Training

In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import TensorDataset, DataLoader, Dataset
from sklearn.utils import shuffle
```

In [2]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [3]:
```
import numpy as np
import random
import numpy.random as npr
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.image as img
from matplotlib import offsetbox
from PIL import Image
import shutil
import pandas as pd
import zipfile as zf
from tqdm.auto import tqdm
import yaml
import collections
import json
import random
import matplotlib.patches as patches
import pickle
import cv2
import time
import os
import copy
import math
from time import time
import warnings
%matplotlib inline
plt.style.use('bmh')
warnings.filterwarnings('ignore')
```

In [5]:
```
train_df = pd.read_csv('Train.csv')
train_df = train_df.dropna(how='any')
train_df = shuffle(train_df)
train_df= train_df.reset_index()
train_df = train_df.drop(columns=['index'])
train_df
```

Out[5]:

	Image	Label
0	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
1	/blue/eel6825/ravipatim/2015/preprocessed_data...	0
2	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
3	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
4	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
...	...	...
13745	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
13746	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
13747	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
13748	/blue/eel6825/ravipatim/2015/preprocessed_data...	1
13749	/blue/eel6825/ravipatim/2015/preprocessed_data...	1

13750 rows × 2 columns

In [6]:
```
vals, counts = np.unique(np.array(train_df['Label']), return_counts=True)

plt.bar(vals, counts)
plt.xticks(range(2),range(2))
plt.xlabel('Classes',size=20)
plt.ylabel('# Samples per Class', size=20)
plt.title('Training Data (Total = '+str(len(np.array(train_df['Image'])))+' samples)'
```

In [8]:
```
def load_images_labels(df):
    X , t =[],[]
    for i in range(len(df)):
        if df['Image'][i]:
            img = cv2.imread(df['Image'][i])
            X.append(np.array(img).reshape(3,300,300))
            t.append(int(df['Label'][i]))
        else:
            break
```

```
    print("Processed all the Images and their corresponding attributes")
    return np.array(X),np.array(t)
```

In [9]:
```
X,t = load_images_labels(train_df)
```

```
Processed all the Images and their corresponding attributes
```

In [10]:
```
X_train,X_test,y_train,y_test = train_test_split(X,t,shuffle=True,test_size=0.2,strati
```

Data Processing

In [11]:
```
transform_train = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

transform_test = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
```

In [12]:
```
train_data = TensorDataset(torch.Tensor(X_train), torch.LongTensor(y_train))
```

In [13]:
```
train_dataloader = DataLoader(train_data, batch_size=32, shuffle=True)
```

In [14]:
```
test_data = TensorDataset(torch.Tensor(X_test), torch.LongTensor(y_test))
```

In [15]:
```
test_dataloader = DataLoader(test_data, batch_size=32, shuffle=False)
```

Model Files

In [16]:
```
class ResNet101(nn.Module):
    def __init__(self, num_classes):
        super(ResNet101, self).__init__()
        self.resnet = nn.Sequential(*list(torchvision.models.resnet101(pretrained=True
        self.classifier = nn.Linear(2048, num_classes)

    def forward(self, x):
        x = self.resnet(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

In [17]:
```
def train_model(model,dataloader, criterion, optimizer, scheduler):
    model.train()
    total = 0
    running_loss = 0.0
    running_corrects = 0

    for images, labels in dataloader:
        inputs = images.to(device)
        labels = labels.to(device)
```

```
        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward + backward + optimize
        outputs = model(inputs)
        _, preds = torch.max(outputs.data, 1)
        loss = criterion(outputs, labels)
        total += labels.size(0)
        loss.backward()
        optimizer.step()

        # Print statistics
        running_loss += loss.item()
        running_corrects += (preds == labels).sum().item()

    scheduler.step()
    epoch_loss = running_loss / len(dataloader)
    epoch_acc = running_corrects / total
    return epoch_loss,epoch_acc
```

In [18]:
```
def validate_model(model,dataloader,criterion):
    model.eval()
    total = 0
    running_loss = 0.0
    running_corrects = 0

    with torch.no_grad():
        for images, labels in dataloader:
            inputs = images.to(device)
            labels = labels.to(device)
            # Forward
            outputs = model(inputs)
            _, preds = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)
            total += labels.size(0)

            # Print statistics
            running_loss += loss.item()
            running_corrects += (preds == labels).sum().item()

    epoch_loss = running_loss / len(dataloader)
    epoch_acc = running_corrects / total
    return epoch_loss,epoch_acc
```

Hyper Parameter Optimization

In [19]:
```
def model_optim(model, train_dataloader,val_dataloader,criterion, optimizer, scheduler
    ep,loss1,acc1,loss2,acc2 =[],[],[],[],[]
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):

        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
        ep.append(epoch)
```

```
        # model training
        train_loss,train_acc = train_model(model, train_dataloader, criterion, optimiz
        acc1.append(train_acc)
        loss1.append(train_loss)

        #model validation
        val_loss,val_acc = validate_model(model, val_dataloader, criterion)
        acc2.append(val_acc)
        loss2.append(val_loss)

        print(f"Train Accuracy: {train_acc:.4f}, Test Accuracy: {val_acc:.4f}")

        # saving a copy of best model parameters
        if  val_acc > best_acc:
            best_acc = val_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print()
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elaps
    print('Best Test Accuracy:{:4f}'.format(best_acc))
    # load best model weights
    model.load_state_dict(best_model_wts)
    return model,best_acc*100
```

In [20]:
```
import time
```

In [21]:
```
model = ResNet101(num_classes=2)

criterion = nn.CrossEntropyLoss()

model = model.to(device)
```

