```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import torch
import torchvision
%matplotlib inline
matplotlib.rcParams['figure.facecolor']='white'
```

```
dataset=torchvision.datasets.MNIST(root="data/",download=True,transform=torchvision.transforms.ToTensor())
```

```
100%|██████████| 9.91M/9.91M [00:02<00:00, 4.55MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 135kB/s]
100%|██████████| 1.65M/1.65M [00:01<00:00, 1.26MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 7.64MB/s]
```

```
len(dataset)
```

```
60000
```

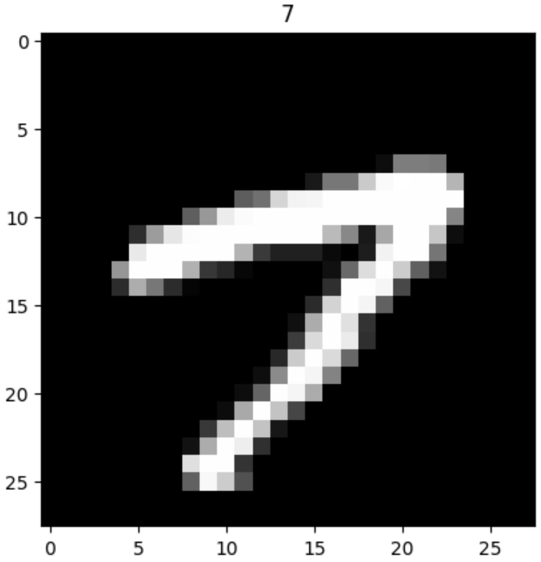
```
type(dataset)
```

```
torchvision.datasets.mnist.MNIST
def __init__(root: Union[str, Path], train: bool=True, transform: Optional[Callable]=None,
target_transform: Optional[Callable]=None, download: bool=False) -> None

/usr/local/lib/python3.11/dist-packages/torchvision/datasets/mnist.py
`MNIST <http://yann.lecun.com/exdb/mnist/>`_ Dataset.

Args:
    root (str or ``pathlib.Path``): Root directory of dataset where ``MNIST/raw/train-images
        and  ``MNIST/raw/t10k-images-idx3-ubyte`` exist
```

```
image,label=dataset[193]
plt.imshow(image[0],cmap='gray')
plt.title(label)
plt.show()
```



```
val_size=10000
train_size=len(dataset)-val_size
train_ds,val_ds=torch.utils.data.random_split(dataset,(train_size,val_size))
```

```
len(train_ds)
```

```
50000
```

```
len(val_ds)
```

```
10000
```

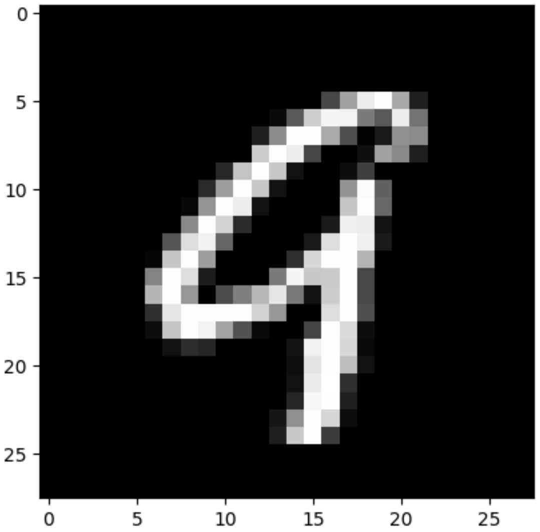
```
test_dataset=torchvision.datasets.MNIST(root="data/",train=False,transform=torchvision.transforms.ToTensor())
```

```
batch_size=64
train_dataloader=torch.utils.data.DataLoader(train_ds,batch_size,shuffle=True,pin_memory=True,num_workers=4)
val_dataloader=torch.utils.data.DataLoader(val_ds,batch_size,shuffle=True,pin_memory=True,num_workers=4)
test_dataloader=torch.utils.data.DataLoader(test_dataset,batch_size,shuffle=True,pin_memory=True,num_workers=4)
```

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number of worke
  warnings.warn(
```

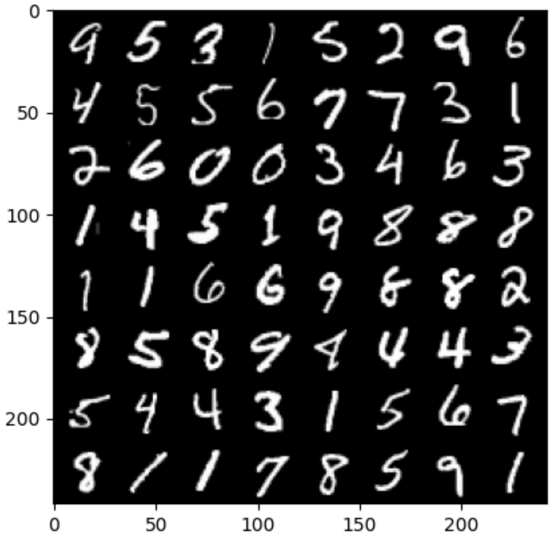
```
for batch in train_dataloader:
    images,labels=batch
    print(images.shape)
    plt.imshow(images[0,0],cmap="gray")
    plt.show()
    print(labels[0].item())
    break
```

