MINOR PROJECT

SIGN LANGUAGE TO TEXT

1.Reedam Ranjan (1805686)
2.Nandini Kalita (1805675)
3.Sourodeep Dhar (1805711)

Step 1 — Preparing the Sign Language Classification Dataset(Preprocessing)

In [1]:

```python
from torch.utils.data import Dataset
from torch.autograd import Variable
import torchvision.transforms as transforms
import torch.nn as nn
import numpy as np
import torch

from typing import List

import csv


class SignLanguageMNIST(Dataset):
    """Sign Language classification dataset.

    Utility for loading Sign Language dataset into PyTorch. Dataset posted on
    Kaggle in 2017, by an unnamed author with username `tecperson`:
    https://www.kaggle.com/datamunge/sign-language-mnist

    Each sample is 1 x 1 x 28 x 28, and each label is a scalar.
    """

    @staticmethod
    def get_label_mapping():
        """
        We map all labels to [0, 23]. This mapping from dataset labels [0, 23]
        to letter indices [0, 25] is returned below.
        """
        mapping = list(range(25))
        mapping.pop(9)
        return mapping

    @staticmethod
    def read_label_samples_from_csv(path: str):
        """
        Assumes first column in CSV is the label and subsequent 28^2 values
        are image pixel values 0-255.
        """
        mapping = SignLanguageMNIST.get_label_mapping()
        labels, samples = [], []
        with open(path) as f:
            _ = next(f)  # skip header
            for line in csv.reader(f):
                label = int(line[0])
                labels.append(mapping.index(label))
                samples.append(list(map(int, line[1:])))
        return labels, samples

    def __init__(self,
            path: str="data/sign_mnist_train.csv",
            mean: List[float]=[0.485],
            std: List[float]=[0.229]):
        """
        Args:
            path: Path to `.csv` file containing `label`, `pixel0`, `pixel1`...
        """
        labels, samples = SignLanguageMNIST.read_label_samples_from_csv(path)
        self._samples = np.array(samples, dtype=np.uint8).reshape((-1, 28, 28, 1))
        self._labels = np.array(labels, dtype=np.uint8).reshape((-1, 1))

        self._mean = mean
        self._std = std

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

    def __getitem__(self, idx):
        transform = transforms.Compose([
            transforms.ToPILImage(),
            transforms.RandomResizedCrop(28, scale=(0.8, 1.2)),
            transforms.ToTensor(),
            transforms.Normalize(mean=self._mean, std=self._std)])

        return {
            'image': transform(self._samples[idx]).float(),
            'label': torch.from_numpy(self._labels[idx]).float()
        }
```

In [4]:

```python
def get_train_test_loaders(batch_size=32):
    trainset = SignLanguageMNIST('data/sign_mnist_train.csv')
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)

    testset = SignLanguageMNIST('data/sign_mnist_test.csv')
    testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)
    return trainloader, testloader
```

In [5]:

```python
loader, _ = get_train_test_loaders(2)
print(next(iter(loader)))
```

```
{'image': tensor([[[[0.8961, 0.9132, 0.9132,  ..., 0.8961, 0.9132, 0.9132],
          [0.9132, 0.9132, 0.9132,  ..., 0.9132, 0.9132, 0.9303],
          [0.9303, 0.9303, 0.9303,  ..., 0.9303, 0.9303, 0.9474],
          ...,
          [1.1187, 1.1015, 1.1700,  ..., 1.2557, 1.2043, 1.2043],
          [1.1358, 1.1187, 1.1529,  ..., 1.2043, 1.2043, 1.1872],
          [1.1358, 1.1358, 1.1700,  ..., 1.2043, 1.2043, 1.2214]]],


        [[[0.6221, 0.6392, 0.6563,  ..., 0.4337, 0.4166, 0.3994],
          [0.6906, 0.6906, 0.6906,  ..., 0.4851, 0.4851, 0.4508],
          [0.7248, 0.7248, 0.7248,  ..., 0.5364, 0.5193, 0.5022],
          ...,
          [1.2557, 1.2385, 1.2214,  ..., 1.1015, 1.0844, 1.0673],
          [1.2557, 1.2557, 1.2557,  ..., 1.1187, 1.1015, 1.0673],
          [1.2557, 1.2385, 1.2557,  ..., 1.1187, 1.0844, 1.0673]]]]), 'label': tensor([[11.],
        [13.]])}
```

Step 2 — Building and Training the Sign Language Classifier Using Deep Learning

In [6]:

```python
from torch.utils.data import Dataset
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch




class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 6, 3)
        self.conv3 = nn.Conv2d(6, 16, 3)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 48)
        self.fc3 = nn.Linear(48, 25)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


def main():
    net = Net().float()
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

    trainloader, _ = get_train_test_loaders()
    for epoch in range(12):  # loop over the dataset multiple times
        train(net, criterion, optimizer, trainloader, epoch)
        scheduler.step()
    torch.save(net.state_dict(), "checkpoint.pth")



def train(net, criterion, optimizer, trainloader, epoch):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs = Variable(data['image'].float())
        labels = Variable(data['label'].long())
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels[:, 0])
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 100 == 0:
            print('[%d, %5d] loss: %.6f' % (epoch, i, running_loss / (i + 1)))
```

In [7]:

```python
main()
```

```
[0,     0] loss: 3.193408
[0,   100] loss: 3.209208
[0,   200] loss: 3.193350
[0,   300] loss: 3.020502
[0,   400] loss: 2.767063
[0,   500] loss: 2.484486
[0,   600] loss: 2.243822
[0,   700] loss: 2.038367
[0,   800] loss: 1.863631
[1,     0] loss: 0.660421
[1,   100] loss: 0.454788
[1,   200] loss: 0.431698
[1,   300] loss: 0.401833
[1,   400] loss: 0.367726
[1,   500] loss: 0.342959
[1,   600] loss: 0.325167
[1,   700] loss: 0.306999
[1,   800] loss: 0.291484
[2,     0] loss: 0.047885
[2,   100] loss: 0.160940
[2,   200] loss: 0.149098
[2,   300] loss: 0.146902
[2,   400] loss: 0.142737
[2,   500] loss: 0.138513
[2,   600] loss: 0.132887
[2,   700] loss: 0.130138
[2,   800] loss: 0.125890
[3,     0] loss: 0.031571
[3,   100] loss: 0.090978
[3,   200] loss: 0.084181
[3,   300] loss: 0.081451
[3,   400] loss: 0.085871
[3,   500] loss: 0.084438
[3,   600] loss: 0.088302
[3,   700] loss: 0.087049
[3,   800] loss: 0.087946
[4,     0] loss: 0.017777
[4,   100] loss: 0.057266
[4,   200] loss: 0.061120
[4,   300] loss: 0.060221
[4,   400] loss: 0.058293
[4,   500] loss: 0.059010
[4,   600] loss: 0.058586
[4,   700] loss: 0.058511
[4,   800] loss: 0.057770
[5,     0] loss: 0.007457
[5,   100] loss: 0.068351
[5,   200] loss: 0.060739
[5,   300] loss: 0.055211
[5,   400] loss: 0.058636
[5,   500] loss: 0.058849
[5,   600] loss: 0.057620
[5,   700] loss: 0.056358
[5,   800] loss: 0.055398
[6,     0] loss: 0.072571
[6,   100] loss: 0.032231
[6,   200] loss: 0.040233
[6,   300] loss: 0.036612
[6,   400] loss: 0.036078
[6,   500] loss: 0.041315
[6,   600] loss: 0.042208
[6,   700] loss: 0.040826
[6,   800] loss: 0.040325
[7,     0] loss: 0.008531
[7,   100] loss: 0.030616
[7,   200] loss: 0.031134
[7,   300] loss: 0.030806
[7,   400] loss: 0.032173
[7,   500] loss: 0.033607
[7,   600] loss: 0.038027
[7,   700] loss: 0.038285
[7,   800] loss: 0.037273
[8,     0] loss: 0.158042
[8,   100] loss: 0.041839
[8,   200] loss: 0.041024
[8,   300] loss: 0.039950
[8,   400] loss: 0.045207
[8,   500] loss: 0.046381
[8,   600] loss: 0.043453
[8,   700] loss: 0.040535
[8,   800] loss: 0.039876
[9,     0] loss: 0.000369
[9,   100] loss: 0.038694
[9,   200] loss: 0.037872
[9,   300] loss: 0.033917
[9,   400] loss: 0.034002
[9,   500] loss: 0.030806
[9,   600] loss: 0.032566
[9,   700] loss: 0.032744
[9,   800] loss: 0.033113
[10,     0] loss: 0.000153
[10,   100] loss: 0.017307
[10,   200] loss: 0.015655
[10,   300] loss: 0.013979
[10,   400] loss: 0.012866
[10,   500] loss: 0.011855
[10,   600] loss: 0.011260
[10,   700] loss: 0.010566
[10,   800] loss: 0.010019
[11,     0] loss: 0.000162
[11,   100] loss: 0.006919
[11,   200] loss: 0.006931
[11,   300] loss: 0.005598
[11,   400] loss: 0.007027
[11,   500] loss: 0.006762
[11,   600] loss: 0.006077
[11,   700] loss: 0.005840
[11,   800] loss: 0.005476
```

In [8]:

```python
main()
```

```
[0,     0] loss: 3.218370
[0,   100] loss: 3.209821
[0,   200] loss: 3.202712
[0,   300] loss: 3.133060
[0,   400] loss: 2.929191
[0,   500] loss: 2.699897
[0,   600] loss: 2.455835
[0,   700] loss: 2.228534
[0,   800] loss: 2.027808
[1,     0] loss: 0.507166
[1,   100] loss: 0.500827
[1,   200] loss: 0.461239
[1,   300] loss: 0.417974
[1,   400] loss: 0.390625
[1,   500] loss: 0.369000
[1,   600] loss: 0.344547
[1,   700] loss: 0.321961
[1,   800] loss: 0.304202
[2,     0] loss: 0.435642
[2,   100] loss: 0.179268
[2,   200] loss: 0.155368
[2,   300] loss: 0.157750
[2,   400] loss: 0.152462
[2,   500] loss: 0.146353
[2,   600] loss: 0.142867
[2,   700] loss: 0.138218
[2,   800] loss: 0.135241
[3,     0] loss: 0.203142
[3,   100] loss: 0.093122
[3,   200] loss: 0.092518
[3,   300] loss: 0.089966
[3,   400] loss: 0.090251
[3,   500] loss: 0.088381
[3,   600] loss: 0.083922
[3,   700] loss: 0.085312
[3,   800] loss: 0.085651
[4,     0] loss: 0.039505
[4,   100] loss: 0.085106
[4,   200] loss: 0.075297
[4,   300] loss: 0.066901
[4,   400] loss: 0.069145
[4,   500] loss: 0.065298
[4,   600] loss: 0.065675
[4,   700] loss: 0.065393
[4,   800] loss: 0.069289
[5,     0] loss: 0.034225
[5,   100] loss: 0.050080
[5,   200] loss: 0.046112
[5,   300] loss: 0.047215
[5,   400] loss: 0.046595
[5,   500] loss: 0.043227
[5,   600] loss: 0.043054
[5,   700] loss: 0.044323
[5,   800] loss: 0.044103
[6,     0] loss: 0.061230
[6,   100] loss: 0.031948
[6,   200] loss: 0.040579
[6,   300] loss: 0.048135
[6,   400] loss: 0.046492
[6,   500] loss: 0.044303
[6,   600] loss: 0.044723
[6,   700] loss: 0.041663
[6,   800] loss: 0.043448
[7,     0] loss: 0.021581
[7,   100] loss: 0.055558
[7,   200] loss: 0.043741
[7,   300] loss: 0.039236
[7,   400] loss: 0.038254
[7,   500] loss: 0.038604
[7,   600] loss: 0.036711
[7,   700] loss: 0.037918
[7,   800] loss: 0.035991
[8,     0] loss: 0.002241
[8,   100] loss: 0.038937
[8,   200] loss: 0.033439
[8,   300] loss: 0.032727
[8,   400] loss: 0.034187
[8,   500] loss: 0.035973
[8,   600] loss: 0.033995
[8,   700] loss: 0.032971
[8,   800] loss: 0.033962
[9,     0] loss: 0.004806
[9,   100] loss: 0.034603
[9,   200] loss: 0.031797
[9,   300] loss: 0.031906
[9,   400] loss: 0.034440
[9,   500] loss: 0.033291
[9,   600] loss: 0.033475
[9,   700] loss: 0.031474
[9,   800] loss: 0.031825
[10,     0] loss: 0.002800
[10,   100] loss: 0.010861
[10,   200] loss: 0.011170
[10,   300] loss: 0.012243
[10,   400] loss: 0.012188
[10,   500] loss: 0.011350
[10,   600] loss: 0.011152
[10,   700] loss: 0.010790
[10,   800] loss: 0.010602
[11,     0] loss: 0.008281
[11,   100] loss: 0.004162
[11,   200] loss: 0.005755
[11,   300] loss: 0.006706
[11,   400] loss: 0.006712
[11,   500] loss: 0.006371
[11,   600] loss: 0.006017
[11,   700] loss: 0.005722
[11,   800] loss: 0.005538
```

Step 3 — Evaluating the Sign Language Classifier

In [9]:

```python
from torch.utils.data import Dataset
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch
import numpy as np

import onnx
import onnxruntime as ort




def evaluate(outputs: Variable, labels: Variable) -> float:
    """Evaluate neural network outputs against non-one-hotted labels."""
    Y = labels.numpy()
    Yhat = np.argmax(outputs, axis=1)
    return float(np.sum(Yhat == Y))


def batch_evaluate(
        net: Net,
        dataloader: torch.utils.data.DataLoader) -> float:
    """Evaluate neural network in batches, if dataset is too large."""
    score = n = 0.0
    for batch in dataloader:
        n += len(batch['image'])
        outputs = net(batch['image'])
        if isinstance(outputs, torch.Tensor):
            outputs = outputs.detach().numpy()
        score += evaluate(outputs, batch['label'][:, 0])
    return score / n


def validate():
    trainloader, testloader = get_train_test_loaders()
    net = Net().float().eval()

    pretrained_model = torch.load("checkpoint.pth")
    net.load_state_dict(pretrained_model)

    print('=' * 10, 'PyTorch', '=' * 10)
    train_acc = batch_evaluate(net, trainloader) * 100.
    print('Training accuracy: %.1f' % train_acc)
    test_acc = batch_evaluate(net, testloader) * 100.
    print('Validation accuracy: %.1f' % test_acc)

    trainloader, testloader = get_train_test_loaders(1)

    # export to onnx
    fname = "signlanguage.onnx"
    dummy = torch.randn(1, 1, 28, 28)
    torch.onnx.export(net, dummy, fname, input_names=['input'])

    # check exported model
    model = onnx.load(fname)
    onnx.checker.check_model(model)  # check model is well-formed

    # create runnable session with exported model
    ort_session = ort.InferenceSession(fname)
    net = lambda inp: ort_session.run(None, {'input': inp.data.numpy()})[0]

    print('=' * 10, 'ONNX', '=' * 10)
    train_acc = batch_evaluate(net, trainloader) * 100.
    print('Training accuracy: %.1f' % train_acc)
    test_acc = batch_evaluate(net, testloader) * 100.
    print('Validation accuracy: %.1f' % test_acc)
```

In [10]:

```python
validate()
```

```
========== PyTorch ==========
Training accuracy: 99.8
Validation accuracy: 97.2
========== ONNX ==========
Training accuracy: 99.8
Validation accuracy: 96.9
```

Step 4 — Linking the Camera Feed

In [11]:

```python
import cv2
import numpy as np
import onnxruntime as ort


def center_crop(frame):
    h, w, _ = frame.shape
    start = abs(h - w) // 2
    if h > w:
        return frame[start: start + w]
    return frame[:, start: start + h]


def main_ca():
    # constants
    index_to_letter = list('ABCDEFGHIKLMNOPQRSTUVWXY')
    mean = 0.485 * 255.
    std = 0.229 * 255.

    # create runnable session with exported model
    ort_session = ort.InferenceSession("signlanguage.onnx")

    cap = cv2.VideoCapture(0)
    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()

        # preprocess data
        frame = center_crop(frame)
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        x = cv2.resize(frame, (28, 28))
        x = (x - mean) / std

        x = x.reshape(1, 1, 28, 28).astype(np.float32)
        y = ort_session.run(None, {'input': x})[0]

        index = np.argmax(y, axis=1)
        letter = index_to_letter[int(index)]

        cv2.putText(frame, letter, (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0, 255, 0), thickness=2)
        cv2.imshow("Sign Language Translator", frame)

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

    cap.release()
    cv2.destroyAllWindows()
```

In [ ]:

```python
main_ca()
```