

代码查看与使用

这里解释一下各个文件所包含的内容, 利于读者查看代码. 由于使用的VScode, 所以自己写了CMakeList.txt和.vscode后build目录下的东西均是自动生成的(如果有比较方便的编译器而不需要自己来手动编译都可忽略). .h与.cpp一一对应, .h申明, .cpp定义. utils.h为使用到的一些基础函数, 包括快速幂, 对权值的初始化(使用高斯分布初始化). utile_other.h 是另外的一些基础模板函数, 均是new出空间和delete掉空间. load_data.h为装载图片的类. layer.h是上文中介绍的网络中各个层的类. network.h为网络结构类. main.cpp为最终训练程序.

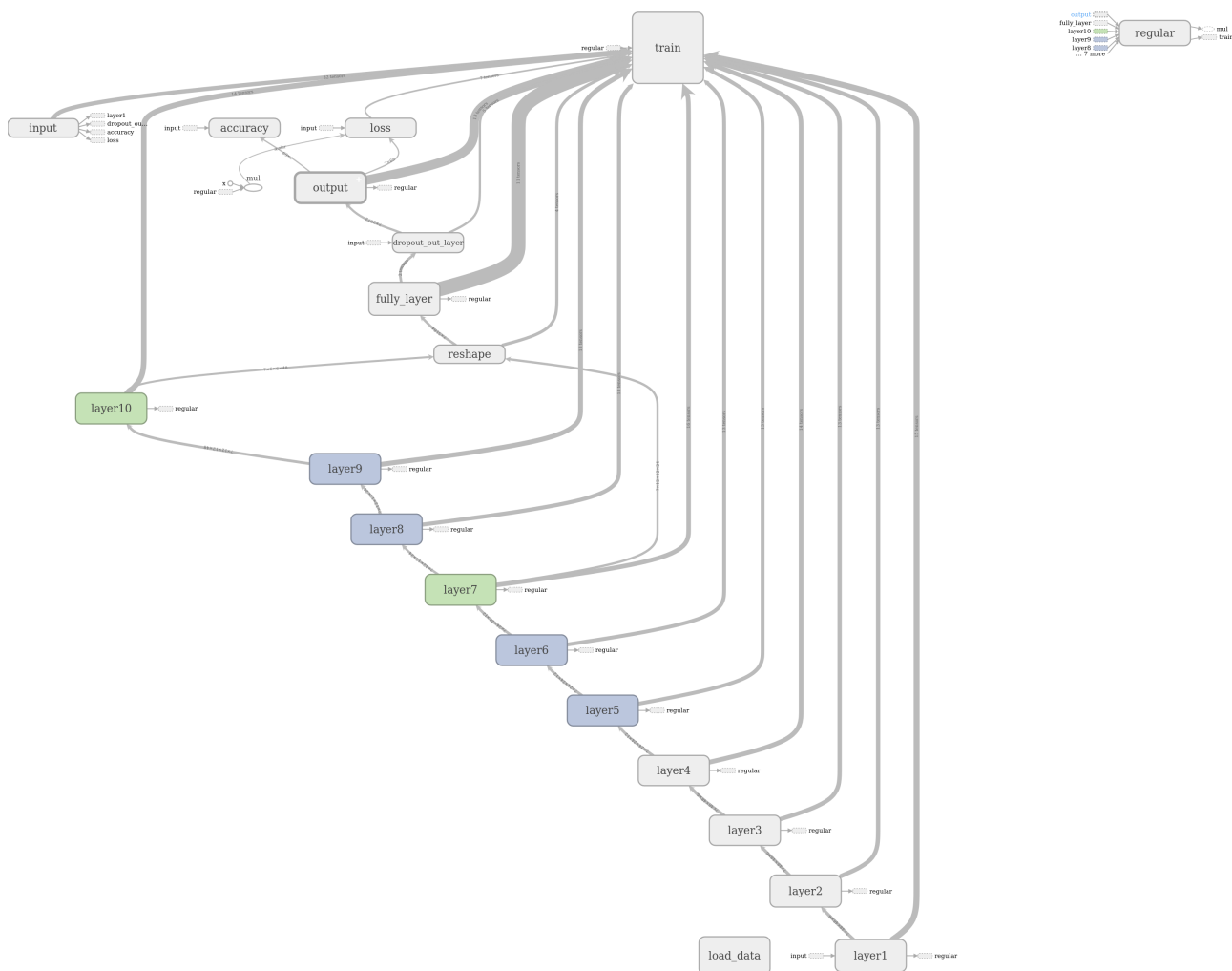
对于已经知道网络实现原理的读者, 下面可以不用看了, 如果想要了解各个部分实现原理可以继续阅览.

缘由:

上学期加入了实验室(实习生), 刚好期末时一门课程大作业为实现班级的人脸识别. 当时使用tensorflow框架完成, 总感觉意犹未尽, 又想加深一下理解, 顺便把落下了近一年的c++给捡起来, 于是决定假期回去使用C++手写一遍. 可确实没想到这么困难, 一直到前几天才结束. (感谢写框架还开源的各位大大!!!)

网络结构:

我采用了最简单的网络结构, 类似与VGG, 单纯的卷积, 激活, 池化的交替进行. 只是在最后的全连接层组合了两级的特征(最后一层卷积和第七层卷积). 由于是完全复现了使用的tensorflow的版本的网络, 所以直接贴当时的图了.



(注意:上图基本试用,但不完全相同,在手写实现时我主要是为了搞清楚各个部的原理及实现,并未添加对权重系数的正则项,读者要是想添加也十分简单,只需要在含有权值的层(修改类)添加一个正则系数,在进行权值优化时减去对应系数一半的权值即可)

各个层的原理:

卷积层:

卷积层实现了高度的参数共享,并使得网络对于平移具有不变形.可以看做一个极其强的先验(网络的整体结构都可看做一个极强的先验,即用户相信这样的网络可以实现要求).

前向传播:

$A^l = Z^{l-1} * W^l + B^l$ (其中 Z^{l-1} 为上一层的输出, W^l 和 B^l 分别为当前层的卷积核和偏置,对于小批量而言, Z, W应为四维, B为一维(对应于输出通道)). (Z四维解释:第一维: 图片数量; 第二维, 第三维: 图片尺寸. 第四维: 通道数.W四维解释: 第一维,第二维: 卷积核尺寸(常用 $3 * 3, 5 * 5$); 第三维:输入通道数(即Z的通道数); 第四网维:输出通道,自己设计, 决定网络宽度.b仅一维, 对应于w的输出通道, 即第四维.) 使用二维张量简化表示如下

$$A = \begin{pmatrix} z_{00} & \cdots & z_{0n} \\ \vdots & \ddots & \vdots \\ z_{m0} & \cdots & z_{mn} \end{pmatrix} * \begin{pmatrix} w_{00} & \cdots & w_{0x} \\ \vdots & \ddots & \vdots \\ w_{y0} & \cdots & w_{yx} \end{pmatrix} + b$$
 卷积运算大家应该都知道(否则自行百度). 为了保证卷积后输入输出尺寸一致, 关

键在于填充. 我采用的是0填充方式, 并且默认卷积核尺寸为奇数, 此时对外填充数量为卷积核尺寸的一半(向下取整). 比如 3×5 的卷积核, 则x方向(二维矩阵第一维)向外填充一个0, y方向(对应矩阵第二维)向外填充两个0即可. 注意: 填充并不会影响反向传播, 具体接下来会详细介绍. 此时即完成了前向传播.

反向传播:

在解释反向传播之前, 我们应该先了解一个求导原则. 若 $J = f(y_1, y_1, \dots, y_n, \theta)$ 且 $y_i = g(x, \theta_i)$ 则有 $\frac{\partial J}{\partial x} = \sum_{i=1}^n \frac{\partial J}{\partial y_i} * \frac{\partial y_i}{\partial x}$ 还有一个则是反向传播的链式求导法则, 比较简单, 不明白可以百度. 同时计算反向传播时, $\frac{\partial J}{\partial A^l}$ 应该已经在之前的反向传播中算出.

计算 $\frac{\partial J}{\partial W^l}$

$\frac{\partial J}{\partial W^l} = \frac{\partial J}{\partial A^l} * \frac{\partial A^l}{\partial W^l}$, 对单独的 w_{ij}^l 分析, 有 $\frac{\partial J}{\partial w_{ij}^l} = \sum_{pq} \frac{\partial J}{\partial a_{pq}^l} * \frac{\partial a_{pq}^l}{\partial w_{ij}^l}$ 其中 p, q 为 A 内有 w_{ij}^l 部分的元素索引, 根据卷积定义(且滑动均为1)可以得对于每个 w_{ij}^l 具有乘积关系的元素个数均与 Z^{l-1} 所含元素个数相同, 对于处于中间的 w_{ij}^l , 恰好是整个 Z^{l-1} , 其余的 w_{ij}^l 取决于其对中间位置的偏移, 对应的则是将 Z^{l-1} 填充后从中心进行对应偏移后的与 Z^{l-1} 大小相同的块. 所以得到 $\frac{\partial J}{\partial W^l} = Z^{l-1} * \frac{\partial J}{\partial A^l}$ 其中也首先对 Z^{l-1} 进行与前向传播时一样的扩充. 这是由于前向传播时对 Z^{l-1} 进行了扩充, 对应零的位置应该在反向传播时依然保留, 零在前向传播中进行了乘积的运算, 反向传播依然要计算. 这就是之前所述填充不影响反向传播.

计算 $\frac{\partial J}{\partial B^l}$

由前向传播公式可得, 所以 A^l 中的元素均与 b 有关, 又 b 系数为1, 所以 $\frac{\partial J}{\partial B^l} = \frac{\partial J}{\partial A^l} * 1 = \sum_{p,q} \frac{\partial J}{\partial a_{pq}^l}$, 即 $\frac{\partial J}{\partial B^l}$ 为所有 $\frac{\partial J}{\partial A^l}$ 元素和.

计算 $\frac{\partial J}{\partial Z^{l-1}}$

$\frac{\partial J}{\partial Z^{l-1}} = \frac{\partial J}{\partial A^l} * \frac{\partial A^l}{\partial Z^{l-1}}$, 同样对于单个元素进行分析, $\frac{\partial J}{\partial z_{ij}^{l-1}} = \sum_{p,q} \frac{\partial J}{\partial a_{pq}^l} * \frac{\partial a_{pq}^l}{\partial z_{ij}^{l-1}}$ 其中 $\{p,q\}$ 为 A 内有 z_{ij}^l 部分的元素索引. 由 $A = \begin{pmatrix} z_{00} & \dots & z_{0n} \\ \vdots & \ddots & \vdots \\ z_{m0} & \dots & z_{mn} \end{pmatrix} * \begin{pmatrix} w_{00} & \dots & w_{0x} \\ \vdots & \ddots & \vdots \\ w_{y0} & \dots & w_{yx} \end{pmatrix} + b$ 可以看出 A 内有 z_{ij}^l 部分的元素. 大部分元素都是 W 内的全部元素, 少部分(边缘)是 W 的一部分元素. 且应当注意, 第一个与 z_{ij}^l 相乘的元素为 W 的右下角的元素, 最后一个与 z_{ij}^l 相乘的元素为 W 的左上角的元素, 而相乘后的位置由 W 中心的位置决定(可以看做 z_{ij}^l 不动, W 的元素从右下角到左上角依次划过 z_{ij}^l). 此时得到的与 z_{ij}^l 有关区域与 W 的大小一致, 不过对应区域中与 W 有关的元素却与 W 内本身的元素发生了180度的旋转. 于是可以得到 $\frac{\partial J}{\partial Z^{l-1}} = \frac{\partial J}{\partial A^l} * \text{rot180}(W^l)$ 同样首先对 $\frac{\partial J}{\partial A^l}$ 进行与前向传播时一样的扩充.

全连接层

全连接层即是矩阵相乘, 相较于卷积来说也不用扩充, 相比之下简单横多. 推导过程大家可以模仿卷积层的方式, 这里仅仅给出最终结果.

前向传播

$$A^l = Z^{l-1} W^l + b$$

反向传播

计算 $\frac{\partial J}{\partial W^l}$

$$\frac{\partial J}{\partial W^l} = Z^{l-1T} \frac{\partial J}{\partial A^l}$$

计算 $\frac{\partial J}{\partial B^l}$

$$\frac{\partial J}{\partial B^l} = \frac{\partial J}{\partial A^l} * 1 = \sum_{p,q} \frac{\partial J}{\partial A^l}$$

计算 $\frac{\partial J}{\partial Z^{l-1}}$

$$\frac{\partial J}{\partial W^l} = \frac{\partial J}{\partial A^l} W^{lT}$$

ReLU层

ReLU层比较简单, 前向传播时记住大于0的位置即可, 前向传播时将小于0的位置置0, 反向传播时将上一层反向传播的输出乘以记录的矩阵即可. 即只需要将前向传播中置零的位置再次置零, 其余位置不变即可.

MaxPooling层

池化层可以有不同的选择, 主要有最大值池化和均值池化. 这里我使用的是最大值池化. 这里为了简化考虑, 我默认用户输入满足最大值池化的核尺寸与步长恰好满足约束条件, 不需要零填充. 池化同样可以帮助表示输入的平移不变性. 同时池化是一个降采样的过程, 可以增加之后网络的感受野.

前向传播

对于给的核的大小, 选出最大的作为下一层的参数, 同时记录选取的元素的位置.

反向传播 $\frac{\partial J}{\partial Z^{l-1}}$

将上一层的反向传播的输出作为输入, 将其映射回对应的前向传播的输入尺寸, 利用前向传播记录的每个元素的位置进行还原, 未被记录的位置则为0

Reshape层

这一层也比较简单, 前向传播为将输入的四维张量转换为二维向量, 第一维为batch大小, 第二为对应每张图片提取出来的特征. 反向传播相当于前向传播的逆运算, 将输入的二维张量还原成四维张量.

Dropout层

一般在全连接层的后面, SVM层(也是一个全连接层)的前面都会有一个dropout层(当然卷积后亦可以使用, 只不过比较少见), dropout可看做一个正则化的操作, 减少模型的过拟合. 对于dropout能够起到作用的理论解释目前还不完善, 但使用的效果确实很好. 一个比较非正式的解释是在最终分类时减少了提取出的特征数量, 以此来训练模型以保证提取出来的特征更加有效(训练时减去一部分特征, 要求网络依然能够正常工作, 则要求剩余部分特征足够好, 而每次减去的特征位置都不一样, 这就要求所有的特征都要足够好, 以此达到训练目的). 注意, 虽然会减少特征值的数量, 但特征值总体大小应该保持不变(保证训练与测试时一致). dropout在验证和测试时不需要.

前向传播

对于输入的一个k(float)数, 我们看做其将整个网络保留特征比例. 由于直接想要保留对应比例实现相对复杂(至少我是不知道咋样确保比例又要随机), 所以我对每一个特征实施dropout即对每一个点随机生成一个[0, 1]之间的小数, 如果小于k, 则该数除以k作为输出, 否则置零. 由概率论的知识可以知道, 这样对于整个提取的特征保留比例应该接近k, 当提取的特征数目较多时, 偏差应该不会太多. 在保留与置零时同样要进行记录, 用于反向传播时使用.

反向传播

利用前向传播时的记录, 记录的位置除以k即可, 未记录的位置置零.

拼接层

当考虑到不同尺度的特征时, 我们可以将网络不同深度提取出来的特征进行拼接, 一起作为最终预测的参数, 或者输入有多个数据, 提取出所以数据的特征后进行拼接, 之后来进行预测. 该层十分简单, 只是简单的将两个矩阵拼接即可, 前向传播反向传播都不困难. 前向传播将不同特征拼接到一起, 反向传播将更高维的导数拆分还原到不同位置. 需要注意的是, 拆分完之后的几部分都要进行反向传播, 直到几部分最终有合并到同一层(合并到同一层时两部分的导数相加), 或者均到达各自网络的起点为止.

Softmax层

公式: $\text{softmax}(Z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

对于损失函数使用交叉熵, 交叉熵中的 \log 函数可以使 softmax 中的指数抵消掉, 利于训练(具体解释在loss函数部分). 需要注意, softmax 并不是所以时候都适用, 主要用在分类且损失函数为交叉熵的情况下. 交叉熵之外的很多损失函数都不适用于 softmax . 具体来说, 不使用对数来抵消 softmax 中的指数的损失函数都不适用 softmax , 这是由于当指数函数的变量取值十分小的负值时会造成梯度的消失, 从而无法学习. 当输入值差异变得极端时, 极端大大的值的输出饱和到一, 极端小的值饱和到0, 此时很多基于 softmax 的代价函数也会饱和, 除非代价函数能够转化饱和的激活函数(如交叉熵).

前向传播

$\text{softmax}(Z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$ 对之前输出的每个元素按照此公式转化.

反向传播

$\frac{\partial J}{\partial z_i} = \sum_j \frac{\partial J}{\partial \text{softmax}(Z)_j} \frac{\partial \text{softmax}(Z)_j}{\partial z_i}$ 其中 $\frac{\partial J}{\partial \text{softmax}(Z)_j}$ 为更高层的损失函数反向传播时已经求出来的, 主要问题就是求解 $\frac{\partial \text{softmax}(Z)_j}{\partial z_i}$, 根据前向传播公式, 此处求导也不困难. 不过需要分开讨论.

当 $i = j$ 时:

$$\frac{\partial \text{softmax}(Z)_j}{\partial z_i} = \frac{e^{z_i} * (\sum_j e^{z_j}) - e^{z_i} * e^{z_i}}{(\sum_j e^{z_j})^2} = \text{softmax}(Z)_i - \text{softmax}(Z)_i^2$$

当 $i \neq j$ 时:

$$\frac{\partial softmax(Z)_j}{\partial z_i} = \frac{-e^{z_i} * e^{z_j}}{(\sum_j e^{z_j})^2} = -softmax(Z)_i * softmax(Z)_j$$

之后再用上述公式即可。

损失函数

如前文所述, 我采用的是交叉熵损失函数. 对于交叉熵损失的解释如下(才疏学浅, 有很多错误望大佬们多多赐教): 由于整个网络的结构是一个无穷大的先验, 我们认为这个结构最终能够实现我们的分类目标. 因此, 对于网络的输出, 我们希望它们都是正确的, 因此采用最大似然估计的策略来不断优化网络中的参数(最大似然估计详见概率论). 正如我们在学习概率论中最大似然估计时一样, 为了简化计算, 我们会对乘积取 \log 以此来简化计算. 损失函数中也依然如此, 更重要的是取 \log 之后我们可以将预测的不同类别进行拆分. 然而, 虽然我们希望网络输出是正确的, 但并不一定会如此(训练前期往往不会), 而我们利用取 \log 之后把想要优化的部分进行了拆分, 这时, 真正的标签就有作用了, 如果是单纯的极大似然估计, 会将所有的预测均进行优化, 错误的同样进行, 最后相当于没有优化, 而这时对于每一部分乘以一个正确标签概率(权值, 一般正确的位置是1, 错误是0), 我们就能够知道真正该对哪一部分进行优化, 这里同样可以解释为何 $softmax$ 一般与交叉熵一同使用. $\log(softmax(Z)_i) = z_i - \log(\sum_j e^{z_j})$ 这时就可以将 $softmax$ 中的取指数给抵消掉. 同时, 我们一般要最小化损失函数, 因此再在前面乘以负一.

前向传播

$$loss = - \sum_j y_j^{lab} \log(y_j^{pre}) \text{ 其中 } y_j^{lab} \text{ 为标签, } y_j^{pre} \text{ 为网络预测.}$$

上式整体没有问题, 但是在电脑上运行时可能会报错, 主要原因是当某一个预测值十分小时, 电脑精度达不到, \log 内可能是0, 这是非法的, 因此为了防止这样的事情发生, 应该在内部再添加一个十分小的数来避免非法运算. 所以最终公式为

$$loss = - \sum_j y_j^{lab} \log(y_j^{pre} + 1e - 8)$$

反向传播

反向传播比较简单, 直接求导即可, 结果为:

$$\frac{\partial loss}{\partial y_j^{pre}} = -y_j^{lab} \frac{1}{y_j^{pre} + 1e - 8}$$

优化

我选取的优化算法是能够自适应调整学习率的 $Adam$, 由于能力有限, 对这个函数原理的知识讲解大家自己百度吧. 算法如下:

Require: 步长 ϵ (建议步长: 0.001)

Require: 矩估计的指数衰减速率 ρ_1 和 ρ_2 在区间 $[0,1)$ 内.(建议默认为: 分别为0.9和0.99)

Require: 用于数值稳定的小常数 δ (建议默认: $1e-8$)

Require: 初始参数 θ

初始化一阶和二阶矩变量 $s = 0, r = 0$

初始化时间步 $t = 0$

while 没有达到终止条件do

 从训练集中采包含 m 的样本 $x^{(1)}, \dots, x^{(m)}$ 的小批量, 对应标签为 $y^{(i)}$.

 计算梯度: $g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x^{(i)}, \theta), y^{(i)})$

$t \leftarrow t + 1$

 更新有偏一阶矩估计: $s \leftarrow \rho_1 s + (1 - \rho_1)g$

 更新有偏二阶矩估计: $s \leftarrow \rho_2 s + (1 - \rho_2)g \odot g$

 修正一阶矩的偏差: $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$

 修正二阶矩的偏差: $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$

 计算更新: $\Delta\theta \leftarrow -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$ (逐元素操作)

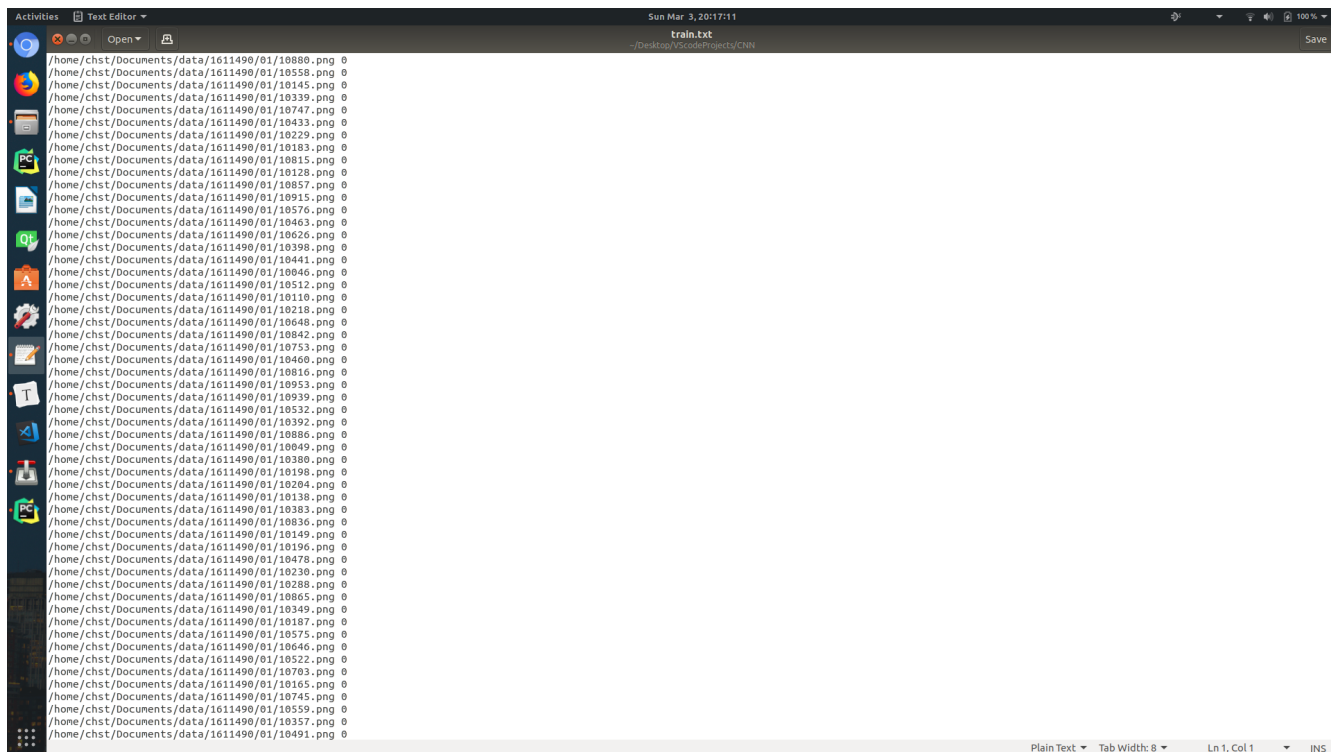
 应用更新: $\theta \leftarrow \theta + \Delta\theta$

end while

其它问题

数据集

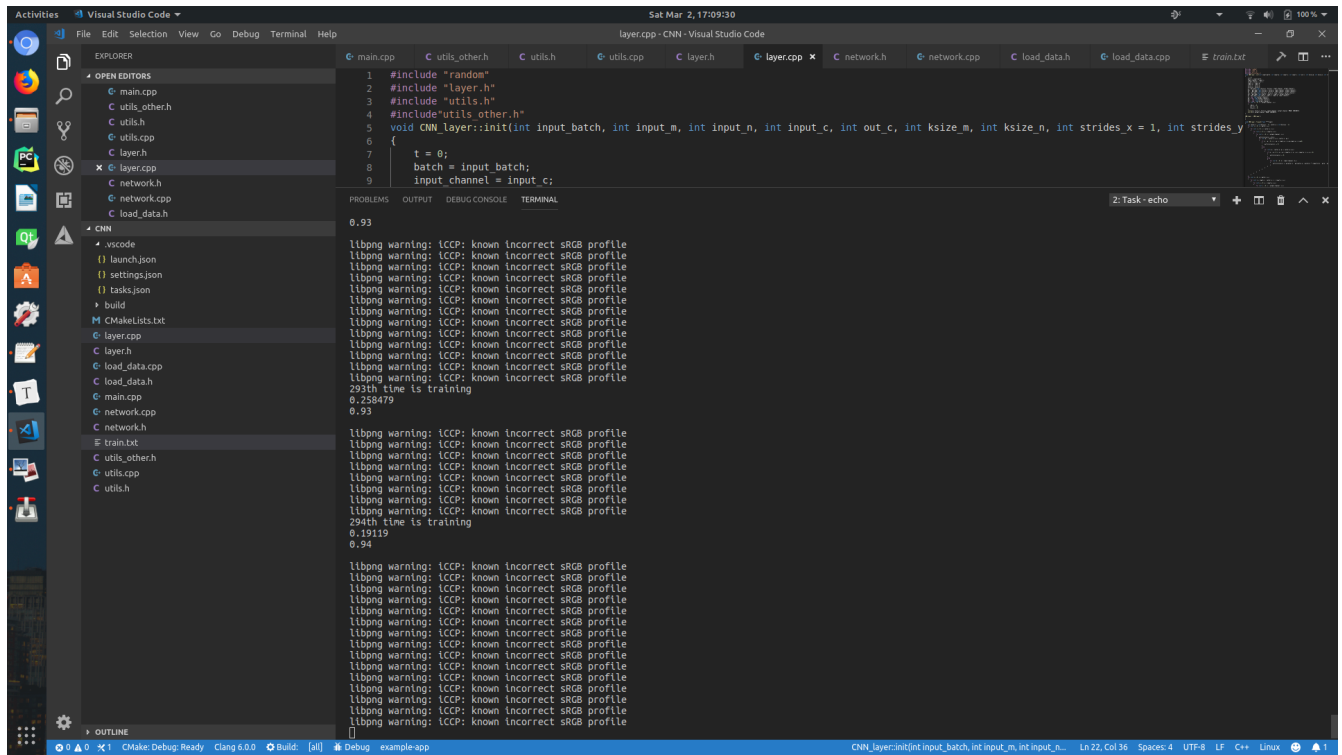
我的数据集为班级同学照片, 先生成一个txt文件用于保存图片路径和对应标签. 训练时直接读txt内文件路径, 使用opencv加载图片. 我的txt文件这样的:



其次，在读取数据之后一定要将数据打乱，否则训练效果奇差。

代码中的问题

我的代码中卷积层初始化中有提供步长的参数,但实际并没有用, 代码实际步长都是1, 太懒不想删了. 读者可以在自己的代码中删去或者完善使得可以更改步长. 同时, 我的代码还有一个优化算法即含有动量的随机梯度下降. 我自己并没有试效果如何, 所以前文并未介绍(但代码肯定没问题). 还有最重要的一点, 这份代码完全使用CPU进行运算, 所以会十分慢, 建议大家尝试时提供一个比较小的数据集. 我的数据集是1500张照片, 运行二两小时左右才能有一个不错的结果. 当然你可以尝试将代码迁移为使用CUDA加速的版本(作者不会,正在学习). 同时, 这份代码只是为了理解卷积网络内部是如何工作的, 并未追求十分好的效果.因此也没有验证集与测试集. 所以最终效果如下图.



最后, 毕竟才疏学浅, 文章或代码中可能有很多错误, 大家发现希望能够联系我, 即时改正. 当然大家如果有什么问题也可以联系我哦.

联系方式：

QQ: 2322253097

电话: 15222125166

email: 1611447@mail.nankai.edu.cn

本文主要参考:

深度学习 [美]伊恩·古德费洛等

<https://www.cnblogs.com/pinard/p/6494810.html>