Peer Review Final Assignment

Introduction

In this lab, you will build an image classifier using the VGG16 pre-trained model, and you will evaluate it and compare its performance to the model we built in the last module using the ResNet50 pre-trained model. Good luck!

Download Data

Use the `wget` command to download the data for this assignment from here: https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/concrete_data_week4.zip (https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/concrete_data_week4.zip)

Use the following cells to download the data.

In [ ]:
```
#!wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/concrete_data_week4.zip
```

In [ ]:
```
#!zip concrete_data_week4.zip
```

After you unzip the data, you fill find the data has already been divided into a train, validation, and test sets.

Part 1

In this part, you will design a classifier using the VGG16 pre-trained model. Just like the ResNet50 model, you can import the model `VGG16` from `keras.applications`.

You will essentially build your classifier as follows:

1. Import libraries, modules, and packages you will need. Make sure to import the *preprocess_input* function from `keras.applications.vgg16`.
2. Use a batch size of 100 images for both training and validation.
3. Construct an ImageDataGenerator for the training set and another one for the validation set. VGG16 was originally trained on 224×224 images, so make sure to address that when defining the ImageDataGenerator instances.
4. Create a sequential model using Keras. Add VGG16 model to it and dense layer.
5. Compile the mode using the adam optimizer and the categorical_crossentropy loss function.
6. Fit the model on the augmented data using the ImageDataGenerators.

Use the following cells to create your classifier.

In [1]:
```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
```

In [2]:
```
train_dir = r'C:\Users\Dennis\Desktop\AI Capstone Project with Deep Learning\con
crete_data_week4\train'
validation_dir = r'C:\Users\Dennis\Desktop\AI Capstone Project with Deep Learnin
g\concrete_data_week4\valid'
test_dir =r'C:\Users\Dennis\Desktop\AI Capstone Project with Deep Learning\concr
ete_data_week4\test'
```

In [3]:
```
# Generating batches of tensor image data
train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(224, 224),
        batch_size=100,
        class_mode='categorical')

validation_generator = valid_datagen.flow_from_directory(
        validation_dir,
        target_size=(224, 224),
        batch_size=100,
        class_mode='categorical')
```

```
Found 30000 images belonging to 2 classes.
Found 9500 images belonging to 2 classes.
```

In [4]:
```
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(224, 224, 3))
```

In [5]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (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_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_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_pool (MaxPooling2D)   (None, 7, 7, 512)         0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [6]:
```
conv_base.trainable = False
```

In [7]:
```
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(2,activation='softmax'))
```

In [8]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
vgg16 (Model)               (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)           (None, 25088)             0
_________________________________________________________________
dense (Dense)               (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [9]:
```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

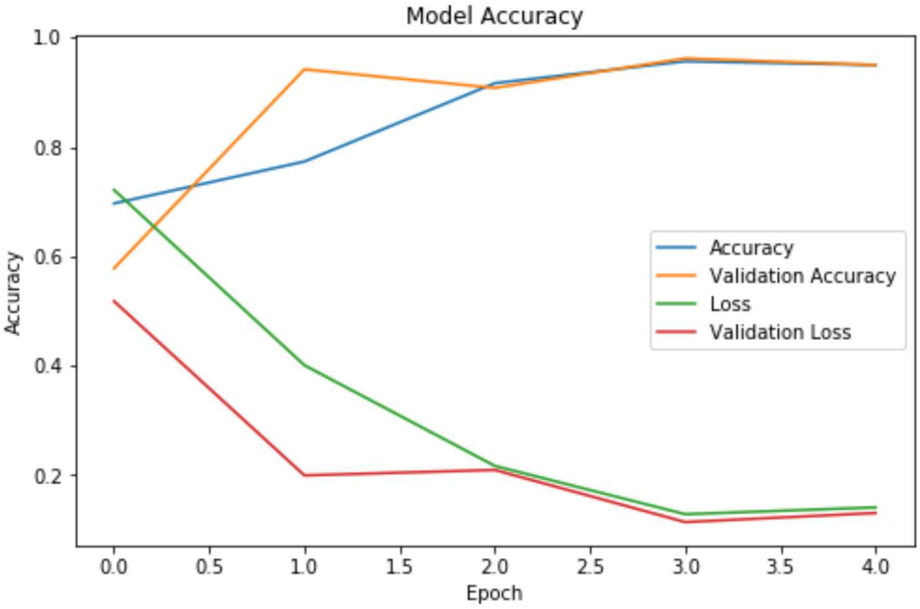
In [10]:
```
checkpoint = ModelCheckpoint("vgg16.h5",monitor='val_loss',verbose=1,save_best_only=True)
```

In [11]:
```
early = EarlyStopping(monitor='val_loss',min_delta=0,patience=3,verbose=1)
```

In [12]:
```
history = model.fit_generator(generator=train_generator,steps_per_epoch=3,epochs
=5,verbose=1,
                             validation_data=validation_generator,validation_ste
ps=5,callbacks=[checkpoint,early])
```

