人工智能导论大作业二

深度学习

植物叶片分类 & 植物叶脉提取

班级：自 66　自 64

姓名：夏卓凡　魏书凝

学号：2016011496　2016011450

2019 年 5 月 21 日

目录

1 任务综述

本次大作业包括两个任务，一个是实例分类任务，一个是细小物体分割任务，这两个人物都是现在计算机视觉领域常见的监督学习任务。我们将尝试使用一些现在比较流行的方案完成这些任务。本项目的基本流程为数据集准备，模型构建，前向——反向传播（训练），前向传播——输出（测试）；这一流程对两个任务是通用的，所以我们也构建了统一的程序执行框架。

植物叶片分类 由同学们收集校园中的三种叶子的图片，然后进行分类。三类植物分别为冬青、紫丁香和五叶爬山虎，使用类别编号为 0、1 和 2 进行标记。总的来说这个任务非常简单，只需要使用比较常见的识别任务模型，如 VGG、ResNet 等架构即可达到不错的效果。

植物叶脉提取 对于干净（没有嘈杂背景、光线适中、叶子占据图片主要部分）的叶子图片，勾勒出叶子上的叶脉。这个任务由于数据集标注质量不高，而且部分叶子图片拍摄质量不足，我们尝试了多种架构的分割任务模型，但效果差强人意，并没有实现非常理想的精准分割。

2 数据集处理

我们将数据集整理为如下结构，其中 `data1` 为任务一的相关数据，按照类别分为 3 个文件夹，`data2` 为任务二的相关数据，按照叶片和叶脉分为两个文件夹，作为任务二的 image 和 label。注意要将 `data2/leaf/2016013282 (1~10).jpg` 重命名为 `data2/leaf/2016013282 (1~10).jpg`；并且要注意将所有扩展名为`.JPG` 的文件重命名为`.jpg`。

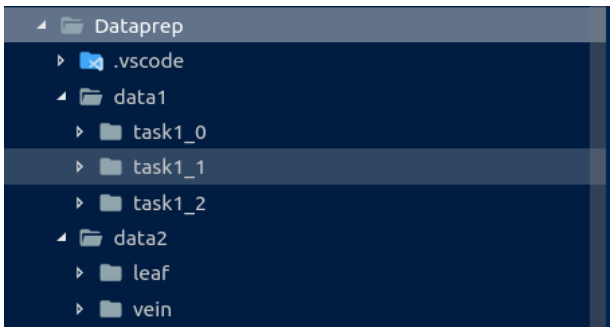

图 1: 处理前的数据集结构

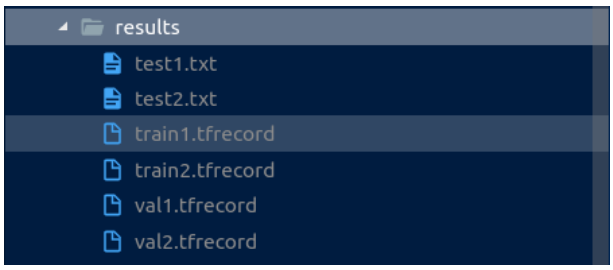

图 2: 处理后的数据集结构

接下来我们将对这些数据进行整理，代码见 `data_prepare.py`。对于任务一，我们在 3 个类别中各随机选取 50 张图片作为测试集、50 张图片作为验证集，其余每类约 850 张组成训练集；对于这些图片，我们将其降采样到 128×128 大小，训练集和验证集压缩进 `tfrecord` 文件，测试集仅保留

图片路径和类别编号，存成文本文件。对于任务二，我们在 536 张图片中随机选取 20 张图片作为测试集、20 张图片作为验证集，其余 496 张图片作为训练集；同样，我们将图片降采样到 1024×1024 后切块为 256×256 大小，训练集和验证集压缩进 `tfrecord` 文件，测试集保留 image 和 label 的路径，存成文本文件。准备完成后的数据集如图 2 所示。

3 任务一——植物叶片分类

3.1 Pipeline 说明

如下图所示，任务一的 pipeline 非常简单，即将数据样本归一化之后送入网络；网络的输入为 `tf.float32` 的三分类 logits，计算 loss 和 metric 时由于函数接受 logits 而无需转换为 prediction 的概率；在训练阶段，使用 ADAM 优化器 [1] 对模型参数进行更新；在测试阶段需要给出预测值的时候再对 logits 使用 softmax 函数转换为概率值，取最大的下标作为分类结果输出。

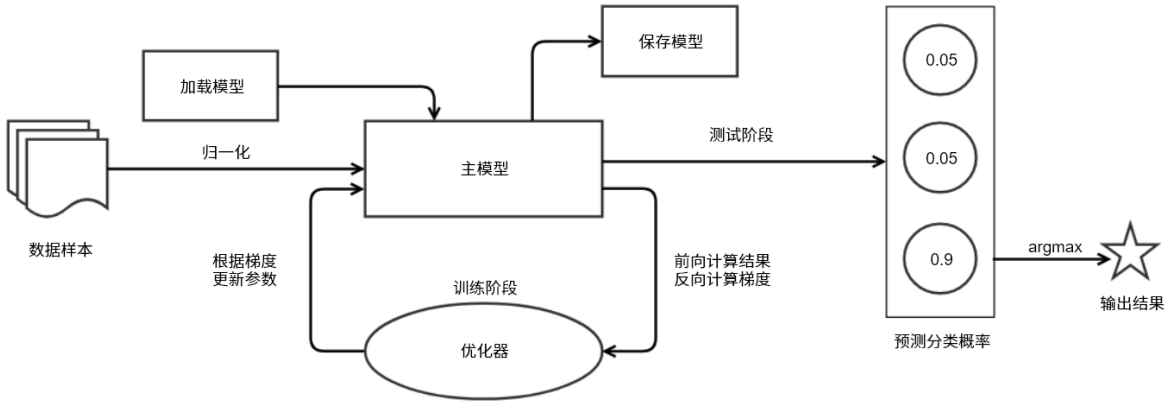

图 3: 任务一的 Pipeline 说明

3.2 简化的 VGG 模型

3.2.1 VGG 模型

我们的模型借鉴了 VGG-16 网络 [2] 的结构，VGG-16 与 VGG-19 是牛津大学视觉几何组（Visual Geometry Group）的 *Karen Simonyan* 和 *Andrew Zisserman* 提出的，发表于 ICLR2015。在当时，这类具有“卷积——池化——全连接”的模型在分类问题上表现出了良好的性能。

图 4: VGG 网络的结构

VGG-16 与 VGG-19 结构相似，不同配置如下图所示，VGG-16 对应表中的 D，VGG-19 对应表中的 E，训练数据为 ImageNet 这样的超大规模数据集。而针对任务一，我们认为这样的网络结构太大，由于其没有 ResNet 中的 skip connection[3] 以及 batch normalization 层 [4] 帮助训练，容

