Human Eye Project Final Report

郭丹琪 张晨阳 信息学院

2020 年 12 月 18 日

1 介绍

基本项目任务：收集尽可能多的视觉数据，包括图像或视频，构建识别模型，从视觉内容中生成文本信息。

我们实现的扩展任务：在能识别出图片上物体类别的基础上，在类别内部进行更细致的识别和分辨。提供一个植物图像识别应用，命名为 PLANET(Plant Expert)，根据植物的图片，识别植物种类，帮助人们快速便捷地辨别植物。

接下来，我们将分三部分介绍我们的工作。第二部分介绍项目背景与相关工作。第三部分介绍训练模型的过程。第四部分介绍对训练好模型的应用。附录给出了训练模型的具体数据。

2 背景与相关工作学习

2.1 图像识别模型

从图像中提取信息首先要做的是目标检测，即从图像中检测并定位到待识别的物体。我们对当前已有的目标检测方法进行了调研。传统目标检测系统采用 Deformable Parts Models (DPM)，通过滑动框方法提出目标区域，然后采用分类器来实现识别。此外，还有 R-CNN 系列算法，例如 RCNN、Fast RCNN 和 Faster RCNN，这些算法采用的是 Region Proposal Methods。首先生成潜在的 bounding boxes，然后采用分类器识别这些 bounding boxes 区域。最后通过 post-processing 去除重复 bounding boxes 来进行优化。这类方法流程复杂，存在速度慢和训练困难的问题。

YOLO [1] 是图像识别的经典模型，只需一眼（you only look once，YOLO）即可检测目标类别和位置，预测流程简单，速度快。YOLO 采用单个卷积神经网络来预测多个 bounding boxes 和类别概率，与滑动窗口方法和 Region Proposal Methods 不同，YOLO 在训练和预测过程中可以利用全图信息，正确率较高。

2.2 数据集

我们通过一篇计算机视觉领域训练集的 Survey 论文 [2] 对数据集进行了调研，从中挑选了 Leafsnap 数据集 [3]，并对其进行了学习。Leafsnap 数据集包含 185 种树木种类，23147 张高质量实验室叶子照片和 7719 张普通照片。其中实验室照片用比色尺和刻度尺标量了叶子的颜色和大小。Leafsnap 数据集的构建团队提出了一个用于植物种类自动识别的视觉识别系统，目前这个系统已经做成了 APP，不进行开源。

2.3 项目提出与分析

我们提出了首个开源的植物图像识别工具，帮助人快速辨别植物。以下是我们项目开发的两个动机。

- 当人们见到有趣的植物，想要知道它的品种时，只能通过查阅植物百科的方式。这种方式对于不随身携带植物百科书的人来说，十分不方便，且需要人为翻书搜索，耗时很长。现在没有开源图像识别框架能准确识别植物。因此我们希望开发一个开源植物图片识别框架，来填补这个缺口，帮助人们快速、准确地识别并分辨植物。
- 现有的图像识别框架大部分是很一般、概括性的识别，例如 YOLO 可以识别出人和狗，但不能分辨狗的种类是藏獒还是吉娃娃。我们希望进一步探究这些图像识别框架的能力。在本项目中，我们探究这些框架和神经网络在某个特定小领域上，给近似图像分类的准确度。更具体的，我们探究现有框架和神经网络在植物领域，分辨植物类别的准确度。这样的探究帮助我们更好地了解现有图像识别框架和神经网络的能力和限制，为后续科研工作者的研究与模型改进提供思路。

3 模型训练

3.1 数据预处理

在这一部分，我们阐述对 Leafsnap 数据集的预处理过程。首先，我们将 Leafsnap 数据集中的所有图片，按照 4：1 的比例，随机分成了训练集和测试集。然后对训练集进行了数据清理工作，具体方法在下面展示。

观察 Leafsnap 数据集中的原始叶子图片见图 1，发现图片有很大的边框，通过尺子和比色尺来展示叶子的大小。这些边框部分对训练模型没有帮助，因此我们对原始叶子图片进行了裁剪，剪去了不必要的边框部分。首先调用 OpenCV 的函数 cv2.findContours( )，找到叶子的轮廓。在这个过程中，遇到的难点是 cv2.threshold 的选取，过大或过小的 cv2.threshold 会导致识别到错误的轮廓。然后调用 scipy.misc.imresize( ) 重定义图片大小为 224×224，统一训练数据。图 1展示了裁剪前后的图片。

为了方便后续训练，我们调整所有叶子图片大小为 224×224 像素。图片大小可以通过更改 *utils.py* 中 *img = misc.imresize(img, (width,length))* 中的参数 *width* 和 *length*，

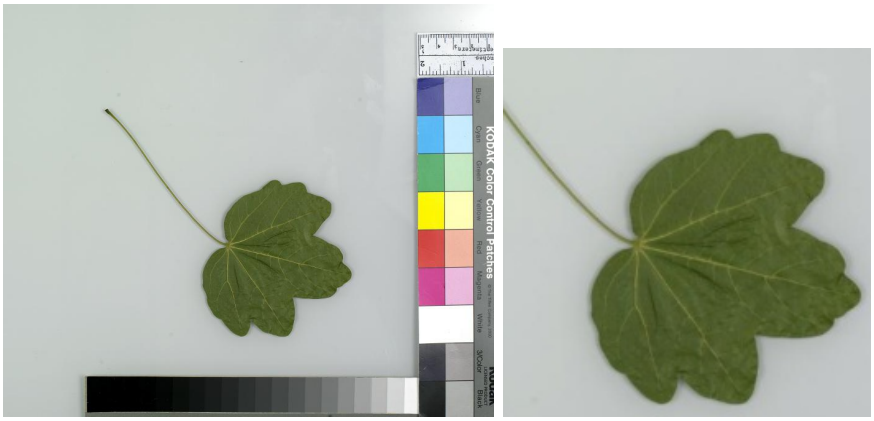

图 1: 裁剪前后的叶子图片:after

生成 $width \times length$ 的图片数据。

这样的数据清理过程

- 统一训练图片的尺寸，为后续训练作准备。
- 消除了数据集中的冗余信息,减小了数据集占用的存储空间。数据集从原始的 890MB 缩小到 580MB。
- 突出了数据图片的重点，从而提高模型的训练效率与准确率。

3.2 模型训练

训练平台是 8 核 16 线程的 Intel Core i9-9900K CPU 和有 10GB Memory 的 NVIDIA GeForce RTX 2080Ti。

参考博客 [4] 学习了 pytorch 在 GPU 上训练神经网络的方法后，我们训练了 ResNet-101 和 VGG-16。为了使模型训练更快，我们通过以下 API，在 python 代码中调用了 CUDA，从而利用高性能 GPU 训练模型。在使用 pytorch 训练的过程中，受限于 GPU Memory，ResNet-101 和 VGG-16 的 batch size 最大分别能取到 64 和 32。

```
model = torch.nn.DataParallel(model).cuda()
criterion = criterion.cuda()
input = input.cuda(async=True)
target = target.cuda(async=True)
```

3.3 训练优化

我们对模型进行了如下优化:

- 动态调整 learning rate: 随着训练 epoch 增加，learning rate 递减，但始终大于 0.0001。在训练初期使收敛速度更快，训练后期收敛慢、保证训练出更高的准确率。

- 选取 batch size：在实验中发现，训练 epoch 相同时，小的 batch size 的准确率更低。并且，随着 epoch 的增加，batch size 小的模型准确率的提升较慢。因此选取了训练平台能满足的最大 batch size。
- 尝试迁移学习：冻结已经训练好的模型的某些层的参数，进行 fine-tuning。具体的，冻结 ResNet 卷积层的参数，只训练全连接层。

3.4 训练中的问题与解决

CUDA out of memory 训练和检验准确率的过程中，程序经常会报出 *CUDA out of memory* 的错误。造成这个错误的主要原因是

- CUDA Memory 中存储了无用的缓存信息，导致 Memory 被占满。
- 数据集较大，且 batch size 较大。
- GPU Memory 较小，无法训练复杂模型。

针对以上原因，在代码中做出相应调整

- 调用 *torch.cuda.empty_cache()* 及时清理无用缓存。调用 *with torch.no_grad():*，不跟踪计算梯度，减小 Memory 占用。
- 在上一方法仍不能解决问题的情况下，减小 batch size。
- 增加 GPU Memory，或者调整模型。在我们的训练过程中，没有用到这种方法。

Size mismatch 具体报错信息为 *RuntimeError: size mismatch m1: [32 × 50176], m2: [1024 × 185] at /pytorch/aten/src/THC/generic/THCTensorMathBlas.cu:290*。

参考 [5] 知道，如果模型中有 nn.Linear 层，需要在训练前确定该层的 *in_features* 和 *out_features*，因为 pytorch 无法在训练中动态选取正确的值。调用 *model.linear = nn.Linear(50176, 80)* 实现。

3.5 效果展示

我们训练了 ResNet-101 和 VGG-16，在 10 个 epoch 内，ResNet-101 的准确率和速度都显著高于 VGG-16，因此不再继续训练 VGG-16。

尝试迁移学习时，冻结 ResNet-101 的卷积层，只训练全连接层，训练速度显著提升。训练 30 个 epoch 后 top1 和 top5 的准确率只有 88.26%, 95.52%，不如完全训练的 ResNet-101。

因此我们训练的最佳模型是 26 个 epoch 后的完全训练 ResNet-101 图 2展示这个最佳模型的训练效果，具体的训练数据在 APPENDIX 中给出。横轴表示训练的 epoch。纵轴表示模型的准确率 (precision)。Top 1 precision 是预测概率第一的预测结果的准确率，

Top 5 precision 是预测概率前五的预测结果的准确率。模型的预测准确率最高达到 Top 1 precision 为 93.203%，Top 5 precision 为 99.407%。这样的训练效果已经足够好，因此我们停止了训练。

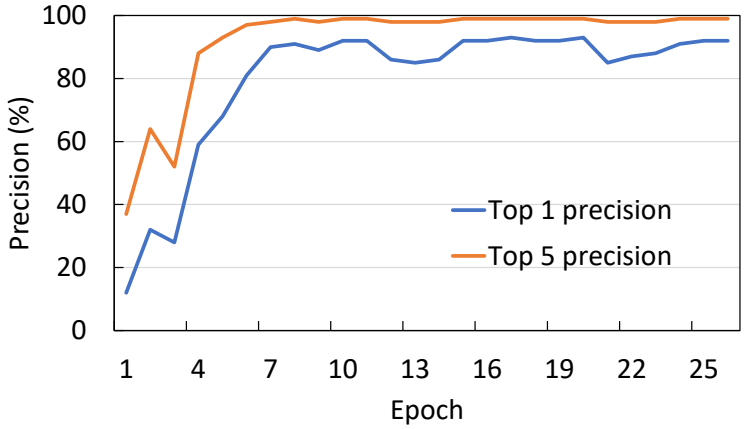

图 2: ResNet-101 训练效果

4 应用

4.1 Inference

由于植物识别的实际应用是在个人电脑的 CPU 上，而模型是在服务器 GPU 上训练的，所以需要将训练好的模型保存下来，并在电脑 CPU 上进行加载和应用。

4.1.1 处理步骤

首先将训练好的模型读入，加载只需要使用 torch 中的 load 操作。但加载后的模型还不能直接使用，需要先对其进行 eval 操作。这是因为在应用中每次只会传入一张图片，对于模型来说 batchsize 过小，在处理过程中容易导致参数改变，且使得图像失真。所以需要 eval 操作来将模型转换为测试模型，这样才不会改变模型中已经训练好的参数值。

加载完模型后，在实际应用阶段需要对传入的植物叶子图片进行预处理，操作与训练集图片的预处理类似。先将图片转换为 torch.FloatTensor 的数据形式，然后对该 tensor 数据进行正则化处理，就可以得到模型可以处理的图片数据。