```
Epoch 1/5
2/3 [===================>.........] - ETA: 2:51 - loss: 0.9101 - accuracy: 0.
5800
Epoch 00001: val_loss improved from inf to 0.51783, saving model to vgg16.h5
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
3/3 [==============================] - 767s 256s/step - loss: 0.7215 - accurac
y: 0.6967 - val_loss: 0.5178 - val_accuracy: 0.5780
Epoch 2/5
2/3 [===================>.........] - ETA: 2:51 - loss: 0.4982 - accuracy: 0.
6800
Epoch 00002: val_loss improved from 0.51783 to 0.19915, saving model to vgg16.
h5
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
3/3 [==============================] - 783s 261s/step - loss: 0.4007 - accurac
y: 0.7733 - val_loss: 0.1991 - val_accuracy: 0.9420
Epoch 3/5
2/3 [===================>.........] - ETA: 2:57 - loss: 0.2160 - accuracy: 0.
9300
Epoch 00003: val_loss did not improve from 0.19915
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
3/3 [==============================] - 755s 252s/step - loss: 0.2165 - accurac
y: 0.9167 - val_loss: 0.2091 - val_accuracy: 0.9080
Epoch 4/5
2/3 [===================>.........] - ETA: 2:35 - loss: 0.1507 - accuracy: 0.
9400
Epoch 00004: val_loss improved from 0.19915 to 0.11376, saving model to vgg16.
h5
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
3/3 [==============================] - 704s 235s/step - loss: 0.1282 - accurac
y: 0.9567 - val_loss: 0.1138 - val_accuracy: 0.9620
Epoch 5/5
2/3 [===================>.........] - ETA: 2:25 - loss: 0.0886 - accuracy: 0.
9650
Epoch 00005: val_loss did not improve from 0.11376
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not
available. Available metrics are: loss,accuracy,val_loss,val_accuracy
3/3 [==============================] - 709s 236s/step - loss: 0.1406 - accurac
y: 0.9500 - val_loss: 0.1304 - val_accuracy: 0.9500
```

In [13]:
```
#Plot Graph to see the result
plt.figure(figsize=(8,5))
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","Loss","Validation Loss"])
plt.show()
```



In [14]:
```
#Save the model
model.save('vgg16.h5')
```

In [16]:
```
del model
```

In [17]:
```
vgg16 = load_model('vgg16.h5')
```

In [18]:
```
vgg16.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

Part 2

In this part, you will evaluate your deep learning models on a test data. For this part, you will need to do the following:

1. Load your saved model that was built using the ResNet50 model.
2. Construct an ImageDataGenerator for the test set. For this ImageDataGenerator instance, you only need to pass the directory of the test images, target size, and the **shuffle** parameter and set it to False.
3. Use the **evaluate_generator** method to evaluate your models on the test data, by passing the above ImageDataGenerator as an argument. You can learn more about **evaluate_generator** [here (https://keras.io/models/sequential/)](https://keras.io/models/sequential/).
4. Print the performance of the classifier using the VGG16 pre-trained model.
5. Print the performance of the classifier using the ResNet pre-trained model.

Use the following cells to evaluate your models.

