In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from PIL import Image
from sklearn import metrics
import os
from glob import glob
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.preprocessing.image import ImageDataGenerator
import warnings
import cv2
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
warnings.filterwarnings('ignore')
```

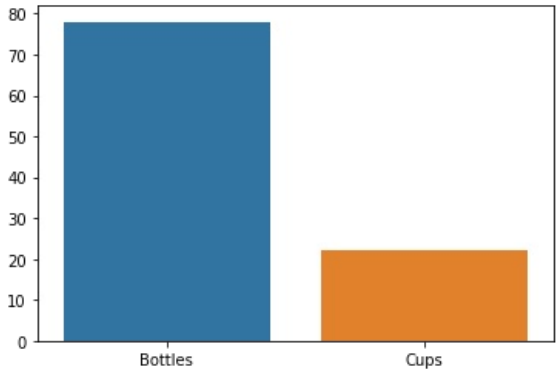
In [2]:
```
# Extracting the compressed dataset

from zipfile import ZipFile
data_path = 'Img.zip'

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

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

In [3]:
```
path = 'Img'
classes = os.listdir(path)
classes
```

Out[3]:
```
['Bottles', 'Cups']
```

In [4]:
```
ans = []
for i in classes:
    ans.append(len(os.listdir(f'{path}/{i}')))
sb.barplot(classes,ans)
plt.show()
```



In [5]:
```
# Scaling and Splitting data

data = tf.keras.utils.image_dataset_from_directory('Img')
data = data.map(lambda x,y: (x/255,y))

n = len(data)
train_size = int(n*.7)
val_size = int(n*.2)+1
test_size = int(n*.1)+1

train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

```
Found 100 files belonging to 2 classes.
```

In [6]:
```
# Building the model

model = keras.models.Sequential([layers.Conv2D(16,(3,3),activation='relu',input_shape=(256,256,3)),
                                 layers.MaxPooling2D(2,2),
                                 layers.Conv2D(32,(3,3),activation='relu'),
                                 layers.MaxPooling2D(2,2),
                                 layers.Conv2D(16,(3,3),activation='relu'),
                                 layers.MaxPooling2D(2,2),
                                 layers.Flatten(),
```

```
                                    layers.Dense(256,activation='relu'),
                                    layers.BatchNormalization(),
                                    layers.Dense(1,activation='sigmoid')])
```

In [7]:
```
# Compiling the model

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

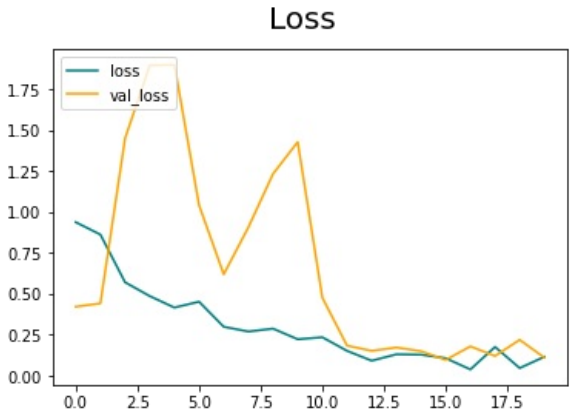
In [8]:
```
logdir = 'logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = logdir)
```

In [9]:
```
# Training the model

hist = model.fit(train, epochs = 20, validation_data = val, callbacks = [tensorboard_callback])
```

```
Epoch 1/20
2/2 [==============================] - 14s 5s/step - loss: 0.9367 - accuracy: 0.5625 - val_loss: 0.4206 - val_acc
uracy: 0.8438
Epoch 2/20
2/2 [==============================] - 9s 5s/step - loss: 0.8612 - accuracy: 0.7031 - val_loss: 0.4411 - val_accu
racy: 0.9062
Epoch 3/20
2/2 [==============================] - 9s 5s/step - loss: 0.5698 - accuracy: 0.7031 - val_loss: 1.4480 - val_accu
racy: 0.1875
Epoch 4/20
2/2 [==============================] - 9s 5s/step - loss: 0.4856 - accuracy: 0.7812 - val_loss: 1.8936 - val_accu
racy: 0.3125
Epoch 5/20
2/2 [==============================] - 10s 6s/step - loss: 0.4154 - accuracy: 0.8594 - val_loss: 1.8979 - val_acc
uracy: 0.2188
Epoch 6/20
2/2 [==============================] - 12s 6s/step - loss: 0.4513 - accuracy: 0.7812 - val_loss: 1.0408 - val_acc
uracy: 0.3125
Epoch 7/20
2/2 [==============================] - 10s 5s/step - loss: 0.2983 - accuracy: 0.9062 - val_loss: 0.6183 - val_acc
uracy: 0.5000
Epoch 8/20
2/2 [==============================] - 10s 5s/step - loss: 0.2692 - accuracy: 0.9531 - val_loss: 0.9057 - val_acc
uracy: 0.4062
Epoch 9/20
2/2 [==============================] - 10s 5s/step - loss: 0.2868 - accuracy: 0.9531 - val_loss: 1.2317 - val_acc
uracy: 0.4375
Epoch 10/20
2/2 [==============================] - 9s 5s/step - loss: 0.2222 - accuracy: 0.9531 - val_loss: 1.4263 - val_accu
racy: 0.2812
Epoch 11/20
2/2 [==============================] - 9s 5s/step - loss: 0.2342 - accuracy: 0.9688 - val_loss: 0.4755 - val_accu
racy: 0.8750
Epoch 12/20
2/2 [==============================] - 9s 5s/step - loss: 0.1512 - accuracy: 0.9844 - val_loss: 0.1831 - val_accu
racy: 1.0000
Epoch 13/20
2/2 [==============================] - 9s 5s/step - loss: 0.0910 - accuracy: 1.0000 - val_loss: 0.1514 - val_accu
racy: 0.9375
Epoch 14/20
2/2 [==============================] - 10s 5s/step - loss: 0.1301 - accuracy: 0.9688 - val_loss: 0.1717 - val_acc
uracy: 0.9062
Epoch 15/20
2/2 [==============================] - 9s 5s/step - loss: 0.1280 - accuracy: 0.9219 - val_loss: 0.1492 - val_accu
racy: 1.0000
Epoch 16/20
2/2 [==============================] - 10s 5s/step - loss: 0.1064 - accuracy: 0.9844 - val_loss: 0.0958 - val_acc
uracy: 1.0000
Epoch 17/20
2/2 [==============================] - 9s 5s/step - loss: 0.0377 - accuracy: 1.0000 - val_loss: 0.1786 - val_accu
racy: 0.8750
Epoch 18/20
2/2 [==============================] - 9s 5s/step - loss: 0.1753 - accuracy: 0.9375 - val_loss: 0.1198 - val_accu
racy: 1.0000
Epoch 19/20
2/2 [==============================] - 9s 5s/step - loss: 0.0460 - accuracy: 1.0000 - val_loss: 0.2187 - val_accu
racy: 0.9688
Epoch 20/20
2/2 [==============================] - 9s 5s/step - loss: 0.1150 - accuracy: 0.9531 - val_loss: 0.1097 - val_accu
racy: 1.0000
```

