Untitled

April 4, 2021

1 Associative Analysis of CT images

The goal is to run a correlative or associative study between CT scan images and patients' age and contrast (applied or not).

[1]:
```
import pandas as pd
import numpy as np
import os
import torchvision
import torch
import libtiff
from pydicom import dcmread
from libtiff import TIFF
import matplotlib.pyplot as plt
from torchvision import transforms
```

[2]:
```
data_description = pd.read_csv("overview.csv")
```

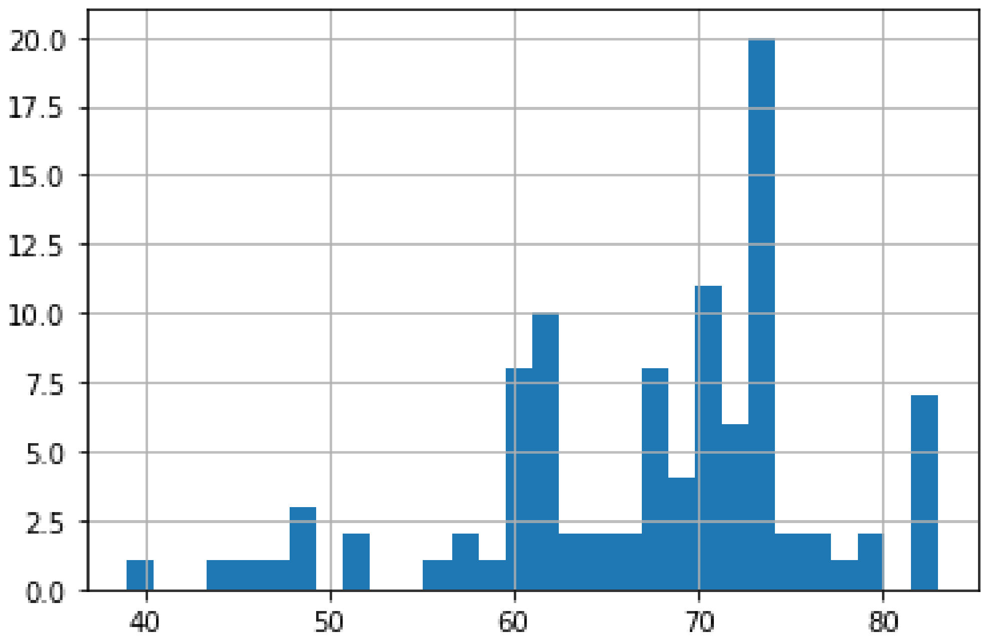
[3]:
```
data_description.head()
```

[3]:
```
   Unnamed: 0  Age  Contrast ContrastTag  \
0           0   60      True        NONE
1           1   69      True        NONE
2           2   74      True     APPLIED
3           3   75      True        NONE
4           4   56      True        NONE

                                     raw_input_path  id  \
0   ../data/50_50_dicom_cases\Contrast\00001 (1).dcm   0
1  ../data/50_50_dicom_cases\Contrast\00001 (10).dcm   1
2  ../data/50_50_dicom_cases\Contrast\00001 (11).dcm   2
3  ../data/50_50_dicom_cases\Contrast\00001 (12).dcm   3
4  ../data/50_50_dicom_cases\Contrast\00001 (13).dcm   4

                         tiff_name                         dicom_name
0  ID_0000_AGE_0060_CONTRAST_1_CT.tif  ID_0000_AGE_0060_CONTRAST_1_CT.dcm
1  ID_0001_AGE_0069_CONTRAST_1_CT.tif  ID_0001_AGE_0069_CONTRAST_1_CT.dcm
2  ID_0002_AGE_0074_CONTRAST_1_CT.tif  ID_0002_AGE_0074_CONTRAST_1_CT.dcm
```

```
3  ID_0003_AGE_0075_CONTRAST_1_CT.tif  ID_0003_AGE_0075_CONTRAST_1_CT.dcm
4  ID_0004_AGE_0056_CONTRAST_1_CT.tif  ID_0004_AGE_0056_CONTRAST_1_CT.dcm
```

[4]:
```python
data_description.Age.hist(bins = 30)
```

[4]: <AxesSubplot:>



Age column is pretty skewed. A suggestible idea would be to create a categorical data to deal with age. From histogram, probably best idea would be create a variable to indicate whether the patient is above or below 70 years old.

