Human Eye Project Report - Stage Two

郭丹琪 张晨阳 信息学院

2020 年 11 月 28 日

1 介绍

项目任务：提供一个植物图像识别应用 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第二阶段的项目提案与进展。我们在第二部分介绍对已有数据集的调研和学习，在第三部分介绍数据集预处理过程，在第四部分详细阐述模型的训练过程与训练效果。最后，我们在第五部分提出下一阶段的工作计划。附录中给出了 ResNet-101 训练过程中，最后两个 epoch 的训练数据。

2 数据集调研与学习

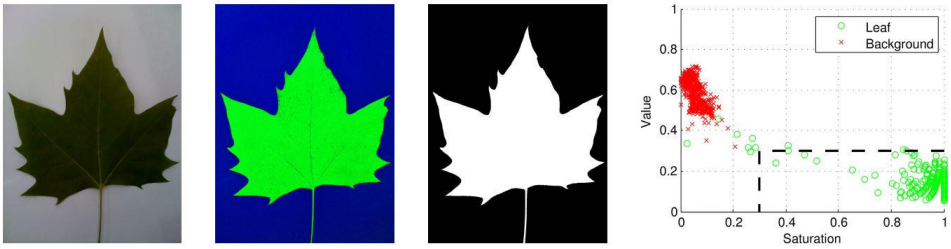
我们通过一篇计算机视觉领域训练集的 Survey 论文 [1] 对数据集进行了调研，从中挑选了 Leafsnap 数据集 [2]，并对其进行了学习。

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。我们对这个识别系统的方法论进行了学习。识别的过程包括 Classifying、Segmenting、Extracting 和 Comparing 这四个阶段。Classifying 和 Segmenting 相当于数据预处理阶段，Extracting 属于特征提取阶段，Comparing 属于最后的识别。

2.1 Classifying

首先需要判断输入图像是否为有效的叶子图片，以此决定是否进行后续处理。在所有输入图像上运行一个二元的叶/非叶分类器，该分类器利用图像上计算的 gist 特征，以 RBF 核作为分类函数，将其输入支持向量机，使用了支持向量机的 libsvm 实现和 gist 的 LEAR 实现。



(a) (b) (c) (d)

图 1: 基于颜色的分离方法

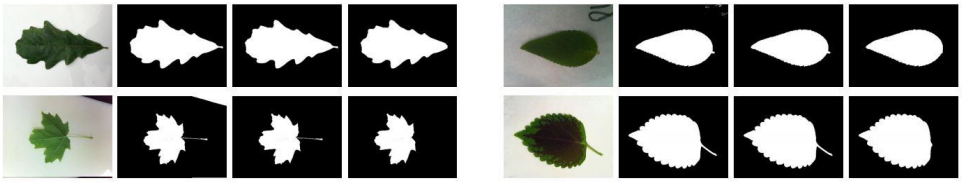


图 2: 分割处理过程

2.2 Segmenting

该识别系统将叶片的形状作为唯一识别的特征，这是因为叶片的形状在类别间区别较大，在类别内基本相似，而其他例如颜色、纹理脉络这样的特征哪怕是在类别内区别都可能很大，无法作为主要的判断依据。因此在该阶段需要移除不必要的特征影响，突出叶片的形状特征，从原始输入图片中分离出叶子的黑白二维图片。使用的是基于颜色的分离方法，通过估计 HSV 色彩空间的饱和度-值空间中的前景和背景颜色分布分割图像，并利用它们对每个像素进行独立分类。基于颜色的像素化方法比基于边缘、基于区域的方法好在可以处理类似松针这样的具有复杂分割边界的叶片。分割中首先使用 Expectation-Maximization 算法进行初始分割，在饱和度-值空间中进行像素聚类，将像素 x 的概率分布建模为两个高斯分布的和。

这个方法在图 1中展示，其中（a）为原始 RGB 图片，（b）为转化到饱和度-值空间的图片，（c）为使用了 EM 算法初始分割后的图片，（d）为图片中的背景和叶片像素在饱和度-值空间中的分布，空间的右下方轮廓部分倾向于包含叶像素。EM 期间，落在这个区域内的像素被加权，使它们的和与该区域外的像素的和匹配。

在 EM 步骤完成后，根据每个像素隶属于两个高斯分布中的哪一个将其分配给叶或背景。同时去掉初始分割后的假阳性区域和叶片的茎。这个步骤在图 2中展示。

2.3 Extracting

利用多尺度的生态测量方法可以有效地表示叶片的形状。曲率是形状的一个基本属性，该阶段从叶子的黑白二维图片中提取曲率特征，计算每个尺度上曲率值的直方图，并将这些直方图连接在一起形成尺度上曲率 (HoCS) 特征的直方图，以此表示叶片形状的特征。图 3展示了特征提取中各步的结果。

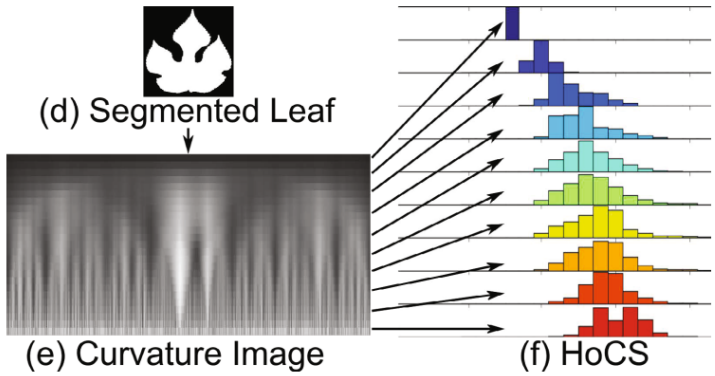


图 3: 特征提取中各步结果

2.4 Comparing

将提取的 HoCS 特征与已标记的叶片图像数据库中的特征进行比较，返回与之最接近的物种。使用的是最近邻方法，将直方图交集作为距离度量。

3 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。

3.1 数据清理

观察 Leafsnap 数据集中的原始叶子图片，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。图 4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*，生成 *width*×*length* 的图片数据。

这样的数据清理过程

- 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 划分训练集与预测集

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。

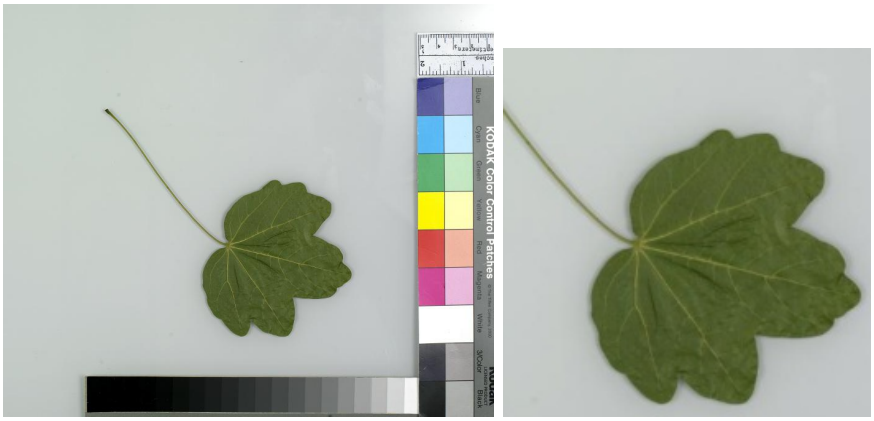


图 4: 裁剪前后的叶子图片

4 模型构建

4.1 模型训练

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

我们基于 pytorch 框架，训练了 ResNet-101 和 DenseNet-121。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。

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

在使用 pytorch 训练的过程中，受限于 CUDA Memory 的大小，这两个模型的 batch size 最大能取到 64 和 32。观察 ResNet-101 在不同 batch size 下的训练情况，发现 batch size 越小，随着 epoch 的增加，准确率的提升较慢。因此训练模型时，我们分别选取了可取到的最大 batch size 64 和 32。

4.2 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 [3] 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。

4.3 效果展示

图 5展示 ResNet-101 的训练效果。横轴表示训练的 epoch，一共训练了 26 个 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。

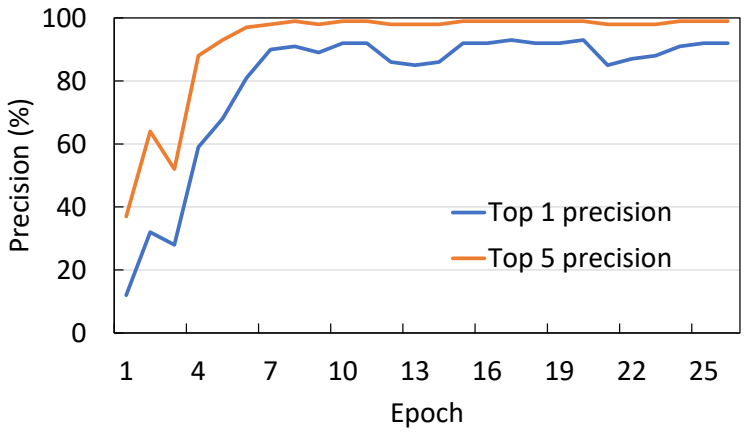


图 5: ResNet-101 训练效果

5 下阶段计划

1. 实现训练好的模型上的快速 inference 程序。
2. 用自己拍的照片测试模型的实际效果。
3. 可能会实现一个动态网页或微信小程序，将 PLANET 做成应用。

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] 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.

[2] 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.

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