In [38]:
```
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')
```

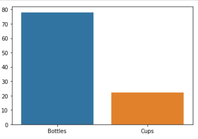
In [39]:
```
# Extracting the compressed dataset

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 [40]:
```
path = 'Img'
classes = os.listdir(path)
classes
```

Out[40]:
```
['Bottles', 'Cups']
```

In [41]:
```
ans = []
for i in classes:
    ans.append(len(os.listdir(f'{path}/{i}')))
sb.barplot(classes,ans)
plt.show()
```



In [42]:
```
# 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 [43]:
```
# 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')])
```

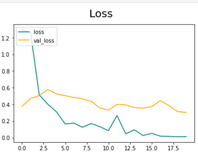
In [44]:
```
# Compiling the model

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

In [45]:
```
logdir = 'logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = logdir)
```

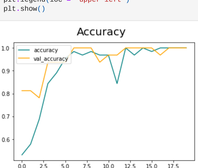
In [46]:
```
# Training the model

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

```
Epoch 1/20
2/2 [==============================] - 17s 6s/step - loss: 1.2951 - accuracy: 0.5312 - val_loss: 0.3763 - val_accuracy: 0.8125
Epoch 2/20
2/2 [==============================] - 11s 5s/step - loss: 1.2627 - accuracy: 0.5781 - val_loss: 0.4692 - val_accuracy: 0.8125
Epoch 3/20
2/2 [==============================] - 10s 5s/step - loss: 0.5123 - accuracy: 0.6875 - val_loss: 0.5051 - val_accuracy: 0.7812
Epoch 4/20
2/2 [==============================] - 10s 5s/step - loss: 0.4006 - accuracy: 0.8438 - val_loss: 0.5772 - val_accuracy: 0.9375
Epoch 5/20
2/2 [==============================] - 10s 5s/step - loss: 0.3092 - accuracy: 0.8906 - val_loss: 0.5251 - val_accuracy: 0.9375
Epoch 6/20
2/2 [==============================] - 10s 5s/step - loss: 0.1646 - accuracy: 0.9531 - val_loss: 0.5047 - val_accuracy: 0.9375
Epoch 7/20
2/2 [==============================] - 12s 7s/step - loss: 0.1746 - accuracy: 0.9844 - val_loss: 0.4813 - val_accuracy: 1.0000
Epoch 8/20
2/2 [==============================] - 10s 5s/step - loss: 0.1246 - accuracy: 0.9688 - val_loss: 0.4677 - val_accuracy: 1.0000
Epoch 9/20
2/2 [==============================] - 11s 6s/step - loss: 0.1696 - accuracy: 0.9844 - val_loss: 0.4365 - val_accuracy: 1.0000
Epoch 10/20
2/2 [==============================] - 10s 6s/step - loss: 0.1320 - accuracy: 0.9688 - val_loss: 0.3581 - val_accuracy: 0.9375
Epoch 11/20
2/2 [==============================] - 10s 5s/step - loss: 0.0835 - accuracy: 0.9688 - val_loss: 0.3298 - val_accuracy: 0.9688
Epoch 12/20
2/2 [==============================] - 9s 5s/step - loss: 0.2650 - accuracy: 0.8438 - val_loss: 0.3989 - val_accuracy: 0.9688
Epoch 13/20
2/2 [==============================] - 8s 4s/step - loss: 0.0460 - accuracy: 1.0000 - val_loss: 0.3940 - val_accuracy: 1.0000
Epoch 14/20
2/2 [==============================] - 8s 4s/step - loss: 0.0938 - accuracy: 0.9688 - val_loss: 0.3611 - val_accuracy: 1.0000
Epoch 15/20
2/2 [==============================] - 8s 4s/step - loss: 0.0254 - accuracy: 1.0000 - val_loss: 0.3539 - val_accuracy: 1.0000
Epoch 16/20
2/2 [==============================] - 8s 4s/step - loss: 0.0513 - accuracy: 0.9844 - val_loss: 0.3722 - val_accuracy: 1.0000
Epoch 17/20
2/2 [==============================] - 9s 5s/step - loss: 0.0180 - accuracy: 1.0000 - val_loss: 0.4442 - val_accuracy: 0.9688
Epoch 18/20
2/2 [==============================] - 8s 4s/step - loss: 0.0147 - accuracy: 1.0000 - val_loss: 0.3847 - val_accuracy: 1.0000
Epoch 19/20
2/2 [==============================] - 9s 5s/step - loss: 0.0110 - accuracy: 1.0000 - val_loss: 0.3130 - val_accuracy: 1.0000
Epoch 20/20
2/2 [==============================] - 10s 5s/step - loss: 0.0128 - accuracy: 1.0000 - val_loss: 0.3026 - val_accuracy: 1.0000
```

In [47]:
```
# 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 [48]:
```
# 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 [49]:
```
# Compute precision, recall and accuracy of the model

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

In [50]:
```
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 [==============================] - 0s 367ms/step
```

In [51]:
```
print([pre.result().numpy(),re.result().numpy(),acc.result().numpy()])
```

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

In [52]:
```
# 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 [53]:
```
resize = tf.image.resize(imag, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```

In [54]:
```
yhat = model.predict(np.expand_dims(resize/255,0))
print(yhat)
```

```
1/1 [==============================] - 0s 47ms/step
[[0.20976979]]
```

In [55]:
```
if yhat < 0.5:
    print("Predicted class is Bottles")
else:
    print("Predicted class is Cups")
```

```
Predicted class is Bottles
```

In [ ]:
```

```