# 

**第二组目标检测文档说明**

|  |  |  |
| --- | --- | --- |
| **学号** | **姓名** | **分工部分** |
| **2023214250** | **李诚辉** | **代码实现与结果测试** |
| **2023264124** | **刘源** | **数据集处理** |
| **2023214247** | **胡腾中** | **文档说明** |
| **2023214249** | **蒋靖** | **需求文档与测试文档** |

**目 录**

**[1 引言 1](#_Toc32087)**

**[2 YOLOv8架构设计 2](#_Toc27578)**

[2.1网络架构 2](#_Toc6289)

[2.2损失函数 3](#_Toc13436)

[2.3代码结构 3](#_Toc6682)

**[3 模型训练与优化 4](#_Toc6932)**

[3.1数据准备 4](#_Toc10458)

[3.2训练策略 5](#_Toc6873)

**[4 YOLOv8使用 5](#_Toc19094)**

[4.1代码获取 5](#_Toc25378)

[4.2环境准备 5](#_Toc14179)

[4.3在python中使用 5](#_Toc24928)

[4.4 helmet.yaml文件解读 8](#_Toc11737)

[4.4.1类型/模式参数 8](#_Toc27917)

[4.4.2训练参数 9](#_Toc30148)

[4.4.3验证/测试参数 10](#_Toc11290)

[4.4.4模型导出参数 10](#_Toc27995)

[4.4.5超参数 11](#_Toc15294)

**[5 结果展示 11](#_Toc7504)**

**[6优势与挑战 13](#_Toc18649)**

[6.1优势 13](#_Toc29233)

[6.2挑战 13](#_Toc23619)

**[7 结论 14](#_Toc28141)**

# 1 引言

施工建造行业已经成为我国国民经济的重要物质生产部门和支柱产业之一，同时也是最危险的工作领域之一。过去20年建筑安全事故率虽整体有所下降，但仍是各行业平均事故率的3倍左右,而安全帽在保护施工人员免受事故伤害方面起着至关重要的作用，所以必须采取监督措施，要求施工人员增强安全意识，及时佩戴安全帽。 现实情况中，目前对安全帽佩戴的检测主要依靠施工现场管理人员的检查和视屏监控的捕获、存储和回放等，繁琐且耗时。针对这一情况，迫切需要更加经济有效的方式使该过程自动化，减少头部伤残事故发生，然而，与施工现场出入口相比，施工作业面工况更复杂，图1左为某施工作业面的监控画面，图1右为各种情况下的安全帽样本，可看出施工作业面的各种模板、支架容易对目标造成遮挡，且人员移动、图像视角多变；同时，由于外界因素的影响如：天气多变、光照强度变化、背景对比度变化等，图像中目标尺寸不一、样本明暗程度不一，如图1左中安全帽目标在图像中所占比例范围为0.017%~3.9%，其中的小目标，由于分辨率低，容易受干扰导致漏检，因此，施工作业面安全帽检测更困难。

# IMG_256IMG_257

**图1 某施工作业面的监控画面与各种情况下的安全帽样本**

传统的安全帽目标检测方法包括利用梯度直方图特征和分类器支持向量机先定位到人体区域，进而采用头盔区域的颜色特征进行识别定位；利用安全帽的Haar（Haar-like）特征和霍夫变换进行检测。刘晓慧等人以工人的面部特征与肤色等综合信息将人脸区域进行定位，并通过头发与安全帽的色差、比较了不同的分类器，以此来判测工人是否佩戴了安全帽。李云波等则基于安全帽颜色信息统计分布的情况，结合视频流中人的动态目标及其色度值，进一步通过匹配安全帽颜色来判测安全帽佩戴的情况。上述方法都需要工人做工期间的正脸区域检测信息和反馈数据，并且考虑到施工作业面工况复杂，目标视角多变，这些传统的目标检测算法识别效果不理想。

