冠军的试炼

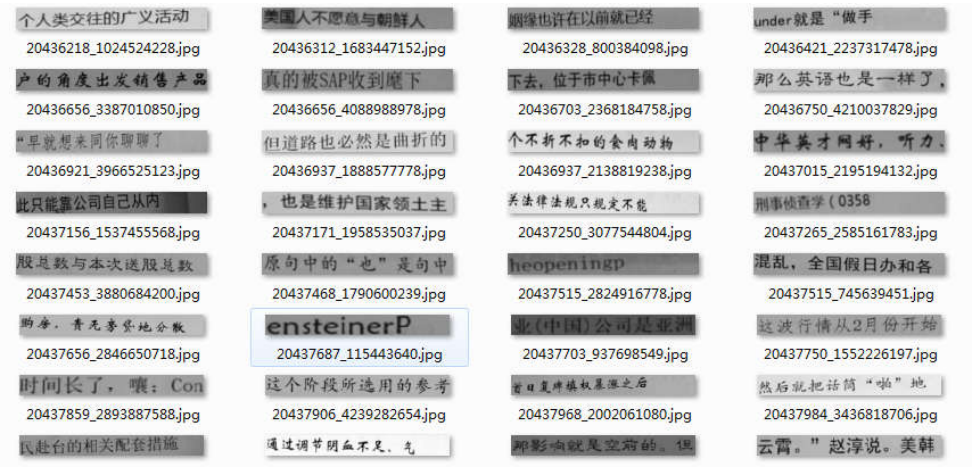
悟已往之不谏，知来者之可追

【OCR技术系列之八】端到端不定长文本识别CRNN代码实现

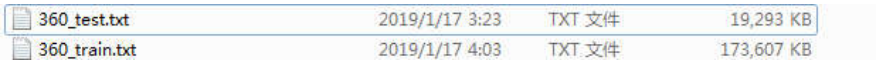
CRNN是OCR领域非常经典且被广泛使用的识别算法，其理论基础可以参考我上一篇文章，本文将着重讲解CRNN代码实现过程以及识别效果。

数据处理

利用图像处理技术我们手工大批量生成文字图像，一共360万张图像样本，效果如下：



我们划分了训练集和测试集（10:1），并单独存储为两个文本文件：



文本文件里的标签格式如下：

公告

昵称：Madcola
园龄：2年7个月
粉丝：1334
关注：30
+加关注

2019年8月

日	一	二	三	四	五	六
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类(69)

C++(1)
CUDA(1)
Linux编程(12)
OCR系列(8)
opencv探索(28)
STL(2)
波折岁月(4)
工具技巧(1)
机器学习之旅(5)
深度学习(4)
数字图像处理(3)

随笔档案(69)

2019年2月 (1)
2019年1月 (1)

```
364366  72685562_3468761464.jpg 司减资后的注册资本将
364367  72686187_1999452158.jpg 孙长官迟迟不愿过早调
364368  72687140_2765922188.jpg 雨交加云开雾散等效果
364369  72687375_3194773556.jpg 马刺帕克全场砍下最高
364370  72692390_3459797313.jpg 职卸掉了来自老板的压
364371  72694093_2567517260.jpg 己的女儿过了30岁后
364372  72694921_4262303020.jpg 经常做饭的，皆二八姝
364373  72695578_3201054776.jpg 果你想成为一个科级的
364374  72695921_299209679.jpg 有"顺义"、"大红门
364375  72697593_1275663487.jpg "永远是无名而质朴的
364376  72699109_1748104088.jpg 问那个学生:健全县、
364377  72699734_3823229382.jpg 见尊堂增强了学校服务
364378  72700906_2312372128.jpg 如其分地迎合其主流理
364379  72701515_18398291.jpg 域K对于域P的加罗华
364380  72701703_4284999016.jpg 以最大限度的增加理财
364381  72702671_4002101734.jpg 亚洲人民必须保持高度
364382  72703687_484902289.jpg 了，"该飞机完全符合
364383  72704703_2788033355.jpg :必不坐视而差不多同
364384  72705031_4117617091.jpg 要求相应的赔偿。联系
364385  72707187_3556392116.jpg 国政治关系中随时可能
364386  72708609_4040978918.jpg 罗迪克非常轻松的击败
364387  72708921_2293767637.jpg 域网和局域网都不能连
364388  72709390_1391827818.jpg 到去了以后手气特别背
364389  72711234_2844950077.jpg 样才可令身体真正吸收
364390  72711500_2076288204.jpg 股弃权。管理信息系统
364391  72711968_2897609052.jpg ，一本作"兵强则不胜
364392  72713343_149810275.jpg 6先"模仿"，所谓最
364393  72714562_113576196.jpg 立反卫星（ASAT）
364394  72721093_3493675887.jpg 等风俗。递交:200
364395  72721718_2056206001.jpg 必须注意尊重美国的利
364396  72722718_3485632692.jpg 但是湖人队的状态低迷
364397  72722937_1567927785.jpg 又过年"是我们形容幸
364398  72723218_3304231823.jpg 施—以色列F-16
364399  72723296_1145360198.jpg el复制了Lotus
364400  72726968_976070203.jpg 空军就袭击了这个机场
364401
```

我们获取到的是最原始的数据集，在图像深度学习训练中我们一般都会把原始数据集转化为lmdb格式以方便后续的网络训练。因此我们也需要对该数据集进行lmdb格式转化。下面代码就是用于lmdb格式转化，思路比较简单，就是首先读入图像和对应的文本标签，先使用字典将该组合存储起来（cache），再利用lmdb包的put函数把字典(cache)存储的k,v写成lmdb格式存储好（cache当有了1000个元素就put一次）。

```
import lmdb
import cv2
import numpy as np
import os


def checkImageIsValid(imageBin):
    if imageBin is None:
        return False
    try:
        imageBuf = np.fromstring(imageBin, dtype=np.uint8)
        img = cv2.imdecode(imageBuf, cv2.IMREAD_GRAYSCALE)
        imgH, imgW = img.shape[0], img.shape[1]
    except:
        return False
    else:
        if imgH * imgW == 0:
            return False
    return True


def writeCache(env, cache):
    with env.begin(write=True) as txn:
        for k, v in cache.items():
            txn.put(k, v)
```

2018年12月 (2)
2018年10月 (1)
2018年9月 (3)
2018年5月 (1)
2018年4月 (2)
2018年2月 (6)
2018年1月 (3)
2017年12月 (4)
2017年11月 (3)
2017年10月 (1)
2017年9月 (4)
2017年8月 (3)
2017年7月 (5)
2017年6月 (4)
2017年5月 (17)
2017年4月 (1)
2017年2月 (2)
2017年1月 (5)

积分与排名

积分 - 213285
排名 - 1747

最新评论

1. Re:【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）
感谢博主的无私奉献
--寒冬夜行人lee

2. Re:我的2018：OCR、实习和秋招
学长好厉害！我实习内容也是在做OCR，每天照着你的好多博客看...我是18级研究生2020年毕业然后最近也在边实习边准备秋招（但是我好菜），可不可以加学长微信交流交流呀，我的微信是Dreaminice.....
--Cocoalate

3. Re:【Keras】基于SegNet和U-Net的遥感图像语义分割
@wenny-bell我也有这样的问题，请问您解决了吗？可以加下qq讨论下，2724858160...
--嗯哼！！！

4. Re:【Keras】基于SegNet和U-Net的遥感图像语义分割
@wenny-bell我也有这样的问题，请问您解决了吗？可以加下qq讨论下，2724858160...
--嗯哼！！！

5. Re:OpenCV探索之路（十一）：轮廓查找和多边形包围轮廓
请教下，int thresh_size = (100 / 4) * 2 + 1; //自适应二值化阈值这个阈值的定义是怎么给出的呢？用你设定的参数确实得到了很好的效果，但是不知道为什么这样设置.谢谢~.....
--foxlucia

阅读排行榜

1. 卷积神经网络CNN总结(219292)
2. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster

```
def createDataset(outputPath, imagePathList, labelList, lexiconList=None, checkValid=True):
    """
    Create LMDB dataset for CRNN training.
    ARGS:
        outputPath    : LMDB output path
        imagePathList : list of image path
        labelList     : list of corresponding groundtruth texts
        lexiconList   : (optional) list of lexicon lists
        checkValid    : if true, check the validity of every image
    """
    assert (len(imagePathList) == len(labelList))
    nSamples = len(imagePathList)
    env = lmdb.open(outputPath, map_size=1099511627776)
    cache = {}
    cnt = 1
    for i in range(nSamples):
        imagePath = ''.join(imagePathList[i]).split()[0].replace('\n', '').replace('\r\n',
'')
        # print(imagePath)
        label = ''.join(labelList[i])
        print(label)
        # if not os.path.exists(imagePath):
        #     print('%s does not exist' % imagePath)
        #     continue

        with open('.' + imagePath, 'r') as f:
            imageBin = f.read()

        if checkValid:
            if not checkImageIsValid(imageBin):
                print('%s is not a valid image' % imagePath)
                continue
        imageKey = 'image-%09d' % cnt
        labelKey = 'label-%09d' % cnt
        cache[imageKey] = imageBin
        cache[labelKey] = label
        if lexiconList:
            lexiconKey = 'lexicon-%09d' % cnt
            cache[lexiconKey] = ' '.join(lexiconList[i])
        if cnt % 1000 == 0:
            writeCache(env, cache)
            cache = {}
            print('Written %d / %d' % (cnt, nSamples))
        cnt += 1
        print(cnt)
    nSamples = cnt - 1
    cache['num-samples'] = str(nSamples)
    writeCache(env, cache)
    print('Created dataset with %d samples' % nSamples)


OUT_PATH = '../crnn_train_lmdb'
IN_PATH = './train.txt'

if __name__ == '__main__':
    outputPath = OUT_PATH
    if not os.path.exists(OUT_PATH):
        os.mkdir(OUT_PATH)
    imgdata = open(IN_PATH)
    imagePathList = list(imgdata)

    labelList = []
    for line in imagePathList:
        word = line.split()[1]
        labelList.append(word)
    createDataset(outputPath, imagePathList, labelList)
```

我们运行上面的代码，可以得到训练集和测试集的lmdb

```
[ljs@localhost data]$ ls
test_lmdb  train_lmdb
```

在数据准备部分还有一个操作需要强调的，那就是文字标签数字化，即我们用数字来表示每一个文字（汉字，英

R-CNN(206007)
3. OpenCV探索之路（二十四）图像拼接和图像融合技术(88662)
4. CNN网络架构演进：从LeNet到DenseNet(58613)
5. OpenCV探索之路（二十三）：特征检测和特征匹配方法汇总(54127)
6. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(51232)
7. C++ STL快速入门(45935)
8. OpenCV探索之路（六）：边缘检测（canny、sobel、laplacian）(45154)
9. Linux编程之UDP SOCKET全攻略(41918)
10. OpenCV探索之路（十四）：绘制点、直线、几何图形(37782)

评论排行榜

1. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(82)
2. 【Keras】基于SegNet和U-Net的遥感图像语义分割(73)
3. OpenCV探索之路（二十四）图像拼接和图像融合技术(67)
4. 【OCR技术系列之八】端到端不定长文本识别CRNN代码实现(59)
5. 【Keras】从两个实际任务掌握图像分类(33)

推荐排行榜

1. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster R-CNN(92)
2. 卷积神经网络CNN总结(61)
3. 我的2018：OCR、实习和秋招(22)
4. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(21)
5. 【Keras】基于SegNet和U-Net的遥感图像语义分割(20)
6. CNN网络架构演进：从LeNet到DenseNet(20)
7. OpenCV探索之路（二十四）图像拼接和图像融合技术(18)
8. 我在北京实习的四个月(15)
9. 【OCR技术系列之一】字符识别技术总览(13)
10. 读研以来的一些感想：名校好在哪里?(13)

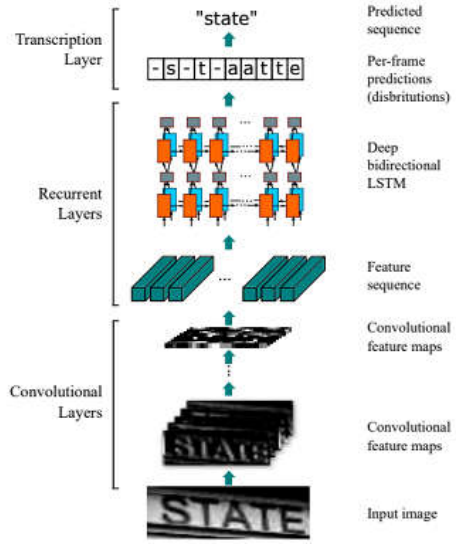
文字母，标点符号）。比如“我”字对应的id是1，“I”对应的id是1000，“？”对应的id是90，如此类推，这种编解码工作使用字典数据结构存储即可，训练时先把标签编码（encode），预测时就将网络输出结果解码(decode)成文字输出。

```
class strLabelConverter(object):
    """Convert between str and label.

    NOTE:
        Insert `blank` to the alphabet for CTC.

    Args:
        alphabet (str): set of the possible characters.
        ignore_case (bool, default=True): whether or not to ignore all of the case.
    """

    def __init__(self, alphabet, ignore_case=False):
        self._ignore_case = ignore_case
        if self._ignore_case:
            alphabet = alphabet.lower()
        self.alphabet = alphabet + '-'  # for `-1` index

        self.dict = {}
        for i, char in enumerate(alphabet):
            # NOTE: 0 is reserved for 'blank' required by wrap_ctc
            self.dict[char] = i + 1

    def encode(self, text):
        """Support batch or single str.

        Args:
            text (str or list of str): texts to convert.

        Returns:
            torch.IntTensor [length_0 + length_1 + ... length_{n - 1}]: encoded texts.
            torch.IntTensor [n]: length of each text.
        """

        length = []
        result = []
        for item in text:
            item = item.decode('utf-8', 'strict')

            length.append(len(item))
            for char in item:

                index = self.dict[char]
                result.append(index)

        text = result
        # print(text,length)
        return (torch.IntTensor(text), torch.IntTensor(length))

    def decode(self, t, length, raw=False):
        """Decode encoded texts back into strs.

        Args:
            torch.IntTensor [length_0 + length_1 + ... length_{n - 1}]: encoded texts.
            torch.IntTensor [n]: length of each text.

        Raises:
            AssertionError: when the texts and its length does not match.

        Returns:
            text (str or list of str): texts to convert.
        """
        if length.numel() == 1:
            length = length[0]
            assert t.numel() == length, "text with length: {} does not match declared length:
{}".format(t.numel(),

length)
            if raw:
                return ''.join([self.alphabet[i - 1] for i in t])
```

```
            else:
                char_list = []
                for i in range(length):
                    if t[i] != 0 and (not (i > 0 and t[i - 1] == t[i])):
                        char_list.append(self.alphabet[t[i] - 1])
                return ''.join(char_list)
        else:
            # batch mode
            assert t.numel() == length.sum(), "texts with length: {} does not match declared
length: {}".format(
                t.numel(), length.sum())
            texts = []
            index = 0
            for i in range(length.numel()):
                l = length[i]
                texts.append(
                    self.decode(
                        t[index:index + l], torch.IntTensor([l]), raw=raw))
                index += l
            return texts
```

网络设计

根据CRNN的论文描述，CRNN是由CNN-》RNN-》CTC三大部分架构而成，分别对应卷积层、循环层和转录层。首先CNN部分用于底层的特征提取，RNN采取了BiLSTM，用于学习关联序列信息并预测标签分布，CTC用于序列对齐，输出预测结果。



为了将特征输入到Recurrent Layers，做如下处理：

- 首先会将图像缩放到 32×W×3 大小
- 然后经过CNN后变为 1×（W/4）× 512
- 接着针对LSTM，设置 T=(W/4)， D=512 ，即可将特征输入LSTM。

以上是理想训练时的操作，但是CRNN论文提到的网络输入是归一化好的100×32大小的灰度图像，即高度统一为32个像素。下面是CRNN的深度神经网络结构图，CNN采取了经典的VGG16，值得注意的是，在VGG16的第3第4个max pooling层CRNN采取的是1×2的矩形池化窗口(w×h)，这有别于经典的VGG16的2×2的正方形池化窗口，这个改动是因为文本图像多数都是高较小而宽较长，所以其feature map也是这种高小宽长的矩形形状，如果使用1×2的池化窗口则更适合英文字母识别（比如区分i和l）。VGG16部分还引入了BatchNormalization模块，旨在加速模型收敛。还有值得注意一点，CRNN的输入是灰度图像，即图像深度为1。CNN部分的输出是512x1x16（c×h×w）的特征向量。

Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k:2 × 2, s:1, p:0
MaxPooling	Window:1 × 2, s:2
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
MaxPooling	Window:1 × 2, s:2
Convolution	#maps:256, k:3 × 3, s:1, p:1
Convolution	#maps:256, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:128, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:64, k:3 × 3, s:1, p:1
Input	W × 32 gray-scale image

接下来分析RNN层。RNN部分使用了双向LSTM，隐藏层单元数为256，CRNN采用了两层BiLSTM来组成这个RNN层，RNN层的输出维度将是（s,b,class_num），其中class_num为文字类别总数。

值得注意的是：Pytorch里的LSTM单元接受的输入都必须是3维的张量（Tensors）.每一维代表的意思不能弄错。第一维体现的是序列（sequence）结构，第二维度体现的是小块（mini-batch）结构，第三位体现的是输入的元素（elements of input）。如果在应用中不适用小块结构，那么可以将输入的张量中该维度设为1，但必须要体现出这个维度。

LSTM的输入

```
input of shape (seq_len, batch, input_size): tensor containing the features of the input
sequence.
The input can also be a packed variable length sequence.
input shape(a,b,c)
a:seq_len  -> 序列长度
b:batch
c:input_size   输入特征数目
```

根据LSTM的输入要求，我们要对CNN的输出做些调整，即把CNN层的输出调整为[seq_len, batch, input_size]形式，下面为具体操作：先使用squeeze函数移除h维度，再使用permute函数调整各维顺序，即从原来[w, b, c]的调整为[seq_len, batch, input_size]，具体尺寸为[16,batch,512]，调整好之后即可以将该矩阵送入RNN层。

```
x = self.cnn(x)
b, c, h, w = x.size()
# print(x.size()): b,c,h,w
assert h == 1   # "the height of conv must be 1"
x = x.squeeze(2)  # remove h dimension, b *512 * width
x = x.permute(2, 0, 1)  # [w, b, c] = [seq_len, batch, input_size]
x = self.rnn(x)
```

RNN层输出格式如下，因为我们采用的是双向BiLSTM，所以输出维度将是hidden_unit * 2

```
Outputs: output, (h_n, c_n)
output of shape (seq_len, batch, num_directions * hidden_size)
h_n of shape (num_layers * num_directions, batch, hidden_size)
c_n (num_layers * num_directions, batch, hidden_size)
```

然后我们再通过线性变换操作 `self.embedding1 = torch.nn.Linear(hidden_unit * 2, 512)` 是的输出维度再次变为512，继续送入第二个LSTM层。第二个LSTM层后继续接线性操作 `torch.nn.Linear(hidden_unit * 2, class_num)` 使得整个RNN层的输出为文字类别总数。

```
import torch
import torch.nn.functional as F


class Vgg_16(torch.nn.Module):
```

```
    def __init__(self):
        super(Vgg_16, self).__init__()
        self.convolution1 = torch.nn.Conv2d(1, 64, 3, padding=1)
        self.pooling1 = torch.nn.MaxPool2d(2, stride=2)
        self.convolution2 = torch.nn.Conv2d(64, 128, 3, padding=1)
        self.pooling2 = torch.nn.MaxPool2d(2, stride=2)
        self.convolution3 = torch.nn.Conv2d(128, 256, 3, padding=1)
        self.convolution4 = torch.nn.Conv2d(256, 256, 3, padding=1)
        self.pooling3 = torch.nn.MaxPool2d((1, 2), stride=(2, 1)) # notice stride of the non-
square pooling
        self.convolution5 = torch.nn.Conv2d(256, 512, 3, padding=1)
        self.BatchNorm1 = torch.nn.BatchNorm2d(512)
        self.convolution6 = torch.nn.Conv2d(512, 512, 3, padding=1)
        self.BatchNorm2 = torch.nn.BatchNorm2d(512)
        self.pooling4 = torch.nn.MaxPool2d((1, 2), stride=(2, 1))
        self.convolution7 = torch.nn.Conv2d(512, 512, 2)

    def forward(self, x):
        x = F.relu(self.convolution1(x), inplace=True)
        x = self.pooling1(x)
        x = F.relu(self.convolution2(x), inplace=True)
        x = self.pooling2(x)
        x = F.relu(self.convolution3(x), inplace=True)
        x = F.relu(self.convolution4(x), inplace=True)
        x = self.pooling3(x)
        x = self.convolution5(x)
        x = F.relu(self.BatchNorm1(x), inplace=True)
        x = self.convolution6(x)
        x = F.relu(self.BatchNorm2(x), inplace=True)
        x = self.pooling4(x)
        x = F.relu(self.convolution7(x), inplace=True)
        return x  # b*512x1x16


class RNN(torch.nn.Module):
    def __init__(self, class_num, hidden_unit):
        super(RNN, self).__init__()
        self.Bidirectional_LSTM1 = torch.nn.LSTM(512, hidden_unit, bidirectional=True)
        self.embedding1 = torch.nn.Linear(hidden_unit * 2, 512)
        self.Bidirectional_LSTM2 = torch.nn.LSTM(512, hidden_unit, bidirectional=True)
        self.embedding2 = torch.nn.Linear(hidden_unit * 2, class_num)

    def forward(self, x):
        x = self.Bidirectional_LSTM1(x)   # LSTM output: output, (h_n, c_n)
        T, b, h = x[0].size()   # x[0]: (seq_len, batch, num_directions * hidden_size)
        x = self.embedding1(x[0].view(T * b, h))  # pytorch view() reshape as [T * b, nOut]
        x = x.view(T, b, -1)  # [16, b, 512]
        x = self.Bidirectional_LSTM2(x)
        T, b, h = x[0].size()
        x = self.embedding2(x[0].view(T * b, h))
        x = x.view(T, b, -1)
        return x  # [16,b,class_num]


# output: [s,b,class_num]
class CRNN(torch.nn.Module):
    def __init__(self, class_num, hidden_unit=256):
        super(CRNN, self).__init__()
        self.cnn = torch.nn.Sequential()
        self.cnn.add_module('vgg_16', Vgg_16())
        self.rnn = torch.nn.Sequential()
        self.rnn.add_module('rnn', RNN(class_num, hidden_unit))

    def forward(self, x):
        x = self.cnn(x)
        b, c, h, w = x.size()
        # print(x.size()): b,c,h,w
        assert h == 1   # "the height of conv must be 1"
        x = x.squeeze(2)  # remove h dimension, b *512 * width
        x = x.permute(2, 0, 1)  # [w, b, c] = [seq_len, batch, input_size]
        # x = x.transpose(0, 2)
        # x = x.transpose(1, 2)
        x = self.rnn(x)
        return x
```

损失函数设计

刚刚完成了CNN层和RNN层的设计，现在开始设计转录层，即将RNN层输出的结果翻译成最终的识别文字结果，从而实现不定长的文字识别。pytorch没有内置的CTC loss，所以只能去Github下载别人实现的CTC loss来完成损失函数部分的设计。安装CTC-loss的方式如下：

```
git clone https://github.com/SeanNaren/warp-ctc.git
cd warp-ctc
mkdir build; cd build
cmake ..
make
cd ../pytorch_binding/
python setup.py install
cd ../build
cp libwarpctc.so ../../usr/lib
```

待安装完毕后，我们可以直接调用CTC loss了，以一个小例子来说明ctc loss的用法。

```
import torch
from warpctc_pytorch import CTCLoss
ctc_loss = CTCLoss()
# expected shape of seqLength x batchSize x alphabet_size
probs = torch.FloatTensor([[[0.1, 0.6, 0.1, 0.1, 0.1], [0.1, 0.1, 0.6, 0.1,
0.1]]]).transpose(0, 1).contiguous()
labels = torch.IntTensor([1, 2])
label_sizes = torch.IntTensor([2])
probs_sizes = torch.IntTensor([2])
probs.requires_grad_(True)  # tells autograd to compute gradients for probs
cost = ctc_loss(probs, labels, probs_sizes, label_sizes)
cost.backward()
```

```
CTCLoss(size_average=False, length_average=False)
    # size_average (bool): normalize the loss by the batch size (default: False)
    # length_average (bool): normalize the loss by the total number of frames in the batch.
If True, supersedes size_average (default: False)

forward(acts, labels, act_lens, label_lens)
    # acts: Tensor of (seqLength x batch x outputDim) containing output activations from
network (before softmax)
    # labels: 1 dimensional Tensor containing all the targets of the batch in one large
sequence
    # act_lens: Tensor of size (batch) containing size of each output sequence from the
network
    # label_lens: Tensor of (batch) containing label length of each example
```

从上面的代码可以看出，CTCLoss的输入为[probs, labels, probs_sizes, label_sizes]，即预测结果、标签、预测结果的数目和标签数目。那么我们仿照这个例子开始设计CRNN的CTC LOSS。

```

preds = net(image)
preds_size = Variable(torch.IntTensor([preds.size(0)] * batch_size))  # preds.size(0)=w=16
cost = criterion(preds, text, preds_size, length) / batch_size   # 这里的length就是包含每个文本标签的长度的list，除以batch_size来求平均loss
cost.backward()
```

网络训练设计

接下来我们需要完善具体的训练流程，我们还写了个trainBatch函数用于bacth形式的梯度更新。

```
def trainBatch(net, criterion, optimizer, train_iter):
    data = train_iter.next()
    cpu_images, cpu_texts = data
    batch_size = cpu_images.size(0)
    lib.dataset.loadData(image, cpu_images)
    t, l = converter.encode(cpu_texts)
    lib.dataset.loadData(text, t)
    lib.dataset.loadData(length, l)

    preds = net(image)
    #print("preds.size=%s" % preds.size)
    preds_size = Variable(torch.IntTensor([preds.size(0)] * batch_size))  #
preds.size(0)=w=22
```

```
    cost = criterion(preds, text, preds_size, length) / batch_size  # length= a list that
contains the len of text label in a batch
    net.zero_grad()
    cost.backward()
    optimizer.step()
    return cost
```

整个网络训练的流程如下：CTC-LOSS对象->CRNN网络对象->image,text,len的tensor初始化->优化器初始化，然后开始循环每个epoch，指定迭代次数就进行模型验证和模型保存。CRNN论文提到所采用的优化器是Adadelta，但是经过我实验看来，Adadelta的收敛速度非常慢，所以改用了RMSprop优化器，模型收敛速度大幅度提升。

```
    criterion = CTCLoss()

    net = Net.CRNN(n_class)
    print(net)

    net.apply(lib.utility.weights_init)

    image = torch.FloatTensor(Config.batch_size, 3, Config.img_height, Config.img_width)
    text = torch.IntTensor(Config.batch_size * 5)
    length = torch.IntTensor(Config.batch_size)

    if cuda:
        net.cuda()
        image = image.cuda()
        criterion = criterion.cuda()

    image = Variable(image)
    text = Variable(text)
    length = Variable(length)

    loss_avg = lib.utility.averager()

    optimizer = optim.RMSprop(net.parameters(), lr=Config.lr)
    #optimizer = optim.Adadelta(net.parameters(), lr=Config.lr)
    #optimizer = optim.Adam(net.parameters(), lr=Config.lr,
                           #betas=(Config.beta1, 0.999))

    for epoch in range(Config.epoch):
        train_iter = iter(train_loader)
        i = 0
        while i < len(train_loader):
            for p in net.parameters():
                p.requires_grad = True
            net.train()

            cost = trainBatch(net, criterion, optimizer, train_iter)
            loss_avg.add(cost)
            i += 1

            if i % Config.display_interval == 0:
                print('[%d/%d][%d/%d] Loss: %f' %
                      (epoch, Config.epoch, i, len(train_loader), loss_avg.val()))
                loss_avg.reset()

            if i % Config.test_interval == 0:
                val(net, test_dataset, criterion)

            # do checkpointing
            if i % Config.save_interval == 0:
                torch.save(
                    net.state_dict(), '{0}/netCRNN_{1}_{2}.pth'.format(Config.model_dir,
epoch, i))
```

训练过程与测试设计

下面这幅图表示的就是CRNN训练过程，文字类别数为6732，一共训练20个epoch，batch_Szie设置为64，所以一共是51244次迭代/epoch。

```
(env_pytorch3.5) [ljs@localhost crnn]$ python train.py
Using cuda
alphabet class num is 6736
CRNN(
  (cnn): Sequential(
    (vgg_16): Vgg_16(
      (convolution1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (pooling1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (convolution2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (pooling2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (convolution3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (convolution4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (pooling3): MaxPool2d(kernel_size=(1, 2), stride=(2, 1), padding=0, dilation=1, ceil_mode=Fals
      (convolution5): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (BatchNorm1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (convolution6): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (BatchNorm2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (pooling4): MaxPool2d(kernel_size=(1, 2), stride=(2, 1), padding=0, dilation=1, ceil_mode=Fals
      (convolution7): Conv2d(512, 512, kernel_size=(2, 2), stride=(1, 1))
    )
  )
  (rnn): Sequential(
    (rnn): RNN(
      (Bidirectional_LSTM1): LSTM(512, 256, bidirectional=True)
      (embedding1): Linear(in_features=512, out_features=512, bias=True)
      (Bidirectional_LSTM2): LSTM(512, 256, bidirectional=True)
      (embedding2): Linear(in_features=512, out_features=6736, bias=True)
    )
  )
)
[0/20][20/51244] Loss: 2.495454
[0/20][40/51244] Loss: 3.031315
[0/20][60/51244] Loss: 2.354279
[0/20][80/51244] Loss: 2.066727
[0/20][100/51244] Loss: 3.310245
[0/20][120/51244] Loss: 2.344869
[0/20][140/51244] Loss: 2.478002
[0/20][160/51244] Loss: 3.452201
[0/20][180/51244] Loss: 2.199530
[0/20][200/51244] Loss: 2.204273
```

在迭代4个epoch时，loss降到0.1左右，acc上升到0.98。

```
[4/20][37920/51244] Loss: 0.082352
[4/20][37940/51244] Loss: 0.091124
[4/20][37960/51244] Loss: 0.083361
[4/20][37980/51244] Loss: 0.126384
[4/20][38000/51244] Loss: 0.095119
Start val
Test loss: 0.127284, accuray: 0.977344
[4/20][38020/51244] Loss: 0.155867
```

接下来我们设计推断预测部分的代码，首先需初始化CRNN网络，载入训练好的模型，读入待预测的图像并resize为高为32的灰度图像，接着讲该图像送入网络，最后再将网络输出解码成文字即可输出。

```
import time
import torch
import os
from torch.autograd import Variable
import lib.convert
import lib.dataset
from PIL import Image
import Net.net as Net
import alphabets
import sys
import Config

os.environ['CUDA_VISIBLE_DEVICES'] = "4"

crnn_model_path = './bs64_model/netCRNN_9_48000.pth'
IMG_ROOT = './test_images'
running_mode = 'gpu'
alphabet = alphabets.alphabet
nclass = len(alphabet) + 1


def crnn_recognition(cropped_image, model):
    converter = lib.convert.strLabelConverter(alphabet)  # 标签转换

    image = cropped_image.convert('L')  # 图像灰度化

    ### Testing images are scaled to have height 32. Widths are
    # proportionally scaled with heights, but at least 100 pixels
    w = int(image.size[0] / (280 * 1.0 / Config.infer_img_w))
    #scale = image.size[1] * 1.0 / Config.img_height
    #w = int(image.size[0] / scale)
```

```
    transformer = lib.dataset.resizeNormalize((w, Config.img_height))
    image = transformer(image)
    if torch.cuda.is_available():
        image = image.cuda()
    image = image.view(1, *image.size())
    image = Variable(image)

    model.eval()
    preds = model(image)

    _, preds = preds.max(2)
    preds = preds.transpose(1, 0).contiguous().view(-1)

    preds_size = Variable(torch.IntTensor([preds.size(0)]))
    sim_pred = converter.decode(preds.data, preds_size.data, raw=False)  # 预测输出解码成文字
    print('results: {0}'.format(sim_pred))


if __name__ == '__main__':

    # crnn network
    model = Net.CRNN(nclass)

    # 载入训练好的模型，CPU和GPU的载入方式不一样，需分开处理
    if running_mode == 'gpu' and torch.cuda.is_available():
        model = model.cuda()
        model.load_state_dict(torch.load(crnn_model_path))
    else:
        model.load_state_dict(torch.load(crnn_model_path, map_location='cpu'))

    print('loading pretrained model from {0}'.format(crnn_model_path))

    files = sorted(os.listdir(IMG_ROOT))  # 按文件名排序
    for file in files:
        started = time.time()
        full_path = os.path.join(IMG_ROOT, file)
        print("================================================")
        print("ocr image is %s" % full_path)
        image = Image.open(full_path)

        crnn_recognition(image, model)
        finished = time.time()
        print('elapsed time: {0}'.format(finished - started))
```

识别效果和总结

首先我从测试集中抽取几张图像送入模型识别，识别全部正确。

我也随机在一些文档图片、扫描图像上截取了一段文字图像送入我们该模型进行识别，识别效果也挺好的，基本识别正确，表明模型泛化能力很强。

我还截取了增值税扫描发票上的文本图像来看看我们的模型能否还可以表现出稳定的识别效果：

这里做个小小的总结：对于端到端不定长的文字识别，CRNN是最为经典的识别算法，而且实战看来效果非常不错。上面识别结果可以看出，虽然我们用于训练的数据集是自己生成的，但是我们该模型对于pdf文档、扫描图像等都有很不错的识别结果，如果需要继续提升对特定领域的文本图像的识别，直接大量加入该类图像用于训练即可。CRNN的完整代码可以参考我的Github。

分类： OCR系列

Madcola
关注 - 30
粉丝 - 1334
+加关注

5 0

« 上一篇：【OCR技术系列之七】端到端不定长文字识别CRNN算法详解

posted @ 2019-02-01 11:44 Madcola 阅读(10989) 评论(59) 编辑 收藏

< Prev 1 2

评论列表

#51楼 2019-07-11 15:10 **明天周六了**

CNN用的是VGG16吗？为什么层数有点不太一样啊？

支持(0) 反对(0)

#52楼 2019-07-15 11:18 **puma360**

楼主你好， 请问train/test.txt里的数据已经是数字化过了吗？谢谢

支持(0) 反对(0)

#53楼 2019-07-15 15:57 **puma360**

想请问一下大家有什么办法可以将train和test.txt里已经数字化的部分转回文字

支持(0) 反对(0)

#54楼 2019-07-15 16:49 **puma360**

@ 西决英豪
你好，想请问一下将数据集转换为imdb的时候，是train/test.txt 和 .jpg图像集都需要吗？图像集就是那360w的那个吗，谢谢~

支持(0) 反对(0)

#55楼 2019-07-15 16:49 **puma360**

@ Keep_exercising
你好，想请问一下将数据集转换为imdb的时候，是train/test.txt 和 .jpg图像集都需要吗？图像集就是那360w的那个吗，谢谢~
支持(0) 反对(0)

#56楼 2019-07-15 16:51 **puma360**

@ ruz_zhang
你好， 请问生产imdb的时候是train/test.txt 和.jpg图像集都需要嘛 谢谢!
支持(0) 反对(0)

#57楼 2019-07-17 11:27 **puma360**

@ 走吧！走吧
引用
各位大佬，我标签设置的为带空格的字符串，例如 "80 33 181 758 2 662 19 94 71 73"，但是验证集正确率为0，就表示我的标签格式不对，那么到底应该怎么弄呢？因为汉字之间可以不加空格，但是汉字转换为数字后必须有空格，标签格式是啥样的?

我也有同样的疑惑。我们用的训练集和测试集里的文本已经是数字化的了，xxxx.jpg后面的一行字，每个字用数字表示并且用空格分开，word = line.split()[1]这一句只把每行的第一个数字读到labelList里了，请问这样的处理正确吗
支持(0) 反对(0)

#58楼 2019-07-27 09:36 **rider99**

不知是否有人遇到下面这个错误

RuntimeError: DataLoader worker (pid 2675) exited unexpectedly with exit code 1. Details are lost due to multiprocessing. Rerunning with num_workers=0 may give better error trace.
支持(0) 反对(1)

#59楼 2019-08-02 04:42 **杜沐清**

@ 寂寞的小乞丐
https://github.com/suntreeDu/easy_ocr
支持(0) 反对(0)

< Prev 1 2

刷新评论 刷新页面 返回顶部

相关博文:
· pytoch word_language_model 代码阅读
· Kerasvs.PyTorchinTransferLearning
· 【Pytorch】CIFAR-10分类任务
· Pytorch常用操作
· 使用神经网络分分彩源码下载拟合argmax函数