



fastai lecture2

2018年7月



上海交通大學

SHANGHAI JIAO TONG UNIVERSITY



概述



lecture two核心是介绍几种在深度学习过程中使用的小技巧，可以帮助模型快速达到很好的训练和测试效果。

课程的介绍会分为两部分，第一部分为对各种小技巧的介绍，给出一个完整的训练流程；第二部分为使用介绍的一套训练流程，对不同的数据集进行试验比较。



1

技巧介绍

2

实验测试





1

技巧介绍

2

实验测试





训练流程



- 1、开启数据增强，并设置`precompute=True`；
- 2、使用`lr_find()`函数确定初始学习率，该学习率保证依旧可以明显降低损失函数值；
- 3、在固定原模型参数的基础上，对新加的最后几层训练若干轮；
- 4、设置`cycle_len=1`，使用SGDR再次训练最后一层若干轮；
- 5、Unfreeze模型固定的参数；
- 6、设置浅层的学习率为深层的十分之一到三分之一；
- 7、再次使用`lr_find()`函数寻找学习率；
- 8、设置`cycle_mult=2`，训练整个网络直到收敛。



核心技巧



- 1、学习率的控制
- 2、epoch的设置
- 3、数据增强

- 1) 训练的速度与效果
- 2) 总共训练的次数
- 3) 数据量

相关概念



epoch——所有的训练图像全部通过网络训练的次数

iteration——数据完成一次前向和后向训练的操作

batchsize——在每次iteration中训练图片的数量

e.g. 假如有1280000张图片， $\text{batchsize}=256$ ， 则1个epoch需要 $1280000/256=5000$ 次iteration， 假如系统最大支持的迭代数为450000， 则有 $450000/5000=90$ 个epoch。

相关概念



保证分类结果相同的情况下

裁剪

数据增强

缩放

旋转

翻转

亮度

对比度





相关概念



欠拟合 (underfitting) ——网络训练并没有达到最佳的结果，参数未收敛到较优解，在训练和测试时的loss都较大，准确度较低

过拟合 (overfitting) ——网络在训练集上得到的结果要明显好于在测试集上得到的结果，即训练集上的loss低于测试集上的loss。对应数据较少而模型参数过多的情况

1、开启数据增强，并设置precompute=True

数据增强——对图像的裁剪，目的是为了更方便统一训练和测试。

precompute=True——训练使用的网络为已经训练好的

保证各层参数和输出的activation以及对应的输入的图像都是固定的
典型的图像尺寸为 224×224 或 299×299 。

```
data=ImageClassifierData.from_paths(PATH, tfms=tfms_from_model(arch, sz))
```

数据集的路径 数据集进行处理的函数 训练好的模型 需要裁剪得到的大小

1、开启数据增强，并设置precompute=True

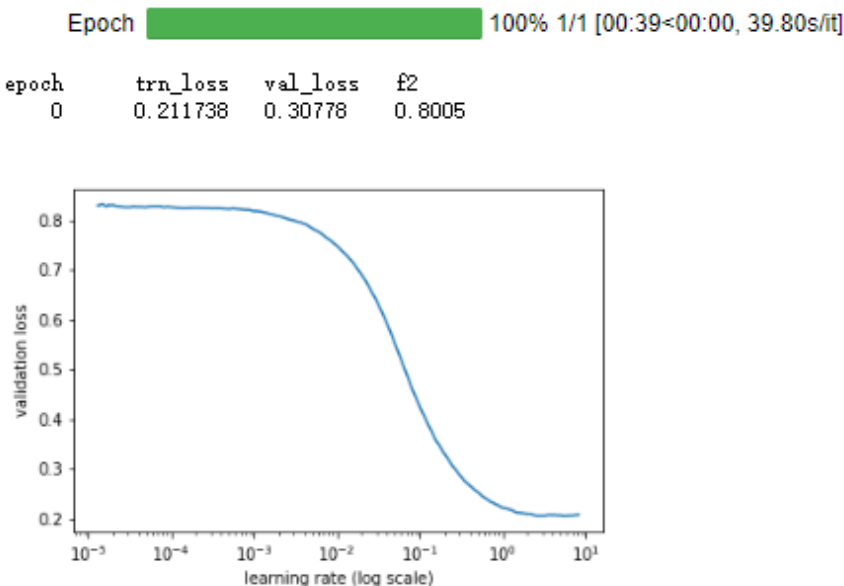
Tips——对于卷积神经网络而言，加入全连接层就意味着输入图像的尺寸必须确定，但是对其他网络而言，卷积层并非一定要求图像的尺寸固定。

同时在开启precompute=True时，数据增强的图像裁剪将失去效果。

2、使用lr_find()函数确定初始学习率，该学习率保证依旧可以明显降低损失函数值

```
In [21]: learn = ConvLearner.pretrained(f_model, data, metrics=metrics)
```

```
In [22]: lrf=learn.lr_find()  
learn.sched.plot()
```



lr_find() 函数会找到使 loss 下降最快的学习率

基本思路是随机选择一个初始点，先使用小的学习率，然后逐步增大，使 loss 下降；当得到下降最快的学习率时记录作为实际使用；之后学习率继续增加，loss 下降速度变缓，直到出现 loss 上升时函数停止。

2、使用lr_find()函数确定初始学习率，该学习率保证依旧可以明显降低损失函数值

Tips——虽然与训练相同，通过读入数据计算得到loss，但是这不属于训练，只是为了找到最优的学习率。

3、在固定原模型参数的基础上，对新加的最后几层训练若干轮



模型的整体架构——训练好的网络+最后一层全连接层

训练方式——图像经过一个已经确定的预处理网络，得到的输出结果作为新加一层的输入，然后仅对新加的一层训练几个epoch

4、设置cycle_len=1，使用SGDR再次训练最后一层若干轮



首先需要知道在训练过程中，会设置学习率逐渐降低。

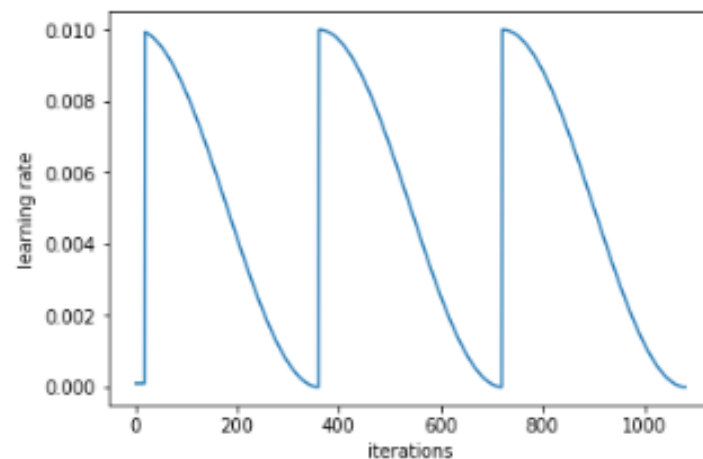
常用的方法为学习率线性下降以及cosine函数下降。

4、设置cycle_len=1



```
learn.fit(1e-2, 3, cycle_len=1)  
learn.sched.plot_lr()
```

In [46]: `learn.sched.plot_lr()`

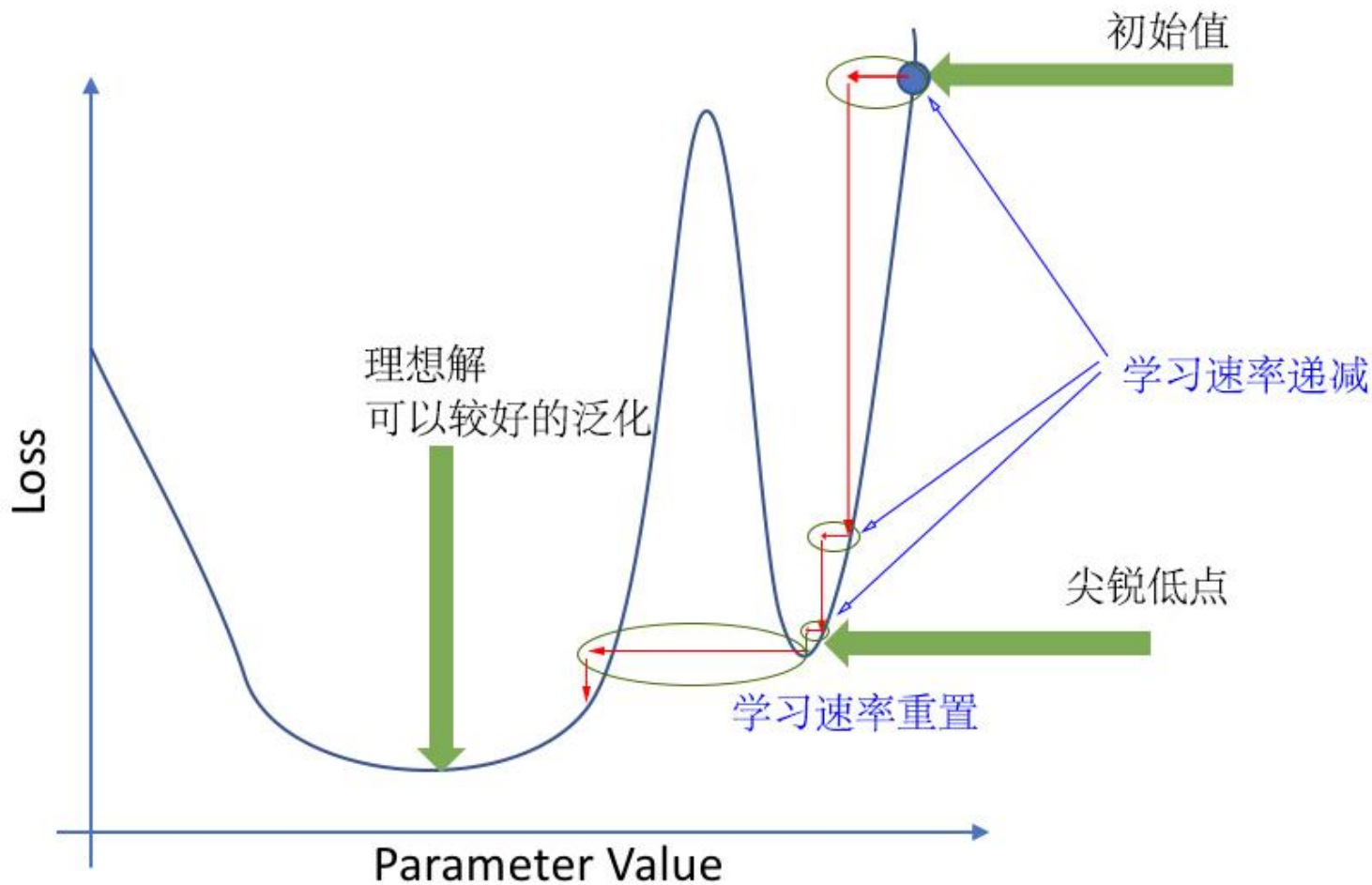


其中1e-2代表初始的学习率

cycle_len=1代表单次学习率下降的epoch数

3表示学习速率重置次数

4、设置cycle_len=1



[1]

4、使用数据增强再次训练最后一层若干轮。



在保证图像分类结果不变的情况下，对图像进行反转、缩放，从而实现数据扩增

```
tfms=tfms_from_model(resnet34,sz,aug_tfms=transforms  
_side_on,max_zoom=1.1)
```

其中transforms_side_on表明图片是从侧向拍摄，即是一般的摄影图像，区别于卫星图像，对这种图像会进行平移、旋转、翻转操作；max_zoom指定了对图像进行缩放的幅度，即最大缩放不超过1.1倍。

4、使用数据增强再次训练最后一层若干轮。



对应从上至下拍摄的例如卫星图像的参数为transforms_top_down

```
In [14]: def get_data(sz):  
    tfms = tfms_from_model(f_model, sz, aug_tfms=transforms_top_down, max_zoom=1.05)  
    return ImageClassifierData.from_csv(PATH, 'train-jpg', label_csv, tfms=tfms,  
                                       suffix='.jpg', val_idxs=val_idxs, test_name='test-jpg')  
  
    sz=256  
    tfms = tfms_from_model(f_model, sz, aug_tfms=transforms_top_down, max_zoom=1.1)  
    def get_augs():  
        data = ImageClassifierData.from_csv(PATH, 'train-jpg', label_csv, tfms=tfms,  
                                           suffix='.jpg', val_idxs=val_idxs, test_name='test-jpg', bs=2, num_workers=1)  
        x, _ = next(iter(data.aug_dl))  
        return data.trn_ds.denorm(x)[1]
```

```
In [15]: data = get_data(256)  
ims = np.stack(get_augs() for i in range(9))  
plots(ims, rows=3)
```



4、设置cycle_len=1，使用SGDR再次训练最后一层若干轮



tip——由于输入图片被改变，训练好的网络生成的结果会发生改变，无法再直接使用固定的activation，因此需要设置precompute=False。但是同时要注意，此处模型的参数还是固定的。

5、Unfreeze模型固定的参数



实验中使用unfreeze()函数，具体操作是将之前固定的网络参数重新纳入到训练的范围内，根据反向的梯度下降传播将之前固定的参数在原有的基础上进行训练修改，特别是卷积层。

tip——解冻的参数包括所有的层次，对具体的层次解冻并没有实际上的意义。

6、设置浅层的学习率为深层的十分之一到三分之一



由于不同层次间提取出的特征性质和方面不同，对网络大致分成浅层、中部、深层，其中浅层的学习率要较低。

学习率层次间比例的选择会根据不同图像的前端得到的特征与本身的差异，例如卫星图像的前后层次学习率比例选择三分之一，而猫狗分类使用十分之一。

7、再次使用lr_find()函数寻找学习率



方法和原理与第二步相同。

8、设置cycle_mult=2，训练整个网络直到收敛



cycle_mult=2代表每次学习速率重置时，遍历数据集的次数就翻倍，即epoch乘以2。

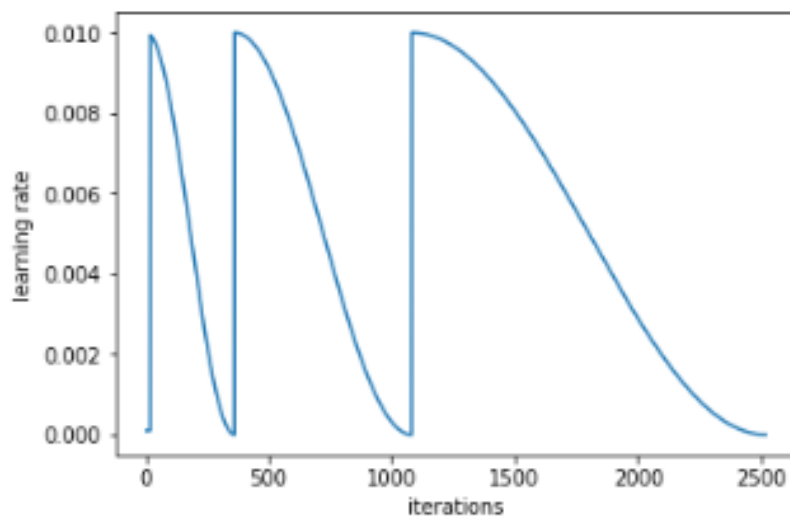
使用这种方式的原因在于初始点随机选择，需要在训练初通过较快的学习率变化得到好的泛化性；之后放缓学习率的降低速度，提升loss最小值的精度。

8、设置cycle_mult=2



实验中学习率的变化如图，三次学习率的重新设置周期为1,2,4个epoch。

```
In [64]: learn.sched.plot_lr()
```



8、训练整个网络直到过拟合情况的出现



需要注意的是实验表明Cycle_mult的使用对大结构的模型相对不敏感，而对小结构的模型可以加快fitting，因为大模型参数多，容易迅速度过underfitting的过程，达到overfitting。

8、训练整个网络直到过拟合情况的出现



测试集由保证运行方便的裁剪带来误差

测试的数据集的增强，即Test data augmentation (TTA)

与训练数据上的处理保证分类结果不同，对测试数据的检测结果为随机4组变换结果的平均值

```
log_preds,y = learn.TTA()
```

```
probs = np.mean(np.exp(log_preds),0)
```

其中log_preds即预测的概率值的对数，np.mean即计算平均值。



1

技巧介绍

2

实验测试



dogbreed数据集



来源于Kaggle比赛，对应120种狗的种类总共有将近20千张图片，该数据集的主要问题在于数据量本身的不足，对应每个种类不到200张图片。格式也相较lecture1中的猫狗分类不同，dogbreed数据集在不同文件夹内放不同标签对应的图像，然后通过csv格式作为索引，id为对应文件夹的名称代码，breed对应分类。

```
In [46]: label_df.head()           #列出csv文件中的一些格式
```

Out[46]:

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffa9c82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

dogbreed数据集



tip

该数据集没有测试集，因此需要从训练集中选取，实验中选择抽取20%作为 cross validation data set

```
val_idx = get_cv_idx(n)
```

n为总共的数据量。

第一步数据读入



```
tfms = tfms_from_model(arch, sz, aug_tfms=transforms_side_on,  
max_zoom=1.1)
```

```
data = ImageClassifierData.from_csv(PATH, 'train', f'{PATH}labels.csv',  
test_name='test', val_idxs=val_idxs, suffix='.jpg', tfms=tfms, bs=bs)
```

其中相较于猫狗不同的是，`from_csv()`指定了数据格式csv文件，'train'为训练数据集所在目录，'f'{PATH}labels.csv'为csv文件目录，`val_idxs`为验证数据的索引目录，`suffix`为图片文件扩展名。

第一步数据读入



tip——由于gpu显存会限制batchsize的大小，需要合理选择bs

理论上bs越大，训练效果会更好

```
In [12]: fn = PATH+data.trn_ds.fnames[0]; fn
```

```
Out[12]: 'data/dogbreed/train/001513dfcb2ffa82cccf4d8bbaba97.jpg'
```

```
In [13]: img = PIL.Image.open(fn); img
```

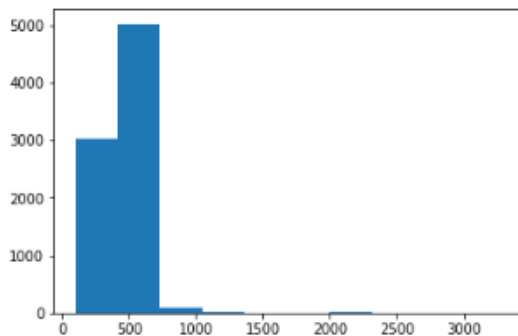
```
Out[13]:
```



第一步数据读入

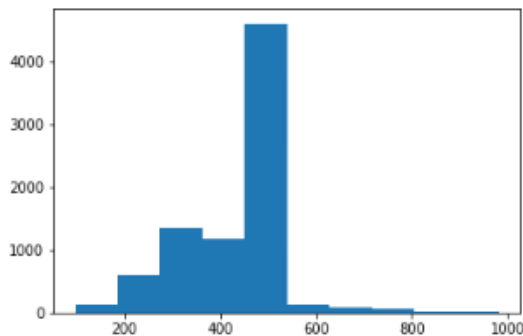


```
In [19]: plt.hist(row_sz):
```



```
In [20]: plt.hist(row_sz[row_sz<1000])
```

```
Out[20]: (array([ 135.,  592., 1347., 1164., 4599., 128.,  76.,  62.,  14.,  11.]),  
array([ 97., 185.5, 274., 362.5, 451., 539.5, 628., 716.5, 805., 893.5, 982. ]),  
<a list of 10 Patch objects>)
```



实验中可以通过
matplotlib得到图像的
histogram图，观察图
片的分布。

第三步使用precomputed网络训练最后一层

```
In [29]: learn = ConvLearner.pretrained(arch, data, precompute=True)
```

```
100% ██████████ 141/141 [01:00<00:00, 2.32it/s]
100% ██████████ 36/36 [00:14<00:00, 2.42it/s]
100% ██████████ 179/179 [01:15<00:00, 2.38it/s]
```

```
In [30]: learn.fit(1e-2, 5)
```

```
Epoch ██████████ 100% 5/5 [00:11<00:00, 2.38s/it]

epoch    trn_loss    val_loss    accuracy
0         0.898589    0.377881    0.904599
1         0.41594     0.300815    0.915362
2         0.29536     0.279074    0.916341
3         0.225841    0.263981    0.915851
4         0.190398    0.251431    0.920744
```

```
Out[30]: [array([0.25143]), 0.9207436481450635]
```

```
In [33]: learn = ConvLearner.pretrained(arch, data, precompute=True, ps=0.5)
```

```
In [34]: learn.fit(1e-2, 2)
```

```
Epoch ██████████ 100% 2/2 [00:05<00:00, 2.82s/it]

epoch    trn_loss    val_loss    accuracy
0         0.7136     0.354169    0.895793
1         0.397633    0.292147    0.909491
```

```
Out[34]: [array([0.29215]), 0.9094911933878397]
```

实验中采用resnext101网络

需要注意的是，对不同网络参数的读取需要先在fastai/weight/目录下存储对应的权重数据。

第四步设置precompute=False, cycle_len=1 训练



```
In [35]: learn.precompute=False
```

```
In [36]: learn.fit(1e-2, 5, cycle_len=1)
```

Epoch  100% 5/5 [05:50<00:00, 70.07s/it]

epoch	trn_loss	val_loss	accuracy
0	0.392563	0.254466	0.919276
1	0.383206	0.244328	0.924168
2	0.375068	0.243537	0.922211
3	0.329621	0.239132	0.917808
4	0.288349	0.231184	0.926614

```
Out[36]: [array([0.23118]), 0.9266144810590725]
```


可以发现训练时间变长，训练结果改善

第五六七八步合并



unfreeze，设置分层学习率，再次训练，设置 `cycle_mult=2`，采用TTA方式增强分类效果

```
In [22]: learn.fit(1e-2, 3, cycle_len=1, cycle_mult=2)
```

Epoch  100% 7/7 [21:12<00:00, 181.78s/it]

epoch	trn_loss	val_loss	accuracy
0	1.077906	0.531894	0.910959
1	0.557997	0.279189	0.919276
2	0.433319	0.261675	0.921722
3	0.390885	0.239877	0.924168
4	0.321396	0.229493	0.925147
5	0.279613	0.214058	0.931996
6	0.281415	0.215205	0.928571

```
Out[22]: [array([0.2152]), 0.9285714288047149]
```

```
In [25]: log_preds, y = learn.TTA()  
probs = np.mean(np.exp(log_preds), 0)  
  
accuracy_np(probs, y)
```


```
Out[25]: 0.9354207436399217
```

可以看到学习时间大大增长，结果有一定改善，TTA提升了测试的结果。

设置不同参数



```
In [28]: learn.fit(1e-2, 1, cycle_len=2)
```

Epoch  100% 2/2 [06:33<00:00, 196.91s/it]

epoch	trn_loss	val_loss	accuracy
0	0.297071	0.212636	0.929061
1	0.247344	0.212412	0.926614

```
Out[28]: [array([0.21241]), 0.9266144816422882]
```

```
In [31]: log_preds, y = learn.TTA()  
         probs = np.mean(np.exp(log_preds), 0)  
  
         accuracy_np(probs, y)
```

```
Out[31]: 0.9305283757338552
```

可以看到这次的学习率是单调下降，总共两个epoch。

额外技巧

由于数据集为图像，
可以从小尺寸的图像
开始训练，然后转到
大尺寸图像继续训练，
这是可以有效避免过
拟合的手段。

```
In [39]: learn.set_data(get_data(299, bs))  
learn.freeze()
```

100% 6/6 [00:00<00:00, 199.88it/s]

```
In [40]: learn.fit(1e-2, 3, cycle_len=1)
```

Epoch 100% 3/3 [06:16<00:00, 125.65s/it]

epoch	trn_loss	val_loss	accuracy
0	0.295386	0.220928	0.928082
1	0.259001	0.219994	0.928082
2	0.249648	0.218785	0.927593

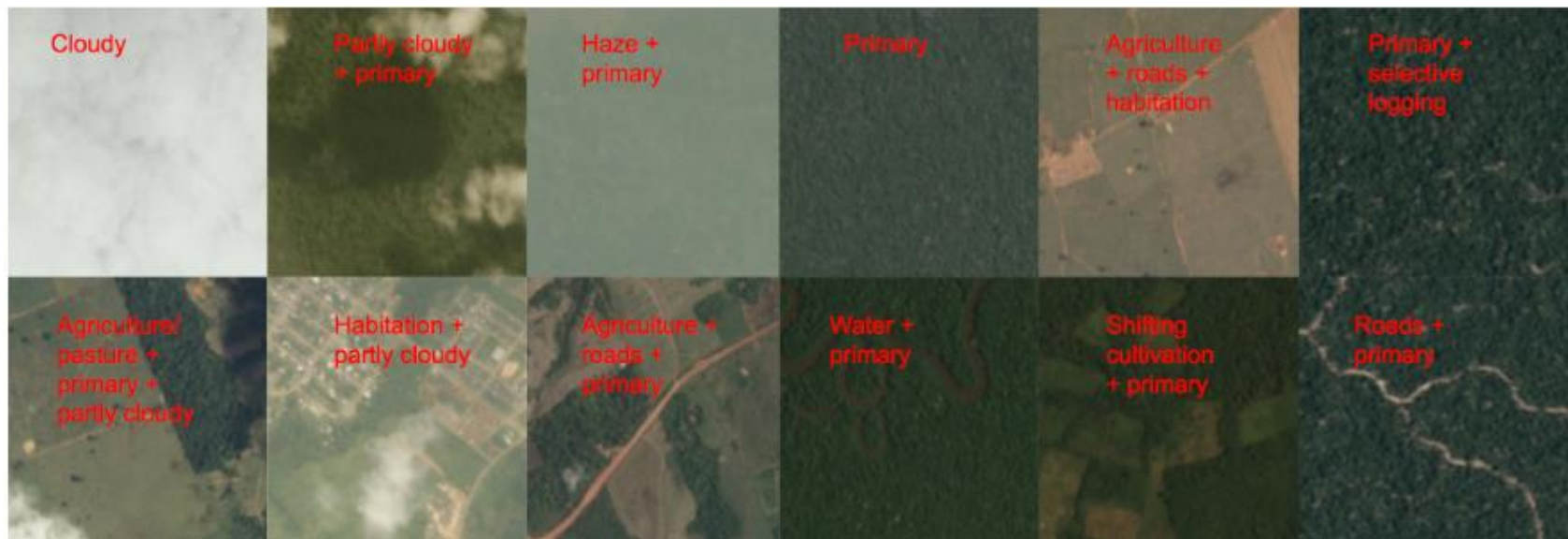
```
Out[40]: [array([0.21879]), 0.9275929552235015]
```

课程中得到的结果为0.1998， 0.941。

planet数据集



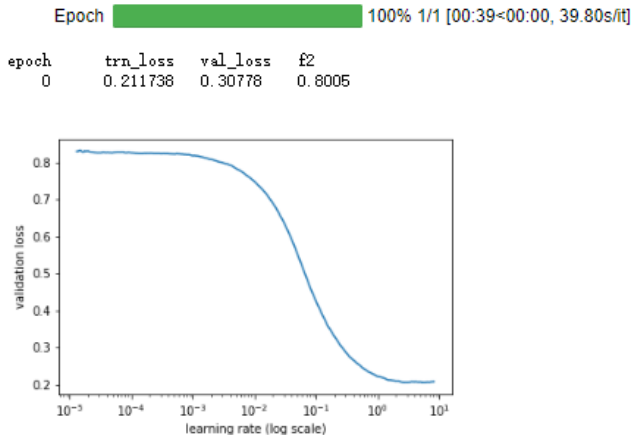
planet数据集是kaggle中描述亚马逊森林被砍伐情况的卫星图像数据集，包含一个train.csv的索引文件，和对应的[train/test]-jpg文件。该数据集为多标签的，即对于一张图片而言，可以涵盖多个标签。数据集中总共有40000多张训练图像，标签总体可以分为大气情况——晴朗、局部多云等；常见的土地覆盖和使用类型——雨林、农业等；罕见的土地覆盖和使用类型——砍伐并燃烧、选择性砍伐等。



planet数据集

```
In [21]: learn = ConvLearner.pretrained(f_model, data, metrics=metrics)
```

```
In [22]: lrf=learn.lr_find()  
learn.sched.plot()
```



```
In [23]: lr = 0.2
```

```
In [24]: learn.fit(lr, 3, cycle_len=1, cycle_mult=2)
```

Epoch 100% 7/7 [05:33<00:00, 47.64s/it]

epoch	trn_loss	val_loss	f2
0	0.145256	0.132826	0.883728
1	0.140274	0.126987	0.8908
2	0.135761	0.125613	0.892363
3	0.141125	0.124899	0.893977
4	0.138134	0.12247	0.895053
5	0.133957	0.121725	0.896007
6	0.130785	0.121326	0.896197

```
Out[24]: [array([0.12133]), 0.8961973269766308]
```

根据lr_find()的结果，设置学习率为0.2

planet数据集

进一步设置分层学习率
lrs=[lr/9,lr/3,lr]

```
In [26]: learn.unfreeze()
learn.fit(lrs, 3, cycle_len=1, cycle_mult=2)
```

Epoch 100% 7/7 [18:11<00:00, 155.86s/it]

epoch	trn_loss	val_loss	f2
0	0.118761	0.108237	0.909744
1	0.114574	0.104288	0.914397
2	0.106223	0.099939	0.916511
3	0.111104	0.102052	0.914098
4	0.106162	0.099148	0.917358
5	0.100136	0.096975	0.918994
6	0.099311	0.096039	0.919674

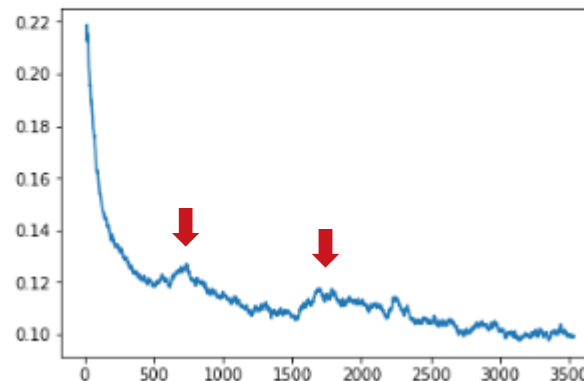
Out[26]: [array([0.09604]), 0.9196741531315286]

存储模型参数在
{PATH}/tmp/{arch}/models目录下。

```
In [27]: learn.save(f'{sz}')
```

```
In [28]: learn.sched.plot_loss()
```

loss和iteration曲线中可以看到中间存在两次loss的反向跳变，这里对应的就是学习率重置导致跳出尖锐低点的操作。



修改图像大小

对于卫星图而言，裁剪得到小图可以更好的表示特征和结果

```
In [30]: learn.set_data(get_data(sz))
learn.freeze()
learn.fit(lr, 3, cycle_len=1, cycle_mult=2)
```

Epoch 100% 7/7 [04:57<00:00, 42.45s/it]

epoch	trn_loss	val_loss	f2
0	0.101277	0.094537	0.919985
1	0.101668	0.092946	0.921148
2	0.097919	0.092019	0.922036
3	0.097525	0.091503	0.923523
4	0.096259	0.091389	0.922979
5	0.095818	0.091285	0.922441
6	0.093934	0.090915	0.923569

Epoch 100% 7/7 [14:06<00:00, 120.95s/it]

epoch	trn_loss	val_loss	f2
0	0.093407	0.086013	0.928978
1	0.094844	0.086977	0.928777
2	0.090303	0.085146	0.929617
3	0.091551	0.08635	0.928657
4	0.09023	0.085988	0.928524
5	0.08414	0.084284	0.929333
6	0.080356	0.083945	0.929854

Out[30]: [array([0.09091]), 0.92356872915424]

使用resnet50模型对猫狗再次分类

```
In [4]: tfms = tfms_from_model(arch, sz, aug_tfms=transforms_side_on, max_zoom=1.1)
data = ImageClassifierData.from_paths(PATH, tfms=tfms, bs=bs, num_workers=4)
learn = ConvLearner.pretrained(arch, data, precompute=True, ps=0.5)
```

```
100%|██████████████████| 822/822 [03:58<00:00, 3.45it/s]
100%|██████████████████| 72/72 [00:20<00:00, 3.47it/s]
```

```
In [5]: learn.fit(1e-2, 1)
learn.precompute=False
```

```
Epoch ██████████ 100% 1/1 [00:12<00:00, 12.09s/it]
```

epoch	trn_loss	val_loss	accuracy
0	0.040608	0.021008	0.993

```
In [6]: learn.fit(1e-2, 2, cycle_len=1)
```

```
Epoch ██████████ 100% 2/2 [08:47<00:00, 263.74s/it]
```

epoch	trn_loss	val_loss	accuracy
0	0.041105	0.015768	0.9955
1	0.036698	0.016612	0.996

```
Out[6]: [array([0.01661]), 0.99600000010728836]
```

使用resnet50模型对猫狗再次分类

```
In [5]: learn.fit(lr, 3, cycle_len=1)
```

Epoch  100% 3/3 [10:59<00:00, 219.96s/it]

epoch	trn_loss	val_loss	accuracy
0	0.062964	0.020269	0.9945
1	0.060163	0.018756	0.9935
2	0.05242	0.018989	0.9935

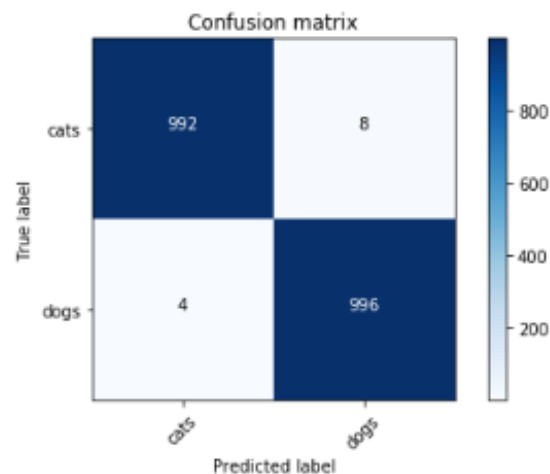
```
Out[5]: [array([0.01899]), 0.9935]
```

```
In [8]: log_preds, y = learn.TTA()  
probs = np.mean(np.exp(log_preds), 0)  
accuracy_np(probs, y)
```

```
Out[8]: 0.994
```

```
In [10]: plot_confusion_matrix(cm, data.classes)
```

```
[[992  8]  
 [  4 996]]
```



使用resnet50模型对猫狗再次分类

```
In [6]: learn.fit(1e-2, 2, cycle_len=1)
```

resnet50模型

Epoch  100% 2/2 [08:47<00:00, 263.74s/it]

epoch	trn_loss	val_loss	accuracy
0	0.041105	0.015768	0.9955
1	0.036698	0.016612	0.996

```
Out[6]: [array([0.01661]), 0.9960000010728836]
```

```
learn.fit(1e-2, 3, cycle_len=1)
```

resnet34模型

HBox(children=(IntProgress(value=0, description='Epoch', max=3), HTML(value='')))

epoch	trn_loss	val_loss	accuracy
0	0.05121	0.024065	0.992
1	0.04339	0.023791	0.99
2	0.047053	0.022904	0.991

实验问题

- 1、由于没有使用paperspace，会有许多需要预先安装的部分找不到，例如数据集、文件夹、网络的weights不存在，解决方法可以通过网上下载过来放在对应的文件夹即可
- 2、代码中存在的错误，由于版本的更新，语法存在的错误，往往会出现问题，例如TTA部分的代码

代码解决

```
In [33]: logs_preds, y = learn.TTA()  
         probs = np.exp(logs_preds)  
         accuracy(logs_preds, y), metrics.log_loss(y, probs)
```

```
TypeError                                Traceback (most recent call last)  
<ipython-input-33-1f68f5c22374> in <module>()  
      1 logs_preds, y = learn.TTA()  
      2 probs = np.exp(logs_preds)  
----> 3 accuracy(logs_preds, y), metrics.log_loss(y, probs)  
  
~/fastai/courses/dl1/fastai/metrics.py in accuracy(preds, targs)  
      7  
      8 def accuracy(preds, targs):  
----> 9     preds = torch.max(preds, dim=1)[1]  
     10     return (preds==targs).float().mean()  
     11  
  
TypeError: torch.max received an invalid combination of arguments - got (numpy.ndarray, dim=int), but expected one of:  
 * (torch.FloatTensor source)  
 * (torch.FloatTensor source, torch.FloatTensor other)  
   didn't match because some of the keywords were incorrect: dim  
 * (torch.FloatTensor source, int dim)  
 * (torch.FloatTensor source, int dim, bool keepdim)
```

```
In [25]: log_preds, y = learn.TTA()  
         probs = np.mean(np.exp(log_preds), 0)  
  
         accuracy_np(probs, y)
```

修改一下语法即可

谢谢！

【1】 <https://blog.csdn.net/suredied/article/details/80822678>



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