In [27]:
```
ne=[8,16]
lrv = [0.0001,0.001]
ss =[7,9]
param=[]
sc=[]
print("\n Performing Custom Search For Best Parameters")
since = time.time()
for i in range(len(ne)):
    for j in range(len(lrv)):
        for k in range(len(ss)):
          tmp =(ne[i],lrv[j],ss[k])
          print("\n Training a model with a learning rate of {0},step size {1} and an
          param.append(tmp)
          optimizer = optim.Adam(model.parameters(), lr=lrv[j])
          step_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=ss[k], gamma=0.
          model,acc = model_optim(model,train_dataloader,test_dataloader, criterion, c
          sc.append(acc)
bacc = max(sc)
loc = np.where(np.array(sc)==bacc)[0][0]
ne_v = param[loc][0]
lr_v = param[loc][1]
ss_v = param[loc][2]
time_elapsed = time.time() - since
print("The search for optimal parameters for the model has completed after:{:.0f}m {:.
print("\nThe optimal parameters are given as follows - learning rate:{0},step size:{1}
```

```
 Performing Custom Search For Best Parameters

 Training a model with a learning rate of 0.0001,step size 7 and an epoch count of 8
Epoch 0/7
----------
train Loss: 0.3504 Acc: 0.8643
test Loss: 0.2226 Acc: 0.9187

Epoch 1/7
----------
train Loss: 0.1303 Acc: 0.9540
test Loss: 0.1349 Acc: 0.9529

Epoch 2/7
----------
train Loss: 0.0844 Acc: 0.9717
test Loss: 0.1405 Acc: 0.9545

Epoch 3/7
----------
train Loss: 0.0636 Acc: 0.9794
test Loss: 0.1117 Acc: 0.9643

Epoch 4/7
----------
train Loss: 0.0498 Acc: 0.9842
test Loss: 0.0693 Acc: 0.9779

Epoch 5/7
----------
train Loss: 0.0419 Acc: 0.9869
test Loss: 0.1628 Acc: 0.9461

Epoch 6/7
----------
train Loss: 0.0354 Acc: 0.9892
test Loss: 0.0624 Acc: 0.9805

Epoch 7/7
----------
train Loss: 0.0096 Acc: 0.9972
test Loss: 0.0353 Acc: 0.9891

Training complete in 191m 44s
Best test Acc: 0.989062

 Training a model with a learning rate of 0.0001,step size 9 and an epoch count of 8
Epoch 0/7
----------
train Loss: 0.0321 Acc: 0.9906
test Loss: 0.0729 Acc: 0.9773

Epoch 1/7
----------
train Loss: 0.0301 Acc: 0.9904
test Loss: 0.0563 Acc: 0.9819

Epoch 2/7
----------
train Loss: 0.0250 Acc: 0.9927
```

```
test Loss: 0.0617 Acc: 0.9828

Epoch 3/7
----------
train Loss: 0.0247 Acc: 0.9926
test Loss: 0.0548 Acc: 0.9829

Epoch 4/7
----------
train Loss: 0.0217 Acc: 0.9937
test Loss: 0.0683 Acc: 0.9814

Epoch 5/7
----------
train Loss: 0.0215 Acc: 0.9934
test Loss: 0.0532 Acc: 0.9846

Epoch 6/7
----------
train Loss: 0.0193 Acc: 0.9947
test Loss: 0.0712 Acc: 0.9793

Epoch 7/7
----------
train Loss: 0.0177 Acc: 0.9951
test Loss: 0.0681 Acc: 0.9813

Training complete in 182m 51s
Best test Acc: 0.984553

 Training a model with a learning rate of 0.001,step size 7 and an epoch count of 8
Epoch 0/7
----------
train Loss: 0.2908 Acc: 0.8902
test Loss: 0.2228 Acc: 0.9233

Epoch 1/7
----------
train Loss: 0.1696 Acc: 0.9394
test Loss: 0.1346 Acc: 0.9529

Epoch 2/7
----------
train Loss: 0.1143 Acc: 0.9602
test Loss: 0.1179 Acc: 0.9594

Epoch 3/7
----------
train Loss: 0.0852 Acc: 0.9710
test Loss: 0.1519 Acc: 0.9473

Epoch 4/7
----------
train Loss: 0.0696 Acc: 0.9770
test Loss: 0.0978 Acc: 0.9702

Epoch 5/7
----------
train Loss: 0.0571 Acc: 0.9815
test Loss: 0.1023 Acc: 0.9704
```

```
Epoch 6/7
----------
train Loss: 0.0478 Acc: 0.9847
test Loss: 0.0743 Acc: 0.9768

Epoch 7/7
----------
train Loss: 0.0141 Acc: 0.9956
test Loss: 0.0433 Acc: 0.9868

Training complete in 184m 1s
Best test Acc: 0.986849

