LSGOMYP 课程内容提交

caf4e32 2 days ago

1 contributor

Datawhale 计算机视觉基础-图像处理（上）-Task01 OpenCV框架与图像插值算法

1.1 简介

在图像处理中，平移变换、旋转变换以及放缩变换是一些基础且常用的操作。这些几何变换并不改变图象的象素值，只是在图象平面上进行象素的重新排列。在一幅输入图象$[u，v]$中，灰度值仅在整数位置上有定义。然而，输出图象[x，y]的灰度值一般由处在非整数坐标上的$（u，v）$值来决定。这就需要插值算法来进行处理，常见的插值算法有最近邻插值、双线性插值和三次样条插值。

1.2 学习目标

- 了解插值算法与常见几何变换之间的关系
- 理解插值算法的原理
- 掌握OpenCV框架下插值算法API的使用

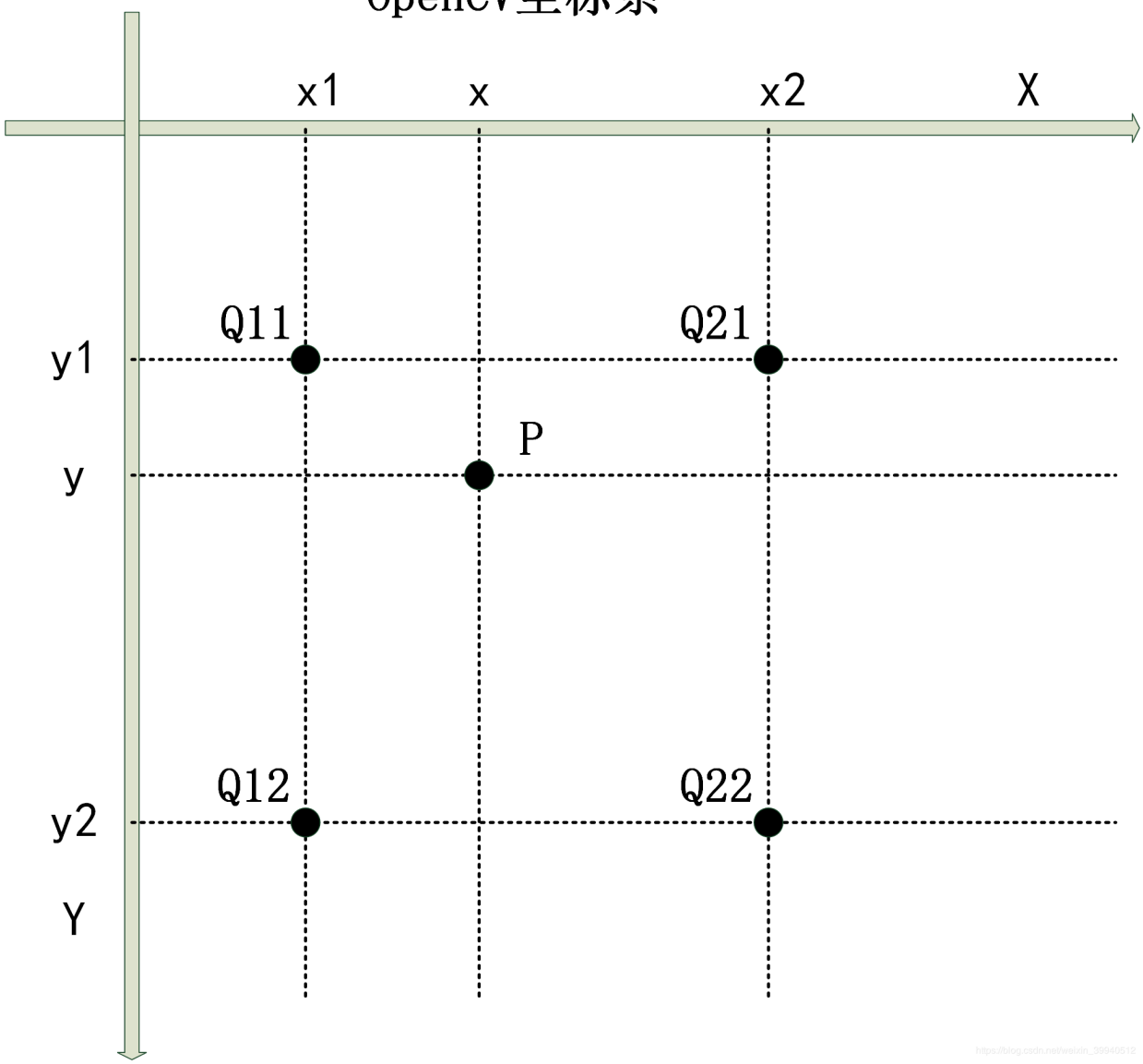
1.3 内容介绍

1. 插值算法原理介绍
    - 最近邻插值算法
    - 双线性插值算法
2. OpenCV代码实践
    - cv.resize()各项参数及含义
3. 动手实现（由读者自己完成）

1.4 算法理论介绍与推荐

1.4.1 最近邻插值算法原理

最近邻插值，是指将目标图像中的点，对应到源图像中后，找到最相邻的整数点，作为插值后的输出。



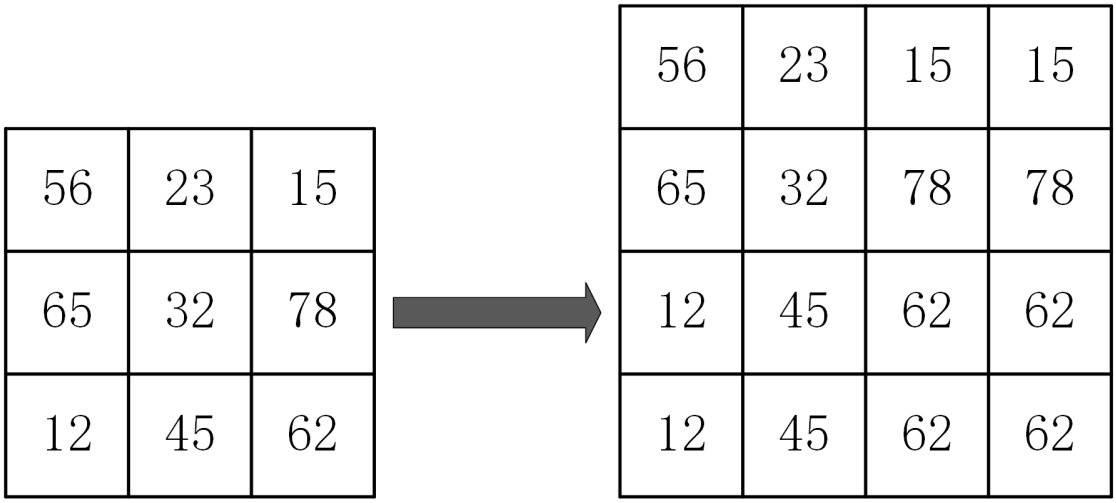
如上图所示，目标图像中的某点投影到原图像中的位置为点P,此时易知，$f(P) = f(Q11)$.

一个例子：

如下图所示，将一幅3X3的图像放大到4X4，用$f(x, y)$表示目标图像，$h(x, y)$表示原图像，我们有如下公式：

$$ \begin{array}{c} f(dst_{X}, dst_{Y}) = h(\frac{dst_{X}src_{Width}} {dst_{Width}}, \frac{dst_{Y}src_{Height}} {dst_{Height}}) \end{array} $$

$$ \begin{array}{c} f(0,0)=h(0,0) \ f(0,1)=h(0,0.75)=h(0,1) \ f(0,2)=h(0,1.50)=h(0,2) \ f(0,3)=h(0,2.25)=h(0,2) \ ...\ \end{array} $$



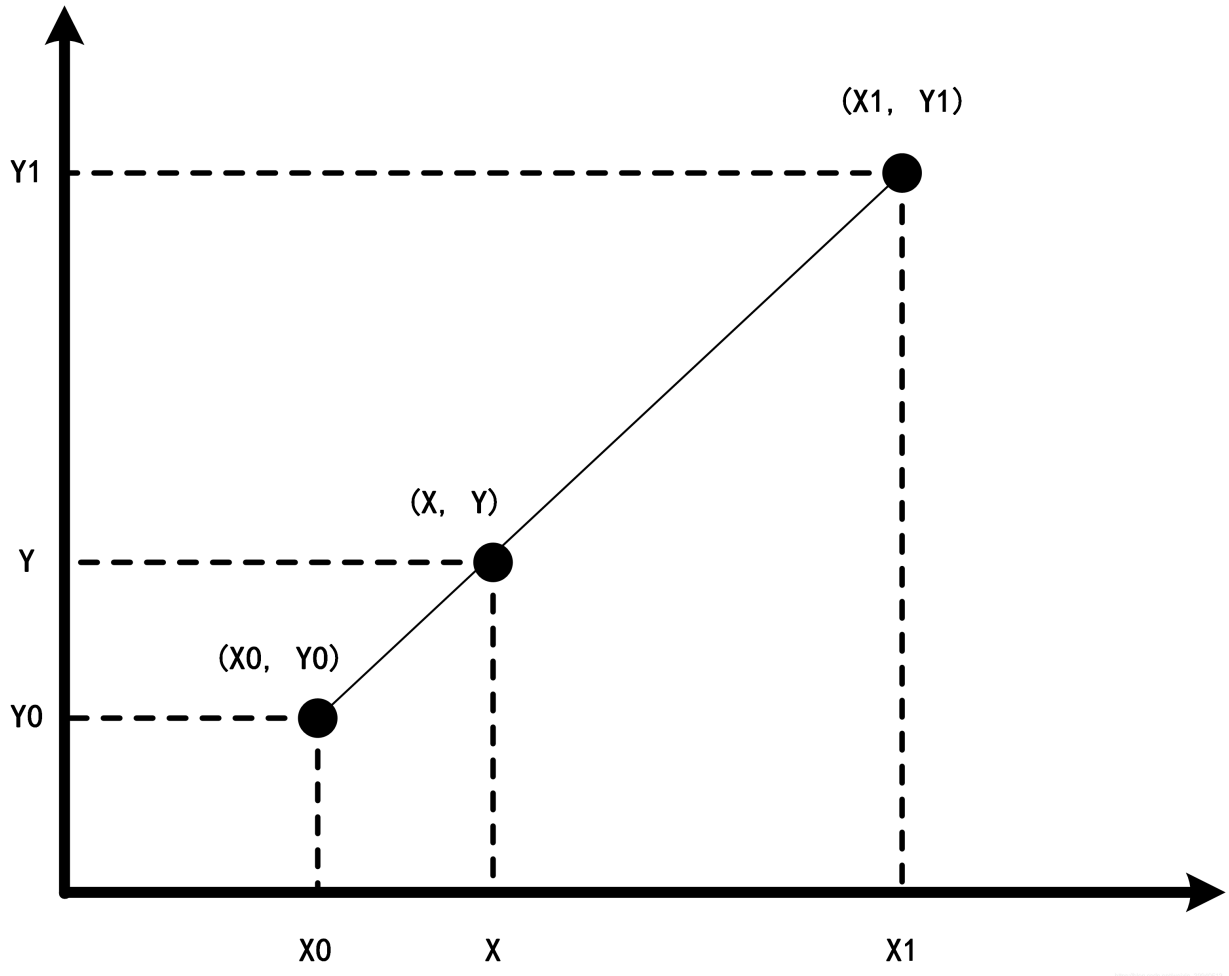
.

缺点： 用该方法作放大处理时，在图象中可能出现明显的块状效应

1.4.2 双线性插值

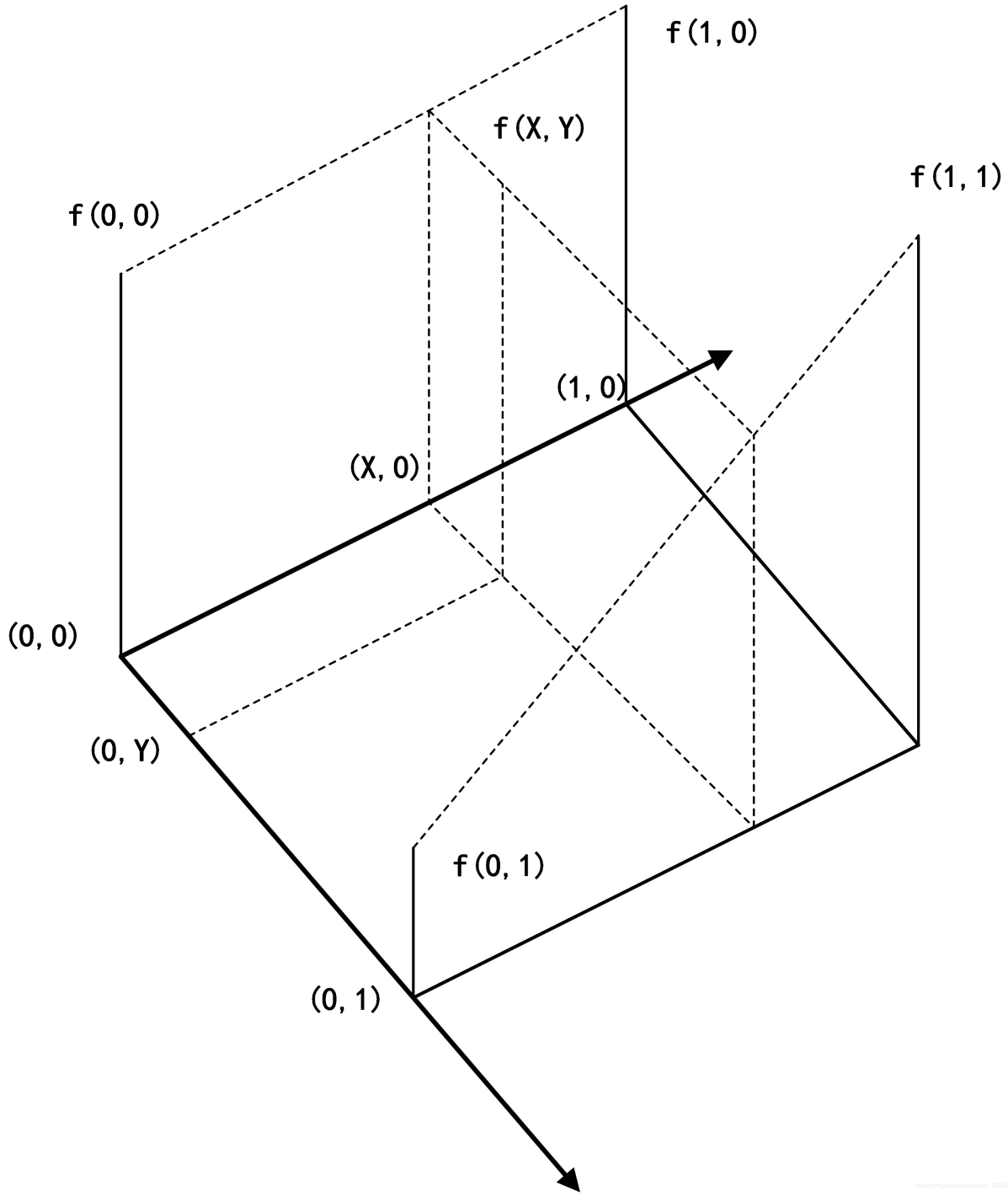
在讲双线性插值之前先看以一下线性插值，线性插值多项式为：

$$ f(x)=a_{1} x+a_{0} $$



$$ y=y_{0}+\left(x-x_{0}\right) \frac{y_{1}-y_{0}}{x_{1}-x_{0}}=y_{0}+\frac{\left(x-x_{0}\right) y_{1}-\left(x-x_{0}\right) y_{0}}{x_{1}-x_{0}} $$

双线性插值就是线性插值在二维时的推广,在两个方向上做三次线性插值，具体操作如下图所示：



令$f(x，y)$为两个变量的函数，其在单位正方形顶点的值已知。假设我们希望通过插值得到正方形内任意点的函数值。则可由双线性方程: $$ f(x, y)=a x+b y+c x y+d $$

来定义的一个双曲抛物面与四个已知点拟合。

首先对上端的两个顶点进行线性插值得:

$$ f(x, 0)=f(0,0)+x[f(1,0)-f(0,0)] $$

类似地，再对底端的两个顶点进行线性插值有： $$ f(x, 1)=f(0,1)+x[f(1,1)-f(0,1)] $$

最后，做垂直方向的线性插值，以确定:

$$ f(x, y)=f(x, 0)+y[f(x, 1)-f(x, 0)] $$

整理得:

$$ \begin{array}{l} f(x, y)=[f(1,0)-f(0,0)] x+[f(0,1)-f(0,0)] y \ +[f(1,1)+f(0,0)-f(0,1)-f(1,0)] x y+f(0,0) \end{array} $$

1.4.3 映射方法

向前映射法

可以将几何运算想象成一次一个象素地转移到输出图象中。如果一个输入象素被映射到四个输出象素之间的位置，则其灰度值就按插值算法在4个输出象素之间进行分配。称为向前映射法，或象素移交影射。

> 注：从原图象坐标计算出目标图象坐标镜像、平移变换使用这种计算方法

向后映射法

向后映射法（或象素填充算法）是输出象素一次一个地映射回到输入象素中，以便确定其灰度级。如果一个输出象素被映射到4个输入象素之间，则其灰度值插值决定，向后空间变换是向前变换的逆。

> 注：从结果图象的坐标计算原图象的坐标

- 旋转、拉伸、放缩可以使用
- 解决了漏点的问题，出现了马赛克

1.5 基于OpenCV的实现

1.5.1 C++

函数原型：

> void cv::resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)

```
src:输入图像
dst:输出图像
dsize:输出图像尺寸
fx、fy:x,y方向上的缩放因子
INTER_LINEAR：插值方法，总共五种
    1. INTER_NEAREST – 最近邻插值法
    2. INTER_LINEAR – 双线性插值法（默认）
    3. INTER_AREA – 基于局部像素的重采样(resampling using pixel area relation)。对于图像抽取(image decimation)来说，这可能是一个更好的方法。但如果是放大图像时，它和最近邻法的效果类似。
    4. INTER_CUBIC – 基于4x4像素邻域的3次插值法
    5. INTER_LANCZOS4 – 基于8x8像素邻域的Lanczos插值
```

代码实践：

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
        Mat img = imread("D:/image/yuner.jpg");
        if (img.empty())
        {
                cout << "无法读取图像" << endl;
                return 0;
        }

        int height = img.rows;
        int width = img.cols;
        // 缩小图像，比例为(0.2, 0.2)
        Size dsize = Size(round(0.2 * width), round(0.2 * height));
        Mat shrink;
    //使用双线性插值
        resize(img, shrink, dsize, 0, 0, INTER_LINEAR);

        // 在缩小图像的基础上，放大图像，比例为(1.5, 1.5)
        float fx = 1.5;
        float fy = 1.5;
        Mat enlarge1, enlarge2;
        resize(shrink, enlarge1, Size(), fx, fy, INTER_NEAREST);
        resize(shrink, enlarge2, Size(), fx, fy, INTER_LINEAR);

        // 显示
        imshow("src", img);
        imshow("shrink", shrink);
```

```
        imshow("INTER_NEAREST", enlarge1);
        imshow("INTER_LINEAR", enlarge2);
        waitKey(0);
    return 0;
}
```

原图

0.2倍缩小，双线性插值

1.5倍放大，最近邻插值

1.5倍放大，双线性插值

1.5.2 Python

函数原型：

> cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])

参数：

参数	描述
src	【必需】原图像
dsize	【必需】输出图像所需大小
fx	【可选】沿水平轴的比例因子
fy	【可选】沿垂直轴的比例因子
interpolation	【可选】插值方式

插值方式：

cv.INTER_NEAREST	最近邻插值
cv.INTER_LINEAR	双线性插值
cv.INTER_CUBIC	基于4x4像素邻域的3次插值法
cv.INTER_AREA	基于局部像素的重采样

> 通常，缩小使用cv.INTER_AREA，放缩使用cv.INTER_CUBIC(较慢)和cv.INTER_LINEAR(较快效果也不错)。默认情况下，所有的放缩都使用cv.INTER_LINEAR。

代码实践：

```
import cv2

if __name__ == "__main__":
    img = cv2.imread('D:/image/yuner.jpg', cv2.IMREAD_UNCHANGED)

    print('Original Dimensions : ',img.shape)

    scale_percent = 30       # percent of original size
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    # resize image
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_LINEAR)

    fx = 1.5
    fy = 1.5

    resized1 = cv2.resize(resized, dsize=None, fx=fx, fy=fy, interpolation = cv2.INTER_NEAREST)

    resized2 = cv2.resize(resized, dsize=None, fx=fx, fy=fy, interpolation = cv2.INTER_LINEAR)
    print('Resized Dimensions : ',resized.shape)

    cv2.imshow("Resized image", resized)
    cv2.imshow("INTER_NEAREST image", resized1)
    cv2.imshow("INTER_LINEAR image", resized2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

0.3倍缩小，双线性插值

1.5倍放大，最近邻插值

1.5倍放大，双线性插值

- 推荐书籍：学习OpenCV中文版
- 推荐博客：https://blog.csdn.net/hongbin_xu/category_6936122.html

1.6 总结

插值算法是很多几何变换的基础和前置条件，对插值算法细节的掌握有助于对其他算法的理解，为自己的学习打下坚实的基础。

Task01 OpenCV框架与图像插值算法 END.

--- By: Aaron

> 博客：https://sandy1230.github.io/

> 博客：https://blog.csdn.net/weixin_39940512

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。