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

一、 实验目的

1. 掌握图像物体检测；
2. 掌握语义分割；

二、 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；

三、 实验的具体实现

3.1 基本算子定义

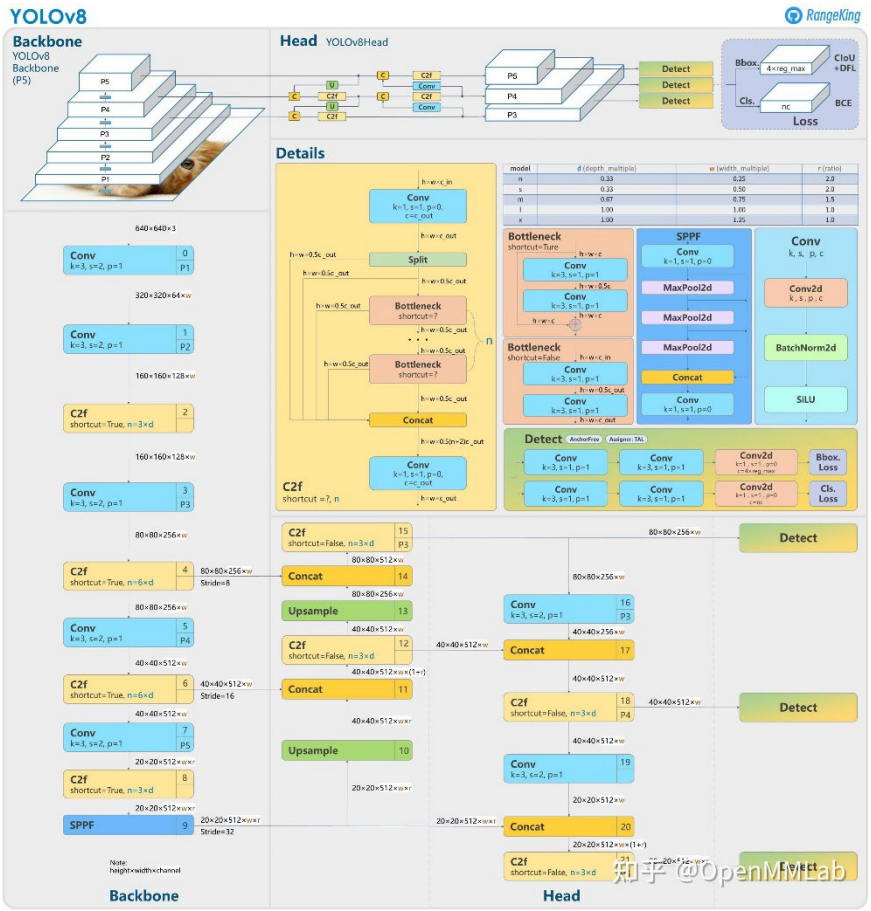

图 1 YOLOv8 结构及其算子示意图

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 ten
sor."""
        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.r
```

```
eg_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 probabiliti
es."""
        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(sel
f.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

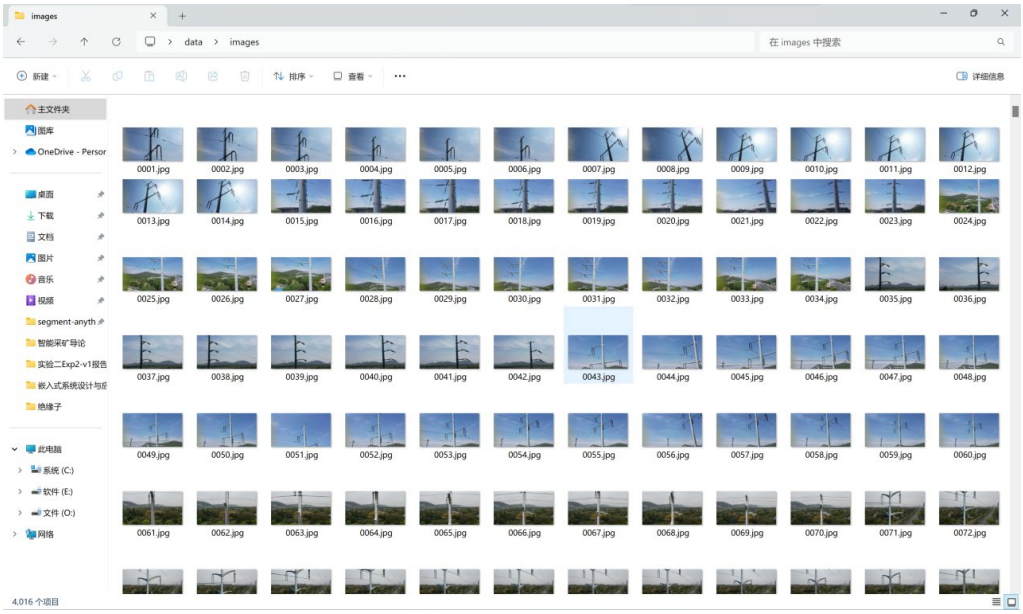

图 2 数据集图像

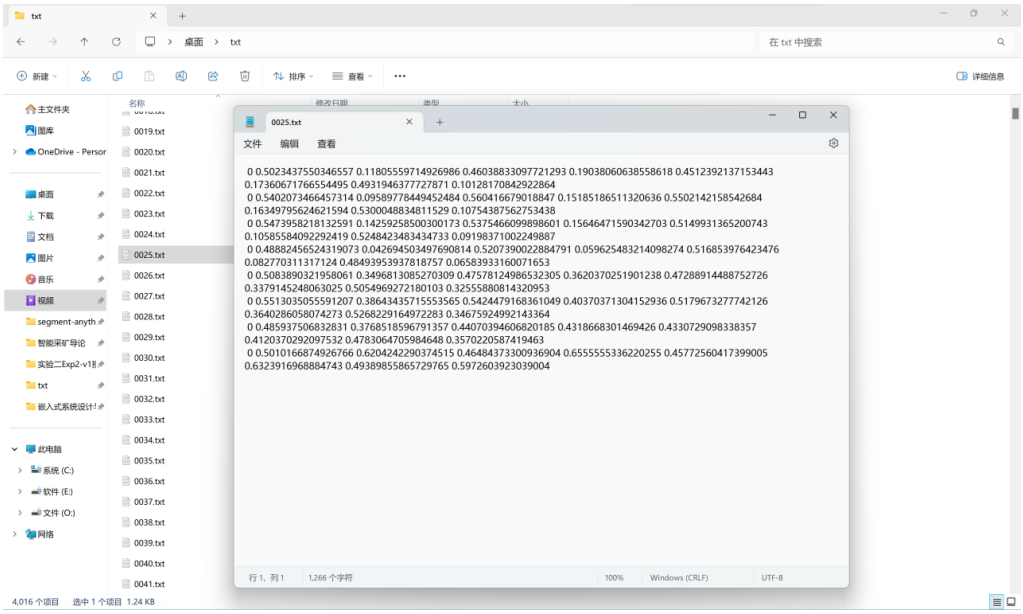

图 3 数据集标签

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 l
ayers, 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}' a
nd "
           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 pla
ces
       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 w
ith weight_decay
    optimizer.add_param_group({"params": g[1], "weight_decay": 0.0})  # add g1 (Ba
tchNorm2d 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
```

2) 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 prop
erties 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.t
ype(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 s
ize."""
        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_tenso
r=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_sc
