In [2]:
```
#Load Libraries
import os
import numpy as np
import torch
import glob
import torch.nn as nn
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torch.optim import Adam
from torch.autograd import variable
import torchvision
import pathlib
```

In [3]:
```
#checking for device
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [4]:
```
print(device)
```

```
cpu
```

In [5]:
```
#Transforms
transformer=transforms.Compose([
    transforms.Resize((150,150)),
    transforms.RandomHorizontalFlip(),
    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 [6]:
```
#Dataloader

#Path for training and testing directory
train_path='E:\CNN for Bone fracture Image 02 -\Fracture detection\seg_train\se
test_path='E:\CNN for Bone fracture Image 02 -\Fracture detection\seg_test\seg_

train_loader=DataLoader(
    torchvision.datasets.ImageFolder(train_path,transform=transformer),
    batch_size=256, shuffle=True
)
test_loader=DataLoader(
    torchvision.datasets.ImageFolder(test_path,transform=transformer),
    batch_size=128, shuffle=True
)
```

In [7]:
```
#categories
root=pathlib.Path(train_path)
classes=sorted([j.name.split('/')[-1] for j in root.iterdir()])
```

In [8]:
```
print(classes)
```

```
['Fracture Image', 'Non-fracture Image']
```

In [9]:
```
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 [10]:

```
model=ConvNet(num_classes=2).to(device)
```

In [11]:

```
#Optimizer and loss function
optimizer=Adam(model.parameters(),lr=0.001,weight_decay=0.0001)
loss_function=nn.CrossEntropyLoss()
```

In [12]:

```
num_epochs=14
```

In [13]:

```
#calculating the size of training and testing images
train_count=len(glob.glob(train_path+'/**/*.jpg'))
test_count=len(glob.glob(test_path+'/**/*.jpg'))
```

In [14]:

```
print(train_count,test_count)
```

```
9193 8907
```

In [15]:

```
#Model training and saving best model

best_accuracy=0.0

for epoch in range(num_epochs):

    #Evaluation and training on training dataset
    model.train()
    train_accuracy=0.0
    train_loss=0.0

    for i, (images,labels) in enumerate(train_loader):
        if torch.cuda.is_available():
            images=Variable(images.cuda())
            labels=Variable(labels.cuda())

        optimizer.zero_grad()

        outputs=model(images)
        loss=loss_function(outputs,labels)
        loss.backward()
        optimizer.step()


        train_loss+= loss.cpu().data*images.size(0)
        _,prediction=torch.max(outputs.data,1)

        train_accuracy+=int(torch.sum(prediction==labels.data))

    train_accuracy=train_accuracy/train_count
    train_loss=train_loss/train_count


     # Evaluation on testing dataset
    model.eval()

    test_accuracy=0.0
    for i, (images,labels) in enumerate(test_loader):
        if torch.cuda.is_available():
            images=Variable(images.cuda())
            labels=Variable(labels.cuda())

        outputs=model(images)
        _,prediction=torch.max(outputs.data,1)
        test_accuracy+=int(torch.sum(prediction==labels.data))

    test_accuracy=test_accuracy/test_count


    print('Epoch: '+str(epoch)+' Train Loss: '+str(train_loss)+' Train Accurac

    #Save the best model
    if test_accuracy>best_accuracy:
        torch.save(model.state_dict(),'best_checkpoint.model')
        best_accuracy=test_accuracy
```

```
Epoch: 0 Train Loss: tensor(3.3721) Train Accuracy: 0.6284129228761014 Test A
ccuracy: 0.7495228471988323
Epoch: 1 Train Loss: tensor(0.4713) Train Accuracy: 0.8138801261829653 Test A
ccuracy: 0.9091725609071517
Epoch: 2 Train Loss: tensor(0.4813) Train Accuracy: 0.8305232241923203 Test A
ccuracy: 0.8132929156842933
Epoch: 3 Train Loss: tensor(0.4554) Train Accuracy: 0.8495594474056347 Test A
ccuracy: 0.9480184124845628
Epoch: 4 Train Loss: tensor(0.1532) Train Accuracy: 0.9452844555640161 Test A
ccuracy: 0.9779948355226227
Epoch: 5 Train Loss: tensor(0.0916) Train Accuracy: 0.9730229522462743 Test A
ccuracy: 0.9812507016952958
Epoch: 6 Train Loss: tensor(0.0580) Train Accuracy: 0.9840095725008159 Test A
ccuracy: 0.9910183002133154
Epoch: 7 Train Loss: tensor(0.0573) Train Accuracy: 0.9845534645926248 Test A
ccuracy: 0.9940496238913215
Epoch: 8 Train Loss: tensor(0.0409) Train Accuracy: 0.9905362776025236 Test A
ccuracy: 0.977658021780622
Epoch: 9 Train Loss: tensor(0.0484) Train Accuracy: 0.98607636244969 Test Acc
uracy: 0.9905692152239811
Epoch: 10 Train Loss: tensor(0.0263) Train Accuracy: 0.9945610790819102 Test
Accuracy: 0.9960705063433255
Epoch: 11 Train Loss: tensor(0.0188) Train Accuracy: 0.9974980963776787 Test
Accuracy: 0.995172336364657
Epoch: 12 Train Loss: tensor(0.0176) Train Accuracy: 0.9973893179593168 Test
Accuracy: 0.9964073200853262
Epoch: 13 Train Loss: tensor(0.0155) Train Accuracy: 0.9979332100511259 Test
Accuracy: 0.9976423038059953
```