

# 机器学习基础研究报告

设计题目：基于深度学习的无人机识别与定位模型

学生姓名：王熠辉

专 业：软件工程

班 级：软件工程23-01

学 号：2023213406

指导教师：徐亮

完成日期：2024年6月11日

# （一）需求和规格说明

**问题描述：**

采用机器学习或深度学习等方法实现图像或视频中的无人机识别与定位。

需要附代码和运行结果截图，截图上须有电脑桌面背景或程序输出的姓名学号等信息。

# （二）实验过程

## 1．问题分析

考虑到图像中的无人机定位，基于Pytorch架构卷积神经网络，并对比成熟的yolov8，进行比较学习。

## 2．设计思想

**环境准备**

Python 3.10环境

Pytorch 2.3.1+cu118

YOLOv8

opencv-python

torchvision

tqdm

等

**数据集准备**

无人机的图像数据集DRONES\_NEW

数据已经被作者分为训练集、验证集和测试集

**程序设计**

a. 自定义卷积神经网络（CNN）结构

设计一个基于Pytorch的卷积神经网络，用于无人机识别与定位

网络结构实际需求调整，目前为根据VGG16改编的网络

使用AdamW为优化器，mse作为损失函数

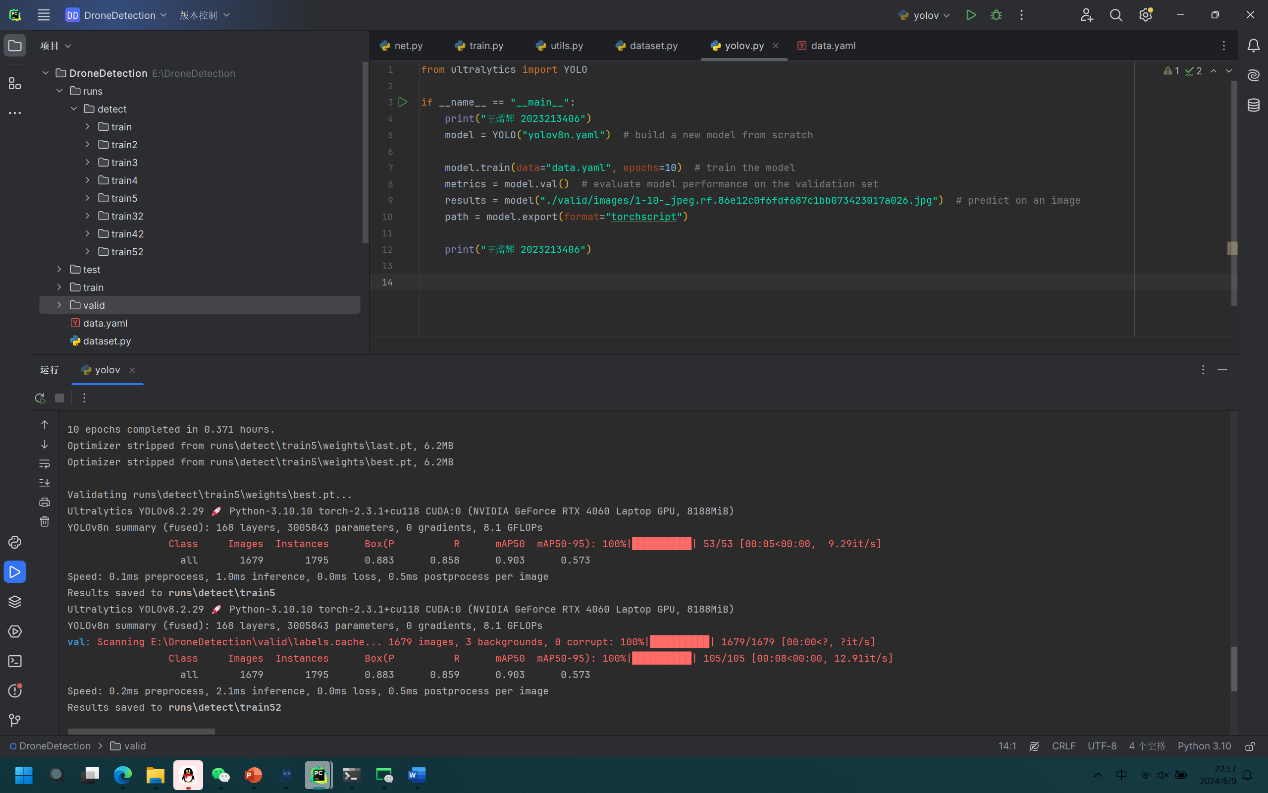
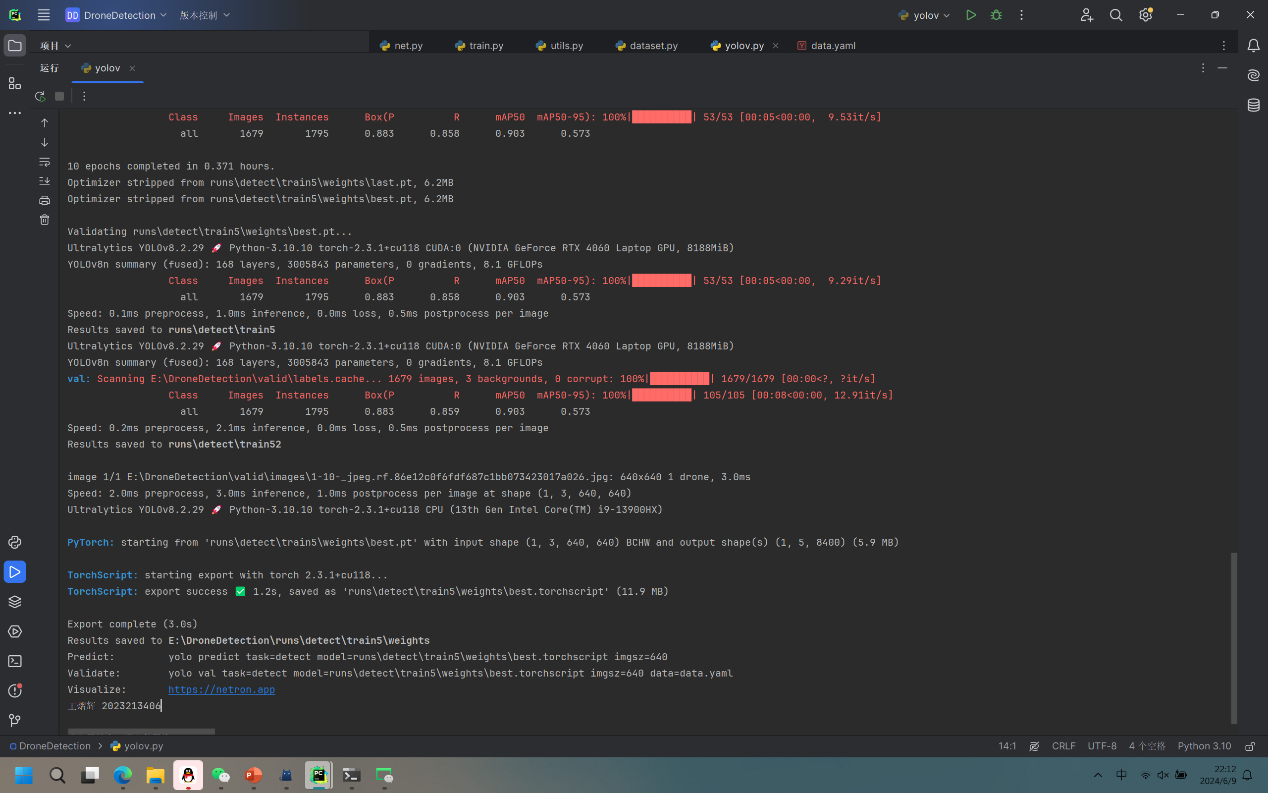
b. YOLOv8模型训练与测试

使用官方提供的YOLOv8模型

将数据集转换为YOLOv8格式，进行模型训练和测试

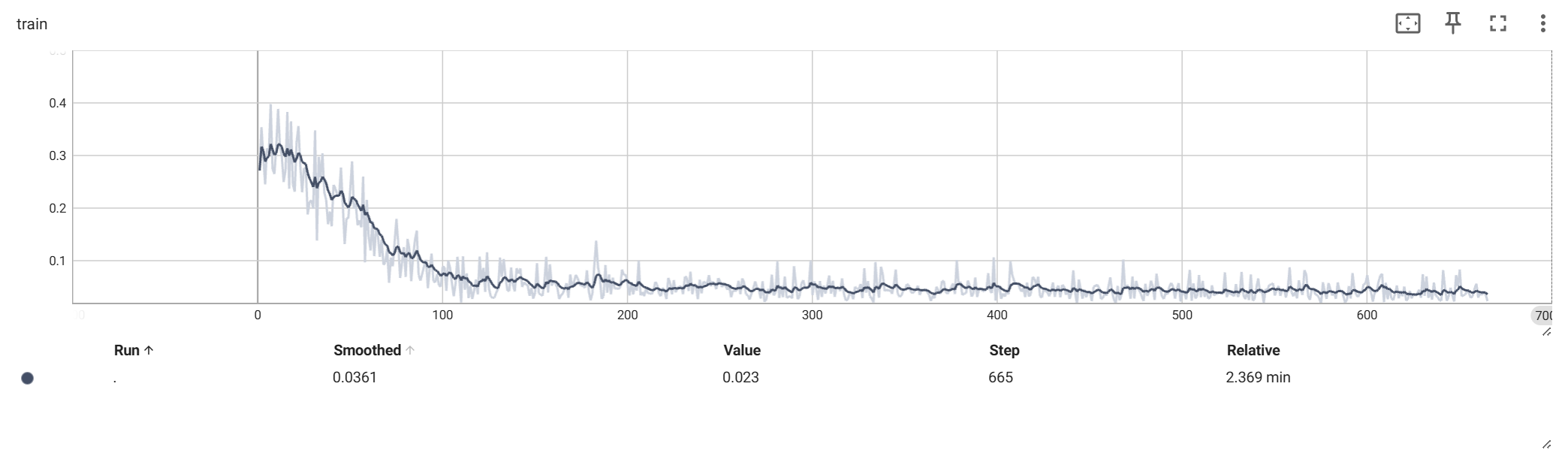
调整超参数，如学习率、迭代次数等，以获得最佳性能

## 3. 实验过程截图

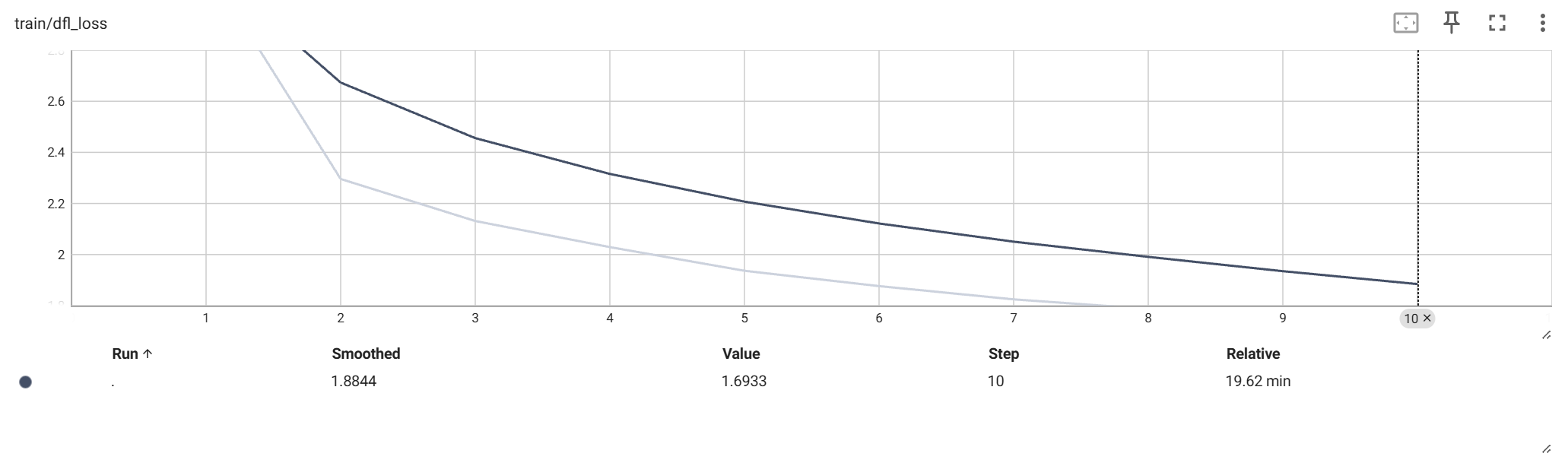
****

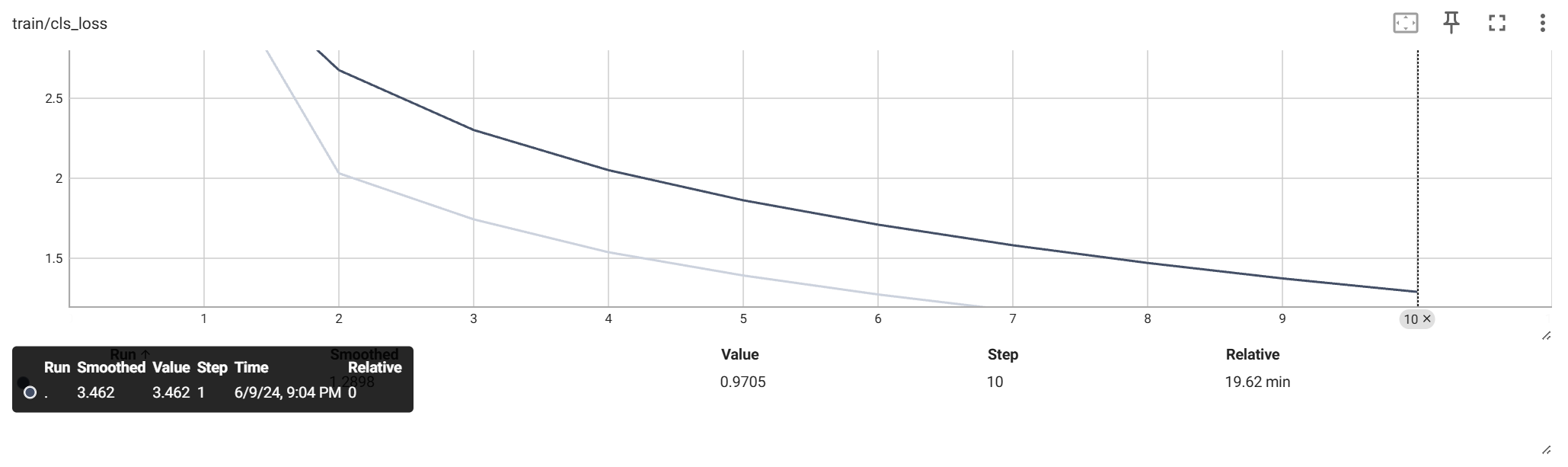
**姓名学号信息已经在图片中展示**

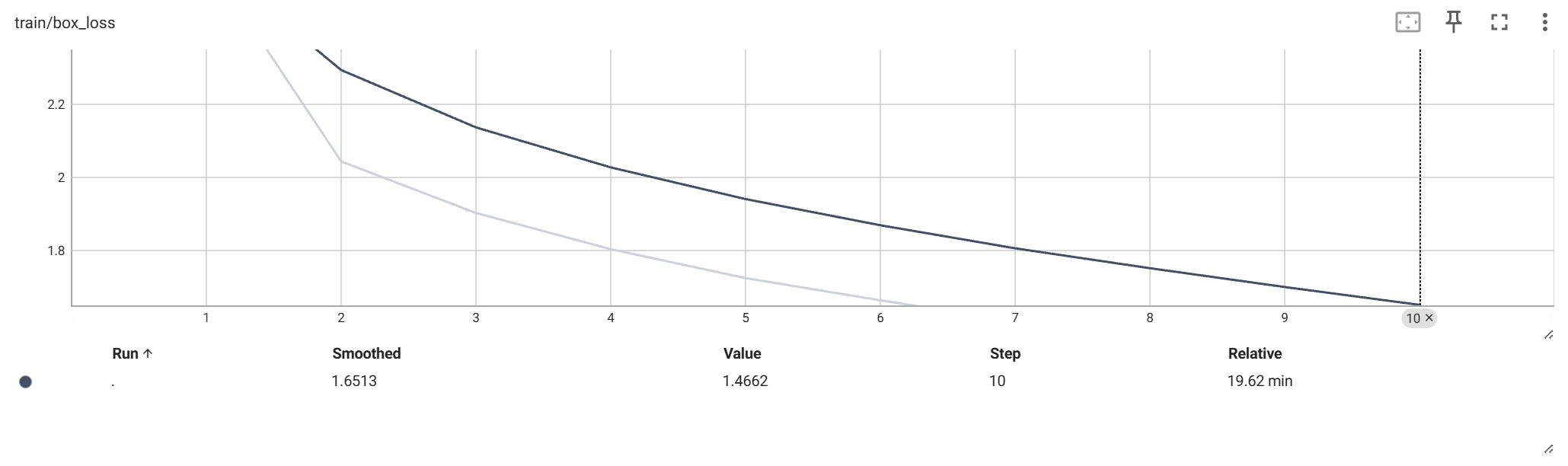
