Model: - Convolutional + ASL.Net

Dataset: - ASL Data-Set

Language: - Python

Code: -

import os

import shutil

import argparse

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from PIL import Image

import matplotlib.pyplot as plt

class ASLDataset(Dataset):

    def \_\_init\_\_(self, root\_dir, transform=None):

        self.root\_dir = root\_dir

        self.transform = transform

        self.classes = self.\_get\_classes()  # Get all classes (0-9, A-Z)

        self.image\_paths = self.\_load\_image\_paths()

    def \_get\_classes(self):

        # Include both digits 0-9 and letters A-Z

        digit\_classes = [str(i) for i in range(10)]

        letter\_classes = [chr(i) for i in range(ord('A'), ord('Z')+1)]

        return digit\_classes + letter\_classes

    def \_load\_image\_paths(self):

        image\_paths = []

        labels = []

        for class\_name in self.classes:

            class\_path = os.path.join(self.root\_dir, class\_name)

            if os.path.exists(class\_path):

                for img\_name in os.listdir(class\_path):

                    if img\_name.endswith(('.jpg', '.jpeg', '.png')):

                        image\_paths.append((os.path.join(class\_path, img\_name), self.classes.index(class\_name)))

        return image\_paths

    def \_\_len\_\_(self):

        return len(self.image\_paths)

    def \_\_getitem\_\_(self, idx):

        img\_path, label = self.image\_paths[idx]

        image = Image.open(img\_path).convert('RGB')

        if self.transform:

            image = self.transform(image)

        return image, label

class ASLNet(nn.Module):

    def \_\_init\_\_(self, num\_classes=36):  # 10 digits + 26 letters = 36 classes

        super(ASLNet, self).\_\_init\_\_()

        self.features = nn.Sequential(

            nn.Conv2d(3, 64, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(64, 128, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(128, 256, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

            nn.Conv2d(256, 512, kernel\_size=3, padding=1),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel\_size=2, stride=2),

        )

        self.classifier = nn.Sequential(

            nn.Linear(512 \* 4 \* 4, 1024),

            nn.ReLU(inplace=True),

            nn.Dropout(0.5),

            nn.Linear(1024, num\_classes)

        )

    def forward(self, x):

        x = self.features(x)

        x = torch.flatten(x, 1)

        x = self.classifier(x)

        return x

def train(model, train\_loader, criterion, optimizer, device, epoch):

    model.train()

    running\_loss = 0.0

    correct = 0

    total = 0

    for batch\_idx, (data, target) in enumerate(train\_loader):

        data, target = data.to(device), target.to(device)

        optimizer.zero\_grad()

        outputs = model(data)

        loss = criterion(outputs, target)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

        \_, predicted = outputs.max(1)

        total += target.size(0)

        correct += predicted.eq(target).sum().item()

        if batch\_idx % 10 == 0:

            print(f'Epoch: {epoch}, Batch: {batch\_idx}/{len(train\_loader)}, '

                  f'Loss: {running\_loss/(batch\_idx+1):.3f}, '

                  f'Acc: {100.\*correct/total:.2f}%')

def test(model, test\_loader, criterion, device):

    model.eval()

    test\_loss = 0

    correct = 0

    total = 0

    with torch.no\_grad():

        for data, target in test\_loader:

            data, target = data.to(device), target.to(device)

            outputs = model(data)

            test\_loss += criterion(outputs, target).item()

            \_, predicted = outputs.max(1)

            total += target.size(0)

            correct += predicted.eq(target).sum().item()

    acc = 100.\*correct/total

    print(f'\nTest set: Average loss: {test\_loss/len(test\_loader):.4f}, '

          f'Accuracy: {correct}/{total} ({acc:.2f}%)\n')

    return acc

def main():

    # Arguments

    parser = argparse.ArgumentParser(description='ASL Recognition')

    parser.add\_argument('--data-dir', type=str,

                      default=r'E:\Win\_sem 2024-25\BCSE332L-Deep Learning\LAB\Assessment -1DL\PredRNN\data',

                      help='data directory')

    parser.add\_argument('--batch-size', type=int, default=32)

    parser.add\_argument('--epochs', type=int, default=50)

    parser.add\_argument('--lr', type=float, default=0.001)

    parser.add\_argument('--no-cuda', action='store\_true', default=False)

    args = parser.parse\_args()

    # Setup

    use\_cuda = not args.no\_cuda and torch.cuda.is\_available()

    device = torch.device("cuda" if use\_cuda else "cpu")

    print(f"Using device: {device}")

    # Data transforms

    transform = transforms.Compose([

        transforms.Resize((64, 64)),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.485, 0.456, 0.406],

                           std=[0.229, 0.224, 0.225])

    ])

    # Create dataset

    full\_dataset = ASLDataset(root\_dir=args.data\_dir, transform=transform)

    # Split dataset

    train\_size = int(0.8 \* len(full\_dataset))

    test\_size = len(full\_dataset) - train\_size

    train\_dataset, test\_dataset = torch.utils.data.random\_split(

        full\_dataset, [train\_size, test\_size])

    # Create data loaders

    train\_loader = DataLoader(train\_dataset, batch\_size=args.batch\_size,

                            shuffle=True, num\_workers=2)

    test\_loader = DataLoader(test\_dataset, batch\_size=args.batch\_size,

                           shuffle=False, num\_workers=2)

    # Initialize model, criterion, and optimizer

    model = ASLNet().to(device)

    criterion = nn.CrossEntropyLoss()

    optimizer = optim.Adam(model.parameters(), lr=args.lr)

    # Training loop

    best\_acc = 0

    for epoch in range(1, 10):

        print(f'\nEpoch: {epoch}')

        train(model, train\_loader, criterion, optimizer, device, epoch)

        acc = test(model, test\_loader, criterion, device)

        if acc > best\_acc:

            best\_acc = acc

            torch.save(model.state\_dict(), 'asl\_recognition\_model.pth')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Code: - (Real time recognition)

import cv2

import torch

import torchvision.transforms as transforms

from PIL import Image

import numpy as np

# Import your model architecture

from run2 import ASLNet

def load\_model(model\_path):

    model = ASLNet()

    model.load\_state\_dict(torch.load(model\_path))

    model.eval()

    return model

def preprocess\_image(image):

    transform = transforms.Compose([

        transforms.Resize((64, 64)),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

    ])

    image = Image.fromarray(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

    return transform(image).unsqueeze(0)

def get\_prediction(model, image):

    with torch.no\_grad():

        outputs = model(image)

        \_, predicted = torch.max(outputs, 1)

        return predicted.item()

def main():

    model\_path = 'asl\_recognition\_model.pth'

    model = load\_model(model\_path)

    # Define the classes (0-9 and A-Z)

    classes = [str(i) for i in range(10)] + [chr(i) for i in range(ord('A'), ord('Z')+1)]

    # Start video capture

    cap = cv2.VideoCapture(0)

    while True:

        ret, frame = cap.read()

        if not ret:

            break

        # Preprocess the frame

        processed\_frame = preprocess\_image(frame)

        # Get prediction

        prediction = get\_prediction(model, processed\_frame)

        predicted\_class = classes[prediction]

        # Draw the prediction on the frame

        cv2.putText(frame, f"Prediction: {predicted\_class}", (10, 30),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

        # Display the frame

        cv2.imshow('ASL Recognition', frame)

        # Break the loop if 'q' is pressed

        if cv2.waitKey(1) & 0xFF == ord('q'):

            break

    # Release the capture and close windows

    cap.release()

    cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

    main()









 