 Training a model with a learning rate of 0.001,step size 9 and an epoch count of 8
Epoch 0/7
----------
train Loss: 0.0417 Acc: 0.9872
test Loss: 0.0724 Acc: 0.9791

Epoch 1/7
----------
train Loss: 0.0359 Acc: 0.9886
test Loss: 0.0699 Acc: 0.9770

Epoch 2/7
----------
train Loss: 0.0324 Acc: 0.9898
test Loss: 0.0556 Acc: 0.9821

Epoch 3/7
----------
train Loss: 0.0303 Acc: 0.9908
test Loss: 0.0718 Acc: 0.9785

Epoch 4/7
----------
train Loss: 0.0267 Acc: 0.9924
test Loss: 0.0643 Acc: 0.9798

Epoch 5/7
----------
train Loss: 0.0241 Acc: 0.9926
test Loss: 0.1012 Acc: 0.9708

Epoch 6/7
----------
train Loss: 0.0225 Acc: 0.9933
test Loss: 0.0538 Acc: 0.9832

Epoch 7/7
----------
train Loss: 0.0215 Acc: 0.9936
test Loss: 0.0558 Acc: 0.9840

Training complete in 182m 47s
Best test Acc: 0.983969

 Training a model with a learning rate of 0.0001,step size 7 and an epoch count of 16
Epoch 0/15
```

```
----------
train Loss: 0.0056 Acc: 0.9981
test Loss: 0.0390 Acc: 0.9897

Epoch 1/15
----------
train Loss: 0.0016 Acc: 0.9996
test Loss: 0.0438 Acc: 0.9896

Epoch 2/15
----------
train Loss: 0.0014 Acc: 0.9996
test Loss: 0.0445 Acc: 0.9894

Epoch 3/15
----------
train Loss: 0.0010 Acc: 0.9997
test Loss: 0.0529 Acc: 0.9884

Epoch 4/15
----------
train Loss: 0.0009 Acc: 0.9997
test Loss: 0.0479 Acc: 0.9896

Epoch 5/15
----------
train Loss: 0.0006 Acc: 0.9998
test Loss: 0.0501 Acc: 0.9899

Epoch 6/15
----------
train Loss: 0.0005 Acc: 0.9998
test Loss: 0.0521 Acc: 0.9894

Epoch 7/15
----------
train Loss: 0.0003 Acc: 0.9999
test Loss: 0.0529 Acc: 0.9897

Epoch 8/15
----------
train Loss: 0.0001 Acc: 1.0000
test Loss: 0.0553 Acc: 0.9897

Epoch 9/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0541 Acc: 0.9896

Epoch 10/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0588 Acc: 0.9895

Epoch 11/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0608 Acc: 0.9897

Epoch 12/15
```

```
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0602 Acc: 0.9897

Epoch 13/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0625 Acc: 0.9898

Epoch 14/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0633 Acc: 0.9896

Epoch 15/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0608 Acc: 0.9897

Training complete in 370m 23s
Best test Acc: 0.989939

 Training a model with a learning rate of 0.0001,step size 9 and an epoch count of 16
Epoch 0/15
----------
train Loss: 0.0006 Acc: 0.9998
test Loss: 0.0544 Acc: 0.9894

Epoch 1/15
----------
train Loss: 0.0004 Acc: 0.9999
test Loss: 0.0568 Acc: 0.9896

Epoch 2/15
----------
train Loss: 0.0004 Acc: 0.9999
test Loss: 0.0542 Acc: 0.9898

Epoch 3/15
----------
train Loss: 0.0004 Acc: 0.9999
test Loss: 0.0562 Acc: 0.9892

Epoch 4/15
----------
train Loss: 0.0004 Acc: 0.9999
test Loss: 0.0573 Acc: 0.9892

Epoch 5/15
----------
train Loss: 0.0002 Acc: 0.9999
test Loss: 0.0566 Acc: 0.9899

Epoch 6/15
----------
train Loss: 0.0002 Acc: 0.9999
test Loss: 0.0578 Acc: 0.9896

Epoch 7/15
----------
```

```
train Loss: 0.0002 Acc: 1.0000
test Loss: 0.0661 Acc: 0.9890

Epoch 8/15
----------
train Loss: 0.0001 Acc: 1.0000
test Loss: 0.0643 Acc: 0.9894

Epoch 9/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0675 Acc: 0.9894

Epoch 10/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0663 Acc: 0.9895

Epoch 11/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0682 Acc: 0.9895

Epoch 12/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0714 Acc: 0.9895

