

实验报告

第 32 组

仇亦禹 208011239 张昱恒 208011235 郭彦江 2018010835

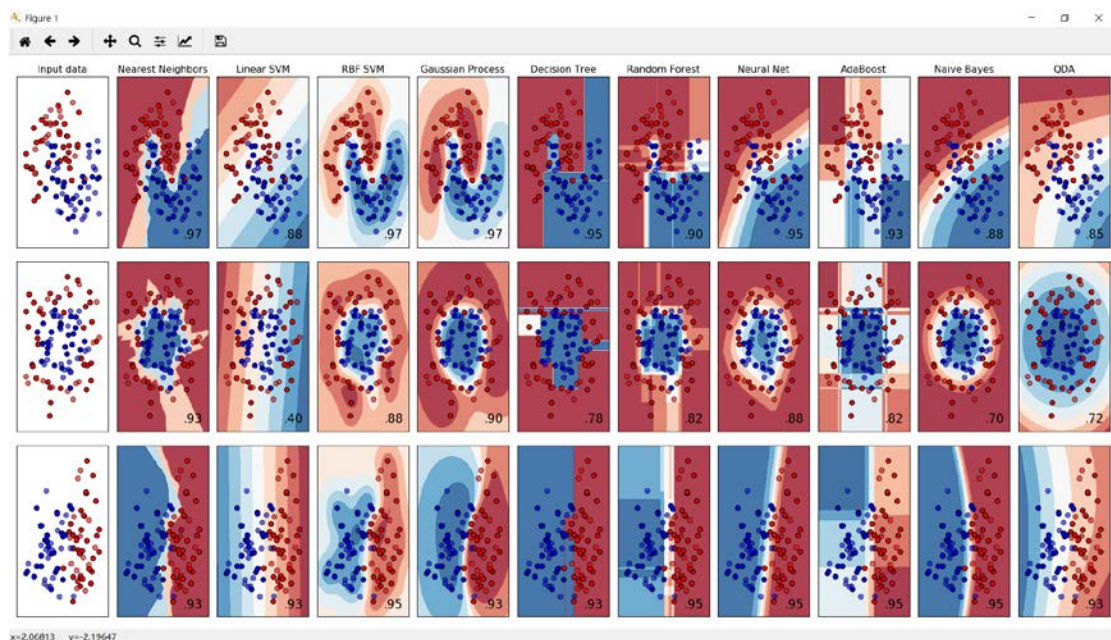
一、 任务一

1. 文献调研

本人利用传统机器学习方法对 19 类交通标志进行分类，即在特征提取的基础上利用分类器进行分类操作。

常用的特征提取方法有 sift, hog, lbp 等等，调研对比多种特征提取方法的特点与适用情况，发现 hog 对于刚性物体的提取具有良好的特性，因此考虑使用 hog 方法进行特征提取。

本次实验为包含 19 个类别的多维分类任务，调查发现针对此种任务的分类方法有 svm, adaboost, random forest, naïve Bayes 等等，文献中针对三种简单的非线性样本测试了不同分类器的分类效果，其结果如下图：



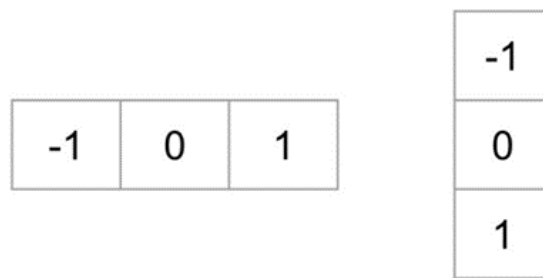
对比之下，选择分类效果相对较好的 svm 方法。

特征提取与分类的具体方法在下文阐明。

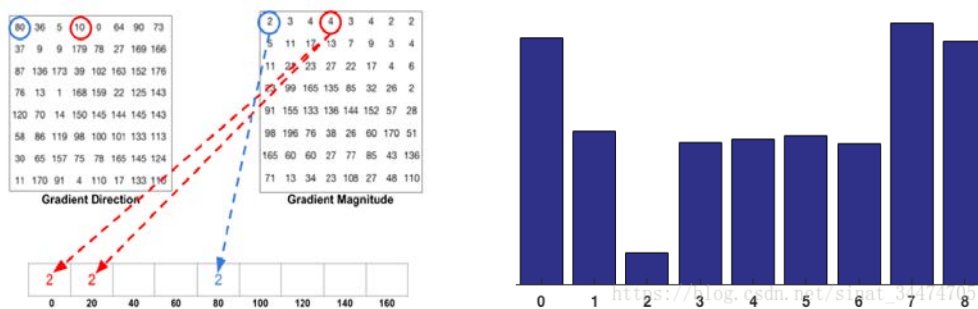
2. 方法原理

与其他特征描述子相同，hog 将一张大小为 $\text{width} \times \text{height} \times 3$ 的图片转化为一个长度为 n 的一维特征向量。对于 hog，其提取了图像中梯度方向的分布，对于交通标志这种线条、边框分明的物体应该有较好的提取效果。

其具体原理如下：首先计算图像在水平和数值方向的梯度，反应在代码上即用下图所示核对图像进行卷积操作。

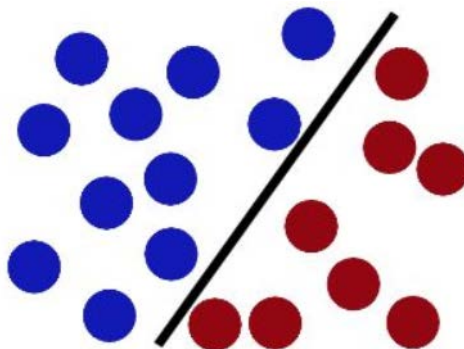


计算出 g_x, g_y , 此后每个像素点梯度 $g = \sqrt{g_x^2 + g_y^2}, \theta = \arctan \frac{g_y}{g_x}$, 再根据梯度的大小与方向创建梯度直方图, 每个 cell 的大小根据图像的大小可自行进行调整。

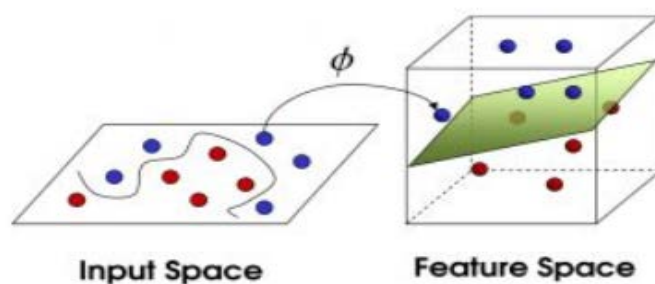


对得到的梯度直方图进行归一化操作, 以此忽略图像亮度对数据的影响。

有关 SVM, 简短直观地说, 对于如下图所示的线性可分的数据, 这种分类器会找到一个距离合适的超平面将两种数据分割开来。



而对于线性不可分的数据, SVM 会通过核函数将其投影到更高的维度, 从而实现线性可分, 如下图所示:



由此看来 SVM 本身是一个二分类器, 要实现多分类问题需要进行一定的改进, 目前已知的方法有两种, 第一类方法主要思想是在优化公式的同时考虑所有的类别数据, 该算法在

经典的 SVM 理论的基础上，重新构造多类分类型，同时考虑多个类别，然后将问题也转化为一个解决二次规划(Quadratic Programming, 简称 QP)问题，从而实现多分类。该算法由于涉及到的变量繁多，选取的目标函数复杂，实现起来比较困难，计算复杂度高。

第二类方法的基本思想是通过组合多个二分类器实现对多分类器的构造，常见的构造方法有“一对一”和“一对其余”两种。其中“一对一”方法需要对 n 类训练数据两两组合，构建 $n(n-1)/2$ 个支持向量机，每个支持向量机训练两种不同类别的数据，最后分类的时候采取“投票”的方式决定分类结果。“一对其余”方法对 n 分类问题构建 n 个支持向量机，每个支持向量机负责区分本类数据和非本类数据。该分类器为每个类构造一个支持向量机，第 k 个支持向量机在第 k 类和其余 $n-1$ 个类之间构造一个超平面，最后结果由输出离分界面距离 $wx+b$ 最大的那个支持向量机决定。

3. 方案设计

利用 `pytorch` 中的 `imagefolder` 函数读取文件夹保存图片数据，并将图片大小统一为 64×64 像素，随后利用 `opencv` 中的 `HOGDescriptor` 方法进行图片的特征提取。

初始化 `HOGDescriptor` 参数，`winSize` 为 16×16 ，即将整张图片分割成 16×16 的窗口，窗口扫描步长为 `winStride=8`，并在每个窗口中，用大小为 8×8 ，每次移动步长为 4 的 `block` 进行扫描，而每个 `block` 又分为四个 4×4 的 `cell`，在每个 `cell` 中计算 6 个方向（即每个方向 30° ）的梯度大小。

最终得到：窗口数量 \times 每个窗口 `block` 个数 \times 每个 `block` 中 `cell` 个数 \times 梯度方向个数即 $49 \times 9 \times 4 \times 6 = 10584$ 维的向量。

利用 `scikitlearn` 中的 `svm` 构建分类器，考虑到实验给出的训练与数据集之比大约为 4:1，利用 `train_test_split` 函数将数据按照 `label` 标签中类别的比例按 4:1 分为训练集与验证集两个部分，并对数据进行随机打乱。

由此准备好了训练集与测试集，就可以开始对分类器进行训练并得到验证集上的训练结果了，通过对验证集上结果的分析调整训练方法。

4. 实验结果

为了直观感受 `hog` 对实验的效果，前几次实验并未采用 `hog` 特征提取，而是将图片的矩阵 `flatten` 之后直接当做训练集进行训练。

首次执行代码的过程中，由于没有仔细地思考，在读取图片并统一尺寸的过程中采用了随机切割的方式，且将图像读成了灰度图的形式，导致了验证集上只有 0.31 的准确率。

随后的第二次尝试过程中，观察发现不同交通标志的颜色差距很大，于是本次尝试中读取了图像的颜色信息，验证集上的准确率达到 0.6，仔细观察训练的结果可以发现，在区分不同颜色的标志时这次的模型已经有很不错的效果，但同一颜色的交通标志，比如不同的数字标志区分的效果很差。

第三次实验，将 `transform` 中的随机切割函数变为了采用插值函数的 `resize` 函数，这次的准确率达到 0.85，但在仍如同第二次实验中提到的，在区分同一颜色不同数字的标志时准确率较低。

第四次实验中加入了 `hog` 特征提取，在验证集上得到了 0.95 的正确率，这已经基本符合我们的预期了。

最终结果如下图：

	precision	recall	f1-score	support
0.0	1.00	0.88	0.94	58
1.0	0.96	0.97	0.97	111
2.0	0.99	0.99	0.99	258
3.0	0.91	0.94	0.92	139
4.0	0.97	0.97	0.97	34
5.0	0.95	0.95	0.95	245
6.0	1.00	0.78	0.88	23
7.0	0.88	0.94	0.91	121
8.0	1.00	1.00	1.00	46
9.0	0.97	0.87	0.92	86
10.0	0.94	0.98	0.96	220
11.0	0.98	0.91	0.94	64
12.0	0.94	0.95	0.94	167
13.0	0.98	0.92	0.95	130
14.0	0.98	0.96	0.97	141
15.0	0.90	0.99	0.94	476
16.0	0.99	0.99	0.99	340
17.0	0.91	0.75	0.82	187
18.0	1.00	0.98	0.99	47
accuracy			0.95	2893
macro avg	0.96	0.93	0.95	2893
weighted avg	0.95	0.95	0.95	2893

5. 代码运行方式

作业中提交的 `task1_train.py` 需要与 `Train` 文件夹放在同一目录下运行，否则应注意修改 `imagefolder` 函数的路径信息，程序运行后会在 `py` 文件目录下生成 `model` 文件。

该代码用到的工具包含 `opencv`, `scikitlearn`, `torchvision` 以及 `matplotlib`。

`task1_test.py` 应与运行 `task1_train.py` 代码得到的 `model` 文件置于同目录下运行，否则应注意修改 `joblib.load` 函数的路径信息。

6. 问题与反思

实验中遇到最大的问题就是在使用 `imagefolder` 函数分文件夹读取图片的时候，该函数将文件夹按照名称排序进行数字编号，在随后编写 `json` 文件时，我想当然地为数字编号是按照 `windows` 文件夹中的名称排序进行的，这就导致了在数字转回交通标志名称时有七个种类的交通标志名称是错误的，这也同样影响到了我们组任务 2,4 的 `json` 文件编写，导致在第一次提交给助教进行测试时我们组任务 1,2,4 的结果都出乎意料的低。如右图：

可见中间七类的分类结果基本是全错的。这也提醒我在日后的编程实践中不能想当然，还是要对结果仔细验证一下。

```

classi2 recall:0.902703
classi4 recall:0.961957
classi5 recall:0.925234
classio recall:0.733607
classip recall:0.994186
classp11 recall:0.000000
classp23 recall:0.011111
classp26 recall:0.000000
classp5 recall:0.000000
classpl30 recall:0.004808
classpl40 recall:0.000000
classpl5 recall:0.000000
classpl50 recall:0.924242
classpl60 recall:0.983051
classpl80 recall:0.983425
classpn recall:0.607362
classpne recall:0.946602
classpo recall:0.794872
classw57 recall:0.933333
Accuracy: 0.572105

```

二、 任务二

1. 文献调研：

分类的任务是卷积网络较擅长的任务。在发展历史上有很多的网络都有很好的效果，从手写数字的识别 `LeNet-5`, `AlexNet`, `VGG16`, 后来引入残差网络 `ResNet`, `DenseNet`。卷积网络基本的思想往往是使用多次的 `conv+relu+pooling` 的组合，不断提取图中的特征，低层次的特征图中包含基本内容多一些例如横竖色彩纹理，高层次的特征图包含多更复杂的内

容，例如不同的组合形状，最后使用 softmax 回归分类。

2. 方法原理

利用卷积神经网络，采用多层 conv+relu+pooling 的组合，不断提取图中的特征。最后用几层的全连接层 FC+Softmax 进行分类。

可以采用迁移学习的方式，将一些已经预训练好的模型作为起点，在这之上 Fine tune 出对自己样本较好的分类网络。

3. 方案设计

3.1 初步分析，分割标注数据

拿到数据集后，首先我观察了一下数据集，数据集总共有 19 类。每一类的数量大不相同，多的有一千多张，少的有一百多张，总共大概有 14000 张训练图片。为了便于今后的研究，我先将带标签的训练集按照 9:1 划分成训练集和验证集，以便于之后对于过拟合的分析，对超参数的调整。

3.2 模仿 LeNet-5 手写数字识别网络

由于分类的种类有 19 类，我首先想到了轻量级的 LeNet-5 网络。该网络在手写数字灰度图上取得了较好的识别效果，原网络分成了 0-9 这 10 类。我稍稍修改了网络，保持其输入的图片尺寸为 32*32，输入 RGB 三张 32*32 尺寸的图片，最后多加了一个全连接层，并将最后一层 softmax 层改为 19 类。由于这是一个较小的网络，参数的总数大概在 13k，相对于总参数大小，我们有较多的图片，因此在这个网络里，我们不太需要担心过拟合的问题。一开始，我担心因为参数个数太少，导致无法学到数据中全部的特征，但是尝试后发现，在 50 个 epoch 的学习后我在训练集获得了 95% 的正确率，在验证集获得了 94% 左右的正确率。已经取得了较好的结果。

3.3 Resnet18 网络迁移学习

接着我打算探究使用更大更深层的网络。查阅资料后我得知，对于深度较大的网络，拥有数量巨大的参数，例如 AlexNet 有大概 60M 个参数，参数数量几千倍于刚才的手写数字网络。如果从随机初始化开始学习这些参数，不但会花费较长的时间，而且数据量不够大的话容易出现过拟合的情况，将图片中的许多噪声也学习到网络中。并且深度较大的网络容易出现梯度消失和梯度爆炸的问题，给我们的训练带来很多的困难。了解了以上的困难后，我打算采用迁移学习的方法，使用一些预训练的网络来加快参数的收敛速度，而且 resnet 这种网络结构可以使得梯度跨层流动，很好地解决了梯度消失的问题，于是使用 Resnet18 作为预训练的网络进行训练。

一开始，我打算固定全连接层之前所有的网络参数，将之前 18 层的卷积网络当作是一个特征提取器，仅调整最后全连接层的参数。最后的全连接层有 512×19 约等于 10k 个参数。冻结前面参数，开始训练后，训练了很多个 epoch 后，发现验证集训练集正确率只能维持在 80% 左右的地方。我们认为，仅仅单层的可调整参数提供的模式是非常有限的，导致最后一层的网络无法较好地学习图片中的所有特征。

于是注释了参数固定的命令，取消了全连接层之前梯度流的冻结。取消了参数的固定的后，这已经变成了一个深层网络。我非常担心出现过拟合的问题，我采取学习率衰减的方式，由于不知道在哪个 epoch 会出现最好的验证集性能，设置了一个 best_model 来记录训练中出现的最好的模型参数，最好的模型参数指在验证集上取得了最高正确率的一轮参数，总共训练了 100 轮，同时采用了学习率衰减的方法。

最好的一个 epoch 得到了 97.9% 的验证集正确率，保存了正确率最高的模型。最后选择使用迁移学习得到的 Resnet18 网络进行测试集分类，首先加载最佳的网络参数，接着将 test

里的图片经过一遍网络，得到最可能的类，进行标注，再写成要求的 json 格式。

4. 实验结果

4.1 修改的 LeNet-5 手写数字识别的最佳网络在训练集获得了 95.0%正确率，在验证集获得了 94.3%正确率。

4.2 Resnet18 迁移学习得到的最佳网络在训练集获得了 99.1%正确率，在验证集获得了 96.7%正确率。

	训练集正确率	测试集正确率
LeNet-5	95.0%	94.3%
冻结卷积层的 Resnet18	82.3%	81.8%
Resnet18	99.1%	96.7%

5. 代码运行方式

5.1 运行 2_LeNet_5.py 可以训练 LeNet5 手写数字识别网络，保存最佳权重参数。

5.2 运行 2_resnet18.py 可以训练 Resnet 迁移学习的网络，保存最佳的权重参数。

5.3 运行 2_test.py 可以加载最佳权重参数，在测试集上进行分类，得到 json 文件。

6. 问题与反思

遇到图像分类问题时，我们首先要对数据的规模有一个大致的了解。根据数据集的规模选择网络。图片的张数、像素大小都应该在考虑范围。训练较大的网络的时候，要防止网络的过拟合，可以采用的办法有：加入一些噪声、在 loss 最低前提前终止网络、将图片翻转切割来增强数据等。

三、 任务三

1. 文献调研

在初期文献调研的过程中，本组阅读了多种算法的相关论文和介绍性质的博客，包括元学习算法（MAML、Reptile），最近邻，迁移学习等。

元学习方法可以从任务一、任务二分类任务的训练集中分割出小的训练样本，反复随机选取小样本训练，在分割出的测试集上预测获得损失函数，更新网络参数，最终获得一个更容易得到结果的、较好的初始网络参数，再用任务三的训练集进行训练。

Fine tune 方法以已经训练好的网络作为基础，对新的分类类别进行训练，能够利用从其他训练集获得的特征提取性能，同时防止训练样本过少造成的过拟合。

权衡之后，为了便于实践，本组决定先从较为简单实用的 fine tune 方法入手，以同为交通标志分类的任务二训练结果作为初始网络参数。

2. 方法原理

由于任务二和任务三的目标都是交通标志的分类，并且任务三训练集较小，因此可以采用迁移学习的方法。交通标志类别不同，但是其中的特征提取有很大一部分是相同的，因此只需要改变任务二网络全连接层的输出个数，并在任务二网络的基础上进行 fine tune，即可获得适合任务三分类任务的卷积神经网络。

3. 方案设计

任务三代码在 cifar10_tutorial 的基础上修改而来，主要分为以下步骤：

- (1) 获取卷积神经网络对象 `net`，载入任务二训练后的网络参数，修改全连接层的输出个数为 11；
- (2) 确定学习率，用训练集进行若干轮次训练；
- (3) 保存训练后的网络参数，准备开始测试；
- (4) 重新获取卷积神经网络对象，载入任务三训练后的网络参数，用测试集数据进行测试，并将预测结果保存在 `json` 文件中。

其中，训练集和测试集可以从任务一、任务二的训练集中分割得到。也可以据此自己验证单样本训练的效果。

为了获得较好的训练效果，用控制变量的方法调整以下超参数，从中选取表现最好的作为最终提交的预测结果：

- (1) 卷积神经网络分别使用 `LeNet-5` 和 `ResNet-18` 两种；
- (2) 修改训练轮次，比较不同训练轮次下的准确率。

4. 实验结果

使用 `LeNet-5` 卷积神经网络对 11 类单样本进行训练，平均召回率（`recall`）在训练 100 个 `epoch` 左右达到最好，约为 60%。

使用 `ResNet-18` 卷积神经，分别采用两种策略：(1)所有层学习率统一；(2)除了最后一个全连接层以外所有其他学习率减小。两种方法相比(1)效果更好，平均召回率（`recall`）在训练 100 个 `epoch` 时为 90%左右。部分测试结果如下表所示。

学习率（ <code>lr</code> ）	1	2	3	平均
<code>lr=1e-3</code>	90.00	90.00	93.00	92.00
<code>[:1]层 lr=0，全连接层 lr=1e-3</code>	87.00	85.00	85.00	85.67

在最终提交给助教的测试集预测结果中，对任务三 11 类交通标志的平均召回率（`recall`）达到 90%，其中对 `p14`、`p27` 类别的预测结果相对不理想。

测试集上的预测结果文件位置在 `/Results/pred3.json`。

5. 代码运行方式

`/Codes/task3.py` 文件完成了(1)网络训练；(2)将测试集上的预测结果写入 `json` 文件。

按以下内容修改代码即可开始运行 `python` 文件：

行数	修改内容
20	修改为训练集所在文件夹地址（按类别保存在 <code>subfolder</code> 中）
21	修改为测试集所在文件夹地址（图片存在 <code>subfolder</code> 中）
22	修改为任务二保存的 <code>ResNet</code> 网络参数（ <code>pth</code> 文件）位置
88	修改为训练后网络参数（ <code>pth</code> 文件）的保存位置
119	修改为测试集上的预测结果（ <code>json</code> 文件）的文件名

6. 问题与反思

在测试集预测结果中，对类别 `p14`、`p27` 的预测相对不理想，原因可能是这两个类别对应的训练集图片亮度较低。实验代码没有对 11 个训练样本做符合图片特性的变换，而是采用了统一的变换方式，可能导致了亮度相对较暗的训练样本对应的类别预测结果较差。

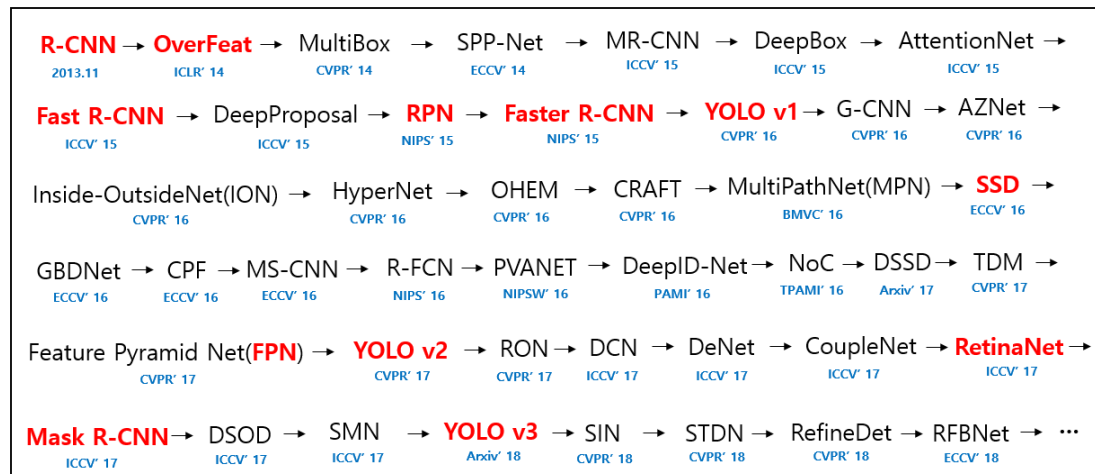
此外，受时间和任务分配限制，本组对任务三仅使用了“在任务二网络参数的基础上 `fine-tune`”一种方法。若要改进，应该更多采用几种方法，如最近邻（`KNN`）、元学习（`MAML`）

模型) 等等, 比较多种方法的效果, 从中选取表现最好的一种。

四、 任务四

1. 文献调研

目标检测任务上, 在深度学习领域涌现了许多优秀的算法和模型。如下图为近年来目标检测领域的部分相关论文。



为了便于实践和调试, 本组决定选取效果较好、代码指导也较为丰富的 YOLOv3 和 Mask R-CNN 两种模型, 其中对于 Mask R-CNN 模型, 分别(1)不输入 mask 数据, 只用 bounding box 数据进行训练, 则网络退化为 Faster R-CNN; (2)输入 mask 数据。也就是说, 本组在任务四中一共使用三种模型进行实验, 比较并从中选取较好的结果对应的方法。

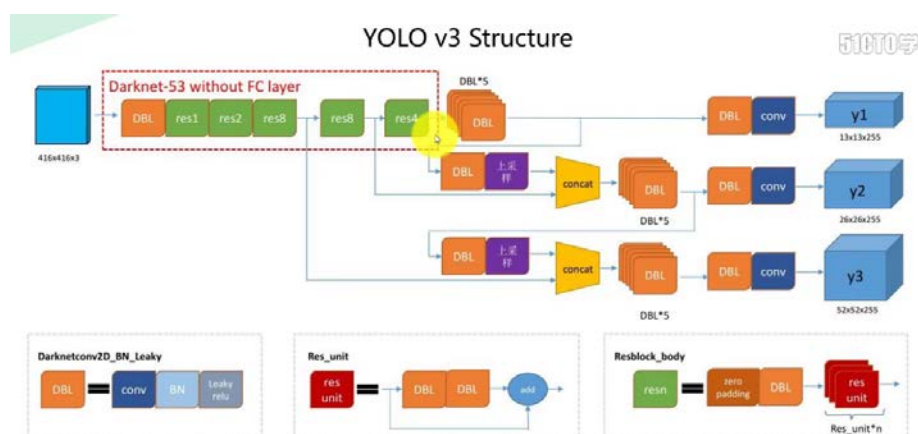
2. 方法原理

1) YOLOv3

yolo 算法不采用窗口滑动, 而是直接将原始图片分割成 $n \times n$ 互不重合的小方块, 然后通过卷积最后生产同样大小的特征图。可以认为特征图的每个元素也是对应原始图片的一个小方块, 然后用每个元素来可以预测那些中心点在该小方格内的目标。

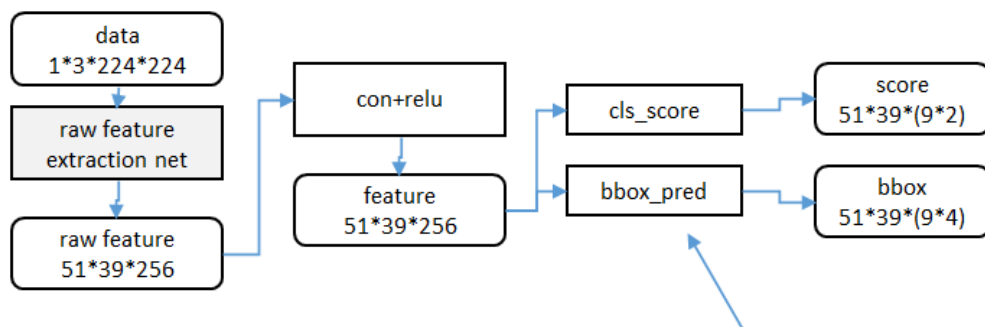
YOLOv3 对之前两代网络做了如下改进:

- ①采用多尺度的融合, 输出三个不同尺度的特征图, 对大小不同的物体进行检测, 也就是按不同大小的方格切割原图。
- ②采用 3 个独立的逻辑分类器替换 softmax 函数, 以计算输入属于特定标签的可能性。



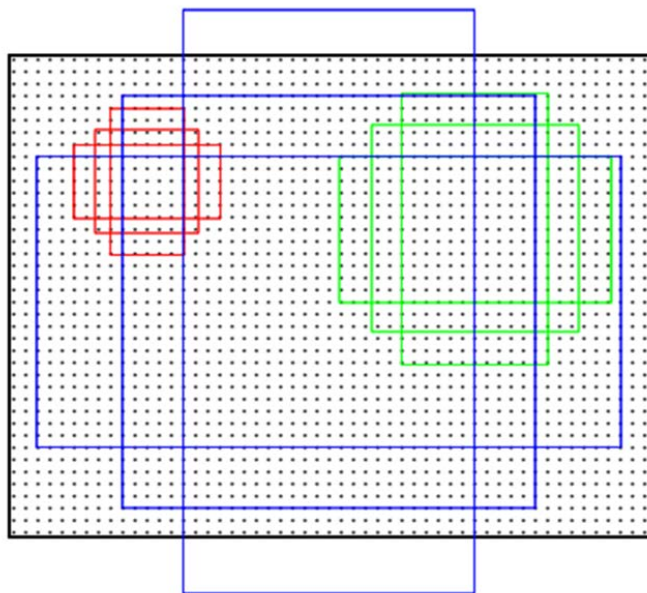
2) Faster R-CNN

Fasterrcnn 的基本思想是在提取好的特征图上对所有可能的候选框进行判别。



Consider 9 'anchors' on each of the 51*39 positions

原始特征提取包含了若干层 conv+relu 网络，最终输出输出 51*39*256 维的特征 (feature)。这个特征可以看作一个尺度 51*39 的 256 通道图像，对于该图像的每一个位置，考虑 9 个可能的候选窗口：三种面积{128,256,512}×三种比例{1:1,1:2,2:1}。这些候选窗口称为 anchors。下图示出 51*39 个 anchor 中心，以及 9 种 anchor 示例。



分类层 (cls_score) 输出每一个位置上，9 个 anchor 属于前景和背景的概率；窗口回归层 (bbox_pred) 输出每一个位置上，9 个 anchor 对应窗口应该平移缩放参数。对于每一个位置来说，分类层从 256 维特征中输出属于前景和背景的概率；窗口回归层从 256 维特征中输出 4 个平移缩放参数。

Fasterrcnn 的损失函数同时最小化两种误差：a.分类误差，b.前景样本的窗口位置偏差。

3) Mask R-CNN

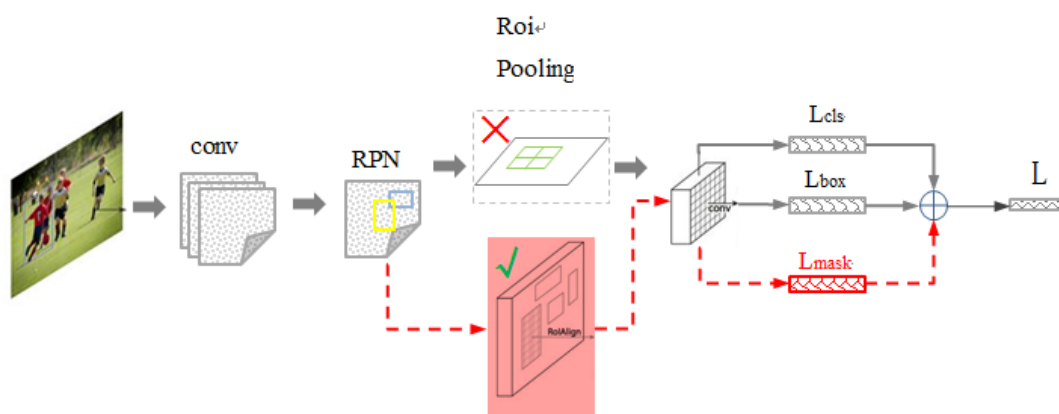
如下图，红色部分为 Mask R-CNN 在 Faster R-CNN 基础上的修改，包括①将 Roi Pooling 层替换成了 RoiAlign；②添加并列的 FCN 层 (mask 层)。

这样的网络结构使该模型具有以下特点：

- ①在边框识别的基础上添加分支网络，用于语义 Mask 识别；

- ②训练简单，相对于 Faster R-CNN 仅增加一个小的 Overhead，可以跑到 5FPS；
- ③可以方便的扩展到其他任务，比如人的姿态估计等；
- ④不借助 Trick，在每个任务上，效果优于目前所有的 single-model entries；

在本实验中，在 Faster R-CNN 的基础上增加了 mask 数据进行训练，以期获得相对更好的目标检测效果。



3. 方案设计

1) YOLOv3

主要参考了一个 GitHub 的项目 <https://github.com/eriklindernoren/PyTorch-YOLOv3>

2) Faster R-CNN

主要参考助教在网络学堂给出的 pytorch 关于 Mask R-CNN 的官方文档，具体步骤如下：

- ①由于本次实验并未提供关于数据集的 mask 数据，因此修改官方文档中的 Dataset 函数，取出其中有关 mask 的方法，并将标定训练集图片的 json 文件中的信息读取到其中；
- ②定义模型调整函数，其中载入预训练模型，注意因为没有 mask 信息，因此载入的预训练模型为 fasterrcnn_resnet50_fpn，且将分类输出类别调整为 2，仅区分背景与交通标志；
- ③在主函数中，运用 pytorch 封装好的函数进行模型训练和评估，并将训练好的模型保存为 task4_fasterrcnn.pth；
- ④在 test 程序中，读取训练好的模型信息，并在测试集上进行预测；
- ⑤利用预测得到的 bbox 数据从测试集图片上切出交通标志的区域，并用任务一、二得到的分类器进行分类，将分类结果保存到 json 文件中。

3) Mask R-CNN

使用 Mask R-CNN 模型的代码在 pytorch 官方文档的基础上修改而来，整体实验方案如下：

- ①构建本实验的 DataSet 类，用标注文件中的多边形/椭圆边界/bbox 边界绘制 mask，使其中方法能按要求返回图片和网络模型需要的各项信息；
- ②定义模型调整函数，其中载入预训练模型，将分类输出类别数修改为 2（本组 Mask R-CNN 仅区分背景和交通标志，分类任务另外完成）；
- ③在主函数中，运用 pytorch 封装好的函数进行模型训练和评估；
- ④将训练后的网络参数保存在 pth 文件中；
- ⑤test 文件中重新载入训练后的网络参数，另一个网络对象载入任务二训练后的网络参

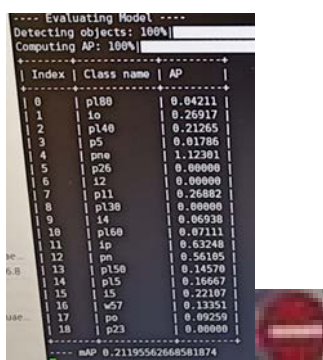
数:

⑥用 Mask R-CNN 网络预测交通标志位置, 从对应图片裁剪出小图, 输入任务二分类器进行分类, 将分类预测结果和 Bounding box 合并成字典写入 json 文件。

其中网络训练不使用训练集最后 50 张图片, 并将其作为测试集自行测试。用控制变量的方法调整网络超参数, 比较预测结果, 从中选取最好的一种。本任务中, 对目标检测网络给出的置信度 score 取不同阈值限制, 进行比较。

4. 实验结果

- 1) Yolov3 网络原本的设计输入为 416*416 像素。在一开始时, 将原本 2048*2048 的图片压缩成了 416*416 大小像素的送进网络进行训练。训练了几十轮后, 发现 accuracy\precision\recall 的值一直无法上升, loss 也停滞不下降。于是停止了训练, 开始观察预测的图片的结果。

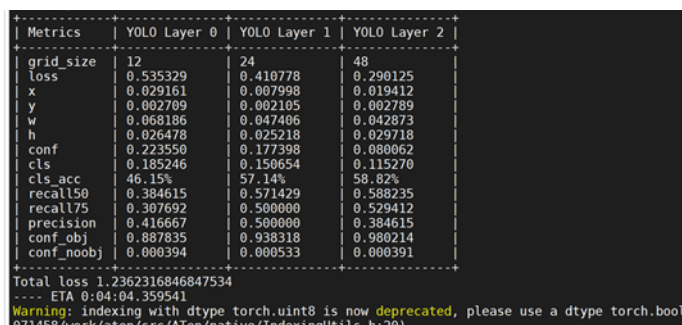


```
---- Evaluating Model ----
Detecting objects: 100%
Computing AP: 100%
+-----+-----+-----+
| Index | Class name | AP |
+-----+-----+-----+
| 0 | pl80 | 0.04211 |
| 1 | lo | 0.26917 |
| 2 | pl40 | 0.21265 |
| 3 | p5 | 0.01786 |
| 4 | pne | 1.12301 |
| 5 | p26 | 0.00000 |
| 6 | l2 | 0.00000 |
| 7 | pl1 | 0.26882 |
| 8 | pl30 | 0.00000 |
| 9 | la | 0.06938 |
| 10 | pl60 | 0.07111 |
| 11 | lp | 0.63248 |
| 12 | pa | 0.56105 |
| 13 | pl50 | 0.14570 |
| 14 | pl5 | 0.16667 |
| 15 | l5 | 0.22187 |
| 16 | w57 | 0.13351 |
| 17 | po | 0.09259 |
| 18 | p23 | 0.00000 |
+-----+-----+-----+
mAP: 0.2119556266581874
```

可以看到有些类别的正确率在较高的水平, 但是有些类别根本无法识别出来, 正确率几乎是零。仔细观察训练集的数据, 发现要检测的图标占的体积大多数都非常小, 大多数图标大概只有 20-40 的像素宽度。于是想到在输入图像时, 对图像进行了压缩, 导致检测到的一个图标只有 5-10 个像素宽度, 在这样的条件下根本不可能对图像进行识别。例如一个数字 8, 从上到下, 应该会由白-黑-白-黑-白-黑-白也有 7 个不同区域, 10 个像素根本不可能刻画出一个可识别的数字。类似 8, 带有数字的图标, 或者一些有复杂图像的图标都根本无法识别出来。

于是想直接将原图送进网络。但是 yolov3 在最后的卷积层到全连接层有一个 $n^2 \times L$ 的矩阵, 图片从原来的 416 放大到 2048 像素时, 最后全连接层的参数增加了 24 倍, 参数规模大大增加, 导致一块 GPU 也无法放下所有的参数, 但是对 YOLO 网络结构的大修改还是不太敢进行。

最后采用了折衷的办法将输入的训练图片压缩为 1024*1024 大小, 进行训练。训练了几十轮后, 大致如下图所示, recall50 一直无法超过 60, 最后检测的效果不是很好。



```
-----+-----+-----+
| Metrics | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+-----+-----+-----+
| grid_size | 12 | 24 | 48 |
| loss | 0.535329 | 0.410778 | 0.290125 |
| x | 0.029161 | 0.007998 | 0.019412 |
| y | 0.002709 | 0.002105 | 0.002789 |
| w | 0.068186 | 0.047406 | 0.042873 |
| h | 0.026478 | 0.025218 | 0.029718 |
| conf | 0.223550 | 0.177398 | 0.080062 |
| cls | 0.185246 | 0.150654 | 0.115270 |
| cls_acc | 46.15% | 57.14% | 58.82% |
| recall50 | 0.384615 | 0.571429 | 0.588235 |
| recall75 | 0.307692 | 0.500000 | 0.529412 |
| precision | 0.416667 | 0.500000 | 0.384615 |
| conf_obj | 0.887835 | 0.938318 | 0.980214 |
| conf_noobj | 0.000394 | 0.000533 | 0.000391 |
+-----+-----+-----+
Total loss 1.2362316846847534
---- ETA 0:04:04.359541
Warning: indexing with dtype torch.uint8 is now deprecated, please use a dtype torch.bool
071458/work/aten/src/ATen/native/IndexingUtils.h:20)
```

2) Faster R-CNN

使用 Faster R-CNN 网络，分别对检测出的对象取不同的置信度（score）阈值限制：若目标检测给出的 score 大于阈值 T，认为是检测出的目标位置；若网络预测给出的 score 小于阈值 T，则认为是误检，不将该对象写入预测结果中。

对 score 分别取不同门限 T，使用不同的分类器，对训练集最后五十张图片进行预测（未用于训练），得到以下 accuracy 和 recall：

分类器	阈值分数 T	Accuracy/%	Recall/%
任务二	0.5	69.80	92.04
	0.7	76.69	90.27
	0.8	81.90	84.07
	0.85	83.78	82.30
	0.9	89.11	79.65
任务一	0.3	58.38	89.38
	0.5	65.77	86.73
	0.7	72.18	84.96
	0.8	76.72	78.76
	0.9	85.15	76.11

可以看到使用任务一所获得的分类器效果是不如任务二的，这也与任务一二分类器得到的测试集准确率相吻合。

3) Mask R-CNN

使用 Mask R-CNN 网络，分别对检测出的对象取不同的置信度（score）阈值限制：若目标检测给出的 score 大于阈值 T，认为是检测出的目标位置；若网络预测给出的 score 小于阈值 T，则认为是误检，不将该对象写入预测结果中。

