In [3]:
```
#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 [4]:
```
#checking for device
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [5]:
```
print(device)
```

```
cpu
```

In [6]:
```
#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 [7]:
```
#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 [8]:
```
#categories
root=pathlib.Path(train_path)
classes=sorted([j.name.split('/')[-1] for j in root.iterdir()])
```

In [9]:
```
print(classes)
```

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

In [10]:
```
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 [11]:

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

In [12]:

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

In [13]:

```
num_epochs=12
```

In [14]:

```
#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 [15]:

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

```
9193 8907
```

In [16]:
```
#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 Accuracy

    #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(6.9940) Train Accuracy: 0.559991297726531 Test Ac
curacy: 0.5396878859324127
Epoch: 1 Train Loss: tensor(1.0258) Train Accuracy: 0.7183726748613075 Test A
ccuracy: 0.7512069159088357
Epoch: 2 Train Loss: tensor(0.4011) Train Accuracy: 0.8477102142934841 Test A
ccuracy: 0.9015381160884698
Epoch: 3 Train Loss: tensor(0.2029) Train Accuracy: 0.9269009028608725 Test A
ccuracy: 0.9660940833052655
Epoch: 4 Train Loss: tensor(0.2345) Train Accuracy: 0.9052539976068747 Test A
ccuracy: 0.8965981812057932
Epoch: 5 Train Loss: tensor(0.1804) Train Accuracy: 0.9316871532687915 Test A
ccuracy: 0.9731671718872796
Epoch: 6 Train Loss: tensor(0.0838) Train Accuracy: 0.9741107364298923 Test A
ccuracy: 0.9877624340406422
Epoch: 7 Train Loss: tensor(0.0552) Train Accuracy: 0.9873817034700315 Test A
ccuracy: 0.9885483327719771
Epoch: 8 Train Loss: tensor(0.0462) Train Accuracy: 0.9899923855107147 Test A
ccuracy: 0.9955091501066576
Epoch: 9 Train Loss: tensor(0.0371) Train Accuracy: 0.9934732948982922 Test A
ccuracy: 0.9958459638486583
Epoch: 10 Train Loss: tensor(0.0255) Train Accuracy: 0.9963015337756989 Test
Accuracy: 0.9977545750533289
Epoch: 11 Train Loss: tensor(0.0245) Train Accuracy: 0.9969542042858697 Test
Accuracy: 0.9978668463006624
```