ml4sci-task2-model2

March 18, 2024

[1]:
```
!pip install pyarrow
```

```
Requirement already satisfied: pyarrow in /usr/local/lib/python3.10/dist-
packages (14.0.2)
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.10/dist-
packages (from pyarrow) (1.25.2)
```

[2]:
```
import pandas as pd
import pyarrow.parquet as parquet
first = 'QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.snappy.parquet'
second = 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.snappy.parquet'

first_file = parquet.ParquetFile('/content/drive/MyDrive/Sci_data/'+first)
```

0.0.1 Since the size of the raw data is very large, We will first visualise a small chunk of X_jet

[3]:
```
chunk_size = 12000
# batches_df = []

for batch in first_file.iter_batches(chunk_size):
    print("RecordBatch")
    batch_df = batch.to_pandas()
    # batches_df.append(batch_df)
    break
    # print("batch_df:", batch_df)
```

```
RecordBatch
```

[4]:
```
batch_df['y'][9]
```

[4]:
```
1.0
```

0.1 Visualizing X_jets

[5]:
```
from torch.utils.data import Dataset

class ImageData(Dataset):
  def __init__(self, img_data, transform):
    self.img_list = []
    self.img_data=img_data
    self.transform=transform
    for number in range(img_data.shape[0]):
      for idx, channels in enumerate(batch_df['X_jets'][number]):
        for i, row in enumerate(channels):
          if i==0:
            img = row
          else:
            img = np.vstack([img, row])
        if idx==0:
          final_img = img
        else:
          final_img = np.dstack([final_img, img])
      self.img_list.append(final_img)

  def __len__(self):
    return len(self.img_list)

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

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

transform = Compose([
    ToTensor(),
    Resize((32,32)),
])
```

[7]:
```
import numpy as np
data = ImageData(batch_df, transform)
```

[8]:
```
del batch_df
```

[9]:
```
from torchvision.models import resnet18
```

[10]:
```
model = resnet18(weights=True)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
```

```
equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
/root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|      | 44.7M/44.7M [00:00<00:00, 59.3MB/s]
```

[11]:
```
import torch.nn as nn

model.fc = nn.Linear(512, 1)

# Enabling gradient for all parameters gives better results
for p in model.parameters():
    p.requires_grad = True
```

[12]:
```
device=torch.device("cuda") if torch.cuda.is_available() else 'cpu'
```

[13]:
```
model = model.to(device)
```

[14]:
```
from torch.utils.data import DataLoader, random_split

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

[15]:
```
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)
```

[16]:
```
from tqdm import tqdm

criterion = nn.BCEWithLogitsLoss()
optimiser = torch.optim.AdamW(model.parameters(), lr=3e-4)
num_epochs = 25
```

[17]:
```
def evaluate(loader):
    model.eval()
    total_samples = 0
    correct_samples = 0
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            pred = model(inputs.float())  # Forward pass
            predicted_labels = (pred.sigmoid().round())
            correct_samples += (predicted_labels.squeeze(-1) == labels).sum().
↪item()
            total_samples += labels.size(0)
    return correct_samples / total_samples * 100
```

[18]:
```python
best_acc = 0
for epoch in range(1, num_epochs+1):
    epoch_loss = 0
    model.train()
    for inputs, labels in tqdm(train_dataloader, desc=f'Epoch {epoch}:'):
        inputs, labels = inputs.to(device), labels.to(device)
        optimiser.zero_grad()
        outputs = model(inputs.float())
        loss = criterion(outputs.squeeze(), labels.float())
        loss.backward()
        optimiser.step()
        epoch_loss += loss.item()
    acc = evaluate(test_dataloader)
    if acc > best_acc:
      best_epoch = epoch
      best_acc= acc
      torch.save(model.state_dict(), 'best_model_resnet.pth')
    print(f'Epoch {epoch}: Loss = {epoch_loss:.4f}, Test Accuracy = {acc:.2f}%')
```

```
Epoch 1:: 100%|      | 150/150 [02:10<00:00,  1.15it/s]

Epoch 1: Loss = 93.4887, Test Accuracy = 69.92%

Epoch 2:: 100%|      | 150/150 [02:09<00:00,  1.16it/s]

Epoch 2: Loss = 86.4873, Test Accuracy = 71.96%

Epoch 3:: 100%|      | 150/150 [02:09<00:00,  1.16it/s]

Epoch 3: Loss = 84.5296, Test Accuracy = 72.42%

Epoch 4:: 100%|      | 150/150 [02:08<00:00,  1.17it/s]

Epoch 4: Loss = 82.6285, Test Accuracy = 71.17%

Epoch 5:: 100%|      | 150/150 [02:08<00:00,  1.17it/s]

Epoch 5: Loss = 81.5309, Test Accuracy = 72.08%

Epoch 6:: 100%|      | 150/150 [02:08<00:00,  1.17it/s]

Epoch 6: Loss = 79.0897, Test Accuracy = 72.12%

Epoch 7:: 100%|      | 150/150 [02:09<00:00,  1.16it/s]

Epoch 7: Loss = 76.8161, Test Accuracy = 72.00%

Epoch 8:: 100%|      | 150/150 [02:09<00:00,  1.16it/s]

Epoch 8: Loss = 75.8674, Test Accuracy = 65.62%

Epoch 9:: 100%|      | 150/150 [02:09<00:00,  1.16it/s]

Epoch 9: Loss = 74.2520, Test Accuracy = 70.38%

Epoch 10:: 100%|      | 150/150 [02:08<00:00,  1.17it/s]
```

```
Epoch 10: Loss = 70.5479, Test Accuracy = 62.79%
Epoch 11:: 100%|     | 150/150 [02:08<00:00,  1.17it/s]
Epoch 11: Loss = 68.5160, Test Accuracy = 71.00%
Epoch 12:: 100%|     | 150/150 [02:08<00:00,  1.17it/s]
Epoch 12: Loss = 64.6753, Test Accuracy = 69.25%
Epoch 13:: 100%|     | 150/150 [02:09<00:00,  1.16it/s]
Epoch 13: Loss = 62.6169, Test Accuracy = 57.63%
Epoch 14:: 100%|     | 150/150 [02:08<00:00,  1.17it/s]
Epoch 14: Loss = 57.6717, Test Accuracy = 67.12%
Epoch 15:: 100%|     | 150/150 [02:08<00:00,  1.17it/s]
Epoch 15: Loss = 54.9255, Test Accuracy = 70.21%
Epoch 16:: 100%|     | 150/150 [02:08<00:00,  1.16it/s]
Epoch 16: Loss = 49.7635, Test Accuracy = 68.79%
Epoch 17:: 100%|     | 150/150 [02:09<00:00,  1.16it/s]
Epoch 17: Loss = 46.0939, Test Accuracy = 69.38%
Epoch 18:: 100%|     | 150/150 [02:09<00:00,  1.16it/s]
Epoch 18: Loss = 42.4472, Test Accuracy = 66.38%
Epoch 19:: 100%|     | 150/150 [02:08<00:00,  1.16it/s]
Epoch 19: Loss = 38.4205, Test Accuracy = 67.62%
Epoch 20:: 100%|     | 150/150 [02:09<00:00,  1.16it/s]
Epoch 20: Loss = 34.0486, Test Accuracy = 69.29%
Epoch 21:: 100%|     | 150/150 [02:09<00:00,  1.16it/s]
Epoch 21: Loss = 30.0833, Test Accuracy = 63.42%
Epoch 22:: 100%|     | 150/150 [02:10<00:00,  1.15it/s]
Epoch 22: Loss = 29.5587, Test Accuracy = 62.96%
Epoch 23:: 100%|     | 150/150 [02:08<00:00,  1.16it/s]
Epoch 23: Loss = 24.9666, Test Accuracy = 66.33%
Epoch 24:: 100%|     | 150/150 [02:08<00:00,  1.17it/s]
Epoch 24: Loss = 23.1163, Test Accuracy = 67.58%
Epoch 25:: 100%|     | 150/150 [02:08<00:00,  1.17it/s]
Epoch 25: Loss = 22.6408, Test Accuracy = 69.00%
```

[23]:
```
del model
```

[24]:
```
model = resnet18()
model.fc = nn.Linear(512, 1)
```

[25]:
```
state_dict = torch.load('best_model_resnet.pth')
model.load_state_dict(state_dict)
```

[25]:
```
<All keys matched successfully>
```

[26]:
```
print(f'Best accuracy = {best_acc} at epoch {best_epoch}')
```

```
Best accuracy = 72.41666666666666 at epoch 3
```

[22]:
```

```