In [1]:
```
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten

# Set the image size and channels
img_size = (256, 256)
channels = 3

# Define the image directories
f1 = 'C:\\Users\\Singh\\Desktop\\New folder (2)\\ML Assignment\\Images\\8x shampoo
f2 = 'C:\\Users\\Singh\\Desktop\\New folder (2)\\ML Assignment\\Images\\saslic'
f3 = 'C:\\Users\\Singh\\Desktop\\New folder (2)\\ML Assignment\\Images\\wow'

# Create lists for images and labels
images = []
labels = []
```

In [2]:
```
f1_count = 0
f2_count = 0
f3_count = 0
# Load the images and labels
for label, folder in enumerate([f1, f2, f3]):
    for imgname in os.listdir(folder):#stores all names of img in a list imgname
        if not imgname.endswith(('.jpg', '.png', '.jpeg','.webp','.avif','jfif')):
            continue
        image_path = os.path.join(folder, imgname) # joins path of folder with the
        try:
            img_array = cv2.imread(image_path)#to read all img in image_path
            #img_array stores array of image's pixels
            if img_array is None or img_array.size == 0:
                print(f"Error reading image: {image_path}")
                continue
            img = cv2.resize(img_array, img_size)#resizes the img_array to the dim
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)#from BGR to RGB becz matplo
            images.append(img)
            labels.append(label)
        except Exception as e:
            print(f"Error processing image: {image_path}: {e}")
            continue

        # Increment directory counter
        if folder == f1:
            f1_count += 1
        elif folder == f2:
            f2_count += 1
        elif folder == f3:
            f3_count += 1

print(f"Number of images in {f1}: {f1_count}")
print(f"Number of images in {f2}: {f2_count}")
print(f"Number of images in {f3}: {f3_count}")
```

```
Error reading image: C:\Users\Singh\Desktop\New folder (2)\ML Assignment\Images\wo
w\7.avif
Number of images in C:\Users\Singh\Desktop\New folder (2)\ML Assignment\Images\8x
shampoo: 8
Number of images in C:\Users\Singh\Desktop\New folder (2)\ML Assignment\Images\sas
lic: 13
Number of images in C:\Users\Singh\Desktop\New folder (2)\ML Assignment\Images\wo
w: 24
```

In [3]:
```
# Convert the images and labels to NumPy arrays
images = np.array(images)
labels = np.array(labels)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2,
```

In [4]:
```
# Normalizing  pixel values so it lies between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0
X_train
```

Out[4]:
```
array([[[[0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99607843, 0.99607843, 0.99607843],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99607843, 0.99607843, 0.99607843],
         ...,
         [0.99607843, 0.99607843, 0.99607843],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        ...,

        [[0.98431373, 0.99607843, 1.        ],
         [0.97254902, 1.        , 1.        ],
         [0.94901961, 0.99607843, 1.        ],
         ...,
         [0.94117647, 1.        , 0.98039216],
         [0.96862745, 1.        , 0.98431373],
         [0.99607843, 1.        , 0.99215686]],

        [[0.99215686, 0.99215686, 1.        ],
         [0.98039216, 1.        , 1.        ],
         [0.96470588, 1.        , 1.        ],
         ...,
         [0.97647059, 1.        , 0.99607843],
         [0.98431373, 1.        , 0.99607843],
         [0.99215686, 0.99607843, 0.99607843]],

        [[1.        , 0.98823529, 1.        ],
         [0.98039216, 0.99607843, 1.        ],
         [0.98039216, 1.        , 0.98823529],
         ...,
         [0.99215686, 0.98431373, 1.        ],
         [0.99607843, 0.98431373, 1.        ],
         [1.        , 0.99215686, 0.99607843]]],


       [[[0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         ...,
         [0.99215686, 1.        , 0.99215686],
         [0.99215686, 1.        , 0.99215686],
         [0.99215686, 1.        , 0.99215686]],

        [[0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.98823529, 0.99215686],
         ...,
         [0.99215686, 1.        , 0.99215686],
```

```
 [0.99215686, 1.        , 0.99215686],
 [0.99215686, 1.        , 0.99215686]],

[[0.99215686, 0.99215686, 0.99215686],
 [0.99215686, 0.98823529, 0.99215686],
 [0.98039216, 0.98039216, 0.98039216],
 ...,
 [0.98823529, 0.99607843, 0.99607843],
 [0.98823529, 1.        , 0.99607843],
 [0.98823529, 1.        , 0.99607843]],

...,

[[1.        , 1.        , 0.98823529],
 [1.        , 1.        , 0.98823529],
 [0.83529412, 0.84705882, 0.82352941],
 ...,
 [0.84705882, 0.86666667, 0.85098039],
 [0.99215686, 1.        , 1.        ],
 [0.99215686, 1.        , 1.        ]],

[[1.        , 1.        , 0.98823529],
 [1.        , 1.        , 0.98823529],
 [0.98039216, 0.98039216, 0.97647059],
 ...,
 [0.98431373, 0.99215686, 0.98823529],
 [0.99215686, 1.        , 1.        ],
 [0.99215686, 1.        , 1.        ]],

[[1.        , 1.        , 0.98823529],
 [1.        , 1.        , 0.98823529],
 [1.        , 1.        , 0.99607843],
 ...,
 [0.99215686, 1.        , 1.        ],
 [0.99215686, 1.        , 1.        ],
 [0.99215686, 1.        , 1.        ]]],


[[[0.98431373, 0.98431373, 0.98431373],
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843],
  [0.99215686, 0.99215686, 0.99215686],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 ...,
```

