Classification between Cracked, and Non-Cracked Images using ResNet-18.

Contents:

1. Introduction

2. How does the ResNet Convolutional Neural Network work ?

3. Concrete Dataset

4. Classification Model (ResNet-18) using Python.

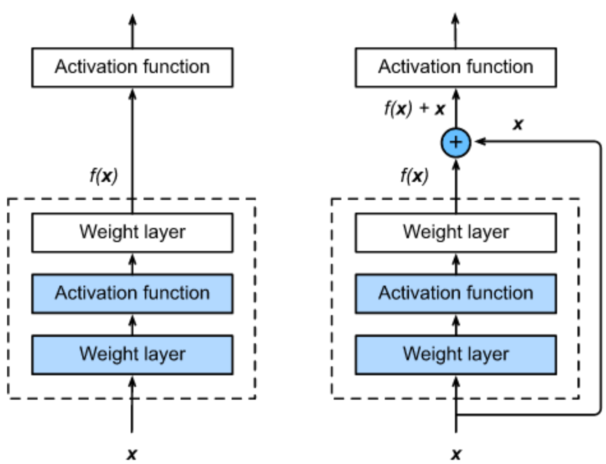
5. Link to Github Repository

6. References

Introduction:-

I am using a concrete dataset with positive and negative images, positive means images with cracks on the wall and negative means otherwise.I will be using a pre-trained model called ResNet18. ResNet-18 is a convolutional neural network that is 18 layers deep. You can load a pre-trained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

How does the ResNet Convolutional Neural Network work ?

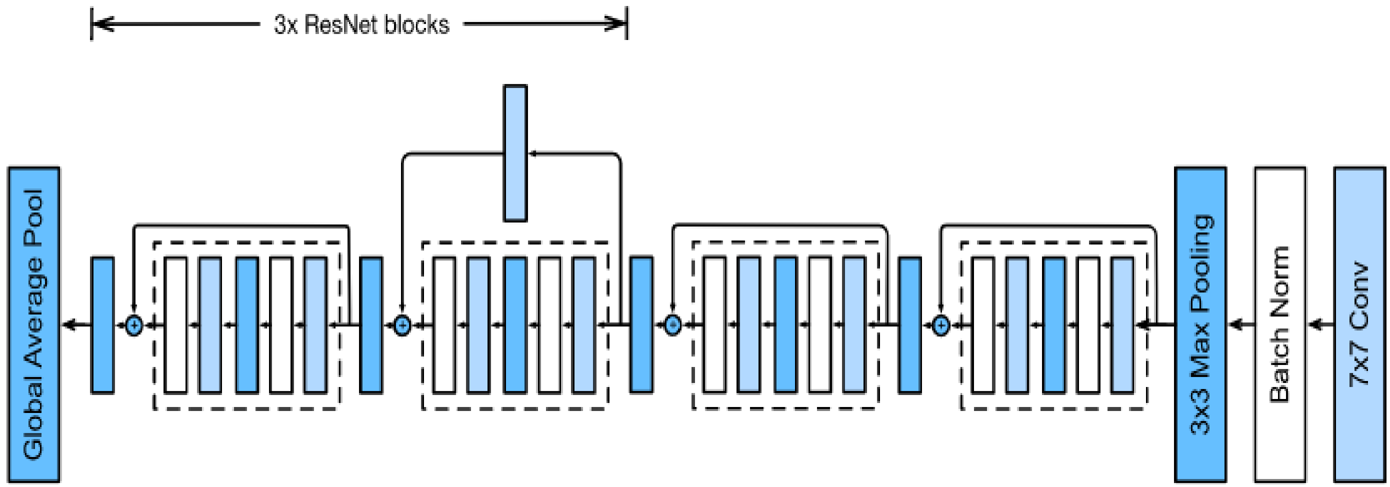
<table>
<tr>
<td>


</td>
<td>
This diagram shows a basic Residual block.
We can see that an input x obtains a learning f(x).
Now what we do is that we use skip connections.With the help of these skip connections we get the output as:-

$x[l+1]=g(z[l+1]+x[l])$.

Where:
$z[l+1]=w[l+1]*x[l] + b[l+1]$

Suppose the values of weight and bias for second network is equal to 0,then our equation will become $x[l+1]=g(x[l])$.Which will make our model learn the identity function more easily
</td>
</tr>
</table>



Resnet-18

Concrete Dataset

This dataset contains two types of images, Positive and Negative,Positive images are walls with cracks and Negative images are walls with no cracks. The size of the dataset is 40000,with 20000 images each in Positive and Negative respectively.Below are the first images from each batch.

