Creating CNN Using Scratch And Transfer Learning

In [1]:
```
# import the libraries as shown below

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
```

In [2]:
```
# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'cell_images/Train'
valid_path = 'cell_images/Test'
```

In [36]:
```
# Import the Vgg 16 library as shown below and add preprocessing layer to the
 front of VGG
# Here we will be using imagenet weights

mobilnet = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top
=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applica
tions/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 204s 3us/step
```

In [37]:
```
# don't train existing weights
for layer in mobilnet.layers:
    layer.trainable = False
```

In [38]:
```
  # useful for getting number of output classes
folders = glob('Dataset/Train/*')
```

In [39]:
```
folders
```

Out[39]:
```
['Dataset/Train\\Parasite', 'Dataset/Train\\Uninfected']
```

In [40]:
```
# our layers - you can add more if you want
x = Flatten()(mobilnet.output)
```

In [42]:
```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=mobilnet.input, outputs=prediction)
```

In [43]:
```
# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 50178
=================================================================
Total params: 20,074,562
Trainable params: 50,178
Non-trainable params: 20,024,384
_________________________________________________________________
```

In [44]:
```
from tensorflow.keras.layers import MaxPooling2D
```

In [45]:
```
### Create Model from scratch using CNN
#model=Sequential()
#model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",in
put_shape=(224,224,3)))
#model.add(MaxPooling2D(pool_size=2))
#model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation ="relu"))
#model.add(MaxPooling2D(pool_size=2))
#model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
#model.add(MaxPooling2D(pool_size=2))
#model.add(Flatten())
#model.add(Dense(500,activation="relu"))
#model.add(Dense(2,activation="softmax"))
#model.summary()
```

In [46]:
```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

In [47]:
```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [48]:
```
# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('Dataset/Train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
Found 416 images belonging to 2 classes.
```

In [49]:
```
training_set
```

Out[49]:
```
<keras_preprocessing.image.directory_iterator.DirectoryIterator at 0x18300512
688>
```

In [50]:
```
test_set = test_datagen.flow_from_directory('Dataset/Test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 134 images belonging to 2 classes.
```

In [51]:
```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=50,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
Epoch 1/50
13/13 [==============================] - 96s 7s/step - loss: 1.4985 - accurac
y: 0.5505 - val_loss: 0.6400 - val_accuracy: 0.6791
Epoch 2/50
13/13 [==============================] - 96s 7s/step - loss: 0.5195 - accurac
y: 0.7740 - val_loss: 0.5961 - val_accuracy: 0.6866
Epoch 3/50
13/13 [==============================] - 97s 7s/step - loss: 0.4107 - accurac
y: 0.8125 - val_loss: 0.4516 - val_accuracy: 0.7761
Epoch 4/50
13/13 [==============================] - 98s 8s/step - loss: 0.4039 - accurac
y: 0.8029 - val_loss: 0.6201 - val_accuracy: 0.6642
Epoch 5/50
13/13 [==============================] - 99s 8s/step - loss: 0.3601 - accurac
y: 0.8413 - val_loss: 0.4251 - val_accuracy: 0.7687
Epoch 6/50
13/13 [==============================] - 98s 8s/step - loss: 0.2975 - accurac
y: 0.8822 - val_loss: 0.4234 - val_accuracy: 0.7985
Epoch 7/50
13/13 [==============================] - 99s 8s/step - loss: 0.3177 - accurac
y: 0.8558 - val_loss: 0.3725 - val_accuracy: 0.8657
Epoch 8/50
13/13 [==============================] - 98s 8s/step - loss: 0.3577 - accurac
y: 0.8341 - val_loss: 0.3226 - val_accuracy: 0.8881
Epoch 9/50
13/13 [==============================] - 99s 8s/step - loss: 0.2688 - accurac
y: 0.8798 - val_loss: 0.3195 - val_accuracy: 0.8881
Epoch 10/50
13/13 [==============================] - 97s 7s/step - loss: 0.2251 - accurac
y: 0.9183 - val_loss: 0.3539 - val_accuracy: 0.8358
Epoch 11/50
13/13 [==============================] - 98s 8s/step - loss: 0.2025 - accurac
y: 0.9279 - val_loss: 0.3037 - val_accuracy: 0.9104
Epoch 12/50
13/13 [==============================] - 98s 8s/step - loss: 0.1867 - accurac
y: 0.9327 - val_loss: 0.2989 - val_accuracy: 0.9030
Epoch 13/50
13/13 [==============================] - 100s 8s/step - loss: 0.1886 - accura
cy: 0.9231 - val_loss: 0.2731 - val_accuracy: 0.9104
Epoch 14/50
13/13 [==============================] - 98s 8s/step - loss: 0.1784 - accurac
y: 0.9543 - val_loss: 0.2872 - val_accuracy: 0.9179
Epoch 15/50
13/13 [==============================] - 98s 8s/step - loss: 0.1897 - accurac
y: 0.9375 - val_loss: 0.3853 - val_accuracy: 0.7687
Epoch 16/50
13/13 [==============================] - 98s 8s/step - loss: 0.1630 - accurac
y: 0.9471 - val_loss: 0.2720 - val_accuracy: 0.9254
Epoch 17/50
13/13 [==============================] - 98s 8s/step - loss: 0.1465 - accurac
y: 0.9591 - val_loss: 0.2874 - val_accuracy: 0.8955
Epoch 18/50
13/13 [==============================] - 99s 8s/step - loss: 0.1366 - accurac
y: 0.9519 - val_loss: 0.2665 - val_accuracy: 0.9030
Epoch 19/50
13/13 [==============================] - 98s 8s/step - loss: 0.1637 - accurac
y: 0.9471 - val_loss: 0.6027 - val_accuracy: 0.7164
```

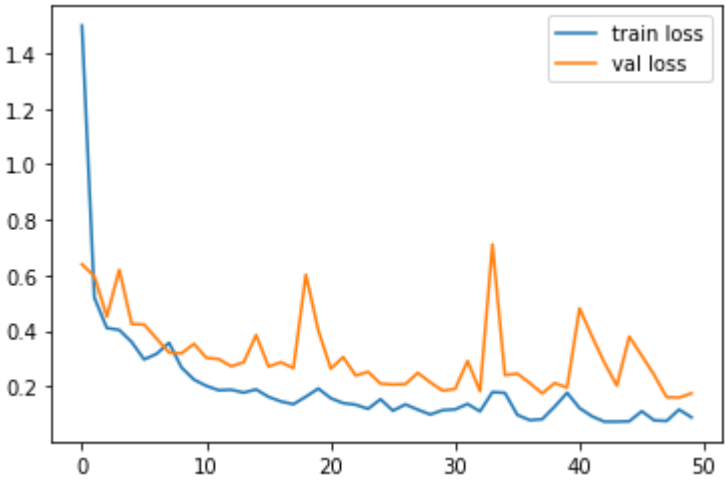
```
Epoch 20/50
13/13 [==============================] - 98s 8s/step - loss: 0.1929 - accurac
y: 0.9207 - val_loss: 0.4020 - val_accuracy: 0.7463
Epoch 21/50
13/13 [==============================] - 99s 8s/step - loss: 0.1583 - accurac
y: 0.9399 - val_loss: 0.2646 - val_accuracy: 0.9254
Epoch 22/50
13/13 [==============================] - 99s 8s/step - loss: 0.1405 - accurac
y: 0.9639 - val_loss: 0.3060 - val_accuracy: 0.8582
Epoch 23/50
13/13 [==============================] - 99s 8s/step - loss: 0.1348 - accurac
y: 0.9543 - val_loss: 0.2398 - val_accuracy: 0.9179
Epoch 24/50
13/13 [==============================] - 100s 8s/step - loss: 0.1199 - accura
cy: 0.9663 - val_loss: 0.2531 - val_accuracy: 0.9104
Epoch 25/50
13/13 [==============================] - 99s 8s/step - loss: 0.1545 - accurac
y: 0.9375 - val_loss: 0.2105 - val_accuracy: 0.9328
Epoch 26/50
13/13 [==============================] - 99s 8s/step - loss: 0.1132 - accurac
y: 0.9712 - val_loss: 0.2074 - val_accuracy: 0.9403
Epoch 27/50
13/13 [==============================] - 99s 8s/step - loss: 0.1359 - accurac
y: 0.9519 - val_loss: 0.2089 - val_accuracy: 0.9104
Epoch 28/50
13/13 [==============================] - 99s 8s/step - loss: 0.1170 - accurac
y: 0.9760 - val_loss: 0.2495 - val_accuracy: 0.8955
Epoch 29/50
13/13 [==============================] - 100s 8s/step - loss: 0.0996 - accura
cy: 0.9688 - val_loss: 0.2152 - val_accuracy: 0.9030
Epoch 30/50
13/13 [==============================] - 99s 8s/step - loss: 0.1152 - accurac
y: 0.9688 - val_loss: 0.1854 - val_accuracy: 0.9254
Epoch 31/50
13/13 [==============================] - 99s 8s/step - loss: 0.1180 - accurac
y: 0.9591 - val_loss: 0.1914 - val_accuracy: 0.9478
Epoch 32/50
13/13 [==============================] - 100s 8s/step - loss: 0.1376 - accura
cy: 0.9423 - val_loss: 0.2922 - val_accuracy: 0.8806
Epoch 33/50
13/13 [==============================] - 101s 8s/step - loss: 0.1108 - accura
cy: 0.9663 - val_loss: 0.1832 - val_accuracy: 0.9478
Epoch 34/50
13/13 [==============================] - 100s 8s/step - loss: 0.1803 - accura
cy: 0.9255 - val_loss: 0.7119 - val_accuracy: 0.7164
Epoch 35/50
13/13 [==============================] - 100s 8s/step - loss: 0.1774 - accura
cy: 0.9231 - val_loss: 0.2411 - val_accuracy: 0.9030
Epoch 36/50
13/13 [==============================] - 100s 8s/step - loss: 0.0983 - accura
cy: 0.9712 - val_loss: 0.2467 - val_accuracy: 0.8955
Epoch 37/50
13/13 [==============================] - 101s 8s/step - loss: 0.0792 - accura
cy: 0.9808 - val_loss: 0.2129 - val_accuracy: 0.9030
Epoch 38/50
13/13 [==============================] - 100s 8s/step - loss: 0.0827 - accura
cy: 0.9784 - val_loss: 0.1754 - val_accuracy: 0.9478
```

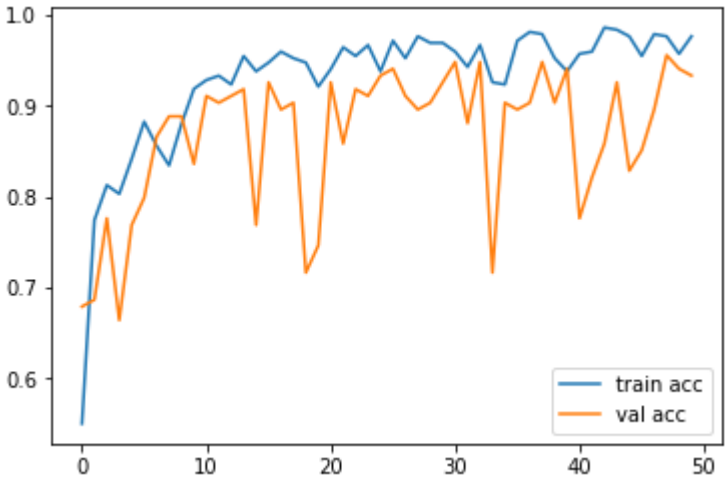
```
Epoch 39/50
13/13 [==============================] - 100s 8s/step - loss: 0.1279 - accura
cy: 0.9519 - val_loss: 0.2127 - val_accuracy: 0.9030
Epoch 40/50
13/13 [==============================] - 100s 8s/step - loss: 0.1778 - accura
cy: 0.9375 - val_loss: 0.1963 - val_accuracy: 0.9403
Epoch 41/50
13/13 [==============================] - 101s 8s/step - loss: 0.1226 - accura
cy: 0.9567 - val_loss: 0.4813 - val_accuracy: 0.7761
Epoch 42/50
13/13 [==============================] - 102s 8s/step - loss: 0.0933 - accura
cy: 0.9591 - val_loss: 0.3809 - val_accuracy: 0.8209
Epoch 43/50
13/13 [==============================] - 101s 8s/step - loss: 0.0738 - accura
cy: 0.9856 - val_loss: 0.2858 - val_accuracy: 0.8582
Epoch 44/50
13/13 [==============================] - 101s 8s/step - loss: 0.0739 - accura
cy: 0.9832 - val_loss: 0.2034 - val_accuracy: 0.9254
Epoch 45/50
13/13 [==============================] - 100s 8s/step - loss: 0.0752 - accura
cy: 0.9760 - val_loss: 0.3799 - val_accuracy: 0.8284
Epoch 46/50
13/13 [==============================] - 102s 8s/step - loss: 0.1114 - accura
cy: 0.9543 - val_loss: 0.3114 - val_accuracy: 0.8507
Epoch 47/50
13/13 [==============================] - 99s 8s/step - loss: 0.0785 - accurac
y: 0.9784 - val_loss: 0.2418 - val_accuracy: 0.8955
Epoch 48/50
13/13 [==============================] - 100s 8s/step - loss: 0.0765 - accura
cy: 0.9760 - val_loss: 0.1621 - val_accuracy: 0.9552
Epoch 49/50
13/13 [==============================] - 99s 8s/step - loss: 0.1173 - accurac
y: 0.9567 - val_loss: 0.1603 - val_accuracy: 0.9403
Epoch 50/50
13/13 [==============================] - 99s 8s/step - loss: 0.0894 - accurac
y: 0.9760 - val_loss: 0.1759 - val_accuracy: 0.9328
```

