1.LeNet网络结构如下：

2.MNIST数据集简介：

(1）数据集包括60000个用于训练的示例和10000个用于测试的示例；

(2）数据集包括0-9共10类手写数字图片，每张图片都做了尺寸归一化，都是28\*28大小。

3.手写字体识别流程：

（1）构建网络模型；

（2）下载、加载数据集MNIST；

（3）定义超参数 ；

（4）定义优化器；

（5）定义训练方法；

（6）定义测试方法；

（7）训练模型并输出预测结果。

4.代码实现

（1）构建网络模型；

class LeNet5(nn.Cell):  
 def \_\_init\_\_(self, num\_class=10, num\_channel=1, include\_top=True):  
 super(LeNet5, self).\_\_init\_\_()  
 self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode='valid')  
 self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode='valid')  
 self.relu = nn.ReLU()  
 self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)  
 self.include\_top = include\_top  
 if self.include\_top:  
 self.flatten = nn.Flatten()  
 self.fc1 = nn.Dense(256, 120, weight\_init=Normal(0.02))  
 self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))  
 self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))  
  
  
 def construct(self, x):  
 x = self.conv1(x)  
 x = self.relu(x)  
 x = self.max\_pool2d(x)  
 x = self.conv2(x)  
 x = self.relu(x)  
 x = self.max\_pool2d(x)  
 if not self.include\_top:  
 return x  
 x = self.flatten(x)  
 x = self.relu(self.fc1(x))  
 x = self.relu(self.fc2(x))  
 x = self.fc3(x)  
 return x

（2）下载、加载数据集MNIST；

def datapipe(path, batch\_size):  
 image\_transforms = [  
 vision.Rescale(1.0 / 255.0, 0),  
 vision.Normalize(mean=(0.1307,), std=(0.3081,)),  
 vision.HWC2CHW()  
 ]  
 label\_transform = transforms.TypeCast(mindspore.int32)  
  
 dataset = MnistDataset(path)  
 dataset = dataset.map(image\_transforms, 'image')  
 dataset = dataset.map(label\_transform, 'label')  
 dataset = dataset.batch(batch\_size)  
 return dataset  
  
train\_dataset = datapipe('MNIST\_Data/train', batch\_size=64)  
test\_dataset = datapipe('MNIST\_Data/test', batch\_size=64)

（3）定义超参数 ；

lr = 0.01  
epochs = 10  
loss\_fn = nn.CrossEntropyLoss()

（4）定义优化器；

optimizer = nn.SGD(model.trainable\_params(), learning\_rate=lr)  
def forward\_fn(data, label):  
 logits = model(data)  
 loss = loss\_fn(logits, label)  
 return loss, logits  
grad\_fn = mindspore.value\_and\_grad(forward\_fn, None, optimizer.parameters, has\_aux=True)

（5）定义训练方法；

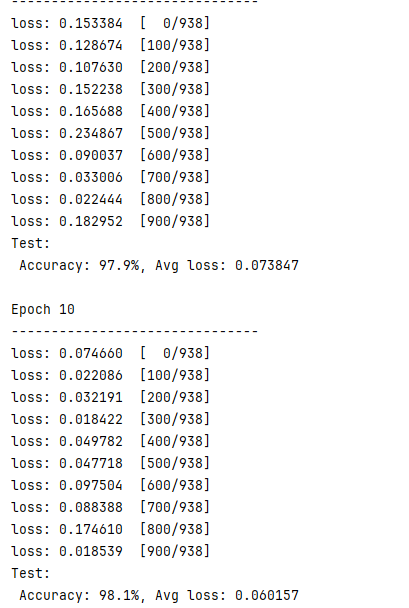
def train\_step(data, label):  
 (loss, \_), grads = grad\_fn(data, label)  
 optimizer(grads)  
 return loss  
def train\_loop(model, dataset):  
 size = dataset.get\_dataset\_size()  
 model.set\_train()  
 for batch, (data, label) in enumerate(dataset.create\_tuple\_iterator()):  
 loss = train\_step(data, label)  
  
 if batch % 100 == 0:  
 loss, current = loss.asnumpy(), batch  
 print(f"loss: {loss:>7f} [{current:>3d}/{size:>3d}]")

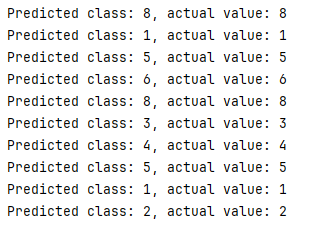
（6）定义测试方法；  
def test\_loop(model, dataset, loss\_fn):  
 num\_batches = dataset.get\_dataset\_size()  
 model.set\_train(False)  
 total, test\_loss, correct = 0, 0, 0  
 for data, label in dataset.create\_tuple\_iterator():  
 pred = model(data)  
 total += len(data)  
 test\_loss += loss\_fn(pred, label).asnumpy()  
 correct += (pred.argmax(1) == label).asnumpy().sum()  
 test\_loss /= num\_batches  
 correct /= total  
 print(f"Test: \n Accuracy: {(100 \* correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")

（7）训练模型并输出预测结果。

for t in range(epochs):  
 print(f"Epoch {t + 1}\n-------------------------------")  
 size = train\_dataset.get\_dataset\_size()  
 model.set\_train()  
 for batch, (data, label) in enumerate(train\_dataset.create\_tuple\_iterator()):  
 (loss, \_), grads = grad\_fn(data, label)  
 optimizer(grads)  
  
 if batch % 100 == 0:  
 loss, current = loss.asnumpy(), batch  
 print(f"loss: {loss:>7f} [{current:>3d}/{size:>3d}]")  
  
 num\_batches = test\_dataset.get\_dataset\_size()  
 model.set\_train(False)  
 total, test\_loss, correct = 0, 0, 0  
 for data, label in test\_dataset.create\_tuple\_iterator():  
 pred = model(data)  
 total += len(data)  
 test\_loss += loss\_fn(pred, label).asnumpy()  
 correct += (pred.argmax(1) == label).asnumpy().sum()  
 test\_loss /= num\_batches  
 correct /= total  
 print(f"Test: \n Accuracy: {(100 \* correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")

5.实验结果

训练中每轮准确率和平均损失：

在测试集上的表现：

在自定义数据集的表现：