	
Positive	Negative

Classification Model (ResNet-18) using Python.

- Upload the Data and Unzip it

```
In [0]: # To upload the Concrete Data with Positive Tensors

        !wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/image
        s/Positive_tensors.zip
        !unzip -q Positive_tensors.zip

--2020-06-14 12:10:59--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/
DL0321EN/data/images/Positive_tensors.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)|67.22
8.254.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2598656062 (2.4G) [application/zip]
Saving to: ‘Positive_tensors.zip’

100%[===================================>] 2,598,656,062 47.2MB/s   in 50s

2020-06-14 12:11:50 (49.3 MB/s) - ‘Positive_tensors.zip’ saved [2598656062/2598656062]
```

```
In [0]: # To upload the Concrete Data with Negative Tensors

        ! wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0321EN/data/imag
        es/Negative_tensors.zip
        !unzip -q Negative_tensors.zip

--2020-06-14 12:15:10--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/
DL0321EN/data/images/Negative_tensors.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)|67.22
8.254.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2111408108 (2.0G) [application/zip]
Saving to: ‘Negative_tensors.zip’

100%[===================================>] 2,111,408,108 47.1MB/s   in 43s

2020-06-14 12:15:54 (46.8 MB/s) - ‘Negative_tensors.zip’ saved [2111408108/2111408108]
```

- Install necessary libraries

```
In [0]: !pip install torchvision

Collecting torchvision
  Downloading https://files.pythonhosted.org/packages/61/51/aa2770a70f612ce9a2fc7da3a1a93f9ecf8746788256fe
d6b691f9b31ca9/torchvision-0.6.0-cp36-cp36m-manylinux1_x86_64.whl (6.6MB)
     |████████████████████████████████| 6.6MB 6.0MB/s eta 0:00:01
Requirement already satisfied: numpy in /opt/conda/envs/Python36/lib/python3.6/site-packages (from torchvi
sion) (1.15.4)
Requirement already satisfied: pillow>=4.1.1 in /opt/conda/envs/Python36/lib/python3.6/site-packages (from
torchvision) (5.4.1)
Collecting torch==1.5.0 (from torchvision)
  Downloading https://files.pythonhosted.org/packages/13/70/54e9fb010fe1547bc4774716f11ececb81ae5b306c05f0
90f4461ee13205/torch-1.5.0-cp36-cp36m-manylinux1_x86_64.whl (752.0MB)
     |████████████████████████████████| 752.0MB 16kB/s s eta 0:00:01     |█                              |
24.1MB 31.1MB/s eta 0:00:24██                            | 157.7MB 29.7MB/s eta 0:00:21/s eta 0:00:14MB/s et
a 0:00:07��██████████████      | 661.4MB 25.5MB/s eta 0:00:04     |█████████████████████████████   | 700.
1MB 35.9MB/s eta 0:00:02
Requirement already satisfied: future in /opt/conda/envs/Python36/lib/python3.6/site-packages (from torch=
=1.5.0->torchvision) (0.17.1)
Installing collected packages: torch, torchvision
Successfully installed torch-1.5.0 torchvision-0.6.0
```

```
In [0]: !pip install torchsummary

Collecting torchsummary
  Downloading https://files.pythonhosted.org/packages/7d/18/1474d06f721b86e6a9b9d7392ad68bed711a02f3b61ac4
3f13c719db50a6/torchsummary-1.5.1-py3-none-any.whl
Installing collected packages: torchsummary
Successfully installed torchsummary-1.5.1
```

- Import important libraries

```
In [0]: import torchvision.models as models
from PIL import Image
import pandas
from torchvision import transforms
import torch.nn as nn
import time
import torch
import matplotlib.pylab as plt
import numpy as np
from torch.utils.data import Dataset, DataLoader
import h5py
import os
import glob
torch.manual_seed(0)
from matplotlib.pyplot import imshow
import matplotlib.pylab as plt
from PIL import Image
import pandas as pd
import os
import torch
```

- Create dataset

```
class Dataset(Dataset):

    # Constructor
    def __init__(self,transform=None,train=True):
        directory="/home/dsxuser/work"
        positive="Positive_tensors"
        negative='Negative_tensors'

        positive_file_path=os.path.join(directory,positive)
        negative_file_path=os.path.join(directory,negative)
        positive_files=[os.path.join(positive_file_path,file) for file in os.listdir(positive_file_path) i
f file.endswith(".pt")]
        negative_files=[os.path.join(negative_file_path,file) for file in os.listdir(negative_file_path) i
f file.endswith(".pt")]
        number_of_samples=len(positive_files)+len(negative_files)
        self.all_files=[None]*number_of_samples
        self.all_files[::2]=positive_files
        self.all_files[1::2]=negative_files
        # The transform is goint to be used on image
        self.transform = transform
        #torch.LongTensor
        self.Y=torch.zeros([number_of_samples]).type(torch.LongTensor)
        self.Y[::2]=1
        self.Y[1::2]=0

        if train:
            self.all_files=self.all_files[0:30000]
            self.Y=self.Y[0:30000]
            self.len=len(self.all_files)
            print('Number of images in Training Dataset :-',len(self.all_files))
        else:
            self.all_files=self.all_files[30000:]
            self.Y=self.Y[30000:]
            self.len=len(self.all_files)
            print('Number of images in Validation Dataset :-',len(self.all_files))


    # Get the Length
    def __len__(self):
        return self.len
```

```

    # Get the Length
    def __len__(self):
        return self.len

    # Getter
    def __getitem__(self, idx):

        image=torch.load(self.all_files[idx])
        y=self.Y[idx]

        # If there is any transform method, apply it onto the image
        if self.transform:
            image = self.transform(image)

        return image, y

print("done")
```

```
done
```

5.1 : Split the dataset into Training and Validating Datasets.

```
# Training Data
train_dataset = Dataset(train=True)
```

```
Number of images in Training Dataset :- 30000
```

```
# Validating Data
validation_dataset = Dataset(train=False)
```

```
Number of images in Validation Dataset :- 10000
```

- Data Visualization

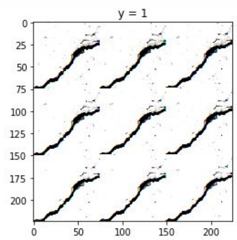
6.1 : Function to visualize the data

```
def visualize(data_sample):
    plt.imshow(data_sample[0].numpy().reshape(224, 224, 3))
    plt.title('y = ' + str(data_sample[1].item()))
```

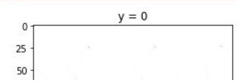
6.2 : Visualizing the data before classification

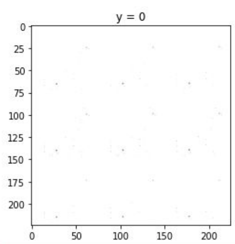
```
#Visualizing the first four images in training dataset
for i, (x, y) in enumerate(train_dataset):
    if i<4:
        visualize(train_dataset[i])
        plt.show()
    else:
        break
# I have titled every image with either 0 or 1
# 0 :- Non-cracked image
# 1 :- Cracked image
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
s).
```

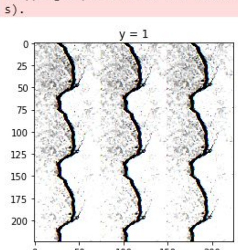



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
s).
```

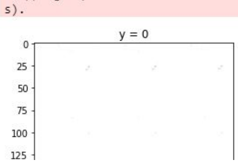





```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
s).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
s).
```



- Create the Model

7.1 : Importing the ResNet18 model and setting the parameter pre-trained and progress to True

```
In [0]: import torchvision.models as models
        model = models.resnet18(pretrained=True,progress=True)
```

7.2 : Setting the attribute requires_grad to False. As a result, the parameters will not be affected by training.

```
In [0]: for param in model.parameters():
            param.requires_grad = False
```

7.3 : Replacing the output layer model.fc of the neural network with a nn.Linear object, to classify 2 different classes.

```
In [0]: model.fc = nn.Linear(512, 2)
```

7.4 : Summary of our ResNet18 model.

