1.caffe-net.py

运行python caffe-net.py C:/Users/Administrator/Desktop/Caffe\_Using/caffe-master/myself/my\_train\_val.prototxt my\_train\_val.png

2.http://blog.csdn.net/cfyzcc/article/details/51318200

3.

#net.blobs['prob'].data[0]输出的就是分类的概率

4.caffe-python-API 接口：

http://blog.csdn.net/langb2014/article/details/53082704

5.明天继续：

1.http://www.linuxidc.com/Linux/2016-11/136774p17.htm

2.http://blog.csdn.net/cfyzcc/article/details/51318200（主要看这个网址）

6.参数解答：http://blog.csdn.net/guoyilin/article/details/42873747

7.caffe中的基本类Blob，Layer,Net:

http://blog.csdn.net/mounty\_fsc/article/details/51085654

----------

7-1:

Blob:是caffe中用来处理数据的，标准数组和统一内存接口

Blob（）：data\_(),diff\_(),count(0),capacity\_(0){}

explicit Blob(const int num, const int channels, const int height,const int width);

explicit Blob(const vector<int>& shape);

由源代码中可以注意到Blob有个成员变量：vector shape\_

其作用：

1.对于图像数据，shape可以定义为4维的数组(Num, Channels, Height, Width)或(n, k, h, w)，所以Blob数据维度为n\*k\*h\*w，

Blob是row-major保存的，因此在(n, k, h, w)位置的值物理位置为((n \* K + k) \* H + h) \* W + w。

其中Number是数据的batch size，对于256张图片为一个training batch的ImageNet来说n = 256(即训练的图片数);

Channel是特征维度，如RGB图像k = 3

2.对于全连接网络，使用2D blobs (shape (N, D))，然后调用InnerProductLayer

3.对于参数，维度根据该层的类型和配置来确定。对于有3个输入96个输出的卷积层，Filter核 11 x 11，

则blob为96（出） x 3（入） x 11 x 11. 对于全连接层，1000个输出，1024个输入，则blob为1000（出） x 1024（入）.

--------------

7-2：

Layer:是Caffe的基础以及基本计算单元。

Caffe十分强调网络的层次性，可以说，一个网络的大部分功能都是以Layer的形式去展开的，

如convolute,pooling,loss等等。

在创建一个Caffe模型的时候，也是以Layer为基础进行的，

需按照src/caffe/proto/caffe.proto中定义的网络及参数格式定义网络 prototxt文件

layer {

name: "conv1"

type: "Convolution"

bottom: "data" #（输入）

top: "conv1" #（输出）

....

}

说明：每一层定义了三种操作：

1.Setup：Layer的初始化

2.Forward：前向传导计算，根据bottom计算top，调用了Forward\_cpu（必须实现）和Forward\_gpu

（可选，若未实现，则调用cpu的）

3.Backward：反向传导计算，根据top计算bottom的梯度，其他同上

Layer派生类分类:

1.Vision Layers :

Vison 层主要用于处理视觉图像相关的层，以图像作为输入，产生其他的图像。其主要特点是具有空间结构。

包含Convolution(conv\_layer.hpp)、Pooling(pooling\_layer.hpp)、Local Response Normalization(LRN)(lrn\_layer.hpp)、

im2col等，注：老版本的Caffe有头文件include/caffe/vision\_layers.hpp，

新版本中用include/caffe/layer/conv\_layer.hpp等取代

2.Loss Layers：

这些层产生loss，如Softmax(SoftmaxWithLoss)、Sum-of-Squares / Euclidean(EuclideanLoss)、

Hinge / Margin(HingeLoss)、Sigmoid Cross-Entropy(SigmoidCrossEntropyLoss)、

Infogain(InfogainLoss)、Accuracy and Top-k等

3.Activation / Neuron Layers：

元素级别的运算，运算均为同址计算（in-place computation，返回值覆盖原值而占用新的内存）。

如：ReLU / Rectified-Linear and Leaky-ReLU(ReLU)、Sigmoid(Sigmoid)、TanH / Hyperbolic Tangent(TanH)、

Absolute Value(AbsVal)、Power(Power)、BNLL(BNLL)等

4.Data Layers：

网络的最底层，主要实现数据格式的转换，如：Database(Data)、In-Memory(MemoryData)、HDF5 Input(HDF5Data)、

HDF5 Output(HDF5Output)、Images(ImageData)、Windows(WindowData)、Dummy(DummyData)等

5.Common Layers：

Caffe提供了单个层与多个层的连接。如：Inner Product(InnerProduct)、Splitting(Split)、

Flattening(Flatten)、Reshape(Reshape)、Concatenation(Concat)、Slicing(Slice)、Elementwise(Eltwise)、

Argmax(ArgMax)、Softmax(Softmax)、Mean-Variance Normalization(MVN)等

-----------

7-3：

Net:一个Net由多个Layer组成。一个典型的网络从data layer（从磁盘中载入数据）出发到loss layer结束。

如图是一个简单的逻辑回归分类器。

8.Caffe的进阶练习：https://www.zhihu.com/question/27982282

问题：

1.为什么显示不了原图？

解：

img = caffe.io.load\_image(caffe\_root +'\\myself\\classification\_test\\501.jpg')

traformed\_image = transformer.preprocess('data',img)

plt.imshow(img)

plt.show()

2.caffe输入的原图的格式是什么？

解：

在pycaffe文件中io.py源码中：

img = caffe.io.load\_image(caffe\_root +'\\myself\\classification\_test\\501.jpg')

Image IO文件中：

def load\_image(filename, color=True):

True (default) loads as RGB while False loads as intensity (if image is already grayscale).

Returns

-------

image : an image with type np.float32 in range [0, 1]

of size (H x W x 3) in RGB or

of size (H x W x 1) in grayscale.

总结：刚开始load\_image读入的图像是（H\*W\*3）->preprocess(3\*H\*W)

------------------

cv2.imread()接口读图像读进来的直接就是BGR格式（0-255），所以不需要缩放到【0,255】和通道变换【2，1,0】

不需要transformer.set\_raw\_scale('data',255)和transformer.set\_channel\_swap('data',(2,1,0))

------------------

若是caffe.io.load\_image()读进来是RGB格式和0~1(float）

所以在进行特征提取之前要在transformer中设置transformer.set\_raw\_scale('data',255)(缩放至0~255）

------------------

以及transformer.set\_channel\_swap('data',(2,1,0)(将RGB变换到BGR）

http://blog.csdn.net/summermaoz/article/details/64442707

plt.imshow(transformer.deprocess('data', net.blobs['data'].data[0]))

原来输入的图像是h\*w\*c->c\*h\*w(NET需要的)

3.

plt(RGB)和opencv（BGR）读图片的区别：http://blog.csdn.net/lights\_joy/article/details/45933907

opencv是以BGR的颜色空间读的图片

所以需要转换：

(r,g,b)=cv2.split(img)

img=cv2.merge([b,g,r])

plt.imshow(img)

plt.show()

4.python 中的\_\_init\_\_(self,....):问题和list,tuple,dict,set:http://blog.csdn.net/liuyanfeier/article/details/53731239

总结：

一，列表【list】

定义一个列表使用一对中(方)括号[ ]

python内置的一种数据类型是列表：

list是一种有序的数据集合，可以随意的添加和删除其中的数据。

比如列出班里所有的同学的名字，列出所有工厂员工的工号等都是可以用到列表的。

L.append(var) #追加元素

L.insert(index,var)

L.pop(var) #返回最后一个元素，并从list中删除之

L.remove(var) #删除第一次出现的该元素

L.count(var) #该元素在列表中出现的个数

L.index(var) #该元素的位置,无则抛异常

L.extend(list6) #追加list６，即合并list到L上,这里注意，使用extend函数可以一次在一个列表中插入任意多个值，而不必须每次只使用append()一次一值的插入

L.sort() #排序

L.reverse() #倒序

del L[1] #删除指定下标的元素

del L[1:3] #删除指定下标范围的元素

#复制list:

L1 = L #L1为L的别名，用C来说就是指针地址相同，对L1操作即对L操作。

L1 = L[:] #L1为L的克隆，即另一个拷贝。

------------------

二，元组（Tuple）

定义一个元组使用一对小(圆)括号( ) 。

和列表类似，元组也是一种有序列表，虽然tuple和list非常之类似，但是list初始化之后使可以改变的，但是，元组一旦初始化之后就不可以改变。

这点与Python中的字符串类似，所以我们说元组和字符串都是不可变的序列。

现在tuple不能变了，它也没有append()，insert()这样的方法。

其他获取元素的方法和list是一样的，你可以正常地使用tuple[0]，tuple[-1]，但不能赋值成另外的元素。

不可变的tuple有什么意义？因为tuple不可变，所以代码更安全。如果可能，能用tuple代替list就尽量用tuple。

tuple的陷阱：

1.当你定义一个tuple时，在定义的时候，tuple的元素就必须被确定下来；

2.定义只有一个元素的Tuple的时候，需要这样:

tuple1 = (123,)

后面要加上一个逗号，这是因为括号()既可以表示tuple，又可以表示数学公式中的小括号，这就产生了歧义。

tuple2 = (123) ＃如果你这样定义你定义的将是１２３这个元素，而不是一个元组。

python在显示只有1个元素的tuple时，也会加一个逗号,，以免你误解成数学计算意义上的括号。

元组的内置函数：

1.比较两个元组元素：cmp(tuple1,tuple2)相等的话返回0，不相等返回1；

2.计算元组的长度：len(tuple

3.返回元组中的最大值最小值：max(tuple),min(tuple)；

4.将列表转换成元组：Tuple = tuple(list)

-------------------

三，字典{Dict}

定义 Dictionary 使用一对大(花)括号 { }

字典(Dictionary) 是 Python 的内置数据类型之一，它定义了键和值之间一对一的关系,但它们是以无序的方式储存的。

字典中的“值”通过键来引用。

与列表区别：字典是无序的，在字典中通过键来访问成员。

字典是可变的，可以包含任何其他类型

常用字典操作：

dic.clear()清空字典

dic.keys()获得键的列表

dic.values()获得值的列表

dic.copy()复制字典

dic.pop(k)删除键k

dic.get(k)获得键k的值

dic.update()更新成员，若成员不存在，相当于加入

dic.items()获得由键和值组成的列表

get()语法：

dict.get(key, default=None)

参数

key – 字典中要查找的键。

default – 如果指定键的值不存在时，返回该默认值值。

返回值

返回指定键的值，如果值不在字典中返回默认值None。

---------------------

四，集合（Set）

Python的集合(set)和其他语言类似, 是一个无序不重复元素集, 基本功能包括关系测试和消除重复元素.

#定义一个集合

set1 = {1, 2, 3, 4, 5}

# 或者使用 set 函数

list1 = [6, 7, 7, 8, 8, 9]

set2 = set(list1)

set2.add(10) # 添加新元素

print set2 # set([8, 9, 6, 7]) 去掉重复内容,而且是无序的

set3 = frozenset(list1) #固定集合

set3.add(10) # 固定集合不能添加元素

#方法（所有的集合方法）：

s.issubset(t) #如果s是t的子集,返回True，否则返回False

s.issuperset(t) #如果s是t的超集,返回True，否则返回False

s.union(t) #返回一个新集合, 该集合是s和t的并集

s.intersection(t) #返回一个新集合, 该集合是s和t的交集

s.difference(t) #返回一个新集合, 该集合是s的成员, 但不是t的成员, 即返回s不同于t的元素

s.symmetric\_defference(t) #返回所有s和t独有的(非共同拥有)元素集合

s.copy() #返回一个s的浅拷贝, 效率比工厂要好

#方法（仅适用于可变集合）:以下方法参数必须是可哈希的

s.update(t) #用t中的元素 修改s，即s现在包含s或t的成员

s.intersection\_update(t) #s中的成员是共同属于s和t的元素

s.difference\_update(t) #s中的成员是属于s但不包含在t中的元素

s.symmetric\_difference\_update(t) #s中的成员更新为那些包含在s或t中，但不是s和t共有的元素

s.add(obj) #在集合s中添加对象obj

s.remove(obj) #从集合s中删除对象obj，如果obj不是集合s中的元素（obj not in s）,将引发keyError错误

s.discard(obj) #如果obj是集合s中的元素，从集合s中删除对象obj

s.pop() #删除集合s中得任意一个对象，并返回它

s.clear() #删除集合s中的所有元素

## 集合有并集，交集，求差操作

## 并集：intersection() 方法返回一个新集合，包含在两个集合中同时出现的所有元素。

## 交集：union() 方法返回一个新集合，包含在两个 集合中出现的元素。

## 差集：difference() 方法返回的新集合中，包含所有在 集合A出现但未在集合B中的元素。

## symmetric\_difference() 方法返回一个新集合，包含所有只在其中一个集合中出现的元素。

# 删除元素

set2.discard(6) # 当元素不存在时,不会引发异常

set2.remove(6) # 与discard的区别在于，如果没有要删除的元素，remove会引发一个异常

set2.pop() # 因为set是无序的，所以pop会随机的从set中删除一个元素