Code :

CNN.py

```
import torch.nn as nn
class Mymodel(nn.Module):
    def __init__(self):
        super().__init__()
        self.model=nn.Sequential(

nn.Conv2d(in_channels=1,out_channels=8,kernel_size=3,padding='same'),
                nn.ReLU(),
                nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,
padding='same'),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=2),
                nn.Dropout(0.25),
                nn.Flatten(),
                nn.Linear(in_features=16*4*4,out_features=128),
                nn.ReLU(),
                nn.Dropout(0.5),
                nn.Linear(in_features=128,out_features=10)
        )

    def forward(self,input):
        output=self.model(input)
        return output
```

classifier.py

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import Dataset,DataLoader
import torch.nn as nn
import torch
from torch.utils.tensorboard import SummaryWriter
from torchsummary import summary
from CNN import Mymodel
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import random
```

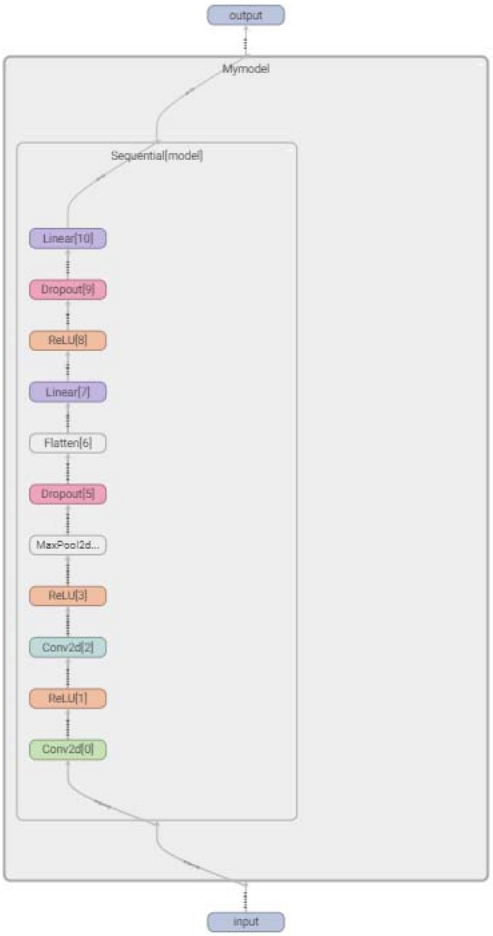
```
class Mydata(Dataset):
    def __init__(self,X,y):
        self.X=X
        self.y=y

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

    def __getitem__(self, idx):
        img=self.X[idx]
        label=self.y[idx]
        img=torch.tensor(data=img,dtype=torch.float32)
        label=torch.tensor(data=label)
        label = label.type(torch.LongTensor)
        return img,label

def train(model, writer,device, train_loader, optimizer, loss_fn,epoch):
    model.train()
    train_loss=0
    train_correct=0
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        pred = output.argmax(dim=1)
        train_correct+=pred.eq(target.view_as(pred)).sum().item()
        loss = loss_fn(output, target)
        train_loss += loss.item()
        loss.backward()
        optimizer.step()

    print('The loss of the model on the train dataset:{}'.format(train_loss))
    writer.add_scalar(tag='train_loss',scalar_value=train_loss,global_step=epoch)
    accuracy=train_correct/len(train_loader.dataset)
    writer.add_scalar(tag='train_acc', scalar_value=accuracy, global_step=epoch)
    print('The predictive accuracy of the model on the train
dataset:{}'.format(accuracy))

def test(model, writer,device, test_loader,loss_fn,epoch,test_acc):
    model.eval()
    test_loss = 0
    test_correct = 0
    with torch.no_grad():
```

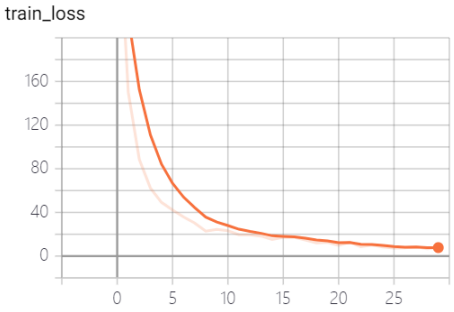
```
            for data, target in test_loader:
                data, target = data.to(device), target.to(device)
                output = model(data)
                test_loss += loss_fn(output, target)
                pred = output.argmax(dim=1, keepdim=True)
                test_correct += pred.eq(target.view_as(pred)).sum().item()

        print('The loss of the model on the test dataset:{}'.format(test_loss))
        writer.add_scalar(tag='test_loss', scalar_value=test_loss, global_step=epoch)
        accuracy = test_correct / len(test_loader.dataset)
        writer.add_scalar(tag='test_acc', scalar_value=accuracy, global_step=epoch)
        print('The predictive accuracy of the model on the test
dataset:{}'.format(accuracy))
        if accuracy>test_acc:
            torch.save(model,'./model/BEST_CNN_MODEL.pth')
        return accuracy

if __name__ == '__main__':
        dataset = datasets.load_digits()
        x_data = dataset.data
        y_data = dataset.target
        x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,test_size=0.2,
random_state=0)
        x_train=x_train.reshape((len(x_train),1,8,8,))
        x_test=x_test.reshape((len(x_test),1,8,8))
        train_dataset=Mydata(x_train,y_train)
        test_dataset=Mydata(x_test,y_test)
        train_dataloader=DataLoader(dataset=train_dataset,batch_size=8)
        test_dataloader=DataLoader(dataset=test_dataset,batch_size=8)
        writer=SummaryWriter('logs')
        for i,(img,label) in enumerate(train_dataloader):

writer.add_images(tag='train',img_tensor=img,global_step=i,dataformats="NCHW")

        for i,(img,label) in enumerate(test_dataloader):

writer.add_images(tag='test',img_tensor=img,global_step=i,dataformats="NCHW")

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

        model=Mymodel()
        writer.add_graph(model=model,input_to_model=torch.randn(8,1,8,8))
        summary(model=model,input_size=(1,8,8))
        model=model.to(device)
```

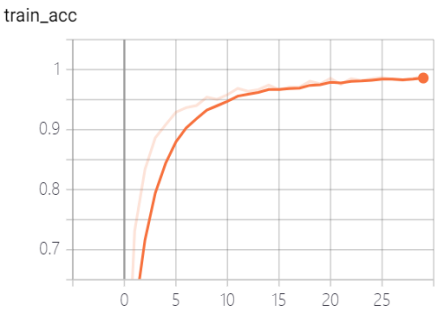
```
    optimizer=torch.optim.SGD(model.parameters(), lr=0.01)
    scheduler = StepLR(optimizer, step_size=20, gamma=0.9)
    loss_fn=nn.CrossEntropyLoss()
    loss_fn.to(device)
    EPOCH=30
    test_acc=0
    for epoch in range(EPOCH):

print('EPOCH**************************{}/{}'.format(epoch+1,EPOCH))
        train(model=model, writer=writer,device=device,
train_loader=train_dataloader, optimizer=optimizer, loss_fn=loss_fn,epoch=epoch)
        test_acc=test(model=model, writer=writer,device=device,
test_loader=test_dataloader, loss_fn=loss_fn,epoch=epoch,test_acc=test_acc)
        scheduler.step()
    writer.close()

    # confusion matrix
    x_test=torch.tensor(x_test,dtype=torch.float32)
    y_pred=model(x_test).argmax(dim=1)
    cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=range(10))
    disp.plot()
    plt.savefig('./figure/cm.png',dpi=600)