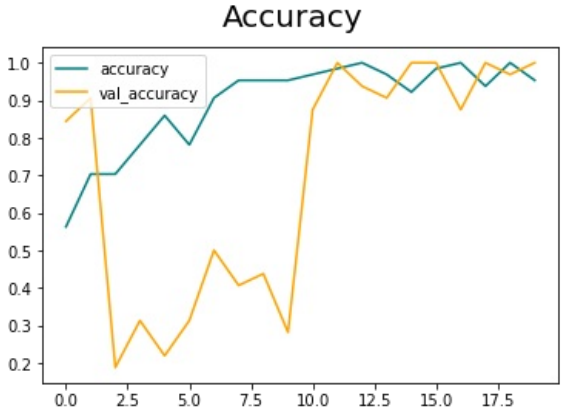
In [10]:
```
# plot performance
```

```
# plot performance

fig = plt.figure()
plt.plot(hist.history['loss'], color = 'teal', label = 'loss')
plt.plot(hist.history['val_loss'], color = 'orange', label = 'val_loss')
fig.suptitle('Loss', fontsize = 20)
plt.legend(loc = "upper left")
plt.show()
```



In [11]:

```
# plot performance

fig = plt.figure()
plt.plot(hist.history['accuracy'], color = 'teal', label = 'accuracy')
plt.plot(hist.history['val_accuracy'], color = 'orange', label = 'val_accuracy')
fig.suptitle('Accuracy', fontsize = 20)
plt.legend(loc = "upper left")
plt.show()
```



In [12]:

```
# Compute precision, recall and accuracy of the model

pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

In [13]:

```
for batch in test.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    pre.update_state(y,yhat)
    re.update_state(y,yhat)
    acc.update_state(y,yhat)
```

```
1/1 [==============================] - 1s 656ms/step
```

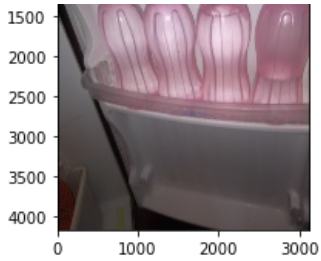
In [14]:

```
print([pre.result().numpy(),re.result().numpy(),acc.result().numpy()])
```

```
[1.0, 1.0, 1.0]
```

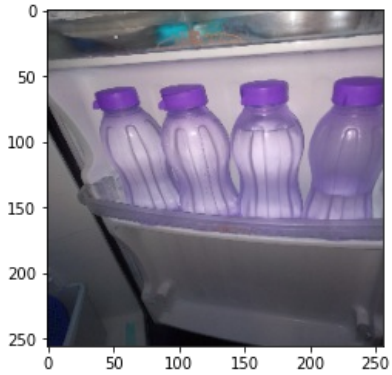
In [15]:

```
# Predicting the model result on a random image

imag = cv2.imread('Bottles (4).jpg')
plt.imshow(cv2.cvtColor(imag,cv2.COLOR_BGR2RGB))
plt.show()
```



In [16]:

```
resize = tf.image.resize(imag, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```



In [17]:

```
yhat = model.predict(np.expand_dims(resize/255,0))
print(yhat)
```

```
1/1 [==============================] - 0s 42ms/step
[[0.08217014]]
```

In [18]:

```
if yhat < 0.5:
    print("Predicted class is Bottles")
else:
    print("Predicted class is Cups")
```

```
Predicted class is Bottles
```