IBM Developer
SKILLS NETWORK

Loading Data

Objective

- How to download and visualize the image dataset.

Introduction

Crack detection has vital importance for structural health monitoring and inspection. In this series of labs, you learn everything you need to efficiently build a classifier using a pre-trained model that would detect cracks in images of concrete. For problem formulation, we will denote images of cracked concrete as the positive class and images of concrete with no cracks as the negative class.

In this lab, I will walk you through the process of loading and visualizing the image dataset.

Please note: You will encounter questions that you will need to answer in order to complete the quiz for this module.

Table of Contents

Download Data

For your convenience, I have placed the data on a server which you can retrieve easily using the **wget** command. So let's run the following line of code to get the data. Given the large size of the image dataset, it might take some time depending on your internet speed.

[1]:
```
# get the data
!wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/images/concrete_crack_images_for_cla
```

```
--2021-03-20 23:12:52--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/images/concrete_c
rack_images_for_classification.zip
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.254.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 245259777 (234M) [application/zip]
Saving to: 'concrete_crack_images_for_classification.zip'

concrete_crack_imag 100%[===================>] 233.90M  19.0MB/s    in 13s

2021-03-20 23:13:06 (18.3 MB/s) - 'concrete_crack_images_for_classification.zip' saved [245259777/245259777]
```

Support

And now if you check the left directory pane, you should see the zipped file *concrete_crack_images_for_classification.zip* appear. So, let's go ahead and unzip the file to access the images. Given the large number of images in the dataset, this might take a couple of minutes, so please be patient, and wait until the code finishes running.

and wait until the code finishes running.

[*]:
```
!unzip concrete_crack_images_for_classification.zip
```

```
  inflating: Negative/07285.jpg
  inflating: Negative/11264.jpg
  inflating: Negative/00433.jpg
  inflating: Negative/03635.jpg
  inflating: Negative/12346.jpg
  inflating: Negative/12692.jpg
  inflating: Negative/08240.jpg
  inflating: Negative/17764.jpg
  inflating: Negative/16253.jpg
  inflating: Negative/10591.jpg
  inflating: Negative/17964.jpg
  inflating: Negative/02733.jpg
  inflating: Negative/02005.jpg
```

Now, you should see two folders appear in the left pane: *Positive* and *Negative*. *Negative* is the negative class like we defined it earlier and it represents the concrete images with no cracks. *Positive* on the other hand is the positive class and represents the concrete images with cracks.

Important Note: There are thousands and thousands of images in each folder, so please don't attempt to double click on the folders. This may consume all of your memory and you may end up with a **50*** error. So please **DO NOT DO IT**.

Import Libraries and Packages

Before we proceed, let's import the libraries and packages that we will need to complete the rest of this lab.

[3]:
```
import os
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
```

Load Images

Next, we will use the standard approach of loading all images into memory and demonstrate how this approach is not efficient at all when it comes to building deep learning models for classifying images.

Let's start by reading in the negative images. First, we will use **os.scandir** to build an iterator to iterate through *./Negative* directory that contains all the images with no cracks.

[4]:
```
negative_files = os.scandir('./Negative')
negative_files
```

[4]:
```
<posix.ScandirIterator at 0x7fa78f18dcf0>
```

Then, we will grab the first file in the directory.

[5]:
```
file_name = next(negative_files)
file_name
```

[5]:
```
<DirEntry '10796.jpg'>
```

Since the directory can contain elements that are not files, we will only read the element if it is a file.

[6]:
```
os.path.isfile(file_name)
```

[6]:
```
True
```

Get the image name.

[7]:
```
image_name = str(file_name).split("'")[1]
image_name
```

[7]:
```
'10796.jpg'
```

Read in the image data.

[8]:
```
image_data = plt.imread('./Negative/{}'.format(image_name))
image_data
```

[8]:
```
array([[[215, 211, 199],
        [218, 214, 202],
        [216, 212, 200],
        ...,
        [202, 193, 184],
        [199, 190, 181],
        [195, 186, 177]],

       [[213, 209, 197],
        [216, 212, 200],
        [216, 212, 200],
        ...,
        [202, 193, 184],
```

Question: What is the dimension of a single image according to **image_data**?

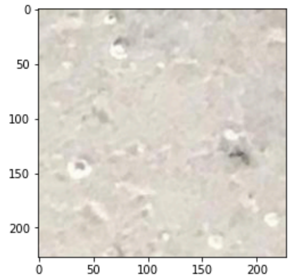
[9]:
```python
## You can use this cell to type your code to answer the above question
image_data.shape
```

[9]:
```
(227, 227, 3)
```

Let's view the image.

