Physcalの大魔導書

某HFUT的蒟蒻，ICT/VIPL的直博狗，SeetaTech的码农，还是当大魔导师好了(=￣ω￣=)。

Spatial Transformer Networks(空间变换神经网络)

Reference：Spatial Transformer Networks [Google.DeepMind]
Reference：[Theano源码，基于Lasagne]

闲扯：大数据不如小数据

这是一份很新的Paper(2015.6)，来自于Google旗下的新锐AI公司DeepMind的四位剑桥Phd研究员。

他们针对CNN的特点，构建了一个新的局部网络层，称为空间变换层，如其名，它能将输入图像做任意空间变换。

在我的论文 [深度神经网络在面部情感分析系统中的应用与改良] 中，提出了一个有趣观点：

大数据不如小数据，如果大数据不能被模型有效利用。

该现象是比较常见的，如ML实战的一个经典问题：数据不均衡，这样模型就会对大类数据过拟合，忽略小类数据。

另外，就是 [Evolving Culture vs Local Minima：文化、进化与局部最小值] 提到的课程学习观点：

将大数据按照难易度剖分，分批学习，要比直接全部硬塞有效得多。

当前，我们炙手可热的模型仍然是蒟蒻的，而数据却是巧夺天工、超乎想象的。

因而，想要通过模型完全摸清数据的Distribution是不现实的，发明、改良模型结构仍然是第一要务，

而不单纯像Li Feifei教授剑走偏锋，用ImageNet这样的大数据推进深度学习进程。

空间变换的重要意义

在我的论文 [深度神经网络在面部情感分析系统中的应用与改良] 中，分析了CNN的三个强大原因：

[局部性]、**[平移不变性]**、**[缩小不变性]**，还对缺失的**[旋转不变性]**做了相应的实验。

这些不变性的本质就是图像处理的经典手段，**[裁剪]**、**[平移]**、**[缩放]**、**[旋转]**。

这些手段又属于一个家族：空间变换，又服从于同一方法：坐标矩阵的仿射变换。

那么，神经网络是否有办法，用一种统一的结构，自适应实现这些变换呢？DeepMind用一种简易的方式实现了。

图像处理技巧：仿射矩阵、逆向坐标映射、双线性插值

1.1 仿射变换矩阵

实现**[裁剪]**、**[平移]**、**[缩放]**、**[旋转]**，只需要一个$[2,3]$的变换矩阵：

$$\begin{bmatrix}\theta_{11} & \theta_{12} & \theta_{13}\\ \theta_{21} & \theta_{22} & \theta_{23}\end{bmatrix}$$

对于平移操作，坐标仿射矩阵为：

$$\begin{bmatrix}1 & 0 & \theta_{13}\\ 0 & 1 & \theta_{23}\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix}=\begin{bmatrix}x+\theta_{13}\\ y+\theta_{23}\end{bmatrix}$$

对于缩放操作，坐标仿射矩阵为：

$$\begin{bmatrix}\theta_{11} & 0 & 0\\ 0 & \theta_{22} & 0\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix}=\begin{bmatrix}\theta_{11}x\\ \theta_{22}y\end{bmatrix}$$

对于旋转操作，设绕原点顺时针旋转α度，坐标仿射矩阵为：

$$\begin{bmatrix}cos(\alpha) & sin(\alpha) & 0\\ -sin(\alpha) & cos(\alpha) & 0\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix}=\begin{bmatrix}cos(\alpha)x+sin(\alpha)y\\ -sin(\alpha)x+cos(\alpha)y\end{bmatrix}$$

公告

这是一个属于轻松描写魔导书平凡日常的故事，请不要过度期待。还有，请保持屋内明亮离开电视3米以上再观看。(=￣ω￣=)

昵称：Physcal
园龄：3年6个月
粉丝：427
关注：19
+加关注

最新随笔

1. [深度学习大讲堂]从NN...
2. [深度学习大讲堂]文化、...
3. 前馈网络求导概论(一)·S...
4. 从零开始山寨Caffe·拾贰...
5. 从零开始山寨Caffe·拾：...
6. 从零开始山寨Caffe·玖：...
7. 从零开始山寨Caffe·捌：...
8. 从零开始山寨Caffe·柒：...
9. 从零开始山寨Caffe·陆：...
10. 从零开始山寨Caffe·伍...

随笔分类(164)

ACM(113)
Haskell(3)
Qt(1)
并行计算(3)
机器学习理论(28)
机器学习系统设计(12)
模式识别(4)

随笔档案(163)

2016年12月 (1)
2016年6月 (2)
2016年3月 (8)
2016年2月 (5)
2015年11月 (1)
2015年10月 (1)
2015年9月 (2)
2015年8月 (7)
2015年7月 (1)
2015年6月 (8)
2015年5月 (19)
2015年4月 (3)
2015年3月 (7)

这里有个trick，由于图像的坐标不是中心坐标系，所以只要做下Normalization，把坐标调整到[-1,1]。

这样，就绕图像中心旋转了，下文中会使用这个trick。

至于裁剪操作，没有看懂Paper的关于左2x2 sub-matrix的行列式值的解释，但可以从坐标范围解释：

只要$x^{'}$ 、$y^{'}$ 的范围比x ，y 小，那么就可以认为是目标图定位到了源图的局部。

这种这种仿射变换没有具体的数学形式，但肯定是可以在神经网络搜索过程中使用的。

1.2 逆向坐标映射

注：感谢网友@载重车提出疑问，修正了这部分的内容。具体请移步评论区。

★本部分作为一个对论文的错误理解，保留。

在线性代数计算中，一个经典的求解思路是：

$$\begin{bmatrix}\theta_{11} & \theta_{12} & \theta_{13}\\ \theta_{21} & \theta_{22} & \theta_{23}\end{bmatrix}\begin{bmatrix}x^{Source}\\ y^{Source}\\ 1\end{bmatrix}=\begin{bmatrix}x^{Target}\\ y^{Target}\end{bmatrix}$$

这种做法在做图像处理时，会给并行矩阵程序设计造成尴尬——需要牺牲额外的空间存储映射源，：

由于(x^{Target},y^{Target}) 必然是离散的，当我们需要得到$Pixel(x^{Target},y^{Target})$ 的值时，

如果不及时保存(x^{Source},y^{Source}) ，那么就必须即时单点复制$Pixel(x^{Source},y^{Source})->Pixel(x^{Target},y^{Target})$

显然，这种方法的实现依赖于For 循环：

$$\begin{array}{l}For(0....i....Height)\\ \quad For(0....j....Width)\\ \quad\quad Calculate\&Copy\end{array}$$

为了能让矩阵并行计算成为可能，我们需要逆转一下思路：

$$\begin{bmatrix}\theta_{11} & \theta_{12} & \theta_{13}\\ \theta_{21} & \theta_{22} & \theta_{23}\end{bmatrix}^{'}\begin{bmatrix}x^{Target}\\ y^{Target}\\ 1\end{bmatrix}=\begin{bmatrix}x^{Source}\\ y^{Source}\end{bmatrix}$$

之后，构建变换目标图就转化成了，数组下标取元素问题：

$PixelMatrix^{Target}=PixelMatrix^{Source}[x^{Source},y^{Source}]$

这依赖于仿射矩阵的一个性质：

$$\begin{bmatrix}\theta_{11} & \theta_{12} & \theta_{13}\\ \theta_{21} & \theta_{22} & \theta_{23}\end{bmatrix}^{'}=\begin{bmatrix}\theta_{11} & \theta_{12} & \theta_{13}\\ \theta_{21} & \theta_{22} & \theta_{23}\end{bmatrix}^{-1}$$

即，由Target变换为Source时，新仿射矩阵为源仿射矩阵的逆矩阵。

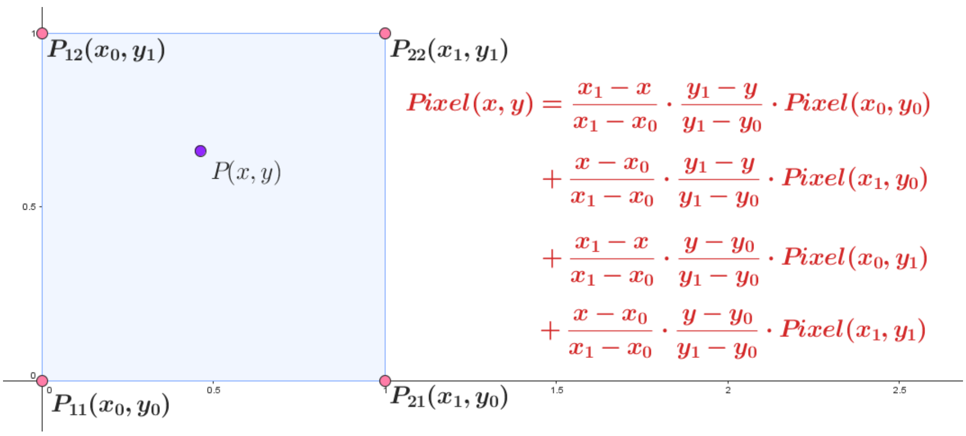
1.3 双线性插值

考虑一个$[1,10]$ 图像放大10倍问题，我们需要将10个像素，扩展到为100的数轴上，整个图像应该有100个像素。

但其中90个对应Source图的坐标是非整数的，是不存在的，如果我们用黑色$(RGB(0,0,0)$)填充，此时图像是惨不忍睹的。

所以需要对缺漏的像素进行插值，利用图像数据的局部性近似原理，取邻近像素做平均生成。

双线性插值是一个兼有质量与速度的方法(某些电子游戏里通常这么排列：线性插值、双线性插值....)：



如果(x^{Source},y^{Source}) 是实数坐标，那么先取整(截尾)，然后沿轴扩展d 个坐标单位，得到P_{21} 、P_{12} 、P_{22}

一般的(源码中)，取$d=1$ ，式中分母全被消去，再利用图中双线性插值式进行插值，得到$Pixel(x^{Source},y^{Source})$ 的近似值。

神经网络

2.1 块状神经元

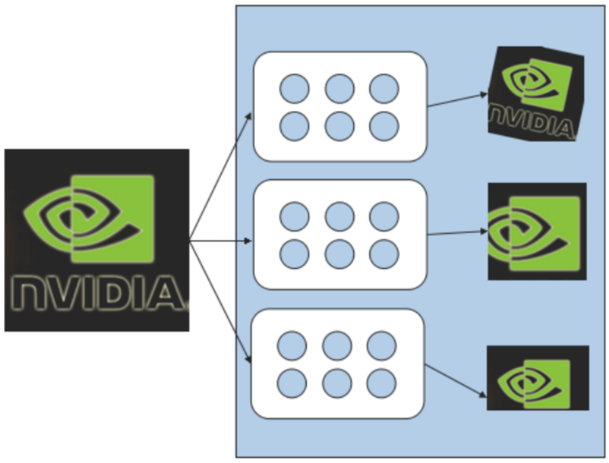
CNN是一个变革的先驱者模型，它率先提出局部连接观点，减少网络广度，增加网络深度。

队友の魔導書

esxgx
MaticsL
Pentium
战亿熊猫

局部连接让神经元呈块状，单参数成参数组；让网络2D化，切合2D图像；让权值共享，大幅度减少参数量。

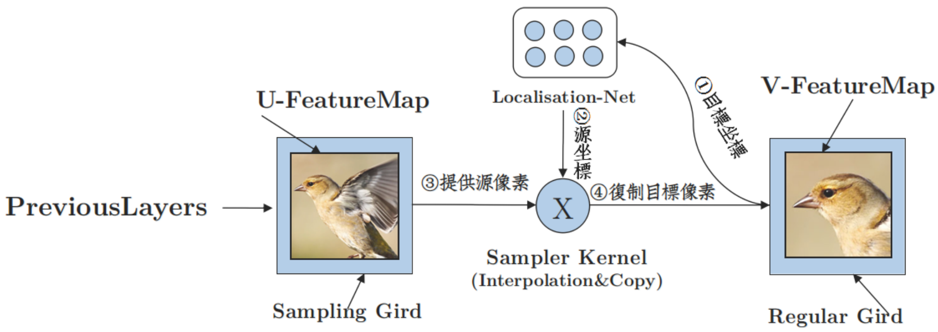
仿射矩阵自适应学习理论，因此而得以实现：



將仿射矩陣作爲神經元嵌入到神經网絡中

2.2 基本结构与前向传播

论文中的结构图描述得不是很清楚，个人做了部分调整，如下：



★★★空間變換層的前向傳播机制，①②③④依次執行

DeepMind为了描述这个空间变换层，首先添加了坐标网格计算的概念，即：

对应输入源特征图像素的坐标网格——Sampling Grid，保存着(x^{Source}, y^{Source})

对应输出源特征图像素的坐标网格——Regluar Grid ，保存着(x^{Target}, y^{Target})

然后，将仿射矩阵神经元组命名为定位网络 **(Localisation Network)**。

对于一次神经元提供参数，坐标变换计算，记为 $\tau_\theta(G)$ ，根据1.2，有：

$$\tau_\theta(G_i) = \begin{bmatrix}\theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23}\end{bmatrix}' \cdot \begin{bmatrix}x_i^{Target} \\ y_i^{Target} \\ 1\end{bmatrix} = \begin{bmatrix}x_i^{Source} \\ y_i^{Source}\end{bmatrix} \quad where \quad i = 1,2,3,4..,H*W$$

该部分对应于图中的①②，但是与论文中的图有些变化，可能是作者并没有将逆向计算的Trick搬到结构图中来。

所以你看到的仍然是Sampling Grid提供坐标给定位网络，而具体实现的时候恰好是相反的，坐标由Regluar Grid提供。

——

Regluar Grid提供的坐标组是顺序逐行扫描坐标的序列，序列长度为 $[Heght * Width]$ ，即：

将2D坐标组全部1D化，根据在序列中的位置即可立即算出，在Regluar Grid中位置。

这么做的最大好处在于，无须额外存储Regluar Grid坐标(x^{Target}, y^{Target}) 。

因为从输入特征图U 数组中，按下标取出的新像素值序列，仍然是逐行扫描顺序，简单分隔一下，便得到了输出特征图V 。

该部分对应于图中的③。

——

(1.3)中提到了，直接简单按照(x^{Source}, y^{Source}) ，从源像素数组中复制像素值是不可行的。

因为仿射变换后的(x^{Source}, y^{Source}) 可以为实数，但是像素位置坐标必须是整数。

为了解决像素值缺失问题，必须进行插值。插值核函数很多，源码中选择了论文中提供的第二种插值方式——双线性插值。

(1.3)的插值式非常不优雅，DeepMind在论文利用max与abs函数，改写成一个简洁、优雅的插值等式：

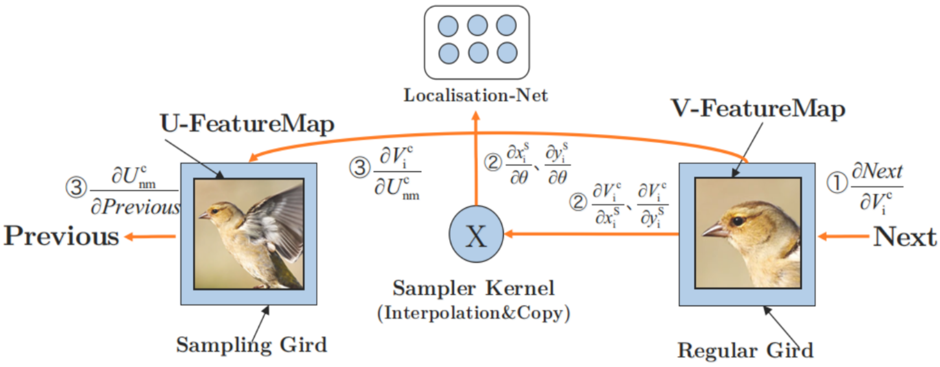
$V_i^c=\sum_n^H\sum_m^W U_{nm}^c\max(0,1-|x_i^S-m|)\max(0,1-|y_i^S-n|)\quad where\quad i\in[1,H^{'}W^{'}],c\in[1,3]$

两个 $\sum$ 实际上只筛出了四个邻近插值点，虽然写法简洁，但白循环很多，所以源码中选择了直接算4个点，而不是用循环筛。

该部分对应图中的④。

2.3 梯度流动与反向传播

添加空间变换层之后，梯度流动变得有趣了，如图：



★★★空間變換層反向傳播梯度流動，①②③代表分支流

形成了三股分支流：

(I)后の流：

$ErrorGradient\rightarrow.....\rightarrow\frac{\partial Next}{\partial V_i^c}$

这是Back Propagation从后层继承的动力源泉，没有它，你就不可能完成Back Propagation。

(II)里の流：

$$\begin{cases}\frac{\partial V_i^c}{\partial x_i^S}\rightarrow\frac{\partial x_i^S}{\partial\theta}\\\frac{\partial V_i^c}{\partial y_i^S}\rightarrow\frac{\partial y_i^S}{\partial\theta}\end{cases}$$

个人对这股流的最好描述就是：一江春水流进了小黑屋。

是的，你没有看错，这股流根本就没有流到网络开头，而是在定位网络处就断流了。

由此来看，定位网络就好像是在主网络旁侧偷建的小黑屋，是一个违章建筑。

所以也无怪乎作者说，定位网络直接变成了一个回归模型，因为更新完参数，流就断了，独立于主网络。

(III)前の流：

$\frac{\partial V_i^c}{\partial U_{nm}^i}\rightarrow\frac{\partial U_{nm}^i}{\partial Previous}$

这是Back Propagation传宗接代的根本保障，没有它，Back Propagation就断子绝孙了。

2.4* 局部梯度

论文中多次出现**[局部梯度]**(Sub-Gradient) 的概念。

作者们反复强调，他们写的，优雅简洁的采样核函数，是不连续的，不能如下直接求导：

$g=\frac{\partial V_i^c}{\partial\theta}$

而应该是分两步，先对x_i^S 、x_i^S 求局部梯度: $\frac{\partial V_i^c}{\partial x_i^c}$ 、 $\frac{\partial V_i^c}{\partial y_i^c}$ ，后有：

$$\begin{cases}g=\frac{\partial V_i^c}{\partial x_i^S}\cdot\frac{\partial x_i^S}{\partial\theta}\\g=\frac{\partial V_i^c}{\partial y_i^S}\cdot\frac{\partial y_i^S}{\partial\theta}\end{cases}$$

有趣的是，对于Theano这种自动求导的Tools，局部梯度可以直接被忽视。

因为Theano的Tensor机制，会聪明地讨论并且解离非连续函数，追踪每一个可导子式，即便你用了作者们的优雅的采样函数，

Tensor.grad函数也能精确只对筛出的4个点求导，所以在Theano里讨论非连续函数和局部梯度，是会被贻笑大方的。

分类: 机器学习理论 , 模式识别

+加关注

« 上一篇： PRML读书后记(一): 拟合学习
» 下一篇： 关于过拟合、局部最小值、以及Poor Generalization的思考

评论列表

#1楼 2015-11-09 12:58 载重车

还是不太理解这个思路“为了能让矩阵并行计算成为可能，我们需要逆转一下思路：”

支持(0) 反对(0)

#2楼 [楼主] 2015-11-09 14:21 Physcal

@ 载重车
就是说，如果你正向用Source坐标转换，(1,1)这个Source点，可能对应着(233,233)这个Target点。我们写代码的时候，大可设一个二维数组。Target[233][233]=Tuple(1,1)。现在我们要构建Target这个图，Target(233,233)就可以用O(1)复杂度直接取出来了，因为已经我们打
了表了。这是一个离线算法。要是不用数组呢？那么算出(233,233)点的时候，就要立刻复制Source(1,1)像素到Target(233,233)像素，不然你复制的时候，还要重算一遍。这是一个强制在线算法，在正向转换的时候，你只能用For循环实现，实时在线处理。

但如果你用逆向转换，即便不额外打表一个二维数组，我们也可以用PixelSource加上结果矩阵作为索引，O（1）直接得到像素。因为这时候
，结果矩阵相当于一个隐形的记录表。这样的离线处理是非常赞的。

综上，逆向转换的是时间复杂度和空间复杂度最低的实现，非常适合矩阵并行计算。因为它不用在计算时，开额外的空间辅助存储，也不用For循环。如果你学过矩阵并行计算，就会知道，矩阵并行计算是分割小矩阵-合并的整体过程，它的实际运行时间复杂度远低于O(n^3)，甚至低于Strassen算法的O(n^(2.7))[见算法导论]。你不可能在它的For循环中间插入取像素的代码。除非你自己去改写CUDA/OpenCL/SMP的底层代码，这是不现实的。

支持(1) 反对(0)

#3楼 2015-11-09 14:53 载重车

@ Physcal 我大概理解你的思路，但是实际操作时，但是Taget = theta*Source 为什么一定要转化为 Source = Theta'*Target?

支持(0) 反对(0)

#4楼 [楼主] 2015-11-09 17:51 Physcal

@ 载重车
我仔细想了一下。
好像正向Target[Res[1,1].x,Res[1,1].y]=SourcePixel[1,1]
和逆向Target[1,1]=SourcePixel[Res[1,1].x,Res[1,1].y]是差不多的。

但你得考虑一下Res[1,1].x、Res[1,1].y 可能是浮点数，你是要做插值的。你如果能够想出来，怎么拿浮点数Target坐标Res[1,1]做整数Source坐标的插值运算，我觉得也是可以的。

因为正向的插值过程，都是引用像素坐标是浮点数（如我的图），这时候正好啊对应逆向变换，得到的Source坐标是浮点数，然后正好用Source像素插值。

如果你非要先正向变换，那么插值过程就要逆向过来，这反而比较麻烦。

从计算角度方面，是我之前理解有问题。我认为正向变换不好离线取出像素，这是不对的。

支持(0) 反对(0)

#5楼 2015-11-09 19:44 载重车

@ Physcal
我现在是这样理解的，target map是固定的坐标，通过6个参数做仿射变换得到在source map中的对应的坐标区域，然后对该区域进行双线性插值。所以应该是source = theta*target.

支持(0) 反对(0)

#6楼 [楼主] 2015-11-09 19:46 Physcal

@ 载重车
对，拿到Source的坐标后，非常容易实现插值。
反之，拿到Target坐标，就麻烦了。

支持(0) 反对(0)

#7楼 2016-01-15 17:21 matscilearn

最近刚好看了这篇文章，有几个问题一起讨论一下~
关于2.2的第一个图
原文章中localisation-Net应该是根据原图出来的，而不是从提取出来的图出来的，前向传播中，后面一层是在有了参数和

原图才可以得到后面的feature map的。也就是，坐标是根据前图得到的。
关于逆向转换
大抵是因为feature map是需要是方形的，正向的话要保证是方形在边界的插值会很麻烦，而且逆向的转换更适合把梯度迭代到前一层对应的激活值，用逆向转换建立映射之后meshgrid，等同于建立一个映射关系，正向转换和逆向转换本身是没有什么区别的，换成逆向仅仅是因为实现起来更简单。
而且，虽然矩阵乘法并行起来很有优势，但是在建立宏观的网络时，很少考虑这种事情，当整体的网络构思清楚后，一切就都可以写为矩阵操作，把细节运算和宏观构思分离开来更能帮助理解。
ps：博客写的真心不错~

支持(1) 反对(0)

#8楼 2017-11-27 00:33 zcy5417

我觉得博主后半部分弄错了，
看完文章我想的是，Localisation net的输入是U，输出θ，θ告诉怎么在U上选取格点，然后把选到的格点给sampler生成V
U的改变会导致θ的改变，所以在训练的时候，流经Localisation net的梯度流会流到U，不会断
U前面的weight变化导致了V的变化，那么这个weight的变化实际上导致了U的变化，从而导致θ的变化，导致在变化了的U上选取格点方式的变化。这个过程是途径Localisation net的
我去跑源码看下

支持(0) 反对(0)

#9楼 2017-12-04 15:50 星际快递员

@ zcy5417
U作为Localisation net的入口貌似必须得认为是固定不变的，所以即使变化从Localisation net 分枝流到U也不能再对其求偏导了。

假设 U 是个 hide layer 那么对 U 的偏导必须经过主网络，也就是直接从 samples 变化插值函数那里一步求得, 这里 samples 变化参数G(θ)是上次迭代出来的缓存值也是固定的。

所以感觉原博主写的还是对的。

另外先前两位楼主的讨论话题：source = theta*target. 问题。一眼看去貌似方向反了，直接认为反了的原因是，疏忽了这个变化真正目的不是得到 target 中像素的坐标，而是 target像素的值。source = theta*target 也只是整个目标的一个小步骤。整体过程是：想得到 target（x=1， y=1）处的像素值，现通过 source = theta*target 变化拿到（Xs，Ys）坐标，经过插值近似直接把（Xs，Ys）附近的像素值Ux copy给 target（x=1， y=1）。然后继续target（x=2， y=2）最终拿到整个 output V。

支持(0) 反对(0)

#10楼 2017-12-10 00:42 zcy5417

@ 星际快递员
我觉得关键是搞清楚误差流的意思，
误差流从后面传到前面是指前面层的weight更新需要后面的weight参与计算，
U前面的weight更新是需要locnet里的weight参与计算的啊
求导就能看出来，我上传不了图片。。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· 从蚂蚁金服到lazada，彭蕾进击东南亚
· 快手的算法，和这个社会的高雅低俗
· 最近几起互联网大事背后的底层逻辑
· Surface Pro 4新固件带来Surface Dial屏幕上交互支持
· 物尽其用，空客计划将货舱改造成卧铺
» 更多新闻...

最新知识库文章:
· 写给自学者的入门指南
· 和程序员谈恋爱
· 学会学习
· 优秀技术人的管理陷阱
· 作为一个程序员，数学对你到底有多重要
» 更多知识库文章...

历史上的今天:
2014-10-21 HDU 3065 (AC自动机模板题)
2014-10-21 HDU 2896 (AC自动机模板题)
2014-10-21 HDU 2222（AC自动机模板题)