Lọc Trung Bình

*import* cv2 *as* cv

*import* numpy *as* np

*from* matplotlib *import* pyplot *as* plt

*# Load and blur image*

img = cv.imread('rose\_gauss.jpg')

img2 = cv.imread('rose\_salt\_and\_pepper.jpg')

blur = cv.blur(img,(5,5))

blur2 = cv.blur(img2,(5,5))

*# Convert color from bgr (OpenCV default) to rgb*

img\_rgb = cv.cvtColor(img, cv.COLOR\_BGR2RGB)

blur\_rgb = cv.cvtColor(blur, cv.COLOR\_BGR2RGB)

img\_rgb2 = cv.cvtColor(img2, cv.COLOR\_BGR2RGB)

blur\_rgb2 = cv.cvtColor(blur2, cv.COLOR\_BGR2RGB)

*# Display*

plt.subplot(221),plt.imshow(img\_rgb),plt.title('Gauss Noise')

plt.xticks([]), plt.yticks([])

plt.subplot(222),plt.imshow(blur\_rgb),plt.title('Gauss Noise - Blurred')

plt.xticks([]), plt.yticks([])

plt.subplot(223),plt.imshow(img\_rgb2),plt.title('Salt&Pepper Noise')

plt.xticks([]), plt.yticks([])

plt.subplot(224),plt.imshow(blur\_rgb2),plt.title('Salt&Pepper Noise - Blurred')

plt.xticks([]), plt.yticks([])

plt.show()

Lọc Sobel

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <stdlib.h>

#include <stdio.h>

using namespace cv;

enum

{CV\_BGR2GRAY = 6};

int main(int argc, char\*\* argv)

{ Mat src, src\_gray;

Mat grad;

const char\* window\_name = "Sobel Demo - Simple Edge Detector";

int scale = 1;

int delta = 0;

int ddepth = CV\_16S;

/// Load an image

src = imread("long.jpg");

if (!src.data) return -1;

/// Convert it to gray

cvtColor(src, src\_gray, CV\_BGR2GRAY);

/// Create window

namedWindow(window\_name, WINDOW\_AUTOSIZE);

/// Generate grad\_x and grad\_y

Mat grad\_x, grad\_y;

Mat abs\_grad\_x, abs\_grad\_y;

/// Gradient X

Sobel(src\_gray, grad\_x, ddepth, 1, 0, 3, scale, delta, BORDER\_DEFAULT);

convertScaleAbs(grad\_x, abs\_grad\_x);

/// Gradient Y

Sobel(src\_gray, grad\_y, ddepth, 0, 1, 3, scale, delta, BORDER\_DEFAULT);

convertScaleAbs(grad\_y, abs\_grad\_y);

/// Total Gradient (approximate)

addWeighted(abs\_grad\_x, 0.5, abs\_grad\_y, 0.5, 0, grad);

imshow("Original", src);

imshow(window\_name, grad);

waitKey(0);

return 0;

}

**Lọc laplance**

#include "opencv2/imgproc.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

using namespace cv;

int main(int argc, char\*\* argv)

{ Mat src, srcc, src\_gray, dst;

int kernel\_size = 3;

int scale = 1;

int delta = 0;

int ddepth = CV\_16S;

const char\* window\_name = "Laplace Demo";

src = imread("long.jpg"); // Load an image

// Check if image is loaded fine

if (src.empty()) return -1;

// Reduce noise by blurring with a Gaussian filter ( kernel size = 3 )

cvtColor(src, src\_gray, COLOR\_BGR2GRAY); // Convert the image to grayscale

Mat abs\_dst;

Laplacian(src\_gray, dst, ddepth, kernel\_size, scale, delta, BORDER\_DEFAULT);

// converting back to CV\_8U

convertScaleAbs(dst, abs\_dst);

imshow("Original", src);

imshow(window\_name, abs\_dst);

waitKey(0);

return 0;

}

Lọc gause

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <stdlib.h>

#include <stdio.h>

using namespace cv;

enum

{CV\_BGR2GRAY = 6};

int main(int argc, char\*\* argv)

{ Mat src, s;

int ddepth = CV\_16S;

/// Load an image

src = imread("long.jpg");

if (!src.data) return -1;

//loc Gaussian

GaussianBlur(src, s, Size(3, 3), 0, 0, BORDER\_DEFAULT);

imshow("Original", src);

imshow("Gaussian", s);

waitKey(0);

return 0;

}

**Mở camera**

#include "opencv2/opencv.hpp"

using namespace cv;

int main(int argc, char\*\* argv)

{VideoCapture cam;

// Mở camera mặc định

// Kiểm tra tài liệu VideoCapture.

if (!cam.open(0)) return 0;

for (;;)//thời gian mở camera

{Mat frame;

cam >> frame;

if (frame.empty()) break; // kết thúc luồng video

imshow("show face", frame);

if (waitKey(10) == 13) break; // nhấn nút Enter để đóng camera

}

return 0;

}

**Nhận diện mặt**

#include <opencv2\opencv.hpp>

#include <vector>

using namespace cv;

using namespace std;

int main()

{double scale = 5.0;

CascadeClassifier face;//để phát hiện các đối tượng trong luồng video

face.load("C:\\opencv\\build\\etc\\haarcascades\\haarcascade\_frontalface\_alt.xml");//để tải tập tin phân loại .xml

//mở camera

VideoCapture cap(0);

if (!cap.isOpened()) return - 1;

for (;;)//Vòng lặp vô hạn từ webcam

{ Mat khung;

cap >> khung;

//tạo khung để nhận diện thấy khuôn mặt

Mat gray;

cvtColor(khung, gray, COLOR\_BGR2GRAY);//chuyển đổi không gian màu(1 số chức năng của opencv phải chuyển qua thang độ xám mới thực hiện đc)

resize(gray, gray, Size(gray.size().width / scale, gray.size().height / scale));//thay đổi kích thước

//kích thước hình ảnh đầu ra

//nơi chứa hình ảnh khi phát hiện mặt

vector<Rect> faces;

face.detectMultiScale(gray, faces, 1.1, 3, 0, Size(30, 30));//phát hiện mặt

//kích thước hình ảnh được giảm ở mỗi tỷ lệ hình ảnh.

//số lượng hàng xóm

//kích thước đối tượng tối thiểu, kích thước đối tượng tối đa

for (Rect area : faces)//vòng lặp xét khu vực của mặt

{

Scalar color = Scalar(0, 0, 255);//màu của khung là màu đỏ

//tạo khung hình chữ nhật

rectangle(khung, Point(cvRound(area.x \* scale), cvRound(area.y \* scale)),

Point(cvRound((area.x + area.width - 1) \* scale), cvRound((area.y + area.height - 1) \* scale)), color);

} //Point:đỉnh của hình chữ nhật(trên-trái); đỉnh đối diện của hình chữ nhật(dưới-phải)

imshow("Camera ", khung);

if (waitKey(30) >= 0) break;

}

return 0;

}

**Machine learing**

#include <opencv2/core.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/imgcodecs.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/ml.hpp>

using namespace cv;

using namespace cv::ml;

int main(int, char\*\*)

{// Set up training data

int labels[6] = {1, 1, -1, -1, 1, -1};

float trainingData[6][2] = { {500, 300}, {255, 10}, {501, 255}, {10, 100}, {250, 200}, {300,100} };

Mat trainingDataMat(6, 2, CV\_32F, trainingData);

Mat labelsMat(6, 1, CV\_32SC1, labels);

// Train the SVM

Ptr<SVM> svm = SVM::create();

svm->setType(SVM::C\_SVC);

svm->setKernel(SVM::LINEAR);

svm->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER, 100, 1e-6));

svm->train(trainingDataMat, ROW\_SAMPLE, labelsMat);