[5]:
```python
def age_cat(row):
    if row['Age'] > 70:
        return "Above"
    else:
        return "Below"
data_description['Age_Cat'] = data_description.apply(lambda row: age_cat(row), axis = 1)
```

[6]:
```python
data_description.Age_Cat.value_counts()
```

[6]:
```
Below    52
Above    48
Name: Age_Cat, dtype: int64
```

Our new target variable, is relatively balanced, therefore we would not have balance problem when training a model based on this target. Let's take a look at other target variable, even though in this specific analysis I am not going to use them for training.

[7]:
```
data_description.Contrast.value_counts()
```

[7]:
```
True     50
False    50
Name: Contrast, dtype: int64
```

Balanced regarding contrast distribution

[8]:
```
from torchvision import utils
from torch.utils.data import Dataset, DataLoader
```

[9]:
```
data_transforms =  transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

class CT_Images(Dataset):

    def __init__(self, csv_file_root, tiff_root, dicom_root, transform = None):

        self.csv_file = pd.read_csv(csv_file_root)
        self.tiff_root = tiff_root
        self.dicom_root = dicom_root
        self.transform = transform



    def __len__(self):
        return len(self.csv_file)



    def __getitem__(self, index):

        if torch.is_tensor(index):
            index = index.to_list()


        image_name_tiff = os.path.join(self.tiff_root, self.csv_file.
 ↪tiff_name[index])

        """
        If one is interested in loading DICOM data, it can be loaded and␣
 ↪returned.
        However due to our interest only in pixel data, this can be unnecessary
```

```
        and we can do our analysis with only TIFF images
        """

        image_name_dicom = os.path.join(self.dicom_root, self.csv_file.
↪dicom_name[index])
        #dicom_image = dcmread(image_name_dicom)
        image_TIFF = TIFF.open(image_name_tiff)
        for image in image_TIFF.iter_images():
            x = np.array(image)
        """
        Since Images are 1-channeled, we have to make them suitable for later␣
↪training
        by pre-trained models. Therefore, repeat them through 3-channels.
        """
        x = np.repeat(x[..., np.newaxis], 3, -1)
        x = x.transpose(2,1,0)


        contrast_image = self.csv_file.Contrast[index]
        string_contrast = "With Contrast" if contrast_image else "No Contrast"
        age_category = 1 if self.csv_file.Age[index] >= 70 else 0
        x = torch.from_numpy(x)
        if self.transform:
            x = self.transform(x)


        return x, age_category, string_contrast
```

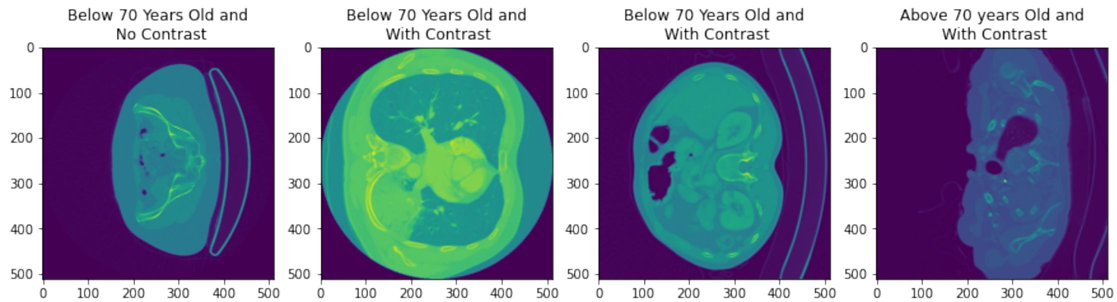
[10]:

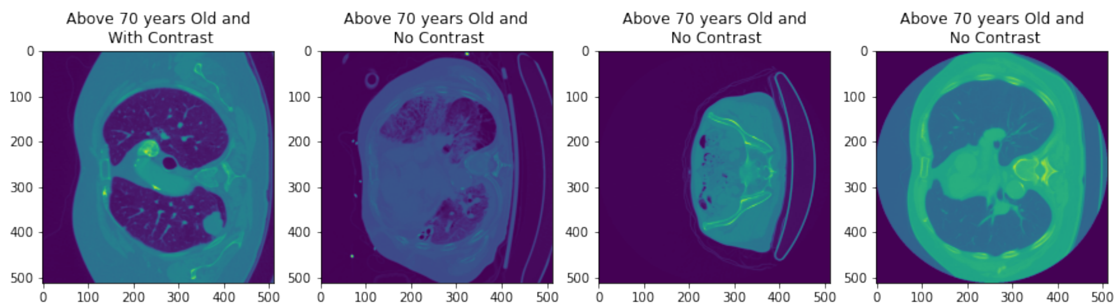
```
CT_images_dataset = CT_Images("overview.csv", "tiff_images", "dicom_dir",␣
↪data_transforms)
train_set, test_set = torch.utils.data.random_split(CT_images_dataset, [80, 20])
image_datasets = {'train': train_set, 'test':test_set}

dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=8,
                                             shuffle=True, num_workers=2)
              for x in ['train', 'test']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'test']}
class_names = ['Below 70 Years Old', 'Above 70 years Old']
```

[11]:

```
"Plotting a random sample of CT Scan Images"
images, classes, contrast_classes = next(iter(dataloaders['train']))
fig = plt.subplots(2,4, figsize=(15,15))
for i in range(8):
    plt.subplot(2,4,i+1)
```

```
    plt.title(class_names[classes[i]] + " and\n " + contrast_classes[i])
    plt.imshow(images[i][1].numpy().reshape(512,512))
```





2 Training and Fine-Tuning

We want to see whether possible to predict age of patient from CT scan slicing data. Due to very low number of data points, our only choice is transfer learning and fine-tuning. We use a VGG net pre-trained model and then fine tune it on data.

[12]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
import copy
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):

    chosen_model_weights = copy.deepcopy(model.state_dict())
    best_accuracy = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
```

```python
        print('-' * 10)

        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels,_ in dataloaders[phase]:
                inputs = inputs.to(device, dtype=torch.float)
                labels = labels.to(device, dtype=torch.long)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()


                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_accuracy = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Accuracy: {:.4f}'.format(
                phase, epoch_loss, epoch_accuracy))

            """
            We save the model's weight with best accuracy
            for later use.
            """
            if phase == 'test' and epoch_accuracy > best_accuracy:
                best_accuracy = epoch_accuracy
                chosen_model_weights = copy.deepcopy(model.state_dict())

        print()
```

```
    print('Best Test Accuracy: {:4f}'.format(best_accuracy))
    model.load_state_dict(chosen_model_weights)
    return model
```

[13]:
```
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms

"Getting Pre-Trained Resnet Model"
model_resnet = models.resnet18(pretrained=True)
model_features = model_resnet.fc.in_features

"Changing Last Layer to Suit Two Class Training"
model_resnet.fc = nn.Linear(model_features, 2)
model_resnet = model_resnet.to(device)

criterion = nn.CrossEntropyLoss()
optimizer_model = optim.SGD(model_resnet.parameters(), lr=0.001, momentum=0.9)
"Decaying learning rate, since we use a pre-trianed model"
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_model, step_size=8, gamma=0.1)
```

[14]:
```
model_resnet = train_model(model_resnet, criterion, optimizer_model,␣
 ↪exp_lr_scheduler, num_epochs=25)
```

```
Epoch 0/24
----------
train Loss: 0.8000 Accuracy: 0.4625
test Loss: 0.6792 Accuracy: 0.5500

Epoch 1/24
----------
train Loss: 0.6648 Accuracy: 0.5875
test Loss: 0.6549 Accuracy: 0.5000

Epoch 2/24
----------
train Loss: 0.5877 Accuracy: 0.6750
test Loss: 0.6457 Accuracy: 0.5000

Epoch 3/24
----------
train Loss: 0.5549 Accuracy: 0.7250
test Loss: 0.5120 Accuracy: 0.8000

Epoch 4/24
----------
```

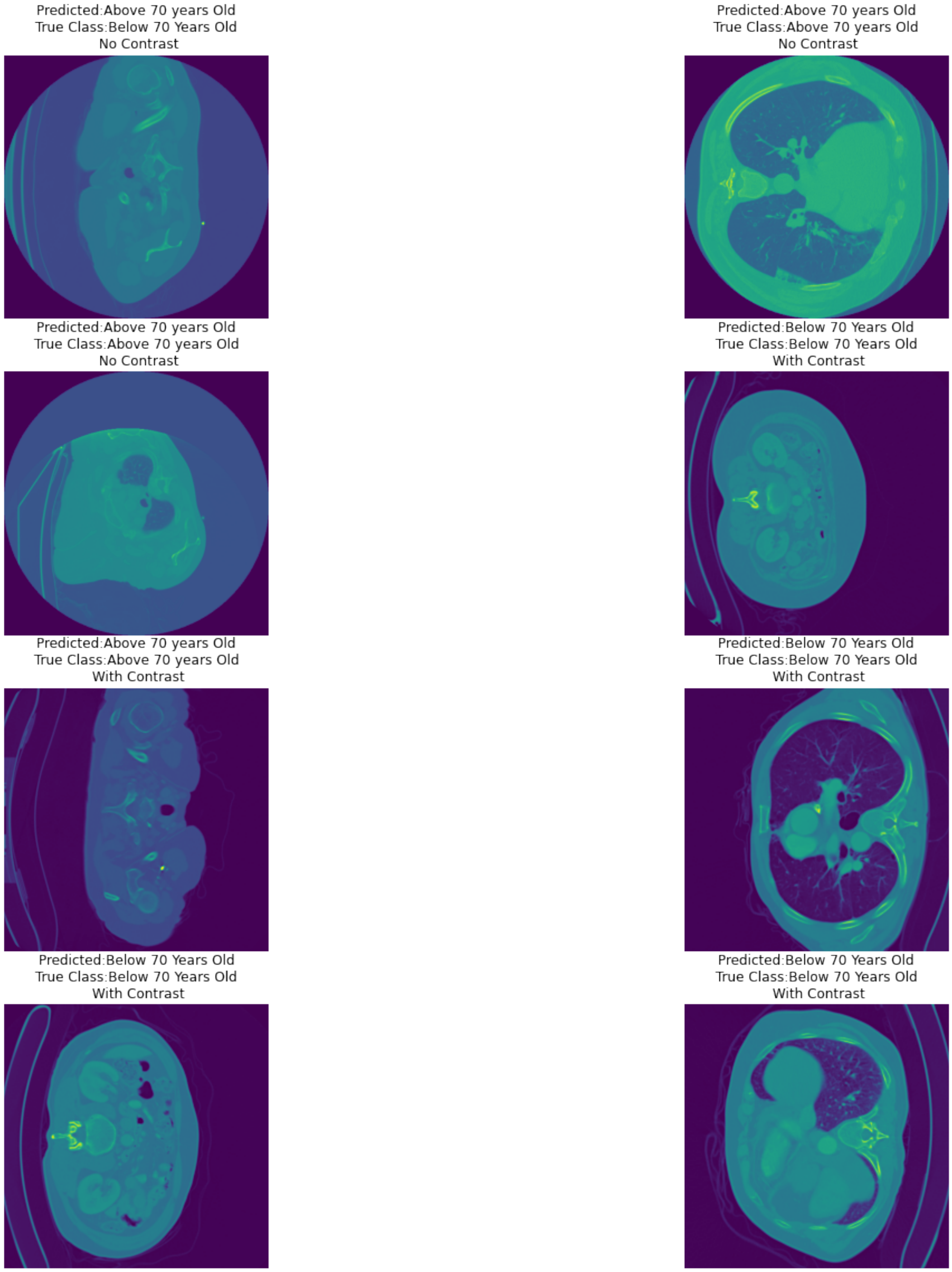
```
train Loss: 0.4620 Accuracy: 0.8375
test Loss: 0.5603 Accuracy: 0.8000

Epoch 5/24
----------
train Loss: 0.4197 Accuracy: 0.8000
test Loss: 0.5167 Accuracy: 0.7500

Epoch 6/24
----------
train Loss: 0.3407 Accuracy: 0.8750
test Loss: 0.4602 Accuracy: 0.8000

Epoch 7/24
----------
train Loss: 0.3984 Accuracy: 0.8250
test Loss: 0.4336 Accuracy: 0.8000

Epoch 8/24
----------
train Loss: 0.2553 Accuracy: 0.9375
test Loss: 0.3920 Accuracy: 0.9000

Epoch 9/24
----------
train Loss: 0.2140 Accuracy: 0.9750
test Loss: 0.4360 Accuracy: 0.7500

Epoch 10/24
----------
train Loss: 0.2698 Accuracy: 0.9375
test Loss: 0.4101 Accuracy: 0.9000

Epoch 11/24
----------
train Loss: 0.2625 Accuracy: 0.9250
test Loss: 0.4313 Accuracy: 0.7000

Epoch 12/24
----------
train Loss: 0.2249 Accuracy: 0.9750
test Loss: 0.4342 Accuracy: 0.8000

Epoch 13/24
----------
train Loss: 0.2211 Accuracy: 0.9750
test Loss: 0.4196 Accuracy: 0.8000
```

```
Epoch 14/24
----------
train Loss: 0.2299 Accuracy: 0.9500
test Loss: 0.3874 Accuracy: 0.9500

Epoch 15/24
----------
train Loss: 0.2005 Accuracy: 0.9625
test Loss: 0.3777 Accuracy: 0.9000

Epoch 16/24
----------
train Loss: 0.2098 Accuracy: 0.9750
test Loss: 0.4058 Accuracy: 0.8000

Epoch 17/24
----------
train Loss: 0.1647 Accuracy: 1.0000
test Loss: 0.4024 Accuracy: 0.7500

Epoch 18/24
----------
train Loss: 0.2526 Accuracy: 0.9750
test Loss: 0.4075 Accuracy: 0.8000

Epoch 19/24
----------
train Loss: 0.1928 Accuracy: 0.9875
test Loss: 0.3909 Accuracy: 0.8500

Epoch 20/24
----------
train Loss: 0.2185 Accuracy: 0.9625
test Loss: 0.3925 Accuracy: 0.9000

Epoch 21/24
----------
train Loss: 0.2048 Accuracy: 0.9625
test Loss: 0.3801 Accuracy: 0.9500

Epoch 22/24
----------
train Loss: 0.2123 Accuracy: 0.9625
test Loss: 0.3975 Accuracy: 0.8500

Epoch 23/24
----------
train Loss: 0.1750 Accuracy: 0.9875
```

```
test Loss: 0.4032 Accuracy: 0.8000

Epoch 24/24
----------
train Loss: 0.2700 Accuracy: 0.8875
test Loss: 0.3998 Accuracy: 0.9500

Best Test Accuracy: 0.950000
```

[15]:
```
"""
Using some images from test dataset
to illustrate performance of model
"""
def visualize_model(model, num_images = 8):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure(figsize=(20,20))

    with torch.no_grad():
        for i, (inputs, labels, contrast_class) in␣
 ↪enumerate(dataloaders['test']):
            inputs = inputs.to(device,dtype=torch.float)
            labels = labels.to(device,dtype=torch.long)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)


            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('Predicted:{}\n True Class:{} \n {}'.
 ↪format(class_names[preds[j]], class_names[labels[j]], contrast_class[j]))
                plt.imshow(inputs.cpu().data[j][1].numpy().reshape(512,512))

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

[16]:
```
visualize_model(model_resnet)
```



Even though this version of training, shows a great deal of accuracy, with some more trial and training, one could realize there is a great variance in test accuracy and perphaps then great deal of variance of model in relation to training data. The analysis needs more training points to reduce this variance and therefore not very wise to further analyze data with deep learning models.

```
[ ]:
```