《神经网络与深度学习》第四次作业

# 1 作业要求

基于Mindspore框架，结合具体应用（数据集），设计一个神经网络模型，训练、测试该模型。上传可运行的程序源代码，并提供简单说明文档，包含软件的使用说明，以及模型的训练、测试效果。

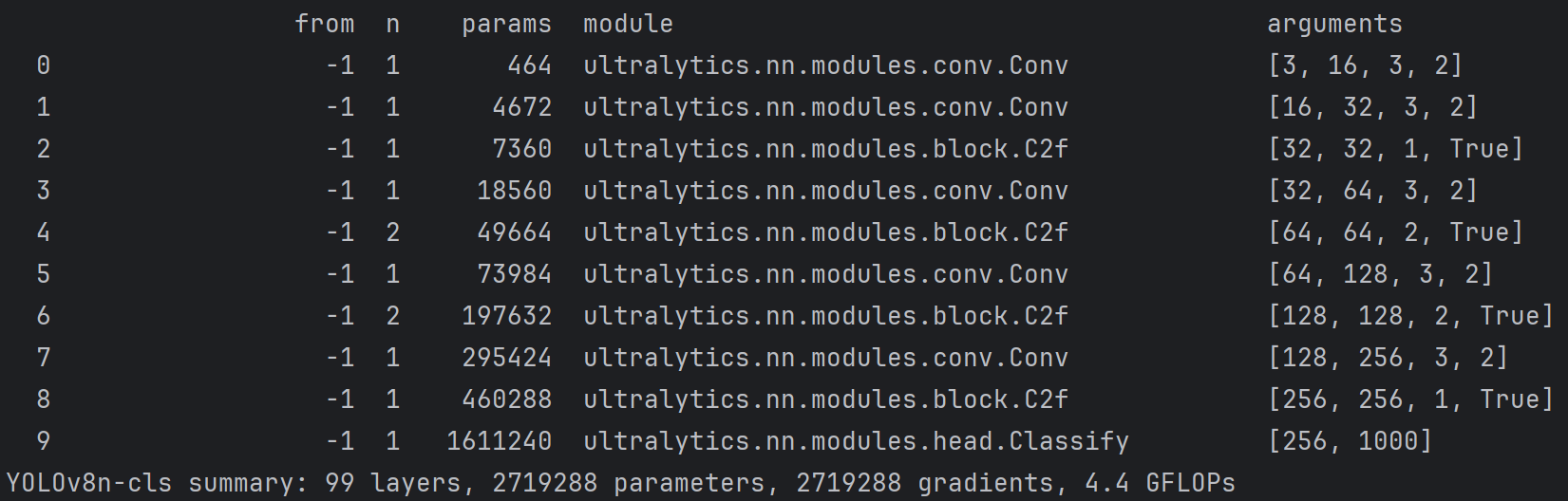