Epoch 13/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0733 Acc: 0.9894

Epoch 14/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0723 Acc: 0.9895

Epoch 15/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0754 Acc: 0.9894

Training complete in 373m 42s
Best test Acc: 0.989897

 Training a model with a learning rate of 0.001,step size 7 and an epoch count of 16
Epoch 0/15
----------
train Loss: 0.0169 Acc: 0.9952
test Loss: 0.0628 Acc: 0.9833

Epoch 1/15
----------
train Loss: 0.0187 Acc: 0.9945
test Loss: 0.0602 Acc: 0.9840

Epoch 2/15
----------
train Loss: 0.0161 Acc: 0.9953
```

```
test Loss: 0.0577 Acc: 0.9828

Epoch 3/15
----------
train Loss: 0.0168 Acc: 0.9952
test Loss: 0.0603 Acc: 0.9825

Epoch 4/15
----------
train Loss: 0.0151 Acc: 0.9957
test Loss: 0.0596 Acc: 0.9845

Epoch 5/15
----------
train Loss: 0.0148 Acc: 0.9957
test Loss: 0.0521 Acc: 0.9861

Epoch 6/15
----------
train Loss: 0.0136 Acc: 0.9961
test Loss: 0.0523 Acc: 0.9847

Epoch 7/15
----------
train Loss: 0.0043 Acc: 0.9988
test Loss: 0.0413 Acc: 0.9895

Epoch 8/15
----------
train Loss: 0.0009 Acc: 0.9998
test Loss: 0.0457 Acc: 0.9898

Epoch 9/15
----------
train Loss: 0.0007 Acc: 0.9998
test Loss: 0.0480 Acc: 0.9898

Epoch 10/15
----------
train Loss: 0.0007 Acc: 0.9998
test Loss: 0.0517 Acc: 0.9891

Epoch 11/15
----------
train Loss: 0.0004 Acc: 0.9999
test Loss: 0.0513 Acc: 0.9895

Epoch 12/15
----------
train Loss: 0.0005 Acc: 0.9999
test Loss: 0.0523 Acc: 0.9895

Epoch 13/15
----------
train Loss: 0.0002 Acc: 0.9999
test Loss: 0.0535 Acc: 0.9892

Epoch 14/15
----------
train Loss: 0.0002 Acc: 0.9999
```

```
test Loss: 0.0531 Acc: 0.9896

Epoch 15/15
----------
train Loss: 0.0000 Acc: 1.0000
test Loss: 0.0539 Acc: 0.9896

Training complete in 369m 3s
Best test Acc: 0.989813

 Training a model with a learning rate of 0.001,step size 9 and an epoch count of 16
Epoch 0/15
----------
train Loss: 0.0112 Acc: 0.9969
test Loss: 0.0582 Acc: 0.9852

Epoch 1/15
----------
train Loss: 0.0120 Acc: 0.9967
test Loss: 0.0480 Acc: 0.9861

Epoch 2/15
----------
train Loss: 0.0126 Acc: 0.9966
test Loss: 0.0494 Acc: 0.9865

Epoch 3/15
----------
train Loss: 0.0116 Acc: 0.9970
test Loss: 0.0496 Acc: 0.9857

Epoch 4/15
----------
train Loss: 0.0107 Acc: 0.9973
test Loss: 0.0504 Acc: 0.9856

Epoch 5/15
----------
train Loss: 0.0097 Acc: 0.9975
test Loss: 0.0489 Acc: 0.9862

Epoch 6/15
----------
train Loss: 0.0111 Acc: 0.9971
test Loss: 0.0482 Acc: 0.9866

Epoch 7/15
----------
train Loss: 0.0097 Acc: 0.9973
test Loss: 0.0473 Acc: 0.9865

Epoch 8/15
----------
train Loss: 0.0089 Acc: 0.9975
test Loss: 0.0513 Acc: 0.9857

Epoch 9/15
----------
train Loss: 0.0034 Acc: 0.9992
test Loss: 0.0402 Acc: 0.9901
```

```
Epoch 10/15
----------
train Loss: 0.0008 Acc: 0.9998
test Loss: 0.0453 Acc: 0.9905

Epoch 11/15
----------
train Loss: 0.0002 Acc: 0.9999
test Loss: 0.0517 Acc: 0.9901

Epoch 12/15
----------
train Loss: 0.0004 Acc: 0.9999
test Loss: 0.0500 Acc: 0.9904

Epoch 13/15
----------
train Loss: 0.0001 Acc: 0.9999
test Loss: 0.0532 Acc: 0.9904

Epoch 14/15
----------
train Loss: 0.0002 Acc: 0.9999
test Loss: 0.0586 Acc: 0.9900

Epoch 15/15
----------
train Loss: 0.0002 Acc: 1.0000
test Loss: 0.0565 Acc: 0.9905

Training complete in 369m 37s
Best test Acc: 0.990481
The search for optimal parameters for the model has completed after:2224m 8s

The optimal parameters are given as follows - learning rate:0.001,step size:7 and a f
ixed number of epochs:16
```

Final Model Training

In [22]:

```
def model_main(model, train_dataloader,val_dataloader,criterion, optimizer, scheduler,
    ep,loss1,acc1,loss2,acc2 =[],[],[],[],[]
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):

        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
        ep.append(epoch)
        # model training
        train_loss,train_acc = train_model(model, train_dataloader, criterion, optimiz
        acc1.append(train_acc)
        loss1.append(train_loss)

        #model validation
        val_loss,val_acc = validate_model(model, val_dataloader, criterion)
```

```
        acc2.append(val_acc)
        loss2.append(val_loss)

        print(f"Train Accuracy: {train_acc:.4f}, Test Accuracy: {val_acc:.4f}")

        # saving a copy of best model parameters
        if  val_acc > best_acc:
            best_acc = val_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print()
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elaps
    print('Best Test Accuracy:{:4f}'.format(best_acc))
    # load best model weights
    model.load_state_dict(best_model_wts)
    return model,acc1,acc2,np.array(loss1),np.array(loss2),np.array(ep)
```

In [23]:
```
import time
```

In [25]:
```
model = ResNet101(num_classes=2)

criterion = nn.CrossEntropyLoss()

model = model.to(device)

optimizer = optim.Adam(model.parameters(), lr=0.001)

step_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=ss_v, gamma=0.1)

model,acc1,acc2,loss1,loss2,epochs = model_main(model,train_dataloader,test_dataloader
```

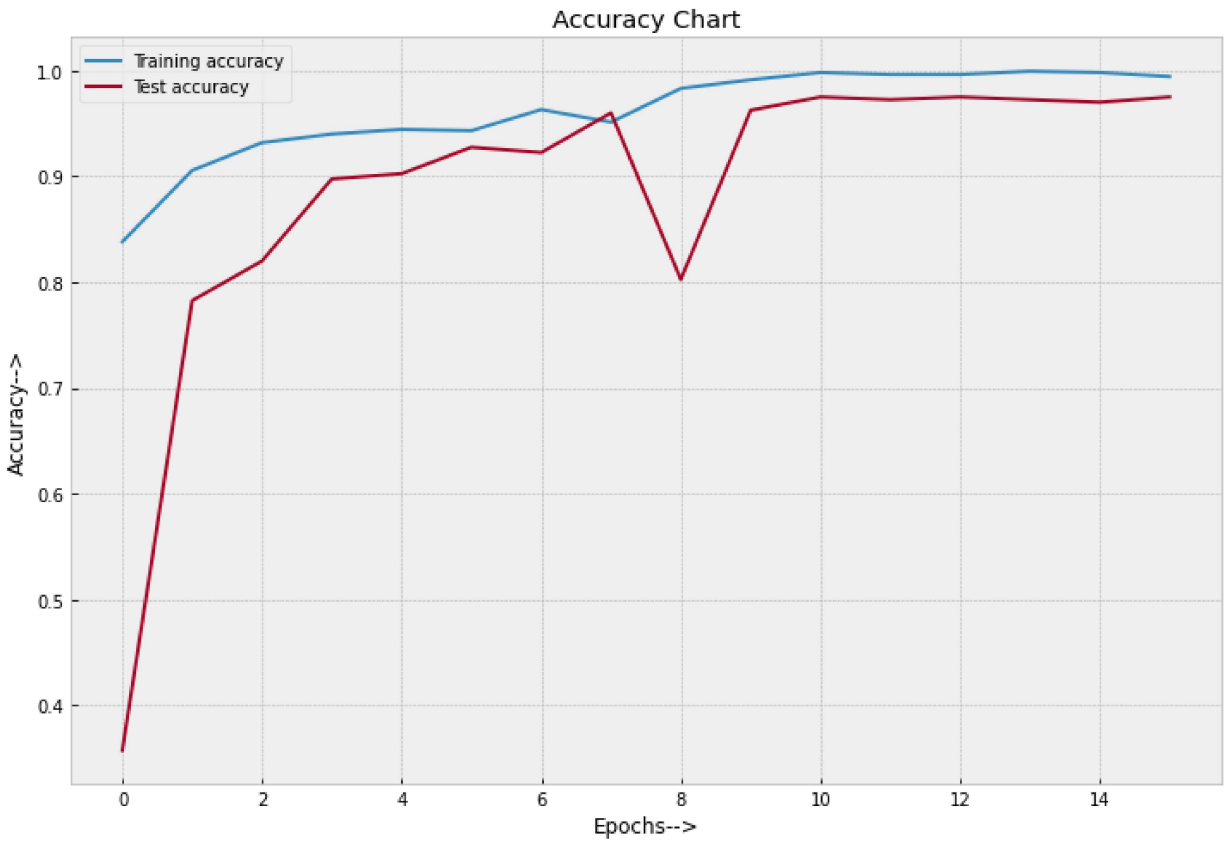
```
Epoch 0/15
----------
Train Accuracy: 0.8381, Test Accuracy: 0.3575
Epoch 1/15
----------
Train Accuracy: 0.9056, Test Accuracy: 0.7825
Epoch 2/15
----------
Train Accuracy: 0.9319, Test Accuracy: 0.8200
Epoch 3/15
----------
Train Accuracy: 0.9400, Test Accuracy: 0.8975
Epoch 4/15
----------
Train Accuracy: 0.9444, Test Accuracy: 0.9025
Epoch 5/15
----------
Train Accuracy: 0.9431, Test Accuracy: 0.9275
Epoch 6/15
----------
Train Accuracy: 0.9631, Test Accuracy: 0.9225
Epoch 7/15
----------
Train Accuracy: 0.9513, Test Accuracy: 0.9600
Epoch 8/15
----------
Train Accuracy: 0.9831, Test Accuracy: 0.8025
Epoch 9/15
----------
Train Accuracy: 0.9912, Test Accuracy: 0.9625
Epoch 10/15
----------
Train Accuracy: 0.9981, Test Accuracy: 0.9750
Epoch 11/15
----------
Train Accuracy: 0.9962, Test Accuracy: 0.9725
Epoch 12/15
----------
Train Accuracy: 0.9962, Test Accuracy: 0.9750
Epoch 13/15
----------
Train Accuracy: 0.9994, Test Accuracy: 0.9725
Epoch 14/15
----------
Train Accuracy: 0.9981, Test Accuracy: 0.9700
Epoch 15/15
----------
Train Accuracy: 0.9944, Test Accuracy: 0.9750

Training complete in 3m 33s
Best Test Accuracy:0.975000
```

