

CS634101 – DATA MINING

PROJECT: LUNG CANCER
DETECTION SYSTEM

Final Report

Saketh Reddy Garlapati (sg2485@njit.edu)
Olive Satya Priya Gundla(og84@njit.edu)

Department of Computer science
New Jersey Institute of Technology (NJIT)
Newark, New Jersey -07102

Abstract

Detecting malignant lung nodules from Histopathological Lung Tissue images is a hard and time-consuming task for radiologists. To alleviate this burden, computer- aided diagnosis (CAD) systems have been proposed. In recent years, deep learning approaches have shown impressive results outperforming classical methods in various fields. Nowadays, researchers are trying different deep learning techniques to increase the performance of CAD systems in lung cancer screening. In this work, we review recent state-of-the-art deep learning algorithms and architectures proposed as CAD systems for lung cancer detection. Recently, image processing techniques are widely used in several medical areas for image improvement in earlier detection and treatment stages, where the time factor is very important to discover the abnormality issues in target images, especially in various cancer tumours such as lung cancer, breast cancer, etc.

The human radiologists have to distinguish lung lesions from bones, pulmonary veins and other complex structures in the lung cavity this may lead to errors in diagnosis of lung cancer. The error in diagnosis of lung cancer can lead to legal issues and also unnecessary medical expenses. The observer may overlook small lesions in lungs which are cancerous. In order to overcome the human error and biases a computer based diagnostic system can be designed which uses high-end image processing algorithms and deep convolutional networks that uses lung tissue images to predict the occurrence of cancer. This project makes use of transfer learning a machine learning method which saves time and also uses limited computing power.

Introduction

Lung cancer is considered as the deadliest cancer worldwide. It seems to be the common cause of death among people throughout the world. Early detection of lung cancer can increase the chance of survival among people. The overall 5-year survival rate for lung cancer patients increases from 14 to 49% if the disease is detected in time. The abnormal growth of cells causes cancer or tumour. The body part where these abnormal cells are present decides the type of cancer. This abnormal cell growth in lungs are called lung nodules, lung nodules are small masses of tissue in the lungs. The tumour can be malignant or benign. Benign tumours are noncancerous if they are of normal size, they aren't harmful, they do not spread to other tissues or organs. If the benign tumour grows for some reason, it can be dangerous. Malignant tumours are cancerous they spread to other parts of the body. The tumours appear as round, white shadow in the CT scan images. Lung nodules are usually 0.2 inch (5 millimetres) to 1.2 inches (30 millimetres) in size. Based on the appearance of tumour cells in microscope, lung cancer can be classified as small cell lung cancer (SCLC) and Non-small cell lung cancer (NSCLC). These 2 types of lung cancer spreads and are treated in different way so making this distinction is important. SCLC account for about 10%-15% of lung cancer. This is the most aggressive type of lung cancer as it grows rapidly. SCLC is caused strongly due to cigarette smoking. SCLC rapidly grows to many parts in the body and is discovered after extensive spreading. SCLC is difficult to detect as the tumour cells will be very small and they resemble normal cells. NSCLC is the most common type of lung cancer accounting for 75-80% of the total lung cancer cases. Symptoms of Lung cancer:

1. Loss of appetite
2. Wheezing
3. Tiredness or weakness
4. Pain in the chest, shoulders, back
5. Cough producing blood
6. Swelling in the face and neck

With the expected increase in the number of preventive/early-detection measures, scientists are working in computerized solutions that help alleviate the work of doctors, improve diagnostics' precision by reducing the subjectivity factor, speedup the analysis and reduce medical costs.

Based on the 10000 projects in 2017 alone, approximately 18.1 million rare new types of disease cancer occurred globally, and this caused the deaths approximately 15.6% of the death . 520 individuals having between one as well as eight pulmonary nodules. Some of them, thirty-one with nodules very improbable of being malignant ,from the Figure 1 we can clearly state that 64% with nodules sensibly improbable of being malignant, 149 with nodules with undetermined malignant cells, 78% with nodules moderately suspicious of being malevolent and 62% with nodules very suspicious of being malevolent.

Tests to diagnose lung cancer are as follows: Imaging tests: The X-Ray image of a chest cavity can reveal abnormal mass of tissues in the lungs and CT scans can be used to determine small lesions in the lungs which cannot be detected by X-Ray. Sputum Cytology: The patient who has cough and is producing sputum, the observation of the sputum under the microscope can reveal this cancerous cells in the lungs Biopsy: The sample of abnormal cells in the lungs can be removed from the lungs and careful analysis in the lab can help in the detection of lung cancer The human radiologists have to distinguish lung lesions from bones, pulmonary veins and other complex structures in the lung cavity this may lead to errors in diagnosis of lung cancer. The error in diagnosis of lung cancer can lead to legal issues and also unnecessary medical expenses. The observer may overlook small lesions in lungs which are cancerous. In order to overcome the human error and biases a computer based diagnostic system can be designed which uses high-end image processing algorithms and deep convolutional networks that uses chest CT scan images to predict the occurrence of cancer.

Problem Statement:

Lung cancer detection – Detection of Lung cancer at an early stage is crucial and an effective solution. Therefore, an efficient and accurate predictive model is required to detect the cancer. In order to detect malignant nodules, specific features need to be recognized and measured. Based on the detected features and their combination, cancer probability can be assessed. However, this task is very difficult, even for an experienced medical doctor, since nodule presence and positive cancer diagnosis are not easily related. Common computer aided diagnosis (CAD) approaches use previously studied features which are somehow related to cancer suspiciousness, such as volume, shape, subtlety, solidity, spiculation, sphericity, among others.

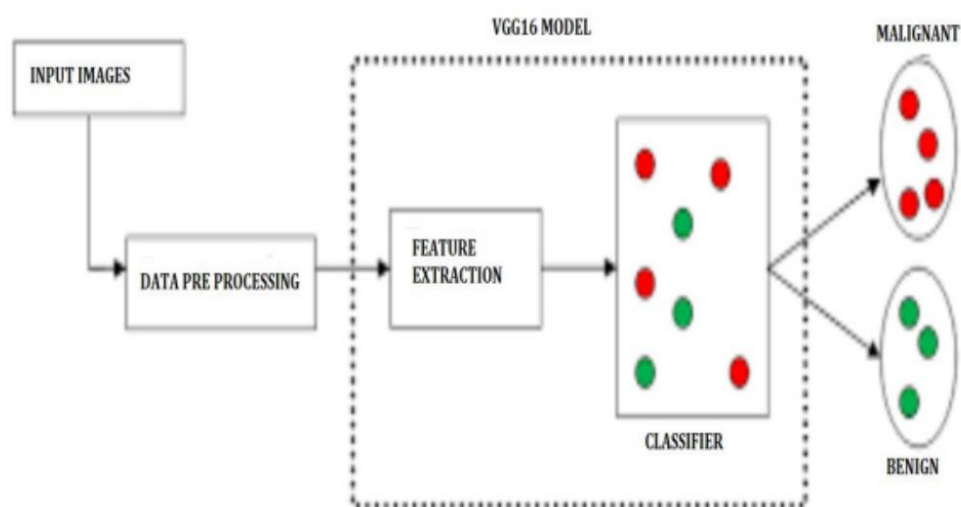
Implementation (Methodology)

Proposed System:

Training a Neural network from scratch requires large dataset and also training time will be high. Instead of doing this we can use transfer learning where we transfer the weights which is already learned by the network and use it to solve the target problem. Transfer learning is a machine learning methodology where knowledge gained when solving one problem is stored and it is applied to some other related problem. The knowledge here is nothing but a model which was developed for that problem along with the pretrained weights. Here we use models created by others as starting point to create model for our problem by making changes to the chosen model. Transfer learning is used for computer vision and natural language processing applications which requires heavy computing power. The main idea here is to use pretrained models which are previously trained on large dataset using high end GPUs. Pretrained models are models which are developed by someone else to solve some problem using large dataset and high computing power. Keras library has this pretrained models built in which comes with the weights and biases. These pretrained models are previously trained network, which is trained on large dataset to be used in multi class classification. Some of the commonly available pretrained models are VGG16, VGG19, Inception V3, Mobilenet, Resnet

50 etc. Pretrained models may not be 100% accurate but it saves lot of time required for training. This pretrained model can be used in an application by making some changes to it. Either architecture of the pretrained model can be chosen by initializing all the weights randomly or training the model from scratch using the dataset. This pretrained model can be used as a feature extractor by removing the output layer; it could extract some basic features like edges, curves, shapes and lines etc. User can also partially train the model by freezing some layers and fine tuning the others. Many pretrained model architecture are readily available on Keras library. These pretrained models are trained on Imagenet which is a database of millions of images. These pretrained architectures are capable of classifying 1000 different categories. The convolution layers in the pretrained models are used for feature extraction and initial layers in the pretrained models are composed of convolutional layers. These convolutional layers can be used to extract features in the above application, so this layer is freezed so that same weights can be used for extracting features. These layers are capable of detecting edges and shapes so the same layers can be utilized for feature extraction in cancer detection. The VGG16 model is pretrained on large image dataset we can use the pretrained weights which are learned during the training process. The initial few layers of vgg16 model are used for detecting the basic features, middle conv layers are used for detection of some shapes and top layers are used for some complex feature extraction and classification. We will freeze all the layers in vgg16 except the last 3 layers so that the frozen layers won't be retrained and the pretrained weights can be retained. The output layer in vgg16 can be removed since it has output shape of 1000 i.e., it can classify 1000 classes. In our application of lung cancer detection, we only want to classify images belonging to 2 classes we can replace the output layer with the dense layer of shape 2 which can be used for classifying the images into cancerous and non-cancerous. Since we are freezing only the bottom layers and fine tuning the top layers, we do not need large dataset and also training time will also be less compared to training the CNN from scratch.

System Architecture:



Architecture of the Model

Methodology:

We are using Kaggle which is a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges. We are also using PyTorch which is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- Deep neural networks built on a type-based automatic differentiation system

PyTorch integrates acceleration libraries such as Intel MKL (Math Kernel Library) and the Nvidia cuDNN (CUDA Deep Neural Network) and NCCL (Nvidia Collective Communications) libraries to maximize speed. Its core CPU and GPU tensor and neural network back ends—TH (Torch), THC (Torch CUDA), THNN (Torch Neural Network), and THCUNN (Torch CUDA Neural Network)—are written as independent libraries with a C99 API. At the same time, PyTorch is not a Python binding into a monolithic C++ framework, but designed to be deeply integrated with Python and to allow the use of other Python libraries. The memory usage in PyTorch is efficient compared to Torch and some of the alternatives. One of the optimizations is a set of custom memory allocators for the GPU, since available GPU memory can often limit the size of deep learning models that can be solved at GPU speeds. CUDA is Nvidia's API for its general purpose GPUs. GPUs are much faster than CPUs for training and making predictions from deep neural networks; so are Google's TPUs (tensor processing units) and FPGAs (field programmable gate arrays), which are available for use on AWS, Microsoft Azure, and elsewhere. In some cases, the use of advanced chips (GPUs, TPUs, or FPGAs) can speed up computations over CPUs by 50x per chip used, reducing training times from weeks to hours or from hours to minutes.

Pytorch Optimizers:

Most of the weight update rules (optimizers) used to find the minimum error take the gradient of the loss function as the initial direction to change the values for the next step, multiplied by a small learning rate to reduce the magnitude of the step. The basic algorithm is called steepest descent. For machine learning, the usual variant is stochastic gradient descent, or SGD, which uses multiple batches of data points and often goes through the data multiple times (epochs). More sophisticated versions of stochastic gradient descent, for example Adam and RMSprop, may compensate for biases, fold in momentum and velocity with the gradient,

average gradients, or use adaptive learning rates. PyTorch currently supports 10 optimization methods.

Pytorch Neural Networks:

The `torch.nn` class defines modules and other containers, module parameters, 11 kinds of layers, 17 loss functions, 20 activation functions, and two kinds of distance functions. Each kind of layer has many variants, for example six convolution layers and 18 pooling layers.

The `torch.nn.functional` class defines 11 categories of functions. Somewhat confusingly, both `torch.nn` and `torch.nn.functional` contain loss and activation member functions. In many cases, however, the `torch.nn` member is little more than a wrapper for the corresponding `torch.nn.functional` member.

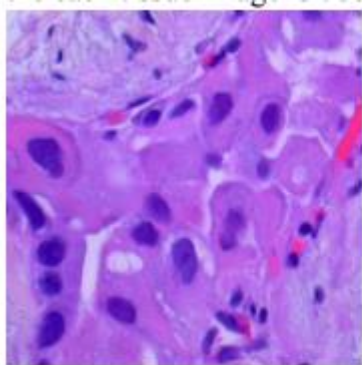
Training a Neural network from scratch requires large dataset and also training time will be high. Instead of doing this we can use transfer learning where we transfer the weights which is already learned by the network and use it to solve the target problem. Transfer learning is a machine learning methodology where knowledge gained when solving one problem is stored and it is applied to some other related problem. The knowledge here is nothing but a model which was developed for that problem along with the pretrained weights. Here we use models created by others as starting point to create model for our problem by making changes to the chosen model. Transfer learning is used for computer vision and natural language processing applications which requires heavy computing power. The main idea here is to use pretrained models which are previously trained on large dataset using high end GPUs.

Pretrained models are models which are developed by someone else to solve some problem using large dataset and high computing power. PyTorch library has this pretrained models built in which comes with the weights and biases. These pretrained models are previously trained network, which is trained on large dataset to be used in multi class classification. Some of the commonly available pretrained models are VGG16, VGG19, Inception V3, Mobilenet, Resnet 50 etc. Pretrained models may not be 100% accurate but it saves lot of time required for training. This pretrained model can be used in an application by making some changes to it. Either architecture of the pretrained model can be chosen by initializing all the weights randomly or training the model from scratch using the dataset. This pretrained model can be used as a feature extractor by removing the output layer; it could extract some basic features like edges, curves, shapes and lines etc. User can also partially train the model by freezing some layers and fine tuning the others. Many pretrained model architecture are readily available on PyTorch library. These pretrained models are trained on Imagenet which is a database of millions of images. These pretrained architectures are capable of classifying 1000 different categories. The convolution layers in the pretrained models are used for feature extraction and initial layers in the pretrained models are composed of convolutional layers. These convolutional layers can be used to extract features in the above application, so this layer is freezed so that same weights can be used for extracting features. These layers are capable of detecting edges and shapes so the same layers can be utilized for feature extraction in cancer detection. VGG16 takes a input image of fixed size i.e. 224X224 RGB image. Here image is passed through stack of convolutional layers with a filter of size 3X3 and convolutional stride is fixed to one pixel. Maxpooling layers is used to perform spatial pooling which follows some of the convolutional layers it uses a window of size 2X2 with stride two. The stack of convolutional layers are followed by 3 fully connected layers, the final

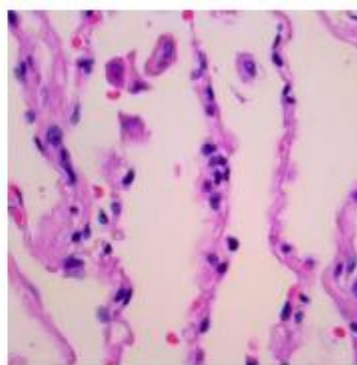
fully connected layer acts as a output layer which can classify 1000 classes hence it has 1000 channels. The Pooling and flattening layers does not learn anything so it has 030 trainable parameters. The pooling layer has a window size of 2X2 with stride 2; hence it outputs a pixel for every 4 pixels and jumps by 2 pixels to do the next calculations. The VGG16 model is pre-trained on large image dataset we can use the pre-trained weights which are learned during the training process. The initial few layers of vgg16 model is used for detecting the basic features, middle conv layers are used for detection of some shapes and top layers are used for some complex feature extraction and classification. We will freeze all the layers in vgg16 except the last 3 layers so that the freezed layers won't be retrained and the pre-trained weights can be retained. The output layer in vgg16 can be removed since it has output shape of 1000 i.e. it can classify 1000 classes. In our application of lung cancer detection we only want to classify images belonging to 3 classes we can replace the output layer with the dense layer of shape 2 which can be used for classifying the images into cancerous and non-cancerous. Since we are freezing only the bottom layers and fine tuning the top layers we do not need large dataset and also training time will also be less compared to training the CNN from scratch.

Implementation

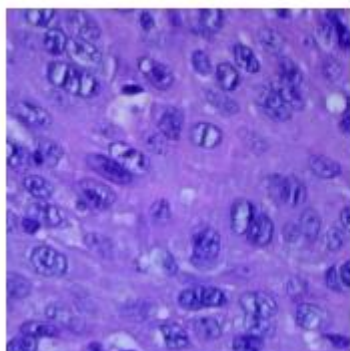
The model implementation is initiated by acquiring the dataset required for the project. The dataset we acquired is the Lung cancer histopathological images which contains three classes that are the lung adenocarcinoma tissues, lung benign tissues and Lung squamous cell carcinoma tissues where the first class and the last class tissues are the cancerous tissues and the lung benign tissues are the non-cancerous tissues. The size of the each tissue image is 768*768 pixels.



Lung adenocarcinoma tissue



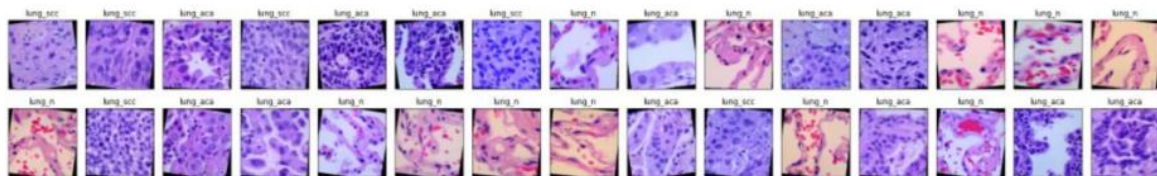
Lung benign tissue



Lung squamous cell carcinoma

This data undergoes various pre-processing methods of torch library like resize, centre crop, random horizontal flip, random rotation, color jitter, to Tensor to make data effective and usable for the implementation of the model.

After the data pre-processing we visualize the data using matplotlib library. This step done to simply interpret research and easily leverage their information. It also allows health care professionals to access the data of other patients with similar symptoms to understand common timelines for recovery.



Data visualization of pre-processed data

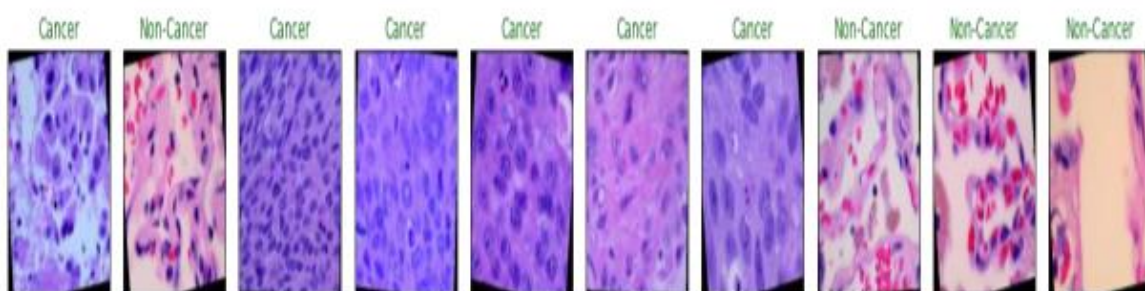
After the data visualization we import and load the model into our environment which is a pre-trained Transfer learning algorithm i.e.; VGG16 model. Then we alter the out parameters according to our model requirements .

Now the model is pushed onto the GPU using cuda and then it is trained on training dataset(which is obtained by splitting the dataset into train dataset, validation dataset and testing dataset into 8:1:1 ratio) an then it is evaluated by using the validation dataset. During the training and evaluating the model we calculate the loss and optimize the model using the Adam optimizer. At last we calculate the average train loss and average valid loss and print them.

```
Epoch : 1 , Batch : 49 , Loss : 0.19250731768753426
Valid Loss decreased from inf ----> 0.09685513941571117
```

Training and validation loss

After training the model we test the model and print the accuracy. Then the visualization of the output is done using matplotlib library where the tissues images are predicted as the cancerous or non-cancerous tissues. The correctly predicted output are titled in green colour and the wrong predictions as titled using red colour.



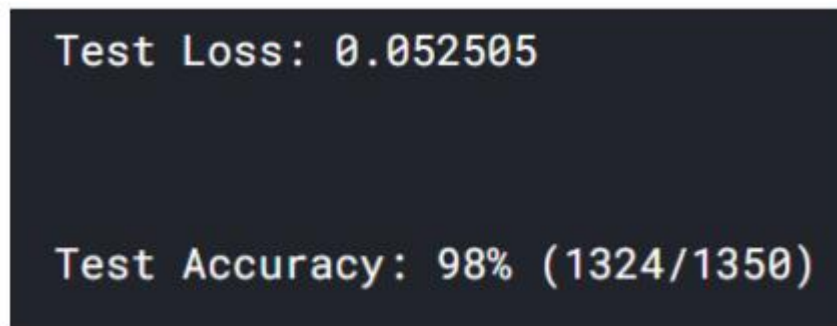
Data visualization of the output

Testing:

The testing of the model is done by defining a test method which accept the parameters such as the model, criterion, cuda, etc. Initially the test method pushes the model and test data (which is acquired by splitting the dataset into training dataset, validation dataset and testing dataset) into the GPU using cuda. Along with the data the target output is also loaded into GPU to verify the correctness of the predictions. Once we have the data and model we predict the output for the given test data and then we calculate the loss for the test dataset. Once the prediction is done we verify each data output to their target output and count number of correct predicitions. Then we calculate the accuracy of the model using the formula

$$\text{Accuracy} = (\text{correct prediction} / \text{total predicitions}) * 100$$

After which we print the test loss and the accuracy of the model.

A terminal window with a dark background and light-colored text. The first line displays 'Test Loss: 0.052505' and the second line displays 'Test Accuracy: 98% (1324/1350)'.

```
Test Loss: 0.052505  
  
Test Accuracy: 98% (1324/1350)
```

Test loss and accuracy

Experiment Analysis:

Code:

```
import numpy as np  
import matplotlib.pyplot as plt  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
import torchvision  
import torchvision.transforms as transforms  
import torchvision.datasets as dataset
```

```

from torch.utils.data import DataLoader

transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(),
    transforms.ToTensor()
])

Data = dataset.ImageFolder('../input/lung-and-colon-cancer-histopathological-
images/lung_colon_image_set/lung_image_sets' , transform = transform)

datalen = len(data)

dataidx = np.array(list(range(datalen)))
np.random.shuffle(dataidx)

splitfrac = 0.9

split_idx = int(splitfrac * datalen)

train_idx = dataidx[:split_idx]
valid_idx = dataidx[split_idx:]

testsplit = 0.1

testidx = int(testsplit * len(train_idx))

test_idx = train_idx[:testidx]
train_idx = train_idx[testidx:]

np.random.shuffle(test_idx)

batch_size = 30

train_samples = torch.utils.data.SubsetRandomSampler(train_idx)
valid_samples = torch.utils.data.SubsetRandomSampler(valid_idx)
test_samples = torch.utils.data.SubsetRandomSampler(test_idx)

dataloader = DataLoader(data , batch_size = batch_size , sampler = train_samples)
validloader = DataLoader(data , batch_size = batch_size , sampler = valid_samples)
testloader = DataLoader(data , batch_size = batch_size , sampler = test_samples)

images , labels = iter(dataloader).next()

```

```

images , labels = images.numpy() , labels.numpy()
fig = plt.figure(figsize = (25,4))
for i in range(batch_size):
    ax = fig.add_subplot(2 , batch_size/2 , i+1 , xticks = [] , yticks = [])
    ax.imshow(images[i].transpose(2,1,0).squeeze())
    ax.set_title(data.classes[labels[i]])
plt.tight_layout()

from torchvision import models
model = models.vgg16(pretrained = True)
# Parameters -> Do not perform gradient decent -> Freeze
for param in model.parameters():
    param.no_grad_ = True
model.classifier[6] = nn.Linear(4096 , 3)
model.cuda()
criterion = nn.CrossEntropyLoss().cuda()
optimizer = optim.Adam(model.parameters() , lr = 1e-5)
%timeit
n_epochs = 1
for e in range(n_epochs):
    train_loss = 0.0
    valid_loss = 0.0
    min_valid_loss = np.inf
    model.train()
    for batch_idx , (data , target) in enumerate(dataloader):
        data = data.cuda()
        target = target.cuda()
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output , target)
        loss.backward()

```

```

optimizer.step()

train_loss += loss.item()

model.eval()

for batch_idx , (data , target) in enumerate(validloader):

    data = data.cuda()

    target = target.cuda()

    optimizer.zero_grad()

    output = model(data)

    loss = criterion(output , target)

    valid_loss += loss.item()

train_loss = train_loss / len(dataloader)

valid_loss = valid_loss / len(validloader)

print("Epoch : { } , Batch : { } , Loss : { }".format(e+1 , batch_idx , train_loss))

if valid_loss < min_valid_loss:

    print("Valid Loss decreased from { } ---> { }".format(min_valid_loss , valid_loss))

    torch.save(model.state_dict() , "model_lung_canc.pt")

    min_valid_loss = valid_loss

def test(loaders, model, criterion, use_cuda = True):

    # monitor test loss and accuracy

    test_loss = 0.

    correct = 0.

    total = 0.

    model.eval()

    for batch_idx, (data, target) in enumerate(loaders):

        # move to GPU

        if use_cuda:

            data, target = data.cuda(), target.cuda()

        # forward pass: compute predicted outputs by passing inputs to the model

        output = model(data)

        # calculate the loss

```

```

loss = criterion(output, target)

# update average test loss
test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))

# convert output probabilities to predicted class
pred = output.data.max(1, keepdim=True)[1]

# compare predictions to true label
correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().numpy())

total += data.size(0)

print("Test Loss: {:.6f}\n".format(test_loss))

print("\nTest Accuracy: %2d%% (%2d/%2d)' % (
    100. * correct / total, correct, total))

test(testloader, model, criterion, use_cuda = True)

dataiter = iter(testloader)
images, labels = dataiter.next()

# get predictions
preds = np.squeeze(model(images.cuda()).data.max(1, keepdim=True)[1].cpu().numpy())

images = images.cpu().numpy()

fig = plt.figure(figsize=(25, 4))

for idx in np.arange(10):
    ax = fig.add_subplot(2, batch_size/2, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(images[idx].transpose(1,2,0)), cmap='gray')
    if(labels[idx] == 0 or labels[idx] == 2):
        ax.set_title("Cancer",
            color=("green" if preds[idx]==labels[idx] else "red"))
    else:
        ax.set_title("Non-Cancer",
            color=("green" if preds[idx]==labels[idx] else "red"))

plt.tight_layout()

```

Screenshots:

```
In [1]: import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as dataset

from torch.utils.data import DataLoader
```

```
In [2]: transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(),
    transforms.ToTensor()
])
```

```
In [3]: data = dataset.ImageFolder('../input/lung-and-colon-cancer-histopathological-ima
```

```
In [4]: datalen = len(data)
dataidx = np.array(list(range(datalen)))
np.random.shuffle(dataidx)

splitfrac = 0.9
split_idx = int(splitfrac * datalen)
train_idxes = dataidx[:split_idx]
valid_idxes = dataidx[split_idx:]

testsplit = 0.1
testidxes = int(testsplit * len(train_idxes))

test_idxes = train_idxes[:testidxes]
train_idxes = train_idxes[testidxes:]

np.random.shuffle(test_idxes)
```

```
In [5]: batch_size = 30
```

```
In [6]: train_samples = torch.utils.data.SubsetRandomSampler(train_idxes)
valid_samples = torch.utils.data.SubsetRandomSampler(valid_idxes)
test_samples = torch.utils.data.SubsetRandomSampler(test_idxes)
dataloader = DataLoader(data , batch_size = batch_size , sampler = train_samples)
validloader = DataLoader(data , batch_size = batch_size , sampler = valid_samples)
testloader = DataLoader(data , batch_size = batch_size , sampler = test_samples)
```

```
In [7]: images , labels = iter(dataloader).next()
images , labels = images.numpy() , labels.numpy()

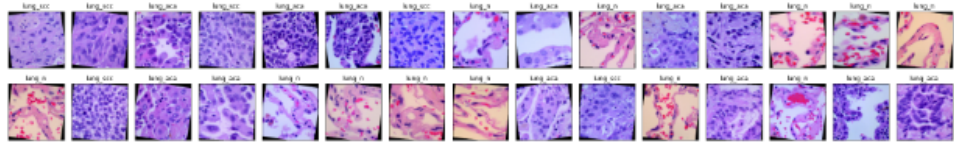
fig = plt.figure(figsize = (25,4))

for i in range(batch_size):
    ax = fig.add_subplot(2 , batch_size/2 , i+1 , xticks = [] , yticks = [])
    ax.imshow(images[i].transpose(2,1,0).squeeze())
    ax.set_title(data.classes[labels[i]])

plt.tight_layout()
```

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:7: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.

```
import sys
```



```
In [8]: from torchvision import models
model = models.vgg16(pretrained = True)

# Parameters -> Do not perform gradient decent -> Freeze

for param in model.parameters():
    param.no_grad_ = True

model.classifier[6] = nn.Linear(4096 , 3)

Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
0%|          | 0.00/528M [00:00<?, ?B/s]
```



```
In [9]: model.cuda()
```

```
Out[9]: VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace=True)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace=True)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=3, bias=True)
```

```
In [10]: criterion = nn.CrossEntropyLoss().cuda()
optimizer = optim.Adam(model.parameters() , lr = 1e-5)
```

```
In [11]: %timeit
n_epochs = 1

for e in range(n_epochs):
    train_loss = 0.0
    valid_loss = 0.0
    min_valid_loss = np.inf
    model.train()
    for batch_idx , (data , target) in enumerate(dataloader):
        data = data.cuda()
        target = target.cuda()
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output , target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    model.eval()
    for batch_idx , (data , target) in enumerate(validloader):
        data = data.cuda()
        target = target.cuda()
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output , target)
        valid_loss += loss.item()

    train_loss = train_loss / len(dataloader)
    valid_loss = valid_loss / len(validloader)
    print("Epoch : {} , Batch : {} , Loss : {}".format(e+1 , batch_idx , train_l

    if valid_loss < min_valid_loss:
        print("Valid Loss decreased from {} ---> {}".format(min_valid_loss , val
        torch.save(model.state_dict() , "model_lung_canc.pt")
        min_valid_loss = valid_loss
```

```
Epoch : 1 , Batch : 49 , Loss : 0.18040520711169566
Valid Loss decreased from inf ---> 0.057006154088303444
```

```
In [12]: def test(loaders, model, criterion, use_cuda = True):

    # monitor test loss and accuracy
    test_loss = 0.
    correct = 0.
    total = 0.

    model.eval()
    for batch_idx, (data, target) in enumerate(loaders):
        # move to GPU
        if use_cuda:
            data, target = data.cuda(), target.cuda()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model(data)
        # calculate the loss
        loss = criterion(output, target)
        # update average test loss
        test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
        # convert output probabilities to predicted class
        pred = output.data.max(1, keepdim=True)[1]
        # compare predictions to true label
        correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu()).n
        total += data.size(0)

    print('Test Loss: {:.6f}\n'.format(test_loss))

    print('\nTest Accuracy: %2d%% (%2d/%2d)' % (
        100. * correct / total, correct, total))
```

```
In [13]: test(testloader, model, criterion, use_cuda = True)
```

Test Loss: 0.052505

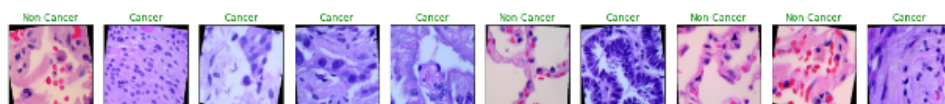
Test Accuracy: 98% (1324/1350)

```
In [14]: dataiter = iter(testloader)
images, labels = dataiter.next()
# get predictions
preds = np.squeeze(model(images.cuda()).data.max(1, keepdim=True)[1].cpu()).numpy()
images = images.cpu().numpy()

fig = plt.figure(figsize=(25, 4))

for idx in np.arange(10):
    ax = fig.add_subplot(2, batch_size/2, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(images[idx].transpose(1,2,0)), cmap='gray')
    if(labels[idx] == 0 or labels[idx] == 2):
        ax.set_title("Cancer",
            color=("green" if preds[idx]==labels[idx] else "red"))
    else:
        ax.set_title("Non-Cancer",
            color=("green" if preds[idx]==labels[idx] else "red"))
plt.tight_layout()
```

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:10: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.
Remove the CWD from sys.path while we load stuff.



Conclusion:

We have built a computer-based application for lung cancer detection to help doctors to make better and informed decision when diagnosing the lung cancer. we have explored various software tools in python, libraries (PYTORCH) which are used to implement machine learning algorithms. We have implemented a Convolutional Neural network using a VGG16 pre trained model which can be trained to classify the lung tissue images into cancerous and non-cancerous.

Application of computer-based lung cancer detection system to help doctors to make better and informed decision when diagnosing the lung cancer. We have studied different research papers in the field of lung cancer detection using machine learning, image processing, deep learning.

The proposed system outperformed many available programs of lung cancer detection in accuracy, sensitivity, and specificity.

Future Scope:

Diagnosing cancer at its earliest stages often provides the best chance for a cure. With this in mind, talk with your doctor about what types of cancer screening may be appropriate for you. For a few cancers, studies show that screening tests can save lives by diagnosing cancer early. For other cancers, screening tests are recommended only for people with increased risk. A variety of medical organizations and patient-advocacy groups have recommendations and guidelines for cancer screening. Review the various guidelines with your doctor and together you can determine what's best for you based on your own risk factors for cancer. We intend to enhance the model by designing it to predict other cancerous tissues present in parts of human body.