**2 YOLOv8架构设计**

**2.1网络架构**

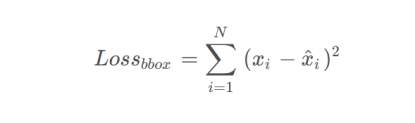
YOLOv8采用了改进的主干网络（Backbone）和检测头（Head），其中主干网络负责提取图像的特征，检测头负责预测目标的位置和类别。 YOLOv8的配置文件定义了模型的关键参数和结构，包括类别数、模型尺寸、骨干（backbone）和头部（head）结构。这些配置决定了模型的性能和复杂性。 Backbone主干网络是模型的基础，负责从输入图像中提取特征。这些特征是后续网络层进行目标检测的基础。 在YOLOv8中，主干网络采用了类似于CSPDarknet的结构。 Head头部网络是目标检测模型的决策部分，负责产生最终的检测结果。 Neck颈部网络位于主干网络和头部网络之间，它的作用是进行特征融合和增强。 与前代相比，YOLOv8在以下几个方面进行了优化： 深度可分离卷积：采用深度可分离卷积，减少计算量，提高计算效率。 特征金字塔网络（FPN）和路径聚合网络（PAN）：融合多尺度特征，增强检测效果。 新型激活函数：使用更高效的激活函数，如SiLU或Mish，提高模型的非线性表达能力。

## IMG_256

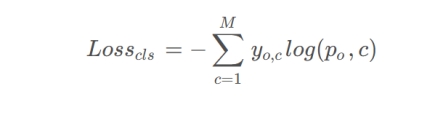
**图2 YOLOv8网络结构**

## 2.2损失函数

YOLOv8采用了改进的损失函数，结合了分类损失、边界框回归损失和置信度损失，确保了检测结果的准确性和鲁棒性。 Bbox Loss（边界框回归损失）用于计算预测边界框与真实边界框之间的差异。均方误差（MSE）是一个常用的损失函数，它在较大误差时赋予更高的惩罚，这有助于模型快速修正大的预测错误。因此，Bbox Loss计算预测与实际坐标之间的差异的平方和，其计算公式如下：



Cls Loss（分类损失）用于衡量模型预测的类别分布与真实标签之间的差异。交叉熵损失函数是分类任务中常用的一种损失函数，对于错误预测给出了很大的惩罚，尤其是在预测的概率和实际标签相差很大时。因此，Cls Loss帮助模型在分类问题中优化其预测，使预测概率分布尽可能接近真实的标签分布，



## 2.3代码结构

helmet\_detection

├──dataset #数据集

│ ├── images #图片

│ │ ├── train #训练集

│ │ │ ├── xxx1.jpg

│ │ │ │ ···

│ │ ├── val #验证集

│ │ │ ├── xxx1.jpg

│ │ │ │ ···

│ ├── labels #坐标标识

│ │ ├── train #训练集

│ │ │ ├── xxx1.txt

│ │ │ │ ···

│ │ ├── val #验证集

│ │ │ ├── xxx1.txt

│ │ │ │ ···

│ │ ├── train.cache #训练集图像转为txt

│ │ ├── val.cache #验证集图像转为txt

│ ├── label\_list.txt #类别

│ ├── test\_name.txt #测试集图像名称

│ ├── train.txt #训练数据集文件列表

│ └── valid.txt #测试数据集文件列表

├── models #所用模型合集

│ ├── yolov8n.py #YOLOv8模型

├── utils #数据清理和转换

│ ├── data\_clean.txt #数据清理

│ └── split\_data.py #划分数据集为训练集、测试集和验证集

│ └── xml2txt.py #把xml格式转换为txt格式

├── run #训练结构分析

├── test #测试环境可行性和分析结果

│ ├── test\_cuda.py #测试cuda可行

│ └── analytics.py #结果可视化

├── main.py #开始

├── requirements.txt #所需库

└── README.md #实现步骤说明

**3 模型训练与优化**

**3.1数据准备**

训练YOLOv8需要大量标注精确的图像数据。数据增强技术（如随机裁剪、颜色变换、旋转）用于提高模型的泛化能力。本文采用飞桨AI Studio平台所提供的安全帽检测数据集（5000个），针对安全帽检测数据集其中包含三类标记：“head”、“helmet”，在模型训练之前对数据集中大小不一的图片进行预处理。

