fastai lecture 8

2018年7月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

目录 Contents

1、前情回顾

已有的学习内容：

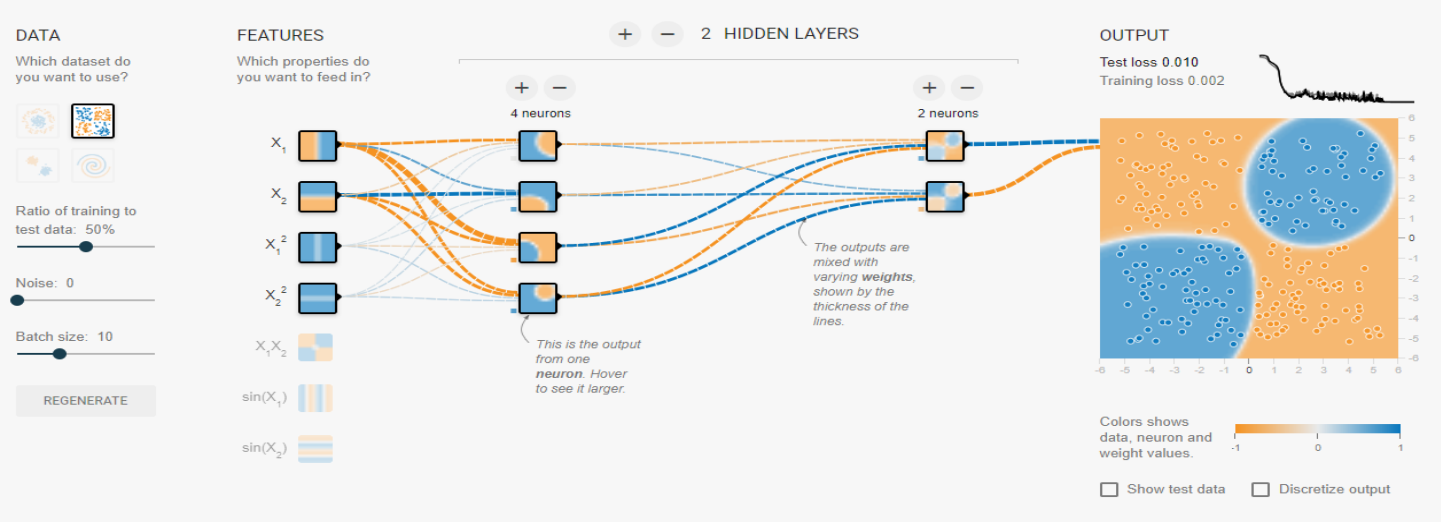
1.1 可微分的编程

Yann LeCun一直在推行一种观念，即常说的“深度学习”其实可以被认为是“可微分的编程”，我们在part1学习中所做的工作其实是创建一个分类函数以及一个损失函数（用来描述相关参数是否足够好）。

按照课程中的说法：“可微分编程不只是对现代深度学习技术的重塑，就像深度学习是对具有两层以上神经网络的重塑一样，它更重要的意义是，人们现在正在通过组装参数化功能块的网络来构建一种“新型软件”，并通过使用某种形式的基于梯度的优化来训练它们。”

下面的链接是不同神经网络结构与效果的动画显示。

https://playground.tensorflow.org



1.2 迁移学习

删除最后一层并添加多个随机层（全连接层）来替代最后一层，利用原始网络的同时，微调最后的几层以执行我们所需要的目的。这样做可以使用更少的数据（这种减少是数量级上的），却得到更准确的结果同时训练速度更快。

Transfer learning makes nearly everything easier, faster, and more accurate

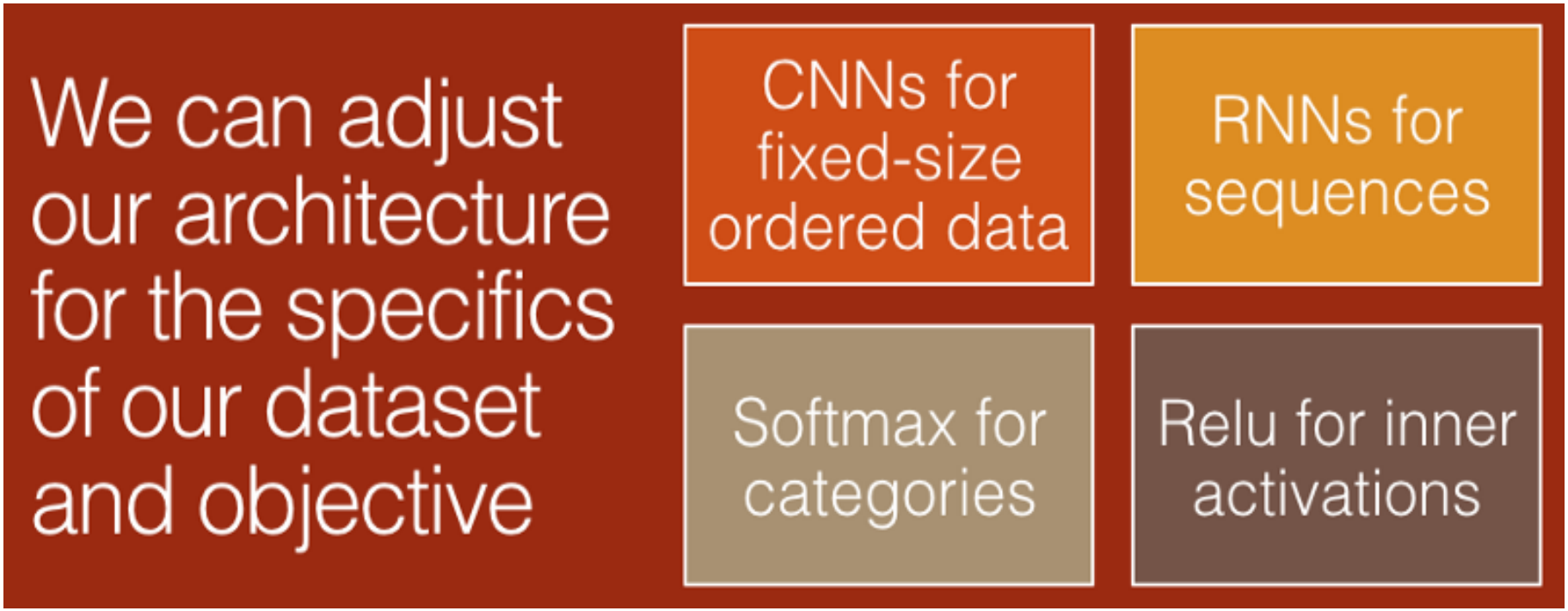
1.3 结构设计

在深度学习领域，有一些结构往往在很多时候工作效果不错：

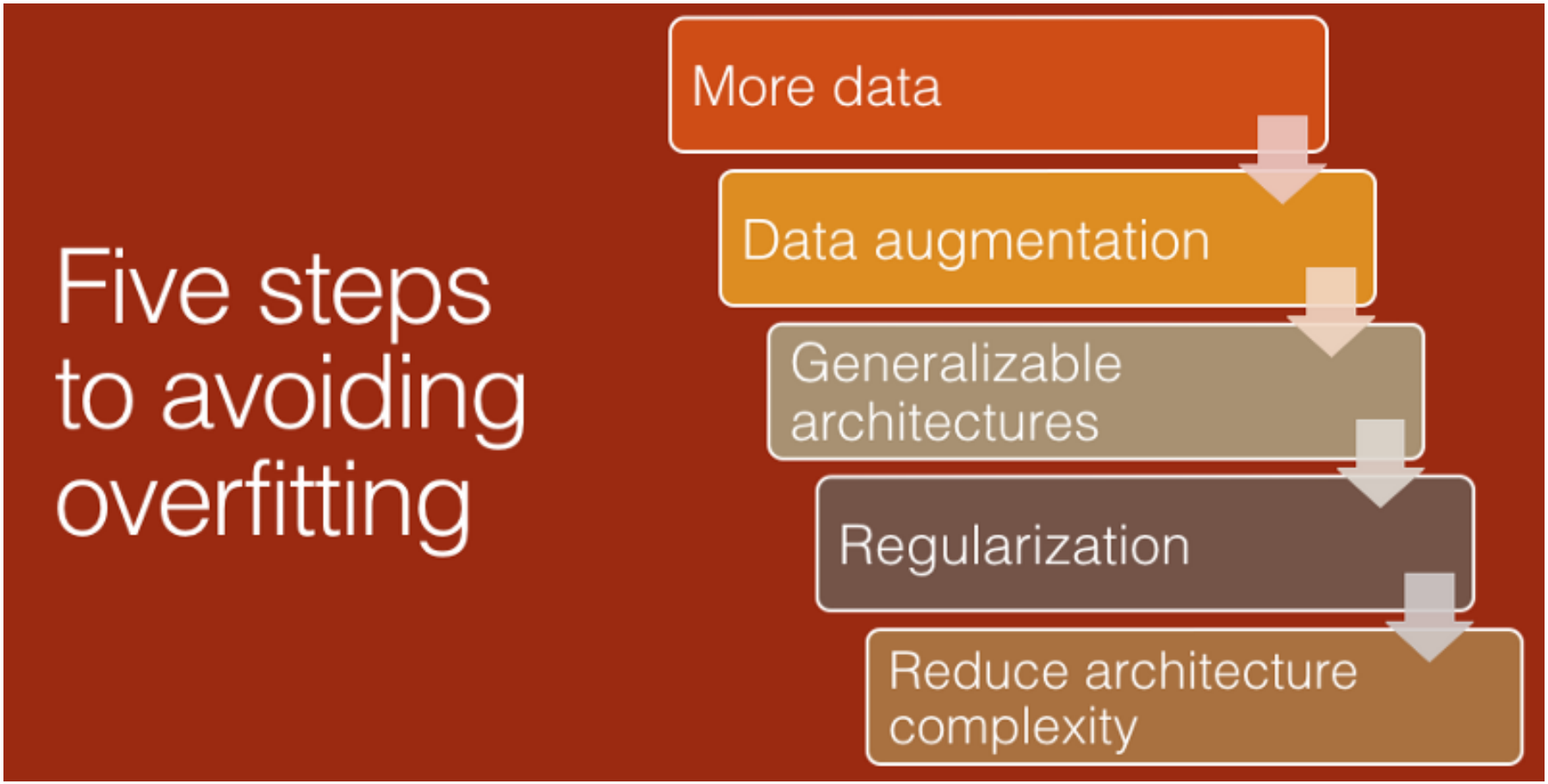
用CNN解决固定大小的有序数据的问题、RNN用于解决具有其他特别状态的序列；

一定的选择激活函数的经验，如果是单一的分类结果，多用softmax；如果您有多个分类结果，则用sigmoid；

……



1.4 避免过拟合的方法


Five steps
to avoiding
overfitting
More data
Data augmentation
Generalizable
architectures
Regularization
Reduce architecture
complexity

1.5 嵌入层（Embeddings）

嵌入层的使用让神经网络处理数据的类型更加广泛。

Embeddings allow us to use categorical data too

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1									*original data*				user embedding						movie embedding				
2		Users	Embeddings						userId	movieId	rating	user idx	*1*	*2*	*3*	*4*	*5*	movie idx	*1*	*2*	*3*	*4*	*5*
3	idx	Row Lab -							*14*	*417*	2	1	0.19	0.63	0.31	0.44	0.51	14	0.75	0.47	0.05	0.91	0.59
4	1	14	0	0.63	0.31	0.44	0.51		*29*	*417*	4	2	0.25	0.83	0.71	0.96	0.59	14	0.75	0.47	0.05	0.91	0.59
5	2	29	0.25	0.83	0.71	0.96	0.59		*72*	*417*	5	3	0.30	0.44	0.19	0.00	0.72	14	0.75	0.47	0.05	0.91	0.59
6	3	72	0.30	0.44	0.19	0.00	0.72		*211*	*417*	3	4	0.02	0.72	0.69	0.35	0.25	14	0.75	0.47	0.05	0.91	0.59
7	4	211	0.02	0.72	0.69	0.35	0.25		*212*	*417*	3	5	0.60	0.87	0.76	0.30	0.04	14	0.75	0.47	0.05	0.91	0.59
8	5	212	0.60	0.87	0.76	0.30	0.04		*293*	*417*	4	6	0.73	0.70	0.44	0.47	0.29	14	0.75	0.47	0.05	0.91	0.59
9	6	293	0.73	0.70	0.44	0.47	0.29		*310*	*417*	3	7	0.23	0.81	0.36	0.47	0.12	14	0.75	0.47	0.05	0.91	0.59
10	7	310	0.23	0.81	0.36	0.47	0.12		*379*	*417*	4	8	0.68	0.90	0.20	0.92	0.74	14	0.75	0.47	0.05	0.91	0.59
11	8	379	0.68	0.90	0.20	0.92	0.74		*451*	*417*	3.5	9	0.81	0.41	0.81	0.15	0.17	14	0.75	0.47	0.05	0.91	0.59
12	9	451	0.81	0.41	0.81	0.15	0.17		*467*	*417*	4	10	0.70	0.61	0.90	0.89	0.24	14	0.75	0.47	0.05	0.91	0.59
13	10	467	0.70	0.61	0.90	0.89	0.24		*508*	*417*	3	11	0.50	0.27	0.73	0.44	0.83	14	0.75	0.47	0.05	0.91	0.59
14	11	508	0.50	0.27	0.73	0.44	0.83		*546*	*417*	3.5	12	0.16	0.21	0.75	0.48	0.98	14	0.75	0.47	0.05	0.91	0.59
15	12	546	0.16	0.21	0.75	0.48	0.98		*563*	*417*	4	13	0.91	0.75	0.75	0.24	0.06	14	0.75	0.47	0.05	0.91	0.59
16	13	563	0.91	0.75	0.75	0.24	0.06		*579*	*417*	4	14	0.55	0.58	0.68	0.93	0.66	14	0.75	0.47	0.05	0.91	0.59
17	14	579	0.55	0.58	0.68	0.93	0.66		*623*	*417*	5	15	0.94	0.25	0.46	0.16	0.30	14	0.75	0.47	0.05	0.91	0.59
18	15	623	0.94	0.25	0.46	0.16	0.30		*14*	*27*	3	1	0.19	0.63	0.31	0.44	0.51	1	0.71	0.81	0.74	0.04	0.04
19									*29*	*27*	5	2	0.25	0.83	0.71	0.96	0.59	1	0.71	0.81	0.74	0.04	0.04

1.6 Part2内容概述

What we'll study in Part 2

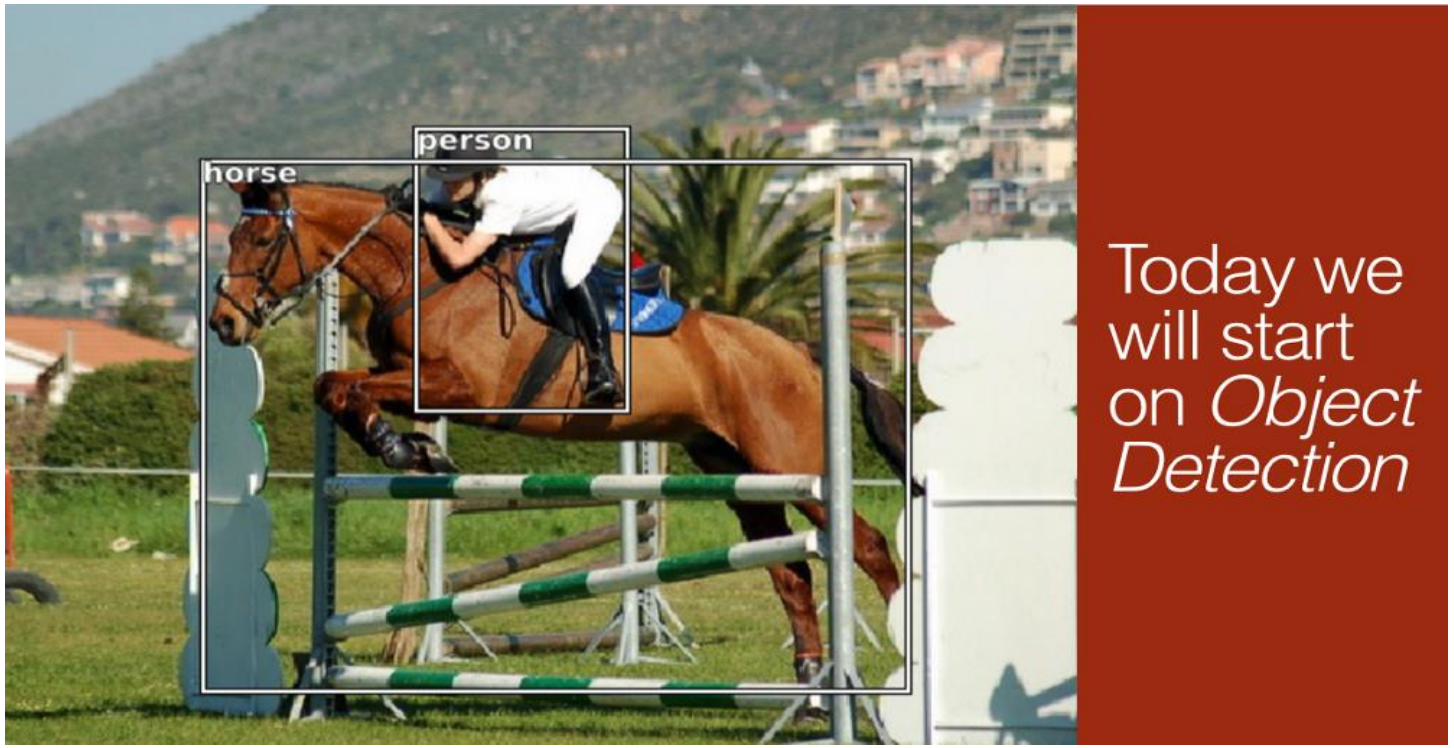
Generative Models

CNNs beyond classification	NLP beyond classification	Large data sets

• Localization
• Enhancement:
  • Colorization
  • Super-resolution
  • Artistic style
• GANs
	• Translation
	• Seq2seq
	• Attention
	• Large vocabularies
		• Large images
		• Lots of data points
		• Large outputs

1.7 本节实验目的

1、 检测出图片中最大的物体；
2、 识别目标物体、为其画上边框；
3、 同时完成以上两步。



目录 Contents

2.1 生成器(generator)简介

生成器适用的情形：

1、当我们需要一个公用的，按需生成的数据时；

2、某个进程先执行一部分，某个事件发生后再执行下一部分，需要实现异步时。

在python的函数定义中，只要出现了yield表达式（Yield expression），那么事实上定义的是一个生成器函数（ generator function ）， 调用这个生成器函数的返回值会是一个生成器，这点跟普通的函数调用有所区别。

2.1 生成器(generator)简介

```
def gen_generator():
    yield 1

def gen_value():
    return 1

if __name__ == '__main__':
    ret = gen_generator()
    print ret, type(ret)    #<generator object gen_generator at 0x02645648> <type 'generator'>
    ret = gen_value()
    print ret, type(ret)    # 1 <type 'int'>
```

从上面的代码可以看出，gen_generator函数返回的是一个生成器实例，此生成器有以下特点：

1、遵循迭代器（iterator）协议，迭代器协议需要实现__iter__、next等接口；

2、能够多次进入、多次返回，能够暂停函数体中代码的执行。

2.1 生成器(generator)简介

```
>>> gen = gen_example()
>>> gen.next()        # 第一次调用next
before any yield
'first yield'
>>> gen.next()        # 第二次调用next
between yields
'second yield'
>>> gen.next()        # 第三次调用next
no yield anymore
```

```
>>> def gen_example():
...     print 'before any yield'
...     yield 'first yield'
...     print 'between yields'
...     yield 'second yield'
...     print 'no yield anymore'
...
```

2.1 生成器(generator)简介

需要注意的是右图示例中的generator被转化成一个list表示，，且因为for语句能自动捕获StopIteration异常，所以generator（本质上是任何iterator）较为常用的方法是在循环中使用

```
1 def generator_example():
2     yield 1
3     yield 2
4
5 if __name__ == '__main__':
6     for e in generator_example():
7         print e
8         # output 1 2
```

生成器函数（generator function）产生生成器（generator）与普通函数的区别：

1、普通函数每次都是从第一行开始运行，而生成器函数从上一次yield开始的地方运行。

2、普通函数调用一次返回一个（一组）值，而生成器函数可以多次返回。

3、普通函数可以被无数次重复调用，而一个生成器函数实例在yield最后一个值或者return之后就不能继续调用了。

2.2 下载Pascal VOC 数据集

Paperspace中的主机上并没有提前下载pascal数据集，需要自行下载。
将Paperspace中的主机当做没有图形界面的Ubuntu即可，运用以下语句便可实现数据集下载并解压到特定位置。

```
cd ~/fastai/courses/dl2
ln -s ~/data data && cd $_
mkdir pascal && cd $_
curl -OL http://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
curl -OL https://storage.googleapis.com/coco-dataset/external/PASCAL_VOC.zip
tar -xf VOCtrainval_06-Nov-2007.tar
unzip PASCAL_VOC.zip
mv PASCAL_VOC/*.json .
rmdir PASCAL_VOC
```

2.2 下载Pascal VOC 数据集

实际操作过程中可能因为本地网络问题而界面卡住，此时直接刷新即可，这并不会损失进度，只是之前的输出就无法查看了。

但是实际运行中可能仍会存在误差，需要手动找到解压位置然后调整PATH（借鉴我们在第一课猫和狗的分类中所做）。

```
(fastai) paperspace@ps77xhsgq:~/fastai/courses/dl2/data$ tar -xf VOCtrainval_06-Nov-2007.tar
(fastai) paperspace@ps77xhsgq:~/fastai/courses/dl2/data$ unzip PASCAL_VOC.zip
Archive:  PASCAL_VOC.zip
   creating: PASCAL_VOC/
  inflating: PASCAL_VOC/pascal_test2007.json
  inflating: PASCAL_VOC/pascal_train2007.json
  inflating: PASCAL_VOC/pascal_train2012.json
  inflating: PASCAL_VOC/pascal_val2007.json
  inflating: PASCAL_VOC/pascal_val2012.json
(fastai) paperspace@ps77xhsgq:~/fastai/courses/dl2/data$ mv PASCAL_VOC/*.json .
(fastai) paperspace@ps77xhsgq:~/fastai/courses/dl2/data$  rmdir PASCAL_VOC
(fastai) paperspace@ps77xhsgq:~/fastai/courses/dl2/data$ rmdir PASCAL_VOC
rmdir: failed to remove 'PASCAL_VOC': No such file or directory
(fastai) paperspace@ps77xhsgq:~/fastai/courses/dl2/data$
```

2.3 数据集的查看与调整

本次实验中采用pathlib的方法查看数据集，具体代码如下：

```
In [4]: trn_j = json.load((PATH/'pascal_train2007.json').open())
        trn_j.keys()

Out[4]: dict_keys(['images', 'type', 'annotations', 'categories'])

In [5]: IMAGES,ANNOTATIONS,CATEGORIES = ['images', 'annotations', 'categories']
        trn_j[IMAGES][:5]

Out[5]: [{'file_name': '000012.jpg', 'height': 333, 'id': 12, 'width': 500},
         {'file_name': '000017.jpg', 'height': 364, 'id': 17, 'width': 480},
         {'file_name': '000023.jpg', 'height': 500, 'id': 23, 'width': 334},
         {'file_name': '000026.jpg', 'height': 333, 'id': 26, 'width': 500},
         {'file_name': '000032.jpg', 'height': 281, 'id': 32, 'width': 500}]
```

Out[5]输出几个标签的意义分别是：

文件名：与文件名相关的图像　　高度：图像的高度的大小

id：用于连接到其他数据集的图像ID（每个图像独一无二）

宽度：图像的宽度的大小。

2.3 数据集的查看与调整

因为实验中我们需要具体查看图片中待检测的物体及其标签，所以还特别需要查看以下几条：

bbox：图片中物体的方框（四个数字分别是column，row，height，width）、Category：此物体的种类、id：从属于哪张图片（每张图片都是特定id）。

```
trn_j[ANNOTATIONS][:2]
```

```
[{'area': 34104,
  'bbox': [155, 96, 196, 174],
  'category_id': 7,
  'id': 1,
  'ignore': 0,
  'image_id': 12,
  'iscrowd': 0,
  'segmentation': [[155, 96, 155, 270, 351, 270, 351, 96]]},
 {'area': 13110,
  'bbox': [184, 61, 95, 138],
  'category_id': 15,
  'id': 2,
  'ignore': 0,
  'image_id': 17,
  'iscrowd': 0,
  'segmentation': [[184, 61, 184, 199, 279, 199, 279, 61]]}]
```

```
trn_anno[17]
```

```
[(array([61, 184, 198, 278]), 15), (array([77, 89, 335, 402]), 13)]
```

```
cats[15],cats[13]
```

```
('person', 'horse')
```

例如此处，id为17的图片中有两个物体，他们的类别分别为15（人类）和13（马）

2.3 数据集的查看与调整

将所需数据先从字典中提取出来：

```
FILE_NAME,ID,IMG_ID,CAT_ID,BBOX = 'file_name','id','image_id','category_id','bbox'

cats = dict((o[ID], o['name']) for o in trn_j[CATEGORIES])
trn_fns = dict((o[ID], o[FILE_NAME]) for o in trn_j[IMAGES])
trn_ids = [o[ID] for o in trn_j[IMAGES]]
```

创建一个新的字典，从物体种类、物体方框映射到图片id，并将原来表示bbox的column、row、height、width转换成用左上到右下的numpy数组来表示。

```
def hw_bb(bb): return np.array([bb[1], bb[0], bb[3]+bb[1]-1, bb[2]+bb[0]-1])

trn_anno = collections.defaultdict(lambda:[])
for o in trn_j[ANNOTATIONS]:
    if not o['ignore']:
        bb = o[BBOX]
        bb = hw_bb(bb)
        trn_anno[o[IMG_ID]].append((bb,o[CAT_ID]))

len(trn_anno)
```

2.3 数据集的查看与调整

可以使用类似im = open_image（IMG_PATH/ im0_d[FILE_NAME]）的语句来查看图片；

Open_image函数的结构其实比较复杂，我们可以使用VS code 或者在pychrom 的IDE下打开fastai库，然后查看它的源码，具体如下：

```
def open_image(fn):
    """ Opens an image using OpenCV given the file path.

    Arguments:
        fn: the file path of the image

    Returns:
        The numpy array representation of the image in the RGB format
    """
    flags = cv2.IMREAD_UNCHANGED+cv2.IMREAD_ANYDEPTH+cv2.IMREAD_ANYCOLOR
    if not os.path.exists(fn):
        raise OSError('No such file or directory: {}'.format(fn))
    elif os.path.isdir(fn):
        raise OSError('Is a directory: {}'.format(fn))
    else:
        try:
            im = cv2.imread(str(fn), flags).astype(np.float32)/255
            if im is None: raise OSError(f'File not recognized by opencv: {fn}')
            return cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
        except Exception as e:
            raise OSError('Error handling image at: {}'.format(fn)) from e
```

2.4 目标检测方法

学会使用Matplotlib的API，举例如下：

```
def show_img(im, figsize=None, ax=None):
    if not ax: fig,ax = plt.subplots(figsize=figsize)
    ax.imshow(im)
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    return ax
```

此处plt.subplot函数在创建图层（括号内的参数为图片大小），内部的False即取消默认的x，y轴坐标。

```
def draw_outline(o, lw):
    o.set_path_effects([patheffects.Stroke(
        linewidth=lw, foreground='black'), patheffects.Normal()])
```

此处draw_outline函数为设定所画方框的颜色，采用白底黑边或者黑底白边即可在所有底色中都能看见我们的描边。

2.4 目标检测方法

接着画出检测物体的边框并标出类型。

```
def draw_rect(ax, b):
    patch = ax.add_patch(patches.Rectangle(b[:2], *b[-2:], fill=False, edgecolor='white', lw=2))
    draw_outline(patch, 4)
```

```
def draw_text(ax, xy, txt, sz=14):
    text = ax.text(*xy, txt,
        verticalalignment='top', color='white', fontsize=sz, weight='bold')
    draw_outline(text, 1)
```

```
ax = show_img(im)
b = bb_hw(im0_a[0])
draw_rect(ax, b)
draw_text(ax, b[:2], cats[im0_a[1]])
```

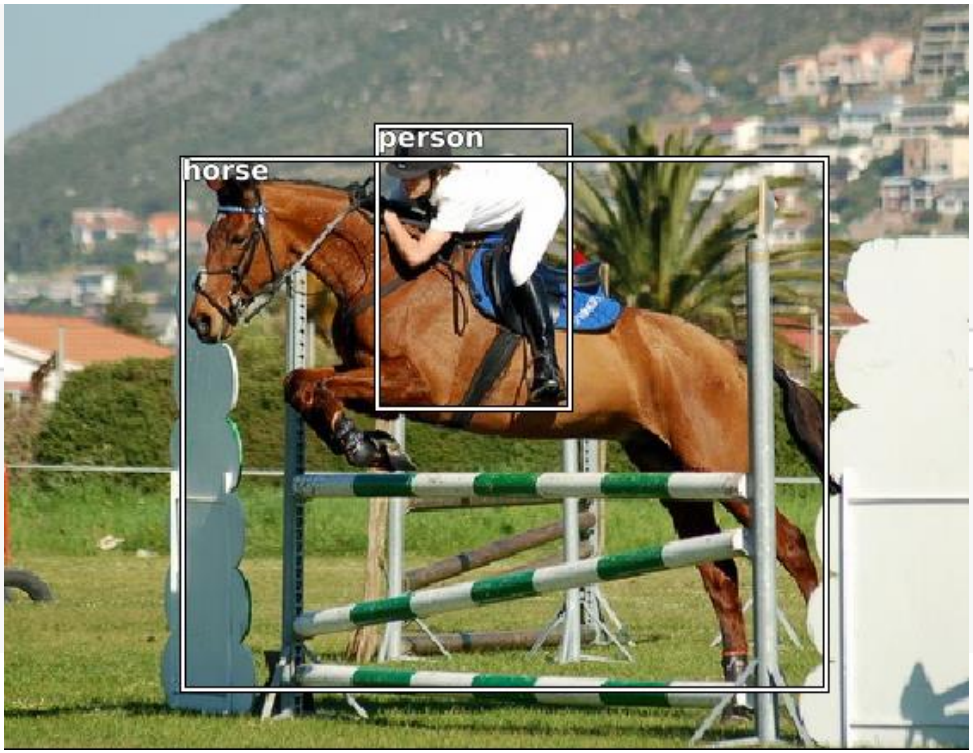
综上，基本流程是先读取图片、标签内容；然后根据已有数据画出边框并标上代表所框物体类型的标签。

2.4 目标检测方法

将目前已有的工作打包，可以得到一个画图工具。

```
def draw_im(im, ann):
    ax = show_img(im, figsize=(16,8))
    for b,c in ann:
        b = bb_hw(b)
        draw_rect(ax, b)
        draw_text(ax, b[:2], cats[c], sz=16)
```

```
def draw_idx(i):
    im_a = trn_anno[i]
    im = open_image(IMG_PATH/trn_fns[i])
    print(im.shape)
    draw_im(im, im_a)
```



如上图，i即为图片的id，

当我们运行draw_idx（17）即可得到右上的图片。

目录 Contents

3.1 最大目标识别

遍历图片中所有的边框，以边框左上减右下再相乘的面积作为方框的面积，然后排序找出最大值，即认为对应物体是最大的；下面的函数使得我们得到一个字典——从图片id映射到该图片中最大的物体的边框，具体代码如下图：

```
def get_lrg(b):
    if not b: raise Exception()
    b = sorted(b, key=lambda x: np.product(x[0][-2:]-x[0][:2]), reverse=True)
    return b[0]
```

```
trn_lrg_anno = {a: get_lrg(b) for a,b in trn_anno.items()}
```

```
b,c = trn_lrg_anno[23]
b = bb_hw(b)
ax = show_img(open_image(IMG_PATH/trn_fns[23]), figsize=(5,10))
draw_rect(ax, b)
draw_text(ax, b[:2], cats[c], sz=16)
```

此处返回的b、c分别代表id为23的图片中最大物体的边框和其类别代号。

3.2 数据再处理

直接读取csv文件是读取数据集时较简单的操作，如果没有csv数据集，则应将数据集整理成csv形式以便操作，而不要自定义其他格式。

```
(PATH/'tmp').mkdir(exist_ok=True)
CSV = PATH/'tmp/lrg.csv'
```

我们通常使用panda自带的Dataframe函数来创建一个字典，如下：

```
df = pd.DataFrame({'fn': [trn_fns[o] for o in trn_ids],
    'cat': [cats[trn_lrg_anno[o][1]] for o in trn_ids]}, columns=['fn','cat'])
df.to_csv(CSV, index=False)
```

其中值得注意的是，字典的key其实是没有顺序的，所以columns=[‘fn’,’cat’]中的顺序就很重要了。
最后将字典输出为一个csv文件以便读取调用。

3.2 数据再处理

需要注意的是最后的Crop_type=CropType.NO 此处为不剪裁图片，而是压缩。如下：

```
f_model = resnet34
sz=224
bs=64
```

```
tfms = tfms_from_model(f_model, sz, aug_tfms=transforms_side_on, crop_type=CropType.NO)
md = ImageClassifierData.from_csv(PATH, JPEGS, CSV, tfms=tfms, bs=bs)
```

```
x,y=next(iter(md.val_dl))
```

```
show_img(md.val_ds.denorm(to_np(x))[0]);
```

此处即用到了刚刚所说的generator的方法，定义了一个迭代对象，然后执行next的时候只返回一个batch（默认一般大小为64），此处使用迭代器（iter）是为了方便从数据集的开头开始取mini_batch。
右图即为输出压缩图片展示。

3.2 数据再处理

我们单独地看看一个batch内的数据：此时我们会发现数组无法直接用open_imag打开，这是因为：

1. 数组不是标准的Numpy数组。
2. 数组不是CPU的运算数组。
3. 分布的形状是错误的，我们需要3*224*224，而此处为2*3*224*224。
4. 数组不是标准的从0到1的分布。

此时我们使用tfms_from_stats函数来变换这个数组，使之能被open_image直接打开。

```
x[:2]
```

```
( 1 , 0 ,.,.) =
  2.1343e+00  2.1323e+00  2.1205e+00  ...  -2.0870e+00 -2.0843e+00 -2.0837e+00
  2.1703e+00  2.1498e+00  2.1186e+00  ...  -2.0939e+00 -2.0863e+00 -2.0861e+00
  2.1454e+00  2.1375e+00  2.0940e+00  ...  -2.0922e+00 -2.0923e+00 -2.0933e+00
                 ...                 ⋱                 ...
 -1.8905e+00 -1.8818e+00 -1.8374e+00  ...  -1.8581e+00 -1.8778e+00 -1.8109e+00
 -1.8530e+00 -1.8759e+00 -1.8143e+00  ...  -1.8267e+00 -1.8599e+00 -1.8636e+00
 -1.8764e+00 -1.8944e+00 -1.9561e+00  ...  -1.8202e+00 -1.7856e+00 -1.8230e+00

( 1 , 1 ,.,.) =
  2.2305e+00  2.1392e+00  1.7919e+00  ...  -2.0042e+00 -2.0013e+00 -2.0007e+00
  2.2569e+00  2.1649e+00  1.8062e+00  ...  -2.0112e+00 -2.0034e+00 -2.0032e+00
  2.2407e+00  2.1826e+00  1.8103e+00  ...  -2.0095e+00 -2.0096e+00 -2.0105e+00
                 ...                 ⋱                 ...
 -1.8032e+00 -1.7944e+00 -1.7489e+00  ...  -1.7557e+00 -1.7819e+00 -1.7218e+00
 -1.7649e+00 -1.7883e+00 -1.7254e+00  ...  -1.7107e+00 -1.7613e+00 -1.7757e+00
 -1.7888e+00 -1.8072e+00 -1.8703e+00  ...  -1.7174e+00 -1.6914e+00 -1.7343e+00

( 1 , 2 ,.,.) =
  1.8575e+00  1.3323e+00  6.5693e-01  ...  -1.7730e+00 -1.7702e+00 -1.7696e+00
  1.9155e+00  1.3577e+00  6.7138e-01  ...  -1.7800e+00 -1.7723e+00 -1.7721e+00
  1.8913e+00  1.3744e+00  6.8420e-01  ...  -1.7783e+00 -1.7784e+00 -1.7794e+00
                 ...                 ⋱                 ...
 -1.5730e+00 -1.5642e+00 -1.5189e+00  ...  -1.5329e+00 -1.5559e+00 -1.4919e+00
 -1.5348e+00 -1.5581e+00 -1.4955e+00  ...  -1.4945e+00 -1.5366e+00 -1.5456e+00
 -1.5586e+00 -1.5769e+00 -1.6397e+00  ...  -1.4945e+00 -1.4639e+00 -1.5043e+00
[torch.cuda.FloatTensor of size 2x3x224x224 (GPU 0)]
```

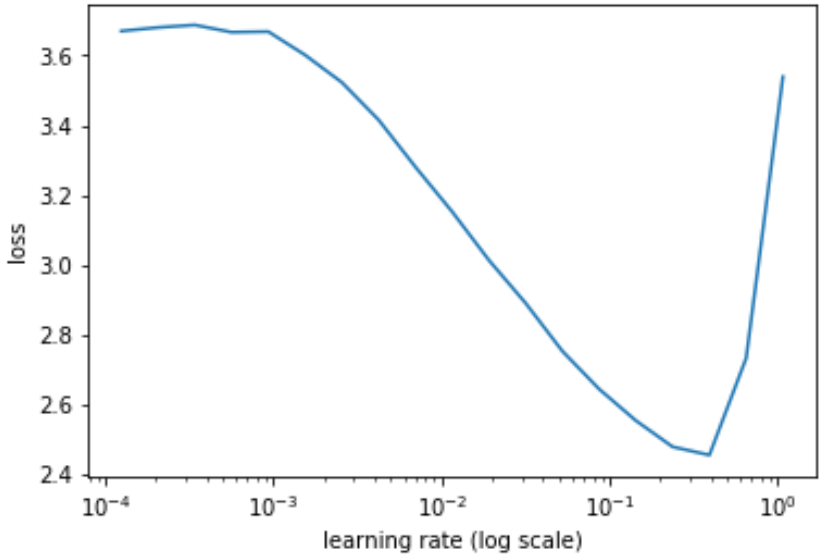
3.3 训练过程

```
learn = ConvLearner.pretrained(f_model, md, metrics=[accuracy])
learn.opt_fn = optim.Adam
```

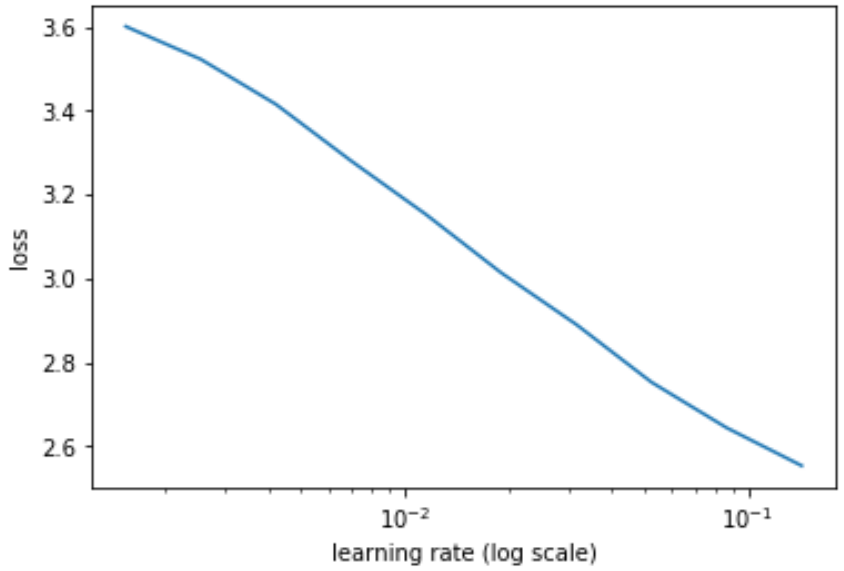
```
lrf=learn.lr_find(1e-5,100)
```

此处在寻找最佳学习率

```
learn.sched.plot(n_skip=5, n_skip_end=1)
```




```
learn.sched.plot()
```



3.3 训练过程

初步进行训练，会发现正确率并不高，这是因为大部分图片中都具有多个物体，并非第一课猫与狗分类中单个物体识别的简单情况

```
lr = 2e-2
```

```
learn.fit(lr, 1, cycle_len=1)
```

```
A Jupyter Widget
epoch      trn_loss   val_loss   accuracy
    0      1.335532   0.6443     0.804838
```

设定多个学习率，解冻前两层，使网络结构更加复杂，再进行训练得到结果。可以发现正确率有了明显提高，但是还是不能让人满意。

```
lrs = np.array([lr/1000,lr/100,lr])
```

```
learn.freeze_to(-2)
```

```
learn.fit(lrs/5, 1, cycle_len=1)
```

```
A Jupyter Widget
epoch      trn_loss   val_loss   accuracy
    0      0.780925   0.575539   0.821064

[0.57553864, 0.82106370478868484]
```

3.3 训练过程

解冻全部层次之后，我们再次训练，就得到了更好的结果。

```
learn.unfreeze()
learn.fit(lrs/5, 1, cycle_len=2)

epoch      trn_loss   val_loss   accuracy
    0      0.676254   0.546998   0.834285
    1      0.460609   0.533741   0.833233
```

目录 Contents

最大目标检测

画出最大物体的边框乍一看似乎是没有学习过的问题，但实际上这个问题和之前所学习的内容区别并不大，只需要将找出最大物体和为其标明边框两个学过的任务结合起来即可。

值得注意的是之前我们网络的输出往往是一个具体的数或者类别，以达到“离散的”分类。我们知道在最后的全连接层不适用sigmod函数或者softmax函数，与此同时我们使用均方差误差作为损失函数，这样这个网络就能预测连续的数而非进行“离散的”分类。将两者联合起来思考，输出方框即输出层预测的结果应为4个数字，我们设置最后输出层具有4个激活，并用均方差误差作为损失函数。

最大目标检测

同样读取并处理原始数据：

```
BB_CSV = PATH/'tmp/bb.csv'
```

```
bb = np.array([trn_lrg_anno[o][0] for o in trn_ids])
bbs = [' '.join(str(p) for p in o) for o in bb]

df = pd.DataFrame({'fn': [trn_fns[o] for o in trn_ids], 'bbox': bbs}, columns=['fn','bbox'])
df.to_csv(BB_CSV, index=False)
```

```
BB_CSV.open().readlines()[:5]
```

```
['fn,bbox\n',
 '000012.jpg,96 155 269 350\n',
 '000017.jpg,77 89 335 402\n',
 '000023.jpg,1 2 461 242\n',
 '000026.jpg,124 89 211 336\n']
```

```
f_model=resnet34
sz=224
bs=64
```

现在我们将尝试找到最大对象的边界框，这是一个简单的回归，有4个输出，所以我们可以使用一个带有多个标签的CSV。注意此处文件的多个标签必须是空间分隔的，并且文件名是逗号分隔的。

最大目标检测

设置continuous=True说明这是个回归问题而不是分类问题，bs=4表示batch size=4 。CropType.NO表示是进行图片压缩而不是剪裁。

```
tfms = tfms_from_model(f_model, sz, crop_type=CropType.NO, tfm_y=TfmType.COORD, aug_tfms=augs)
md = ImageClassifierData.from_csv(PATH, JPEGS, BB_CSV, tfms=tfms, continuous=True, bs=4)
```

```
idx=3
fig,axes = plt.subplots(3,3, figsize=(9,9))
for i,ax in enumerate(axes.flat):
    x,y=next(iter(md.aug_dl))
    ima=md.val_ds.denorm(to_np(x))[idx]
    b = bb_hw(to_np(y[idx]))
    print(b)
    show_img(ima, ax=ax)
    draw_rect(ax, b)
```

```
[ 48.  34. 112. 188.]
[ 65.  36. 107. 185.]
[ 49.  27. 131. 195.]
[ 24.  18. 147. 204.]
[ 61.  34. 113. 188.]
[ 55.  31. 121. 191.]
[ 52.  19. 144. 203.]
[  7.   0. 193. 222.]
[ 52.  38. 105. 182.]
```

最大目标检测

Fastai中允许我们使用custom_head参数在已经训练好的网络主干上添加自定义的层，这里因为我们需要一个有4激活的输出层，所以添加了如下网络。

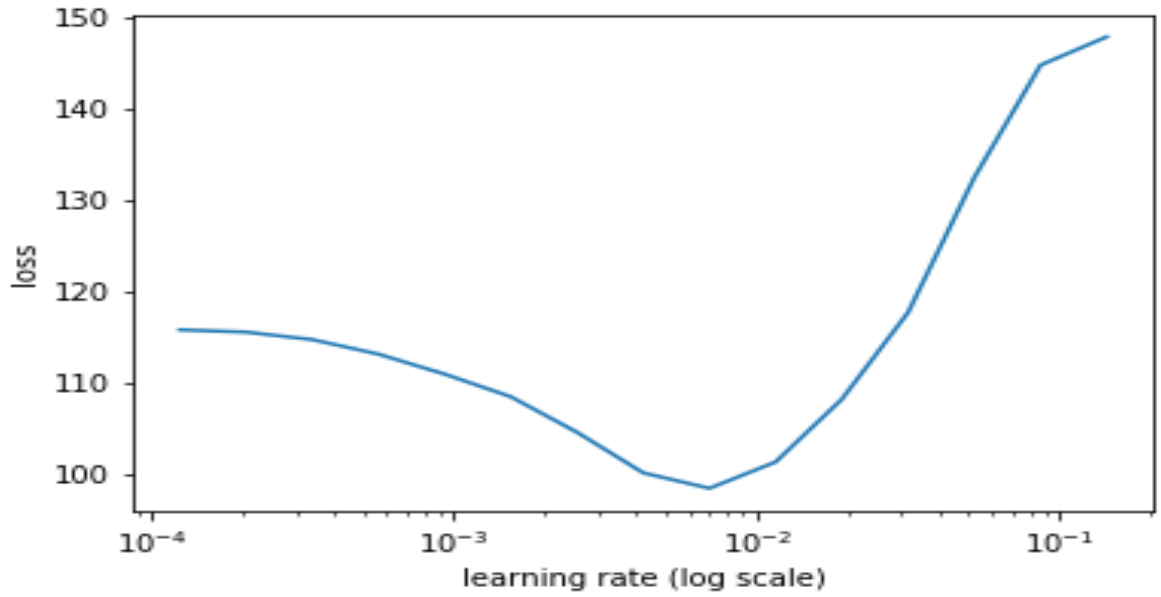
```
head_reg4 = nn.Sequential(Flatten(), nn.Linear(25088,4))
learn = ConvLearner.pretrained(f_model, md, custom_head=head_reg4)
learn.opt_fn = optim.Adam
learn.crit = nn.L1Loss()
```

最大目标检测

```
learn.lr_find(1e-5,100)
learn.sched.plot(5)
```

A Jupyter Widget

78%|████████████ | 25/32 [00:04<00:01, 6.16it/s, loss=395]



最大目标检测

仅训练最后一层的结果：

```
lr = 2e-3
```

```
learn.fit(lr, 2, cycle_len=1, cycle_mult=2)
```

```
A Jupyter Widget
epoch      trn_loss   val_loss
    0      49.523444  34.764141
    1      36.864003  28.007317
    2      30.925234  27.230705
```

解冻最后两层并训练的结果：

```
learn.fit(lrs, 2, cycle_len=1, cycle_mult=2)
```

```
A Jupyter Widget
epoch      trn_loss   val_loss
    0      25.616161  22.83597
    1      21.812624  21.387115
    2      17.867176  20.335539
```

解冻最后三层并训练的结果：

```
learn.freeze_to(-3)
```

```
learn.fit(lrs, 1, cycle_len=2)
```

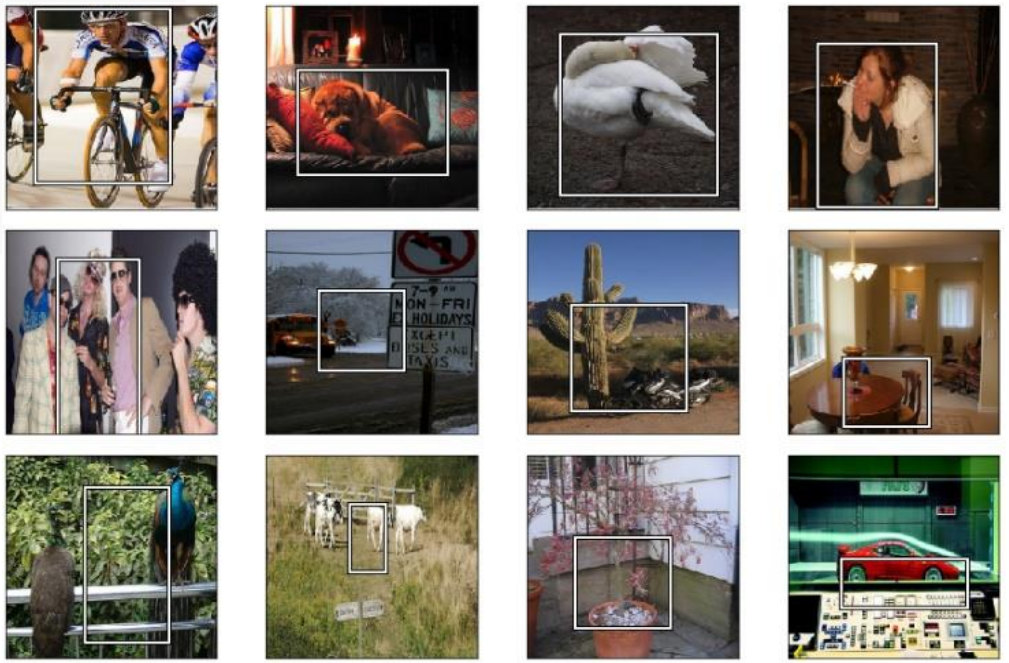
```
A Jupyter Widget
epoch      trn_loss   val_loss
    0      16.571885  20.948696
    1      15.072718  19.925312
```

最大目标检测

训练完之后批量查看结果：

```
x,y = next(iter(md.val_dl))
learn.model.eval()
preds = to_np(learn.model(VV(x)))
```

```
fig, axes = plt.subplots(3, 4, figsize=(12, 8))
for i,ax in enumerate(axes.flat):
    ima=md.val_ds.denorm(to_np(x))[i]
    b = bb_hw(preds[i])
    ax = show_img(ima, ax=ax)
    draw_rect(ax, b)
plt.tight_layout()
```



谢 谢！