resnet-15

March 18, 2024

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

[2]:
```
import h5py
electron_dataset = h5py.File('/kaggle/input/electron-photon-dataset/
↪SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5', 'r')
photon_dataset = h5py.File('/kaggle/input/electron-photon-dataset/
↪SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5', 'r')
```

[3]:
```
electron_dataset.keys()
```

[3]:
```
<KeysViewHDF5 ['X', 'y']>
```

[5]:
```
from torch.utils.data import DataLoader, random_split
from torch.utils.data import Dataset, ConcatDataset
```

```python
class FullDataset(Dataset):
    def __init__(self, data, transform):
        self.data = data
        self.transform = transform

    def __len__(self):
        return self.data['y'].shape[0]

    def __getitem__(self, idx):
        return self.transform(self.data['X'][idx]), self.data['y'][idx]
```

[6]:
```python
import torch
from torchvision.transforms import ToTensor, Compose, Resize, Lambda

transform = Compose([
    ToTensor(),
    Resize((128,128)),
    Lambda(lambd= lambda x: torch.cat([x, torch.zeros([1, 128, 128])], dim=0))
])
```

[7]:
```python
electron = FullDataset(electron_dataset, transform=transform)
photon = FullDataset(photon_dataset, transform=transform)

full = ConcatDataset([electron, photon])

train_data, test_data = random_split(full, [0.8, 0.2])
```

[8]:
```python
from torch.utils.data import Subset
import numpy as np


random_indices = np.random.choice(len(train_data), size=40000, replace=False)

small_train_dataset = Subset(train_data, random_indices)

batch_size = 64

train_dataloader = DataLoader(small_train_dataset, batch_size=batch_size,␣
↪shuffle=True,)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
```

[13]:
```python
device = torch.device('cuda')
```

[14]:
```python
import torch.nn as nn
import torch.nn.functional as F
```

```python
class ResNet15(nn.Module):
    def __init__(self, in_channels):
        super().__init__()

        # Initial convolution
        self.conv1 = conv_block(in_channels, 64)

        # Residual blocks
        self.res1 = nn.Sequential(conv_block(64, 64), conv_block(64, 64))
        self.conv2 = conv_block(64, 128, stride=2)  # Downsample using stride
        self.res2 = nn.Sequential(conv_block(128, 128), conv_block(128, 128), conv_block(128, 128))
        self.conv3 = conv_block(128, 512, stride=2)  # Downsample using stride
        self.res3 = nn.Sequential(conv_block(512, 512), conv_block(512, 512))
        self.conv4 = conv_block(512, 1024, stride=2)  # Downsample using stride
        self.res4 = nn.Sequential(conv_block(1024, 1024), conv_block(1024, 1024))

        # Classifier
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),  # Global average pooling
            nn.Flatten(),
            nn.Dropout(0.2),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 1)  # Adjust output size for classification (assuming 2 classes)
        )

    def forward(self, x):
        x = self.conv1(x)
        x = self.res1(x) + x  # Residual connection
        x = self.conv2(x)
        x = self.res2(x) + x  # Residual connection
        x = self.conv3(x)
        x = self.res3(x) + x  # Residual connection
        x = self.conv4(x)
        x = self.res4(x) + x  # Residual connection
        x = self.classifier(x)
        return x

def conv_block(in_channels, out_channels, kernel_size=3, stride=1, padding=1):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU()
    )
```

```
model = ResNet15(3).float().to(device)
```

[15]:
```
from tqdm import tqdm

criterion = nn.BCELoss()
optimiser = torch.optim.AdamW(model.parameters(), lr=3e-4)
num_epochs = 5

for epoch in range(1, num_epochs+1):
    epoch_loss = 0
    for batch in tqdm(train_dataloader, desc=f'Epoch {epoch}:'):
        # print(f'{i}/{len(train_dataloader)}')
        # print(batch[1],batch[1].shape)

        pred = model(batch[0].to(device))
        # print(pred, pred.shape)
        loss = criterion(torch.sigmoid(pred).squeeze(-1), batch[1].float().
 ↪to(device))
        optimiser.zero_grad()
        loss.backward()
        optimiser.step()
        epoch_loss += loss.item()
    print(f'Loss after {epoch} epochs = {epoch_loss}')
    torch.save(model.state_dict(), f'/kaggle/working/Resnet15-Epoch_{epoch}.
 ↪pth')
```

```
Epoch 1::   0%|          | 0/625 [00:00<?, ?it/s]/opt/conda/lib/python3.10/site-
packages/torchvision/transforms/functional.py:1603: UserWarning: The default
value of the antialias parameter of all the resizing transforms (Resize(),
RandomResizedCrop(), etc.) will change from None to True in v0.17, in order to
be consistent across the PIL and Tensor backends. To suppress this warning,
directly pass antialias=True (recommended, future default), antialias=None
(current default, which means False for Tensors and True for PIL), or
antialias=False (only works on Tensors - PIL will still use antialiasing). This
also applies if you are using the inference transforms from the models weights:
update the call to weights.transforms(antialias=True).
  warnings.warn(
Epoch 1:: 100%|      | 625/625 [1:49:18<00:00, 10.49s/it]

Loss after f1 epochs = 415.2847344279289

Epoch 2:: 100%|      | 625/625 [1:49:13<00:00, 10.49s/it]

Loss after f2 epochs = 391.9993373155594

Epoch 3:: 100%|      | 625/625 [1:48:21<00:00, 10.40s/it]

Loss after f3 epochs = 382.91221472620964
```

```
Epoch 4:: 100%|      | 625/625 [1:48:38<00:00, 10.43s/it]

Loss after f4 epochs = 376.4430049955845

Epoch 5:: 100%|      | 625/625 [1:49:05<00:00, 10.47s/it]

Loss after f5 epochs = 371.85460218787193
```

[ ]:
```
for epoch in range(1, num_epochs+1):
    epoch_loss = 0
    for batch in tqdm(train_dataloader, desc=f'Epoch {epoch}:'):
        # print(f'{i}/{len(train_dataloader)}')
        # print(batch[1],batch[1].shape)

        pred = model(batch[0].to(device))
        # print(pred, pred.shape)
        loss = criterion(torch.sigmoid(pred).squeeze(-1), batch[1].float().
 ↪to(device))
        optimiser.zero_grad()
        loss.backward()
        optimiser.step()
        epoch_loss += loss.item()
    print(f'Loss after {epoch+num_epochs} epochs = {epoch_loss}')
    torch.save(model.state_dict(), f'/kaggle/working/
 ↪Resnet15-Epoch_{epoch+num_epochs}.pth')
```

```
Epoch 1:: 100%|      | 625/625 [1:49:14<00:00, 10.49s/it]

Loss after 6 epochs = 366.11900609731674

Epoch 2::  48%|       | 302/625 [52:38<56:36, 10.51s/it]
```

[ ]:
```

```