**3.2训练策略**

YOLOv8采用了多阶段训练策略，包括预训练、微调和精细调整。预训练阶段使用大规模数据集（如ImageNet）进行初始化，微调阶段在目标数据集上进行训练，精细调整阶段进行超参数优化。将所有图片缩放到480\*480\*3（长\*宽\*通道）的尺寸，从而提高模型的识别率，同时考虑到输入值太大会导致在反向传播时候传递到输入层的梯度就会变得很大，最终使学习率很低，所以将图像的灰度进行归一化，作为初始的模型输入，并将训练集和验证集划分为4000、1000；其次，由训练集的可视化知施工作业面复杂背景下安全帽检测存在的遮挡，光照多变，且目标尺寸不一等问题，因此本文将训练集输入到YOLOv8基线模型中，使用平均精度（mAP）作为模型的评价指标，使用验证集得到的损失曲线图、mAP图、P-R图分析模型的缺点，为解决基线模型存在的问题，本文使用标签平滑正则化、更换主干网络、增加图片分辨率、数据增强方法对基线模型进行优化，训练改进的YOLOv8模型得到损失曲线图、mAP图、P-R图和对结果可视化，将其和基线模型做对比。最后为了检验训练模型是否具有数据库依赖性，本文还使用对口罩检测数据集的重新训练测试了该模型，得出改进的YOLOv8模型泛化能力。

# 4 YOLOv8使用

## 4.1代码获取

切换至指定目录，使用git克隆指令拷贝代码

|  |
| --- |
| git clone https://github.com/ultralytics/ultralytics |

## 4.2环境准备

切换至ultralytics目录，安装依赖包

|  |
| --- |
| cd ultralytics |

//或者使用conda，具体看read.me

|  |
| --- |
| pip install -r requirements.txt |

## 4.3在python中使用

（1）数据预处理

splitDataset.py主要用于将图像文件和相应的标签文件（文本文件）分配到训练集、验证集和测试集中。

import os, shutil

from sklearn.model\_selection import train\_test\_split

val\_size = 0.1

test\_size = 0.2

postfix = 'png'

imgpath = 'helmet\_raw/images'

txtpath = 'helmet\_raw/txt'

os.makedirs('images/train', exist\_ok=True)

os.makedirs('images/val', exist\_ok=True)

os.makedirs('images/test', exist\_ok=True)

os.makedirs('labels/train', exist\_ok=True)

os.makedirs('labels/val', exist\_ok=True)

os.makedirs('labels/test', exist\_ok=True)

listdir = [i for i in os.listdir(txtpath) if 'txt' in i and i != 'classes.txt']

train, test = train\_test\_split(listdir, test\_size=test\_size, shuffle=True, random\_state=0)

train, val = train\_test\_split(train, test\_size=val\_size, shuffle=True, random\_state=0)

for i in train:

shutil.copy('{}/{}.{}'.format(imgpath, i[:-4], postfix), 'images/train/{}.{}'.format(i[:-4], postfix))

shutil.copy('{}/{}'.format(txtpath, i), 'labels/train/{}'.format(i))

for i in val:

shutil.copy('{}/{}.{}'.format(imgpath, i[:-4], postfix), 'images/val/{}.{}'.format(i[:-4], postfix))

shutil.copy('{}/{}'.format(txtpath, i), 'labels/val/{}'.format(i))

for i in test:

shutil.copy('{}/{}.{}'.format(imgpath, i[:-4], postfix), 'images/test/{}.{}'.format(i[:-4], postfix))

shutil.copy('{}/{}'.format(txtpath, i), 'labels/test/{}'.format(i))

（2）使用xml2txt.py，将XML格式的标注文件转换为用于训练目标检测模型的文本格式

import xml.etree.ElementTree as ET

import os, cv2

import numpy as np

from os import listdir

from os.path import join

classes = []

def convert(size, box):

dw = 1. / (size[0])

dh = 1. / (size[1])

x = (box[0] + box[1]) / 2.0 - 1

y = (box[2] + box[3]) / 2.0 - 1

w = box[1] - box[0]

h = box[3] - box[2]

x = x \* dw

w = w \* dw

y = y \* dh

h = h \* dh

return (x, y, w, h)

def convert\_annotation(xmlpath, xmlname):

with open(xmlpath, "r", encoding='utf-8') as in\_file:

txtname = xmlname[:-4] + '.txt'

txtfile = os.path.join(txtpath, txtname)

tree = ET.parse(in\_file)

root = tree.getroot()

filename = root.find('filename')

img = cv2.imdecode(np.fromfile('{}/{}.{}'.format(imgpath, xmlname[:-4], postfix), np.uint8), cv2.IMREAD\_COLOR)

