Datawhale 计算机视觉基础-图像处理（上）-Task02 几何变换

2.1 简介

该部分将对基本的几何变换进行学习，几何变换的原理大多都是相似，只是变换矩阵不同，因此，我们以最常用的平移和旋转为例进行学习。在深度学习领域，我们常用平移、旋转、镜像等操作进行数据增广；在传统CV领域，由于某些拍摄角度的问题，我们需要对图像进行矫正处理，而几何变换正是这个处理过程的基础，因此了解和学习几何变换也是有必要的。

这次我们带着几个问题进行，以旋转为例：

- 1：变换的形式（公式）是什么？
- 2：旋转中心是什么？毕竟以不同位置为旋转中心得到的结果是不一样的。
- 3：采用前向映射还是反向映射？(反向映射更为有效)
- 4：采用反向映射后，采用何种插值算法？最常用的的是双线性插值，OpenCV也是默认如此。

2.2 学习目标

- 了解几何变换的概念与应用
- 理解平移、旋转的原理
- 掌握在OpenCV框架下实现平移、旋转操作

2.3 内容介绍

1、平移、旋转的原理

2、OpenCV代码实践

3、动手实践并打卡（读者完成）

2.4 算法理论介绍

变换形式

先看第一个问题，变换的形式。与OpencV不同的是这里采取冈萨雷斯的《数字图像处理_第三版》的变换矩阵方式，关于OpenCV的策略可以看它的官方文档。根据冈萨雷斯书中的描述，仿射变换的一般形式如下：

$$[x \quad y \quad 1] = [v \quad w \quad 1]\mathbf{T} = [v \quad w \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

式中的T就是变换矩阵，其中 (v,w)为原坐标，(x,y) 为变换后的坐标，不同的变换对应不同的矩阵，这里也贴出来吧，一些常见的变换矩阵及作用如下表：

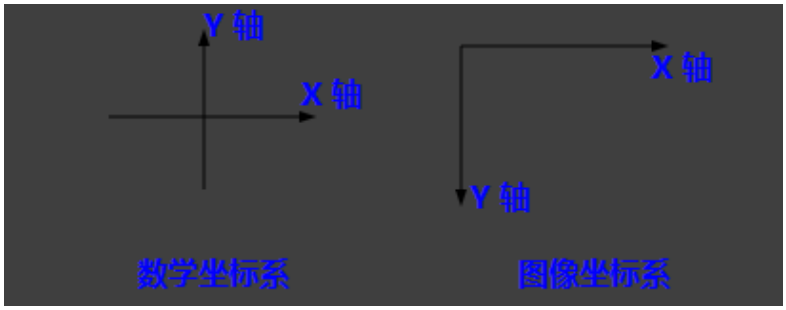
变换名称	仿射变换矩阵$\mathbf{T}$	坐标公式
恒等变换	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{cases} x = v, \\ y = w \end{cases}$
尺度变换	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{cases} x = vc_x, \\ y = wc_y \end{cases}$
旋转变换（以逆时针为正）	$\begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{cases} x = vcos(\theta) - wsin(\theta), \\ y = vsin(\theta) + wcos(\theta) \end{cases}$
旋转变换（以顺时针为正）	$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{cases} x = vcos(\theta) + wsin(\theta), \\ y = -vsin(\theta) + wcos(\theta) \end{cases}$
平移变换	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$\begin{cases} x = v + t_x, \\ y = w + t_y \end{cases}$
偏移变换（水平）	$\begin{bmatrix} 1 & 0 & 0 \\ s_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{cases} x = v + ws_h, \\ y = w \end{cases}$
偏移变换（垂直）	$\begin{bmatrix} 1 & s_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{cases} x = v, \\ y = vs_v + w \end{cases}$

也就是说，我们根据自己的目的选择不同变换矩阵就可以了。

坐标系变换

再看第二个问题，变换中心，对于缩放、平移可以以图像坐标原点（图像左上角为原点）为中心变换，这不用坐标系变换，直接按照一般形式计算即可。而对于旋转和偏移，一般是以图像中心为原点，那么这就涉及坐标系转换了。

我们都知道，图像坐标的原点在图像左上角，水平向右为 X 轴，垂直向下为 Y 轴。数学课本中常见的坐标系是以图像中心为原点，水平向右为 X 轴，垂直向上为 Y 轴，称为笛卡尔坐标系。看下图:

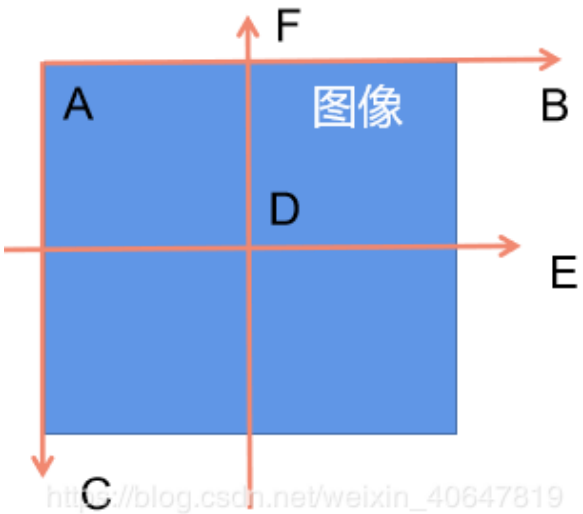


因此，对于旋转和偏移，就需要3步（3次变换）：

- 将输入原图图像坐标转换为笛卡尔坐标系；
- 进行旋转计算。旋转矩阵前面已经给出了；
- 将旋转后的图像的笛卡尔坐标转回图像坐标。

图像坐标系与笛卡尔坐标系转换关系：

先看下图：



在图像中我们的坐标系通常是AB和AC方向的,原点为A，而笛卡尔直角坐标系是DE和DF方向的，原点为D。 **令图像表示为M×N的矩阵，对于点A而言，两坐标系中的坐标分别是(0，0)和(-N/2,M/2)，则图像某像素点(x',y')转换为笛卡尔坐标（x,y）转换关系为，x为列，y为行：**

$$x = x'-N/2$$
$$y = -y'+M/2$$

逆变换为：

$$x' = x + N/2$$
$$y' = -y + M/2$$

于是，根据前面说的3个步骤（3次变换），旋转(顺时针旋转)的变换形式就为，3次变换就有3个矩阵：

$$(x,y,1) = (x',y',1)T = (x',y',1)\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5*N & 0.5*M & 1 \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5*N & 0.5*M & 1 \end{bmatrix}$$

反向映射

看第3个问题，在冈萨雷斯的《数字图像处理_第三版》中说的很清楚，前向映射就是根据原图用变换公式直接算出输出图像相应像素的空间位置，那么这会导致一个问题：可能会有多个像素坐标映射到输出图像的同一位置，也可能输出图像的某些位置完全没有相应的输入图像像素与它匹配，也就是没有被映射到，造成有规律的空洞（黑色的蜂窝状）。更好的一种方式是采用 反向映射（Inverse Mapping）：扫描输出图像的位置(x,y)，通过 $[v,w,1] = [x,y,1] * T^{-1}$ （为T的逆矩阵）计算输入图像对应的位置(v,w)，通过插值方法决定输出图像该位置的灰度值。

插值

第4个问题，采用反向映射后，需通过插值方法决定输出图像该位置的值，因此需要选择插值算法。通常有最近邻插值、双线性插值，双三次插值等，OpencV默认采用双线性插值，我们也就采用双线性插值。

2.5 基于OpenCV的实现

- 工具：OpenCV3.1.0+VS2013
- 平台：WIN10

函数原型（c++）

OpenCV仿射变换相关的函数一般涉及到warpAffine和getRotationMatrix2D这两个：

- 使用OpenCV函数warpAffine 来实现一些简单的重映射.
- OpenCV函数getRotationMatrix2D 来获得旋转矩阵。

1、warpAffined函数详解

```
void boxFilter( InputArray src, OutputArray dst,
                int ddepth,
                Size ksize,
                Point anchor = Point(-1,-1),
```

```
                bool normalize = true,
                int borderType = BORDER_DEFAULT );
```

- 第一个参数，InputArray类型的src，输入图像，即源图像，填Mat类的对象即可。
- 第二个参数，OutputArray类型的dst，函数调用后的运算结果存在这里，需和源图片有一样的尺寸和类型。
- 第三个参数，InputArray类型的M，2×3的变换矩阵。
- 第四个参数，Size类型的dsize，表示输出图像的尺寸。
- 第五个参数，int类型的flags，插值方法的标识符。此参数有默认值INTER_LINEAR(线性插值)，可选的插值方式如下： INTER_NEAREST - 最近邻插值 INTER_LINEAR - 线性插值（默认值） INTER_AREA - 区域插值 INTER_CUBIC –三次样条插值 INTER_LANCZOS4 -Lanczos插值 CV_WARP_FILL_OUTLIERS - 填充所有输出图像的象素。如果部分象素落在输入图像的边界外，那么它们的值设定为fillval. CV_WARP_INVERSE_MAP –表示M为输出图像到输入图像的反变换，即 。因此可以直接用来做象素插值。否则, warpAffine函数从M矩阵得到反变换。
- 第六个参数，int类型的borderMode，边界像素模式，默认值为BORDER_CONSTANT。
- 第七个参数，const Scalar&类型的borderValue，在恒定的边界情况下取的值，默认值为Scalar()，即0。

2、getRotationMatrix2D函数详解

```
C++: Mat getRotationMatrix2D(Point2f center, double angle, double scale)
```

参数：

- 第一个参数，Point2f类型的center，表示源图像的旋转中心。
- 第二个参数，double类型的angle，旋转角度。角度为正值表示向逆时针旋转（坐标原点是左上角）。
- 第三个参数，double类型的scale，缩放系数。## 2.6 总结

实现示例（c++)

1、旋转

```
        cv::Mat src = cv::imread("lenna.jpg");
        cv::Mat dst;

        //旋转角度
        double angle = 45;

        cv::Size src_sz = src.size();
        cv::Size dst_sz(src_sz.height, src_sz.width);
        int len = std::max(src.cols, src.rows);

        //指定旋转中心（图像中点）
        cv::Point2f center(len / 2., len / 2.);
```

```
        //获取旋转矩阵（2x3矩阵）
        cv::Mat rot_mat = cv::getRotationMatrix2D(center, angle, 1.0);

        //根据旋转矩阵进行仿射变换
        cv::warpAffine(src, dst, rot_mat, dst_sz);

        //显示旋转效果
        cv::imshow("image", src);
        cv::imshow("result", dst);

        cv::waitKey(0);

        return 0;
```

2、平移

```
        cv::Mat src = cv::imread("lenna.jpg");
        cv::Mat dst;

        cv::Size dst_sz = src.size();

        //定义平移矩阵
        cv::Mat t_mat =cv::Mat::zeros(2, 3, CV_32FC1);

        t_mat.at<float>(0, 0) = 1;
        t_mat.at<float>(0, 2) = 20; //水平平移量
        t_mat.at<float>(1, 1) = 1;
        t_mat.at<float>(1, 2) = 10; //竖直平移量

        //根据平移矩阵进行仿射变换
        cv::warpAffine(src, dst, t_mat, dst_sz);

        //显示平移效果
        cv::imshow("image", src);
        cv::imshow("result", dst);

        cv::waitKey(0);

        return 0;
```

进阶实现(根据原理自己实现)

1、旋转

```
/*图像旋转（以图像中心为旋转中心）*/
void affine_trans_rotate(cv::Mat& src, cv::Mat& dst, double Angle){
        double angle = Angle*CV_PI / 180.0;
        //构造输出图像
        int dst_rows = round(fabs(src.rows * cos(angle)) + fabs(src.cols
        int dst_cols = round(fabs(src.cols * cos(angle)) + fabs(src.rows

        if (src.channels() == 1) {
```

```
                dst = cv::Mat::zeros(dst_rows, dst_cols, CV_8UC1); //灰度图
        }
        else {
                dst = cv::Mat::zeros(dst_rows, dst_cols, CV_8UC3); //RGB图
        }

        cv::Mat T1 = (cv::Mat_<double>(3,3) << 1.0,0.0,0.0 , 0.0,-1.0,0.0
        cv::Mat T2 = (cv::Mat_<double>(3,3) << cos(angle),-sin(angle),0.0
        double t3[3][3] = { { 1.0, 0.0, 0.0 }, { 0.0, -1.0, 0.0 }, { 0.5*
        cv::Mat T3 = cv::Mat(3.0,3.0,CV_64FC1,t3);
        cv::Mat T = T1*T2*T3;
        cv::Mat T_inv = T.inv(); // 求逆矩阵

        for (double i = 0.0; i < dst.rows; i++){
                for (double j = 0.0; j < dst.cols; j++){
                        cv::Mat dst_coordinate = (cv::Mat_<double>(1, 3)
                        cv::Mat src_coordinate = dst_coordinate * T_inv;
                        double v = src_coordinate.at<double>(0, 0); // 原
                        double w = src_coordinate.at<double>(0, 1); // 原
                //      std::cout << v << std::endl;

                        /*双线性插值*/
                        // 判断是否越界
                        if (int(Angle) % 90 == 0) {
                                if (v < 0) v = 0; if (v > src.cols - 1) v
                                if (w < 0) w = 0; if (w > src.rows - 1) w
                        }

                        if (v >= 0 && w >= 0 && v <= src.cols - 1 && w <=
                                int top = floor(w), bottom = ceil(w), lef
                                double pw = w - top ; //pw为坐标 行 的小数部
                                double pv = v - left; //pv为坐标 列 的小数部
                                if (src.channels() == 1){
                                        //灰度图像
                                        dst.at<uchar>(i, j) = (1 - pw)*(1
                                }
                                else{
                                        //彩色图像
                                        dst.at<cv::Vec3b>(i, j)[0] = (1 -
                                        dst.at<cv::Vec3b>(i, j)[1] = (1 -
                                        dst.at<cv::Vec3b>(i, j)[2] = (1 -
                                }
                        }
                }
        }
}
```

2、平移

```
/*平移变换*（以图像左顶点为原点）/
/****************************************
tx: 水平平移距离 正数向右移动 负数向左移动
ty: 垂直平移距离 正数向下移动 负数向上移动
****************************************/
void affine_trans_translation(cv::Mat& src, cv::Mat& dst, double tx, doub
        //构造输出图像
```

```
        int dst_rows = src.rows;//图像高度
        int dst_cols = src.cols;//图像宽度

        if (src.channels() == 1) {
                dst = cv::Mat::zeros(dst_rows, dst_cols, CV_8UC1); //灰度
        }
        else {
                dst = cv::Mat::zeros(dst_rows, dst_cols, CV_8UC3); //RGB
        }

        cv::Mat T = (cv::Mat_<double>(3, 3) << 1,0,0 , 0,1,0 , tx,ty,1);
        cv::Mat T_inv = T.inv(); // 求逆矩阵

        for (int i = 0; i < dst.rows; i++){
                for (int j = 0; j < dst.cols; j++){
                        cv::Mat dst_coordinate = (cv::Mat_<double>(1, 3)
                        cv::Mat src_coordinate = dst_coordinate * T_inv;
                        double v = src_coordinate.at<double>(0, 0); // 原
                        double w = src_coordinate.at<double>(0, 1); // 原

                        /*双线性插值*/
                        // 判断是否越界

                        if (v >= 0 && w >= 0 && v <= src.cols - 1 && w <=
                                int top = floor(w), bottom = ceil(w), lef
                                double pw = w - top; //pw为坐标 行 的小数部
                                double pv = v - left; //pv为坐标 列 的小数部
                                if (src.channels() == 1){
                                        //灰度图像
                                        dst.at<uchar>(i, j) = (1 - pw)*(1
                                }
                                else{
                                        //彩色图像
                                        dst.at<cv::Vec3b>(i, j)[0] = (1 -
                                        dst.at<cv::Vec3b>(i, j)[1] = (1 -
                                        dst.at<cv::Vec3b>(i, j)[2] = (1 -
                                }
                        }
                }
        }
}
```

效果

1、旋转45度

2、平移

相关技术文档、博客、教材、项目推荐

opencv文档:
https://docs.opencv.org/3.1.0/da/d54/group__imgproc__transform.html#ga0203d9ee5fcd28d40dbc4a1ea4451983
博客：https://blog.csdn.net/weixin_40647819/article/details/87912122
https://www.jianshu.com/p/18cd12e776e1
https://blog.csdn.net/whuhan2013/article/details/53814026
python版本：https://blog.csdn.net/g11d111/article/details/79978582
https://www.kancloud.cn/aollo/aolloopencv/264331
http://www.woshicver.com/FifthSection/4_2_%E5%9B%BE%E5%83%8F%E5%87%A0%E4%BD%95%E5%8F%98%E6%8D%A2/

2.6 总结

该部分对几何变换的平移和旋转进行了介绍，读者可根据提供的资料对相关原理进行学习，然后参考示例代码自行实现。另外读者可以尝试学习并实现其他几何变换，如偏移。

Task02 几何变换 END.