In [19]:
```
resnet = load_model("resnet.h5")
```

```
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, yo
ur model is starting with a freshly initialized optimizer.
```

In [20]:
```
resnet.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 2048)              23587712
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 4098
=================================================================
Total params: 23,591,810
Trainable params: 23,538,690
Non-trainable params: 53,120
_________________________________________________________________
```

In [21]:
```
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(224, 224),
        batch_size=100,
        class_mode='categorical',
        shuffle=False)
```

```
Found 500 images belonging to 2 classes.
```

In [22]:
```
len(test_generator)
```

Out[22]:
```
5
```

In [23]:
```
loss = resnet.evaluate_generator(test_generator,steps=5,verbose=1)
```

```
5/5 [==============================] - 838s 168s/step - loss: 1.3265 - accurac
y: 0.5000
```

In [24]:
```
loss
```

Out[24]:
```
[1.3265284791588783, 0.5]
```

In [25]:
```
resnet.metrics_names
```

Out[25]:
```
['loss', 'accuracy']
```

In [27]:
```
print("Accuracy for ResNet is %.2f%%" % (loss[1]*100))
```

```
Accuracy for ResNet is 50.00%
```

In [28]:
```
loss2 = vgg16.evaluate_generator(test_generator,steps=5,verbose=1)
```

```
5/5 [==============================] - 367s 73s/step - loss: 0.1464 - accurac
y: 0.9480
```

In [29]:
```
print("Accuracy for VGG is %.2f%%" % (loss2[1]*100))
```

```
Accuracy for VGG is 94.80%
```

Part 3

In this model, you will predict whether the images in the test data are images of cracked concrete or not. You will do the following:

1. Use the **predict_generator** method to predict the class of the images in the test data, by passing the test data ImageDataGenerator instance defined in the previous part as an argument. You can learn more about the **predict_generator** method [here (https://keras.io/models/sequential/)](https://keras.io/models/sequential/).
2. Report the class predictions of the first five images in the test set. You should print something list this:

Positive
Negative
Positive
Positive
Negative

Use the following cells to make your predictions.

In [30]:
```
resnet_predict = resnet.predict_generator(generator=test_generator,steps=5,verbo
se=1)
```

```
5/5 [==============================] - 844s 169s/step
```

In [31]:

```
resnet_predict
```

```
Out[31]: array([[0.9216869 , 0.07831309],
       [0.92157716, 0.07842289],
       [0.9215356 , 0.07846433],
       [0.92253685, 0.07746314],
       [0.9214253 , 0.07857466],
       [0.92207825, 0.0779217 ],
       [0.9212901 , 0.07870995],
       [0.92132145, 0.07867851],
       [0.9213929 , 0.0786071 ],
       [0.92163324, 0.07836676],
       [0.9213261 , 0.07867392],
       [0.9213351 , 0.07866489],
       [0.92126745, 0.07873252],
       [0.9213651 , 0.07863495],
       [0.92131376, 0.07868626],
       [0.921574  , 0.07842605],
       [0.9212273 , 0.07877272],
       [0.9214263 , 0.07857364],
       [0.92137116, 0.07862884],
       [0.92178464, 0.07821533],
       [0.9210988 , 0.07890116],
       [0.92114717, 0.07885284],
       [0.9213146 , 0.07868544],
       [0.92123556, 0.07876439],
       [0.92121005, 0.07878993],
       [0.92123306, 0.07876692],
       [0.9215979 , 0.07840209],
       [0.92157644, 0.0784236 ],
       [0.9213207 , 0.07867935],
       [0.92119765, 0.0788024 ],
       [0.92277926, 0.07722078],
       [0.92114717, 0.07885284],
       [0.92187744, 0.07812262],
       [0.9210526 , 0.07894742],
       [0.9212905 , 0.07870954],
       [0.9209829 , 0.0790171 ],
       [0.9214592 , 0.07854079],
       [0.9231534 , 0.07684664],
       [0.9210671 , 0.07893284],
       [0.92161804, 0.0783819 ],
       [0.9214698 , 0.07853014],
       [0.9212276 , 0.07877243],
       [0.92099243, 0.07900761],
       [0.9221099 , 0.07789013],
       [0.9216296 , 0.07837044],
       [0.9214747 , 0.07852531],
       [0.92124903, 0.07875095],
       [0.9221125 , 0.07788745],
       [0.9228376 , 0.07716238],
       [0.92120045, 0.07879959],
       [0.92253053, 0.07746945],
       [0.92123544, 0.07876457],
       [0.92119914, 0.07880089],
       [0.9212064 , 0.07879357],
       [0.9210919 , 0.07890806],
       [0.9210202 , 0.07897975],
       [0.92116696, 0.07883306],
       [0.92123294, 0.07876705],
       [0.92111355, 0.07888643],
       [0.92140496, 0.078595  ],
       [0.92322266, 0.07677736],
       [0.9213832 , 0.07861682],
       [0.9211256 , 0.07887437],
       [0.9211652 , 0.07883476],
       [0.9212341 , 0.07876594],
       [0.9217736 , 0.07822638],
       [0.92124444, 0.07875548],
```

