validation_tqdm

January 17, 2024

[ ]:
```
import torch
import torch
import torch.nn as nn
from torch.nn import Linear, Conv2d, BatchNorm1d, BatchNorm2d, PReLU,␣
↪Sequential, Module
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import torch.optim as optim
from tqdm import tqdm
from sklearn.model_selection import ParameterGrid

device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
# device = torch.device("cpu")
print(f"Using device: {device}")
```

```
Using device: mps
```

[ ]:
```
import os
import pandas as pd
from PIL import Image
from torch.utils.data import Dataset
import torch

class RAFDBDataset(Dataset):
    def __init__(self, csv_file, img_dir, transform=None):
        self.labels = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform

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

    def __getitem__(self, idx):
```

```
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.img_dir, self.labels.iloc[idx, 0])
        image = Image.open(img_name)
        label = self.labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)

        return image, label
```

[ ]:
```
from rafdb_dataset import RAFDBDataset

transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.Grayscale(num_output_channels=3),
    # transforms.RandomHorizontalFlip(),
    # transforms.RandomApply([
    #     transforms.RandomRotation(5),
    #     transforms.RandomCrop(64, padding=8)
    # ], p=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    # transforms.RandomErasing(scale=(0.02,0.25)),
])

rafdb_dataset_train = RAFDBDataset(csv_file='archive/train_labels.csv',
                            img_dir='archive/DATASET/train/',
                            transform=transform)
data_train_loader = DataLoader(rafdb_dataset_train, batch_size=16,␣
 ↪shuffle=True, num_workers=4)
train_image, train_label = next(iter(data_train_loader))
print(f"Train batch: image shape {train_image.shape}, labels shape {train_label.
 ↪shape}")

rafdb_dataset_vali = RAFDBDataset(csv_file='dataset/vali_labels.csv',
                            img_dir='dataset/validation_set',
                            transform=transform)
data_vali_loader = DataLoader(rafdb_dataset_vali, batch_size=16, shuffle=False,␣
 ↪num_workers=0)
vali_image, vali_label = next(iter(data_vali_loader))
print(f"Vali batch: image shape {vali_image.shape}, labels shape {vali_label.
 ↪shape}")

rafdb_dataset_test = RAFDBDataset(csv_file='archive/test_labels.csv',
                            img_dir='archive/DATASET/test',
                            transform=transform)
```

```
data_test_loader = DataLoader(rafdb_dataset_test, batch_size=16, shuffle=False,␣
↪num_workers=0)
test_image, test_label = next(iter(data_test_loader))
print(f"Test batch: image shape {test_image.shape}, labels shape {test_label.
↪shape}")
```

```
Train batch: image shape torch.Size([16, 3, 64, 64]), labels shape
torch.Size([16])
Vali batch: image shape torch.Size([16, 3, 64, 64]), labels shape
torch.Size([16])
Test batch: image shape torch.Size([16, 3, 64, 64]), labels shape
torch.Size([16])
```

[ ]:
```
# for images, labels in data_train_loader:
#     labels = labels - 1
#     if labels.min() < 0 or labels.max() > 5:
#         print("Found label outside the expected range [0, 5]")
#         break

# for images, labels in data_vali_loader:
#     labels = labels - 1
#     if labels.min() < 0 or labels.max() > 5:
#         print("Found label outside the expected range [0, 5]")
#         break

# for images, labels in data_test_loader:
#     labels = labels - 1
#     if labels.min() < 0 or labels.max() > 5:
#         print("Found label outside the expected range [0, 5]")
#         break
```

[ ]:
```
class SEBlock(nn.Module): # Squeeze-and-Excitation (SE) blocks apply␣
↪channel-wise attention.
    def __init__(self, input_channels, reduction=16):
        super(SEBlock, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Linear(input_channels, input_channels // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(input_channels // reduction, input_channels, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
```

```
        return x * y.expand_as(x)
```

[ ]:
```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,␣
↪stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,␣
↪stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1,␣
↪stride=stride, padding=0),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

[ ]:
```
# Residual
# class EmotionClassifier(nn.Module):
#     def __init__(self):
#         super(EmotionClassifier, self).__init__()
#         self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
#         self.bn1 = nn.BatchNorm2d(64)
#         self.relu = nn.ReLU(inplace=True)
#         self.se1 = SEBlock(64)

#         # Using Residual Blocks
#         self.res_block1 = ResidualBlock(64, 128, stride=2)
#         self.res_block2 = ResidualBlock(128, 256, stride=2)
#         self.res_block3 = ResidualBlock(256, 512, stride=2)
#         self.res_block4 = ResidualBlock(512, 1024, stride=2)

#         self.pool = nn.AdaptiveAvgPool2d((1, 1))
#         self.fc1 = nn.Linear(1024, 2048)
#         self.fc2 = nn.Linear(2048, 1024)
#         self.dropout1 = nn.Dropout(0.5)
#         self.fc3 = nn.Linear(1024, 6)
```

```
#     def forward(self, x):
#         x = self.relu(self.bn1(self.conv1(x)))
#         x = self.se1(x)

#         x = self.res_block1(x)
#         x = self.res_block2(x)
#         x = self.res_block3(x)
#         x = self.res_block4(x)

#         x = self.pool(x)
#         x = x.view(x.size(0), -1)
#         x = F.relu(self.fc1(x))
#         x = self.dropout1(x)
#         x = F.relu(self.fc2(x))
#         x = self.fc3(x)
#         return x

# model = EmotionClassifier().to(device)
```

[ ]:
```
class EmotionClassifier(nn.Module):
    def __init__(self):
        super(EmotionClassifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        # self.se1 = SEBlock(64)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        self.conv5 = nn.Conv2d(512, 1024, kernel_size=3, padding=1)
        self.bn5 = nn.BatchNorm2d(1024)

        self.pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc1 = nn.Linear(1024, 2048)
        self.fc2 = nn.Linear(2048, 1024)
        self.dropout1 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(1024, 6)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        # x = self.se1(x)
        # x = F.relu(self.se1(self.conv1(x)))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.bn2(self.conv2(x)))
```

```python
        x = F.max_pool2d(x, 2)
        x = F.relu(self.bn3(self.conv3(x)))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.bn4(self.conv4(x)))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.bn5(self.conv5(x)))
        x = F.max_pool2d(x, 2)

        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = EmotionClassifier().to(device)
```

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

# class BasicBlock(nn.Module):
#     expansion = 1

#     def __init__(self, in_planes, planes, stride=1):
#         super(BasicBlock, self).__init__()
#         self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3,␣
↪stride=stride, padding=1, bias=False)
#         self.bn1 = nn.BatchNorm2d(planes)
#         self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1,␣
↪padding=1, bias=False)
#         self.bn2 = nn.BatchNorm2d(planes)

#         self.shortcut = nn.Sequential()
#         if stride != 1 or in_planes != self.expansion * planes:
#             self.shortcut = nn.Sequential(
#                 nn.Conv2d(in_planes, self.expansion * planes, kernel_size=1,␣
↪stride=stride, bias=False),
#                 nn.BatchNorm2d(self.expansion * planes)
#             )

#     def forward(self, x):
#         out = F.relu(self.bn1(self.conv1(x)))
#         out = self.bn2(self.conv2(out))
#         out += self.shortcut(x)
#         out = F.relu(out)
```

```
#         return out

# class ResNet(nn.Module):
#     def __init__(self, block, num_blocks, num_classes=6):
#         super(ResNet, self).__init__()
#         self.in_planes = 64

#         self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,␣
↪bias=False)
#         self.bn1 = nn.BatchNorm2d(64)
#         self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
#         self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
#         self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
#         self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
#         self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
#         self.fc = nn.Linear(512 * block.expansion, num_classes)

#     def _make_layer(self, block, planes, num_blocks, stride):
#         strides = [stride] + [1]*(num_blocks-1)
#         layers = []
#         for stride in strides:
#             layers.append(block(self.in_planes, planes, stride))
#             self.in_planes = planes * block.expansion
#         return nn.Sequential(*layers)

#     def forward(self, x):
#         out = F.relu(self.bn1(self.conv1(x)))
#         out = self.layer1(out)
#         out = self.layer2(out)
#         out = self.layer3(out)
#         out = self.layer4(out)
#         out = self.avgpool(out)
#         out = out.view(out.size(0), -1)
#         out = self.fc(out)
#         return out

# def EmotionClassifierResNet18():
#     return ResNet(BasicBlock, [2, 2, 2, 2])

# model = EmotionClassifierResNet18().to(device)
```

[ ]:
```
# import torch.nn as nn
# import torch.nn.functional as F

# class VGGEmotionClassifier(nn.Module):
#     def __init__(self):
#         super(VGGEmotionClassifier, self).__init__()
```

```
#         self.features = nn.Sequential(
#             nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3,␣
↪padding=1),
#             nn.BatchNorm2d(64),
#             nn.ReLU(inplace=True),
#             nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,␣
↪padding=1),
#             nn.BatchNorm2d(64),
#             nn.ReLU(inplace=True),
#             nn.MaxPool2d(kernel_size=2, stride=2),

#             nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3,␣
↪padding=1),
#             nn.BatchNorm2d(128),
#             nn.ReLU(inplace=True),
#             nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3,␣
↪padding=1),
#             nn.BatchNorm2d(128),
#             nn.ReLU(inplace=True),
#             nn.MaxPool2d(kernel_size=2, stride=2),

#             nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3,␣
↪padding=1),
#             nn.BatchNorm2d(256),
#             nn.ReLU(inplace=True),
#             nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3,␣
↪padding=1),
#             nn.BatchNorm2d(256),
#             nn.ReLU(inplace=True),
#             nn.MaxPool2d(kernel_size=2, stride=2),
#         )

#         self.classifier = nn.Sequential(
#             nn.Linear(16384, 4096),
#             nn.ReLU(inplace=True),
#             nn.Dropout(0.5),
#             nn.Linear(4096, 1024),
#             nn.ReLU(inplace=True),
#             nn.Dropout(0.5),
#             nn.Linear(1024, 6)
#         )

#     def forward(self, x):
#         x = self.features(x)
#         x = x.view(x.size(0), -1)
```

```
#         x = self.classifier(x)
#         return x

# model = VGGEmotionClassifier().to(device)
```

[ ]:
```
# param_grid = {
#     'lr': [0.1, 0.01, 0.001, 0.0001],
#     'batch_size': [8, 16, 32, 64],
# }
# grid = ParameterGrid(param_grid)
# results = []
```

[ ]:
```
# for params in grid: # Hyperparameter tuning
#     data_train_loader = DataLoader(rafdb_dataset_train,␣
↪batch_size=params['batch_size'], shuffle=True, num_workers=4)
#     data_vali_loader = DataLoader(rafdb_dataset_vali,␣
↪batch_size=params['batch_size'], shuffle=False, num_workers=0)

#     model = EmotionClassifier().to(device)
#     optimizer = optim.Adam(model.parameters(), lr=params['lr'])
#     criterion = nn.CrossEntropyLoss()

#     best_val_acc = 0
#     num_epochs = 15

#     for epoch in range(num_epochs):
#         model.train()
#         for i, data in enumerate(tqdm(data_train_loader, desc=f"Epoch␣
↪{epoch+1}/{num_epochs}"), 0):
#             inputs, labels = data[0].to(device), data[1].to(device)
#             optimizer.zero_grad()
#             outputs = model(inputs)
#             loss = criterion(outputs, labels)
#             loss.backward()
#             optimizer.step()

#     model.eval()
#     val_correct = 0
#     val_total = 0
#     with torch.no_grad():
#         for data in data_vali_loader:
#             inputs, labels = data[0].to(device), data[1].to(device)
#             outputs = model(inputs)
#             _, predicted = torch.max(outputs.data, 1)
#             val_total += labels.size(0)
#             val_correct += (predicted == labels).sum().item()
```

```
#     val_acc = val_correct / val_total
#     best_val_acc = max(best_val_acc, val_acc)

#     results.append({
#         'lr': params['lr'],
#         'batch_size': params['batch_size'],
#         'best_val_acc': best_val_acc,
#     })

# for result in results:
#     print(f"LR: {result['lr']}, Batch Size: {result['batch_size']}, Best Val␣
↪Acc: {result['best_val_acc']}")

# best_params = max(results, key=lambda x: x['best_val_acc'])
# print(f"Best params: {best_params}")
```

[ ]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.
↪1) # test 5/0.5 later

patience = 5
best_val_acc = 0
patience_counter = 0

num_epochs = 40
```

[ ]:
```
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []
test_losses = []
test_accuracies = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for data in tqdm(data_train_loader, desc=f"Epoch {epoch+1}/{num_epochs}"):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
```

```
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_loss = running_loss / len(data_train_loader)
    train_acc = correct / total
    train_losses.append(train_loss)
    train_accuracies.append(train_acc)

    model.eval()
    test_running_loss = 0.0
    test_correct = 0
    test_total = 0
    with torch.no_grad():
        for data in data_test_loader:
            inputs, labels = data[0].to(device), data[1].to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_loss = test_running_loss / len(data_test_loader)
    test_acc = test_correct / test_total
    test_losses.append(test_loss)
    test_accuracies.append(test_acc)

    model.eval()
    val_running_loss = 0.0
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for data in data_vali_loader:
            inputs, labels = data[0].to(device), data[1].to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            val_total += labels.size(0)
            val_correct += (predicted == labels).sum().item()

    val_loss = val_running_loss / len(data_vali_loader)
    val_acc = val_correct / val_total
```

```
    val_losses.append(val_loss)
    val_accuracies.append(val_acc)

    print(f"Epoch {epoch+1}, Train Loss: {train_loss}, Train Accuracy:␣
↪{train_acc}, Test Loss: {test_loss}, Test Accuracy: {test_acc}, Validation␣
↪Loss: {val_loss}, Validation Accuracy: {val_acc}")

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        patience_counter = 0
        torch.save(model.state_dict(), 'best_vgg.pth')
    else:
        patience_counter += 1
        print(f"No improvement in validation accuracy for {patience_counter}␣
↪epochs.")

    if patience_counter > patience:
        print("Stopping early due to lack of improvement in validation accuracy.
↪")
        break
```

```
Epoch 1/40: 100%|      | 2127/2127 [04:35<00:00,  7.71it/s]

Epoch 1, Train Loss: 1.038158256471241, Train Accuracy: 0.44907135300340895,
Test Loss: 0.8766948894659679, Test Accuracy: 0.6461474036850922, Validation
Loss: 1.0915394597931911, Validation Accuracy: 0.3789649415692821

Epoch 2/40: 100%|      | 2127/2127 [04:34<00:00,  7.76it/s]

Epoch 2, Train Loss: 0.8062941928564465, Train Accuracy: 0.552838838603503, Test
Loss: 0.6887196026245753, Test Accuracy: 0.7139865996649917, Validation Loss:
1.0637079963558598, Validation Accuracy: 0.38731218697829717

Epoch 3/40: 100%|      | 2127/2127 [04:31<00:00,  7.83it/s]

Epoch 3, Train Loss: 0.7100688778547036, Train Accuracy: 0.5888386035029975,
Test Loss: 0.5940767848491668, Test Accuracy: 0.7432998324958124, Validation
Loss: 0.9950733749490035, Validation Accuracy: 0.4056761268781302

Epoch 4/40: 100%|      | 2127/2127 [04:31<00:00,  7.83it/s]

Epoch 4, Train Loss: 0.6360879438825487, Train Accuracy: 0.6116727400963912,
Test Loss: 0.6520130230983099, Test Accuracy: 0.7336683417085427, Validation
Loss: 0.9441205121968922, Validation Accuracy: 0.4357262103505843

Epoch 5/40: 100%|      | 2127/2127 [04:30<00:00,  7.85it/s]

Epoch 5, Train Loss: 0.5640135205262807, Train Accuracy: 0.6404431644528036,
Test Loss: 0.5285604581236839, Test Accuracy: 0.7587939698492462, Validation
Loss: 0.9108067884256965, Validation Accuracy: 0.49248747913188645

Epoch 6/40: 100%|      | 2127/2127 [04:31<00:00,  7.83it/s]
```

```
Epoch 6, Train Loss: 0.496881178409898, Train Accuracy: 0.6702127659574468, Test
Loss: 0.6308415601650874, Test Accuracy: 0.7206867671691792, Validation Loss:
0.908859438017795, Validation Accuracy: 0.4774624373956594
No improvement in validation accuracy for 1 epochs.

Epoch 7/40: 100%|     | 2127/2127 [04:31<00:00,  7.84it/s]

Epoch 7, Train Loss: 0.4237294014609849, Train Accuracy: 0.6989244151874926,
Test Loss: 0.5103230973084768, Test Accuracy: 0.7768006700167505, Validation
Loss: 0.8057699360345539, Validation Accuracy: 0.5609348914858097

Epoch 8/40: 100%|     | 2127/2127 [04:31<00:00,  7.82it/s]

Epoch 8, Train Loss: 0.35802674052915456, Train Accuracy: 0.7229046667450335,
Test Loss: 0.4837748867770036, Test Accuracy: 0.7730318257956449, Validation
Loss: 0.7484175233464492, Validation Accuracy: 0.5459098497495827
No improvement in validation accuracy for 1 epochs.

Epoch 9/40: 100%|     | 2127/2127 [35:13<00:00,  1.01it/s]

Epoch 9, Train Loss: 0.3060540745392106, Train Accuracy: 0.7488244974726695,
Test Loss: 0.5215935582915942, Test Accuracy: 0.7768006700167505, Validation
Loss: 0.883939485016622, Validation Accuracy: 0.5542570951585977
No improvement in validation accuracy for 2 epochs.

Epoch 10/40: 100%|     | 2127/2127 [04:32<00:00,  7.80it/s]

Epoch 10, Train Loss: 0.24410162753980305, Train Accuracy: 0.7712765957446809,
Test Loss: 0.4762981908147534, Test Accuracy: 0.7705192629815746, Validation
Loss: 0.7543475165178901, Validation Accuracy: 0.5592654424040067
No improvement in validation accuracy for 3 epochs.

Epoch 11/40: 100%|     | 2127/2127 [04:31<00:00,  7.84it/s]

Epoch 11, Train Loss: 0.20427155406231812, Train Accuracy: 0.7882626072646056,
Test Loss: 0.49015229113710423, Test Accuracy: 0.7893634840871022, Validation
Loss: 0.8891089176268954, Validation Accuracy: 0.5692821368948247

Epoch 12/40: 100%|     | 2127/2127 [04:32<00:00,  7.79it/s]

Epoch 12, Train Loss: 0.17004134656908385, Train Accuracy: 0.802162924650288,
Test Loss: 0.7857648380349079, Test Accuracy: 0.733249581239531, Validation
Loss: 1.4519559019490291, Validation Accuracy: 0.48580968280467446
No improvement in validation accuracy for 1 epochs.

Epoch 13/40: 100%|     | 2127/2127 [04:30<00:00,  7.85it/s]

Epoch 13, Train Loss: 0.14110336791115127, Train Accuracy: 0.8127130598330786,
Test Loss: 0.538908489793539, Test Accuracy: 0.7646566164154104, Validation
Loss: 0.863646683724303, Validation Accuracy: 0.5726210350584308

Epoch 14/40: 100%|     | 2127/2127 [04:33<00:00,  7.77it/s]

Epoch 14, Train Loss: 0.12739907055513602, Train Accuracy: 0.8192077112965793,
Test Loss: 0.615200116597116, Test Accuracy: 0.7809882747068677, Validation
Loss: 1.069829094174661, Validation Accuracy: 0.5909849749582637
```

```
Epoch 15/40: 100%|     | 2127/2127 [04:33<00:00,  7.78it/s]

Epoch 15, Train Loss: 0.1175731341872931, Train Accuracy: 0.8232044198895028,
Test Loss: 0.5340123608211677, Test Accuracy: 0.7646566164154104, Validation
Loss: 0.7495129916228747, Validation Accuracy: 0.5976627712854758

Epoch 16/40: 100%|     | 2127/2127 [04:33<00:00,  7.79it/s]

Epoch 16, Train Loss: 0.10165356366667071, Train Accuracy: 0.827730104619725,
Test Loss: 0.6439926016454895, Test Accuracy: 0.7763819095477387, Validation
Loss: 1.1798048584084762, Validation Accuracy: 0.5742904841402338
No improvement in validation accuracy for 1 epochs.