ores_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_scor
es, 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

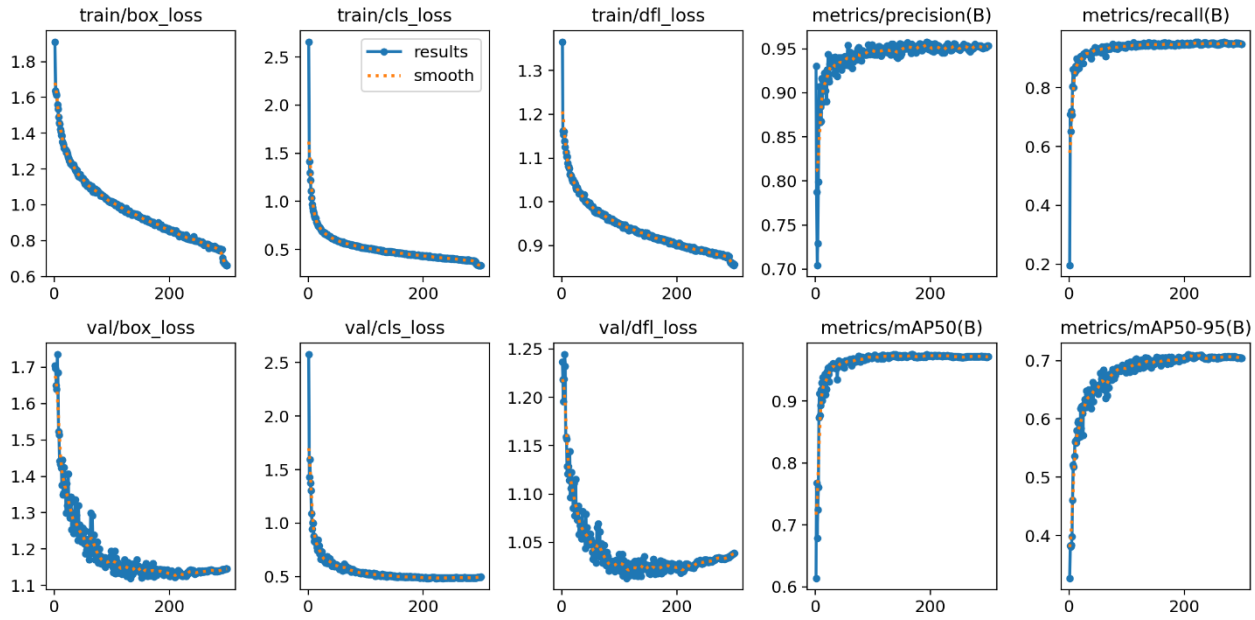

图 4 YOLOv8n 训练结果

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

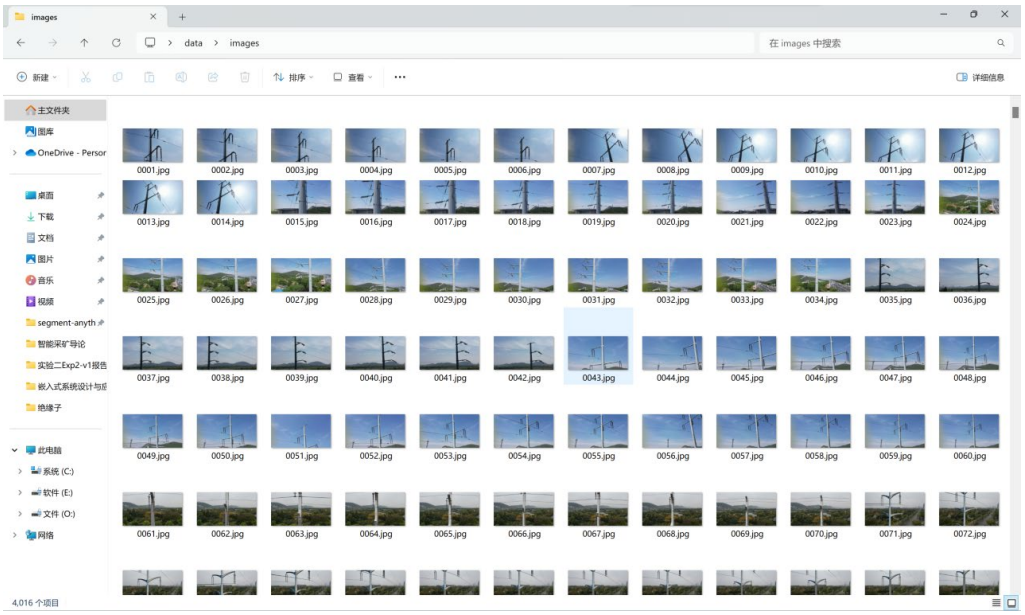

图 6 数据集图像

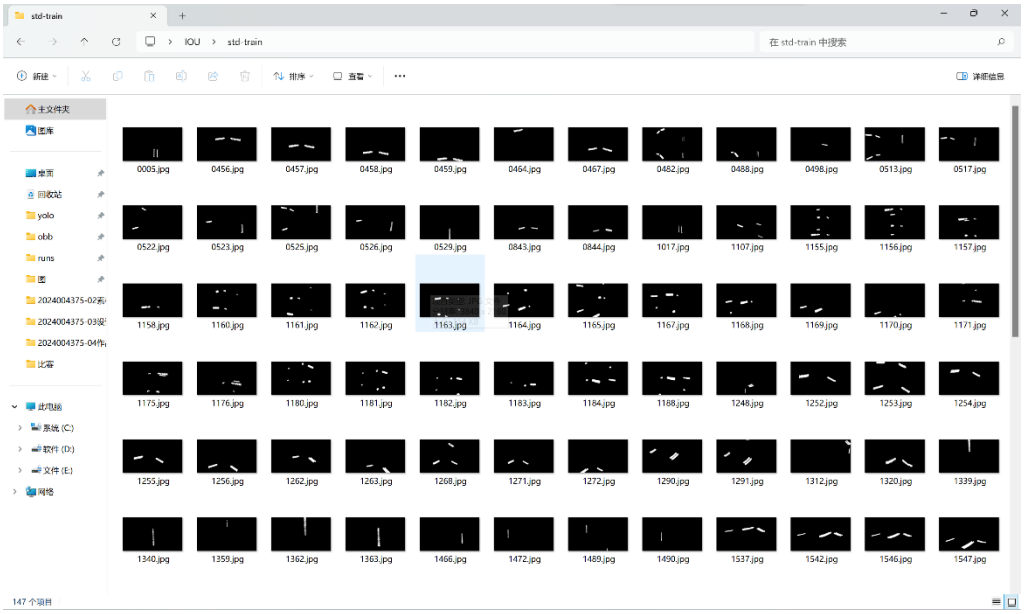

图 7 数据集掩膜

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 yolo v8-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 l
ayers, 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}' a
```

```
nd "
            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 pla
ces
        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 w
ith weight_decay
    optimizer.add_param_group({"params": g[1], "weight_decay": 0.0})  # add g1 (Ba
tchNorm2d 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
```