In [32]:
```
len(resnet_predict)
```
Out[32]:
```
500
```

In [33]:
```
np.round(a=resnet_predict,decimals=3)
```

```
Out[33]: array([[0.922, 0.078],
                [0.922, 0.078],
                [0.922, 0.078],
                [0.923, 0.077],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.923, 0.077],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.923, 0.077],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.922, 0.078],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.923, 0.077],
                [0.921, 0.079],
                [0.923, 0.077],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.923, 0.077],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.921, 0.079],
                [0.922, 0.078],
                [0.921, 0.079],
```

In [34]:

```
vgg_predict = vgg16.predict_generator(generator=test_generator,steps=5,verbose=
1)
```

```
5/5 [==============================] - 327s 65s/step
```

In [35]:
```
vgg_predict
```

```
Out[35]: array([[9.78436530e-01, 2.15634406e-02],
                [6.47004917e-02, 9.35299516e-01],
                [9.66751296e-03, 9.90332425e-01],
                [2.45390132e-01, 7.54609883e-01],
                [9.77425218e-01, 2.25748252e-02],
                [9.20497298e-01, 7.95026794e-02],
                [9.48500633e-01, 5.14993444e-02],
                [9.75193679e-01, 2.48063263e-02],
                [9.78029191e-01, 2.19708458e-02],
                [9.76324797e-01, 2.36752164e-02],
                [9.71785069e-01, 2.82149743e-02],
                [9.45410550e-01, 5.45894057e-02],
                [9.78415310e-01, 2.15847194e-02],
                [9.77163970e-01, 2.28359848e-02],
                [6.42237902e-01, 3.57762069e-01],
                [9.62787509e-01, 3.72124314e-02],
                [9.71709430e-01, 2.82905605e-02],
                [9.80994999e-01, 1.90049447e-02],
                [9.67856646e-01, 3.21433507e-02],
                [1.85327381e-01, 8.14672709e-01],
                [9.58345711e-01, 4.16543335e-02],
                [9.65750992e-01, 3.42490077e-02],
                [9.80306804e-01, 1.96931977e-02],
                [9.79576111e-01, 2.04238780e-02],
                [9.56659794e-01, 4.33402695e-02],
                [9.57621813e-01, 4.23782282e-02],
                [5.33222079e-01, 4.66777891e-01],
                [9.30389881e-01, 6.96101189e-02],
                [6.24859929e-01, 3.75140041e-01],
                [9.60869491e-01, 3.91304828e-02],
                [2.80978769e-01, 7.19021261e-01],
                [9.50272262e-01, 4.97277938e-02],
                [6.23670995e-01, 3.76329035e-01],
                [9.67287123e-01, 3.27128880e-02],
                [9.71206188e-01, 2.87937596e-02],
                [9.55041289e-01, 4.49587218e-02],
                [2.69104898e-01, 7.30895102e-01],
                [2.06871420e-01, 7.93128550e-01],
                [9.79833364e-01, 2.01667026e-02],
                [9.52963948e-01, 4.70360070e-02],
                [9.77202952e-01, 2.27971021e-02],
                [7.35875249e-01, 2.64124751e-01],
                [9.83925223e-01, 1.60747562e-02],
                [9.22190309e-01, 7.78096765e-02],
                [3.51966619e-01, 6.48033381e-01],
                [9.68531668e-01, 3.14682610e-02],
                [9.60019171e-01, 3.99808511e-02],
                [3.14741373e-01, 6.85258567e-01],
                [1.17344216e-01, 8.82655799e-01],
                [9.68562245e-01, 3.14378217e-02],
                [1.54721618e-01, 8.45278382e-01],
                [9.73627508e-01, 2.63725519e-02],
                [9.73936200e-01, 2.60638446e-02],
                [9.78680611e-01, 2.13193540e-02],
                [9.73119557e-01, 2.68804152e-02],
                [9.55021381e-01, 4.49786484e-02],
                [9.73639727e-01, 2.63603479e-02],
                [9.81054425e-01, 1.89456120e-02],
                [8.04405630e-01, 1.95594415e-01],
                [9.60171878e-01, 3.98281068e-02],
                [3.47396642e-01, 6.52603388e-01],
                [9.74164665e-01, 2.58352850e-02],
                [9.79163945e-01, 2.08360404e-02],
                [9.54891562e-01, 4.51083779e-02],
                [9.76826370e-01, 2.31735520e-02],
                [7.46428847e-01, 2.53571153e-01],
                [9.79618669e-01, 2.03813501e-02],
```

In [36]:
```
len(vgg_predict)
```
Out[36]:
```
500
```

In [37]:
```
np.round(a=vgg_predict,decimals=3)
```

```
Out[37]: array([[0.978, 0.022],
               [0.065, 0.935],
               [0.01 , 0.99 ],
               [0.245, 0.755],
               [0.977, 0.023],
               [0.92 , 0.08 ],
               [0.949, 0.051],
               [0.975, 0.025],
               [0.978, 0.022],
               [0.976, 0.024],
               [0.972, 0.028],
               [0.945, 0.055],
               [0.978, 0.022],
               [0.977, 0.023],
               [0.642, 0.358],
               [0.963, 0.037],
               [0.972, 0.028],
               [0.981, 0.019],
               [0.968, 0.032],
               [0.185, 0.815],
               [0.958, 0.042],
               [0.966, 0.034],
               [0.98 , 0.02 ],
               [0.98 , 0.02 ],
               [0.957, 0.043],
               [0.958, 0.042],
               [0.533, 0.467],
               [0.93 , 0.07 ],
               [0.625, 0.375],
               [0.961, 0.039],
               [0.281, 0.719],
               [0.95 , 0.05 ],
               [0.624, 0.376],
               [0.967, 0.033],
               [0.971, 0.029],
               [0.955, 0.045],
               [0.269, 0.731],
               [0.207, 0.793],
               [0.98 , 0.02 ],
               [0.953, 0.047],
               [0.977, 0.023],
               [0.736, 0.264],
               [0.984, 0.016],
               [0.922, 0.078],
               [0.352, 0.648],
               [0.969, 0.031],
               [0.96 , 0.04 ],
               [0.315, 0.685],
               [0.117, 0.883],
               [0.969, 0.031],
               [0.155, 0.845],
               [0.974, 0.026],
               [0.974, 0.026],
               [0.979, 0.021],
               [0.973, 0.027],
               [0.955, 0.045],
               [0.974, 0.026],
               [0.981, 0.019],
               [0.804, 0.196],
               [0.96 , 0.04 ],
               [0.347, 0.653],
               [0.974, 0.026],
               [0.979, 0.021],
               [0.955, 0.045],
               [0.977, 0.023],
               [0.746, 0.254],
               [0.98 , 0.02 ],
```

In [38]:

```
classes = np.round(a=vgg_predict,decimals=3)
```

In [39]:

```
classes
```

```
Out[39]: array([[0.978, 0.022],
                [0.065, 0.935],
                [0.01 , 0.99 ],
                [0.245, 0.755],
                [0.977, 0.023],
                [0.92 , 0.08 ],
                [0.949, 0.051],
                [0.975, 0.025],
                [0.978, 0.022],
                [0.976, 0.024],
                [0.972, 0.028],
                [0.945, 0.055],
                [0.978, 0.022],
                [0.977, 0.023],
                [0.642, 0.358],
                [0.963, 0.037],
                [0.972, 0.028],
                [0.981, 0.019],
                [0.968, 0.032],
                [0.185, 0.815],
                [0.958, 0.042],
                [0.966, 0.034],
                [0.98 , 0.02 ],
                [0.98 , 0.02 ],
                [0.957, 0.043],
                [0.958, 0.042],
                [0.533, 0.467],
                [0.93 , 0.07 ],
                [0.625, 0.375],
                [0.961, 0.039],
                [0.281, 0.719],
                [0.95 , 0.05 ],
                [0.624, 0.376],
                [0.967, 0.033],
                [0.971, 0.029],
                [0.955, 0.045],
                [0.269, 0.731],
                [0.207, 0.793],
                [0.98 , 0.02 ],
                [0.953, 0.047],
                [0.977, 0.023],
                [0.736, 0.264],
                [0.984, 0.016],
                [0.922, 0.078],
                [0.352, 0.648],
                [0.969, 0.031],
                [0.96 , 0.04 ],
                [0.315, 0.685],
                [0.117, 0.883],
                [0.969, 0.031],
                [0.155, 0.845],
                [0.974, 0.026],
                [0.974, 0.026],
                [0.979, 0.021],
                [0.973, 0.027],
                [0.955, 0.045],
                [0.974, 0.026],
                [0.981, 0.019],
                [0.804, 0.196],
                [0.96 , 0.04 ],
                [0.347, 0.653],
                [0.974, 0.026],
                [0.979, 0.021],
                [0.955, 0.045],
                [0.977, 0.023],
                [0.746, 0.254],
                [0.98 , 0.02 ],
```

In [40]:
```
filenames=test_generator.filenames
filenames
```

```
Out[40]: ['negative\\19751.jpg',
 'negative\\19752.jpg',
 'negative\\19753.jpg',
 'negative\\19754.jpg',
 'negative\\19755.jpg',
 'negative\\19756.jpg',
 'negative\\19757.jpg',
 'negative\\19758.jpg',
 'negative\\19759.jpg',
 'negative\\19760.jpg',
 'negative\\19761.jpg',
 'negative\\19762.jpg',
 'negative\\19763.jpg',
 'negative\\19764.jpg',
 'negative\\19765.jpg',
 'negative\\19766.jpg',
 'negative\\19767.jpg',
 'negative\\19768.jpg',
 'negative\\19769.jpg',
 'negative\\19770.jpg',
 'negative\\19771.jpg',
 'negative\\19772.jpg',
 'negative\\19773.jpg',
 'negative\\19774.jpg',
 'negative\\19775.jpg',
 'negative\\19776.jpg',
 'negative\\19777.jpg',
 'negative\\19778.jpg',
 'negative\\19779.jpg',
 'negative\\19780.jpg',
 'negative\\19781.jpg',
 'negative\\19782.jpg',
 'negative\\19783.jpg',
 'negative\\19784.jpg',
 'negative\\19785.jpg',
 'negative\\19786.jpg',
 'negative\\19787.jpg',
 'negative\\19788.jpg',
 'negative\\19789.jpg',
 'negative\\19790.jpg',
 'negative\\19791.jpg',
 'negative\\19792.jpg',
 'negative\\19793.jpg',
 'negative\\19794.jpg',
 'negative\\19795.jpg',
 'negative\\19796.jpg',
 'negative\\19797.jpg',
 'negative\\19798.jpg',
 'negative\\19799.jpg',
 'negative\\19800.jpg',
 'negative\\19801.jpg',
 'negative\\19802.jpg',
 'negative\\19803.jpg',
 'negative\\19804.jpg',
 'negative\\19805.jpg',
 'negative\\19806.jpg',
 'negative\\19807.jpg',
 'negative\\19808.jpg',
 'negative\\19809.jpg',
 'negative\\19810.jpg',
 'negative\\19811.jpg',
 'negative\\19812.jpg',
 'negative\\19813.jpg',
 'negative\\19814.jpg',
 'negative\\19815.jpg',
 'negative\\19816.jpg',
 'negative\\19817.jpg',
```

In [41]:
```
results = pd.DataFrame({"file": filenames,"prediction":vgg_predict[:,0],"class":
classes[:,0]})
```

In [42]:
```
results
```

Out[42]:

	file	prediction	class
0	negative\19751.jpg	0.978437	0.978
1	negative\19752.jpg	0.064700	0.065
2	negative\19753.jpg	0.009668	0.010
3	negative\19754.jpg	0.245390	0.245
4	negative\19755.jpg	0.977425	0.977
...	...	...	...
495	positive\19996.jpg	0.001427	0.001
496	positive\19997.jpg	0.190317	0.190
497	positive\19998.jpg	0.005833	0.006
498	positive\19999.jpg	0.001227	0.001
499	positive\20000.jpg	0.042057	0.042

500 rows × 3 columns

Assumption is prediction > 0.5 , class is 1 else prediction < 0.5, class is 0

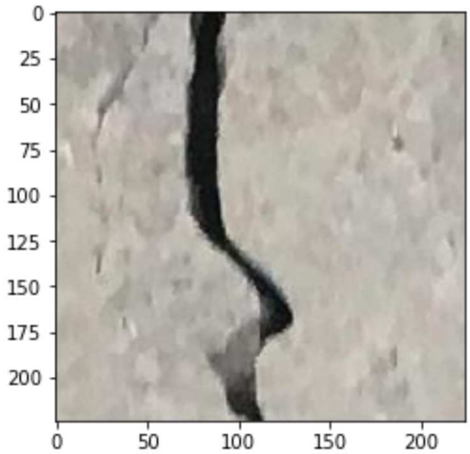
Method 2: Sampling 5 images randomly and predict class

In [48]:
```
from tensorflow.keras.preprocessing import image
```

In [62]:

```python
img1 = image.load_img("19751.jpg",target_size=(224,224))

# img1 = tf.cast(img1, tf.float32)
img1 = np.asarray(img1)

plt.imshow(img1)
#img1 = np.expand_dims(img1, axis=0)

output1 = vgg16.predict(img1)
print(output1)

if output1[0][1] == 0:
    print("negative")
else:
    print('positive')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-62-ec2468f651f1> in <module>
      7 #img1 = np.expand_dims(img1, axis=0)
      8 
----> 9 output1 = vgg16.predict(img1)
     10 print(output1)
     11 

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\training.py in predict(self, x, batch_size, verbose, steps, callbacks, max_q
ueue_size, workers, use_multiprocessing)
    907         max_queue_size=max_queue_size,
    908         workers=workers,
--> 909         use_multiprocessing=use_multiprocessing)
    910 
    911   def reset_metrics(self):

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\training_v2.py in predict(self, model, x, batch_size, verbose, steps, callba
cks, **kwargs)
    460     return self._model_iteration(
    461         model, ModeKeys.PREDICT, x=x, batch_size=batch_size, verbose=v
erbose,
--> 462         steps=steps, callbacks=callbacks, **kwargs)
    463 
    464 

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\training_v2.py in _model_iteration(self, model, mode, x, y, batch_size, verb
ose, sample_weight, steps, callbacks, **kwargs)
    394           sample_weights=sample_weight,
    395           steps=steps,
--> 396           distribution_strategy=strategy)
    397       total_samples = _get_total_number_of_samples(adapter)
    398       use_sample = total_samples is not None

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\training_v2.py in _process_inputs(model, x, y, batch_size, epochs, sample_we
ights, class_weights, shuffle, steps, distribution_strategy, max_queue_size, w
orkers, use_multiprocessing)
    592         batch_size=batch_size,
    593         check_steps=False,
--> 594         steps=steps)
    595   adapter = adapter_cls(
    596       x,

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\training.py in _standardize_user_data(self, x, y, sample_weight, class_weigh
t, batch_size, check_steps, steps_name, steps, validation_split, shuffle, extr
act_tensors_from_dataset)
   2470           feed_input_shapes,
   2471           check_batch_axis=False,  # Don't enforce the batch size.
-> 2472           exception_prefix='input')
   2473 
   2474     # Get typespecs for the input data and sanitize it if necessary.

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\training_utils.py in standardize_input_data(data, names, shapes, check_batch
_axis, exception_prefix)
    563                            ': expected ' + names[i] + ' to have ' +
    564                            str(len(shape)) + ' dimensions, but got arr
ay '
--> 565                            'with shape ' + str(data_shape))
    566         if not check_batch_axis:
    567           data_shape = data_shape[1:]
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

Thank you for completing this lab!

This notebook was created by Alex Aklson.

This notebook is part of a course on **Coursera** called *AI Capstone Project with Deep Learning*. If you accessed this notebook outside the course, you can take this course online by clicking [here (https://cocl.us/DL0321EN_Coursera_Week4_LAB1)](https://cocl.us/DL0321EN_Coursera_Week4_LAB1).