Epoch 17/40: 100%|     | 2127/2127 [04:32<00:00,  7.81it/s]

Epoch 17, Train Loss: 0.09943411914710484, Train Accuracy: 0.8303455977430352,
Test Loss: 0.5641713500209152, Test Accuracy: 0.7948073701842546, Validation
Loss: 1.0125738680362701, Validation Accuracy: 0.5926544240400667
No improvement in validation accuracy for 2 epochs.

Epoch 18/40: 100%|     | 2127/2127 [04:32<00:00,  7.80it/s]

Epoch 18, Train Loss: 0.08994970858037227, Train Accuracy: 0.8335782296931938,
Test Loss: 0.5471182976306106, Test Accuracy: 0.7977386934673367, Validation
Loss: 1.024513742249263, Validation Accuracy: 0.5976627712854758
No improvement in validation accuracy for 3 epochs.

Epoch 19/40: 100%|     | 2127/2127 [04:32<00:00,  7.80it/s]

Epoch 19, Train Loss: 0.07943363850364206, Train Accuracy: 0.837428000470201,
Test Loss: 0.601752273112846, Test Accuracy: 0.7872696817420436, Validation
Loss: 1.1811322526712167, Validation Accuracy: 0.5759599332220368
No improvement in validation accuracy for 4 epochs.

Epoch 20/40: 100%|     | 2127/2127 [04:32<00:00,  7.80it/s]

Epoch 20, Train Loss: 0.07760589793114128, Train Accuracy: 0.835958622311038,
Test Loss: 0.5253865905106068, Test Accuracy: 0.7969011725293133, Validation
Loss: 0.9357573864491362, Validation Accuracy: 0.6060100166944908

Epoch 21/40: 100%|     | 2127/2127 [04:32<00:00,  7.79it/s]

Epoch 21, Train Loss: 0.06859378968538718, Train Accuracy: 0.8388973786293641,
Test Loss: 0.6550833213205138, Test Accuracy: 0.7692629815745393, Validation
Loss: 1.0431401433521195, Validation Accuracy: 0.6026711185308848
No improvement in validation accuracy for 1 epochs.

Epoch 22/40: 100%|     | 2127/2127 [04:32<00:00,  7.81it/s]

Epoch 22, Train Loss: 0.06718066186183375, Train Accuracy: 0.8390443164452803,
Test Loss: 0.6963489640317857, Test Accuracy: 0.7868509212730318, Validation
Loss: 1.2220798252444518, Validation Accuracy: 0.5926544240400667
No improvement in validation accuracy for 2 epochs.

Epoch 23/40: 100%|     | 2127/2127 [04:33<00:00,  7.76it/s]
```

```
Epoch 23, Train Loss: 0.05667143662150525, Train Accuracy: 0.8422769483954391,
Test Loss: 0.7877611050444344, Test Accuracy: 0.7935510887772195, Validation
Loss: 1.5479834440507387, Validation Accuracy: 0.5859766277128547
No improvement in validation accuracy for 3 epochs.

Epoch 24/40: 100%|     | 2127/2127 [04:33<00:00,  7.77it/s]

Epoch 24, Train Loss: 0.05858069760403162, Train Accuracy: 0.8421300105795227,
Test Loss: 0.7133151440194342, Test Accuracy: 0.7843383584589615, Validation
Loss: 1.1273613050580025, Validation Accuracy: 0.6093489148580968

Epoch 25/40: 100%|     | 2127/2127 [04:32<00:00,  7.79it/s]

Epoch 25, Train Loss: 0.05833942163364176, Train Accuracy: 0.8432761255436699,
Test Loss: 0.7167338895025508, Test Accuracy: 0.7876884422110553, Validation
Loss: 1.3211990362523418, Validation Accuracy: 0.5993322203672788
No improvement in validation accuracy for 1 epochs.

Epoch 26/40: 100%|     | 2127/2127 [04:33<00:00,  7.78it/s]

Epoch 26, Train Loss: 0.05278913704836877, Train Accuracy: 0.8455095803455978,
Test Loss: 0.6841571927687619, Test Accuracy: 0.785175879396985, Validation
Loss: 1.2453941389134056, Validation Accuracy: 0.6010016694490818
No improvement in validation accuracy for 2 epochs.

Epoch 27/40: 100%|     | 2127/2127 [04:32<00:00,  7.79it/s]

Epoch 27, Train Loss: 0.05425615350117347, Train Accuracy: 0.8440989773128013,
Test Loss: 0.7837989750556881, Test Accuracy: 0.7906197654941374, Validation
Loss: 1.4191058604536873, Validation Accuracy: 0.6043405676126878
No improvement in validation accuracy for 3 epochs.

Epoch 28/40: 100%|     | 2127/2127 [04:33<00:00,  7.79it/s]

Epoch 28, Train Loss: 0.05061246426776571, Train Accuracy: 0.8454801927824145,
Test Loss: 0.6675986280441673, Test Accuracy: 0.7948073701842546, Validation
Loss: 1.2572345541496026, Validation Accuracy: 0.6110183639398998

Epoch 29/40: 100%|     | 2127/2127 [04:32<00:00,  7.80it/s]

Epoch 29, Train Loss: 0.04894330673879113, Train Accuracy: 0.8453038674033149,
Test Loss: 0.7051294089093183, Test Accuracy: 0.788107202680067, Validation
Loss: 1.4890289679169655, Validation Accuracy: 0.5859766277128547
No improvement in validation accuracy for 1 epochs.

Epoch 30/40: 100%|     | 2127/2127 [04:33<00:00,  7.78it/s]

Epoch 30, Train Loss: 0.043532936140868284, Train Accuracy: 0.8471258963206771,
Test Loss: 0.9987100430003678, Test Accuracy: 0.7638190954773869, Validation
Loss: 1.9004389694646786, Validation Accuracy: 0.5692821368948247
No improvement in validation accuracy for 2 epochs.

Epoch 31/40: 100%|     | 2127/2127 [04:31<00:00,  7.84it/s]
```

```
Epoch 31, Train Loss: 0.04990433166783608, Train Accuracy: 0.8455095803455978,
Test Loss: 0.6789765183371492, Test Accuracy: 0.7847571189279732, Validation
Loss: 1.1027874840717566, Validation Accuracy: 0.6260434056761269

Epoch 32/40: 100%|      | 2127/2127 [04:32<00:00,  7.82it/s]

Epoch 32, Train Loss: 0.04153778113902246, Train Accuracy: 0.8473609968261432,
Test Loss: 0.7082144226983655, Test Accuracy: 0.7906197654941374, Validation
Loss: 1.46780813014821, Validation Accuracy: 0.5959933222036727
No improvement in validation accuracy for 1 epochs.

Epoch 33/40: 100%|      | 2127/2127 [04:33<00:00,  7.79it/s]

Epoch 33, Train Loss: 0.03800791553515266, Train Accuracy: 0.8478311978370754,
Test Loss: 0.9559638269568131, Test Accuracy: 0.791038525963149, Validation
Loss: 1.7312354708188458, Validation Accuracy: 0.5976627712854758
No improvement in validation accuracy for 2 epochs.

Epoch 34/40: 100%|      | 2127/2127 [04:33<00:00,  7.79it/s]

Epoch 34, Train Loss: 0.04708815529021551, Train Accuracy: 0.8459503937933467,
Test Loss: 0.8340883854031563, Test Accuracy: 0.7470686767169179, Validation
Loss: 0.980157943148362, Validation Accuracy: 0.6243739565943238
No improvement in validation accuracy for 3 epochs.

Epoch 35/40: 100%|      | 2127/2127 [04:33<00:00,  7.78it/s]

Epoch 35, Train Loss: 0.041739497996898176, Train Accuracy: 0.8477724227107089,
Test Loss: 0.7098684623957767, Test Accuracy: 0.7885259631490787, Validation
Loss: 1.3593406594897572, Validation Accuracy: 0.5976627712854758
No improvement in validation accuracy for 4 epochs.

Epoch 36/40: 100%|      | 2127/2127 [04:33<00:00,  7.77it/s]

Epoch 36, Train Loss: 0.042951817305408424, Train Accuracy: 0.8479193605266251,
Test Loss: 0.8280993535387825, Test Accuracy: 0.7818257956448911, Validation
Loss: 1.4862253218889236, Validation Accuracy: 0.6093489148580968
No improvement in validation accuracy for 5 epochs.

Epoch 37/40: 100%|      | 2127/2127 [04:32<00:00,  7.79it/s]

Epoch 37, Train Loss: 0.03859294675208923, Train Accuracy: 0.848800987422123,
Test Loss: 0.8216257402518128, Test Accuracy: 0.7989949748743719, Validation
Loss: 1.7455876108847166, Validation Accuracy: 0.5843071786310517
No improvement in validation accuracy for 6 epochs.
Stopping early due to lack of improvement in validation accuracy.
```

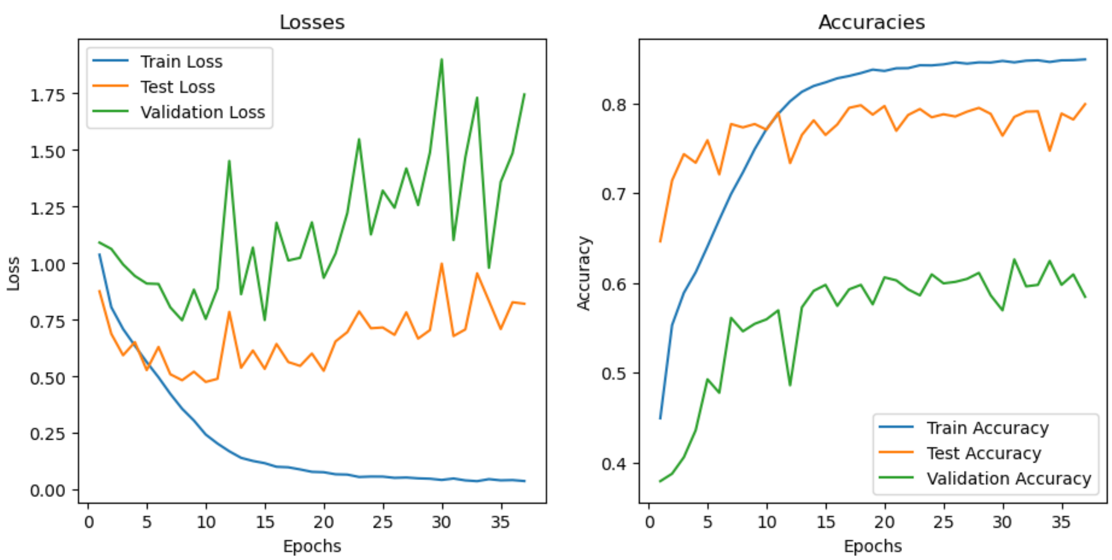
[ ]:

```
plt.figure(figsize=(17, 5))
plt.subplot(1, 3, 1)
plt.plot(range(1, 38), train_losses, label='Train Loss') # change this number␣
 ↪after '(1, _)' to num_epochs+1
plt.plot(range(1, 38), test_losses, label='Test Loss') # change this number␣
 ↪after '(1, _)' to num_epochs+1
```

```
plt.plot(range(1, 38), val_losses, label='Validation Loss') # change this␣
↪number after '(1, _)' to num_epochs+1
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Losses')
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(range(1, 38), train_accuracies, label='Train Accuracy') # change this␣
↪number after '(1, _)' to num_epochs+1
plt.plot(range(1, 38), test_accuracies, label='Test Accuracy') # change this␣
↪number after '(1, _)' to num_epochs+1
plt.plot(range(1, 38), val_accuracies, label='Validation Accuracy') # change␣
↪this number after '(1, _)' to num_epochs+1
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracies')
plt.legend()

plt.show()
```



[ ]:
```
df = pd.DataFrame({
    'Epoch': range(1, 38), # change this number after '(1, _)' to num_epochs+1
    'Train Loss': train_losses,
    'Test Loss': test_losses,
    'Validation Loss': val_losses,
    'Train Accuracy': train_accuracies,
    'Test Accuracy': test_accuracies,
```

```python
    'Validation Accuracy': val_accuracies
})
df.to_csv('training_metrics.csv', index=False)
```

[ ]:

```python

```