Import Statements

In [4]:
```
import os
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNorma
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('dark_background')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

In [2]:
```
!pip install keras
```

```
Requirement already satisfied: keras in c:\users\arivu\appdata\local\programs\pyth
on\python310\lib\site-packages (2.10.0)
[notice] A new release of pip available: 22.3 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

One Hot Encoding the Target Classes

In [5]:
```
encoder = OneHotEncoder()
encoder.fit([[0], [1]])

# 0 - cancer
# 1 - normal
```

Out[5]:
```
OneHotEncoder()
```

Creating 3 Important Lists --

1. data list for storing image data in numpy array form
2. paths list for storing paths of all images
3. result list for storing one hot encoded form of target class whether normal or cancer

In [6]:
```
data = []
paths = []
result = []

for r, d, f in os.walk(r"C:\Users\arivu\OneDrive\Dataset_vani"):
    for file in f:
        if '.bmp' in file:
            paths.append(os.path.join(r, file))

for path in paths:
    img = Image.open(path)
    img = img.resize((128,128))
    img = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[0]]).toarray())
```

In [8]:
```
paths = []
for r, d, f in os.walk(r"C:\Users\arivu\OneDrive\Dataset_vani"):
    for file in f:
        if '.bmp' in file:
            paths.append(os.path.join(r, file))

for path in paths:
    img = Image.open(path)
    img = img.resize((128,128))
    img = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[1]]).toarray())
```

In [9]:
```
data = np.array(data)
data.shape
```

Out[9]:
```
(1365, 128, 128, 3)
```

In [11]:
```
result = np.array(result)
result = result.reshape(1365,2)
```

Splitting the Data into Training & Testing

In [12]:
```
x_train,x_test,y_train,y_test = train_test_split(data, result, test_size=0.2, shuf
```

In [13]:
```
# Normalization
x_train = x_train/255.0
x_test = x_test/255.0
```

Model Building

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

In [14]:
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), input_shape=(128, 128, 3), padding = 'Same
model.add(Conv2D(32, kernel_size=(2, 2),  activation ='relu', padding = 'Same'))


model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size = (2,2), activation ='relu', padding = 'Same'))
model.add(Conv2D(64, kernel_size = (2,2), activation ='relu', padding = 'Same'))

model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
```

```
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

model.compile(loss = "categorical_crossentropy", optimizer='Adamax')
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      416

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      4128

 batch_normalization (BatchN  (None, 128, 128, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0
 )

 dropout (Dropout)           (None, 64, 64, 32)        0

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        8256

 conv2d_3 (Conv2D)           (None, 64, 64, 64)        16448

 batch_normalization_1 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 32, 32, 64)        0

 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 512)               33554944

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 33,585,602
Trainable params: 33,585,410
Non-trainable params: 192
_________________________________________________________________
None
```

In [23]:
```
y_train.shape
```

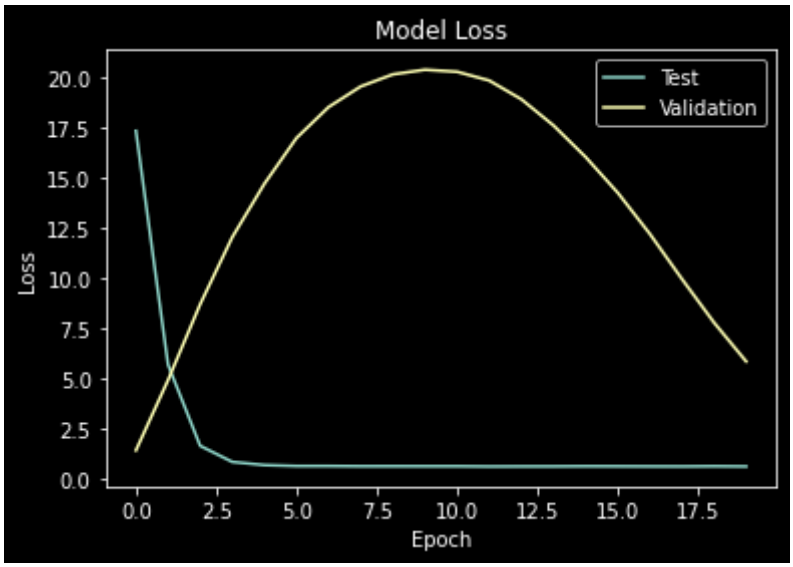
Out[23]:
```
(4004, 2)
```

In [15]:
```
history = model.fit(x_train, y_train, epochs = 20, batch_size = 40, verbose = 1,va
```

