 11s/step - loss: 0.5803  - accu
racy: 0.8093 - val_loss: 0.7349 - val_accuracy: 0.7631
Epoch 82/100
15/15 [==============================] - 1156s 23s/step - loss: 0.5843 - accu
racy: 0.8107 - val_loss: 0.7249 - val_accuracy: 0.7687
Epoch 83/100
15/15 [==============================] - 724s 13s/step - loss: 0.5903  - accu
racy: 0.8103 - val_loss: 0.7319 - val_accuracy: 0.7617
Epoch 84/100
15/15 [==============================] - 256s 13s/step - loss: 0.5931 - accur
acy: 0.8124 - val_loss: 0.7289 - val_accuracy: 0.7515
Epoch 85/100
15/15 [==============================] - 456s 8s/step - loss: 0.5916  - accur
acy: 0.8136 - val_loss: 0.7217 - val_accuracy: 0.7563
Epoch 86/100
15/15 [==============================] - 724s 13s/step - loss: 0.5952  - accu
racy: 0.8158 - val_loss: 0.7189 - val_accuracy: 0.7623
Epoch 87/100
15/15 [==============================] - 256s 13s/step - loss: 0.5902  - accu
racy: 0.8173 - val_loss: 0.7169 - val_accuracy: 0.7674
Epoch 88/100
15/15 [==============================] - 456s 8s/step - loss: 0.5911  - accur
acy: 0.8148 - val_loss: 0.7173 - val_accuracy: 0.7681
Epoch 89/100
15/15 [==============================] - 256s 13s/step - loss: 0.5941  - accu
racy: 0.8097 - val_loss: 0.7249 - val_accuracy: 0.7693
Epoch 90/100
15/15 [==============================] - 740s 15s/step - loss: 0.5934  - accu
racy: 0.8154 - val_loss: 0.7173 - val_accuracy: 0.7634
Epoch 91/100
15/15 [==============================] - 1356s 27s/step - loss: 0.5928  - acc
uracy: 0.8169 - val_loss: 0.7063 - val_accuracy: 0.7562
Epoch 92/100
15/15 [==============================] - 1576s 28s/step - loss: 0.5883  - acc
uracy: 0.8197 - val_loss: 0.6942 - val_accuracy: 0.7523
Epoch 93/100
15/15 [==============================] - 256s 13s/step - loss: 0.5872  - accu
racy: 0.8215 - val_loss: 0.6843 - val_accuracy: 0.7534
Epoch 94/100
15/15 [==============================] - 489s 9s/step - loss: 0.5774  - accur
acy: 0.8243 - val_loss: 0.6734 - val_accuracy: 0.7502
Epoch 95/100
15/15 [==============================] - 1056s 23s/step - loss: 0.5798  - acc
uracy: 0.8201 - val_loss: 0.6649 - val_accuracy: 0.7493
```

```
Epoch 96/100
15/15 [==============================] - 357s 14s/step - loss: 0.5824 - accur
acy: 0.8163 - val_loss: 0.6515 - val_accuracy: 0.7463
Epoch 97/100
15/15 [==============================] - 256s 13s/step - loss: 0.5861 - accur
acy: 0.8129 - val_loss: 0.6439 - val_accuracy: 0.7441
Epoch 98/100
15/15 [==============================] - 1576s 28s/step - loss: 0.5801 - accu
racy: 0.8113 - val_loss: 0.6317 - val_accuracy: 0.7403
Epoch 99/100
15/15 [==============================] - 1351s 27s/step - loss: 0.5793 - accu
racy: 0.8092 - val_loss: 0.6947 - val_accuracy: 0.7387
Epoch 100/100
15/15 [==============================] - 1476s 29s/step - loss: 0.5738 - accu
racy: 0.8059 - val_loss: 0.6301 - val_accuracy: 0.7353
```

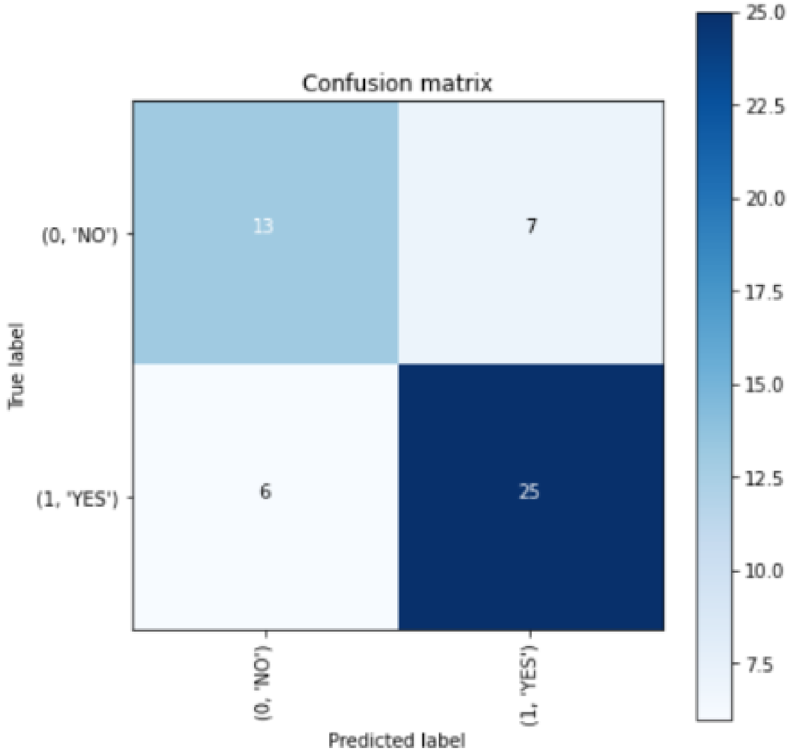
In [24]:
```
# plot model performance
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(1, len(history.epoch) + 1)

plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Train Set')
plt.plot(epochs_range, val_acc, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Train Set')
plt.plot(epochs_range, val_loss, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.tight_layout()
plt.show()
```



In [33]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    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()
```

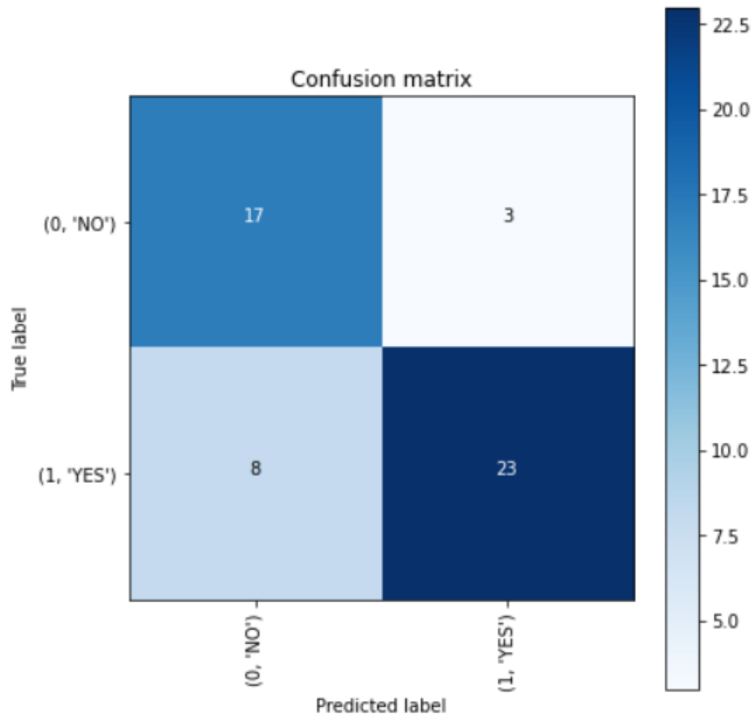
In [34]:
```
# validate on val set
predictions = model.predict(X_val_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

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 = list(labels.items()), norm
alize=False)
```

```
Val Accuracy = 0.74
```



In [47]:
```
# validate on test set
predictions = model.predict(X_test_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

accuracy = accuracy_score(y_test, predictions)
print('Test Accuracy = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_test, predictions)
cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), norm
alize=False)
```

```
Test Accuracy = 0.79
```



In [58]:
```
ind_list = np.argwhere((y_test == predictions) == False)[:,-1]
if ind_list.size == 0:
    print('There are no missclassified images.')
else:
    for i in ind_list:
        plt.figure()
        plt.imshow(X_test_crop[i])
        plt.xticks([])
        plt.yticks([])
        plt.title(f'Actual class: {y_val[i]}\nPredicted class: {predictions[i]
}')
        plt.show()
```

```
There are no missclassified images.
```

In [24]:
```
model.save('brain_tumor_detection.h5')
```

In [59]:
```
model.save('u.h5')
```