目录

第 3 章 opencv 基本理论知识

3.1 Mat 像素统计技术

3.1.1 opencv 常用类和方法

3.1.1.1 Mat 类

Mat 是一个基本图像容器，也是一个类，数据由两个部分组成：

矩阵头（包含矩阵尺寸，存储方法，存储地址等信息）和一个指向存储所有像素值的矩阵（根据所选存储方法的不同矩阵可以是不同的维数）的指针。

3.1.1.2 imread()

```
Mat imread( const String& filename, int flags = IMREAD_COLOR );
```

*功能：*读取图片文件中的数据

参数：

filename：图片路径

flags：指定读取图片的颜色类型，注意：opencv 版本不同，宏可能不一样　　opencv3.2.0 中

IMREAD_GRAYSCALE 值为 0，表示显示灰度图

IMREAD_COLOR 值为 1，表示显示原图

*返回值：*Mat 类对象

3.1.1.3 imshow()

```
void imshow(const String& winname, InputArray mat);
```

*功能：*显示照片

参数：

winname：显示照片窗口的名称

mat：imread 的返回值

*返回值：*无

3.1.1.4 waitKey()

```
int waitKey(int delay = 0)
```

*功能：*在一个给定的时间内(单位 ms)等待用户按键触发,如果用户没有按下键,则一直阻塞

参数：

delay：用于设置在显示完一帧，图像后程序等待“delay”ms 再显示下一帧视频，如果 waitKey（0）则只会显示第一帧视频

*返回值：*按键的 ASCII 码值

3.1.1.5 putText()

```
void putText( Mat& img, const string& text,
              Point org,  int fontFace,
              double fontScale,    Scalar color,
              int thickness=1, int lineType=8  );
```

*功能：*在图像上添加文字

参数：

img：待添加文字的图像

text：字符串,不支持中文

org：待写入的首字符左下角坐标

fontFace：字体类型，FONT_HERSHEY_SIMPLEX ，FONT_HERSHEY_PLAIN ，FONT_HERSHEY_DUPLEX 等

fontScale：字体大小

color：字体颜色，颜色用 Scalar（BGR）表示

Thickness：字体粗细

lineType：线型，默认值是 8

*返回值：*无

3.1.2 opencv 代码编译

案例：

```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace std;
using namespace cv;

int main(int argc, char const *argv[])
{
    //实例化对象保存图片的信息
    Mat image = imread("./dog.jpg");

    //判断图片是否为空
    if(image.empty()){
        cout << "Do not load image..." << endl;
        return -1;
    }
    //显示图片
    imshow("This is a image", image);

    //等待按键被按下，如果不按下，则阻塞
```

```
    //waitKey(0);
#if 1
    //键盘任意键按下或者 5 秒后代码继续运行
    waitKey(5000);
    image = imread("./dog.jpg", 0);
    imshow("This is a image", image);
    waitKey(0);
#endif
    return 0;
}
```

3.1.2.1 直接命令行编译

```
g++  -o opencv_t  t.cpp   `pkg-config --cflags --libs opencv`
```

3.1.2.2 通过 Makefile 编译

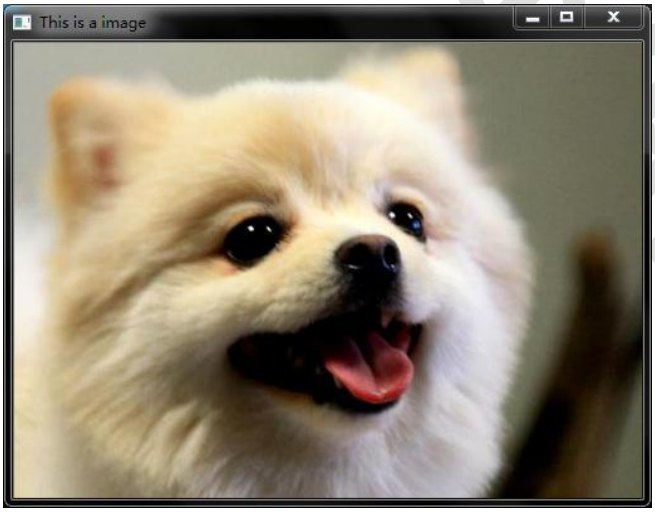
从 opencv-3.2.0/samples/cpp/example_cmake/拷贝一个 CMakeLists.txt，将文中三处 example 改成你的 cpp 文件名，并添加下面一行：

```
set(OpenCV_DIR /home/edu/ai/opencv-3.2.0/mybuild)
```

然后执行配置编译命令：

```
cmake .
make
```

运行结果：



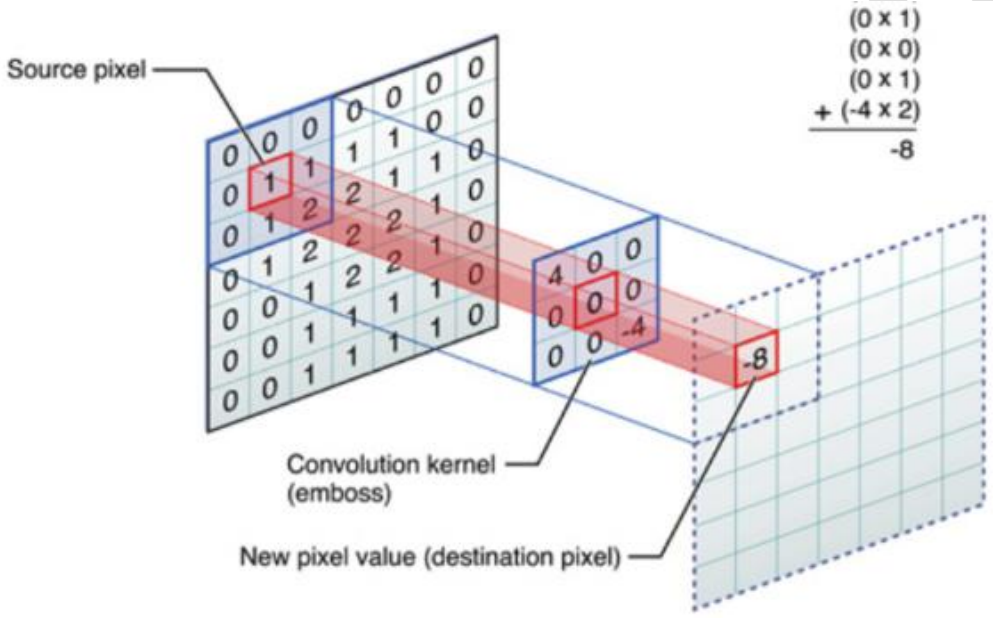
3.1.3 锐化操作

在对像素邻域进行计算时，通常用一个核心矩阵来表示。这个核心矩阵展现了如何将与计算相关的像素组合起来，才能得到预期结果。针对本节使用的锐化滤波器，核心矩阵可以是这样的：

0	-1	0

-1	5	-1
0	-1	0

除非另有说明，当前像素用核心矩阵中心单元格表示，又叫卷积核或相关核，核心矩阵中的每个单元格表示相关像素的乘法系数，结果像素通过核心矩阵得到的结果，即是这些乘积的累加。核心矩阵的大小就是邻域的大小（这里是 3×3），计算过程如下：



滤波处理接口 void filter2D()

```
void filter2D( InputArray src, OutputArray dst, int ddepth, InputArray kernel,Point anchor
= Point(-1,-1), double delta = 0, int borderType = BORDER_DEFAULT );
```

*功能：*通过给定卷积核对图像进行滤波处理

参数：

src：源图像

dst：与 src 相同大小、相同通道数的输出图像

ddepth：目标图像期望的深度（一般保持和源图像一致）

kernel：构造核心矩阵内核（协同图像传入 filter2D）

delta：选填过滤后的像素（一般取缺省值 0）

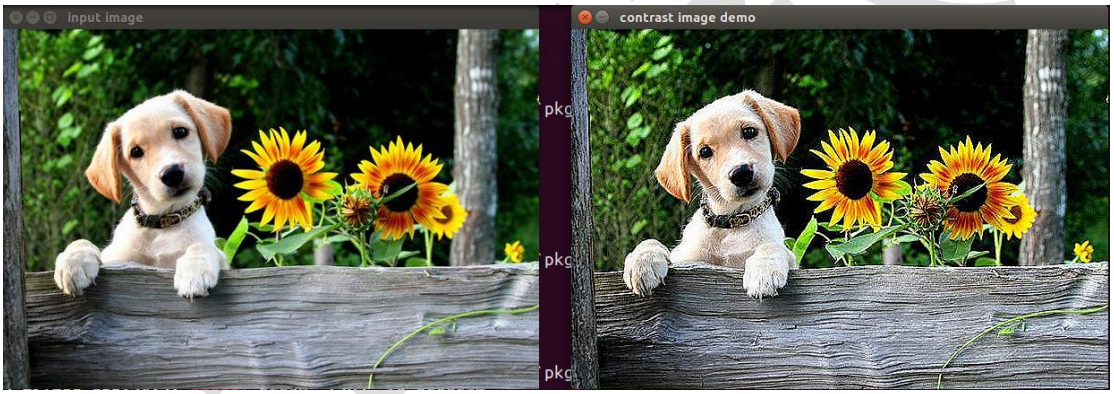
borderType：边框（一般取缺省值 BORDER_DEFAULT）

案例：

```
#include<iostream>
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;

int main(int argc, char const *argv[])
{
    if(argc < 2){
        cout<<"./opencv t  *.jpg"<<endl;
        return -1;
    }
    Mat src,dst;
    //读取照片
    src = imread(argv[1]);
    if(!src.data){
        cout<<"cound not load image..."<<endl;
        return -1;
    }
    imshow("input image", src);
    //锐化方式
    Mat kernel = (Mat <char>(3,3)<< 0,-1,0,-1,5,-1,0,-1,0);
    filter2D(src, dst, src.depth(), kernel3);
    imshow("contrast image demo", dst);
    waitKey(0);
    return 0;
}
```

执行结果



还可以做如下尝试：

```
    Mat kernel = Mat::ones(5,5,CV_32F)/(float)(25);//模糊处理
    Mat kernel = (Mat_<int>(2,2)<< 1,0,0,-1);//边缘处理
```

3.1.4 图像重叠操作

像素混合接口 void addWeighted()

```
void addWeighted(InputArray src1, double alpha, InputArray src2,double beta, double gamma,
OutputArray dst, int dtype = -1);
```

*功能：*图像合并处理,合并图片

参数：

src1： 源图像 1

alpha： 图像的权重(0~1 之间)

src2： 源图像 2

beta： 第二张源图像的权重(一般为 1.0-alpha)

gamma： 如果两张相加后亮度不理想 可以使用 gamma 使其亮度效果更好，设置亮度

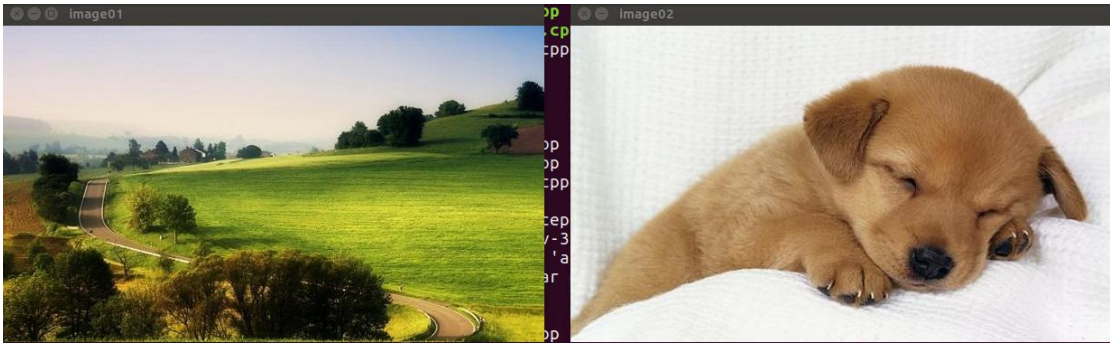
dst： 两张图像输出的目标图像

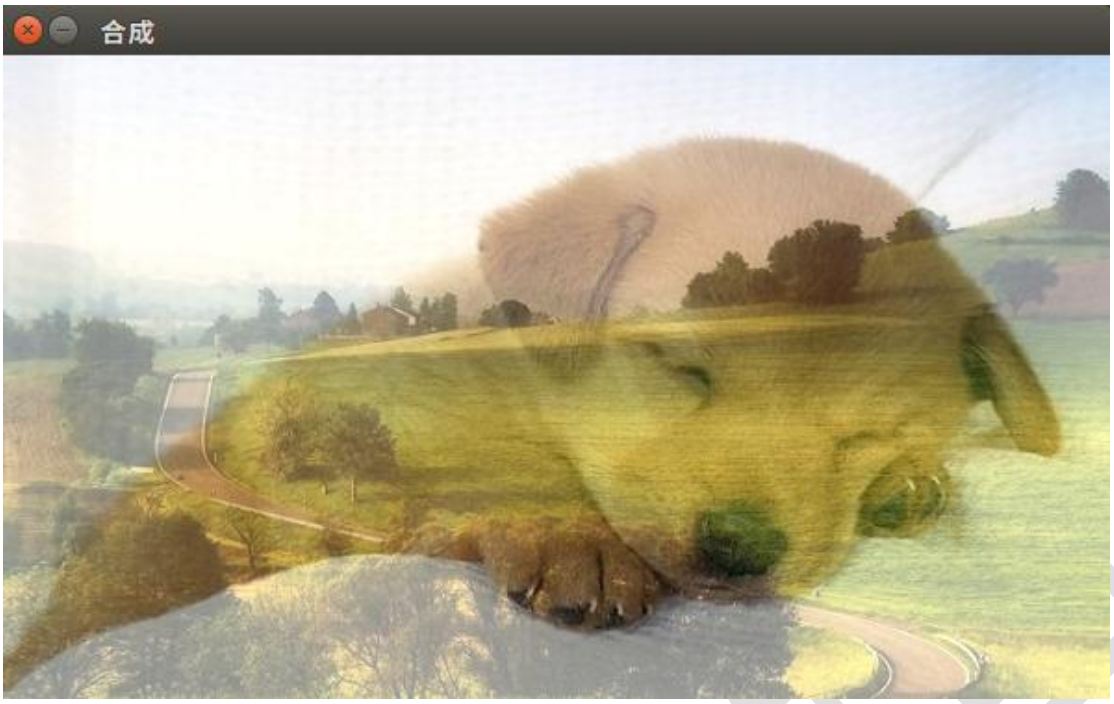
dtype: 使用缺省值

案例:

```
#include<iostream>
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main(int argc, char const *argv[])
{
    if(argc < 3){
        cout<<"./opencv t  *.jpg *.jpg"<<endl;
        return -1;
    }
    Mat src1,src2, dst;
    //读取第一张照片
    src1 = imread(argv[1]);
    if(!src1.data){
        cout<<"cound not load image..."<<endl;
        return -1;
    }
    //读取第二张照片
    src2 = imread(argv[2]);
    if(!src2.data){
        cout<<"cound not load image..."<<endl;
        return -1;
    }
    //必须保证图片大小一致
    if(src1.rows != src2.rows || src1.cols != src2.cols || src1.type() != src2.type()){
        resize(src1, src1,Size(300,300));
        resize(src2, src2,Size(300,300));
    }
    double alpha = 0.5;
    addWeighted(src1, alpha, src2, 0.6, 0.0, dst);
    imshow("src1", src1);
    imshow("src2", src2);
    imshow("合成", dst);
    waitKey(0);
    return 0;
}
```

执行结果





3.2 计算 N 维数据关系

统计学里最基本的概念就是样本的均值、方差、标准差、协方差等，下面我们给定一个包含 n 个样本的集合，分别进行分析。

3.2.1 均值

未经分组的均值计算公式：

$$\overline{x}=\frac{x_1+x_2+\cdots x_n}{n}=\frac{\sum_{i=1}^{n}x_i}{n}$$

3.2.2 方差

均值描述的是样本集合的中间点,它告诉我们的信息是有限的,而方差给我们描述的是样本集合的各个样本点到均值之间的平均距离。单一正态总体方差计算公式：

$$s^2 = \frac{\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2}{n-1}$$

3.2.3 标准差

方差对平均距离计算了平方，为了还原回原来的数量级，就有了标准差，标准差是对方差开根号，计算公式：

$$s = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2}{n-1}}$$

标准差描述各个点与均值距离的平均值，值越小表示数据越集中，例如：[0，8，12，20]和[8，9，11，12]，两个集合的均值都是 10，但显然两个集合各个值差别是很大的，计算两者的标准差，前者是 8.3，后者是 1.8，显然后者较为集中。

3.2.4 协方差

如果有另一个样本集合，也就是两个以上的样本集合，那么这两个样本集合间的各个点在时间或空间上有什么关系，其中一个样本中的点会不会像另一个样本中的点保持着一样的变化趋势，均值、方差、标准差都是解决一维内部各数据间的相关性问题（我们村的贫富差距问题）。当出现多维集合时，各个维度间的数据有无关联，可以参照一维的方法，首先将每个维度样本集合中每一个点的数据值减去该维度的平均值，再乘以另外一个维度的同样的差值，最后除以 n-1 就是协方差(n 就是每个维度样本个数，各维度一样)，这个协方差就可以反映两个维度间各数据的相关性，计算公式：

$$\mu_x = \frac{\sum_{i=0}^{n} x_i}{n} \qquad \mu_y = \frac{\sum_{i=0}^{n} y_i}{n}$$

$$\mathrm{COV}_{xy} = \frac{\sum_{i=0}^{n}(x_i - \mu_x)(y_i - \mu_y)}{(n-1)}$$

协方差的结果有什么意义？如果结果为正值，则说明两者是正相关的，如果结果为负值，则说明两者是负相关的，如果结果为 0，则表示两者之间没有关系。协方差只是说明了线性相关的方向问题，即从正无穷到负无穷，不能说明相关的程度，因为这个值可能很大也可能很小，所以还引出了相关系数=两个维度的协方差/(两个维度的标准差)，其值始终在-1 到 1 之间变化。

3.2.5 协方差矩阵

当出现多维数据时，若要对多维数据的相关性进行分析，那么就要用到协方差矩阵。

$$V = \begin{bmatrix} \Sigma x_1^2/N & \Sigma x_1 x_2/N & \dots & \Sigma x_1 x_c/N \\ \Sigma x_2 x_1/N & \Sigma x_2^2/N & \dots & \Sigma x_2 x_c/N \\ \dots & \dots & \dots & \dots \\ \Sigma x_c x_1/N & \Sigma x_c x_2/N & \dots & \Sigma x_c^2/N \end{bmatrix}$$

Student	Math	English	Art
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

$$\Rightarrow \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix} \quad A$$

$$a = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix} (1/5)$$

$$a = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix} - \begin{bmatrix} 66 & 60 & 60 \\ 66 & 60 & 60 \\ 66 & 60 & 60 \\ 66 & 60 & 60 \\ 66 & 60 & 60 \end{bmatrix} = \begin{bmatrix} 24 & 0 & 30 \\ 24 & 30 & -30 \\ -6 & 0 & 0 \\ -6 & 0 & 30 \\ -36 & -30 & -30 \end{bmatrix}$$

$$a'a = \begin{bmatrix} 24 & 24 & -6 & -6 & -36 \\ 0 & 30 & 0 & 0 & -30 \\ 30 & -30 & 0 & 30 & -30 \end{bmatrix} \begin{bmatrix} 24 & 0 & 30 \\ 24 & 30 & -30 \\ -6 & 0 & 0 \\ -6 & 0 & 30 \\ -36 & -30 & -30 \end{bmatrix} = \begin{bmatrix} 2520 & 1800 & 900 \\ 1800 & 1800 & 0 \\ 900 & 0 & 3600 \end{bmatrix}$$

$$V = a'a/n = \begin{bmatrix} 2520/5 & 1800/5 & 900/5 \\ 1800/5 & 1800/5 & 0/5 \\ 900/5 & 0/5 & 3600/5 \end{bmatrix} = \begin{bmatrix} 504 & 360 & 180 \\ 360 & 360 & 0 \\ 180 & 0 & 720 \end{bmatrix}$$

通过 opencv 接口完成计算的例子：

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

using namespace std;
using namespace cv;

int main(int argc, char const *argv[])
{
    Mat samples = (Mat <double>(5, 3) << 90, 60, 90, 90, 90, 30, 60, 60, 60, 60, 60, 90,
30, 30, 30);
    Mat cov, mu;
    //mu: 保存均值
    //cov: 保存协方差
    calcCovarMatrix(samples, cov, mu, CV COVAR NORMAL | CV COVAR ROWS);//CV COVAR COLS
    cout << "means : "<< endl;
    cout << mu << endl;
    cout << cov << endl;
    cout << "cov : " << endl;
    cout << cov/5 << endl;
    waitKey(0);
    return 0;
}
```

执行结果

```
edu@edu:~/debug/opencv$ ./opencv_t
means :
[66, 60, 60]
[2520, 1800, 900;
 1800, 1800, 0;
 900, 0, 3600]
cov :
[504, 360, 180;
 360, 360, 0;
 180, 0, 720]
edu@edu:~/debug/opencv$
```

3.3 特征值和特征向量

3.3.1 特征值与特征向量的定义

Ax= λ x

设 A 是 n 阶方阵，如果数值 λ 和 n 维非零列向量 x 使关系式 Ax= λ x 成立(即只伸缩不旋转)，那么这样的数 λ 称为矩阵 A 的“特征值”。

$$\begin{bmatrix} .8 & .3 \\ .2 & .7 \end{bmatrix} \quad \begin{bmatrix} .70 & .45 \\ .30 & .55 \end{bmatrix} \quad \begin{bmatrix} .650 & .525 \\ .350 & .475 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} .6000 & .6000 \\ .4000 & .4000 \end{bmatrix}$$

$$A \qquad A^2 \qquad A^3 \qquad\qquad A^{100}$$

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ A\mathbf{x} - \lambda\mathbf{x} &= 0 \\ A\mathbf{x} - \lambda I\mathbf{x} &= 0 \\ (A - \lambda I)\mathbf{x} &= 0 \end{aligned}$$

定义 设 $A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$，则

称 $|A-\lambda I|=\begin{vmatrix} a_{11}-\lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22}-\lambda & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn}-\lambda \end{vmatrix}$

为矩阵 A 的特征多项式，记作 $f(\lambda)$

$$A=\begin{bmatrix}3&6&-8\\0&0&6\\0&0&2\end{bmatrix} \qquad \det(A-\lambda I)$$

$$A-\lambda I=\begin{bmatrix}3&6&-8\\0&0&6\\0&0&2\end{bmatrix}-\begin{bmatrix}\lambda&0&0\\0&\lambda&0\\0&0&\lambda\end{bmatrix}=\begin{bmatrix}3-\lambda&6&-8\\0&-\lambda&6\\0&0&2-\lambda\end{bmatrix}$$

$$\begin{aligned}\det(A-\lambda I)&=(3-\lambda)(-\lambda)(2-\lambda)+(6)(6)(0)+(-8)(0)(0)\\&\quad-(0)(-\lambda)(-8)-(0)(6)(3-\lambda)-(-\lambda)(0)(6)\\&=-\lambda(3-\lambda)(2-\lambda)\end{aligned}$$

特征值有三个 $\lambda=0,\ 2,\ \text{or}\ 3$

当特征值为0时

$$\begin{aligned}(A-\lambda I)\mathbf{x}&=\mathbf{0}\\(A-0I)\mathbf{x}&=\mathbf{0}\\A\mathbf{x}&=\mathbf{0}\end{aligned}$$

$$\begin{bmatrix}3&6&-8\\0&0&6\\0&0&2\end{bmatrix}\mathbf{x}=\mathbf{0}$$

$$\mathbf{x}=\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=x_2\begin{bmatrix}-2\\1\\0\end{bmatrix}$$

当特征值为2时

$$\begin{aligned}(A-\lambda I)\mathbf{x}&=\mathbf{0}\\(A-2I)\mathbf{x}&=\mathbf{0}\end{aligned}$$

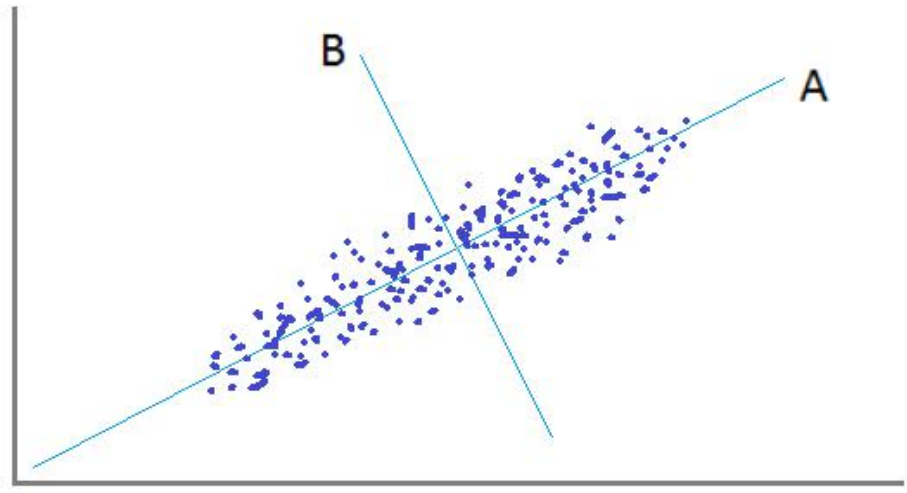
$$\left(\begin{bmatrix}3&6&-8\\0&0&6\\0&0&2\end{bmatrix}-\begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}\right)\mathbf{x}=\mathbf{0}$$

$$\begin{bmatrix}1&6&-8\\0&-2&6\\0&0&0\end{bmatrix}\mathbf{x}=\mathbf{0}$$

最终得到特征值为2时特征向量

$$\mathbf{x}=\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=x_3\begin{bmatrix}-10\\3\\1\end{bmatrix}.$$

计算得到特征值和特征向量的意义？特征值与特征向量表达了一个线性变换的特征，特征向量将一个矩阵进行正交分解，判断出在哪些方向只拉伸不扭曲来简化计算量，得到了特征值与特征向量就是得到了某个矩阵导致的伸缩比例和伸缩方向，其目的主要用于降维。



上图通过分析特征值与特征向量就可以将二维数据变成一维数据分析。

3.3.2 opencv 接口计算特征值与特征向量

```
bool eigen(InputArray src, OutputArray eigenvalues, OutputArray eigenvectors = noArray());
```

*功能：*获取特征值和特征向量

参数：

src：原图或者数据

eigenvalues：特征值

eigenvectors：特征向量

案例

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

using namespace std;
using namespace cv;

int main(int argc, char const *argv[])
{
    Mat data = (Mat <double>(2, 2) <<
    1, 2,
    2, 1);
    Mat eigen values, eigen vector;
    eigen(data, eigen values, eigen vector);
    for(int i = 0; i < eigen values.rows; i++){
        printf("eigen value %d: %.3f\n", i, eigen values.at<double>(i));
    }
    cout << "eigen vector: " << endl << eigen vector << endl;
    waitKey(0);
    return 0;
}
```

执行结果

```
lzx@qfedu:~/share/work/test/opencv$ ./a.out
eigen value 0: 3.000
eigen value 1: -1.000
eigen vector:
[0.7071067811865475, 0.7071067811865475;
 0.7071067811865475, -0.7071067811865475]
lzx@qfedu:~/share/work/test/opencv$
```

可以将矩阵扩大两倍看看，得到的特征值与特征向量如何变化。