In [1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
from fastai import *
from fastai.vision import *
import coco_text
```

Load selection of image ids and annotations with coco api

In [3]:
```
ct = coco_text.COCO_Text('COCO_Text.json')
```

```
loading annotations into memory...
0:00:02.132191
creating index...
index created!
```

In [4]:
```
ct.info()
```

```
url: http://vision.cornell.edu/se3/coco-text/
date_created: 2017-03-28
version: 1.4
description: This is 1.4 version of the 2017 COCO-Text dataset.
author: COCO-Text group
```

In [5]:
```
imgIds = ct.getImgIds(imgIds=ct.train,
                      catIds=[('legibility','legible'),('class','machine printed')])
```

In [6]:
```
annots = ct.getAnnIds(imgIds=ct.val,
                          catIds=[('legibility','legible'),('class','machine printed')],
                          areaRng=[0, 6000])
```

Show an example:

In [7]:
```
imgId = imgIds[10]
imgMetas = ct.loadImgs(ids=imgId)
file_name = imgMetas[0]['file_name']
path = Path('train/train2014')/file_name
image = open_image(path)
image
```

Out[7]:

In [8]:
```
#get annotations
annIds = ct.getAnnIds(imgIds=imgId)
anns = ct.loadAnns(annIds)
#show the bounding box coordinates of the first annotation:
anns[0]['bbox']
```

Out[8]:
```
[137.9307403564453, 190.3381929397583, 64.26467895507812, 31.616369247436523]
```

Create custom functions to get the annotations

In [9]:
```
classes = {}
annotations = []
images = []

imgMetas = ct.loadImgs(ids=imgIds)
annotIds = ct.getAnnIds(imgIds=imgIds)
annotMetas = ct.loadAnns(annotIds)

for imgMeta in imgMetas:
    images.append( {'id': imgMeta['id'], 'file_name': imgMeta['file_name']} )

for annotMeta in annotMetas:
    bbox = annotMeta['bbox']
    image_id = annotMeta['image_id']
    category_id = annotMeta['class']
    classes[annotMeta['class']] = annotMeta['class']
    annotations.append({'bbox': bbox, 'image_id': image_id, 'category': category_id})

annot_dict = {'annotations': annotations, 'images': images, 'classes': classes}
```

In [10]:
```
def get_annotations(annot_dict, prefix=None):
    id2images, id2bboxes, id2cats = {}, collections.defaultdict(list), collections.defaultdict(list)

    classes = annot_dict['classes']
    for o in annot_dict['annotations']:
        bb = o['bbox']
        id2bboxes[o['image_id']].append([bb[1],bb[0], bb[3]+bb[1], bb[2]+bb[0]])
        id2cats[o['image_id']].append(classes[o['category']])
    for o in annot_dict['images']:
        if o['id'] in id2bboxes:
            id2images[o['id']] = ifnone(prefix, '') + o['file_name']
    ids = list(id2images.keys())
    return [id2images[k] for k in ids], [[id2bboxes[k], id2cats[k]] for k in ids]
```

In [11]:
```
images, lbl_bbox = get_annotations(annot_dict)
img2bbox = dict(zip(images, lbl_bbox))
get_y_func = lambda o:img2bbox[o.split('/')[-1]]
# img2bbox
```

Check data inputs

In [12]:
```
img_df = pd.DataFrame(images, columns=['file_name'])
img_df.head()
```

Out[12]:

	file_name
0	COCO_train2014_000000294914.jpg
1	COCO_train2014_000000196610.jpg
2	COCO_train2014_000000458756.jpg
3	COCO_train2014_000000491526.jpg
4	COCO_train2014_000000032778.jpg

In [13]:
```
annot_df = pd.DataFrame(lbl_bbox, columns=['bbox', 'classes'])
annot_df.head()
```

Out[13]:

	bbox	classes
0	[[315.4072997275363, 342.5074517548974, 327.34...	[machine printed, machine printed, machine pri...
1	[[334.95858707783384, 348.54771784232366, 434....	[machine printed, machine printed, machine pri...
2	[[256, 266, 287, 309], [93.56962025316459, 324...	[machine printed, machine printed, machine pri...
3	[[198.76056338028167, 225.51095071353063, 232....	[machine printed, machine printed, machine pri...
4	[[478, 164, 491, 220], [478, 226, 492, 323]]	[machine printed, machine printed]

Create data object

In [14]:
```
cocoPath = Path('/home/jupyter/coco')

