Detecting COVID-19 with Chest X-Ray using PyTorch

Guided Project completed by Suhaimi William Chan

Instructor: Amit Yadav

We will use a ResNet-18 model and train it on a COVID-19 Radiography dataset. This dataset has nearly 3000 Chest X-Ray scans which are categorized in three classes - Normal, Viral Pneumonia and COVID-19. Our objective in this project is to create an image classification model that can predict Chest X-Ray scans that belong to one of the three classes with a reasonably high accuracy.

Please note that this dataset, and the model that we train in the project, can not be used to diagnose COVID-19 or Viral Pneumonia. We are only using this data for educational purpose.

We should be familiar with programming in Python. We should also have a theoretical understanding of Convolutional Neural Networks, and optimization techniques such as gradient descent. This is a hands on, practical project that focuses primarily on implementation, and not on the theory behind Convolutional Neural Networks.

Course Objectives

In this course, we are going to focus on the following learning objectives:

- Create custom Dataset and DataLoader in PyTorch
- Train a ResNet-18 model in PyTorch to perform Image Classification

By the end of this course, we will be able to create Convolutional Neural Networks, and will be able to train it to classify Chest X-Ray scans with reasonably high accuracy.

Project Structure

The hands on project on **Detecting COVID-19 with Chest X-Ray using PyTorch** is divided into following tasks:

Task 1: Introduction

Task 2: Importing Libraries

Task 3: Creating Custom Dataset

Task 4: Image Transformations

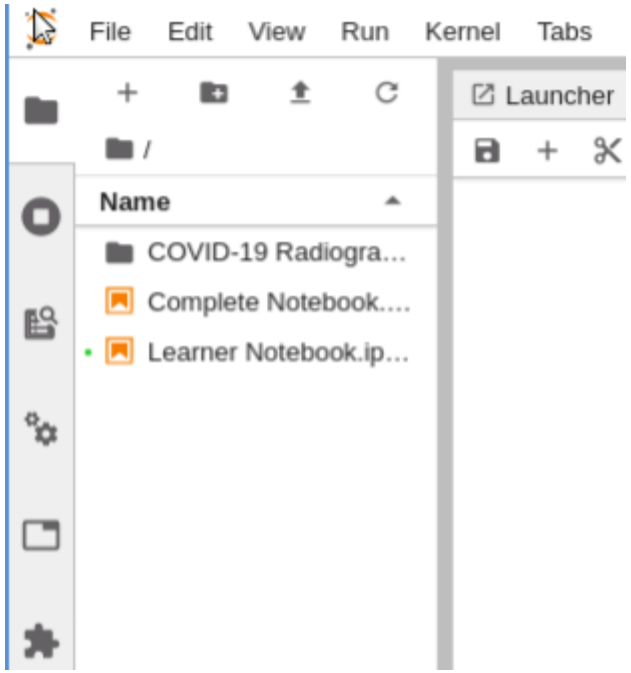
Task 5: Prepare DataLoader

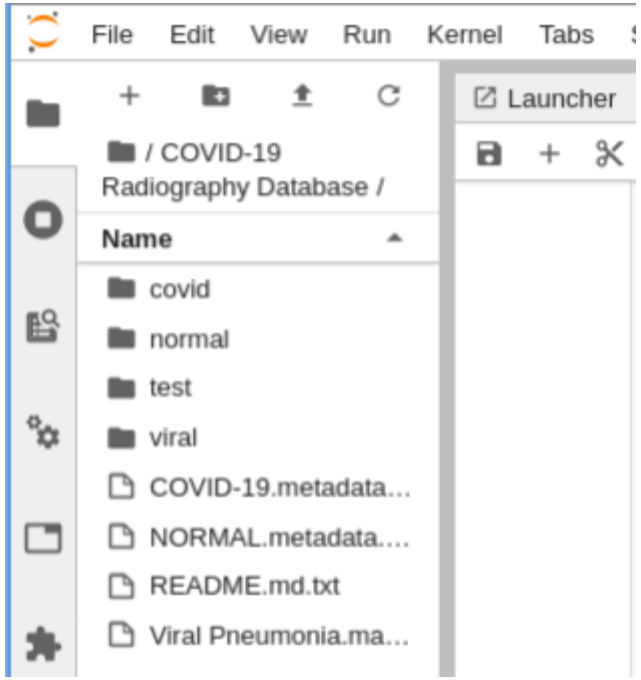
Task 6: Data Visualization

Task 7: Creating the Model

Task 8: Training the Model

Task 9: Final Results


File
Edit
View
Run
Kernel
Tabs
Launcher
/
Name
COVID-19 Radiogra...
Complete Notebook....
Learner Notebook.ip...


File
Edit
View
Run
Kernel
Tabs
Launcher
/ COVID-19
Radiography Database /
Name
covid
normal
test
viral
COVID-19.metadata...
NORMAL.metadata....
README.md.txt
Viral Pneumonia.ma...

```
# Detecting COVID-19 with Chest X Ray using PyTorch

Image classification of Chest X Rays in one of three classes: Normal, Viral
Pneumonia, COVID-19

Notebook created for the guided project [Detecting COVID-19 with Chest X Ray
using PyTorch](https://www.coursera.org/projects/covid-19-detection-x-ray) on
Coursera

Dataset from [COVID-19 Radiography Dataset](https://www.kaggle.com
/tawsifurrahman/covid19-radiography-database) on Kaggle
```

```
# Importing Libraries
```

```
%matplotlib inline

import os
import shutil
import random
import torch
import torchvision
import numpy as np

from PIL import Image
from matplotlib import pyplot as plt

torch.manual_seed(0)
```

```
print('Using PyTorch version', torch.__version__)
```

```
Using PyTorch version 1.5.1
```

```
# Preparing Training and Test Sets
```

```
class_names = ['normal', 'viral', 'covid']
root_dir = 'COVID-19 Radiography Database'
source_dirs = ['NORMAL', 'Viral Pneumonia', 'COVID-19']

if os.path.isdir(os.path.join(root_dir, source_dirs[1])):
    os.mkdir(os.path.join(root_dir, 'test'))

    for i, d in enumerate(source_dirs):
        os.rename(os.path.join(root_dir, d), os.path.join(root_dir, class_names[i]))

    for c in class_names:
        os.mkdir(os.path.join(root_dir, 'test', c))

    for c in class_names:
        images = [x for x in os.listdir(os.path.join(root_dir, c))
                  if x.lower().endswith('png')]
        selected_images = random.sample(images, 30)
        for image in selected_images:
            source_path = os.path.join(root_dir, c, image)
            target_path = os.path.join(root_dir, 'test', c, image)
            shutil.move(source_path, target_path)
```

```
# Creating Custom Dataset
```

```
class ChestXRayDataset(torch.utils.data.Dataset):
    def __init__(self, image_dirs, transform):
        def get_images(class_name):
            images = [x for x in os.listdir(image_dirs[class_name])
                      if x[-3:].lower().endswith('png')]
            print(f'Found {len(images)} {class_name} examples')
            return images

        self.images = {}
        self.class_names = ['normal', 'viral', 'covid']

        for class_name in self.class_names:
            self.images[class_name] = get_images(class_name)

        self.image_dirs = image_dirs
        self.transform = transform


    def __len__(self):
        return sum([len(self.images[class_name]) for class_name in self.class_names])


    def __getitem__(self, index):
        class_name = random.choice(self.class_names)
        index = index % len(self.images[class_name])
        image_name = self.images[class_name][index]
        image_path = os.path.join(self.image_dirs[class_name], image_name)
        image = Image.open(image_path).convert('RGB')
        return self.transform(image), self.class_names.index(class_name)
```

```
# Image Transformations
```

```
train_transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize(size=(224, 224)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
])

test_transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize(size=(224, 224)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```
# Prepare DataLoader
```

```
train_dirs = {
    'normal': 'COVID-19 Radiography Database/normal',
    'viral': 'COVID-19 Radiography Database/viral',
    'covid': 'COVID-19 Radiography Database/covid'
}

train_dataset = ChestXRayDataset(train_dirs, train_transform)
```

```
Found 1311 normal examples
Found 1315 viral examples
Found 189 covid examples
```

```
test_dirs = {
    'normal': 'COVID-19 Radiography Database/test/normal',
    'viral': 'COVID-19 Radiography Database/test/viral',
    'covid': 'COVID-19 Radiography Database/test/covid'
}

test_dataset = ChestXRayDataset(test_dirs, test_transform)
```

```
Found 30 normal examples
Found 30 viral examples
Found 30 covid examples
```

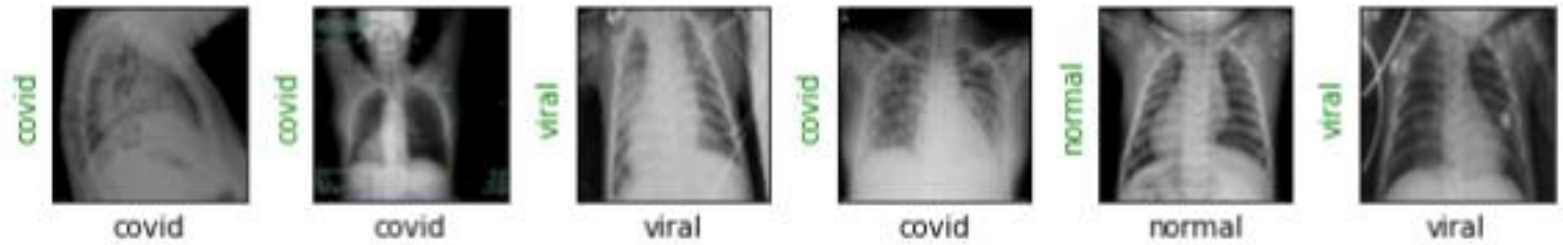
```
batch_size = 6

dl_train = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
                                       shuffle=True)
dl_test = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
                                      shuffle=True)

print('Number of training batches', len(dl_train))
print('Number of test batches', len(dl_test))
```

```
Number of training batches 470
Number of test batches 15
```

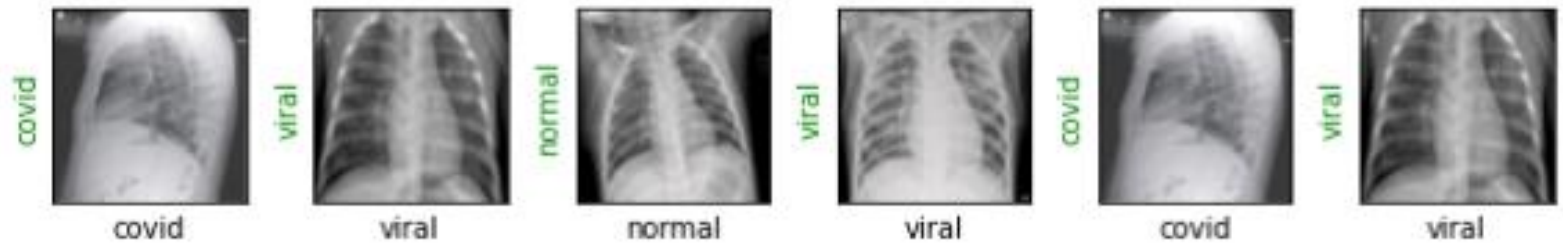
```
# Data Visualization
```

```
class_names = train_dataset.class_names


def show_images(images, labels, preds):
    plt.figure(figsize=(8, 4))
    for i, image in enumerate(images):
        plt.subplot(1, 6, i + 1, xticks=[], yticks=[])
        image = image.numpy().transpose((1, 2, 0))
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        image = image * std + mean
        image = np.clip(image, 0., 1.)
        plt.imshow(image)
        col = 'green'
        if preds[i] != labels[i]:
            col = 'red'

        plt.xlabel(f'{class_names[int(labels[i].numpy())]}')
        plt.ylabel(f'{class_names[int(preds[i].numpy())]}', color=col)
    plt.tight_layout()
    plt.show()
```

```
images, labels = next(iter(dl_train))
show_images(images, labels, labels)
```




```
images, labels = next(iter(dl_test))
show_images(images, labels, labels)
```



```
# Creating the Model
```

```
resnet18 = torchvision.models.resnet18(pretrained=True)

print(resnet18)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fals
e)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tru
e)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fals
e)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=
False)
```

```
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

```
resnet18.fc = torch.nn.Linear(in_features=512, out_features=3)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet18.parameters(), lr=3e-5)
```

```
def show_preds():
    resnet18.eval()
    images, labels = next(iter(dl_test))
    outputs = resnet18(images)
    _, preds = torch.max(outputs, 1)
    show_images(images, labels, preds)
```

```
show_preds()
```

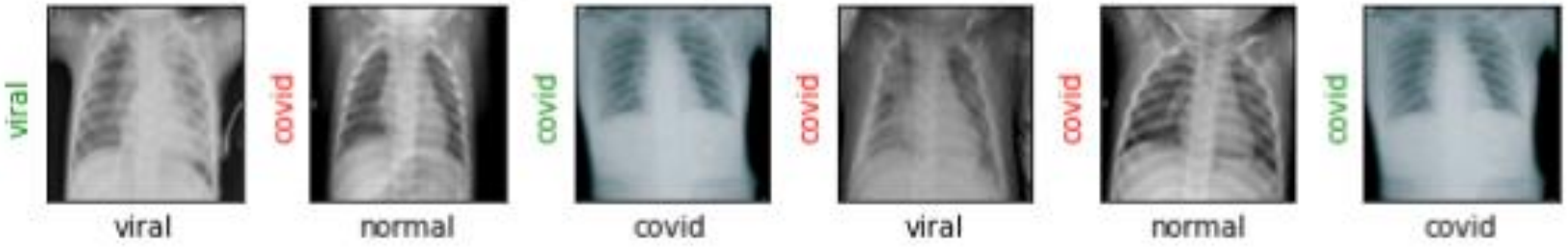
```
# Training the Model
```

```
def train(epochs):
    print('Starting training..')
    for e in range(0, epochs):
        print('='*20)
        print(f'Starting epoch {e + 1}/{epochs}')
        print('='*20)

        train_loss = 0.
        val_loss = 0.

        resnet18.train() # set model to training phase

        for train_step, (images, labels) in enumerate(dl_train):
            optimizer.zero_grad()
            outputs = resnet18(images)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
            if train_step % 20 == 0:
                print('Evaluating at step', train_step)

                accuracy = 0
```

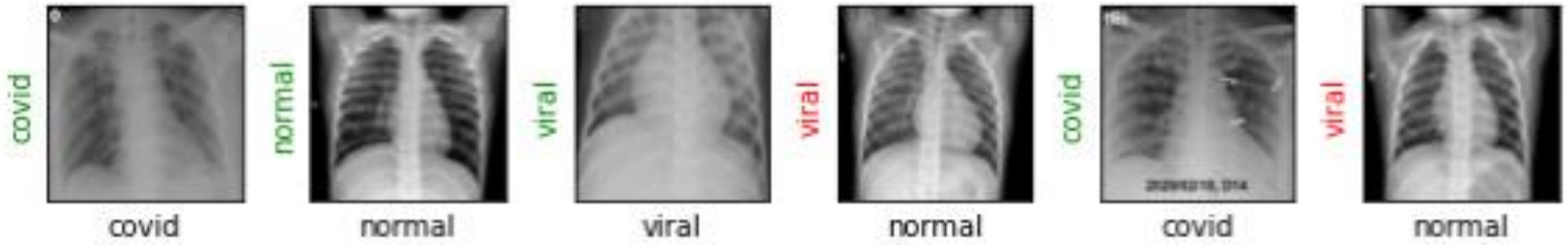
```
            resnet18.eval() # set model to eval phase

            for val_step, (images, labels) in enumerate(dl_test):
                outputs = resnet18(images)
                loss = loss_fn(outputs, labels)
                val_loss += loss.item()

                _, preds = torch.max(outputs, 1)
                accuracy += sum((preds == labels).numpy())

            val_loss /= (val_step + 1)
            accuracy = accuracy/len(test_dataset)
            print(f'Validation Loss: {val_loss:.4f}, Accuracy: {accuracy:.4f}')

            show_preds()

            resnet18.train()

            if accuracy >= 0.95:
                print('Performance condition satisfied, stopping..')
                return

        train_loss /= (train_step + 1)

        print(f'Training Loss: {train_loss:.4f}')
    print('Training complete..')
```

```
%%time

train(epochs=1)
```

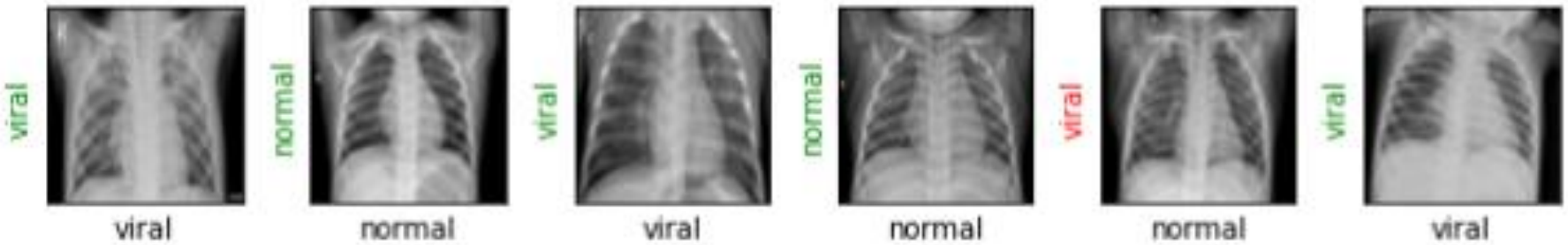
```
Starting training..
==========
Starting epoch 1/1
==========
Evaluating at step 0
Validation Loss: 1.2700, Accuracy: 0.2444
```



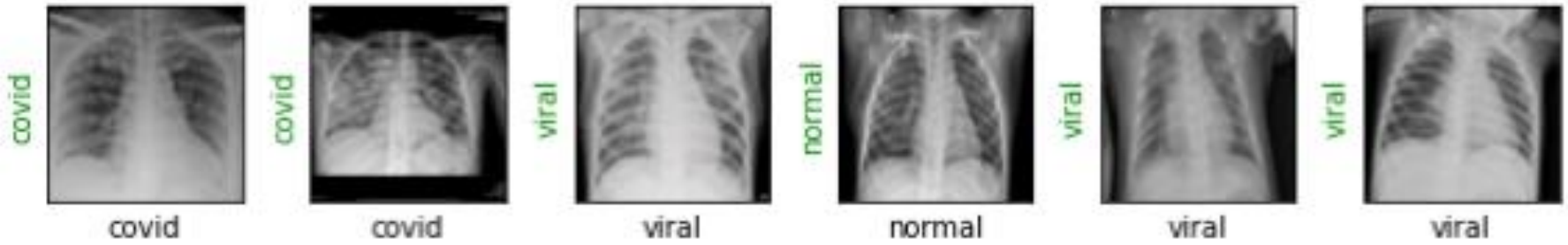

```
Evaluating at step 20
Validation Loss: 0.9085, Accuracy: 0.5778
```



```
Evaluating at step 40
Validation Loss: 0.5653, Accuracy: 0.7889
```




```
Evaluating at step 60
Validation Loss: 0.2190, Accuracy: 0.9778
```




```
Performance condition satisfied, stopping..
CPU times: user 1min 42s, sys: 2.38 s, total: 1min 44s
Wall time: 1min 45s
```

```
# Final Results
```

```
show_preds()
```