In [ ]:

In [52]:
```
# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```






```
<Figure size 432x288 with 0 Axes>
```

In [53]:
```
# save it as a h5 file


from tensorflow.keras.models import load_model

model.save('model_vgg19.h5')
```

In [ ]:
```

```

In [54]:
```
y_pred = model.predict(test_set)
```

In [55]:
```
y_pred
```

Out[55]:
```
array([[7.33568192e-01, 2.66431868e-01],
       [9.99980807e-01, 1.92336047e-05],
       [8.01913381e-01, 1.98086604e-01],
       [1.00000000e+00, 1.31814044e-08],
       [8.65602374e-01, 1.34397611e-01],
       [1.04434900e-02, 9.89556491e-01],
       [2.77041905e-02, 9.72295761e-01],
       [1.16299026e-01, 8.83701026e-01],
       [5.37910238e-02, 9.46208954e-01],
       [2.93028634e-02, 9.70697105e-01],
       [9.99999285e-01, 6.78014885e-07],
       [9.99504447e-01, 4.95534972e-04],
       [1.99335113e-01, 8.00664902e-01],
       [9.99957442e-01, 4.25866310e-05],
       [9.99995708e-01, 4.34720141e-06],
       [9.98122752e-01, 1.87728333e-03],
       [9.99790609e-01, 2.09422928e-04],
       [2.82993298e-02, 9.71700609e-01],
       [1.44461364e-01, 8.55538666e-01],
       [2.19181478e-02, 9.78081822e-01],
       [9.81848955e-01, 1.81510355e-02],
       [9.99973655e-01, 2.63286001e-05],
       [9.99836445e-01, 1.63586170e-04],
       [8.15682530e-01, 1.84317455e-01],
       [1.79526001e-01, 8.20473969e-01],
       [4.26607013e-01, 5.73393047e-01],
       [9.99854922e-01, 1.45101134e-04],
       [9.99862194e-01, 1.37764408e-04],
       [2.26018913e-02, 9.77398098e-01],
       [8.54977012e-01, 1.45022959e-01],
       [9.99835610e-01, 1.64366837e-04],
       [1.00000000e+00, 5.56785418e-09],
       [5.62469661e-01, 4.37530398e-01],
       [1.34621561e-02, 9.86537814e-01],
       [9.80807245e-01, 1.91927906e-02],
       [6.69995546e-02, 9.33000445e-01],
       [9.99995351e-01, 4.68705957e-06],
       [1.02938088e-02, 9.89706159e-01],
       [4.82946366e-01, 5.17053604e-01],
       [9.89062250e-01, 1.09377559e-02],
       [9.99584496e-01, 4.15479502e-04],
       [7.60960579e-02, 9.23903942e-01],
       [9.75712121e-01, 2.42878925e-02],
       [9.99994755e-01, 5.23468771e-06],
       [9.99727428e-01, 2.72591918e-04],
       [9.97438550e-01, 2.56145676e-03],
       [6.58888638e-01, 3.41111332e-01],
       [6.72240779e-02, 9.32775974e-01],
       [8.12208176e-01, 1.87791809e-01],
       [1.13631934e-01, 8.86368096e-01],
       [3.51420522e-01, 6.48579478e-01],
       [5.22698089e-02, 9.47730184e-01],
       [9.98033464e-01, 1.96657726e-03],
       [9.99961019e-01, 3.89583693e-05],
       [1.37733385e-01, 8.62266600e-01],
       [3.01815663e-02, 9.69818354e-01],
       [9.99721229e-01, 2.78760330e-04],
```