处理完后就可以将图片数据传入模型。模型对图片中的植物种类进行识别，返回其属于数据集中各植物类别的概率，从中选取最大的 5 个值对应的类别作为识别结果。

4.1.2 遇到的问题及解决

首先是在 CPU 上运行 GPU 上训练好的模型的问题。在运行的时候会报错 *RuntimeError: module must have its parameters and buffers on device cuda:0 (device ids[0])*

but found one of them on device:cpu。如果从字面意思理解该错误信息，就会误以为是把模型加载到了 GPU 上，然后把部分数据加载到了 CPU 上。但实际问题在于训练模型的过程中使用了多个 GPU 加速训练，用到 torch.nn.DataParallel(model).cuda()。但是在 CPU 环境中不能直接使用导入的 GPU 训练的 DataParallel 模型，需要额外用 model.module 语句得到其中实际的模型，在 CPU 环境中才可以使用。

其次在得到了识别结果后，发现得到的结果只是植物类别在模型中对应的序号，无法确定类别实际是什么。所以只能用 torchvision.datasets.ImageFolder 重新进行模型训练数据的读入和预处理操作，再通过 class-to-idx 得到模型对各植物类别的编号，将编号和类别的对应关系存入文件，以便后续的使用。

4.2 动态网页应用

为了让模型得到更好的应用，实现了植物种类识别的动态网页。在前端把用户在网页上上传的图片传到后端，后端在得到图片后将图片传入 inference 程序，进行预处理和类别识别，最后再把得到的该图片中的植物最可能属于的 5 个类别返回给前端输出。

主要用了两个界面，初始界面用来上传图片，如图 3，结果页面用来显示识别出的植物类别，如图 4。

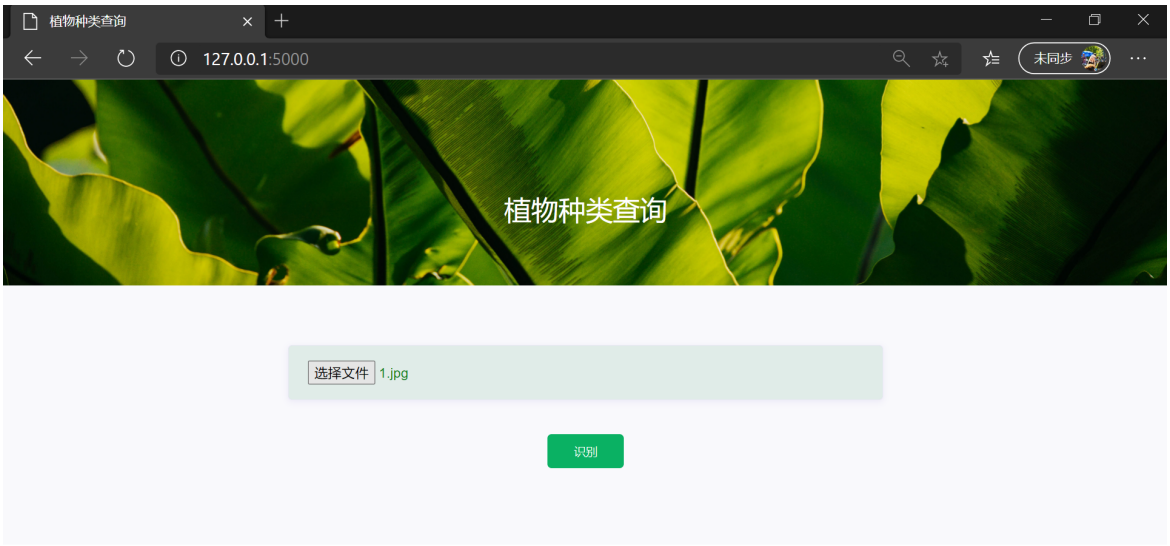

图 3: 图片上传界面

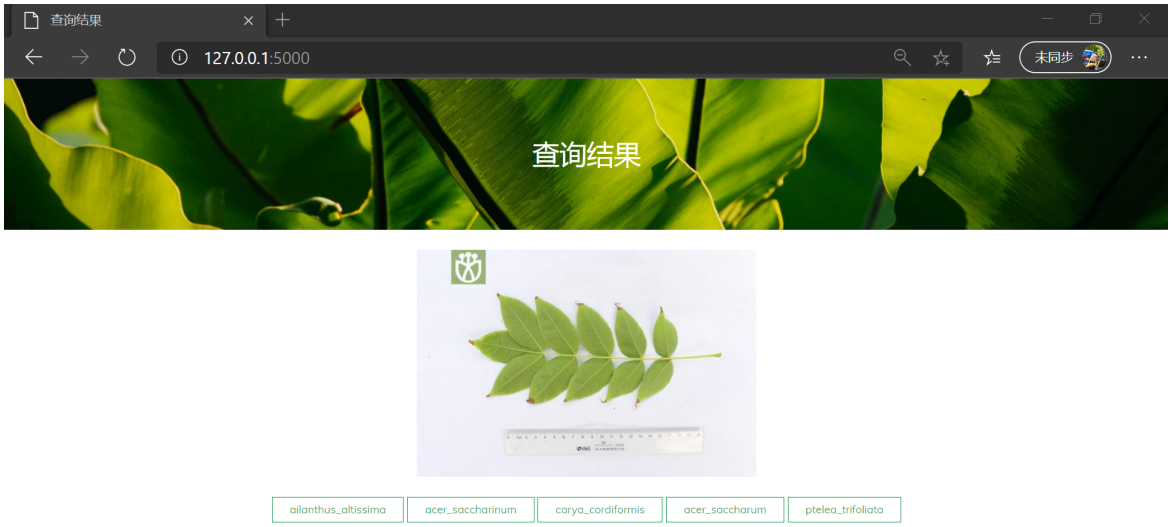

图 4: 结果显示界面

5 项目总结

5.1 项目难点

- 数据预处理时，要识别叶子的轮廓并剪切，threshold 的选取需要反复试验和调整
- 本门课程是我们所学的第一门深度学习课程，课上所学的深度神经网络模型较复杂，实现较难。初次尝试用 pytorch 在 GPU 上训练复杂的神经网络模型，且数据集较大，期间遇到了很多问题，在结合课上所学内容和课外自学内容后逐一解决
- 训练神经网络模型时，进行了多次实验：尝试不同模型（ResNet101，VGG16），试验 batch size 对准确率的影响并调整到合适的 batch size，尝试迁移学习。
- 用训练好的神经网络进行推理时，要把从 GPU 上训练并保存的模型放在 CPU 上运行，需要转换代码。

5.2 收获与感受

- 学习了计算机视觉领域的热门算法和模型，对计算机视觉领域有了更深入的了解。
- 在完成项目的过程中，学习并实践了完整实现一个项目的流程：调研领域背景与相关工作，进行项目可行性分析，搜集所需要的资料、代码、数据集，落实到工程实现，工程调优，效果展示，最终制作用户界面将项目落实到应用。

- 实现一个 end-to-end 的，可以实际应用的程序，包括前期复杂的神经网络训练和后期的推理、动态网页用户界面，最终实现用户交互，项目可以供用户使用，很有成就感。

APPENDIX

附录中给出了 ResNet-101 训练过程中最后两个 epoch 的训练数据。

```
[Learning Rate] 0.000100
Epoch: [25][0/1535] \Time 0.564 (0.564) Data 0.159 (0.159) Loss 0.0093
   (0.0093) Prec@1 100.000 (100.000) Prec@5 100.000 (100.000)
Epoch: [25][100/1535] \Time 0.394 (0.396) Data 0.008 (0.010) Loss 0.0139
   (0.0511) Prec@1 100.000 (98.407) Prec@5 100.000 (99.985)
Epoch: [25][200/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0302
   (0.0515) Prec@1 100.000 (98.414) Prec@5 100.000 (99.984)
Epoch: [25][300/1535] \Time 0.395 (0.395) Data 0.008 (0.009) Loss 0.0405
   (0.0507) Prec@1 96.875 (98.458) Prec@5 100.000 (99.984)
Epoch: [25][400/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0422
   (0.0519) Prec@1 98.438 (98.395) Prec@5 100.000 (99.984)
Epoch: [25][500/1535] \Time 0.395 (0.395) Data 0.008 (0.009) Loss 0.0899
   (0.0520) Prec@1 96.875 (98.422) Prec@5 100.000 (99.984)
Epoch: [25][600/1535] \Time 0.395 (0.395) Data 0.008 (0.009) Loss 0.0573
   (0.0524) Prec@1 98.438 (98.412) Prec@5 100.000 (99.982)
Epoch: [25][700/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0397
   (0.0528) Prec@1 98.438 (98.400) Prec@5 100.000 (99.984)
Epoch: [25][800/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0299
   (0.0527) Prec@1 100.000 (98.404) Prec@5 100.000 (99.986)
Epoch: [25][900/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0220
   (0.0524) Prec@1 100.000 (98.417) Prec@5 100.000 (99.988)
Epoch: [25][1000/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0305
   (0.0524) Prec@1 100.000 (98.425) Prec@5 100.000 (99.989)
Epoch: [25][1100/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0441
   (0.0525) Prec@1 96.875 (98.442) Prec@5 100.000 (99.989)
Epoch: [25][1200/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0338
   (0.0526) Prec@1 98.438 (98.447) Prec@5 100.000 (99.988)
Epoch: [25][1300/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0405
   (0.0531) Prec@1 100.000 (98.430) Prec@5 100.000 (99.987)
Epoch: [25][1400/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0356
```

```
   (0.0533) Prec@1 100.000 (98.422) Prec@5 100.000 (99.988)
Epoch: [25][1500/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0439
   (0.0536) Prec@1 98.438 (98.401) Prec@5 100.000 (99.989)
Test: [0/93] Time 0.242 (0.242) Loss 0.0799 (0.0799) Prec@1 98.438
   (98.438) Prec@5 100.000 (100.000)
Test: [10/93] Time 0.136 (0.146) Loss 0.3807 (0.1230) Prec@1 85.938
   (95.739) Prec@5 96.875 (99.574)
Test: [20/93] Time 0.136 (0.141) Loss 0.1952 (0.1302) Prec@1 95.312
   (95.610) Prec@5 100.000 (99.628)
Test: [30/93] Time 0.136 (0.140) Loss 0.1926 (0.1523) Prec@1 93.750
   (94.859) Prec@5 100.000 (99.597)
Test: [40/93] Time 0.136 (0.139) Loss 0.0159 (0.1550) Prec@1 100.000
   (94.931) Prec@5 100.000 (99.581)
Test: [50/93] Time 0.136 (0.138) Loss 0.3943 (0.1814) Prec@1 92.188
   (94.271) Prec@5 98.438 (99.510)
Test: [60/93] Time 0.136 (0.138) Loss 0.7876 (0.2528) Prec@1 70.312
   (91.931) Prec@5 98.438 (99.360)
Test: [70/93] Time 0.136 (0.138) Loss 0.0380 (0.2537) Prec@1 98.438
   (91.769) Prec@5 100.000 (99.362)
Test: [80/93] Time 0.136 (0.137) Loss 0.2839 (0.2507) Prec@1 89.062
   (91.917) Prec@5 100.000 (99.344)
Test: [90/93] Time 0.135 (0.137) Loss 0.3474 (0.2503) Prec@1 93.750
   (92.136) Prec@5 96.875 (99.296)
 * Prec@1 92.220 Prec@5 99.305

[INFO] Saved Model to leafsnap_model.pth

[Learning Rate] 0.000100
Epoch: [26][0/1535] \Time 0.535 (0.535) Data 0.130 (0.130) Loss 0.0397
   (0.0397) Prec@1 100.000 (100.000) Prec@5 100.000 (100.000)
Epoch: [26][100/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0550
   (0.0551) Prec@1 96.875 (98.407) Prec@5 100.000 (100.000)
Epoch: [26][200/1535] \Time 0.395 (0.395) Data 0.008 (0.009) Loss 0.0422
   (0.0526) Prec@1 100.000 (98.562) Prec@5 100.000 (100.000)
Epoch: [26][300/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0475
   (0.0529) Prec@1 98.438 (98.515) Prec@5 100.000 (100.000)
Epoch: [26][400/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0745
```

```
   (0.0532) Prec@1 98.438 (98.484) Prec@5 100.000 (99.996)
Epoch: [26][500/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0304
   (0.0537) Prec@1 100.000 (98.447) Prec@5 100.000 (99.997)
Epoch: [26][600/1535] \Time 0.394 (0.395) Data 0.008 (0.009) Loss 0.0244
   (0.0533) Prec@1 100.000 (98.443) Prec@5 100.000 (99.997)
Epoch: [26][700/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0697
   (0.0531) Prec@1 96.875 (98.455) Prec@5 100.000 (99.996)
Epoch: [26][800/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0525
   (0.0533) Prec@1 96.875 (98.436) Prec@5 100.000 (99.996)
Epoch: [26][900/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0693
   (0.0532) Prec@1 98.438 (98.431) Prec@5 100.000 (99.997)
Epoch: [26][1000/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0734
   (0.0533) Prec@1 95.312 (98.439) Prec@5 100.000 (99.997)
Epoch: [26][1100/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0596
   (0.0535) Prec@1 96.875 (98.436) Prec@5 100.000 (99.997)
Epoch: [26][1200/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0892
   (0.0538) Prec@1 98.438 (98.435) Prec@5 98.438 (99.993)
Epoch: [26][1300/1535] \Time 0.395 (0.395) Data 0.008 (0.009) Loss 0.0351
   (0.0539) Prec@1 98.438 (98.433) Prec@5 100.000 (99.994)
Epoch: [26][1400/1535] \Time 0.399 (0.395) Data 0.011 (0.009) Loss 0.0916
   (0.0539) Prec@1 98.438 (98.431) Prec@5 100.000 (99.993)
Epoch: [26][1500/1535] \Time 0.395 (0.395) Data 0.009 (0.009) Loss 0.0557
   (0.0537) Prec@1 98.438 (98.439) Prec@5 100.000 (99.994)
Test: [0/93] Time 0.243 (0.243) Loss 0.0816 (0.0816) Prec@1 98.438
   (98.438) Prec@5 100.000 (100.000)
Test: [10/93] Time 0.136 (0.146) Loss 0.4409 (0.1380) Prec@1 87.500
   (95.739) Prec@5 96.875 (99.574)
Test: [20/93] Time 0.136 (0.141) Loss 0.1991 (0.1512) Prec@1 93.750
   (95.461) Prec@5 100.000 (99.628)
Test: [30/93] Time 0.136 (0.140) Loss 0.2239 (0.1779) Prec@1 92.188
   (94.506) Prec@5 100.000 (99.546)
Test: [40/93] Time 0.136 (0.139) Loss 0.0161 (0.1756) Prec@1 100.000
   (94.817) Prec@5 100.000 (99.543)
Test: [50/93] Time 0.137 (0.138) Loss 0.4916 (0.2034) Prec@1 87.500
   (93.964) Prec@5 96.875 (99.479)
Test: [60/93] Time 0.136 (0.138) Loss 0.8256 (0.2813) Prec@1 68.750
   (91.522) Prec@5 98.438 (99.308)
```

```
Test: [70/93] Time 0.136 (0.138) Loss 0.0351 (0.2806) Prec@1 98.438
   (91.219) Prec@5 100.000 (99.318)
Test: [80/93] Time 0.136 (0.138) Loss 0.3174 (0.2769) Prec@1 89.062
   (91.397) Prec@5 100.000 (99.306)
Test: [90/93] Time 0.135 (0.137) Loss 0.4948 (0.2800) Prec@1 87.500
   (91.518) Prec@5 96.875 (99.245)
 * Prec@1 91.627 Prec@5 99.254

[INFO] Saved Model to leafsnap_model.pth
```

Reference

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[2] Y. Lu and S. Young, "A survey of public datasets for computer vision tasks in precision agriculture," *Computers and Electronics in Agriculture*, vol. 178, p. 105760, 2020.

[3] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. Soares, "Leafsnap: A computer vision system for automatic plant species identification," in *European conference on computer vision*. Springer, 2012, pp. 502–516.

[4] "Pytorch 更新模型参数以及并固定该部分参数," https://blog.csdn.net/qq_39852676/article/details/106327513?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromBaidu-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromBaidu-1.control, 2020.

[5] "Stackoverflow: Pytorch runtimeerror: size mismatch," https://stackoverflow.com/questions/53500838/pytorch-runtimeerror-size-mismatch-m1-1-x-7744-m2-400-x-120, 2020.