resnet-small

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
```

[1]:
```
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')

electron_dataset.keys()
```

[1]:
```
<KeysViewHDF5 ['X', 'y']>
```

[2]:
```
from torch.utils.data import DataLoader, random_split
from torch.utils.data import Dataset, ConcatDataset
import torch
from torchvision.transforms import ToTensor, Compose, Resize, Lambda
```

```
from torch.utils.data import Subset
import numpy as np
import torch.nn as nn
import torch.nn.functional as F


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]


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

electron = FullDataset(electron_dataset, transform=transform)
photon = FullDataset(photon_dataset, transform=transform)

full = ConcatDataset([electron, photon])

random_indices = np.random.choice(len(full), size=10000, replace=False)

small_dataset = Subset(full, random_indices)

train_data, test_data = random_split(small_dataset, [0.8, 0.2])


batch_size = 64

train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True,)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=True)

device = 'cuda'

from tqdm import tqdm

class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
```

```python
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = None
        if stride != 1 or in_channels != out_channels:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        out = self.relu(out)
        return out

class Model(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(Model, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.residual_block = ResidualBlock(out_channels, out_channels)
        self.fc = nn.Linear(320, 2)  # Linear layer after the ResNet block

    def forward(self, x):
        out = self.conv(x)
        out = self.residual_block(out)
        out = F.avg_pool2d(out, 4)  # Example downsampling operation
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

model = Model(2, 5).float().to(device)
```

```python
from torch.utils.data import Subset
import numpy as np


def evaluate(loader, model):
    model.eval()
    total_loss = 0.0
    correct = 0
    total_samples = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for batch in tqdm(loader):

            pred = model(batch[0].to(device))

            pred_labels = torch.sigmoid(pred).argmax(dim=-1)
            correct += (pred_labels == batch[1].to(device)).sum().item()
            total_samples += len(batch[1].to(device))

    return correct/total_samples
```

[3]:
```python
num_epochs = 15
best_acc = 0

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

for epoch in range(1, num_epochs+1):
    epoch_loss = 0
    model.train()
    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(pred, batch[1].long().to(device))
        optimiser.zero_grad()
        loss.backward()
        optimiser.step()
        epoch_loss += loss.item()
    acc = evaluate(test_dataloader, model)
    print(f'Loss after f{epoch} epochs = {epoch_loss}, Val acc = {100*acc}%')
    if acc > best_acc:
        best_acc = acc
```

```
        best_epoch = epoch
        torch.save(model.state_dict(), f'/kaggle/working/{epoch}.pth')
```

```
Epoch 1:: 100%|      | 125/125 [20:47<00:00,  9.98s/it]
100%|      | 32/32 [05:11<00:00,  9.73s/it]

Loss after f1 epochs = 86.44747442007065, Val acc = 56.599999999999994%

Epoch 2:: 100%|      | 125/125 [20:48<00:00,  9.99s/it]
100%|      | 32/32 [05:11<00:00,  9.73s/it]

Loss after f2 epochs = 85.28908115625381, Val acc = 57.550000000000004%

Epoch 3:: 100%|      | 125/125 [20:54<00:00, 10.04s/it]
100%|      | 32/32 [05:14<00:00,  9.82s/it]

Loss after f3 epochs = 84.45879954099655, Val acc = 57.49999999999999%

Epoch 4:: 100%|      | 125/125 [20:48<00:00,  9.98s/it]
100%|      | 32/32 [05:11<00:00,  9.72s/it]

Loss after f4 epochs = 83.70755857229233, Val acc = 58.75%

Epoch 5:: 100%|      | 125/125 [20:51<00:00, 10.01s/it]
100%|      | 32/32 [05:14<00:00,  9.82s/it]

Loss after f5 epochs = 83.34814894199371, Val acc = 59.4%

Epoch 6:: 100%|      | 125/125 [20:47<00:00,  9.98s/it]
100%|      | 32/32 [05:12<00:00,  9.77s/it]

Loss after f6 epochs = 82.91225200891495, Val acc = 58.8%

Epoch 7:: 100%|      | 125/125 [20:46<00:00,  9.97s/it]
100%|      | 32/32 [05:10<00:00,  9.72s/it]

Loss after f7 epochs = 82.87983494997025, Val acc = 59.8%

Epoch 8:: 100%|      | 125/125 [20:42<00:00,  9.94s/it]
100%|      | 32/32 [05:09<00:00,  9.68s/it]

Loss after f8 epochs = 82.60226565599442, Val acc = 59.35%

Epoch 9:: 100%|      | 125/125 [20:45<00:00,  9.97s/it]
100%|      | 32/32 [05:14<00:00,  9.83s/it]

Loss after f9 epochs = 82.43799072504044, Val acc = 59.650000000000006%

Epoch 10:: 100%|      | 125/125 [20:45<00:00,  9.96s/it]
100%|      | 32/32 [05:11<00:00,  9.73s/it]

Loss after f10 epochs = 82.24004679918289, Val acc = 59.25%

Epoch 11:: 100%|      | 125/125 [20:41<00:00,  9.93s/it]
100%|      | 32/32 [05:10<00:00,  9.70s/it]

Loss after f11 epochs = 82.43570989370346, Val acc = 60.25%
```

```
Epoch 12:: 100%|      | 125/125 [20:39<00:00,  9.92s/it]
100%|      | 32/32 [05:09<00:00,  9.67s/it]

Loss after f12 epochs = 82.04974013566971, Val acc = 59.95%

Epoch 13:: 100%|      | 125/125 [20:50<00:00, 10.01s/it]
100%|      | 32/32 [05:13<00:00,  9.79s/it]

Loss after f13 epochs = 81.91537237167358, Val acc = 59.95%

Epoch 14:: 100%|      | 125/125 [20:47<00:00,  9.98s/it]
100%|      | 32/32 [05:12<00:00,  9.75s/it]

Loss after f14 epochs = 81.78760993480682, Val acc = 59.650000000000006%

Epoch 15:: 100%|      | 125/125 [20:46<00:00,  9.97s/it]
100%|      | 32/32 [05:12<00:00,  9.78s/it]

Loss after f15 epochs = 81.78811627626419, Val acc = 60.35%
```

[ ]:
```
for epoch in range(1, num_epochs+1):
    epoch_loss = 0
    model.train()
    for batch in tqdm(train_dataloader, desc=f'Epoch {num_epochs+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(pred, batch[1].long().to(device))
        optimiser.zero_grad()
        loss.backward()
        optimiser.step()
        epoch_loss += loss.item()
    acc = evaluate(test_dataloader, model)
    print(f'Loss after {num_epochs+epoch} epochs = {epoch_loss}, Val acc =␣
 ↪{100*acc}%')
    if acc > best_acc:
        best_acc = acc
        best_epoch = epoch
        torch.save(model.state_dict(), f'/kaggle/working/{num_epochs+epoch}.
 ↪pth')
```

```
Epoch 16:: 100%|      | 125/125 [20:46<00:00,  9.97s/it]
100%|      | 32/32 [05:10<00:00,  9.71s/it]

Loss after 16 epochs = 81.50043708086014, Val acc = 60.050000000000004%

Epoch 17:: 100%|      | 125/125 [20:44<00:00,  9.96s/it]
100%|      | 32/32 [05:11<00:00,  9.74s/it]
```

```
Loss after 17 epochs = 81.71386760473251, Val acc = 59.4%

Epoch 18:: 100%|      | 125/125 [20:47<00:00,  9.98s/it]
100%|      | 32/32 [05:11<00:00,  9.73s/it]

Loss after 18 epochs = 81.30803221464157, Val acc = 60.35%

Epoch 19:: 100%|      | 125/125 [20:51<00:00, 10.01s/it]
100%|      | 32/32 [05:14<00:00,  9.84s/it]

Loss after 19 epochs = 81.26681661605835, Val acc = 60.25%

Epoch 20:: 100%|      | 125/125 [20:59<00:00, 10.08s/it]
100%|      | 32/32 [05:14<00:00,  9.83s/it]

Loss after 20 epochs = 81.20397907495499, Val acc = 60.550000000000004%

Epoch 21:: 100%|      | 125/125 [20:56<00:00, 10.05s/it]
100%|      | 32/32 [05:12<00:00,  9.77s/it]

Loss after 21 epochs = 81.19562822580338, Val acc = 60.199999999999996%

Epoch 22:: 100%|      | 125/125 [20:53<00:00, 10.03s/it]
100%|      | 32/32 [05:13<00:00,  9.81s/it]

Loss after 22 epochs = 80.97035294771194, Val acc = 59.650000000000006%

Epoch 23:: 100%|      | 125/125 [21:02<00:00, 10.10s/it]
100%|      | 32/32 [05:15<00:00,  9.85s/it]

Loss after 23 epochs = 80.81147682666779, Val acc = 60.3%

Epoch 24:: 100%|      | 125/125 [20:59<00:00, 10.07s/it]
100%|      | 32/32 [05:14<00:00,  9.83s/it]

Loss after 24 epochs = 80.93522256612778, Val acc = 60.35%

Epoch 25:: 100%|      | 125/125 [21:00<00:00, 10.09s/it]
100%|      | 32/32 [05:14<00:00,  9.82s/it]

Loss after 25 epochs = 80.62047773599625, Val acc = 60.550000000000004%

Epoch 26:: 100%|      | 125/125 [20:58<00:00, 10.07s/it]
100%|      | 32/32 [05:15<00:00,  9.85s/it]

Loss after 26 epochs = 80.64774167537689, Val acc = 60.35%

Epoch 27:: 100%|      | 125/125 [21:01<00:00, 10.09s/it]
100%|      | 32/32 [05:15<00:00,  9.86s/it]

Loss after 27 epochs = 80.67484217882156, Val acc = 60.0%

Epoch 28::  49%|        | 61/125 [10:16<10:44, 10.08s/it]
```

[ ]:

[ ]:

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```