h, w = img.shape[:2]

res = []

for obj in root.iter('object'):

cls = obj.find('name').text

if cls not in classes:

classes.append(cls)

cls\_id = classes.index(cls)

xmlbox = obj.find('bndbox')

b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text), float(xmlbox.find('ymin').text),

float(xmlbox.find('ymax').text))

bb = convert((w, h), b)

res.append(str(cls\_id) + " " + " ".join([str(a) for a in bb]))

if len(res) != 0:

with open(txtfile, 'w+') as f:

f.write('\n'.join(res))

if \_\_name\_\_ == "\_\_main\_\_":

postfix = 'png'

imgpath = 'helmet\_raw/Images'

xmlpath = 'helmet\_raw/txt'

if not os.path.exists(txtpath):

os.makedirs(txtpath, exist\_ok=True)

list = os.listdir(xmlpath)

error\_file\_list = []

for i in range(0, len(list)):

try:

path = os.path.join(xmlpath, list[i])

if ('.xml' in path) or ('.XML' in path):

convert\_annotation(path, list[i])

print(f'file {list[i]} convert success.')

else:

print(f'file {list[i]} is not xml format.')

except Exception as e:

print(f'file {list[i]} convert error.')

print(f'error message:\n{e}')

error\_file\_list.append(list[i])

print(f'this file convert failure\n{error\_file\_list}')

print(f'Dataset Classes:{classes}')

(3)调用main.py

from ultralytics import YOLO

# 加载模型

# 从头开始构建新模型

model = YOLO("yolov8n.yaml")

# 加载预训练模型（建议用于训练）

model = YOLO("yolov8n.pt")

# 使用模型

## 训练模型

model.train(data="helmet.yaml", epochs=150)

# 在验证集上评估模型性能

metrics = model.val()

# 将模型导出为 ONNX 格式

success = model.export(format="onnx")

(4)修改 helmet.yaml文件中的训练路径

## 4.4 helmet.**yaml**文件解读

helmet.yaml配置文件用于设置Yolov8模型的训练和预测参数。

**4.4.1类型/模式参数**

task：detect 指定Yolov8的任务类型，默认为detect，您也可根据实际应用场景设置为segment、classify、pose等；

mode: train 指定Yolov8的运行模式，默认为train，您也可根据实际操作设置为val、predict、export、track、benchmark等。

**4.4.2训练参数**

model 模型文件的路径，可以是预训练的模型权重文件（如yolov8n.pt）或模型配置文件（如yolov8n.yaml）；

data 数据集配置文件的路径，指定用于训练的数据集配置文件（如coco128.yaml）；

epochs: 100 训练过程中整个数据集将被迭代多少次；

patience: 50 用于控制训练过程中的停止机制，当模型在一定数量的轮次内没有观察到适应性的改善时，训练将被提前停止。这个参数默认设置为50，即在训练过程中，如果模型在连续的50个轮次内没有发现适应性的改善，训练将被提前停止，并且不会再继续进行下去；

batch: 8 每个批次中的图像数量；

imgsz: 640 用于设置输入图像尺寸；

save: True 指定是否保存训练过程中的模型检查点（即模型的权重和参数）以及预测结果（训练模式下，设置save: True，训练过程中的模型检查点将被保存，以便在训练完成后进行模型的加载和继续训练；预测模型下，设置save: True将保存预测结果）；

save\_period: -1 用于指定训练过程中模型检查点的保存周期（当 save\_period 设置为大于等于 1 的整数时，模型检查点将每隔指定的epochs保存一次。例如，save\_period: 10 表示每隔 10 个epochs保存一次模型检查点；当 save\_period 设置为小于1的值（如 -1）时，模型检查点的保存功能将被禁用，不会在训练过程中自动保存。）；

cache: False 用于设置数据加载时是否使用缓存。默认不使用（当设置为 True 时，表示使用缓存来加速数据加载。数据将被缓存到RAM或磁盘上，以便在训练过程中快速读取；当设置为 False 时，表示禁用缓存，数据将每次从原始数据源重新加载）；

device 一个可选参数，用于指定模型运行的设备（如果设置为整数（如 device: 0），表示将模型加载到指定的CUDA设备上进行训练或推理；如果设置为列表（如 device: 0,1,2,3），表示将模型加载到多个CUDA设备上并行运行；如果设置为字符串（如 device: cuda），表示将模型加载到可用的CUDA设备上运行，如果没有可用的CUDA设备，则会自动切换到CPU运行；如果设置为字符串（如device: cpu），表示将模型加载到CPU上进行训练或推理）；

workers: 8 设置数据加载过程中的线程数；

seed: 0 设置随机种，以实现结果的可重现性；

deterministic: True 启用确定性模式；

single\_cls: False 指定是否将多类别数据集作为单类别进行训练；

rect: False 确定在train/test模式下是否使用矩形训练数据；

cos\_lr: False 确定是否使用余弦学习率调度器；

close\_mosaic: 10 确定是否在最后几个训练周期中禁用马赛克数据增强；

resume: False 确定是否从上一个checkpoint继续训练模型；

amp: True 确定是否使用自动混合精度（AMP）进行训练；

fraction: 1.0 确定训练时要使用的数据集比例；

profile: False 确定是否在训练过程中启动性能分析，并将结果记录到日志中；

**4.4.3验证/测试参数**

val: True 确定在训练过程中是否进行验证或测试；

split: val 确定用于验证的数据集划分类型；

save\_json: False 确定是否将结果(可能包括训练损失、验证准确率、模型权重等信息)保存到JSON文件中；

save\_hybrid: False 确定是否保存标签的混合版本；

conf 确定目标检测中的目标置信度阈值。

**4.4.4模型导出参数**

format: torchscript 导出模型的格式，可以选择将模型导出为TorchScript格式；

keras: False 是否使用Keras；

optimize: False 是否对TorchScript进行优化；

int8: False 是否进行CoreML或TensorFlow的INT8量化；

dynamic: False 是否使用动态轴；

simplify: False 是否简化ONNX模型；

opset ONNX的opset版本号，为整数类型，可选择性设置；

workspace: 4 TensorRT的工作空间大小，以GB为单位；

nms: False 是否在CoreML中添加非最大抑制（NMS）。

**4.4.5超参数**

learning\_rate=0.005 / 12 / 50, # 学习率

warmup\_steps=600, # 预热步数

warmup\_start\_lr=0.0, # 预热起始学习率

save\_interval\_epochs=5, # 每5个轮次保存一次，有验证数据时，自动评估

lr\_decay\_epochs=[85, 135], # step学习率衰减

# 5 结果展示

（1）0代表佩戴了头盔，1代表没有佩戴头盔。



1. 可以看到图像中无论是大目标检测、中目标检测还是小目标检测，都可以准确找到头部位置并且判断是否佩戴头盔。



1. 在不同尺寸图像中、在不同光照条件下，都可以准确判断图像中的头部位置与头盔是否存在。



1. 图像中存在不同的头盔类别、头盔颜色，算法也能够准确检测头盔位置并标注。



**6优势与挑战**

**6.1优势**

通过标签平滑避免了过拟合风险；

通过数据增强降低了由于数据集体量小导致的特征学习不全面；

通过使用做出不同类别不同大小P-R曲线图对基线模型进行了更为全面的分析。

可扩展性：对于企业，以及工作人员而言，检测效率至关重要。而随着深度学习的发展，YOLO模型更是保障工业实时检测的第一保障。当人员手动识别目标时，人为监管耗时耗力，而且不易实时监管，过程繁琐、消耗人力且实时性较差。在类似的工业问题中可以推广该模型：如零件检测，口罩检测，车牌检测等。

**6.2挑战**

在模型优化方面，可以细化模型参数，从而扩大超参数的调参范围，优化模型参数，提高模型精确度。本次实验由于时间问题，没有对模型进行进一步的超参数修改；

在模型选择方面，可以选用的 GoogLeNet 和 VGG 等神经网络模型，增加神经网络的深度，从而在一定程度上显示对比性，但同时会产生时间开销需要后续优化与解决。

# 7 结论

YOLOv8作为YOLO系列的较新版本，通过在网络架构、损失函数、训练策略和数据增强等方面的改进，在目标检测领域实现了新的突破。其高效性、易用性和鲁棒性使其成为计算机视觉应用中的重要工具。在未来的发展中，YOLOv8有望在模型轻量化、自适应学习和多模态融合等方面继续提升，为更多应用场景提供高效、准确的目标检测解决方案。