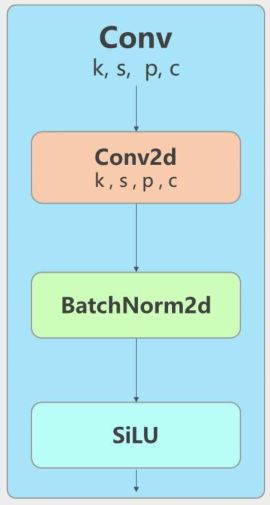
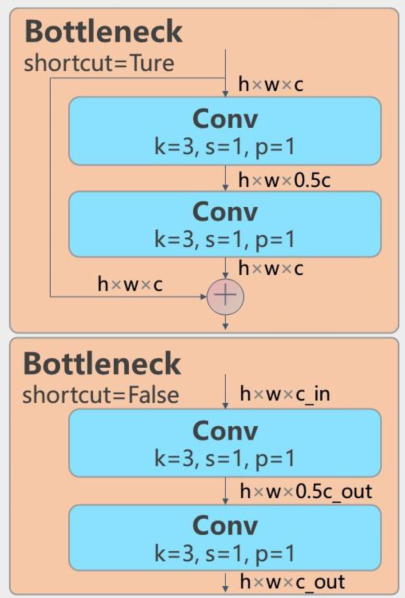
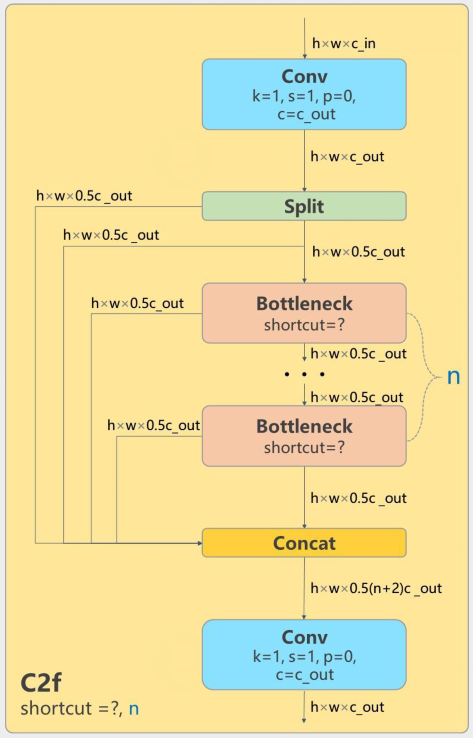
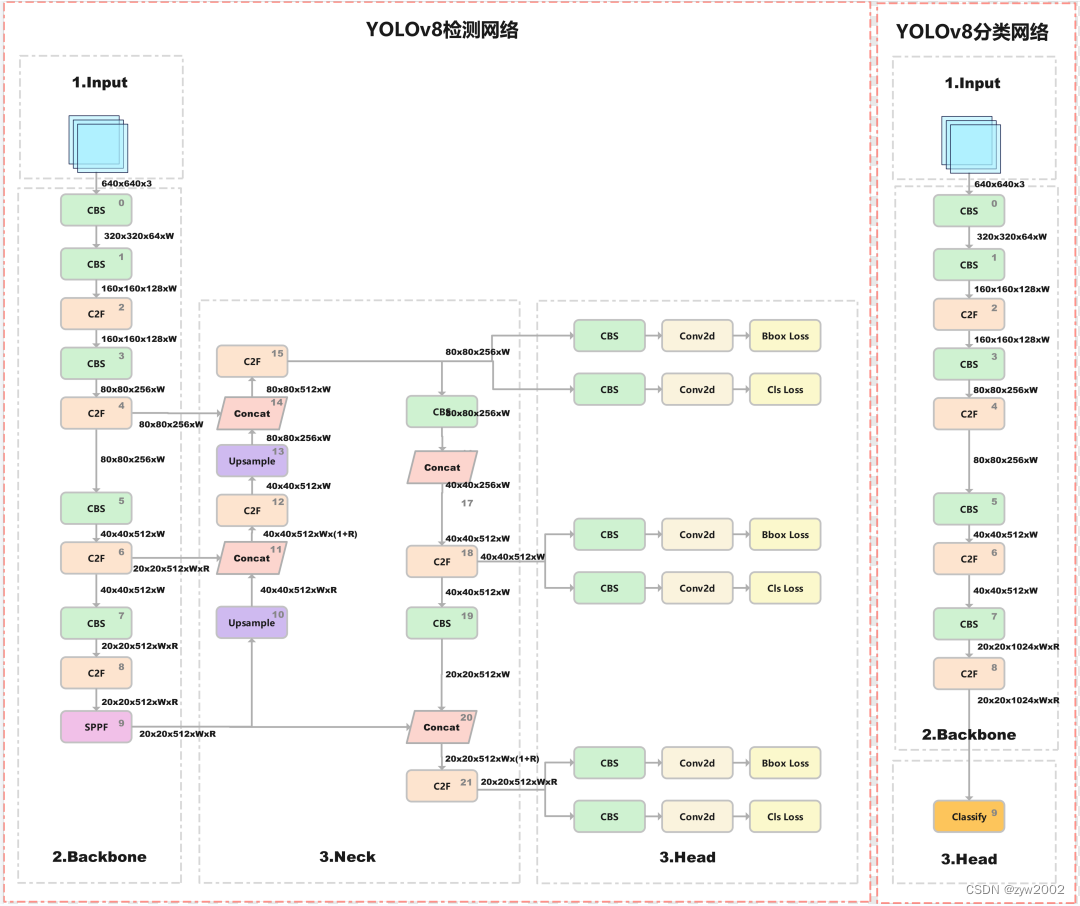
# 2 实现

