olives_project_augmented

April 25, 2025

[1]:
```
import torch
from torch.utils.data import Dataset, DataLoader, random_split, Subset
import torchvision.transforms as T
import torch.nn as nn
import torchvision.models as models
import torch.optim as optim
from datasets import load_dataset
from sklearn.metrics import f1_score, accuracy_score
import numpy as np
import os
```

```
c:\Users\siyin\AppData\Local\Programs\Python\Python310\lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[2]:
```
class OLIVESDataset(Dataset):
    def __init__(self, hf_dataset, transform=None):
        self.transform = transform
        self.filtered_data = []
        for sample in hf_dataset:
            if any(sample.get(k) is None for k in ["B1", "B2", "B3", "B4",␣
 ↪"B5", "B6"]):
                continue
            if sample.get("BCVA") is None or sample.get("CST") is None:
                continue
            self.filtered_data.append(sample)

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

    def __getitem__(self, idx):
        sample = self.filtered_data[idx]
        image = sample["Image"].convert("L")
        if self.transform:
            image = self.transform(image)
```

```
        labels = torch.tensor([sample[f"B{i}"] for i in range(1, 7)],␣
↪dtype=torch.float32)
        extra_features = torch.tensor([sample["BCVA"], sample["CST"]],␣
↪dtype=torch.float32)
        return image, extra_features, labels
```

[3]:
```
def prepare_data_simple(sample_size=1000, batch_size=16):
    olives = load_dataset("gOLIVES/OLIVES_Dataset", "biomarker_detection")

    train_transform = T.Compose([
        T.Resize((256, 256)),
        T.RandomResizedCrop(224, scale=(0.8, 1.0)),
        T.RandomHorizontalFlip(p=0.5),
        T.RandomRotation(degrees=10),
        T.ColorJitter(brightness=0.2, contrast=0.2),
        T.ToTensor()
    ])

    test_transform = T.Compose([
        T.Resize((224, 224)),
        T.ToTensor()
    ])

    small_train_data = olives["train"].select(range(sample_size))
    full_dataset = OLIVESDataset(hf_dataset=small_train_data, transform=None)

    train_size = int(0.8 * len(full_dataset))
    val_size = len(full_dataset) - train_size
    train_subset, val_subset = random_split(full_dataset, [train_size,␣
↪val_size])

    # wrap again with transform
    train_dataset = OLIVESDataset(hf_dataset=[full_dataset.filtered_data[i] for␣
↪i in train_subset.indices], transform=train_transform)
    val_dataset = OLIVESDataset(hf_dataset=[full_dataset.filtered_data[i] for i␣
↪in val_subset.indices], transform=test_transform)

    train_loader = DataLoader(train_dataset, batch_size=batch_size,␣
↪shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size)

    test_dataset = OLIVESDataset(hf_dataset=olives["test"],␣
↪transform=test_transform)
    test_loader = DataLoader(test_dataset, batch_size=batch_size)

    return train_loader, val_loader, test_loader, train_dataset
```

[4]:
```
class MultimodalNet(nn.Module):
    def __init__(self):
        super().__init__()
        resnet = models.resnet50(pretrained=True)
        self.cnn = nn.Sequential(*list(resnet.children())[:-1])
        self.img_out_dim = 2048
        self.extra_mlp = nn.Sequential(nn.Linear(2, 64), nn.ReLU(), nn.
 ↪Linear(64, 128), nn.ReLU())
        self.fusion = nn.Sequential(
            nn.Linear(self.img_out_dim + 128, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 6)
        )

    def forward(self, image, extra_features):
        if image.shape[1] == 1:
            image = image.repeat(1, 3, 1, 1)
        img_feat = self.cnn(image).view(image.size(0), -1)
        extra_feat = self.extra_mlp(extra_features)
        combined = torch.cat((img_feat, extra_feat), dim=1)
        return self.fusion(combined)
```

[5]:
```
def train_one_fold(model, train_loader, val_loader, fold=0, device="cuda",␣
 ↪num_epochs=10, lr=1e-4, save_dir="checkpoints"):
    model = model.to(device)
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    best_val_loss = float("inf")
    patience_counter = 0
    os.makedirs(save_dir, exist_ok=True)
    save_path = os.path.join(save_dir, f"best_model_fold_{fold+1}.pt")

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for images, extra_features, labels in train_loader:
            images, extra_features, labels = images.to(device), extra_features.
 ↪to(device), labels.to(device)
            optimizer.zero_grad()
            loss = criterion(model(images, extra_features), labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * images.size(0)

        val_loss = 0.0
        model.eval()
```

```
        with torch.no_grad():
            for images, extra_features, labels in val_loader:
                images, extra_features, labels = images.to(device),␣
↪extra_features.to(device), labels.to(device)
                val_loss += criterion(model(images, extra_features), labels).
↪item() * images.size(0)

        avg_val_loss = val_loss / len(val_loader.dataset)
        print(f"Fold {fold+1} | Epoch {epoch+1} | Train Loss: {running_loss/
↪len(train_loader.dataset):.4f} | Val Loss: {avg_val_loss:.4f}")

        if avg_val_loss < best_val_loss:
            torch.save(model.state_dict(), save_path)
            best_val_loss = avg_val_loss
            patience_counter = 0
        else:
            patience_counter += 1

    return best_val_loss
```

[6]:
```
def evaluate_on_test(model, test_loader, device="cuda"):
    model = model.to(device).eval()
    criterion = nn.BCEWithLogitsLoss()
    all_preds, all_labels = [], []
    test_loss = 0.0
    with torch.no_grad():
        for images, extra_features, labels in test_loader:
            images, extra_features, labels = images.to(device), extra_features.
↪to(device), labels.to(device)
            outputs = model(images, extra_features)
            test_loss += criterion(outputs, labels).item() * images.size(0)
            preds = torch.sigmoid(outputs) > 0.5
            all_preds.append(preds.cpu())
            all_labels.append(labels.cpu())
    all_preds = torch.cat(all_preds).numpy()
    all_labels = torch.cat(all_labels).numpy()
    print(f"\n Test Loss: {test_loss / len(test_loader.dataset):.4f}")
    print(f" Test Accuracy: {accuracy_score(all_labels, all_preds):.4f}")
    print(f" Test F1 Score (macro): {f1_score(all_labels, all_preds,␣
↪average='macro'):.4f}")
```

[7]:
```
if __name__ == "__main__":
    sample_sizes = [75000]
    for sample_size in sample_sizes:
        train_loader, val_loader, test_loader, _ =␣
↪prepare_data_simple(sample_size=sample_size)
        model = MultimodalNet()
```

```
        train_one_fold(model, train_loader, val_loader, fold=0, num_epochs=35)
        evaluate_on_test(model, test_loader)
```

```
c:\Users\siyin\AppData\Local\Programs\Python\Python310\lib\site-
packages\torchvision\models\_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
c:\Users\siyin\AppData\Local\Programs\Python\Python310\lib\site-
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=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
Fold 1 | Epoch 1 | Train Loss: 0.4038 | Val Loss: 0.2799
Fold 1 | Epoch 2 | Train Loss: 0.3069 | Val Loss: 0.2564
Fold 1 | Epoch 3 | Train Loss: 0.2760 | Val Loss: 0.2458
Fold 1 | Epoch 4 | Train Loss: 0.2515 | Val Loss: 0.2146
Fold 1 | Epoch 5 | Train Loss: 0.2317 | Val Loss: 0.1955
Fold 1 | Epoch 6 | Train Loss: 0.2169 | Val Loss: 0.1888
Fold 1 | Epoch 7 | Train Loss: 0.2019 | Val Loss: 0.1842
Fold 1 | Epoch 8 | Train Loss: 0.1872 | Val Loss: 0.1773
Fold 1 | Epoch 9 | Train Loss: 0.1800 | Val Loss: 0.1999
Fold 1 | Epoch 10 | Train Loss: 0.1710 | Val Loss: 0.1697
Fold 1 | Epoch 11 | Train Loss: 0.1581 | Val Loss: 0.1703
Fold 1 | Epoch 12 | Train Loss: 0.1478 | Val Loss: 0.1539
Fold 1 | Epoch 13 | Train Loss: 0.1451 | Val Loss: 0.1486
Fold 1 | Epoch 14 | Train Loss: 0.1369 | Val Loss: 0.1687
Fold 1 | Epoch 15 | Train Loss: 0.1285 | Val Loss: 0.1444
Fold 1 | Epoch 16 | Train Loss: 0.1237 | Val Loss: 0.1514
Fold 1 | Epoch 17 | Train Loss: 0.1188 | Val Loss: 0.1285
Fold 1 | Epoch 18 | Train Loss: 0.1139 | Val Loss: 0.1223
Fold 1 | Epoch 19 | Train Loss: 0.1096 | Val Loss: 0.1295
Fold 1 | Epoch 20 | Train Loss: 0.1052 | Val Loss: 0.1176
Fold 1 | Epoch 21 | Train Loss: 0.0994 | Val Loss: 0.1310
Fold 1 | Epoch 22 | Train Loss: 0.0952 | Val Loss: 0.1255
Fold 1 | Epoch 23 | Train Loss: 0.0912 | Val Loss: 0.1166
Fold 1 | Epoch 24 | Train Loss: 0.0876 | Val Loss: 0.1203
Fold 1 | Epoch 25 | Train Loss: 0.0826 | Val Loss: 0.1086
Fold 1 | Epoch 26 | Train Loss: 0.0810 | Val Loss: 0.1207
Fold 1 | Epoch 27 | Train Loss: 0.0775 | Val Loss: 0.1064
Fold 1 | Epoch 28 | Train Loss: 0.0757 | Val Loss: 0.1045
Fold 1 | Epoch 29 | Train Loss: 0.0722 | Val Loss: 0.1062
Fold 1 | Epoch 30 | Train Loss: 0.0683 | Val Loss: 0.1070
Fold 1 | Epoch 31 | Train Loss: 0.0714 | Val Loss: 0.1024
Fold 1 | Epoch 32 | Train Loss: 0.0669 | Val Loss: 0.1022
```

```
Fold 1 | Epoch 33 | Train Loss: 0.0610 | Val Loss: 0.1208
Fold 1 | Epoch 34 | Train Loss: 0.0639 | Val Loss: 0.1131
Fold 1 | Epoch 35 | Train Loss: 0.0575 | Val Loss: 0.1052

 Test Loss: 0.8502
 Test Accuracy: 0.2979
 Test F1 Score (macro): 0.6339
```