resnet18_pytorch_transfer Learning

January 4, 2022

[1]:
```
# import Library
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision
from torch.autograd import Variable
from torchvision import datasets, models, transforms
import os
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Data Aumentation step
data_transforms = {'train':transforms.Compose([transforms.
 ↪RandomResizedCrop(400),
                                              transforms.RandomHorizontalFlip(),
                                              transforms.ToTensor(),
                                              transforms.Normalize([0.485, 0.
 ↪456, 0.406], [0.229, 0.224, 0.225])
                                              ]),
                   'test': transforms.Compose([transforms.Resize(460),
                                              transforms.CenterCrop(400),
                                              transforms.ToTensor(),
                                              transforms.Normalize([0.485, 0.
 ↪456, 0.406], [0.229, 0.224, 0.225])
                                              ]),
                  }
```

[3]:
```
# Load Local dataset and make it trainable
data_dir = 'data'
#Create a dictionary that contains the information of the images in both the␣
 ↪training and validation set
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir,␣
 ↪x),data_transforms[x]) for x in ['train', 'test']}
#Create a dictionary that contians the data loader
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
                                              batch_size=4,
                                                   shuffle=True) for x in ['train',␣
↪'test']}

#Create a dictionary that contains the size of each dataset (training and␣
↪validation)
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'test']}
#Get the class names
class_names = image_datasets['train'].classes
#Print out the results
print("Class Names: {}".format(class_names))
print("There are {} batches in the training set".
↪format(len(dataloaders['train'])))
print("There are {} batches in the test set".format(len(dataloaders['test'])))
print("There are {} training images".format(dataset_sizes['train']))
print("There are {} testing images".format(dataset_sizes['test']))
```

```
Class Names: ['Mask', 'NoMask']
There are 1072 batches in the training set
There are 210 batches in the test set
There are 4286 training images
There are 837 testing images
```

[4]:
```
# load ResNet18
model = models.resnet18(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
```

[5]:
```
num_neural_last_layer = model.fc.in_features  # No of neural in last layer
# print(num_neural_last_layer)
model.fc = nn.Linear(num_neural_last_layer, 2)  # as we have only 2 class
```

[6]:
```
if torch.cuda.is_available():
    model = model.cuda()
```

[7]:
```
# Set loss, optimizer, lr
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.fc.parameters(), lr=0.001, momentum=0.9)
# Decay LR by a fector of 0.1 every 7 epochs
exp_lr_schedular = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

[8]:
```
#Training
num_epochs = 50
for epoch in range(num_epochs):
    correct = 0
    for (images, labels) in dataloaders['train']:
        images = Variable(images)
        labels = Variable(labels)
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()
    exp_lr_schedular.step()
    train_acc = 100*correct / dataset_sizes['train']
    print('Epoch: {}/{}, Loss: {:.4f}, Accuracy: {:.2f}%'.format(epoch+1,␣
↪num_epochs, loss.item(), train_acc))
```

```
/home/artpark/anaconda3/envs/ew/lib/python3.9/site-packages/PIL/Image.py:975:
UserWarning: Palette images with Transparency expressed in bytes should be
converted to RGBA images
  warnings.warn(
```

```
Epoch: 1/50, Loss: 0.6577, Accuracy: 72.94%
Epoch: 2/50, Loss: 0.2220, Accuracy: 78.84%
Epoch: 3/50, Loss: 0.0063, Accuracy: 80.12%
Epoch: 4/50, Loss: 2.1686, Accuracy: 78.86%
Epoch: 5/50, Loss: 0.0137, Accuracy: 78.35%
Epoch: 6/50, Loss: 1.8531, Accuracy: 80.66%
Epoch: 7/50, Loss: 1.6590, Accuracy: 80.10%
Epoch: 8/50, Loss: 1.3701, Accuracy: 83.43%
Epoch: 9/50, Loss: 0.0020, Accuracy: 84.81%
Epoch: 10/50, Loss: 0.9358, Accuracy: 84.32%
Epoch: 11/50, Loss: 0.0217, Accuracy: 84.39%
Epoch: 12/50, Loss: 0.1178, Accuracy: 83.11%
Epoch: 13/50, Loss: 0.0366, Accuracy: 84.13%
Epoch: 14/50, Loss: 0.0713, Accuracy: 84.32%
Epoch: 15/50, Loss: 0.9521, Accuracy: 84.46%
Epoch: 16/50, Loss: 2.6079, Accuracy: 85.00%
Epoch: 17/50, Loss: 0.7549, Accuracy: 84.58%
Epoch: 18/50, Loss: 0.0422, Accuracy: 84.55%
Epoch: 19/50, Loss: 0.4942, Accuracy: 84.53%
Epoch: 20/50, Loss: 0.9487, Accuracy: 84.72%
Epoch: 21/50, Loss: 0.0183, Accuracy: 85.58%
Epoch: 22/50, Loss: 0.0884, Accuracy: 85.11%
Epoch: 23/50, Loss: 0.0733, Accuracy: 84.46%
Epoch: 24/50, Loss: 0.9455, Accuracy: 84.79%
Epoch: 25/50, Loss: 0.0719, Accuracy: 84.13%
Epoch: 26/50, Loss: 0.5935, Accuracy: 84.34%
Epoch: 27/50, Loss: 0.1599, Accuracy: 85.98%
```

```
Epoch: 28/50, Loss: 1.2331, Accuracy: 85.18%
Epoch: 29/50, Loss: 0.7383, Accuracy: 85.00%
Epoch: 30/50, Loss: 0.0525, Accuracy: 84.53%
Epoch: 31/50, Loss: 0.5534, Accuracy: 84.86%
Epoch: 32/50, Loss: 1.0819, Accuracy: 84.09%
Epoch: 33/50, Loss: 0.2173, Accuracy: 84.62%
Epoch: 34/50, Loss: 0.0160, Accuracy: 84.72%
Epoch: 35/50, Loss: 0.0630, Accuracy: 85.09%
Epoch: 36/50, Loss: 1.4906, Accuracy: 85.86%
Epoch: 37/50, Loss: 0.8208, Accuracy: 84.48%
Epoch: 38/50, Loss: 1.0557, Accuracy: 83.88%
Epoch: 39/50, Loss: 0.3642, Accuracy: 85.42%
Epoch: 40/50, Loss: 1.0310, Accuracy: 84.04%
Epoch: 41/50, Loss: 0.0285, Accuracy: 84.76%
Epoch: 42/50, Loss: 1.3116, Accuracy: 85.21%
Epoch: 43/50, Loss: 0.0600, Accuracy: 84.25%
Epoch: 44/50, Loss: 1.1354, Accuracy: 85.25%
Epoch: 45/50, Loss: 0.7491, Accuracy: 85.09%
Epoch: 46/50, Loss: 0.2079, Accuracy: 84.37%
Epoch: 47/50, Loss: 0.0052, Accuracy: 83.99%
Epoch: 48/50, Loss: 0.7249, Accuracy: 84.67%
Epoch: 49/50, Loss: 0.0343, Accuracy: 84.58%
Epoch: 50/50, Loss: 0.0091, Accuracy: 84.62%
```

[10]:

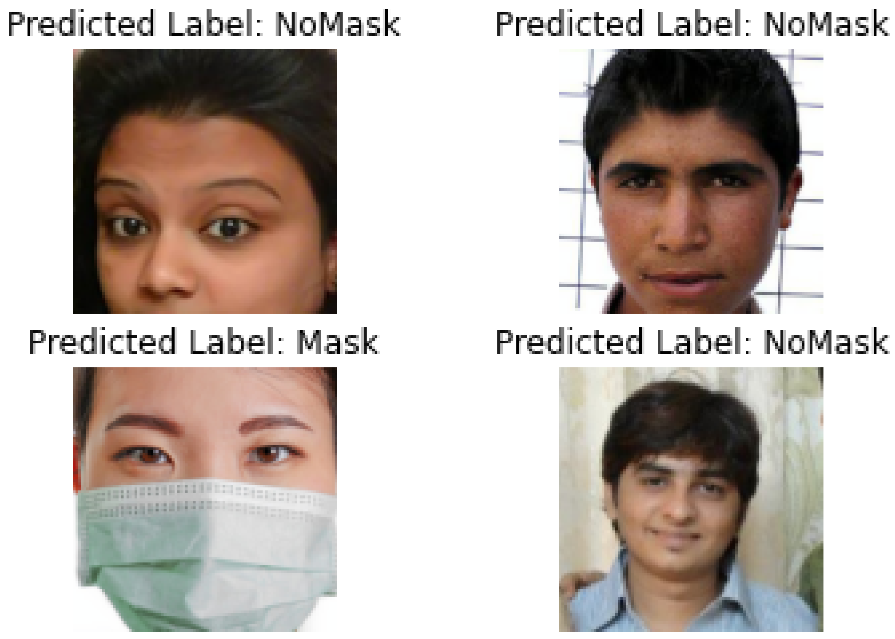
```
# Testing
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in dataloaders['test']:
        images = Variable(images)
        labels = Variable(labels)
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    print('Accuracy on test data is {}%'.format(100*correct//total))
```

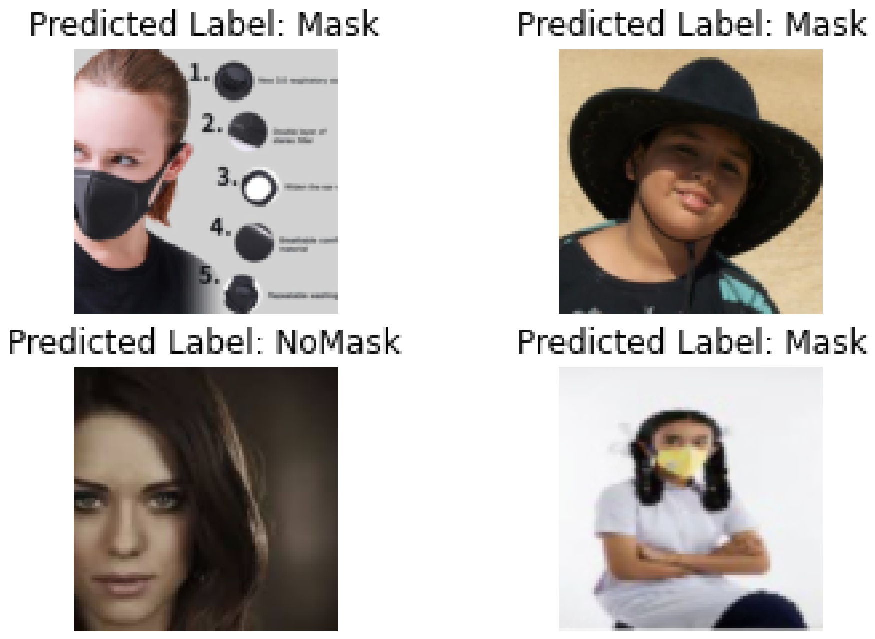
```
Accuracy on test data is 91%
```

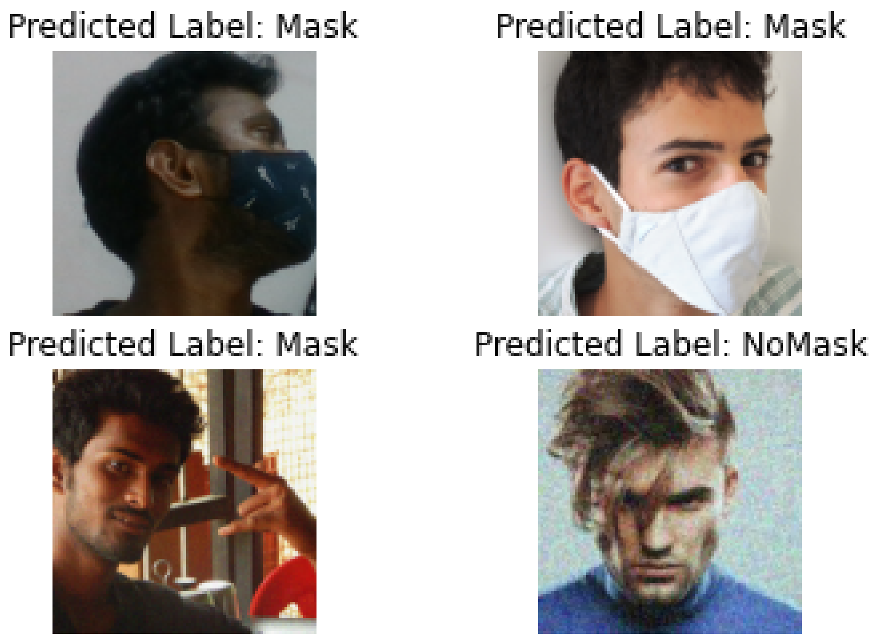
[27]:

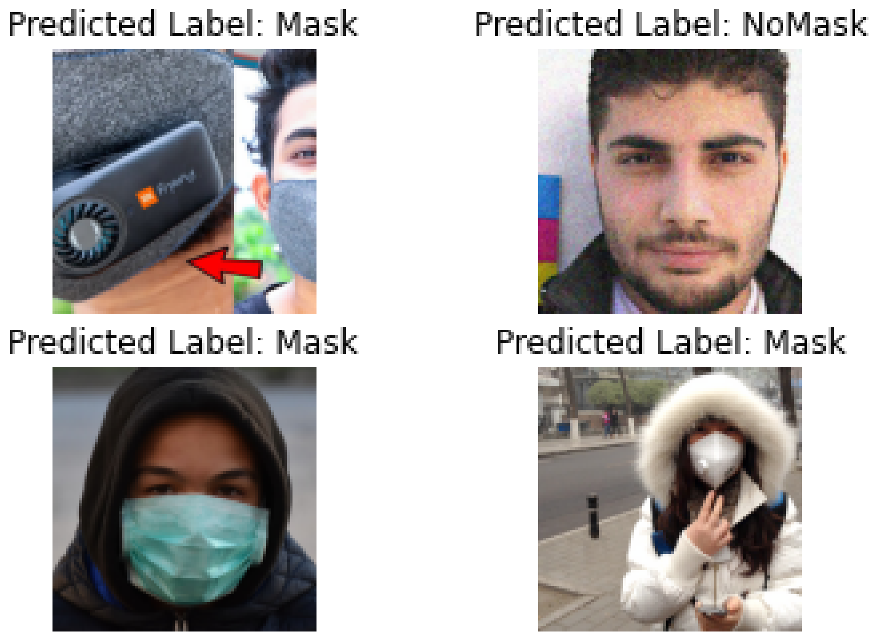
```
#Visualize some predictions
import matplotlib.pyplot as plt
# fig = plt.figure()
```

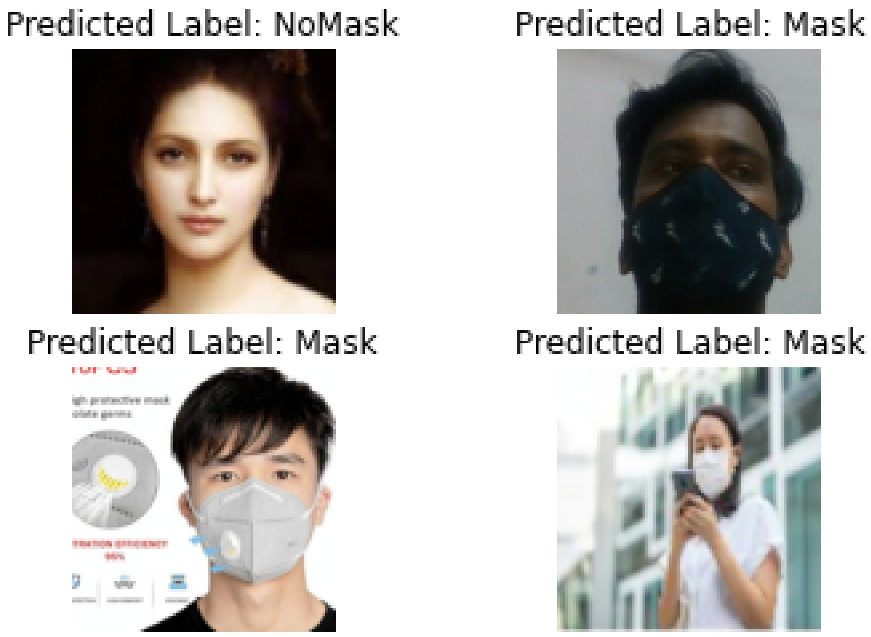
```
shown_batch = 0
with torch.no_grad():
    for (images, labels) in dataloaders['test']:
        index = 0
        if shown_batch == 5:
            break
        shown_batch += 1
        images = Variable(images)
        labels = Variable(labels)
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()

        outputs = model(images)                            #The output is of␣
 ↪shape (4,2)
        _, preds = torch.max(outputs, 1)                        #The pred is of␣
 ↪shape (4) --> [ 0,  0,  0,  1]

        for i in range(4):
            index += 1
#             fig = plt.figure(index)
            ax = plt.subplot(2,2,index)
            ax.axis('off')
            ax.set_title('Predicted Label: {}'.format(class_names[preds[i]]))
            input_img = images.cpu().data[i]                    #Get the tensor␣
 ↪of the image, and put it to cpu
            inp = input_img.numpy().transpose((1, 2, 0))        #If we have a␣
 ↪tensor of shape (2,3,4) --> it becomes (3,4,2)
            mean = np.array([0.485, 0.456, 0.406])
            std = np.array([0.229, 0.224, 0.225])
            inp = std * inp + mean
            inp = np.clip(inp, 0, 1)
            plt.imshow(inp)
        plt.show()
```


Predicted Label: NoMask
Predicted Label: NoMask
Predicted Label: Mask
Predicted Label: NoMask


Predicted Label: Mask
Predicted Label: Mask
1.
2.
3.
4.
5.
Predicted Label: NoMask
Predicted Label: Mask


Predicted Label: Mask
Predicted Label: Mask
Predicted Label: Mask
Predicted Label: NoMask


Predicted Label: Mask
Predicted Label: NoMask
Predicted Label: Mask
Predicted Label: Mask



```
[19]: torch.save(model.state_dict(), 'FaceMaskDetectionModel.pth.tar')
```

```
[ ]:
```