[10]:
```python
plt.imshow(image_data)
```

[10]:
```
<matplotlib.image.AxesImage at 0x7fa77c358a58>
```



Now that we are familiar with the process of reading in an image data, let's loop through all the image in the *./Negative* directory and read them all in and save them in the list **negative_images**. We will also time it to see how long it takes to read in all the images.

[ ]:
```python
%%time

negative_images = []
for file_name in negative_files:
    if os.path.isfile(file_name):
        image_name = str(file_name).split("'")[1]
        image_data = plt.imread('./Negative/{}'.format(image_name))
        negative_images.append(image_data)

negative_images = np.array(negative_images)
```

Oops! The kernel died due to an out-of-memory error. Since the kernel died, you may have to run the above cell to load the libraries and packages again.

Loading images into memory is definitely not the right approach when working with images as you can hit your limit on memory and other resources fairly quickly. Therefore, let's repeat the previous process but let's save the paths to the images in a variable instead of loading and saving the images themselves.

So instead of using **os.scandir**, we will use **os.listdir**.

[11]:
```python
negative_images = os.listdir('./Negative')
negative_images
```

[11]:
```
['10796.jpg',
 '05507.jpg',
 '19194.jpg',
 '12767.jpg',
 '11856.jpg',
 '05641.jpg',
 '16794.jpg',
 '11687.jpg',
 '11166.jpg',
 '11918.jpg',
 '05081.jpg',
 '13959.jpg',
 '15188.jpg',
```

Notice how the images are not sorted, so let's call the `sort` method to sort the images.

[12]:
```python
negative_images.sort()
negative_images
```

[12]:
```
['00001.jpg',
 '00002.jpg',
 '00003.jpg',
 '00004.jpg',
 '00005.jpg',
 '00006.jpg',
 '00007.jpg',
 '00008.jpg',
 '00009.jpg',
 '00010.jpg',
 '00011.jpg',
 '00012.jpg',
 '00013.jpg',
```

Before we can show an image, we need to open it, which we can do using the **Image** module in the **PIL** library. So to open the first image, we run the following:

[13]:
```
image_data = Image.open('./Negative/{}'.format(negative_images[0]))
```

Then to view the image, you can simply run:

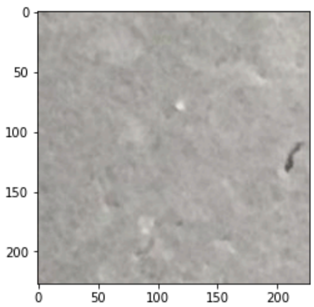
[14]:
```
image_data
```

[14]:

or use the `imshow` method as follows:

[15]:
```
plt.imshow(image_data)
```

[15]:
```
<matplotlib.image.AxesImage at 0x7fa773c26f98>
```



Let's loop through all the images in the `./Negative` directory and add save their paths.

[16]:
```
negative_images_dir = ['./Negative/{}'.format(image) for image in negative_images]
negative_images_dir
```

[16]:
```
['./Negative/00001.jpg',
 './Negative/00002.jpg',
 './Negative/00003.jpg',
 './Negative/00004.jpg',
 './Negative/00005.jpg',
 './Negative/00006.jpg',
 './Negative/00007.jpg',
 './Negative/00008.jpg',
 './Negative/00009.jpg',
 './Negative/00010.jpg',
 './Negative/00011.jpg',
 './Negative/00012.jpg',
 './Negative/00013.jpg',
```

Let's check how many images with no cracks exist in the dataset.

[17]:
```
len(negative_images_dir)
```

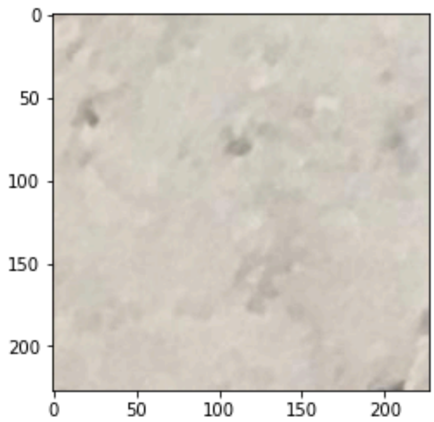
[17]:
```
20000
```

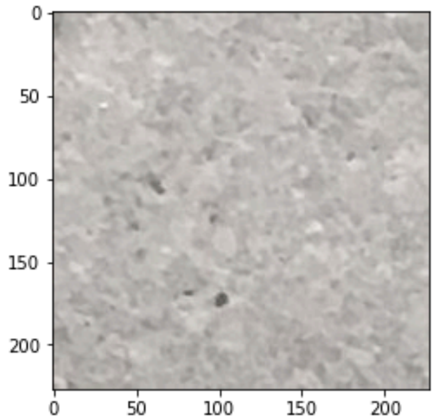
Question: Show the next four images.