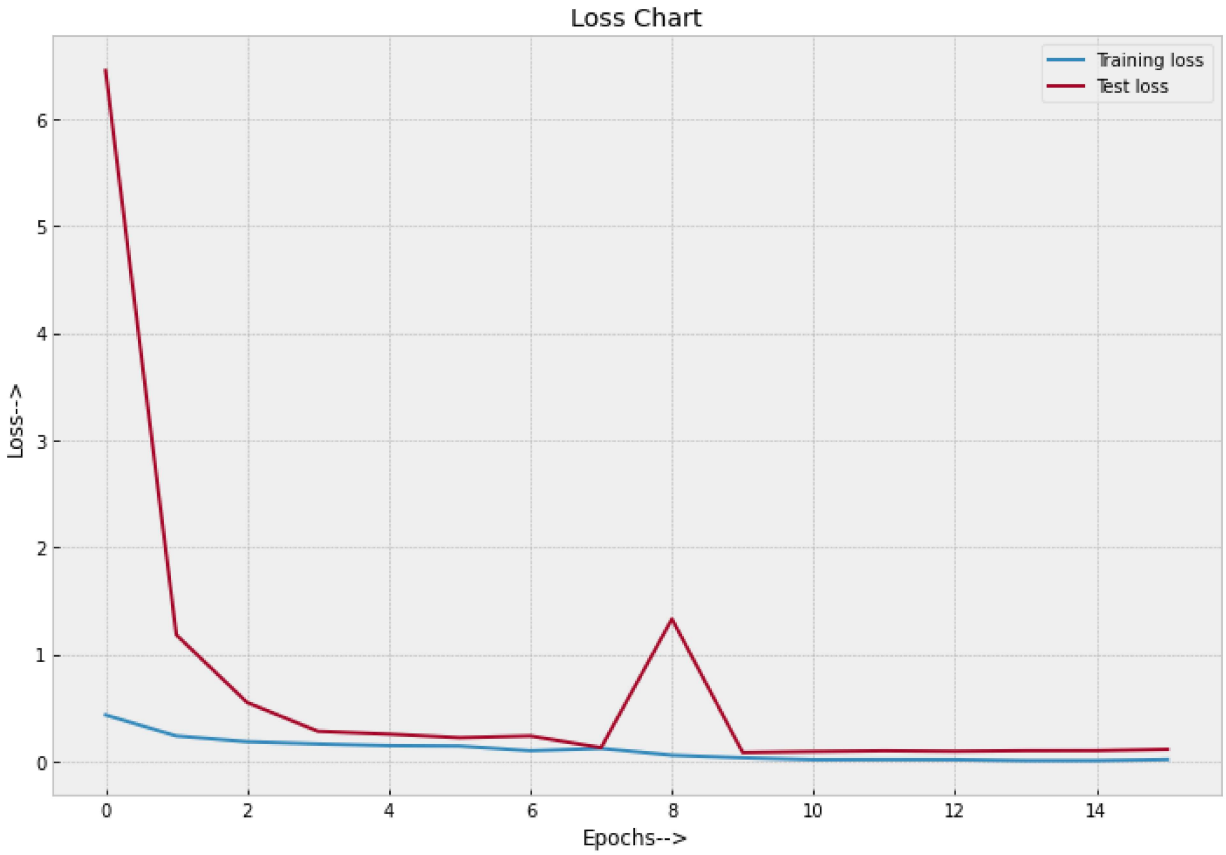
In [26]:

```
plt.figure(figsize =(12, 8))
plt.grid(visible=True)
plt.plot(epochs,acc1, label='Training accuracy')
plt.plot(epochs,acc2, label='Test accuracy')
plt.title('Accuracy Chart')
plt.xlabel('Epochs-->')
plt.ylabel('Accuracy-->')
```

```
plt.legend()
plt.savefig('Accuracy.png');
```



In [27]:
```
plt.figure(figsize =(12, 8))
plt.grid(visible=True)
plt.plot(epochs,loss1, label='Training loss')
plt.plot(epochs,loss2, label='Test loss')
plt.title('Loss Chart')
plt.xlabel('Epochs-->')
plt.ylabel('Loss-->')
plt.legend()
plt.savefig('Loss.png');
```



In [28]:
```
print('Finished Training')
device = torch.device("cuda:0")
model.to(device)
PATH = './FPLD-ResNet-101-CLS.pth'
torch.save(model, PATH)
```

```
Finished Training
```