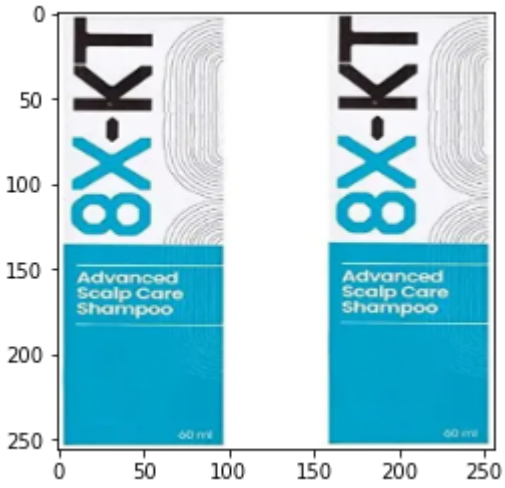
```
 [[1.        , 0.99215686, 1.        ],
  [1.        , 0.99215686, 1.        ],
  [1.        , 0.99215686, 1.        ],
  ...,
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686]],

 [[1.        , 0.99215686, 1.        ],
  [1.        , 0.99215686, 1.        ],
  [1.        , 0.99215686, 1.        ],
  ...,
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  [0.98823529, 0.98823529, 0.98823529]],

 [[1.        , 0.99215686, 1.        ],
  [1.        , 0.99215686, 1.        ],
  [1.        , 0.99215686, 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]]],


...,


[[[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [0.99215686, 1.        , 0.99215686],
  [0.99215686, 1.        , 0.99215686],
  [0.99215686, 1.        , 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [0.99215686, 1.        , 0.99215686],
  [0.99215686, 1.        , 0.99215686],
  [0.99215686, 1.        , 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [0.99215686, 1.        , 1.        ],
  [0.99215686, 1.        , 1.        ],
  [0.99215686, 1.        , 1.        ]],

 ...,

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 0.99607843, 0.99607843],
  ...,
  [0.99215686, 1.        , 0.99607843],
  [0.98823529, 1.        , 0.99215686],
  [0.98823529, 1.        , 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
```

```
  [1.        , 1.        , 0.99607843],
  ...,
  [1.        , 1.        , 0.99607843],
  [0.99215686, 1.        , 0.99215686],
  [0.98823529, 1.        , 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 0.99607843, 0.99607843],
  ...,
  [1.        , 1.        , 0.99607843],
  [0.99215686, 1.        , 0.99215686],
  [0.98823529, 1.        , 0.99215686]]],


[[[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.        , 0.99215686],
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 0.99215686]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 0.99215686]]],
```

```
[[[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.        ]],

 [[0.98823529, 0.98823529, 0.98823529],
  [0.98823529, 0.98823529, 0.98823529],
  [0.99607843, 0.99607843, 0.99607843],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 ...,

 [[0.99215686, 0.98039216, 0.99607843],
  [0.99607843, 0.99215686, 1.        ],
  [1.        , 0.99607843, 1.        ],
  ...,
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686],
  [0.99215686, 0.99215686, 0.99215686]],

 [[0.99215686, 0.98039216, 0.99607843],
  [0.99607843, 0.99215686, 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843]],

 [[0.99215686, 0.98039216, 0.99607843],
  [0.99607843, 0.99215686, 1.        ],
  [1.        , 0.99607843, 1.        ],
  ...,
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843],
  [0.99607843, 0.99607843, 0.99607843]]])
```

In [5]:
```
plt.imshow(X_train[0])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x248d142c820>
```



In [6]:
```
plt.imshow(X_test[0])
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x248d2a6bb50>
```

In [7]:
```
# Define the CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_size[0], img_size
#32 filters with 3,3 size
model.add(MaxPooling2D((2, 2)))
# 2,2 is filter size
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax')) #final layer softmax becz of multi-class
```

In [8]:
```
# Compile the model and loss is sparse_categorical_crossentropy becz our final resu
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a



# Train the model and store the history to plot the graph
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test),
```

```python
# Evaluate the model on the testing set
loss, accuracy = model.evaluate(X_test, y_test)
print(loss,accuracy)
print(f"Test loss: {loss:.4f}")
print(f"Test accuracy: {accuracy:.4f}")
```

```
Epoch 1/50
2/2 [==============================] - 3s 516ms/step - loss: 1.2294 - accuracy: 0.
4167 - val_loss: 0.9047 - val_accuracy: 0.3333
Epoch 2/50
2/2 [==============================] - 2s 350ms/step - loss: 1.0071 - accuracy: 0.
3611 - val_loss: 0.8959 - val_accuracy: 1.0000
Epoch 3/50
2/2 [==============================] - 2s 414ms/step - loss: 0.9341 - accuracy: 0.
7778 - val_loss: 0.6039 - val_accuracy: 0.6667
Epoch 4/50
2/2 [==============================] - 2s 406ms/step - loss: 0.7891 - accuracy: 0.
5278 - val_loss: 0.3206 - val_accuracy: 1.0000
Epoch 5/50
2/2 [==============================] - 2s 406ms/step - loss: 0.4391 - accuracy: 0.
9722 - val_loss: 0.1207 - val_accuracy: 1.0000
Epoch 6/50
2/2 [==============================] - 2s 375ms/step - loss: 0.4079 - accuracy: 0.
7500 - val_loss: 0.0351 - val_accuracy: 1.0000
Epoch 7/50
2/2 [==============================] - 2s 407ms/step - loss: 0.2733 - accuracy: 0.
8889 - val_loss: 0.0331 - val_accuracy: 1.0000
Epoch 8/50
2/2 [==============================] - 2s 368ms/step - loss: 0.1100 - accuracy: 0.
9722 - val_loss: 0.0681 - val_accuracy: 1.0000
Epoch 9/50
2/2 [==============================] - 2s 406ms/step - loss: 0.1289 - accuracy: 0.
9722 - val_loss: 0.1038 - val_accuracy: 1.0000
Epoch 10/50
2/2 [==============================] - 2s 391ms/step - loss: 0.1649 - accuracy: 0.
9444 - val_loss: 0.0623 - val_accuracy: 1.0000
Epoch 11/50
2/2 [==============================] - 2s 406ms/step - loss: 0.1126 - accuracy: 1.
0000 - val_loss: 0.0321 - val_accuracy: 1.0000
Epoch 12/50
2/2 [==============================] - 2s 359ms/step - loss: 0.0904 - accuracy: 0.
9722 - val_loss: 0.0096 - val_accuracy: 1.0000
Epoch 13/50
2/2 [==============================] - 2s 375ms/step - loss: 0.0657 - accuracy: 0.
9722 - val_loss: 0.0056 - val_accuracy: 1.0000
Epoch 14/50
2/2 [==============================] - 2s 335ms/step - loss: 0.0458 - accuracy: 0.
9722 - val_loss: 0.1681 - val_accuracy: 0.8889
Epoch 15/50
2/2 [==============================] - 2s 365ms/step - loss: 0.1125 - accuracy: 0.
9167 - val_loss: 5.0149e-05 - val_accuracy: 1.0000
Epoch 16/50
2/2 [==============================] - 2s 344ms/step - loss: 0.0132 - accuracy: 1.
0000 - val_loss: 6.9821e-05 - val_accuracy: 1.0000
Epoch 17/50
2/2 [==============================] - 2s 371ms/step - loss: 0.0039 - accuracy: 1.
0000 - val_loss: 7.1754e-05 - val_accuracy: 1.0000
Epoch 18/50
2/2 [==============================] - 2s 375ms/step - loss: 0.0054 - accuracy: 1.
0000 - val_loss: 7.3395e-05 - val_accuracy: 1.0000
Epoch 19/50
2/2 [==============================] - 2s 375ms/step - loss: 0.0118 - accuracy: 1.
0000 - val_loss: 1.1429e-04 - val_accuracy: 1.0000
Epoch 20/50
2/2 [==============================] - 2s 391ms/step - loss: 0.0059 - accuracy: 1.
0000 - val_loss: 1.6140e-04 - val_accuracy: 1.0000
1/1 [==============================] - 0s 109ms/step - loss: 1.6140e-04 - accurac
y: 1.0000
0.0001613981439732015 1.0
```

```
Test loss: 0.0002
Test accuracy: 1.0000
```

In [9]:
```
yprob=model.predict(X_test)
```

```
1/1 [==============================] - 0s 154ms/step
```

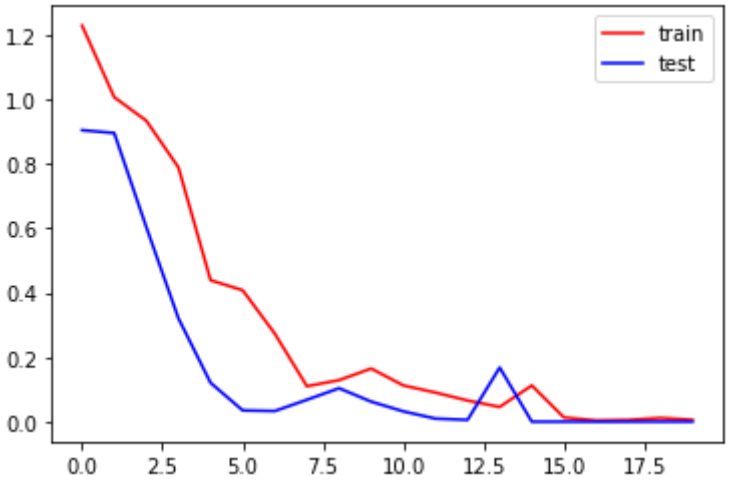
In [10]:
```
ypred=yprob.argmax(axis=1)
ypred
```

Out[10]:
```
array([2, 2, 2, 2, 2, 2, 0, 1, 1], dtype=int64)
```

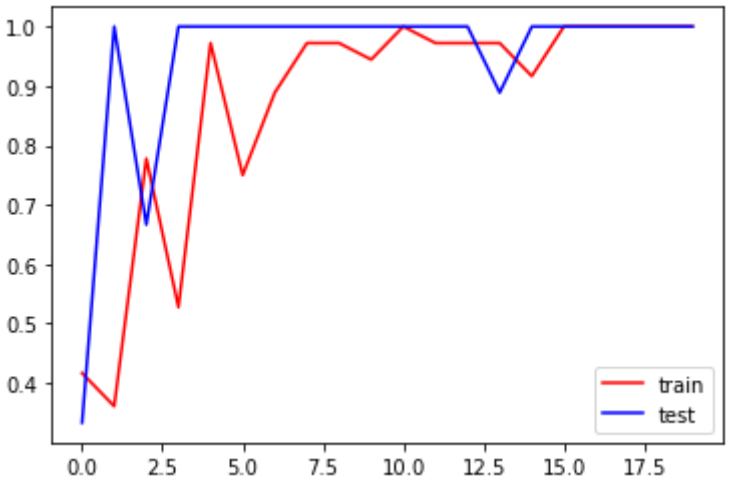
In [11]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         6

    accuracy                           1.00         9
   macro avg       1.00      1.00      1.00         9
weighted avg       1.00      1.00      1.00         9
```

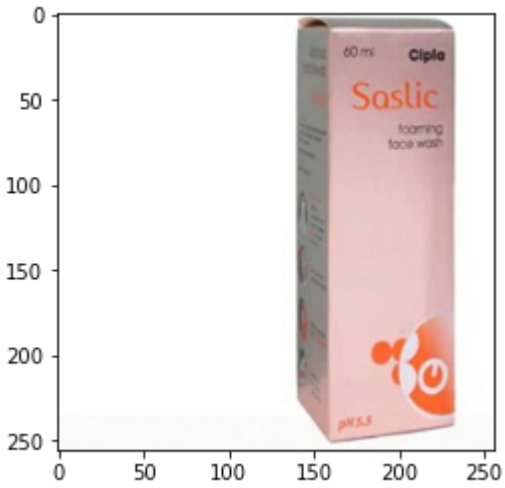
In [12]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='test')
plt.legend()
plt.show()
#model gives good result for both training and test data
```



In [13]:
```
# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], color='red', label='train')
plt.plot(history.history['val_accuracy'], color='blue', label='test')
plt.legend()
plt.show()
# as epochs increase our accuracy increases and there is no gap between 2 lines so
```



In [14]:
```python
import random

idx = random.randint(0, len(y_test) - 1)
plt.imshow(X_test[idx,:])
plt.show()
y_pred = model.predict(X_test[idx,:].reshape(1,256,256,3))
print(y_pred )
print()
print('The Product is :')
threshold = 0.85 # set a threshold value for probability
if y_pred[0][0] >threshold:
    print('*'*5,'8x shampoo','*'*5)
elif y_pred[0][1] > threshold:
    print('*'*5,'saslic','*'*5)
elif y_pred[0][2] > threshold:
    print('*'*5,'wow','*'*5)

#TThis code classifies an input image using a trained model.
#It prints the label with the highest predicted probability, as long as the probabi
#The code demonstrates a simple way to classify images based on predicted probabili
```




```
1/1 [==============================] - 0s 16ms/step
[[1.3820357e-03 9.9854892e-01 6.9006892e-05]]

The Product is :
***** saslic *****
```

In [ ]:

In [ ]: