

# 期中作业：微调与目标检测

何益涵 20307110032

吕文韬 23210180109

**摘要:** 本项目的仓库地址为[该超链接<sup>1</sup>](#). 详见其中的文件夹 `group_task2`. 其中包含微调任务 `task1_finetuning` 和目标检测任务 `task2_object_detection`.

本项目提供了训练好的模型, 详见[百度网盘<sup>2</sup>](#), 若链接失效, 也可以通过 Github Issue 联系作者.

---

<sup>1</sup><https://github.com/HeDesertFox/Neural-Networks-and-Deep-Learning-Homework-Group-Tasks.git>

<sup>2</sup>提取码 7gky

# 第一章 微调在 ImageNet 上预训练的卷积神经网络实现鸟类识别

## 第 1 节 任务描述

1. 修改现有的 CNN 架构（如 AlexNet，ResNet-18）用于鸟类识别，通过将其输出层大小设置为 200 以适应数据集中的类别数量，其余层使用在 ImageNet 上预训练得到的网络参数进行初始化；
2. 在 [CUB-200-2011] 数据集上从零开始训练新的输出层，并对其余参数使用较小的学习率进行微调；
3. 观察不同的超参数，如训练步数、学习率，及其不同组合带来的影响，并尽可能提升模型性能；
4. 与仅使用 CUB-200-2011 数据集从随机初始化的网络参数开始训练得到的结果进行对比，观察预训练带来的提升.

## 第 2 节 项目架构

此任务的所有文件在 `task1_finetuning` 文件夹中，其中包含以下文件：

1. `data_loading_preprocessing.py` 文件：包含数据下载函数与预处理函数.
2. `model.py` 文件：包含网络构造函数.
3. `training_fine_tuning.py` 文件：包含训练和超参数调优函数.
4. `main_notebook.ipynb` 文件：包含微调任务的全部流程，如数据加载、调参和训练可视化.

## 第 3 节 实验设置

### 3.1. 数据增广

本实验根据 CUB-200-2011 中的 `train_test_split.txt` 文件对数据集进行划分，形成训练集和验证集. 在数据预处理中，对训练集的数据进行了图像增强，其中包括了

随机水平翻转和随机旋转.

### 3.2. 模型选择

此项目的模型构建函数可以生成两种 CNN 架构: Alexnet 和 ResNet-18, 并且可以选择采用 Imagenet 预训练的参数初始化或随机初始化. 生成的模型的最后一层被替换为输出维数为 200 的全连接层以适应本任务的要求.

为了实现更高的模型性能, 本实验采用 ResNet-18 架构, 使用者可以在模型构造函数中输入参数"alexnet" 将模型改为 Alexnet.

### 3.3. 优化器设置

本实验的优化器均选择 SGD 优化器, 在所有实验中都采用 0.9 的动量以加速收敛, 采用 1e-3 的权重衰减缓解过拟合.

在后续的调参和训练过程中, 预训练模型的学习率在最后一层都正常设置, 其余层的学习率都除 10.

## 第 4 节 调参结果

本实验调节两个参数以尽可能提高模型性能: 学习率 (lr) 和训练轮数 (epoch).

总体来说, 增加训练轮数不会减少验证集上的准确率. 这是因为 ResNet-18 的表现力足够强, 一般可以在训练集上将精度提升到接近 100%, 此时梯度几乎为零, 因此最终验证集精度不会因为训练过久而下降.

经过若干次调参, 预训练模型的最后一次调参的参数列表定为 lr: {0.5e-3, 1e-3, 2e-3}, epoch: 20. 最优参数为 lr = 2e-3, epoch = 20.

实验表明, 在这三种学习率之下, 精度都比较接近, 略高于 70%, 其中 lr = 2e-3 时的表现略好.

```
Tuning hyperparameters for pretrained model...
Training with lr=0.005, num_epochs=20
Final validation accuracy: 0.7164
New best accuracy: 0.7164 with lr=0.005 and num_epochs=20
Training with lr=0.01, num_epochs=20
Final validation accuracy: 0.7245
New best accuracy: 0.7245 with lr=0.01 and num_epochs=20
Training with lr=0.02, num_epochs=20
Final validation accuracy: 0.7370
New best accuracy: 0.7370 with lr=0.02 and num_epochs=20
Best Params: lr=0.02, num_epochs=20, Accuracy=0.7370
Best hyperparameters for pretrained model: {'lr': 0.02, 'num_epochs': 20,
```

图 1.1: 最后一次调参结果

随机初始化模型也可以做调参. 本实验中随机初始化模型采用和预训练模型一样的参数，保证比较的公平性.

## 第 5 节 实验结果

### 5.1. 数据分析

以下图片中，**橙线**都是**预训练模型**的数据曲线，**蓝线**都是**随机初始化模型**的数据曲线. 完整的数据文件在文件夹 `run` 中.

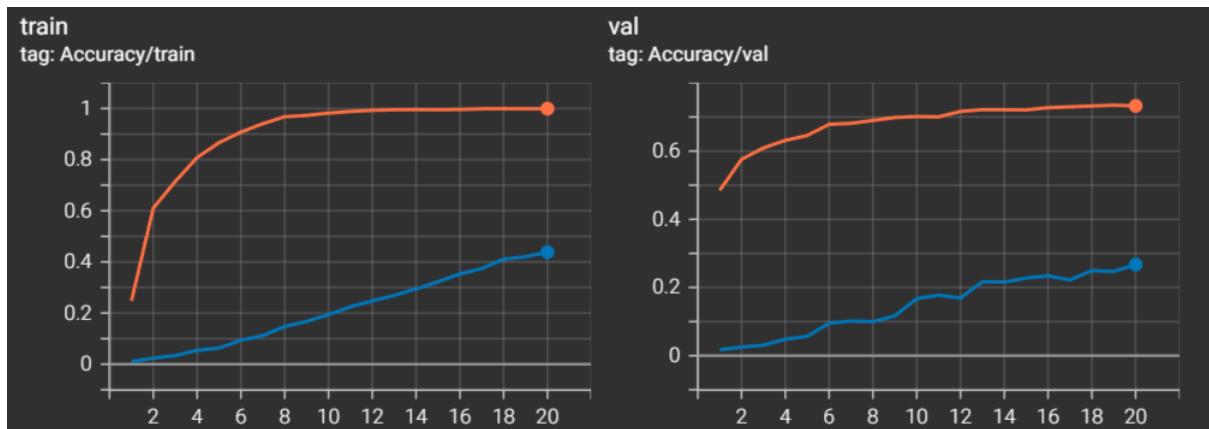


图 1.2: 准确率对比

预训练模型的训练准确率迅速上升，并且在初期就已经接近 1，这表明模型能够很好地学习训练数据. 准确率接近 1 通常表明模型对训练数据的拟合非常好. 随机初始化模型的训练准确率相比之下增长较慢，最终也没有接近 1，这可能意味着模型学习较慢，或者是模型容量不足以完全学习数据.

预训练模型的验证准确率也很快上升并保持在较高水平，这说明预训练模型在未见数据上也具有很好的泛化能力。验证准确率的高稳定性同时也表明模型没有出现显著的过拟合。随机初始化模型的验证准确率虽有所提高，但整体上显著低于预训练模型，这表明其泛化能力较弱。验证准确率的较低水平也表明该模型可能受限于其初始参数设置，未能充分利用训练数据。

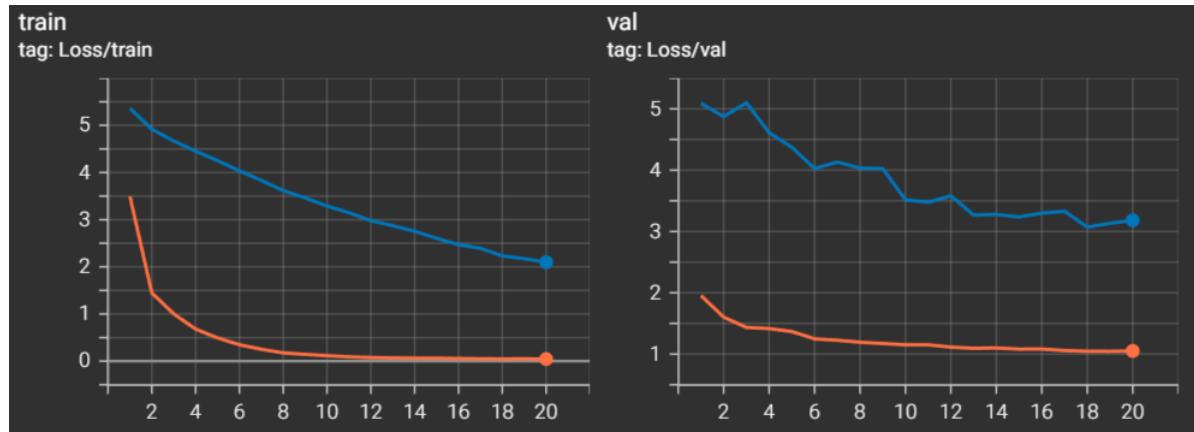


图 1.3: 损失函数对比

预训练模型的训练损失迅速下降并趋于平稳，这与高训练准确率一致，表明模型有效地减少了误差。随机初始化模型的训练损失下降较慢，结束时仍高于预训练模型的损失，这进一步证实了其学习效率较低。

预训练模型的验证损失下降并保持较低，与高的验证准确率相符，说明模型在未见数据上的表现良好。验证损失的低水平和稳定性表明没有过拟合。随机初始化模型的验证损失相比之下较高并稍有波动，可能指示模型在未见数据上的性能不够稳定，这可能是由于模型参数初始化不佳或者模型结构不够优化。

## 5.2. 结论

预训练模型明显优于随机初始化模型，无论是在训练过程还是在验证过程中。这表明利用预训练的权重可以显著提升模型的学习效率和泛化能力。

随机初始化模型可能需要更多的训练周期、更复杂的网络结构或者更多的调优来提高其性能。

基于以上分析，使用预训练模型在类似的任务中是一个有效的策略，特别是当可用的训练数据量不足以从头训练一个复杂模型时。对于随机初始化的模型，可能需要探索额外的技术，如更深的网络、正则化策略或更精细的超参数调整，以提高其性能。

## 第二章 在 VOC 数据集上训练并测试目标检测模型 Faster R-CNN 和 YOLO V3

### 第 1 节 任务描述

1. 学习使用现成的目标检测框架——如 mmdetection 或 detectron2——在 VOC 数据集上训练并测试目标检测模型 Faster R-CNN 和 YOLO V3;
2. 挑选 4 张测试集中的图像，通过可视化对比训练好的 Faster R-CNN 第一阶段产生的 proposal box 和最终的预测结果.
3. 搜集三张不在 VOC 数据集内包含有 VOC 中类别物体的图像，分别可视化并比较两个在 VOC 数据集上训练好的模型在这三张图片上的检测结果（展示 bounding box、类别标签和得分）

### 第 2 节 实验环境

#### 2.1. MMDetection 设置：Config is All You Need

MMDetection 是一个基于 PyTorch 的开源目标检测工具箱，由香港中文大学多媒体实验室（CUHK Multimedia Lab）开发和维护. 它是 OpenMMLab 项目的一部分，旨在提供一个灵活、高效、易于使用的目地检测框架，支持各种主流目地检测算法.

MMDetection 对整个目地检测模型的训练流程进行了高程度的封装，用户只需要写出正确的 config 文件而无需过度关注模型训练和测试的工程细节. MMDetection 同时提供了一键部署分布式训练、绘制损失曲线等接口，使得部署十分方便. 本次实验就使用了 mmdeet 的相关功能.

#### 2.2. 软硬件环境

该任务在 Ubuntu 22.04.1 LTS 上完成，具体如下：

```
1 -----
2 System environment:
3     sys.platform: linux
4     Python: 3.10.14 (main, May  6 2024, 19:42:50) [GCC 11.2.0]
5     CUDA available: True
```

## 第二章 在 VOC 数据集上训练并测试目标检测模型 FASTER R-CNN 和 YOLO V3 6

```
6 MUSA available: False
7 numpy_random_seed: 1586568599
8 GPU 0,1,2,3: Tesla V100-SXM2-16GB
9 CUDA_HOME: /usr/local/cuda-12.0
10 NVCC: Cuda compilation tools, release 12.0, V12.0.76
11 GCC: gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0
12 PyTorch: 2.1.2
13 PyTorch compiling details: PyTorch built with:
14 - GCC 9.3
15 - C++ Version: 201703
16 - Intel(R) oneAPI Math Kernel Library Version 2023.1-Product Build
   20230303 for Intel(R) 64 architecture applications
17 - Intel(R) MKL-DNN v3.1.1 (Git Hash 64
   f6bcbab628e96f33a62c3e975f8535a7bde4)
18 - OpenMP 201511 (a.k.a. OpenMP 4.5)
19 - LAPACK is enabled (usually provided by MKL)
20 - NNPACK is enabled
21 - CPU capability usage: AVX512
22 - CUDA Runtime 11.8
23 - NVCC architecture flags: -gencode;arch=compute_50,code=sm_50;-
   gencode;arch=compute_60,code=sm_60;-gencode;arch=compute_61,code=
   sm_61;-gencode;arch=compute_70,code=sm_70;-gencode;arch=compute_75
   ,code=sm_75;-gencode;arch=compute_80,code=sm_80;-gencode;arch=
   compute_86,code=sm_86;-gencode;arch=compute_37,code=sm_37;-gencode
   ;arch=compute_90,code=sm_90;-gencode;arch=compute_37,code=
   compute_37
24 - CuDNN 8.7
25 - Magma 2.6.1
26 - Build settings: BLAS_INFO=mkl, BUILD_TYPE=Release, CUDA_VERSION
   =11.8, CUDNN_VERSION=8.7.0, CXX_COMPILER=/opt/rh/devtoolset-9/root
   /usr/bin/c++, CXX_FLAGS= -D_GLIBCXX_USE_CXX11_ABI=0 -fabi-version
   =11 -fvisibility-inlines-hidden -DUSE_PTHREADPOOL -DNDEBUG -
   DUSE_KINETO -DLIBKINETO_NOROCTRACER -DUSE_FBGEMM -DUSE_QNNPACK -
   DUSE_PYTORCH_QNNPACK -DUSE_XNNPACK -
   DSYMBOLICATE_MOBILE_DEBUG_HANDLE -O2 -fPIC -Wall -Wextra -Werror=
   return-type -Werror=non-virtual-dtor -Werror=bool-operation -
   Wnarrowing -Wno-missing-field-initializers -Wno-type-limits -Wno-
   array-bounds -Wno-unknown-pragmas -Wno-unused-parameter -Wno-
   unused-function -Wno-unused-result -Wno-strict-overflow -Wno-
   strict-aliasing -Wno-stringop-overflow -Wno-psabi -Wno-error=
```

```

pedantic -Wno-error=old-style-cast -Wno-invalid-partial-
specialization -Wno-unused-private-field -Wno-aligned-allocation-
unavailable -Wno-missing-braces -fdiagnostics-color=always -
faligned-new -Wno-unused-but-set-variable -Wno-maybe-uninitialized
-fno-math-errno -fno-trapping-math -Werror=format -Werror=cast-
function-type -Wno-stringop-overflow, LAPACK_INFO=mkl,
PERF_WITH_AVX=1, PERF_WITH_AVX2=1, PERF_WITH_AVX512=1,
TORCH_DISABLE_GPU_ASSERTS=ON, TORCH_VERSION=2.1.2, USE_CUDA=ON,
USE_CUDNN=ON, USE_EXCEPTION_PTR=1, USE_GFLAGS=OFF, USE_GLOG=OFF,
USE_MKL=ON, USE_MKLDNN=ON, USE_MPI=OFF, USE_NCCL=ON, USE_NNPACK=ON
, USE_OPENMP=ON, USE_ROCM=OFF,
27
28 TorchVision: 0.16.2
29 OpenCV: 4.9.0
30 MMEngine: 0.10.4
31
32 Runtime environment:
33 cudnn_benchmark: False
34 mp_cfg: {'mp_start_method': 'fork', 'opencv_num_threads': 0}
35 dist_cfg: {'backend': 'nccl'}
36 seed: 1586568599
37 Distributed launcher: pytorch
38 Distributed training: True
39 GPU number: 4
40 -----

```

## 第 3 节 FasterRCNN 实验

### 3.1. 实验设置

在本次实验中，我们在 VOC2007 和 VOC2012 数据集上微调以 ResNet-50 为骨干网络的 FasterRCNN，其预训练权重可以从 `torchvision` 中下载：

```
1 resnet50 = models.resnet50(pretrained=True)
```

同时，由于 FasterRCNN 的模型复杂度较高，我们在训练过程中仅对输入图像进行随机翻转这一项噪声加入，最后得到的效果较好。

本次实验在 ROI 头和 RPN 头上都使用了相同的损失函数，其中 bbox 部分使用的是  $L^1$ -损失函数，分类部分则使用了交叉熵作为损失函数。

训练设置 `max_epoch = 10`, 但是由于观察到模型在测试集上的表现不再随时间增长, 我们在第 7 个轮次结束后便采取了早停措施. `batch_size = 8/(per GPU)`, 因此总共进行了 10864 次迭代.

### 3.2. 实验结果

我们将所有测试中最好的测试结果置于下表, 其余的结果可以见于下面的 mAP 曲线和 loss 曲线:

1	-----+-----+-----+-----+-----+
2	class   gts   dets   recall   ap
3	-----+-----+-----+-----+-----+
4	aeroplane   285   877   0.800   0.721
5	bicycle   337   1125   0.938   0.834
6	bird   459   1159   0.874   0.784
7	boat   263   1482   0.795   0.602
8	bottle   469   1340   0.642   0.527
9	bus   213   938   0.930   0.833
10	car   1201   3230   0.883   0.796
11	cat   358   1147   0.969   0.884
12	chair   756   4585   0.860   0.632
13	cow   244   921   0.934   0.830
14	diningtable   206   2173   0.942   0.690
15	dog   489   1527   0.969   0.862
16	horse   348   1083   0.937   0.853
17	motorbike   325   1347   0.932   0.830
18	person   4528   12702   0.878   0.776
19	pottedplant   480   2250   0.771   0.525
20	sheep   242   733   0.876   0.750
21	sofa   239   1419   0.954   0.770
22	train   282   1295   0.950   0.823
23	tvmonitor   308   1169   0.873   0.750
24	-----+-----+-----+-----+-----+
25	mAP         0.754
26	-----+-----+-----+-----+-----+

下图是我们的损失曲线和 mAP 曲线:

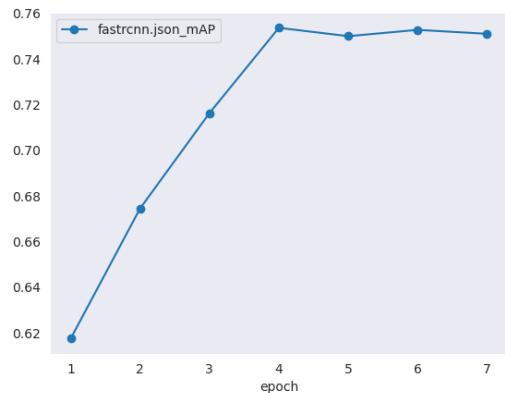


图 2.1: FasterRCNN mAP 曲线

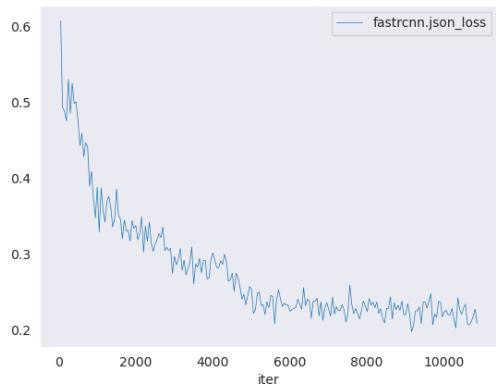


图 2.2: FasterRCNN 损失函数曲线

### 3.3. 可视化检测

我们选取了四张图像，对比了其在 RPN Head 中产生的 proposal box 和最终的预测结果，见下图：



图 2.3: 对象 1: Proposal Box



图 2.4: 对象 1: 预测结果

可以看到，通过将 RPN Head 的参数 `num_classes` 设置为 1，Proposal Boxes 较好地完成了区分前后景的任务，最终的 FasterRCNN 预测结果也较为准确。

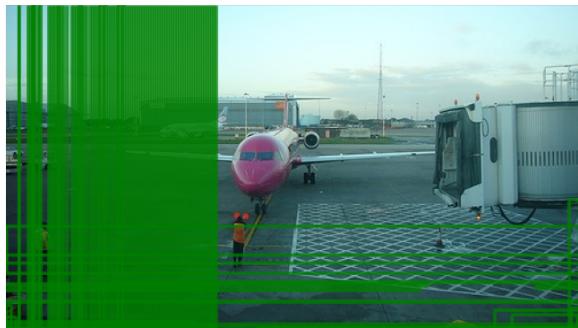


图 2.5: 对象 2: Proposal Box

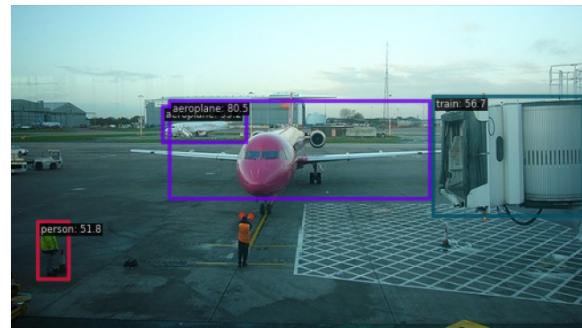


图 2.6: 对象 2: 预测结果



图 2.7: 对象 3: Proposal Box

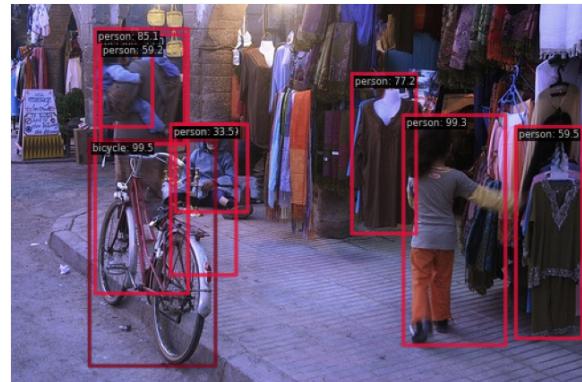


图 2.8: 对象 3: 预测结果

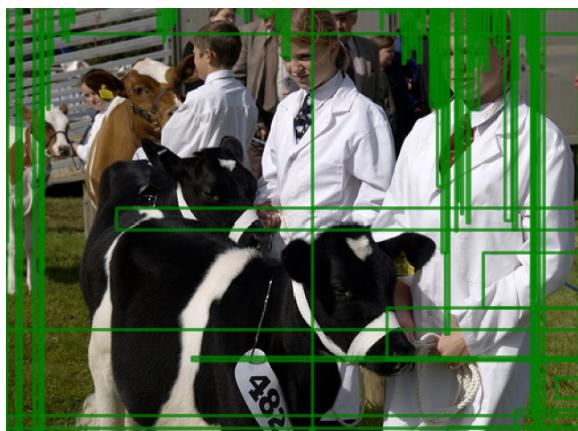


图 2.9: 对象 4: Proposal Box

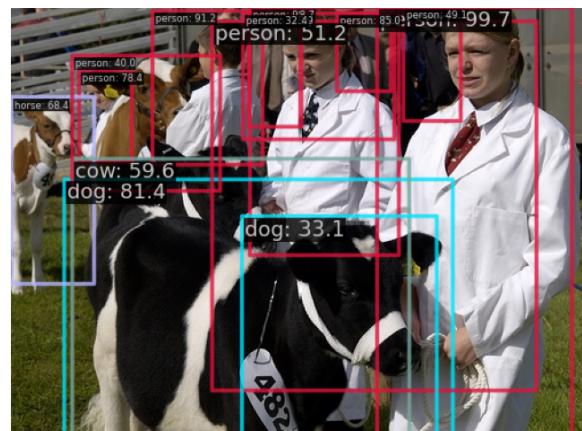


图 2.10: 对象 4: 预测结果

### 3.4. YOLOv3 实验

### 3.5. 实验设置

在本次实验中，我们在 VOC2007 和 VOC2012 数据集上微调以 DarkNet-53 为骨干网络的 YOLOv3，其预训练权重可以从 `torchvision` 中下载。

我们的实验重点包括以下两点：

1. 数据预处理。
2. 在实验过程中添加噪声。

在数据预处理部分，我们将每张图像的 RGB 值除以 255，再将图像尺寸填充为能被 32 整除的大小，具体如下：

```
1 data_preprocessor = dict(  
2     bgr_to_rgb=True,  
3     mean=[  
4         0,  
5         0,  
6         0,  
7     ],  
8     pad_size_divisor=32,  
9     std=[  
10        255.0,  
11        255.0,  
12        255.0,  
13    ],  
14    type='DetDataPreprocessor')
```

实验中添加的噪声则是为了模型的鲁棒性考虑，为了弥补 YOLOv3 的模型差距，在实验中我们采用了更激进的方法向图像增加噪声来进行数据增广，我们使用的方法包括：

1. 拓展图像大小
2. 随机调整图像尺寸
3. 对图像进行随机裁剪
4. 对图像进行光度扭曲
5. 随机翻转图像

同时，在训练中我们还使用了早停机制，当观察到模型在测试集上的 mAP 在几次测试中趋向稳定时，我们提前停止训练，避免模型的过拟合。

和 FasterRCNN 相同，我们使用了  $L^1$ -损失和交叉熵来作为损失函数。

### 3.6. 实验结果

下图是 YOLOv3 训练的 mAP 曲线和 loss 曲线：

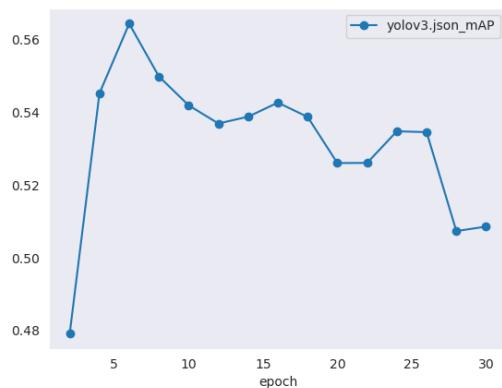


图 2.11: YOLOv3 mAP 曲线

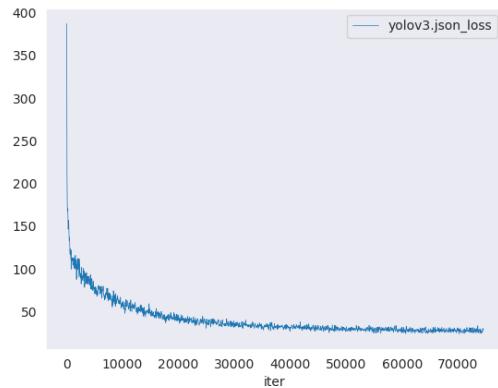


图 2.12: YOLOv3 损失函数曲线

YOLOv3 是作者在半夜训练的，所以没有进行早停，实际上通过 mAP 可以发现，训练到第 30 个 epoch 时模型已经严重过拟合。

## 第 4 节 实验对比

最后，我们使用不在训练测试集中的三张图片对比 YOLOv3 和 FasterRCNN 的性能。



图 2.13: 对象 1: FasterRCNN 表现



图 2.14: 对象 1: YOLOv3 表现



图 2.15: 对象 2: FasterRCNN 表现

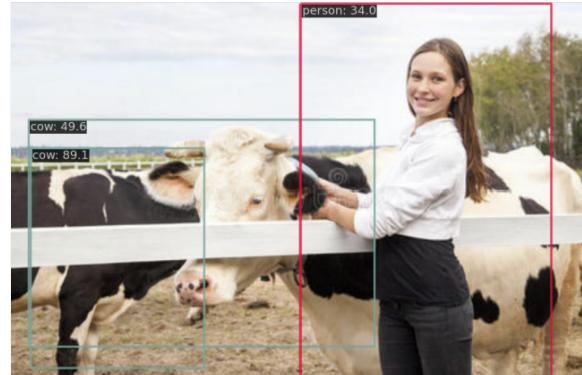


图 2.16: 对象 2: YOLOv3 表现



图 2.17: 对象 3: FasterRCNN 表现



图 2.18: 对象 3: YOLOv3 表现

## 第二章 在 VOC 数据集上训练并测试目标检测模型 FASTER R-CNN 和 YOLO V314

可以看到，在识别表现上 YOLOv3 显著落后于 FasterRCNN，在第 3 张实验图像上，YOLOv3 甚至没有识别出火车。