易发生训练不稳定，难以收敛等问题。因此我们决定用最简单的结构“Conv-ReLU-MaxPooling”来搭建一个较为简单的模型。

3.2.2 适用于任务一的简化 VGG 模型

我们最终简化的模型如下图示。其中“k3n64s1”表示卷积核尺寸为 3×3，输出通道数为 64，卷积步长为 1，“4096”表示全连接层输出维度为 4096，所有最大池化层的步长均为 2，“Flatten”表示将图像数据展开成一个列向量。

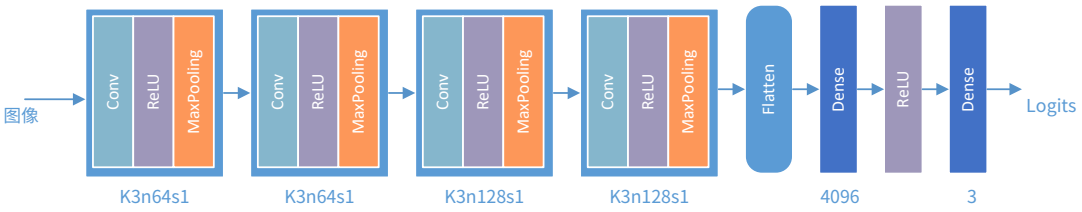

图 5: 简化的 VGG 网络

使用的 loss 函数为分类交叉熵（Categorial Cross Entropy）函数，定义为

$$L_{CCE}(\hat{y},y)=\sum_{i=0}^{C-1}-y_i\log\hat{y}_i$$

这个损失函数是分类问题中最常使用的损失函数，接下来的所有分类问题都将使用这个损失函数。

3.3 ResNet-50 模型

3.3.1 ResNet 的原理

Resnet 网络 [3] 最早是由微软研究院的何恺明等四人提出，发表于 CVPR2016，主要使为了解决随着网络层数的加深，准确率下降的问题。其主要的思想是在网络中增加直连通道，类似于 Highway Network[5] 的思想，即允许原始的输入信息跳过中间的几层，直接传到后面的层中，即得到了我们的残差学习单元，这样的模块就可以不用学习整个的输出，而是学习上一个网络输出的残差，因此 ResNet 也叫做残差网络。

3.3.2 残差模块

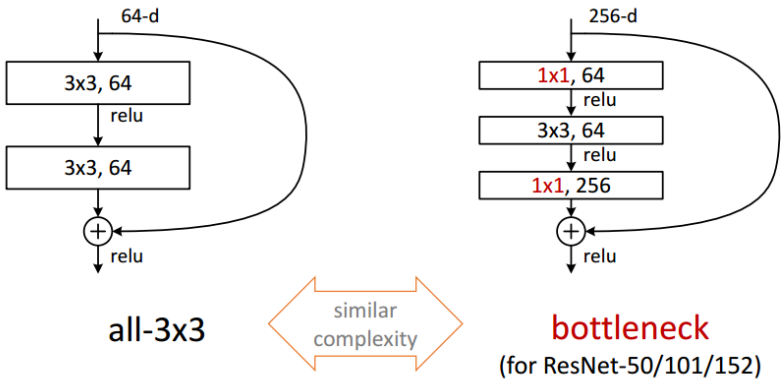

图 6: 两种 Building Block：基线型（baseline）和瓶颈型（bottleneck）

ResNet 中可能会用到两种残差模块，一种是两个 3×3 的卷积网络串接在一起作为一个残差单元（基线型），例外一种是 1×1，3×3，1×1 的 3 个卷积网络串接在一起作为一个残差模块（瓶颈型），如上图所示。

3.3.3 网络结构选择

ResNet 有不同的网络层数，比较常用的是 50 层、101 层和 152 层, 其网络结构如下图所示。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 7: ResNet 网络配置

本次作业中，主要采用了 3 层的残差学习单元（瓶颈型），构建了 ResNet-50 网络，如下图所示。其中我们将最后的 Dense 层的输出维度改为 3，以适应任务一。代码实现过程中我们把这样的一个残差单元封装成一个小的 bottle neck 模型，而在小的 bottle neck 中引入了 BN 来优化模型。

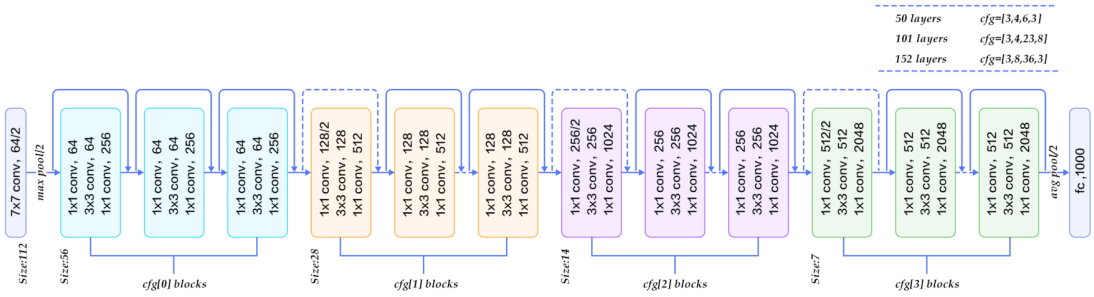

图 8: ResNet-50 网络结构

3.3.4 跳层连接中的 channel 数

channel 数相同直接相加，即恒等映射，这被证明是最优的情况

$$y = F(x) + x$$

channel 数不同时卷积后调整维度再相加，即

$$y = F(x) + Wx$$

其中，加入 W 可以选择恒等映射或者 1×1 卷积对输入 x 进行升维经过 batch normalization 层，本次大作业中采取“恒等映射”的方法。

3.3.5 Batch Normalization 的引入

批量归一化（batch normalization，BN）[4] 是一种对输出结构均值方差规范化的表达层，在神经网络训练时遇到收敛速度慢或梯度爆炸等无法训练的状况时，可以使用 BN 来解决，用来提高模型精度加快训练精度。

BN 的原理为

$$\hat{x}^{(k)} = \frac{x^{(k)} - E(x^{(k)})}{\sqrt{Var(x^{(k)})}}$$

$$y^{(k)} = \gamma x^{(k)} + \beta$$

事实上，这一层将每一个功能层（如卷积、全连接等）的输入端进行归一化，训练时使用数据均值和标准差的移动平均估计，测试时使用训练时固定的均值和标准差，使得输入总是满足标准正态分布。另一方面，数据的线性变换被单独建模，γ 和 β 用于将归一化后的数据映射到原输入。这样做减轻了功能层的负担，使这些层无需拟合与数据分布有关的信息，而专注于特征提取，有效地减少了过拟合现象而且使得收敛更加容易。

3.3.6 ResNet 的优点

综上所述，ResNet 模型有以下优点：

- 学习过程更加灵活更加鲁棒，网络可以在更广范围的学习率下工作；
- 削弱了层间参数的依赖关系；
- 有正则化的效果。

4 任务二——植物叶脉提取

4.1 Pipeline 说明

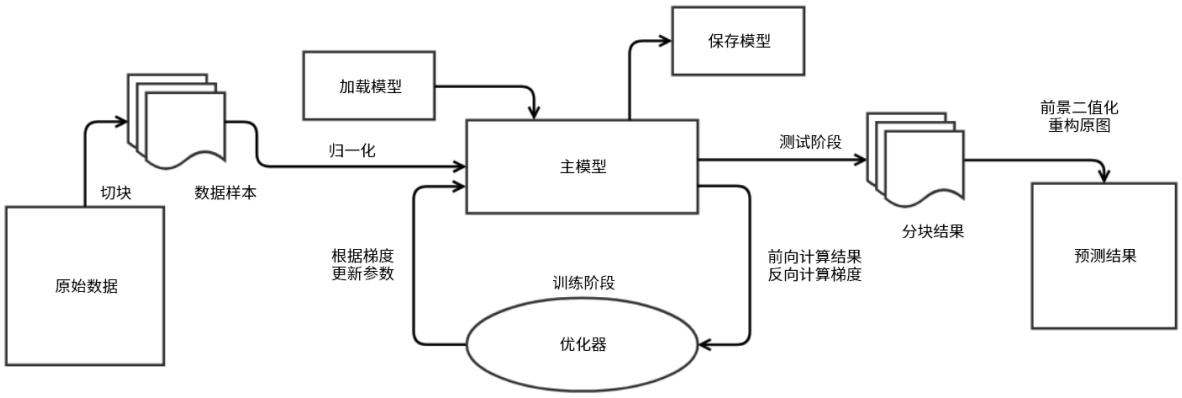

图 9: 任务二的 Pipeline 说明

与任务一相似，任务二的 pipeline 如上图所示。值得注意的是，对于分割任务，分辨率是比较重要的考量之一。考虑到原始图片的分辨率在 1000 到 5000 之间，而现有的显存十分有限（12GB），

如此大的分辨率不可能整体送入网络，只能在分辨率上做一些 trade-off。另一方面，过低的分辨率也会不利于模型的表现，过细的 label 和 image 中的叶脉会让模型训练不稳定。所以，我们先将数据集缩放到 1024×1024，再切成 16 块大小为 256×256 的 patch 送入网络。得到的输出取前景预测 map，使用 0.5 阈值进行二值化，得到输出的图像块，再重组为 1024×1024 大小，最后缩放回输入大小。使用的优化器仍是 ADAM 优化器 [1]。

4.2 简化的 U-Net 模型

4.2.1 U-Net 模型

U-Net 网络结构 [6] 由 *Olaf Ronneberger* 等人提出，发表于 ICMICCAI2015。它最初用于医疗图像的分割，但其架构非常有指导意义，对一般的图像分割任务也能起到非常好的效果。图像分割通常被建模为密集预测问题，即 Pixelwise 层次的分类，这与一般的分类问题有较大不同；其次，其输出为一张应与输入大小相同的预测结果，通常这被成为 mask。全卷积网络的结构在这个问题上非常受欢迎，即网络全由卷积层、激活层、池化层和 BN 层等组成，不包含全连接层。U-Net 也是全卷积结构，主要特点为不同分辨率的特征会在重建时进行拼接（concatenate），达到兼顾全局与细节的效果，其主要结构如下图所示。

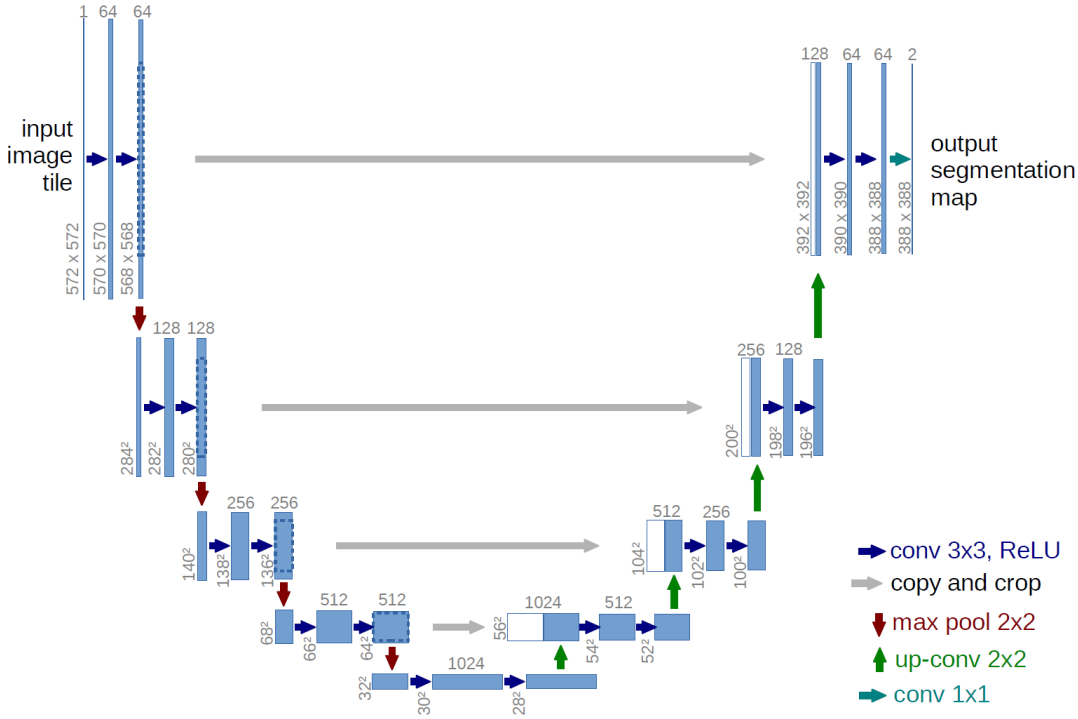

图 10: U-Net 模型结构

4.2.2 简化的 U-Net 模型

针对任务二，我们对 U-Net 模型进行了相应的简化，由于输入图像大小为 256×256，比原来 572×572 的一半还小，所以我们减少了一个“降采样——升采样”环节，并且修改了卷积层的相关参数，使之产生大小更规则的 feature map。简化后的模型如下图示。

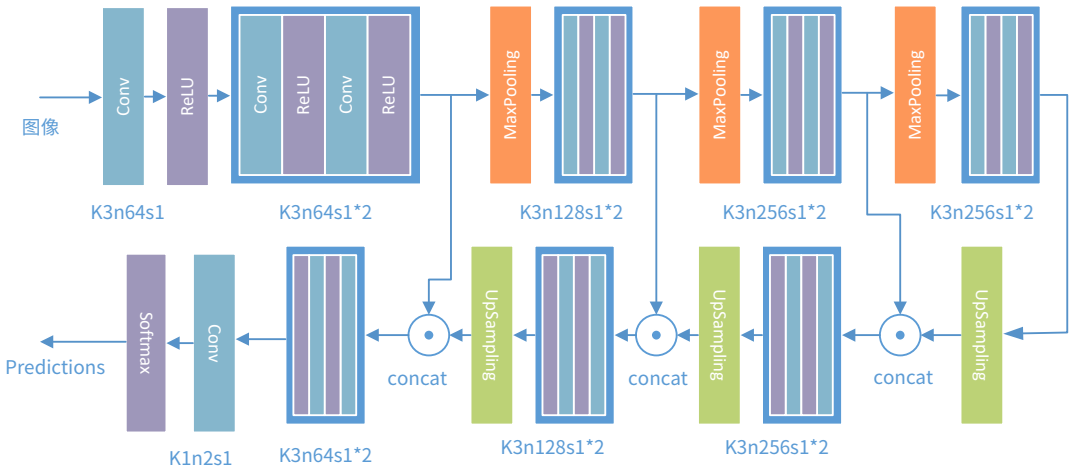

图 11: 简化的 U-Net 网络

为了形成有效的监督，我们在最后一层输出令网络输出 2 张 prediction map，分别对前景预测结果和背景预测结果进行监督，这一策略有效地抑制了对于纯背景 patch 的误判现象。我们依据 V-Net 论文 [7] 中提出的改进 dice loss 设计了损失函数，定义如下

$$L_{DICE}(y,\hat{y}) = \frac{2\sum_{i=1}^{N} y_i\hat{y}_i + \epsilon}{\sum_{i=1}^{N} y_i^2 + \sum_{i=1}^{N} \hat{y}_i^2 + \epsilon}$$

相应的 metric 也为 dice 系数，定义为

$$\mathrm{DICE}(y,\hat{y}) = \frac{2\sum_{i=1}^{N} |y_i\hat{y}_i| + \epsilon}{\sum_{i=1}^{N} |y_i| + \sum_{i=1}^{N} |\hat{y}_i| + \epsilon}$$

除了定义阶数有不同之外，在作为 metric 时，输入为 hard 量，即预测值 $\hat{y}$ 也通过阈值转换为 0-1 值，而 loss 中预测值 $\hat{y}$ 为 soft 量，即在 $[0,1]$ 的浮点数值。接下来的所有模型的 loss 与 metric 都用这两个。

4.3 FCN 模型

4.3.1 原理与结构

传统意义上的 CNN 能够实现图像级别的分类，通过多层网络结构，学习图像的多个图像的特征，较浅的层具有较小的感知域，能够学习一些局部特征，而较深的卷积层具有更大的感知域，可以得到一些抽象特征，这些抽象特征对物体的大小位置方向敏感度更低，从而有助于识别物体。

但是对于输出图像中每个像素具体属于哪一个物体即解决图像分割领域的问题，CNN 就会出现存储开销大计算效率低下的问题。因此 *Jonathan Long* 等人提出了 FCN[8]，发表于 CVPR2015，试图从图像分类级别过渡到像素分类级别。

FCN 有两种结构，分别是 FCN-8 与 FCN-32，结构如下图所示。其结构上的核心思想：

- 不包含全连接层，而是使用全卷积，从而适应任意尺寸输入；
- 上采样即加入反卷积层，输出更加精细的结果；
- 加入 skip 结构，能够融合多层的输出，因为底层网络的感受野更好能够看到小的 pixel，从而预测使输出包含更多的位置信息。

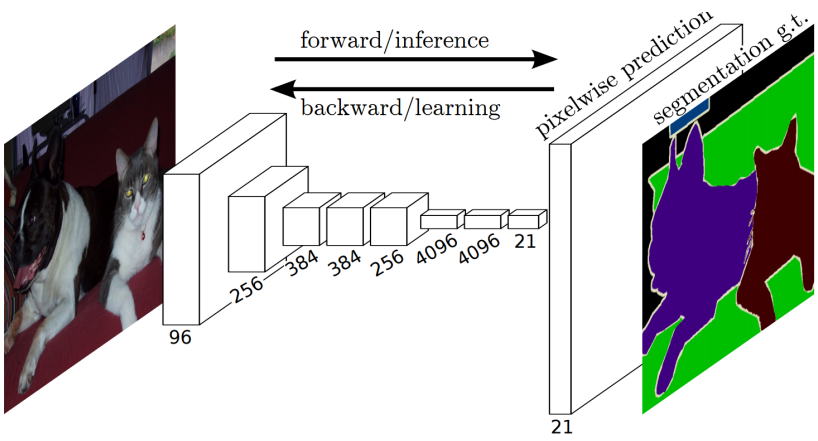

图 12: FCN-32 模型整体结构

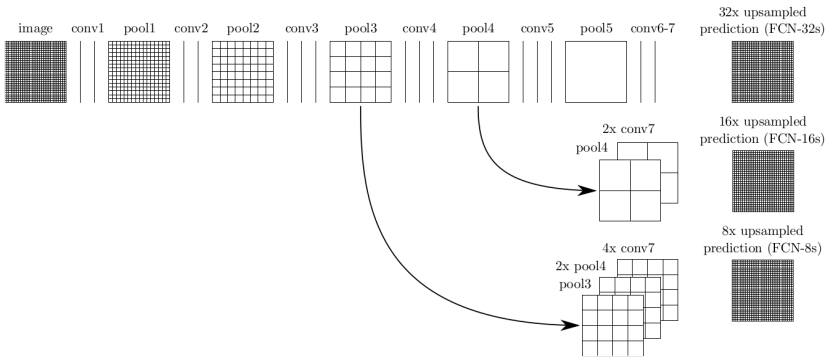

图 13: FCN-8 模型改进后的结构

4.3.2 模型的缺点

- 得到的结果不够精细，上采样的结果比较模糊和平滑，对图像中的细节不够敏感；
- 对各个像素进行分类，没有充分考虑像素与像素之间的关系，忽略了像素与像素之间的空间关系。

4.4 Hourglass 模型

4.4.1 Hourglass 构建块

Hourglass[9] 模型由 *Newell Alejandro* 等人提出，发表于 ECCV2016，是一种用来进行人体姿态估计的算法。这个模型可以以 pixel wise 的形式给出人体骨架的 heatmap，我们认为这个结构事实上也可以用在任务二上。除此之外，另一篇发表于 CVPR2017 的文章 *FSRNet: End-to-end learning face super-resolution with facial priors*[10]，由 *Yu Chen* 等人完成的人脸超分辨率工作中使用到了人脸 parsing map 的先验，而其类似语义分割的任务则恰好用到了这个 hourglass 结构，这也启发我们在任务二中引入这个结构。

Hourglass 块由一组降采样和升采样的残差块构成，每个残差块均完成一次特征传递，而升降采样的过程使用步长为 2 的卷积/解卷积层完成。与 U-Net 架构不同，这里采用残差连接，即相加，对各个尺度的特征进行融合，而不是采用拼接将各个尺度特征进行融合。标准的模块结包含 4 层升降采样和相应尺度的残差连接，具体构建块如下图所示。

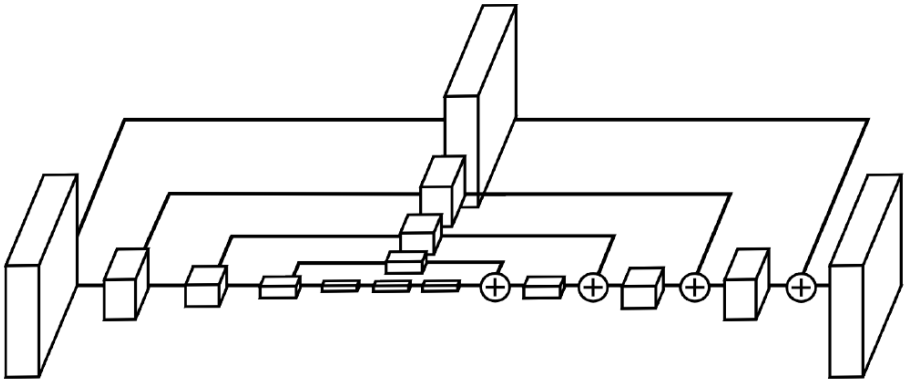

图 14: Hourglass 模块

4.4.2 将 Hourglass 结构用于任务二

由于一个 Hourglass 在 256×256 输入大小的情况下已经足够占满一张 *NVIDIA GeForce GTX 1080 Ti*，我们设计的网络结构仅包含一个构建块，具体的结构如下图所示。

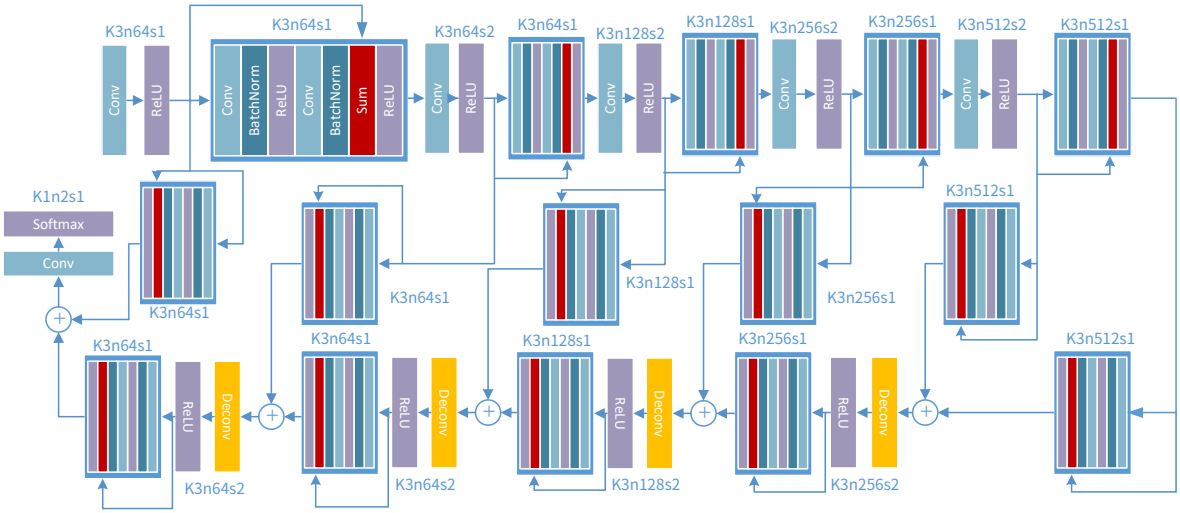

图 15: Hourglass 网络

5 程序编写与实验

5.1 程序结构说明

程序使用 Python3.6 开发，运行于 Ubuntu16.04 系统。深度学习框架为 TensorFlow1.13.1（GPU 版），需要额外安装的包有 Pillow，opencv-python、matplotlib、seaborn 以及 scikit-learn。代码已经上传至我的 Github[1]。

程序整体分为 main、solver、arch 和 data 共四个部分，程序结构仿照 SRFBN 网络[2]进行构建。main 部分见 `main.py`，加载配置文件，控制程序的启动，以及进行 GPU 有关的设定。solver 的代码

[1] https://github.com/Vladimir2506/Leaves

[2] https://github.com/Paper99/SRFBN_CVPR19

用于解决对应的问题，我们为任务一和任务二各写了一个 solver，处理整套 pipeline 以及实现 train，test 和 play 三个功能。arch 为实际上各个模型的定义，所有模型用继承 `tf.keras.Model` 实现，这样可以统一调用的方式。data 主要是用来处理数据集，准备训练、验证和测试用的数据。

为了跟进技术的发展，迎接 TensorFlow2.0 时代，我们使用 Eager Execution 模式规划程序的流程，和 `tf.keras` 作为高阶 API 构建模型，并使用 `tf.data` 构建数据管道以及相关数据增强操作。这一切极大地提高了我们的开发速度，也方便我们实时调试模型。

5.2 任务一实验

我们针对任务一进行了如下设置的实验。对简化 VGG 模型，设定 batch_size = 64，初始学习率为 1×10^{-5} 进行训练，学习率衰减策略为每 5000 减为原来的一半，使用随机反转作为数据增强，一共训练 20000 步，为防止过拟合加入权重衰减的系数为 0.01。对 ResNet-50 模型，设定 batch_size = 32，初始学习率为 1×10^{-4}，进行训练，学习率衰减策略为每 5000 减为原来的一半，使用随机反转作为数据增强，一共训练 20000 步，为防止过拟合加入了概率为 0.5 的 dropout，连接在卷积层 flatten 后的位置。以上实验的指标为验证集准确率，当验证集准确率较高时（$> 95\%$），我们认为模型训练完成，因此我们不做刻意的学习曲线监测。以上实验均在 *NVIDIA GeForce GTX 1080 Ti* 显卡上完成。

我们在基于上面随机划分的测试集中进行对模型进行测试，模型读入一张原始图片，缩放为 128×128 的大小，之后送入网络，网络输出对应的 logits，这些 logits 在经过 softmax 函数转换为置信概率，从而给出可能性最大的下标作为输出类别。下表显示了两个模型在测试集上的准确率，以及混淆矩阵。

模型	简化 VGG	ResNet-50
测试集准确率	92.00%	96.00%

表 1: 任务一测试集准确率

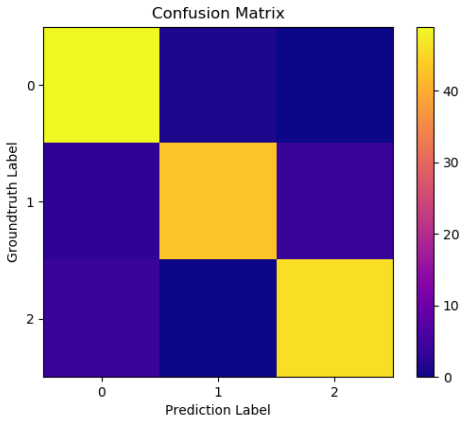

图 16: 简化 VGG 模型的混淆矩阵

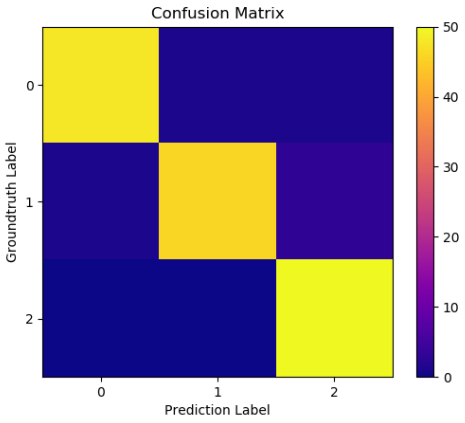

图 17: ResNet-50 模型的混淆矩阵

可见两个模型都在识别任务上取得了不错的效果，而 ResNet-50 模型的准确率稍微高一些。一开始我认为简化 VGG 的效果远远低于 ResNet-50，但实际上他们的准确率相差不大。从混淆矩阵可以看出来，类 1 的分类难度可能较大。而对于任务一而言，可以认为简化的 VGG 模型已经可以

得到令人接受结果。这说明对于一些小的数据集，可以使用更加简单的模型，复杂的模型并不一定在任何情况下都是更好的。

5.3 任务二实验

我们针对任务二进行了如下设置的实验。对任务二的四种模型，设定 batch_size = 16，初始学习率为 1×10^{-4} 进行训练，学习率衰减策略为每 5000 减为原来的一半，使用随机反转作为数据增强，一共训练 20000 步，为防止过拟合加入权重衰减的系数为 0.001。以上实验均在 *NVIDIA GeForce GTX 1080 Ti* 显卡上完成。

由于各种数值度量的效果均不够理想，我们仅在下面给出一些视觉结果（Qualitative Results），并简要讨论数据集的质量问题。下面各图中从左到右分别为输入图片，分割结果以及标注。

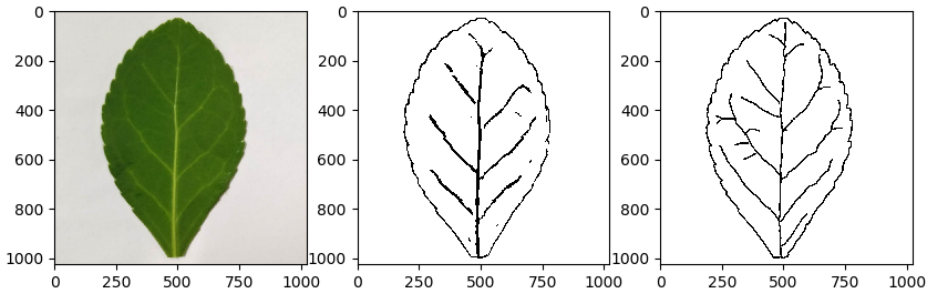

图 18: 简化 U-Net 模型在样例 1 上的分割表现

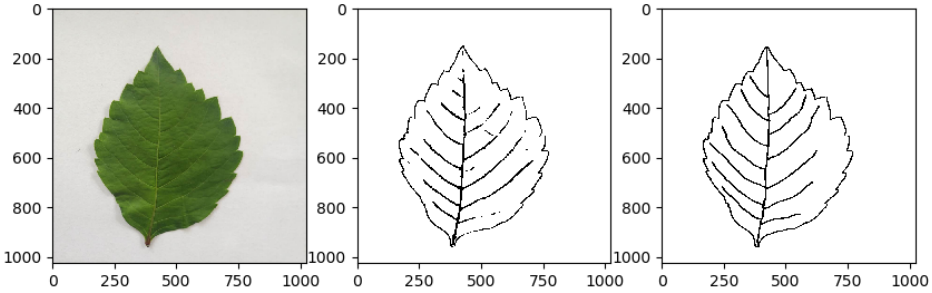

图 19: 简化 U-Net 模型在样例 2 上的分割表现

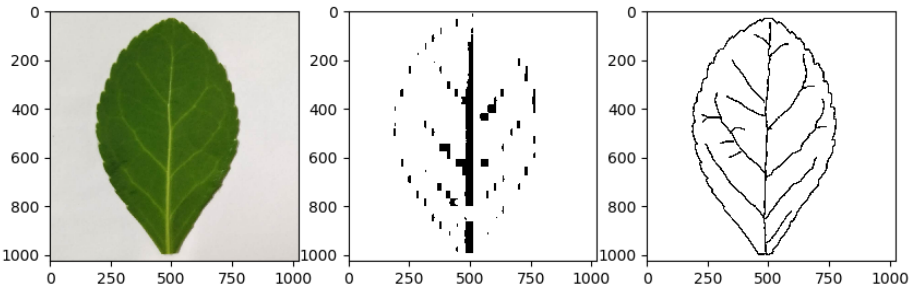

图 20: FCN-32 模型在样例 1 上的分割表现

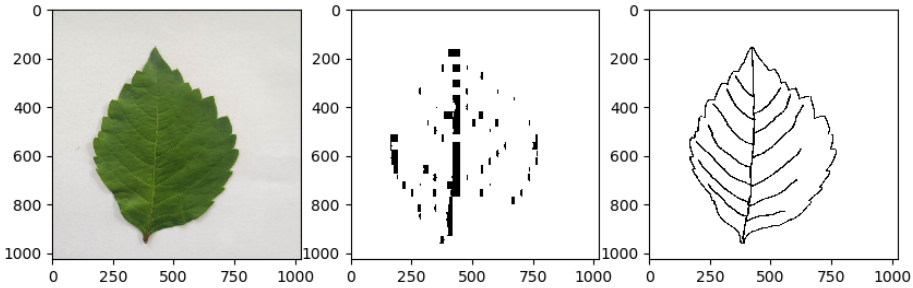

图 21: FCN-32 模型在样例 2 上的分割表现

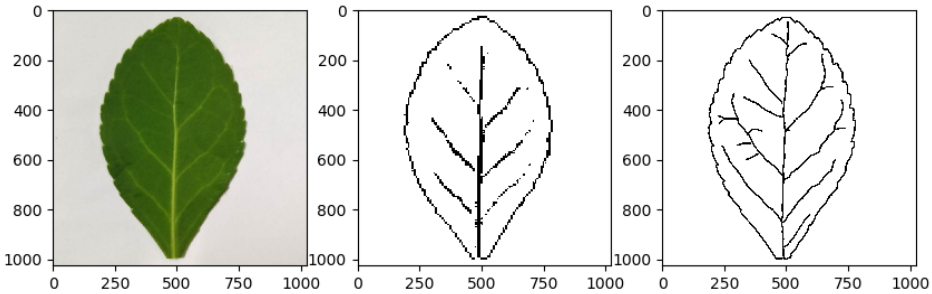

图 22: FCN-8 模型在样例 1 上的分割表现

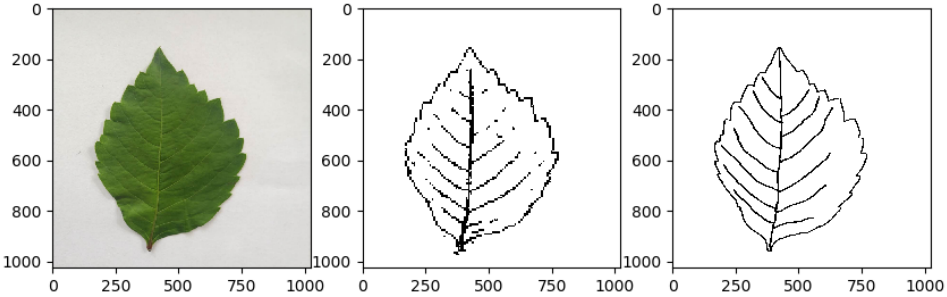

图 23: FCN-8 模型在样例 2 上的分割表现

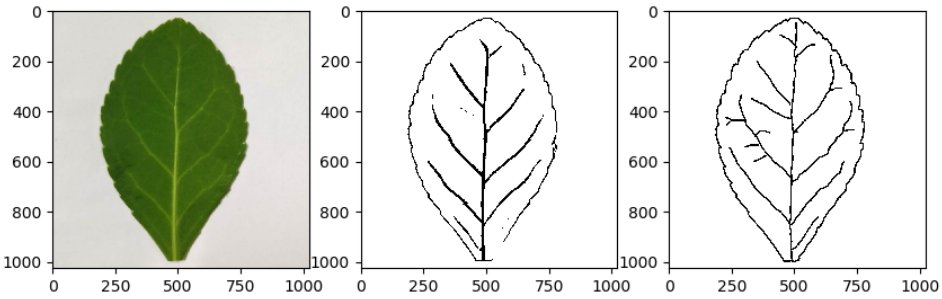

图 24: Hourglass 模型在样例 1 上的分割表现

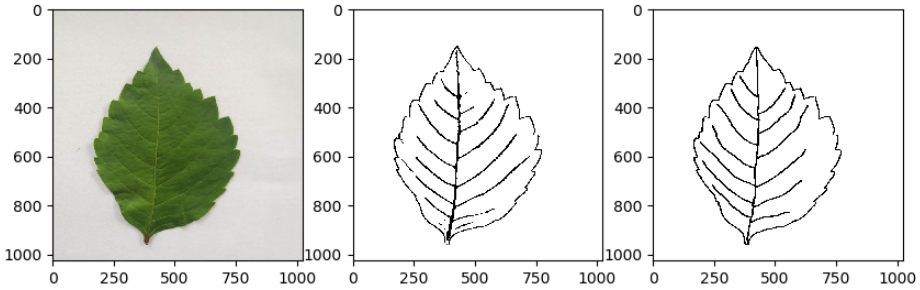

图 25: Hourglass 模型在样例 2 上的分割表现

上面为一些好的样本，即分割效果可以接受的样本。从这些样本的对比我们可以看出，FCN-32 模型的细节效果不够理想，原因可能在于其上采样所用的分辨率过低，以至于无法给出精细的结果。FCN-32 的改进模型 FCN-8 事实上也存在相应的细节不足的问题，而简化 U-Net 模型和 Hourglass 模型可以获得较好的分割结果。

事实上，仔细观察可以发现，模型的分割结果与标注总是有一些差距。但这些分割结果与输入图像更加接近，我们认为这很有可能是出现了过度标注或者错误标注的问题，这也可能是数值指标（Quantitative Results）不佳，无法衡量分割效果的原因之一。下面展示一些分割效果不佳和过度标注的样本，以 Hourglass 模型的结果为例。

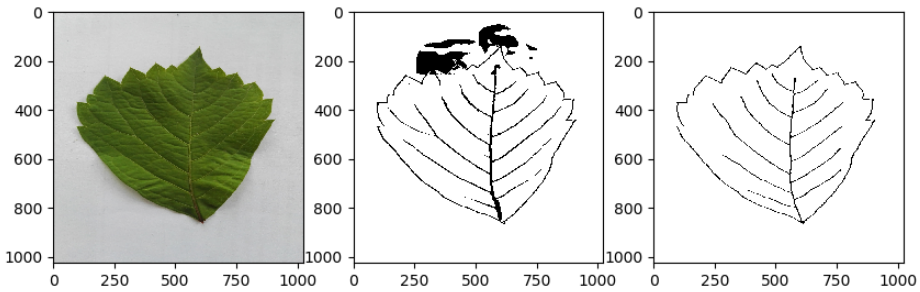

图 26: 分割效果不佳

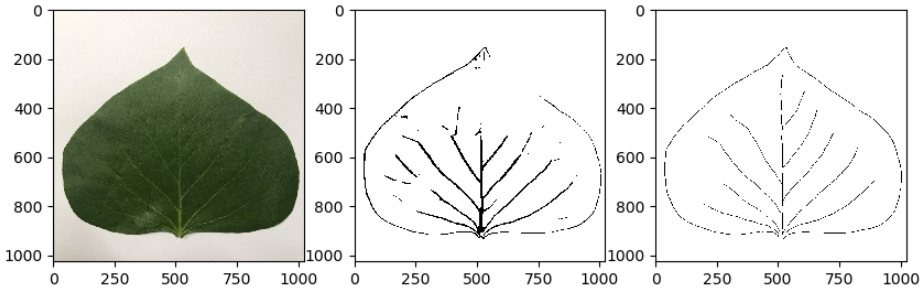

图 27: 分割效果不佳 & 过度标注

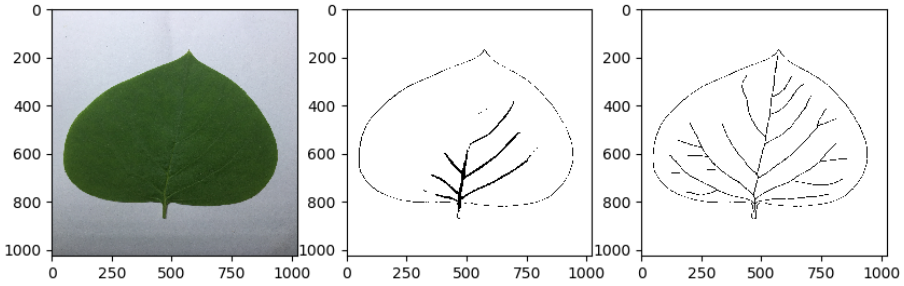

图 28: 过度标注

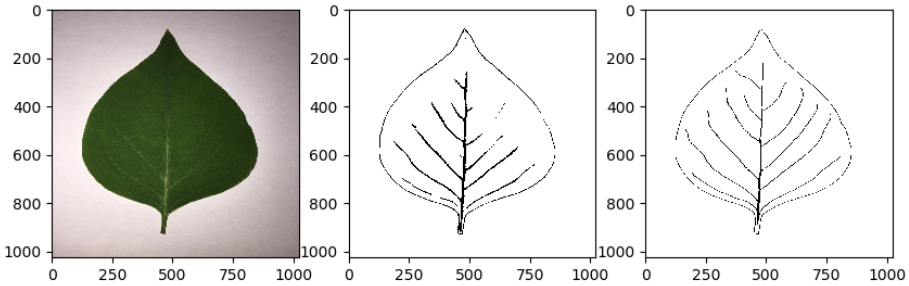

图 29: 过度标注

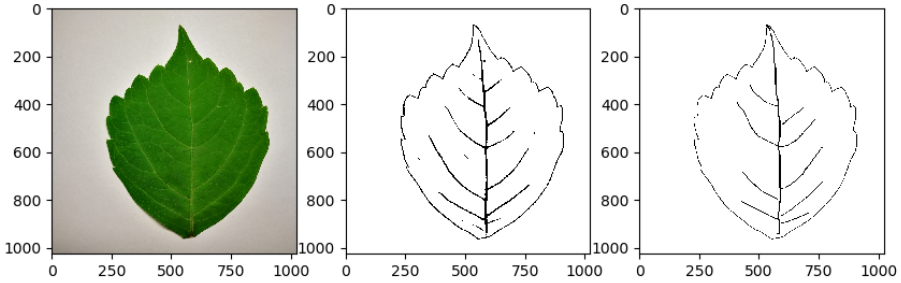

图 30: 过度标注

6 总结与反思

6.1 夏卓凡

本次大作业验证了机器学习中的一个道理“你的模型就是你喂进去的数据”，无论是任务一还是任务二，无论我们采取什么样的提高模型性能的 trick，模型最终的性能还是要靠训练数据的质量决定。任务一中数据量适中而数据中所蕴含的模式比较有限，甚至我们用比较简单的模型也能取得可以接受的结果；ResNet-50 模型固然强大，但几百张树叶图片的三分类问题也并不能完全发挥其威力。任务二中的数据由于标注质量不高，并且叶脉的分割属于分割任务中较难的细小物体分割，这两个因素共同决定了模型的效果很难做到非常理想。

本次大作业从数据采集到数据预处理都让我感到数据的重要性，以及一个没有高质量数据的深度学习问题解决起来是多么的困难。对比我在实验室中的项目，它们大多有着高质量标注的数据集，这样才能使各种超参数调优技巧成为可能。而本次大作业没有这样的条件，在这种低质量数据集上能跑通模型就已经十分不易了。我也通过这次锻炼体会到了高质量数据集的重要性以及数据标注人员的辛苦。

6.2 魏书凝

过拟合问题的解决 过拟合，是指当一个模型过为复杂时，它可以很好的记忆每一个训练数据中随机噪声的部分而忘记了要去“学习”训练数据中的的通用趋势，造成模型在训练集上能够跑到很高的精确度，但是在测试集上效果很差。为了解决这一问题，主要有以下几种方法：

- 正则化

  主要思想是在损失函数中加入用于刻画模型复杂度的指标，希望通过限制权重的大小，使得模型不能任意拟合训练的随机噪音，即损失函数的优化目标由

  $$J(\theta) \to J(\theta) + \mu R(w)$$

  其中 $R(w)$ 利用层的权重参数刻画模型复杂度。

  正则化分为以下两种方式

  - L1 正则化 $R(w) = ||w||_1 = \sum_i |w_i|$
  - L2 正则化 $R(w) = ||w||_2^2 = \sum_i w_i^2$

L1 正则化能使参数变得稀疏，类似于特征化，L2 正则化能弱化小的参数，在模型优化中可以忽略，本次作业中大量使用了 L2 正则化，FCN32 中使用了 0.005 的 L2 正则化。

- Dropout 层加入

  其主要思想是使得某一层的神经元随机失活，从而增强模型的泛化能力，此方法在 Resnet-50 和 FCN 中均有使用。

- BN 层的加入（上文已提到）

Keras 模型的搭建 Keras 中主要提供了两种模型搭建的方法：一种是序列模型（`sequential()`），另一种是使用函数式 API 的 `Model` 模型，本次作业中主要模型的搭建采用对 `Model` 类的继承，使用重写 `__init__()` 和 `call()` 的方式搭建自己的模型，此种方法搭建的 `Model` 更为灵活。

参考文献

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[7] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[9] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

[10] Yu Chen, Ying Tai, Xiaoming Liu, Chunhua Shen, and Jian Yang. Fsrnet: End-to-end learning face super-resolution with facial priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2492–2501, 2018.

A 程序执行方法

检查作业 将模型文件从清华云盘[3]上下载下来,放在./Models/下。若执行任务一,则修改./run1.sh 中要执行的配置文件,并打开./Options/中的配置文件进行修改。指定 gpu_id 来指定显卡编号,在 datasets 的 play 中指定 dir 为输入图片路径列表的文本文件,并将 is_training 设为 play。完成后执行

```
$sh ./run1.sh
```

若执行任务二,则修改./run2.sh 中要载入的配置文件。并需要在 datasets 的 play 中指定 in 为输入图片路径列表的文本文件,out 为输出图片的路径,默认为./Plays/predictions/。完成后执行

```
$sh ./run2.sh
```

训练模型 将 is_training 设为 train,将数据集[4][5][6]按照报告所述进行处理并按照配置文件中的配置修改相应路径与参数,设定完毕后执行 shell 脚本即可。

测试模型 将 is_training 设为 test,测试模式下任务一会输出混淆矩阵,任务二会输出视觉结果,设定完毕后执行 shell 脚本即可。

B 任务分工

魏书凝 任务一中完成 ResNet-50 的搭建调试,以及相应的配置文件,并编写相应的可供测试的函数 Solver1.play();任务二中完成 FCN8 和 FCN32 的搭建调试及相应的配置文件。

夏卓凡 整体程序框架的编写,包括数据处理,Solver 中 pipeline 的构建,以及 Solver2.play() 功能和相关测试代码;任务一中简化 VGG 模型的搭建,任务二中简化 U-Net 模型和 Hourglass 模型的搭建,以及上述模型的调试。

[3] https://cloud.tsinghua.edu.cn/f/2c53f5ab33bc4c73bac7/
[4] https://cloud.tsinghua.edu.cn/f/4c4598e345504caea3cf/
[5] https://cloud.tsinghua.edu.cn/f/16e98ca0bef04233a38d/
[6] https://cloud.tsinghua.edu.cn/f/887a1cc3bd494669b6d5/