## 2.1 项目选择与功能

实现基于Mindspore框架的YOLOv8n-cls图像分类模型，实现手写体数字识别。

数据集：Mindspore官网提供的手写体数字识别数据集

YOLOv8n-cls网络构架示意图：



## 2.2 基本模块实现

1. CBS模块

CBS模块是Conv+Batch Norm+SiLU的简称，实现代码如下：

class Conv(nn.Cell):  
 *"""CBS：Conv+BatchNorm+SiLU"""* def \_\_init\_\_(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  
  
 super().\_\_init\_\_()  
 self.pad = autopad(k, p)  
 self.padmode = None  
 if self.pad == 0:  
 self.padmode = 'same'  
 elif self.pad == 1:  
 self.padmode = 'pad'  
  
  
 self.conv = nn.Conv2d(c1, c2, k, s, padding=self.pad, pad\_mode=self.padmode, group=g, has\_bias=False)  
 self.bn = nn.BatchNorm2d(c2)  
 self.act = SiLU() if act is True else (act if isinstance(act, nn.Cell) else ops.Identity())  
  
 def construct(self, x):  
 *"""构造函数，类似torch的forward()"""* x=self.act(self.bn(self.conv(x)))  
  
 return x

class SiLU(nn.Cell): *# SiLU激活函数*  
 def \_\_init\_\_(self):  
 super(SiLU, self).\_\_init\_\_()  
 self.sigmoid = ops.Sigmoid()  
  
 def construct(self, x):  
 return x \* self.sigmoid(x)

def autopad(k, p=None): *#自适应padding* if p is None:  
 p = k // 2 if isinstance(k, int) else [x // 2 for x in k] *# auto-pad* return p

1. Bottlene模块

参与构成C2F模块

class Bottleneck(nn.Cell):  
  
 def \_\_init\_\_(self, c1, c2, shortcut=True, g=1, k=(3, 3), e=0.5):  
 super().\_\_init\_\_()  
 c\_ = int(c2 \* e) *# hidden channels* self.cv1 = Conv(c1, c\_, k[0], 1)  
 self.cv2 = Conv(c\_, c2, k[1], 1, g=g)  
 self.add = shortcut and c1 == c2  
  
 def construct(self, x):  
 return x + self.cv2(self.cv1(x)) if self.add else self.cv2(self.cv1(x))

1. C2F模块

class C2f(nn.Cell):  
  
 def \_\_init\_\_(self, c1, c2, n=1, shortcut=False, g=1, e=0.5):  
 super().\_\_init\_\_()  
 self.c = int(c2 \* e)  
 self.cv1 = Conv(c1, 2 \* self.c, 1, 1)  
 self.cv2 = Conv((2 + n) \* self.c, c2, 1) *# optional act=FReLU(c2)* self.concat=ops.Concat(1)  
 self.m = nn.CellList([Bottleneck(self.c, self.c, shortcut, g, k=(3,3), e=1.0) for \_ in range(n)])  
  
 def construct(self, x):  
 y = list(self.cv1(x).split((self.c, self.c), 1))  
 y.extend(m(y[-1]) for m in self.m)  
 x= self.cv2(self.concat(y))  
 return x

1. Classify检测头

class Classify(nn.Cell):  
  
 def \_\_init\_\_(self, c1, c2, k=1, s=1, p=None, g=1):  
 super().\_\_init\_\_()  
 c\_ = 1280  
 self.conv = Conv(c1, c\_, k, s, p, g)  
 self.flatten=nn.Flatten()  
 self.pool = nn.AdaptiveAvgPool2d(1)  
 self.drop = nn.Dropout(keep\_prob=1.0)  
 self.linear = nn.Dense(c\_, c2)  
 self.expand=ops.ExpandDims()  
  
 def construct(self, x):  
 x=self.conv(x)  
 x=self.pool(x)  
 x=self.flatten(x)  
 x=self.drop(x)  
 x = self.linear(x)  
 return x

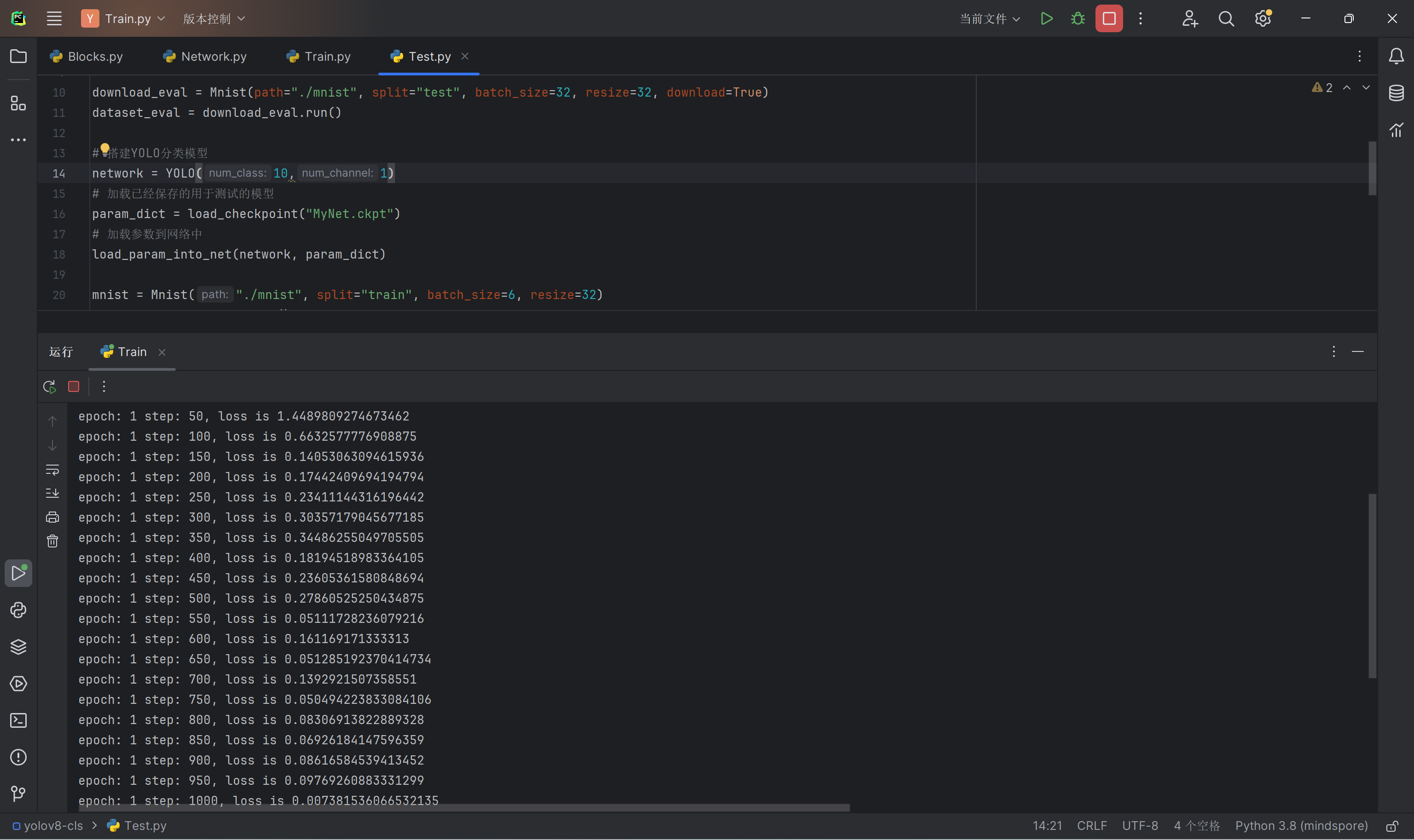
## 2.3 YOLOv8n-cls网络构架

注意：本部分由于Mindspore提供的数据集尺寸很小，所以将C2F模块前的Conv只做1×1卷积，用于扩张通道，这与原来YOLOv8-cls的构架是不符合的。

import mindspore.nn as nn  
import mindspore.ops as ops  
from Blocks import Conv,C2f,Classify  
  
class YOLO(nn.Cell):  
  
 def \_\_init\_\_(self, num\_class=10, num\_channel=3):  
 super(YOLO, self).\_\_init\_\_()  
 *#Backbone:  
 #Conv(c1, c2, k=1, s=1, p=None, g=1, d=1, act=True)  
 #C2f(c1, c2, n=1, shortcut=False, g=1, e=0.5)  
 #Classfiy(c1, c2, k=1, s=1, p=None, g=1)* self.conv=Conv(num\_channel,16,3,2)  
 self.conv1=Conv(16,32,3,2)  
 self.c2f1=C2f(32,32,1,True)  
 self.conv2=Conv(32,64,1,1)  
 self.c2f2=C2f(64,64,2,True)  
 self.conv3 = Conv(64,128,1,1)  
 self.c2f3 = C2f(128,128,2,True)  
 self.conv4 = Conv(128,256,1,1)  
 self.c2f4= C2f(256,256,1,True)  
 *#Head:* self.classfiy=Classify(256,num\_class)  
  
 def construct(self, x):  
 x = self.conv(x)  
 x = self.conv1(x)  
 x = self.c2f1(x)  
 x = self.conv2(x)  
 x = self.c2f2(x)  
 x = self.conv3(x)  
 x = self.c2f3(x)  
 x = self.conv4(x)  
 x = self.c2f4(x)  
 x = self.classfiy(x)  
 return x

## 2.4 模型训练

from mindvision.dataset import Mnist  
import mindspore as ms  
from mindspore import load\_checkpoint, load\_param\_into\_net  
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig,LossMonitor  
import mindspore.nn as nn  
from mindspore.train import Model  
from Network import YOLO  
  
*# 下载并处理MNIST数据集*download\_train = Mnist(path="./mnist", split="train", batch\_size=32, repeat\_num=1, shuffle=True, resize=32, download=True)  
  
download\_eval = Mnist(path="./mnist", split="test", batch\_size=32, resize=32, download=True)  
  
dataset\_train = download\_train.run()  
dataset\_eval = download\_eval.run()  
  
*# 搭建YOLO分类模型*network = YOLO(10,1) *# 类别数，输入通道数*  
*# 定义损失函数*net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')  
  
*# 定义优化器函数*net\_opt = nn.Momentum(network.trainable\_params(), learning\_rate=0.01, momentum=0.9)  
  
*# 初始化模型参数*model = Model(network, loss\_fn=net\_loss, optimizer=net\_opt, metrics={'accuracy'})  
  
*# 训练网络模型，并保存权重为MyNet.ckpt文件*model.train(10, dataset\_train,callbacks=[LossMonitor(50)])  
  
*# 定义的网络模型为net，一般在训练前或者训练后使用*ms.save\_checkpoint(network, "./MyNet.ckpt")



## 2.5 模型测试

from mindvision.dataset import Mnist  
from Network import YOLO  
from mindspore import load\_checkpoint, load\_param\_into\_net  
from mindspore.train import Model  
import numpy as np  
from mindspore import Tensor  
import matplotlib.pyplot as plt  
  
  
download\_eval = Mnist(path="./mnist", split="test", batch\_size=32, resize=32, download=True)  
dataset\_eval = download\_eval.run()  
  
*# 搭建YOLO分类模型*network = YOLO(10,1)  
*# 加载已经保存的用于测试的模型*param\_dict = load\_checkpoint("MyNet.ckpt")  
*# 加载参数到网络中*load\_param\_into\_net(network, param\_dict)  
  
mnist = Mnist("./mnist", split="train", batch\_size=6, resize=32)  
dataset\_infer = mnist.run()  
ds\_test = dataset\_infer.create\_dict\_iterator()  
data = next(ds\_test)  
images = data["image"].asnumpy()  
labels = data["label"].asnumpy()  
  
plt.figure()  
for i in range(1, 7):  
 plt.subplot(2, 3, i)  
 plt.imshow(images[i-1][0], interpolation="None", cmap="gray")  
plt.show()  
  
model = Model(network)  
*# 使用函数model.predict预测image对应分类*output = model.predict(Tensor(data['image']))  
predicted = np.argmax(output.asnumpy(), axis=1)  
  
*# 输出预测分类与实际分类*print(f'Predicted: "{predicted}", Actual: "{labels}"')

