**中国矿业大学**

**计算机科学与技术学院**

**2021级本科生课程设中期计报告**

课程名称 图像处理与视觉感知

设计题目 《图像处理与视觉感知》课程报告

开课学期 2023-2024学年第2学期

报告时间 2024.05.08

学生姓名 杨学通

学 号 08213129

班 级 人工智能2021-1班

专 业 人工智能

任课教师 姚睿

# 实验三 视觉感知

## 实验目的

1. 掌握图像物体检测；
2. 掌握语义分割；

## 实验内容与要求

1. 使用任意深度学习框架搭建图像物体检测网络；
2. 使用任意深度学习框架搭建语义分割网络；

## 实验的具体实现

3.1基本算子定义

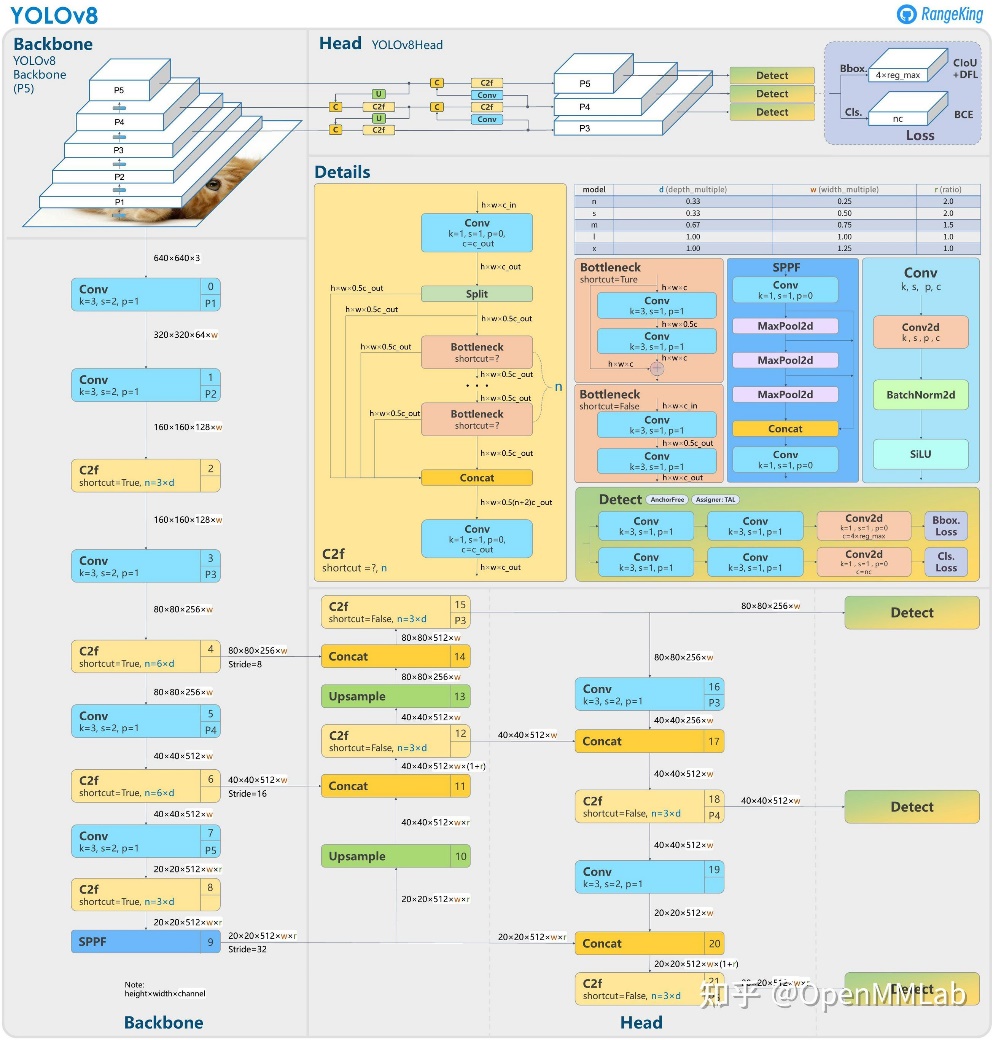


图 1 YOLOv8结构及其算子示意图

#### 3.1.1 Conv模块

class Conv2(Conv):  
 *"""Simplified RepConv module with Conv fusing."""* def \_\_init\_\_(self, c1, c2, k=3, s=1, p=None, g=1, d=1, act=True):  
 *"""Initialize Conv layer with given arguments including activation."""* super().\_\_init\_\_(c1, c2, k, s, p, g=g, d=d, act=act)  
 self.cv2 = nn.Conv2d(c1, c2, 1, s, autopad(1, p, d), groups=g, dilation=d, bias=False) *# add 1x1 conv* def forward(self, x):  
 *"""Apply convolution, batch normalization and activation to input tensor."""* return self.act(self.bn(self.conv(x) + self.cv2(x)))  
  
 def forward\_fuse(self, x):  
 *"""Apply fused convolution, batch normalization and activation to input tensor."""* return self.act(self.bn(self.conv(x)))  
  
 def fuse\_convs(self):  
 *"""Fuse parallel convolutions."""* w = torch.zeros\_like(self.conv.weight.data)  
 i = [x // 2 for x in w.shape[2:]]  
 w[:, :, i[0] : i[0] + 1, i[1] : i[1] + 1] = self.cv2.weight.data.clone()  
 self.conv.weight.data += w  
 self.\_\_delattr\_\_("cv2")  
 self.forward = self.forward\_fuse

#### 3.1.2 C2F模块

class C2f(nn.Module):  
 *"""Faster Implementation of CSP Bottleneck with 2 convolutions."""* def \_\_init\_\_(self, c1, c2, n=1, shortcut=False, g=1, e=0.5):  
 *"""Initialize CSP bottleneck layer with two convolutions with arguments ch\_in, ch\_out, number, shortcut, groups,  
 expansion.  
 """* super().\_\_init\_\_()  
 self.c = int(c2 \* e) *# hidden channels* self.cv1 = Conv(c1, 2 \* self.c, 1, 1)  
 self.cv2 = Conv((2 + n) \* self.c, c2, 1) *# optional act=FReLU(c2)* self.m = nn.ModuleList(Bottleneck(self.c, self.c, shortcut, g, k=((3, 3), (3, 3)), e=1.0) for \_ in range(n))  
  
 def forward(self, x):  
 *"""Forward pass through C2f layer."""* y = list(self.cv1(x).chunk(2, 1))  
 y.extend(m(y[-1]) for m in self.m)  
 return self.cv2(torch.cat(y, 1))  
  
 def forward\_split(self, x):  
 *"""Forward pass using split() instead of chunk()."""* y = list(self.cv1(x).split((self.c, self.c), 1))  
 y.extend(m(y[-1]) for m in self.m)  
 return self.cv2(torch.cat(y, 1))

#### 3.1.3 SPPF模块

class SPPF(nn.Module):  
 *"""Spatial Pyramid Pooling - Fast (SPPF) layer for YOLOv5 by Glenn Jocher."""* def \_\_init\_\_(self, c1, c2, k=5):  
 *"""  
 Initializes the SPPF layer with given input/output channels and kernel size.  
  
