



**Improved YOLOv5 with attention
mechanism research on object detection
without forms**

Cardiff University

School of Computer Science and Informatics

Supervisor – Dr Charith Perera

Moderator – Alia I Abdelmoty

Author: JunLin LI

Student Number: C21051490

Abstract

Deep learning outperforms other machine learning algorithms in many areas. Deep learning techniques is prevalent in computer vision with convolutional neural networks (CNN). Computer vision has long been interested in object detection. The advanced convolutional neural network-based YOLO algorithm. The YOLO technique was derived from Joseph Redmon's in 2015 Conference on Computer Vision and Pattern Recognition paper. The YOLOv5 detection system is accurate and powerful for shaped objects. Many objects in existence differ from conventional shapes and colours, as shown by fire, steam, and water on the floor. Thus, the detection framework yields poor accuracy and practicality. The YOLOv5 object identification framework is integrated with an attention technology module in this dissertation. The model prioritises important regions in an image by giving attention weights, improving object detection accuracy. The framework's essential characteristics and important outcome criteria have been improved by merging three typical comparative study attention modules, SENet, CBAM, and CoordAtt. Overall design optimisation occurred during consolidation. The YOLOv5 framework's attention technology module improves Mean Average Precision (mAP). SENet has 86.1% mAP, CoordAtt 85.2%, and CBAM 88.3%. These figures are 1.3%, 0.4%, and 3.5% better than the original YOLOv5 framework's 84.8% mAP. The results show that adding the attention technology module to the YOLOv5 framework improves object detection for irregular forms, reducing the issue of decreasing detection accuracy. This study develops and provides technological support for a detection system that can identify objects without a steady shape.

Keywords: without shape detection, attention mechanism, convolutional neural networks, mAP, YOLOv5

Acknowledgements

First of all, I would like to thank my supervisor Dr Charith Perera. After several meetings, we jointly determined the general direction of my dissertation and gave me guiding opinions. Secondly, I would like to thank my parents for giving me this opportunity to study abroad. Finally, I would like to thank my fiancée, I wasn't with her this year and she took a lot of pressure from society. She did not complain at all, and always gave me support and accompanied me through many sleepless nights.

content

Abstract.....	2
Acknowledgements	3
List of Figures.....	6
List of Tables.....	25
1. Introduction	28
1.1 Overview.....	28
1.2 Challenging.....	29
1.3 Aims and Objectives	29
1.4 Scope	30
1.5 Learning Outcomes	30
2. Background.....	31
2.1 Wider Context.....	31
2.2 Identified Problem Area	32
2.3 History	33
2.4 Two-stage detector CNN	34
2.5 Single-stage detector YOLOv5	35
2.6 Attention mechanism	36
2.7 Capture Feature Information	37
3. Specification and Design.....	38
3.1 Requirement Analysis.....	38
3.2 Design goals and objectives	39
3.3 Design method	41
3.4 Data Collection	43
3.5 Data Preprocessing	44
3.6 Architecture	46
3.6.1 YOLOv5.....	46
3.6.2 Attention mechanisms.....	51
3.7 Implementation Plan	56
3.8 Risks and Challenges	57
4. Implementation.....	58

4.1	Tools.....	58
4.1.1	Tools.....	59
4.2	Code.....	61
4.2.1	train.py.....	62
4.2.2	detect.py	64
4.2.3	myData.yaml.....	64
4.2.4	yolov5s.yaml.....	65
4.2.5	CBAM	69
4.2.6	CoordAtt.....	72
4.2.7	SENet	75
4.3	Development Process.....	77
4.3.1	Analyzing the underlying neural network.....	77
4.3.2	SPPF	78
4.3.3	Tuning the underlying neural network	79
4.3.4	Specific implementation process.....	81
5.	Result and Evaluation	87
5.1	Experimental environment	87
5.2	Experimental Methods	87
5.3	Analysis of results.....	90
6.	Conclusion and Future work.....	92
6.1	Conclusion.....	92
6.2	Main contributions.....	93
6.3	Comparison of methods.....	94
6.4	Limitations and Future work.....	96
6.4.1	Limitations.....	96
6.4.2	Future work.....	96
7.	Reflection on Learning	97
	References.....	100

List of Figures

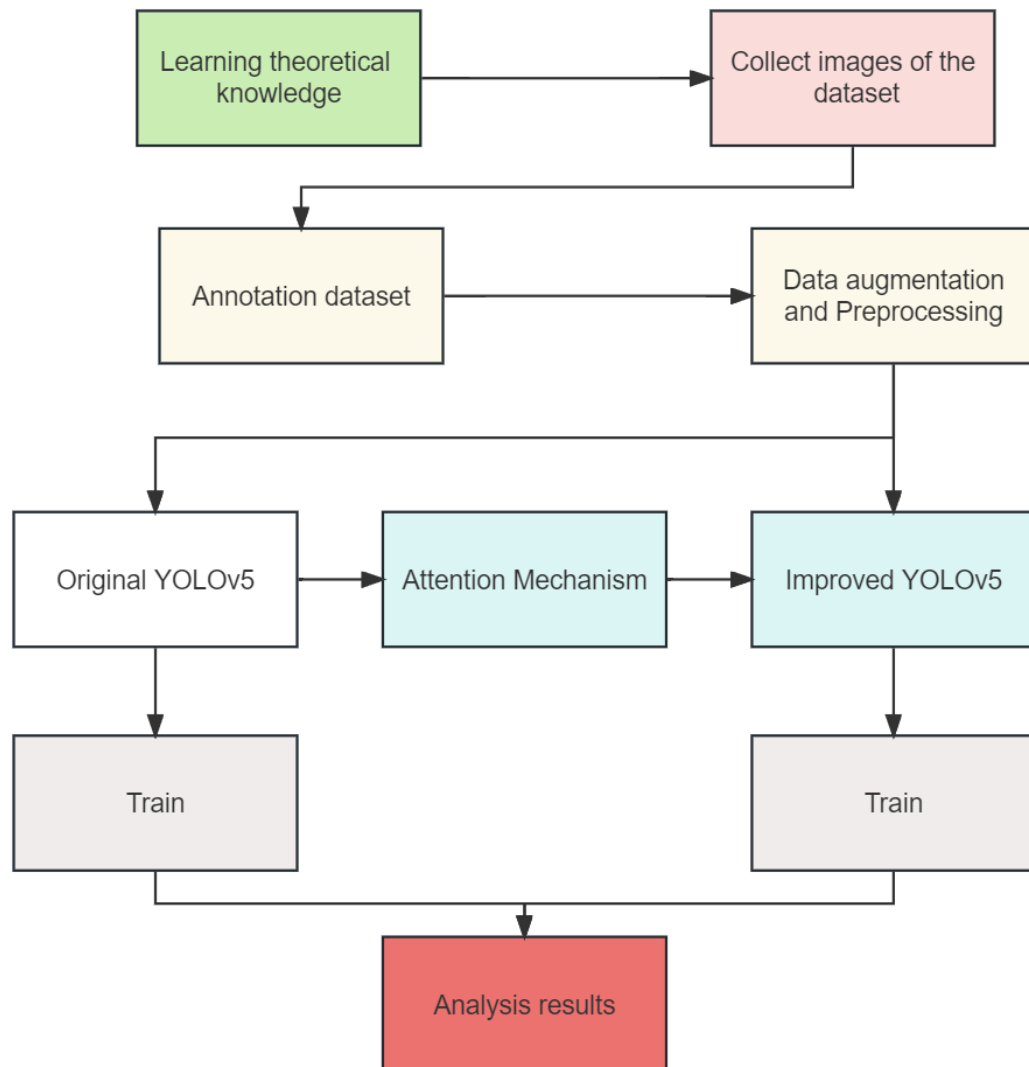


Figure 1-Flowchart of the design method

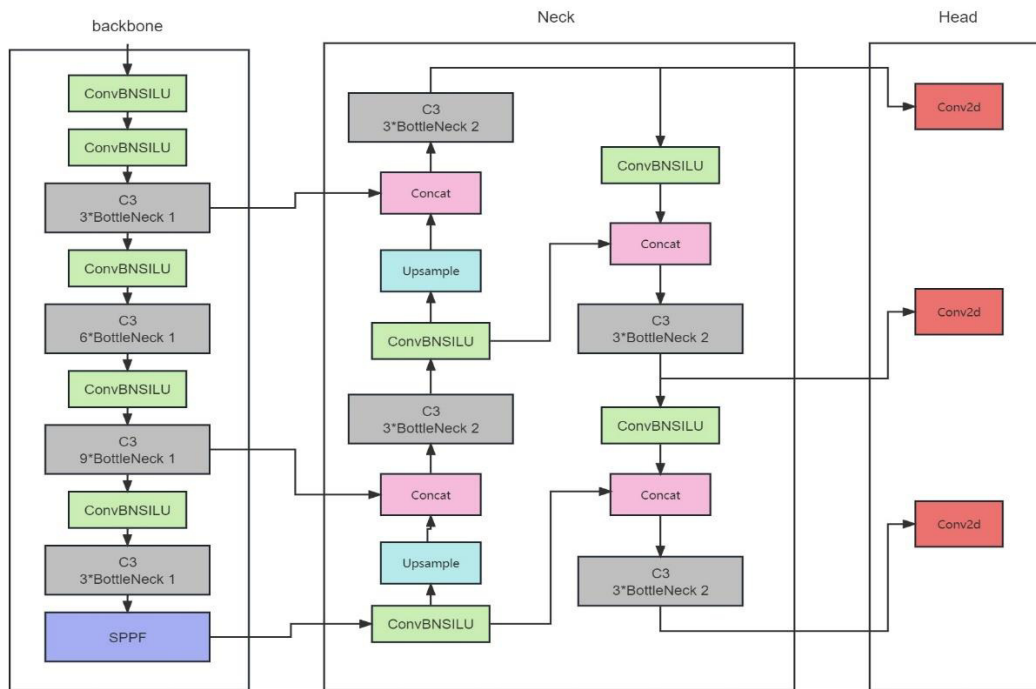


Figure 2-Basic Neural Network Architecture (Ultralytics 2023)

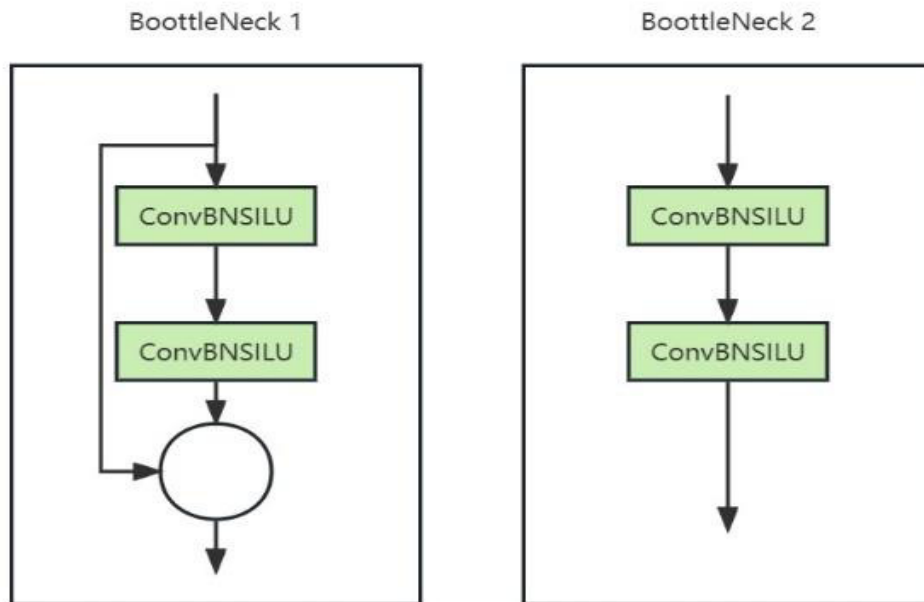


Figure 3-The structure of BottleNeck (Ultralytics 2023)

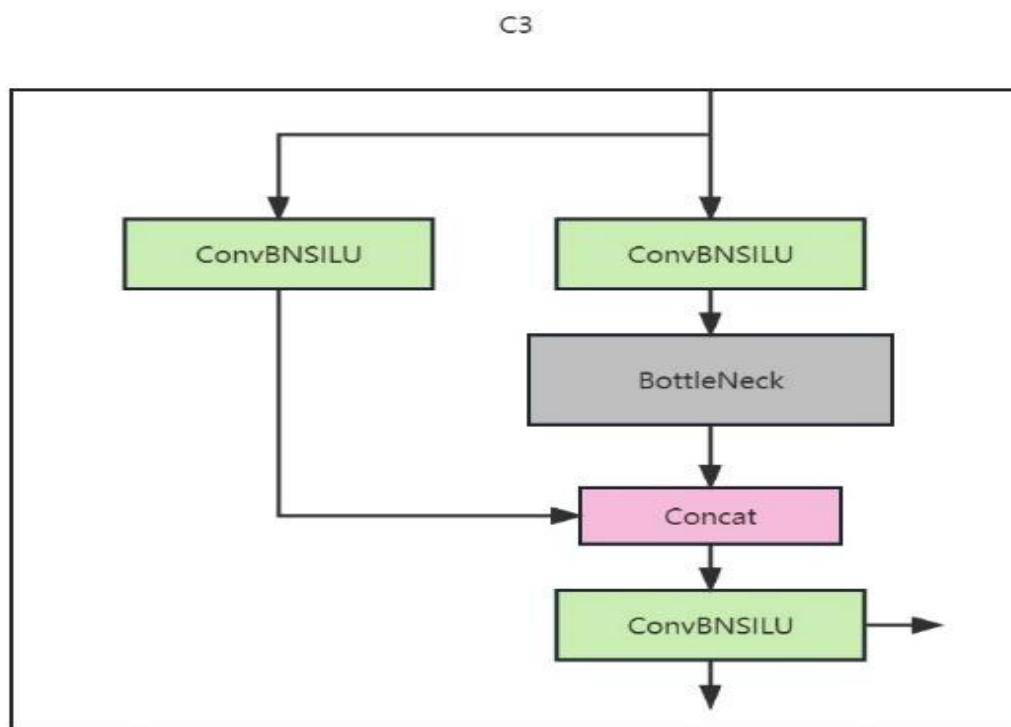


Figure 4-The structure of C3 (Ultralytics 2023)

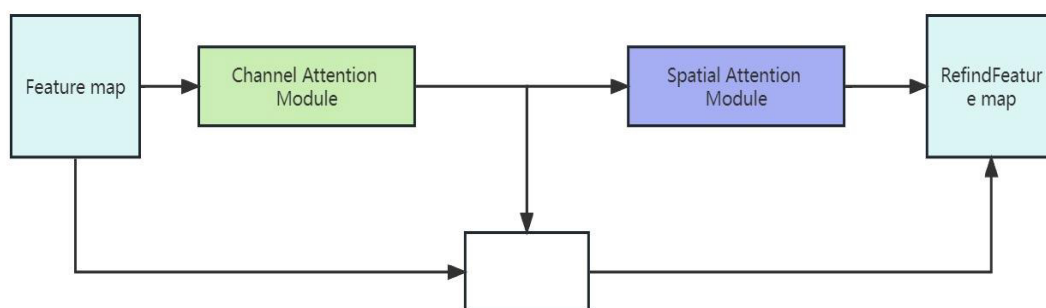


Figure 5-CBAM module

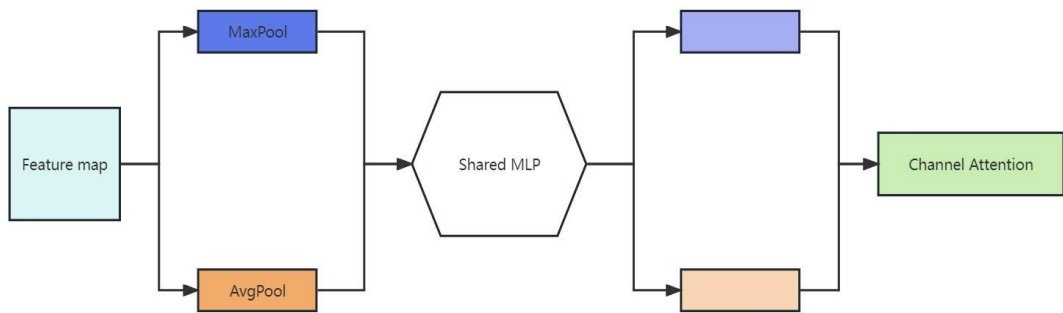


Figure 6-Channel module

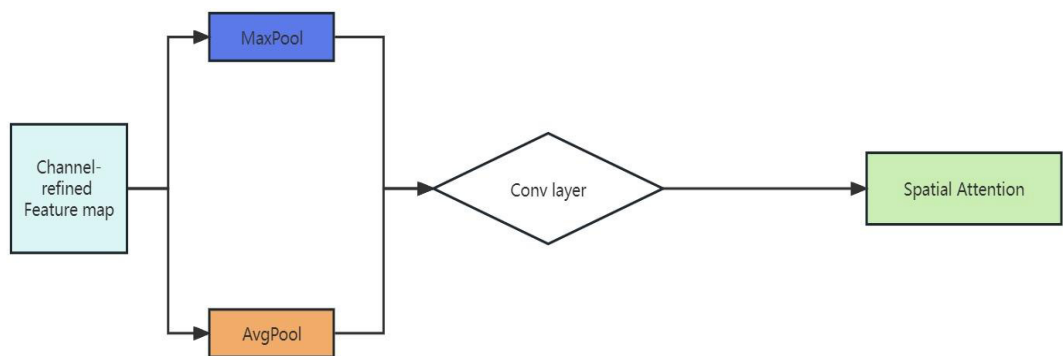


Figure 7-Spatial module

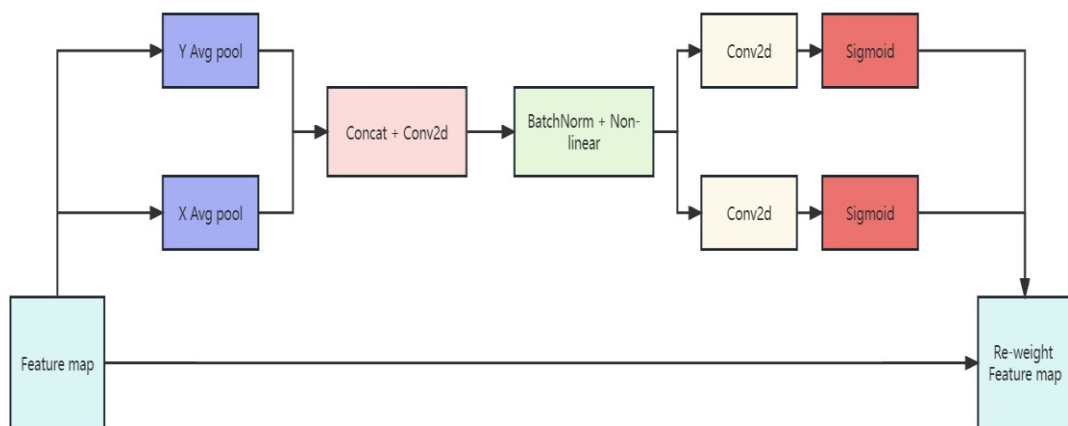


Figure 8-CoordAtt module

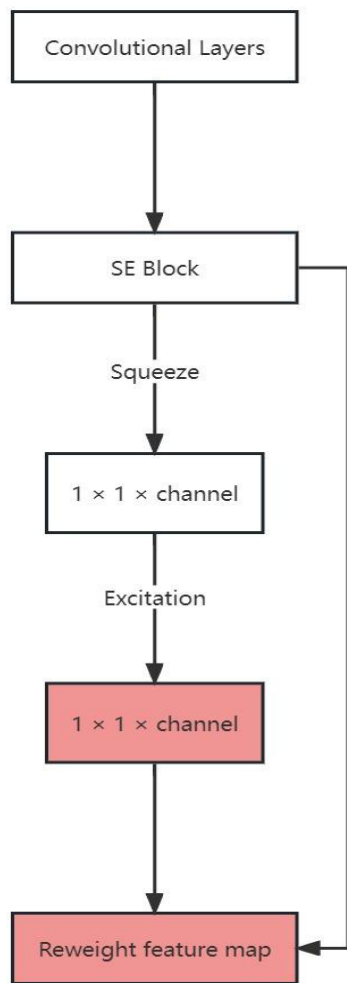


Figure 9-SENet module

```

443 def parse_opt(known=False):
444     parser = argparse.ArgumentParser()
445     parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
446     parser.add_argument('--cfg', type=str, default=ROOT / 'models/yolov5s-CoordAtt.yaml', help='model.yaml path')
447     parser.add_argument('--data', type=str, default=ROOT / 'data/myData.yaml', help='dataset.yaml path')
448     parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
449     parser.add_argument('--epochs', type=int, default=200, help='total training epochs')
450     parser.add_argument('--batch-size', type=int, default=4, help='total batch size for all GPUs, -1 for autobatch')
451     parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
452     parser.add_argument('--rect', action='store_true', help='rectangular training')
453     parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
454     parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
455     parser.add_argument('--noval', action='store_true', help='only validate final epoch')
456     parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
457     parser.add_argument('--noplots', action='store_true', help='save no plot files')
458     parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
459     parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
460     parser.add_argument('--cache', type=str, nargs='?', const='ram', help='image --cache ram/disk')
461     parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
462     parser.add_argument('--device', default='0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
463     parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%')
464     parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
465     parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
466     parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
467     parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
468     parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
469     parser.add_argument('--name', default='exp', help='save to project/name')
470     parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
471     parser.add_argument('--quad', action='store_true', help='quad dataloader')
472     parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
473     parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
474     parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
475     parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
476     parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
477     parser.add_argument('--seed', type=int, default=0, help='Global training seed')
478     parser.add_argument('--local_rank', type=int, default=-1, help='Automatic DDP Multi-GPU argument, do not modify')

```

Figure 10 – The main code of train.py

```

219 def parse_opt():
220     parser = argparse.ArgumentParser()
221     parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
222     parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(webcam)')
223     parser.add_argument('--data', type=str, default=ROOT / 'data/myData.yaml', help='(optional) dataset.yaml path')
224     parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
225     parser.add_argument('--conf-thres', type=float, default=0.5, help='confidence threshold')
226     parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
227     parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
228     parser.add_argument('--device', default='0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
229     parser.add_argument('--view-img', action='store_true', help='show results')
230     parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
231     parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
232     parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
233     parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
234     parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
235     parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
236     parser.add_argument('--augment', action='store_true', help='augmented inference')
237     parser.add_argument('--visualize', action='store_true', help='visualize features')
238     parser.add_argument('--update', action='store_true', help='update all models')
239     parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
240     parser.add_argument('--name', default='exp', help='save results to project/name')
241     parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
242     parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
243     parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
244     parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
245     parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
246     parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
247     parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
248     opt = parser.parse_args()
249     opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1 # expand
250     print_args(vars(opt))
251     return opt

```

Figure 11 – The main code of detect.py

```

1 # YOLOv5 by Ultralytics, AGPL-3.0 license
2 # COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics
3 # Example usage: python train.py --data coco128.yaml
4 # parent
5 # └─ yolo5
6 #   └─ datasets
7 #     └─ coco128 ← downloads here (7 MB)
8
9
10 # Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ...]
11 path: ../dataset7.15 # dataset root dir
12 train: train/images # train images (relative to 'path') 128 images
13 val: valid/images # val images (relative to 'path') 128 images
14 test: test/images # test images (optional)
15
16 # Classes
17 names:
18 0: dirt
19 1: fire
20 2: steam

```

Figure 12 – The code of myData.yaml


```

1 # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3 # Parameters
4 nc: 3 # number of classes
5 depth_multiple: 0.33 # model depth multiple
6 width_multiple: 0.50 # layer channel multiple
7 anchors:
8   - [10,13, 16,30, 33,23] # P3/8
9   - [30,61, 62,45, 59,119] # P4/16
10  - [116,90, 156,198, 373,326] # P5/32
11
12 # YOLOv5 v6.0 backbone
13 backbone:
14   # [from, number, module, args]
15   [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17    [-1, 3, C3, [128]],
18    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19    [-1, 6, C3, [256]],
20    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21    [-1, 9, C3, [512]],
22    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23    [-1, 3, C3, [1024]],
24    [-1, 1, SPPF, [1024, 5]], # 9
25   ]

```

Figure 13 – The code of yolov5s.yaml backbone

```

26
27 # YOLOv5 v6.0 head
28 head:
29   [[-1, 1, Conv, [512, 1, 1]],
30    [-1, 1, nn.Upsample, [None, 2, 'nearest']],
31    [[-1, 6], 1, Concat, [1]], # cat backbone P4
32    [-1, 3, C3, [512, False]], # 13
33
34    [-1, 1, Conv, [256, 1, 1]],
35    [-1, 1, nn.Upsample, [None, 2, 'nearest']],
36    [[-1, 4], 1, Concat, [1]], # cat backbone P3
37    [-1, 3, C3, [256, False]], # 17 (P3/8-small)
38
39    [-1, 1, Conv, [256, 3, 2]],
40    [[-1, 14], 1, Concat, [1]], # cat head P4
41    [-1, 3, C3, [512, False]], # 20 (P4/16-medium)
42
43    [-1, 1, Conv, [512, 3, 2]],
44    [[-1, 10], 1, Concat, [1]], # cat head P5
45    [-1, 3, C3, [1024, False]], # 23 (P5/32-large)
46
47    [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
48 ]

```

Figure 14 – The code of yolov5s.yaml head

```

955 # @inproceedings{hou2021coordinate,
956 #   title={Convolutional Block Attention Module},
957 #   author={Sanghyun Woo, Jongchan Park and Joon-Young Lee and So Kweon},
958 #   booktitle={ECCV},
959 #   year={2018}
960 # }
961
962 # CBAM注意力机制
963 new *
964 class ChannelAttention(nn.Module):
965     new *
966     def __init__(self, in_planes, ratio=16):
967         super(ChannelAttention, self).__init__()
968         self.avg_pool = nn.AdaptiveAvgPool2d(1)
969         self.max_pool = nn.AdaptiveMaxPool2d(1)
970
971         self.fc1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
972         self.relu1 = nn.ReLU()
973         self.fc2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)
974
975         self.sigmoid = nn.Sigmoid()
976
977     new *
978     def forward(self, x):
979         avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
980         max_out = self.fc2(self.relu1(self.fc1(self.max_pool(x))))
981         out = avg_out + max_out
982         return self.sigmoid(out)

```

Figure 15 – The code of CBAM's ChannelAttention

```

2 usages new *
983 class SpatialAttention(nn.Module):
    new *
984     def __init__(self, kernel_size=7):
985         super(SpatialAttention, self).__init__()
986
987         assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
988         padding = 3 if kernel_size == 7 else 1
989
990         self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
991         self.sigmoid = nn.Sigmoid()
992
    new *
993     def forward(self, x):
994         avg_out = torch.mean(x, dim=1, keepdim=True)
995         max_out, _ = torch.max(x, dim=1, keepdim=True)
996         x = torch.cat([avg_out, max_out], dim=1)
997         x = self.conv1(x)
998         return self.sigmoid(x)

```

Figure 16 – The code of CBAM's SpatialAttention

```

1 usage new *
1000 class CBAM(nn.Module):
    new *
1001     def __init__(self, in_planes, ratio=16, kernel_size=7):
1002         super(CBAM, self).__init__()
1003         self.ca = ChannelAttention(in_planes, ratio)
1004         self.sa = SpatialAttention(kernel_size)
1005
    new *
1006     def forward(self, x):
1007         x = self.ca(x) * x
1008         x = self.sa(x) * x
1009         return x

```

Figure 17 – The code of main CBAM

```

877 # CA
878 # @inproceedings{hou2021coordinate,
879 #   title={Coordinate Attention for Efficient Mobile Network Design},
880 #   author={Hou, Qibin and Zhou, Daquan and Feng, Jiashi},
881 #   booktitle={CVPR},
882 #   year={2021}
883 # }
2 usages new *
884 class h_sigmoid(nn.Module):
    new *
885     def __init__(self, inplace=True):
886         super(h_sigmoid, self).__init__()
887         self.relu = nn.ReLU6(inplace=inplace)
    new *
888     def forward(self, x):
889         return self.relu(x + 3) / 6
2 usages new *
890 class h_swish(nn.Module):
    new *
891     def __init__(self, inplace=True):
892         super(h_swish, self).__init__()
893         self.sigmoid = h_sigmoid(inplace=inplace)
    new *
894     def forward(self, x):
895         return x * self.sigmoid(x)

```

Figure 18 – The activation function of CoordAtt

```

897 class CoordAtt(nn.Module):
    new *
898     def __init__(self, inp, oup, reduction=32):
899         super(CoordAtt, self).__init__()
900         self.pool_h = nn.AdaptiveAvgPool2d((None, 1))
901         self.pool_w = nn.AdaptiveAvgPool2d((1, None))
902         mip = max(8, inp // reduction)
903         self.conv1 = nn.Conv2d(inp, mip, kernel_size=1, stride=1, padding=0)
904         self.bn1 = nn.BatchNorm2d(mip)
905         self.act = h_swish()
906         self.conv_h = nn.Conv2d(mip, oup, kernel_size=1, stride=1, padding=0)
907         self.conv_w = nn.Conv2d(mip, oup, kernel_size=1, stride=1, padding=0)
    new *
908     def forward(self, x):
909         identity = x
910         n, c, h, w = x.size()
911         #c*1*W
912         x_h = self.pool_h(x)
913         #c*H*1
914         #C*1*h
915         x_w = self.pool_w(x).permute(0, 1, 3, 2)
916         y = torch.cat([x_h, x_w], dim=2)
917         #C*1*(h+w)
918         y = self.conv1(y)
919         y = self.bn1(y)
920         y = self.act(y)
921         x_h, x_w = torch.split(y, [h, w], dim=2)
922         x_w = x_w.permute(0, 1, 3, 2)
923         a_h = self.conv_h(x_h).sigmoid()
924         a_w = self.conv_w(x_w).sigmoid()
925         out = identity * a_w * a_h
926         return out

```

Figure 19 – The main code of CoordAtt

```

936 # SENet注意力机制
937 1 usage new *
938 class SELayer(nn.Module):
939     new *
940     def __init__(self, channel, reduction = 16):
941         super(SELayer, self).__init__()
942         self.avg_pool = nn.AdaptiveAvgPool2d(1)
943         self.fc = nn.Sequential(
944             nn.Linear(channel, channel // reduction, bias=False),
945             nn.ReLU(inplace=True),
946             nn.Linear(channel // reduction, channel, bias=False),
947             nn.Sigmoid()
948         )
949     def forward(self, x):
950         b, c, _, _ = x.size()
951         y = self.avg_pool(x).view(b, c)
952         y = self.fc(y).view(b, c, 1, 1)
953         return x * y.expand_as(x)

```

Figure 20 – The code of SENet

```

314 args[j] = eval(a) if isinstance(a, str) else a # eval strings
315
316 n = n_ = max(round(n * gd), 1) if n > 1 else n # depth gain
317 if m in {
318     Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, SPPE, DWConv, MixConv2d, Focus, CrossConv,
319     BottleneckCSP, C3, C3TR, C3SPP, C3Ghost, nn.ConvTranspose2d, DWConvTranspose2d, C3x, SELayer, CBAM, CoordAtt}:
320     c1, c2 = ch[f], args[0]
321     if c2 != no: # if not output
322         c2 = make_divisible(c2 * gw, 8)
323
324     args = [c1, c2, *args[1:]]
325     if m in {BottleneckCSP, C3, C3TR, C3Ghost, C3x}:
326         args.insert(2, n) # number of repeats
327         n = 1
328 elif m is nn.BatchNorm2d:

```

Figure 21 – Add Attention mechanism class name


```

1 # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3 # Parameters
4 nc: 3 # number of classes
5 depth_multiple: 0.33 # model depth multiple
6 width_multiple: 0.50 # layer channel multiple
7 anchors:
8   - [10,13, 16,30, 33,23] # P3/8
9   - [30,61, 62,45, 59,119] # P4/16
10  - [116,90, 156,198, 373,326] # P5/32
11
12 # YOLOv5 v6.0 backbone
13 backbone:
14   # [from, number, module, args]
15   [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17    [-1, 3, C3, [128]],
18    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19    [-1, 6, C3, [256]],
20    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21    [-1, 9, C3, [512]],
22    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23    [-1, 3, C3, [1024]],
24    [-1, 1, CBAM, [1024]], # CBAM注意力机制
25    [-1, 1, SPPF, [1024, 5]], # 9
26  ]

```

Figure 22 – yolov5s-CBAM neural network architecture backbone


```

27
28 # YOLOv5 v6.0 head
29 head:
30     [[-1, 1, Conv, [512, 1, 1],
31        [-1, 1, nn.Upsample, [None, 2, 'nearest']],
32        [[-1, 6], 1, Concat, [1]], # cat backbone P4
33        [-1, 3, C3, [512, False]], # 13
34
35        [-1, 1, Conv, [256, 1, 1],
36        [-1, 1, nn.Upsample, [None, 2, 'nearest']],
37        [[-1, 4], 1, Concat, [1]], # cat backbone P3
38        [-1, 3, C3, [256, False]], # 17 (P3/8-small)
39
40        [-1, 1, Conv, [256, 3, 2],
41        [[-1, 15], 1, Concat, [1]], # cat head P4
42        [-1, 3, C3, [512, False]], # 20 (P4/16-medium)
43
44        [-1, 1, Conv, [512, 3, 2],
45        [[-1, 11], 1, Concat, [1]], # cat head P5
46        [-1, 3, C3, [1024, False]], # 23 (P5/32-large)
47
48        [[18, 21, 24], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
49    ]
50

```

Figure 23 - yolov5s-CBAM neural network architecture head

```

1  # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3  # Parameters
4  nc: 3  # number of classes
5  depth_multiple: 0.33  # model depth multiple
6  width_multiple: 0.50  # layer channel multiple
7  anchors:
8      - [10,13, 16,30, 33,23]  # P3/8
9      - [30,61, 62,45, 59,119]  # P4/16
10     - [116,90, 156,198, 373,326]  # P5/32
11
12  # YOLOv5 v6.0 backbone
13  backbone:
14      # [from, number, module, args]
15      [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16       [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17       [-1, 3, C3, [128]],
18       [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19       [-1, 6, C3, [256]],
20       [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21       [-1, 9, C3, [512]],
22       [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23       [-1, 3, C3, [1024]],
24       [-1, 1, CoordAtt, [1024]], # CoordAtt注意力机
25       [-1, 1, SPPF, [1024, 5]], # 9
26  ]

```

Figure 24 - yolov5s-CoordAtt neural network architecture head

```

1 # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3 # Parameters
4 nc: 3 # number of classes
5 depth_multiple: 0.33 # model depth multiple
6 width_multiple: 0.50 # layer channel multiple
7 anchors:
8   - [10,13, 16,30, 33,23] # P3/8
9   - [30,61, 62,45, 59,119] # P4/16
10  - [116,90, 156,198, 373,326] # P5/32
11
12 # YOLOv5 v6.0 backbone
13 backbone:
14   # [from, number, module, args]
15   [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17    [-1, 3, C3, [128]],
18    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19    [-1, 6, C3, [256]],
20    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21    [-1, 9, C3, [512]],
22    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23    [-1, 3, C3, [1024]],
24    [-1, 1, SELayer, [1024]], # SENet注意力机制
25    [-1, 1, SPPF, [1024, 5]], # 9
26  ]
27

```

Figure 25 - yolov5s-SENet neural network architecture head

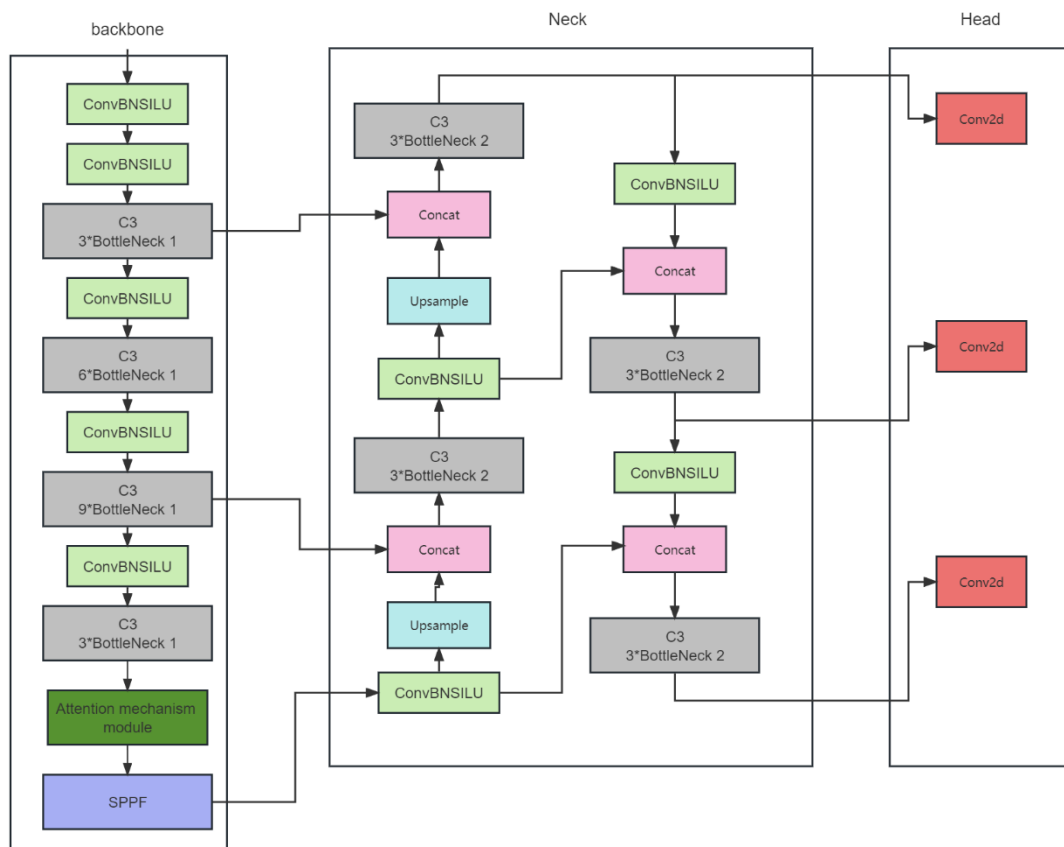


Figure 26 – The yolov5s network structure of adding attention mechanism

```
Fusing layers...
YOLOv5s summary: 157 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
Class      Images  Instances   P      R    mAP50  mAP50-95: 100%| 8/8 [00:00<00:00, 8.76it/s]
all         60       67    0.809   0.806   0.848    0.527
dirt         60       21    0.814   0.81    0.856    0.492
fire         60       25    0.812   0.8     0.839    0.583
steam        60       21    0.801   0.81    0.849    0.506
Results saved to runs\train\exp
```

Figure 27 – Training results of YOLOv5s

```
Fusing layers...
YOLOv5s-CBAM summary: 168 layers, 7019338 parameters, 0 gradients, 15.8 GFLOPs
Class      Images  Instances   P      R    mAP50  mAP50-95: 100%| 8/8 [00:00<00:00, 8.67it/s]
all         60       67    0.896   0.833   0.883    0.516
dirt         60       21    0.864   0.81    0.836    0.468
fire         60       25    0.824   0.88    0.894    0.548
steam        60       21     1     0.811   0.918    0.53
Results saved to runs\train\exp
```

Figure 28 – Training results of YOLOv5s-CBAM

```

YOLOv5s-CoordAtt summary: 167 layers, 7043864 parameters, 0 gradients, 15.8 GFLOPs
Class      Images  Instances   P      R    mAP50  mAP50-95: 100%| 8/8 [00:00<00:00, 9.43it/s]
all         60         67    0.864    0.82    0.852    0.525
dirt         60         21    0.833    0.81    0.845    0.534
fire         60         25    0.822    0.84    0.865    0.56
steam        60         21    0.937    0.81    0.846    0.482
Results saved to runs\train\exp

```

Figure 29 – Training results of YOLOv5s-CoordAtt

```

YOLOv5s-SENet summary: 164 layers, 7019240 parameters, 0 gradients, 15.8 GFLOPs
Class      Images  Instances   P      R    mAP50  mAP50-95: 100%| 8/8 [00:00<00:00, 9.78it/s]
all         60         67    0.891    0.874    0.861    0.507
dirt         60         21    0.819    0.81    0.776    0.462
fire         60         25    0.853    0.88    0.85    0.512
steam        60         21    0.932    0.957    0.957    0.546
Results saved to runs\train\exp

```

Figure 30 – Training results of YOLOv5s-SENet

List of Tables

Total image	Training Dataset	Test Dataset	Validation Dataset
1088	969	59	60

Table 1. The details of Dataset

Model	Size (pixels)	mAP^{val}_{50-95}	mAP^{val}_{50}	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	Params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.6
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Table 2. YOLOv5 pre-trained network model (Jocher 2020)

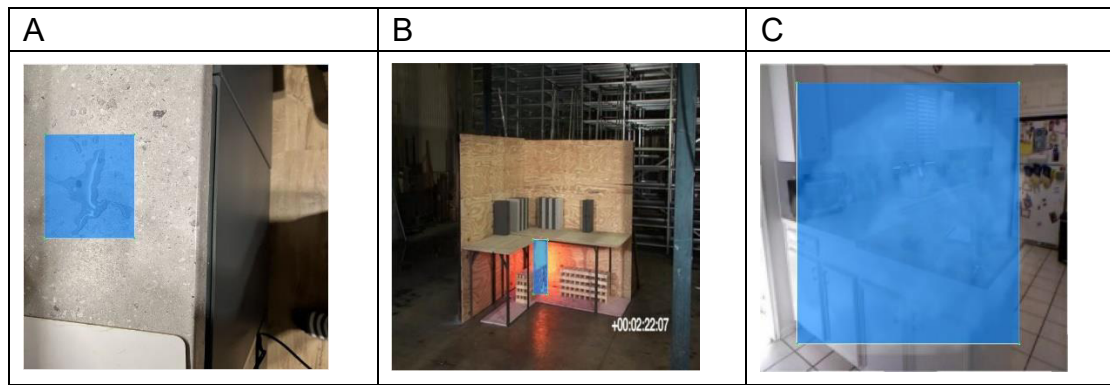


Table 3. Pictures of three kinds of objects

Augmentation	Description
Flip	Horizontal, Vertical
Rotation	Between -15° and $+15^{\circ}$
Brightness	Between -25% and $+25\%$

Tabel 4. Data-Augmentation used in this article

Name	Value
depth_multiple	0.33
width_multiple	0.50
initial learning rate	0.01
final learning rate	0.01
Stochastic Gradient Descent (SGD)	0.937
Weight decay for the optimizer	0.05
Warmup epochs	3
initial momentum during warmup	0.8
Training epochs	200
batch-size	4
photo size	640×640 pixels

Table 5 – The hyperparameters set for training

Name	P	R	F1	mAP50	mAP50-95
YOLOv5s	0.809	0.806	0.807	0.848	0.527
YOLOv5s-CBAM	0.896	0.833	0.863	0.883	0.516
YOLOv5s-CoordAtt	0.864	0.82	0.841	0.852	0.525
YOLOv5s-SENet	0.891	0.874	0.882	0.861	0.507

Table 6 – The training results for all models

1. Introduction

1.1 Overview

Due to its many applications and recent technological advances, object detection has received a lot of attention recently. It has been applied in numerous real-world contexts, including home security (Surantha & Wicaksono, 2018), autonomous driving (Feng et al. 2021), traffic information (Asmara et al., 2020), and robot vision (Ekvall et al., 2007). The discipline of object detection has made substantial use of deep learning models at the same time because of the development of deep convolutional networks and the development of GPU with a large number of parallel threads and multiple cores (Nickolls & Dally, 2010). Deep learning network object detection tasks are commonly classified into two types: detectors with two stages and detectors with single stage. Fast R-CNN (Girshick, 2015) represents two-stage detectors, and YOLO (Redmon et al. 2015) represents single-stage detectors. Based on the YOLOv5 method, this dissertation suggests a detection strategy for objects without a defined shape. In order to improve the resilience of the model under the influence of many factors and avoid overfitting, the data augmentation approach is used initially to enhance and expand the dataset. It also increases the sample size and complexity. The improved modules are depicted in the following manner: by implementing an attention mechanism layer to the framework's underlying neural network structure to extract key features, The model has the capacity to allocate greater attention to the goal and effectively execute picture recognition tasks. These three attention mechanism modules are CBAM (Woo et al. 2018), CoordAtt (Hou et al. 2021) and SENet (Hu et al. 2017). The three improved methods proposed in this research can better recognize items without definite forms, are more accurate than the YOLOv5 detection algorithm's native algorithm, and can detect and position objects without fixed forms, according to experiments.

1.2 Challenging

Previous studies in the discipline of object identification have the majority of focused on utilizing detection algorithms to identify items having a generic shape. The experimental utilization of the YOLOv5 method, designed for object detection tasks involving objects lacking stable shapes, yields suboptimal accuracy and precision. Certain flames can manifest as diminutive targets in everyday situations, perhaps accompanied by a smoky periphery that hampers their discovery. The detection algorithm frequently encounters challenges in accurately detecting objects that lack stable forms, mostly due to characteristics such as color, shape, noise interference, and background influences. The incorporation of the attention mechanism module in the YOLOv5 algorithm aims to address the primary issue of low level of precision in detection. This enhancement enables the system to effectively recognize objects that lack stable forms.

1.3 Aims and Objectives

In everyday life, numerous tasks involving object recognition rely on commonly encountered physical objects. The inherent stability of real objects facilitates the extraction of pertinent characteristics by neural networks, hence enhancing the learning process. Nevertheless, when dealing with objects that lack a defined shape, such as flames, water spots on the floor, and steam, it becomes challenging to extract distinctive characteristics due to their inherent shape variability. The objective of this study is to investigate techniques for effectively capturing the attributes of the aforementioned entity within neural network, with the aim of enhancing the precision of detection.

Three primary objectives were identified:

The YOLOv5 object recognition framework is a widely used system for detecting and classifying objects in images and videos.

The attention mechanism is a computational mechanism that enables a model to choose concentrate on specific segments of input data, hence augmenting its capacity to analyse and comprehend information.

One potential enhancement for the YOLOv5 architecture involves the integration of an attention mechanism.

1.4 Scope

The objective of this project is to enhance a deep learning-based framework for object detection, with the primary aim of assessing the accuracy of detection tasks. This study aims to assess the efficacy of three distinct attention processes in the identification of amorphous objects throughout both training and detection phases. The objective is to identify the most optimal enhancement strategy for this task.

The object detection framework YOLOv5 and the attention module are established technologies that have been implemented with open-source code and development kits. Hence, it is imperative for this project to deliberate upon the optimal level inside the neural network at which the inclusion of the attention module would provide enhanced accuracy. Additionally, it is essential to ascertain the appropriate method of incorporating the attention module into YOLOv5 through the implementation of code.

1.5 Learning Outcomes

The learning outcomes may be succinctly summarized as follows:

The objective of this inquiry is to get comprehension regarding the detection and training procedures employed by the YOLOv5 object detection algorithm.

Comprehend the structural framework and functional significance of the underlying neural network in YOLOv5.

The comprehension of the underlying rationale behind the attention mechanism technique in the realm of deep learning is crucial.

The development of the underlying neural network is carried out with open-source software.

The aim of this analysis is to assess the viability of the enhanced YOLOv5 algorithm.

Provide suggestions for enhancing future detection methodologies of amorphous entities.

2. Background

2.1 Wider Context

The YOLO algorithm has gained significant popularity as an object identification method due to its ability to do real-time and rapid detection. The premise of YOLO pertains to the regression problem, wherein it aims to circumvent the conventional object detection algorithm utilization of sliding windows and region extraction procedures. Due to its superior accuracy and efficiency, YOLOv5 has gained significant popularity among users, following the successive iterations of the YOLO framework. YOLOv5 algorithm has demonstrated favorable performance and accuracy in several research investigations. However, its effectiveness can be further enhanced when confronted with intricate circumstances, including amorphous or irregular shapes, as well as other visually demanding problems like flame, steam, and water stains. The problem can be effectively addressed through the utilization of methods that rely on attentional mechanisms.

Attention mechanisms have emerged as a result of the integration of neural networks into natural language processing. These mechanisms enable models to effectively analyze textual material by assigning varying weights to inputs based on their relative significance. Consequently, the model may concentrate its processing power on the most essential features of the data. The efficacy of

the attention mechanism has been demonstrated in several computer vision tasks, such as classification (Obeso et al. 2022), semantic segmentation (Wu et al. 2021), and target recognition (Qiu et al. 2021).

A novel network model is proposed by integrating YOLOv5 with the attention mechanism. This model aims to enhance the detection performance of shapeless objects by effectively focusing on the crucial regions of the input image. The study incorporates the Convolutional Block Attention Module (Woo et al. 2018), Coordinate Attention (Hou et al. 2021), and Squeeze-and-Excitation Network (Hu et al. 2017) separated into the network architecture of YOLOv5. Experimental evaluations are undertaken to validate the efficacy of these modules. Furthermore, this methodology has the potential to be advantageous in other visual applications, including the real-time monitoring of amorphous entities and the delineation of amorphous entities inside images. In general, the YOLOv5 algorithm, when incorporating the attention mechanism, exhibits significant promise and holds substantial application value.

2.2 Identified Problem Area

The detection of shapeless or irregularly formed objects, such as smoke, clouds, and fires, poses a challenging problem within the domain of computer vision (Han et al. 2021). Accurate detection of these items poses a challenge due to their lack of set shapes or defined borders. Despite the notable accomplishments of the YOLOv5 algorithm in target detection, there is ample opportunity for enhancing the detection of amorphous objects. The attention mechanism is a valuable tool that enhances a model's ability to concentrate on significant portions within a picture. The challenge at hand involves finding an appropriate approach to incorporate the attention mechanism into YOLOv5, hence enhancing its performance in detecting objects with amorphous shapes. The challenge associated with training data lies in the presence of diverse and

irregularly shaped objects, which poses difficulties in acquiring extensive and high-quality training data. Moreover, the process of assigning labels to such data is a laborious and challenging undertaking.

There exists a trade-off between the accuracy and efficiency of a model, as enhancing the model's performance necessitates careful consideration of its computing efficiency. The inclusion of the attention mechanism has the potential to result in a rise in the computational difficulty of the model. Therefore, it is necessary to figure out a balance between enhancing performance and maintaining efficiency, which is a matter of concern. The integration of the attention mechanism into YOLOv5 for network design and parameter tuning is a complex task that necessitates meticulous network structure design, extensive experimentation, and thorough parameter tuning. The technical task at hand involves the design and optimization of a structure to achieve optimal detection performance. The ability of a model to generalize: the extent to which the model can sustain strong generalization ability across a diverse range of amorphous objects, is an additional factor that warrants consideration.

2.3 History

Computer vision has emerged as a prominent area of research, with object detection being a significant and complex subfield within this domain. The primary objective of object detection is to accurately categorize various things. The initial proposition of the artificial neuron model as a foundational framework for deep learning was made by Pitts. This model elucidates the mechanisms by which neurons transfer and process information through the use of binary signals (McCulloch and Pitts 1943). The concept of convolutional networks originated during the 1960s when Hubel and Wiesel conducted a study that examined the visual pathway from the retina to the striated cortex and

investigated the functions of various brain units in visual perception (Hubel and Wiesel 1962). The basic objective of the convolutional layer in the object detection task is to extract essential information from the image. Simultaneously, the integration of low-level convolutional network and high-level convolutional network enables the extraction of significant characteristics from various levels within the image.

Object detection techniques for deep learning networks are divided into two categories: two-stage and single-stage. The category of two-stage detectors is exemplified by the Fast R-CNN (Girshick 2015), whereas the category of single-stage detectors is exemplified by the YOLO model (Redmon et al. 2015). Fast R-CNN, serving as an exemplar of a two-level detector neural network, initiates the production of prospective regions for the input image during the initial level. This process aims to diminish the quantity of regions that necessitate examination, hence enhancing the speed of detection. The second level of the system employs a convolutional neural network (CNN) to extract features from the output proposal; generated by the first level. These features are then processed using real-time object detecting system. In contrast to the conventional two-stage detector technique, YOLO employs a singular stage detector to accomplish the task of object detection. Firstly, the YOLO algorithm partitions the image into $S \times S$ grids. Within each grid, it predicts B bounding boxes together with their associated probabilities. Additionally, the algorithm predicts C conditional class probabilities. In the case of classes with low degrees, a class prediction is obtained for each grid.

2.4 Two-stage detector CNN

The fundamental principle underlying the two-stage detector involves the partitioning of acquired features into two distinct modules. The initial module, referred to as the Region Proposal Network (RPN), is a fully convolutional

network responsible for producing object recommendations. These proposals are subsequently fed into the second module, which utilizes a shared convolutional module to refine and enhance the proposals, resulting in a more comprehensive set of proposals. Simultaneously, the convolution layer that is shared among several networks can also produce object recommendations of superior quality in networks with significant depth (Jiang and Learned-Miller 2017). Li's study employed a two-stage detector, specifically the Fast R-CNN, to examine the distinct characteristics of pedestrian behavior and morphology across various scales. Additionally, the model incorporates multiple sub-networks to extract feature maps within specific input scale ranges through convolution layers. The sub-network's output process is both scale aware and weighted, leading to the attainment of precise pedestrian identification results (Li et al. 2017). The application of the two-stage detector Faster R-CNN has led to improved detection outcomes in identifying malaria cell images and detecting microscopic images of malaria-infected blood, despite the presence of uncertainty related to cell shape and density (Hung et al. 2018). In contrast to the single-stage detector, the two-stage detector has superior detection accuracy. The division of the prediction into two modules enhances the accuracy of the detection results.

2.5 Single-stage detector YOLOv5

The YOLO single-stage detector framework considers the object detection challenge as a regression problem. The assignment of regions inside the image to specific object classes is performed, and the corresponding probabilities of such assignments are computed. Zhang conducted a study wherein they enhanced a real-time detection model based on YOLOv5. They employed the Flip-mosaic data augmentation technique as a means to tackle the problem of sample imbalance, resulting in improved detection accuracy and reduced missed detection rate. The primary objective of this research was to achieve

real-time vehicle detection and monitoring (Zhang et al. 2022). This proposal presents valuable methodologies for the development of real-time intelligent transportation systems. The implementation of smartphone webcam-based pest detection in agriculture with deep learning as the underlying technology has also been observed (Ahmad et al. 2022). The implementation of these advancements has significantly improved the efficacy of crop production and pest management. The attention mechanism has been employed in industrial applications to enhance the processing of feature area channels in the image, detect and classify defects on the steel surface, and enhance the quality of steel within the enterprise (Shi et al. 2022). Furthermore, there exist instances where deep neural networks have been modified to enhance the precision of object detection. Lan introduced a modification to the YOLO network by incorporating three passthrough layers. These layers were designed to establish connections between high-resolution and low-resolution object features (Lan et al. 2018). Features were able to be more comprehensively passed to the deep neural network optimization algorithm. The objective of this modification was to enhance the accuracy of object detection, minimize missed detections, and reduce the rate of false detections. In contrast to the two-stage detector, the single-stage detector possesses a straightforward architecture that is better suited for the integration of additional technical modules. Moreover, the single-stage detector exhibits a significantly faster detection speed. The current detection rate has the capability to achieve up to 45 frames, making it particularly well-suited for tasks involving small-scale or real-time target detection.

2.6 Attention mechanism

The attention mechanism is a data processing technique utilized in the field of machine learning. Within the domain of computer vision, the primary principle underlying attention mechanisms is to facilitate the model in developing the

capacity to focus on relevant information while discarding irrelevant elements. The CoordAtt mechanism is a hybrid attention mechanism that aims to capture the important link between channel and space (Hou et al. 2021). By performing calculations, CoordAtt enhances the model's understanding of the relationship between space and channel features, hence improving feature capture. The CBAM (Woo et al. 2018) method is a hybrid attention mechanism that consists of two sub-modules: channel module and spatial module. These sub-modules are designed to enhance the network's capacity to concentrate on the salient characteristics of input data.

Additionally, CBAM incorporates global pooling to reinforce the representation of crucial information. SENet, as a model incorporating channel attention mechanism, is fundamentally based on the concept of squeezing and excitation (Hu et al. 2017). Using dimension reduction techniques to get the features of low-level channels. Subsequently, the low-level importance information was reintroduced into the original channel through dimension enhancement. This process aimed to enhance the linkages between channels and improve the model's ability to communicate information. The attention mechanism can serve as an efficient strategy for allocating computer resources, enabling the processing of more crucial information within the constraints of low computing capacity (Niu et al., 2021). In Sun's research, a hybrid attention mechanism in their study to optimize the object detection model. This mechanism incorporates local and global feature fusion techniques to enhance the expression capacity of the feature map and enhance the accuracy of both object recognition and background recognition (Sun et al. 2023). The integration of attention mechanism in the object detection framework has proven to be effective in dealing with complicated environments that are characterized by fuzziness and object density (Zhu et al. 2021).

2.7 Capture Feature Information

Some studies have demonstrated that the inclusion of crucial feature information can enhance the efficacy of object detection. The initial object detector demonstrates efficacy in detecting common things as well as objects of substantial size. The original detector exhibits a significant decrease in its efficacy when confronted with small objects and certain objects characterised by intricate patterns and backgrounds. Enhancing the extraction of feature information on small objects by developing an efficient feature extraction module. Their objective was to increase the acquisition of deep semantic features, minimise the variation in feature capture across different scales, and ultimately enhance the performance of object detection (Li et al. 2022). Zheng's study proposes a network architecture known as the feature enhancements fusion network. This network is specifically built to convolve multi-level features, hence enabling the fusing of multi-level feature maps to enhance the original features. The primary objective of this enhancement is to enable accurate object detection in complicated and dense environments, even when objects are oriented in various directions. The network model can be enhanced in its learning of object features through the construction of multiple feature information capturing modules (Zheng et al. 2022). According to Cheng's research, the network's capacity is enhanced through the incorporation of a module that captures multiple features. This enhancement enables the network to acquire a deeper understanding of the region of interest, as well as strengthen its ability to learn the contextual features of the image, including the background and objects that the detector aims to identify. Consequently, the module's integration reduces the impact of complex scenes on the accuracy of object detection in optical remote sensing images (Cheng et al. 2021).

3. Specification and Design

3.1 Requirement Analysis

Functional Requirements

1. Shapeless object detection: The improved YOLOv5 should be able to accurately detect shapeless objects such as flame, steam, water stains.
2. Algorithm integration: The attention mechanism should be able to seamlessly integrate into the YOLOv5 algorithm without affecting its basic workflow.

Non-functional Requirements

1. Portability: The algorithm should be able to run efficiently on multiple hardware platforms, including GPUs and CPUs, to meet different application scenarios.
2. Ease to use: The algorithm should provide easy-to-use interfaces for use in a variety of computer vision applications.
3. Robustness: The algorithm should be able to work effectively in a variety of complex environments and lighting conditions.

Performance Requirements

1. Accuracy: The algorithm should achieve accuracy better than or equal to current leading algorithms on standard datasets for shapeless object detection.

Constraints

1. Resource constraints: possible hardware and computational resource constraints, such as limited GPU memory, are taken into account during the development and testing phases.
2. Time constraints: The project may have a certain development and delivery timeline.

3.2 Design goals and objectives

When formulating a project utilizing the CBAM, SENet, and CoordAtt attention mechanisms to enhance the YOLOv5 algorithm's capability in detecting

amorphous objects, the potential design aims and objectives can be outlined as follows.

1. Improving the detection efficacy of amorphous objects.

The main aim of this study is to enhance the YOLOv5 algorithm's capability to accurately and effectively detect amorphous objects, such as flames, water stains, and steam. The objective might be accomplished by the incorporation of additional network layers or by enhancing the current network architecture. Subsequently, the parameters acquired during the experiment are examined to determine if this enhancement can effectively yield the intended outcomes.

2. The incorporation of attention mechanisms.

The core technology goal of this study is to incorporate attention mechanisms within the YOLOv5 algorithm in order to enhance the network's focus on crucial components of objects. This integration aims to enhance the accuracy of recognising objects with amorphous shapes. The achievement of this objective can be facilitated through the use of diverse attention models, including the Convolutional Block Attention Module (CBAM), Squeeze-and-Excitation Networks (SENet), and Coordinate Attention (CoordAtt).

3. The topic of concern is the maintenance of performance.

The design objective is to take into consideration the notable real-time performance of the YOLO family of algorithms, which is regarded as one of their primary advantages. After modifying the network architecture, it is imperative to ensure the preservation of, or at the very least, minimise any substantial decline in performance. The design and implementation process may necessitate the examination of computational complexity and the utilisation of optimisation methodologies.

4. Guarantee the algorithm's adaptability and portability.

The design objectives should take into account the algorithm's adaptability and portability, enabling it to efficiently operate on diverse hardware platforms and effectively process objects of varying shapes.

5. Provision of assistance for the training and evaluation of models.

The algorithm should possess the capability to facilitate both model training and evaluation on diverse object identification datasets with irregular shapes. This would enable users to customise the model to suit their own requirements.

3.3 Design method

In order to develop an enhanced YOLOv5 algorithm that incorporates attention mechanisms such as CBAM, SEnet or CoordAtt for the detection of amorphous objects, the following methodologies may be employed.

Gaining Proficiency and Acquaintance with the YOLOv5 Algorithm

To commence, it is important to get a comprehensive comprehension of the operational principles and structural components of the YOLOv5 algorithm. By means of diligent examination and practical application, one can acquire comprehension of the primary benefits offered by YOLOv5 and its utilization in the realm of target identification.

Comprehending Attention Mechanisms

Subsequently, it is imperative to comprehend the underlying mechanics of attention, specifically focusing on models such as CBAM, SEnet, and CoordAtt. It is imperative to get a comprehensive understanding of the mechanisms by which model performance is enhanced as well as the methods for effectively integrating these improvements into the YOLOv5 framework.

Collect and preparation of the dataset.

The objective is to gather a dataset including visual representations of flames, water stains, and steam, which will be utilized for the purpose of training and detecting a deep learning model. The dataset is preprocessed through the utilization of picture annotation techniques for annotation purposes, as well as

data augmentation methods to enhance the sample size and mitigate algorithm overfitting.

The process of developing the network architecture.

After ensuring the dataset is prepared for labelling and acquiring a comprehensive understanding of these fundamental concepts, we may proceed with the development of the novel network architecture. The determination of the appropriate layers for the insertion of the Attention Module and the adjustment of network parameters are essential considerations for achieving optimal performance.

The execution of the model.

Once the design has been established, the next step is to proceed with the coding phase in order to implement our model. This may entail leveraging deep learning expertise or utilising an open-source package or code.

The process of training and tuning

Subsequently, it is imperative to conduct model training on a designated training dataset, followed by the refinement and optimisation of the model's parameters through the utilisation of a validation dataset. This process can undergo multiple iterations and experiments.

Assessing Performance

Ultimately, it is imperative to assess the efficacy of our model on the designated test dataset. The success of our methodology can be assessed by the utilisation of diverse indicators, including mean Average Precision (mAP), Frames Per Second (FPS), and others.

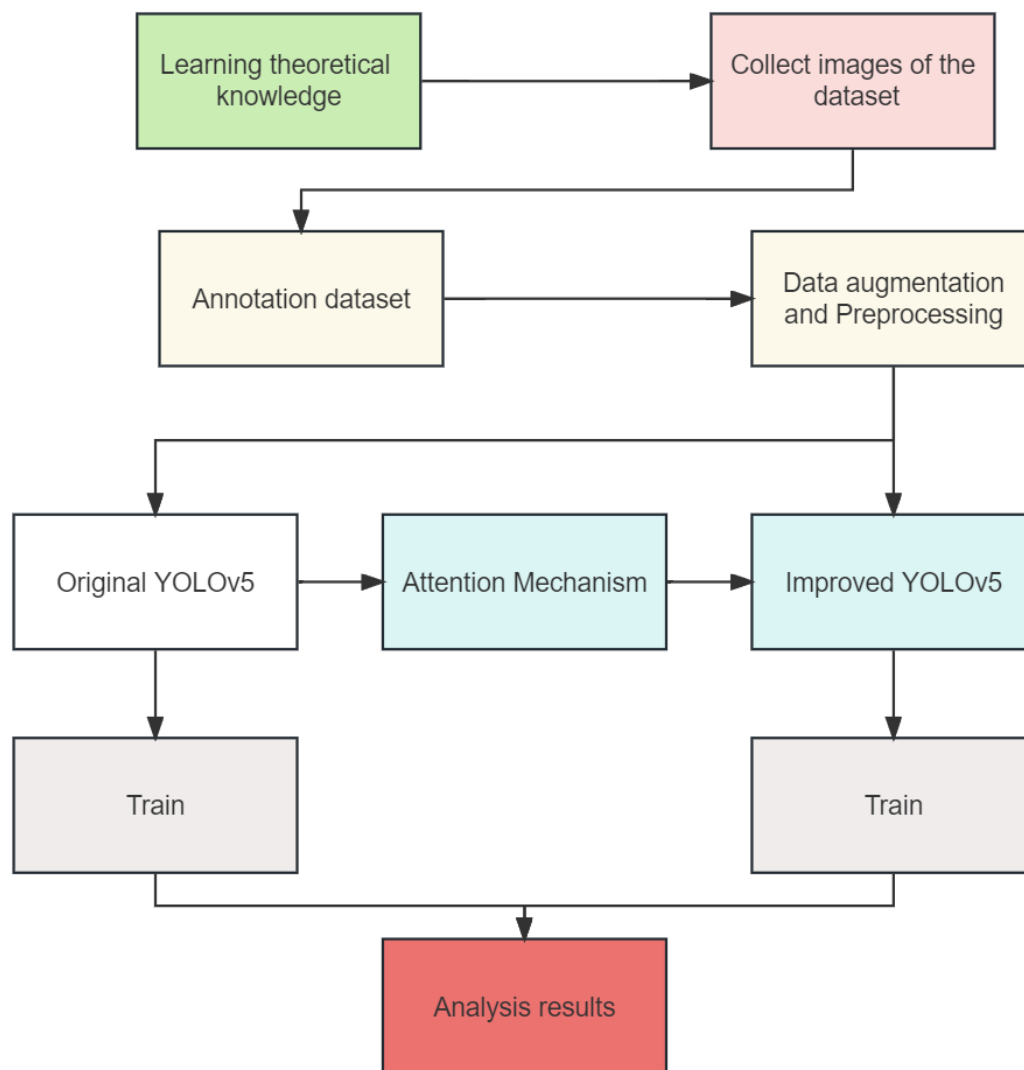


Figure 31-Flowchart of the design method

3.4 Data Collection

The source of the data set in this paper can be divided into two parts, one is from the public data set with license CC BY 4.0 on the Internet, and the other is photos taken using smartphones. The details of the public dataset can be found in the README.md file under dataset7.15 of the code. A collection of 450 distinct image sets was initially available, which were subsequently augmented using data augmentation techniques resulting in a total of 1088 photos. Aggregation of diverse datasets Please refer to the table 1 provided below for additional details:

Total image	Training Dataset	Test Dataset	Validation Dataset
1088	969	59	60

Table 1. The details of Dataset

3.5 Data Preprocessing

The YOLOv5 algorithm includes a variety of network models, and the specific conditions and various parameters are shown in Table 2. In order to achieve high-performance real-time classification on small devices, this study chooses YOLOv5s as the network model for experimental training from the perspective of minimizing the computational cost.

Model	Size (pixels)	mAP^{val}_{50-95}	mAP^{val}_{50}	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	Params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.6
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Table 2. YOLOv5 pre-trained network model (Jocher 2020)

YOLOv5s is the smallest of the YOLOv5 family of models, and it has a significant advantage in terms of runtime speed over other large models such as YOLOv5l and YOLOv5x. For real-time object detection tasks, YOLOv5s provides an important baseline, and because of the YOLOv5s model is smaller, it is easier to understand and modify. This is important for research and test new ideas and techniques without requiring significant computational resources

and time. This is the reason why YOLOv5s was chosen as the basic network model for my dissertation.

Image annotation was performed by labellmg (Tzutalin 2015) The unified management of the dataset and data enhancement techniques are all performed by the Roboflow (Roboflow: Give your software the power to see objects in images and video 2023) framework. Accurate image annotation can improve the accuracy of the YOLO object recognition algorithm. In Table 3, Figure A is the labeled water stain image on the floor, Figure B is the labeled flame image, and Figure C is the labeled steam image.




A	B	C
		

Table 3. Pictures of three kinds of objects

Data augmentation approaches enhance the quality of model training by increasing the number and diversity of datasets, by altered replicas of preexisting data, or generating novel synthetic data based on existing data, such as brightness adjustment, image inversion, adding noise, etc. Data augmentation enables algorithm generalization in time by reducing overfitting on the dataset. The generalization ability of neural networks can be improved using data augmentation techniques (Morabbi et al. 2023). Since this research design does not have object detection with fixed morphological objects, we use some transformations to simulate pictures of objects in complex environments. In Table 3, the following methods are used for image enhancement.

Augmentation	Description
Flip	Horizontal, Vertical
Rotation	Between -15° and $+15^{\circ}$
Brightness	Between -25% and +25%

Tabel 4. Data-Augmentation used in this article

3.6 Architecture

3.6.1 YOLOv5

The YOLO algorithm, devised by Joseph Redmon, is a single-stage detector algorithm. The utilisation of global features entails an end-to-end methodology wherein the direct prediction of both the object bounding box and class is achieved. This approach diverges from prior two-stage detector networks that necessitate feature extraction across many stages. The YOLOv5 algorithm was initially introduced by Glenn Jocher in the year 2020. When compared to alternative object detection algorithms, YOLOv5 has superior inference speed and detection accuracy. The YOLOv5 network model is designed for picture pixel 640 and consists of five variants, namely YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. This study aims to boost the accuracy of object detection for objects without fixed forms by improving the YOLOv5s method, taking into account the performance of the machine. The algorithmic architecture of YOLOv5 comprises three levels, including Backbone, Neck, and Head layers (Ultralytics 2023). Figure 2 illustrates the algorithmic framework of YOLOv5.

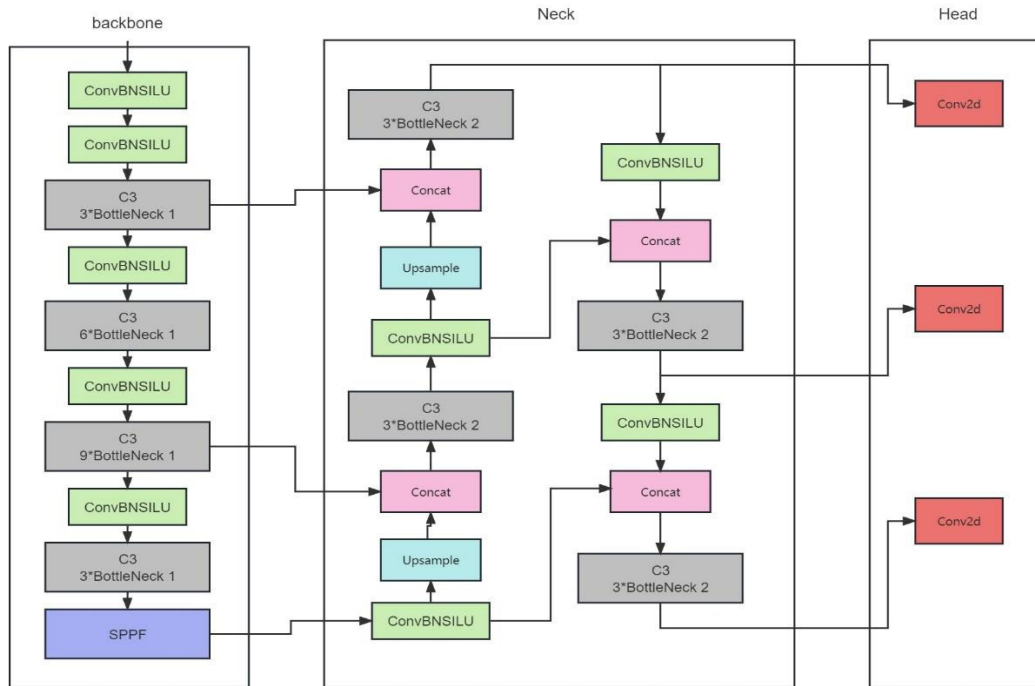


Figure 32-Basic Neural Network Architecture (Ultralytics 2023)

1. Module Explanation

ConvBNSILU layer is a module of YOLOv5, which is divided into three parts to operate, namely Conv, BN and SILU. Conv refers to Convolution, which serves as a crucial component in feature extraction and spatial feature learning from the input data. Batch Normalisation (BN) is an optimization approach employed in neural networks to enhance the efficiency of training processes. The Sigmoid Linear Unit (SILU) is a type of nonlinear activation function. The primary objective of this component is to provide nonlinearity into the model, hence enabling the model to acquire a deeper understanding of intricate mapping functions. Based on the findings presented in Figure 2, it is evident that the module in question holds significant prominence within the YOLOv5 framework. Specifically, this module plays a pivotal role in both feature extraction and feature conversion processes.

BottleNeck2 at C3 Y module shows that the BottleNeck2 module is used Y times in a C3 module. In a BottleNeck module, the number of channels is first reduced

by convolution, and then the features are extracted by 3×3 convolution, and the number of channels is doubled. The network structure known as the bottleneck is depicted in Figure 3.

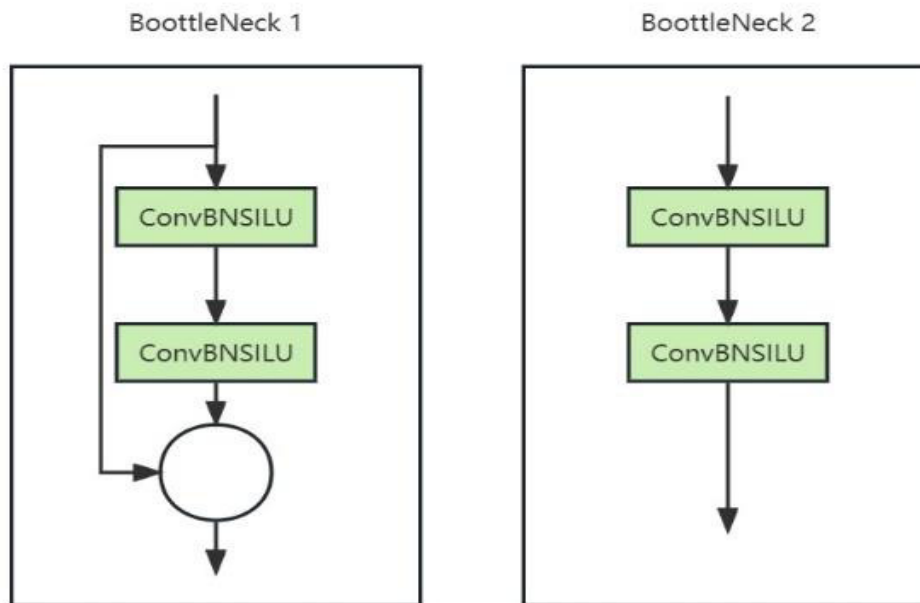


Figure 33-The structure of Bottleneck (Ultralytics 2023)

C3 module is the main idea of the Bottleneck module is to split the feature map into two parts, one part is directly connected to the next layer, and the other part is used for the subsequent convolution of the residual connection. In this way, the neural network layer can directly learn the gap with the previous layer instead of learning a new output. The network structure of the C3 module is seen in Figure 4.

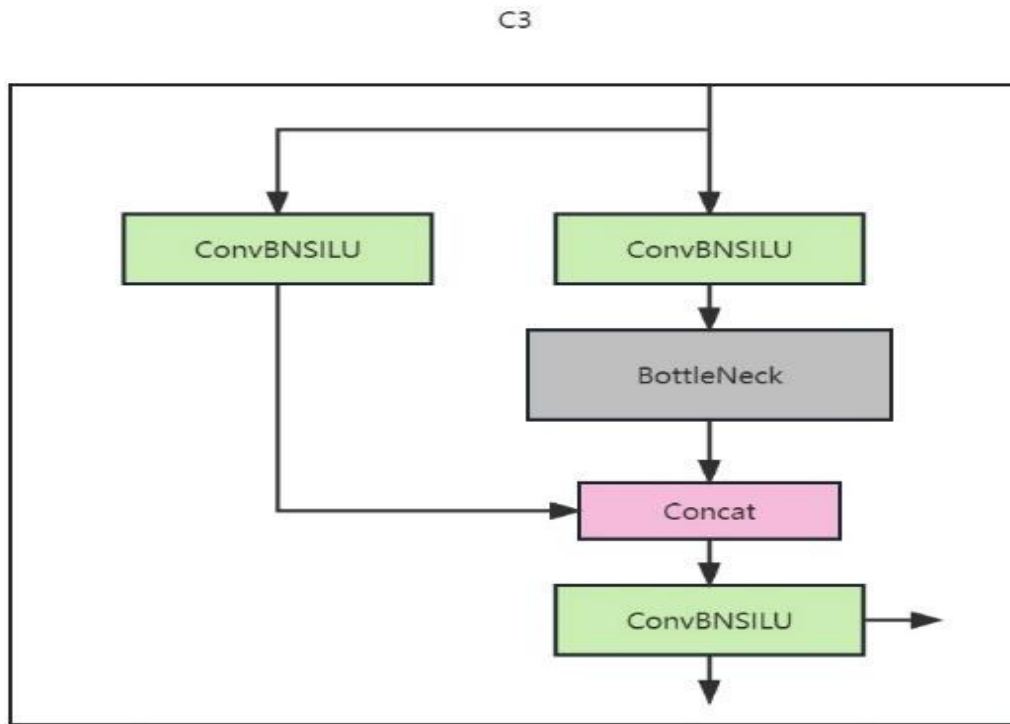


Figure 34-The structure of C3 (Ultralytics 2023)

2. Backbone

The backbone of the YOLOv5 is designed based on Convolutional Neural Network (O'Shea and Nash 2015). As one of the most popular neural networks, Convolutional layers, Nonlinear layers, Pooling layers and fully connected layers together make up it. (Albawi et al. 2017). The primary function of this design is to extract feature information from the input image, which will then be utilized for the purpose of target detection. In YOLOv5, its backbone network is CSPDarknet53 (Bochkovskiy et al. 2020), The proposed model is built upon an enhanced iteration of Darknet 53, incorporating a Cross Stage Partial (CSP) architecture to enhance the network's efficacy and overall performance. CSPDarknet53 introduces the CSP structure on top of Darknet53. The CSP structure is a novel network structure design that divides the feature map of the network into two-part, one part is directly down sampled and passed to the next layer, while the other part goes to the remaining layers for computation. This design effectively reduces the number of features maps and reduces the

computational complexity while maintaining the richness of features (Fu et al. 2021).

In the backbone network, the input image undergoes several operations such as convolution, pooling, and activation function, each of which generates a new feature map. Through these operations, the network is able to extract multi-scale and multi-level feature information from the input image. The experiment focuses on the neural network at the backbone level and the attention technology module is expected to be added in the backbone layer.

3. Neck

The Neck structure of YOLOv5 uses Feature Pyramid Network and Path Aggregation Network. The FPN adopts a top-down design to generate semantic feature maps across various scales, thereby mitigating the drawbacks associated with computationally demanding and memory-intensive processes (Lin et al. 2016). The PAN architecture is a network design that improves the flow of information in instance segmentation, reducing the distance between the lower and higher levels of information and establishing connections between the feature network and the feature level, enables the direct transmission of important information from each level to subsequent sub-networks (Liu et al. 2018). The proposed PAN framework incorporates a bottom-up bidirectional path in addition to the existing FPN architecture. This modification aims to enhance the performance of the model in obtaining more accurate results. Hierarchical traits are present. The integration of residual connections and the utilisation of an adaptable pyramid structure in YOLOv5 lead to improved acquisition of feature maps for objects with varying scales.

4. Head

The Head architecture in YOLOv5 is commonly employed as the concluding component of the network, responsible for extracting and integrating the feature maps required for the ultimate target prediction. The primary function of the

Head component is to accurately classify the object's class and determine the precise location of the bounding box. This enables the detection of objects of varying sizes across multiple scales. The Head module comprises several convolutional layers, with each layer dedicated to a certain scale. These layers provide feature maps of varying scales, where each feature map corresponds to a particular object class or bounding box characteristics, such as centre location, height, and width. The incorporation of this architecture enhances the multi-scale detection capability of YOLOv5. During the training process of YOLOv5, the predicted labels are compared with the actual target labels.

3.6.2 Attention mechanisms

The attention mechanism is a technological approach employed for data processing, which involves the cognitive ability to imitate human behaviour. The inclusion of an attention module within a neural network has been shown to enable selective processing of selected regions of an image, hence reducing the need for the full image to be processed (Ghaffarian et al. 2021). The attention mechanism allocates more weights to significant channels and lesser weights to inconsequential channels, so directing the network model's attention towards channels that possess crucial properties. This research presents a proposal for the integration of three distinct attention mechanisms inside the detection framework, with the aim of enhancing its performance.

1. Convolutional Block Attention Module (CBAM)

Convolutional Block Attention Module (Woo et al. 2018), as a hybrid attention mechanism. As shown in the figure 5, there are two sub-modules called channel module and spatial module which can facilitate the neural network in enhancing its attention towards the significant characteristics of the input data. The channel module focuses on discovering important feature channels, while the space module discovers important feature locations.

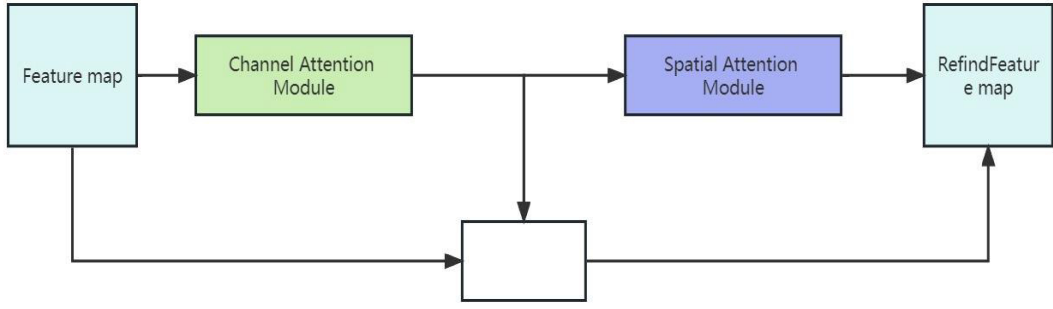


Figure 35-CBAM module

As shown in the figure 6, in the channel module, the global pooling operation as well as the Max pooling operation are first performed on the input feature map of the convolutional block, through a multi-layer perceptron (shared MLP) respectively generates the channel weights of average pooling and maximum pooling, and finally applies the two weights back to the original input feature map to strengthen the important channels and weaken the role of unimportant channels.

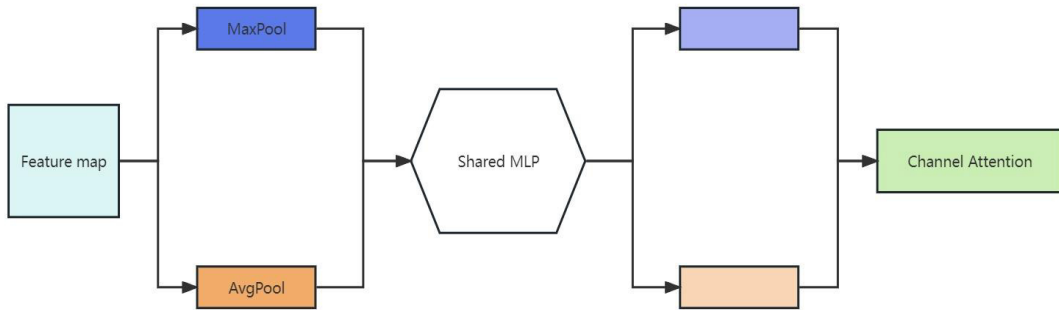


Figure 36-Channel module

The mathematical expression for the Channel module can be expressed as follows:

$$M_c = \sigma \left(MLP(AvgPool(F)) + MLP(MaxPool(F)) \right) \quad (1)$$

As shown in the figure 7, in the spatial module, the feature maps processed by the channel module are firstly averaged and Max pooled, and the spatial weights of the feature maps are generated through a convolutional layer. The

obtained spatial weights are applied to the original feature maps to strengthen the important feature map positions and weaken the unimportant positions.

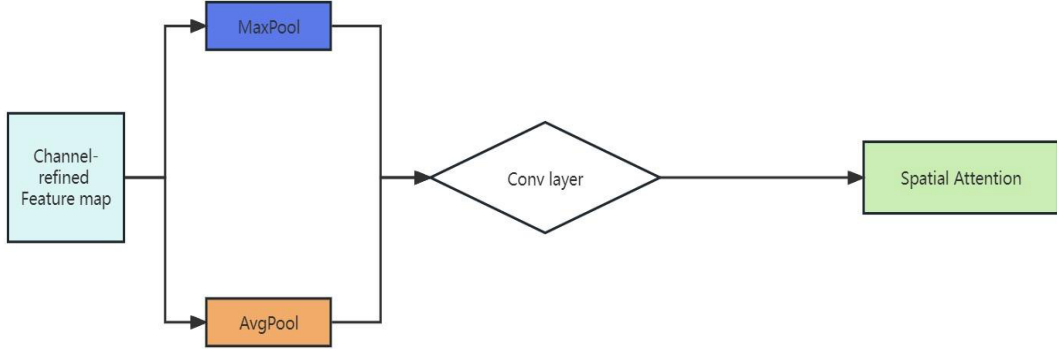


Figure 37-Spatial module

The mathematical expression for the Spatial module can be expressed as follows:

$$M_s(F) = \sigma \left(f^{7 \times 7}([AvgPool(F)]; MaxPool(F)) \right) \quad (2)$$

The acronym MLP refers to a fully connected layer, AvgPool denotes the global average pooling operation and MaxPool represents the global maximum pooling operation. The symbol σ is an activation function, ; is the operation by virtue of which the two pooled features are combined along the channel scale, $f^{7 \times 7}$ specifies a 7×7 convolution operation.

2. Coordinate Attention (CoordAtt)

Coordinate Attention (Hou et al. 2021), displayed in figure 8, is another hybrid attention mechanism that considers channel and position information. Coordinate attention initially encodes characteristics by global pooling from both x and y directions, then encoding of pooling is performed for each channel, considering both horizontal and vertical coordinates. This aids the attention module in effectively capturing spatial input by incorporating accurate location information. The network calculates feature coordinate attention to understand their spatial relationship. Finally, the computed coordinate attention is implemented on the input feature map, and the most important output strengthens significant features and weakens unimportant ones.

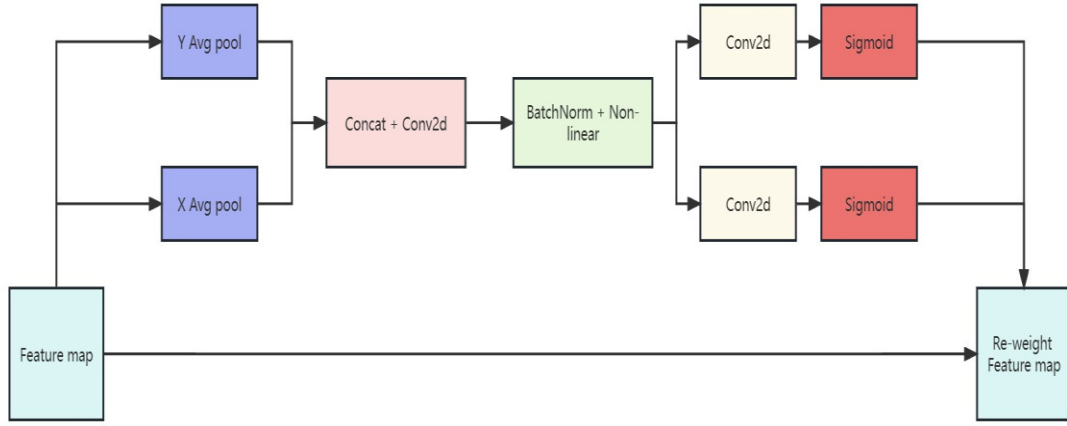


Figure 38-CoordAtt module

The expression of the output along the c -th channel at height h is provided in equation (3).

$$z_c^h(h) = \frac{1}{W} \sum_{0 \leq i \leq W} x_c(h, i) \quad (3)$$

The expression of the output along the width w of the c -th channel is provided in equation (4), These two approaches will acquire both global and positional information.

$$z_c^w(w) = \frac{1}{H} \sum_{0 \leq i \leq H} x_c(j, w) \quad (4)$$

The pooling outcomes from equations (3) and (4) are combined horizontally and vertically using the concatenation operation F1, which is a non-linear activation function.

$$f = \sigma(F1([z^h, z^w])) \quad (5)$$

The resulting connection is divided into two sets of feature maps based on their horizontal and vertical orientations.

$$f^h \in R^{C/r \times H} \text{ and } f^w \in R^{C/r \times W} \quad (6)$$

The weight data is obtained by applying the Sigmoid function to these feature maps.

$$g^w = \sigma(F_h(f^h)) \quad (7)$$

$$g^h = \sigma(F_w(f^w)) \quad (8)$$

By utilizing the horizontal and vertical weights, the coordinates of the attention output feature map are determined.

$$y_c(i, j) = x_c(i, j) + g_c^h(i) + g_c^w(j) \quad (9)$$

3. Squeeze-and-Excitation Network (SENet)

Squeeze-and-Excitation Network (Hu et al. 2017), as a representative of the attention model of the channel mechanism, the SE module proposed by SENet is shown in Figure 9. Following the convolutional layer, the extrusion operation is conducted by means of global average pooling, which serves to decrease the dimensionality to match the number of input feature channels, and again raises the dimension to the number of original input feature channels through excitation Importance. Finally, the output of the excitation operation is multiplied with the output of the original convolutional layer to draw the learned feature map, and the acquisition of the relationship and magnitude between several channels is achieved.

The mathematical expression for the Squeeze step of the c -th channel, when provided with the input X , can be represented as follows:

$$z_c = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W x_c(i, j) \quad (10)$$

where z_c represents the output corresponding to the c -th channel. The input X is obtained straight from a convolutional layer with a predetermined kernel size, therefore it may be regarded as a set of local descriptors. The variables H and W denote the dimensions of the compressed space.

Then this channel descriptors passes through a fully connected (FC) layer, commonly using ReLU and Sigmoid as activation functions, to obtain the weights of each channel. The formulation for the Excitation can be express as follows:

$$s_c = \sigma(\beta ReLU(\alpha z_c)) \quad (11)$$

α and β are the weight of the fully-connected layer. σ is the Sigmoid activation function.

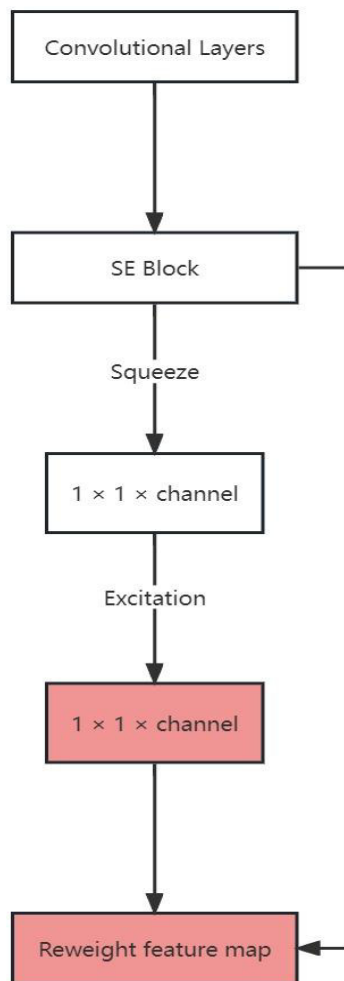


Figure 39-SENet module

3.7 Implementation Plan

Stage One: Learn the work

Read YOLOv5, CBAM, CoordAtt, and SENet literature. Understand the YOLOv5 principle and neural network design, then analyse each layer's role using references. The three attention mechanisms' properties and focus areas are evaluated when learning to analyse them. Know the principles of the three attention mechanisms.

Stage Two: Data prep and experiments

Select and compile a shapeless object detection dataset. After data labelling and preprocessing, the original YOLOv5 model can execute experiments, and the output performance benchmark can indicate if the data set can be used for the next experiment.

Stage Three: Integrate CBAM, SENet and CoordAtt attention mechanism modules to YOLOv5.

To ensure experiment fairness, three attention mechanisms were introduced to a neural network layer that was best for adding them. The redesigned model is trained, and hyperparameter optimisation improves performance.

Stage Four: Separate experiments and analyse them using feature parameters. Under the same hyperparameter configuration, the same training rounds are established, and precision, recall, and mAP value for IOU 0.5 are analysed individually. Performance comparison of the two algorithms.

Stage Five: Summarizing and Thesis.

Summarise and analyse how different attention mechanisms improve the YOLOv5 model based on experimental results.

3.8 Risks and Challenges

When doing research and implementing enhanced YOLOv5 algorithms for the detection of amorphous objects using attention mechanisms like CBAM, CoordAtt, and SENet, certain risks and obstacles may arise.

1. Model complexity: The inclusion of attention modules will result in an augmentation of the model's complexity. This phenomenon has the potential to result in an escalation in the duration of training, as well as an elevation in the demand for computational resources.

2. Risk of overfitting: The risk of overfitting arises when models become excessively complicated, resulting in a loss of generalizability. Additionally, the presence of inaccurately obtained datasets can also contribute to the occurrence of overfitting. The model exhibits high performance on the training set, but demonstrates suboptimal performance on the test set.
3. Choosing the right attention model: When selecting an attention model, it is important to consider many options such as CBAM, SENet, and CoordAtt. Each of these attention modules has its own distinct strengths and weaknesses. The selection of appropriate modules, as well as their location and parameterization within the network, is a significant difficulty.
4. Stability of model training: The stability of model training is a significant concern in the field of deep learning. The inclusion of additional attention modules has the potential to impact the stability of the training process. This may need the implementation of more refined optimisation algorithms and the adjustment of the learning rate.
5. Properties of shapeless objects: The detection of shapeless objects presents a problem because to their lack of set shapes and clear limits. Developing models that possess the ability to successfully address this circumstance represents a significant challenge.
6. Performance evaluation: The evaluation of performance: In light of the distinctive characteristics exhibited by amorphous entities, that may be imperative to devise innovative metrics and approaches to adequately evaluate the performance of the model. The lack of established evaluation methodologies presents a significant difficulty in accurately and impartially evaluating the efficacy of models.

4. Implementation

4.1 Tools

4.1.1 Tools

The following programming languages and applications were used in my dissertation to merge YOLOv5 and attention mechanism.

1. PyCharm Community Edition 2023.1.2 (PyCharm Community Edition and Professional Edition Explained: Licenses and More 2017)

PyCharm Community Edition 2023.1.2 (2017: Explaining Licences and More)
JetBrains' PyCharm is a Python IDE and integrated development environment. It offers entire Python application development tools. YOLOv5 is built on a Python algorithm, hence PyCharm is used as the programming environment and editor for eventual attention module integration. I also trained the neural network in PyCharm on a Windows 10 laptop. The community-based PyCharm is free, therefore all code is edited there.

2. Git (Spinellis 2012)

Git is a distributed version control system created by Linux founder Linus Torvalds in 2005. Its main goals are to improve the efficiency of collaboration among developers and to manage and track the version history of project. Git has the following features like Version Control, Branching and Merging, Distributed system, Data Integrity, Ease of Collaboration. In this dissertation I used Git to get YOLOv5 open-source code.

3. Roboflow (Roboflow: Give your software the power to see objects in images and video 2023)

Roboflow supports annotation, preprocessing, model training, and development for computer vision tasks. Roboflow processed my training and testing sets before developing the model. Pruning, normalisation, and noise in Roboflow preprocessing increase data quality. Roboflow offers data improvement capabilities including flipping, rotating, and cropping to promote dataset diversity and model generalisation. Roboflow simplifies dataset management. Uploading, organising, and sharing photos and annotations streamlines data management. In my dissertation on shapeless object

detection, Roboflow has saved me a lot of time and effort from data preprocessing to model training and evaluation. The URL for my Roboflow project is <https://universe.roboflow.com/yolov5-x4zrk/2023.7.14datasets>.

4. Labellmg (Tzutalin 2015)

Labellmg helped me label my model training and testing datasets in my dissertation. Labellmg is a Python and Qt-based image annotation application that creates rectangular labelled boxes. It helps create object detection annotations. Easy and intuitive using Labellmg, I could draw rectangular boxes on images and categorise them. Labellmg supports pascal, VOC, and YOLO label formats, so I may choose the appropriate one for the model and framework. Labellmg helped me train and evaluate my model by fast and accurately labelling my datasets.

5. Python3.8 (Welcome to Python.org 2023)

Python is a simple, accessible, interpreted, dynamically-typed programming language with considerable library support. Python 3.8 gave my dissertation a solid, flexible programming environment. Its cross-platform nature, substantial library support, and user-friendly syntax have helped me code faster.

6. PyTorch1.12.1 (PyTorch 2017)

PyTorch1.12.1 is my major deep learning framework for model design, training, and evaluation in my dissertation. PyTorch1.12.1 supports NVIDIA GPUs, which I used to accelerate model training and get better results in less time. It also gave me a powerful, easy-to-use, and efficient deep learning framework that allowed me to quickly iterate and get great results in my dissertation on shapeless object detection.

7. YOLOv5 (Jocher 2020)

In my dissertation, I have chosen YOLOv5 as the main object detection framework for shapeless objects. YOLOv5 (You Only Look Once version 5) is a popular, efficient, real-time object detection algorithm development and open-sourced by Ultralytics. It inherits the real-time performance characteristics of

the YOLO series, with several optimizations in accuracy and speed. I modified YOLOv5 compact and efficient architecture to identify shapeless faster. YOLOv5 allowed me to swiftly train and tune the model parameters. The YOLOv5 codebase is well-structured and supported by thorough documentation, making it easy to learn, edit and extend.

8. CBAM (Woo et al. 2018)

CBAM adds attention to deep convolutional neural networks. It focuses on specific neural networks and channel correlation, introducing two levels of attention in the model: spatial attention and channel attention. CBAM can be easily embedded into any block of a convolutional neural network without major changes. I added the CBAM block to YOLOv5's layers to improve shapeless object detection.

9. CoordAtt (Hou et al. 2021)

CoordAtt is a deep learning model attention mechanism. It emphasises coordinate points on the feature graph to help the model prioritise significant features. Modular and easy to integrate, the CoordAtt module can be added to deep neural networks. I fused it into YOLOv5 layers to improve shapeless object identification.

10. SENet (Hu et al. 2017)

SENet is a strong network structure that automatically learns feature channel relationships and adjusts channel weights. Any deep convolutional neural network can easily embed SENet module. I added it to YOLOv5's layers to boost shapeless object recognition.

4.2 Code

In this section of the code, we use publicly accessible software development kits and publicly available common code that can be employed during the coding process. The code references are indicated inside the code itself. This

section provides an analysis of the actions performed by the specific codes inside the used code.

Pre-work

First of all, you need to get the whole code of YOLOv5 framework from GitHub, the specific GitHub repository URL is <https://github.com/ultralytics/yolov5>. Following the document install the YOLOv5.

4.2.1 train.py

The following figure 10 is interception the main code of train.py in the YOLOv5 framework, in which we need to understand a few important parameters. First of all, we need to know that this entire code uses Python argparse module to define command line arguments. Among all the parameters, we will focus on the important ones related to the project. For example, the 445 line of code, '--weight' is the name of the command line parameter --weight, the parameter accepts the type of string, default represents that if the user does not provide the parameter, then it will be based on the default value of the value of the assignment, and the format of the end of each parameter added to the end of the description to help the user to understand the parameter.

```

443 def parse_opt(known=False):
444     parser = argparse.ArgumentParser()
445     parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
446     parser.add_argument('--cfg', type=str, default=ROOT / 'models/yolov5s-CoordAtt.yaml', help='model.yaml path')
447     parser.add_argument('--data', type=str, default=ROOT / 'data/myData.yaml', help='dataset.yaml path')
448     parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
449     parser.add_argument('--epochs', type=int, default=200, help='total training epochs')
450     parser.add_argument('--batch-size', type=int, default=4, help='total batch size for all GPUs, -1 for autobatch')
451     parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
452     parser.add_argument('--rect', action='store_true', help='rectangular training')
453     parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
454     parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
455     parser.add_argument('--noval', action='store_true', help='only validate final epoch')
456     parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
457     parser.add_argument('--noplots', action='store_true', help='save no plot files')
458     parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
459     parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
460     parser.add_argument('--cache', type=str, nargs='?', const='ram', help='image --cache ram/disk')
461     parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
462     parser.add_argument('--device', default='0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
463     parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%')
464     parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
465     parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
466     parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
467     parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
468     parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
469     parser.add_argument('--name', default='exp', help='save to project/name')
470     parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
471     parser.add_argument('--quad', action='store_true', help='quad dataloader')
472     parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
473     parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
474     parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
475     parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
476     parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
477     parser.add_argument('--seed', type=int, default=0, help='Global training seed')
478     parser.add_argument('--local_rank', type=int, default=-1, help='Automatic DDP Multi-GPU argument, do not modify')

```

Figure 40 – The main code of train.py

In the line of 445, we first use the locally downloaded YOLOv5s as the initial network weight. In line 446, it is necessary to make modifications to our code in order to adhere to a particular training profile. The configuration file is responsible for documenting the structural framework of the neural network, which will be elaborated upon in the subsequent yolov5s.yaml code. In line 447, Here we discussing the setup of data set file paths. The criteria for establishing the quantity of training iterations are specified on line 449. The line of 450 determines the number of samples input to the model during forward and backward propagation. The correct batch-size affects model convergence and performance. Large batch-size can speed up training each epoch by taking full advantage of hardware acceleration, but it can also cause out-of-memory or overflow errors. Small batch-size means define the number of samples used in each model weight update to avoid overfitting. The decision to transition from CPU to GPU training is shown in line 462.

4.2.2 detect.py

Both detect.py and train.py are important script file in the YOLOv5 codebase, train.py is responsible for training, detect.py is responsible for detect the object. Here I also focus on describing the main parameters adjusted in the project.

```
219 def parse_opt():
220     parser = argparse.ArgumentParser()
221     parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
222     parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(webcam)')
223     parser.add_argument('--data', type=str, default=ROOT / 'data/myData.yaml', help='(optional) dataset.yaml path')
224     parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
225     parser.add_argument('--conf-thres', type=float, default=0.5, help='confidence threshold')
226     parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
227     parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
228     parser.add_argument('--device', default='0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
229     parser.add_argument('--view-img', action='store_true', help='show results')
230     parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
231     parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
232     parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
233     parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
234     parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
235     parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
236     parser.add_argument('--augment', action='store_true', help='augmented inference')
237     parser.add_argument('--visualize', action='store_true', help='visualize features')
238     parser.add_argument('--update', action='store_true', help='update all models')
239     parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
240     parser.add_argument('--name', default='exp', help='save results to project/name')
241     parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
242     parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
243     parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
244     parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
245     parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
246     parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
247     parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
248     opt = parser.parse_args()
249     opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1 # expand
250     print_args(vars(opt))
251     return opt
```

Figure 41 – The main code of detect.py

The location of the model weight file is specified in line 221, the location of the data file to be detected is specified in line 222, the yaml file that provides information about the dataset during training is specified in line 223, and the parameter of utmost significance is specified in line 225. If the execution of the detection falls below this level, the forecast will be disregarded.

4.2.3 myData.yaml

This document provides an overview of the dataset, encompassing the dataset path, training set route, validation set path, and the total number of distinct item types present inside the dataset. This dissertation encompasses three distinct categories of recognition items. The three states of matter being referred to in

this context are as follows: 0-dirt (specifically water stains), 1-fire (in the form of a flame), and 2-steam (in its gaseous state).

```
1 # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2 # COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics
3 # Example usage: python train.py --data coco128.yaml
4 # parent
5 # └─ yolov5
6 #   └─ datasets
7 #     └─ coco128 ← downloads here (7 MB)
8
9
10 # Train/val/test sets as 1) dir: path/to/images, 2) file: path/to/images.txt, or 3) list: [path/to/images1, path/to/images2, ..]
11 path: ../dataset7.15 # dataset root dir
12 train: train/images # train images (relative to 'path') 128 images
13 val: valid/images # val images (relative to 'path') 128 images
14 test: test/images # test images (optional)
15
16 # Classes
17 names:
18   0: dirt
19   1: fire
20   2: steam
```

Figure 42 – The code of myData.yaml

4.2.4 yolov5s.yaml

Figure 13 shows the backbone network architecture and hyperparameters of yolov5s in detail.

```

1  # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3  # Parameters
4  nc: 3 # number of classes
5  depth_multiple: 0.33 # model depth multiple
6  width_multiple: 0.50 # layer channel multiple
7  anchors:
8    - [10,13, 16,30, 33,23] # P3/8
9    - [30,61, 62,45, 59,119] # P4/16
10   - [116,90, 156,198, 373,326] # P5/32
11
12 # YOLOv5 v6.0 backbone
13 backbone:
14   # [from, number, module, args]
15   [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17    [-1, 3, C3, [128]],
18    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19    [-1, 6, C3, [256]],
20    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21    [-1, 9, C3, [512]],
22    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23    [-1, 3, C3, [1024]],
24    [-1, 1, SPPF, [1024, 5]], # 9
25   ]

```

Figure 43 – The code of yolov5s.yaml backbone

1. **Depth_multiple** And **Width_multiple**: The `depth_multiple` parameter is employed to modify the depth of the model, namely the number of layers. The default configured depth will be modified according to the provided multiple. In smaller models, it is possible for the multiplier to be less than 1 in order to minimise the number of layers and minimise both the size and computational requirements of the model. Similar to the concept of `Width_multiple`, `Width_multiple` is a parameter that modifies the width of a model or the number of channels in each layer of the model.

2. **Anchor**s: A priori anchor frames play a variety of roles in target detection such as normalizing predictions, providing references, covering a wide range of object shapes and sizes, helping to allocate samples, and are an integral part of modern target detection methods. In anchors, three scales are defined for the size of the a priori anchor frames which are used to predict the bounding box. Different scales correspond to different feature map sizes.
3. **BackBone**: This part defines the main structure of yolov5, from input image to feature extraction. Each code line defines a model. The elements in the list are: which layer to start from, number of modules, modules type and module parameters. Let's take code line 16 as an example. -1 means the previous layer, 1 means just one module, Conv indicates a convolution module, 128 channels, 3×3 convolution kernel, step size 2 and no padding. C3 is a module specific to YOLOv5, representing CIOU loss, which has three convolutional layers. SPPF is the spatial pyramid pooling module, which helps the model to capture features at multiple scales.

Figure 14 defines the head structure of yolov5, which is mainly responsible for generating predictions of the target. The head structure of YOLO comprises a sequence of convolutional layers, Upsampling layers and other operations, which ultimately performs the detection of the target at different scales. nn.Upsample and Concat are two new module types found in the head architecture.


```

26
27 # YOLOv5 v6.0 head
28 head:
29   [[-1, 1, Conv, [512, 1, 1]],
30    [-1, 1, nn.Upsample, [None, 2, 'nearest']],
31    [[-1, 6], 1, Concat, [1]], # cat backbone P4
32    [-1, 3, C3, [512, False]], # 13
33
34    [-1, 1, Conv, [256, 1, 1]],
35    [-1, 1, nn.Upsample, [None, 2, 'nearest']],
36    [[-1, 4], 1, Concat, [1]], # cat backbone P3
37    [-1, 3, C3, [256, False]], # 17 (P3/8-small)
38
39    [-1, 1, Conv, [256, 3, 2]],
40    [[-1, 14], 1, Concat, [1]], # cat head P4
41    [-1, 3, C3, [512, False]], # 20 (P4/16-medium)
42
43    [-1, 1, Conv, [512, 3, 2]],
44    [[-1, 10], 1, Concat, [1]], # cat head P5
45    [-1, 3, C3, [1024, False]], # 23 (P5/32-large)
46
47    [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
48 ]

```

Figure 44 – The code of yolov5s.yaml head

1. **nn.Upsample**: The purpose of this technique is to augment the dimensions of the feature map. This technique is frequently employed in cases where there is a requirement to restore the spatial resolution of the initial input or to integrate low-quality characteristics with high-resolution ones.
2. **Concat**: Concat is an operation used to join two or more feature maps. The connection is usually done along the channel dimension, which keeps the spatial dimension of the feature graph constant but increase the number of channels.
3. **head**: The format of the parameters in head is exactly the same as the format of the parameters in backbone. In the code line 30, this is an upsampling layer that scales up the size of the feature map by a factor of 2, using nearest neighbor interpolation. In the code line 31, this is a

concatenation (concat) of the feature maps is performed. It concatenates the current layer (-1) with the output of layer 6. [1] denotes the concatenation along the channel dimensions. In the code line 32, this is a C3 block with an output of 512 channels. False may denote that this block does not contain additional operations such as residual concatenation.

4. **Detect:** In the code line of 47, this is the detection layer. It uses the pervious three feature map scales of 17, 20 and 23 layers for detection. The parameters nc denotes the number of categories, anchors define a priori anchor frames for each scale.

4.2.5 CBAM

In the CBAM attention mechanism three main modules are defined namely the channel attention module, the spatial attention module and the main CBAM's module.

```

955 # @inproceedings{hou2021coordinate,
956 #   title={Convolutional Block Attention Module},
957 #   author={Sanghyun Woo, Jongchan Park and Joon-Young Lee and So Kweon},
958 #   booktitle={ECCV},
959 #   year={2018}
960 # }
961
962 # CBAM注意力机制
963 new *
964 class ChannelAttention(nn.Module):
965     new *
966     def __init__(self, in_planes, ratio=16):
967         super(ChannelAttention, self).__init__()
968         self.avg_pool = nn.AdaptiveAvgPool2d(1)
969         self.max_pool = nn.AdaptiveMaxPool2d(1)
970
971         self.fc1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
972         self.relu1 = nn.ReLU()
973         self.fc2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)
974
975         self.sigmoid = nn.Sigmoid()
976
977     new *
978     def forward(self, x):
979         avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
980         max_out = self.fc2(self.relu1(self.fc1(self.max_pool(x))))
981         out = avg_out + max_out
982         return self.sigmoid(out)

```

Figure 45 – The code of CBAM's ChannelAttention

1. **Initialization method `__init__`:** **`in_planes`** is the number of channels in the input feature map. **Ratio** is a hyperparameter, generally used to minimise intermediate channel attention feature dimensionality. After that is initialization components, it's included **`self.avg_pool`** calculates the average value of each channel in the input map of features. **`self.max_pool`**: adaptive maximum pooling layer for input feature map channel maximum values. **`self.fc1`**: the first convolutional layer whose purpose is to reduce the channel dimension. **`self.relu1`**: ReLU activation function. **`self.fc2`**: a second convolutional layer whose purpose is to restore the dimensionality to its original size. **`self.sigmoid`**: Sigmoid activation function that ensures that the output weights lie between 0 and 1.

2. **Forward propagation method forward:** First, `self.avg_pool` calculates the input feature map channel averages. A feature map of size `(batch_size, in_planes, 1, 1)` will result. Convolution the average pooling output with `self.fc1` and apply the ReLU activation function. `Self.fc2` is used to restore channel dimensions and convolution to get `avg_out`. Use `self.max_pool` to calculate the maximum value of each input feature map channel and repeat the convolution and activation process to get `max_out`. Add `avg_out` and `max_out` for the final attention weight. `Self.sigmoid` should generate output weights between 0 and 1. The module outputs a feature map of channel attention weights `(batch_size, in_planes, 1, 1)` as the final result. Multiplying these weights by the input feature map adjusts channel priority in subsequent processing.

```

2 usages new *
983 class SpatialAttention(nn.Module):
    new *
984     def __init__(self, kernel_size=7):
985         super(SpatialAttention, self).__init__()
986
987         assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
988         padding = 3 if kernel_size == 7 else 1
989
990         self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
991         self.sigmoid = nn.Sigmoid()
992
993     new *
994     def forward(self, x):
995         avg_out = torch.mean(x, dim=1, keepdim=True)
996         max_out, _ = torch.max(x, dim=1, keepdim=True)
997         x = torch.cat([avg_out, max_out], dim=1)
998         x = self.conv1(x)
999         return self.sigmoid(x)

```

Figure 46 – The code of CBAM's SpatialAttention

The initialization approach involves the execution of the convolution layer and the Sigmoid activation function. The user is provided with the option to determine the size of the convolution kernel. In the forward procedure, the `x` tensor is initially provided as input, followed by the computation of the mean

and maximum values of the tensor along the channel dimension. By performing concatenation of a novel value and subsequently feeding this new value into the convolution layer, the resulting convolution layer acquires knowledge of both global and local information following the concatenation process. Ultimately, the output of the Sigmoid activation function is obtained.

```
1 usage  new *
1000  class CBAM(nn.Module):
        new *
1001      def __init__(self, in_planes, ratio=16, kernel_size=7):
1002          super(CBAM, self).__init__()
1003          self.ca = ChannelAttention(in_planes, ratio)
1004          self.sa = SpatialAttention(kernel_size)
1005
        new *
1006      def forward(self, x):
1007          x = self.ca(x) * x
1008          x = self.sa(x) * x
1009          return x
```

Figure 47 – The code of main CBAM

4.2.6 CoordAtt

This code implements a couple of custom PyTorch modules, mainly on activation functions for convolutional neural networks and an attention module called CoordAtt.

```

877 # CA
878 # @inproceedings{hou2021coordinate,
879 #   title={Coordinate Attention for Efficient Mobile Network Design},
880 #   author={Hou, Qibin and Zhou, Daquan and Feng, Jiashi},
881 #   booktitle={CVPR},
882 #   year={2021}
883 # }
2 usages new *
884 class h_sigmoid(nn.Module):
    new *
885     def __init__(self, inplace=True):
886         super(h_sigmoid, self).__init__()
887         self.relu = nn.ReLU6(inplace=inplace)
    new *
888     def forward(self, x):
889         return self.relu(x + 3) / 6
2 usages new *
890 class h_swish(nn.Module):
    new *
891     def __init__(self, inplace=True):
892         super(h_swish, self).__init__()
893         self.sigmoid = h_sigmoid(inplace=inplace)
    new *
894     def forward(self, x):
895         return x * self.sigmoid(x)

```

Figure 48 – The activation function of CoordAtt

h_sigmoid

1. Initialization: In the initialization function, it defines a ReLU6 layers, ReLU6 is a variant of ReLU which limits the output to between 0 and 6.
2. Forward Propagation: In forward propagation, it first adds 3 to the input and then activates it through ReLU6. This means that the input range is shifted, from the original $[-3, 3]$ to $[0, 6]$. The result is then divided by 6, making the output range between $[0, 1]$, similar to the range of the sigmoid function.

h_swish

1. Initialization: It defines an h_sigmoid activation function in the initialisation function.
2. Forward propagation: In forward propagation, it multiplies the input with the output of the h_sigmoid function. This means that it performs a non-linear

transformation on its input, similar to the original swish function, but using a hard-limited version of the sigmoid function.

This code defines a module called CoordAtt that implements a CoordAtt attention mechanism. The main idea is to generate vertical and horizontal attention for the input feature maps and weight the original input using these two attention maps.

```

897 class CoordAtt(nn.Module):
    new *
898     def __init__(self, inp, oup, reduction=32):
899         super(CoordAtt, self).__init__()
900         self.pool_h = nn.AdaptiveAvgPool2d((None, 1))
901         self.pool_w = nn.AdaptiveAvgPool2d((1, None))
902         mip = max(8, inp // reduction)
903         self.conv1 = nn.Conv2d(inp, mip, kernel_size=1, stride=1, padding=0)
904         self.bn1 = nn.BatchNorm2d(mip)
905         self.act = h_swish()
906         self.conv_h = nn.Conv2d(mip, oup, kernel_size=1, stride=1, padding=0)
907         self.conv_w = nn.Conv2d(mip, oup, kernel_size=1, stride=1, padding=0)
    new *
908     def forward(self, x):
909         identity = x
910         n, c, h, w = x.size()
911         #C*1*W
912         x_h = self.pool_h(x)
913         #C*H*1
914         #C*1*h
915         x_w = self.pool_w(x).permute(0, 1, 3, 2)
916         y = torch.cat([x_h, x_w], dim=2)
917         #C*1*(h+w)
918         y = self.conv1(y)
919         y = self.bn1(y)
920         y = self.act(y)
921         x_h, x_w = torch.split(y, [h, w], dim=2)
922         x_w = x_w.permute(0, 1, 3, 2)
923         a_h = self.conv_h(x_h).sigmoid()
924         a_w = self.conv_w(x_w).sigmoid()
925         out = identity * a_w * a_h
926         return out

```

Figure 49 – The main code of CoordAtt

Initialization

First define two adaptive average pooling pooling vertical and horizontal input, the code line of 902 is in the middle of the passage, by convolution will connect the characteristics of the vertical and horizontal pooling the channel number of compression for the middle number of mip, through batch to a layer, and finally generate two convolution layer for vertical and horizontal attention.

Forward

The initial step involves storing the original input tensor, followed by extracting the dimensions of the input tensor. Subsequently, pooling operations are executed based on the height dimension, resulting in a tensor with dimensions $c \times 1 \times w$. The tensor is subsequently subjected to pooling based on the height of the input tensor. Pooling is conducted in the horizontal direction to get a tensor with dimensions $c \times h \times 1$. This tensor is subsequently transformed into a tensor with dimensions $c \times 1 \times h$. To generate a tensor of size $c \times 1 \times (h + w)$, the height and width note mappings are merged along the height dimension. The channel dimension of y should be reduced to the minimum integer programming (MIP) level. The application of batch normalisation is recommended. Utilise the hard offset activation technique. The tensor y can be partitioned into attention maps based on its height and width. Revert the width attention map to its initial dimensional arrangement. Produce the ultimate height attention map. Produce the ultimate attention map width. The final output is obtained by performing element-wise multiplication between the original input tensor and the elements of the height and width attention map.

4.2.7 SENet

The code defines a module called SELayer that implements the Squeeze-and-Excitation operation for strengthening or weakening the importance of certain channels in a deep learning model.

```

936 # SENet注意力机制
937 1 usage new *
937 class SELayer(nn.Module):
938     new *
938     def __init__(self, channel, reduction = 16):
939         super(SELayer, self).__init__()
940         self.avg_pool = nn.AdaptiveAvgPool2d(1)
941         self.fc = nn.Sequential(
942             nn.Linear(channel, channel // reduction, bias=False),
943             nn.ReLU(inplace=True),
944             nn.Linear(channel // reduction, channel, bias=False),
945             nn.Sigmoid()
946         )
947     new *
947     def forward(self, x):
948         b, c, _, _ = x.size()
949         y = self.avg_pool(x).view(b, c)
950         y = self.fc(y).view(b, c, 1, 1)
951         return x * y.expand_as(x)

```

Figure 50 – The code of SENet

Initialization

By first passing an adaptive averaging pooling layer that effectively decreases the spatial dimensions to a fixed size of 1x1, irrespective of the initial spatial dimensions of the input data. The process being described can be understood as a form of global averaging pooling, wherein the spatial locations of each channel are averaged together. The second layer is organised in a linear series, wherein first nn.Linear each subsequent channel has a reduced number of channels, resulting in effective compression characteristics. The ReLU activation function is subsequently employed. The second nn.Linear increases the number of channels from a reduced channel size to the original channel size, hence performing an expansion process. The Sigmoid function is employed in order to ensure that the resulting output value can be interpreted as the significance or magnitude of each channel.

Forward

1. `b, c, _, _ = x.size()`: this line of code gets the batch size (b) and the number of channels (c) from the input tensor.
2. `y = self.avg_pool(x).view(b, c)`: performs global average pooling on input x and reshapes it from `[batch_size, channels, 1, 1]` to `[batch_size, channels]`.
3. `y = self.fc(y).view(b, c, 1, 1)`: The pooled output is processed through the previously defined fully-connected network to obtain weights for each channel. These weights are then reshaped to `[batch_size, channels, 1, 1]`.
4. `return x * y.expand_as(x)`: Here, the input x is multiplied element-by-element with the weights y. `y.expand_as(x)` ensures that y is shaped the same way as x so that the multiplication operation can be performed on each spatial position.

4.3 Development Process

The attention mechanism can facilitate model learning of complex features and weight the feature based on learned weights, strengthening important features and weakening unimportant ones (Vaswani et al. 2023). In the context of a target detection job, this feature enhances the model's ability to precisely identify the target, particularly when confronted with intricate backdrops or diminutive targets. While the introduction of the attention mechanism introduces additional computation, in some designs this can be offset by reducing the computational requirements for subsequent processing (Howard et al. 2017). The attention mechanism allows the model to assign different weights to features based on the importance of each scale, helping to better capture targets at various scales.

4.3.1 Analyzing the underlying neural network

YOLOv5, or "You Only Look Once version 5", is a popular real-time target detection algorithm. Since its first release, YOLO has gone through several iterations, each optimized in terms of speed, accuracy and model size.

1. Backbone

The dorsal bone is the core of the neural network which uses multiple convolutional layers, residual layers and other specific network structures to extract features from the image. These features are then fed into the head for target detection. The main structures include Conv, C3, SPPF. We mainly focus on some features of the SPPF layer. SPPF: Spatial Pyramid Pooling, this technique enables the extraction of features at several scales and enhances the model's sensory field.

2. Head

The head is the part of the model that translates the features in the dorsal bone into the final detection output. The head of YOLOv5 has several unique features that make it excel in multi-scale target detection.

3. Strengths and Considerations

YOLOv5 employs the strategies of jump-joining and multi-scale prediction, which enables it to detect targets of various sizes efficiently. YOLOv5 is designed to strike a balance between speed and accuracy. This makes it ideal for real-time applications such as drones and surveillance. By simply modifying parameters or structure such as neural network architecture, make the model can captures the multi-scale prediction strategy ensures that YOLOv5 can effectively detect targets of all sizes.

4.3.2 SPPF

The SPPF (Spatial Pyramid Pooling - Fast) layer is an important structure in computer vision and deep learning whose main purpose is to extract features at different spatial scales. The concept is mainly based on the traditional Spatial Pyramid Pooling (SPP) method, which has been shown to be effective in matching image descriptors (He et al. 2014). However, in the context of deep learning, the SPPF layer is mainly used to capture features of objects of different size and scales.

1. Multi-scale feature extraction

When working with real-world images, target objects may appear in wide variety of sizes and scales. To ensure that the target detector can effectively detect objects of different sizes, multi-scale features need to be extracted from the image. The SPPF layer enables the network to capture image features at different spatial scales, which is crucial for detection objects of different sizes.

2. Fixed-size output

Conventional convolutional layers require a fixed-size input, which means that the image needs to be resized to a specific size before it can be processed. However, in SPPF, due to its special structure, it can accept any size of input image and output a fixed size feature map. This is particularly important because it allows the model to be more flexible in processing inputs of various sizes.

3. Information-rich feature representation

Since the SPPF layer extracts features at multiple spatial scales, the resulting feature maps incorporate a large amount of contextual information. This means that each spatial location contains information not only about itself but also about its neighbourhood. Such a feature representation is usually more robust and provides more information to the subsequent network layers.

4. Efficiency and computational savings

While the SPPF layer adds some computational complexity, it also avoids the need to sample the entire image multiple times. With a single forward propagation, the network can acquire features at multiple scales and increasing computational efficiency.

4.3.3 Tuning the underlying neural network

Adding the attention mechanism to the SPPF (Spatial Pyramid Pooling) layer before can bring a number of advantages to YOLO or other convolutional neural networks. The core idea of the attention mechanism is to weight the input features so that the network focuses more on those parts that are more

important to the task. Listed below are a few reasons for including the attention mechanism in front of the SPPF layer and the logic behind it:

1. Improvement of feature differentiation

When the target is located in some specific regions of the image, the features in these regions are particularly important for target detection. By applying the attention mechanism, the model can adaptively strengthen its focus on these regions, thus improving the accuracy of detection.

2. Context-awareness

The use of the attention mechanism enables the model to direct its focus towards not just the immediate aspects but also the surrounding contextual information that is relevant to the target. The inclusion of contextual information plays a crucial role in the spatial pyramid pooling process, as it facilitates the extraction of multi-scale features.

3. Improved Robustness

In practical applications, it is common for images to be affected by a range of circumstances, including occlusion and blurring. The utilisation of the attention mechanism enables the model to allocate greater emphasis on undisturbed components, hence enhancing the model's resilience.

4. Computational Efficiency

Although the attention mechanism increases the computational complexity, it allows the model to focus more on the key parts of the image, thus reducing the computational need for unimportant features. This focusing effect is particularly useful prior to SPPF, which requires manipulation of the entire feature map.

In summary, adding an attentional mechanism before the SPPF layer can bring a range of advantages to the model, thus improving its performance in the target detection task. This combination strategy combines the flexibility of attention with the multi-scale feature extraction capabilities of SPPF, providing a powerful tool for achieving efficient and accurate target detection.

4.3.4 Specific implementation process

Paste the code for the three plug-and-play attention mechanisms into the last line of `common.py` in the `models` folder under YOLOv5 (Specific code screenshots in 4.2.5 – 4.2.7). This step adds our attention mechanisms to the YOLOv5 neural network architecture, making it easier for us to make subsequent adjustments to the overall architecture and to write it.

Find the `yolo.py` file in the `models` folder, and find the function called `parse_model` on line 299. The aforementioned function is employed for the purpose of constructing the framework of the model. The function is responsible for parsing and generating the layers of the neural network, utilizing either a configuration file or a list of configurations. This design approach makes it relatively easy to change the structure of the model and experiment with different configurations, since only the configuration files or lists need to be changed. Rather than rewriting the code manually. Add the class names of the three attention mechanisms at the end of line 319, as shown in Figure 21.

```
314         args[j] = eval(a) if isinstance(a, str) else a # eval strings
315
316         n = n_ = max(round(n * gd), 1) if n > 1 else n # depth gain
317         if m in {
318             Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, SPPE, DWConv, MixConv2d, Focus, CrossConv,
319             BottleneckCSP, C3, C3TR, C3SPP, C3Ghost, nn.ConvTranspose2d, DWConvTranspose2d, C3x, SElayer, CBAM, CoordAtt}:
320             c1, c2 = ch[f], args[0]
321             if c2 != no: # if not output
322                 c2 = make_divisible(c2 * gw, 8)
323
324             args = [c1, c2, *args[1:]]
325             if m in {BottleneckCSP, C3, C3TR, C3Ghost, C3x}:
326                 args.insert(2, n) # number of repeats
327                 n = 1
328         elif m is nn.BatchNorm2d:
```

Figure 51 – Add Attention mechanism class name

In terms of modifying the neural network architecture, we chose to add different attentional mechanisms to the same layer to compare with the native framework due to the fairness of the experiments. Since most of the code modifications are the same, I will use the neural network architecture with CBAM as an example to describe the implementation steps.

1. We first copy the yolov5s.yaml file in the models folder and rename it to yolov5s-CBAM.yaml. This file describes the native yolov5s neural network architecture in detail, and we'll build on it. Add the attention mechanism network after the 23rd line of code (Format [-1, 1, CBAM, [1024]]). Where -1 represents the input from the previous layer, and 1 represents the addition of a CBAM Attention Module, which is the class name of the attention mechanism we just wrote into yolo.py and common.py, [1024] means there are 1024 output channels. In this step, we insert the CBAM attention mechanism into the ninth layer of neural network.
2. With the addition of the attention layer, the backbone originally contained nine layers of neural networks became ten layers. As the number of layers in the neural network changes, the number of layers in some of the feature layers also changes. We also need to change the architecture of the neural network in the head from the previous 9 layers.

```

1 # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3 # Parameters
4 nc: 3 # number of classes
5 depth_multiple: 0.33 # model depth multiple
6 width_multiple: 0.50 # layer channel multiple
7 anchors:
8   - [10,13, 16,30, 33,23] # P3/8
9   - [30,61, 62,45, 59,119] # P4/16
10  - [116,90, 156,198, 373,326] # P5/32
11
12 # YOLOv5 v6.0 backbone
13 backbone:
14   # [from, number, module, args]
15   [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17    [-1, 3, C3, [128]],
18    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19    [-1, 6, C3, [256]],
20    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21    [-1, 9, C3, [512]],
22    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23    [-1, 3, C3, [1024]],
24    [-1, 1, CBAM, [1024]], # CBAM注意力机制
25    [-1, 1, SPPF, [1024, 5]], # 9
26  ]

```

Figure 52 – yolov5s-CBAM neural network architecture backbone

3. In line 41 of code and in line 45 of the code, the concatenated concatenation of the feature maps of the previous feature layers is used. Since 1 is added to all neural networks after layer 9, in line 41 of the code we change the layer 14 to 15 and in line 45 of code we change the layer 10 to 11. Meanwhile, in the detection layer, the three feature map scales used for detection are all in the feature layer after layer 9, which is modified to be 18, 21, 24 layer.

```

27
28 # YOLOv5 v6.0 head
29 head:
30     [[-1, 1, Conv, [512, 1, 1],
31        [-1, 1, nn.Upsample, [None, 2, 'nearest']],
32        [[-1, 6], 1, Concat, [1]], # cat backbone P4
33        [-1, 3, C3, [512, False]], # 13
34
35        [-1, 1, Conv, [256, 1, 1],
36        [-1, 1, nn.Upsample, [None, 2, 'nearest']],
37        [[-1, 4], 1, Concat, [1]], # cat backbone P3
38        [-1, 3, C3, [256, False]], # 17 (P3/8-small)
39
40        [-1, 1, Conv, [256, 3, 2],
41        [[-1, 15], 1, Concat, [1]], # cat head P4
42        [-1, 3, C3, [512, False]], # 20 (P4/16-medium)
43
44        [-1, 1, Conv, [512, 3, 2],
45        [[-1, 11], 1, Concat, [1]], # cat head P5
46        [-1, 3, C3, [1024, False]], # 23 (P5/32-large)
47
48        [[18, 21, 24], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
49    ]
50

```

Figure 53 - yolov5s-CBAM neural network architecture head

The incorporation SENet and CoordAtt into the core YOLOv5 network architecture follows a similar approach as that of CBAM.


```

1  # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3  # Parameters
4  nc: 3  # number of classes
5  depth_multiple: 0.33  # model depth multiple
6  width_multiple: 0.50  # layer channel multiple
7  anchors:
8      - [10,13, 16,30, 33,23]  # P3/8
9      - [30,61, 62,45, 59,119]  # P4/16
10     - [116,90, 156,198, 373,326]  # P5/32
11
12  # YOLOv5 v6.0 backbone
13  backbone:
14      # [from, number, module, args]
15      [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16       [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17       [-1, 3, C3, [128]],
18       [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19       [-1, 6, C3, [256]],
20       [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21       [-1, 9, C3, [512]],
22       [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23       [-1, 3, C3, [1024]],
24       [-1, 1, CoordAtt, [1024]], # CoordAtt注意力机
25       [-1, 1, SPPF, [1024, 5]], # 9
26  ]

```

Figure 54 - yolov5s-CoordAtt neural network architecture head

```

1 # YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
2
3 # Parameters
4 nc: 3 # number of classes
5 depth_multiple: 0.33 # model depth multiple
6 width_multiple: 0.50 # layer channel multiple
7 anchors:
8   - [10,13, 16,30, 33,23] # P3/8
9   - [30,61, 62,45, 59,119] # P4/16
10  - [116,90, 156,198, 373,326] # P5/32
11
12 # YOLOv5 v6.0 backbone
13 backbone:
14   # [from, number, module, args]
15   [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
16    [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
17    [-1, 3, C3, [128]],
18    [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
19    [-1, 6, C3, [256]],
20    [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
21    [-1, 9, C3, [512]],
22    [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
23    [-1, 3, C3, [1024]],
24    [-1, 1, SELayer, [1024]], # SENet注意力机制
25    [-1, 1, SPPF, [1024, 5]], # 9
26   ]
27

```

Figure 55 - yolov5s-SENet neural network architecture head

After the addition of the attention mechanism and the modification of the neural network, we can test to find out if the modified model can pass the running test.

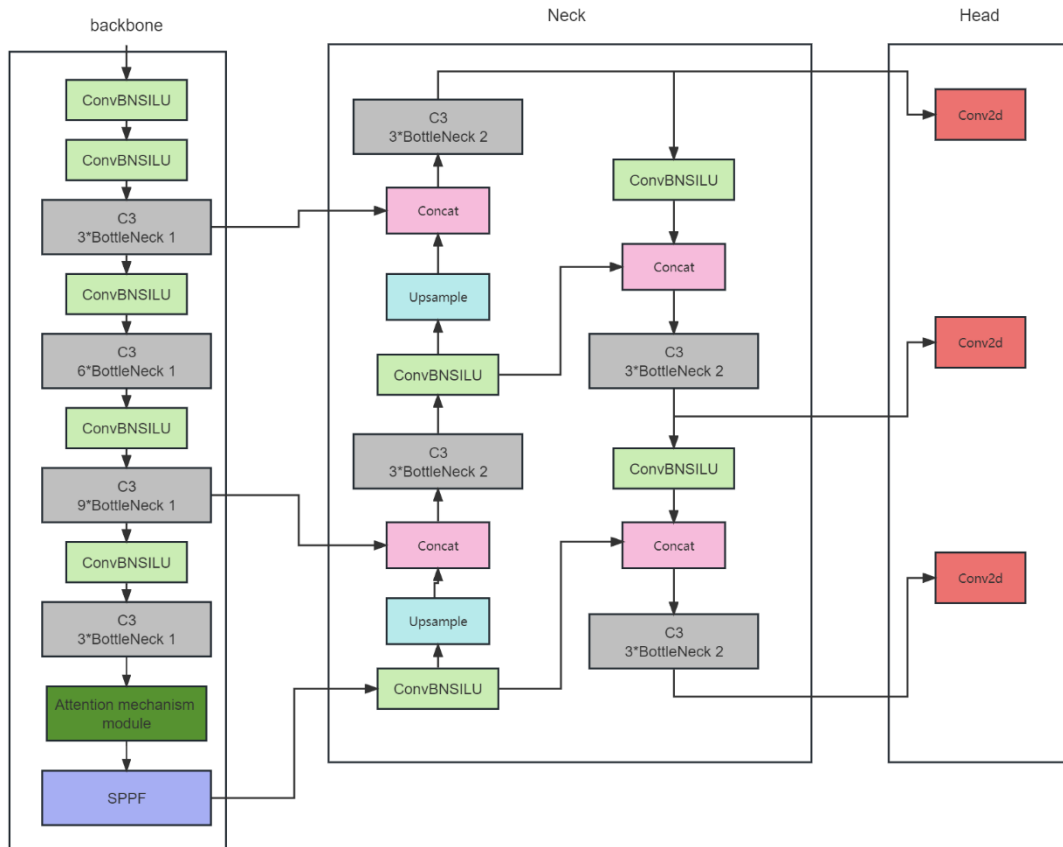


Figure 56 – The yolov5s network structure of adding attention mehcanism

5. Result and Evaluation

5.1 Experimental environment

python3.8 and pytorch1.12.1 are used for development. The experiments are run on Windows10 with an AMD Ryzen 7 5800H CPU with a base frequency of 3.20GHz, an Nvidia GeForce RTX 3070 Laptop GPU and Computer Device Architecture (CUDA)11.3 for GPU training acceleration. All code is developed with open-source tools and packages.

5.2 Experimental Methods

In this dissertation, we use same dataset to test the different model. And set a fixed hyper-parameter to ensure the fairness and impartiality of the test. The table 5 below presents the hyperparameters used during the training process.

Name	Value
depth_multiple	0.33
width_multiple	0.50
initial learning rate	0.01
final learning rate	0.01
Stochastic Gradient Descent (SGD)	0.937
Weight decay for the optimizer	0.05
Warmup epochs	3
initial momentum during warmup	0.8
Training epochs	200
batch-size	4
photo size	640 × 640 pixels

Table 5 – The hyperparameters set for training

In the object detection task, the framework running results and the performance of the classifiers are metrics to measure the effectiveness of the neural network design and the model. The following values are commonly used to define the performance of a detection model: IoU, Precision, F1 score, Recall, mAP.

IoU starts with the Jaccard Index (Jaccard index. 2023), used as a common metric for predicting the overlap between the bounding box and the real bounding box in the object detection task as shown in Equation (12), The term "Area of intersection" denotes the region where the predicted bounding box and the actual bounding box overlap, whereas the term "Area of Union" represents the region where the predicted bounding box and the actual bounding box combine.. Usually, the higher the value of IoU, the higher the prediction accuracy. In some evaluation protocols, when the IoU exceeds a certain threshold (He et al. 2018), the prediction is correct.

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \quad (12)$$

In Equation (13)-(15), describes the calculation of precision recall as well as F1 score. True positive (TP) refers to a test result that correctly identifies the presence of a specific condition or feature. A false positive (FP) refers to a situation in which a test incorrectly suggests the presence of a specific condition or trait. A false negative (FN) refers to a diagnostic test that incorrectly suggests the absence of a specific ailment or feature. F1 score is a widely metric used to assessing the efficacy of a model in the context of a classification task. The harmonic mean of precision and recall is calculated. (Precision and recall. 2023).

$$\text{Precision} = \frac{TP}{TP + FP} \quad (13)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (14)$$

$$F1 \text{ score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (15)$$

The Equation (16) expresses the average accuracy, which is a frequently used evaluation metric in information retrieval and computer vision. Parameters n is the number of detected or retrieved results. P(k) is the precision in the first k detection result. $\Delta r(k)$ is the percentage of recall at the first k detection result with the first k-1 result by the amount of change between them (Evaluation measures (information retrieval). 2023).

$$AveP = \sum_{k=1}^n P(k) \Delta r(k) \quad (16)$$

This formula calculates mAP (mean Average Precision). mAP is an evaluation metric often used in object detection and other related tasks. Q is the total number of queries or categories. In object detection, this usually corresponds

to the total number of categories in the dataset. $AveP(q)$ is q average precision of the query or category (Evaluation measures (information retrieval). 2023).

$$mAP = \frac{\sum_{q=1}^Q AveP(q)}{Q} \quad (17)$$

5.3 Analysis of results

The integration of attention mechanisms into deep neural networks, especially object detection frameworks, has become a topic of great interest to enhance model interpretability and performance. This research aimed to probe the effectiveness of amalgamating three disparate attention mechanisms with the YOLOv5s architecture to discern which one affords a higher detection accuracy for amorphous objects. Herein, we unveil the findings.

Name	P	R	F1	mAP50	mAP50-95
YOLOv5s	0.809	0.806	0.807	0.848	0.527
YOLOv5s-CBAM	0.896	0.833	0.863	0.883	0.516
YOLOv5s-CoordAtt	0.864	0.82	0.841	0.852	0.525
YOLOv5s-SENet	0.891	0.874	0.882	0.861	0.507

Table 6 – The training results for all models

```

Fusing layers...
YOLOv5s summary: 157 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
  Class  Images  Instances   P      R   mAP50  mAP50-95: 100%| 8/8 [00:00<00:00, 8.76it/s]
    all     60       67   0.809   0.806   0.848   0.527
    dirt    60       21   0.814    0.81   0.856   0.492
    fire    60       25   0.812    0.8   0.839   0.583
    steam   60       21   0.801    0.81   0.849   0.506
Results saved to runs\train\exp

```

Figure 57 – Training results of YOLOv5s

```

Freeing layers...
YOLOv5s-CBAM summary: 168 layers, 7019338 parameters, 0 gradients, 15.8 GFLOPs

```

Class	Images	Instances	P	R	mAP50	mAP50-95: 100%
all	60	67	0.896	0.833	0.883	0.516
dirt	60	21	0.864	0.81	0.836	0.468
fire	60	25	0.824	0.88	0.894	0.548
steam	60	21	1	0.811	0.918	0.53

```

Results saved to runs\train\exp

```

Figure 58 – Training results of YOLOv5s-CBAM

```

YOLOv5s-CoordAtt summary: 167 layers, 7043864 parameters, 0 gradients, 15.8 GFLOPs

```

Class	Images	Instances	P	R	mAP50	mAP50-95: 100%
all	60	67	0.864	0.82	0.852	0.525
dirt	60	21	0.833	0.81	0.845	0.534
fire	60	25	0.822	0.84	0.865	0.56
steam	60	21	0.937	0.81	0.846	0.482

```

Results saved to runs\train\exp

```

Figure 59 – Training results of YOLOv5s-CoordAtt

```

Freeing layers...
YOLOv5s-SENet summary: 164 layers, 7019240 parameters, 0 gradients, 15.8 GFLOPs

```

Class	Images	Instances	P	R	mAP50	mAP50-95: 100%
all	60	67	0.891	0.874	0.861	0.507
dirt	60	21	0.819	0.81	0.776	0.462
fire	60	25	0.853	0.88	0.85	0.512
steam	60	21	1	0.932	0.957	0.546

```

Results saved to runs\train\exp

```

Figure 60 – Training results of YOLOv5s-SENet

Table 4 and Figure 27 – 30 show the experimental results. The Convolutional Block Attention Module (CBAM) maximized precision at 0.896. This shows that the CBAM mechanism improves the models without shape object detection, minimizing false positives. The enhanced mAP50 score supports this. However, its mAP50-95, a stricter metric, is slightly lower than vanilla YOLOv5s, indicating that performance improves more at higher IoU thresholds. The YOLOv5s model had 0.874 recall with SENet integration. SENet's ability to reduce false negatives captures more without shape objects. However, its lower mAP50-95 score raises concerns regarding its effectiveness throughout a wider IoU threshold range. Over the vanilla model, Coordinate Attention (CoordAtt) performed better. It didn't top any statistic, but its balanced scores, especially an F1 score of 0.841, suggest its use in situations where precision and recall are equally important.

In real life, numerous application scenarios require us to evaluate the three integration frameworks separately. Critical surveillance systems, where false positives are costly, may benefit more from YOLOv5s-CBAM. YOLOv5s-SENet

may be better for situations where missing an object is harmful, such as tumour detection in medical imaging. For precision-recall balance, YOLOv5-CoordAtt is a good choice.

6. Conclusion and Future work

6.1 Conclusion

The objective of this research article is to address the issue pertaining to the precision of the YOLOv5 detection algorithm in accurately identifying objects that lack well-defined forms. This study employs attention mechanisms to augment the detection of small targets and small characteristics, thereby enhancing the model's learning capabilities and improving the accuracy of object detection in scenarios where objects lack fixed shapes. In order to conduct a comparative analysis of critical parameters, three attention mechanisms, namely CBAM, CoordAtt, and SENet, have been chosen for integration into both the YOLOv5 detection algorithm and the original YOLOv5 target detection method. The test findings indicate that the three integrated models demonstrate significant enhancements in the major detection parameters. This highlights the potential of incorporating attentional mechanisms into object detection algorithms for effective performance in the detection task, as well as suggesting a viable avenue for future development.

The analysis involved the utilisation of recall value, precision, mAP value with an IoU value set to 0.5, and the average mAP value. The IoU value was incrementally increased from 0.5 to 0.95 for the purpose of the analysis. The experimental results demonstrate that the mean accuracy of the three enhanced algorithms (YOLOv5s-CBAM, YOLOv5s-CoordAtt, YOLOv5s-SENet) discussed in this study surpasses that of the original YOLOv5 algorithm. Notably, the YOLOv5s-CBAM variant exhibits the most evident optimisation.

The enhanced YOLOv5 algorithm, which incorporates the CBAM attention mechanism, demonstrates notable enhancements across all measured parameters. In brief, the integration of the attention mechanism into the YOLOv5 algorithm yields significant enhancements in detecting objects lacking a predetermined shape, while also improving the extraction of intricate details inside the detection process.

In conclusion, the integration of attention mechanisms has been shown to greatly enhance the efficacy of object detection algorithms, particularly in complex domains characterised by the detection of small targets lacking fixed shapes. In the context of medical diagnostic tasks, prioritising recall as a crucial factor, it is not recommended to opt for a CBAM-integrated system without careful consideration. Instead, it is advised to select a detection system combined with SENet, which offers a greater recall rate. Gaining a comprehensive understanding of the intricate nature and distinctiveness of each mechanism might facilitate the selection of a more suitable option.

6.2 Main contributions

Attention processes let neural networks focus on essential features in computer vision and object detection, boosting detection performance. The YOLOv5 base network was tested with three popular attention mechanisms: CBAM, SENet, and CoordAtt. Our major goal was to determine which approach improves shapeless item detection best.

YOLOv5s-CBAM boasts the highest precision (0.896), implying that when it predicts an object, it's most likely correct. This suggests that the integration of CBAM helps in narrowing down the model's focus on actual positive instances, thereby reducing false positives. YOLOv5s-SENet showcases the highest recall (0.874), signifying that it successfully identifies the majority of actual

positive cases. The SENet attention mechanism appears to capture more instances of amorphous objects, even if they might be challenging to detect. The harmonic mean of precision and recall, F1, peaks for YOLOv5-SENet (0.882). It presents a balanced perspective, making SENet a favorable choice when both precision and recall are of significance. YOLOv5s-CBAM leads mAP50 with 0.883, implying that at a 50% intersection over Union (IoU) threshold, It outperforms the others. This metric is crucial for applications where moderate object overlap is permissible.

The YOLOv5s models is robust in its right, the integration of attention mechanisms, CBAM, CoordAtt and SENet, each offers distinct enhancements. This solidifies the idea that attention mechanisms, by providing focused contextual information, can improve object detection tasks, especially in complex scenarios like detecting amorphous objects.

6.3 Comparison of methods

CBAM (Convolutional Block Attention Module):

CBAM works by fusing both spatial and channel-wise attentions sequentially, ensuring that the network focuses on salient features in both domains. It dynamically recalibrates the feature maps in two steps: channel attention followed by spatial attention. Due to its dual-attention mechanism, CBAM increased the depth and computational complexity of YOLOv5. However, this overhead was offset by the notable gains in mAP scores for shapeless objects.

Strengths and Weaknesses: Provides a comprehensive attention mechanism that looks at both spatial relevance and channel importance. Particularly beneficial for highlighting nuanced patterns which might be vital for shapeless objects. However, the sequential nature of its attention modules could introduce

redundancy. Additionally, the increased model complexity can be a concern for real-time applications.

SENet (Squeeze-and-Excitation Network)

SENet recalibrates channels. Global average pooling and gating allow it to weight channels by significance. SENet balances efficiency and detection with less computing power than CBAM. It increased mAP slightly less than CBAM.

Strengths and Weaknesses: Lightweight mechanism focusing on channel interdependencies. This reduces computational overhead and streamlines attention. By focusing exclusively on channels, it may miss spatial details needed to detect shapeless things.

CoordAtt (Coordinate Attention)

CoordAtt emphasises feature map coordinate correlations. It calculates attention scores based on coordinate relevance to help the model focus on important locations. Adding CoordAtt to YOLOv5 was the most balance of the three. The balance between performance gain and computational cost makes it an appealing basic architectural addition.

Strengths and Weaknesses: Captures spatial relationships efficiently without computing overhead. The shapeless object detection score improvement was noteworthy. This new method may present unexpected issues when deploying in different settings or combining with other architectural changes.

Adding attention methods to object detection systems like YOLOv5 illuminates model enhancing nuances. Each method has pros and cons, but comparing them helps academics and practitioners customise their models for specific problems like shapeless object recognition.

6.4 Limitations and Future work

6.4.1 Limitations

While conducting our research, we integrated various datasets to evaluate the effectiveness of CBAM, SENet, and CoordAtt in conjunction with YOLOv5 for the detection of objects with indistinct shapes. However, it is important to note that these datasets may not encompass all possible scenarios. Certain distinctive characteristics or variations of amorphous entities might not be accounted for, thus leading to a skewed assessment of the efficacy of each attention mechanism. The dissertation assessments were conducted on predetermined hardware and some solid background setups. The utilisation of various graphics processing units (GPUs) or processing units might potentially impact the speed, memory utilisation, and even the outcomes of detection to a certain degree. Life may involve more complex background interference, requiring more datasets to train the model. Our primary objective is to incorporate single attention mechanisms into the YOLOv5 base network. The act of combining or integrating these elements at varying levels of depth can potentially produce diverse outcomes. When selecting hyperparameters, it is common practise to utilise the same hyperparameter values across different models. However, it is important to note that each attention mechanism necessitates distinct hyperparameter configurations in order to achieve best performance. This dissertation focuses on the YOLOv5 framework. Although the findings offer useful insights, their direct applicability to alternative object detection architectures may be limited.

6.4.2 Future work

In this section, we will discuss potential areas for further research and development. To enhance the comprehensiveness of our findings, it is

recommended that forthcoming studies incorporate a broader array of datasets, encompassing those expressly designed for the purpose of detecting imperceptible objects in diverse environmental contexts. Additionally, the exploration of hybrid models that integrate the advantageous features of CBAM, SENet, and CoordAtt has the potential to generate fresh insights and enhance performance. The existence of numerous object detection models allows for the assessment of attentional mechanisms' influence on alternative object detection architectures, hence determining their overall suitability. One may consider conducting a comprehensive investigation into the topic of video-based invisible object detection, which entails several obstacles such as object persistence and motion-based anomalies.

7. Reflection on Learning

This dissertation was difficult because I had never studied artificial intelligence or computer vision. I chose this project because recognising the thing in a picture sounded tough and cool. However, I often fretted about finishing my dissertation on time. After numerous discussions with my supervisor, I chose my research topic. Since the project began, I have met with my supervisor every half-month or emailed him for advice on my progress and intentions.

Initially, the robot's eyes were to detect video streams, but due to a tiny problem, I discussed with my supervisor and updated the project to use the YOLOv5 framework to boost home security, which is equivalent to developing a home security system. My extremely responsible supervisor advised me to shift the topic to something more difficult and modify the data processing and network architecture to improve the framework. After several meetings and Teams communication, we chose to detect shapeless objects using a modified YOLOv5 framework with an attention mechanism.

I examined the YOLOv5 framework and each neural network module's capabilities. YOLOv5 on github and YouTube tutorials explained the framework. I discovered the importance of each module for object detection and feature extraction by reviewing the source code. Once I grasped the architecture and training procedures, I couldn't wait to train on the dataset, but the results were disappointing. Papers and Google helped me discover that the framework's hyperparameter settings may be to blame, and I began to understand each hyperparameter. After that, I understood YOLOv5's architecture better.

I discovered the Attention Mechanism, an NLP mechanism that emphasises keywords for more accurate semantics, while reading the literature. After reviewing more material, I realised the attention process can be applied in object detection. In feature map acquisition, the attention mechanism boosts weights to learn essential channels. I had many issues integrating the attention mechanism with YOLOv5. The greatest issues were choosing an attention mechanism, adding it, and placing it. The first and second queries were answered by mainstream attention mechanisms, while CBAM, CoordAtt, and SENet stood out as old and novel. More significantly, adding them is easy. The architecture description on the YOLOv5 website vaguely stated where to add attention methods. After some experimentation and tutorials, I found how to integrate them to the YOLOv5 network architecture.

For final testing and evaluation, I prepared over 7 dataset variants. Finally, after over 40 experiments yielded the expected results. Many experiments have taught me useful reflections that I will utilise in my studies and profession. I studied deep learning models, architectures, and improvement processes, which laid the groundwork for my future job in deep learning. I've learned to critically assess each outcome in bugs and unsatisfactory experiments and to

deeply investigate their origins. I also learned to parallelize and manage my time. Reading papers or watching tutorials as the machine trains.

In reflection, this dissertation was a challenge, learning, and growing adventure. I examined attention mechanisms' tremendous potential with deep learning. My dissertation is completed, but my discoveries, learnings, and investigations continue.

References

- Ahmad, I. et al. 2022. Deep Learning Based Detector YOLOv5 for Identifying Insect Pests. *Applied Sciences* 12(19), p. 10167. doi: 10.3390/app121910167.
- Albawi, S., Mohammed, T.A. and Al-Zawi, S. 2017. Understanding of a convolutional neural network. In: 2017 International Conference on Engineering and Technology (ICET). pp. 1–6. doi: 10.1109/ICEngTechnol.2017.8308186.
- Asmara, R.A., Syahputro, B., Supriyanto, D. and Handayani, A.N. 2020. Prediction of Traffic Density Using YOLO Object Detection and Implemented in Raspberry Pi 3b + and Intel NCS 2. In: 2020 4th International Conference on Vocational Education and Training (ICOVET). Malang, Indonesia: IEEE, pp. 391–395. Available at: <https://ieeexplore.ieee.org/document/9230145/> [Accessed: 29 July 2023].
- Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y.M. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. Available at: <http://arxiv.org/abs/2004.10934> [Accessed: 4 August 2023].
- Cheng, G., Lang, C., Wu, M., Xie, X., Yao, X. and Han, J. 2021. Feature Enhancement Network for Object Detection in Optical Remote Sensing Images. *Journal of Remote Sensing* 2021, p. 2021/9805389. doi: 10.34133/2021/9805389.
- Ekvall, S., Kragic, D. and Jensfelt, P. 2007. Object detection and mapping for service robot tasks. *Robotica* 25(2), pp. 175–187. doi: 10.1017/S0263574706003237.

Evaluation measures (information retrieval). 2023. Available at: [https://en.wikipedia.org/w/index.php?title=Evaluation_measures_\(information_retrieval\)&oldid=1170522929#Average_precision](https://en.wikipedia.org/w/index.php?title=Evaluation_measures_(information_retrieval)&oldid=1170522929#Average_precision) [Accessed: 20 August 2023].

Feng, D. et al. 2021. Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges. *IEEE Transactions on Intelligent Transportation Systems* 22(3), pp. 1341–1360. doi: 10.1109/TITS.2020.2972974.

Fu, H., Song, G. and Wang, Y. 2021. Improved YOLOv4 Marine Target Detection Combined with CBAM. *Symmetry* 13(4), p. 623. doi: 10.3390/sym13040623.

Ghaffarian, S., Valente, J., van der Voort, M. and Tekinerdogan, B. 2021. Effect of Attention Mechanism in Deep Learning-Based Remote Sensing Image Processing: A Systematic Literature Review. *Remote Sensing* 13(15), p. 2965. doi: 10.3390/rs13152965.

Girshick, R. 2015. Fast R-CNN. Available at: <https://arxiv.org/abs/1504.08083> [Accessed: 29 July 2023].

Han, W. et al. 2021. Methods for Small, Weak Object Detection in Optical High-Resolution Remote Sensing Images: A survey of advances and challenges. *IEEE Geoscience and Remote Sensing Magazine* 9(4), pp. 8–34. doi: 10.1109/MGRS.2020.3041450.

He, K., Gkioxari, G., Dollár, P. and Girshick, R. 2018. Mask R-CNN. Available at: <http://arxiv.org/abs/1703.06870> [Accessed: 20 August 2023].

He, K., Zhang, X., Ren, S. and Sun, J. 2014. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. Available at: <https://arxiv.org/abs/1406.4729> [Accessed: 1 August 2023].

Hou, Q., Zhou, D. and Feng, J. 2021. Coordinate Attention for Efficient Mobile Network Design. Available at: <https://arxiv.org/abs/2103.02907> [Accessed: 29 July 2023].

Howard, A.G. et al. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Available at: <http://arxiv.org/abs/1704.04861> [Accessed: 19 August 2023].

Hu, J., Shen, L., Albanie, S., Sun, G. and Wu, E. 2017. Squeeze-and-Excitation Networks. Available at: <https://arxiv.org/abs/1709.01507> [Accessed: 29 July 2023].

Hubel, D.H. and Wiesel, T.N. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology* 160(1), pp. 106–154. doi: 10.1113/jphysiol.1962.sp006837.

Hung, J. et al. 2018. Applying Faster R-CNN for Object Detection on Malaria Images. Available at: <https://arxiv.org/abs/1804.09548> [Accessed: 29 July 2023].

Jaccard index. 2023. Available at: https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1169994638 [Accessed: 20 August 2023].

Jiang, H. and Learned-Miller, E. 2017. Face Detection with the Faster R-CNN. In: 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017). Washington, DC, DC, USA: IEEE, pp. 650–657. Available at: <http://ieeexplore.ieee.org/document/7961803/> [Accessed: 29 July 2023].

Jocher, G. 2020. YOLOv5 by Ultralytics. Available at: <https://github.com/ultralytics/yolov5> [Accessed: 9 August 2023].

Lan, W., Dang, J., Wang, Y. and Wang, S. 2018. Pedestrian Detection Based on YOLO Network Model. In: 2018 IEEE International Conference on Mechatronics and Automation (ICMA). Changchun: IEEE, pp. 1547–1551. Available at: <https://ieeexplore.ieee.org/document/8484698/> [Accessed: 29 July 2023].

Li, J., Liang, X., Shen, S., Xu, T., Feng, J. and Yan, S. 2017. Scale-aware Fast R-CNN for Pedestrian Detection. IEEE Transactions on Multimedia, pp. 1–1. doi: 10.1109/TMM.2017.2759508.

Li, Y., Wang, L. and Wang, Z. 2022. Single-Shot Object Detection via Feature Enhancement and Channel Attention. Sensors 22(18), p. 6857. doi: 10.3390/s22186857.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. and Belongie, S. 2016. Feature Pyramid Networks for Object Detection. Available at: <https://arxiv.org/abs/1612.03144> [Accessed: 1 August 2023].

Liu, S., Qi, L., Qin, H., Shi, J. and Jia, J. 2018. Path Aggregation Network for Instance Segmentation. Available at: <https://arxiv.org/abs/1803.01534> [Accessed: 1 August 2023].

McCulloch, W.S. and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5(4), pp. 115–133. doi: 10.1007/BF02478259.

Morabbi, S., Soltanizadeh, H., Mozaffari, S. and Fadaeieslam, M.J. 2023. Improving generalization in deep neural network using knowledge transformation based on fisher criterion. *The Journal of Supercomputing*. Available at: <https://doi.org/10.1007/s11227-023-05448-0> [Accessed: 30 July 2023].

Nickolls, J. and Dally, W.J. 2010. The GPU Computing Era. *IEEE Micro* 30(2), pp. 56–69. doi: 10.1109/MM.2010.41.

Niu, Z., Zhong, G. and Yu, H. 2021. A review on the attention mechanism of deep learning. *Neurocomputing* 452, pp. 48–62. doi: 10.1016/j.neucom.2021.03.091.

Obeso, A.M., Benois-Pineau, J., García Vázquez, M.S. and Acosta, A.Á.R. 2022. Visual vs internal attention mechanisms in deep neural networks for image classification and object detection. *Pattern Recognition* 123, p. 108411. doi: 10.1016/j.patcog.2021.108411.

O'Shea, K. and Nash, R. 2015. An Introduction to Convolutional Neural Networks. Available at: <http://arxiv.org/abs/1511.08458> [Accessed: 4 August 2023].

Precision and recall. 2023. Available at: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=1163655774 [Accessed: 20 August 2023].

PyCharm Community Edition and Professional Edition Explained: Licenses and More. 2017. Available at: <https://blog.jetbrains.com/pycharm/2017/09/pycharm-community-edition-and-professional-edition-explained-licenses-and-more/> [Accessed: 5 August 2023].

PyTorch. 2017. Available at: <https://www.pytorch.org> [Accessed: 9 August 2023].
Qiu, S., Wu, Y., Anwar, S. and Li, C. 2021. Investigating Attention Mechanism in 3D Point Cloud Object Detection. In: 2021 International Conference on 3D Vision (3DV). pp. 403–412. doi: 10.1109/3DV53792.2021.00050.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. 2015. You Only Look Once: Unified, Real-Time Object Detection. Available at: <https://arxiv.org/abs/1506.02640> [Accessed: 29 July 2023].

Roboflow: Give your software the power to see objects in images and video. 2023. Available at: <https://roboflow.com/> [Accessed: 30 July 2023].

Shi, J., Yang, J. and Zhang, Y. 2022. Research on Steel Surface Defect Detection Based on YOLOv5 with Attention Mechanism. Electronics 11(22), p. 3735. doi: 10.3390/electronics11223735.

Spinellis, D. 2012. Git. IEEE Software 29(3), pp. 100–101. doi: 10.1109/MS.2012.61.

Sun, G., Wang, S. and Xie, J. 2023. An Image Object Detection Model Based on Mixed Attention Mechanism Optimized YOLOv5. *Electronics* 12(7), p. 1515. doi: 10.3390/electronics12071515.

Surantha, N. and Wicaksono, W.R. 2018. Design of Smart Home Security System using Object Recognition and PIR Sensor. *Procedia Computer Science* 135, pp. 465–472. doi: 10.1016/j.procs.2018.08.198.

Tzutalin. 2015. LabelImg. Available at: <https://github.com/tzutalin/labelImg>.
Ultralytics. 2023. Architecture Summary. Available at: https://docs.ultralytics.com/yolov5/tutorials/architecture_description [Accessed: 1 August 2023].

Vaswani, A. et al. 2023. Attention Is All You Need. Available at: <http://arxiv.org/abs/1706.03762> [Accessed: 19 August 2023].

Welcome to Python.org. 2023. Available at: <https://www.python.org/> [Accessed: 9 August 2023].

Woo, S., Park, J., Lee, J.-Y. and Kweon, I.S. 2018. CBAM: Convolutional Block Attention Module. Available at: <https://arxiv.org/abs/1807.06521> [Accessed: 29 July 2023].

Wu, T., Huang, J., Gao, G., Wei, X., Wei, X., Luo, X. and Liu, C.H. 2021. Embedded Discriminative Attention Mechanism for Weakly Supervised Semantic Segmentation. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 16760–16769. doi: 10.1109/CVPR46437.2021.01649.

Zhang, Y., Guo, Z., Wu, J., Tian, Y., Tang, H. and Guo, X. 2022. Real-Time Vehicle Detection Based on Improved YOLO v5. *Sustainability* 14(19), p. 12274. doi: 10.3390/su141912274.

Zheng, J., Wang, T., Zhang, Z. and Wang, H. 2022. Object Detection in Remote Sensing Images by Combining Feature Enhancement and Hybrid Attention. *Applied Sciences* 12(12), p. 6237. doi: 10.3390/app12126237.

Zhu, X., Lyu, S., Wang, X. and Zhao, Q. 2021. TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios. Available at: <https://arxiv.org/abs/2108.11539> [Accessed: 29 July 2023].