```
Epoch 1/20
28/28 [==============================] - 51s 2s/step - loss: 17.3473 - val_loss:
1.4399
Epoch 2/20
28/28 [==============================] - 49s 2s/step - loss: 5.7070 - val_loss: 4.
9389
Epoch 3/20
28/28 [==============================] - 52s 2s/step - loss: 1.6675 - val_loss: 8.
7149
Epoch 4/20
28/28 [==============================] - 48s 2s/step - loss: 0.8654 - val_loss: 1
2.0532
Epoch 5/20
28/28 [==============================] - 43s 2s/step - loss: 0.7079 - val_loss: 1
4.7119
Epoch 6/20
28/28 [==============================] - 39s 1s/step - loss: 0.6610 - val_loss: 1
6.9932
Epoch 7/20
28/28 [==============================] - 40s 1s/step - loss: 0.6606 - val_loss: 1
8.5327
Epoch 8/20
28/28 [==============================] - 39s 1s/step - loss: 0.6500 - val_loss: 1
9.5581
Epoch 9/20
28/28 [==============================] - 40s 1s/step - loss: 0.6508 - val_loss: 2
0.1596
Epoch 10/20
28/28 [==============================] - 40s 1s/step - loss: 0.6494 - val_loss: 2
0.3900
Epoch 11/20
28/28 [==============================] - 40s 1s/step - loss: 0.6496 - val_loss: 2
0.2893
Epoch 12/20
28/28 [==============================] - 48s 2s/step - loss: 0.6380 - val_loss: 1
9.8524
Epoch 13/20
28/28 [==============================] - 41s 1s/step - loss: 0.6417 - val_loss: 1
8.9206
Epoch 14/20
28/28 [==============================] - 40s 1s/step - loss: 0.6428 - val_loss: 1
7.6169
Epoch 15/20
28/28 [==============================] - 40s 1s/step - loss: 0.6498 - val_loss: 1
6.0451
Epoch 16/20
28/28 [==============================] - 42s 1s/step - loss: 0.6469 - val_loss: 1
4.2701
Epoch 17/20
28/28 [==============================] - 40s 1s/step - loss: 0.6431 - val_loss: 1
2.2198
Epoch 18/20
28/28 [==============================] - 40s 1s/step - loss: 0.6405 - val_loss: 9.
9718
Epoch 19/20
28/28 [==============================] - 41s 1s/step - loss: 0.6498 - val_loss: 7.
8009
Epoch 20/20
28/28 [==============================] - 40s 1s/step - loss: 0.6396 - val_loss: 5.
8662
```

Plotting Losses

In [16]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Test', 'Validation'], loc='upper right')
plt.show()
```

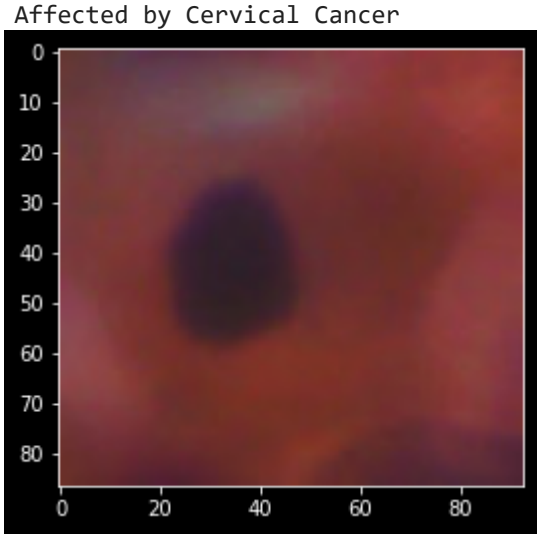


Just Checking the Model

In [20]:
```
def names(number):
    if number==0:
        return 'Not Affected by Cervical Cancer'
    else:
        return 'Affected by Cervical Cancer'
```

In [21]:
```
from matplotlib.pyplot import imshow
img = Image.open(r"C:\Users\arivu\OneDrive\003_03.jpg")
x = np.array(img.resize((128,128)))
x = x.reshape(1,128,128,3)
res = model.predict_on_batch(x
                             )
classification = np.where(res == np.amax(res))[1][0]
imshow(img)
print( names(classification))
```

```
Affected by Cervical Cancer
```



In [ ]: