l691899397的博客

Paper Reading：Spatial Transformer Networks（with code explanation）

标签： 神经网络 网络 cnn
2016年12月14日 17:25:35 2119人阅读 评论(6) 收藏 举报
分类： 深度学习（13） 论文阅读（1）

原文：Spatial Transformer Networks

前言：卷积神经网络（CNN）已经可以构建出一个强大的分类模型，但它仍然缺乏能力来应对输入数据的空间变换，比如：平移、缩放、旋转，尽管convolutional层和pooling层在一定程度上解决了平移和缩放的问题，但很多时候面对旋转、大尺度的缩放等情况时仍然无能为力。这篇文章提出了一种叫做空间变换网络（Spatial Transform Networks， STN）的模型，它能自动学习变换参数，对上一层的图像进行处理，在一定程度上自适应实现这些变换，从而实现对空间变换的不变性。当输入数据的空间变换差异较大时，STN可以将输入的数据进行“矫正”，进而提高分类的准确性。

前导知识：

1：图像的二维仿射变换

图像的二维仿射变换包括图像的平移（Translation）、缩放（Scale）、旋转（Rotation）等变换，实现这些变换只需要一个2*3维的变换矩阵。

（1）平移变换（Translation）

平移变换完成的操作是：

$$x' = x + T_x$$
$$y' = y + T_y$$

写成矩阵形式为：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + T_x \\ y + T_y \end{bmatrix}$$

（2）缩放（Scale）

$$x' = x * S_x$$
$$y' = y * S_y$$

写成矩阵形式为：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x * S_x \\ y * S_y \end{bmatrix}$$

（3）旋转

旋转的矩阵形式为：

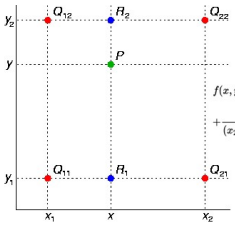
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x * cos\theta + y * sin\theta \\ -x * sin\theta + y * cos\theta \end{bmatrix}$$

由于图像的坐标系的原点在左上角，所以这里会做一个Normalization，把坐标归一化到[-1,1]，这样就是绕图像的中心进行旋转了。不然的话会绕图片的左上角进行旋转。

2、双线性插值

如果把输入图像的一部分映射的输出图像，那么输出图像V中的每一个点，在输入图像U中不一定对应整数点，比如输入图像中的5*5的区域对应输出图像10*10的区域，那么输出图像中很多点对应的输入图像的坐标并不是整数点。这个时候要使用插值法来填充这些像素。

论文中提到了2种方法来填充这些像素，在代码实现中使用了双线性插值的方法。
如下图，中间点P是待插值的点，其中f(x)在我们这里是位于点x处的像素值。

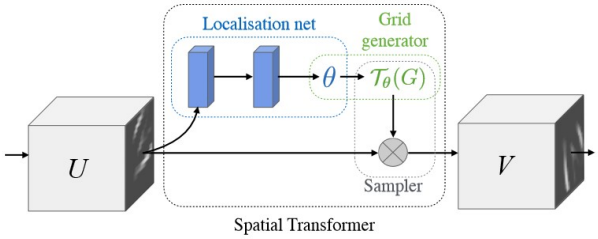


$$f(x,y) \approx \frac{f(Q_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y) + \frac{f(Q_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y) + \frac{f(Q_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1) + \frac{f(Q_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1).$$

STN网络详解

1、正向计算过程：

网络结构：



Spatial Transformer

这幅图是论文中出现的网络结构，可以看出，STN网络为主网络的一个分支，整个STN分支分为三个部分，分别为：

1、Localisation Network
这部分主要是通过一个子网络生成变换参数θ，即2*3维的变换矩阵。子网络主要是由全连接层或者卷积层组成。

2、 Parameterised Sampling Grid
该部分将输入坐标转换为输出坐标。假设输入U每个像素的坐标为(x_i^s ,y_i^s)，输出V的每个像素坐标为(x_i^t ，y_i^t),空间变换函数Tθ为仿射变换函数，那么 (x_i^s ,y_i^s) 和 (x_i^t ,y_i^t) 的对应关系可以写为：

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathbf{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

这里有一点需要注意，论文中用的是输入图像的坐标(x_i^s ,y_i^s) =θ* (x_i^t ,y_i^t)（输出图像的坐标），即根据输出图像V中一点的坐标找该点在输入图像U中的对应坐标，这正好是我们前边仿射变换的逆过程。

3、Differentiable Image Sampling

最后，就可以采用不同的插值方法将输入图像映射到输出图像。入下式 k为不同的sampling kernel。

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \quad \forall i \in [1 \dots H'W'] \quad \forall c \in [1 \dots C]$$

若采用双线性插值的方法，则插值公式为

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

2、反向传播过程：

该层的输入梯度为$\frac{\partial loss}{\partial V}$ ，我们需要计算的包括loss对输入U的导数，以及loss对仿射矩阵θ的导数。
前边的双线性插值的公式如下，在后面的计算过程中会用到：

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

1、loss对输入U的导数

根据双线性插值的公式，对U求导得：

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

该导数的值就是双线性插值的系数。然后根据链式求导法则即可得出loss对U的导数。

2、loss对仿射矩阵θ的导数

$$\frac{\partial V_i^c}{\partial \theta} = \begin{pmatrix} \frac{\partial V_i^c}{\partial x_i^s} \cdot \frac{\partial x_i^s}{\partial \theta} \\ \frac{\partial V_i^c}{\partial y_i^s} \cdot \frac{\partial y_i^s}{\partial \theta} \end{pmatrix}$$

根据链式求导法则 ，所以我们需要求$\frac{\partial V}{\partial x}$ 和$\frac{\partial x}{\partial \theta}$ 。
文中已经给出：

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

所以双线性插值只需对周围4个点进行计算即可，其他的点均为0。$\frac{\partial V}{\partial y}$ 的求法也一样。

而对于$\frac{\partial x}{\partial \theta}$ 和$\frac{\partial y}{\partial \theta}$ ，可以根据下面公式:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathbf{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

很容易求到:

$$\frac{\partial x_i^s}{\partial \theta_{11}} = x_i^t, \quad \frac{\partial x_i^s}{\partial \theta_{12}} = y_i^t, \quad \frac{\partial x_i^s}{\partial \theta_{11}} = 1$$

$$\frac{\partial y_i^s}{\partial \theta_{21}} = x_i^t, \quad \frac{\partial y_i^s}{\partial \theta_{22}} = y_i^t, \quad \frac{\partial xy_i^s}{\partial \theta_{23}} = 1$$

根据链式法则即可求出$\frac{\partial V}{\partial \theta}$ 。

至此，整个stn的推导便结束了。

代码分析

LayerSetUp:

```
void SpatialTransformerLayer<Dtype>::LayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {

    string prefix = "\t\tSpatial Transformer Layer:: LayerSetUp: \t";

    if(this->layer_param_.st_param().transform_type() == "affine") {
        transform_type_ = "affine";
    } else {
        CHECK(false) << prefix << "Transformation type only supports affine now!" << std::en
    }

    if(this->layer_param_.st_param().sampler_type() == "bilinear") {
        sampler_type_ = "bilinear";
    } else {
        CHECK(false) << prefix << "Sampler type only supports bilinear now!" << std::endl;
    }

    if(this->layer_param_.st_param().to_compute_du()) {
        to_compute_dU_ = true;
    }

    std::cout<<prefix<<"Getting output_H_ and output_W_"<<std::endl;

    output_H_ = bottom[0]->shape(2);
    if(this->layer_param_.st_param().has_output_h()) {
        output_H_ = this->layer_param_.st_param().output_h();
    }
    output_W_ = bottom[0]->shape(3);
    if(this->layer_param_.st_param().has_output_w()) {
```

关闭

```
    }

    std::cout<<prefix<<"output_H_ = "<<output_H_<<", output_W_ = "<<output_W_<<std::endl;

    std::cout<<prefix<<"Getting pre-defined parameters"<<std::endl;

    is_pre_defined_theta[0] = false;
    if(this->layer_param_.st_param().has_theta_1_1()) {
        is_pre_defined_theta[0] = true;
        ++ pre_defined_count;
        pre_defined_theta[0] = this->layer_param_.st_param().theta_1_1();
        std::cout<<prefix<<"Getting pre-defined theta[1][1] = "<<pre_defined_theta[0]<<std::en
    }

    is_pre_defined_theta[1] = false;
    if(this->layer_param_.st_param().has_theta_1_2()) {
        is_pre_defined_theta[1] = true;
        ++ pre_defined_count;
        pre_defined_theta[1] = this->layer_param_.st_param().theta_1_2();
        std::cout<<prefix<<"Getting pre-defined theta[1][2] = "<<pre_defined_theta[1]<<std::en
    }

    is_pre_defined_theta[2] = false;
    if(this->layer_param_.st_param().has_theta_1_3()) {
        is_pre_defined_theta[2] = true;
        ++ pre_defined_count;
        pre_defined_theta[2] = this->layer_param_.st_param().theta_1_3();
        std::cout<<prefix<<"Getting pre-defined theta[1][3] = "<<pre_defined_theta[2]<<std::en
    }

    is_pre_defined_theta[3] = false;
    if(this->layer_param_.st_param().has_theta_2_1()) {
        is_pre_defined_theta[3] = true;
        ++ pre_defined_count;
        pre_defined_theta[3] = this->layer_param_.st_param().theta_2_1();
        std::cout<<prefix<<"Getting pre-defined theta[2][1] = "<<pre_defined_theta[3]<<std::en
    }

    is_pre_defined_theta[4] = false;
    if(this->layer_param_.st_param().has_theta_2_2()) {
        is_pre_defined_theta[4] = true;
        ++ pre_defined_count;
        pre_defined_theta[4] = this->layer_param_.st_param().theta_2_2();
        std::cout<<prefix<<"Getting pre-defined theta[2][2] = "<<pre_defined_theta[4]<<std::en
    }

    is_pre_defined_theta[5] = false;
    if(this->layer_param_.st_param().has_theta_2_3()) {
        is_pre_defined_theta[5] = true;
        ++ pre_defined_count;
        pre_defined_theta[5] = this->layer_param_.st_param().theta_2_3();
        std::cout<<prefix<<"Getting pre-defined theta[2][3] = "<<pre_defined_theta[5]<<std::en
    }

    // check the validation for the parameter theta
    CHECK(bottom[1]->count(1) + pre_defined_count == 6) << "The dimension of theta is not six!"
            << " Only " << bottom[1]->count(1) << " + " << pre_defined_count << std::endl;
    CHECK(bottom[1]->shape(0) == bottom[0]->shape(0)) << "The first dimension of theta and " <<
            "U should be the same" << std::endl;

    // initialize the matrix for output grid
    std::cout<<prefix<<"Initializing the matrix for output grid"<<std::endl;

    vector<int> shape_output(2);
    shape_output[0] = output_H_ * output_W_; shape_output[1] = 3;
    output_grid.Reshape(shape_output);

    Dtype* data = output_grid.mutable_cpu_data();
    //这里初始化了保存输出V的坐标的矩阵，并做了Normalization，将坐标归一化到了[-1,1]，每个点的坐标为[
    for(int i=0; i<output_H_ * output_W_; ++i) {
        data[3 * i] = (i / output_W_) * 1.0 / output_H_ * 2 - 1;
        data[3 * i + 1] = (i % output_W_) * 1.0 / output_W_ * 2 - 1;
        data[3 * i + 2] = 1;
    }

    // initialize the matrix for input grid
    std::cout<<prefix<<"Initializing the matrix for input grid"<<std::endl;

    vector<int> shape_input(3);
    shape_input[0] = bottom[1]->shape(0); shape_input[1] = output_H_ * output_W_; shape_input[
    input_grid.Reshape(shape_input);

    std::cout<<prefix<<"Initialization finished."<<std::endl;
}
```

关闭

Forward：

```
void SpatialTransformerLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {

    string prefix = "\t\tSpatial Transformer Layer:: Forward_cpu: \t";

    // CHECK(false) << "Don't use the CPU implementation! If you really want to, delete the" <<
           // " CHECK in st_layer.cpp file. Line number: 240-241." << std::endl;

    if(global_debug) std::cout<<prefix<<"Starting!"<<std::endl;
    //U为输入的图像（可以为整个网络的输入，也可以为某一层的feature map）
    //theta为前边计算得到的仿射矩阵
    const Dtype* U = bottom[0]->cpu_data();
    const Dtype* theta = bottom[1]->cpu_data();
    const Dtype* output_grid_data = output_grid.cpu_data();

    Dtype* input_grid_data = input_grid.mutable_cpu_data();
    Dtype* V = top[0]->mutable_cpu_data();

    caffe_set(input_grid.count(), (Dtype)0, input_grid_data);
    caffe_set(top[0]->count(), (Dtype)0, V);

    // for each input
    for(int i = 0; i < N; ++i) {

        Dtype* coordinates = input_grid_data + (output_H_ * output_W_ * 2) * i;
        //计算输出V中的每一个点的坐标在输入U中对应的坐标
        caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasTrans, output_H_ * output_W_, 2, 3, (Dtype)1.,
              output_grid_data, theta + 6 * i, (Dtype)0., coordinates);

        int row_idx; Dtype px, py;

        for(int j = 0; j < C; ++j)
            for(int s = 0; s < output_H_; ++s)
                for(int t = 0; t < output_W_; ++t) {

                    row_idx = output_W_ * s + t;

                    px = coordinates[row_idx * 2];
                    py = coordinates[row_idx * 2 + 1];
                    //该函数通过双线性插值得到V中每个点的像素值，具体实现见下一段代码
                    V[top[0]->offset(i, j, s, t)] = transform_forward_cpu(
                            U + bottom[0]->offset(i, j, 0, 0), px, py);
                }
    }

    if(global_debug) std::cout<<prefix<<"Finished."<<std::endl;
}
```

transform_forward_cpu：

```
Dtype SpatialTransformerLayer<Dtype>::transform_forward_cpu(const Dtype* pic, Dtype px, Dtyp

    bool debug = false;

    string prefix = "\t\tSpatial Transformer Layer:: transform_forward_cpu: \t";

    if(debug) std::cout<<prefix<<"Starting!\t"<<std::endl;
    if(debug) std::cout<<prefix<<"(px, py) = ("<<px<<", "<<py<<")"<<std::endl;

    Dtype res = (Dtype)0.;
    //将Normalization到[-1,1]区间的坐标值还原到原来区间
    Dtype x = (px + 1) / 2 * H;
    Dtype y = (py + 1) / 2 * W;

    if(debug) std::cout<<prefix<<"(x, y) = ("<<x<<", "<<y<<")"<<std::endl;

    int m, n; Dtype w;
    //下面4部分分别为双线性插值对应的4个点，res为插值得到的像素值
    m = floor(x); n = floor(y); w = 0;
    if(debug) std::cout<<prefix<<"1: (m, n) = ("<<m<<", "<<n<<")"<<std::endl;

    if(m >= 0 && m < H && n >= 0 && n < W) {
        w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));
        res += w * pic[m * W + n];
        if(debug) std::cout<<prefix<<"w = "<<w<<", pic[m, n] = "<<pic[m * W + n]<<std::endl;
    }

    m = floor(x) + 1; n = floor(y); w = 0;
    if(debug) std::cout<<prefix<<"2: (m, n) = ("<<m<<", "<<n<<")"<<std::endl;

    if(m >= 0 && m < H && n >= 0 && n < W) {
        w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));
        res += w * pic[m * W + n];
        if(debug) std::cout<<prefix<<"w = "<<w<<", pic[m, n] = "<<pic[m * W + n]<<std::endl;
```

关闭

```
36
37      m = floor(x); n = floor(y) + 1; w = 0;
38      if(debug) std::cout<<prefix<<"3: (m, n) = ("<<m<<", "<<n<<")"<<std::endl;
39
40      if(m >= 0 && m < H && n >= 0 && n < W) {
41          w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));
42          res += w * pic[m * W + n];
43          if(debug) std::cout<<prefix<<"w = "<<w<<", pic[m, n] = "<<pic[m * W + n]<<std::endl;
44      }
45
46      m = floor(x) + 1; n = floor(y) + 1; w = 0;
47      if(debug) std::cout<<prefix<<"4: (m, n) = ("<<m<<", "<<n<<")"<<std::endl;
48
49      if(m >= 0 && m < H && n >= 0 && n < W) {
50          w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));
51          res += w * pic[m * W + n];
52          if(debug) std::cout<<prefix<<"w = "<<w<<", pic[m, n] = "<<pic[m * W + n]<<std::endl;
53      }
54
55      if(debug) std::cout<<prefix<<"Finished. \tres = "<<res<<std::endl;
56
57      return res;
58  }
```

backward:

```
1   void SpatialTransformerLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
2       const vector<bool>& propagate_down,
3       const vector<Blob<Dtype>*>& bottom) {
4
5           string prefix = "\t\tSpatial Transformer Layer:: Backward_cpu: \t";
6
7           CHECK(false) << "Don't use the CPU implementation! If you really want to, delete the
8                   " CHECK in st_layer.cpp file. Line number: 420-421." << std::endl;
9
10          if(global_debug) std::cout<<prefix<<"Starting!"<<std::endl;
11
12          const Dtype* dV = top[0]->cpu_diff();
13          const Dtype* input_grid_data = input_grid.cpu_data();
14          const Dtype* U = bottom[0]->cpu_data();
15
16          Dtype* dU = bottom[0]->mutable_cpu_diff();
17          Dtype* dTheta = bottom[1]->mutable_cpu_diff();
18          Dtype* input_grid_diff = input_grid.mutable_cpu_diff();
19
20          caffe_set(bottom[0]->count(), (Dtype)0, dU);
21          caffe_set(bottom[1]->count(), (Dtype)0, dTheta);
22          caffe_set(input_grid.count(), (Dtype)0, input_grid_diff);
23
24          for(int i = 0; i < N; ++i) {
25
26              const Dtype* coordinates = input_grid_data + (output_H_ * output_W_ * 2) * i;
27              Dtype* coordinates_diff = input_grid_diff + (output_H_ * output_W_ * 2) * i;
28
29              int row_idx; Dtype px, py, dpx, dpy, delta_dpx, delta_dpy;
30
31              for(int s = 0; s < output_H_; ++s)
32                  for(int t = 0; t < output_W_; ++t) {
33
34                      row_idx = output_W_ * s + t;
35
36                      px = coordinates[row_idx * 2];
37                      py = coordinates[row_idx * 2 + 1];
38
39                      for(int j = 0; j < C; ++j) {
40
41                          delta_dpx = delta_dpy = (Dtype)0.;
42                          //计算dx和dU，具体实现见下一段代码
43                          transform_backward_cpu(dV[top[0]->offset(i, j, s, t)], U + bottom[0]
44                                  px, py, dU + bottom[0]->offset(i, j, 0, 0), delta_dpx, delta_
45
46                          coordinates_diff[row_idx * 2] += delta_dpx;
47                          coordinates_diff[row_idx * 2 + 1] += delta_dpy;
48                      }
49
50                      dpx = coordinates_diff[row_idx * 2];
51                      dpy = coordinates_diff[row_idx * 2 + 1];
52
53                      //这里对应了上面求dx/dTheta和dx/dTheta的部分
54                      dTheta[6 * i] += dpx * (s * 1.0 / output_H_ * 2 - 1);
55                      dTheta[6 * i + 1] += dpx * (t * 1.0 / output_W_ * 2 - 1);
56                      dTheta[6 * i + 2] += dpx;
57                      dTheta[6 * i + 3] += dpy * (s * 1.0 / output_H_ * 2 - 1);
58                      dTheta[6 * i + 4] += dpy * (t * 1.0 / output_W_ * 2 - 1);
59                      dTheta[6 * i + 5] += dpy;
60                  }
61          }
62
```

关闭

```
        if(global_debug) std::cout<<prefix<<"Finished."<<std::endl;
}
```

transform_backward_cpu：

```
void SpatialTransformerLayer<Dtype>::transform_backward_cpu(Dtype dV, const Dtype* U, const 
        const Dtype py, Dtype* dU, Dtype& dpx, Dtype& dpy) {

    bool debug = false;

    string prefix = "\t\tSpatial Transformer Layer:: transform_backward_cpu: \t";

    if(debug) std::cout<<prefix<<"Starting!"<<std::endl;

    //将Normalization到[-1,1]区间的坐标值还原到原来区间
    Dtype x = (px + 1) / 2 * H;
    Dtype y = (py + 1) / 2 * W;
    if(debug) std::cout<<prefix<<"(x, y) = ("<<x<<", "<<y<<")"<<std::endl;

    int m, n; Dtype w;

    //下面4部分也对应了双线性插值的那4个点
    m = floor(x); n = floor(y); w = 0;
    if(debug) std::cout<<prefix<<"(m, n) = ("<<m<<", "<<n<<")"<<std::endl;

    if(m >= 0 && m < H && n >= 0 && n < W) {
        w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));

        //若U中的(m,n)是V中(s,t)对应的点的坐标，则梯度向前传播
        dU[m * W + n] += w * dV;

        //对应上面的dV/dx和dV/dy
        if(abs(x - m) < 1) {
            if(m >= x) {
                dpx += max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
                if(debug) std::cout<<prefix<<"dpx += "<<max(0, 1 - abs(y - n))<<" * "<<U[m *
            } else {
                dpx -= max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
                if(debug) std::cout<<prefix<<"dpx -= "<<max(0, 1 - abs(y - n))<<" * "<<U[m *
            }
        }

        if(abs(y - n) < 1) {
            if(n >= y) {
                dpy += max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
                if(debug) std::cout<<prefix<<"dpy += "<<max(0, 1 - abs(x - m))<<" * "<<U[m *
            } else {
                dpy -= max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
                if(debug) std::cout<<prefix<<"dpy -= "<<max(0, 1 - abs(x - m))<<" * "<<U[m *
            }
        }
    }

    m = floor(x) + 1; n = floor(y); w = 0;
    if(debug) std::cout<<prefix<<"(m, n) = ("<<m<<", "<<n<<")"<<std::endl;

    if(m >= 0 && m < H && n >= 0 && n < W) {
        w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));

        dU[m * W + n] += w * dV;

        if(abs(x - m) < 1) {
            if(m >= x) {
                dpx += max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
                if(debug) std::cout<<prefix<<"dpx += "<<max(0, 1 - abs(y - n))<<" * "<<U[m *
            } else {
                dpx -= max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
                if(debug) std::cout<<prefix<<"dpx -= "<<max(0, 1 - abs(y - n))<<" * "<<U[m *
            }
        }

        if(abs(y - n) < 1) {
            if(n >= y) {
                dpy += max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
                if(debug) std::cout<<prefix<<"dpy += "<<max(0, 1 - abs(x - m))<<" * "<<U[m *
            } else {
                dpy -= max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
                if(debug) std::cout<<prefix<<"dpy -= "<<max(0, 1 - abs(x - m))<<" * "<<U[m *
            }
        }
    }

    m = floor(x); n = floor(y) + 1; w = 0;
    if(debug) std::cout<<prefix<<"(m, n) = ("<<m<<", "<<n<<")"<<std::endl;

    if(m >= 0 && m < H && n >= 0 && n < W) {
        w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));

```

关闭

```
84          dU[m * W + n] += w * dV;
85
86          if(abs(x - m) < 1) {
87              if(m >= x) {
88                  dpx += max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
89                  if(debug) std::cout<<prefix<<"dpx += "<<max(0, 1 - abs(y - n))<<" * "<<U[m * W
90              } else {
91                  dpx -= max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
92                  if(debug) std::cout<<prefix<<"dpx -= "<<max(0, 1 - abs(y - n))<<" * "<<U[m * W
93              }
94          }
95
96          if(abs(y - n) < 1) {
97              if(n >= y) {
98                  dpy += max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
99                  if(debug) std::cout<<prefix<<"dpy += "<<max(0, 1 - abs(x - m))<<" * "<<U[m * W
100             } else {
101                 dpy -= max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
102                 if(debug) std::cout<<prefix<<"dpy -= "<<max(0, 1 - abs(x - m))<<" * "<<U[m * W
103             }
104         }
105     }
106
107     m = floor(x) + 1; n = floor(y) + 1; w = 0;
108     if(debug) std::cout<<prefix<<"(m, n) = ("<<m<<", "<<n<<")"<<std::endl;
109
110     if(m >= 0 && m < H && n >= 0 && n < W) {
111         w = max(0, 1 - abs(x - m)) * max(0, 1 - abs(y - n));
112
113         dU[m * W + n] += w * dV;
114
115         if(abs(x - m) < 1) {
116             if(m >= x) {
117                 dpx += max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
118                 if(debug) std::cout<<prefix<<"dpx += "<<max(0, 1 - abs(y - n))<<" * "<<U[m * W
119             } else {
120                 dpx -= max(0, 1 - abs(y - n)) * U[m * W + n] * dV * H / 2;
121                 if(debug) std::cout<<prefix<<"dpx -= "<<max(0, 1 - abs(y - n))<<" * "<<U[m * W
122             }
123         }
124
125         if(abs(y - n) < 1) {
126             if(n >= y) {
127                 dpy += max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
128                 if(debug) std::cout<<prefix<<"dpy += "<<max(0, 1 - abs(x - m))<<" * "<<U[m * W
129             } else {
130                 dpy -= max(0, 1 - abs(x - m)) * U[m * W + n] * dV * W / 2;
131                 if(debug) std::cout<<prefix<<"dpy -= "<<max(0, 1 - abs(x - m))<<" * "<<U[m * W
132             }
133         }
134     }
135
136     if(debug) std::cout<<prefix<<"Finished."<<std::endl;
137 }
```


- 上一篇 CCF-201503-3
- 下一篇 Paper Reading：Regional Multi-person Pose Estimation

关闭

您还没有登录,请[登录]或[注册]

查看评论

paopaoxr 2楼 2017-02-23 08:20发表

博主，你好！好厉害，膜拜你！我加了SpatialTransformerLayer层后，编译caffe的时候出现了以下错误：
F0222 16:49:31.335220 1862 st_layer.cpp:232] Check failed: false Don't use the CPU implementation! If you really want to, delete the CHECK in st_layer.cpp file. Lin

e number: 240-241.
*** Check failure stack trace: ***
@ 0x7f9c01e08e6d (unknown)
这是什么原因造成的呀？麻烦博主指点一下！！！

查看更多评论

深度学习方法（十二）：卷积神经网络结构变化——Spatial Transformer Networks

今天具体介绍一个Google DeepMind在15年提出的Spatial Transformer Networks，相当于在传统的一层Convolution中间，装了一个“插件”，可以使得传统的卷积...

xbinworld 2017年04月03日 23:45 7316

【论文笔记】Spatial Transformer Networks

shaoxiaohu1 2016年07月02日 12:46 14785

卷积神经网络（CNN）已经被证明能够训练一个能力强大的分类模型，但与传统的模式识别方法类似，它也会受到数据在空间上多样性的影响。这篇Paper提出了一种叫做空间变换网络（Spatial Transfo...

