CSE404 Project 2

201811118 이 구

프로젝트를 수행하기 전, 먼저 제공된 데이터에 관한 전처리를 수행하였다. 주어진 이미지의 해상도가 모두 달랐기 때문에, MNIST dataset의 형식에 맞춰 28x28 사이즈의 grayscale 이미지로 만들어주었다. 원본 이미지의 비율이 바뀌지 않도록 하기 위해, square 형태로 padding 후 28x28 size로 resize 하였다. 그 결과는 아래와 같다. 왼쪽 두 개의 이미지가 제공된 이미지이며, 오른쪽 두 개의 이미지가 각각에 해당하는 전 처리 결과이다. 왼쪽부터 해상도는 각각 59x76, 79x72, 28x28, 28x28이다. 이미지의 channel 고려하면, 정보량을 약 18배 가까이 줄여서 사용했다는 것을 알 수 있다.

<이미지 전처리 전 후 결과>

자세한 과정은 맨 뒤에 첨부한 소스코드 중, preprocessing.py에서 확인할 수 있다. 전체 프로젝트에서 별도의 언급이 없는 경우, preprocessing을 마친 dataset (28x28, grayscale)을 이용하였다. 프로젝트를 수행하기 위해 사용한 method 및 구현 방식은 아래와 같다.

1. Bayes Classifier with PCA

제공된 bayes classifier sample code를 기반으로 구현하였다. 사용한 feature는 784-dim (28*28)의 이미지 정보에 PCA(Principal Component Analysis)를 적용하여 구했다. 이때, PCA는 train dataset에 대해서만 fitting 하였으며, test dataset에는 이미 fitting한 정보만을 이용하여 dimension reduction을 수행하였다. 축소된 feature vector는 41-dim vector로, 이는 실험적으로 가장 높은 성능을 보이는 값을 찾은 것이다. 전체적인 과정은 첨부한 소스코드 중, 1_bc_with_PCA.py에서 확인할 수 있다.

2. CNN

모델 구조의 경우, 제공된 CNN sample code 중 conv_module의 MaxPool2D layer만을 제거한 후 사용하였다. 즉, 5개의 conv module과 하나의 global average pooling layer를 갖는다. 안정적인 학습을 위해 learning rate scheduler(이하 lr sched)를 추가적으로 구현 및 적용하였다. 사용한 lr shed는 StepLR scheduler이며, 이는 특정 step 마다 learning rate에 gamma 값을 곱하여 사용하는 방식이다. 학습에 사용한 parameter 정보는 다음과 같다. (epoch=30000, batch_size=80, init_lr = 0.001, lr_sched_step=10000, lr_sched_gamma=0.7) 전체적인 과정은 첨부한 소스코드 중, 2_cnn.py에서 확인할 수 있다.

3. Shallow CNN with enormous data

최근 발표되는 논문들에 따르면, 모델의 세부적인 구조가 아닌, 양질의 거대한 데이터셋이 딥러닝 모델의 강인하고, 뛰어난 성능을 가능하게 하는 주요 원인이다. 이러한 아이디어를 Data-centric AI라고 칭하며, 이에 기반하여 더욱 얕은 CNN network로도 뛰어난 성능을 보일 수 있다는 것을 실험을 통해 확인하고자 하였다. 이를 위해 3개의 conv module과 하나의 global average pooling layer를 갖는 CNN을 사용하였으며, 두 번째 방식과 같이 lr sched를 적용하였다. 학습에 사용한 데이터셋은, MNIST dataset의 train/test dataset에서 1, 2, 3, 4에 해당하는 data들을 parsing한 후, LMS에 제공된 train dataset에 preprocessing을 적용한 데이터셋을 합쳐서 사용하였다. MNIST dataset의 경우, 우리가 숫자로 인식하는 부분이 흰색으로, 나머지 배경에 해당하는 부분이 검은색으로 표현되어 있다. 하지만, 최종 목표인 LMS의 test dataset에서는 숫자 부분이 낮은 intensity를, 배경 부분이 높은 intensity를 가졌기에 MNIST dataset에서 parsing하는 과정에서 적절한 수정을

거쳤다. 추가적으로 parsing하여 사용한 MNIST dataset은 1에 해당하는 이미지 6742장, 2에 해당하는 이미지 5958장, 3에 해당하는 이미지 6131장, 4에 해당하는 이미지 5842장이다. 학습에 사용한 parameter 정보는 다음과 같다. (epoch=420, batch_size=100, init_lr=0.001, lr_sched_step=200, lr_sched_gamma=0.7) 전체적인 과정은 첨부한 소스코드 중, save_orig_MNIST.py와 3_cnn_modified.py에서 확인할 수 있다.

최종적으로, 20개의 test case에 대해 평가한 결과는 아래와 같다.

Method	Bayes Classifier with PCA	CNN	Shallow CNN with enormous dataset
Accuracy	45	95	95

```
PCA n_components :  41
Test lables       : [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
Predicted lables  : [1, 1, 0, 0, 0, 3, 1, 1, 3, 0, 0, 3, 0, 0, 0, 0, 3, 3, 3, 3]
Test Accuracy of the model on the 20 test images: 45.0 %
```

```
Training labels : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
Test lables       : [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
Predicted lables  : [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 2]
Test Accuracy of the model on the 20 test images: 95.0 %
```

```
Number of training labels : 24753
Train epochs       : 420
Test lables       : [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
Predicted lables  : [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 2]
Test Accuracy of the model on the 20 test images: 95.0 %
```

위에서부터 각각 첫 번째, 두 번째, 세 번째 method의 실행 및 예측 결과이다. 출력된 정보 중 label은 directory 순서에 따라 정해진 값이므로 출력된 값에 1을 더한 것이 실제 해당하는 숫자이다. (0 -> 1, 1 -> 2, 2 -> 3, 3 -> 4)

첫 번째 방법(Bayes Classifier with PCA)의 경우, 50%에도 미치지 못하는 성능을 보였다. 흑백 이미지의 784개의 pixel 중, 배경에 해당하는 pixel이 더욱 많기 때문에, 필요 없는 정보를 제거하기 위해 PCA를 적용하여 Bayes Classifier에서 사용할 feature를 결정하였다. 주로 사용되는 dimensionality reduction 방식인 PCA를 이용하면, 사람이 직접 manual하게 feature를 선택하지 않고, SVD를 통해 수학적으로 기존 데이터의 특성을 최대한 유지하면서 기존보다 낮은 차원을 갖는 feature vector를 얻을 수 있다는 장점이 있기에 이를 적용하였다. 하지만 결과를 직접 확인하는 과정에서 variance가 높다고 해서 항상 좋은 feature가 될 수 있는 것이 아니라는 것을 알게 되었다. 그 예로, 이미지의 중앙 부분은 우리가 숫자를 판별할 때 주의 깊게 보는 영역이지만, 모든 숫자에서 항상 어둡게 표현(숫자에 해당하는 pixel)되니 variance는 낮을 것이다. 이러한 점들이 첫 번째 방식(PCA + Bayes Classifier)의 낮은 정확도에 영향을 미쳤을 것이라 생각한다.

두번째 방식(CNN)과 세 번째 방식(Shallow CNN with enormous dataset)에서는 모두 우수한 성능을 보였다. 여기서 주목해야 할 것은, 세 번째 방식에서는 두 번째 방식에 비해 훨씬 적은 수의 parameter를 갖는다는 점이다. 실제 확인한 parameter 수는 두 번째 방식에서는 402,828개이며 세 번째 방식에서는 26,124개로 약 1/16에 해당한다. 딥러닝 모델의 우수함은 여러가지 지표로 비교할 수 있고, 주로 사용되는 것이 정확도 및 추론 속도이다. 연산량이 적다는 것은 곧 추론 속도가 빠르다는 것을 의미하고, 따라서, 세 번째 방식이 두 번째 방식보다 바람직한 접근 방식이라고 생각한다.

또, 현재는 동일한 정확도를 보이고 있지만, 만약 더욱 다양한 종류의 테스트 케이스들이 추가된다면 세 번째 방식이 월등히 높은 성능을 보일 것이라고 생각한다. 제공된 학습 데이터인 각 클래스 별 20장, 총 80장은 general한 feature들을 뽑을 수 있는 convolution filter를 학습하기에는 터무니없이 부족한 숫자이며, 데이터 수가 적으니 학습 iteration을 늘릴 경우 overfitting 되기도 쉽기 때문이다.

[Source Code – preprocessing.py]

```
import os
from PIL import Image
import numpy as np
import cv2

def saveimage(img, name):
    img=np.uint8(img)
    img=Image.fromarray(img)
    img.save(name)
    return

def make_MNIST(img, col, row, grayscale=False):
    size = max(col, row)
    if grayscale:
        template = 255 * np.ones((size, size, 1))
    else:
        template = 255 * np.ones((size, size, 3))

    if size == col:
        template[(size-row)//2:(size-row)//2+row, :] = img
    else:
        template[:, (size-col)//2:(size-col)//2+col] = img

    img = cv2.resize(template, (28, 28))
    return np.array(img)

if __name__=="__main__":
    dir_path = "./data/Provided"

    for (root, directories, files) in os.walk(dir_path):
        for file in files:
            file_path = os.path.join(root, file)
            temp = file_path.split('/')
            file_name = temp[-1]
            print("filename: ", file_name)

            temp[2] = "MNIST_rgb_data"
            MNIST_rgb_path = '/'.join(temp)
            temp[2] = "MNIST_gray_data"
            MNIST_gray_path = '/'.join(temp)

            img = Image.open(file_path).convert('RGB') # RGBA -> RGB
            col, row = img.size
            img = np.array(img)

            rgb_img = make_MNIST(img, col, row, grayscale=False)
            rcimg = img[:,:,0] # r-channel img
            gcimg = img[:,:,1] # g-channel img
            bcimg = img[:,:,2] # b-channel img
            gimg = np.uint8(0.299 * rcimg + 0.587 * gcimg + 0.114 * bcimg).reshape(row,
col, 1) # Convert a color image to a grayscale image
            gray_img = make_MNIST(gimg, col, row, grayscale=True)

            cv2.imwrite(MNIST_rgb_path, rgb_img)
            saveimage(gray_img, MNIST_gray_path)
```

[Source Code – save_orig_MNIST.py]

```
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import numpy as np
import cv2
from PIL import Image


def saveimage(img, name):
    cv2.imwrite(name, img)
    return

if __name__=="__main__":
    train_dataset = datasets.MNIST(root='./data', train=True,
transform=transforms.ToTensor(), download=True)
    test_dataset = datasets.MNIST(root='./data', train=False,
transform=transforms.ToTensor(), download=True)


    for i in range(len(train_dataset)):
        data = train_dataset[i]
        img = 255 * (1 - np.array(data[0]).reshape(28, 28, 1))

        label = data[1]
        filname=f"img_{i}.png"

        if label == 1:
            path = "./MNIST_original/1/" + filname
        elif label == 2:
            path = "./MNIST_original/2/" + filname
        elif label == 3:
            path = "./MNIST_original/3/" + filname
        elif label == 4:
            path = "./MNIST_original/4/" + filname
        else:
            continue

        saveimage(img, path)

    for i in range(len(test_dataset)):
        data = test_dataset[i]
        img = np.array(data[0]).reshape(28, 28, 1)
        label = data[1]
        filname=f"img_{i}.png"

        if label == 1:
            path = "./MNIST_original/1/" + filname
        elif label == 2:
            path = "./MNIST_original/2/" + filname
        elif label == 3:
            path = "./MNIST_original/3/" + filname
        elif label == 4:
            path = "./MNIST_original/4/" + filname
        else:
            continue
        saveimage(img, path)
```

[Source code – 1_bc_with_PCA.py]

```
import numpy as np
import os
from PIL import Image
from torch.utils.data import Dataset, DataLoader
# import torchvision.dataset as datasets
import numpy as np
import torch
import cv2
from sklearn.decomposition import PCA

class CustomImageDataset(Dataset):
    def read_data_set(self):
        all_img_files = []
        all_labels = []

        class_names = os.walk(self.data_set_path).__next__()[1]

        for index, class_name in enumerate(class_names):
            label = index
            img_dir = os.path.join(self.data_set_path, class_name)
            img_files = os.walk(img_dir).__next__()[2]

            for img_file in img_files:
                img_file = os.path.join(img_dir, img_file)
                img = Image.open(img_file)
                if img is not None:
                    all_img_files.append(img_file)
                    all_labels.append(label)

        return all_img_files, all_labels, len(all_img_files), len(class_names)

    def __init__(self, data_set_path, transforms=None):
        self.data_set_path = data_set_path
        self.image_files_path, self.labels, self.length, self.num_classes =
self.read_data_set()
        self.transforms = transforms

    def __getitem__(self, index):
        image = Image.open(self.image_files_path[index])
        image = image.convert("RGB")

        if self.transforms is not None:
            image = self.transforms(image)

        return {'image': image, 'label': self.labels[index]}

    def __len__(self):
        return self.length


def bc(inp, meanp, cv, pr): # cv is covariance matrix of {cl} * {descriptor} x
{descriptor}, pr is probability of classes {cl}
    cl, dim = np.shape(meanp) # 4, 784
    dfn = np.full(cl, 0.0)
    mdfn = 10 ** 100
    inp = np.matrix(inp)
    meanp = np.matrix(meanp)
    for i in range(cl):
        cv[i] = np.matrix(cv[i])

    for i in range(cl):
```

```
        dfn[i] = np.log(pr[i]) - 0.5*np.linalg.det(cv[i]) + 0.5*(inp-meanp[i,:]) *
np.linalg.inv(cv[i])*np.transpose(inp-meanp[i,:])

        if dfn[i] < mdfn:
            mdfn = dfn[i]
            mcl = i
    return dfn, mcl


def run_exp(number):
    torch.manual_seed(7)
    np.random.seed(7)
    selected_dataset = 'Gray'

    if selected_dataset== 'RGB':
        print("Select RGB Dataset")
        train_data_set = CustomImageDataset(data_set_path="./data/MNIST_rgb_data/train")
        train_loader = DataLoader(train_data_set, batch_size=1, shuffle=False)

        test_data_set = CustomImageDataset(data_set_path='./data/MNIST_rgb_data/test')
        test_loader = DataLoader(test_data_set, batch_size=1, shuffle=False)

    else:
        print("Select Gray Dataset")
        train_data_set = CustomImageDataset(data_set_path="./data/MNIST_gray_data/train")
        train_loader = DataLoader(train_data_set, batch_size=1, shuffle=False)

        test_data_set = CustomImageDataset(data_set_path='./data/MNIST_gray_data/test')
        test_loader = DataLoader(test_data_set, batch_size=1, shuffle=False)

    # class 별 데이터 읽어오기. cl1, cl2, ...
    cl1 = [] # class 별 이미지들
    cl2 = []
    cl3 = []
    cl4 = []

    train_label = []
    for data in train_data_set:
        if selected_dataset=='RGB':
            img = np.array(data['image']).flatten() # 784 * 3

        else:
            img = np.array(data['image'])[:,:,1].flatten() # 784

        label = data['label']
        train_label.append(label)
        if label == 0:
            cl1.append(img)
        elif label == 1:
            cl2.append(img)
        elif label == 2:
            cl3.append(img)
        elif label == 3:
            cl4.append(img)

    train_total = np.array(cl1 + cl2 + cl3 + cl4) # total img (80 * 784)

    pca = PCA(n_components=number)
    train_feature = pca.fit_transform(train_total)


    # ... 모든 class 값 읽어오기
    test_imgs = [] # test data로 추정
    test_labels = []
```

```
    for data in test_data_set:
        if selected_dataset=='RGB':
            img = np.array(data['image']).flatten() # 784 * 3

        else:
            img = np.array(data['image'])[:,:,1].flatten() # 784
        label = data['label']
        test_imgs.append(img)
        test_labels.append(label)


    cl1_feature = train_feature[:20]
    cl2_feature = train_feature[20:40]
    cl3_feature = train_feature[40:60]
    cl4_feature = train_feature[60:]

    cl1c = np.cov(cl1_feature.T)
    cl2c = np.cov(cl2_feature.T)
    cl3c = np.cov(cl3_feature.T)
    cl4c = np.cov(cl4_feature.T)

    mp = np.array([np.mean(cl1_feature, axis=0), np.mean(cl2_feature, axis=0),
np.mean(cl3_feature, axis=0), np.mean(cl4_feature, axis=0)])

    cv = np.array([cl1c, cl2c, cl3c, cl4c])
    pr = np.array([1/len(cv) for _ in range(len(cv))])


    test_feature = pca.transform(test_imgs)

    test_pred = []
    correct = 0

    for i in range(len(test_feature)):
        input1 = test_feature[i] # 32-dim feature vector, results from PCA that is fitted
with training data
        df, decide = bc(input1, mp, cv, pr)
        test_pred.append(decide)
        if decide == test_labels[i]:
            correct += 1

    total = len(test_labels)
    print("PCA n_components : ", number)
    print("Test lables       :", test_labels)
    print("Predicted lables  :", test_pred)
    print("Test Accuracy of the model on the {} test images: {} %".format(len(test_labels),
100 * correct / total))
    print()
    return

if __name__=="__main__":
    run_exp(41)
```