```
In [0]: from torchsummary import summary
        summary(model, (3, 227, 227))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 114, 114]           9,408
       BatchNorm2d-2         [-1, 64, 114, 114]             128
              ReLU-3         [-1, 64, 114, 114]               0
         MaxPool2d-4           [-1, 64, 57, 57]               0
            Conv2d-5           [-1, 64, 57, 57]          36,864
       BatchNorm2d-6           [-1, 64, 57, 57]             128
              ReLU-7           [-1, 64, 57, 57]               0
            Conv2d-8           [-1, 64, 57, 57]          36,864
       BatchNorm2d-9           [-1, 64, 57, 57]             128
             ReLU-10           [-1, 64, 57, 57]               0
       BasicBlock-11           [-1, 64, 57, 57]               0
           Conv2d-12           [-1, 64, 57, 57]          36,864
      BatchNorm2d-13           [-1, 64, 57, 57]             128
             ReLU-14           [-1, 64, 57, 57]               0
           Conv2d-15           [-1, 64, 57, 57]          36,864
      BatchNorm2d-16           [-1, 64, 57, 57]             128
             ReLU-17           [-1, 64, 57, 57]               0
       BasicBlock-18           [-1, 64, 57, 57]               0
           Conv2d-19          [-1, 128, 29, 29]          73,728
      BatchNorm2d-20          [-1, 128, 29, 29]             256
       BasicBlock-34          [-1, 128, 29, 29]               0
           Conv2d-35          [-1, 256, 15, 15]         294,912
      BatchNorm2d-36          [-1, 256, 15, 15]             512
             ReLU-37          [-1, 256, 15, 15]               0
           Conv2d-38          [-1, 256, 15, 15]         589,824
      BatchNorm2d-39          [-1, 256, 15, 15]             512
           Conv2d-40          [-1, 256, 15, 15]          32,768
      BatchNorm2d-41          [-1, 256, 15, 15]             512
             ReLU-42          [-1, 256, 15, 15]               0
       BasicBlock-43          [-1, 256, 15, 15]               0
           Conv2d-44          [-1, 256, 15, 15]         589,824
      BatchNorm2d-45          [-1, 256, 15, 15]             512
             ReLU-46          [-1, 256, 15, 15]               0
           Conv2d-47          [-1, 256, 15, 15]         589,824
      BatchNorm2d-48          [-1, 256, 15, 15]             512
             ReLU-49          [-1, 256, 15, 15]               0
       BasicBlock-50          [-1, 256, 15, 15]               0
           Conv2d-51            [-1, 512, 8, 8]       1,179,648
      BatchNorm2d-52            [-1, 512, 8, 8]           1,024
             ReLU-53            [-1, 512, 8, 8]               0
           Conv2d-54            [-1, 512, 8, 8]       2,359,296
      BatchNorm2d-55            [-1, 512, 8, 8]           1,024
           Conv2d-56            [-1, 512, 8, 8]         131,072
      BatchNorm2d-57            [-1, 512, 8, 8]           1,024
             ReLU-58            [-1, 512, 8, 8]               0
       BasicBlock-59            [-1, 512, 8, 8]               0
           Conv2d-60            [-1, 512, 8, 8]       2,359,296
      BatchNorm2d-61            [-1, 512, 8, 8]           1,024
             ReLU-62            [-1, 512, 8, 8]               0
           Conv2d-63            [-1, 512, 8, 8]       2,359,296
      BatchNorm2d-64            [-1, 512, 8, 8]           1,024
             ReLU-65            [-1, 512, 8, 8]               0
       BasicBlock-66            [-1, 512, 8, 8]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                    [-1, 2]           1,026
================================================================
Total params: 11,177,538
Trainable params: 1,026
Non-trainable params: 11,176,512
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 67.01
Params size (MB): 42.64
Estimated Total Size (MB): 110.24
----------------------------------------------------------------
```

- Perform Classification with the trained model

8.1 : Create the loss function

```
In [0]: criterion = nn.CrossEntropyLoss()
```

8.2 : Creating a training loader and validation loader object,with the batch size equal to 100 samples each.

```
In [0]: train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100)
        validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_size=100)
```

8.3 : Used Adam optimizer to minimise the loss

```
In [0]: optimizer = torch.optim.Adam([parameters  for parameters in model.parameters() if parameters.requires_gra
        d],lr=0.001)
```

8.4 : Running the model on the dataset

```
In [0]: import time
        n_epochs=1
        loss_list=[]
        accuracy_list=[]
        correct=0
        N_test=len(validation_dataset)
        N_train=len(train_dataset)
        start_time = time.time()
        #n_epochs
        j=1
        i=1
        k=0
        Loss=0
        start_time = time.time()

        for epoch in range(n_epochs):

            print('TRAINING PHASE')
            print('************************************************************')
            for x, y in train_loader:
                print('Loop Number:-',i,'/300')
                t = time.time()
                model.train()
                #clear gradient
                optimizer.zero_grad()
```