论文笔记：Spatial Transformer Networks（空间变换网络）

上一篇博客 Spatial Transformer Networks论文笔记（一）——仿射变换和双线性插值介绍了仿射变换和双线性插值，为更好地理解STN打基础。本篇博客是记录的是阅读原文Spatial...

sinat_34474705 2017年07月17日 21:18 1203

STN系列之Spatial Transformer Networks

dreamer_on_air 2017年09月04日 13:32 457

Spatial Transformer Networks. Max Jaderberg Karen Simonyan Andrew Zisserman Koray KavukcuogluGoogle ...

空间映射网络--Spatial Transformer Networks

zhangjunhit 2017年03月28日 09:29 2032

Spatial Transformer Networks 主要对目标在特征空间做不变性归一化 解决 角度、尺度等变形引入的影响 Code: https://github.com/skaae/t...

论文阅读《Spatial Transformer Networks》

zzchust 2015年11月09日 12:26 5169

STN module inserted to CNN without any extra training feature maps learn invariance to translation, ...

[深度学习论文笔记][CVPR 17 oral] Inverse Compositional Spatial Transformer...

这篇文章是针对Spatial Transformer Network进一步改进的工作。从研究领域来看，该工作是对增强深度网络之于输入图片空间不变性的研究。...

u010158659 2017年04月23日 00:44 6542

2016-Spatial Transformer Networks理解

PPT

2018年01月17日 10:00 680KB 下载

空间映射网络--Spatial Transformer Networks

u014568921 2017年05月31日 21:54 690

深度学习方法（十二）：卷积神经网络结构变化——Spatial Transformer Networks 空间映射网络--Spatial Transformer Networks...

详细解读Spatial Transformer Networks（STN）-一篇文章让你完全理解STN了

目录 STN的作用 1.1 灵感来源 1.2 什么是STN？ STN的基本架构 Localisation net是如何实现参数的选取的？ 3.1 实现平移 3.2 实现缩放 3.3 实现旋转...

qq_39422642 2017年12月22日 10:36 693

Tensorflow1.0空间变换网络(SpatialTransformer Networks)实现

空间变换网络简单介绍： 通过locatnet，提取输入图像的theta(将用于仿射变换)；根据输入图像的width和height以及仿射变换(或者TPS)的参数theta，可以生成目标位...

weixin_36368407 2017年03月01日 10:01 1459

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

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...

brandon2015 2017年05月12日 17:56 1260

Spatial Transformer Networks

keyanxiaocaicai 2017年06月11日 21:06 245

caffe-stn移植到高版本的方法 参考：http://blog.csdn.net/kuaitoukid/article/details/51035028 https://github.com/ha...

空间变换网络--spatial transform network

u011961856 2017年09月10日 11:11 2190

CNN分类时，通常需要考虑输入样本的局部性、平移不变性、缩小不变性，旋转不变性等，以提高分类的准确度。这些不变性的本质就是图像处理的经典方法，即图像的裁剪、平移、缩放、旋转，而这些方法实际上就是对图像...

【Unet】不使用NetWorkTransform组件 进行同步位移和旋转

using System.Collections; using System.Collections.Generic; using UnityEngine; using UnityEngine.Net...

ldy597321444 2017年04月12日 17:49 1583

caffe-stn移植到高版本的方法

kuaitoukid 2016年04月01日 11:00 3041

本文介绍了将github上开源的stn-caffe代码移植到新版本的caffe的方法

Spatial Transformation Network

DuinoDu 2017年04月07日 12:46 877

STN

论文阅读《Spatial Transformer Network》

yaoqi_isee 2017年05月27日 22:01 618

Google DeepMindAbstract:作者说明了CNN对于输入的数据缺乏空间变换不变形(lack of spatially invariant ability to input data)，...

[深度学习论文笔记][Attention] Spatial Transformer Networks

Jaderberg, Max, Karen Simonyan, and Andrew Zisserman. “Spatial transformer networks.” Advances in Ne...

Hao_Zhang_Vision 2016年11月15日 22:02 1303

A brief scanning of paper “Spatial Transformer Network”

Part.1: what’s the problems the method proposed in this paper solved? When we using a convolutio...

sinat_24002967 2017年11月09日 22:06 60