Task-1-pytorch-Xception

April 3, 2023

[2]:
```
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
import os
import h5py
import math
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import init
from torch.utils.data import Dataset, random_split, DataLoader
from torchvision import transforms
import torch.optim as optim
from torchmetrics.classification import MulticlassAUROC, MulticlassAccuracy
```

[3]:
```
# clearing cuda cache memory
import gc
torch.cuda.empty_cache()
gc.collect()
```

[3]:
```
0
```

[4]:
```
# dataset directory
# this directory contains all the datasets related for ML4SCI tests.
os.listdir("../dataset")
```

[4]:
```
['QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.snappy.parquet',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.snappy.parquet',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494.test.snappy.parquet',
 'SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5',
 'SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5']
```

[5]:
```
# import dataset

# importing electron dataset and seperating images and labels
```

```python
electron_dataset = h5py.File("../dataset/SingleElectronPt50_IMGCROPS_n249k_RHv1.
↪hdf5","r")
electron_imgs=np.array(electron_dataset["X"])
electron_labels=np.array(electron_dataset["y"],dtype=np.int64)

# importing photon dataset and seperating images and labels
photon_dataset = h5py.File("../dataset/SinglePhotonPt50_IMGCROPS_n249k_RHv1.
↪hdf5","r")
photon_imgs=np.array(photon_dataset["X"])
photon_labels=np.array(photon_dataset["y"],dtype=np.int64)
```

[6]:
```python
# concatenate electron and photon images/labels
img_arrs = torch.Tensor(np.vstack((photon_imgs,electron_imgs)))
labels = torch.Tensor(np.hstack((photon_labels,electron_labels))).to(torch.
↪int64)
```

[8]:
```python
# dataset class
# this will ease image/label reading at runtime
class SingleElectronPhotonDataset(Dataset):
    def __init__(self,split_inx, transform=None,target_transform= None):
        self.img_arrs_split = img_arrs[split_inx]
        self.labels_split = labels[split_inx]
        self.transform = transform
        self.target_transform = target_transform
    def __len__(self):
        return self.labels_split.shape[0]
    def __getitem__(self,idx):
        image=self.img_arrs_split[idx,:,:,:]
        # changing the dim of image to channels, height, width by transposing␣
↪the
        # original image tensor.
        image = image.permute(2,1,0)
        label = self.labels_split[idx]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image,label
```

[9]:
```python
class SeparableConv2d(nn.Module):
    def␣
↪__init__(self,in_channels,out_channels,kernel_size=1,stride=1,padding=0,bias=False):
↪
        """
        Seperable convolution layer in Xception model, as specified in
        https://arxiv.org/pdf/1610.02357.pdf
        """
```

```
        super(SeparableConv2d,self).__init__()

        self.conv1 = nn.Conv2d(in_channels,in_channels,kernel_size,stride,padding,groups=in_channels,bias=bias)
        self.pointwise = nn.Conv2d(in_channels,out_channels,1,1,0,1,1,bias=bias)

    def forward(self,x):
        x = self.conv1(x)
        x = self.pointwise(x)
        return x


class Block(nn.Module):
    def __init__(self,in_channels,out_channels,reps,strides=1,start_with_relu=True,expand_first=Tru
        '''
        reps: total number of separable conv layers in the block
              note that separable conv layers are preceded by relu and followed batch normalization.
        start_with_relu: if true start with relu
        expand_first: if True latent embedding dim of the block will be expanded to out_channels
                      at the beginning  else latent dim will be expanded at the end
        '''
        super(Block, self).__init__()

        if out_channels != in_channels or strides!=1:
            self.skip = nn.Conv2d(in_channels,out_channels,1,stride=strides, bias=False)
            self.skipbn = nn.BatchNorm2d(out_channels)
        else:
            self.skip=None

        self.relu = nn.ReLU(inplace=True)
        rep=[]

        filters=in_channels
        if expand_first:
            rep.append(self.relu)
            rep.append(SeparableConv2d(in_channels,out_channels,3,stride=1,padding=1,bias=False))
            rep.append(nn.BatchNorm2d(out_channels))
            filters = out_channels
```

```
        for i in range(reps-1):
            rep.append(self.relu)
            rep.
↪append(SeparableConv2d(filters,filters,3,stride=1,padding=1,bias=False))
            rep.append(nn.BatchNorm2d(filters))

        if not expand_first:
            rep.append(self.relu)
            rep.
↪append(SeparableConv2d(in_channels,out_channels,3,stride=1,padding=1,bias=False))
            rep.append(nn.BatchNorm2d(out_channels))

        if not start_with_relu:
            rep = rep[1:]
        else:
            rep[0] = nn.ReLU(inplace=False)

        if strides != 1:
            rep.append(nn.MaxPool2d(3,strides,1))
        self.rep = nn.Sequential(*rep)

    def forward(self,inp):
        x = self.rep(inp)

        if self.skip is not None:
            skip = self.skip(inp)
            skip = self.skipbn(skip)
        else:
            skip = inp

        x+=skip
        return x



class Xception(nn.Module):
    """
    Xception model, as specified in
    https://arxiv.org/pdf/1610.02357.pdf
    """
    def __init__(self, num_classes=2):
        """ Constructor
        Args:
            num_classes: number of classes
        """
        super(Xception, self).__init__()
```

```
        self.num_classes = num_classes

        self.conv1 = nn.Conv2d(2, 32, 3,2, 0, bias=False)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu = nn.ReLU(inplace=True)

        self.conv2 = nn.Conv2d(32,64,3,bias=False)
        self.bn2 = nn.BatchNorm2d(64)

        self.block1=Block(64,128,2,2,start_with_relu=False,expand_first=True)
        self.block2=Block(128,256,2,2,start_with_relu=True,expand_first=True)
        self.block3=Block(256,728,2,2,start_with_relu=True,expand_first=True)

        self.block4=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block5=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block6=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block7=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block8=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block9=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block10=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block11=Block(728,728,3,1,start_with_relu=True,expand_first=True)

        self.block12=Block(728,1024,2,2,start_with_relu=True,expand_first=False)

        self.conv3 = SeparableConv2d(1024,1536,3,1,1)
        self.bn3 = nn.BatchNorm2d(1536)

        self.conv4 = SeparableConv2d(1536,2048,3,1,1)
        self.bn4 = nn.BatchNorm2d(2048)

        self.fc = nn.Linear(2048, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu(x)

        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
```

```
        x = self.block6(x)
        x = self.block7(x)
        x = self.block8(x)
        x = self.block9(x)
        x = self.block10(x)
        x = self.block11(x)
        x = self.block12(x)

        x = self.conv3(x)
        x = self.bn3(x)
        x = self.relu(x)


        x = self.conv4(x)
        x = self.bn4(x)
        x = self.relu(x)


        x = F.adaptive_avg_pool2d(x, (1, 1))
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return F.softmax(x,dim=1)

    def __str__(self):
        return "Xception"
```

[10]:
```
# declare the device and the loss function
device = torch.device("cuda:0" if torch.cuda.is_available() else torch.
 ↪device("cpu"))
multicls_criterion = torch.nn.CrossEntropyLoss()
```

[11]:
```
model = Xception(num_classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 23
```

[12]:
```
# preprocess
preprocess = transforms.Compose([
    transforms.Resize(96),
    transforms.Normalize(mean=[0.5, 0.5], std=[0.5, 0.5]),
])

# random split of train, validation, tests set
# seed it set to 42 for reproducability of results
train_inx, valid_inx, test_inx = random_split(range(labels.shape[0]),[0.7,0.2,0.
 ↪1],generator=torch.Generator()
```

```
                                              .manual_seed(42))

train_data = SingleElectronPhotonDataset(split_inx=train_inx,transform =␣
 ↪preprocess)
valid_data = SingleElectronPhotonDataset(split_inx=valid_inx,transform =␣
 ↪preprocess)
test_data = SingleElectronPhotonDataset(split_inx=test_inx,transform =␣
 ↪preprocess)

# data loaders
train_dataloader = DataLoader(train_data,batch_size = 64, shuffle = True)
valid_dataloader = DataLoader(valid_data,batch_size = 64, shuffle = True)
test_dataloader = DataLoader(test_data,batch_size = 64, shuffle = True)
```

[13]:
```
# training loop

def train(model, device, loader, optimizer):
    model.train()

    loss_accum = 0
    for step, batch in enumerate(tqdm(loader, desc="Iteration")):
        inputs, labels = batch
        inputs = inputs.to(device)
        labels = labels.to(device)
        output = model(inputs)
        loss= 0
        optimizer.zero_grad()
        loss += multicls_criterion(output, labels)
        loss.backward()
        optimizer.step()

        loss_accum += loss.item()

    print('Average training loss: {}'.format(loss_accum / (step + 1)))
```

[14]:
```
# evaluation loop

def evaluate(model, device, loader,evaluator= "roauc",isTqdm=False):
    model.eval()

    preds_list = []
    target_list = []
    iterator = enumerate(loader)
    if isTqdm:
        iterator = enumerate(tqdm(loader))
    for step, batch in iterator:
        inputs, labels = batch
```

```
        inputs = inputs.to(device)
        labels = labels.to(device)
        with torch.no_grad():
            output = model(inputs)
            preds_list.extend(output.tolist())
        target_list += batch[1].tolist()
    if evaluator == "roauc":
        metric = MulticlassAUROC(num_classes=2, average="macro",␣
↪thresholds=None)
    if evaluator == "acc":
        metric = MulticlassAccuracy(num_classes=2, average="macro")
    # print("AUC-ROC metric score : ",metric(torch.Tensor(preds_list),torch.
↪Tensor(target_list)).item())
    return metric(torch.Tensor(preds_list),torch.Tensor(target_list).to(torch.
↪int64)).item()
```

[15]:
```
# setup for loading and saving checkpoints
checkpoints_path = "../models"
checkpoints = os.listdir(checkpoints_path)
checkpoint_path = list(filter(lambda i : str(model) in i, checkpoints))
```

0.0.1 Training and evaluating the Xception model.

Refer the readme for performance analysis

[16]:
```
train_curves = []
valid_curves = []

starting_epoch = 1
if len(checkpoint_path)>0:
    checkpoint = torch.load(f"{checkpoints_path}/{checkpoint_path[0]}")
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    starting_epoch = checkpoint['epoch']+1

for epoch in range(starting_epoch, epochs + 1):
    print("=====Epoch {}".format(epoch))
    print('Training...')
    train(model, device, train_dataloader, optimizer)

    print("Saving model...")
    # save checkpoint of current epoch
    torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            }, f"{checkpoints_path}/{str(model)}-{epoch}.pt")
```

```
    # delete checkpoint of previous epoch
    if epoch>1:
        os.remove(f"{checkpoints_path}/{str(model)}-{epoch-1}.pt")

    print("Evaluating...")
    train_perf_roauc = evaluate(model,device,train_dataloader)
    valid_perf_roauc = evaluate(model,device,valid_dataloader)
    test_perf_roauc = evaluate(model,device,test_dataloader)

    print('ROAUC scores: ',{'Train': train_perf_roauc, 'Validation':␣
 ↪valid_perf_roauc})

print('\nFinished training!')
print('\nROAUC Test score: {}'.format(evaluate(model,device,test_dataloader)))
```

```
=====Epoch 23
Training…

Iteration: 100%|      | 5447/5447 [27:49<00:00,  3.26it/s]

Average training loss: 0.5227140573508064
Saving model…
Evaluating…
ROAUC scores:  {'Train': 0.8334630727767944, 'Validation': 0.7872711420059204}

Finished training!

ROAUC Test score: 0.7892882823944092
```

0.0.2 Testing the model on the entire dataset.

[17]:
```
tot_dataloader =␣
 ↪DataLoader(SingleElectronPhotonDataset(split_inx=list(range(labels.
 ↪shape[0])),
                                                                      ␣
 ↪         transform = preprocess))
print('\nROAUC Total score: {}'.
 ↪format(evaluate(model,device,tot_dataloader,isTqdm=True)))
```

```
100%|      | 498000/498000 [1:48:26<00:00, 76.54it/s]

ROAUC Total score: 0.8199927806854248
```