Министерство образования и науки Российской Федерации

Южно-Уральский государственный университет

Кафедра Автоматика и Управление

Курсовая работа

на тему

Реализация метода главных компонент с помощью библиотеки OpenCV

Выполнил: Пушников А.А.

Группа: ПС-669

Проверил Разнополов К.О.

Дата «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2006 г.

Челябинск

2006 гОглавление

[Метод главных компонент 2](#_Toc152777056)

[Реализация метода главных компонент в OpenCV 3](#_Toc152777057)

[Текст программы 4](#_Toc152777058)

# Метод главных компонент

Метод главных компонент (Principal Component Analysis, PCA) применяется для сжатия информации без существенных потерь информативности. Он состоит в линейном ортогональном преобразовании входного вектора X размерности N в выходной вектор Y размерности M, N. При этом компоненты вектора Y являются некоррелированными и общая дисперсия после преобразования остаётся неизменной. Матрица X состоит из всех примеров изображений обучающего набора. Решив уравнение , получаем матрицу собственных векторов , где – ковариационная матрица для X, а – диагональная матрица собственных чисел. Выбрав из подматрицу , соответствующую M наибольшим собственным числам, получим, что преобразование , где – нормализованный вектор с нулевым математическим ожиданием, характеризует большую часть общей дисперсии и отражает наиболее существенные изменения X.



Выбор первых M главных компонент разбивает векторное пространство на главное (собственное) пространство , содержащее главные компоненты, и его ортогональное дополнение .



Применение для задачи распознавания изображений имеет следующий вид. Входные вектора представляют собой отцентрированные и приведённые к единому масштабу изображения. Собственные вектора, вычисленные для всего набора изображений, называются собственными объектами (eigenobject). С помощью вычисленных ранее матриц входное изображение разлагается на набор линейных коэффициентов, называемых главными компонентами. Сумма главных компонент, умноженных на соответствующие собственные вектора, является реконструкцией изображения.

Для каждого изображения лица вычисляются его главные компоненты. Обычно берётся от 5 до 200 главных компонент. Остальные компоненты кодируют мелкие различия между эталоном и шум. Процесс распознавания заключается в сравнении главных компонент неизвестного изображения с компонентами всех остальных изображений. Для этого обычно применяют какую-либо метрику (простейший случай – Евклидово расстояние). При этом предполагается, что изображения, соответствующие одному эталону, сгруппированы в кластеры в собственном пространстве. Из базы данных (или тренировочного набора) выбираются изображения-кандидаты, имеющие наименьшее расстояние от входного (неизвестного) изображения.

Дальнейшее совершенствование заключалось в использовании метрики Махаланобиса и Гауссовского распределения для оценки близости изображений. Для учёта различных ракурсов в этой же работе использовалось многомодальное распределение изображений в собственном пространстве.

Основное преимущество применения анализа главных компонент – это хранение и поиск изображений в больших базах данных, реконструкция изображений.

Основной недостаток – высокие требования к условиям съёмки изображений. Изображения должны быть получены в близких условиях освещённости, одинаковом ракурсе. Должна быть проведена качественная предварительная обработка, приводящая изображения к стандартным условиям (масштаб, поворот, центрирование, выравнивание яркости, отсечение фона).

# Реализация метода главных компонент в OpenCV

Библиотека OpenCV реализует описанный выше алгоритм следующими функциями:

Функция, вычисляет собственные объекты эталонов:

void cvCalcEigenObjects( int nObjects, void\* input, void\* output, int ioFlags, int ioBufSize, void\* userData, CvTermCriteria\* calcLimit, IplImage\* avg, float\* eigVals ),

где

nObjects – число эталонов

input - указатель на массив изображений-эталонов (изображения глубиной 8 бит)

output – (выход функции) указатель на массив собственных объектов (изображения глубиной 32 бит)

ioFlags – флаги ввода/вывода. Для работы с памятью.

ioBufSize - размер буфера. Для работы с памятью.

userData – указатель на структуру для работы с памятью.

calcLimit – критерий прекращения вычислений. Два варианта: по количеству итераций и по ко точности (?)

avg – (выход функции) усредненное изображение эталонов

eigVals (выход функции) указатель на собственные числа (может быть NULL)

Функция, вычисляет коэффициенты разложения:

void cvEigenDecomposite( IplImage\* obj, int eigenvec\_count, void\* eigInput, int ioFlags, void\* userData, IplImage\* avg, float\* coeffs ),

где

obj – исследуемое изображение

eigenvec\_count – число собственных объектов

eigInput - указатель на массив собственных объектов (изображения глубиной 32 бит)

ioFlags – флаги ввода/вывода. Для работы с памятью.

userData – указатель на структуру для работы с памятью.

avg - (выход функции) усредненное изображение эталонов

coeffs - (выход функции) коэффициенты разложения (?)

Функция, вычисляет проекцию исследуемого изображения на пространство собственных объектов:

void cvEigenProjection( void\* input\_vecs, int eigenvec\_count, int io\_flags, void\* userdata, float\* coeffs, IplImage\* avg, IplImage\* proj ),

где

input\_vec - указатель на массив собственных объектов (изображения глубиной 32 бит)

eigenvec\_count – число собственных объектов

io\_flags – флаги ввода/вывода. Для работы с памятью.

userdata – указатель на структуру для работы с памятью.

coeffs - коэффициенты разложения (?)

avg - усредненное изображение эталонов

proj - проекция исследуемого изображения на пространство собственных объектов

В полученной проекции имеет смысл убрать излишние компоненты (например, с помощью функции cvThreshold – отсечение по порогу). Далее полученный результат можно сравнивать с эталонами, для принятия решения. Способов сравнения много, это может быть, например, минимальное расстояние (Евклидово) или корреляция с эталонами.

# Текст программы

//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

#include "cxcore.h"

#include "cv.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

IplImage \*\*Objs, \*Pro, \*Object;

int obj\_number=3;

HINSTANCE highgui,cv,cvaux;

IplImage\* (\_\_stdcall \*cvLoadImage)( const char\* filename, int iscolor);

int (\_\_stdcall \*cvSaveImage)( const char\* filename, const CvArr\* image);

int (\_\_stdcall \*cvNamedWindow)( const char\* name, int flags );

void (\_\_stdcall \*cvShowImage)( const char\* name, const CvArr\* image );

IplImage\* (\_\_stdcall \*cvCreateImage\_)( CvSize size, int depth, int channels );

double (\_\_stdcall \*cvDotProduct\_)(const CvArr\* src1, const CvArr\* src2 );

void (\_\_stdcall \*cvMul\_)(const CvArr\* src1, const CvArr\* src2, CvArr\* dst, double scale=1 );

void (\_\_stdcall \*cvThreshold\_)(const CvArr\* src, CvArr\* dst, double threshold,double max\_value, int threshold\_type);

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

//---------------------------------------------------------------------------

void show\_im(TCanvas\*c,IplImage \*p)

{

for(int i=0;i<p->width;i++)

for(int j=0;j<p->height;j++)

{

int a=p->imageDataOrigin[p->widthStep\*j+i];

c->Pixels[i][j]=a&0x0000ff|(a<<8)&0x00ff00|(a<<16)&0xff0000;

}

}

void pca(int obj\_number, IplImage \*\*Objs,CvTermCriteria limit, IplImage \*Object,IplImage \*Pro)

{

CvSize size;

int m1=obj\_number;

IplImage \*\*EigObjs, \*Avg;

float \*coeffs;

HINSTANCE hDLL = LoadLibrary("cvaux100.dll");

if (!hDLL) return;

void (\_\_stdcall \*cvCalcEigenObjects)( int nObjects, void\* input, void\* output, int ioFlags, int ioBufSize, void\* userData, CvTermCriteria\* calcLimit, IplImage\* avg, float\* eigVals );

cvCalcEigenObjects = (void(\_\_stdcall \*)( int nObjects, void\* input, void\* output, int ioFlags, int ioBufSize, void\* userData, CvTermCriteria\* calcLimit, IplImage\* avg, float\* eigVals ))GetProcAddress(hDLL, "cvCalcEigenObjects");

if (!cvCalcEigenObjects) return;

void (\_\_stdcall \*cvEigenDecomposite)( IplImage\* obj, int nEigObjs, void\* eigInput, int ioFlags, void\* userData, IplImage\* avg, float\* coeffs );

cvEigenDecomposite = (void(\_\_stdcall \*)( IplImage\* obj, int nEigObjs, void\* eigInput, int ioFlags, void\* userData, IplImage\* avg, float\* coeffs ))GetProcAddress(hDLL, "cvEigenDecomposite");

if (!cvEigenDecomposite) return;

void (\_\_stdcall \*cvEigenProjection)( void\* eigInput, int nEigObjs, int ioFlags, void\* userData, float\* coeffs, IplImage\* avg, IplImage\* proj );

cvEigenProjection = (void(\_\_stdcall \*)( void\* eigInput, int nEigObjs, int ioFlags, void\* userData, float\* coeffs, IplImage\* avg, IplImage\* proj ))GetProcAddress(hDLL, "cvEigenProjection");

if (!cvEigenProjection) return;

EigObjs=new IplImage\*[m1];

coeffs=new float[m1];

size.width = Object->width; size.height = Object->height;

Avg = cvCreateImage\_( size, IPL\_DEPTH\_32F, 1 );

for(int i=0; i<m1; i++ )

{

EigObjs[i] = cvCreateImage\_( size, IPL\_DEPTH\_32F, 1 );

}

cvCalcEigenObjects( obj\_number, (void\*)Objs, (void\*)EigObjs, 0, 0, NULL, &limit, Avg, NULL );

cvEigenDecomposite( Object, m1, (void\*)EigObjs, 0, NULL, Avg, coeffs );

cvEigenProjection ( (void\*)EigObjs, m1, 0, NULL, coeffs, Avg, Pro );

FreeLibrary(hDLL);

// cvReleaseImage( &Avg );

// for(int i=0; i<m1; i++ )

// {

// cvReleaseImage( &EigObjs[i] );

// }

// cvFree( &coeffs);

}

void \_\_fastcall TForm1::FormCreate(TObject \*Sender)

{

highgui = LoadLibrary("highgui100.dll");

if (!highgui) return;

cvLoadImage = (IplImage\*(\_\_stdcall \*)( const char\* filename, int iscolor))GetProcAddress(highgui, "cvLoadImage");

if (!cvLoadImage) return;

cvSaveImage = (int(\_\_stdcall \*)( const char\* filename, const CvArr\* image))GetProcAddress(highgui, "cvSaveImage");

if (!cvSaveImage) return;

cvNamedWindow = (int(\_\_stdcall \*)( const char\* name, int flags ))GetProcAddress(highgui, "cvNamedWindow");

if (!cvNamedWindow) return;

cvShowImage = (void(\_\_stdcall \*)( const char\* name, const CvArr\* image ))GetProcAddress(highgui, "cvShowImage");

if (!cvShowImage) return;

cv = LoadLibrary("cxcore100.dll");

if (!cv) return;

cvCreateImage\_ = (IplImage\*(\_\_stdcall \*)( CvSize size, int depth, int channels ))GetProcAddress(cv, "cvCreateImage");

if (!cvCreateImage\_) return;

cvDotProduct\_ = (double(\_\_stdcall \*)( const CvArr\* src1, const CvArr\* src2))GetProcAddress(cv, "cvDotProduct");

if (!cvDotProduct\_) return;

cvMul\_ = (void(\_\_stdcall \*)( const CvArr\* src1, const CvArr\* src2, CvArr\* dst, double scale=1))GetProcAddress(cv, "cvMul");

if (!cvMul\_) return;

cvaux = LoadLibrary("cv100.dll");

if (!cvaux) return;

cvThreshold\_ = (void(\_\_stdcall \*)(const CvArr\* src, CvArr\* dst, double threshold,double max\_value, int threshold\_type))GetProcAddress(cvaux, "cvThreshold");

if (!cvThreshold\_) return;

Objs=new IplImage\*[obj\_number];

Objs[0] = cvLoadImage( ".\\et\\1.bmp", 0);

show\_im(Image1->Canvas,Objs[0]);

Objs[1] = cvLoadImage( ".\\et\\2.bmp", 0);

show\_im(Image2->Canvas,Objs[1]);

Objs[2] = cvLoadImage( ".\\et\\3.bmp", 0);

show\_im(Image3->Canvas,Objs[2]);

String fname="6.bmp";

Object = cvLoadImage((".\\in\\"+fname).c\_str(), 0);

show\_im(Image4->Canvas,Object);

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

float e[3];

CvTermCriteria limit;

CvSize size;

size.width = Object->width; size.height = Object->height;

Pro = cvCreateImage\_( size, IPL\_DEPTH\_8U, 1 );

limit.type = CV\_TERMCRIT\_EPS;

limit.max\_iter = 1;

limit.epsilon = 0.1;

show\_im(Image4->Canvas,Object);

pca(obj\_number,Objs,limit,Object,Pro);

show\_im(Image5->Canvas,Pro);

cvThreshold\_(Pro,Object,200,255,CV\_THRESH\_BINARY);

show\_im(Image6->Canvas,Object);

cvMul\_(Object,Objs[0],Pro);

show\_im(Image7->Canvas,Pro);

cvMul\_(Object,Objs[1],Pro);

show\_im(Image8->Canvas,Pro);

cvMul\_(Object,Objs[2],Pro);

show\_im(Image9->Canvas,Pro);

e[0]=cvDotProduct\_(Object,Objs[0])/cvDotProduct\_(Objs[0],Objs[0]);

e[1]=cvDotProduct\_(Object,Objs[1])/cvDotProduct\_(Objs[1],Objs[1]);

e[2]=cvDotProduct\_(Object,Objs[2])/cvDotProduct\_(Objs[2],Objs[2]);

Label1->Caption=FloatToStr(int(e[0]\*1000)/1000.);

Label2->Caption=FloatToStr(int(e[1]\*1000)/1000.);

Label3->Caption=FloatToStr(int(e[2]\*1000)/1000.);

if(e[0]>e[1])

if(e[0]>e[2])

ShowMessage("1");

if(e[1]>e[0])

if(e[1]>e[2])

ShowMessage("2");

if(e[2]>e[1])

if(e[2]>e[0])

ShowMessage("3");

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Image1Click(TObject \*Sender)

{

if(OpenPictureDialog1->Execute())

{

Objs[0] = cvLoadImage(OpenPictureDialog1->FileName.c\_str(), 0);

show\_im(Image1->Canvas,Objs[0]);

}

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Image2Click(TObject \*Sender)

{

if(OpenPictureDialog1->Execute())

{

Objs[1] = cvLoadImage(OpenPictureDialog1->FileName.c\_str(), 0);

show\_im(Image2->Canvas,Objs[1]);

}

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Image3Click(TObject \*Sender)

{

if(OpenPictureDialog1->Execute())

{

Objs[2] = cvLoadImage(OpenPictureDialog1->FileName.c\_str(), 0);

show\_im(Image3->Canvas,Objs[2]);

}

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Image4Click(TObject \*Sender)

{

if(OpenPictureDialog1->Execute())

{

Object = cvLoadImage(OpenPictureDialog1->FileName.c\_str(), 0);

show\_im(Image4->Canvas,Object);

}

}

//---------------------------------------------------------------------------