 This module is equivalent to SPP(k=(5, 9, 13)).  
 """* super().\_\_init\_\_()  
 c\_ = c1 // 2 *# hidden channels* self.cv1 = Conv(c1, c\_, 1, 1)  
 self.cv2 = Conv(c\_ \* 4, c2, 1, 1)  
 self.m = nn.MaxPool2d(kernel\_size=k, stride=1, padding=k // 2)  
  
 def forward(self, x):  
 *"""Forward pass through Ghost Convolution block."""* x = self.cv1(x)  
 y1 = self.m(x)  
 y2 = self.m(y1)  
 return self.cv2(torch.cat((x, y1, y2, self.m(y2)), 1))

#### 3.1.4 Detect检测头

class Detect(nn.Module):  
 *"""YOLOv8 Detect head for detection models."""* dynamic = False *# force grid reconstruction* export = False *# export mode* shape = None  
 anchors = torch.empty(0) *# init* strides = torch.empty(0) *# init* def \_\_init\_\_(self, nc=80, ch=()):  
 *"""Initializes the YOLOv8 detection layer with specified number of classes and channels."""* super().\_\_init\_\_()  
 self.nc = nc *# number of classes* self.nl = len(ch) *# number of detection layers* self.reg\_max = 16 *# DFL channels (ch[0] // 16 to scale 4/8/12/16/20 for n/s/m/l/x)* self.no = nc + self.reg\_max \* 4 *# number of outputs per anchor* self.stride = torch.zeros(self.nl) *# strides computed during build* c2, c3 = max((16, ch[0] // 4, self.reg\_max \* 4)), max(ch[0], min(self.nc, 100)) *# channels* self.cv2 = nn.ModuleList(  
 nn.Sequential(Conv(x, c2, 3), Conv(c2, c2, 3), nn.Conv2d(c2, 4 \* self.reg\_max, 1)) for x in ch  
 )  
 self.cv3 = nn.ModuleList(nn.Sequential(Conv(x, c3, 3), Conv(c3, c3, 3), nn.Conv2d(c3, self.nc, 1)) for x in ch)  
 self.dfl = DFL(self.reg\_max) if self.reg\_max > 1 else nn.Identity()  
  
 def forward(self, x):  
 *"""Concatenates and returns predicted bounding boxes and class probabilities."""* for i in range(self.nl):  
 x[i] = torch.cat((self.cv2[i](x[i]), self.cv3[i](x[i])), 1)  
 if self.training: *# Training path* return x  
  
 *# Inference path* shape = x[0].shape *# BCHW* x\_cat = torch.cat([xi.view(shape[0], self.no, -1) for xi in x], 2)  
 if self.dynamic or self.shape != shape:  
 self.anchors, self.strides = (x.transpose(0, 1) for x in make\_anchors(x, self.stride, 0.5))  
 self.shape = shape  
  
 if self.export and self.format in ("saved\_model", "pb", "tflite", "edgetpu", "tfjs"): *# avoid TF FlexSplitV ops* box = x\_cat[:, : self.reg\_max \* 4]  
 cls = x\_cat[:, self.reg\_max \* 4 :]  
 else:  
 box, cls = x\_cat.split((self.reg\_max \* 4, self.nc), 1)  
 dbox = self.decode\_bboxes(box)  
  
 if self.export and self.format in ("tflite", "edgetpu"):  
 *# Precompute normalization factor to increase numerical stability  
 # See https://github.com/ultralytics/ultralytics/issues/7371* img\_h = shape[2]  
 img\_w = shape[3]  
 img\_size = torch.tensor([img\_w, img\_h, img\_w, img\_h], device=box.device).reshape(1, 4, 1)  
 norm = self.strides / (self.stride[0] \* img\_size)  
 dbox = dist2bbox(self.dfl(box) \* norm, self.anchors.unsqueeze(0) \* norm[:, :2], xywh=True, dim=1)  
  
 y = torch.cat((dbox, cls.sigmoid()), 1)  
 return y if self.export else (y, x)  
  
 def bias\_init(self):  
 *"""Initialize Detect() biases, WARNING: requires stride availability."""* m = self *# self.model[-1] # Detect() module  
 # cf = torch.bincount(torch.tensor(np.concatenate(dataset.labels, 0)[:, 0]).long(), minlength=nc) + 1  
 # ncf = math.log(0.6 / (m.nc - 0.999999)) if cf is None else torch.log(cf / cf.sum()) # nominal class frequency* for a, b, s in zip(m.cv2, m.cv3, m.stride): *# from* a[-1].bias.data[:] = 1.0 *# box* b[-1].bias.data[: m.nc] = math.log(5 / m.nc / (640 / s) \*\* 2) *# cls (.01 objects, 80 classes, 640 img)* def decode\_bboxes(self, bboxes):  
 *"""Decode bounding boxes."""* return dist2bbox(self.dfl(bboxes), self.anchors.unsqueeze(0), xywh=True, dim=1) \* self.strides

#### 3.1.5 Segment检测头

class Segment(Detect):  
 *"""YOLOv8 Segment head for segmentation models."""* def \_\_init\_\_(self, nc=80, nm=32, npr=256, ch=()):  
 *"""Initialize the YOLO model attributes such as the number of masks, prototypes, and the convolution layers."""* super().\_\_init\_\_(nc, ch)  
 self.nm = nm *# number of masks* self.npr = npr *# number of protos* self.proto = Proto(ch[0], self.npr, self.nm) *# protos* self.detect = Detect.forward  
  
 c4 = max(ch[0] // 4, self.nm)  
 self.cv4 = nn.ModuleList(nn.Sequential(Conv(x, c4, 3), Conv(c4, c4, 3), nn.Conv2d(c4, self.nm, 1)) for x in ch)  
  
 def forward(self, x):  
 *"""Return model outputs and mask coefficients if training, otherwise return outputs and mask coefficients."""* p = self.proto(x[0]) *# mask protos* bs = p.shape[0] *# batch size* mc = torch.cat([self.cv4[i](x[i]).view(bs, self.nm, -1) for i in range(self.nl)], 2) *# mask coefficients* x = self.detect(self, x)  
 if self.training:  
 return x, mc, p  
 return (torch.cat([x, mc], 1), p) if self.export else (torch.cat([x[0], mc], 1), (x[1], mc, p))

3.2 YOLOv8目标检测实现

#### 3.2.1 数据集

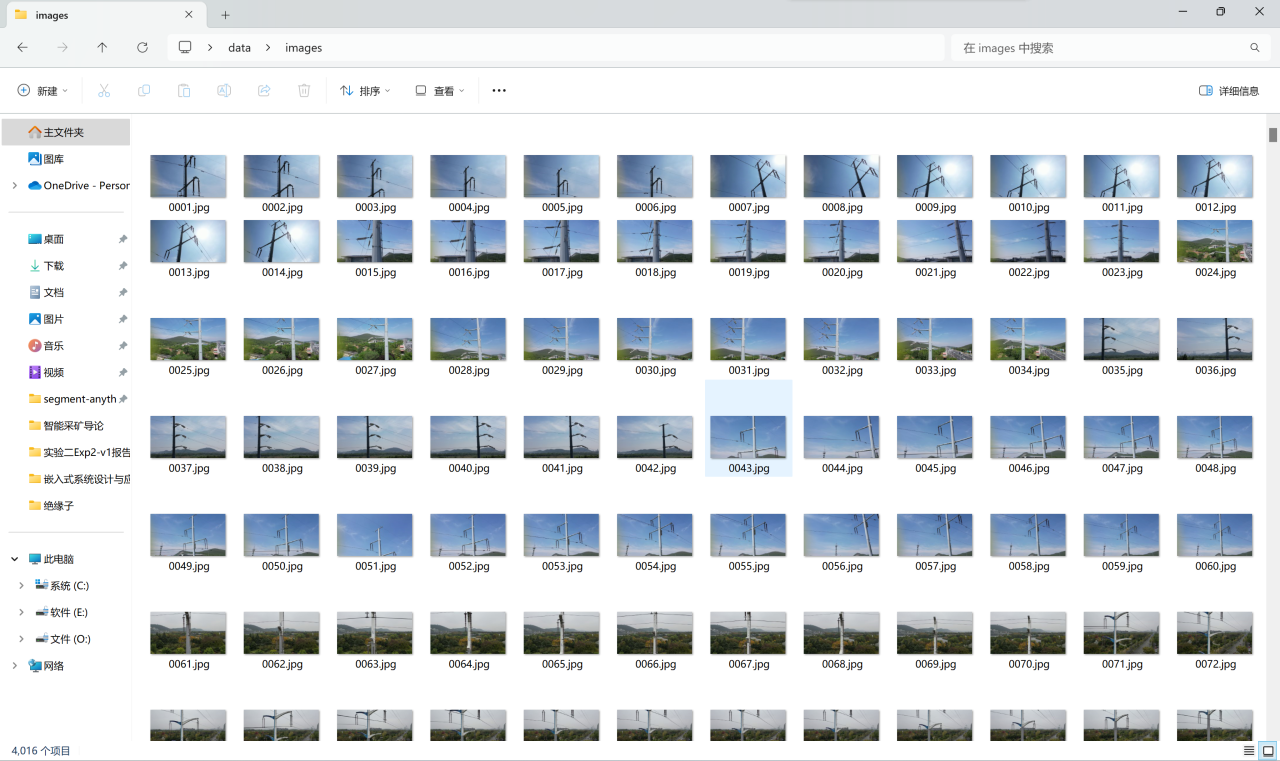


图 2 数据集图像

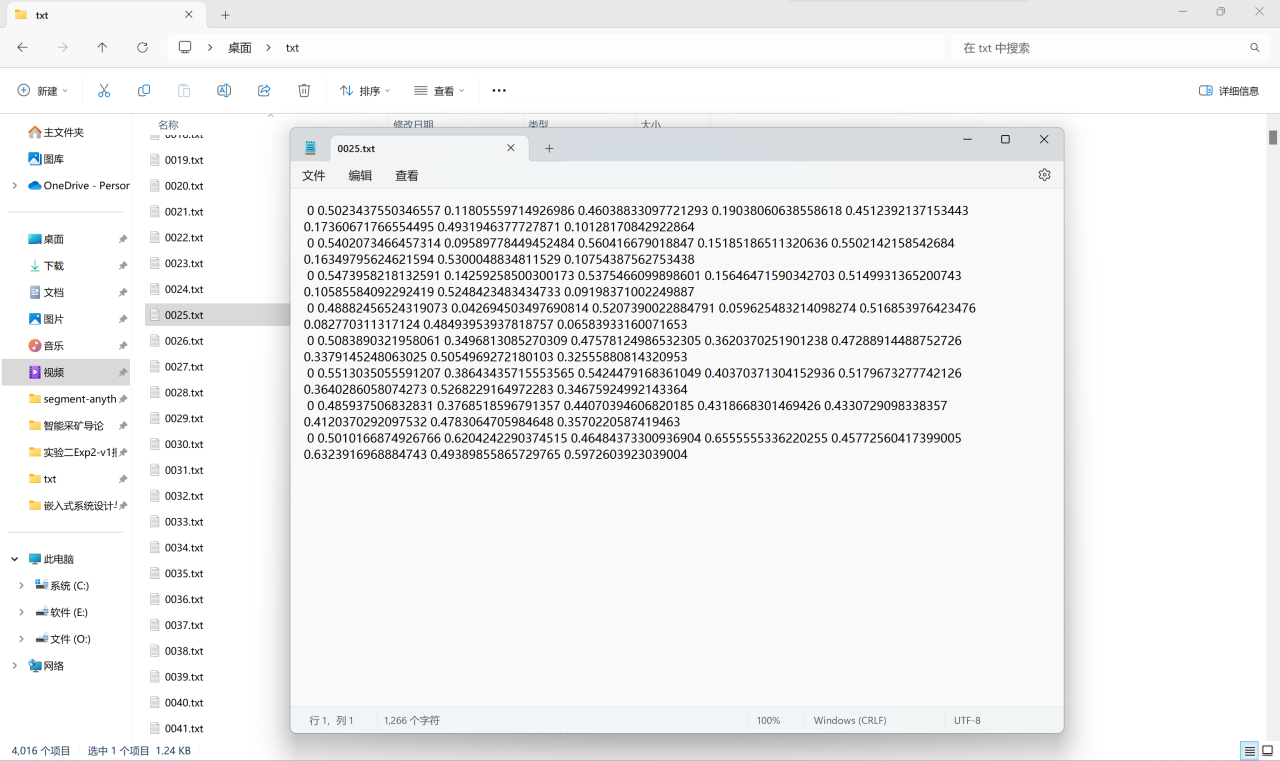


图 3 数据集标签

#### 3.2.2 网络构架

# Parameters

nc: 80 # number of classes

scales: # model compound scaling constants, i.e. 'model=yolov8n.yaml' will call yolov8.yaml with scale 'n'

# [depth, width, max\_channels]

n: [0.33, 0.25, 1024]

s: [0.33, 0.50, 1024]

m: [0.67, 0.75, 768]

l: [1.00, 1.00, 512]

x: [1.00, 1.25, 512]

backbone:

# [from, repeats, module, args]

- [-1, 1, Conv, [64, 3, 2]] # 0-P1/2

- [-1, 1, Conv, [128, 3, 2]] # 1-P2/4

- [-1, 3, C2f, [128, True]]

- [-1, 1, Conv, [256, 3, 2]] # 3-P3/8

- [-1, 6, C2f, [256, True]]

- [-1, 1, Conv, [512, 3, 2]] # 5-P4/16

- [-1, 6, C2f, [512, True]]

- [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32

- [-1, 3, C2f, [1024, True]]

- [-1, 1, SPPF, [1024, 5]] # 9

head:

- [-1, 1, nn.Upsample, [None, 2, 'nearest']]

- [[-1, 6], 1, Concat, [1]] # cat backbone P4

- [-1, 3, C2f, [512]] # 12

- [-1, 1, nn.Upsample, [None, 2, 'nearest']]

- [[-1, 4], 1, Concat, [1]] # cat backbone P3

- [-1, 3, C2f, [256]] # 15 (P3/8-small)

- [-1, 1, Conv, [256, 3, 2]]

- [[-1, 12], 1, Concat, [1]] # cat head P4

- [-1, 3, C2f, [512]] # 18 (P4/16-medium)

- [-1, 1, Conv, [512, 3, 2]]

- [[-1, 9], 1, Concat, [1]] # cat head P5

- [-1, 3, C2f, [1024]] # 21 (P5/32-large)

- [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)

