验证模型

使用测试集验证模型

```
#
import torch, torchvision
import matplotlib.pyplot as plt
from FNN_MODEL import MyNet
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
# 张量转换器
to_tensor = ToTensor()
# 加载测试集
test_set = torchvision.datasets.MNIST("", transform=to_tensor, train=False, download=False)
test_loader = DataLoader(test_set,shuffle=True)
# 迭代器
iterator = iter(test_loader)
# 加载模型
model = MyNet()
model.load_state_dict(torch.load("FNN_MODEL_PARARM.pt"))
# 循环9张图片
for i in range(9):
    images, labels = next(iterator)
    img = images[0]
    # 使用模型验证
    outputs = model.forward(img)
    # 取最大值，即认为的数字
    prediction = torch.argmax(outputs, dim=1)
    # 输出到九宫格
    plt.subplot(3, 3, i + 1)
    plt.title(int(prediction))
    plt.imshow(img[0],"gray")

plt.show()
```

使用自己写的数字图片验证模型

使用画图工具写的数字是白底黑字，而训练时的图片是黑底白字，且像素大小不一致。

将手写图片转换为28*28的单通道黑底白字图片。

```
import torch
from PIL import Image, ImageOps
from torchvision.transforms import ToTensor
# from FNN_MODEL import MyNet
from CNN_MODEL import MyNet
img_path_list = [
    "真实图片/0.png",
    "真实图片/1.png",
```

```
    "真实图片/2.png",
    "真实图片/3.png",
    "真实图片/4.png",
    "真实图片/5.png",
    "真实图片/6.png",
    "真实图片/7.png",
    "真实图片/8.png",
    "真实图片/9.png",
]

# 创建张量转换器对象
to_tensor = ToTensor()
# 创建用于最终验证的张量集合对象
img_tensor_list = []
# 遍历图片，转换格式
for img_path in img_path_list:
    img = Image.open(img_path)
    # 修改为单通道灰度图
    img = img.convert("L")
    # 调整尺寸
    img = img.resize((28, 28))
    # 反转颜色
    img = ImageOps.invert(img)
    # 转换为张量
    img_tensor = to_tensor(img)
    # 添加到张量集合对象中
    img_tensor_list.append(img_tensor)

# 将张量集合对象转换为用于验证模型的张量对象
img_tensor_list = torch.stack(img_tensor_list)

# 创建模型，加载训练好的模型参数
model = MyNet()
# model.load_state_dict(torch.load("FNN_MODEL_PARARM.pt"))
model.load_state_dict(torch.load("CNN_MODEL_PARAM.pt"))
# 使用模型验证
outputs = model.forward(img_tensor_list)
prediction = torch.argmax(outputs, dim=1)
print(prediction)
```

使用Tkinker测试

```
import tkinter as tk
from tkinter import Canvas, Label

from torchvision.transforms import ToTensor
from PIL import Image, ImageDraw, ImageOps
import numpy as np
import torch
# 修改为自己的模型对象
from CNN_MODEL import MyNet
```

```
class HandwritingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("MNIST数字识别")

        # 创建Canvas
        self.canvas = Canvas(root, width=256, height=256, bg='white')
        self.canvas.pack(padx=10, pady=10)

        # 绑定鼠标事件
        self.canvas.bind("<Button-1>", self.on_draw_start)
        self.canvas.bind("<B1-Motion>", self.on_draw_move)

        # 创建一个空的PIL图像用于保存绘制结果
        self.drawing = Image.new("RGB", (256, 256), 'white')
        self.draw = ImageDraw.Draw(self.drawing)
        self.img_tensor = None

        # 保存按钮
        self.save_button = tk.Button(root, text="识别", command=self.on_save_button_clicked)
        self.save_button.pack(pady=20)

        # 清除按钮
        self.clear_button = tk.Button(root, text="清除",
command=self.on_clear_button_clicked)
        self.clear_button.pack(pady=10)

        # 显示预测结果的标签
        self.prediction_label = Label(root, text="", width=20)
        self.prediction_label.pack(pady=20)

        # 加载模型参数
        self.model = MyNet()
        self.model.load_state_dict(torch.load("CNN_MODEL_PARAM.pt"))

    def on_draw_start(self, event):
        self.lastx, self.lasty = event.x, event.y

    def on_draw_move(self, event):
        x, y = event.x, event.y
        self.draw.line((self.lastx, self.lasty, x, y), fill='black', width=10)
        self.canvas.create_line(self.lastx, self.lasty, x, y, fill='black', width=10,
capstyle=tk.ROUND, smooth=tk.TRUE,
                                splinesteps=36)
        self.lastx, self.lasty = x, y

    def save_as_tensor(self):
        img_gray = self.drawing.convert('L')
        img_gray = ImageOps.invert(img_gray)
        resized_image = img_gray.resize((28, 28))
        to_tensor = ToTensor()
```

```
        img_tensor = to_tensor(resized_image).float()
        self.img_tensor = img_tensor.unsqueeze(0)

    def on_save_button_clicked(self):
        self.save_as_tensor()
        outputs = self.model.forward(self.img_tensor)
        prediction = torch.argmax(outputs)
        self.prediction_label.config(text=f"识别结果: {int(prediction)}")

    def on_clear_button_clicked(self):
        self.canvas.delete("all")
        self.drawing = Image.new("RGB", (256, 256), 'white')
        self.draw = ImageDraw.Draw(self.drawing)
        self.img_tensor = None
        self.prediction_label.config(text="")


root = tk.Tk()
app = HandwritingApp(root)
root.mainloop()
```