```
        t = time.time()
        model.train()
        #clear gradient
        optimizer.zero_grad()
        #make a prediction
        z = model(x)
        # calculate loss
        loss = criterion(z, y)
        # calculate gradients of parameters
        #loss_sublist.append(loss.data.item())
        loss.backward()
        # update parameters
        optimizer.step()
        loss_list.append(loss.data)
        print(loss.data)
        k=k+1
        i=i+1
        print("Finished this iteration of training phase in {} (s)".format(time.time()-t))
        print('-------------------------------------------------------------------')


    print('VALIDATION PHASE')
    print('************************************************************')
    correct=0
    for x_test, y_test in validation_loader:
        print("Loop Number:-",j,'/100')
        t = time.time()
        # set model to eval
        model.eval()
        #make a prediction
        z = model(x_test)
        #find max
        _, yhat = torch.max(z.data, 1)

        #Calculate misclassified  samples in mini-batch
        #hint +=(yhat==y_test).sum().item()
        correct += (yhat==y_test).sum().item()
        j=j+1
        print()
        print("Finished this iteration of validation phase in {} (s)".format(time.time()-t))
        print('-------------------------------------------------------------------')
    print('************************************************************')

    accuracy=correct/N_test
    accuracy_list.append(accuracy)
```

- Accuracy and Misclassified Data

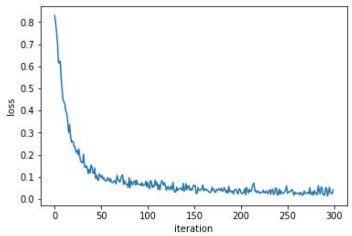
8.5 : Model Accuracy

In [0]:
```
print(accuracy*100,'%')
```

```
99.33 %
```

8.6 : Loss plot

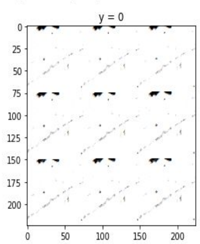
In [0]:
```
plt.plot(loss_list)
plt.xlabel("iteration")
plt.ylabel("loss")
plt.show()
```



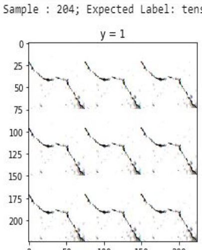
6.3 : Visualizing the misclassified images

In [0]:
```
count=0
validation = torch.utils.data.DataLoader(dataset=validation_dataset, batch_size=1)
for i, (x_test, y_test) in enumerate(validation):
    z = model(x_test)
    _, yhat = torch.max(z.data, 1)
    if yhat != y_test:
        print("Sample : {}; Expected Label: {}; Obtained Label: {}".format(str(i), str(y_test), str(yha
t)))
        visualize((x_test,y_test))
        plt.show()
        count=count+1
        if count>=4:
            break
```

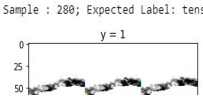
```
Sample : 195; Expected Label: tensor([0]); Obtained Label: tensor([1])
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
s).
Sample : 204; Expected Label: tensor([1]); Obtained Label: tensor([0])
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
s).
Sample : 280; Expected Label: tensor([1]); Obtained Label: tensor([0])
```



Link To Github Repository:-

https://github.com/supragyabajpai/DeepLearning_Repository/blob/master/Concrete_Crack.ipynb

References :-

- Weblinks :
  - 7.6. Residual Networks (ResNet) — Dive into Deep Learning 0.8.0 documentation
  - torchvision.models — PyTorch 1.5.0 documentation
  - PyTorch ResNet: Building, Training and Scaling Residual Networks on PyTorch

- Step wise:-
  - Python Template : 4_1_resnet18_PyTorch_(2).ipynb
  - Data Visualization : How to Load and Manipulate Images for Deep Learning in Python With PIL/Pillow
  - ResNet 18:
    - Show notebooks in Drive
    - C4W2L03 Resnets
    - C4W2L04 Why ResNets Work

  - Cross Entropy :Understand the Softmax Function in Minutes - Data Science Bootcamp
  - Adam optimizer: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam%20is%20an%20optimization%20algorithm,iterative%20based%20in%20training%20data.&text=The%20algorithm%20is%20called%20Adam.
- ResNet 18 research paper :-https://arxiv.org/pdf/1512.03385.pdf