ml4sci-task2

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 Schema of the dataset

[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]:
```
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]
```

[5]:
```
import torch
from torchvision.transforms import ToTensor, Compose, Resize, Lambda

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

[6]:
```
import numpy as np
data = ImageData(batch_df, transform)
```

[7]:
```
del batch_df
```

[8]:
```
from torchvision.models import vgg11
```

[9]:
```
model = vgg11(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=VGG11_Weights.IMAGENET1K_V1`. You can also use
`weights=VGG11_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

0.0.2 12 Layer VGG Model

[10]:
```
model
```

[10]:
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

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

num_features = model.classifier[0].in_features
# add ing 2 classification layers to make 12 layers in total
model.classifier = nn.Sequential(
    nn.Linear(num_features, 1024),
    nn.ReLU(),
    nn.Linear(1024, 1)
)
```

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

[12]:
```
device=torch.device("cuda")
```

[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 = 15
```

[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())
            predicted_labels = (pred.sigmoid().round())
            correct_samples += (predicted_labels.squeeze(-1) == labels).sum().
 ↪item()
            total_samples += labels.shape[0]
    return correct_samples / total_samples * 100
```

[18]:
```
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_vgg.pth')
    print(f'Epoch {epoch}: Loss = {epoch_loss:.4f}, Test Accuracy = {acc:.2f}%')
```

```
Epoch 1:: 100%|      | 150/150 [00:10<00:00, 14.90it/s]
Epoch 1: Loss = 93.1758, Test Accuracy = 71.67%
Epoch 2:: 100%|      | 150/150 [00:08<00:00, 17.31it/s]
Epoch 2: Loss = 86.0021, Test Accuracy = 70.58%
Epoch 3:: 100%|      | 150/150 [00:08<00:00, 17.25it/s]
Epoch 3: Loss = 84.5525, Test Accuracy = 71.50%
Epoch 4:: 100%|      | 150/150 [00:09<00:00, 16.18it/s]
Epoch 4: Loss = 83.9741, Test Accuracy = 70.33%
Epoch 5:: 100%|      | 150/150 [00:08<00:00, 18.12it/s]
Epoch 5: Loss = 82.9197, Test Accuracy = 71.67%
Epoch 6:: 100%|      | 150/150 [00:08<00:00, 17.70it/s]
Epoch 6: Loss = 81.6663, Test Accuracy = 72.21%
Epoch 7:: 100%|      | 150/150 [00:08<00:00, 17.05it/s]
Epoch 7: Loss = 79.2461, Test Accuracy = 72.21%
Epoch 8:: 100%|      | 150/150 [00:08<00:00, 17.26it/s]
Epoch 8: Loss = 77.7110, Test Accuracy = 72.17%
Epoch 9:: 100%|      | 150/150 [00:08<00:00, 17.69it/s]
Epoch 9: Loss = 75.9713, Test Accuracy = 71.62%
Epoch 10:: 100%|      | 150/150 [00:08<00:00, 18.17it/s]
Epoch 10: Loss = 72.6666, Test Accuracy = 69.54%
Epoch 11:: 100%|      | 150/150 [00:08<00:00, 17.04it/s]
Epoch 11: Loss = 70.5110, Test Accuracy = 69.12%
Epoch 12:: 100%|      | 150/150 [00:08<00:00, 17.18it/s]
Epoch 12: Loss = 65.8131, Test Accuracy = 68.17%
```

```
Epoch 13:: 100%|      | 150/150 [00:08<00:00, 17.61it/s]
Epoch 13: Loss = 60.3439, Test Accuracy = 69.25%
Epoch 14:: 100%|      | 150/150 [00:08<00:00, 18.41it/s]
Epoch 14: Loss = 56.3622, Test Accuracy = 69.50%
Epoch 15:: 100%|      | 150/150 [00:08<00:00, 17.18it/s]
Epoch 15: Loss = 49.6004, Test Accuracy = 69.88%
```

[19]:
```
del model
```

[20]:
```
model = vgg11()

model.classifier = nn.Sequential(
    nn.Linear(25088, 1024),
    nn.ReLU(),
    nn.Linear(1024, 1)
)
```

[21]:
```
state_dict = torch.load('best_model_vgg.pth')
model.load_state_dict(state_dict)
```

[21]:
```
<All keys matched successfully>
```

[21]:
```

```