对 score 分别取不同门限 T，对训练集最后五十张图片进行预测（未用于训练），得到以下 accuracy 和 recall：

阈值分数 T	Accuracy/%	Recall/%
0.0	43.82	97.35
0.1	52.88	97.35
0.2	59.34	95.58
0.3	63.53	95.58
0.4	68.59	94.69
0.5	71.92	92.92
0.6	73.57	91.15
0.7	76.74	87.61
0.8	84.07	84.07
0.9	87.25	78.76

当阈值取 T=0.8 时，accuracy 和 recall 表现均较好。在此阈值下对测试集进行预测，提交给助教检验的结果为：accuracy:0.8452871072589383，recall:0.8674672003557927。

5. 代码运行方式

1) 由于 YOLOV3 网络得到的结果不太理想，不提供可运行的代码。

2) Faster R-CNN

/Codes/task4_fasterrcnn_train.py 完成了网络训练和保存网络参数。注意到除了 pytorch 自带的库以外，本程序的运行还需要使用 cocopytools 工具包，其安装方式如下：

```
>>pip install cython
>>git clone https://github.com/cocodataset/cocoapi.git
>>cd cocoapi/PythonAPI
>>python setup.py build_ext install
```

```
>>git clone https://github.com/pytorch/vision.git
>> cd vision
>> git checkout v0.4.0
>> cp references/detection/utils.py ../
>> cp references/detection/transforms.py ../
>> cp references/detection/coco_eval.py ../
>>cp references/detection/engine.py ../
>>cp references/detection/coco_utils.py ../
```

其余有关代码路径修改的详情可以在代码注释中看到。

/Codes/task4_fasterrcnn_test.py 完成了测试集预测和写 json 文件。需要在相同目录下读取 task4_fasterrcnn_train.py 生成的 pth 文件，运行后在程序目录下生成检测用的 json 文件。

3) Mask R-CNN

/Codes/task4_mask_rcnn.py 完成了网络训练和保存网络参数。按以下内容修改代码即可开始运行 python 文件：

行数	修改内容
15	将 os.listdir()内替换为训练集文件夹地址
18	将 open()内替换为训练集标注文件地址
125	将 torch.load()内替换为 maskrcnn_resnet50_fpn_coco 预训练模型地址
164	如果第 15、18 行改过了这个不改也没关系。如果训练集、标注文件相对位置和服务上一致，也可以只改此处为数据集总目录地址，15/18 不改
210	修改为训练后网络参数（pth 文件）的保存位置

/Codes/task4_mask_rcnn_test.py 完成了测试集预测和写 json 文件。按以下内容修改代码即可开始运行 python 文件：

行数	修改内容
22	将 torch.load()内替换为 maskrcnn_resnet50_fpn_coco 预训练模型地址
50	修改为训练后网络参数（pth 文件）的保存位置
57	修改为任务二保存的 ResNet 网络参数（pth 文件）位置
61	修改为任务四测试集（test）文件夹地址
123	修改为测试集预测结果（json 文件）的文件名

此外，代码中 import 了一些文件，需要预先下载。可以预先执行以下命令：

```
>>pip install cython
>>git clone https://github.com/cocodataset/cocoapi.git
```

```
>>cd cocoapi/PythonAPI
>>python setup.py build_ext install

>>git clone https://github.com/pytorch/vision.git
>> cd vision
>> git checkout v0.4.0
>> cp references/detection/utils.py ../
>> cp references/detection/transforms.py ../
>> cp references/detection/coco_eval.py ../
>>cp references/detection/engine.py ../
>>cp references/detection/coco_utils.py ../
```

6. 问题与反思

1) YOLOv3 曾经是目标检测领域里程碑式的作品。YOU ONLY LOOK ONCE 的快速推断手段曾经带来了令人惊喜的效果，但是 YOLO 在设计是更善于检测大型目标，由于采用了分割区域而不是滑窗的方法，在小目标检测上还是有很多的缺陷，性能上远不如之后提出的 Faster-RCNN、Mask-RCNN 等网络。

五、 成员贡献

仇亦禹在小组中担任组长，完成了任务三的全部内容；在任务四中，使用 Mask R-CNN 架构完成了目标检测和分类任务，获得了组内 accuracy 和 recall 最好的预测结果。

张昱恒在小组中担任组员，完成了任务一的全部内容；在任务四中，使用 Faster R-CNN 架构完成了目标检测和分类任务，获得了组内 accuracy 和 recall 较好的预测结果。

郭彦江在小组中担任组员，完成了任务二的全部内容；在任务四中，使用 YOLO v3 架构完成了目标检测和分类任务，在组内最先完成，为其他方法的实现提供了代码基础。

六、 参考文献

- [1] 图像处理之特征提取[EB/OL].<https://www.jianshu.com/p/d94e558ebe26>,2018-11-
- [2] 梯度方向直方图 Histogram of Oriented Gradients[EB/OL].
https://blog.csdn.net/sinat_34474705/article/details/80219617,2018-05-08.
- [3] 支持向量机通俗导论（理解 SVM 的三层境界）
[EB/OL].http://blog.csdn.net/v_july_v/article/details/7624837,2012-06-01.
- [4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks [J]. CoRR, abs/1703.03400, 2017.
- [5] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. [J] ArXiv, abs/1803.02999, 2018
- [6] Sasank Chilamkurthy, Shishir Patil, Will Feng, et al. TRAINING A CLASSIFIER [EB/OL].
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html, 2019-10-4.
- [7] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. ArXiv, abs/1803.02999, 2018.
- [8] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

- [9] He, Kaiming, et al. "Deep residual learning for image recognition." arXiv preprint arXiv:1512.03385 (2015). [pdf] (ResNet, Very very deep networks, CVPR best paper)
- [10] Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.
- [11] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." arXiv preprint arXiv:1506.02640 (2015). [pdf] (YOLO, Outstanding Work, really practical)
- [12] He, Gkioxari, et al. "Mask R-CNN" arXiv preprint arXiv:1703.06870 (2017).