// Data for visual representation

int width = 512, height = 512;

Mat image = Mat::zeros(height, width, CV\_8UC3);

// Show the decision regions given by the SVM

Vec3b green(0,255,0), blue(255,0,0);

for (int i = 0; i < image.rows; i++)

{for (int j = 0; j < image.cols; j++)

{Mat sampleMat = (Mat\_<float>(1,2) << j,i);

float response = svm->predict(sampleMat);

if (response == 1)

image.at<Vec3b>(i,j) = green;

else if (response == -1)

image.at<Vec3b>(i,j) = blue;

}

}

// Show the training data

int thickness = -1;

circle( image, Point(500, 300), 5, Scalar(0, 0, 0), thickness );

circle( image, Point(255, 10), 5, Scalar(0, 0, 0), thickness );

circle( image, Point(501, 255), 5, Scalar(255, 255, 255), thickness );

circle( image, Point(10, 100), 5, Scalar(255, 255, 255), thickness );

circle( image, Point(250, 200), 5, Scalar(0, 0, 0), thickness);

circle( image, Point(300, 100), 5, Scalar(255, 255, 255), thickness);

// Show support vectors

thickness = 2;

Mat sv = svm->getUncompressedSupportVectors();

for (int i = 0; i < sv.rows; i++)

{const float\* v = sv.ptr<float>(i);

circle(image, Point( (int) v[0], (int) v[1]), 6, Scalar(128, 128, 128), thickness);

}

imwrite("result.png", image); // save the image

imshow("SVM Simple Example", image); // show it to the user

waitKey();

return 0;

}

**Histogram**

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/imgproc.hpp"

#include <iostream>

using namespace cv;

using namespace std;

int main(int argc, char\*\* argv)

{ CommandLineParser parser(argc, argv, "{@input |long.jpg | input image}");

Mat input = imread(samples::findFile(parser.get<String>("@input")), IMREAD\_COLOR);

if (input.empty())

{cout << "Could not open or find the image!\n" << endl;

cout << "Usage: " << argv[0] << " <Input image>" << endl;

return -1;

}

cvtColor(input, input, COLOR\_BGR2GRAY);

Mat dst;

equalizeHist(input, dst);//hàm histogram

imshow("Source image", input);

imshow("Equalized Image", dst);

waitKey();

return 0;

}

**Phân vùng**

#include <opencv2/opencv.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main(int, char\*\* argv)

{// Load the image

Mat src = imread("tien.jpg");

if (!src.data)

return -1;

// Show source image

imshow("Source Image", src);

for (int x = 0; x < src.rows; x++) {

for (int y = 0; y < src.cols; y++) {

if (src.at<Vec3b>(x, y) == Vec3b(255, 255, 255)) {

src.at<Vec3b>(x, y)[0] = 0;

src.at<Vec3b>(x, y)[1] = 0;

src.at<Vec3b>(x, y)[2] = 0;

}

}

}

// Tạo một hạt nhân mà chúng ta sẽ sử dụng để tích lũy / làm sắc nét hình ảnh của chúng ta

Mat kernel = (Mat\_<float>(3, 3) <<

1, 1, 1,

1, -8, 1,

1, 1, 1);

Mat imgLaplacian;

Mat sharp = src; //sao chép hình ảnh nguồn sang một cái khác tạm thời filter2D(sharp, imgLaplacian, CV\_32F, kernel);

src.convertTo(sharp, CV\_32F);

Mat imgResult = sharp - imgLaplacian;

imgResult.convertTo(imgResult, CV\_8UC3);

imgLaplacian.convertTo(imgLaplacian, CV\_8UC3);

src = imgResult; // copy back

// Create binary image from source image

Mat bw;

cvtColor(src, bw, COLOR\_BGR2GRAY);

// Perform the distance transform algorithm

Mat dist;

distanceTransform(bw, dist, 2, 3);

normalize(dist, dist, 0, 1.0, NORM\_MINMAX);

threshold(dist, dist, 0.4, 1.0, 0);

