代码解读：Fast AutoAugment

原创 Zerg_Wang 于 2019-11-26 16:52:34 发布 2061 收藏 5 编辑 版权

分类专栏： Machine Learning 文章标签： 深度学习 augment 数据增强 autoaugment python

官方代码地址：https://github.com/kakaobrain/fast-autoaugment

对Fast AutoAugment（以下简称为FAA）代码的简要笔记。此外，代码中还用了一些作者自己编写的python库，简单看了看，对这些库的使用方法做个笔记。

代码主体

简要来说，作者首先使用自定义的手动增强策略，训练cv_num个模型，然后在这些模型上使用FAA方法搜索策略。搜索过程中产生的策略及相应文件默认保存在~/ray_results下。得到相应的策略（在代码中，存储在变量final_policy_set中）后，作者会再训练num_experiments个使用默认增强策略的模型以及num_experiments个使用搜索得来的策略增强的模型，最后比较这些模型的精度。

一些参数

模型、数据集的指定、训练时的超参数、增强方法等，通过在confs目录下的yaml文件指定，并在代码中通过Config库调用。此外，还有通过argparse传入的参数：

args.num-policy：指定每个策略由多少个子策略组成

args.num-op：指定每个子策略由多少个操作组成args.num-search：搜索深度B（尝试args.num-search个策略，然后从中选最好的，这个数值与~/ray_results下中每个cv中的文件夹数量一致）

args.decay：weight decay参数

args.cv_ratio：训练集和验证集的比例，例如，设为0.4的话，代表给出的非测试集数据（训练集+验证集）中，验证集占40%。

args.until：个人认为没用

args.redis：指定redis服务器，如果是本地运行，可直接屏蔽，然后将后面的ray.init(redis_address=args.redis)改成ray.init()即可。（然而事实是我不会用redis……）

args.dataroot：指定数据集地址

其他参数

cv_num，num_experiments：前文提过，即训练cv_num个模型并分别在这些模型上搜索策略，之后使用搜索得来的策略，训练num_experiments个模型。因为论文中作者提出的密度匹配概念，因此cv_num参数会影响最后搜索出来的增强策略，进而影响增强效果。

num_result_per_cv：每次实验从args.num-search中选出最好的num_result_per_cv个策略。（cv_num乘以num_result_per_cv等于final_policy_set中的策略数量）

Package ：pystopwatch2

一个通过标签（tag）管理的计时器，使用方法如下：

```
from pystopwatch2 import PyStopwatch
w = PyStopwatch()
```

然后在要计时的程序段前：

```
w.start(tag='a')
```

在要计时的程序段后：

```
w.pause(tag='a')
```

输出该程序段执行时间：

```
print(w.get_elapsed('a'))
```

全程序只需一个PyStopwatch对象即可，不同时间段可用不同tag来计时（tag为str类型，可有多个字符），同样通过标签可以嵌套计时。

直接输出w，可看到：（假如之前设置过标签a、b）

```
b: state=ClockState.PAUSE elapsed=2.0007 prev_time=1569658360.29341030
a: state=ClockState.PAUSE elapsed=1.0000 prev_time=1569658361.29409599
```

第一项表示该标签是否被暂停（PAUSE or RUN），第二项表示该标签对应时间段经过的时间，第三个为该标签start的时间，这个值为1970年开始至该时间所经过的秒数，即time.time()。

清理用过的标签：

```
w.clear('a')
```

Package: theconf

该包用于管理参数信息，有两个类Config和ConfigArgumentParser。

在FAA的代码中，作者将每个模型对应的参数信息都编辑相应的yaml文件。然而，虽然都遵守yaml的格式，但不同模型参数信息和参数量都不同，此外，程序实际运行过程中可能不会使用yaml中的默认参数值，需要从控制台中输入自己所需的参数值，因此对参数的管理会较为麻烦，为此作者使用theconf包处理这一过程。而仅需以下代码（也是FAA中所用的代码）即可实现以上功能：

```
from theconf import Config, ConfigArgumentParser
parser = ConfigArgumentParser(conflict_handler='resolve')
parser.add_argument('--SomethingYouWantToAdd', type=str, default='')
... #这里可以加入要额外添加的参数
args = parser.parse_args()
```

第二行代码中，通过ConfigArgumentParser（该类继承自argparse.ArgumentParser）实例化对象parser，这里强制添加了参数--config（简写为-c），用于指定所使用的模型的参数信息——yaml文件的路径。

```
model:
  type: resnet50
dataset: imagenet
aug: inception
cutout: 0
batch: 128 # this value is onl
epoch: 270
lr: 0.05
lr_schedule:
  type: 'resnet'
  warmup:
    multiplier: 32
    epoch: 5
optimizer:
  type: sgd
  nesterov: False
  decay: 0.0001
  clip: 0
```


以上为resnet50_b4906.yaml的内容，可见该文件涵盖了模型大部分的信息。通过theconf包，这些参数信息的值都可从控制台中指定，如果不指定，则默认使用yaml文件中的值。

第三行代码可自己添加yaml文件中没有的参数信息，在FAA中，就添加了模型保存位置（--save）、数据位置（--dataroot）等等。

以上的参数，无论是来自yaml的默认值、自己修改的值，还是额外添加的参数，都统一保存到了Config的类中，该类无需实例化即可使用，例如：

```
Config('resnet50_b4096.yaml')
print(Config.get()['batch'])    # 128
print(Config.get()['optimizer']['type'])    # sgd
```

yaml以及用户自定义的参数都以dict的形式保存在Config中，直接调用方法Config.get()即可调用。

一旦对Config指定了yaml文件，在当前程序中就不能指定另一个（一次只能运行一个模型，再次Config('XX.yaml')会报错），防止了冲突。

参考资料

https://github.com/wbaek/theconf

https://github.com/ildoonet/pystopwatch2

https://github.com/kakaobrain/fast-autoaugment