#### 3.2.3 优化器和损失函数

1. 优化器

def build\_optimizer(self, model, name="auto", lr=0.001, momentum=0.9, decay=1e-5, iterations=1e5):  
 g = [], [], [] *# optimizer parameter groups* bn = tuple(v for k, v in nn.\_\_dict\_\_.items() if "Norm" in k) *# normalization layers, i.e. BatchNorm2d()* if name == "auto":  
 LOGGER.info(  
 f"{colorstr('optimizer:')} 'optimizer=auto' found, "  
 f"ignoring 'lr0={self.args.lr0}' and 'momentum={self.args.momentum}' and "  
 f"determining best 'optimizer', 'lr0' and 'momentum' automatically... "  
 )  
 nc = getattr(model, "nc", 10) *# number of classes* lr\_fit = round(0.002 \* 5 / (4 + nc), 6) *# lr0 fit equation to 6 decimal places* name, lr, momentum = ("SGD", 0.01, 0.9) if iterations > 10000 else ("AdamW", lr\_fit, 0.9)  
 self.args.warmup\_bias\_lr = 0.0 *# no higher than 0.01 for Adam* for module\_name, module in model.named\_modules():  
 for param\_name, param in module.named\_parameters(recurse=False):  
 fullname = f"{module\_name}.{param\_name}" if module\_name else param\_name  
 if "bias" in fullname: *# bias (no decay)* g[2].append(param)  
 elif isinstance(module, bn): *# weight (no decay)* g[1].append(param)  
 else: *# weight (with decay)* g[0].append(param)  
  
 if name in ("Adam", "Adamax", "AdamW", "NAdam", "RAdam"):  
 optimizer = getattr(optim, name, optim.Adam)(g[2], lr=lr, betas=(momentum, 0.999), weight\_decay=0.0)  
 elif name == "RMSProp":  
 optimizer = optim.RMSprop(g[2], lr=lr, momentum=momentum)  
 elif name == "SGD":  
 optimizer = optim.SGD(g[2], lr=lr, momentum=momentum, nesterov=True)  
 else:  
 raise NotImplementedError(  
 f"Optimizer '{name}' not found in list of available optimizers "  
 f"[Adam, AdamW, NAdam, RAdam, RMSProp, SGD, auto]."  
 "To request support for addition optimizers please visit https://github.com/ultralytics/ultralytics."  
 )  
  
 optimizer.add\_param\_group({"params": g[0], "weight\_decay": decay}) *# add g0 with weight\_decay* optimizer.add\_param\_group({"params": g[1], "weight\_decay": 0.0}) *# add g1 (BatchNorm2d weights)* LOGGER.info(  
 f"{colorstr('optimizer:')} {type(optimizer).\_\_name\_\_}(lr={lr}, momentum={momentum}) with parameter groups "  
 f'{len(g[1])} weight(decay=0.0), {len(g[0])} weight(decay={decay}), {len(g[2])} bias(decay=0.0)'  
 )  
 return optimizer

1. 损失函数

class v8DetectionLoss:  
 *"""Criterion class for computing training losses."""* def \_\_init\_\_(self, model): *# model must be de-paralleled  
 """Initializes v8DetectionLoss with the model, defining model-related properties and BCE loss function."""* device = next(model.parameters()).device *# get model device* h = model.args *# hyperparameters* m = model.model[-1] *# Detect() module* self.bce = nn.BCEWithLogitsLoss(reduction="none")  
 self.hyp = h  
 self.stride = m.stride *# model strides* self.nc = m.nc *# number of classes* self.no = m.no  
 self.reg\_max = m.reg\_max  
 self.device = device  
  
 self.use\_dfl = m.reg\_max > 1  
  
 self.assigner = TaskAlignedAssigner(topk=10, num\_classes=self.nc, alpha=0.5, beta=6.0)  
 self.bbox\_loss = BboxLoss(m.reg\_max - 1, use\_dfl=self.use\_dfl).to(device)  
 self.proj = torch.arange(m.reg\_max, dtype=torch.float, device=device)  
  
 def preprocess(self, targets, batch\_size, scale\_tensor):  
 *"""Preprocesses the target counts and matches with the input batch size to output a tensor."""* if targets.shape[0] == 0:  
 out = torch.zeros(batch\_size, 0, 5, device=self.device)  
 else:  
 i = targets[:, 0] *# image index* \_, counts = i.unique(return\_counts=True)  
 counts = counts.to(dtype=torch.int32)  
 out = torch.zeros(batch\_size, counts.max(), 5, device=self.device)  
 for j in range(batch\_size):  
 matches = i == j  
 n = matches.sum()  
 if n:  
 out[j, :n] = targets[matches, 1:]  
 out[..., 1:5] = xywh2xyxy(out[..., 1:5].mul\_(scale\_tensor))  
 return out  
  
 def bbox\_decode(self, anchor\_points, pred\_dist):  
 *"""Decode predicted object bounding box coordinates from anchor points and distribution."""* if self.use\_dfl:  
 b, a, c = pred\_dist.shape *# batch, anchors, channels* pred\_dist = pred\_dist.view(b, a, 4, c // 4).softmax(3).matmul(self.proj.type(pred\_dist.dtype))  
 *# pred\_dist = pred\_dist.view(b, a, c // 4, 4).transpose(2,3).softmax(3).matmul(self.proj.type(pred\_dist.dtype))  
 # pred\_dist = (pred\_dist.view(b, a, c // 4, 4).softmax(2) \* self.proj.type(pred\_dist.dtype).view(1, 1, -1, 1)).sum(2)* return dist2bbox(pred\_dist, anchor\_points, xywh=False)  
  
 def \_\_call\_\_(self, preds, batch):  
 *"""Calculate the sum of the loss for box, cls and dfl multiplied by batch size."""* loss = torch.zeros(3, device=self.device) *# box, cls, dfl* feats = preds[1] if isinstance(preds, tuple) else preds  
 pred\_distri, pred\_scores = torch.cat([xi.view(feats[0].shape[0], self.no, -1) for xi in feats], 2).split(  
 (self.reg\_max \* 4, self.nc), 1  
 )  
  
 pred\_scores = pred\_scores.permute(0, 2, 1).contiguous()  
 pred\_distri = pred\_distri.permute(0, 2, 1).contiguous()  
  
 dtype = pred\_scores.dtype  
 batch\_size = pred\_scores.shape[0]  
 imgsz = torch.tensor(feats[0].shape[2:], device=self.device, dtype=dtype) \* self.stride[0] *# image size (h,w)* anchor\_points, stride\_tensor = make\_anchors(feats, self.stride, 0.5)  
  
 *# Targets* targets = torch.cat((batch["batch\_idx"].view(-1, 1), batch["cls"].view(-1, 1), batch["bboxes"]), 1)  
 targets = self.preprocess(targets.to(self.device), batch\_size, scale\_tensor=imgsz[[1, 0, 1, 0]])  
 gt\_labels, gt\_bboxes = targets.split((1, 4), 2) *# cls, xyxy* mask\_gt = gt\_bboxes.sum(2, keepdim=True).gt\_(0)  
  
 *# Pboxes* pred\_bboxes = self.bbox\_decode(anchor\_points, pred\_distri) *# xyxy, (b, h\*w, 4)* \_, target\_bboxes, target\_scores, fg\_mask, \_ = self.assigner(  
 pred\_scores.detach().sigmoid(),  
 (pred\_bboxes.detach() \* stride\_tensor).type(gt\_bboxes.dtype),  
 anchor\_points \* stride\_tensor,  
 gt\_labels,  
 gt\_bboxes,  
 mask\_gt,  
 )  
  
 target\_scores\_sum = max(target\_scores.sum(), 1)  
  
 *# Cls loss  
 # loss[1] = self.varifocal\_loss(pred\_scores, target\_scores, target\_labels) / target\_scores\_sum # VFL way* loss[1] = self.bce(pred\_scores, target\_scores.to(dtype)).sum() / target\_scores\_sum *# BCE  
  
 # Bbox loss* if fg\_mask.sum():  
 target\_bboxes /= stride\_tensor  
 loss[0], loss[2] = self.bbox\_loss(  
 pred\_distri, pred\_bboxes, anchor\_points, target\_bboxes, target\_scores, target\_scores\_sum, fg\_mask  
 )  
  
 loss[0] \*= self.hyp.box *# box gain* loss[1] \*= self.hyp.cls *# cls gain* loss[2] \*= self.hyp.dfl *# dfl gain* return loss.sum() \* batch\_size, loss.detach() *# loss(box, cls, dfl)*

#### 3.2.4 训练和结果

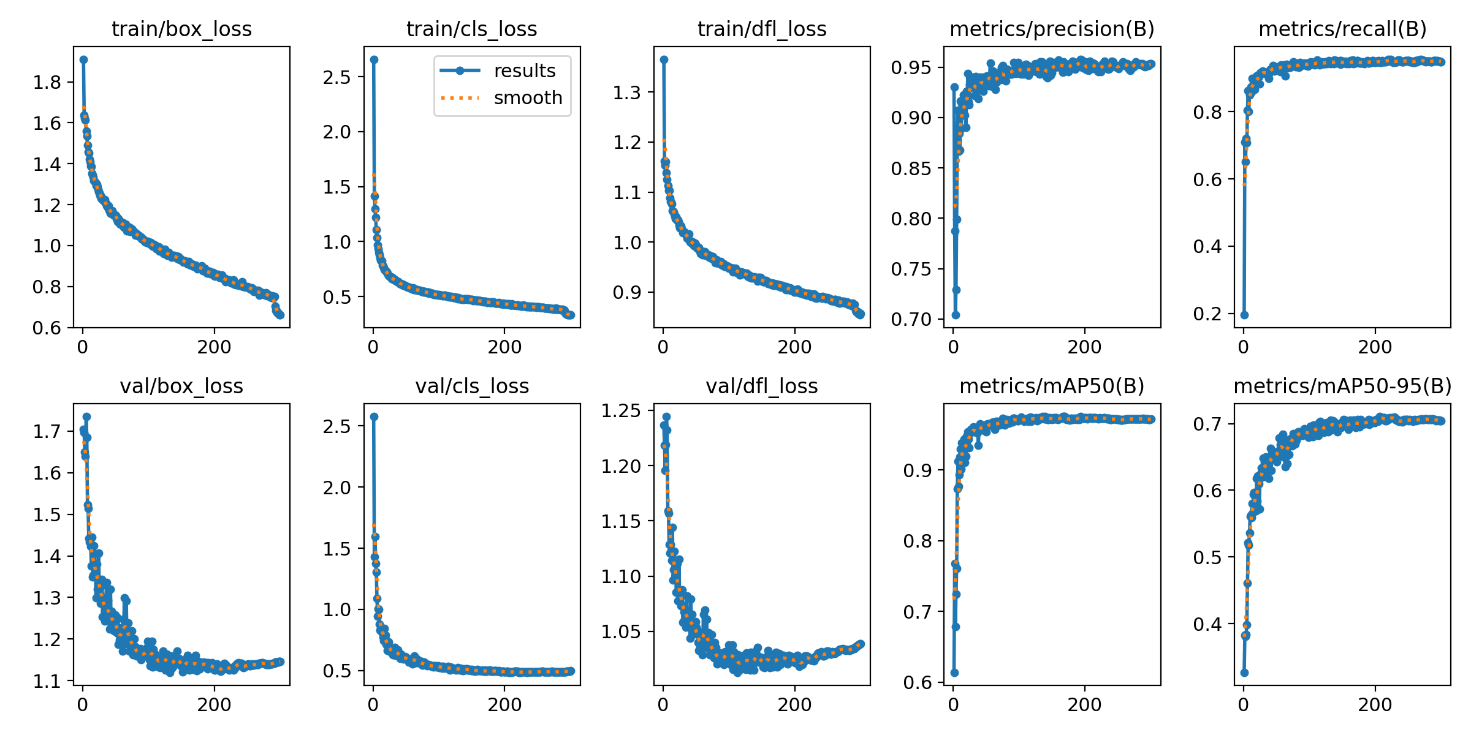


图 4 YOLOv8n训练结果

表中记录了yolov8n在300轮训练过程中的的边界框损失（box\_loss）、类别损失（cls\_loss）以及特征点损失（dfl\_loss）。同时记录了每一轮的准确率（precision）和召回率（recall）。可以看到，随着训练轮次的增加，三种损失在不断降低，模型的准确率和召回率在不断上升。



图 5 YOLOv8n预测结果

3.3 YOLOv8语义分割实现

#### 3.3.1 数据集

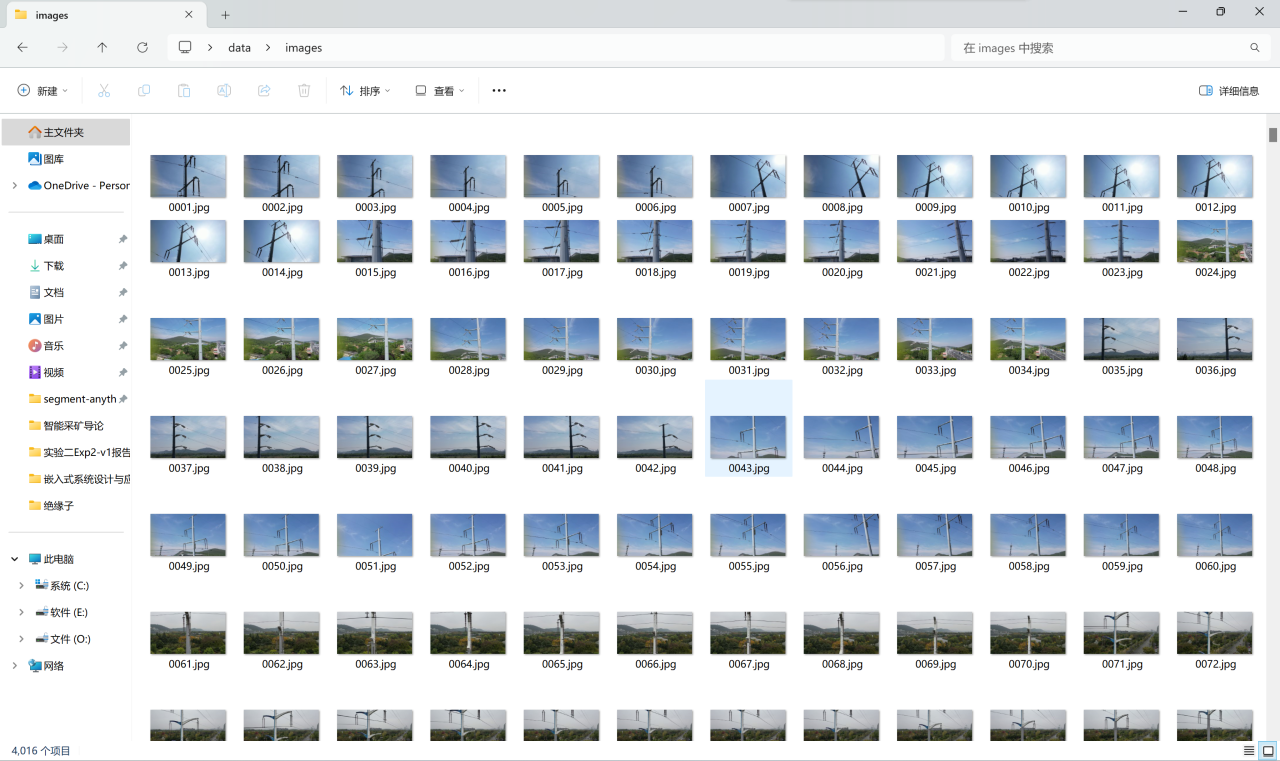


图 6 数据集图像

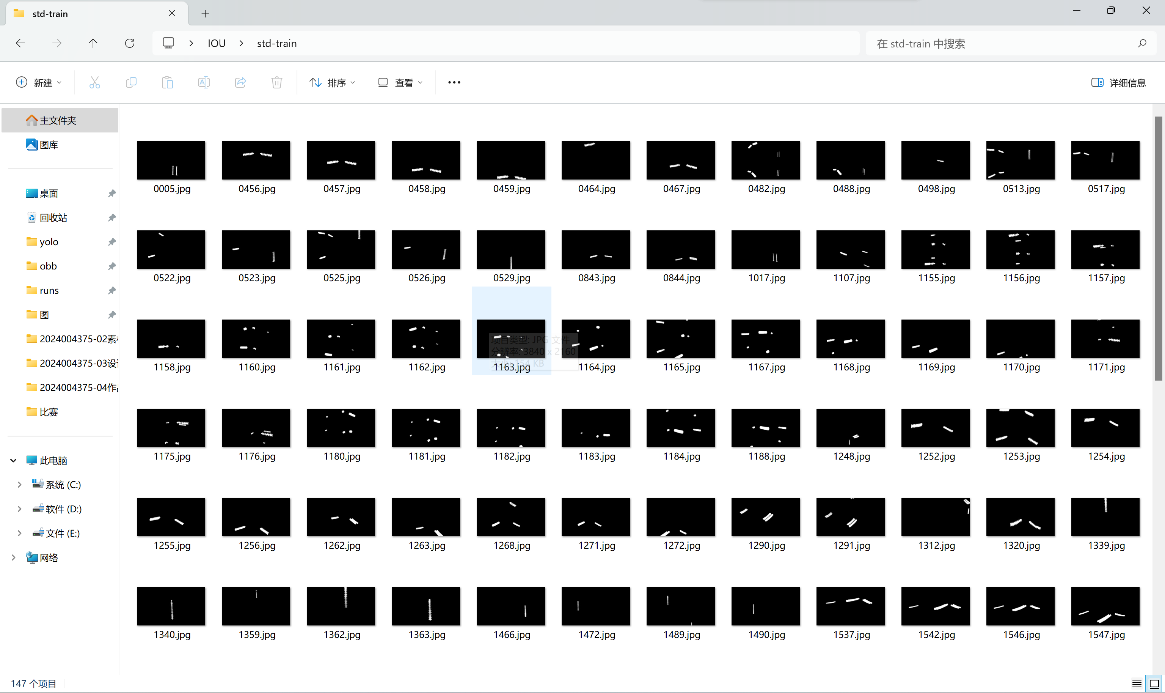


图 7 数据集掩膜

#### 3.3.2 网络架构

# Parameters

nc: 80 # number of classes

scales: # model compound scaling constants, i.e. 'model=yolov8n-seg.yaml' will call yolov8-seg.yaml with scale 'n'

# [depth, width, max\_channels]

n: [0.33, 0.25, 1024]

s: [0.33, 0.50, 1024]

m: [0.67, 0.75, 768]

l: [1.00, 1.00, 512]

x: [1.00, 1.25, 512]

backbone:

# [from, repeats, module, args]

- [-1, 1, Conv, [64, 3, 2]] # 0-P1/2

- [-1, 1, Conv, [128, 3, 2]] # 1-P2/4

- [-1, 3, C2f, [128, True]]

- [-1, 1, Conv, [256, 3, 2]] # 3-P3/8

- [-1, 6, C2f, [256, True]]

- [-1, 1, Conv, [512, 3, 2]] # 5-P4/16

- [-1, 6, C2f, [512, True]]

- [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32

- [-1, 3, C2f, [1024, True]]

- [-1, 1, SPPF, [1024, 5]] # 9

head:

- [-1, 1, nn.Upsample, [None, 2, 'nearest']]

- [[-1, 6], 1, Concat, [1]] # cat backbone P4

- [-1, 3, C2f, [512]] # 12

- [-1, 1, nn.Upsample, [None, 2, 'nearest']]

- [[-1, 4], 1, Concat, [1]] # cat backbone P3

- [-1, 3, C2f, [256]] # 15 (P3/8-small)

- [-1, 1, Conv, [256, 3, 2]]

- [[-1, 12], 1, Concat, [1]] # cat head P4

- [-1, 3, C2f, [512]] # 18 (P4/16-medium)

- [-1, 1, Conv, [512, 3, 2]]

- [[-1, 9], 1, Concat, [1]] # cat head P5

- [-1, 3, C2f, [1024]] # 21 (P5/32-large)

- [[15, 18, 21], 1, Segment, [nc, 32, 256]] # Segment(P3, P4, P5)

#### 3.3.3 优化器和损失函数

1. 优化器

def build\_optimizer(self, model, name="auto", lr=0.001, momentum=0.9, decay=1e-5, iterations=1e5):  
 g = [], [], [] *# optimizer parameter groups* bn = tuple(v for k, v in nn.\_\_dict\_\_.items() if "Norm" in k) *# normalization layers, i.e. BatchNorm2d()* if name == "auto":  
 LOGGER.info(  
 f"{colorstr('optimizer:')} 'optimizer=auto' found, "  
 f"ignoring 'lr0={self.args.lr0}' and 'momentum={self.args.momentum}' and "  
 f"determining best 'optimizer', 'lr0' and 'momentum' automatically... "  
 )  
 nc = getattr(model, "nc", 10) *# number of classes* lr\_fit = round(0.002 \* 5 / (4 + nc), 6) *# lr0 fit equation to 6 decimal places* name, lr, momentum = ("SGD", 0.01, 0.9) if iterations > 10000 else ("AdamW", lr\_fit, 0.9)  
 self.args.warmup\_bias\_lr = 0.0 *# no higher than 0.01 for Adam* for module\_name, module in model.named\_modules():  
 for param\_name, param in module.named\_parameters(recurse=False):  
 fullname = f"{module\_name}.{param\_name}" if module\_name else param\_name  
 if "bias" in fullname: *# bias (no decay)* g[2].append(param)  
 elif isinstance(module, bn): *# weight (no decay)* g[1].append(param)  
 else: *# weight (with decay)* g[0].append(param)  
  
 if name in ("Adam", "Adamax", "AdamW", "NAdam", "RAdam"):  
 optimizer = getattr(optim, name, optim.Adam)(g[2], lr=lr, betas=(momentum, 0.999), weight\_decay=0.0)  
 elif name == "RMSProp":  
 optimizer = optim.RMSprop(g[2], lr=lr, momentum=momentum)  
 elif name == "SGD":  
 optimizer = optim.SGD(g[2], lr=lr, momentum=momentum, nesterov=True)  
 else:  
 raise NotImplementedError(  
 f"Optimizer '{name}' not found in list of available optimizers "  
 f"[Adam, AdamW, NAdam, RAdam, RMSProp, SGD, auto]."  
 "To request support for addition optimizers please visit https://github.com/ultralytics/ultralytics."  
 )  
  
 optimizer.add\_param\_group({"params": g[0], "weight\_decay": decay}) *# add g0 with weight\_decay* optimizer.add\_param\_group({"params": g[1], "weight\_decay": 0.0}) *# add g1 (BatchNorm2d weights)* LOGGER.info(  
 f"{colorstr('optimizer:')} {type(optimizer).\_\_name\_\_}(lr={lr}, momentum={momentum}) with parameter groups "  
 f'{len(g[1])} weight(decay=0.0), {len(g[0])} weight(decay={decay}), {len(g[2])} bias(decay=0.0)'  
 )  
 return optimizer

1. 损失函数

class v8SegmentationLoss(v8DetectionLoss):  
 *"""Criterion class for computing training losses."""* def \_\_init\_\_(self, model): *# model must be de-paralleled  
 """Initializes the v8SegmentationLoss class, taking a de-paralleled model as argument."""* super().\_\_init\_\_(model)  
 self.overlap = model.args.overlap\_mask  
  
 def \_\_call\_\_(self, preds, batch):  
 *"""Calculate and return the loss for the YOLO model."""* loss = torch.zeros(4, device=self.device) *# box, cls, dfl* feats, pred\_masks, proto = preds if len(preds) == 3 else preds[1]  
 batch\_size, \_, mask\_h, mask\_w = proto.shape *# batch size, number of masks, mask height, mask width* pred\_distri, pred\_scores = torch.cat([xi.view(feats[0].shape[0], self.no, -1) for xi in feats], 2).split(  
 (self.reg\_max \* 4, self.nc), 1  
 )  
  
 *# B, grids, ..* pred\_scores = pred\_scores.permute(0, 2, 1).contiguous()  
 pred\_distri = pred\_distri.permute(0, 2, 1).contiguous()  
 pred\_masks = pred\_masks.permute(0, 2, 1).contiguous()  
  
 dtype = pred\_scores.dtype  
 imgsz = torch.tensor(feats[0].shape[2:], device=self.device, dtype=dtype) \* self.stride[0] *# image size (h,w)* anchor\_points, stride\_tensor = make\_anchors(feats, self.stride, 0.5)  
  
 *# Targets* try:  
 batch\_idx = batch["batch\_idx"].view(-1, 1)  
 targets = torch.cat((batch\_idx, batch["cls"].view(-1, 1), batch["bboxes"]), 1)  
 targets = self.preprocess(targets.to(self.device), batch\_size, scale\_tensor=imgsz[[1, 0, 1, 0]])  
 gt\_labels, gt\_bboxes = targets.split((1, 4), 2) *# cls, xyxy* mask\_gt = gt\_bboxes.sum(2, keepdim=True).gt\_(0)  
 except RuntimeError as e:  
 raise TypeError(  
 "ERROR ❌ segment dataset incorrectly formatted or not a segment dataset.\n"  
 "This error can occur when incorrectly training a 'segment' model on a 'detect' dataset, "  
 "i.e. 'yolo train model=yolov8n-seg.pt data=coco8.yaml'.\nVerify your dataset is a "  
 "correctly formatted 'segment' dataset using 'data=coco8-seg.yaml' "  
 "as an example.\nSee https://docs.ultralytics.com/datasets/segment/ for help."  
 ) from e  
  
 *# Pboxes* pred\_bboxes = self.bbox\_decode(anchor\_points, pred\_distri) *# xyxy, (b, h\*w, 4)* \_, target\_bboxes, target\_scores, fg\_mask, target\_gt\_idx = self.assigner(  
 pred\_scores.detach().sigmoid(),  
 (pred\_bboxes.detach() \* stride\_tensor).type(gt\_bboxes.dtype),  
 anchor\_points \* stride\_tensor,  
 gt\_labels,  
 gt\_bboxes,  
 mask\_gt,  
 )  
  
 target\_scores\_sum = max(target\_scores.sum(), 1)  
  
 *# Cls loss  
 # loss[1] = self.varifocal\_loss(pred\_scores, target\_scores, target\_labels) / target\_scores\_sum # VFL way* loss[2] = self.bce(pred\_scores, target\_scores.to(dtype)).sum() / target\_scores\_sum *# BCE* if fg\_mask.sum():  
 *# Bbox loss* loss[0], loss[3] = self.bbox\_loss(  
 pred\_distri,  
 pred\_bboxes,  
 anchor\_points,  
 target\_bboxes / stride\_tensor,  
 target\_scores,  
 target\_scores\_sum,  
 fg\_mask,  
 )  
 *# Masks loss* masks = batch["masks"].to(self.device).float()  
 if tuple(masks.shape[-2:]) != (mask\_h, mask\_w): *# downsample* masks = F.interpolate(masks[None], (mask\_h, mask\_w), mode="nearest")[0]  
  
 loss[1] = self.calculate\_segmentation\_loss(  
 fg\_mask, masks, target\_gt\_idx, target\_bboxes, batch\_idx, proto, pred\_masks, imgsz, self.overlap  
 )  
  
 *# WARNING: lines below prevent Multi-GPU DDP 'unused gradient' PyTorch errors, do not remove* else:  
 loss[1] += (proto \* 0).sum() + (pred\_masks \* 0).sum() *# inf sums may lead to nan loss* loss[0] \*= self.hyp.box *# box gain* loss[1] \*= self.hyp.box *# seg gain* loss[2] \*= self.hyp.cls *# cls gain* loss[3] \*= self.hyp.dfl *# dfl gain* return loss.sum() \* batch\_size, loss.detach() *# loss(box, cls, dfl)* @staticmethod  
 def single\_mask\_loss(  
 gt\_mask: torch.Tensor, pred: torch.Tensor, proto: torch.Tensor, xyxy: torch.Tensor, area: torch.Tensor  
 ) -> torch.Tensor:  
 pred\_mask = torch.einsum("in,nhw->ihw", pred, proto) *# (n, 32) @ (32, 80, 80) -> (n, 80, 80)* loss = F.binary\_cross\_entropy\_with\_logits(pred\_mask, gt\_mask, reduction="none")  
 return (crop\_mask(loss, xyxy).mean(dim=(1, 2)) / area).sum()  
  
 def calculate\_segmentation\_loss(  
 self,  
 fg\_mask: torch.Tensor,  
 masks: torch.Tensor,  
 target\_gt\_idx: torch.Tensor,  
 target\_bboxes: torch.Tensor,  
 batch\_idx: torch.Tensor,  
 proto: torch.Tensor,  
 pred\_masks: torch.Tensor,  
 imgsz: torch.Tensor,  
 overlap: bool,  
 ) -> torch.Tensor:  
 \_, \_, mask\_h, mask\_w = proto.shape  
 loss = 0  
  
 *# Normalize to 0-1* target\_bboxes\_normalized = target\_bboxes / imgsz[[1, 0, 1, 0]]  
  
 *# Areas of target bboxes* marea = xyxy2xywh(target\_bboxes\_normalized)[..., 2:].prod(2)  
  
 *# Normalize to mask size* mxyxy = target\_bboxes\_normalized \* torch.tensor([mask\_w, mask\_h, mask\_w, mask\_h], device=proto.device)  
  
 for i, single\_i in enumerate(zip(fg\_mask, target\_gt\_idx, pred\_masks, proto, mxyxy, marea, masks)):  
 fg\_mask\_i, target\_gt\_idx\_i, pred\_masks\_i, proto\_i, mxyxy\_i, marea\_i, masks\_i = single\_i  
 if fg\_mask\_i.any():  
 mask\_idx = target\_gt\_idx\_i[fg\_mask\_i]  
 if overlap:  
 gt\_mask = masks\_i == (mask\_idx + 1).view(-1, 1, 1)  
 gt\_mask = gt\_mask.float()  
 else:  
 gt\_mask = masks[batch\_idx.view(-1) == i][mask\_idx]  
  
 loss += self.single\_mask\_loss(  
 gt\_mask, pred\_masks\_i[fg\_mask\_i], proto\_i, mxyxy\_i[fg\_mask\_i], marea\_i[fg\_mask\_i]  
 )  
  
 *# WARNING: lines below prevents Multi-GPU DDP 'unused gradient' PyTorch errors, do not remove* else:  
 loss += (proto \* 0).sum() + (pred\_masks \* 0).sum() *# inf sums may lead to nan loss* return loss / fg\_mask.sum()

#### 3.3.4 训练和结果

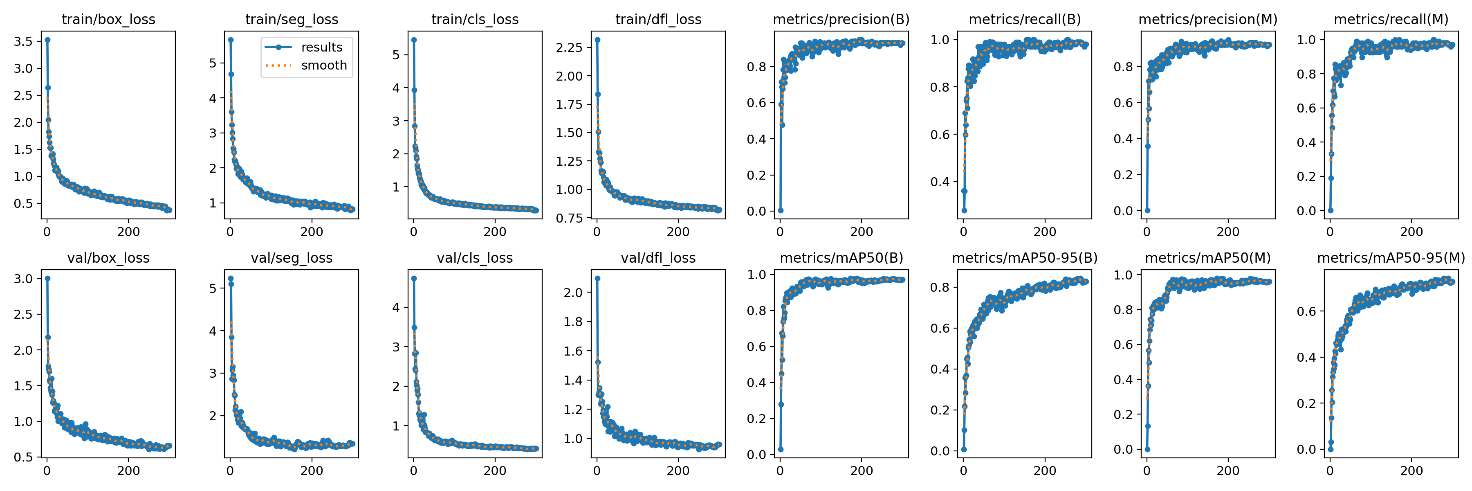


图 8 YOLOv8n-seg训练结果

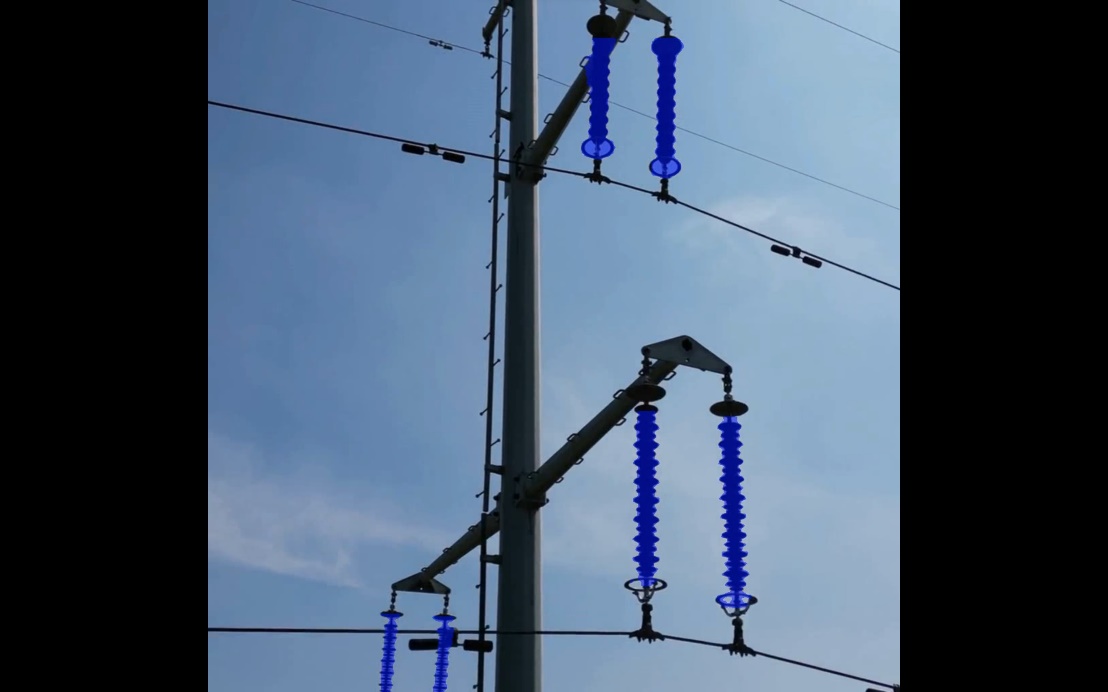


图 9 YOLOv8n-seg预测结果

表中记录了yolov8n-seg在300轮训练过程中的的边界框损失（box\_loss）、语义分割损失（seg\_loss）、类别损失（cls\_loss）以及特征点损失（dfl\_loss）。同时记录了每一轮的准确率（precision）和召回率（recall）。可以看到，随着训练轮次的增加，四种损失在不断降低，模型的准确率和召回率在不断上升。