成　绩	
评阅人	

复 旦 大 学

研 究 生 课 程 论 文

论文题目：　基于 YOLO 和 DAMO-YOLO 的电动车进电梯识别检测

修读课程：　神经网络与深度学习

选课学期：　2023-2024 学年第一学期

选课学生：　韩佳悦（22262010013）

完成日期：　2024.1.4

基于 YOLO 和 DAMO-YOLO 的电动车进电梯识别检测

1. 引言

在当今，电动车作为一种环保、便捷的交通工具，成为很多人城市出行的选择。然而，电动车在日常使用中的管理也面临一系列难题。特别是在多层建筑中，居民将电动车带入电梯的问题难以监管。一方面，电动车的电池可能在特定条件下发生火灾或爆炸，而在封闭的电梯空间内，这类事故将避无可避地对乘客和电梯设备造成极大危险。另一方面，将电动车等较大设备通过电梯运送上楼会占用大量空间，不但影响环境也会阻挡特殊危险情况下居民的逃生通道。

据统计，近年来由于电动车带入电梯所引发的直接或间接安全事故时有发生。行为者多为居民，其中一些人安全意识淡薄或心存侥幸，不服从管理。此外，电梯分布广泛、零散，使得常规的监管手段难以有效实施。

为了减少电动车进入电梯的情况和带来的安全隐患，本文基于 YOLO 和 DAMO-YOLO 目标检测算法实现电动车进入电梯的识别预警功能。该系统不仅能够判断图中是否存在电动车，并标记出人和电动车的位置，还能实时监测电动车在电梯区域的行为，有效提升了管理和安全预防的效果。

2. 数据集的准备：

2.1 数据的获取

通过多种方式获取采集数据，主要包括网络爬取，现场拍摄。将采集到的数据进行预处理和筛选，将质量差，内容不相符合，遮挡严重的样本数据去除后，保留质量较好的 28 张样本图片。

2.2 数据增强

在深度学习中，一般算法要求有充足的样本数量。当前样本数量较少，需要对样本进行数据增强处理，以提高样本质量。

数据增强通过产生相似但又不相同的训练样本，从而扩大训练数据集的规模。图像增广的意义是，随机改变训练样本可以降低模型对某些属性的依赖性，从而提高模型的泛化能力。主要方法和实现如下：

2.2.1 对图像进行一定比例的缩放；

```python
# 缩放
def scale_pic(image, scale):
    return cv2.resize(image, None, fx=scale, fy=scale)
```

2.2.2 对图像进行随机位置的裁剪；

实际场景中被检测物体并不会确定的出现在图像的某一位置，通过随机裁剪让物体以不同的比例出现在图像的不同位置，这样能够降低模型对目标位置的敏感性。

```python
# 旋转+缩放
def rotate_and_scale(img, angele=15, scale=1.1):
    w = img.shape[0]
    h = img.shape[1]
    matrix = cv2.getRotationMatrix2D((w/2, h/2), angele, scale)
    ret = cv2.warpAffine(img, matrix, (w,h))
    return ret
```

2.2.3 对图像进行水平和垂直的翻转：

和垂直翻转是图像处理中常用的增强手段，它们被广泛采用于改善模型的鲁棒性和泛化性能。水平翻转是指沿着图像的水平轴进行翻转，而垂直翻转则是沿垂直轴进行翻转。这两种翻转操作在目标检测领域中被视为经典的数据增强方法，它们的应用通常不会影响物体的类别信息，而更侧重于引入多样性以提高模型对不同场景的适应能力。

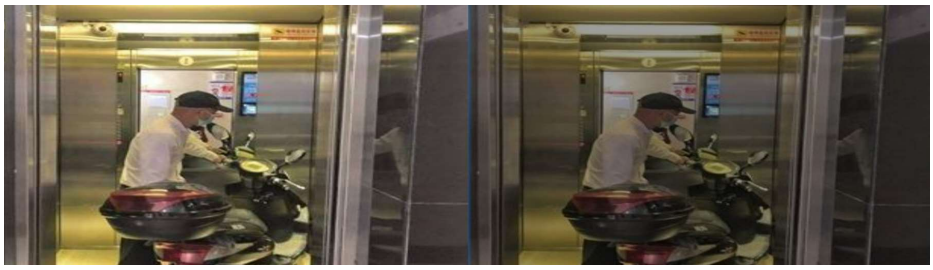
在实际的目标检测任务中，对图像进行水平和垂直的翻转操作有助于缓解模型对镜像和旋转变化的敏感性，从而提升其在复杂场景下的性能。这些翻转操作不仅能够增加数据集的规模，而且通过引入更多变化，有助于模型更全面地理解目标物体的外观特征，使其更具有泛化能力。

```python
# 翻转
def horizontal_reverse(image):
    return cv2.flip(image, 1, dst=None)


def vertical_reverse(image):
    return cv2.flip(image, 0, dst=None)
```

2.2.4 对图像进行亮度、对比度、颜色等的随机变化。

为了进一步丰富数据集，提高模型的鲁棒性以及对于真实场景的适应性，我们采用了图像亮度、对比度和颜色等方面的随机变化操作。这一数据增强手段不仅有助于提高模型的泛化性，还能够模拟真实场景中不同光照和环境条件下的图像变化，使模型更具健壮性。

```python
def brighter_or_darker(img, prec=1.2):
    img_copy = img.copy()
    w = img.shape[0]
    h = img.shape[1]
    for i in range(0, w):
        for j in range(0, h):
            for k in range(3):
                img_copy[i,j,k] =np.clip(int(img[i,j,k])*prec, a_max=255, a_min=0)
    return img_copy
```



2.2.5 在图像中加入随机噪声

为了进一步提升数据集的多样性，我们引入了随机噪声，这有助于使模型更好地适应各种实际环境中可能存在的噪声。在数据增强的过程中，我们使用

了两种常见的随机噪声：椒盐（脉冲）噪声和高斯噪声。下面分别介绍了其实现代码和效果。

1）椒盐（脉冲）噪声：

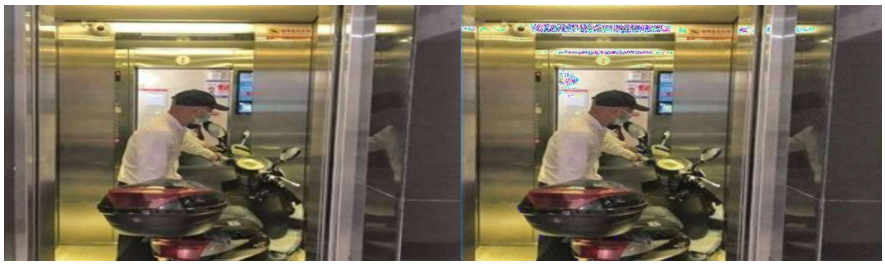
椒盐噪声是一种将图像中的随机像素点替换为黑色或白色的噪声类型。这种噪声模拟了图像中可能存在的强烈干扰或损坏情况。代码与效果如下：

```
# 噪声 - 椒盐噪声
def AddSaltPepperNoise(src, rate=0.05):
    srcCopy = src.copy()
    height, width = srcCopy.shape[0:2]
    noiseCount = int(rate*height*width/2)
    # add salt noise
    X = np.random.randint(width,size=(noiseCount,))
    Y = np.random.randint(height,size=(noiseCount,))
    srcCopy[Y, X] = 255
    # add black peper noise
    X = np.random.randint(width,size=(noiseCount,))
    Y = np.random.randint(height,size=(noiseCount,))
    srcCopy[Y, X] = 0
    return srcCopy
```

2）斯噪声：

高斯噪声是一种在图像中引入具有高斯分布的随机值的方法，模拟了图像中的连续性噪声。实现代码和效果如下

```
# 噪声 - 高斯噪声
def AddGaussNoise(src,sigma=5):
    mean = 0
    # 获取图片的高度和宽度
    height, width, channels = src.shape[0:3]
    gauss = np.random.normal(mean,sigma,(height,width,channels))
    noisy_img = np.uint8(src + gauss)
    return noisy_img
```



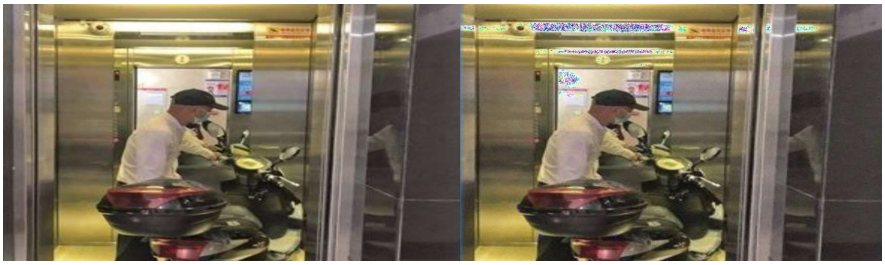
通过这两种噪声的引入，我们使得数据集更具挑战性，模拟了在实际应用中可能会遇到的各种噪声干扰。这有助于提高模型的鲁棒性，使其更好地适应真实世界的复杂条件。这样的数据增强策略对于电动车进入电梯行为检测模

型的训练具有积极的影响。

2.2.6 高斯模糊：

由于实际应用场景中，受到设备采集数据的质量和精度的限制，导致模糊程度存在差异。为应对这一挑战，我们引入了高斯模糊。以下是码及其在任务中产生的效果。

```python
# 高斯模糊
def blur(img):
    return cv2.GaussianBlur(img, (5,5), 1)
```

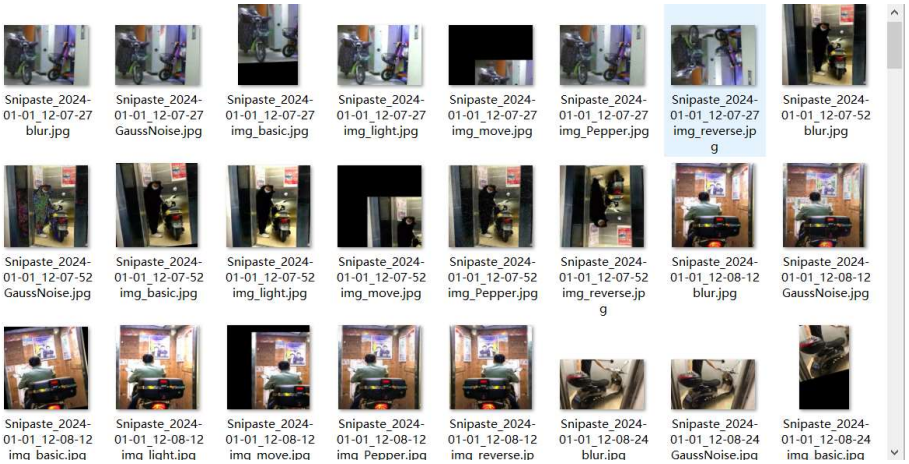


2.2.7 整合实现：

整合上述的数据增强方法，并加入随机量，来整体的进行数据增强，达到扩充数据集，提高样本质量的作用。以下是相应的代码实现：

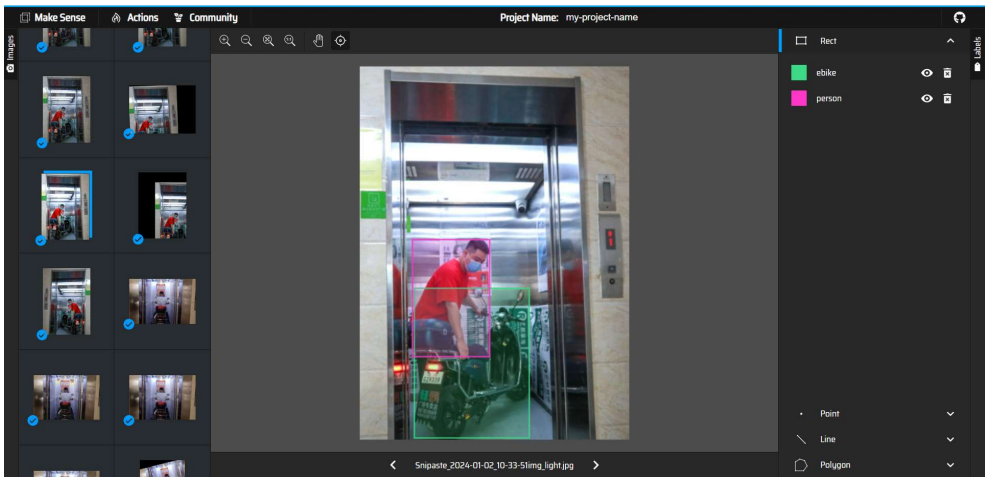
```python
# ----------------------------------------
def opt_on_dir():
    root_path = r"C:\Users\13917\Desktop\2023_deeplearning\data_improvement\ebike\images"
    save_path = r"C:\Users\13917\Desktop\2023_deeplearning\data_improvement\ebike\results"
    for a,b,c in os.walk(root_path):
        for file_i in c:
            file_i_path = os.path.join(a, file_i)
            print(file_i_path)
            img_i = cv2.imread(file_i_path)

            img_blur = blur(img_i)
            cv2.imwrite(os.path.join(save_path, file_i[:-4]+"blur.jpg"), img_blur)

            img_AddGaussNoise = AddGaussNoise(img_i)
            cv2.imwrite(os.path.join(save_path, file_i[:-4] + "GaussNoise.jpg"), img_AddGaussNoise)

            img_Pepper = AddSaltPepperNoise(img_i)
            cv2.imwrite(os.path.join(save_path, file_i[:-4] + "img_Pepper.jpg"), img_Pepper)

            img_light = brighter_or_darker(img_i,random.uniform(0.6, 1.5))
            cv2.imwrite(os.path.join(save_path, file_i[:-4] + "img_light.jpg"), img_light)

            img_basic = rotate_and_scale(img_i, random.randint(0, 15), random.uniform(0.8, 1.2))
            cv2.imwrite(os.path.join(save_path, file_i[:-4] + "img_basic.jpg"), img_basic)

            img_move = move_img(img_i, random.randint(0, int(img_i.shape[0]*0.5)), random.randint(0, int(img_i.shape[1]*0.5)))
            cv2.imwrite(os.path.join(save_path, file_i[:-4] + "img_move.jpg"), img_move)

            if random.randint(0, 1):
                img_reverse = horizontal_reverse(img_i)
            else:
                img_reverse = vertical_reverse(img_i)
            cv2.imwrite(os.path.join(save_path, file_i[:-4] + "img_reverse.jpg"), img_reverse)


if __name__ == '__main__':
    # test_one_sample()
    opt_on_dir()
```

数据集扩充至 224 张



2.3 数据标注

使用 makesense.ai 工具对数据集进行标注，将标签定义为两类，分别为：person 和 ebike。标注完成后，生成并导出 YOLO 格式与 VOC 格式的标注文件，预备下一步处理。



3. YOLO 模型下的训练与优化：

3.1 数据划分与预处理

YOLO 模型使用的数据集格式为 YOLO 格式，在 YOLO 文件夹下创建存储训练数据的文件夹，文件夹目录结构如下，并将之前准备好的数据集图片和 xml 文件，拆分为训练集和测试集，分别放入对应位置，目录结构如下：

- Images
  - Train
  - Test
- Labels
  - Train
  - Test

3.2 模型的选择，训练与优化

YOLO（You Only Look Once）是一种目标检测算法，它的主要特点是能够在一张图像中同时检测多个目标，而不需要对图像进行多次处理。YOLO 将目标检测任务转化为一个回归问题，通过单个前向传递就能够同时预测多个边界框及其对应的类别概率。YOLO 模型的速度快，因为它不需要复杂的多次卷积和非极大值抑制（Non-Maximum Suppression）步骤。由于其性能优越，速度快，易进行模型的集成和定制，更新和维护，广泛应用在目标识别领域。

准备好 YOLO 模型源码，这里使用的是 YOLO v5-6.0 版本，创建虚拟环境，并根据 read.me 的介绍做好相应的环境准备。

YOLO v5 提供 s, m, l, x 版本的不同模型配置文件，相对应的架构增大，训练时间增长。根据本功能的应用场景，我们选择 m，即'yolov5m.pt'。在源码中训练文件中更改使用模型。

```
parser.add_argument('--weights', type=str, default=ROOT /

'yolov5m.pt', help='initial weights path')
```

在 data 文件夹下创建 data/ebike.yaml 文件，定义数据集信息

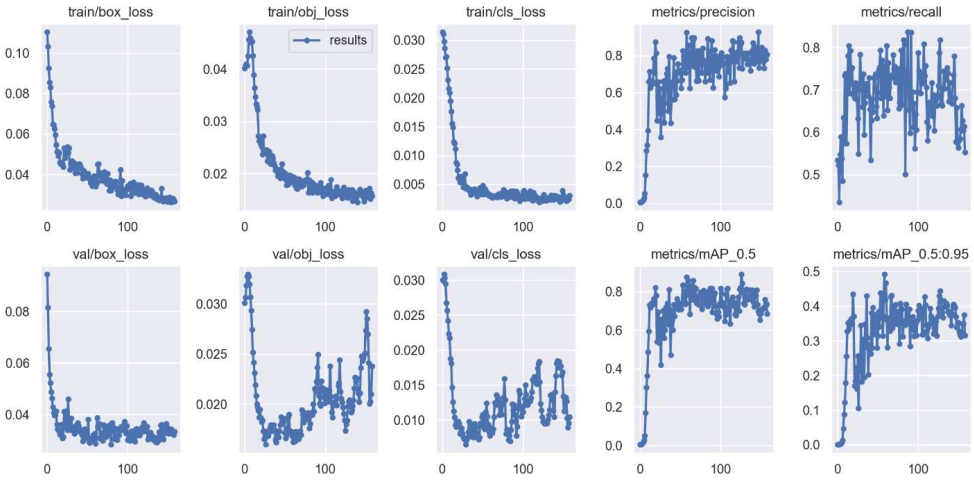
```yaml
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..
path: ./datasets/dataset_ebike_yolo  # dataset root dir
train: images/train  # train images (relative to 'path') 128 images
val: images/test  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes
nc: 2  # number of classes
names: ['ebike', 'person']  # class names
```

在源码中训练文件中将 ebike.yaml 加入，定义使用的数据集：

```
parser.add_argument('--data', type=str, default=ROOT /

'data/ebike.yaml', help='dataset.yaml path')
```

调整 epochs, learning-rate,batch-size 等超参数，并多次进行微调，使结果的准确率尽可能高。

在经过调整后，损失函数降低，识别的准确率和精确度可接受，保存最佳的权重数据 best.pt



3.3 模型的迁移和应用

在模型训练完成后，我们使用 detect 库加载训练得出的最佳权重数据，进行推理和结果的测试。这一阶段涉及将经过训练的模型应用于实际场景，验证其在未见过的数据上的性能表现。

在进行推理过程中，我们选择了一条测试视频，将其输入到经过训练的模型中。通过模型对视频帧进行分析，我们得到了对电梯中的人和电动车进行识别的结果。

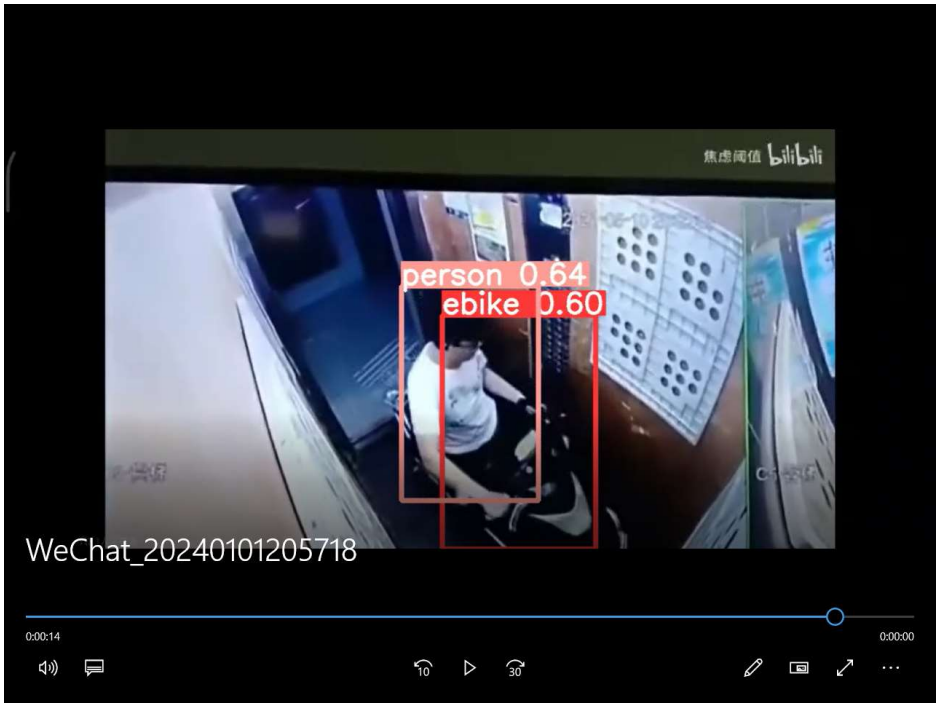
根据推理结果，我们可以在视频中标注出电梯内出现的人和电动车的位置。这项工作为后续分析提供了基础，使得我们能够详细了解模型在真实场景中的行为表现。

标注的结果不仅有助于模型的性能评估，还为进一步的优化提供了有价值的信息。

此外，通过模型的迁移和应用，我们可以评估模型在不同环境、不同数据集上的通用性。这有助于确定模型的泛化能力，从而更好地理解模型在实际应用中的可行性和可靠性。

```
parser.add_argument('--weights', nargs='+', type=str, default=ROOT /

r'runs/train/exp5/weights/best.pt', help='model path(s)')
```

```
parser.add_argument('--source', type=str, default=ROOT /

r'C:\Users\13917\Desktop\2023_deeplearning\WeChat_20240101205718.mp4', help=
'file/dir/URL/glob, 0 for webcam')
```



4. DAMO-YOLO 模型下的训练与优化

4.1 数据划分，预处理与 modelscope 平台的数据集卡片制作：

DAMO-YOLO 模型采用 coco 数据集，由于标注工具的导出形式并没有 coco 类型，我们使用 voc 形式导出，并通过脚本将其转换为 coco 类型，并实现训练集与测试集的划分，代码部分详见附件，运行结果如下：

```
(DAMO-YOLO) C:\Users\13917\Desktop\2023_deeplearning\data_improvement\ebike\dataset_ebike_voc2coco>python voc2coco.py
{'ebike': 1, 'person': 2}
xml_dir is ./datasets/Annotations
训练样本数目是 136
验证样本数目是 19
测试样本数目是 40
------------create ./datasets/train.json done--------------
find 2 categories: dict_keys(['ebike', 'person']) -->>> your pre_define_categories 2: dict_keys(['ebike', 'person'])
category: id --> {'ebike': 1, 'person': 2}
dict_keys(['ebike', 'person'])
dict_values([1, 2])
------------create ./datasets/val.json done--------------
find 2 categories: dict_keys(['ebike', 'person']) -->>> your pre_define_categories 2: dict_keys(['ebike', 'person'])
category: id --> {'ebike': 1, 'person': 2}
dict_keys(['ebike', 'person'])
dict_values([1, 2])
------------create ./datasets/test.json done--------------
find 2 categories: dict_keys(['ebike', 'person']) -->>> your pre_define_categories 2: dict_keys(['ebike', 'person'])
category: id --> {'ebike': 1, 'person': 2}
dict_keys(['ebike', 'person'])
dict_values([1, 2])
```

将数据集按照格式打包，在 modelscope 平台根据要求制作数据集卡片：

4.2 模型的选择，训练与优化

DAMO-YOLO 是由阿里巴巴达摩院智能计算实验室 TinyML 团队研发的一款注重速度与精度兼顾的目标检测框架。相较于传统的 YOLO 框架，DAMO-YOLO 引入了一系列新技术，对整个检测框架进行了全面的优化与改进。

DAMO-YOLO 的模型包括了 DAMO-YOLO-T、DAMO-YOLO-S、DAMO-YOLO-M 等不同版本，以满足不同场景和需求的要求。在考虑我们项目的应用场景以及未来可能的部署环境时，我们精心选择了 DAMO-YOLO-S 模型，以在保证检测精度的同时，充分利用其在速度方面的卓越表现。

选择 DAMO-YOLO-S 的理由在于其轻量级设计和高效率，适用于在有限计算资源下进行目标检测的场景。这符合我们项目对实时性和资源消耗的双重要求。通过采用 DAMO-YOLO-S，

4.3 模型的参数调整和训练

在进行模型的参数调整和训练阶段，我们首先导入了专门为项目创建的数据集。为了确保数据的正常调用，我们采用了迭代器的方式获取数据，确认数据可以正常调用：

```
[18]: # Step 1: 数据集准备，可以使用modelscope上已有的数据集，也可以自己在本地构建COCO数据集
train_dataset =  MsDataset.load('etoiles/ebike_detect',  namespace="modelscope", split='train', download_mode=Downlo
val_dataset = MsDataset.load('etoiles/ebike_detect', namespace="modelscope", split='validation', download_mode=Downl

2024-01-02 14:57:15,693 - modelscope - INFO - No subset_name specified, defaulting to the default
2024-01-02 14:57:16,139 - modelscope - INFO - Generating dataset ebike_detect (/root/.cache/modelscope/hub/dataset
s/modelscope___ebike_detect/master/train)
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]
100%
Downloading data files: 100%|██████████| 1/1 [00:03<00:00,  3.19s/it]
Extracting data files: 100%|██████████| 1/1 [00:00<00:00,  5.29it/s]
2024-01-02 14:57:20,332 - modelscope - INFO - No subset_name specified, defaulting to the default
2024-01-02 14:57:20,761 - modelscope - INFO - Generating dataset ebike_detect (/root/.cache/modelscope/hub/dataset
s/modelscope___ebike_detect/master/validation)
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]
100%
Downloading data files: 100%|██████████| 1/1 [00:03<00:00,  3.16s/it]
Extracting data files: 100%|██████████| 1/1 [00:00<00:00,  5.31it/s]

[19]: print(next(iter(train_dataset)))

('train', '/root/.cache/modelscope/hub/datasets/etoiles/ebike_detect/master/data_files/extracted/095c6b82bc4aa54548
2f29cdc212e02ef2b4efce1c5e7e24fc17d8adc63289f5')
```

设置调整相关参数：

在模型的参数调整和训练阶段，合理设置关键参数对于模型性能的提升至关重要。以下是我们所多次进行的一系列参数调整与设置：

学习率（Learning Rate）的优化： 学习率直接影响模型权重的更新速度，我们通过学习率的调整来平衡模型在训练中的收敛速度和稳定性。采用了初始较大的学习率以快速收敛，然后通过学习率衰减策略，逐渐减小学习率，提高模型对于数据的学习能力，降低过拟合的风险。

批量大小（Batch Size）的选择： 批量大小直接关系到模型权重的更新频率，我们进行了多次实验，选取了适中的批量大小，以在保证模型训练效率的同时，不引入过多的噪声。

迭代次数（Epochs）的设定： 我们通过多次迭代实验，选择了适当的迭代次数，以确保模型在训练过程中充分学习数据集的特征。我们监控训练过程中的损失函数变化，确保模型达到稳定的收敛状态。

权重初始化（Weight Initialization）的策略： 模型参数的初始值对于训练的起始状态至关重要，我们采用了一种有效的权重初始化策略，以提高模型的训练效果。通过这些参数的多次尝试与调整，我们在模型的训练过程中更好地平衡了模型的收敛速度和泛化能力。

```
# Step 2: 相关参数设置
train_root_dir = train_dataset.config_kwargs['split_config']['train']
val_root_dir = val_dataset.config_kwargs['split_config']['validation']
train_img_dir = osp.join(train_root_dir, 'images')
val_img_dir = osp.join(val_root_dir, 'images')
train_anno_path = osp.join(train_root_dir, 'train.json')
val_anno_path = osp.join(val_root_dir, 'val.json')
kwargs = dict(
            model='damo/cv_tinynas_object-detection_damoyolo', # 使用DAMO-YOLO-S模型
            gpu_ids=[  # 指定训练使用的gpu
                0,
            ],
            batch_size=16, # batch_size，每个gpu上的图片数等于batch_size // len(gpu_ids)
            max_epochs=300, # 总的训练epochs
            num_classes=2, # 自定义数据中的类别数
            load_pretrain=True, # 是否载入预训练模型，若为False，则为从头重新训练
            base_lr_per_img=0.001, # 每张图片的学习率，lr=base_lr_per_img*batch_size
            train_image_dir=train_img_dir, # 训练图片路径
            val_image_dir=val_img_dir, # 测试图片路径
            train_ann=train_anno_path, # 训练标注文件路径
            val_ann=val_anno_path, # 测试标注文件路径
            )
```

开启训练任务：

```
[38]: # Step 3: 开启训练任务
trainer = build_trainer(
            name=Trainers.tinynas_damoyolo, default_args=kwargs)
trainer.train()
```

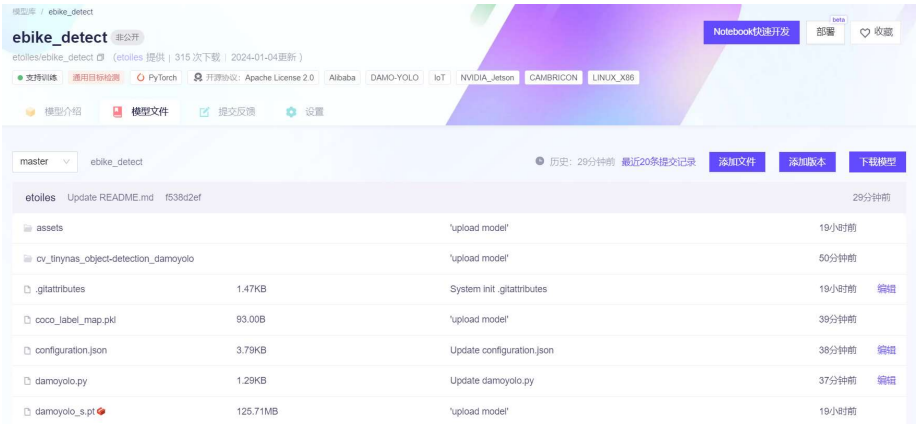
```
2024-01-02 15:11:34,108 - modelscope - WARNING - Model revision not specified, use revision: v1.2.2
2024-01-02 15:11:34,981 - modelscope - WARNING - Model revision not specified, use revision: v1.2.2
2024-01-02 15:11:35,673 - modelscope - WARNING - Model revision not specified, use revision: v1.2.2
2024-01-02 15:11:35,921 - modelscope - INFO - args info: None
2024-01-02 15:11:35,922 - modelscope - INFO - cfg value:
{'miscs': {'print_interval_iters': 50, 'exp_name': 'damoyolo_s', 'seed': 1234, 'eval_interval_epochs': 1, 'ckpt_i
nterval_epochs': 1, 'num_workers': 4, 'output_dir': './workdirs'}, 'model': {'type': 'tinynas-damoyolo', 'weight
s': 'damoyolo_tinynasL25_S.pt', 'backbone': {'name': 'TinyNAS_res', 'structure_file': '/mnt/workspace/.cache/mode
lscope/damo/cv_tinynas_object-detection_damoyolo/tinynas_L25_k1kx.txt', 'out_indices': [2, 4, 5], 'with_spp': Tru
e, 'use_focus': True, 'act': 'relu', 'reparam': True}, 'neck': {'name': 'GiraffeNeckV2', 'depth': 1.0, 'hidden_ra
tio': 0.75, 'in_channels': [128, 256, 512], 'out_channels': [128, 256, 512], 'act': 'relu', 'spp': False, 'block_
name': 'BasicBlock_3x3_Reverse'}, 'head': {'name': 'ZeroHead', 'num_classes': 2, 'in_channels': [128, 256, 512],
'stacked_convs': 0, 'reg_max': 16, 'act': 'silu', 'nms_conf_thre': 0.05, 'nms_iou_thre': 0.7}}, 'dataset': {'trai
n_image_dir': '/root/.cache/modelscope/hub/datasets/machineman/outline_dog_test/master/data_files/extracted/e59ce
57aaed99b03011fc97f1f96830d1a27c8781f661b0960130b631be281da/images', 'val_image_dir': '/root/.cache/modelscope/hu
b/datasets/machineman/outline_dog_test/master/data_files/extracted/74ce61d72b2ae0f7aae44b792ceecf4e65bf69238e1435
f54430138fcfcc4207/images', 'train_ann': '/root/.cache/modelscope/hub/datasets/machineman/outline_dog_test/maste
r/data_files/extracted/e59ce57aaed99b03011fc97f1f96830d1a27c8781f661b0960130b631be281da/train.json', 'val_ann':
```

4.4 模型的迁移和应用

模型训练成功后我们得到权重参数和自己的模型，参照 modelscope 平台文档，将自己的模型和训练后的权重数据上传平台，搭建自己的模型.

```
from modelscope.hub.api import HubApi

YOUR_ACCESS_TOKEN = '36ea7026-a961-4b13-94c5-fe625c3ff90b'
# 请注意ModelScope平台针对SDK访问和git访问两种模式，提供两种不同的访问令牌(token)。此处请使用SDK访问令牌。


api = HubApi()
api.login(YOUR_ACCESS_TOKEN)
api.push_model(
    model_id="etoiles/ebike_detect",
    model_dir="cv_tinyness_damoyolo_trained" # 本地模型目录，要求目录中必须包含configuration.json
)
```

```
2024-01-04 11:25:43,098 - modelscope - INFO - [master a53007f] 'upload model'
 1 file changed, 0 insertions(+), 0 deletions(-)
```



再分别修改 json 文件，py 文件和 pkl 文件中的类别数量数据，进行下一步的推理。实现代码和效果如下：

```
from modelscope.pipelines import pipeline
from modelscope.utils.constant import Tasks
from modelscope.outputs import OutputKeys
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

ebike_detector = pipeline(Tasks.image_object_detection, model='etoiles/ebike_detect')
result = ebike_detector('test_ebike.jpg')
# 打印结果
print(result)
```

```
Downloading: 100%|██████████| 3.79k/3.79k [00:00<00:00, 18.4MB/s]
Downloading: 100%|██████████| 1.29k/1.29k [00:00<00:00, 6.74MB/s]
Downloading: 100%|██████████| 3.50k/3.50k [00:00<00:00, 16.6MB/s]
Downloading: 100%|██████████| 3.29k/3.29k [00:00<00:00, 18.5MB/s]
2024-01-04 11:28:54,142 - modelscope - INFO - initiate model from /mnt/workspace/.cache/modelscope/etoiles/ebike_detect
2024-01-04 11:28:54,143 - modelscope - INFO - initiate model from location /mnt/workspace/.cache/modelscope/etoiles/ebike_detect.
2024-01-04 11:28:54,146 - modelscope - INFO - initialize model from /mnt/workspace/.cache/modelscope/etoiles/ebike_detect
2024-01-04 11:28:54,579 - modelscope - WARNING - No val key and type key found in preprocessor domain of configuration.json file.
2024-01-04 11:28:54,580 - modelscope - WARNING - Cannot find available config to build preprocessor at mode inference, current conf
ig: {'train': {'mosaic_mixup': {'mosaic_prob': 1.0, 'mosaic_scale': [0.1, 2.0], 'mosaic_size': [640, 640], 'mixup_prob': 0.15, 'mix
up_scale': [0.5, 1.5], 'degrees': 10.0, 'translate': 0.2, 'shear': 2.0}, 'transform': {'image_mean': [0.0, 0.0, 0.0], 'image_std':
[1.0, 1.0, 1.0], 'image_max_range': [640, 640], 'flip_prob': 0.5, 'autoaug_dict': {'box_prob': 0.3, 'num_subpolicies': 5, 'scale_sp
lits': [2048, 10240, 51200], 'autoaug_params': [6, 9, 5, 3, 3, 4, 2, 4, 4, 4, 5, 2, 4, 1, 4, 2, 6, 4, 2, 2, 2, 6, 2, 2, 2, 0, 5, 1,
3, 0, 8, 5, 2, 8, 7, 5, 1, 3, 3, 3]}}}, 'evaluation': {'transform': {'image_mean': [0.0, 0.0, 0.0], 'image_std': [1.0, 1.0, 1.0],
'image_max_range': [640, 640], 'flip_prob': 0.0}}, 'model_dir': '/mnt/workspace/.cache/modelscope/etoiles/ebike_detect'}. trying to
build by task and model information.
{'scores': array([0.63988245, 0.63087445], dtype=float32), 'labels': ['ebike', 'ebike'], 'boxes': array([[2534.7864 ,  924.97076, 2
842.7017 , 1280.0604 ],
       [2012.3676 , 1602.6377 , 2555.1528 , 1934.1534 ]], dtype=float32)}
```

最终识别效果如下图，可以看到 person 和 ebike 均被准确识别出。

```
576 928
576 928
0.94510627 ebike [496.47012 358.39752 688.8341  575.07733]
0.87106603 person [396.28577 261.36993 556.4437  579.96545]
```

5. 总结

在本篇文章中，我们实现了基于 YOLOv5 和 DAMO-YOLO 的电动车进入电梯行为检测任务。首先，我们构建了一个多样性的电动车进电梯场景数据集，以确保模型在真实世界的各种复杂环境中表现出色。通过精心设计的数据增强处理，我们提高了数据集的多样性，增强了模型的鲁棒性和泛化性。

在模型训练阶段，我们选择了 YOLOv5 和 DAMO-YOLO 两个先进的目标检测模型。YOLOv5 以其高性能和实时性而备受推崇，而 DAMO-YOLO 则可能融合了一些优化和变种，为我们提供了另一种实用有效便捷使用的模型。通过在构建的数据集上进行训练，我们能供成功地且较为准确地检测到电动车进入电梯的行为。

在模型的迁移与部署阶段， modelscope 平台为我们提供了一个方便而高效的方式，将研究成果迁移到实际应用中。modelscope 的灵活性和易用性使得模型的部署过程更加流畅，为将算法落地提供了可靠的支持。

通过本研究，我们为电动车进电梯行为检测提供了一种可行的解决方案，并展示了两种先进目标检测模型的成功应用，但仍有不足之处。未来，我们将进一步优化模型、扩展数据集，以适应更广泛的场景，并探索其他可能的性能提升途径。