    # Output the figure and the predicted number vs the correct answer, for ten of
test images.
    index=random.sample(range(len(x_test)), 10)
    for i in index:
        plt.figure()
        x=torch.tensor(x_test[i].reshape(1,1,8,8),dtype=torch.float32)
        plt.imshow(x_test[i].reshape(8,8))
        plt.title(y_test[i])
        plt.colorbar()
        y_pred = model(x).argmax(dim=1)
        plt.show(block=True)
        print('true value:{}'.format(y_test[i]))
        print('predicted value:{}'.format(y_pred.item()))
```

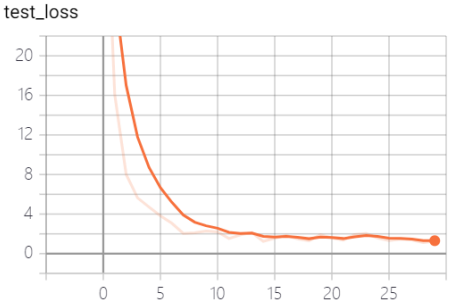
original image :

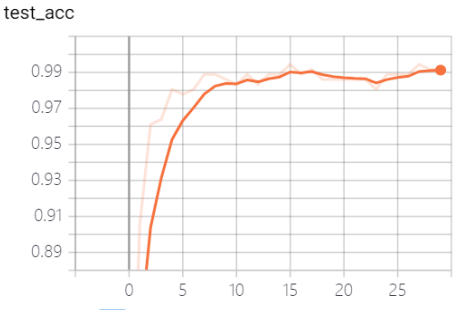
network structure :



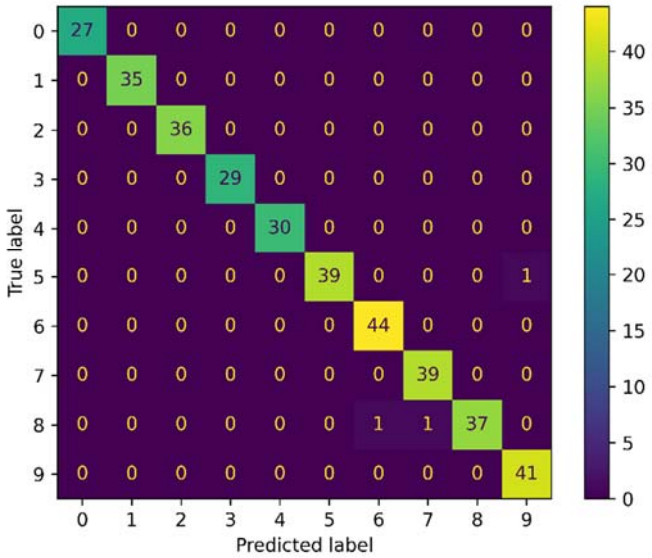




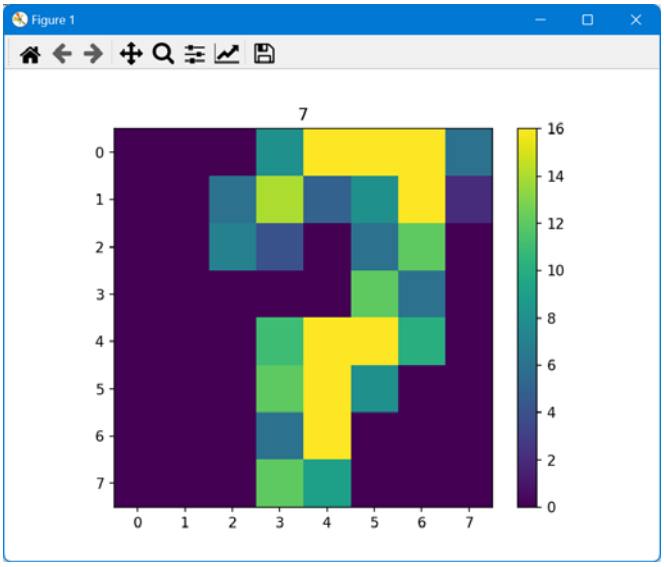




confusion matrix:



Output the figure and the predicted number vs the correct answer:



true value:7

predicted value:7