data = (ObjectItemList.from_df(img_df, cocoPath, folder='train/train2014')
        #Where are the images? -> in coco
        .random_split_by_pct()
        #How to split in train/valid? -> randomly with the default 20% in valid
        .label_from_func(get_y_func)
        #How to find the labels? -> use get_y_func
        .transform(get_transforms(), size=300, tfm_y=True, do_crop=False, padding_mode='zeros')
        #Data augmentation? -> Standard transforms with tfm_y=True
        .databunch(bs=2, collate_fn=bb_pad_collate))
        #Finally we convert to a DataBunch and we use bb_pad_collate
```

In [15]:
```
data.show_batch(1)
```

In [16]:
```
data.show_batch(1)
```

In [17]:
```
data.show_batch(1)
```

In [18]:
```
data.show_batch(1)
```

In [19]:
```
data.show_batch(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-3e33bc45fd7b> in <module>
----> 1 data.show_batch(1)

/opt/anaconda3/lib/python3.6/site-packages/fastai/basic_data.py in show_batch(self, rows, ds_type, **kwargs)
    153     def show_batch(self, rows:int=5, ds_type:DatasetType=DatasetType.Train, **kwargs)->None:
    154         "Show a batch of data in `ds_type` on a few `rows`."
--> 155         x,y = self.one_batch(ds_type, True, True)
    156         if self.train_ds.x._square_show: rows = rows ** 2
    157         xs = [self.train_ds.x.reconstruct(grab_idx(x, i, self._batch_first)) for i in range(rows)]

/opt/anaconda3/lib/python3.6/site-packages/fastai/basic_data.py in one_batch(self, ds_type, detach, denorm)
    134         w = self.num_workers
    135         self.num_workers = 0
--> 136         try:     x,y = next(iter(dl))
    137         finally: self.num_workers = w
    138         if detach: x,y = to_detach(x),to_detach(y)

/opt/anaconda3/lib/python3.6/site-packages/fastai/basic_data.py in __iter__(self)
     67         "Process and returns items from `DataLoader`."
     68         assert not self.skip_size1 or self.batch_size > 1, "Batch size cannot be one if skip_size1 i
s set to True"
---> 69         for b in self.dl:
     70             y = b[1][0] if is_listy(b[1]) else b[1]
     71             if not self.skip_size1 or y.size(0) != 1: yield self.proc_batch(b)

/opt/anaconda3/lib/python3.6/site-packages/torch/utils/data/dataloader.py in __next__(self)
    635                 self.reorder_dict[idx] = batch
    636                 continue
--> 637             return self._process_next_batch(batch)
    638
    639     next = __next__  # Python 2 compatibility

/opt/anaconda3/lib/python3.6/site-packages/torch/utils/data/dataloader.py in _process_next_batch(self, batc
h)
    656         self._put_indices()
    657         if isinstance(batch, ExceptionWrapper):
--> 658             raise batch.exc_type(batch.exc_msg)
    659         return batch
    660
```

```
TypeError: Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.6/site-packages/torch/utils/data/dataloader.py", line 138, in _worker_loo
p
    samples = collate_fn([dataset[i] for i in batch_indices])
  File "/opt/anaconda3/lib/python3.6/site-packages/fastai/vision/data.py", line 42, in bb_pad_collate
    max_len = max([len(s[1].data[1]) for s in samples])
  File "/opt/anaconda3/lib/python3.6/site-packages/fastai/vision/data.py", line 42, in <listcomp>
    max_len = max([len(s[1].data[1]) for s in samples])
  File "/opt/anaconda3/lib/python3.6/site-packages/torch/tensor.py", line 404, in __len__
    raise TypeError("len() of a 0-d tensor")
TypeError: len() of a 0-d tensor
```

In [ ]:

In [ ]: