In [1]:
```
import torch
import torch.nn as nn
from torchvision.transforms import transforms
import numpy as np
from torch.autograd import Variable
from torchvision.models import squeezenet1_1
import torch.functional as F
from io import open
import os
from PIL import Image
import pathlib
import glob
import cv2
```

In [2]:
```
train_path='E:\CNN for Bone fracture Image\Fracture detection\seg_train\seg_tra
test_path='E:\CNN for Bone fracture Image\Fracture detection\seg_test\seg_test
pred_path='E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg_pred
```

In [3]:
```
#categories
root=pathlib.Path(train_path)
classes=sorted([j.name.split('/')[-1] for j in root.iterdir()])
```

In [4]:
```
#CNN Network


class ConvNet(nn.Module):
    def __init__(self,num_classes=6):
        super(ConvNet,self).__init__()

        #Output size after convolution filter
        #((w-f+2P)/s) +1

        #Input shape= (256,3,150,150)

        self.conv1=nn.Conv2d(in_channels=3,out_channels=12,kernel_size=3,strid
        #Shape= (256,12,150,150)
        self.bn1=nn.BatchNorm2d(num_features=12)
        #Shape= (256,12,150,150)
        self.relu1=nn.ReLU()
        #Shape= (256,12,150,150)

        self.pool=nn.MaxPool2d(kernel_size=2)
        #Reduce the image size be factor 2
        #Shape= (256,12,75,75)

        self.conv2=nn.Conv2d(in_channels=12,out_channels=20,kernel_size=3,stri
        #Shape= (256,20,75,75)
        self.bn2=nn.BatchNorm2d(num_features=20)
        #Shape= (256,20,75,75)
        self.relu2=nn.ReLU()
        #Shape= (256,20,75,75)

        self.pool2=nn.MaxPool2d(kernel_size=2)
        #Reduce the image size be factor 2
        #Shape= (256,20,75,75)


        self.conv3=nn.Conv2d(in_channels=20,out_channels=32,kernel_size=3,stri
        #Shape= (256,32,75,75)
        self.bn3=nn.BatchNorm2d(num_features=32)
        #Shape= (256,32,75,75)
        self.relu3=nn.ReLU()
        #Shape= (256,32,75,75)

        self.pool3=nn.MaxPool2d(kernel_size=2)
        #Reduce the image size be factor 2
        #Shape= (256,32,75,75)


        self.fc=nn.Linear(in_features=75 * 75 * 32,out_features=num_classes)


        #Feed forwad function

    def forward(self,input):
        output=self.conv1(input)
        output=self.bn1(output)
        output=self.relu1(output)
```

```
        output=self.pool(output)

        output=self.conv2(output)
        output=self.bn2(output)
        output=self.relu2(output)


        output=self.conv3(output)
        output=self.bn3(output)
        output=self.relu3(output)




            #Above output will be in matrix form, with shape (256,32,75,75)

        output=output.view(-1,32*75*75)


        output=self.fc(output)

        return output
```

In [5]:
```
checkpoint=torch.load('best_checkpoint.model')
model=ConvNet(num_classes=2)
model.load_state_dict(checkpoint)
model.eval()
```

Out[5]:
```
ConvNet(
  (conv1): Conv2d(3, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu1): ReLU()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  (conv2): Conv2d(12, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
e=False)
  (conv3): Conv2d(20, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu3): ReLU()
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mod
e=False)
  (fc): Linear(in_features=180000, out_features=2, bias=True)
)
```

In [6]:
```
#Transforms
transformer=transforms.Compose([
    transforms.Resize((150,150)),
    transforms.ToTensor(),  #0-255 to 0-1, numpy to tensors
    transforms.Normalize([0.5,0.5,0.5], # 0-1 to [-1,1] , formula (x-mean)/std
                         [0.5,0.5,0.5])
])
```

In [7]:
```
#prediction function
def prediction(img_path,transformer):

    image=Image.open(img_path)

    image_tensor=transformer(image).float()


    image_tensor=image_tensor.unsqueeze_(0)

    if torch.cuda.is_available():
        image_tensor.cuda()

    input=Variable(image_tensor)


    output=model(input)

    index=output.data.numpy().argmax()

    pred=classes[index]

    return pred
```

In [21]:
```
images_path=glob.glob(pred_path+'/*.jpg')
```

In [22]:
```
pred_dict={}

idx = 0
for imgPath in images_path:
    print("processing, ", imgPath)
    pred_dict[idx]=prediction(imgPath,transformer)
    print("well done")

    idx = idx + 1
```

```
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\85_jpg.rf.8ea2a0ee640bcf6ecbe162a905cbdbaf.jpg
well done
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\87_jpg.rf.0fe6c38fe536952979272ec167087cd6.jpg
well done
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\89_jpg.rf.31dc58444ebae3a21b38822f8d0b75b6.jpg
well done
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\8_jpg.rf.1a2c67587b5266449e319b6682d062a4.jpg
well done
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\8_jpg.rf.7f03f15f9a90dfca6712a551d6fb5602 - Copy.jpg
well done
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\91_jpg.rf.c1b726bceb3d0a2e6b2450fe88fac3d7.jpg
well done
processing,  E:\CNN for Bone fracture Image\Fracture detection\seg_pred\seg
_pred\94_jpg.rf.60be4f10d9cafc5ab5aa2199a938584d.jpg
```

In [23]:
```
pred_dict
```

Out[23]:
```
{0: 'Non-fracture Image',
 1: 'Non-fracture Image',
 2: 'Non-fracture Image',
 3: 'Non-fracture Image',
 4: 'Fracture Image',
 5: 'Non-fracture Image',
 6: 'Non-fracture Image',
 7: 'Non-fracture Image',
 8: 'Fracture Image',
 9: 'Non-fracture Image',
 10: 'Non-fracture Image',
 11: 'Non-fracture Image',
 12: 'Non-fracture Image',
 13: 'Non-fracture Image',
 14: 'Non-fracture Image',
 15: 'Fracture Image',
 16: 'Fracture Image',
 17: 'Non-fracture Image',
 18: 'Non-fracture Image',
```