# （三）实验结果

**自定义卷积神经网络训练时MSE**

**yolov8训练过程**

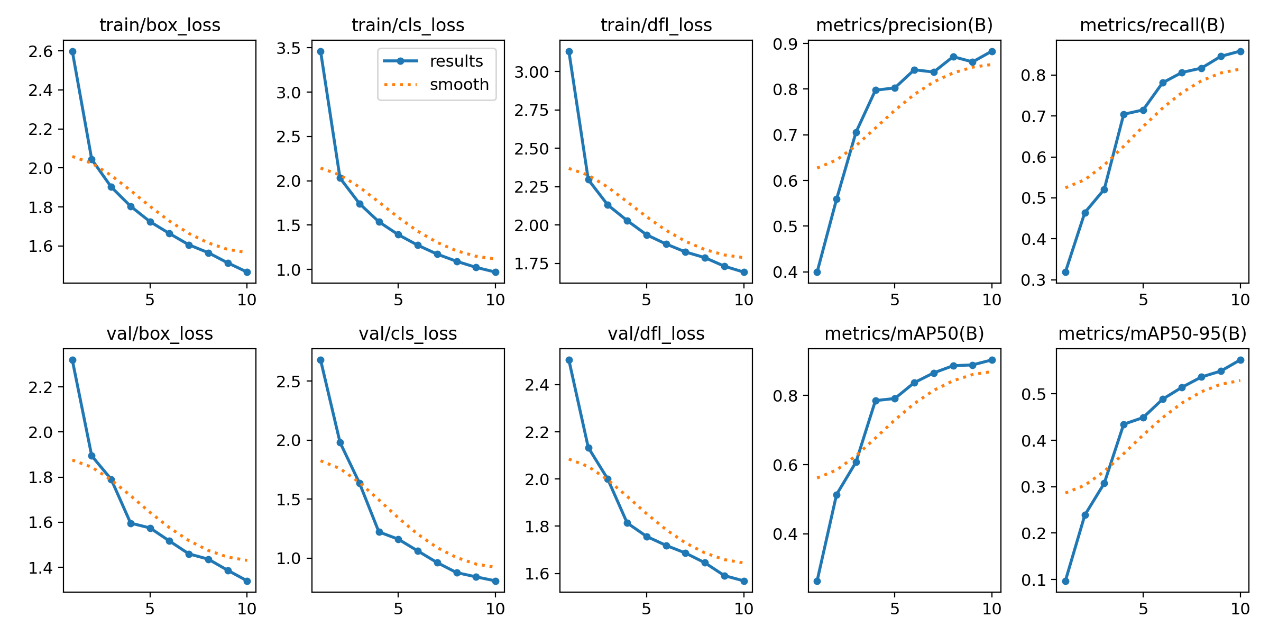
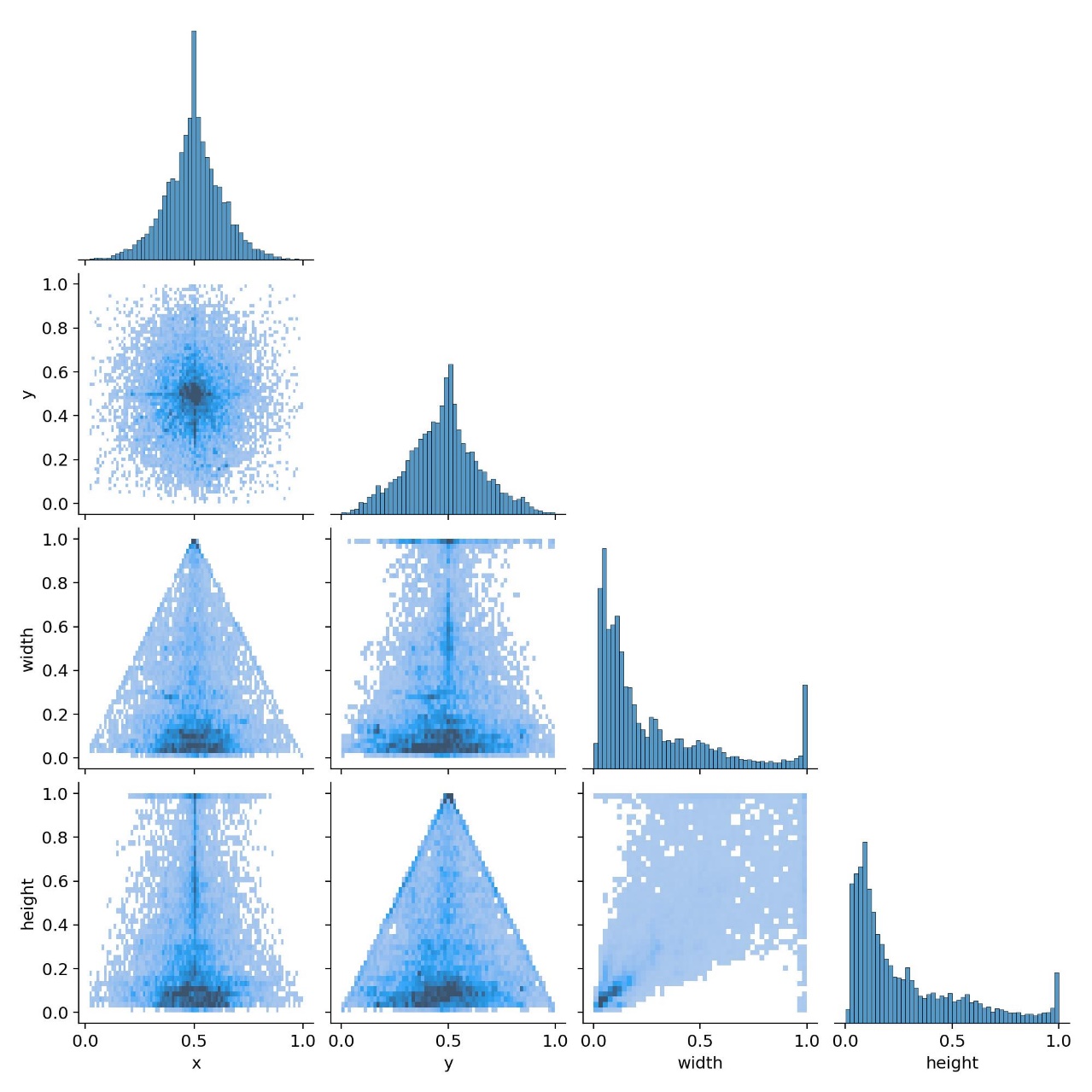


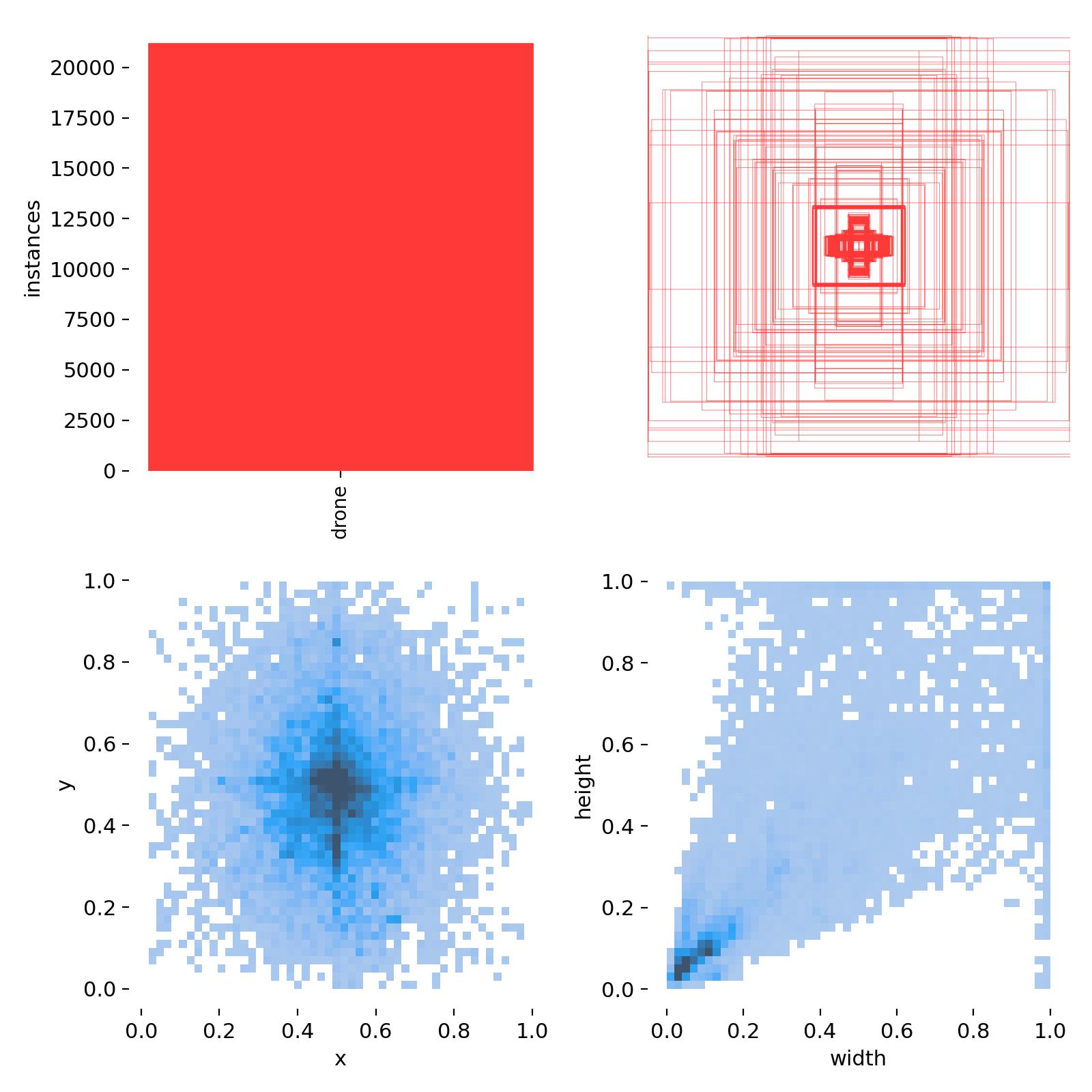
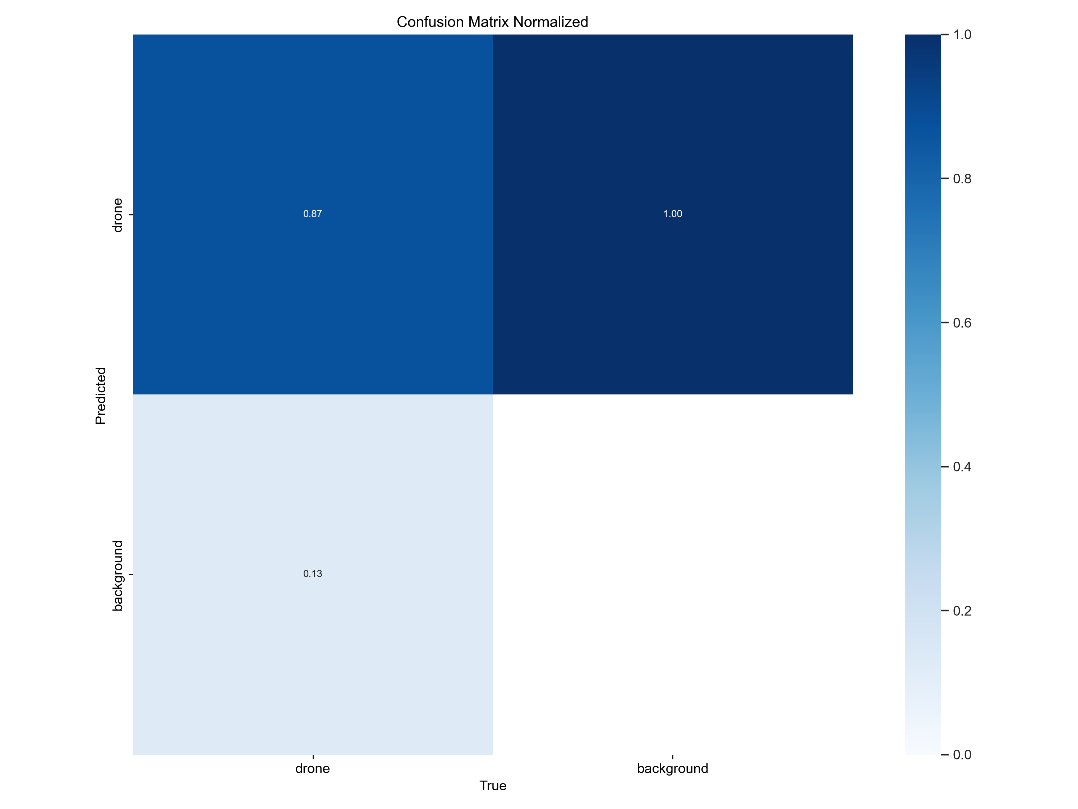


