**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**

In [1]:

**%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**

In [2]:

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**

In [11]:

**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**

In [19]:

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**

In [20]:

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

In [21]:

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

In [22]:

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**

In [23]:

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)

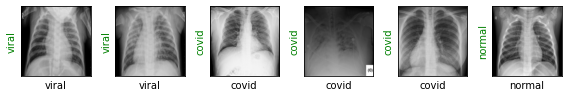
plt**.**tight\_layout()

plt**.**show()

In [24]:

images, labels **=** next(iter(dl\_train))

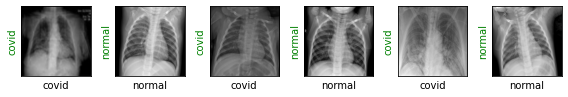
show\_images(images, labels, labels)



In [25]:

images, labels **=** next(iter(dl\_test))

show\_images(images, labels, labels)



**Creating the Model**

In [30]:

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

print(resnet18)

ResNet(

(conv1): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(maxpool): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)

(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(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)

)

)

(layer2): Sequential(

(0): BasicBlock(

(conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(64, 128, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer3): Sequential(

(0): BasicBlock(

(conv1): Conv2d(128, 256, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(downsample): Sequential(

(0): Conv2d(128, 256, kernel\_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(layer4): Sequential(

(0): BasicBlock(

(conv1): Conv2d(256, 512, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=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\_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(512, 512, 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\_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

)

(avgpool): AdaptiveAvgPool2d(output\_size=(1, 1))

(fc): Linear(in\_features=512, out\_features=1000, bias=True)

)

In [32]:

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)

In [33]:

**def** show\_preds():

resnet18**.**eval() *# set to evaluation mode*

images, labels **=** next(iter(dl\_test))

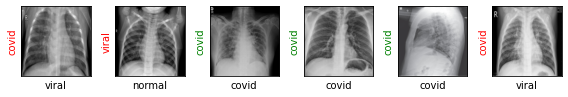
outputs **=** resnet18(images)

\_, preds **=** torch**.**max(outputs, 1)

show\_images(images, labels, preds)

In [34]:

show\_preds()



**Training the Model**

In [35]:

**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..')

In [36]:

**%%time**

train(epochs**=**1)

Starting training..

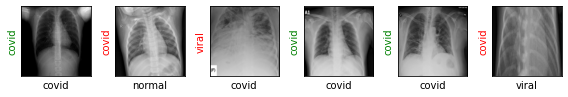
====================

Starting epoch 1/1

====================

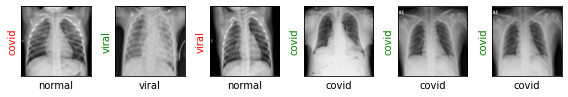
Evaluating at step 0

Validation Loss: 1.3951, Accuracy: 0.2333



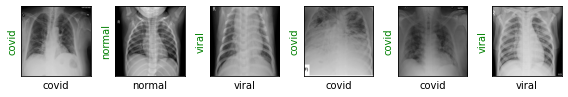
Evaluating at step 20

Validation Loss: 0.8155, Accuracy: 0.6444



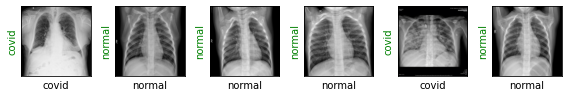
Evaluating at step 40

Validation Loss: 0.4558, Accuracy: 0.9000



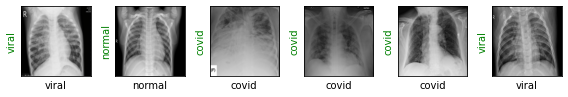
Evaluating at step 60

Validation Loss: 0.2589, Accuracy: 0.9222



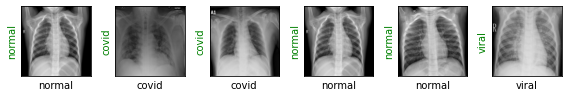
Evaluating at step 80

Validation Loss: 0.1472, Accuracy: 0.9444



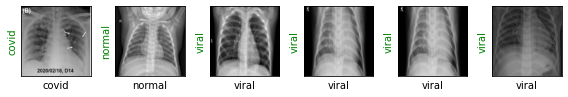
Evaluating at step 100

Validation Loss: 0.1587, Accuracy: 0.9333



Evaluating at step 120

Validation Loss: 0.1394, Accuracy: 0.9556



Performance condition satisfied, stopping..

CPU times: user 3min 24s, sys: 3.23 s, total: 3min 27s

Wall time: 4min 57s

**Final Results**

In [37]:

show\_preds()