[Source code – 2_cnn.py]

```
import torch
import os

from PIL import Image
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
# import torchvision.dataset as datasets
from torchvision import datasets
import torchvision.transforms as transforms

# conda install pytorch==1.5.1 torchvision==0.6.1 cpuonly -c pytorch

class CustomImageDataset(Dataset):
    def read_data_set(self):
        all_img_files = []
        all_labels = []

        class_names = os.walk(self.data_set_path).__next__()[1]

        for index, class_name in enumerate(class_names):
            label = index
            img_dir = os.path.join(self.data_set_path, class_name)
            img_files = os.walk(img_dir).__next__()[2]

            for img_file in img_files:
                img_file = os.path.join(img_dir, img_file)
                img = Image.open(img_file)
                if img is not None:
                    all_img_files.append(img_file)
                    all_labels.append(label)

        return all_img_files, all_labels, len(all_img_files), len(class_names)

    def __init__(self, data_set_path, transforms=None):
        self.data_set_path = data_set_path
        self.image_files_path, self.labels, self.length, self.num_classes =
self.read_data_set()
        self.transforms = transforms

    def __getitem__(self, index):
        image = Image.open(self.image_files_path[index])
        image = image.convert("RGB")

        if self.transforms is not None:
            image = self.transforms(image)

        return {'image': image, 'label': self.labels[index]}

    def __len__(self):
        return self.length


class CustomConvNet(nn.Module):
    def __init__(self, num_classes):
        super(CustomConvNet, self).__init__()
        self.num_classes = num_classes
        self.layer1 = self.conv_module(3, 16)
        self.layer2 = self.conv_module(16, 32)
        self.layer3 = self.conv_module(32, 64)
        self.layer4 = self.conv_module(64, 128)
        self.layer5 = self.conv_module(128, 256)
        self.gap = self.global_avg_pool(256, num_classes)
```

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = self.gap(out)
        out = out.view(-1, self.num_classes)
        return out

    def conv_module(self, in_num, out_num):
        return torch.nn.Sequential(
            torch.nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(out_num),
            torch.nn.LeakyReLU(),
            # torch.nn.MaxPool2d(kernel_size=2, stride=2)
        )
    def global_avg_pool(self, in_num, out_num):
        return torch.nn.Sequential(
            torch.nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(out_num),
            torch.nn.LeakyReLU(),
            torch.nn.AdaptiveAvgPool2d((1,1))
        )

hyper_param_epoch = 30000
hyper_param_batch = 80
hyper_param_learning_rate = 0.001
lr_sched_step = int(hyper_param_epoch//3)
lr_sched_gamma = 0.7

transforms_train = transforms.Compose([
    transforms.RandomRotation(10.),
    transforms.ToTensor()
])

transforms_test = transforms.Compose([
    transforms.ToTensor()
])


selected_dataset = 'Gray'

if selected_dataset== 'RGB':
    print("Select RGB Dataset")
    train_data_set = CustomImageDataset(data_set_path="./data/MNIST_rgb_data/train",
transforms=transforms_train)
    train_loader = DataLoader(train_data_set, batch_size=hyper_param_batch, shuffle=True)
    nmtrainfile=train_data_set.length
    trainlabels=train_data_set.labels

    test_data_set = CustomImageDataset(data_set_path='./data/MNIST_rgb_data/test',
transforms=transforms_test)
    test_loader = DataLoader(test_data_set, batch_size=hyper_param_batch, shuffle=False)
    nmtestfile=test_data_set.length

else:
    print("Select Gray Dataset")
    train_data_set = CustomImageDataset(data_set_path="./data/MNIST_gray_data/train",
transforms=transforms_train)
    train_loader = DataLoader(train_data_set, batch_size=hyper_param_batch, shuffle=True)
    nmtrainfile=train_data_set.length
    trainlabels=train_data_set.labels
```

```
    test_data_set = CustomImageDataset(data_set_path='./data/MNIST_gray_data/test',
transforms=transforms_test)
    test_loader = DataLoader(test_data_set, batch_size=hyper_param_batch, shuffle=False)
    nmtestfile=test_data_set.length

if not (train_data_set.num_classes == test_data_set.num_classes):
    print("error: Numbers of class in training set and test set are not equal")
    exit()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

num_classes = train_data_set.num_classes
custom_model = CustomConvNet(num_classes=num_classes).to(device)

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(custom_model.parameters(), lr=hyper_param_learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=lr_sched_step,
gamma=lr_sched_gamma)

for e in range(hyper_param_epoch):
    custom_model.train()
    for i_batch, item in enumerate(train_loader):
        images = item['image'].to(device)
        labels = item['label'].to(device)

        outputs = custom_model(images)

        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    scheduler.step()

    if (e+1) % 100 == 0:
        print('Epoch [{}/{}], Loss: {:.4f}'.format(e+1, hyper_param_epoch, loss.item()))

custom_model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    testlabels = []
    predictedlabels = []
    for item in test_loader:
        images = item['image'].to(device)
        labels = item['label'].to(device)
        outputs = custom_model(images)
        _, predicted = torch.max(outputs.data, 1)

        testlabels += labels.tolist()
        predictedlabels += predicted.tolist()
        total += len(labels)
        correct += (predicted==labels).sum().item()

    print("\nTraining labels :", trainlabels)
    print("Test lables       :", testlabels)
    print("Predicted lables  :", predictedlabels)
    print("Test Accuracy of the model on the {} test images: {} %".format(total, 100 *
correct / total))
```

[Source code – 3_cnn_modified.py]

```
import torch
import os

from PIL import Image
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
# import torchvision.dataset as datasets
from torchvision import datasets
import torchvision.transforms as transforms
import numpy as np

# conda install pytorch==1.5.1 torchvision==0.6.1 cpuonly -c pytorch

class CustomImageDataset(Dataset):
    def read_data_set(self):
        all_img_files = []
        all_labels = []

        class_names = os.walk(self.data_set_path).__next__()[1]

        for index, class_name in enumerate(class_names):
            label = index
            img_dir = os.path.join(self.data_set_path, class_name)
            img_files = os.walk(img_dir).__next__()[2]

            for img_file in img_files:
                img_file = os.path.join(img_dir, img_file)
                img = Image.open(img_file)
                if img is not None:
                    all_img_files.append(img_file)
                    all_labels.append(label)

        return all_img_files, all_labels, len(all_img_files), len(class_names)

    def __init__(self, data_set_path, transforms=None):
        self.data_set_path = data_set_path
        self.image_files_path, self.labels, self.length, self.num_classes =
self.read_data_set()
        self.transforms = transforms

    def __getitem__(self, index):
        image = Image.open(self.image_files_path[index])
        image = image.convert("RGB")

        if self.transforms is not None:
            image = self.transforms(image)

        return {'image': image, 'label': self.labels[index]}

    def __len__(self):
        return self.length


class CustomConvNet(nn.Module):
    def __init__(self, num_classes):
        super(CustomConvNet, self).__init__()
        self.num_classes = num_classes
        self.layer1 = self.conv_module(3, 16)
        self.layer2 = self.conv_module(16, 32)
        self.layer3 = self.conv_module(32, 64)
        self.gap = self.global_avg_pool(64, num_classes)
```

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.gap(out)
        out = out.view(-1, self.num_classes)
        return out

    def conv_module(self, in_num, out_num):
        return torch.nn.Sequential(
            torch.nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(out_num),
            torch.nn.LeakyReLU(),
            # torch.nn.MaxPool2d(kernel_size=2, stride=2)
        )
    def global_avg_pool(self, in_num, out_num):
        return torch.nn.Sequential(
            torch.nn.Conv2d(in_num, out_num, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(out_num),
            torch.nn.LeakyReLU(),
            torch.nn.AdaptiveAvgPool2d((1,1))
        )

hyper_param_epoch = 420
hyper_param_batch = 100
hyper_param_learning_rate = 0.001
lr_sched_step = 200
lr_sched_gamma = 0.7

transforms_train = transforms.Compose([
    transforms.RandomRotation(10.),
    transforms.ToTensor()
])

transforms_test = transforms.Compose([
    transforms.ToTensor()
])


train_data_set = CustomImageDataset(data_set_path="./data/MNIST_gray_total/train",
transforms=transforms_train)
train_loader = DataLoader(train_data_set, batch_size=hyper_param_batch, shuffle=True)
nmtrainfile=train_data_set.length
trainlabels=train_data_set.labels

test_data_set = CustomImageDataset(data_set_path='./data/MNIST_gray_total/test',
transforms=transforms_test)
test_loader = DataLoader(test_data_set, batch_size=hyper_param_batch, shuffle=False)
nmtestfile=test_data_set.length

if not (train_data_set.num_classes == test_data_set.num_classes):
    print("error: Numbers of class in training set and test set are not equal")
    exit()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

num_classes = train_data_set.num_classes
custom_model = CustomConvNet(num_classes=num_classes).to(device)

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(custom_model.parameters(), lr=hyper_param_learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=lr_sched_step,
gamma=lr_sched_gamma)
```

```
for e in range(hyper_param_epoch):
    for i_batch, item in enumerate(train_loader):
        images = item['image'].to(device)
        labels = item['label'].to(device)

        outputs = custom_model(images)

        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i_batch + 1) * hyper_param_batch >= nmtrainfile:
            print('Epoch [{}/{}], Loss: {:.4f}'.format(e+1, hyper_param_epoch,
loss.item()))
    scheduler.step()

    # if (e+1)%30 == 0:
custom_model.eval()

with torch.no_grad():
    correct = 0
    total = 0
    testlabels = []
    predictedlabels = []
    for item in test_loader:
        images = item['image'].to(device)
        labels = item['label'].to(device)
        outputs = custom_model(images)
        _, predicted = torch.max(outputs.data, 1)

        testlabels += labels.tolist()
        predictedlabels += predicted.tolist()
        total += len(labels)
        correct += (predicted==labels).sum().item()

    print("\nNumber of training labels :", len(trainlabels))
    print("Train epochs      :", hyper_param_epoch)
    print("Test lables       :", testlabels)
    print("Predicted lables  :", predictedlabels)
    print("Test Accuracy of the model on the {} test images: {} %".format(total, 100 *
correct / total))
```