Mat kernel1 = Mat::ones(3, 3, CV\_8UC1);

dilate(dist, dist, kernel1);

Mat dist\_8u;

dist.convertTo(dist\_8u, CV\_8U);

vector<vector<Point> > contours;

Mat markers = Mat::zeros(dist.size(), CV\_32SC1);

for (size\_t i = 0; i < contours.size(); i++)

drawContours(markers, contours, static\_cast<int>(i), Scalar::all(static\_cast<int>(i) + 1), -1);

circle(markers, Point(5, 5), 3, CV\_RGB(255, 255, 255), -1);

watershed(src, markers);

Mat mark = Mat::zeros(markers.size(), CV\_8UC1);

markers.convertTo(mark, CV\_8UC1);

bitwise\_not(mark, mark);

vector<Vec3b> colors;

for (size\_t i = 0; i < contours.size(); i++)

{int b = theRNG().uniform(0, 255);

int g = theRNG().uniform(0, 255);

int r = theRNG().uniform(0, 255);

colors.push\_back(Vec3b((uchar)b, (uchar)g, (uchar)r));

}

// Create the result image

Mat dst = Mat::zeros(markers.size(), CV\_8UC3);

// Fill labeled objects with random colors

for (int i = 0; i < markers.rows; i++)

{for (int j = 0; j < markers.cols; j++)

{int index = markers.at<int>(i, j);

if (index > 0 && index <= static\_cast<int>(contours.size()))

dst.at<Vec3b>(i, j) = colors[index - 1];

else

dst.at<Vec3b>(i, j) = Vec3b(0, 0, 0);

}

}

// Visualize the final image

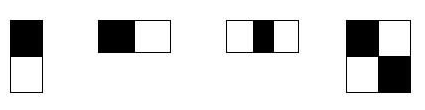
imshow("Final Result", dst);

waitKey(0);

return 0;

}

Các đặc trưng Haar-Like là những hình chữ nhật được phân thành các vùng khác nhau, gồm 4 đặc trưng cơ bản để xác định khuôn mặt người. Mỗi đặc trưng Haar-Like là sự kết hợp của hai hay ba hình chữ nhật trắng hay đen.



Dùng các đặc trưng trên, ta có thể tính được các giá trị của đặc trưng Haar-Like là sự chênh lệch giữa tổng của các pixel của vùng đen và vùng trắng.



Đặc điểm đầu tiên được chọn dường như tập trung vào đặc tính là vùng mắt thường tối hơn vùng mũi và má. Đặc điểm thứ hai được lựa chọn dựa vào đặc tính là mắt tối hơn sống mũi.

Vị trí của các hình chữ nhật này được xác định liên quan đến cửa sổ phát hiện hoạt động giống như một hộp giới hạn với đối tượng đích.

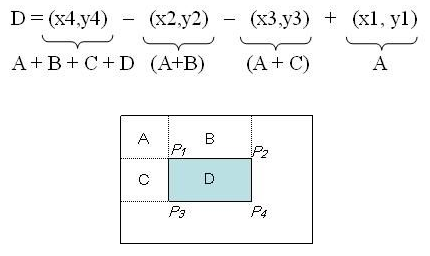
Các giá trị chỉ ra các đặc điểm nhất định của một khu vực cụ thể của hình ảnh.

Tính năng 2 hình chữ nhật có thể chỉ ra vị trí đường viền nằm giữa vùng tối và vùng sáng.

Giả sử ta cần tính tổng giá trị mức xám của vùng D như hình dưới, ta có thể tính được như sau:

D = A + B + C + D – (A+B) – (A+C) + A

Với A + B + C + D chính là giá trị tại điểm P4 trên Integral Image, tương tự như vậy A+B là giá trị tại điểm P2, A+C là giá trị tại điểm P3, và A là giá trị tại điểm P1. Vậy ta có thể viết lại biểu thức tính D ở trên như sau:



Trong đó:

