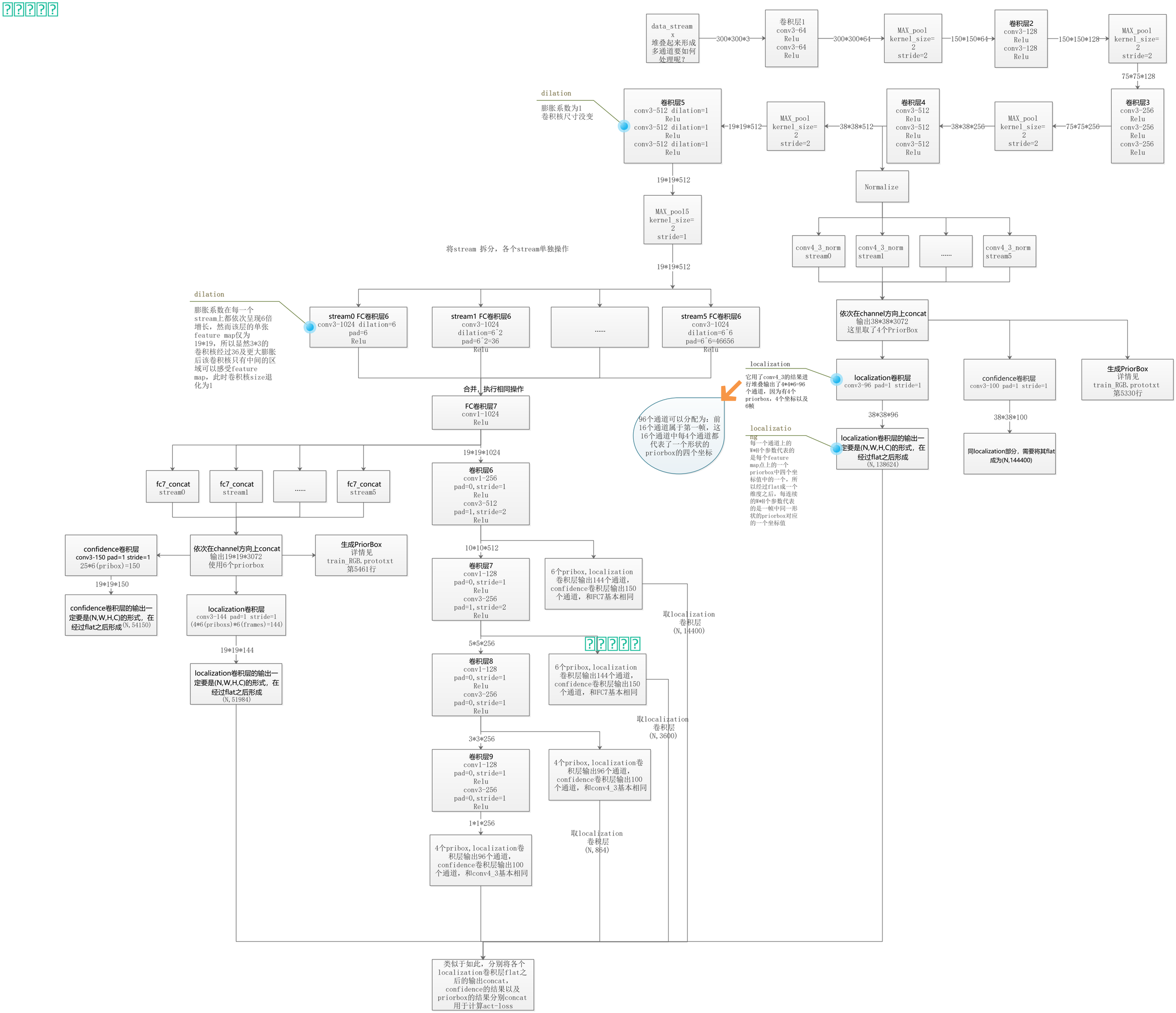

data_stream
x
堆叠起来形成多通道要如何处理呢？
300*300*3
卷积层1
conv3-64
Relu
conv3-64
Relu
300*300*64
MAX_pool
kernel_size=2
stride=2
150*150*64
卷积层2
conv3-128
Relu
conv3-128
Relu
150*150*128
MAX_pool
kernel_size=2
stride=2
75*75*128
卷积层3
conv3-256
Relu
conv3-256
Relu
conv3-256
Relu
75*75*256
MAX_pool
kernel_size=2
stride=2
38*38*256
卷积层4
conv3-512
Relu
conv3-512
Relu
conv3-512
Relu
38*38*512
MAX_pool
kernel_size=2
stride=2
19*19*512
dilation
膨胀系数为1
卷积核尺寸没变
卷积层5
conv3-512 dilation=1
Relu
conv3-512 dilation=1
Relu
conv3-512 dilation=1
Relu
19*19*512
MAX_pool5
kernel_size=2
stride=1
19*19*512
Normalize
conv4_3_norm
stream0
conv4_3_norm
stream1
.....
conv4_3_norm
stream5
依次在channel方向上concat
输出38*38*3072
这里取了4个PriorBox
生成PriorBox
详情见
train_RGB.prototxt
第5330行
将stream 拆分，各个stream单独操作
dilation
膨胀系数在每一个stream上都依次呈现6倍增长，然而该层的单张feature map仅为19*19，所以显然3*3的卷积核经过36及更大膨胀后该卷积核只有中间的区域可以感受feature map，此时卷积核size退化为1
stream0 FC卷积层6
conv3-1024 dilation=6
pad=6
Relu
stream1 FC卷积层6
conv3-1024
dilation=6^2
pad=6^2=36
Relu
.....
stream5 FC卷积层6
conv3-1024
dilation=6^6
pad=6^6=46656
Relu
合并，执行相同操作
FC卷积层7
conv1-1024
Relu
19*19*1024
卷积层6
conv1-256
pad=0,stride=1
Relu
conv3-512
pad=1,stride=2
Relu
10*10*512
卷积层7
conv1-128
pad=0,stride=1
Relu
conv3-256
pad=1,stride=2
Relu
5*5*256
卷积层8
conv1-128
pad=0,stride=1
Relu
conv3-256
pad=0,stride=1
Relu
3*3*256
卷积层9
conv1-128
pad=0,stride=1
Relu
conv3-256
pad=0,stride=1
Relu
1*1*256
4个pribox,localization卷积层输出96个通道，confidence卷积层输出100个通道，和conv4_3基本相同
6个pribox,localization卷积层输出144个通道，confidence卷积层输出150个通道，和FC7基本相同
取localization卷积层
(N,14400)
6个pribox,localization卷积层输出144个通道，confidence卷积层输出150个通道，和FC7基本相同
取localization卷积层
(N,3600)
4个pribox,localization卷积层输出96个通道，confidence卷积层输出100个通道，和conv4_3基本相同
取localization卷积层
(N,864)
localization
它用了conv4_3的结果进行堆叠输出了4*4*6=96个通道，因为有4个priorbox，4个坐标以及6帧
96个通道可以分配为：前16个通道属于第一帧，这16个通道中每4个通道都代表了一个形状的priorbox的四个坐标
localization卷积层
conv3-96 pad=1 stride=1
confidence卷积层
conv3-100 pad=1 stride=1
生成PriorBox
详情见
train_RGB.prototxt
第5330行
38*38*96
38*38*100
localization卷积层的输出一定要是(N,W,H,C)的形式，在经过flat之后形成
(N,138624)
同localization部分，需要将其flat成为(N,144400)
localization
每一个通道上的W*H个参数代表的是每个feature map点上的一个priorbox中四个坐标值中的一个，所以经过flat成一个维度之后，每连续的W*H个参数代表的是一帧中同一形状的priorbox对应的一个坐标值
fc7_concat
stream0
fc7_concat
stream1
.....
fc7_concat
stream5
依次在channel方向上concat
输出19*19*3072
使用6个priorbox
生成PriorBox
详情见
train_RGB.prototxt
第5461行
confidence卷积层
conv3-150 pad=1 stride=1
25*6(pribox)=150
19*19*150
confidence卷积层的输出一定要是(N,W,H,C)的形式，在经过flat之后形成
(N,54150)
localization卷积层
conv3-144 pad=1 stride=1
(4*6(priboxs)*6(frames)=144)
19*19*144
localization卷积层的输出一定要是(N,W,H,C)的形式，在经过flat之后形成
(N,51984)
类似于如此，分别将各个localization卷积层flat之后的输出concat，confidence的结果以及priorbox的结果分别concat用于计算act-loss

本周主要完成基础网络的细化和编程，对 SSD 的网络部分的具体细节根据 prototxt 文件绘制流程图，进一步摸清了其各个步骤的细节，同时进行了编程实现以及将作者提供的 caffemodel 中的预训练参数进行提取和移植到了 pytorch 网络中。

考虑了**论文复现的思路**：为了尽量减少低级错误带来的时间浪费，决定先自己写出 pytorch 版本的代码，然后直接将作者提供的训练好的参数载入，看能否复现结果，保证测试的网络结构没有什么问题，之后再次使用作者提供的预训练参数进行训练，看能否复现结果。

训练数据的获取：

UCF101 数据集内包含 101 个 class，如果只是用于分类只需要各个视频的 split file 和 labels 即可，考虑到要使用 SSD 网络生成 tubes 故需要找到其 ground truth 文件。兜兜转转在论文作者给出的 readme 中找到作者使用的数据集的所有 tubes 的 dataset。

其 cache file 中为每一个视频配备了 gttubes 属性，每一个 tube 都对对应视频的每一帧提供了 5 个参数：帧编号，ground_truth_x1，ground_truth_y1，ground_truth_x2，ground_truth_y2。

在加载 cache 过程中出现错误：

```
UnicodeDecodeError: 'ascii' codec can't decode byte 0x80 in position 2: ordinal not in range(128)
```

这个经过查询这个错误应该是 python2 和 python3 使用的 pickle 不一样导致的，在 load 的时候指定其编码即可，之前也遇到过类似的问题但是是发生在 pip install 中，那次是更新了 pip 就好了。:

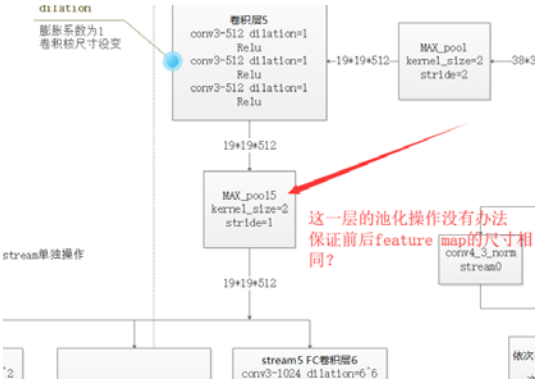
```
cache = pickle.load(fid, encoding='iso-8859-1')
```

但是找到的可以用来训练的 tube cache 只有 24 个类，经过对比，应该是作者只取了 UCF101 中的 24 个类。。

网络细节部分：

为了清楚网络各个部分的细节实现，仔细阅读了作者提供的 train_RGB.prototxt，考虑到其网络复杂，该文件非常长，故花了点时间将其转化为流程图的形式，希望通过这样直观的方法减少编程中的错误，同时加强一遍理解：

对于流程图中的细节，是通过对照 SSD 网络和 prototxt 来完成的，编程实验的时候发现在其中有一个池化操作是无法保证预期的效果，在 kernel_size 为 2 的情况下，无论是否 padding 都没有办法保证前后 feature map 的大小一致！如果这一个池化层的 padding 设置为 0，那么将会从 19*19 变为 18*18，在后续的卷积操作中大小可以恢复正常，但是这里的 pool 损失了一列和一行特征。所以我认为这里的 padding 设置为 1 比较合适，这样经过这一层池化之后的尺寸为 20*20，虽然变大了但是不会损失特征，想请问一下学长，不知道这样的做法合理？



注意到其由 conv5-fc6 的卷积操作使用到了 dilation 操作，膨胀卷积。膨胀卷积计算及其含义，卷积核膨胀是将卷积核扩张到膨胀尺度约束的尺度中，并将原卷积核没有占用的区域填充 0，然后 pad 的选取和膨胀卷积的膨胀系数一样大，保证了大小。

膨胀的卷积核尺寸 = 膨胀系数*（原始卷积核尺寸 - 1）+ 1

Caffe 卷积核膨胀：

https://blog.csdn.net/jiongnima/article/details/69487519

预训练参数的提取：

希望将作者提供的 caffemodel 转化为 pytorch 参数对网络进行 pretrain，之前用 inception 的时候有过将 caffemodel 转为 pytorch，试图通过同样的办法获得预训练参数，但是经过折腾之后发现那一份文件只对 inception 有效。里面 message 的实现中有很多都没有！所以很多参数在 load 的过程中都会出现找不到对应的情况而报错。

为了将 caffemodel 中的数据读取出，最后把原作者的 caffe 重新编译之后才成功将网络的所有卷积层参数读取出，并将使用 pickle 将其保存为 numpy 数组在 pytorch 中读取和加载。

将 caffe 中的 weights 取出后发现其参数形式为(output_channels, input_channels, kernel_size)，这点与 pytorch 的参数对应上了，提取出来之后可以直接导入。

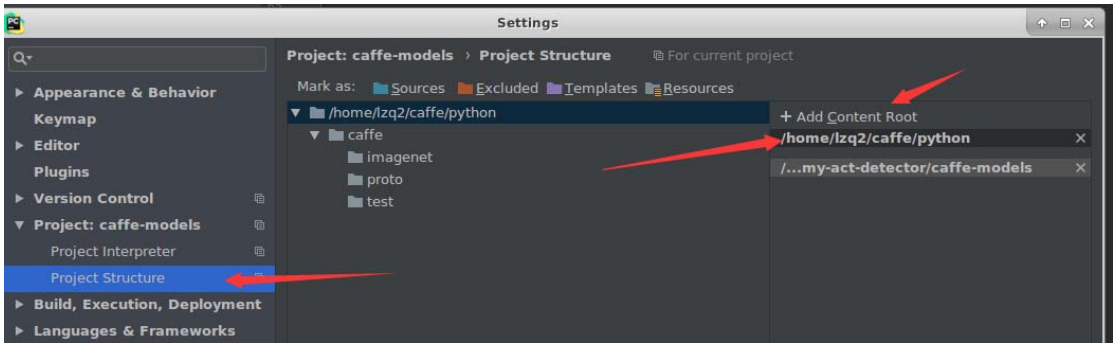
在 prototxt 中发现 caffe 中为每一帧都单独分配了一个卷积层，然后依次进行计算，但是根据之前的使用习惯，思考能否直接将 6 帧进行堆叠，对应的 input_channel 和 output_channel 各自放大 6 倍，写出后发现参数无法进行一一对应，其实后面的流程图就反映了这个思想，但是在编程后发现，这样做考虑了 6 帧之间的关系，卷积层的复杂度很明显上升！但是作者的做法是 6 帧之间完全是独立的，这样很显然是错误的。

本文最后记录了一些在编译 caffe 中遇到的问题及解决方案：

from google.protobuf import text_format

google module not found:　pip install google & pip install protobuf

pycharm 中找不到 caffe，但是在命令行中可以 import：



报找不到 hdf5.h，在 makefile.config 中的 INCLUDE_DIRS 后面添加：

/usr/include/hdf5/serial 即可，hdf5.h 的路径不在/usr/local/include 中

命令窗口添加路径 PYTHONPATH：

export PYTHONPATH=$PYTHONPATH:/home/ershisui

注意：此方法只在当前命令窗口生效，即如果打开一个新的 Terminal 窗口，定位到当

前目录，　打印 PYTHONPATH 是没有刚才加入的路径的．

Cannot find -lhdf5_hl
Cannot find -lhdf5
Caffe 编译的时候出现上面这两个错误，可以先使用
locate libhdf5 对 libhdf5.so 进行定位
然后 ln -s 将找到的库文件链接到/usr/lib/文件夹下面！而不是/usr/bin!

编译 pycaffe 时报错：fatal error: numpy/arrayobject.h 没有那个文件或目录：

```
1 import numpy as np
2 np.get_include()
```

得到：
/usr/local/lib/python2.7/dist-packages/numpy/core/include
在 Makefile.config 找到 PYTHON_INCLUDE，发现有点不同：
PYTHON_INCLUDE := /usr/include/python2.7 \
　　/usr/lib/python2.7/dist-packages/numpy/core/include
要加一个 local，变成：
PYTHON_INCLUDE := /usr/include/python2.7 \
　　/usr/local/lib/python2.7/dist-packages/numpy/core/include
再 make pycaffe 就 ok 了