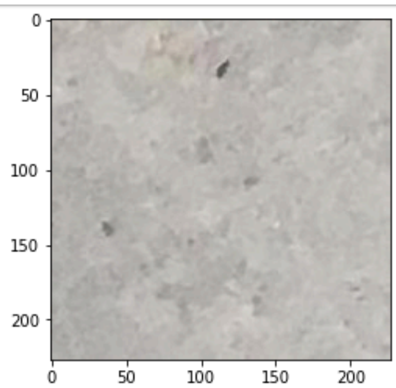
Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. Learn more here.

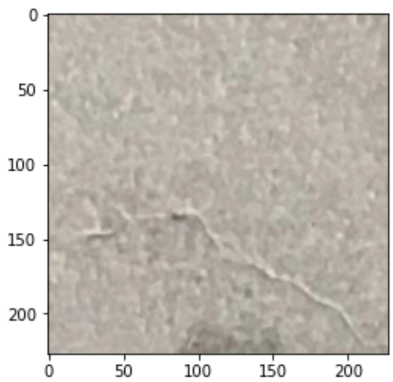
[19]:
```
## You can use this cell to type your code to answer the above question


for i in negative_images_dir[1:5]:
    a= plt.imread(i)
    plt.imshow(a)
    plt.show()
```


0
50
100
150
200
0 50 100 150 200


0
50
100
150
200
0 50 100 150 200


0
50
100
150
200
0 50 100 150 200


0
50
100
150
200
0 50 100 150 200

Your turn: Save the paths to all the images in the *./Positive* directory in a list called **positive_images_dir**. Make sure to sort the paths.

[22]:
```
## Type your answer here

positive_images_dir = os.listdir('Positive')
positive_images_dir.sort()
positive_images_dir = ['./Positive/{}'.format(image) for image in positive_images_dir]
positive_images_dir
```

[22]:
```
['./Positive/00001.jpg',
 './Positive/00002.jpg',
 './Positive/00003.jpg',
 './Positive/00004.jpg',
 './Positive/00005.jpg',
 './Positive/00006.jpg',
 './Positive/00007.jpg',
 './Positive/00008.jpg',
 './Positive/00009.jpg',
 './Positive/00010.jpg',
```

Question: How many images of cracked concrete exist in the *./Positive* directory?

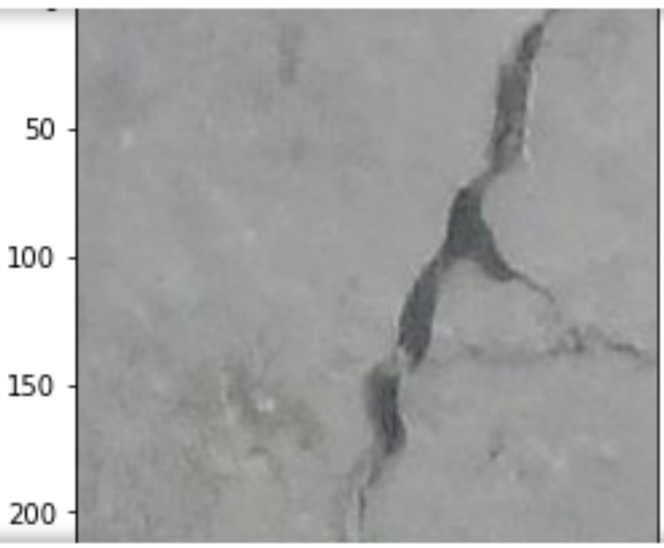
[23]:
```
## You can use this cell to type your code to answer the above question
len(positive_images_dir)
```

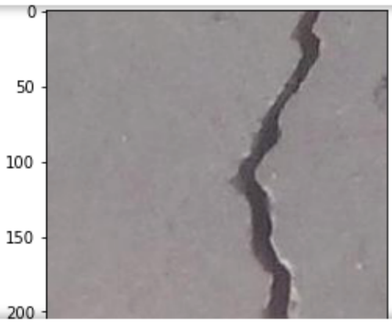
[23]:
```
20000
```

Question: Show the first four images with cracked concrete.

[24]:
```
## You can use this cell to type your code to answer the above question

for i in positive_images_dir[0:4]:
    b = plt.imread(i)
    plt.imshow(b)
    plt.show()
```





Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational.

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.

About the Authors:

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.