```
[9.99998689e-01, 1.34526908e-06],
[9.51245964e-01, 4.87540960e-02],
[9.11167204e-01, 8.88328403e-02],
[9.41605926e-01, 5.83940633e-02],
[9.69726026e-01, 3.02740131e-02],
[4.46416214e-02, 9.55358326e-01],
[4.65420708e-02, 9.53457952e-01],
[4.80751060e-02, 9.51924920e-01],
[9.96749878e-01, 3.25014745e-03],
[9.99680638e-01, 3.19386541e-04],
[4.96625096e-01, 5.03374934e-01],
[7.20607400e-01, 2.79392660e-01],
[9.64657664e-01, 3.53423730e-02],
[3.86598147e-02, 9.61340129e-01],
[7.25330263e-02, 9.27466989e-01],
[2.56446630e-01, 7.43553400e-01],
[8.46040919e-02, 9.15395975e-01],
[6.21868186e-02, 9.37813222e-01],
[7.79348314e-02, 9.22065198e-01],
[2.49052513e-02, 9.75094676e-01],
[9.99147534e-01, 8.52417143e-04],
[3.09107453e-02, 9.69089270e-01],
[3.77726299e-03, 9.96222734e-01],
[9.90637481e-01, 9.36260261e-03],
[2.67133638e-02, 9.73286688e-01],
[9.92535949e-01, 7.46406941e-03],
[9.99937415e-01, 6.25946559e-05],
[6.52000666e-01, 3.47999364e-01],
[7.74592161e-01, 2.25407809e-01],
[9.99927998e-01, 7.19879099e-05],
[6.61823452e-01, 3.38176608e-01],
[9.55748677e-01, 4.42513563e-02],
[3.15008223e-01, 6.84991777e-01],
[9.93485034e-01, 6.51501724e-03],
[9.99985337e-01, 1.46407219e-05],
[9.99700785e-01, 2.99298030e-04],
[1.33023798e-01, 8.66976261e-01],
[9.96966064e-01, 3.03391251e-03],
[8.44473004e-01, 1.55526996e-01],
[7.85597324e-01, 2.14402705e-01],
[9.04192328e-01, 9.58076790e-02],
[9.99096274e-01, 9.03777429e-04],
[9.99068439e-01, 9.31616058e-04],
[5.64740658e-01, 4.35259372e-01],
[9.99316454e-01, 6.83533610e-04],
[3.90607625e-01, 6.09392405e-01],
[9.78302777e-01, 2.16971543e-02],
[9.99998569e-01, 1.47930052e-06],
[9.99913931e-01, 8.60950167e-05],
[7.36469626e-02, 9.26352978e-01],
[9.99930739e-01, 6.92849644e-05],
[9.99940872e-01, 5.91624339e-05],
[7.92534769e-01, 2.07465261e-01],
[8.48820686e-01, 1.51179254e-01],
[2.90788934e-02, 9.70921099e-01],
[9.89815593e-01, 1.01844380e-02],
[2.15925537e-02, 9.78407443e-01],
```

```
[2.66673174e-02, 9.73332644e-01],
[9.99646425e-01, 3.53570882e-04],
[1.24900743e-01, 8.75099301e-01],
[9.94786859e-01, 5.21314982e-03],
[9.13027167e-01, 8.69727731e-02],
[9.52461421e-01, 4.75385636e-02],
[8.48463356e-01, 1.51536673e-01],
[9.33516979e-01, 6.64829910e-02],
[9.26610827e-01, 7.33892098e-02],
[1.81540042e-01, 8.18459928e-01],
[9.37498827e-03, 9.90625024e-01],
[9.53926682e-01, 4.60733287e-02],
[1.42495140e-01, 8.57504845e-01],
[7.08026171e-01, 2.91973799e-01],
[7.95964450e-02, 9.20403600e-01],
[9.72831368e-01, 2.71686390e-02],
[9.97366607e-01, 2.63345707e-03],
[9.98226702e-01, 1.77325646e-03],
[1.78368136e-01, 8.21631849e-01],
[8.14394414e-01, 1.85605541e-01]], dtype=float32)
```

In [56]:
```
import numpy as np
y_pred = np.argmax(y_pred, axis=1)
```

In [57]:
```
y_pred
```

Out[57]:
```
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0], dtype=int64)
```

In [ ]:
```
```

In [58]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

In [59]:
```
model=load_model('model_vgg19.h5')
```

In [ ]:
```
```

In [60]:
```
img=image.load_img('Dataset/Test/Uninfected/2.png',target_size=(224,224))
```

In [61]:
```
x=image.img_to_array(img)
x
```

Out[61]:
```
array([[[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., 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.]]], dtype=float32)
```

In [62]:
```
x.shape
```

Out[62]:
```
(224, 224, 3)
```

In [63]:
```
x=x/255
```

In [64]:
```
x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
img_data.shape
```

Out[64]:
```
(1, 224, 224, 3)
```

In [65]:
```
model.predict(img_data)
```

Out[65]:
```
array([[4.4797333e-05, 9.9995518e-01]], dtype=float32)
```

In [66]:
```
a=np.argmax(model.predict(img_data), axis=1)
```

In [67]:
```
if(a==1):
    print("Uninfected")
else:
    print("Infected")
```

```
Uninfected
```

In [ ]:

In [ ]: