Task-1-pytorch-Xception

April 6, 2023

[1]:
```
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
from torch.utils.tensorboard import SummaryWriter
```

[2]:
```
# clearing cuda cache memory
import gc
torch.cuda.empty_cache()
gc.collect()
```

[2]: 22

[3]:
```
# dataset directory
# this directory contains all the datasets related for ML4SCI tests.
os.listdir("../dataset")
```

[3]:
```
['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',
 'QG_Jets',
 'quark-gluon_data-set_n139306.hdf5',
 'SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5',
 'SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5']
```

[4]:
```
# import dataset

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

[5]:
```
# 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)
```

[6]:
```
# images array shape
img_arrs.shape
```

[6]:
```
torch.Size([498000, 32, 32, 2])
```

[7]:
```
# 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
```

[8]:
```
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"
```

[9]:
```
# 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()
```

[10]:
```
model = Xception(num_classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 20
```

[11]:
```
# 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)
```

[12]:
```
# 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)
        optimizer.zero_grad()
        loss = multicls_criterion(output, labels)
        loss.backward()
        optimizer.step()

        loss_accum += loss.item()

    return loss_accum / (step + 1)
```

[13]:
```
# evaluation loop

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

```
    preds_list = []
    target_list = []
    loss_accum =0
    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())
            if returnLoss:
                loss = multicls_criterion(output, labels)
                loss_accum += loss.item()
        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(),loss_accum/(step+1)
```

[14]:
```
# setup for loading and saving checkpoints
checkpoints_path = "../models"
checkpoints = os.listdir(checkpoints_path)
checkpoint_path = list(filter(lambda i : (str(model) in i) and (len(i.
↪split("-"))==2), checkpoints))
checkpoint_path = sorted(checkpoint_path,key=lambda n : [int(n[:-3].
↪split("-")[1]),n])
```

[15]:
```
# setup for curve plotting using tensorboard
curves_path = "../tensorboard-plots"
writer = SummaryWriter(log_dir = f"{curves_path}/{str(model)}/exp3")
```

[16]:
```
# setting maximum patience for early stopping

maxPatience = 3 # patience for monotonic increase
maxTolerance =5 # patience for gradual increase
```

0.0.1 Training and evaluating the Xception model.

Refer the readme for performance analysis

[17]:
```
# list of values used for plotting
train_losses = [1000]
val_losses = [1000]

# init values for early stopping and plotting
currentPatience = 0
currentTolerance = 0
toleranceValidScore = -1000.0
starting_epoch = 1
max_val_epoch = 0
max_val = 0
# loading previous checkpoints
if len(checkpoint_path)>0:
    checkpoint = torch.load(f"{checkpoints_path}/{checkpoint_path[-1]}")
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    starting_epoch = checkpoint['epoch']+1
    currentPatience = checkpoint['currentPatience']
    maxPatience = max(checkpoint['prevMaxPatience'],maxPatience)
    currentTolerance = checkpoint['currentTolerance']
    maxTolerance = max(checkpoint['prevMaxTolerance'],maxTolerance)
    toleranceValidScore = checkpoint['toleranceValidScore']
    val_losses = [checkpoint['val_loss']]
    train_losses = [checkpoint['train_loss']]
    max_val_epoch = checkpoint['max_val_epoch']
    max_val = checkpoint['max_val']

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

    print("Evaluating...")
    train_perf_auc,_ = evaluate(model,device,train_dataloader,returnLoss=False)
    valid_perf_auc,val_loss =␣
 ↪evaluate(model,device,valid_dataloader,returnLoss=True)

    # keep the maximum val auc and the epoch
    if max_val < valid_perf_auc:
        max_val_epoch = epoch
        max_val = valid_perf_auc

    if currentTolerance >0:
        if toleranceValidScore <= val_loss:
            currentTolerance+=1
        else:
```

```
            '''
            Removed deleting
            '''
            # tolerancePoint = f"{checkpoints_path}/
↪{str(model)}-{epoch-(currentTolerance+1)}.pt"
            # os.remove(tolerancePoint)
            currentTolerance=0

    if train_losses[-1]>train_loss and val_losses[-1]<=val_loss:
        currentPatience +=1
        if currentTolerance == 0:
            toleranceValidScore = val_losses[-1] # set the starting point a.k.a␣
↪tolerance point
            currentTolerance+=1
    else:
        '''
        Removed deleting
        '''
        # for pre_inx in range(2,currentPatience+2):
            # f = f"{checkpoints_path}/{str(model)}-{epoch-pre_inx}.pt"
            # if (currentTolerance != pre_inx) and os.path.exists(f) :
                # os.remove(f)
        currentPatience = 0

    train_losses.append(train_loss)
    val_losses.append(val_loss)

    writer.add_scalars('Loss', {"train" : train_loss,
                                "validation"  : val_loss}, epoch)
    writer.add_scalars("AUC",{'train':train_perf_auc,
                              'validation':valid_perf_auc}, epoch)

    # print('Losses: ',{'Train': train_loss, 'Validation': val_loss})
    print('ROC-AUC scores: ',{'Train': train_perf_auc, 'Validation':␣
↪valid_perf_auc})

    # stopping if overfitting

    # stop if the the val loss has increased monotonically
    if currentPatience == maxPatience:
        print("Early stopping training due to overfitting...")
        print(f"obtain results of epoch {epoch-maxPatience}")
        break

    # stop if the val loss has increased surpassing the tolerance patience
    if currentTolerance == maxTolerance:
        print("Early stopping training due to overfitting...")
```

```
        print(f"obtain results of epoch {epoch-(currentTolerance)}")
        break

    # save checkpoint of current epoch
    torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'train_loss':train_loss,
            'val_loss':val_loss,
            'optimizer_state_dict': optimizer.state_dict(),
            'currentPatience':currentPatience,
            'prevMaxPatience':maxPatience,
            'currentTolerance':currentTolerance,
            'prevMaxTolerance':maxTolerance,
            'toleranceValidScore':toleranceValidScore,
            'max_val_epoch':max_val_epoch,
            'max_val':max_val
            }, f"{checkpoints_path}/{str(model)}-{epoch}.pt")

print('\nFinished training!')

print('\nROC-AUC Test score at last epoch: {}'.
 ↪format(evaluate(model,device,test_dataloader)[0]))

print('\nMax ROC-AUC Validation score: {}'.format(max_val))
print('\nMax ROC-AUC Validation epoch: {}'.format(max_val_epoch))

# loading the model with max val
maxVal_checkpoint = torch.load(f"{checkpoints_path}/
 ↪{str(model)}-{max_val_epoch}.pt")
model.load_state_dict(maxVal_checkpoint['model_state_dict'])

print('\nROC-AUC Test score at epoch {} : {}'.
 ↪format(max_val_epoch,evaluate(model,device,test_dataloader)[0]))

# logging and plotting

if currentPatience == maxPatience:
    model_file = f"{checkpoints_path}/{str(model)}-{epoch-maxPatience}.pt"
    if os.path.exists(model_file):
        pre_model = torch.load(model_file)['model_state_dict']
        model.load_state_dict(pre_model)
        test_roc,test_loss =␣
 ↪evaluate(model,device,test_dataloader,returnLoss=True)
        print('\nROC-AUC Test score in {} prior to overfitting: {}'.
 ↪format(epoch-maxPatience,
```

```
                                                                                ␣
 ↪test_roc))
       print('\nTest loss in {} prior to overfitting: {}'.
 ↪format(epoch-maxPatience,
                                                                                ␣
 ↪test_loss))

elif currentTolerance == maxTolerance:
    model_file = f"{checkpoints_path}/{str(model)}-{epoch-maxTolerance}.pt"
    if os.path.exists(model_file):
        pre_model = torch.load(model_file)['model_state_dict']
        model.load_state_dict(pre_model)
        test_roc,test_loss =␣
 ↪evaluate(model,device,test_dataloader,returnLoss=True)
       print('\nROC-AUC Test score in {} prior to overfitting: {}'.
 ↪format(epoch-maxTolerance,
                                                                                ␣
 ↪test_roc))
       print('\nTest loss in {} prior to overfitting: {}'.
 ↪format(epoch-maxTolerance,
                                                                                ␣
 ↪test_loss))

writer.flush()
writer.close()
```

```
=====Epoch 1
Training…

Iteration: 100%|      | 5447/5447 [43:23<00:00,  2.09it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.4802662134170532, 'Validation': 0.4801178574562073}
=====Epoch 2
Training…

Iteration: 100%|      | 5447/5447 [45:57<00:00,  1.98it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7743293046951294, 'Validation': 0.7717928886413574}
=====Epoch 3
Training…

Iteration: 100%|      | 5447/5447 [46:39<00:00,  1.95it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.6000425815582275, 'Validation': 0.5959718823432922}
=====Epoch 4
Training…

Iteration: 100%|      | 5447/5447 [47:15<00:00,  1.92it/s]
```

```
Evaluating…
ROC-AUC scores:  {'Train': 0.5411525964736938, 'Validation': 0.5417307615280151}
=====Epoch 5
Training…

Iteration: 100%|     | 5447/5447 [46:33<00:00,  1.95it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7988113760948181, 'Validation': 0.7960073947906494}
=====Epoch 6
Training…

Iteration: 100%|     | 5447/5447 [47:25<00:00,  1.91it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.6784870624542236, 'Validation': 0.6708086729049683}
=====Epoch 7
Training…

Iteration: 100%|     | 5447/5447 [48:45<00:00,  1.86it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.559593915939331, 'Validation': 0.5561144948005676}
=====Epoch 8
Training…

Iteration: 100%|     | 5447/5447 [45:44<00:00,  1.98it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7494600415229797, 'Validation': 0.747252345085144}
=====Epoch 9
Training…

Iteration: 100%|     | 5447/5447 [47:55<00:00,  1.89it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7898080348968506, 'Validation': 0.7854228019714355}
=====Epoch 10
Training…

Iteration: 100%|     | 5447/5447 [47:11<00:00,  1.92it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7997227907180786, 'Validation': 0.7943601608276367}
=====Epoch 11
Training…

Iteration: 100%|     | 5447/5447 [38:00<00:00,  2.39it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7981902360916138, 'Validation': 0.7911926507949829}
=====Epoch 12
Training…

Iteration: 100%|     | 5447/5447 [47:20<00:00,  1.92it/s]
```

```
Evaluating…
ROC-AUC scores:  {'Train': 0.7935458421707153, 'Validation': 0.7864241003990173}
=====Epoch 13
Training…

Iteration: 100%|     | 5447/5447 [45:58<00:00,  1.97it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7949322462081909, 'Validation': 0.7859148383140564}
=====Epoch 14
Training…

Iteration: 100%|     | 5447/5447 [47:59<00:00,  1.89it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8055897951126099, 'Validation': 0.7946267127990723}
=====Epoch 15
Training…

Iteration: 100%|     | 5447/5447 [42:19<00:00,  2.15it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8171241283416748, 'Validation': 0.7988213300704956}
=====Epoch 16
Training…

Iteration: 100%|     | 5447/5447 [41:37<00:00,  2.18it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8195126056671143, 'Validation': 0.7981994152069092}
=====Epoch 17
Training…

Iteration: 100%|     | 5447/5447 [40:39<00:00,  2.23it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8239861726760864, 'Validation': 0.7992955446243286}
=====Epoch 18
Training…

Iteration: 100%|     | 5447/5447 [41:41<00:00,  2.18it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8264161348342896, 'Validation': 0.7949784398078918}
=====Epoch 19
Training…

Iteration: 100%|     | 5447/5447 [41:42<00:00,  2.18it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.815655529499054, 'Validation': 0.7794774770736694}
=====Epoch 20
Training…

Iteration: 100%|     | 5447/5447 [44:18<00:00,  2.05it/s]
```

```
Evaluating…
ROC-AUC scores:  {'Train': 0.8242670297622681, 'Validation': 0.790569543838501}

Finished training!

ROC-AUC Test score at last epoch: 0.7918810844421387

Max ROC-AUC Validation score: 0.7992955446243286

Max ROC-AUC Validation epoch: 17

ROC-AUC Test score at epoch 17 : 0.7994509935379028
```