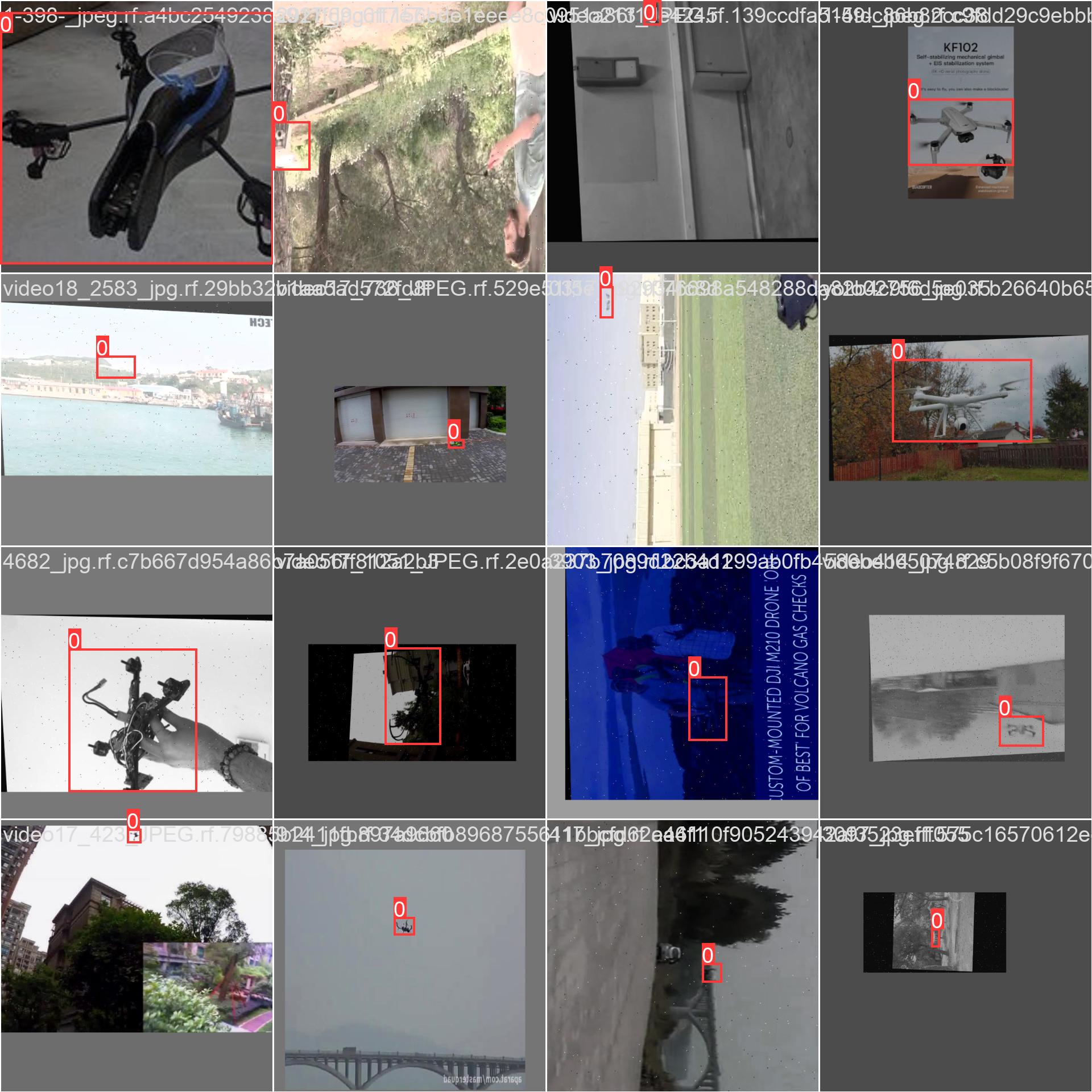


**截取自tensorboard**

**yolov8训练结果**

****

****

****

# （四）程序源代码

./net.py

1. from torch import nn
2. from einops import rearrange
3. from einops.layers.torch import Rearrange
4. class Net(nn.Module):
5. def \_\_init\_\_(self):
6. super().\_\_init\_\_()
7. self.layer1 = nn.Sequential(
8. nn.Conv2d(in\_channels=3, out\_channels=16, kernel\_size=3, stride=1, padding=1),
9. nn.LeakyReLU(inplace=True),
10. nn.Conv2d(in\_channels=16, out\_channels=16, kernel\_size=3, stride=1, padding=1),
11. nn.LeakyReLU(inplace=True),
12. nn.MaxPool2d(kernel\_size=2, stride=2)
13. )
14. self.layer2 = nn.Sequential(
15. nn.Conv2d(in\_channels=16, out\_channels=32, kernel\_size=3, stride=1, padding=1),
16. nn.LeakyReLU(inplace=True),
17. nn.Conv2d(in\_channels=32, out\_channels=32, kernel\_size=3, stride=1, padding=1),
18. nn.LeakyReLU(inplace=True),
19. nn.MaxPool2d(2, 2)
20. )
21. self.layer3 = nn.Sequential(
22. nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, stride=1, padding=1),
23. nn.LeakyReLU(inplace=True),
24. nn.Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3, stride=1, padding=1),
25. nn.LeakyReLU(inplace=True),
26. nn.Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3, stride=1, padding=1),
27. nn.LeakyReLU(inplace=True),
28. nn.MaxPool2d(2, 2)
29. )
30. self.layer4 = nn.Sequential(
31. nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, stride=1, padding=1),
32. nn.LeakyReLU(inplace=True),
33. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
34. nn.LeakyReLU(inplace=True),
35. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
36. nn.LeakyReLU(inplace=True),
37. nn.MaxPool2d(2, 2)
38. )
39. self.layer5 = nn.Sequential(
40. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
41. nn.LeakyReLU(inplace=True),
42. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
43. nn.LeakyReLU(inplace=True),
44. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
45. nn.LeakyReLU(inplace=True),
46. nn.MaxPool2d(2, 2)
47. )
48. self.layer6 = nn.Sequential(
49. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
50. nn.LeakyReLU(inplace=True),
51. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
52. nn.LeakyReLU(inplace=True),
53. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
54. nn.LeakyReLU(inplace=True),
55. nn.MaxPool2d(2, 2)
56. )
57. self.layer7 = nn.Sequential(
58. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
59. nn.LeakyReLU(inplace=True),
60. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
61. nn.LeakyReLU(inplace=True),
62. nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),
63. nn.LeakyReLU(inplace=True),
64. nn.MaxPool2d(2, 2)
65. )
66. self.conv = nn.Sequential(
67. self.layer1,
68. self.layer2,
69. self.layer3,
70. self.layer4,
71. self.layer5,
72. self.layer6,
73. self.layer7,
74. )
75. self.fc = nn.Sequential(
76. Rearrange('b c w h -> b (c w h)'),
77. nn.Linear(3200, 512),
78. nn.LeakyReLU(),
79. nn.Dropout(0.2),
80. nn.Linear(512, 256),
81. nn.LeakyReLU(),
82. nn.Dropout(0.2),
83. )
84. self.cv = nn.Sequential(
85. self.conv,
86. self.fc
87. )
88. self.last\_hidden = nn.Sequential(
89. nn.BatchNorm1d(256),
90. nn.Linear(256, 128),
91. nn.LeakyReLU(inplace=True),
92. nn.Dropout(0.1),
93. nn.Linear(128, 32),
94. nn.LeakyReLU(inplace=True),
95. nn.Dropout(0.1),
96. nn.Linear(32, 16),
97. nn.LeakyReLU(inplace=True),
98. nn.Linear(16, 4),
99. )
100. def forward(self, x):
101. img = rearrange(x, 'b w h c->b c w h')
102. cv\_output = self.cv(img)
103. combine\_output = self.last\_hidden(cv\_output)
104. return combine\_output

./train.py

1. from torch.utils.tensorboard import SummaryWriter
2. import torch
3. from net import Net
4. from dataset import DroneDataset
5. from torch.utils.data import DataLoader
6. learning\_rate = 0.0002
7. epoch = 5
8. total\_strain\_step = 0
9. if \_\_name\_\_ == '\_\_main\_\_':
10. dataset = DroneDataset('./test')
11. net = Net().to('cuda')
12. optimizer = torch.optim.AdamW(net.parameters(), lr=learning\_rate)
13. loss\_fn = torch.nn.MSELoss().to('cuda')
14. writer = SummaryWriter('logs')
15. for epoch in range(epoch):
16. for img, label in DataLoader(dataset, batch\_size=8, shuffle=True):
17. img = img.to('cuda')
18. label = label.to('cuda')
19. output = net(img)
20. loss = loss\_fn(output, label)
21. optimizer.zero\_grad()
22. loss.backward()
23. optimizer.step()
24. total\_strain\_step += 1
25. writer.add\_scalar('train', loss.item(), total\_strain\_step)

./dataset.py

1. import torch
2. from torch.utils.data import Dataset
3. from cv2 import imread, cvtColor, COLOR\_BGR2HSV, resize
4. from einops import rearrange
5. import os
6. import numpy as np
7. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
8. print(device)
9. class DroneDataset(Dataset):
10. def \_\_init\_\_(self, path):
11. self.path = path
12. self.files = os.listdir(os.path.join(path, 'images'))
13. def \_\_len\_\_(self):
14. return len(self.files)
15. def \_\_getitem\_\_(self, idx):
16. img\_path = os.path.join(self.path, 'images', self.files[idx])
17. label\_path = os.path.join(self.path, 'labels', os.path.splitext(self.files[idx])[0] + '.txt')
18. bgr\_img = resize(imread(img\_path), (640, 640))
19. cv\_img = cvtColor(bgr\_img, COLOR\_BGR2HSV) / 255.  *# 转为hsv并且标准化*
20. img: torch.Tensor = torch.from\_numpy(cv\_img).float()
21. img.to(device)
22. label: torch.Tensor = torch.from\_numpy(np.fromfile(label\_path, sep=' ', dtype=np.float32))
23. label.to(device)
24. if len(label) < 5:
25. label = torch.zeros(5).float()
26. label.to(device)
27. return img, label[1:5]
28. if \_\_name\_\_ == '\_\_main\_\_':
29. dataset = DroneDataset('./test')
30. print(dataset[0])

./yolov

1. from ultralytics import YOLO
2. if \_\_name\_\_ == "\_\_main\_\_":
3. print("王熠辉 2023213406")
4. model = YOLO("yolov8n.yaml")  *# build a new model from scratch*
5. model.train(data="data.yaml", epochs=10)  *# train the model*
6. metrics = model.val()  *# evaluate model performance on the validation set*
7. results = model("./valid/images/1-10-\_jpeg.rf.86e12c0f6fdf687c1bb073423017a026.jpg")  *# predict on an image*
8. path = model.export(format="torchscript")
9. print("王熠辉 2023213406")

# （五）感想

通过本次研究，我深入了解了深度学习在计算机视觉领域的应用，特别是无人机识别与定位技术。在这个过程中，我不仅掌握了Pytorch这一强大的深度学习框架，还对比学习了自定义卷积神经网络与当前成熟的YOLOv8模型。

这次研究经历不仅提升了我的技术能力，也培养了我的科研素养。我将继续探索深度学习在计算机视觉领域的应用，为无人机监控和其他实际应用场景提供更好的解决方案。