2) 损失函数

```
class v8SegmentationLoss(v8DetectionLoss):
    """Criterion class for computing training losses."""

    def __init__(self, model):  # model must be de-parallelized
        """Initializes the v8SegmentationLoss class, taking a de-parallelized 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_t
ensor=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 dat
aset.\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 yo
ur 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_sc
ores_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, m
ask_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, ma
sks_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], m
area_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 le
ad to nan loss

        return loss / fg_mask.sum()
```

3.3.4 训练和结果

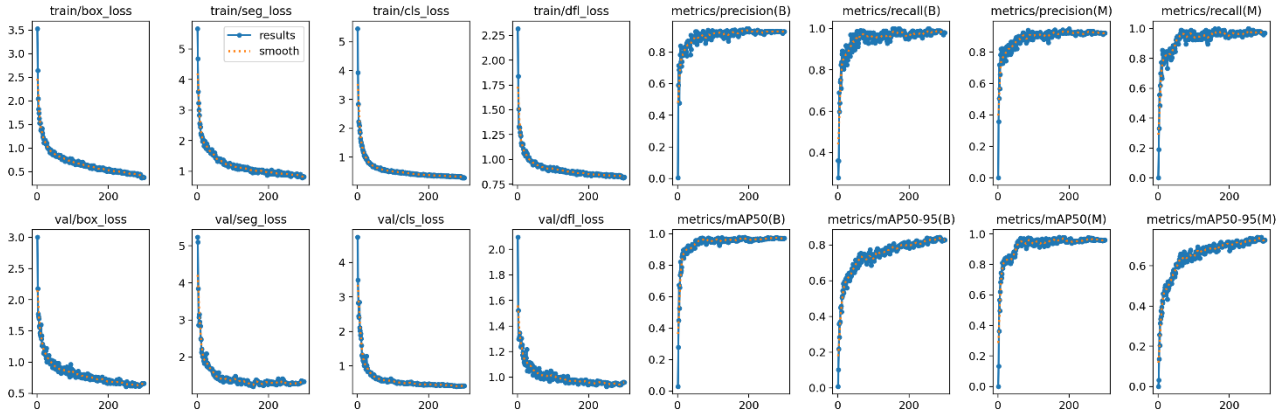

图 8 YOLOv8n-seg 训练结果

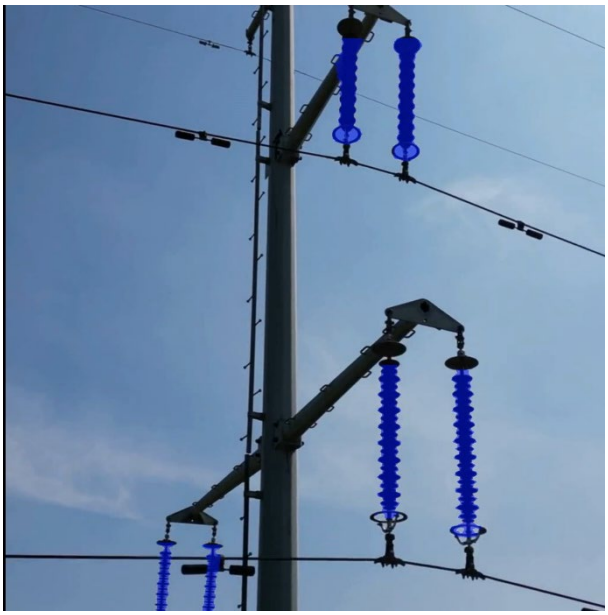

图 9 YOLOv8n-seg 预测结果

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。