```
torch.Size([64, 1, 28, 28])
```



```
9
```

```
images_=torchvision.utils.make_grid(images,nrow=8)
images_=images_.permute(1,2,0)
plt.imshow(images_,cmap='gray')
plt.figure(figsize=(1,1))
plt.axis("off")
plt.show()
```



```
def accuracy(pred,labels):
    _,maxP=torch.max(pred,dim=1)
    return torch.tensor(torch.sum(maxP==labels).item()/len(labels))


class MnistModel(torch.nn.Module):
    def __init__(self,input_size,hidden_size,output_size):
        super().__init__()
        self.layer1=torch.nn.Linear(input_size,hidden_size)
        self.layer2=torch.nn.Linear(hidden_size,output_size)
    def forward(self,X):
        X=X.reshape(-1,784)
        out=self.layer1(X)
        out=torch.nn.functional.relu(out)
        out=self.layer2(out)
        return out
    def training_step(self,batch):
        images,labels=batch
        out=self(images)
        loss=torch.nn.functional.cross_entropy(out,labels)
        return loss
    def validation_step(self,batch):
        images,labels=batch
        out=self(images)
        loss=torch.nn.functional.cross_entropy(out,labels)
        acc=accuracy(out,labels)
        return {"val_acc":acc,"val_loss":loss}
    def validation_step_epoch(self,outputs):
        loss_=[X["val_loss"] for X in outputs]
        loss__=torch.stack(loss_).mean()
        acc_=[X["val_acc"] for X in outputs]
        acc__=torch.stack(acc_).mean()
        return {"val_loss":loss__.item(),"val_acc":acc__.item()}
    def epoch_end(self,epoch,result):
        print("Epoch [{}] Accuracy: {:.4f} Loss: {:.4f}".format(epoch,result["val_acc"],result["val_loss"]))
```

```
def evaluate(model,val_dataloader):
    outputs=[model.validation_step(batch) for batch in val_dataloader]
    return model.validation_step_epoch(outputs)
```

```
def fit(model,epochs,lr,train_dataloader,val_dataloader,opt=torch.optim.SGD):
    history=[]
    optimizer=opt(model.parameters(),lr)
    for epoch in range(epochs):
        for batch in train_dataloader:
            loss=model.training_step(batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        result=evaluate(model,val_dataloader)
        model.epoch_end(epoch,result)
        history.append(result)
    return history
```

```
input_size=784
hidden_size=64
output_size=10
```

```
model1=MnistModel(input_size,hidden_size,output_size)
```

```
history=[evaluate(model1,val_dataloader)]
history
```

```
[{'val_loss': 2.3217451572418213, 'val_acc': 0.10688694566488266}]
```

```
history+=fit(model1,5,0.5,train_dataloader,val_dataloader)
```

```
Epoch [0] Accuracy: 0.9504 Loss: 0.1668
Epoch [1] Accuracy: 0.8983 Loss: 0.3473
Epoch [2] Accuracy: 0.9574 Loss: 0.1363
Epoch [3] Accuracy: 0.9688 Loss: 0.1043
Epoch [4] Accuracy: 0.8929 Loss: 0.4291
```

```
evaluate(model1,test_dataloader)
```

```
{'val_loss': 0.4329960346221924, 'val_acc': 0.8956010937690735}
```

```
torch.cuda.is_available()
```

```
True
```

```
def get_default_device():
    if torch.cuda.is_available():
        return torch.device("cuda")
    return torch.device("cpu")
```

```
device=get_default_device()
```

```
device
```

```
device(type='cuda')
```

```
def to_device(data,device):
    if isinstance(data,(list,tuple)):
        return [to_device(x,device) for x in data]
    return data.to(device,non_blocking=True)
```

```
class dataloader_device:
    def __init__(self,data,device):
        self.data=data
        self.device=device
    def __len__(self):
        return len(self.data)
    def __iter__(self):
        for x in self.data:
            yield to_device(x,self.device)
```

```
train_loader=dataloader_device(train_dataloader,device)
val_loader=dataloader_device(val_dataloader,device)
test_loader=dataloader_device(test_dataloader,device)
```

```
model2=MnistModel(input_size,hidden_size,output_size)
model2=to_device(model2,device)
```

```
history_new=[evaluate(model2,val_loader)]
history_new
```

```
[{'val_loss': 2.299755811691284, 'val_acc': 0.11186305433511734}]
```

```
history_new+=fit(model2,5,0.5,train_loader,val_loader)
```

```
Epoch [0] Accuracy: 0.9499 Loss: 0.1633
Epoch [1] Accuracy: 0.9560 Loss: 0.1464
Epoch [2] Accuracy: 0.9685 Loss: 0.1069
Epoch [3] Accuracy: 0.9493 Loss: 0.1699
Epoch [4] Accuracy: 0.9707 Loss: 0.1004
```

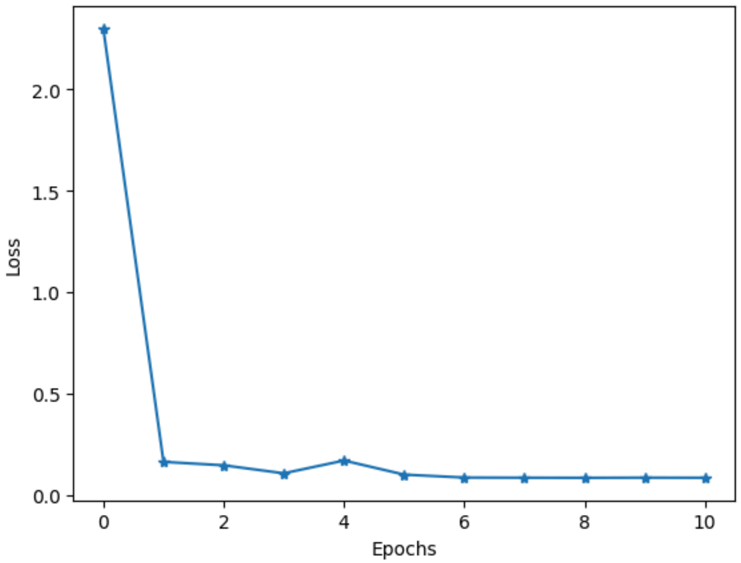
```
history_new+=fit(model2,5,0.05,train_loader,val_loader)
```

```
Epoch [0] Accuracy: 0.9742 Loss: 0.0860
Epoch [1] Accuracy: 0.9740 Loss: 0.0852
Epoch [2] Accuracy: 0.9751 Loss: 0.0846
Epoch [3] Accuracy: 0.9750 Loss: 0.0854
Epoch [4] Accuracy: 0.9746 Loss: 0.0849
```

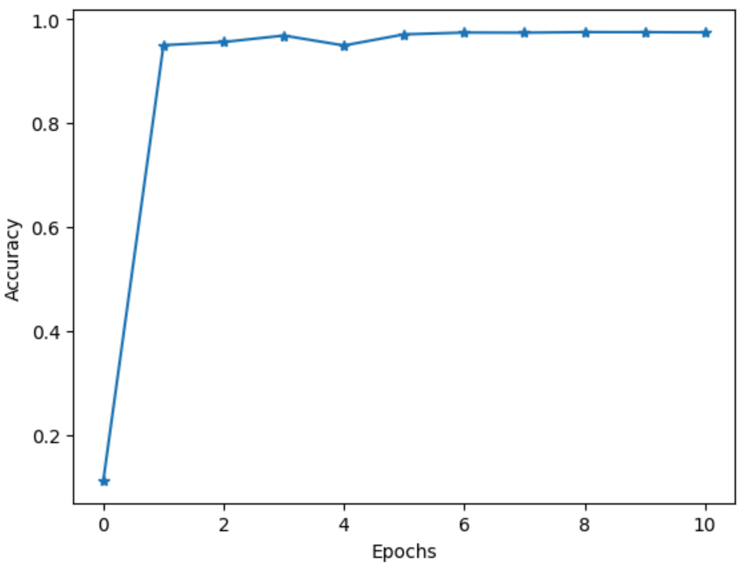
```
evaluate(model2,test_loader)
```

```
{'val_loss': 0.07494346797466278, 'val_acc': 0.9764131903648376}
```

```
loss=[X["val_loss"] for X in history_new]
plt.plot(loss,"-*")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



```
loss=[X["val_acc"] for X in history_new]
plt.plot(loss,"-*")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.show()
```



```
evaluate(model2,test_loader)
```

```
{'val_loss': 0.07445521652698517, 'val_acc': 0.9764131903648376}
```

```
model2
```

```
MnistModel(
  (layer1): Linear(in_features=784, out_features=64, bias=True)
  (layer2): Linear(in_features=64, out_features=10, bias=True)
)
```

```
model2.layer1.weight.numel()+model2.layer1.bias.numel()+model2.layer2.weight.numel()+model2.layer2.bias.numel()
```

```
50890
```

```
def predict(model2,image,device):
    image=to_device(image,device)
    image=image.reshape(-1,784)
    pred=model2(image)
    _,ans=torch.max(pred,dim=1)
    return ans.item()
```

```
image,label=test_dataset[8989]
plt.imshow(image[0],cmap="gray")
plt.show()
print("Output: ",label," Predicted: ",predict(model2,image,device))
```

0
5