

写在前面的话

OpenCV 是计算机视觉中经典的专用库，其支持多语言、跨平台，功能强大。**OpenCV-Python** 为OpenCV提供了Python接口，使得使用者在Python中能够调用C/C++，在保证易读性和运行效率的前提下，实现所需的功能。

OpenCV-Python Tutorials 是官方提供的文档，其内容全面、简单易懂，使得初学者能够快速上手使用。2014年段力辉在当时已翻译过OpenCV3.0，但时隔五年，如今的 **OpenCV4.1** 中许多函数和内容已经有所更新，因此有必要对该官方文档再进行一次翻译。

翻译过程中难免有所疏漏，如发现错误，希望大家指出，谢谢支持。

OpenCV-Python Tutorials官方文档：https://docs.opencv.org/4.1.2/d6/d00/tutorial_py_root.html

目录

OpenCV中文官方文档

- OpenCV简介
 - [0_OpenCV-Python Tutorials](#)
- OpenCV安装
 - [1_1_OpenCV-Python教程简介](#)
 - [1_2_在Windows中安装OpenCV-Python](#)
 - [1_3_在Fedora中安装OpenCV-Python](#)
 - [1_4_在Ubuntu中安装OpenCV-Python](#)
- OpenCV中的GUI特性
 - [2_1_图像入门](#)
 - [2_2_视频入门](#)
 - [2_3_OpenCV中的绘图功能](#)
 - [2_4_鼠标作为画笔](#)

- [2_5_轨迹栏作为调色板](#)
- 核心操作
 - [3_1_图像的基本操作](#)
 - [3_2_图像上的算法运算](#)
 - [3_3_性能衡量和提升技术](#)
- OpenCV中的图像处理
 - [4_1_改变颜色空间](#)
 - [4_2_图像几何变换](#)
 - [4_3_图像阈值](#)
 - [4_4_图像平滑](#)
 - [4_5_形态转换](#)
 - [4_6_图像梯度](#)
 - [4_7_Canny边缘检测](#)
 - [4_8_图像金字塔](#)
 - [4_9_1_OpenCV中的轮廓](#)
 - [4_9_2_轮廓特征](#)
 - [4_9_3_轮廓属性](#)
 - [4_9_4_轮廓：更多属性](#)
 - [4_9_5_轮廓分层](#)
 - [4_10_1_直方图-1：查找，绘制，分析](#)
 - [4_10_2_直方图-2：直方图均衡](#)
 - [4_10_3_直方图3：二维直方图](#)
 - [4_10_4_直方图-4：直方图反投影](#)
 - [4_11_傅里叶变换](#)
 - [4_12_模板匹配](#)
 - [4_13_霍夫线变换](#)
 - [4_14_霍夫圈变换](#)

- [4_15_图像分割与分水岭算法](#)
- [4_16_交互式前景提取使用GrabCut算法](#)
- 特征检测与描述
 - [5_1_理解特征](#)
 - [5_2_哈里斯角检测](#)
 - [5_3_Shi-Tomasi拐角探测器和良好的跟踪功能](#)
 - [5_4_SIFT \(尺度不变特征变换 \) 简介](#)
 - [5_5_SURF简介 \(加速的强大功能 \)](#)
 - [5_6_用于角点检测的FAST算法](#)
 - [5_7_BRIEF \(二进制的鲁棒独立基本特征 \)](#)
 - [5_8_ORB \(定向快速和旋转简要 \)](#)
 - [5_9_特征匹配](#)
 - [5_10_特征匹配+单应性查找对象](#)
- 视频分析
 - [6_1_如何使用背景分离方法](#)
 - [6_2_Meanshift和Camshift](#)
 - [6_3_光流](#)
- 相机校准和3D重建
 - [7_1_相机校准](#)
 - [7_2_姿态估计](#)
 - [7_3_对极几何](#)
 - [7_4_立体图像的深度图](#)
- 机器学习
 - [8_1_理解KNN](#)
 - [8_2_使用OCR手写数据集运行KNN](#)
 - [8_3_理解SVM](#)
 - [8_4_使用OCR手写数据集运行SVM](#)

- [8_5_理解K均值聚类](#)
- [8_6_OpenCV中的K均值](#)
- 计算摄影学
 - [9_1_图像去噪](#)
 - [9_2_图像修补](#)
 - [9_3_高动态范围](#)
- 目标检测
 - [10_1_级联分类器](#)
 - [10_2_级联分类器训练](#)
- OpenCV-Python Binding
 - [11_1_OpenCV-Python Bindings](#)

关于

扫描下方二维码，关注公众号【深度学习与计算机视觉】，获取更多计算机视觉教程，资源。

扫描下方二维码，关注公众号【深度学习与计算机视觉】，发送关键字“**pytorchpdf**”到公众号后台，可获得最新pytorch中文官方文档 PDF 资源。



深度学习与计算机视觉

微信扫描二维码，关注我的公众号

扫描下方二维码或搜索微信ID “mthler”，加我微信，备注“视觉”，拉你进交流群



贝加尔 
阿尔巴尼亚

加我微信，备注“视觉”，拉你进交流群



扫一扫上面的二维码图案，加我微信

OpenCV 中文官方文档

<http://woshicver.com/>

Github 地址

<https://github.com/fendouai/OpenCVTutorials>

0_OpenCV-Python Tutorials

OpenCV-Python教材

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

写在前面的话

OpenCV是计算机视觉中经典的专用库，其支持多语言、跨平台，功能强大。OpenCV-Python为OpenCV提供了Python接口，使得使用者在Python中能够调用C/C++，在保证易读性和运行效率的前提下，实现所需的功能。OpenCV-Python Tutorials是官方提供的文档，其内容全面、简单易懂，使得初学者能够快速上手使用。

2014年段力辉在当时已翻译过OpenCV3.0，但时隔五年，如今的OpenCV4.1中许多函数和内容已经有所更新，因此有必要对该官方文档再进行一次翻译。

翻译过程中难免有所疏漏，如发现错误，希望大家指出，谢谢支持。

OpenCV-Python Tutorials 官方文档：https://docs.opencv.org/master/d6/d00/tutorial_py_root.html

目录

- **OpenCV简介:**

了解如何在计算机上安装OpenCV-Python

- **OpenCV中的GUI特性**

在这里，您将学习如何显示和保存图像和视频，控制鼠标事件以及创建轨迹栏。

- **核心操作**

在本节中，您将学习图像的基本操作、例如像素编辑、几何变换，代码优化、一些数学工具等。

- **OpenCV中的图像处理**

在本节中，您将学习OpenCV内部的不同图像处理函数。

- **特征检测与描述**

在本节中，您将学习有关特征检测和描述符的信息

- **视频分析**

在本部分中，您将学习与对象跟踪等视频配合使用的不同技术。

- **相机校准和3D重建**

在本节中，我们将学习有关相机校准，立体成像等的信息。

- **机器学习**

在本节中，您将学习OpenCV内部的不同图像处理函数。

- **计算摄影学**

在本节中，您将学习不同的计算摄影技术如图像去噪等。

- **目标检测 (objdetect模块)**

在本节中，您将学习目标检测技术，例如人脸检测等。

- **OpenCV-Python Binding**

在本节中，我们将了解如何生成OpenCV-Python Binding

OpenCV-Python教程简介

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

OpenCV

OpenCV由**Gary Bradsky**于1999年在英特尔创立，第一版于2000年问世。**Vadim Pisarevsky**加入Gary Bradsky，一起管理英特尔的俄罗斯软件OpenCV团队。2005年，OpenCV用于Stanley，该车赢得了2005年DARPA挑战赛的冠军。后来，在Willow Garage的支持下，它的积极发展得以继续，由Gary Bradsky和Vadim Pisarevsky领导了该项目。OpenCV现在支持与计算机视觉和机器学习有关的多种算法，并且正在日益扩展。

OpenCV支持多种编程语言，例如C++、Python、Java等，并且可在Windows、Linux、OS X、Android和iOS等不同平台上使用。基于CUDA和OpenCL的高速GPU操作的接口也正在积极开发中。

OpenCV-Python是用于OpenCV的Python API，结合了OpenCV C++ API和Python语言的最佳特性。

OpenCV-Python

OpenCV-Python是旨在解决计算机视觉问题的Python专用库。

Python是由**Guido van Rossum**发起的通用编程语言，很快就非常流行，主要是因为它的简单性和代码可读性。它使程序员可以用较少的代码行表达想法，而不会降低可读性。

与C/C++之类的语言相比，Python速度较慢。也就是说，可以使用C/C++轻松扩展Python，这使我们能够用C/C++编写计算密集型代码并创建可用作Python模块的Python包装器。这给我们带来了两个好处：首先，代码与原始C/C++代码一样快（因为它是在后台运行的实际C++代码），其次，在Python中比C/C++编写代码更容易。OpenCV-Python是原始OpenCV C++实现的Python包装器。

OpenCV-Python利用了**Numpy**，这是一个高度优化的库，用于使用MATLAB样式的语法进行数值运算。所有OpenCV数组结构都与Numpy数组相互转换。这也使与使用Numpy的其他库（例如SciPy和Matplotlib）的集成变得更加容易。

OpenCV-Python教程

OpenCV引入了一组新的教程，它们将指导您完成OpenCV-Python中可用的各种功能。**本指南主要针对OpenCV 3.x版本**（尽管大多数教程也适用于OpenCV 2.x）。

建议先了解Python和Numpy，因为本指南将不介绍它们。**要使用OpenCV-Python编写优化的代码，必须先明白Numpy。**

本教程最初由*Abid Rahman K.在*Alexander Mordvintsev*的指导下作为Google Summer of Code 2013计划的一部分*启动。

OpenCV需要您！

由于OpenCV是开放源代码计划，因此欢迎所有人为这个库，文档和教程做出贡献。如果您在本教程中发现任何错误（从小的拼写错误到代码或概念中的严重错误），请随时通过在GitHub中:<https://github.com/opencv/opencv> 克隆OpenCV 并提交请求请求来更正它。OpenCV开发人员将检查您的请求请求，给您重要的反馈，并且（一旦通过审阅者的批准）它将被合并到OpenCV中。然后，您将成为开源贡献者:-)

随着新模块添加到OpenCV-Python中，本教程将不得不进行扩展。如果您熟悉特定的算法，并且可以编写一个包括算法基本理论和显示示例用法的代码的教程，欢迎你这样做。

记住，我们可以**共同**使这个项目取得巨大成功！

贡献者

以下是向OpenCV-Python提交了教程的贡献者列表。

1. Alexander Mordvintsev (GSoC-2013 导师)
2. Abid Rahman K. (GSoC-2013 实习生)

其他资源

1. Python快速指南- [一小部分Python] : <http://swaroopch.com/notes/python/>
2. 基本的Numpy教程 : http://wiki.scipy.org/Tentative_NumPy_Tutorial
3. numpy示例列表 : http://wiki.scipy.org/Numpy_Example_List
4. OpenCV文档 : <http://docs.opencv.org/>
5. OpenCV论坛 : <http://answers.opencv.org/questions/>

在Windows中安装OpenCV-Python

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本教程中

- 我们将学习在你的Windows系统中设置OpenCV-Python。

下面的步骤在装有Visual Studio 2010和Visual Studio 2012的Windows 7-64位计算机上进行了测试。屏幕截图展示的是VS2012。

从预编译的二进制文件安装OpenCV

1. 下面的Python软件包将被下载并安装到其默认位置。
2. Python的3.X(3.4+)或Python 2.7.x从这里下载：<https://www.python.org/downloads/>。
3. Numpy包(例如使用 `pip install numpy` 命令下载)。
4. Matplotlib(`pip install matplotlib`)(Matplotlib是可选的，但推荐它，因为我们使用了很多在我们的教程)。
5. 将所有软件包安装到其默认位置。`C:/Python27/` 如果使用Python 2.7，将安装Python。
6. 安装后，打开Python IDLE。输入`**import numpy**`并确保Numpy运行正常。
7. 从GitHub：<https://github.com/opencv/opencv/releases> 或SourceForge网站：<https://sourceforge.net/projects/opencvlibrary/files/> 下载最新的OpenCV版本，然后双击将其解压缩。
8. 转到`**opencv/build/python/2.7**`文件夹。
9. 将`**cv2.pyd**`复制到`**C:/Python27/lib/site-packages**`。
10. 打开Python IDLE，然后在Python终端中键入以下代码。

```
>>> import cv2 as cv
>>> print( cv.__version__ )
```

如果打印出来的结果没有任何错误，那就恭喜！你已经成功安装了OpenCV-Python。

从源代码构建OpenCV

1. 下载并安装Visual Studio和CMake。
2. Visual Studio 2012:<http://go.microsoft.com/?linkid=9816768>
3. CMake:<https://cmake.org/download/>
4. 将必要的Python软件包下载并安装到其默认位置
5. Python
6. Numpy

注意 在这种情况下，我们使用的是32位Python软件包二进制文件。但是，如果要将OpenCV用于x64，则将安装Python软件包的64位二进制文件。问题在于，没有Numpy的官方64位二进制文件。你必须自行构建。为此，你必须使用与构建Python相同的编译器。启动Python IDLE时，它会显示编译器详细信息。你可以在此处：<http://stackoverflow.com/q/2676763/1134940> 获得更多信息。因此，你的系统必须具有相同的Visual Studio版本并从源代码构建Numpy。

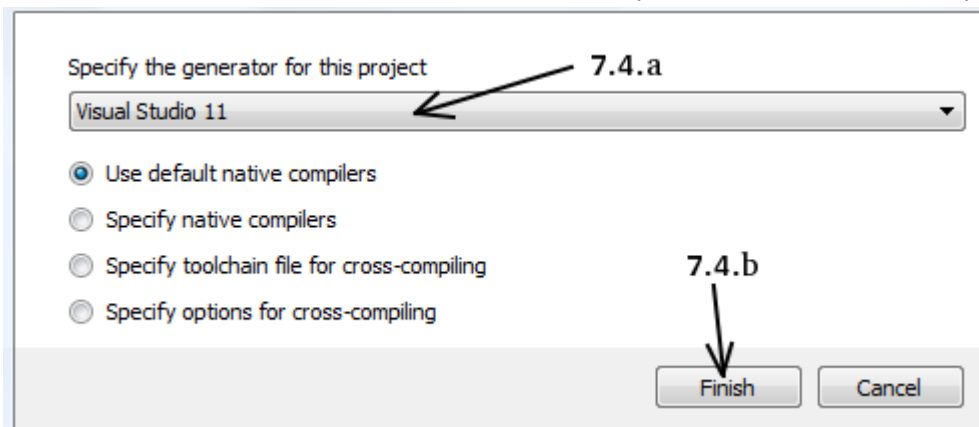
拥有64位Python软件包的另一种方法是使用来自第三方(如Anaconda：<http://www.continuum.io/downloads>、Enthought：<https://www.enthought.com/downloads/>)等现成Python发行版。它的大小会更大，但可以满足你的所有需求。一切都在一个外壳中。你也可以下载32位版本。

1. 确保Python和Numpy正常运行。
2. 下载OpenCV源代码。它可以来自Sourceforge:<http://sourceforge.net/projects/opencvlibrary/> (官方发行版)或来自Github:<https://github.com/opencv/opencv> (最新源)。
3. 将其解压缩到一个文件夹中，在opencv中创建一个新的文件夹。
4. 打开CMake-gui(Start>All Programs> CMake-gui)
5. 如下填写字段(请参见下图)：
 - a. 单击**Browse Source**然后找到opencv文件夹。
 - b. 单击**Browse Build**然后找到我们创建的构建文件夹。

c. 点击**Configure**。

![] (<http://qiniu.aihubs.net/Capture1.jpg>)

d. 它将打开一个新窗口以选择编译器。选择适当的编译器(此处为Visual Studio 11)，然后单击



Finish。

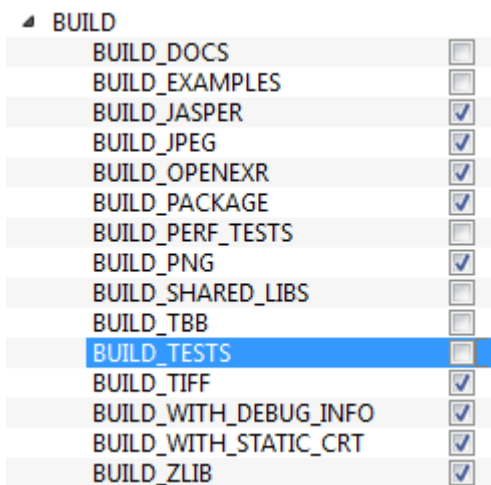
e. 等待分析完

成。

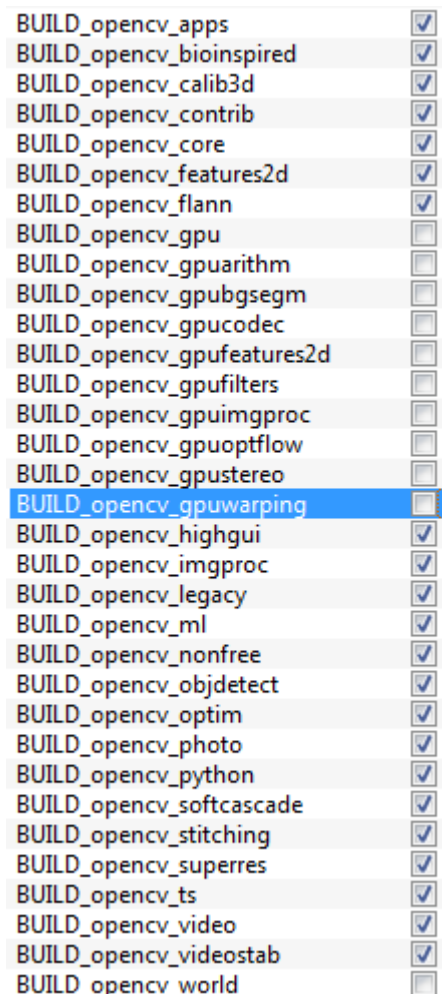
1. 你将看到所有字段都标记为红色。单击**WITH**字段将其展开。它决定了你需要哪些额外的功能。因此，请标记适当的字段。见下图：

WITH	
WITH_1394	☐
WITH_CLP	☐
WITH_CSTRIPES	☐
WITH_CUBLAS	☐
WITH_CUDA	☐
WITH_CUFFT	☐
WITH_DSHOW	☒
WITH_FFMPEG	☒
WITH_GIGEAPI	☐
WITH_GSTREAMER_1_X	☐
WITH_IPP	☐
WITH_JASPER	☒
WITH_JPEG	☒
WITH_MSMF	☒
WITH_NVCUVID	☐
WITH_OPENCL	☐
WITH_OPENCLAMDBLAS	☐
WITH_OPENCLAMDFFT	☐
WITH_OPENEXR	☒
WITH_OPENGL	☐
WITH_OPENNI	☐
WITH_PNG	☒
WITH_PVAPI	☐
WITH_QT	☐
WITH_TIFF	☒
WITH_VFW	☒
WITH_WEBP	☒
WITH_WIN32UI	☒
WITH_XIMEA	☐

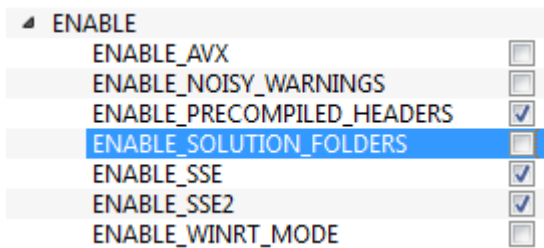
2. 现在，单击**BUILD**字段以将其展开。前几个字段配置构建方法。见下图：



3. 其余字段指定要构建的模块。由于OpenCV-Python尚不支持GPU模块，因此可以完全避免使用它以节省时间(但是如果使用它们，则将其保留在此处)。见下图：



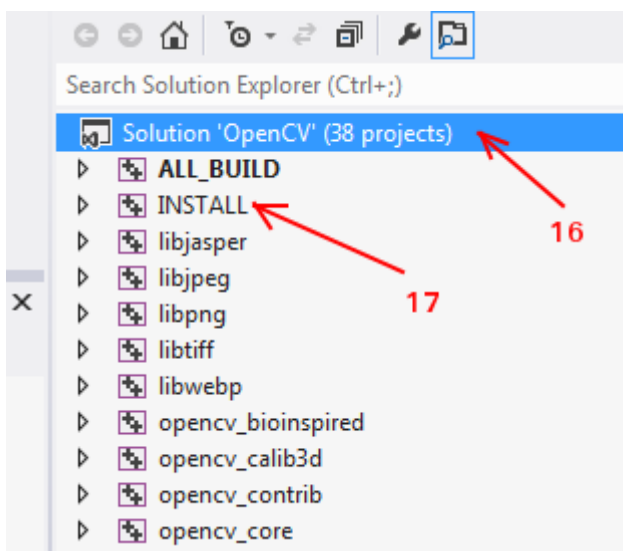
4. 现在单击 **ENABLE**** 字段将其展开。确保未选中**ENABLE_SOLUTION_FOLDERS(Visual Studio Express版本不支持解决方案文件夹)。见下图：



5. 还要确保在**PYTHON**字段中，所有内容都已填充。(忽略PYTHON_DEBUG_LIBRARY)。见下图：

PYTHON	
PYTHON_DEBUG_LIBRARY	PYTHON_DEBUG_LIBRARY-NOTFOUND
PYTHON_EXECUTABLE	C:/Python27/python.exe
PYTHON_INCLUDE_DIR	C:/Python27/include
PYTHON_LIBRARY	C:/Python27/libs/python27.lib
PYTHON_NUMPY_INCLUDE_DIRS	C:/Python27/lib/site-packages/numpy/core/include
PYTHON_PACKAGES_PATH	C:/Python27/Lib/site-packages

6. 最后，单击**Generate**按钮。
7. 现在转到我们的**opencv / build**文件夹。在那里你将找到**OpenCV.sln**文件。用Visual Studio打开它。
8. 将构建模式检查为**Release**而不是**Debug**。
9. 在解决方案资源管理器中，右键单击**Solution***(或**ALL_BUILD***)并进行构建。需要一些时间才能完成。
10. 再次，右键单击**INSTALL**并进行构建。现在将安装OpenCV-Python。



11. 打开Python IDLE，然后输入 `import cv2 as cv`。如果没有错误，则说明已正确安装。

注意 我们没有安装其他支持如TBB、Eigen、Qt、Documentation等。在这里很难解释清楚。我们将添加更详细的视频，或者你可以随意修改。

其他资源

练习题

如果你有Windows计算机，请从源代码编译OpenCV。做各种各样极客。如果遇到任何问题，请访问OpenCV论坛并解释你的问题。

在Fedora中安装OpenCV-Python

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本教程中

- 我们将学习在你的Fedora系统中设置OpenCV-Python。针对Fedora 18 (64位) 和Fedora 19 (32位) 进行以下步骤。

介绍

可以通过两种方式在Fedora中安装OpenCV-Python：1) 从fedora存储库中可用的预构建二进制文件安装，2) 从源代码进行编译。在本节中，我们将同时看到这两种方法。

另一个重要的事情是所需的其他库。OpenCV-Python仅需要**Numpy** (除了其他依赖关系，我们将在后面看到)。但是在本教程中，我们还使用**Matplotlib**进行一些简单而又漂亮的作图 (与OpenCV相比，感觉好多了)。Matplotlib是可选的，但强烈建议安装。同样，我们还将看到**IPython**，这是一个强烈推荐的交互式Python终端。

从预构建的二进制文件安装OpenCV-Python

以root用户身份在终端中使用以下命令安装所有软件包。

```
$ yum install numpy opencv *
```

打开Python IDLE (或IPython)，然后在Python终端中键入以下代码。

```
>>> import cv2 as cv
>>> print( cv.__version__ )
```

如果打印出来的结果没有任何错误，那就恭喜！你已经成功安装了OpenCV-Python。

这很简单。但是这里有一个问题。Yum仓库可能不总是包含最新版本的 OpenCV。例如，在撰写本教程时，yum 库包含2.4.5，而最新的 OpenCV 版本是2.4.6。对于 Python API，最新版本总是包含更好的支持。另外，取决于所使用的驱动程序、ffmpeg、gstreamer软件包等，相机支持，视频播放等可能会出现问题。

所以我个人的偏好是下一种方法，即从源代码编译。在某个时候，如果你想为OpenCV 做贡献，你也需要这个。

从源代码安装OpenCV

从源代码编译起初可能看起来有点复杂，但是一旦你成功了，就没有什么复杂的了。

首先，我们将安装一些依赖项。有些是强制性的，有些是可选的。可选的依赖项，如果不需要，可以跳过。

强制依赖

我们需要**CMake**来配置安装，**GCC**进行编译，**Python-devel**和**Numpy**来创建Python扩展等。

```
yum install cmake
yum install python-devel numpy
yum install gcc gcc-c++
```

接下来，我们需要**GTK**对GUI功能的支持，相机支持(libdc1394，v4l)，媒体支持(ffmpeg，gstreamer)等。

```
yum install gtk2-devel
yum install libdc1394-devel
yum install ffmpeg-devel
yum install gstreamer-plugins-base-devel
```

可选依赖项

以上依赖关系足以在你的fedora计算机中安装OpenCV。但是根据你的要求，你可能需要一些额外的依赖项。此类可选依赖项的列表如下。你可以跳过或安装它，取决于你:)

OpenCV附带了用于图像格式（例如PNG，JPEG，JPEG2000，TIFF，WebP等）的支持文件。但是它可能有些旧。如果要获取最新的库，可以安装这些格式的开发文件。

```
yum install libpng-devel
yum install libjpeg-turbo-devel
yum install jasper-devel
yum install openexr-devel
yum install libtiff-devel
yum install libwebp-devel
```

几个OpenCV功能与**英特尔的线程构建模块** (TBB) 并行。但是，如果要启用它，则需要先安装TBB。(同样在使用CMake配置安装时，请不要忘记设置 `-D WITH_TBB = ON` 。下面更多详细信息。)

```
yum install tbb-devel
```

OpenCV使用另一个**Eigen**库来优化数学运算。因此，如果你的系统中装有Eigen，则可以利用它。(同样在使用CMake配置安装时，请不要忘记设置 `WITH_EIGEN = ON` 。下面更多详细信息。)

```
yum install eigen3-devel
```

如果你要构建**文档** (是的，你可以使用完整的搜索功能以HTML格式在系统中创建OpenCV完整官方文档的脱机版本，这样，如果有任何问题，你就不必总是访问Internet，而且非常快捷!!!)，你需要安装**Doxygen** (文档生成工具)。

```
yum install doxygen
```

下载OpenCV

接下来，我们必须下载OpenCV。你可以从sourceforge网站：<http://sourceforge.net/projects/opencvlibrary/> 下载最新版本的OpenCV。然后解压缩文件夹。

或者，你可以从OpenCV的github存储库下载最新的源代码。(如果你想为OpenCV做出贡献，请选择此项。它始终使你的OpenCV保持最新状态)。为此，你需要先安装**Git**。

```
yum install git
git clone https://github.com/opencv/opencv.git
```

它将在主目录 (或你指定的目录) 中创建一个文件夹OpenCV。克隆可能需要一些时间，具体取决于你的Internet网络。

现在打开一个终端窗口，然后导航到下载的OpenCV文件夹。创建一个新的构建文件夹并导航到它。

```
mkdir build
cd build
```

配置和安装

现在，我们已经安装了所有必需的依赖项，让我们安装OpenCV。必须使用CMake配置安装。它指定要安装的模块，安装路径，要使用的其他库，是否要编译的文档和示例等。下面的命令通常用于配置（从build文件夹执行）。

```
cmake -D CMAKE_BUILD_TYPE = RELEASE -D CMAKE_INSTALL_PREFIX = /usr/local ..
```

它指定构建类型为“发布模式”，安装路径为/usr/local。在每个选项之前标志 `-D`，在最后观察标志 `..`。简而言之，这是一种格式：

```
cmake [-D <flag>] [-D <flag>] ..
```

你可以指定任意数量的标志，但是每个标志前面应带有-D。

因此，在本教程中，我们将安装具有TBB和Eigen支持的OpenCV。我们还构建了文档，但是不包括性能测试和构建示例。我们还会禁用与GPU相关的模块（因为我们使用的是OpenCV-Python，因此我们不需要与GPU相关的模块。这为我们节省了一些时间）。

（以下所有命令都可以在单个cmake语句中完成，但为了便于理解，此处将其拆分。）

- 启用TBB和Eigen支持：

```
cmake -D WITH_TBB=ON -D WITH_EIGEN=ON ..
```

- 启用文档并禁用测试和示例

```
cmake -D BUILD_DOCS=ON -D BUILD_TESTS=OFF -D BUILD_PERF_TESTS=OFF -D
BUILD_EXAMPLES=OFF ..
```

- 禁用所有与GPU相关的模块。

```
cmake -D WITH_OPENCL=OFF -D BUILD_opencv_gpu=OFF -D BUILD_opencv_gpuarithm=OFF -D
BUILD_opencv_gpubgsegm=OFF -D BUILD_opencv_gpucodec=OFF -D
```

```
BUILD_opencv_gpufeatures2d=OFF -D BUILD_opencv_gpufilters=OFF -D  
BUILD_opencv_gpuimgproc=OFF -D BUILD_opencv_gpulegacy=OFF -D  
BUILD_opencv_gpuoptflow=OFF -D BUILD_opencv_gpustereo=OFF -D  
BUILD_opencv_gpuwarping=OFF ..
```

- 设置安装路径和构建类型

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
```

每次输入cmake语句时，它都会打印出结果配置设置。在完成的最终设置中，请确保填写以下字段（以下是我获得的一些重要配置）。这些字段也应在你的系统中适当填写。否则将会发生一些问题。因此，请检查你是否正确执行了上述步骤。

```
...  
-- GUI:  
--   GTK+ 2.x:                YES (ver 2.24.19)  
--   GThread :                YES (ver 2.36.3)  
-- Video I/O:  
--   DC1394 2.x:              YES (ver 2.2.0)  
--   FFMPEG:                  YES  
--     codec:                  YES (ver 54.92.100)  
--     format:                 YES (ver 54.63.104)  
--     util:                   YES (ver 52.18.100)  
--     swscale:                YES (ver 2.2.100)  
--     gentoo-style:           YES  
-- GStreamer:  
--   base:                    YES (ver 0.10.36)  
--   video:                   YES (ver 0.10.36)  
--   app:                     YES (ver 0.10.36)  
--   riff:                    YES (ver 0.10.36)  
--   pbutils:                 YES (ver 0.10.36)  
-- V4L/V4L2:                  Using libv4l (ver 1.0.0)  
-- Other third-party libraries:  
--   Use Eigen:                YES (ver 3.1.4)  
--   Use TBB:                  YES (ver 4.0 interface 6004)  
-- Python:  
--   Interpreter:              /usr/bin/python2 (ver 2.7.5)  
--   Libraries:                /lib/libpython2.7.so (ver 2.7.5)  
--   numpy:                    /usr/lib/python2.7/site-packages/numpy/core/  
include (ver 1.7.1)  
--   packages path:            lib/python2.7/site-packages  
...
```

还有许多其他标志和设置。它留给你以作进一步的探索。

现在，你可以使用 `make` 命令构建文件，并使用 `make install` 命令进行安装。`make install` 应该以 `root` 身份执行。

```
make
su
make install
```

安装结束。所有文件都安装在 `/usr/local/` 文件夹中。但是要使用它，你的Python应该能够找到OpenCV模块。你有两个选择。

1. **将模块移动到Python路径中的任何文件夹**：可以通过在Python终端中输入

`import sys; print(sys.path)` 来找到Python路径。它将打印出许多位置。将 `/usr/local/lib/python2.7/site-packages/cv2.so` 移至该文件夹中的任何一个。例如，`su mv /usr/local/lib/python2.7/site-packages/cv2.so /usr/lib/python2.7/site-packages` 但是，每次安装OpenCV时都必须这样做。

2. **将 `/usr/local/lib/python2.7/site-packages` 添加到 `PYTHON_PATH`**：只需执行一次。只需打

开 `.bashrc` 并向其添加以下行，然后注销并返回即可。`export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.7/site-packages` 至此，OpenCV安装完成。打开终端，然后尝试 `import cv2 as cv`。

要构建文档，只需输入以下命令：

```
make doxygen
```

然后打开 `opencv/build/doc/doxygen/html/index.html` 并将其添加到浏览器中。

其他资源

练习题

1. 在Fedora系统的机器中采用源代码编译OpenCV。

在Ubuntu中安装OpenCV-Python

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

在本教程中，我们将学习在Ubuntu System中设置OpenCV-Python。以下步骤针对Ubuntu 16.04和18.04（均为64位）进行了测试。

可以通过两种方式在Ubuntu中安装OpenCV-Python：

- 从Ubuntu存储库中可用的预构建二进制文件安装
- 从源代码编译。在本节中，我们将同时看到两者。

另一个重要的事情是所需的其他库。OpenCV-Python仅需要**Numpy**（除了其他依赖关系，我们将在后面看到）。但是在本教程中，我们还使用**Matplotlib**进行一些简单而又漂亮的绘图目的（与OpenCV相比，我感觉好多了）。Matplotlib是可选的，但强烈建议使用。同样，我们还将看到**IPython**，这是一个强烈推荐的交互式Python终端。

从预构建的二进制文件安装OpenCV-Python

仅用于编程和开发OpenCV应用程序时，此方法最有效。

在终端（以root用户身份）中使用以下命令安装python-opencv:<https://packages.ubuntu.com/trusty/python-opencv>软件包。

```
$ sudo apt-get install python-opencv
```

打开Python IDLE（或IPython），然后在Python终端中键入以下代码。

```
import cv2 as cv
print(cv.__version__)
```

如果打印出来的结果没有任何错误，那就恭喜！你已经成功安装了OpenCV-Python。

这看起来很容易，但也可能出现一些问题。Apt存储库不一定总是包含最新版本的OpenCV。例如，在编写本教程时，apt存储库包含2.4.8，而最新的OpenCV版本是3.x。关于Python API，最新版本将始终包含更好的支持和最新的错误修复。

因此，要获取最新的源代码，首选方法是从源代码进行编译。同样在某个时间点，如果你想为OpenCV做出贡献，则将通过这种方式。

从源代码构建OpenCV

首先，从源代码进行编译似乎有些复杂，但是一旦成功完成，就没有什么复杂的了。

首先，我们将安装一些依赖项。有些是必需的，有些是可选的。如果不想，可以跳过可选的依赖项。

所需的构建依赖项

我们需要**CMake**来配置安装，需要**GCC**进行编译，需要**Python-devel**和**Numpy**来构建Python依赖项等。

```
sudo apt-get install cmake  
sudo apt-get install gcc g++
```

支持python2: `sudo apt-get install python-dev python-numpy`

支持python3: `sudo apt-get install python3-dev python3-numpy`

接下来，我们需要GUI功能的**GTK**支持，相机支持（v4l），媒体支持（ffmpeg，gstreamer）等。

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev  
sudo apt-get install libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
```

支持gtk2: `sudo apt-get install libgtk2.0-dev`

支持gtk3: `sudo apt-get install libgtk-3-dev`

可选依赖项

以上依赖关系足以在你的Ubuntu计算机中安装OpenCV。但是根据你的需求，你可能需要一些额外的依赖项。此类可选依赖项的列表如下。你可以跳过或安装它，取决于你：)

OpenCV附带了用于图像格式（例如PNG，JPEG，JPEG2000，TIFF，WebP等）的支持文件。但是它可能有些旧。如果要获取最新的库，可以为这些格式的系统库安装开发文件。

```
sudo apt-get install libpng-dev
sudo apt-get install libjpeg-dev
sudo apt-get install libopenexr-dev
sudo apt-get install libtiff-dev
sudo apt-get install libwebp-dev
```

注意 如果你使用的是Ubuntu 16.04，则还可以安装 `libjasper-dev` 以添加对JPEG2000格式的系统级别支持。

下载OpenCV

要从OpenCV的GitHub Repository:<https://github.com/opencv/opencv>下载最新的源代码。（如果你想为OpenCV做出贡献，请选择此项。为此，你需要先安装**Git**）

```
$ sudo apt-get install git
$ git clone https://github.com/opencv/opencv.git
```

它将在当前目录中创建一个文件夹”opencv”。下载可能需要一些时间，具体取决于你的Internet网络。

现在打开一个终端窗口，并导航到下载的”opencv”文件夹。创建一个新的”build”文件夹并导航到它。

```
$ mkdir build
$ cd build
```

配置和安装

现在我们有了所有必需的依赖项，让我们安装OpenCV。必须使用CMake配置安装。它指定要安装的模块，安装路径，要使用的其他库，是否要编译的文档和示例等。大多数工作都是使用配置良好的默认参数自动完成的。

以下命令通常用于配置OpenCV库构建（从构建文件夹执行）：`$ cmake ../`

OpenCV的默认默认设置为”Release”构建类型，安装路径为 `/usr/local`。有关CMake选项的更多信息，请参考OpenCV C++编译指南:https://docs.opencv.org/4.1.2/d7/d9f/tutorial_linux_install.html

你应该在CMake输出中看到以下几行（它们意味着正确找到了Python）：

```
-- Python 2:
--   Interpreter:           /usr/bin/python2.7 (ver 2.7.6)
--   Libraries:            /usr/lib/x86_64-linux-gnu/libpython2.7.so
(ver 2.7.6)
--   numpy:                /usr/lib/python2.7/dist-packages/numpy/core/
include (ver 1.8.2)
--   packages path:        lib/python2.7/dist-packages
--
-- Python 3:
--   Interpreter:           /usr/bin/python3.4 (ver 3.4.3)
--   Libraries:            /usr/lib/x86_64-linux-gnu/libpython3.4m.so
(ver 3.4.3)
--   numpy:                /usr/lib/python3/dist-packages/numpy/core/
include (ver 1.8.2)
--   packages path:        lib/python3.4/dist-packages
```

现在，使用 `make` 命令构建文件，然后使用 `make install` 命令安装文件。

```
$ make
# sudo make install
```

安装结束。所有文件都安装在 `/usr/local/` 文件夹中。打开终端，然后尝试导入 `cv2`。

```
import cv2 as cv
print(cv.__version__)
```

图像入门

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 在这里，你将学习如何读取图像，如何显示图像以及如何将其保存回去
- 你将学习以下功能：**cv.imread()**，**cv.imshow()**，**cv.imwrite()**
- (可选) 你将学习如何使用Matplotlib显示图像

使用OpenCV

读取图像

使用**cv.imread()**函数读取图像。图像应该在工作目录或图像的完整路径应给出。

第二个参数是一个标志，它指定了读取图像的方式。

- **cv.IMREAD_COLOR**：加载彩色图像。任何图像的透明度都会被忽视。它是默认标志。
- **cv.IMREAD_GRAYSCALE**：以灰度模式加载图像
- **cv.IMREAD_UNCHANGED**：加载图像，包括alpha通道

注意 除了这三个标志，你可以分别简单地传递整数1、0或-1。

请参见下面的代码：

```
import numpy as np
import cv2 as cv

# 加载彩色灰度图像
img = cv.imread('messi5.jpg', 0)
```

警告

即使图像路径错误，它也不会引发任何错误，但是 `print img` 会给出 `None`

显示图像

使用函数 `cv.imshow()` 在窗口中显示图像。窗口自动适合图像尺寸。

第一个参数是窗口名称，它是一个字符串。第二个参数是我们的对象。你可以根据需要创建任意多个窗口，但可以使用不同的窗口名称。

```
cv.imshow('image', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

窗口的屏幕截图如下所示（在Fedora-Gnome机器中）：



`cv.waitKey()` 是一个键盘绑定函数。其参数是以毫秒为单位的时间。该函数等待任何键盘事件指定的毫秒。如果您在这段时间内按下任何键，程序将继续运行。如果 `0` 被传递，它将无限期地等待一次敲击键。它也可以设置为检测特定的按键，例如，如果按下键 `a` 等，我们将在下面讨论。

注意 除了键盘绑定事件外，此功能还处理许多其他GUI事件，因此你必须使用它来实际显示图像。

`cv.destroyAllWindows()` 只会破坏我们创建的所有窗口。如果要销毁任何特定的窗口，请使用函数 `cv.destroyWindow()` 在其中传递确切的窗口名称作为参数。

注意 在特殊情况下，你可以创建一个空窗口，然后再将图像加载到该窗口。在这种情况下，你可以指定窗口是否可调整大小。这是通过功能`cv.namedWindow()`完成的。默认情况下，该标志为`cv.WINDOW_AUTOSIZE`。但是，如果将标志指定为`cv.WINDOW_NORMAL`，则可以调整窗口大小。当图像尺寸过大以及向窗口添加跟踪栏时，这将很有帮助。

请参见下面的代码：

```
cv.namedWindow('image', cv.WINDOW_NORMAL)
cv.imshow('image', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

写入图像

使用函数`cv.imwrite()`保存图像。

第一个参数是文件名，第二个参数是要保存的图像。 `cv.imwrite('messigray.png', img)`

这会将图像以PNG格式保存在工作目录中。

总结

在下面的程序中，以灰度加载图像，显示图像，按 `s` 保存图像并退出，或者按 `ESC` 键直接退出而不保存。

```
import numpy as np
import cv2 as cv
img = cv.imread('messi5.jpg', 0)
cv.imshow('image', img)
k = cv.waitKey(0)
if k == 27:          # 等待ESC退出
    cv.destroyAllWindows()
elif k == ord('s'): # 等待关键字，保存和退出
    cv.imwrite('messigray.png', img)
    cv.destroyAllWindows()
```

警告

如果使用的是64位计算机，则必须 `k = cv.waitKey(0)` 按如下所示修改行：`k = cv.waitKey(0) & 0xFF`

使用Matplotlib

Matplotlib是Python的绘图库，可为你提供多种绘图方法。你将在接下来的文章中看到它们。在这里，你将学习如何使用Matplotlib显示图像。你可以使用Matplotlib缩放图像，保存图像等。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg',0)
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
plt.xticks([], plt.yticks([])  # 隐藏 x 轴和 y 轴上的刻度值
plt.show()
```



窗口的屏幕截图如下所示：

还可以看看

Matplotlib中提供了许多绘图选项。请参考Matplotlib文档以获取更多详细信息。一些，我们将在路上看到。

警告

OpenCV加载的彩色图像处于BGR模式。但是Matplotlib以RGB模式显示。因此，如果使用OpenCV读取彩色图像，则Matplotlib中将无法正确显示彩色图像。有关更多详细信息，请参见练习。

其他资源

1. Matplotlib绘图样式和功能 : http://matplotlib.org/api/pyplot_api.html

练习题

1. 当你尝试在OpenCV中加载彩色图像并将其显示在Matplotlib中时，存在一些问题。阅读此讨论 : <http://stackoverflow.com/a/15074748/1134940>)并理解它。

图像入门

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 学习读取视频，显示视频和保存视频。
- 学习从相机捕捉并显示它。
- 你将学习以下功能：**cv.VideoCapture()**，**cv.VideoWriter()**

从相机中读取视频

通常情况下，我们必须用摄像机捕捉实时画面。提供了一个非常简单的界面。让我们从摄像头捕捉一段视频(我使用的是我笔记本电脑内置的网络摄像头)，将其转换成灰度视频并显示出来。只是一个简单的任务开始。

要捕获视频，你需要创建一个 **VideoCapture** 对象。它的参数可以是设备索引或视频文件的名称。设备索引就是指哪个摄像头的数字。正常情况下，一个摄像头会被连接(就像我的情况一样)。所以我简单地传0(或-1)。你可以通过传递1来选择第二个相机，以此类推。在此之后，你可以逐帧捕获。但是在最后，不要忘记释放俘虏。

```
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
    exit()
while True:
    # 逐帧捕获
    ret, frame = cap.read()
    # 如果正确读取帧, ret为True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
```



```
# 我们在框架上的操作到这里
gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
# 显示结果帧e
cv.imshow('frame', gray)
if cv.waitKey(1) == ord('q'):
    break
# 完成所有操作后, 释放捕获器
cap.release()
cv.destroyAllWindows()
```

`cap.read()` 返回布尔值(`True` / `False`)。如果正确读取了帧, 它将为 `True`。因此, 你可以通过检查此返回值来检查视频的结尾。

有时, `cap`可能尚未初始化捕获。在这种情况下, 此代码显示错误。你可以通过 `**cap.isOpened**()`方法检查它是否已初始化。如果是 `True`, 那么确定。否则, 使用 `**cap.open**()`打开它。

你还可以使用 `cap.get(propId)` 方法访问该视频的某些功能, 其中`propId`是0到18之间的一个数字。每个数字表示视频的属性(如果适用于该视频), 并且可以显示完整的详细信息在这里看到: `cv::VideoCapture::get()`。其中一些值可以使用 `cap.set(propId, value)` 进行修改。 `value`是你想要的新值。

例如, 我可以通过 `cap.get(cv.CAP_PROP_FRAME_WIDTH)` 和 `cap.get(cv.CAP_PROP_FRAME_HEIGHT)` 检查框架的宽度和高度。默认情况下, 它的分辨率为640x480。但我想将其修改为320x240。只需使用和即可。 `ret = cap.set(cv.CAP_PROP_FRAME_WIDTH, 320) and ret = cap.set(cv.CAP_PROP_FRAME_HEIGHT, 240)` .

注意 如果出现错误, 请确保使用任何其他相机应用程序(例如Linux中的Cheese)都可以正常使用相机。

从文件播放视频

它与从相机捕获相同, 只是用视频文件名更改摄像机索引。另外, 在显示框架时, 请使用适当的时间 `cv.waitKey()`。如果太小, 则视频将非常快, 而如果太大, 则视频将变得很慢(嗯, 这就是显示慢动作的方式)。正常情况下25毫秒就可以了。

```
import numpy as np
import cv2 as cv
cap = cv.VideoCapture('vtest.avi')
while cap.isOpened():
```

```
ret, frame = cap.read()
# 如果正确读取帧, ret为True
if not ret:
    print("Can't receive frame (stream end?). Exiting ...")
    break
gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
cv.imshow('frame', gray)
if cv.waitKey(1) == ord('q'):
    break
cap.release()
cv.destroyAllWindows()
```

注意 确保安装了正确的 ffmpeg 或 gstreamer 版本。有时, 使用视频捕获(Video Capture)是一件令人头疼的事情, 主要原因是错误地安装了 ffmpeg / gstreamer。

保存视频

所以我们捕捉一个视频, 一帧一帧地处理, 我们想要保存这个视频。对于图像, 它非常简单, 只需使用 `cv.imwrite()`。这里还需要做一些工作。

这次我们创建一个 **VideoWriter** 对象。我们应该指定输出文件名(例如: output.avi)。然后我们应该指定 **FourCC** 代码(详见下一段)。然后传递帧率的数量和帧大小。最后一个是颜色标志。如果为 `True`, 编码器期望颜色帧, 否则它与灰度帧一起工作。

FourCC : <http://en.wikipedia.org/wiki/FourCC> 是用于指定视频编解码器的4字节代码。可用代码列表可在fourcc.org中:<http://www.fourcc.org/codecs.php> 找到。它取决于平台。遵循编解码器对我来说效果很好。

- 在Fedora中: DIVX, XVID, MJPG, X264, WMV1, WMV2。(最好使用XVID。MJPG会生成大尺寸的视频。X264会生成非常小的尺寸的视频)
- 在Windows中: DIVX (尚待测试和添加)
- 在OSX中: MJPG (.mp4), DIVX (.avi), X264 (.mkv)。

FourCC代码作为MJPG的 `cv.VideoWriter_fourcc('M','J','P','G')` or `cv.VideoWriter_fourcc(*'MJPG')` 传递。

在从摄像机捕获的代码下面, 沿垂直方向翻转每一帧并保存。

```
import numpy as np
import cv2 as cv
```

```
cap = cv.VideoCapture(0)
# 定义编解码器并创建VideoWriter对象
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('output.avi', fourcc, 20.0, (640, 480))
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    frame = cv.flip(frame, 0)
    # 写翻转的框架
    out.write(frame)
    cv.imshow('frame', frame)
    if cv.waitKey(1) == ord('q'):
        break
# 完成工作后释放所有内容
cap.release()
out.release()
cv.destroyAllWindows()
```

其他资源

练习题

OpenCV中的绘图功能

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 学习使用OpenCV绘制不同的几何形状
- 您将学习以下功能：`cv.line()`，`cv.circle()`，`cv.rectangle()`，`cv.ellipse()`，`cv.putText()`等。

代码

在上述所有功能中，您将看到一些常见的参数，如下所示：

- `img`：您要绘制形状的图像
- `color`：形状的颜色。对于BGR，将其作为元组传递，例如：`(255,0,0)`对于蓝色。对于灰度，只需传递标量值即可。
- `厚度`：线或圆等的粗细。如果对闭合图形（如圆）传递 `-1`，它将填充形状。默认厚度= 1
- `lineType`：线的类型，是否为8连接线，抗锯齿线等。默认情况下，为8连接线。
`cv.LINE_AA`给出了抗锯齿的线条，看起来非常适合曲线。

画线

要绘制一条线，您需要传递线的开始和结束坐标。我们将创建一个黑色图像，并从左上角到右下角在其上绘制一条蓝线。

```
import numpy as np
import cv2 as cv
# 创建黑色的图像
img = np.zeros((512,512,3), np.uint8)
# 绘制一条厚度为5的蓝色对角线
cv.line(img, (0,0), (511,511), (255,0,0), 5)
```

画矩形

要绘制矩形，您需要矩形的左上角和右下角。这次，我们将在图像的右上角绘制一个绿色矩形。

```
cv.rectangle(img, (384,0), (510,128), (0,255,0), 3)
```

画圆圈

要绘制一个圆，需要其中心坐标和半径。我们将在上面绘制的矩形内绘制一个圆。

```
cv.circle(img, (447,63), 63, (0,0,255), -1)
```

画椭圆

要绘制椭圆，我们需要传递几个参数。一个参数是中心位置 (x , y)。下一个参数是轴长度 (长轴长度，短轴长度)。angle是椭圆沿逆时针方向旋转的角度。startAngle和endAngle表示从主轴沿顺时针方向测量的椭圆弧的开始和结束。即给出0和360给出完整的椭圆。有关更多详细信息，请参阅**cv.ellipse**的文档。下面的示例在图像的中心绘制一个椭圆形。

```
cv.ellipse(img, (256,256), (100,50), 0, 0, 180, 255, -1)
```

画多边形

要绘制多边形，首先需要顶点的坐标。将这些点组成形状为 `ROWSx1x2` 的数组，其中 `ROWS` 是顶点数，并且其类型应为int32。在这里，我们绘制了一个带有四个顶点的黄色小多边形。

```
pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)
pts = pts.reshape((-1,1,2))
cv.polylines(img, [pts], True, (0,255,255))
```

注意 如果第三个参数为False，您将获得一条连接所有点的折线，而不是闭合形状。

cv.polylines()可用于绘制多条线。只需创建要绘制的所有线条的列表，然后将其传递给函数即可。所有线条将单独绘制。与为每条线调用**cv.line**相比，绘制一组线是一种更好，更快的方法。

向图像添加文本：

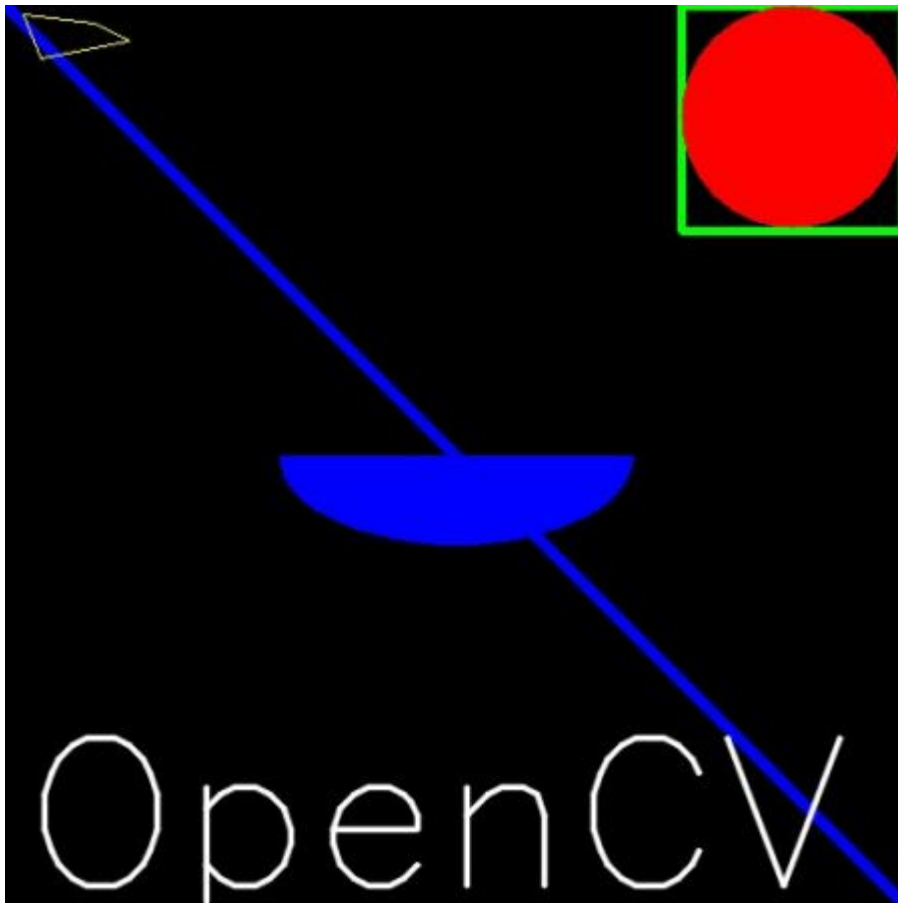
要将文本放入图像中，需要指定以下内容。 - 您要写入的文字数据 - 您要放置它的位置坐标 (即数据开始的左下角)。 - 字体类型 (检查**cv.putText**文档以获取受支持的字体) - 字体比例 (指定字体大小) - 常规的内容，例如颜色，厚度，线条类型等。为了获得更好的外观，建议使用 `lineType = cv.LINE_AA`。

我们将在白色图像上写入**OpenCV**。

```
font = cv.FONT_HERSHEY_SIMPLEX  
cv.putText(img, 'OpenCV', (10, 500), font, 4, (255, 255, 255), 2, cv.LINE_AA)
```

结果

现在是时候查看我们绘图的最终结果了。正如您在以前的文章中学习的那样，显示图像以查看它。



其他资源

1. 椭圆函数中使用的角度不是我们的圆角。有关更多详细信息，请访问此讨论：<http://answers.opencv.org/question/14541/angles-in-ellipse-function/>。

练习题

1. 尝试使用OpenCV中可用的绘图功能创建OpenCV的徽标。

鼠标作为画笔

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 了解如何在OpenCV中处理鼠标事件
- 您将学习以下功能：**cv.setMouseCallback()**

简单演示

在这里，我们创建一个简单的应用程序，无论我们在哪里双击它，都可以在图像上绘制一个圆。

首先，我们创建一个鼠标回调函数，该函数在发生鼠标事件时执行。鼠标事件可以是与鼠标相关的任何事物，例如左键按下，左键按下，左键双击等。它为我们提供了每个鼠标事件的坐标(x, y)。通过此活动和地点，我们可以做任何我们喜欢的事情。要列出所有可用的可用事件，请在Python终端中运行以下代码：

```
import cv2 as cv
events = [i for i in dir(cv) if 'EVENT' in i]
print( events )
```

创建鼠标回调函数具有特定的格式，该格式在所有地方都相同。它仅在功能上有所不同。因此，我们的鼠标回调函数可以做一件事，在我们双击的地方绘制一个圆圈。因此，请参见下面的代码。代码在注释中是不言自明的：

```
import numpy as np
import cv2 as cv
# 鼠标回调函数
def draw_circle(event,x,y,flags,param):
    if event == cv.EVENT_LBUTTONDBLCLK:
        cv.circle(img, (x,y), 100, (255,0,0), -1)
# 创建一个黑色的图像，一个窗口，并绑定到窗口的功能
img = np.zeros((512,512,3), np.uint8)
```

```
cv.namedWindow('image')
cv.setMouseCallback('image', draw_circle)
while(1):
    cv.imshow('image', img)
    if cv.waitKey(20) & 0xFF == 27:
        break
cv.destroyAllWindows()
```

更高级的演示

现在我们去寻找一个更好的应用。在这里，我们通过拖动鼠标来绘制矩形或圆形(取决于我们选择的模式)，就像我们在 Paint 应用程序中所做的那样。所以我们的鼠标回调函数有两部分，一部分用于绘制矩形，另一部分用于绘制圆形。这个具体的例子对于创建和理解一些交互式应用程序非常有帮助，比如目标跟踪，图像分割地图等等。

```
import numpy as np
import cv2 as cv
drawing = False # 如果按下鼠标，则为真
mode = True # 如果为真，绘制矩形。按 m 键可以切换到曲线
ix, iy = -1, -1
# 鼠标回调函数
def draw_circle(event, x, y, flags, param):
    global ix, iy, drawing, mode
    if event == cv.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv.EVENT_MOUSEMOVE:
        if drawing == True:
            if mode == True:
                cv.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)
            else:
                cv.circle(img, (x, y), 5, (0, 0, 255), -1)
    elif event == cv.EVENT_LBUTTONUP:
        drawing = False
        if mode == True:
            cv.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)
        else:
            cv.circle(img, (x, y), 5, (0, 0, 255), -1)
```


其他资源

练习题

1. 在最后一个示例中，我们绘制了填充矩形。您修改代码以绘制一个未填充的矩形。

轨迹栏作为调色板

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 了解将轨迹栏固定到OpenCV窗口
- 您将学习以下功能：`cv.getTrackbarPos`，`cv.createTrackbar`等。

代码演示

在这里，我们将创建一个简单的应用程序，以显示您指定的颜色。您有一个显示颜色的窗口，以及三个用于指定B、G、R颜色的跟踪栏。滑动轨迹栏，并相应地更改窗口颜色。默认情况下，初始颜色将设置为黑色。

对于 `cv.getTrackbarPos()` 函数，第一个参数是轨迹栏名称，第二个参数是它附加到的窗口名称，第三个参数是默认值，第四个参数是最大值，第五个是执行的回调函数每次跟踪栏值更改。回调函数始终具有默认参数，即轨迹栏位置。在我们的例子中，函数什么都不做，所以我们简单地通过。

轨迹栏的另一个重要应用是将其用作按钮或开关。默认情况下，OpenCV不具有按钮功能。因此，您可以使用轨迹栏获得此类功能。在我们的应用程序中，我们创建了一个开关，只有在该开关为ON的情况下，该应用程序才能在其中运行，否则屏幕始终为黑色。

```
import numpy as np
import cv2 as cv
def nothing(x):
    pass
# 创建一个黑色的图像，一个窗口
img = np.zeros((300,512,3), np.uint8)
cv.namedWindow('image')
# 创建颜色变化的轨迹栏
cv.createTrackbar('R','image',0,255,nothing)
cv.createTrackbar('G','image',0,255,nothing)
```

```
cv.createTrackbar('B', 'image', 0, 255, nothing)
# 为 ON/OFF 功能创建开关
switch = '0 : OFF \n1 : ON'
cv.createTrackbar(switch, 'image', 0, 1, nothing)
while(1):
    cv.imshow('image', img)
    k = cv.waitKey(1) & 0xFF
    if k == 27:
        break
    # 得到四条轨迹的当前位置
    r = cv.getTrackbarPos('R', 'image')
    g = cv.getTrackbarPos('G', 'image')
    b = cv.getTrackbarPos('B', 'image')
    s = cv.getTrackbarPos(switch, 'image')
    if s == 0:
        img[:] = 0
    else:
        img[:] = [b, g, r]
cv.destroyAllWindows()
```

该应用程序的屏幕截图如下所示：



练习题

1. 使用轨迹栏创建颜色和画笔半径可调的Paint应用程序。有关绘制的信息，请参阅有关鼠标处理的先前教程。

图像的基本操作

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

学会： - 访问像素值并修改它们 - 访问图像属性 - 设置感兴趣区域(ROI) - 分割和合并图像

本节中的几乎所有操作都主要与Numpy相关，而不是与OpenCV相关。要使用OpenCV编写更好的优化代码，需要Numpy的丰富知识。

(由于大多数示例都是单行代码，因此示例将在Python终端中显示)

访问和修改像素值

让我们先加载彩色图像：

```
>>> import numpy as np
>>> import cv2 as cv
>>> img = cv.imread('messi5.jpg')
```

你可以通过行和列坐标来访问像素值。对于 BGR 图像，它返回一个由蓝色、绿色和红色值组成的数组。对于灰度图像，只返回相应的灰度。

```
>>> px = img[100,100]
>>> print( px )
[157 166 200]
# 仅访问蓝色像素
>>> blue = img[100,100,0]
>>> print( blue )
157
```

你可以用相同的方式修改像素值。

```
>>> img[100,100] = [255,255,255]
>>> print( img[100,100] )
[255 255 255]
```

警告

Numpy是用于快速数组计算的优化库。因此，简单地访问每个像素值并对其进行修改将非常缓慢，因此不建议使用。

注意 上面的方法通常用于选择数组的区域，例如前5行和后3列。对于单个像素访问，Numpy数组方法`array.item()`和`array.itemset()`被认为更好，但是它们始终返回标量。如果要访问所有B，G，R值，则需要分别调用所有的`array.item()`。

更好的像素访问和编辑方法：

```
# 访问 RED 值
>>> img.item(10,10,2)
59
# 修改 RED 值
>>> img.itemset((10,10,2),100)
>>> img.item(10,10,2)
100
```

访问图像属性

图像属性包括行数，列数和通道数，图像数据类型，像素数等。

图像的形状可通过 `img.shape` 访问。它返回行，列和通道数的元组（如果图像是彩色的）：

```
>>> print( img.shape )
(342, 548, 3)
```

注意 如果图像是灰度的，则返回的元组仅包含行数和列数，因此这是检查加载的图像是灰度还是彩色的好方法。

像素总数可通过访问 `img.size`：

```
>>> print( img.size )
562248
```

图像数据类型通过 `img.dtype` 获得：

```
>>> print( img.dtype )
uint8
```

注意 `img.dtype`在调试时非常重要，因为OpenCV-Python代码中的大量错误是由无效的数据类型引起的。

图像感兴趣区域ROI

有时候，你不得不处理一些特定区域的图像。对于图像中的眼睛检测，首先对整个图像进行人脸检测。在获取人脸图像时，我们只选择人脸区域，搜索其中的眼睛，而不是搜索整个图像。它提高了准确性(因为眼睛总是在面部上:D)和性能(因为我们搜索的区域很小)。

使用Numpy索引再次获得ROI。在这里，我要选择球并将其复制到图像中的另一个区域：

```
>>> ball = img[280:340, 330:390]
>>> img[273:333, 100:160] = ball
```

检查以下结果：



拆分和合并图像通道

有时你需要分别处理图像的B，G，R通道。在这种情况下，你需要将BGR图像拆分为单个通道。在其他情况下，你可能需要将这些单独的频道加入BGR图片。你可以通过以下方式简单地做到这一点：

```
>>> b,g,r = cv.split(img) >>> img = cv.merge((b,g,r))
```

要么

```
>>> b = img[:, :, 0]
```

假设你要将所有红色像素都设置为零，则无需先拆分通道。numpy索引更快：

```
>>> img[:, :, 2] = 0
```

警告

`cv.split()` 是一项耗时的操作（就时间而言）。因此，仅在必要时才这样做。否则请进行Numpy索引。

为图像设置边框（填充）

如果要在图像周围创建边框（如相框），则可以使用 `cv.copyMakeBorder()`。但是它在卷积运算，零填充等方面有更多应用。此函数采用以下参数：

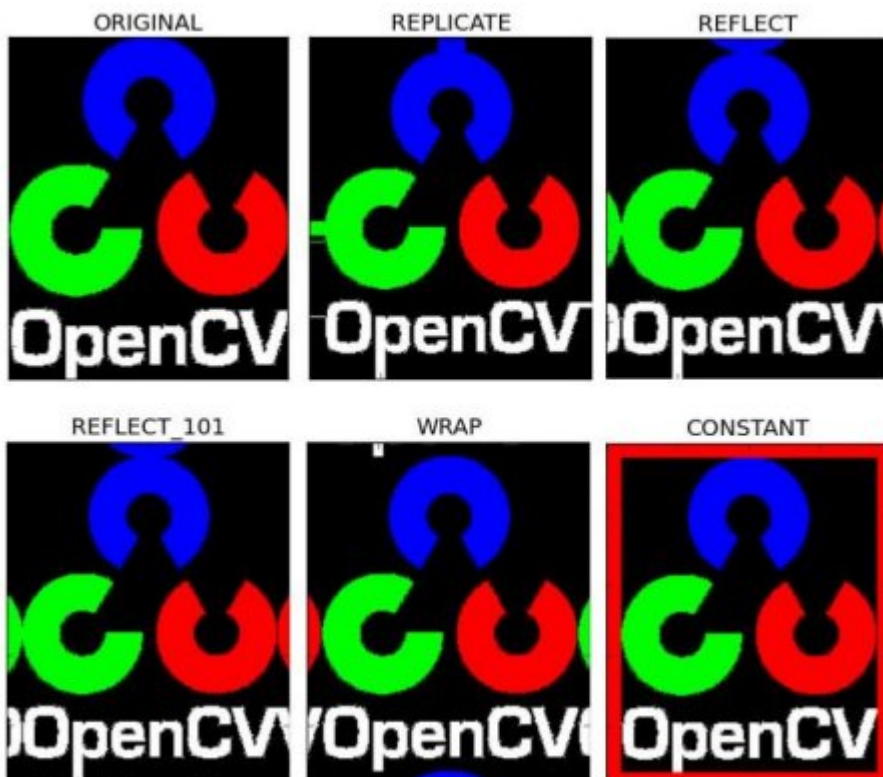
- **src** - 输入图像
- **top** , **bottom** , **left** , **right** 边界宽度（以相应方向上的像素数为单位）
- **borderType** - 定义要添加哪种边框的标志。它可以是以下类型：
 - **cv.BORDER_CONSTANT** - 添加恒定的彩色边框。该值应作为下一个参数给出。
 - **cv.BORDER_REFLECT** - 边框将是边框元素的镜像，如下所示：*fedcba | abcdefgh | hgfedcb*
 - **cv.BORDER_REFLECT_101** 或 **cv.BORDER_DEFAULT** 与上述相同，但略有变化，例如：*gfedcb | abcdefgh | gfedcba*
 - **cv.BORDER_REPLICATE** 最后一个元素被复制，像这样：*aaaaaa | abcdefgh | hhhhhhhh*

- ****cv.BORDER_WRAP****难以解释，它看起来像这样：*cdefgh | abcdefgh | abcdefg*
- **value** -边框的颜色，如果边框类型为****cv.BORDER_CONSTANT****

下面是一个示例代码，演示了所有这些边框类型，以便更好地理解：

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
BLUE = [255,0,0]
img1 = cv.imread('opencv-logo.png')
replicate = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_REPLICATE)
reflect = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_REFLECT)
reflect101 = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_REFLECT_101)
wrap = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_WRAP)
constant= cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_CONSTANT,value=BLUE)
plt.subplot(231),plt.imshow(img1,'gray'),plt.title('ORIGINAL')
plt.subplot(232),plt.imshow(replicate,'gray'),plt.title('REPLICATE')
plt.subplot(233),plt.imshow(reflect,'gray'),plt.title('REFLECT')
plt.subplot(234),plt.imshow(reflect101,'gray'),plt.title('REFLECT_101')
plt.subplot(235),plt.imshow(wrap,'gray'),plt.title('WRAP')
plt.subplot(236),plt.imshow(constant,'gray'),plt.title('CONSTANT')
plt.show()
```

请参阅下面的结果。（图像与matplotlib一起显示。因此红色和蓝色通道将互换）：



其他资源

练习题

图像上的算术运算

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 学习图像的几种算术运算，例如加法，减法，按位运算等。
- 您将学习以下功能：**cv.add**，**cv.addWeighted**等。

图像加法

您可以通过OpenCV函数 `cv.add()` 或仅通过numpy操作 `res = img1 + img2` 添加两个图像。两个图像应具有相同的深度和类型，或者第二个图像可以只是一个标量值。

注意 OpenCV加法和Numpy加法之间有区别。OpenCV加法是饱和运算，而Numpy加法是模运算。

例如，考虑以下示例：

```
>>> x = np.uint8([250])
>>> y = np.uint8([10])
>>> print( cv.add(x,y) ) # 250+10 = 260 => 255
[[255]]
>>> print( x+y )        # 250+10 = 260 % 256 = 4
[4]
```

当添加两个图像时，它将更加可见。OpenCV功能将提供更好的结果。因此，始终最好坚持使用OpenCV功能。

图像融合

这也是图像加法，但是对图像赋予不同的权重，以使其具有融合或透明的感觉。根据以下等式添加图像：

$$G(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

通过将 α 从 $0 \rightarrow 1$ 更改，您可以在一个图像到另一个图像之间执行很酷的过渡。

在这里，我拍摄了两个图像，将它们融合在一起。第一幅图像的权重为0.7，第二幅图像的权重为0.3。 `cv.addWeighted()` 在图像上应用以下公式。

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

在这里， γ 被视为零。

```
img1 = cv.imread('ml.png')
img2 = cv.imread('opencv-logo.png')
dst = cv.addWeighted(img1, 0.7, img2, 0.3, 0)
cv.imshow('dst', dst)
cv.waitKey(0)
cv.destroyAllWindows()
```



检查以下结果：

按位运算

这包括按位 `AND`、`OR`、`NOT` 和 `XOR` 操作。它们在提取图像的任何部分(我们将在后面的章节中看到)、定义和处理非矩形 ROI 等方面非常有用。下面我们将看到一个例子，如何改变一个图像的特定区域。我想把 OpenCV 的标志放在一个图像上面。如果我添加两个图像，它会改变颜色。如果我混合它，我得到一个透明的效果。但我希望它是不透明的。如果是一个矩形区域，我可以使使用 ROI，就像我们在上一章中所做的那样。但是 OpenCV 的 logo 不是长方形的。所以您可以使用如下的按位操作来实现：

我想在图像上方放置 OpenCV 徽标。如果添加两个图像，它将改变颜色。如果混合它，我将获得透明效果。但我希望它不透明。如果是矩形区域，则可以像上一章一样使用 ROI。但是 OpenCV 徽标不是矩形。因此，您可以按如下所示进行按位操作：

```
# 加载两张图片
img1 = cv.imread('messi5.jpg')
img2 = cv.imread('opencv-logo-white.png')
# 我想把logo放在左上角,所以我创建了ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]
# 现在创建logo的掩码,并同时创建其相反掩码
img2gray = cv.cvtColor(img2,cv.COLOR_BGR2GRAY)
ret, mask = cv.threshold(img2gray, 10, 255, cv.THRESH_BINARY)
mask_inv = cv.bitwise_not(mask)
# 现在将ROI中logo的区域涂黑
img1_bg = cv.bitwise_and(roi,roi,mask = mask_inv)
# 仅从logo图像中提取logo区域
img2_fg = cv.bitwise_and(img2,img2,mask = mask)
# 将logo放入ROI并修改主图像
dst = cv.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst
cv.imshow('res',img1)
cv.waitKey(0)
cv.destroyAllWindows()
```

请看下面的结果。左图显示了我们创建的mask。右图显示最终结果。为了更好地理解,显示上面代码中的所有中间映像,特别是 `img1_bg` 和 `img2_fg`。



其他资源

练习题

1. 使用 `cv.addWeighted` 函数在文件夹中创建图像的幻灯片放映,并在图像之间进行平滑过渡

性能衡量和提升技术

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在图像处理中，由于每秒要处理大量操作，因此必须使代码不仅提供正确的解决方案，而且还必须以最快的方式提供。因此，在本章中，你将学习

- 衡量代码的性能。
- 一些提高代码性能的技巧。
- 你将看到以下功能：**cv.getTickCount**，**cv.getTickFrequency**等。

除了OpenCV，Python还提供了一个模块**time**，这有助于衡量执行时间。另一个模块**profile**有助于获取有关代码的详细报告，例如代码中每个函数花费了多少时间，调用了函数的次数等。但是，如果你使用的是IPython，则所有这些功能都集成在用户友好的界面中方式。我们将看到一些重要的信息，有关更多详细信息，请查看“**其他资源**”部分中的链接。

使用OpenCV衡量性能

cv.getTickCount函数返回从参考事件（如打开机器的那一刻）到调用此函数那一刻之间的时钟周期数。因此，如果在函数执行之前和之后调用它，则会获得用于执行函数的时钟周期数。

cv.getTickFrequency函数返回时钟周期的频率或每秒的时钟周期数。因此，要找到执行时间（以秒为单位），你可以执行以下操作：

```
e1 = cv.getTickCount()
# 你的执行代码
e2 = cv.getTickCount()
time = (e2 - e1) / cv.getTickFrequency()
```

我们将通过以下示例进行演示。下面的示例应用中位数过滤，其内核的奇数范围为5到49。（不必担心结果会是什么样，这不是我们的目标）：

```
img1 = cv.imread('messi5.jpg')
e1 = cv.getTickCount()
for i in range(5,49,2):
    img1 = cv.medianBlur(img1,i)
e2 = cv.getTickCount()
t = (e2 - e1)/cv.getTickFrequency()
print( t )
# 我得到的结果是0.521107655秒
```

注意 你可以使用时间模块执行相同的操作。代替cv.getTickCount，使用time.time()函数。然后取两次相差。

OpenCV中的默认优化

许多 OpenCV 函数都是使用 SSE2、AVX 等进行优化的。它还包含未优化的代码。因此，如果我们的系统支持这些特性，我们就应该利用它们(几乎所有现代的处理器的都支持它们)。在编译时默认启用它。因此，如果启用了 OpenCV，它将运行优化的代码，否则它将运行未优化的代码。你可以使用 **cvUseOptimized** 检查是否启用 / 禁用和 **cvSetUseOptimized** 以启用 / 禁用它。让我们看一个简单的例子。

检查是否启用了优化

```
# 检查是否启用了优化
In [5]: cv.useOptimized()
Out[5]: True
In [6]: %timeit res = cv.medianBlur(img,49)
10 loops, best of 3: 34.9 ms per loop
# 关闭它
In [7]: cv.setUseOptimized(False)
In [8]: cv.useOptimized()
Out[8]: False
In [9]: %timeit res = cv.medianBlur(img,49)
10 loops, best of 3: 64.1 ms per loop
```

看，优化的中值滤波比未优化的版本快2倍。如果你检查其来源，你可以看到中值滤波是 SIMD 优化。因此，你可以使用它在代码顶部启用优化(请记住，它是默认启用的)

在IPython中衡量性能

有时你可能需要比较两个类似操作的性能。IPython为你提供了一个神奇的命令计时器来执行此操作。它会多次运行代码以获得更准确的结果。同样，它们适用于测量单行代码。

例如，你知道以下哪个加法运算更好，`x = 5; y = x**2, x = 5; y = x*x, x = np.uint8([5]); y = x*x` 或 `y = np.square(x)`？我们将在IPython shell中使用timeit得到答案。

```
In [10]: x = 5

In [11]: %计时 y=x**2
10000000 loops, best of 3: 73 ns per loop

In [12]: %计时 y=x*x
10000000 loops, best of 3: 58.3 ns per loop

In [15]: z = np.uint8([5])

In [17]: %计时 y=z*z
1000000 loops, best of 3: 1.25 us per loop

In [19]: %计时 y=np.square(z)
1000000 loops, best of 3: 1.16 us per loop
```

你可以看到`x = 5; y = x * x`最快，比Numpy快20倍左右。如果你还考虑阵列的创建，它可能会快100倍。酷吧？（大量开发人员正在研究此问题）

注意 Python标量操作比Numpy标量操作快。因此，对于包含一两个元素的运算，Python标量比Numpy数组好。当数组大小稍大时，Numpy会占优势。

我们将再尝试一个示例。这次，我们将比较`cv.countNonZero`和`np.count_nonzero`对于同一张图片的性能。

```
In [35]: %计时 z = cv.countNonZero(img)
100000 loops, best of 3: 15.8 us per loop
In [36]: %计时 z = np.count_nonzero(img)
1000 loops, best of 3: 370 us per loop
```

看，OpenCV 函数比 Numpy 函数快近25倍。

注意 通常，OpenCV函数比Numpy函数要快。因此，对于相同的操作，首选OpenCV功能。但是，可能会有例外，尤其是当Numpy处理视图而不是副本时。

更多IPython魔术命令

还有其他一些魔术命令可以用来测量性能，性能分析，行性能分析，内存测量等。它们都有很好的文档记录。因此，此处仅提供指向这些文档的链接。建议有兴趣的读者尝试一下。

性能优化技术

有几种技术和编码方法可以充分利用 Python 和 Numpy 的最大性能。这里只注明相关信息，并提供重要信息来源的链接。这里要注意的主要事情是，首先尝试以一种简单的方式实现算法。一旦它运行起来，分析它，找到瓶颈并优化它们。

1. 尽量避免在Python中使用循环，尤其是双/三重循环等。它们本来就很慢。
2. 由于Numpy和OpenCV已针对向量运算进行了优化，因此将算法/代码向量化到最大程度。
3. 利用缓存一致性。
4. 除非需要，否则切勿创建数组的副本。尝试改用视图。数组复制是一项昂贵的操作。

即使执行了所有这些操作后，如果你的代码仍然很慢，或者不可避免地需要使用大循环，请使用Cython等其他库来使其更快。

其他资源

1. Python优化技术：<http://wiki.python.org/moin/PythonSpeed/PerformanceTips>
2. Scipy讲义- 高级Numpy：http://scipy-lectures.github.io/advanced/advanced_numpy/index.html#advanced-numpy
3. IPython中的时序和性能分析：<http://pynash.org/2013/03/06/timing-and-profiling/>

练习题

改变颜色空间

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 在本教程中，你将学习如何将图像从一个色彩空间转换到另一个，像BGR↔灰色，BGR↔HSV等
- 除此之外，我们还将创建一个应用程序，以提取视频中的彩色对象
- 你将学习以下功能：**cv.cvtColor**，**cv.inRange**等。

改变颜色空间

OpenCV中有超过150种颜色空间转换方法。但是我们将研究只有两个最广泛使用的,BGR↔灰色和BGR↔HSV。

对于颜色转换，我们使用cv函数。cvtColor(input_image, flag)，其中flag决定转换的类型。

对于BGR→灰度转换，我们使用标志cv.COLOR_BGR2GRAY。类似地，对于BGR→HSV，我们使用标志cv.COLOR_BGR2HSV。要获取其他标记，只需在Python终端中运行以下命令：

```
>>> import cv2 as cv
>>> flags = [i for i in dir(cv) if i.startswith('COLOR_')]
>>> print( flags )
```

注意 HSV的色相范围为[0,179]，饱和度范围为[0,255]，值范围为[0,255]。不同的软件使用不同的规模。因此，如果你要将OpenCV值和它们比较，你需要将这些范围标准化。

对象追踪

现在我们知道了如何将BGR图像转换成HSV，我们可以使用它来提取一个有颜色的对象。在HSV中比在BGR颜色空间中更容易表示颜色。在我们的应用程序中，我们将尝试提取一个蓝色的对

象。方法如下: - 取视频的每一帧 - 转换从BGR到HSV颜色空间 - 我们对HSV图像设置蓝色范围的阈值 - 现在单独提取蓝色对象, 我们可以对图像做任何我们想做的事情。

下面是详细注释的代码:

```
import cv2 as cv
import numpy as np
cap = cv.VideoCapture(0)
while(1):
    # 读取帧
    _, frame = cap.read()
    # 转换颜色空间 BGR 到 HSV
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    # 定义HSV中蓝色的范围
    lower_blue = np.array([110, 50, 50])
    upper_blue = np.array([130, 255, 255])
    # 设置HSV的阈值使得只取蓝色
    mask = cv.inRange(hsv, lower_blue, upper_blue)
    # 将掩膜和图像逐像素相加
    res = cv.bitwise_and(frame, frame, mask= mask)
    cv.imshow('frame', frame)
    cv.imshow('mask', mask)
    cv.imshow('res', res)
    k = cv.waitKey(5) & 0xFF
    if k == 27:
        break
cv.destroyAllWindows()
```

下图显示了对蓝色对象的跟踪:



注意 图像中有一些噪点。我们将在后面的章节中看到如何删除它们。这是对象跟踪中最简单的方法。一旦学习了轮廓的功能, 你就可以做很多事情, 例如找到该对象的质心并使用它来跟踪对象, 仅通过将手移到相机前面以及其他许多有趣的东西就可以绘制图表。

如何找到要追踪的HSV值？

这是在stackoverflow.com上发现的一个常见问题。它非常简单，你可以使用相同的函数 `**cv.cvtColor()**`。你只需传递你想要的BGR值，而不是传递图像。例如，要查找绿色的HSV值，请在Python终端中尝试以下命令：

```
>>> green = np.uint8([[[0,255,0 ]]])
>>> hsv_green = cv.cvtColor(green,cv.COLOR_BGR2HSV)
>>> print( hsv_green )
[[[ 60 255 255]]]
```

现在把 `[H- 10,100,100]` 和 `[H+ 10,255, 255]` 分别作为下界和上界。除了这个方法之外，你可以使用任何图像编辑工具(如GIMP或任何在线转换器)来查找这些值，但是不要忘记调整HSV范围。

其他资源

练习题

1. 尝试找到一种方法来提取多个彩色对象，例如，同时提取红色，蓝色，绿色对象。

图像的几何变换

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 学习将不同的几何变换应用到图像上，如平移、旋转、仿射变换等。
- 你会看到这些函数: **cv.getPerspectiveTransform**

变换

OpenCV提供了两个转换函数**cv.warpAffine**和**cv.warpPerspective**，您可以使用它们进行各种转换。**cv.warpAffine**采用2x3转换矩阵，而**cv.warpPerspective**采用3x3转换矩阵作为输入。

缩放

缩放只是调整图像的大小。为此，OpenCV带有一个函数**cv.resize()**。图像的大小可以手动指定，也可以指定缩放比例。也可使用不同的插值方法。首选的插值方法是**cv.INTER_AREA**用于缩小，**cv.INTER_CUBIC** (慢) 和**cv.INTER_LINEAR**用于缩放。默认情况下，出于所有调整大小的目的，使用的插值方法为**cv.INTER_LINEAR**。您可以使用以下方法调整输入图像的大小：

```
import numpy as np
import cv2 as cv
img = cv.imread('messi5.jpg')
res = cv.resize(img, None, fx=2, fy=2, interpolation = cv.INTER_CUBIC)
#或者
height, width = img.shape[:2]
res = cv.resize(img, (2*width, 2*height), interpolation = cv.INTER_CUBIC)
```

平移

平移是物体位置的移动。如果您知道在(x,y)方向上的位移，则将其设为(t_x,t_y)，您可以创建转换矩阵 $\mathbf{M}$ ，如下所示：

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

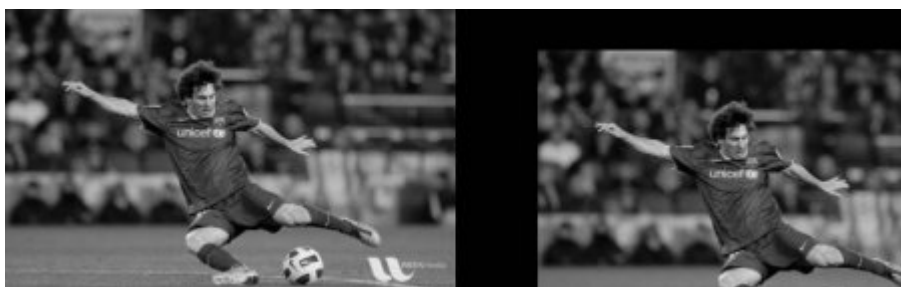
您可以将其放入**np.float32**类型的Numpy数组中，并将其传递给**cv.warpAffine**函数。参见下面偏移为(100, 50)的示例：

```
import numpy as np
import cv2 as cv
img = cv.imread('messi5.jpg',0)
rows,cols = img.shape
M = np.float32([[1,0,100],[0,1,50]])
dst = cv.warpAffine(img,M,(cols,rows))
cv.imshow('img',dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

警告

cv.warpAffine函数的第三个参数是输出图像的大小，其形式应为 (width, height)。记住 width = 列数，height = 行数。

你将看到下面的结果：



旋转

图像旋转角度为 θ 是通过以下形式的变换矩阵实现的：

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

但是OpenCV提供了可缩放的旋转以及可调整的旋转中心，因此您可以在自己喜欢的任何位置旋转。修改后的变换矩阵为

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot \text{center.x} - \beta \cdot \text{center.y} \\ -\beta & \alpha & \beta \cdot \text{center.x} + (1-\alpha) \cdot \text{center.y} \end{bmatrix}$$

其中：

$$\begin{array}{l} \alpha = \text{scale} \cdot \cos \theta, \beta = \text{scale} \cdot \sin \theta \end{array}$$

为了找到此转换矩阵，OpenCV提供了一个函数**cv.getRotationMatrix2D**。请检查以下示例，该示例将图像相对于中心旋转90度而没有任何缩放比例。

```
img = cv.imread('messi5.jpg',0)
rows,cols = img.shape
# cols-1 和 rows-1 是坐标限制
M = cv.getRotationMatrix2D((cols-1)/2.0,(rows-1)/2.0,90,1)
dst = cv.warpAffine(img,M,(cols,rows))
```

查看结果：

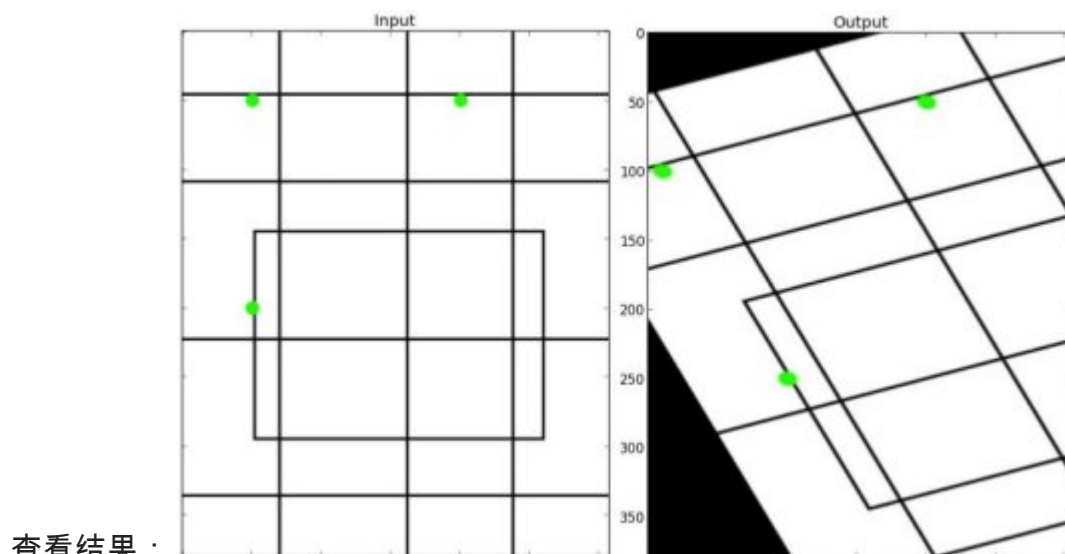


仿射变换

在仿射变换中，原始图像中的所有平行线在输出图像中仍将平行。为了找到变换矩阵，我们需要输入图像中的三个点及其在输出图像中的对应位置。然后**cv.getAffineTransform**将创建一个2x3矩阵，该矩阵将传递给**cv.warpAffine**。

查看以下示例，并查看我选择的点（以绿色标记）：

```
img = cv.imread('drawing.png')
rows,cols,ch = img.shape
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv.getAffineTransform(pts1,pts2)
dst = cv.warpAffine(img,M,(cols,rows))
plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
```



查看结果：

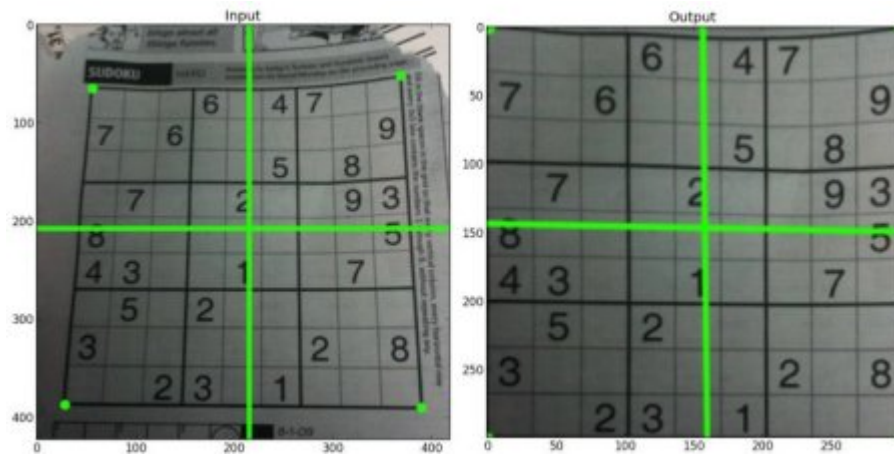
透视变换

对于透视变换，您需要3x3变换矩阵。即使在转换后，直线也将保持直线。要找到此变换矩阵，您需要在输入图像上有4个点，在输出图像上需要相应的点。在这四个点中，其中三个不应共线。然后可以通过函数**cv.getPerspectiveTransform**找到变换矩阵。然后将**cv.warpPerspective**应用于此3x3转换矩阵。

请参见下面的代码：

```
img = cv.imread('sudoku.png')
rows,cols,ch = img.shape
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])
M = cv.getPerspectiveTransform(pts1,pts2)
dst = cv.warpPerspective(img,M,(300,300))
plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

结果：



其他资源

1. "Computer Vision: Algorithms and Applications", Richard Szeliski

练习题

图像阈值

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 在本教程中，您将学习简单阈值，自适应阈值和Otsu阈值。
- 你将学习函数`cv.threshold`和`cv.adaptiveThreshold`。

简单阈值

在这里，问题直截了当。对于每个像素，应用相同的阈值。如果像素值小于阈值，则将其设置为0，否则将其设置为最大值。函数`cv.threshold`用于应用阈值。第一个参数是源图像，它应该是灰度图像。第二个参数是阈值，用于对像素值进行分类。第三个参数是分配给超过阈值的像素值的最大值。OpenCV提供了不同类型的阈值，这由函数的第四个参数给出。通过使用`cv.THRESH_BINARY`类型。所有简单的阈值类型为：

- `cv.THRESH_BINARY`
- `cv.THRESH_BINARY_INV`
- `cv.THRESH_TRUNC`
- `cv.THRESH_TOZERO`
- `cv.THRESH_TOZERO_INV`

请通过类型的文档来观察区别。

该方法返回两个输出。第一个是使用的阈值，第二个输出是阈值后的图像。

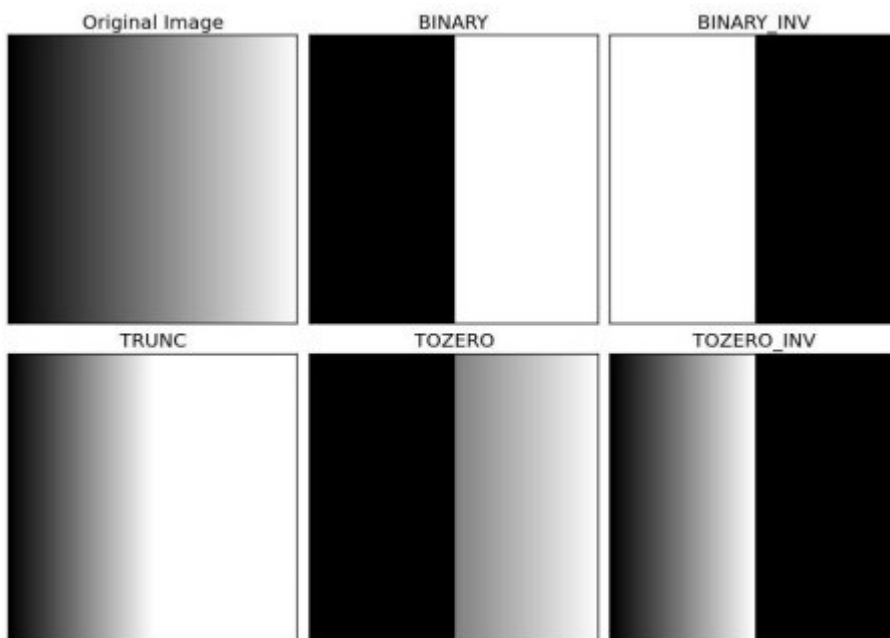
此代码比较了不同的简单阈值类型：

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
```

```
img = cv.imread('gradient.png',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original Image', 'BINARY', 'BINARY_INV', 'TRUNC', 'TOZERO', 'TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in xrange(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))
plt.show()
```

注意 为了绘制多个图像，我们使用 `plt.subplot()` 函数。请查看matplotlib文档以获取更多详细信息。

该代码产生以下结果：



自适应阈值

在上一节中，我们使用一个全局值作为阈值。但这可能并非在所有情况下都很好，例如，如果图像在不同区域具有不同的光照条件。在这种情况下，自适应阈值阈值化可以提供帮助。在此，算法基于像素周围的小区域确定像素的阈值。因此，对于同一图像的不同区域，我们获得了不同的阈值，这为光照度变化的图像提供了更好的结果。

除上述参数外，方法`cv.adaptiveThreshold`还包含三个输入参数：

该`adaptiveMethod`决定阈值是如何计算的：

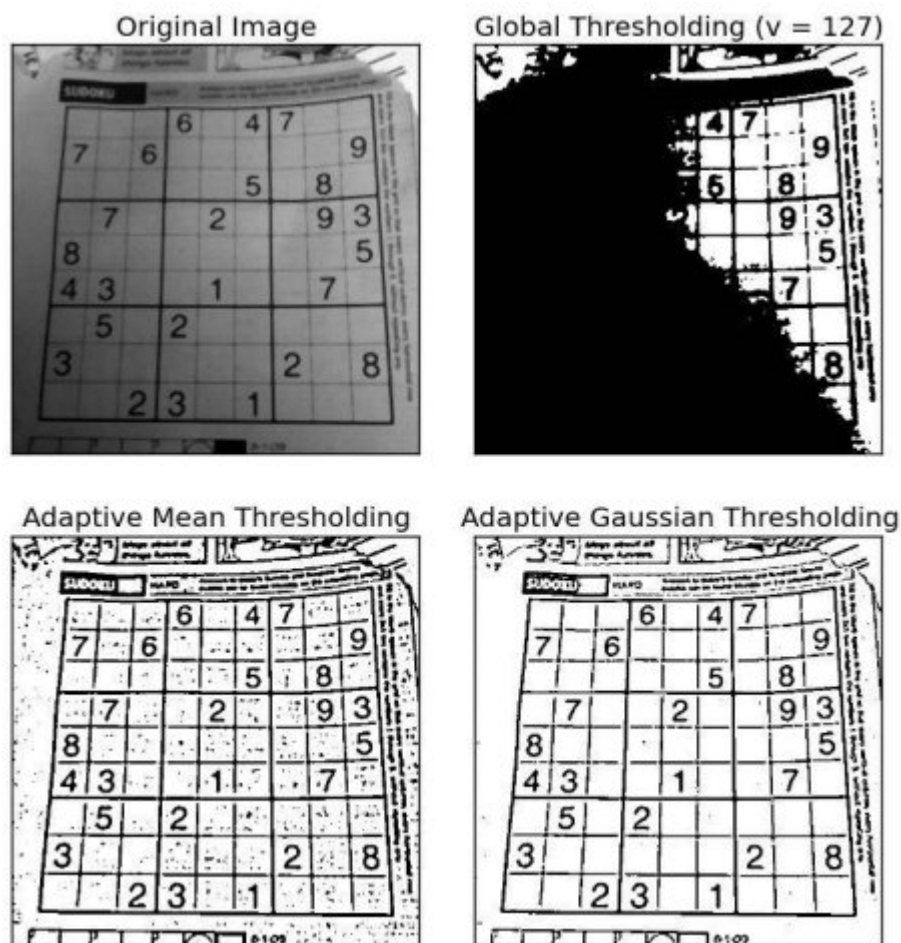
`cv.ADAPTIVE_THRESH_MEAN_C`：阈值是邻近区域的平均值减去常数`C`。

`cv.ADAPTIVE_THRESH_GAUSSIAN_C`：阈值是邻域值的高斯加权总和减去常数`C`。

该`BLOCKSIZE`确定附近区域的大小，`C`是从邻域像素的平均或加权总和中减去的一个常数。

下面的代码比较了光照变化的图像的全局阈值和自适应阈值：

结果：



Otsu的二值化

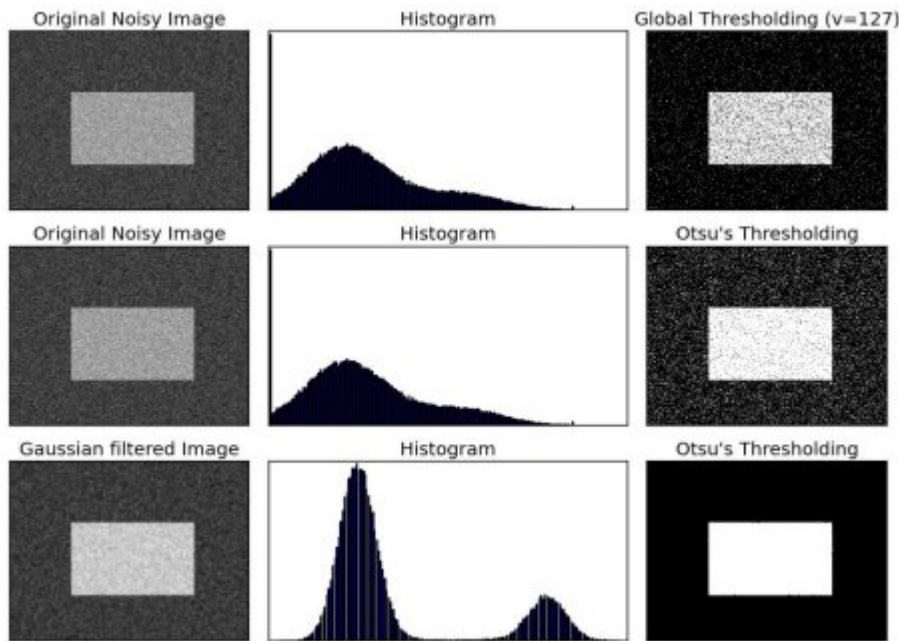
在全局阈值化中，我们使用任意选择的值作为阈值。相反，Otsu的方法避免了必须选择一个值并自动确定它的情况。

考虑仅具有两个不同图像值的图像（双峰图像），其中直方图将仅包含两个峰。一个好的阈值应该在这两个值的中间。类似地，Otsu的方法从图像直方图中确定最佳全局阈值。

为此，使用了**cv.threshold**作为附加标志传递。阈值可以任意选择。然后，算法找到最佳阈值，该阈值作为第一输出返回。

查看以下示例。输入图像为噪点图像。在第一种情况下，采用值为127的全局阈值。在第二种情况下，直接采用Otsu阈值法。在第三种情况下，首先使用5x5高斯核对图像进行滤波以去除噪声，然后应用Otsu阈值处理。了解噪声滤波如何改善结果。

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('noisy2.png',0)
# 全局阈值
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu阈值
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# 高斯滤波后再采用Otsu阈值
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# 绘制所有图像及其直方图
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in xrange(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([], plt.yticks([]))
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([], plt.yticks([]))
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([], plt.yticks([]))
plt.show()
```



结果：

Otsu的二值化如何实现？

本节演示了Otsu二值化的Python实现，以展示其实际工作方式。如果您不感兴趣，可以跳过此步骤。

由于我们正在处理双峰图像，因此Otsu的算法尝试找到一个阈值(t)，该阈值将由关系式给出的**加权类内方差**最小化：

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

其中

$$q_1(t) = \sum_{i=1}^t P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^L P(i)$$

$$\mu_1(t) = \sum_{i=1}^t \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^L \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^L [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

实际上，它找到位于两个峰值之间的 t 值，以使两个类别的差异最小。它可以简单地在Python中实现，如下所示：

```
img = cv.imread('noisy2.png', 0)
blur = cv.GaussianBlur(img, (5, 5), 0)
# 寻找归一化直方图和对应的累积分布函数
hist = cv.calcHist([blur], [0], None, [256], [0, 256])
hist_norm = hist.ravel() / hist.max()
```

```
Q = hist_norm.cumsum()
bins = np.arange(256)
fn_min = np.inf
thresh = -1
for i in xrange(1,256):
    p1,p2 = np.hsplit(hist_norm,[i]) # 概率
    q1,q2 = Q[i],Q[255]-Q[i] # 对类求和
    b1,b2 = np.hsplit(bins,[i]) # 权重
    # 寻找均值和方差
    m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
    v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
    # 计算最小化函数
    fn = v1*q1 + v2*q2
    if fn < fn_min:
        fn_min = fn
        thresh = i
# 使用OpenCV函数找到otsu的阈值
ret, otsu = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print( "{} {}".format(thresh,ret) )
```

其他资源

1. Digital Image Processing, Rafael C. Gonzalez

练习题

1. Otsu的二值化有一些优化。您可以搜索并实现它。

图像平滑

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

学会： - 使用各种低通滤镜模糊图像 - 将定制的滤镜应用于图像（2D卷积）

2D卷积（图像过滤）

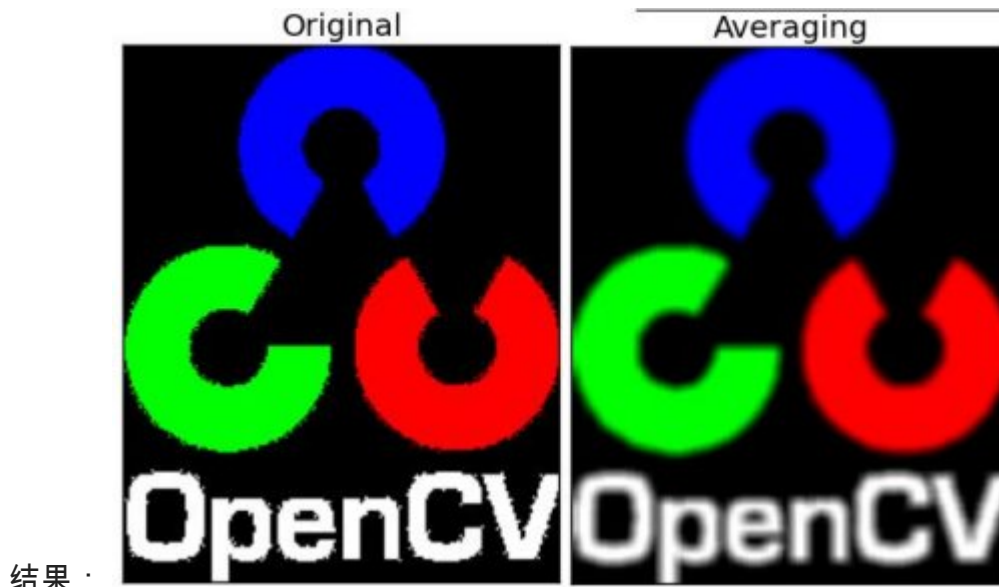
与一维信号一样，还可以使用各种低通滤波器（LPF），高通滤波器（HPF）等对图像进行滤波。LPF有助于消除噪声，使图像模糊等。HPF滤波器有助于在图像中找到边缘。

OpenCV提供了一个函数**cv.filter2D**来将内核与图像进行卷积。例如，我们将尝试对图像进行平均滤波。5x5平均滤波器内核如下所示：

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

操作如下:保持这个内核在一个像素上，将所有低于这个内核的25个像素相加，取其平均值，然后用新的平均值替换中心像素。它将对图像中的所有像素继续此操作。试试这个代码，并检查结果:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('opencv_logo.png')
kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([], plt.yticks([]))
plt.show()
```

结果：

图像模糊（图像平滑）

通过将图像与低通滤波器内核进行卷积来实现图像模糊。这对于消除噪音很有用。它实际上从图像中消除了高频部分（例如噪声，边缘）。因此，在此操作中边缘有些模糊。（有一些模糊技术也可以不模糊边缘）。OpenCV主要提供四种类型的模糊技术。

1.平均

这是通过将图像与归一化框滤镜进行卷积来完成的。它仅获取内核区域下所有像素的平均值，并替换中心元素。这是通过功能**cv.blur()或**cv.boxFilter()完成的。检查文档以获取有关内核的更多详细信息。我们应该指定内核的宽度和高度。3x3归一化框式过滤器如下所示：

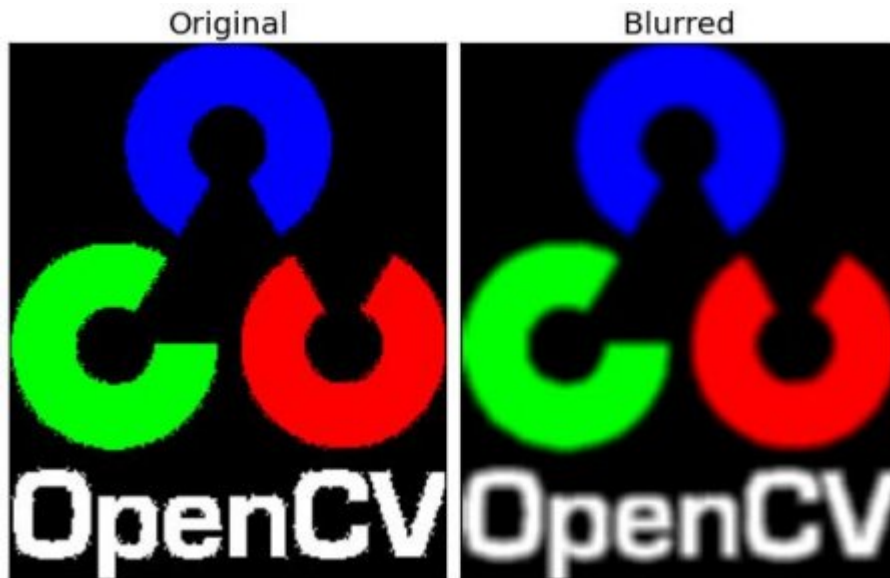
$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

注意 如果您不想使用标准化的框式过滤器，请使用**cv.boxFilter()**。将参数 `normalize = False` 传递给函数。

查看下面的示例演示，其内核大小为5x5：

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('opencv-logo-white.png')
blur = cv.blur(img, (5,5))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
```

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



结果:

2.高斯模糊

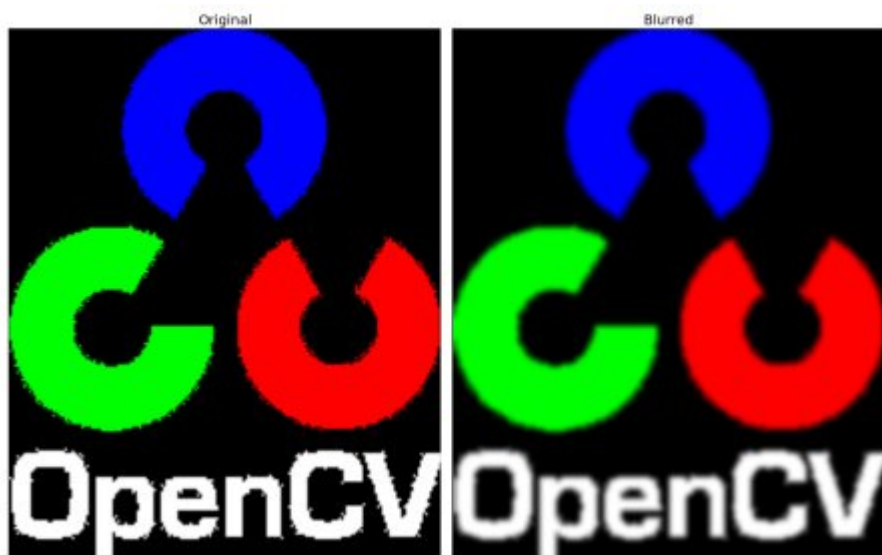
在这种情况下，代替盒式滤波器，使用了高斯核。这是通过功能`cv.GaussianBlur()`完成的。我们应指定内核的宽度和高度，该宽度和高度应为正数和奇数。我们还应指定X和Y方向的标准偏差，分别为`sigmaX`和`sigmaY`。如果仅指定`sigmaX`，则将`sigmaY`与`sigmaX`相同。如果两个都为零，则根据内核大小进行计算。高斯模糊对于从图像中去除高斯噪声非常有效。

如果需要，可以使用函数`cv.getGaussianKernel()`创建高斯内核。

可以修改以上代码以实现高斯模糊：

```
blur = cv.GaussianBlur(img, (5,5), 0)
```

结果：



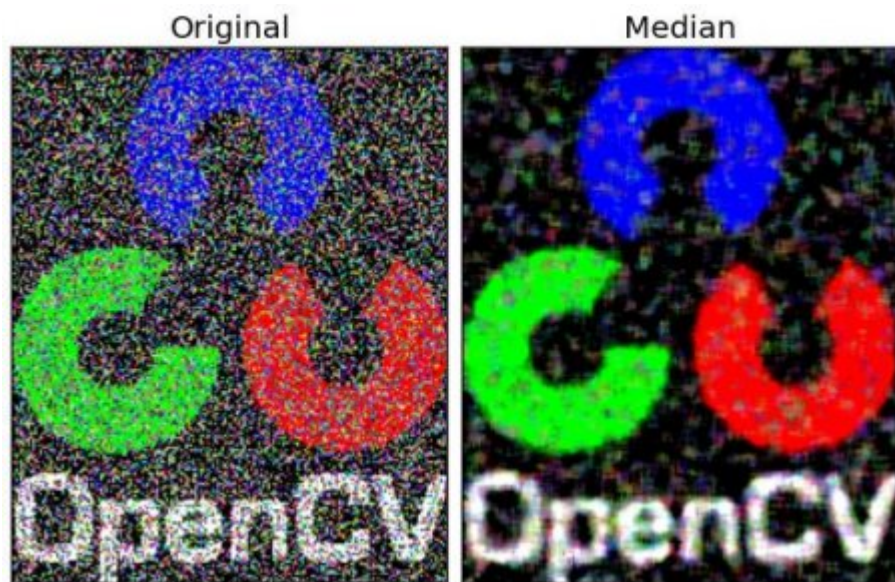
3.中位模糊

在这里，函数`cv.medianBlur()`提取内核区域下所有像素的中值，并将中心元素替换为该中值。这对于消除图像中的椒盐噪声非常有效。有趣的是，在上述过滤器中，中心元素是新计算的值，该值可以是图像中的像素值或新值。但是在中值模糊中，中心元素总是被图像中的某些像素值代替。有效降低噪音。其内核大小应为正奇数整数。

在此演示中，我向原始图像添加了50%的噪声并应用了中值模糊。检查结果：

```
median = cv.medianBlur(img,5)
```

结果：



4. 双边滤波

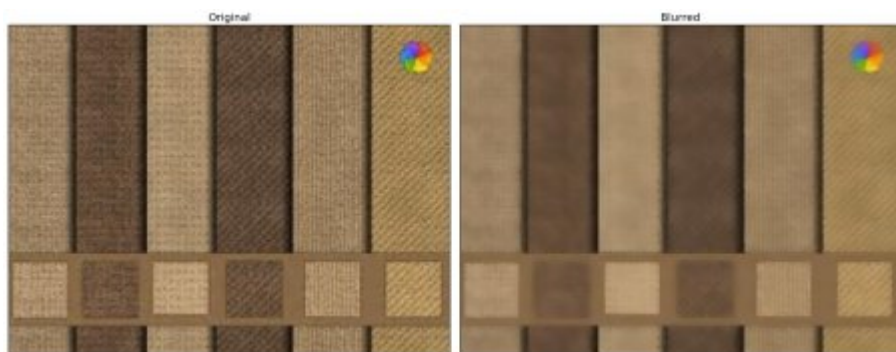
cv.bilateralFilter() 在去除噪声的同时保持边缘清晰锐利非常有效。但是，与其他过滤器相比，该操作速度较慢。我们已经看到，高斯滤波器采用像素周围的邻域并找到其高斯加权平均值。高斯滤波器仅是空间的函数，也就是说，滤波时会考虑附近的像素。它不考虑像素是否具有几乎相同的强度。它不考虑像素是否是边缘像素。因此它也模糊了边缘，这是我們不想做的。

双边滤波器在空间中也采用高斯滤波器，但是又有一个高斯滤波器，它是像素差的函数。空间的高斯函数确保仅考虑附近像素的模糊，而强度差的高斯函数确保仅考虑强度与中心像素相似的那些像素的模糊。由于边缘的像素强度变化较大，因此可以保留边缘。

以下示例显示了使用双边过滤器 (有关参数的详细信息，请访问docs)。

```
blur = cv.bilateralFilter(img, 9, 75, 75)
```

结果：



看到，表面上的纹理消失了，但是边缘仍然保留。

其他资源

1. 有关双边过滤的详细信息：http://people.csail.mit.edu/sparis/bf_course/

练习题

形态学转换

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在这一章当中，我们将学习不同的形态学操作，例如侵蚀，膨胀，开运算，闭运算等。我们将看到不同的功能，例如：`cv.erode()`, `cv.dilate()`, `cv.morphologyEx()`等。

理论

形态变换是一些基于图像形状的简单操作。通常在二进制图像上执行。它需要两个输入，一个是我们的原始图像，第二个是决定**操作性质的结构元素**或**内核**。两种基本的形态学算子是侵蚀和膨胀。然后，它的变体形式（如“打开”，“关闭”，“渐变”等）也开始起作用。在下图的帮助下，我们将一一看到它们：



1. 侵蚀

侵蚀的基本思想就像土壤侵蚀一样，它侵蚀前景物体的边界(尽量使前景保持白色)。它是做什么的呢?内核滑动通过图像(在2D卷积中)。原始图像中的一个像素(无论是1还是0)只有当内核下的所有像素都是1时才被认为是1，否则它就会被侵蚀(变成0)。

结果是，根据内核的大小，边界附近的所有像素都会被丢弃。因此，前景物体的厚度或大小减小，或只是图像中的白色区域减小。它有助于去除小的白色噪声(正如我们在颜色空间章节中看到的)，分离两个连接的对象等。

在这里，作为一个例子，我将使用一个5x5内核，它包含了所有的1。让我们看看它是如何工作的：

```
import cv2 as cv
import numpy as np
img = cv.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
```

结果：



2. 扩张

它与侵蚀正好相反。如果内核下的至少一个像素为“1”，则像素元素为“1”。因此，它会增加图像中的白色区域或增加前景对象的大小。通常，在消除噪音的情况下，腐蚀后会膨胀。因为腐蚀会消除白噪声，但也会缩小物体。因此，我们对其进行了扩展。由于噪音消失了，它们不会回来，但是我们的目标区域增加了。在连接对象的损坏部分时也很有用。

```
dilation = cv.dilate(img,kernel,iterations = 1)
```

结果：



3. 开运算

开放只是**侵蚀然后扩张**的另一个名称。如上文所述，它对于消除噪音很有用。在这里，我们使用函数**cv.morphologyEx**()

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

结果：



4. 闭运算

闭运算与开运算相反，**先扩张然后再侵蚀**。在关闭前景对象内部的小孔或对象上的小黑点时很有用。

```
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```



5. 形态学梯度

这是图像扩张和侵蚀之间的区别。

结果将看起来像对象的轮廓。

```
gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)
```

Uploading gradient.png... (2yruxk2ei)

6. 顶帽

它是输入图像和图像开运算之差。下面的示例针对9x9内核完成。

```
tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)
```

结果：



7. 黑帽

这是输入图像和图像闭运算之差。

```
blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel)
```

结果：



结构元素

在Numpy的帮助下，我们在前面的示例中手动创建了一个结构元素。它是矩形。但是在某些情况下，您可能需要椭圆形/圆形的内核。因此，为此，OpenCV具有一个函数

`cv.getStructuringElement()`。您只需传递内核的形状和大小，即可获得所需的内核。

```
# 矩形内核
>>> cv.getStructuringElement(cv.MORPH_RECT, (5,5))
array([[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]], dtype=uint8)
```



```
# 椭圆内核
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# 十字内核
>>> cv.getStructuringElement(cv.MORPH_CROSS, (5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

其他资源

1. Morphological Operations : <http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm> at HIPR2

练习

图像梯度

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，我们将学习： - 查找图像梯度，边缘等 - 我们将看到以下函数：**cv.Sobel()**，**cv.Scharr()**，**cv.Laplacian()**等

理论

OpenCV提供三种类型的梯度滤波器或高通滤波器，即Sobel，Scharr和Laplacian。我们将看到他们每一种。

1. Sobel 和 Scharr 算子

Sobel算子是高斯平滑加微分运算的联合运算，因此它更抗噪声。逆可以指定要采用的导数方向，垂直或水平（分别通过参数yorder和xorder）。逆还可以通过参数ksize指定内核的大小。如果 `ksize = -1`，则使用3x3 Scharr滤波器，比3x3 Sobel滤波器具有更好的结果。请参阅文档以了解所使用的内核。

2. Laplacian 算子

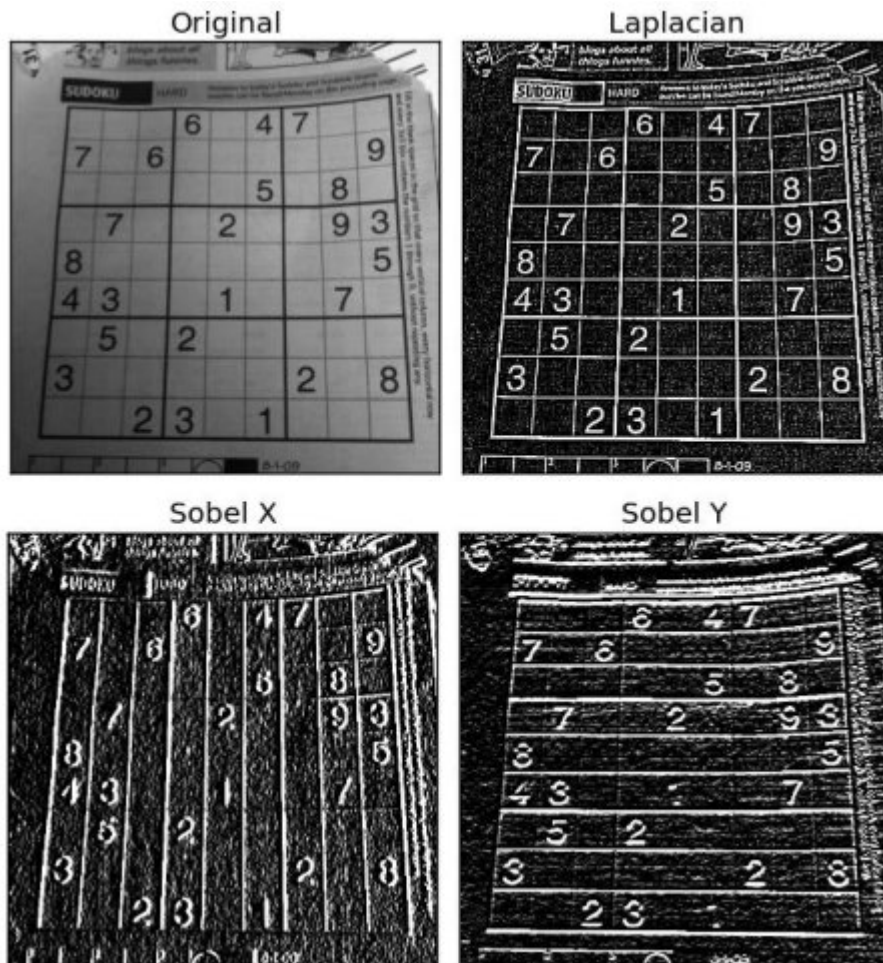
它计算了由关系 $\Delta \text{src} = \frac{\partial^2 \text{src}}{\partial x^2} + \frac{\partial^2 \text{src}}{\partial y^2}$ 给出的图像的拉普拉斯图,它是每一阶导数通过Sobel算子计算。如果 `ksize = 1`,然后使用以下内核用于过滤:

```
kernel = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}
```

代码

下面的代码显示了单个图表中的所有算子。所有内核都是 `5x5` 大小。输出图像的深度通过 `-1` 得到结果的 `np.uint8` 型。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('dave.jpg', 0)
laplacian = cv.Laplacian(img, cv.CV_64F)
sobelx = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=5)
sobely = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=5)
plt.subplot(2, 2, 1), plt.imshow(img, cmap = 'gray')
plt.title('Original'), plt.xticks([], plt.yticks([]))
plt.subplot(2, 2, 2), plt.imshow(laplacian, cmap = 'gray')
plt.title('Laplacian'), plt.xticks([], plt.yticks([]))
plt.subplot(2, 2, 3), plt.imshow(sobelx, cmap = 'gray')
plt.title('Sobel X'), plt.xticks([], plt.yticks([]))
plt.subplot(2, 2, 4), plt.imshow(sobely, cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([], plt.yticks([]))
plt.show()
```



结果:

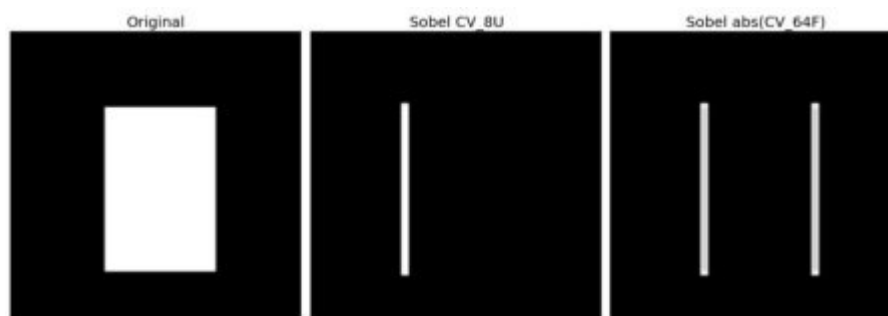
一个重要事项

在我们的最后一个示例中，输出数据类型为 `cv.CV_8U` 或 `np.uint8`。但这有一个小问题。黑色到白色的过渡被视为正斜率（具有正值），而白色到黑色的过渡被视为负斜率（具有负值）。因此，当您将数据转换为 `np.uint8` 时，所有负斜率均设为零。简而言之，您会错过这一边缘信息。

如果要检测两个边缘，更好的选择是将输出数据类型保留为更高的形式，例如 `cv.CV_16S`，`cv.CV_64F` 等，取其绝对值，然后转换回 `cv.CV_8U`。下面的代码演示了用于水平Sobel滤波器和结果差异的此过程。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('box.png', 0)
# Output dtype = cv.CV_8U
sobelx8u = cv.Sobel(img, cv.CV_8U, 1, 0, ksize=5)
# Output dtype = cv.CV_64F. Then take its absolute and convert to cv.CV_8U
sobelx64f = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=5)
```

```
abs_sobel64f = np.absolute(sobelx64f)
sobel_8u = np.uint8(abs_sobel64f)
plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2),plt.imshow(sobelx8u,cmap = 'gray')
plt.title('Sobel CV_8U'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3),plt.imshow(sobel_8u,cmap = 'gray')
plt.title('Sobel abs(CV_64F)'), plt.xticks([]), plt.yticks([])
plt.show()
```



查看以下结果：

附加资源

练习

Canny边缘检测

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，我们将学习 - Canny边缘检测的概念 - OpenCV函数: `cv.Canny()`

理论

Canny Edge Detection是一种流行的边缘检测算法。它由John F. Canny发明

1. 这是一个多阶段算法，我们将经历每个阶段。

2. 降噪

由于边缘检测容易受到图像中噪声的影响，因此第一步是使用5x5高斯滤波器消除图像中的噪声。我们已经在前面的章节中看到了这一点。

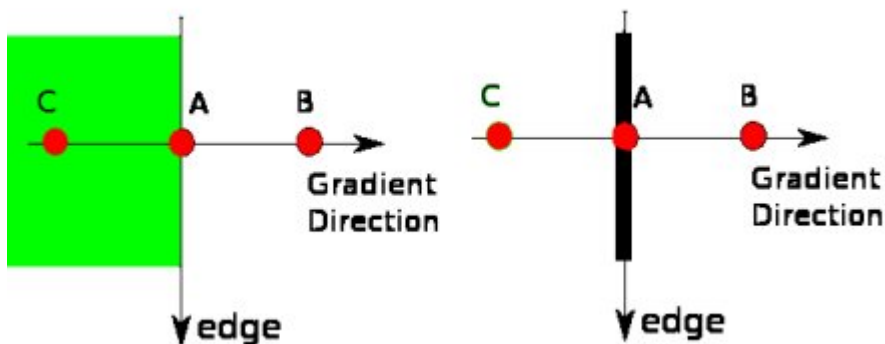
1. 查找图像的强度梯度

然后使用Sobel核在水平和垂直方向上对平滑的图像进行滤波，以在水平方向(G_x)和垂直方向(G_y)上获得一阶导数。从这两张图片中，我们可以找到每个像素的边缘渐变和方向，如下所示：

```
$$
Edge\_Gradient \; (G) = \sqrt{G_x^2 + G_y^2} \; \; Angle \; (\theta) = \tan^{-1}
\bigg(\frac{G_y}{G_x}\bigg)
$$
```

渐变方向始终垂直于边缘。将其舍入为代表垂直，水平和两个对角线方向的四个角度之一。

1. **非极大值抑制** 在获得梯度大小和方向后，将对图像进行全面扫描，以去除可能不构成边缘的所有不需要的像素。为此，在每个像素处，检查像素是否是其在梯度方向上附近的局部最大值。查看下面的图片：



点A在边缘（垂直方向）上。渐变方向垂直于边缘。点B和C在梯度方向上。因此，将A点与B点和C点进行检查，看是否形成局部最大值。如果是这样，则考虑将其用于下一阶段，否则将其抑制（置为零）。简而言之，你得到的结果是带有“细边”的二进制图像。

2. 磁滞阈值

该阶段确定哪些边缘全部是真正的边缘，哪些不是。为此，我们需要两个阈值 `minVal` 和 `maxVal`。强度梯度大于 `maxVal` 的任何边缘必定是边缘，而小于 `minVal` 的那些边缘必定是非边缘，因此将其丢弃。介于这两个阈值之间的对象根据其连通性被分类为边缘或非边缘。如果将它们连接到“边缘”像素，则将它们视为边缘的一部分。否则，它们也将被丢弃。见下图：

```

```

边缘A在 `maxVal` 之上，因此被视为“确定边缘”。尽管边C低于 `maxVal`，但它连接到边A，因此也被视为有效边，我们得到了完整的曲线。但是边缘B尽管在 `minVal` 之上并且与边缘C处于同一区域，但是它没有连接到任何“确保边缘”，因此被丢弃。因此，非常重要的一点是我们必须相应地选择 `minVal` 和 `maxVal` 以获得正确的结果。

在边缘为长线的假设下，该阶段还消除了小像素噪声。

因此，我们最终得到的是图像中的强边缘。

OpenCV中的Canny Edge检测

OpenCV将以上所有内容放在单个函数 `cv.Canny()` 中。我们将看到如何使用它。第一个参数是我们的输入图像。第二个和第三个参数分别是我们的 `minVal` 和 `maxVal`。第三个参数是 `perture_size`。它是用于查找图像渐变的Sobel内核的大小。默认情况下为3。最后一个参数是 `L2gradient`，它指定用于查找梯度幅度的方程式。如果为 `True`，则使用上面提到的更精确的公式，否则使用以下函数： $\text{Edge_Gradient}; (G) = |G_x| + |G_y|$ 。默认情况下，它为 `False`。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([], plt.yticks([]))
plt.show()
```

附加资源

1. Canny edge detector at Wikipedia : http://en.wikipedia.org/wiki/Canny_edge_detector
2. Canny Edge Detection Tutorial : http://dasl.unlv.edu/daslDrexel/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html by Bill Green, 2002.

练习

1. 编写一个小应用程序以找到Canny边缘检测，该检测的阈值可以使用两个跟踪栏进行更改。这样，你可以了解阈值的影响。

图像金字塔

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将学习图像金字塔 - 我们将使用图像金字塔创建一个新的水果“Orapple” - 我们将看到以下功能：**cv.pyrUp()**，**cv.pyrDown()**

理论

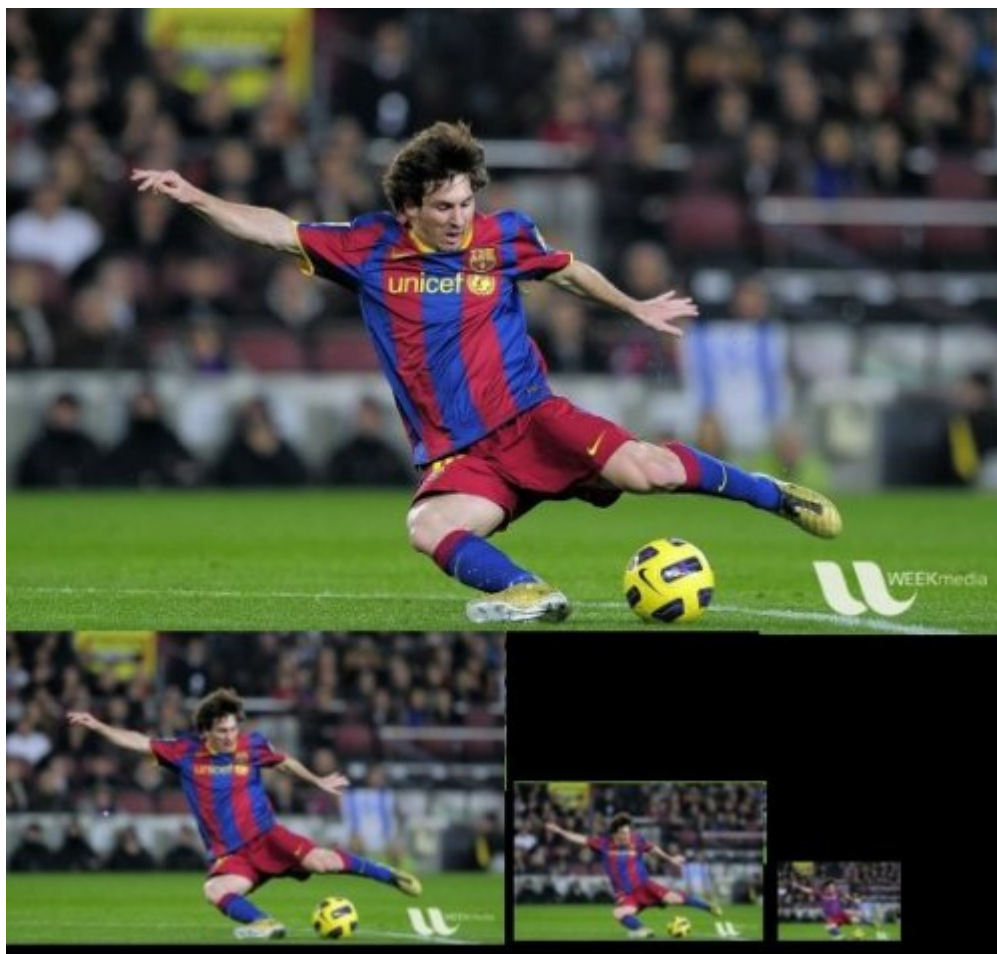
通常，我们过去使用的是恒定大小的图像。但是在某些情况下，我们需要使用不同分辨率的（相同）图像。例如，当在图像中搜索某些东西（例如人脸）时，我们不确定对象将以多大的尺寸显示在图像中。在这种情况下，我们将需要创建一组具有不同分辨率的相同图像，并在所有图像中搜索对象。这些具有不同分辨率的图像集称为“**图像金字塔**”（因为它们堆叠在底部时，最高分辨率的图像位于顶部，最低分辨率的图像位于顶部时，看起来像金字塔）。

有两种图像金字塔。1) **高斯金字塔**和2) **拉普拉斯金字塔**

高斯金字塔中的较高级别（低分辨率）是通过删除较低级别（较高分辨率）图像中的连续行和列而形成的。然后，较高级别的每个像素由基础级别的5个像素的贡献与高斯权重形成。通过这样做， $M \times N$ 图像变成 $M/2 \times N/2$ 图像。因此面积减少到原始面积的四分之一。它称为Octave。当我们在金字塔中越靠上时（即分辨率下降），这种模式就会继续。同样，在扩展时，每个级别的面积变为4倍。我们可以使用**cv.pyrDown()**和**cv.pyrUp()**函数找到高斯金字塔。

```
img = cv.imread('messi5.jpg')
lower_reso = cv.pyrDown(higher_reso)
```

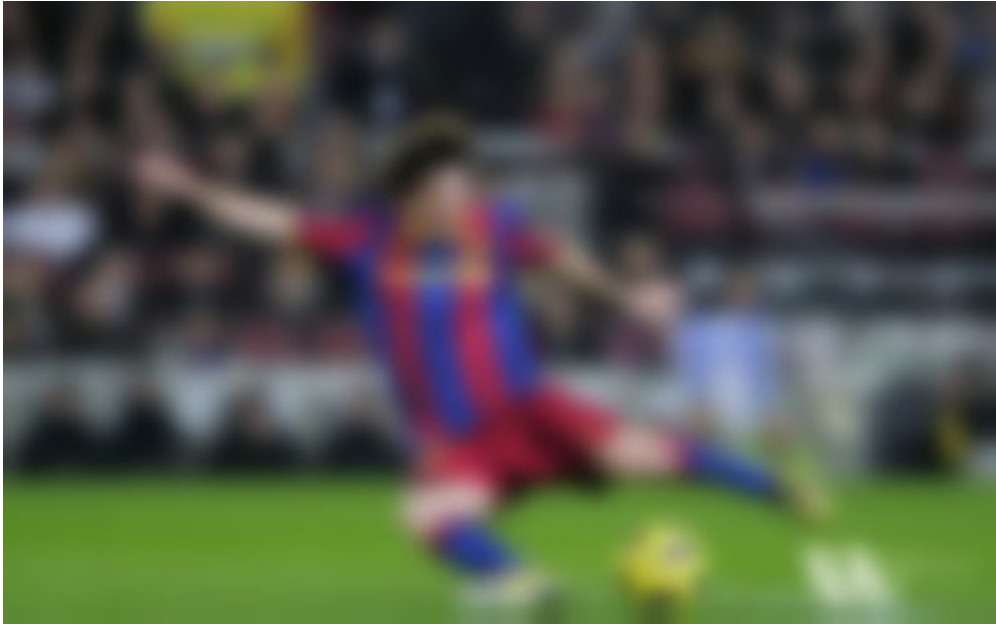
以下是图像金字塔中的4个级别。



现在，您可以使用**cv.pyrUp**()函数查看图像金字塔。

```
higher_reso2 = cv.pyrUp(lower_reso)
```

记住，higher_reso2不等于higher_reso，因为一旦降低了分辨率，就会丢失信息。下面的图像是3层的金字塔从最小的图像在前面的情况下创建。与原图对比:

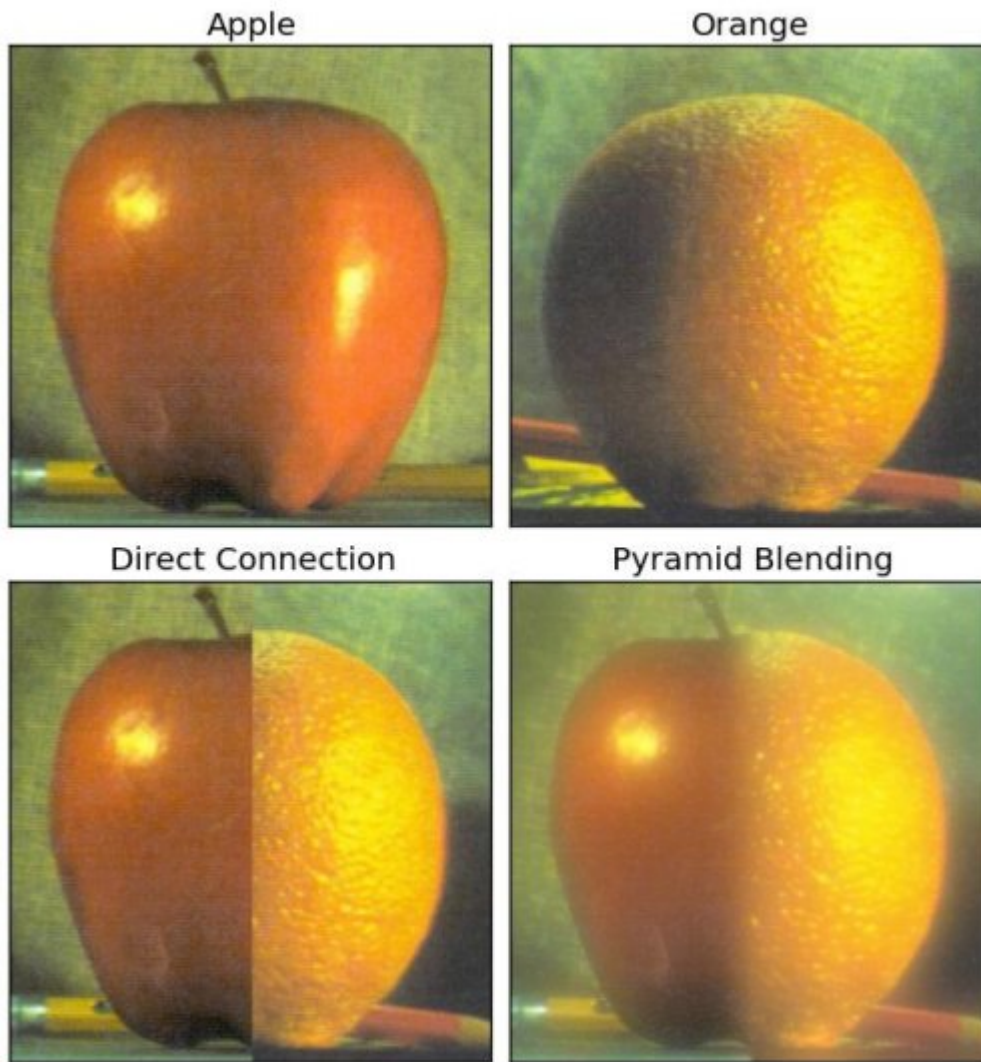


拉普拉斯金字塔由高斯金字塔形成。没有专用功能。拉普拉斯金字塔图像仅像边缘图像。它的大多数元素为零。它们用于图像压缩。拉普拉斯金字塔的层由高斯金字塔的层与高斯金字塔的高层的扩展版本之间的差形成。拉普拉斯等级的三个等级如下所示（调整对比度以增强内容）：



使用金字塔进行图像融合

金字塔的一种应用是图像融合。例如，在图像拼接中，您需要将两个图像堆叠在一起，但是由于图像之间的不连续性，可能看起来不太好。在这种情况下，使用金字塔混合图像可以无缝混合，而不会在图像中保留大量数据。一个经典的例子是将两种水果，橙和苹果混合在一起。现在查看结果本身，以了解我在说什么：



请检查其他资源中的第一个参考，它具有图像混合，拉普拉斯金字塔等的完整图解详细信息。只需完成以下步骤即可：

1. 加载苹果和橙子的两个图像
2. 查找苹果和橙子的高斯金字塔（在此示例中，级别数为6）
3. 在高斯金字塔中，找到其拉普拉斯金字塔
4. 然后在每个拉普拉斯金字塔级别中加入苹果的左半部分和橙子的右半部分
5. 最后从此联合图像金字塔中重建原始图像。

下面是完整的代码。（为简单起见，每个步骤都是单独进行的，这可能会占用更多的内存。如果需要，可以对其进行优化）。

```
import cv2 as cv
import numpy as np, sys
```

```
A = cv.imread('apple.jpg')
B = cv.imread('orange.jpg')
# 生成A的高斯金字塔
G = A.copy()
gpA = [G]
for i in xrange(6):
    G = cv.pyrDown(G)
    gpA.append(G)
# 生成B的高斯金字塔
G = B.copy()
gpB = [G]
for i in xrange(6):
    G = cv.pyrDown(G)
    gpB.append(G)
# 生成A的拉普拉斯金字塔
lpA = [gpA[5]]
for i in xrange(5, 0, -1):
    GE = cv.pyrUp(gpA[i])
    L = cv.subtract(gpA[i-1], GE)
    lpA.append(L)
# 生成B的拉普拉斯金字塔
lpB = [gpB[5]]
for i in xrange(5, 0, -1):
    GE = cv.pyrUp(gpB[i])
    L = cv.subtract(gpB[i-1], GE)
    lpB.append(L)
# 现在在每个级别中添加左右两半图像
LS = []
for la, lb in zip(lpA, lpB):
    rows, cols, dpt = la.shape
    ls = np.hstack((la[:, 0:cols/2], lb[:, cols/2:]))
    LS.append(ls)
# 现在重建
ls_ = LS[0]
for i in xrange(1, 6):
    ls_ = cv.pyrUp(ls_)
    ls_ = cv.add(ls_, LS[i])
# 图像与直接连接的每一半
real = np.hstack((A[:, 0:cols/2], B[:, cols/2:]))
cv.imwrite('Pyramid_blending2.jpg', ls_)
cv.imwrite('Direct_blending.jpg', real)
##
```

附加资源

1. Image Blending : http://pages.cs.wisc.edu/~csverma/CS766_09/ImageMosaic/imagemosaic.html

练习

轮廓：入门

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

- 了解轮廓是什么。
- 学习查找轮廓，绘制轮廓等。
- 你将看到以下功能：**cv.findContours()**，**cv.drawContours()**

什么是轮廓？

轮廓可以简单地解释为连接具有相同颜色或强度的所有连续点（沿边界）的曲线。轮廓是用于形状分析以及对象检测和识别的有用工具。

- 为了获得更高的准确性，请使用二进制图像。因此，在找到轮廓之前，请应用阈值或canny边缘检测。
- 从OpenCV 3.2开始，**findContours()**不再修改源图像。
- 在OpenCV中，找到轮廓就像从黑色背景中找到白色物体。因此请记住，要找到的对象应该是白色，背景应该是黑色。

让我们看看如何找到二进制图像的轮廓：

```
import numpy as np
import cv2 as cv
im = cv.imread('test.jpg')
imggray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imggray, 127, 255, 0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
```


findcontour()函数中有三个参数，第一个是源图像，第二个是轮廓检索模式，第三个是轮廓逼近方法。输出等高线和层次结构。轮廓是图像中所有轮廓的Python列表。每个单独的轮廓是一个(x,y)坐标的Numpy数组的边界点的对象。

注意 稍后我们将详细讨论第二和第三个参数以及有关层次结构。在此之前，代码示例中赋予它们的值将适用于所有图像。

如何绘制轮廓？

要绘制轮廓，请使用**cv.drawContours**函数。只要有边界点，它也可以用来绘制任何形状。它的第一个参数是源图像，第二个参数是应该作为Python列表传递的轮廓，第三个参数是轮廓的索引（在绘制单个轮廓时有用。要绘制所有轮廓，请传递-1），其余参数是颜色，厚度等等

- 在图像中绘制所有轮廓：

```
cv.drawContours(img, contours, -1, (0,255,0), 3)
```

- 绘制单个轮廓，如第四个轮廓：

```
cv.drawContours(img, contours, 3, (0,255,0), 3)
```

- 但是在大多数情况下，以下方法会很有用：

```
cnt = contours[4]
cv.drawContours(img, [cnt], 0, (0,255,0), 3)
```

注意 最后两种方法相似，但是前进时，您会发现最后一种更有用。

轮廓近似方法

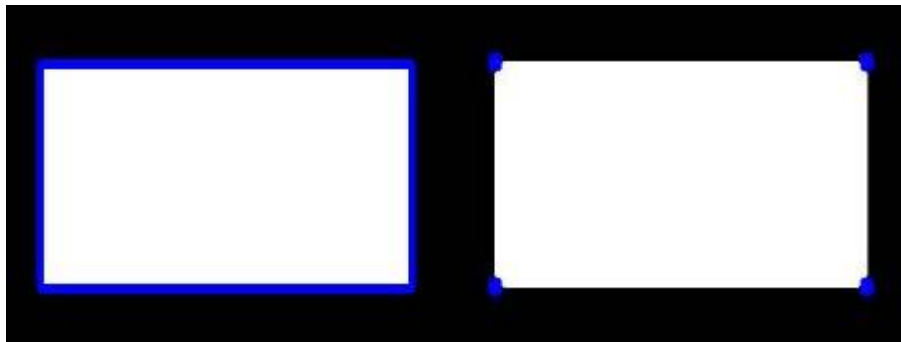
这是**cv.findContours**函数中的第三个参数。它实际上表示什么？

上面我们告诉我们轮廓是强度相同的形状的边界。它存储形状边界的(x,y)坐标。但是它存储所有坐标吗？这是通过这种轮廓近似方法指定的。

如果传递**cv.CHAIN_APPROX_NONE**，则将存储所有边界点。但是实际上我们需要所有这些要点吗？例如，您找到了一条直线的轮廓。您是否需要线上的所有点来代表该线？不，我们只需要

该线的两个端点即可。这就是**cv.CHAIN_APPROX_SIMPLE**所做的。它删除所有冗余点并压缩轮廓，从而节省内存。

下面的矩形图像演示了此技术。只需在轮廓数组中的所有坐标上绘制一个圆（以蓝色绘制）。第一幅图像显示了我用**cv.CHAIN_APPROX_NONE**获得的积分（734个点），第二幅图像显示了我用**cv.CHAIN_APPROX_SIMPLE**获得的效果（只有4个点）。看，它可以节省多少内存！！！！



附加资源

练习

轮廓特征

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本文中，我们将学习 - 如何找到轮廓的不同特征，例如面积，周长，质心，边界框等。 - 您将看到大量与轮廓有关的功能。

1. 特征矩

特征矩可以帮助您计算一些特征，例如物体的质心，物体的面积等。请查看特征矩上的维基百科页面。函数`cv.moments()`提供了所有计算出的矩值的字典。见下文：

```
import numpy as np
import cv2 as cv
img = cv.imread('star.jpg',0)
ret,thresh = cv.threshold(img,127,255,0)
contours,hierarchy = cv.findContours(thresh, 1, 2)
cnt = contours[0]
M = cv.moments(cnt)
print( M )
```

从这一刻起，您可以提取有用的数据，例如面积，质心等。质心由关系给出， $C_x = \frac{M_{10}}{M_{00}}$ 和 $C_y = \frac{M_{01}}{M_{00}}$ 。可以按照以下步骤进行：

```
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

2. 轮廓面积

轮廓区域由函数`cv.contourArea()`或从矩 `M['m00']` 中给出。

```
area = cv.contourArea(cnt)
```

3. 轮廓周长

也称为弧长。可以使用`cv.arcLength()`函数找到它。第二个参数指定形状是闭合轮廓(`True`)还是曲线。

```
perimeter = cv.arcLength(cnt, True)
```

4. 轮廓近似

根据我们指定的精度，它可以将轮廓形状近似为顶点数量较少的其他形状。它是Douglas-Peucker算法的实现。检查维基百科页面上的算法和演示。

为了理解这一点，假设您试图在图像中找到一个正方形，但是由于图像中的某些问题，您没有得到一个完美的正方形，而是一个“坏形状”（如下图所示）。现在，您可以使用此功能来近似形状。在这种情况下，第二个参数称为`epsilon`，它是从轮廓到近似轮廓的最大距离。它是一个精度参数。需要正确选择`epsilon`才能获得正确的输出。

```
epsilon = 0.1*cv.arcLength(cnt, True)
approx = cv.approxPolyDP(cnt, epsilon, True)
```

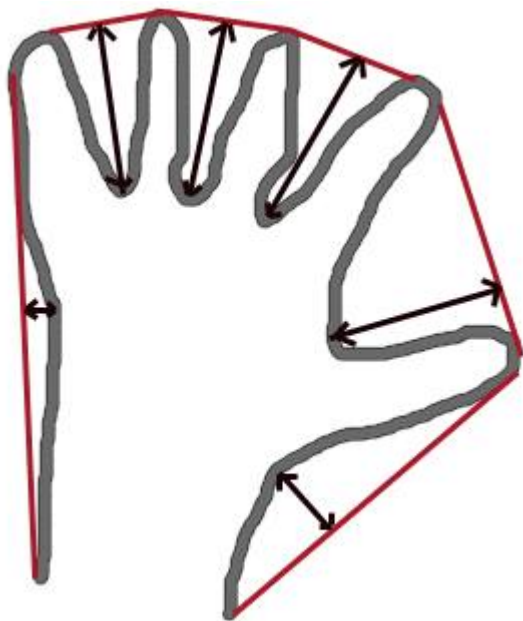
下面，在第二张图片中，绿线显示了 ϵ =弧长的10%时的近似曲线。第三幅图显示了 ϵ =弧长度的1%时的情况。第三个参数指定曲线是否闭合。



5. 轮廓凸包

凸包外观看起来与轮廓逼近相似，但不相似（在某些情况下两者可能提供相同的结果）。在这里，`cv.convexHull()`函数检查曲线是否存在凸凹缺陷并对其进行校正。一般而言，凸曲线是始终

凸出或至少平坦的曲线。如果在内部凸出，则称为凸度缺陷。例如，检查下面的手的图像。红线显示手的凸包。双向箭头标记显示凸度缺陷，这是凸包与轮廓线之间的局部最大偏差。



关于它的语法，有一些需要讨论：

```
hull = cv.convexHull(points[, hull[, clockwise[, returnPoints]]]
```

参数详细信息：- **点****是我们传递到的轮廓。- **凸包****是输出，通常我们忽略它。- **顺时针方向**：方向标记。如果为True，则输出凸包为顺时针方向。否则，其方向为逆时针方向。- **returnPoints**：默认情况下为True。然后返回凸包的坐标。如果为False，则返回与凸包点相对应的轮廓点的索引。

因此，要获得如上图所示的凸包，以下内容就足够了：

```
hull = cv.convexHull(cnt)
```

但是，如果要查找凸度缺陷，则需要传递 `returnPoints = False`。为了理解它，我们将拍摄上面的矩形图像。首先，我发现它的轮廓为 `cnt`。现在，我发现它的带有 `returnPoints = True` 的凸包，得到以下值： `[[[234 202]], [[51 202]], [[51 79]], [[234 79]]]`，它们是四个角矩形的点。现在，如果对 `returnPoints = False` 执行相同的操作，则会得到以下结果： `[[129], [67], [0], [142]]`。这些是轮廓中相应点的索引。例如，检查第一个值： `cnt [129] = [[234, 202]]` 与第一个结果相同（对于其他结果依此类推）。

当我们讨论凸度缺陷时，您将再次看到它。

6. 检查凸度

`cv.isContourConvex()`具有检查曲线是否凸出的功能。它只是返回True还是False。没什么大不了的。

```
k = cv.isContourConvex(cnt)
```

7. 边界矩形

有两种类型的边界矩形。

7.a. 直角矩形

它是一个矩形，不考虑物体的旋转。所以边界矩形的面积不是最小的。它是由函数`cv.boundingRect()`找到的。

令 (x, y) 为矩形的左上角坐标，而 (w, h) 为矩形的宽度和高度。

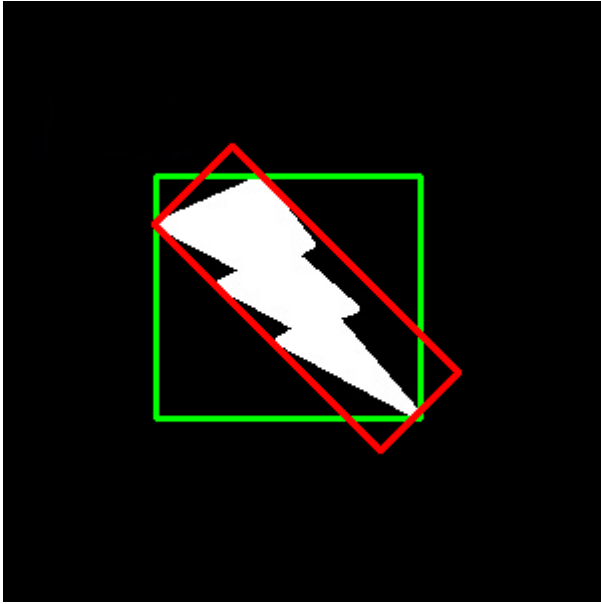
```
x, y, w, h = cv.boundingRect(cnt)
cv.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

7.b. 旋转矩形

这里，边界矩形是用最小面积绘制的，所以它也考虑了旋转。使用的函数是`cv.minAreaRect()`。它返回一个Box2D结构，其中包含以下细节 -(中心(x,y)，(宽度，高度)，旋转角度)。但要画出这个矩形，我们需要矩形的四个角。它由函数`cv.boxPoints()`获得

```
rect = cv.minAreaRect(cnt)
box = cv.boxPoints(rect)
box = np.int0(box)
cv.drawContours(img, [box], 0, (0, 0, 255), 2)
```

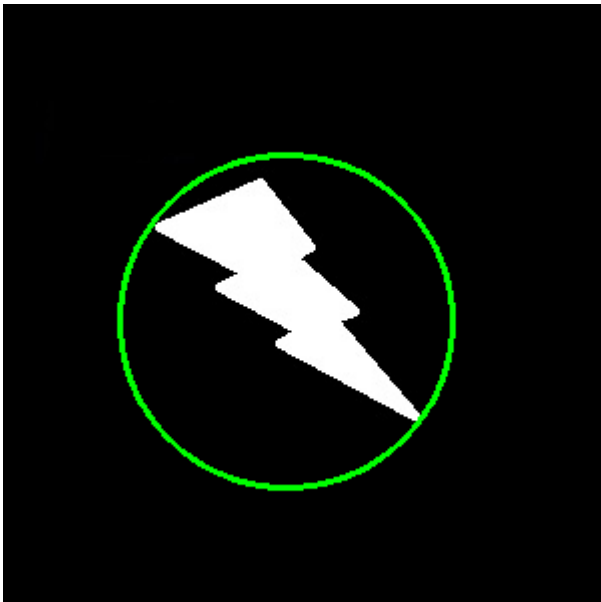
两个矩形都显示在一张单独的图像中。绿色矩形显示正常的边界矩形。红色矩形是旋转后的矩形。



8. 最小闭合圈

接下来，使用函数**`cv.minEnclosingCircle()`查找对象的圆周。它是一个以最小面积完全覆盖物体的圆。

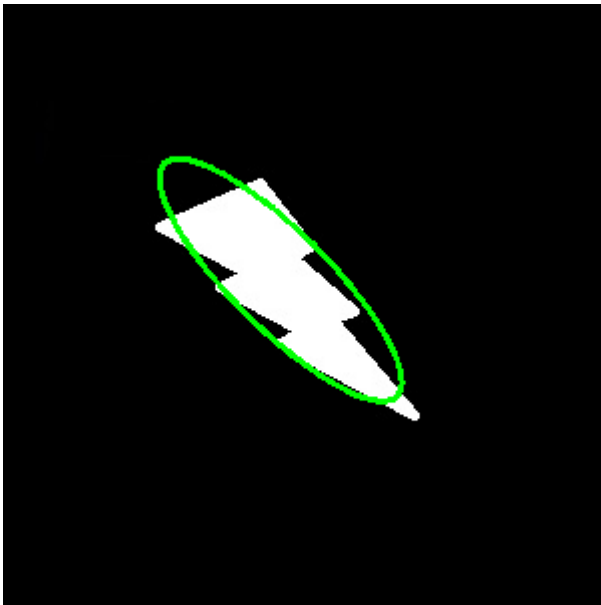
```
(x,y),radius = cv.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
cv.circle(img,center,radius,(0,255,0),2)
```



9. 拟合一个椭圆

下一个是把一个椭圆拟合到一个物体上。它返回内接椭圆的旋转矩形。

```
ellipse = cv.fitEllipse(cnt)
cv.ellipse(img, ellipse, (0, 255, 0), 2)
```



10. 拟合直线

同样，我们可以将一条直线拟合到一组点。下图包含一组白点。我们可以近似一条直线。

```
rows, cols = img.shape[:2]
[vx, vy, x, y] = cv.fitLine(cnt, cv.DIST_L2, 0, 0.01, 0.01)
lefty = int((-x*vy/vx) + y)
righty = int(((cols-x)*vy/vx)+y)
cv.line(img, (cols-1, righty), (0, lefty), (0, 255, 0), 2)
```

附加资源

练习

轮廓属性

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

在这里，我们将学习提取一些常用的物体属性，如坚实度，等效直径，掩模图像，平均强度等。更多的功能可以在Matlab regionprops文档中找到。

(注:质心、面积、周长等也属于这一类，但我们在上一章已经见过)

1. 长宽比

它是对象边界矩形的宽度与高度的比值。

$$\text{Aspect \; Ratio} = \frac{\text{Width}}{\text{Height}}$$

```
x,y,w,h = cv.boundingRect(cnt)
aspect_ratio = float(w)/h
```

2. 范围

范围是轮廓区域与边界矩形区域的比值。

$$\text{Extent} = \frac{\text{Object \; Area}}{\text{Bounding \; Rectangle \; Area}}$$

```
area = cv.contourArea(cnt)
x,y,w,h = cv.boundingRect(cnt)
rect_area = w*h
extent = float(area)/rect_area
```

3. 坚实度

坚实度是等高线面积与其凸包面积之比。

$$\text{Solidity} = \frac{\text{Contour \; Area}}{\text{Convex \; Hull \; Area}}$$

```
area = cv.contourArea(cnt)
hull = cv.convexHull(cnt)
hull_area = cv.contourArea(hull)
solidity = float(area)/hull_area
```

4. 等效直径

等效直径是面积与轮廓面积相同的圆的直径。

Equivalent \; Diameter = $\sqrt{\frac{4 \times \text{Contour \; Area}}{\pi}}$

```
area = cv.contourArea(cnt)
equi_diameter = np.sqrt(4*area/np.pi)
```

5. 取向

取向是物体指向的角度。以下方法还给出了主轴和副轴的长度。

```
(x,y),(MA,ma),angle = cv.fitEllipse(cnt)
```

6. 掩码和像素点

在某些情况下，我们可能需要构成该对象的所有点。可以按照以下步骤完成：

```
mask = np.zeros(imggray.shape,np.uint8)
cv.drawContours(mask,[cnt],0,255,-1)
pixelpoints = np.transpose(np.nonzero(mask))
#pixelpoints = cv.findNonZero(mask)
```

这里提供了两个方法，一个使用Numpy函数，另一个使用OpenCV函数(最后的注释行)。结果也是一样的，只是略有不同。Numpy给出的坐标是 (行、列) 格式，而OpenCV给出的坐标是 (x,y) 格式。所以基本上答案是可以互换的。注意， row = x, column = y。

7. 最大值，最小值和它们的位置

我们可以使用掩码图像找到这些参数。

```
min_val, max_val, min_loc, max_loc = cv.minMaxLoc(imgray,mask = mask)
```

8. 平均颜色或平均强度

在这里，我们可以找到对象的平均颜色。或者可以是灰度模式下物体的平均强度。我们再次使用相同的掩码进行此操作。

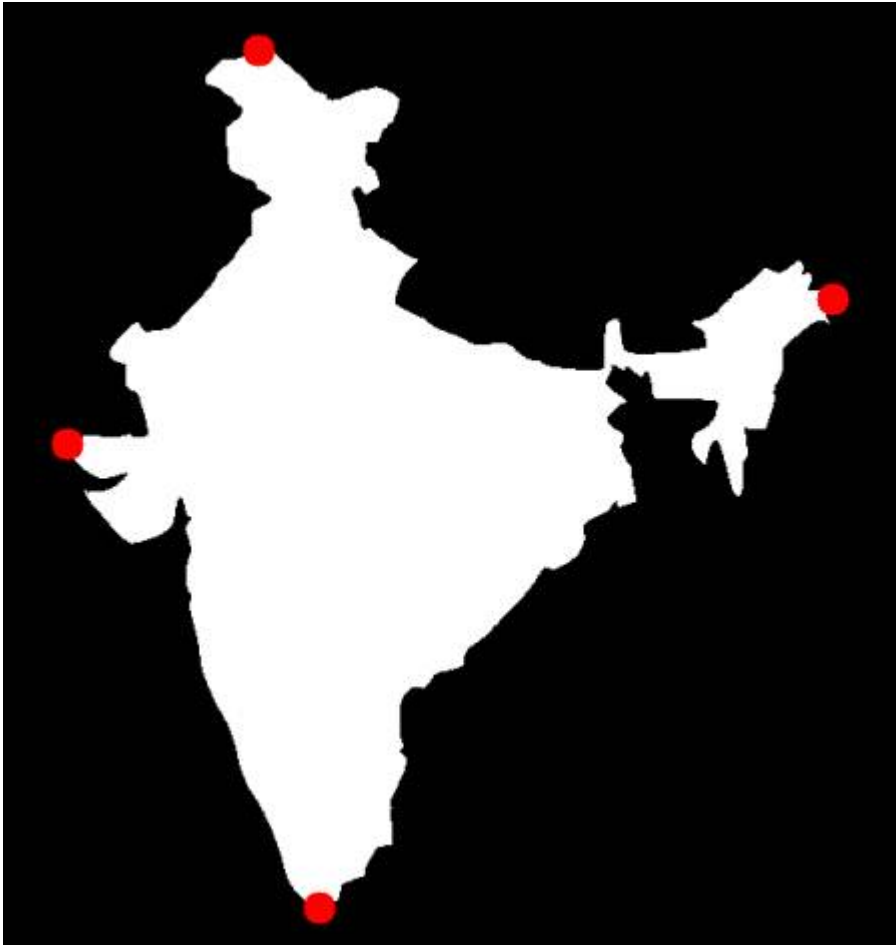
```
mean_val = cv.mean(im,mask = mask)
```

9. 极 endpoint

极点是指对象的最顶部，最底部，最右侧和最左侧的点。

```
leftmost = tuple(cnt[cnt[:, :, 0].argmin()][0])  
rightmost = tuple(cnt[cnt[:, :, 0].argmax()][0])  
topmost = tuple(cnt[cnt[:, :, 1].argmin()][0])  
bottommost = tuple(cnt[cnt[:, :, 1].argmax()][0])
```

例如，如果我将其应用于印度地图，则会得到以下结果：



附加资源

练习

1. matlab的regionprops doc中仍然有一些特性。试着去实现它们。

轮廓：更多属性

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，我们将学习 - 凸性缺陷以及如何找到它们 - 查找点到多边形的最短距离 - 匹配不同的形状

理论和代码

1. 凸性缺陷

我们看到了关于轮廓的第二章的凸包。从这个凸包上的任何偏差都可以被认为是凸性缺陷。OpenCV有一个函数来找到这个，**cv.convexityDefects()**。一个基本的函数调用如下：

```
hull = cv.convexHull(cnt, returnPoints = False)
defects = cv.convexityDefects(cnt, hull)
```

注意 记住,我们必须在发现凸包时,传递 `returnPoints= False` ,以找到凸性缺陷。

它返回一个数组，其中每行包含这些值—**[起点、终点、最远点、到最远点的近似距离]**。我们可以用图像把它形象化。我们画一条连接起点和终点的线，然后在最远处画一个圆。记住，返回的前三个值是cnt的索引。所以我们必须从cnt中获取这些值。

```
import cv2 as cv
import numpy as np
img = cv.imread('star.jpg')
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(img_gray, 127, 255, 0)
contours, hierarchy = cv.findContours(thresh, 2, 1)
cnt = contours[0]
hull = cv.convexHull(cnt, returnPoints = False)
defects = cv.convexityDefects(cnt, hull)
for i in range(defects.shape[0]):
```

```
s,e,f,d = defects[i,0]
start = tuple(cnt[s][0])
end = tuple(cnt[e][0])
far = tuple(cnt[f][0])
cv.line(img,start,end,[0,255,0],2)
cv.circle(img,far,5,[0,0,255],-1)
cv.imshow('img',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

查看结果：



2. 点多边形测试

这个函数找出图像中一点到轮廓线的最短距离。它返回的距离，点在轮廓线外时为负，点在轮廓线内时为正，点在轮廓线上时为零。

例如，我们可以检查点 (50,50) 如下：

```
dist = cv.pointPolygonTest(cnt,(50,50),True)
```

在函数中，第三个参数是measureDist。如果它是真的，它会找到有符号的距离。如果为假，则查找该点是在轮廓线内部还是外部(分别返回+1、-1和0)。

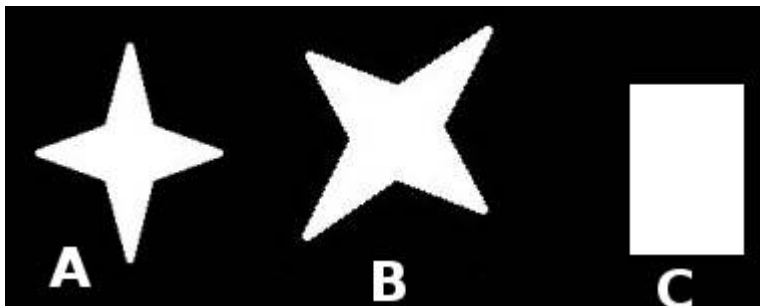
注意 如果您不想找到距离，请确保第三个参数为False，因为这是一个耗时的过程。因此，将其设置为False可使速度提高2-3倍。

3. 形状匹配

OpenCV附带一个函数**cv.matchShapes**()，该函数使我们能够比较两个形状或两个轮廓，并返回一个显示相似性的度量。结果越低，匹配越好。它是根据矩值计算出来的。不同的测量方法在文档中有解释。

```
import cv2 as cv
import numpy as np
img1 = cv.imread('star.jpg',0)
img2 = cv.imread('star2.jpg',0)
ret, thresh = cv.threshold(img1, 127, 255,0)
ret, thresh2 = cv.threshold(img2, 127, 255,0)
contours,hierarchy = cv.findContours(thresh,2,1)
cnt1 = contours[0]
contours,hierarchy = cv.findContours(thresh2,2,1)
cnt2 = contours[0]
ret = cv.matchShapes(cnt1,cnt2,1,0.0)
print( ret )
```

我尝试过匹配下面给出的不同形状的形状：



我得到以下结果: - 匹配的图像A与本身= 0.0 - 匹配图像A与图像B = 0.001946 - 匹配图像A与图像C = 0.326911

看,即使是图像旋转也不会对这个比较产生很大的影响。

参考 Hu矩是平移、旋转和比例不变的七个矩。第七个是无偏斜量。这些值可以使用 `**cv.HuMoments**()` 函数找到。

附加资源

练习

1. 检查文档中的 `**cv.pointPolygonTest**()`，您可以找到红色和蓝色的漂亮图像。它表示从所有像素到白色曲线的距离。曲线内的所有像素都是蓝色的，这取决于距离。外面的点也是红色的。轮廓边缘用白色标记。所以问题很简单。编写一个代码来创建这样的距离表示。
2. 使用 `**cv.matchShapes**()` 比较数字或字母的图像。(这是迈向OCR的简单一步)

轮廓分层

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

这次我们学习轮廓的层次，即轮廓中的父子关系。

理论

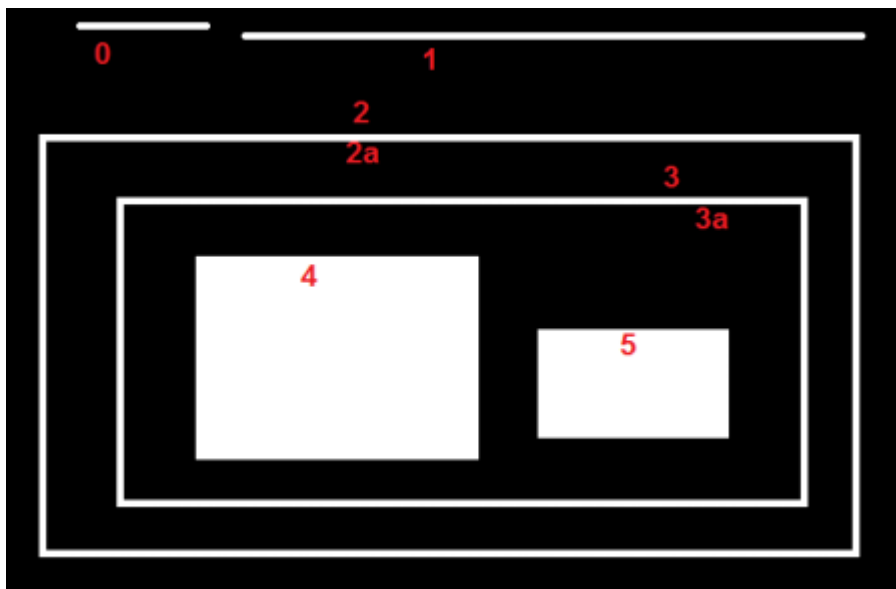
在前几篇关于轮廓的文章中，我们已经讨论了与OpenCV提供的轮廓相关的几个函数。但是当我们使用`cv.findcontour()`函数在图像中找到轮廓时，我们已经传递了一个参数，**轮廓检索模式**。我们通常通过了`cv.RETR_LIST`或`cv.RETR_TREE`，效果很好。但这到底意味着什么呢？

另外，在输出中，我们得到了三个数组，第一个是图像，第二个是轮廓，还有一个我们命名为`hierarchy`的输出(请检查前面文章中的代码)。但我们从未在任何地方使用过这种层次结构。那么这个层级是什么？它是用来做什么的？它与前面提到的函数参数有什么关系？

这就是我们在本文中要讨论的内容。

层次结构是什么？

通常我们使用`cv.findcontour()`函数来检测图像中的对象，对吧？有时对象在不同的位置。但在某些情况下，某些形状在其他形状中。就像嵌套的图形一样。在这种情况下，我们把外部的称为**父类**，把内部的称为**子类**。这样，图像中的轮廓就有了一定的相互关系。我们可以指定一个轮廓是如何相互连接的，比如，它是另一个轮廓的子轮廓，还是父轮廓等等。这种关系的表示称为**层次结构**。



下面是一个例子:

在这张图中，有一些形状我已经从**0-5**开始编号。***2***和***2a***表示最外层盒子的外部和内部轮廓。

这里，等高线0,1,2在****外部或最外面****。我们可以说，它们在****层级-0****中，或者简单地说，它们在****同一个层级****中。

其次是****contour-2a****。它可以被认为是****contour-2的子级****(或者反过来，contour-2是contour-2a的父级)。假设它在****层级-1****中。类似地，contour-3是contour-2的子级，它位于下一个层次结构中。最后，轮廓4,5是contour-3a的子级，他们在最后一个层级。从对方框的编号来看，我认为contour-4是contour-3a的第一个子级(它也可以是contour-5)。

我提到这些是为了理解一些术语，比如****相同层级****，**外部轮廓**，**子轮廓**，**父轮廓**，****第一个子轮廓****等等。现在让我们进入OpenCV。

OpenCV中的分级表示

所以每个轮廓都有它自己的信息关于它是什么层次，谁是它的孩子，谁是它的父母等等。OpenCV将它表示为一个包含四个值的数组: `[Next, Previous, First_Child, Parent]`

“Next表示同一层次的下一个轮廓。”

例如，在我们的图片中取contour-0。谁是下一个同级别的等高线?这是contour-1。简单地令 `Next = 1`。类似地，Contour-1也是contour-2。所以 `Next = 2`。contour-2呢?同一水平线上没有下一条等高线。简单地，让 `Next = -1`。contour-4呢?它与contour-5处于同一级别。它的下一条等高线是contour-5，所以 `next = 5`。

“*Previous*表示同一层次上的先前轮廓。”

和上面一样。contour-1之前的等值线为同级别的contour-0。类似地，contour-2也是contour-1。对于contour-0，没有前项，所以设为-1。

“*First_Child*表示它的第一个子轮廓。”

没有必要作任何解释。对于contour-2, child是contour-2a。从而得到contour-2a对应的指标值。contour-3a呢?它有两个孩子。但我们只关注第一个孩子。它是contour-4。那么 `First_Child = 4` 对contour-3a而言。

“*Parent*表示其父轮廓的索引。”

它与***First_Child*

注意 如果没有子元素或父元素，则该字段被视为-1

现在我们已经了解了OpenCV中使用的层次样式，我们可以借助上面给出的相同图像来检查OpenCV中的轮廓检索模式。一些标志如 `cv.RETR_LIST`, `cv.RETR_TREE`, `cv.RETR_CCMP`, **`cv.RETR_EXTERNAL`

轮廓检索模式

1. RETR_LIST

这是四个标志中最简单的一个(从解释的角度来看)。它只是检索所有的轮廓，但不创建任何亲子关系。在这个规则下，**父轮廓和子轮廓是平等的，他们只是轮廓**。他们都属于同一层级。

这里，第3和第4项总是-1。但是很明显，下一项和上一项都有对应的值。你自己检查一下就可以了。

下面是我得到的结果，每一行是对应轮廓的层次细节。例如，第一行对应于轮廓0。下一条轮廓是轮廓1。所以 `Next = 1`。没有先前的轮廓，所以 `Previous=-1`。剩下的两个，如前所述，是 `-1`。

```
>>> hierarchy
array([[[ 1, -1, -1, -1],
        [ 2,  0, -1, -1],
        [ 3,  1, -1, -1],
        [ 4,  2, -1, -1],
        [ 5,  3, -1, -1],
```

```
[ 6, 4, -1, -1],  
[ 7, 5, -1, -1],  
[-1, 6, -1, -1]]])
```

如果您没有使用任何层次结构特性，那么这是在您的代码中使用的最佳选择。

2. RETR_EXTERNAL

如果使用此标志，它只返回极端外部标志。所有孩子的轮廓都被留下了。**我们可以说，根据这项规则，每个家庭只有长子得到关注。它不关心家庭的其他成员:)**。

所以在我们的图像中，有多少个极端的外轮廓?在等级0级?有3个，即等值线是0 1 2，对吧?现在试着用这个标志找出等高线。这里，给每个元素的值与上面相同。并与上述结果进行了比较。以下是我得到的:

```
>>> hierarchy  
array([[[ 1, -1, -1, -1],  
         [ 2, 0, -1, -1],  
         [-1, 1, -1, -1]]])
```

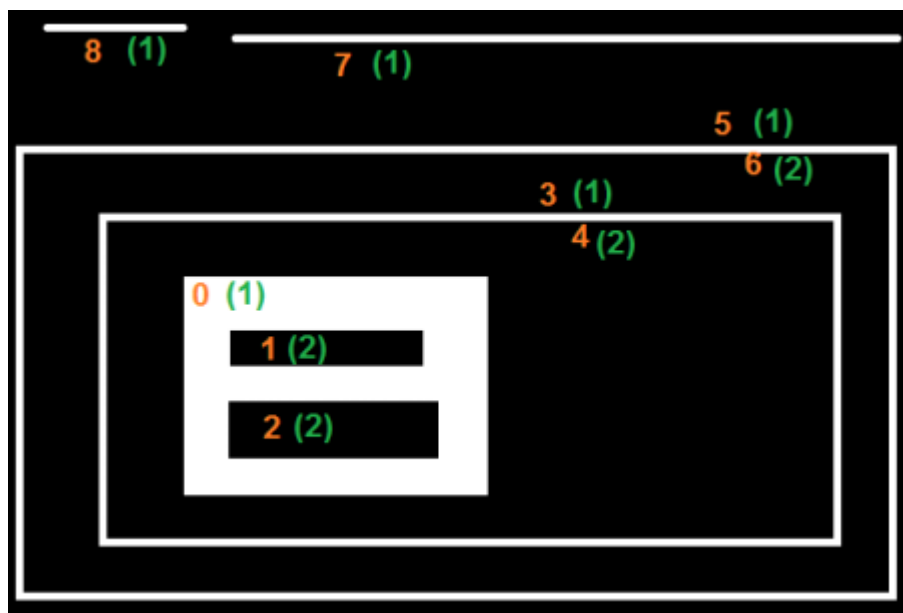
如果只想提取外部轮廓，可以使用此标志。它在某些情况下可能有用。

3. RETR_CCOMP

此标志检索所有轮廓并将其排列为2级层次结构。物体的外部轮廓(即物体的边界)放在层次结构-1中。对象内部孔洞的轮廓(如果有)放在层次结构-2中。如果其中有任何对象，则其轮廓仅在层次结构1中重新放置。以及它在层级2中的漏洞等等。

只需考虑在黑色背景上的“白色的零”图像。零的外圆属于第一级，零的内圆属于第二级。

我们可以用一个简单的图像来解释它。这里我用红色标注了等高线的顺序和它们所属的层次，用绿色标注(1或2)，顺序与OpenCV检测等高线的顺序相同。



考虑第一个轮廓，即contour-0。这是hierarchy-1。它有两个孔，分别是等高线1和2，属于第二级。因此，对于轮廓-0，在同一层次的下一个轮廓是轮廓-3。previous也没有。在hierarchy-2中，它的第一个子结点是contour-1。它没有父类，因为它在hierarchy-1中。所以它的层次数组是

```
[3, -1, 1, -1]
```

现在contour-1。它在层级-2中。相同层次结构中的下一个(在contour-1的父母关系下)是contour-2。没有previous。没有 child，但是 parent 是contour-0。所以数组是

```
[2, -1, -1, 0]
```

类似的contour-2:它在hierarchy-2中。在contour-0下，同一层次结构中没有下一个轮廓。所以没有 Next。previous 是contour-1。没有 child，parent 是contour0。所以数组是

```
[-1, 1, -1, 0]
```

contour-3:层次-1的下一个是轮廓-5。以前是contour-0。child 是contour4，没有 parent。所以数组是

```
[5, 0, 4, -1]
```

contour-4:它在contour-3下的层次结构2中，它没有兄弟姐妹。没有 next，没有 previous，没有 child，parent 是contour-3。所以数组是

```
[-1, -1, -1, 3]
```

剩下的你可以补充。这是我得到的最终答案:

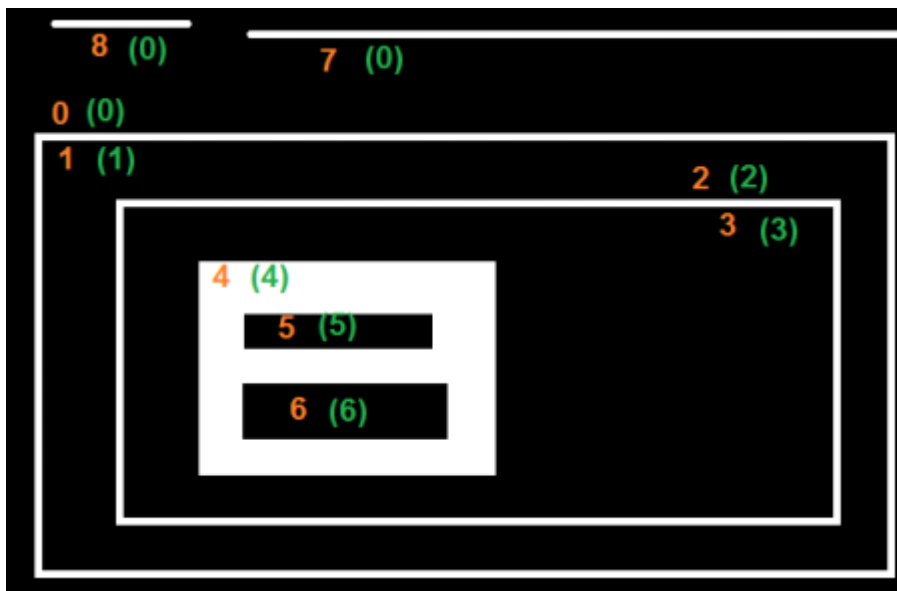
```
>>> hierarchy
array([[[ 3, -1,  1, -1],
        [ 2, -1, -1,  0],
        [-1,  1, -1,  0],
        [ 5,  0,  4, -1],
        [-1, -1, -1,  3],
        [ 7,  3,  6, -1],
        [-1, -1, -1,  5],
```

```
[ 8, 5, -1, -1],
[-1, 7, -1, -1]]])
```

4. RETR_TREE

这是最后一个家伙，完美先生。它检索所有的轮廓并创建一个完整的家族层次结构列表。它甚至告诉，谁是爷爷，父亲，儿子，孙子，甚至更多...:)。

例如，我拿上面的图片，重写了cv的代码。RETR_TREE，根据OpenCV给出的结果重新排序等高线并进行分析。同样，红色的字母表示轮廓数，绿色的字母表示层次顺序。



取 `contour-0` :它在 `hierarchy-0` 中。同一层次结构的 `next` 轮廓是轮廓-7。没有 `previous` 的轮廓。 `child` 是 `contour-1`，没有 `parent`。所以数组是 `[7, -1, 1, -1]`

以 `contour-2` 为例:它在 `hierarchy-1` 中。没有轮廓在同一水平。没有 `previous`。 `child` 是 `contour-3`。父母是 `contour-1`。所以数组是 `[-1, -1, 3, 1]`

剩下的，你自己试试。以下是完整答案:

```
>>> hierarchy
array([[[ 7, -1, 1, -1],
        [-1, -1, 2, 0],
        [-1, -1, 3, 1],
        [-1, -1, 4, 2],
        [-1, -1, 5, 3],
        [ 6, -1, -1, 4],
        [-1, 5, -1, 4],
```

```
[ 8,  0, -1, -1],  
[-1,  7, -1, -1]]])
```

附加资源

练习

直方图-1：查找、绘制和分析

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

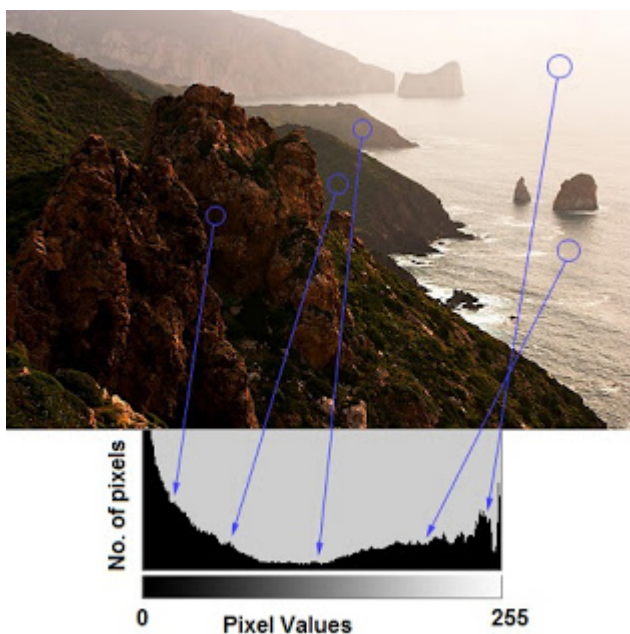
目标

学会 - 使用OpenCV和Numpy函数查找直方图 - 使用OpenCV和Matplotlib函数绘制直方图 - 你将看到以下函数：**cv.calcHist()**，**np.histogram()**等。

理论

那么直方图是什么？您可以将直方图视为图形或绘图，从而可以总体了解图像的强度分布。它是在X轴上具有像素值（不总是从0到255的范围），在Y轴上具有图像中相应像素数的图。

这只是理解图像的另一种方式。通过查看图像的直方图，您可以直观地了解该图像的对比度，亮度，强度分布等。当今几乎所有图像处理工具都提供直方图功能。以下是剑桥彩色网站的图片，我建议您访问该网站以获取更多详细信息。



您可以看到图像及其直方图。（请记住，此直方图是针对灰度图像而非彩色图像绘制的）。直方图的左侧区域显示图像中较暗像素的数量，而右侧区域则显示明亮像素的数量。从直方图中，您可以看到暗区域多于亮区域，而中间调的数量（中间值的像素值，例如127附近）则非常少。

寻找直方图

现在我们有了一个关于直方图的想法，我们可以研究如何找到它。OpenCV和Numpy都为此内置了功能。在使用这些功能之前，我们需要了解一些与直方图有关的术语。

BINS：上面的直方图显示每个像素值的像素数，即从0到255。即，您需要256个值来显示上面的直方图。但是考虑一下，如果您不需要分别找到所有像素值的像素数，而是找到像素值间隔中的像素数怎么办？例如，您需要找到介于0到15之间的像素数，然后找到16到31之间，...，240到255之间的像素数。只需要16个值即可表示直方图。这就是在OpenCV教程中有关直方图的示例中显示的内容。

因此，您要做的就是将整个直方图分成16个子部分，每个子部分的值就是其中所有像素数的总和。每个子部分都称为“BIN”。在第一种情况下，bin的数量为256个（每个像素一个），而在第二种情况下，bin的数量仅为16个。BINS由OpenCV文档中的**histSize**术语表示。

DIMS：这是我们为其收集数据的参数的数量。在这种情况下，我们仅收集关于强度值的一件事的数据。所以这里是1。

RANGE：这是您要测量的强度值的范围。通常，它是 `[0,256]`，即所有强度值。

1. OpenCV中的直方图计算

因此，现在我们使用**cv.calcHist**()函数查找直方图。让我们熟悉一下该函数及其参数：

```
cv.calcHist ( images , channels , mask , histSize , ranges [ , hist [ , accumulate]] )
```

1. images：它是uint8或float32类型的源图像。它应该放在方括号中，即“[img]”。
2. channels：也以方括号给出。它是我们计算直方图的通道的索引。例如，如果输入为灰度图像，则其值为[0]。对于彩色图像，您可以传递[0]，[1]或[2]分别计算蓝色，绿色或红色通道的直方图。
3. mask：图像掩码。为了找到完整图像的直方图，将其指定为“无”。但是，如果要查找图像特定区域的直方图，则必须为此创建一个掩码图像并将其作为掩码。（我将在后面显示一个示例。）

4. histSize : 这表示我们的BIN计数。需要放在方括号中。对于全尺寸, 我们通过[256]。

5. ranges : 这是我们的RANGE。通常为[0,256]。

因此, 让我们从示例图像开始。只需以灰度模式加载图像并找到其完整直方图即可。

```
img = cv.imread('home.jpg',0)
hist = cv.calcHist([img],[0],None,[256],[0,256])
```

hist是256x1的数组, 每个值对应于该图像中具有相应像素值的像素数。

2. numpy的直方图计算

Numpy还为您提供了一个函数**np.histogram**()。因此, 除了**calcHist**()函数外, 您可以尝试下面的代码:

```
hist,bins = np.histogram(img.ravel(),256,[0,256])
```

hist与我们之前计算的相同。但是bin将具有257个元素, 因为Numpy计算出bin的范围为 0-0.99、1-1.99、2-2.99 等。因此最终范围为 255-255.99。为了表示这一点, 他们还在最后添加了256。但是我们不需要256。最多255就足够了。

- 另外 Numpy还有另一个函数**np.bincount**(), 它比np.histogram()快10倍左右。因此, 对于一维直方图, 您可以更好地尝试一下。不要忘记在np.bincount中设置minlength = 256。例如, `hist = np.bincount(img.ravel(), minlength = 256)`

注意 OpenCV函数比np.histogram()快大约40倍。因此, 尽可能使用OpenCV函数。

现在我们应该绘制直方图, 但是怎么绘制?

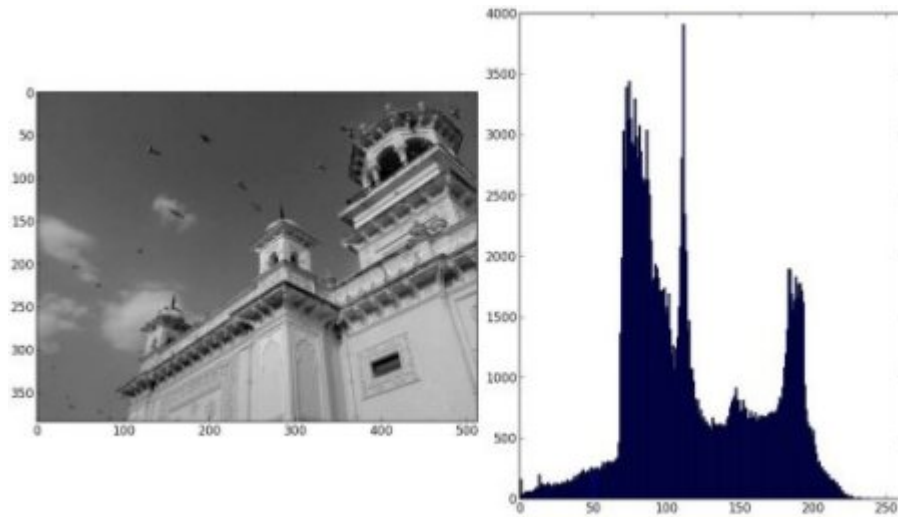
绘制直方图

有两种方法, 1. 简短的方法: 使用Matplotlib绘图功能 2. 稍长的方法: 使用OpenCV绘图功能

1. 使用Matplotlib

Matplotlib带有直方图绘图功能: `matplotlib.pyplot.hist()` 它直接找到直方图并将其绘制。您无需使用**calcHist**()或np.histogram()函数来查找直方图。请参见下面的代码:

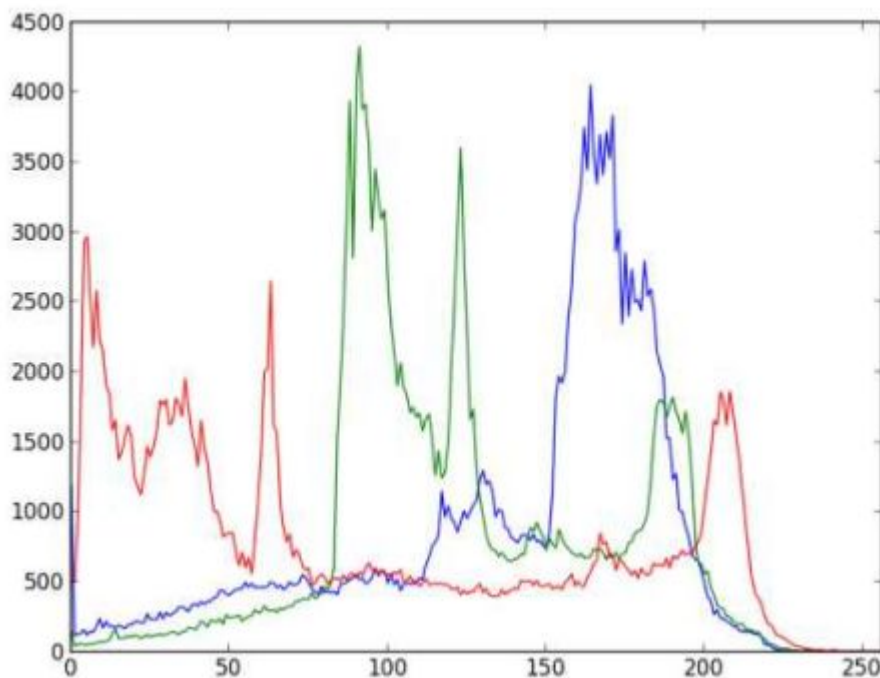
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg',0)
plt.hist(img.ravel(),256,[0,256]); plt.show()
```



你将得到如下的结果：

或者，您可以使用matplotlib的法线图，这对于BGR图是很好的。为此，您需要首先找到直方图数据。试试下面的代码：

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg')
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```



结果：

您可以从上图中得出，蓝色在图像中具有一些高值域（显然这应该是由于天空）

2. 使用 OpenCV

好吧，在这里您可以调整直方图的值及其bin值，使其看起来像x, y坐标，以便您可以使用 `**cv.line**()` 或 `cv.polyline()` 函数绘制它以生成与上述相同的图像。OpenCV-Python2官方示例已经提供了此功能。检查示例/python/hist.py中的代码。

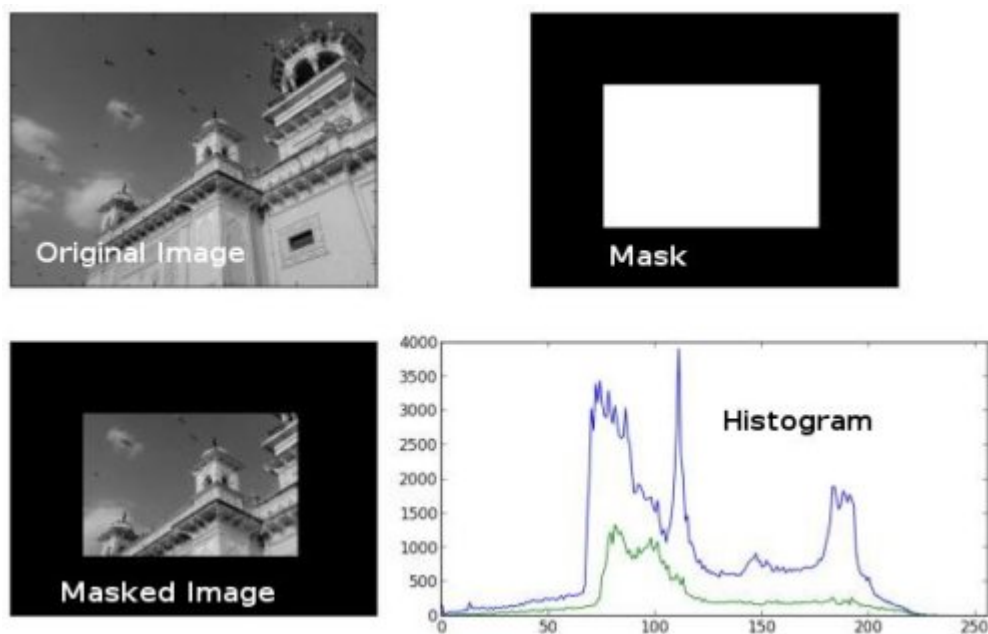
掩码的应用

我们使用了 `cv.calcHist()` 来查找整个图像的直方图。如果你想找到图像某些区域的直方图呢？只需创建一个掩码图像，在你要找到直方图为白色，否则黑色。然后把这个作为掩码传递。

```
img = cv.imread('home.jpg',0)
# create a mask
mask = np.zeros(img.shape[:2], np.uint8)
mask[100:300, 100:400] = 255
masked_img = cv.bitwise_and(img,img,mask = mask)
# 计算掩码区域和非掩码区域的直方图
# 检查作为掩码的第三个参数
hist_full = cv.calcHist([img],[0],None,[256],[0,256])
hist_mask = cv.calcHist([img],[0],mask,[256],[0,256])
plt.subplot(221), plt.imshow(img, 'gray')
plt.subplot(222), plt.imshow(mask, 'gray')
plt.subplot(223), plt.imshow(masked_img, 'gray')
```

```
plt.subplot(224), plt.plot(hist_full), plt.plot(hist_mask)  
plt.xlim([0,256])  
plt.show()
```

查看结果。在直方图中，蓝线表示完整图像的直方图，绿线表示掩码区域的直方图。



附加资源

1. Cambridge in Color website : <http://www.cambridgeincolour.com/tutorials/histograms1.htm>

练习

直方图-2：直方图均衡

作者|OpenCV-Python Tutorials

编译|Vincent

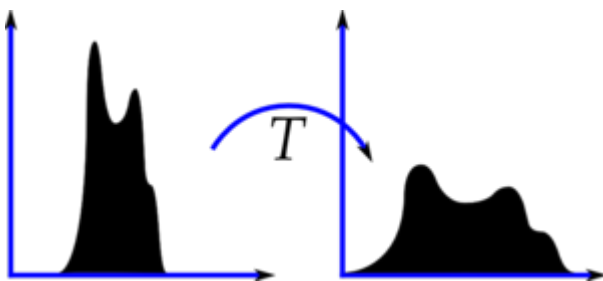
来源|OpenCV-Python Tutorials

目标

在本节中, - 我们将学习直方图均衡化的概念,并利用它来提高图像的对比度。

理论

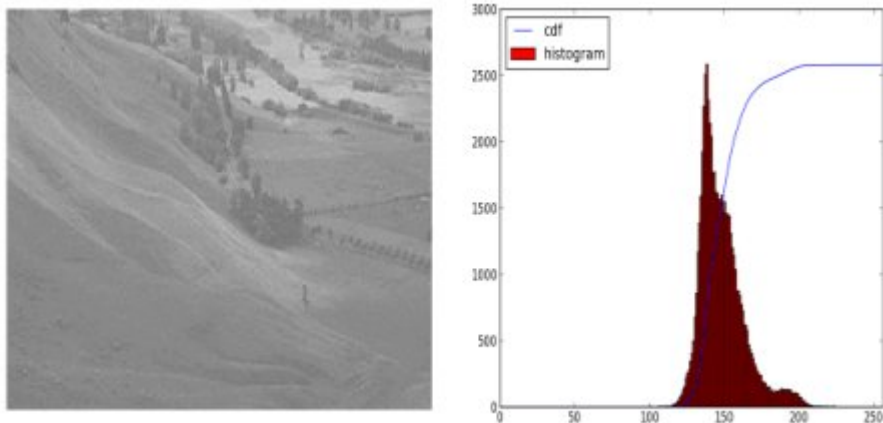
考虑这样一个图像，它的像素值仅局限于某个特定的值范围。例如，较亮的图像将把所有像素限制在高值上。但是一幅好的图像会有来自图像所有区域的像素。因此，您需要将这个直方图拉伸到两端(如下图所示，来自wikipedia)，这就是直方图均衡化的作用(简单来说)。这通常会提高图像的对比度。



我建议您阅读直方图均衡化上的Wikipedia页面，以获取有关它的更多详细信息。它很好地解释了示例，使您在阅读完之后几乎可以理解所有内容。相反，在这里我们将看到其Numpy实现。之后，我们将看到OpenCV功能。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('wiki.jpg',0)
hist,bins = np.histogram(img.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
```

```
plt.xlim([0,256])  
plt.legend(('cdf','histogram'), loc = 'upper left')  
plt.show()
```



你可以看到直方图位于较亮的区域。我们需要全频谱。为此，我们需要一个转换函数，将亮区域的输入像素映射到整个区域的输出像素。这就是直方图均衡化的作用。

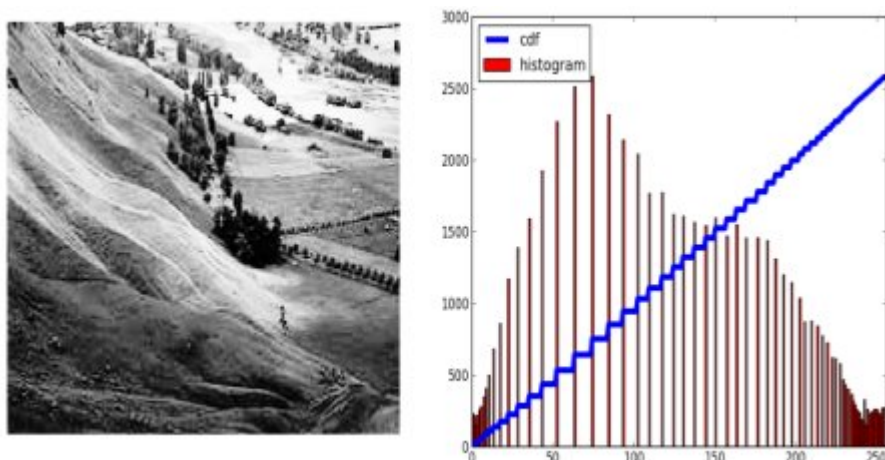
现在我们找到最小的直方图值(不包括0)，并应用wiki页面中给出的直方图均衡化方程。但我在这里用过，来自Numpy的掩码数组概念数组。对于掩码数组，所有操作都在非掩码元素上执行。您可以从Numpy文档中了解更多关于掩码数组的信息。

```
cdf_m = np.ma.masked_equal(cdf,0)  
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())  
cdf = np.ma.filled(cdf_m,0).astype('uint8')
```

现在我们有了查找表，该表为我们提供了有关每个输入像素值的输出像素值是什么的信息。因此，我们仅应用变换。

```
img2 = cdf[img]
```

现在，我们像以前一样计算其直方图和cdf（您这样做），结果如下所示：



另一个重要的特征是，即使图像是一个较暗的图像(而不是我们使用的一个较亮的图像)，经过均衡后，我们将得到几乎相同的图像。因此，这是作为一个“参考工具”，使所有的图像具有相同的照明条件。这在很多情况下都很有用。例如，在人脸识别中，在对人脸数据进行训练之前，对人脸图像进行直方图均衡化处理，使其具有相同的光照条件。

OpenCV中的直方图均衡

OpenCV具有执行此操作的功能`cv.equalizeHist()`。它的输入只是灰度图像，输出是我们的直方图均衡图像。下面是一个简单的代码片段，显示了它与我们使用的同一图像的用法：

```
img = cv.imread('wiki.jpg',0)
equ = cv.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side
cv.imwrite('res.png',res)
```

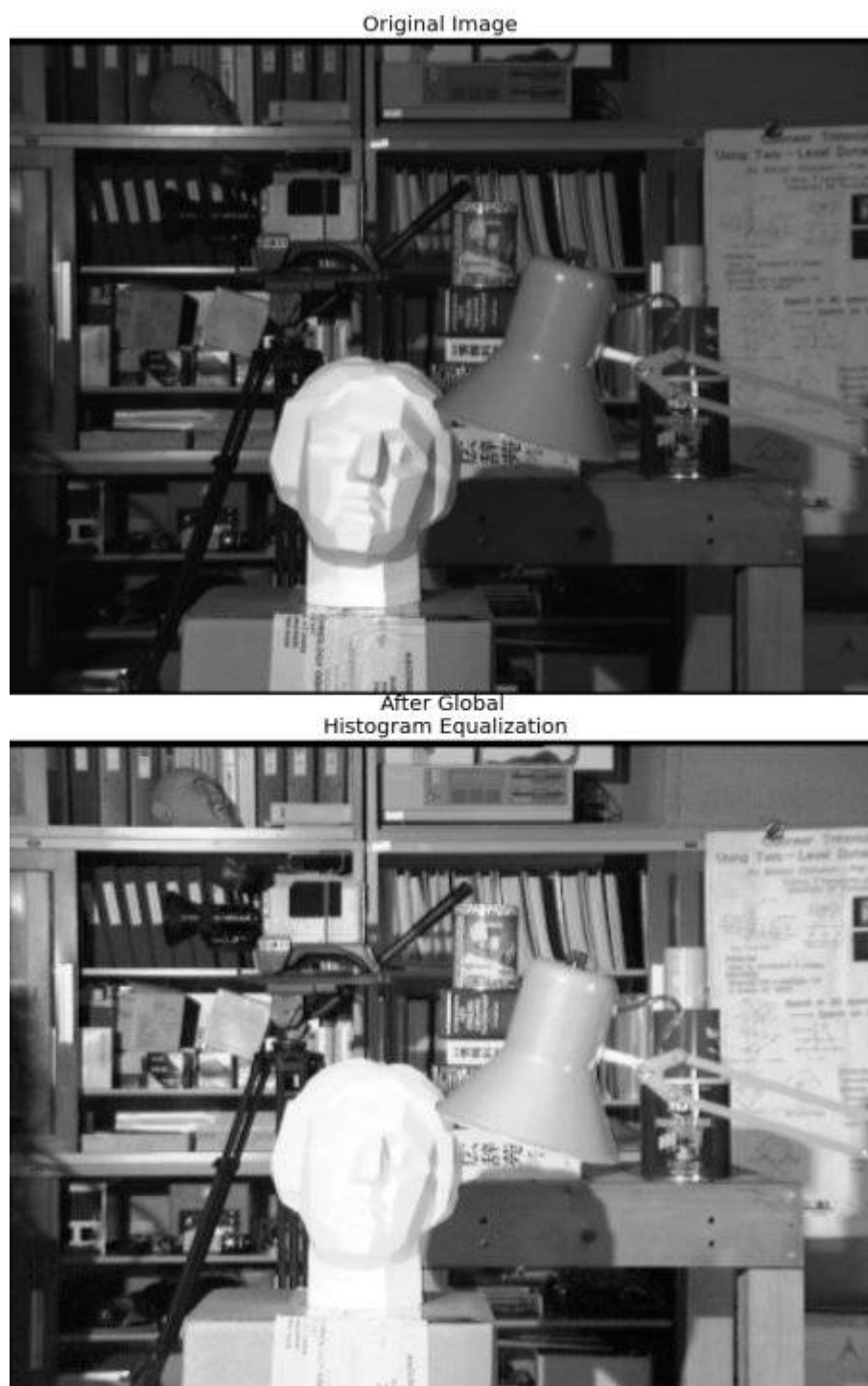


因此，现在您可以在不同的光照条件下拍摄不同的图像，对其进行均衡并检查结果。

当图像的直方图限制在特定区域时，直方图均衡化效果很好。在直方图覆盖较大区域（即同时存在亮像素和暗像素）的强度变化较大的地方，效果不好。请检查其他资源中的SOF链接。

CLAHE (对比度受限的自适应直方图均衡)

我们刚刚看到的第一个直方图均衡化考虑了图像的整体对比度。在许多情况下，这不是一个好主意。例如，下图显示了输入图像及其在全局直方图均衡后的结果。



直方图均衡后，背景对比度确实得到了改善。但是在两个图像中比较雕像的脸。由于亮度过高，我们在那里丢失了大多数信息。这是因为它的直方图不像我们在前面的案例中所看到的那样局限于特定区域（尝试绘制输入图像的直方图，您将获得更多的直觉）。

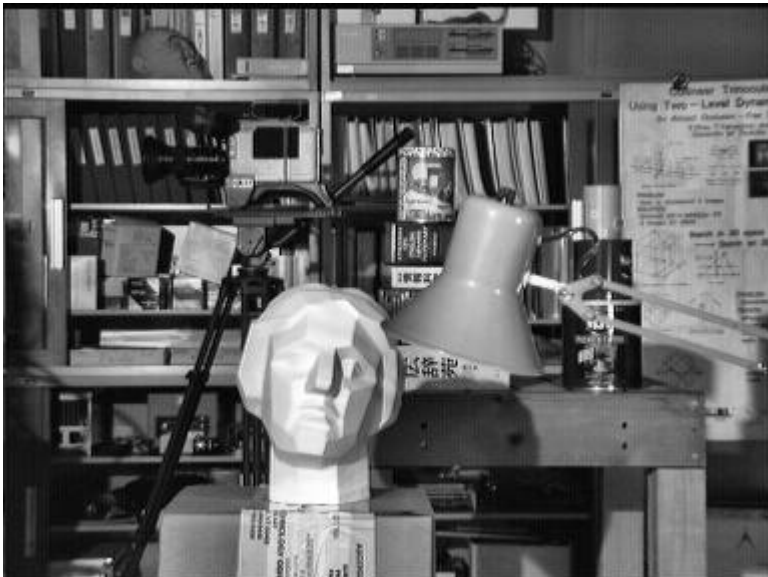
因此，为了解决这个问题，使用了**自适应直方图均衡**。在这种情况下，图像被分成称为“tiles”的小块（在OpenCV中，tileSize默认为 8×8 ）。然后，像往常一样对这些块中的每一个进行直方图均衡。因此，在较小的区域中，直方图将限制在一个较小的区域中（除非存在噪声）。

如果有噪音，它将被放大。为了避免这种情况，应用了对比度限制。如果任何直方图bin超出指定的对比度限制（在OpenCV中默认为40），则在应用直方图均衡之前，将这些像素裁剪并均匀地分布到其他bin。均衡后，要消除图块边界中的伪影，请应用双线性插值。

下面的代码片段显示了如何在OpenCV中应用CLAHE：

```
import numpy as np
import cv2 as cv
img = cv.imread('tsukuba_1.png',0)
# create a CLAHE object (Arguments are optional).
clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
c11 = clahe.apply(img)
cv.imwrite('clahe_2.jpg',c11)
```

查看下面的结果，并将其与上面的结果进行比较，尤其是雕像区域：



附加资源

1. Wikipedia page on Histogram Equalization : http://en.wikipedia.org/wiki/Histogram_equalization
2. Masked Arrays in Numpy : <http://docs.scipy.org/doc/numpy/reference/maskedarray.html>

有关对比度调整的问题：

1. 如何在C中的OpenCV中调整对比度？ <http://stackoverflow.com/questions/10549245/how-can-i-adjust-contrast-in-opencv-in-c>

2. 如何使用opencv均衡图像的对比度和亮度？<http://stackoverflow.com/questions/10561222/how-do-i-equalize-contrast-brightness-of-images-using-opencv>)

练习

直方图-3： 二维直方图

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，我们将学习查找和绘制2D直方图。这将在以后的章节中有所帮助。

介绍

在第一篇文章中，我们计算并绘制了一维直方图。之所以称为一维，是因为我们仅考虑一个特征，即像素的灰度强度值。但是在二维直方图中，您要考虑两个特征。通常，它用于查找颜色直方图，其中两个特征是每个像素的色相和饱和度值。

已经有一个python示例 (`samples / python / color_histogram.py`) 用于查找颜色直方图。我们将尝试了解如何创建这种颜色直方图，这对于理解诸如直方图反向投影之类的更多主题将很有用。

OpenCV中的二维直方图

它非常简单，并且使用相同的函数`cv.calcHist()`进行计算。对于颜色直方图，我们需要将图像从BGR转换为HSV。（请记住，对于一维直方图，我们从BGR转换为灰度）。对于二维直方图，其参数将进行如下修改：

- **channel = [0,1]**，因为我们需要同时处理H和S平面。
- **bins = [180,256]** 对于H平面为180，对于S平面为256。
- **range = [0,180,0,256]** 色相值介于0和180之间，饱和度介于0和256之间。

现在检查以下代码：

```
import numpy as np
import cv2 as cv
img = cv.imread('home.jpg')
```

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
hist = cv.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])
```

就是这样。

Numpy中的二维直方图

Numpy还为此提供了一个特定的函数:`np.histogram2d()`。(记住, 对于一维直方图我们使用了`np.histogram()`。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg')
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
hist, xbins, ybins = np.histogram2d(h.ravel(), s.ravel(), [180, 256], [[0, 180], [0, 256]])
```

第一个参数是H平面, 第二个是S平面, 第三个是每个箱子的数量, 第四个是它们的范围。

现在我们可以检查如何绘制这个颜色直方图。

绘制二维直方图

方法1 : 使用 `cv.imshow()`

我们得到的结果是尺寸为 `80x256` 的二维数组。因此, 可以使用`cv.imshow()`函数像平常一样显示它们。它将是一幅灰度图像, 除非您知道不同颜色的色相值, 否则不会对其中的颜色有太多了解。

方法2 : 使用Matplotlib

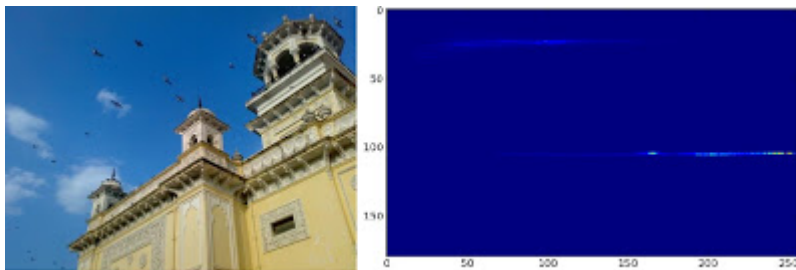
我们可以使用`matplotlib.pyplot.imshow()`函数绘制具有不同颜色图的2D直方图。它使我们对不同的像素密度有了更好的了解。但是, 除非您知道不同颜色的色相值, 否则乍一看并不能使我们知道到底是什么颜色。我还是更喜欢这种方法。它简单而更好。

注意 使用此功能时, 请记住, 插值法应采用最近邻以获得更好的结果。

考虑下面的代码:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg')
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
hist = cv.calcHist( [hsv], [0, 1], None, [180, 256], [0, 180, 0, 256] )
plt.imshow(hist, interpolation = 'nearest')
plt.show()
```

下面是输入图像及其颜色直方图。X轴显示S值，Y轴显示色相。



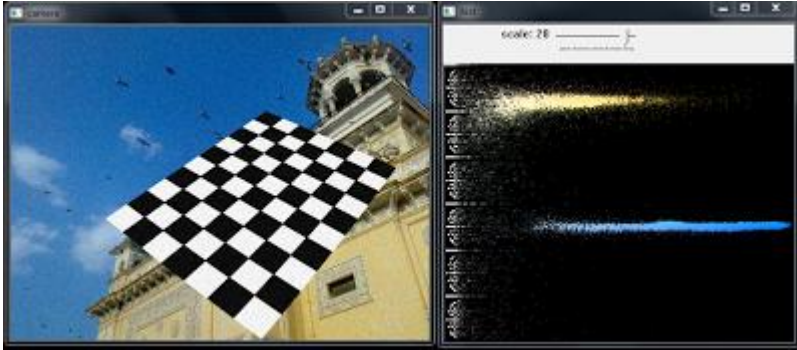
在直方图中，您可以在 $H = 100$ 和 $S = 200$ 附近看到一些较高的值。它对应于天空的蓝色。同样，在 $H = 25$ 和 $S = 100$ 附近可以看到另一个峰值。它对应于宫殿的黄色。您可以使用GIMP等任何图像编辑工具进行验证。

方法3：OpenCV示例样式

OpenCV-Python2示例中有一个颜色直方图的示例代码(samples / python / color_histogram.py)。如果运行代码，则可以看到直方图也显示了相应的颜色。或者简单地，它输出颜色编码的直方图。其结果非常好（尽管您需要添加额外的线束）。

在该代码中，作者在HSV中创建了一个颜色图。然后将其转换为BGR。将所得的直方图图像与此颜色图相乘。他还使用一些预处理步骤来删除小的孤立像素，从而获得良好的直方图。

我将其留给读者来运行代码，对其进行分析并拥有自己的解决方法。下面是与上面相同的图像的代码输出：



您可以在直方图中清楚地看到存在什么颜色，那里是蓝色，那里是黄色，并且由于棋盘的存在而有些白色。很好！

附加资源

练习

直方图4：直方图反投影

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，我们将学习直方图反投影。

理论

这是由**Michael J. Swain**和**Dana H. Ballard**在他们的论文《[通过颜色直方图索引](#)》中提出的。

用简单的话说是什么意思？它用于图像分割或在图像中查找感兴趣的对象。简而言之，它创建的图像大小与输入图像相同（但只有一个通道），其中每个像素对应于该像素属于我们物体的概率。用更简单的话来说，与其余部分相比，输出图像将在可能有对象的区域具有更多的白色值。好吧，这是一个直观的解释。（我无法使其更简单）。直方图反投影与camshift算法等配合使用。

我们该怎么做呢？我们创建一个图像的直方图，其中包含我们感兴趣的对象（在我们的示例中是背景，离开播放器等）。对象应尽可能填充图像以获得更好的效果。而且颜色直方图比灰度直方图更可取，因为对象的颜色对比灰度强度是定义对象的好方法。然后，我们将该直方图“反投影”到需要找到对象的测试图像上，换句话说，我们计算出属于背景的每个像素的概率并将其显示出来。在适当的阈值下产生的输出使我们仅获得背景。

Numpy中的算法

1. 首先，我们需要计算我们要查找的对象（使其为“M”）和要搜索的图像（使其为“I”）的颜色直方图。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
```



```
#roi是我们需要找到的对象或对象区域
roi = cv.imread('rose_red.png')
hsv = cv.cvtColor(roi,cv.COLOR_BGR2HSV)
#目标是我们搜索的图像
target = cv.imread('rose.png')
hsvt = cv.cvtColor(target,cv.COLOR_BGR2HSV)
# 使用calcHist查找直方图。也可以使用np.histogram2d完成
M = cv.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )
I = cv.calcHist([hsvt],[0, 1], None, [180, 256], [0, 180, 0, 256] )
```

1. 求出比值 $R = \frac{M}{I}$ 。然后反向投影R，即使用R作为调色板，并以每个像素作为其对应的目标概率创建一个新图像。即 $B(x, y) = R[h(x, y), s(x, y)]$ 其中h是色调，s是像素在(x, y)的饱和度。之后，应用条件 $B(x, y) = \min[B(x, y), 1]$ 。

```
h,s,v = cv.split(hsvt)
B = R[h.ravel(),s.ravel()]
B = np.minimum(B,1)
B = B.reshape(hsvt.shape[:2])
```

1. 现在对圆盘应用卷积， $B = D \ast B$ ，其中D是圆盘内核。

```
disc = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
cv.filter2D(B,-1,disc,B)
B = np.uint8(B)
cv.normalize(B,B,0,255,cv.NORM_MINMAX)
```

1. 现在最大强度的位置给了我们物体的位置。如果我们期望图像中有一个区域，则对合适的值进行阈值处理将获得不错的结果。

```
ret,thresh = cv.threshold(B,50,255,0)
```

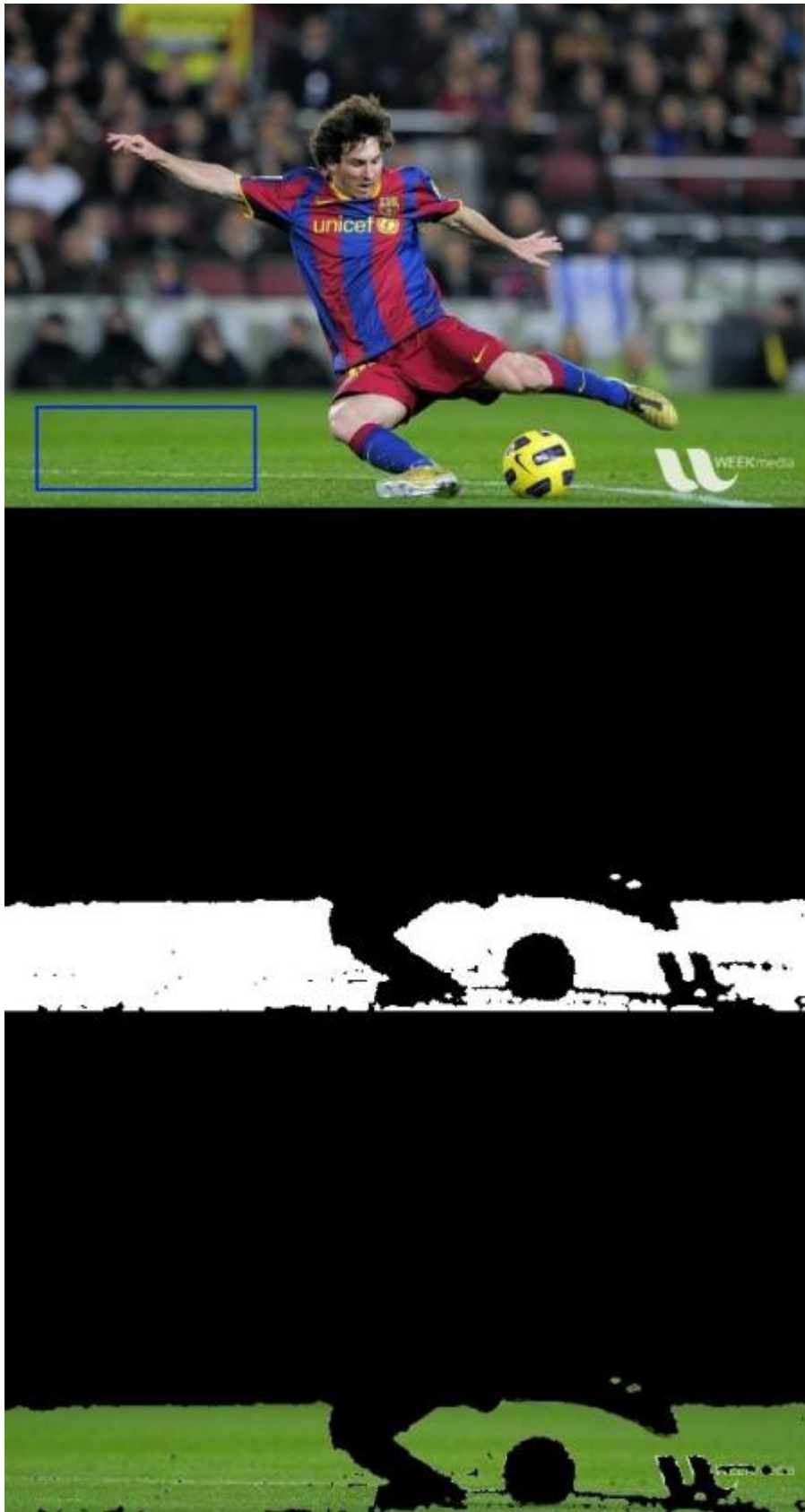
就是这样！！

OpenCV的反投影

OpenCV提供了一个内建的函数**cv.calcBackProject**()。它的参数几乎与**cv.calcHist**()函数相同。它的一个参数是直方图，也就是物体的直方图，我们必须找到它。另外，在传递给backproject函数之前，应该对对象直方图进行归一化。它返回概率图像。然后我们用圆盘内核对图像进行卷积并应用阈值。下面是我的代码和结果:

```
import numpy as np
import cv2 as cv
roi = cv.imread('rose_red.png')
hsv = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
target = cv.imread('rose.png')
hsvt = cv.cvtColor(target, cv.COLOR_BGR2HSV)
# 计算对象的直方图
roihist = cv.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256] )
# 直方图归一化并利用反传算法
cv.normalize(roihist, roihist, 0, 255, cv.NORM_MINMAX)
dst = cv.calcBackProject([hsvt], [0, 1], roihist, [0, 180, 0, 256], 1)
# 用圆盘进行卷积
disc = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
cv.filter2D(dst, -1, disc, dst)
# 应用阈值作与操作
ret, thresh = cv.threshold(dst, 50, 255, 0)
thresh = cv.merge((thresh, thresh, thresh))
res = cv.bitwise_and(target, thresh)
res = np.vstack((target, thresh, res))
cv.imwrite('res.jpg', res)
```

以下是我处理过的一个示例。我将蓝色矩形内的区域用作示例对象，我想提取整个地面。



附加资源

1. “Indexing via color histograms”, Swain, Michael J. , Third international conference on computer vision,1990.

练习

傅里叶变换

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本节中，我们将学习 - 使用OpenCV查找图像的傅立叶变换 - 利用Numpy中可用的FFT函数 - 傅立叶变换的某些应用程序 - 我们将看到以下函数：**cv.dft()**，**cv.idft()**等

理论

傅立叶变换用于分析各种滤波器的频率特性。对于图像，使用**2D离散傅里叶变换**(DFT)查找频域。一种称为**快速傅立叶变换**(FFT)的快速算法用于DFT的计算。关于这些的详细信息可以在任何图像处理或信号处理教科书中找到。请参阅其他资源部分。

对于正弦信号 $x(t) = A \sin(2\pi ft)$ ，我们可以说 f 是信号的频率，如果采用其频域，则可以看到 f 的尖峰。如果对信号进行采样以形成离散信号，我们将获得相同的频域，但是在 $[-\pi, \pi]$ 或 $[0, 2\pi]$ 范围内（对于 N 点DFT为 $[0, N]$ ）是周期性的。您可以将图像视为在两个方向上采样的信号。因此，在 X 和 Y 方向都进行傅立叶变换，可以得到图像的频率表示。

更直观地说，对于正弦信号，如果幅度在短时间内变化如此之快，则可以说它是高频信号。如果变化缓慢，则为低频信号。您可以将相同的想法扩展到图像。图像中的振幅在哪里急剧变化？在边缘点或噪声。因此，可以说边缘和噪声是图像中的高频内容。如果幅度没有太大变化，则它是低频分量。（一些链接已添加到“其他资源”，其中通过示例直观地说明了频率变换）。

现在，我们将看到如何找到傅立叶变换。

Numpy中的傅里叶变换

首先，我们将看到如何使用Numpy查找傅立叶变换。Numpy具有FFT软件包来执行此操作。

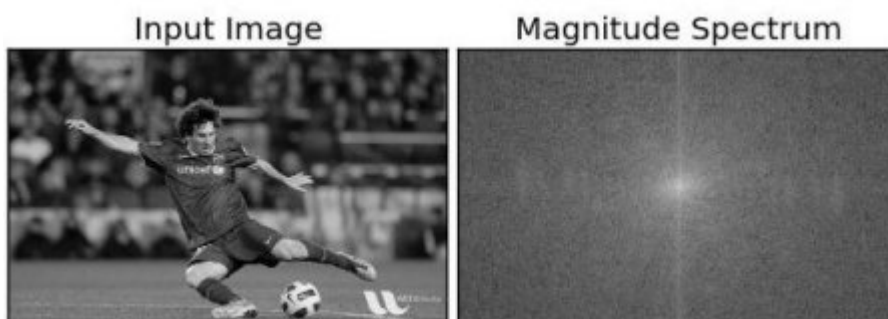
np.fft.fft2()为我们提供了频率转换，它将是一个复杂的数组。它的第一个参数是输入图像，即灰度图像。第二个参数是可选的，它决定输出数组的大小。如果它大于输入图像的大小，则在计算FFT

之前用零填充输入图像。如果小于输入图像，将裁切输入图像。如果未传递任何参数，则输出数组的大小将与输入的大小相同。

现在，一旦获得结果，零频率分量 (DC分量) 将位于左上角。如果要使其居中，则需要在两个方向上将结果都移动 $\frac{N}{2}$ 。只需通过函数`np.fft.fftshift()`即可完成。(它更容易分析)。找到频率变换后，就可以找到幅度谱。

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
Result look like below:
```

结果看起来像下面这样:



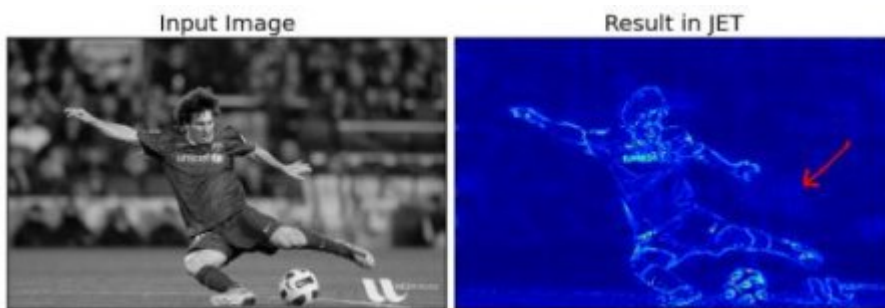
看，您可以在中心看到更多白色区域，这表明低频内容更多。

因此，您发现了频率变换现在，您可以在频域中进行一些操作，例如高通滤波和重建图像，即找到逆DFT。为此，您只需用尺寸为60x60的矩形窗口遮罩即可消除低频。然后，使用`np.fft.ifftshift()`应用反向移位，以使DC分量再次出现在左上角。然后使用`np.ifft2()`函数找到逆FFT。同样，结果将是一个复数。您可以采用其绝对值。

```
rows, cols = img.shape
crow,ccol = rows//2 , cols//2
fshift[crow-30:crow+31, ccol-30:ccol+31] = 0
f_ishift = np.fft.ifftshift(fshift)
```

```
img_back = np.fft.ifft2(f_ishift)
img_back = np.real(img_back)
plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_back)
plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
plt.show()
```

结果看起来像下面这样：



结果表明高通滤波是边缘检测操作。这就是我们在“图像渐变”一章中看到的。这也表明大多数图像数据都存在于频谱的低频区域。无论如何，我们已经看到了如何在Numpy中找到DFT，IDFT等。现在，让我们看看如何在OpenCV中进行操作。如果您仔细观察结果，尤其是最后一张JET颜色的图像，您会看到一些伪像（我用红色箭头标记的一个实例）。它在那里显示出一些波纹状结构，称为**振铃效应**。这是由我们用于遮罩的矩形窗口引起的。此掩码转换为正弦形状，从而导致此问题。因此，矩形窗口不用于过滤。更好的选择是高斯窗口。

OpenCV中的傅里叶变换

OpenCV为此提供了**cv.dft**()和**cv.idft**()函数。它返回与前一个相同的结果，但是有两个通道。第一个通道是结果的实部，第二个通道是结果的虚部。输入图像首先应转换为 `np.float32`。我们来看看怎么做。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg',0)
dft = cv.dft(np.float32(img),flags = cv.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv.magnitude(dft_shift[:, :, 0],dft_shift[:, :, 1]))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

注意 您还可以使用**cv.cartToPolar**(), 它在单个镜头中同时返回幅值和相位

现在我们要做DFT的逆变换。在上一节中, 我们创建了一个HPF, 这次我们将看到如何删除图像中的高频内容, 即我们将LPF应用到图像中。它实际上模糊了图像。为此, 我们首先创建一个高值(1)在低频部分, 即我们过滤低频内容, 0在高频区。

```
rows, cols = img.shape
crow,ccol = rows/2 , cols/2
# 首先创建一个掩码, 中心正方形为1, 其余全为零
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1
# 应用掩码和逆DFT
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = cv.idft(f_ishift)
img_back = cv.magnitude(img_back[:, :, 0],img_back[:, :, 1])
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



注意 通常, OpenCV函数**cv.dft**()和**cv.idft**()比Numpy函数更快。但是Numpy函数更容易使用。有关性能问题的更多细节, 请参见下面的部分。

DFT的性能优化

对于某些数组尺寸, DFT的计算性能较好。当数组大小为2的幂时, 速度最快。对于大小为2、3和5的乘积的数组, 也可以非常有效地进行处理。因此, 如果您担心代码的性能, 可以在找到DFT之

前将数组的大小修改为任何最佳大小(通过填充零)。对于OpenCV, 您必须手动填充零。但是对于Numpy, 您指定FFT计算的新大小, 它将自动为您填充零。

那么如何找到最优的大小呢?OpenCV为此提供了一个函数, `cv.getOptimalDFTSize()`。它同时适用于`cv.dft()`和`np.fft.fft2()`。让我们使用IPython魔术命令`timeit`来检查它们的性能。

```
In [16]: img = cv.imread('messi5.jpg',0)
In [17]: rows,cols = img.shape
In [18]: print("{} {}".format(rows,cols))
342 548
In [19]: nrows = cv.getOptimalDFTSize(rows)
In [20]: ncols = cv.getOptimalDFTSize(cols)
In [21]: print("{} {}".format(nrows,ncols))
360 576
```

参见, 将大小 `(342,548)` 修改为 `(360,576)`。现在让我们用零填充 (对于OpenCV), 并找到其DFT计算性能。您可以通过创建一个新的零数组并将数据复制到其中来完成此操作, 或者使用`cv.copyMakeBorder()`。

```
nimg = np.zeros((nrows,ncols))
nimg[:rows,:cols] = img
```

或者:

```
right = ncols - cols
bottom = nrows - rows
bordertype = cv.BORDER_CONSTANT # 只是为了避免PDF文件中的行中断
nimg = cv.copyMakeBorder(img,0,bottom,0,right,bordertype, value = 0)
```

现在, 我们计算Numpy函数的DFT性能比较:

```
In [22]: %timeit fft1 = np.fft.fft2(img)
10 loops, best of 3: 40.9 ms per loop
In [23]: %timeit fft2 = np.fft.fft2(img,[nrows,ncols])
100 loops, best of 3: 10.4 ms per loop
```

它显示了4倍的加速。现在, 我们将尝试使用OpenCV函数。

```
In [24]: %timeit dft1= cv.dft(np.float32(img),flags=cv.DFT_COMPLEX_OUTPUT)
100 loops, best of 3: 13.5 ms per loop
```

```
In [27]: %timeit dft2= cv.dft(np.float32(nimg),flags=cv.DFT_COMPLEX_OUTPUT)
100 loops, best of 3: 3.11 ms per loop
```

它还显示了4倍的加速。您还可以看到OpenCV函数比Numpy函数快3倍左右。也可以对逆FFT进行测试，这留给您练习。

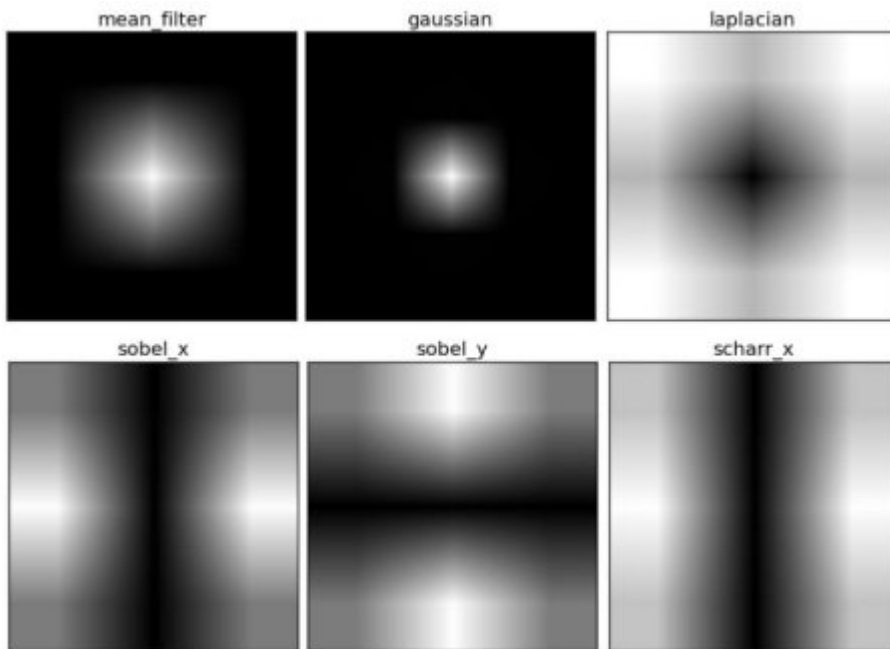
为什么拉普拉斯算子高通滤波器？

在一个论坛上也有人提出了类似的问题。问题是，为什么拉普拉斯变换是高通滤波器？为什么Sobel是HPF？等。第一个答案是关于傅里叶变换的。对于更大的FFT只需要拉普拉斯变换。分析下面的代码：

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
# 没有缩放参数的简单均值滤波器
mean_filter = np.ones((3,3))
# 创建高斯滤波器
x = cv.getGaussianKernel(5,10)
gaussian = x*x.T
# 不同的边缘检测滤波器
# x方向上的scharr
scharr = np.array([[ -3,  0,  3],
                   [-10, 0, 10],
                   [ -3,  0,  3]])
# x方向上的sobel
sobel_x= np.array([[ -1,  0,  1],
                   [-2,  0,  2],
                   [ -1,  0,  1]])
# y方向上的sobel
sobel_y= np.array([[ -1,-2,-1],
                   [ 0,  0,  0],
                   [ 1,  2,  1]])
# 拉普拉斯变换
laplacian=np.array([[ 0,  1,  0],
                    [ 1,-4,  1],
                    [ 0,  1,  0]])
filters = [mean_filter, gaussian, laplacian, sobel_x, sobel_y, scharr]
filter_name = ['mean_filter', 'gaussian','laplacian', 'sobel_x', \
               'sobel_y', 'scharr_x']
fft_filters = [np.fft.fft2(x) for x in filters]
fft_shift = [np.fft.fftshift(y) for y in fft_filters]
mag_spectrum = [np.log(np.abs(z)+1) for z in fft_shift]
for i in xrange(6):
```

```
plt.subplot(2,3,i+1),plt.imshow(mag_spectrum[i],cmap = 'gray')
plt.title(filter_name[i]), plt.xticks([]), plt.yticks([])
plt.show()
```

看看结果：



从图像中，您可以看到每种内核阻止的频率区域以及它允许经过的区域。从这些信息中，我们可以说出为什么每个内核都是HPF或LPF

附加资源

1. 傅里叶变换的直观解释：<http://cns-alumni.bu.edu/~slehar/fourier/fourier.html> by Steven Lehar
2. 傅里叶变换：<http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm> at HIPR
3. 图像中的频率域指什么？<http://dsp.stackexchange.com/q/1637/818>

练习

模板匹配

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，您将学习 - 使用模板匹配在图像中查找对象 - 您将看到以下功能：

cv.matchTemplate()，**cv.minMaxLoc()**

理论

模板匹配是一种用于在较大图像中搜索和查找模板图像位置的方法。为此，OpenCV带有一个函数 **cv.matchTemplate()**。它只是将模板图像滑动到输入图像上（就像在2D卷积中一样），然后在模板图像下比较模板和输入图像的拼图。OpenCV中实现了几种比较方法。（您可以检查文档以了解更多详细信息）。它返回一个灰度图像，其中每个像素表示该像素的邻域与模板匹配的程度。

如果输入图像的大小为 $(W \times H)$ ，而模板图像的大小为 $(w \times h)$ ，则输出图像的大小将为 $(W - w + 1, H - h + 1)$ 。得到结果后，可以使用 **cv.minMaxLoc()** 函数查找最大/最小值在哪。将其作为矩形的左上角，并以 (w, h) 作为矩形的宽度和高度。该矩形是您模板的区域。

注意 如果使用 **cv.TM_SQDIFF** 作为比较方法，则最小值提供最佳匹配。

OpenCV中的模板匹配



作为示例，我们将在梅西的照片中搜索他的脸。所以我创建了一个模板，如下所示：我们将尝试所有比较方法，以便我们可以看到它们的结果如何：

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
```

```
img = cv.imread('messi5.jpg',0)
img2 = img.copy()
template = cv.imread('template.jpg',0)
w, h = template.shape[:,-1]
# 列表中所有的6种比较方法
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
           'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']
for meth in methods:
    img = img2.copy()
    method = eval(meth)
    # 应用模板匹配
    res = cv.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
    # 如果方法是TM_SQDIFF或TM_SQDIFF_NORMED, 则取最小值
    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv.rectangle(img,top_left, bottom_right, 255, 2)
    plt.subplot(121),plt.imshow(res,cmap = 'gray')
    plt.title('Matching Result'), plt.xticks([], plt.yticks([]))
    plt.subplot(122),plt.imshow(img,cmap = 'gray')
    plt.title('Detected Point'), plt.xticks([], plt.yticks([]))
    plt.suptitle(meth)
    plt.show()
```

查看以下结果：

- **cv.TM_CCOEFF**

Matching Result



Detected Point



Matching Result



Detected Point



• **cv.TM_CCOEFF_NORMED**

Matching Result



Detected Point



• **cv.TM_CCORR**

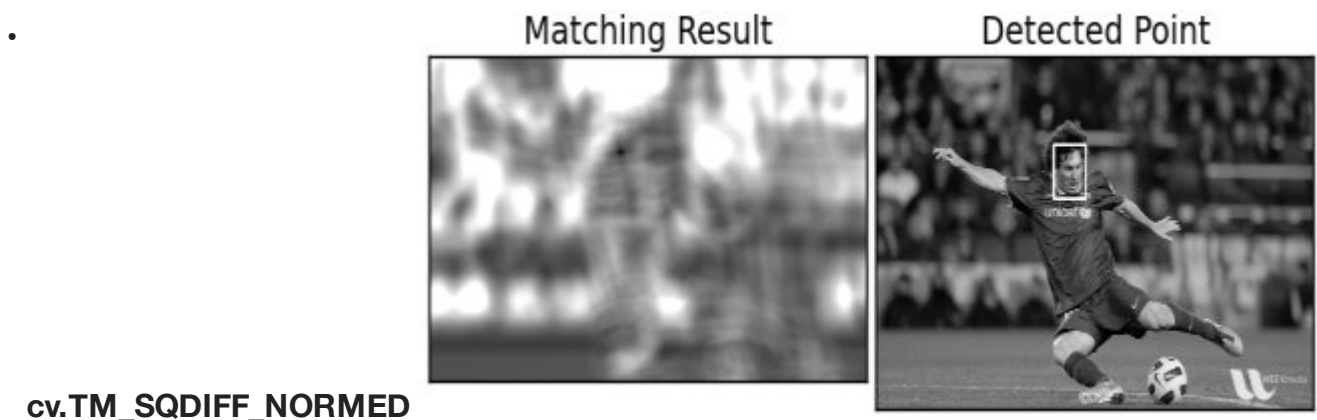
Matching Result



Detected Point



• **cv.TM_CCORR_NORMED**



您会看到，使用**cv.TM_CCORR**的结果并不理想。

多对象的模板匹配

在上一节中，我们在图像中搜索了梅西的脸，该脸在图像中仅出现一次。假设您正在搜索具有多次出现的对象，则**cv.minMaxLoc**()不会为您提供所有位置。在这种情况下，我们将使用阈值化。因此，在此示例中，我们将使用著名游戏**Mario**的屏幕截图，并在其中找到硬币。

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img_rgb = cv.imread('mario.png')
img_gray = cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)
template = cv.imread('mario_coin.png', 0)
w, h = template.shape[: -1]
res = cv.matchTemplate(img_gray, template, cv.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[: -1]):
    cv.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0, 0, 255), 2)
cv.imwrite('res.png', img_rgb)
```

结果:



附加资源

练习

霍夫线变换

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

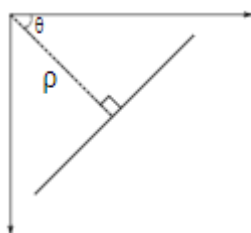
目标

在这一章当中， - 我们将了解霍夫变换的概念。 - 我们将看到如何使用它来检测图像中的线条。 - 我们将看到以下函数：`cv.HoughLines()`，`cv.HoughLinesP()`

理论

如果可以用数学形式表示形状，则霍夫变换是一种检测任何形状的流行技术。即使形状有些破损或变形，也可以检测出形状。我们将看到它如何作用于一条线。

一条线可以表示为 $y = mx + c$ 或以参数形式表示为 $\rho = x \cos \theta + y \sin \theta$ ，其中 ρ 是从原点到该线的垂直距离，而 θ 是由该垂直线和水平轴形成的角度以逆时针方向测量（该方向随您如何表示坐标而变化。此表示形式在OpenCV中使用）。查看下面的图片：

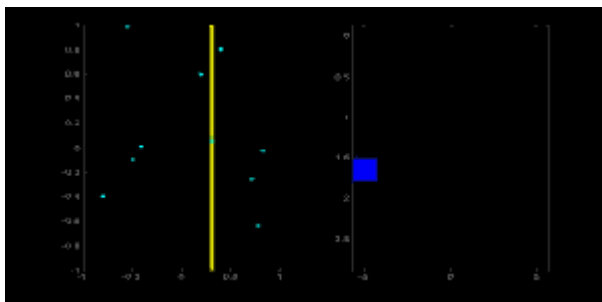


因此，如果线在原点下方通过，则它将具有正的 ρ 且角度小于180。如果线在原点上方，则将角度取为小于180，而不是大于180的角度。 ρ 取负值。任何垂直线将具有0度，水平线将具有90度。

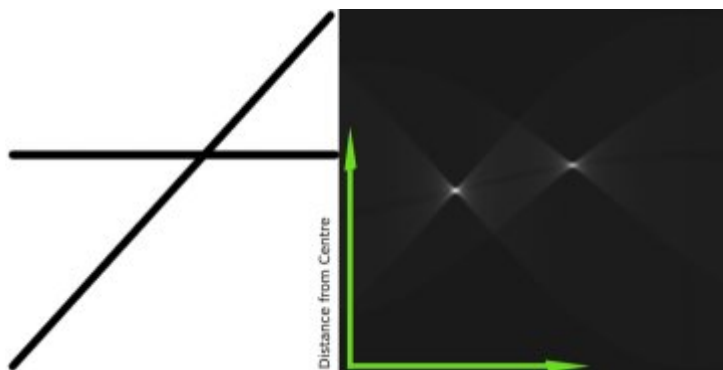
现在，让我们看一下霍夫变换如何处理线条。任何一条线都可以用 (ρ, θ) 这两个术语表示。因此，首先创建2D数组或累加器（以保存两个参数的值），并将其初始设置为0。让行表示 ρ ，列表示 θ 。阵列的大小取决于所需的精度。假设您希望角度的精度为1度，则需要180列。对于 ρ ，最大距离可能是图像的对角线长度。因此，以一个像素精度为准，行数可以是图像的对角线长度。

考虑一个 100×100 的图像，中间有一条水平线。取直线的第一点。您知道它的 (x, y) 值。现在在线性方程式中，将值 $\theta = 0, 1, 2, \dots, 180$ 放进去，然后检查得到 ρ 。对于每对 (ρ, θ) ，在累加器中对应的 (ρ, θ) 单元格将值增加1。所以现在在累加器中，单元格 $(50, 90) = 1$ 以及其他一些单元格。

现在，对行的第二个点。执行与上述相同的操作。递增 (ρ, θ) 对应的单元格中的值。这次，单元格 $(50, 90) = 2$ 。实际上，您正在对 (ρ, θ) 值进行投票。您对线路上的每个点都继续执行此过程。在每个点上，单元格 $(50, 90)$ 都会增加或投票，而其他单元格可能会或可能不会投票。这样一来，最后，单元格 $(50, 90)$ 的投票数将最高。因此，如果您在累加器中搜索最大票数，则将获得 $(50, 90)$ 值，该值表示该图像中的一条线与原点的距离为50，角度为90度。在下面的动画中很好地显示了该图片(图片提供：Amos Storkey)



这就是霍夫变换对线条的工作方式。它很简单，也许您可以自己使用Numpy来实现它。下图显示了累加器。某些位置的亮点表示它们是图像中可能的线条的参数。(图片由维基百科提供)

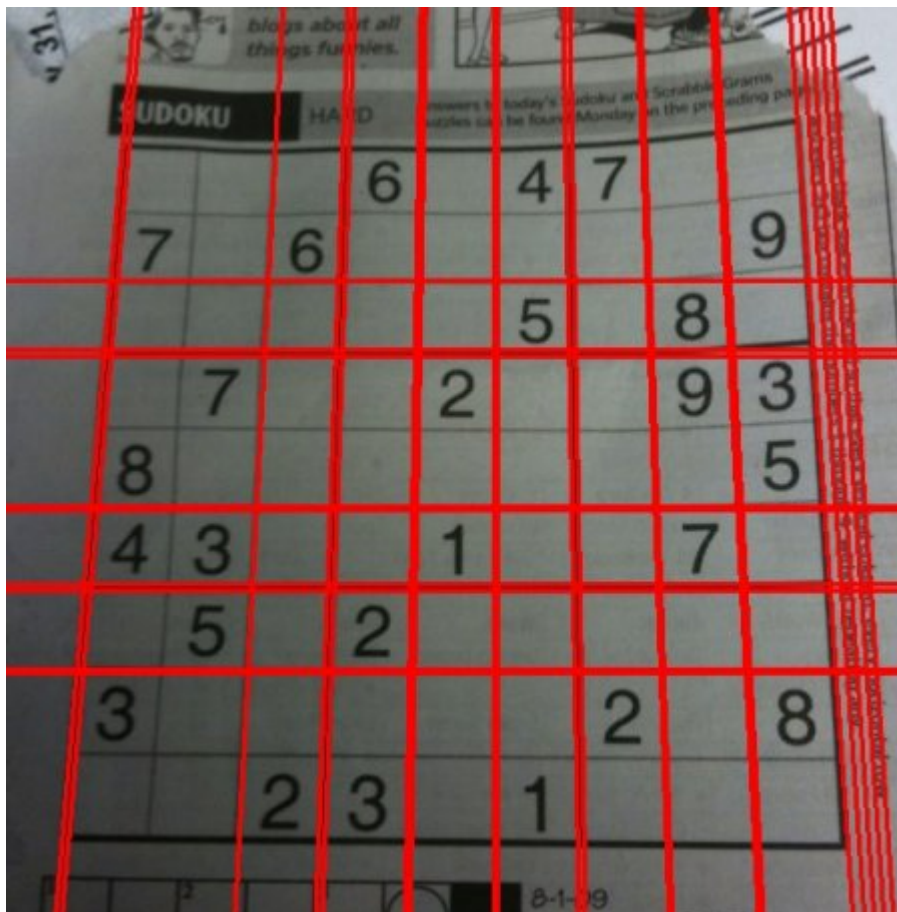


OpenCV中的霍夫曼变换

上面说明的所有内容都封装在OpenCV函数 `cv.HoughLines()` 中。它只是返回一个：

`math: (rho, theta)` 值的数组。 ρ 以像素为单位， θ 以弧度为单位。第一个参数，输入图像应该是二进制图像，因此在应用霍夫变换之前，请应用阈值或使用Canny边缘检测。第二和第三参数分别是 ρ 和 θ 精度。第四个参数是阈值，这意味着应该将其视为行的最低投票。请记住，票数取决于线上的点数。因此，它表示应检测到的最小线长。

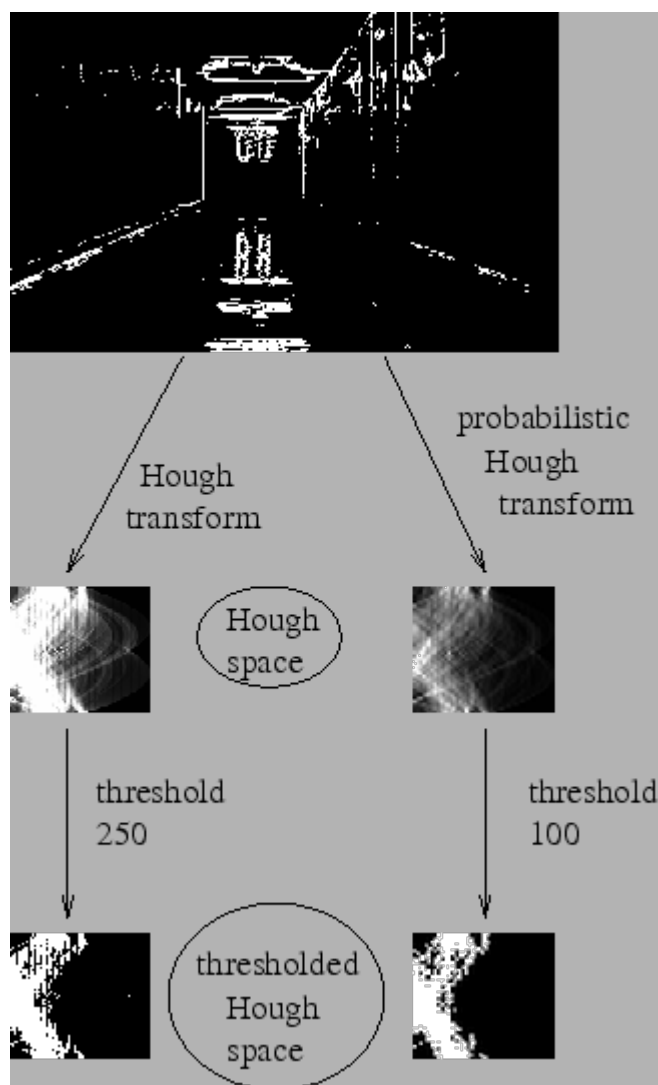
```
import cv2 as cv
import numpy as np
img = cv.imread(cv.samples.findFile('sudoku.png'))
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray, 50, 150, apertureSize = 3)
lines = cv.HoughLines(edges, 1, np.pi/180, 200)
for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)
cv.imwrite('houghlines3.jpg', img)
```



检查下面的结果

概率霍夫变换

在霍夫变换中，您可以看到，即使对于带有两个参数的行，也需要大量计算。概率霍夫变换是我们看到的霍夫变换的优化。它没有考虑所有要点。取而代之的是，它仅采用随机的点子集，足以进行线检测。只是我们必须降低阈值。参见下图，比较了霍夫空间中的霍夫变换和概率霍夫变换。（图片提供：Franck Bettinger的主页）

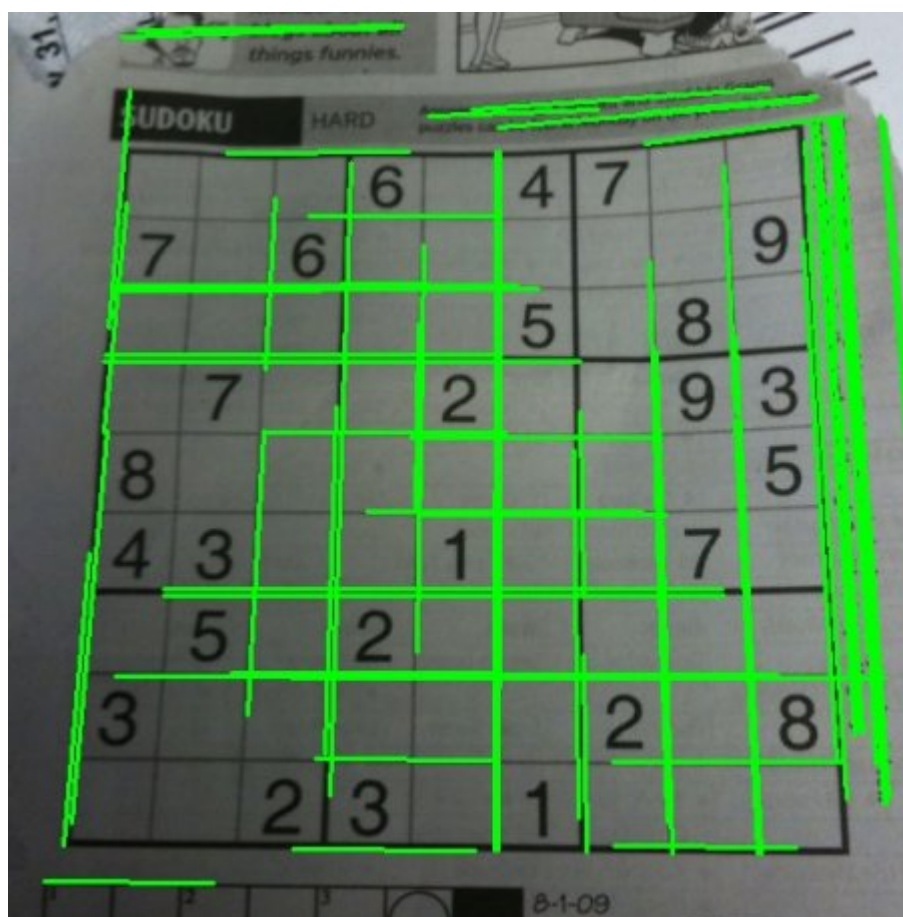


OpenCV的实现基于Matas,J.和Galambos,C.和Kittler, J.V.使用渐进概率霍夫变换对行进行的稳健检测[145]。使用的函数是`cv.HoughLinesP()`。它有两个新的论点。 - **minLineLength** - 最小行长。小于此长度的线段将被拒绝。 - **maxLineGap** - 线段之间允许将它们视为一条线的最大间隙。

最好的是，它直接返回行的两个端点。在以前的情况下，您仅获得线的参数，并且必须找到所有点。在这里，一切都是直接而简单的。

```
import cv2 as cv
import numpy as np
img = cv.imread(cv.samples.findFile('sudoku.png'))
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray, 50, 150, apertureSize = 3)
lines = cv.HoughLinesP(edges, 1, np.pi/180, 100, minLineLength=100, maxLineGap=10)
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
cv.imwrite('houghlines5.jpg', img)
```

看到如下结果：



附加资源

1. Hough Transform on Wikipedia : http://en.wikipedia.org/wiki/Hough_transform

练习

霍夫圈变换

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

学习目标

在本章中，- 我们将学习使用霍夫变换来查找图像中的圆。- 我们将看到以下函数：

cv.HoughCircles()

理论

圆在数学上表示为 $(x-x_{\text{center}})^2+(y-y_{\text{center}})^2 = r^2$ ，其中 $(x_{\text{center}}, y_{\text{center}})$ 是圆的中心， r 是圆的半径。从等式中，我们可以看到我们有3个参数，因此我们需要3D累加器进行霍夫变换，这将非常低效。因此，OpenCV使用更加技巧性的方法，即使用边缘的梯度信息的**Hough梯度方法**。

我们在这里使用的函数是**cv.HoughCircles**()。它有很多参数，这些参数在文档中有很好的解释。因此，我们直接转到代码。

```
import numpy as np
import cv2 as cv
img = cv.imread('opencv-logo-white.png', 0)
img = cv.medianBlur(img, 5)
cimg = cv.cvtColor(img, cv.COLOR_GRAY2BGR)
circles = cv.HoughCircles(img, cv.HOUGH_GRADIENT, 1, 20,
                           param1=50, param2=30, minRadius=0, maxRadius=0)
circles = np.uint16(np.around(circles))
for i in circles[0, :]:
    # 绘制外圆
    cv.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2)
    # 绘制圆心
    cv.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3)
cv.imshow('detected circles', cimg)
cv.waitKey(0)
cv.destroyAllWindows()
```



结果如下：

附加资源

练习

图像分割与Watershed算法

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将学习使用分水岭算法实现基于标记的图像分割 - 我们将看到：`cv.watershed()`

理论

任何灰度图像都可以看作是一个地形表面，其中高强度表示山峰，低强度表示山谷。你开始用不同颜色的水(标签)填充每个孤立的 valleys(局部最小值)。随着水位的上升，根据附近的山峰(坡度)，来自不同山谷的水明显会开始合并，颜色也不同。为了避免这种情况，你要在水融合的地方建造屏障。你继续填满水，建造障碍，直到所有的山峰都在水下。然后你创建的屏障将返回你的分割结果。这就是Watershed背后的“思想”。你可以访问Watershed的CMM网页，了解它与一些动画的帮助。

但是这种方法会由于图像中的噪声或其他不规则性而产生过度分割的结果。因此OpenCV实现了一个基于标记的分水岭算法，你可以指定哪些是要合并的山谷点，哪些不是。这是一个交互式的图像分割。我们所做的是给我们知道的对象赋予不同的标签。用一种颜色(或强度)标记我们确定为前景或对象的区域，用另一种颜色标记我们确定为背景或非对象的区域，最后用 `0` 标记我们不确定的区域。这是我们的标记。然后应用分水岭算法。然后我们的标记将使用我们给出的标签进行更新，对象的边界值将为 `-1`。

代码

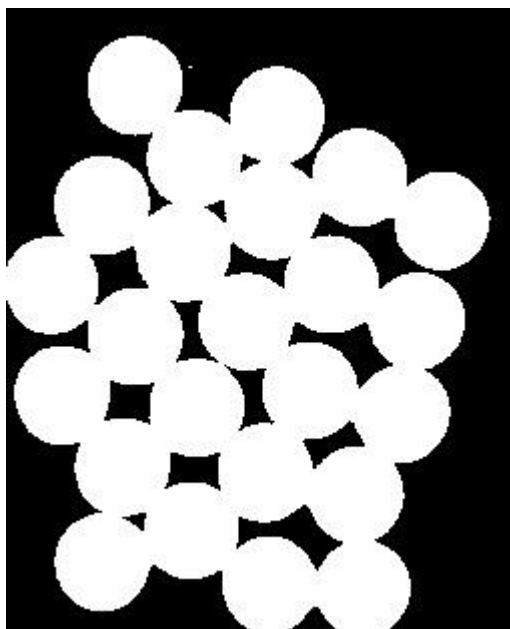
下面我们将看到一个有关如何使用距离变换和分水岭来分割相互接触的对象示例。

考虑下面的硬币图像，硬币彼此接触。即使你设置阈值，它也会彼此接触。



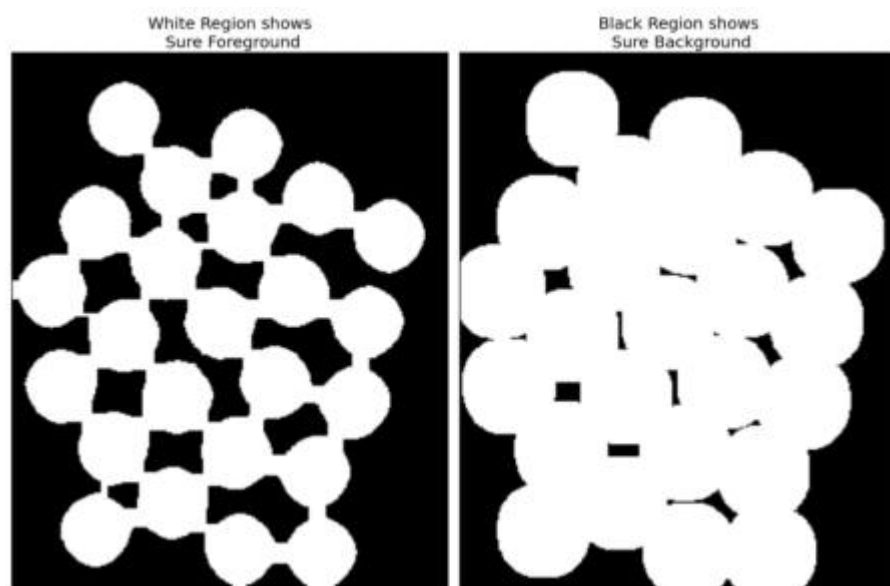
我们先从寻找硬币的近似估计开始。因此，我们可以使用Otsu的二值化。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('coins.png')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
```



现在我们需要去除图像中的任何白点噪声。为此，我们可以使用形态学扩张。要去除对象中的任何小孔，我们可以使用形态学侵蚀。因此，现在我们可以确定，靠近对象中心的区域是前景，而离对象中心很远的区域是背景。我们不确定的唯一区域是硬币的边界区域。

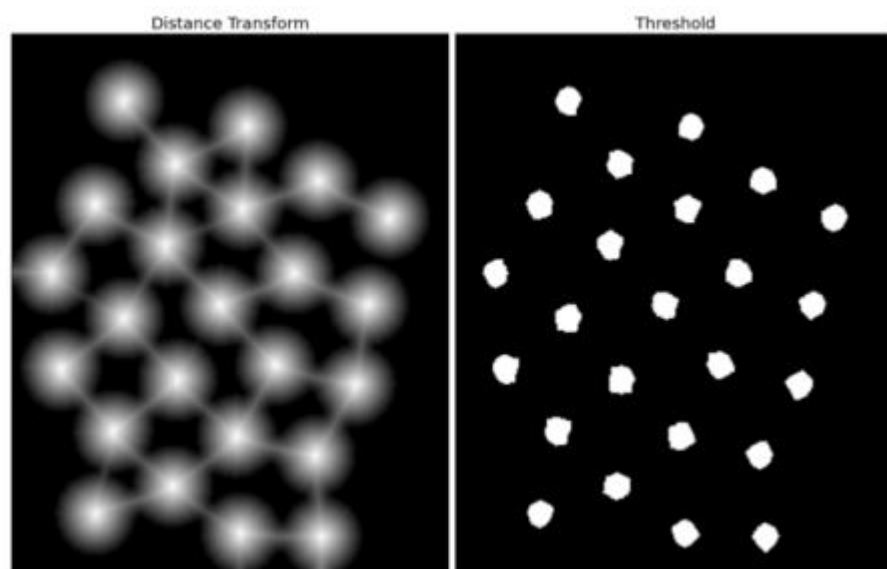
因此，我们需要提取我们可确定为硬币的区域。侵蚀会去除边界像素。因此，无论剩余多少，我们都可以肯定它是硬币。如果物体彼此不接触，那将起作用。但是，由于它们彼此接触，因此另一个好选择是找到距离变换并应用适当的阈值。接下来，我们需要找到我们确定它们不是硬币的区域。为此，我们扩张了结果。膨胀将对象边界增加到背景。这样，由于边界区域已删除，因此我们可以确保结果中背景中的任何区域实际上都是背景。参见下图。



剩下的区域是我们不知道的区域，无论是硬币还是背景。分水岭算法应该找到它。这些区域通常位于前景和背景相遇（甚至两个不同的硬币相遇）的硬币边界附近。我们称之为边界。可以通过从 `sure_bg` 区域中减去 `sure_fg` 区域来获得。

```
# 噪声去除
kernel = np.ones((3,3),np.uint8)
opening = cv.morphologyEx(thresh,cv.MORPH_OPEN,kernel, iterations = 2)
# 确定背景区域
sure_bg = cv.dilate(opening,kernel,iterations=3)
# 寻找前景区域
dist_transform = cv.distanceTransform(opening,cv.DIST_L2,5)
ret, sure_fg = cv.threshold(dist_transform,0.7*dist_transform.max(),255,0)
# 找到未知区域
sure_fg = np.uint8(sure_fg)
unknown = cv.subtract(sure_bg,sure_fg)
```

查看结果。在阈值图像中，我们得到了一些硬币区域，我们确定它们是硬币，并且现在已分离它们。（在某些情况下，你可能只对前景分割感兴趣，而不对分离相互接触的对象感兴趣。在那种情况下，你无需使用距离变换，只需侵蚀就足够了。侵蚀只是提取确定前景区域的另一种方法。）

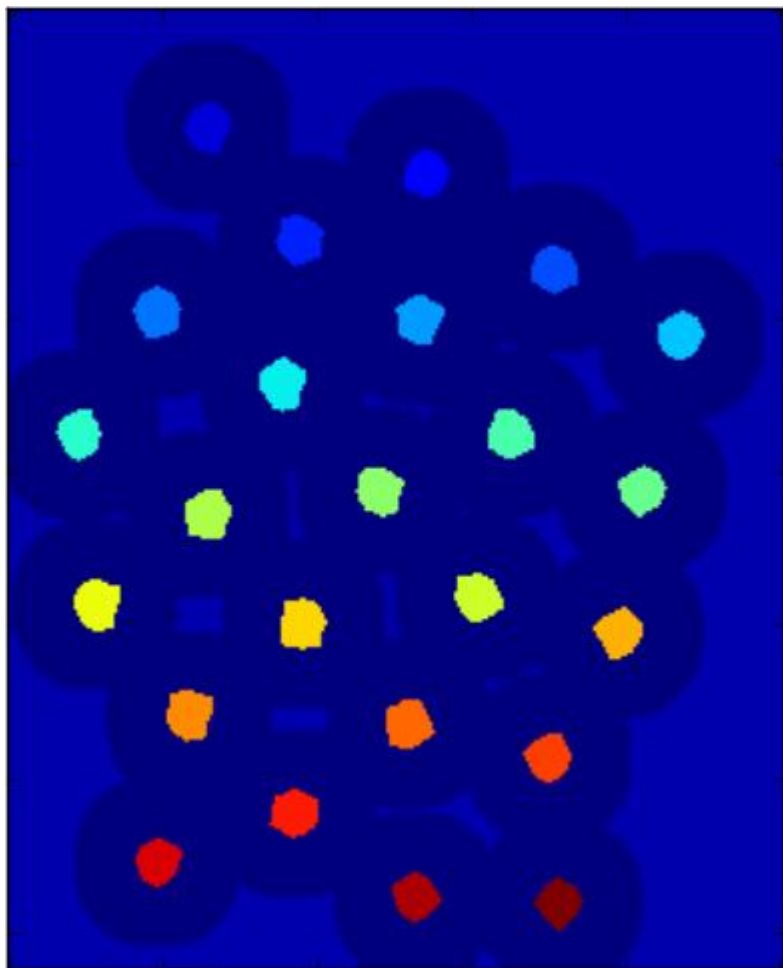


现在我们可以确定哪些是硬币的区域，哪些是背景。因此，我们创建了标记（它的大小与原始图像的大小相同，但具有int32数据类型），并标记其中的区域。我们肯定知道的区域（无论是前景还是背景）都标有任何正整数，但是带有不同的整数，而我们不确定的区域则保留为零。为此，我们使用`cv.connectedComponents()`。它用0标记图像的背景，然后其他对象用从1开始的整数标记。

但是我们知道，如果背景标记为0，则分水岭会将其视为未知区域。所以我们想用不同的整数来标记它。相反，我们将未知定义的未知区域标记为0。

```
# 类别标记
ret, markers = cv.connectedComponents(sure_fg)
# 为所有的标记加1，保证背景是0而不是1
markers = markers+1
# 现在让所有的未知区域为0
markers[unknown==255] = 0
```

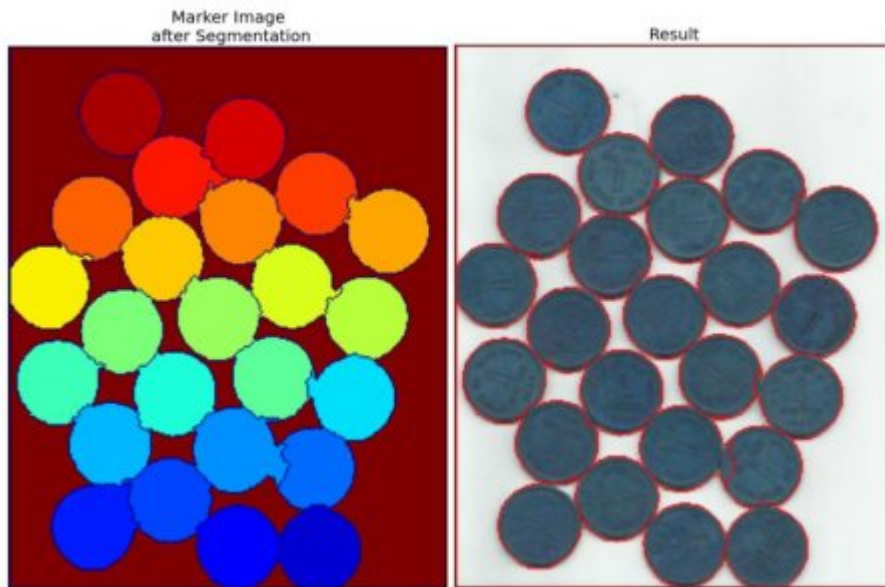
参见JET colormap中显示的结果。深蓝色区域显示未知区域。当然,硬币的颜色不同。剩下,肯定为背景的区域显示在较浅的蓝色，跟未知区域相比。



现在我们的标记已准备就绪。现在是最后一步的时候了，使用分水岭算法。然后标记图像将被修改。边界区域将标记为-1。

```
markers = cv.watershed(img, markers)
img[markers == -1] = [255, 0, 0]
```

请参阅下面的结果。对某些硬币，它们接触的区域被正确地分割，而对于某些硬币，却不是。



附加资源

1. CMM page on [Watershed Transformation](#)

练习

1. OpenCV samples has an interactive sample on watershed segmentation, [watershed.py](#).
Run it, Enjoy it, then learn it.

交互式前景提取使用GrabCut算法

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将看到GrabCut算法来提取图像中的前景 - 我们将为此创建一个交互式应用程序。

理论

GrabCut算法由英国微软研究院的Carsten Rother，Vladimir Kolmogorov和Andrew Blake设计。在他们的论文“GrabCut”中：使用迭代图割的交互式前景提取。需要用最少的用户交互进行前景提取的算法，结果是GrabCut。

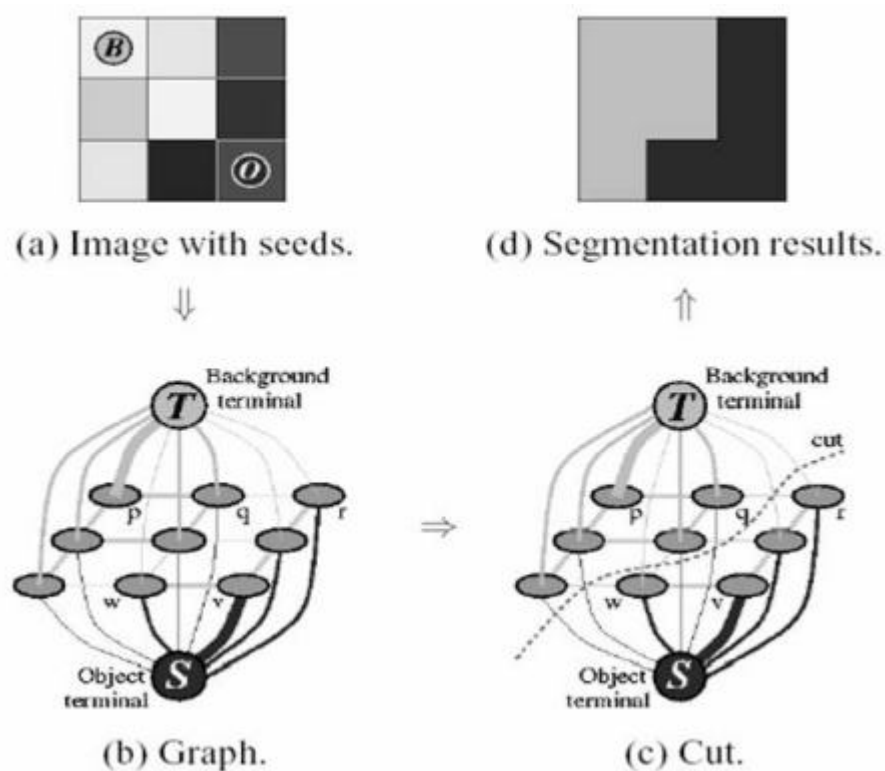
从用户角度来看，它是如何工作的？最初，用户在前景区域周围绘制一个矩形（前景区域应完全位于矩形内部）。然后，算法会对其进行迭代分割，以获得最佳结果。做完了但在某些情况下，分割可能不会很好，例如，可能已将某些前景区域标记为背景，反之亦然。在这种情况下，需要用户进行精修。只需在图像错误分割区域上画些笔画。笔画基本上说“嘿，该区域应该是前景，你将其标记为背景，在下一次迭代中对其进行校正”或与背景相反。然后在下一次迭代中，你将获得更好的结果。

参见下图。第一名球员和橄榄球被封闭在一个蓝色矩形中。然后用白色笔划（表示前景）和黑色笔划（表示背景）进行最后的修饰。而且我们得到了不错的结果。



那么背景发生了什么呢？ - 用户输入矩形。此矩形外部的所有内容都将作为背景（这是在矩形应包含所有对象之前提到的原因）。矩形内的所有内容都是未知的。同样，任何指定前景和背景的用户输入都被视为硬标签，这意味着它们在此过程中不会更改。 - 计算机根据我们提供的数据进行初始标记。它标记前景和背景像素（或对其进行硬标记），现在使用高斯混合模型(GMM)对前景和背景进行建模。 - 根据我们提供的数据，GMM可以学习并创建新的像素分布。也就是说，未知像素根据颜色统计上与其他硬标记像素的关系而被标记为可能的前景或可能的背景（就像聚类一样）。 - 根据此像素分布构建图形。图中的节点为像素。添加了另外两个节点，即“源”节点和“接收器”节点。每个前景像素都连接到源节点，每个背景像素都连接到接收器节点。 - 通过像素是前景/背景的概率来定义将像素连接到源节点/末端节点的边缘的权重。像素之间的权重由边缘信息或像素相似度定义。如果像素颜色差异很大，则它们之间的边缘将变低。 - 然后使用mincut算法对图进行分割。它将图切成具有最小成本函数的两个分离的源节点和宿节点。成本函数是被切割边缘的所有权重的总和。剪切后，连接到“源”节点的所有像素都变为前景，而连接到“接收器”节点的像素都变为背景。 - 继续该过程，直到分类收敛为止。

如下图所示（图片提供：<http://www.cs.ru.ac.za/research/g02m1682/>）



示例

现在我们使用OpenCV进行抓取算法。OpenCV为此具有功能`cv.grabCut()`，我们将首先看到其参数：- `img` - 输入图像 - `mask` - 这是一个掩码图像，在其中我们指定哪些区域是背景，前景或可能的背景/前景等。这是通过以下标志完成的：`cv.GC_BGD`, `cv.GC_FGD`, `cv.GC_PR_BGD`, `cv.GC_PR_FGD`，或直接将 `0, 1, 2, 3` 传递给图像。- `rect` - 它是矩形的坐标，其中包括前景对象，格式为 `(x, y, w, h)` - `bdgModel`, `fgdModel` - 这些是算法内部使用的数组。你只需创建两个大小为 `(1, 65)` 的 `np.float64` 类型零数组。- `iterCount` - 算法应运行的迭代次数。- `model` - 应该是`cv.GC_INIT_WITH_RECT`或`cv.GC_INIT_WITH_MASK`或两者结合，决定我们要绘制矩形还是最终的修饰笔触。

首先让我们看看矩形模式。我们加载图像，创建类似的mask图像。我们创建`*fgdModel*`和`*bgdModel*`。我们给出矩形参数。一切都是直截了当的。让算法运行5次迭代。模式应为`cv.GC_INIT_WITH_RECT`，因为我们使用的是矩形。然后运行`grabcut`。修改mask图像。在新的mask图像中，像素将被标记有四个标记，分别表示上面指定的背景/前景。因此，我们修改mask，使所有0像素和2像素都置为0（即背景），而所有1像素和3像素均置为1（即前景像素）。现在，我们的最终mask已经准备就绪。只需将其与输入图像相乘即可得到分割的图像。

```
import numpy as np
import cv2 as cv
```



```
from matplotlib import pyplot as plt
img = cv.imread('messi5.jpg')
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
rect = (50, 50, 450, 290)
cv.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2) | (mask==0), 0, 1).astype('uint8')
img = img*mask2[:, :, np.newaxis]
plt.imshow(img), plt.colorbar(), plt.show()
```



查看以下结果：

糟糕，梅西的头发不见了。谁会喜欢没有头发的梅西？我们需要把它找回来。因此，我们将使用1像素（确保前景）进行精细修饰。同时，一些不需要的地面也出现在图片里。我们需要删除它们。在那里，我们给出了一些0像素的修饰（确保背景）。因此，如现在所说，我们在以前的情况下修改生成的mask。

我实际上所做的是，我在paint应用程序中打开了输入图像，并在图像中添加了另一层。使用画笔中的画笔工具，我在新图层上用白色标记了错过的前景（头发，鞋子，球等），而用白色标记了不需要的背景（例如logo，地面等）。然后用灰色填充剩余的背景。然后将该mask图像加载到OpenCV中，编辑我们在新添加的mask图像中具有相应值的原始mask图像。

检查以下代码：

```
# newmask是我手动标记过的mask图像
newmask = cv.imread('newmask.png', 0)
# 标记为白色（确保前景）的地方，更改mask = 1
# 标记为黑色（确保背景）的地方，更改mask = 0
mask[newmask == 0] = 0
mask[newmask == 255] = 1
```

```
mask, bgdModel, fgdModel = cv.grabCut(img,mask, None,bgdModel,fgdModel,
5,cv.GC_INIT_WITH_MASK)
mask = np.where((mask==2) | (mask==0),0,1).astype('uint8')
img = img*mask[:,:,np.newaxis]
plt.imshow(img),plt.colorbar(),plt.show()
```

查看以下结果：



就是这样了。在这里，你无需直接在rect模式下初始化，而可以直接进入mask模式。只需用2像素或3像素（可能的背景/前景）标记mask图像中的矩形区域。然后像在第二个示例中一样，将我们的sure_foreground标记为1像素。然后直接在mask模式下应用grabCut功能。

附加资源

练习

1. OpenCV示例包含一个示例catchcut.py这是一个使用grabcut的交互式工具。检查。另请观看有关如何使用它的youtube视频。
2. 在这里，你可以通过绘制矩形和使用鼠标笔触使其成为交互式示例，创建轨迹栏以调整笔触宽度等。

理解特征

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，我们将尝试理解什么是特征，为什么拐角重要等等

解释

你们大多数人都会玩拼图游戏。你会得到很多小图像，需要正确组装它们以形成大的真实图像。问题是，你怎么做？将相同的理论投影到计算机程序上，以便计算机可以玩拼图游戏呢？如果计算机可以玩拼图游戏，为什么我们不能给计算机提供很多自然风光的真实图像，并告诉计算机将所有这些图像拼接成一个大图像呢？如果计算机可以将多个自然图像缝合在一起，那么如何给建筑物或任何结构提供大量图片并告诉计算机从中创建3D模型呢？

好了，问题和想象力还在继续。但这全都取决于最基本的问题：你如何玩拼图游戏？你如何将许多被扰的图像片段排列成一个大的单张图像？你如何将许多自然图像拼接到一张图像上？

答案是，我们正在寻找独特的，易于跟踪和比较的特定模板或特定特征。如果我们对这种特征进行定义，可能会发现很难用语言来表达它，但是我们知道它们是什么。如果有人要求你指出一项可以在多张图像中进行比较的良好特征，则可以指出其中一项。这就是为什么即使是小孩也可以玩这些游戏的原因。我们在图像中搜索这些特征，找到它们，在其他图像中寻找相同的特征并将它们对齐。仅此而已。（在拼图游戏中，我们更多地研究了不同图像的连续性）。所有这些属性都是我们固有的。

因此，我们的一个基本问题扩展到更多，但变得更加具体。这些特征是什么？（答案对于计算机也应该是可以理解的。）

很难说人类如何发现这些特征。这已经在我们的大脑中进行了编码。但是，如果我们深入研究某些图片并搜索不同的模板，我们会发现一些有趣的东西。例如，看以下的图片：

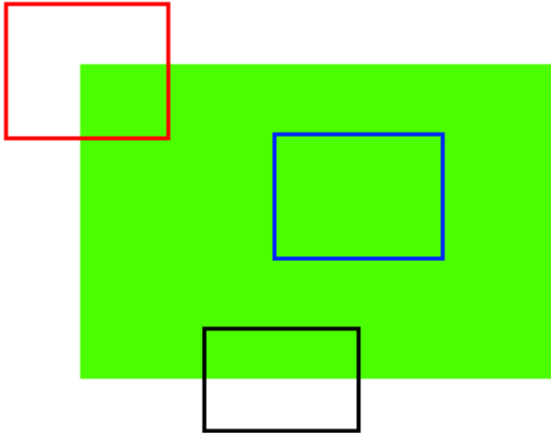


图像非常简单。在图像的顶部，给出了六个小图像块。你的问题是在原始图像中找到这些补丁的确切位置。你可以找到多少正确的结果？

A和B是平坦的表面，它们散布在很多区域上。很难找到这些补丁的确切位置。

C和D更简单。它们是建筑物的边缘。你可以找到一个大概的位置，但是准确的位置仍然很困难。这是因为沿着边缘的每个地方的图案都是相同的。但是，在边缘，情况有所不同。因此，与平坦区域相比，边缘是更好的特征，但不够好（在拼图游戏中比较边缘的连续性很好）。

最后，E和F是建筑物的某些角落。而且很容易找到它们。因为在拐角处，无论将此修补程序移动到何处，它的外观都将有所不同。因此，它们可以被视为很好的特征。因此，现在我们进入更简单（且被广泛使用的图像）以更好地理解。



就像上面一样，蓝色补丁是平坦区域，很难找到和跟踪。无论你将蓝色补丁移到何处，它看起来都一样。黑色补丁有一个边缘。如果你沿垂直方向（即沿渐变）移动它，则它会发生变化。沿着边缘（平行于边缘）移动，看起来相同。对于红色补丁，这是一个角落。无论你将补丁移动到何处，它看起来都不同，这意味着它是唯一的。因此，基本上，拐点被认为是图像中的良好特征。（不仅是角落，在某些情况下，斑点也被认为是不错的功能）。

因此，现在我们回答了我们的问题，“这些特征是什么？”。但是出现了下一个问题。我们如何找到它们？还是我们如何找到角落？我们以一种直观的方式回答了这一问题，即寻找图像中在其周围所有区域中移动（少量）变化最大的区域。在接下来的章节中，这将被投影到计算机语言中。因此，找到这些图像特征称为特征检测。

我们在图像中找到了特征。找到它之后，你应该能够在其他图像中找到相同的图像。怎么做？我们围绕该特征采取一个区域，我们用自己的语言解释它，例如“上部是蓝天，下部是建筑物的区域，在建筑物上有玻璃等”，而你在另一个建筑物中搜索相同的区域图片。基本上，你是在描述特征。同样，计算机还应该描述特征周围的区域，以便可以在其他图像中找到它。所谓的描述称为**特征描述**。获得特征及其描述后，你可以在所有图像中找到相同的功能并将它们对齐，缝合在一起或进行所需的操作。

因此，在此模块中，我们正在寻找OpenCV中的不同算法来查找功能，对其进行描述，进行匹配等。

附加资源

练习

哈里斯角检测

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将了解”Harris Corner Detection”背后的概念。 - 我们将看到以下函数：
`cv.cornerHarris()`, `cv.cornerSubPix()`

理论

在上一章中，我们看到角是图像中各个方向上强度变化很大的区域。**Chris Harris**和**Mike Stephens**在1988年的论文《**组合式拐角和边缘检测器》中做了一次尝试找到这些拐角的尝试，所以现在将其称为哈里斯拐角检测器。他把这个简单的想法变成了数学形式。它基本上找到了(u, v)在所有方向上位移的强度差异。表示如下：**

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \cdot \underbrace{[I(x+u,y+v) - I(x,y)]^2}_{\text{shifted intensity - intensity}}$$

窗口函数要么是一个矩形窗口,要么是高斯窗口,它在下面赋予了值。

我们必须最大化这个函数E(u,v)用于角检测。这意味着,我们必须最大化第二个项。将泰勒扩展应用于上述方程,并使用一些数学步骤(请参考任何你喜欢的标准文本书),我们得到最后的等式:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

其中

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

在此， I_x 和 I_y 分别是在x和y方向上的图像导数。（可以使用**cv.Sobel**()轻松找到）。

然后是主要部分。之后，他们创建了一个分数，基本上是一个等式，它将确定一个窗口是否可以包含一个角。

$$R = \det(M) - k(\text{trace}(M))^2$$

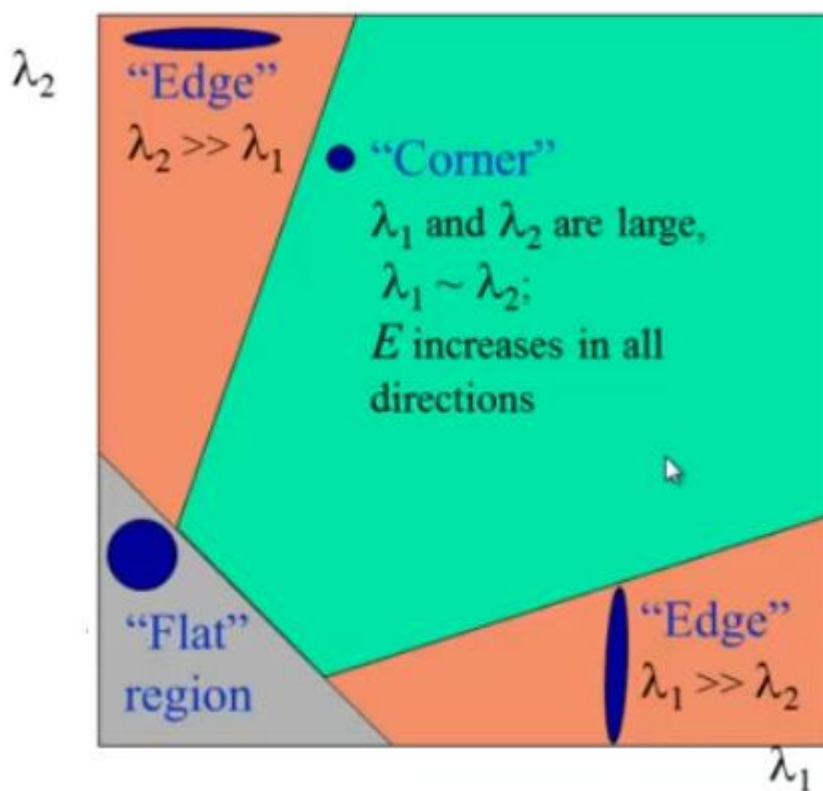
其中

- $\det(M) = \lambda_1 \lambda_2$
- $\text{trace}(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 是 M 的特征值

因此，这些特征值的值决定了区域是拐角，边缘还是平坦。

- 当 $|R|$ 较小，这在 λ_1 和 λ_2 较小时发生，该区域平坦。
- 当 $R < 0$ 时（当 $\lambda_1 \gg \lambda_2$ 时发生，反之亦然），该区域为边。
- 当 R 很大时，这发生在 λ_1 和 λ_2 大且 $\lambda_1 \sim \lambda_2$ 时，该区域是角。

可以用如下图来表示：



因此，Harris Corner Detection的结果是具有这些分数的灰度图像。合适的阈值可为您提供图像的各个角落。我们将以一个简单的图像来完成它。

OpenCV中的哈里斯角检测

为此，OpenCV具有函数**cv.cornerHarris()**。其参数为：- **img** - 输入图像，应为灰度和float32类型。- **blockSize** - 是拐角检测考虑的邻域大小 - **ksize** - 使用的Sobel导数的光圈参数。- **k** - 等式中的哈里斯检测器自由参数。

请参阅以下示例：

```
import numpy as np
import cv2 as cv
filename = 'chessboard.png'
img = cv.imread(filename)
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv.cornerHarris(gray, 2, 3, 0.04)
#result用于标记角点，并不重要
dst = cv.dilate(dst, None)
#最佳值的阈值，它可能因图像而异。
img[dst>0.01*dst.max()]=[0,0,255]
cv.imshow('dst',img)
if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()
```

以下三个结果：

.jpg

SubPixel精度的转角

有时，你可能需要找到最精确的角落。OpenCV附带了一个函数**cv.cornerSubPix**()，它进一步细化了以亚像素精度检测到的角落。下面是一个例子。和往常一样，我们需要先找到哈里斯角。然后我们通过这些角的质心(可能在一个角上有一堆像素，我们取它们的质心)来细化它们。Harris角用红色像素标记，精制角用绿色像素标记。对于这个函数，我们必须定义何时停止迭代的条件。我们在特定的迭代次数或达到一定的精度后停止它，无论先发生什么。我们还需要定义它将搜索角落的邻居的大小。

```
import numpy as np
import cv2 as cv
filename = 'chessboard2.jpg'
img = cv.imread(filename)
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```



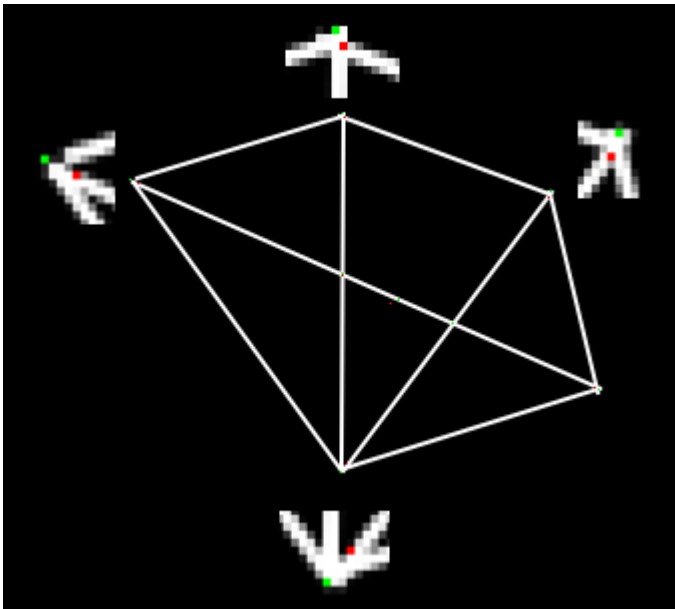
```
# 寻找哈里斯角
gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)
dst = cv.dilate(dst,None)
ret, dst = cv.threshold(dst,0.01*dst.max(),255,0)
dst = np.uint8(dst)

# 寻找质心
ret, labels, stats, centroids = cv.connectedComponentsWithStats(dst)

# 定义停止和完善拐角的条件
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, 0.001)
corners = cv.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

# 绘制
res = np.hstack((centroids,corners))
res = np.int0(res)
img[res[:,1],res[:,0]]=[0,0,255]
img[res[:,3],res[:,2]] = [0,255,0]
cv.imwrite('subpixel5.png',img)
```

以下是结果，其中一些重要位置显示在缩放窗口中以可视化：



附加资源

练习

Shi-tomas拐角检测器和益于跟踪的特征

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将学习另一个拐角检测器：Shi-Tomasi拐角检测器 - 我们将看到以下函数：
cv.goodFeaturesToTrack()

理论

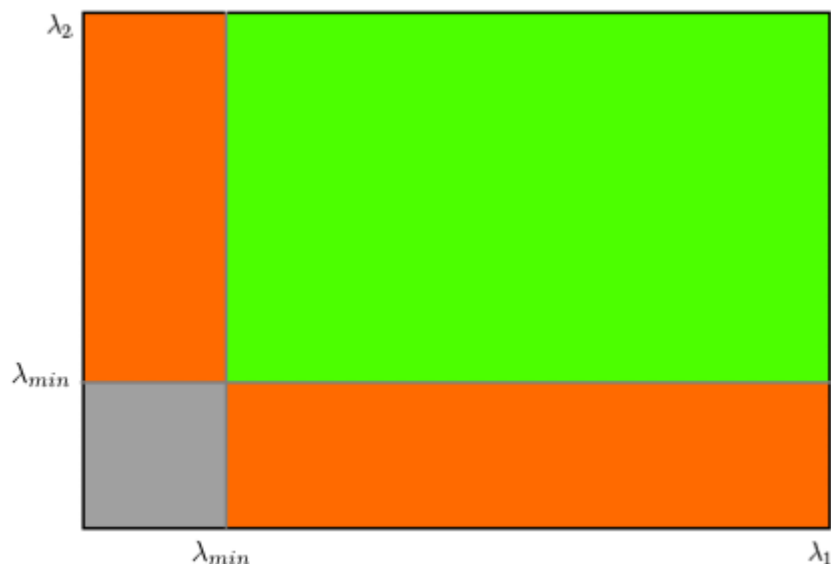
在上一章中，我们看到了Harris Corner Detector。1994年下半年，J. Shi和C. Tomasi在他们的论文《**有益于跟踪的特征**》中做了一个小修改，与Harris Harris Detector相比，显示了更好的结果。哈里斯角落探测器的计分功能由下式给出：

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

取而代之的是，史托马西提出：

$$R = \min(\lambda_1, \lambda_2)$$

如果大于阈值，则将其视为拐角。如果像在Harris Corner Detector中那样在 λ_1 - λ_2 空间中绘制它，则会得到如下图像：



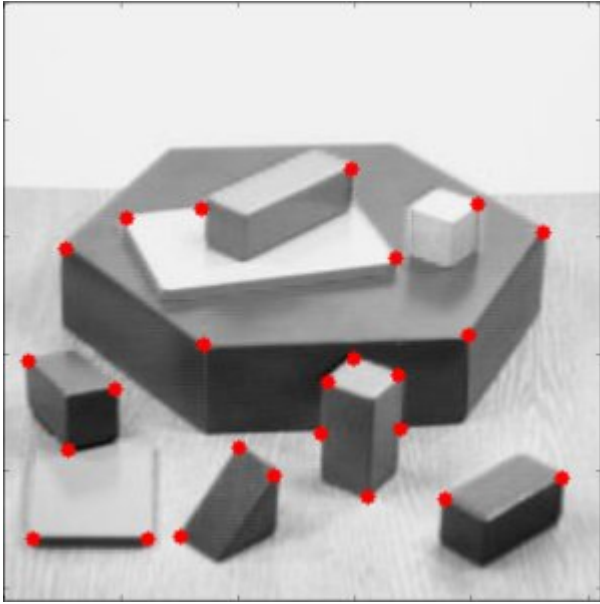
从图中可以看到，只有当 λ_1 和 λ_2 大于最小值 $\lambda_{\min}$ 时，才将其视为拐角（绿色区域）。

代码

OpenCV有一个函数**`cv.goodFeaturesToTrack`**()。它通过Shi-Tomasi方法（或哈里斯角检测，如果指定）找到图像中的N个最强角。像往常一样，图像应该是灰度图像。然后，指定要查找的角数。然后，您指定质量级别，该值是介于0-1之间的值，该值表示每个角落都被拒绝的最低拐角质量。然后，我们提供检测到的角之间的最小欧式距离。利用所有这些信息，该功能可以找到图像中的拐角。低于平均质量的所有拐角点均被拒绝。然后，它会根据质量以降序对剩余的角进行排序。然后函数首先获取最佳拐角，然后丢弃最小距离范围内的所有附近拐角，然后返回N个最佳拐角。在下面的示例中，我们将尝试找到25个最佳弯角：

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('blox.jpg')
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
corners = cv.goodFeaturesToTrack(gray,25,0.01,10)
corners = np.int0(corners)
for i in corners:
    x,y = i.ravel()
    cv.circle(img,(x,y),3,255,-1)
plt.imshow(img),plt.show()
```

查看以下结果：



此功能更适合跟踪。我们将看到使用它的时机

额外资源

练习

SIFT尺度不变特征变换

作者|OpenCV-Python Tutorials

编译|Vincent

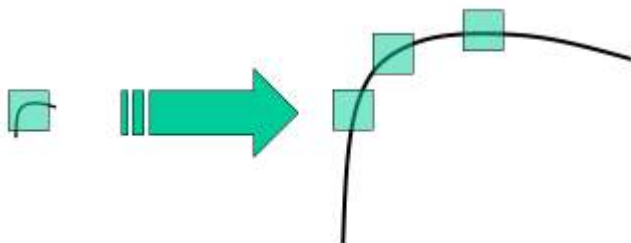
来源|OpenCV-Python Tutorials

目标

在这一章当中， - 我们将学习SIFT算法的概念 - 我们将学习找到SIFT关键点和描述算符。

理论

在前两章中，我们看到了一些像Harris这样的拐角检测器。它们是旋转不变的，这意味着即使图像旋转了，我们也可以找到相同的角。很明显，因为转角在旋转的图像中也仍然是转角。但是缩放呢？如果缩放图像，则拐角可能不是角。例如，检查下面的简单图像。在同一窗口中放大小窗口中小图像中的拐角时，该角是平坦的。因此，Harris拐角不是尺度不变的。



因此，在2004年，不列颠哥伦比亚大学的**D.Lowe**在他的论文《**尺度不变关键点中的独特图像特征**》中提出了一种新算法，即尺度不变特征变换(SIFT)，该算法提取关键点并计算其描述算符。
(改论文易于理解，被认为是学习SIFT的最佳材料。因此，本文只是该论文的简短摘要)。 SIFT算法主要包括四个步骤。我们将一一看到它们。

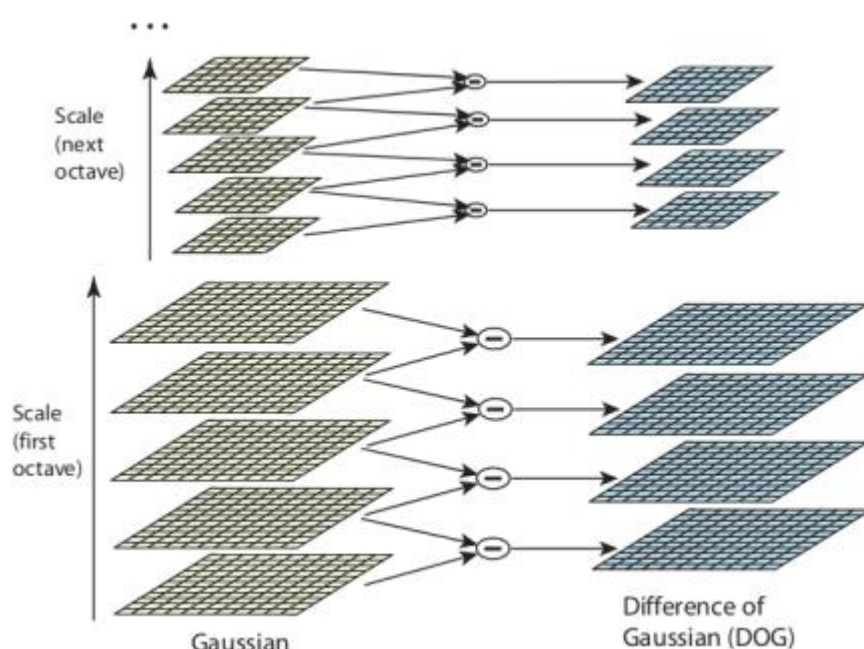
SIFT算法主要包括四个步骤。我们将一一看到它们。

1. 尺度空间极值检测

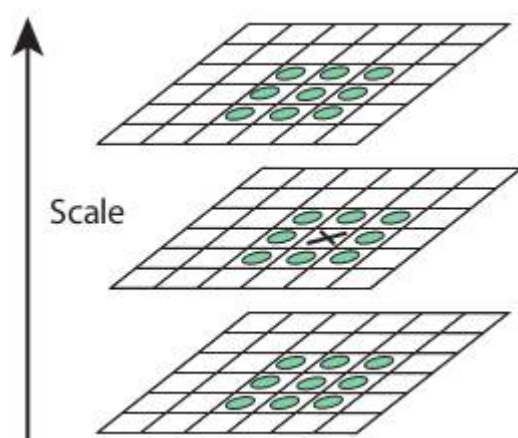
从上图可以明显看出，我们不能使用相同的窗口来检测具有不同比例的关键点。即便小拐角可以。但是要检测更大的拐角，我们将需要更大的窗口。为此，使用了比例空间滤波。在其中，找

到具有各种 σ 值的图像的高斯拉普拉斯算子。LoG用作斑点检测器，可检测由于 σ 的变化而导致的各种大小的斑点。简而言之， σ 用作缩放参数。例如，在上图中，低 σ 的高斯核对于较小的拐角给出较高的值，而高 σ 的高斯核对于较大的拐角而言非常合适。因此，我们可以找到整个尺度和空间上的局部最大值，这给了我们 (x,y,σ) 值的列表，这意味着在 (x,y) 在 σ 尺度上有一个潜在的关键点。

但是这种LoG代价昂贵，因此SIFT算法使用的是高斯差值，它是LoG的近似值。高斯差是作为具有两个不同 σ 的图像的高斯模糊差而获得的，设为 σ 和 $k\sigma$ 。此过程是针对高斯金字塔中图像的不同八度完成的。如下图所示：



一旦找到该DoG，便会在图像上搜索比例和空间上的局部极值。例如，将图像中的一个像素与其8个相邻像素以及下一个比例的9个像素和前一个比例的9个像素进行比较。如果是局部极值，则可能是关键点。从根本上说，关键点是最好的代表。如下图所示：



对于不同的参数，本文给出了一些经验数据，可以概括为：octaves=4，缩放尺度=5，初始 $\sigma=1.6$ ， $k=\sqrt{2}$ 等作为最佳值。

2. 关键点定位

一旦找到潜在的关键点位置，就必须对其进行优化以获取更准确的结果。他们使用了标度空间的泰勒级数展开来获得更精确的极值位置，如果该极值处的强度小于阈值（根据论文为0.03），则将其拒绝。在OpenCV DoG中，此阈值称为**ContrastThreshold**，它对边缘的响应较高，因此也需要删除边缘。

为此，使用类似于哈里斯拐角检测器的概念。他们使用2x2的Hessian矩阵(H)计算主曲率。从哈里斯拐角检测器我们知道，对于边缘，一个特征值大于另一个特征值。因此，这里他们使用了一个简单的函数。

如果该比率大于一个阈值（在OpenCV中称为**edgeThreshold**），则该关键点将被丢弃。论文上写的值为10。

因此，它消除了任何低对比度的关键点和边缘关键点，剩下的就是很可能的目标点。

3. 方向分配

现在，将方向分配给每个关键点，以实现图像旋转的不变性。根据比例在关键点位置附近采取邻域，并在该区域中计算梯度大小和方向。创建了一个具有36个覆盖360度的bin的方向直方图（通过梯度幅度和 σ 等于关键点比例的1.5的高斯加权圆窗加权）。提取直方图中的最高峰，并且将其超过80%的任何峰也视为计算方向。它创建的位置和比例相同但方向不同的关键点。它有助于匹配的稳定性。

4. 关键点描述

现在创建了关键点描述符。在关键点周围采用了16x16的邻域。它分为16个4x4大小的子块。对于每个子块，创建8 bin方向直方图。因此共有128个bin值可用。它被表示为形成关键点描述符的向量。除此之外，还采取了几种措施来实现针对照明变化，旋转等的鲁棒性。

5. 关键点匹配

通过识别两个图像的最近邻，可以匹配两个图像之间的关键点。但是在某些情况下，第二个最接近的匹配可能非常接近第一个。它可能是由于噪音或其他原因而发生的。在那种情况下，采用最接近距离与第二最接近距离之比。如果大于0.8，将被拒绝。根据论文，它可以消除大约90%的错误匹配，而仅丢弃5%的正确匹配。因此，这是SIFT算法的总结。有关更多详细信息和理解，强烈建议阅读原始论文。记住一件事，该算法已申请专利。所以这个算法包含在opencv contrib repo中

OpenCV中的SIFT

现在，让我们来看一下OpenCV中可用的SIFT功能。让我们从关键点检测开始并进行绘制。首先，我们必须构造一个SIFT对象。我们可以将不同的参数传递给它，这些参数是可选的，它们在docs中已得到很好的解释。

```
import numpy as np
import cv2 as cv
img = cv.imread('home.jpg')
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp = sift.detect(gray,None)
img=cv.drawKeypoints(gray,kp,img)
cv.imwrite('sift_keypoints.jpg',img)
```

sift.detect()函数在图像中找到关键点。如果只想搜索图像的一部分，则可以通过掩码。每个关键点是一个特殊的结构，具有许多属性，例如其(x, y)坐标，有意义的邻域的大小，指定其方向的角度，指定关键点强度的响应等。

OpenCV还提供**cv.drawKeyPoints()**函数，该函数在关键点的位置绘制小圆圈。如果将标志**cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS**传递给它，它将绘制一个具有关键点大小的圆，甚至会显示其方向。请参见以下示例。

```
img=cv.drawKeypoints(gray,kp,img,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv.imwrite('sift_keypoints.jpg',img)
```




查看下面的结果：

现在要计算描述符，OpenCV提供了两种方法。1. 由于已经找到关键点，因此可以调用 `**sift.compute**()`，该函数根据我们找到的关键点来计算描述符。例如：`kp, des = sift.compute(gray, kp)` 2. 如果找不到关键点，则可以使用 `**sift.detectAndCompute**()` 函数在单步骤中直接找到关键点和描述符。

我们将看到第二种方法：

```
sift = cv.xfeatures2d.SIFT_create()
kp, des = sift.detectAndCompute(gray, None)
```

这里的kp将是一个关键点列表，而des是一个形状为Number_of_Keypoints×128的数字数组。

这样我们得到了关键点，描述符等。现在我们想看看如何在不同图像中匹配关键点。我们将在接下来的章节中学习。

附加资源

练习

SURF简介 (加速的强大功能)

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

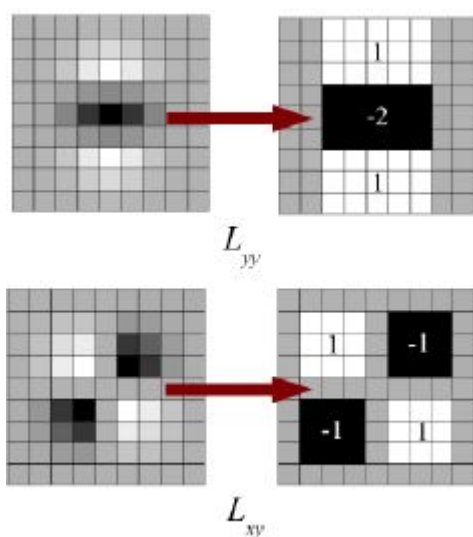
目标

在这一章当中， - 我们将了解SURF的基础 - 我们将在OpenCV中看到SURF函数

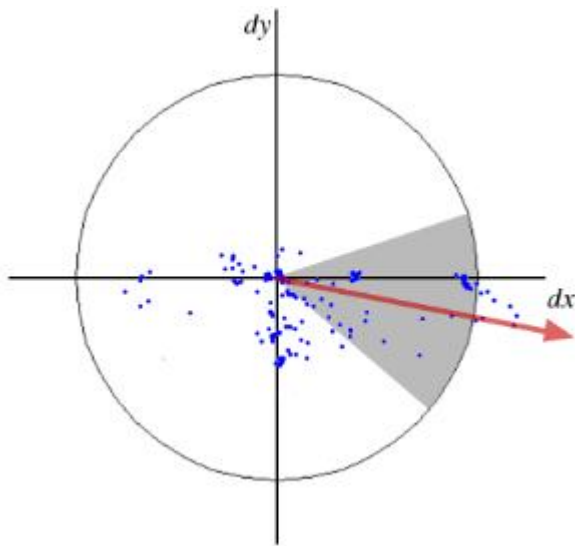
理论

在上一章中,我们看到了SIFT用于关键点检测和描述符。但相对缓慢,人们需要更多的加速版本。2006年,三个人,H .Tuytelaars,T. and Van Gool,L,发表了另一篇论文,“SURF:加速健壮的特征”,引入了一种名为“SURF”的新算法。正如名字所表明的那样,它是一个加速版本的SIFT。

在SIFT中，Lowe用高斯差近似高斯的拉普拉斯算子来寻找尺度空间。SURF走得更远，使用Box Filter近似LoG。下图显示了这种近似值的演示。这种近似的一大优点是，借助积分图像可以轻松地计算出带盒滤波器的卷积。并且可以针对不同规模并行执行。SURF还依赖于Hessian矩阵的行列式来确定尺度和位置。



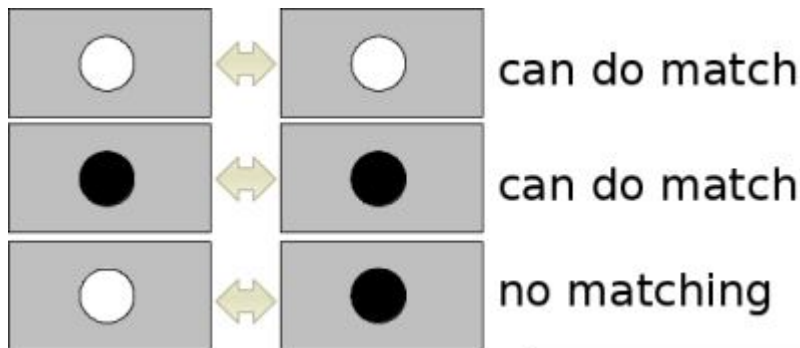
对于方向分配，SURF在水平和垂直方向上对大小为 $6s$ 的邻域使用小波响应。适当的高斯权重也适用于它。然后将它们绘制在下图所示的空间中。通过计算角度为 60° 的滑动方向窗口内所有响应的总和，可以估算主导方向。有趣的是，小波响应可以很容易地使用积分图像在任何规模下发现。对于许多应用，不需要旋转不变性，因此无需查找此方向，从而加快了过程。SURF提供了称为Upright-SURF或U-SURF的功能。它提高了速度，并具有高达 $\pm 15^\circ$ 的鲁棒性。OpenCV根据标志支持两种方式。如果为0，则计算方向。如果为1，则不计算方向并且速度更快。



对于功能描述，SURF在水平和垂直方向上使用小波响应（同样，使用积分图像使事情变得更容易）。在 s 是大小的关键点周围采用大小为 $20s \times 20s$ 的邻域。它分为 4×4 子区域。对于每个子区域，获取水平和垂直小波响应，并像这样形成向量， $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$ 。当表示为向量时，这将为SURF特征描述符提供总共64个维度。尺寸越小，计算和匹配速度越快，但特征的区分性更好。

为了更加独特，SURF特征描述符具有扩展的128维版本。 dx 和 $|dx|$ 的和分别针对 $dy < 0$ 和 $dy \geq 0$ 进行计算。同样， dy 和 $|dy|$ 的总和根据 dx 的符号进行拆分，从而使特征数量加倍。它不会增加太多的计算复杂性。OpenCV通过将分别为64-dim和128-dim（默认值为128-dim）的标志的值设置为0和1来支持这两者（默认为128-dim）。

另一个重要的改进是对基础兴趣点使用了Laplacian算符（海森矩阵的迹）。它不增加计算成本，因为它已在检测期间进行了计算。拉普拉斯算子的标志将深色背景上的明亮斑点与相反的情况区分开。在匹配阶段，我们仅比较具有相同对比类型的特征（如下图所示）。这些最少的信息可加快匹配速度，而不会降低描述符的性能。



简而言之，SURF添加了许多功能来提高每一步的速度。分析表明，它的速度是SIFT的3倍，而性能却与SIFT相当。SURF擅长处理具有模糊和旋转的图像，但不擅长处理视点变化和照明变化。

OpenCV中的SURF

OpenCV提供类似于SIFT的SURF功能。您可以使用一些可选条件（例如64 / 128-dim描述符，Upright / Normal SURF等）来启动SURF对象。所有详细信息在docs中都有详细说明。然后，就像在SIFT中所做的那样，我们可以使用SURF.detect()，SURF.compute()等来查找关键点和描述符。

首先，我们将看到一个有关如何找到SURF关键点和描述符并进行绘制的简单演示。所有示例都在Python终端中显示，因为它与SIFT相同。

```
>>> img = cv.imread('fly.png',0)
# 创建SURF对象。你可以在此处或以后指定参数。
# 这里设置海森矩阵的阈值为400
>>> surf = cv.xfeatures2d.SURF_create(400)
# 直接查找关键点和描述符
>>> kp, des = surf.detectAndCompute(img, None)
>>> len(kp)
699
```

图片中无法显示1199个关键点。我们将其减少到50左右以绘制在图像上。匹配时，我们可能需要所有这些功能，但现在不需要。因此，我们增加了海森阈值。

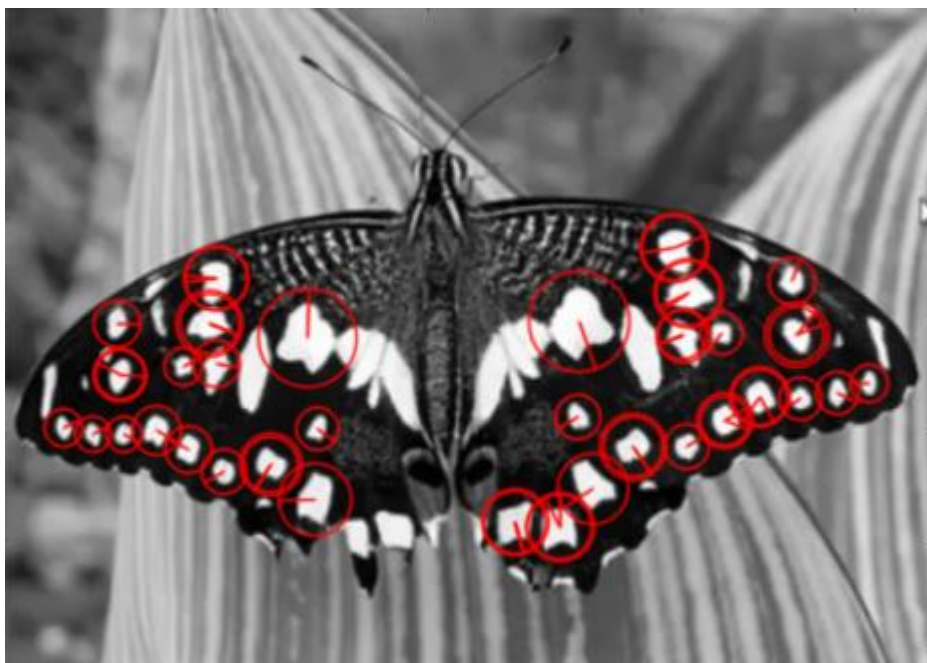
```
# 检查海森矩阵阈值
>>> print( surf.getHessianThreshold() )
400.0
# 我们将其设置为50000。记住，它仅用于表示图片。
# 在实际情况下，最好将值设为300-500
>>> surf.setHessianThreshold(50000)
# 再次计算关键点并检查其数量。
```

```
>>> kp, des = surf.detectAndCompute(img, None)
>>> print( len(kp) )
47
```

它小于50。让我们在图像上绘制它。

```
>>> img2 = cv.drawKeypoints(img, kp, None, (255, 0, 0), 4)
>>> plt.imshow(img2), plt.show()
```

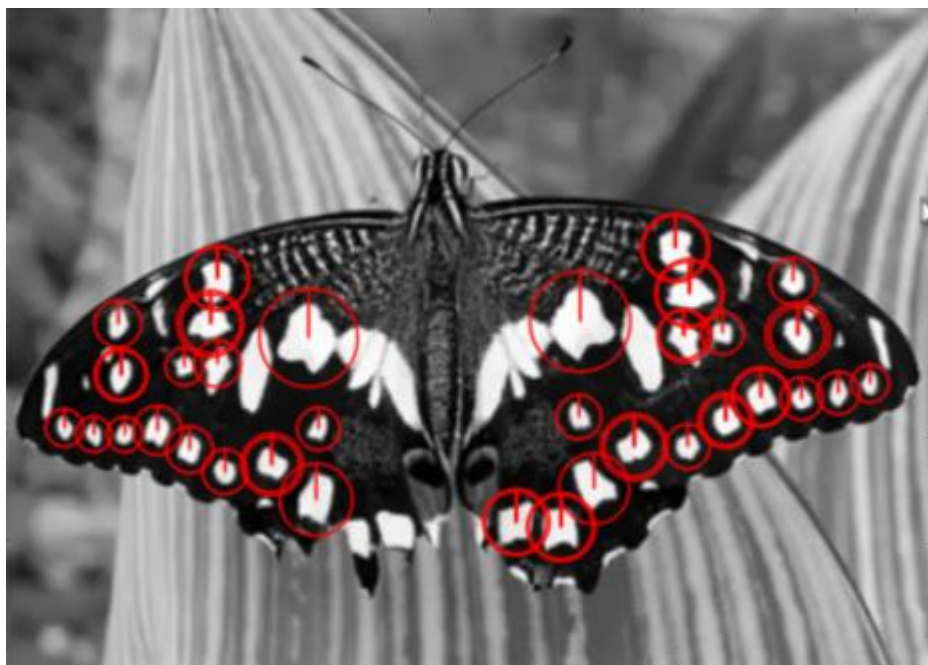
请参阅下面的结果。您可以看到SURF更像是斑点检测器。它检测到蝴蝶翅膀上的白色斑点。您可以使用其他图像进行测试。



现在，我想应用U-SURF，以便它不会找到方向。

```
# 检查flag标志，如果为False，则将其设置为True
>>> print( surf.getUpright() )
False
>>> surf.setUpright(True)
# 重新计算特征点并绘制
>>> kp = surf.detect(img, None)
>>> img2 = cv.drawKeypoints(img, kp, None, (255, 0, 0), 4)
>>> plt.imshow(img2), plt.show()
```

参见下面的结果。所有的方向都显示在同一个方向上。它比以前更快了。如果你工作的情况下，方向不是一个问题(如全景拼接)等，这是更好的。



最后，我们检查描述符的大小，如果只有64维，则将其更改为128。

```
# 找到算符的描述
>>> print( surf.descriptorSize() )
64
# 表示flag "extened" 为False。
>>> surf.getExtended()
False
# 因此，将其设为True即可获取128个尺寸的描述符。
>>> surf.setExtended(True)
>>> kp, des = surf.detectAndCompute(img, None)
>>> print( surf.descriptorSize() )
128
>>> print( des.shape )
(47, 128)
```

其余部分是匹配的，我们将在另一章中进行匹配。

附加资源

练习

用于角点检测的FAST算法

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，- 我们将了解FAST算法的基础知识。- 我们将使用OpenCV功能对FAST算法进行探索。

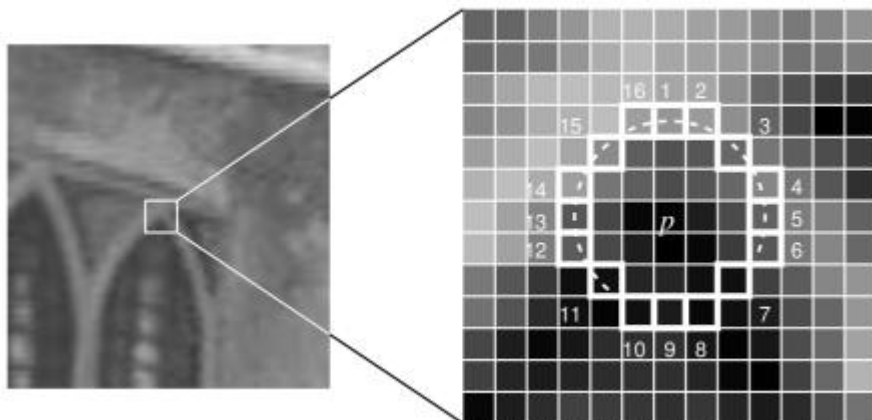
理论

我们看到了几个特征检测器，其中很多真的很棒。但是，从实时应用程序的角度来看，它们不够快。最好的例子是计算资源有限的SLAM（同时定位和制图）移动机器人

作为对此的解决方案，Edward Rosten和Tom Drummond在2006年的论文“用于高速拐角检测的机器学习”中提出了FAST（加速分段测试的特征）算法（后来在2010年对其进行了修订）。该算法的基本内容如下。有关更多详细信息，请参阅原始论文（所有图像均取自原始论文）。

使用FAST进行特征检测

1. 选择图像中是否要识别为兴趣点的像素 p ，使其强度为 I_p
2. 选择适当的阈值 t
3. 考虑被测像素周围有16个像素的圆圈。（见下图）



4. 现在，如果圆中存在一组（共16个像素） n 个连续的像素，它们均比 $I_p + t$ 亮，或者比 $I_p - t$ 都暗，则像素 p 是一个角。（在上图中显示为白色虚线）。 n 被选为12。
5. 建议使用高速测试以排除大量的非角区域。此测试仅检查1、9、5和13处的四个像素（如果第一个1和9太亮或太暗，则对其进行测试。如果是，则检查5和13）。如果 p 是一个角，则其中至少三个必须全部比 $I_p + t$ 亮或比 $I_p - t$ 暗。如果以上两种情况都不是，则 p 不能为角。然后，可以通过检查圆中的所有像素，将完整的分段测试标准应用于通过的候选项。该检测器本身具有很高的性能，但有几个缺点：
6. 它不会拒绝 $n < 12$ 的候选对象。
7. 像素的选择不是最佳的，因为其效率取决于问题的顺序和角落外观的分布。
8. 高速测试的结果被丢弃了。
9. 彼此相邻地检测到多个特征。

机器学习的方法解决了前三点。使用非最大抑制来解决最后一个问题。

让机器学习一个角检测器

1. 选择一组图像进行训练（最好从目标应用程序域中进行训练）
2. 在每个图像中运行FAST算法以查找特征点。
3. 对于每个特征点，将其周围的16个像素存储为矢量。对所有图像执行此操作以获得特征向量 P 。
4. 这16个像素中的每个像素（例如 x ）可以具有以下三种状态之一：

$$S_{p \rightarrow x} = \begin{cases} d, & I_{p \rightarrow x} \leq I_p - t & \text{(darker)} \\ s, & I_p - t < I_{p \rightarrow x} < I_p + t & \text{(similar)} \\ b, & I_p + t \leq I_{p \rightarrow x} & \text{(brighter)} \end{cases}$$

1. 取决于这些状态，特征矢量 P 被细分为3个子集， P_d, P_s, P_b 。
2. 定义一个新的布尔变量 K_p ，如果 p 是一个角，则为true，否则为false。
3. 使用ID3算法（决策树分类器）使用变量 K_p 查询每个子集，以获取有关真实类的知识。它选择 x ，该 x 通过 K_p 的熵测得的有关候选像素是否为角的信息最多。
4. 递归地将其应用于所有子集，直到其熵为零为止。

5. 这样创建的决策树用于其他图像的快速检测。

非最大抑制

在相邻位置检测多个兴趣点是另一个问题。通过使用非极大抑制来解决。

1. 计算所有检测到的特征点的得分函数V。V是p与16个周围像素值之间的绝对差之和。
2. 考虑两个相邻的关键点并计算它们的V值。
3. 丢弃较低V值的那个。

总结

它比其他现有的拐角检测器快几倍。

但是它对高水平的噪声并不鲁棒。它取决于阈值。

OpenCV中的高速拐角检测器

它被称为OpenCV中的任何其他特征检测器。 如果需要，您可以指定阈值，是否要应用非极大抑制，要使用的邻域等。 对于邻域，定义了三个标志，分别为

`cv.FAST_FEATURE_DETECTOR_TYPE_5_8`，`cv.FAST_FEATURE_DETECTOR_TYPE_7_12` 和 `cv.FAST_FEATURE_DETECTOR_TYPE_9_16`。 以下是有关如何检测和绘制FAST特征点的简单代码。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('simple.jpg',0)
# 用默认值初始化FAST对象
fast = cv.FastFeatureDetector_create()
# 寻找并绘制关键点
kp = fast.detect(img, None)
img2 = cv.drawKeypoints(img, kp, None, color=(255,0,0))
# 打印所有默认参数
print( "Threshold: {}".format(fast.getThreshold()) )
print( "nonmaxSuppression:{}".format(fast.getNonmaxSuppression()) )
print( "neighborhood: {}".format(fast.getType()) )
print( "Total Keypoints with nonmaxSuppression: {}".format(len(kp)) )
cv.imwrite('fast_true.png',img2)
# 关闭非极大抑制
fast.setNonmaxSuppression(0)
kp = fast.detect(img, None)
print( "Total Keypoints without nonmaxSuppression: {}".format(len(kp)) )
```

```
img3 = cv.drawKeypoints(img, kp, None, color=(255,0,0))  
cv.imwrite('fast_false.png',img3)
```

查看结果。第一张图片显示了带有nonmaxSuppression的FAST，第二张图片显示了没有nonmaxSuppression的FAST：

附加资源

1. Edward Rosten and Tom Drummond, “Machine learning for high speed corner detection” in 9th European Conference on Computer Vision, vol. 1, 2006, pp. 430–443.
2. Edward Rosten, Reid Porter, and Tom Drummond, “Faster and better: a machine learning approach to corner detection” in IEEE Trans. Pattern Analysis and Machine Intelligence, 2010, vol 32, pp. 105-119.

练习

BRIEF(二进制的鲁棒独立基本特征)

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将看到BRIEF算法的基础知识

理论

我们知道SIFT使用128维矢量作为描述符。由于它使用浮点数，因此基本上需要512个字节。同样，SURF最少也需要256个字节（用于64像素）。为数千个功能部件创建这样的向量会占用大量内存，这对于资源受限的应用程序尤其是嵌入式系统而言是不可行的。内存越大，匹配所需的时间越长。

但是实际匹配可能不需要所有这些尺寸。我们可以使用PCA，LDA等几种方法对其进行压缩。甚至使用LSH（局部敏感哈希）进行哈希的其他方法也可以将这些SIFT描述符中的浮点数转换为二进制字符串。这些二进制字符串用于使用汉明距离匹配要素。这提供了更快的速度，因为查找汉明距离仅是应用XOR和位数，这在具有SSE指令的现代CPU中非常快。但是在这里，我们需要先找到描述符，然后才可以应用散列，这不能解决我们最初的内存问题。

现在介绍BRIEF。它提供了一种直接查找二进制字符串而无需查找描述符的快捷方式。它需要平滑的图像补丁，并以独特的方式（在纸上展示）选择一组 $n_d(x, y)$ 位置对。然后，在这些位置对上进行一些像素强度比较。例如，令第一位置对为 p 和 q 。如果 $I(p) < I(q)$ ，则结果为1，否则为0。将其应用于所有 n_d 个位置对以获得 n_d 维位串。

该 n_d 可以是128、256或512。OpenCV支持所有这些，但默认情况下将是256（OpenCV以字节为单位表示，因此值将为16、32和64）。因此，一旦获得此信息，就可以使用汉明距离来匹配这些描述符。

重要的一点是，BRIEF是特征描述符，它不提供任何查找特征的方法。因此，您将不得不使用任何其他特征检测器，例如SIFT，SURF等。本文建议使用CenSurE，它是一种快速检测器，并且BIM对于CenSurE点的工作原理甚至比对SURF点的工作要好一些。

简而言之，BRIEF是一种更快的方法特征描述符计算和匹配。除了平面内旋转较大的情况，它将提供很高的识别率。

OpenCV中的BRIEF

下面的代码显示了借助CenSurE检测器对Brief描述符的计算。（在OpenCV中，CenSurE检测器称为STAR检测器）注意，您需要使用opencv contrib）才能使用它。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('simple.jpg',0)
# 初始化FAST检测器
star = cv.xfeatures2d.StarDetector_create()
# 初始化BRIEF提取器
brief = cv.xfeatures2d.BriefDescriptorExtractor_create()
# 找到STAR的关键点
kp = star.detect(img, None)
# 计算BRIEF的描述符
kp, des = brief.compute(img, kp)
print( brief.descriptorSize() )
print( des.shape )
```

函数 `brief.getDescriptorSize()` 给出以字节为单位的n_d大小。默认情况下为32。下一个是匹配项，这将在另一章中进行。

附加资源

1. Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, “BRIEF: Binary Robust Independent Elementary Features”, 11th European Conference on Computer Vision (ECCV), Heraklion, Crete. LNCS Springer, September 2010.
2. [LSH \(Locality Sensitive Hashing\)](#) at wikipedia.

ORB(面向快速和旋转的BRIEF)

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中，- 我们将了解ORB的基础知识

理论

作为OpenCV的狂热者，关于ORB的最重要的事情是它来自“OpenCV Labs”。该算法由Ethan Rublee，Vincent Rabaud，Kurt Konolige和Gary R. Bradski在其论文《**ORB：SIFT或SURF的有效替代方案**》中提出。2011年，正如标题所述，它是计算中SIFT和SURF的良好替代方案成本，匹配性能以及主要是专利。是的，SIFT和SURF已获得专利，你应该为其使用付费。但是ORB不是！！！！

ORB基本上是FAST关键点检测器和Brief描述符的融合，并进行了许多修改以增强性能。首先，它使用FAST查找关键点，然后应用Harris角测度在其中找到前N个点。它还使用金字塔生成多尺度特征。但是一个问题是，FAST无法计算方向。那么旋转不变性呢？作者提出以下修改。

它计算角点位于中心的贴片的强度加权质心。从此角点到质心的矢量方向确定了方向。为了改善旋转不变性，使用x和y计算矩，它们应该在半径r的圆形区域中，其中r是斑块的大小。

现在，对于描述符，ORB使用Brief描述符。但是我们已经看到，BRIEF的旋转性能很差。因此，ORB所做的就是根据关键点的方向“引导”BRIEF。对于位置 (x_i, y_i) 上n个二进制测试的任何特征集，定义一个 $2 \times n$ 矩阵S，其中包含这些像素的坐标。然后使用面片的方向 θ ，找到其旋转矩阵并旋转S以获得转向（旋转）版本 S_θ 。

ORB将角度离散化为 $\frac{2\pi}{30}$ （12度）的增量，并构造了预先计算的Brief模式的查找表。只要关键点方向 θ 在各个视图中一致，就将使用正确的点集 S_θ 来计算其描述符。

BRIEF具有一个重要的特性，即每个位特征具有较大的方差，且均值接近0.5。但是，一旦沿关键点方向定向，它就会失去此属性，变得更加分散。高方差使功能更具区分性，因为它对输入的响

应不同。另一个理想的特性是使测试不相关，因为从那时起每个测试都会对结果有所贡献。为了解决所有这些问题，ORB在所有可能的二进制测试中进行贪婪搜索，以找到方差高且均值接近0.5且不相关的测试。结果称为rBRIEF。

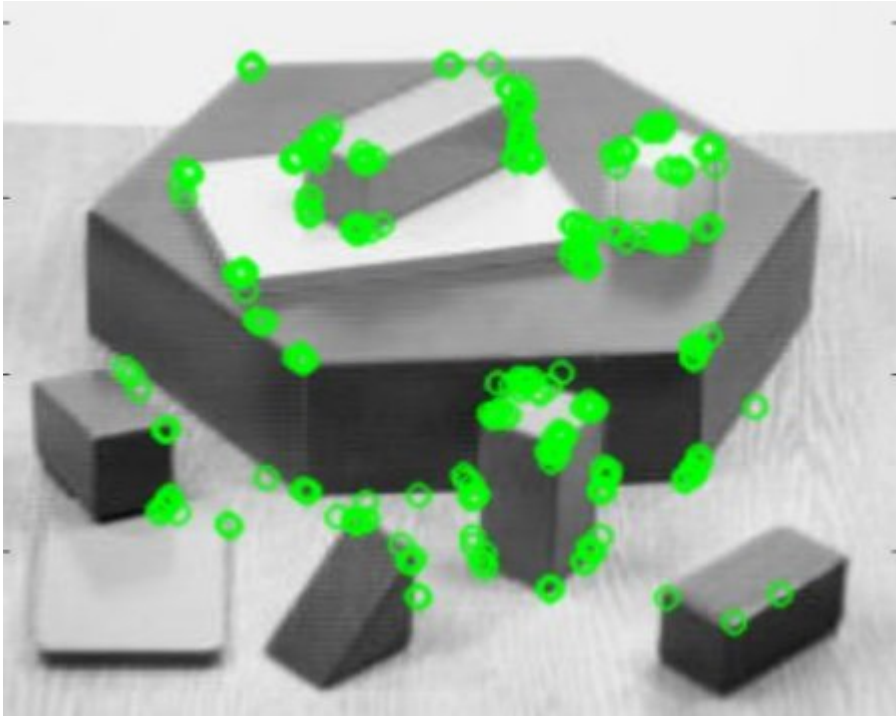
对于描述符匹配，使用了对传统LSH进行改进的多探针LSH。该论文说，ORB比SURF快得多，而SIFT和ORB描述符比SURF更好。在全景拼接等低功耗设备中，ORB是一个不错的选择。

OpenCV中的ORB

与往常一样，我们必须使用函数`cv.ORB`或使用`feature2d`通用接口来创建ORB对象。它具有许多可选参数。最有用的是`nFeatures`，它表示要保留的最大特征数（默认为500），`scoreType`表示是对特征进行排名的Harris分数还是FAST分数（默认为Harris分数）等。另一个参数`WTA_K`决定点数产生定向的BRIEF描述符的每个元素。默认情况下为两个，即一次选择两个点。在这种情况下，为了匹配，将使用`NORM_HAMMING`距离。如果`WTA_K`为3或4，则需要3或4个点来生成Brief描述符，则匹配距离由`NORM_HAMMING2`定义。下面是显示ORB用法的简单代码。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('simple.jpg', 0)
# 初始化ORB检测器
orb = cv.ORB_create()
# 用ORB寻找关键点
kp = orb.detect(img, None)
# 用ORB计算描述符
kp, des = orb.compute(img, kp)
# 仅绘制关键点的位置，而不绘制大小和方向
img2 = cv.drawKeypoints(img, kp, None, color=(0, 255, 0), flags=0)
plt.imshow(img2), plt.show()
```

查看以下结果：



ORB特征匹配，我们将在另一章中进行。

附加资源

1. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary R. Bradski: ORB: An efficient alternative to SIFT or SURF. ICCV 2011: 2564-2571.

练习

特征匹配

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中， - 我们将看到如何将一个图像中的特征与其他图像进行匹配。 - 我们将在OpenCV中使用Brute-Force匹配器和FLANN匹配器

Brute-Force匹配器的基础

蛮力匹配器很简单。它使用第一组中一个特征的描述符，并使用一些距离计算将其与第二组中的所有其他特征匹配。并返回最接近的一个。

对于BF匹配器，首先我们必须使用`cv.BFMatcher()`创建BFMatcher对象。它需要两个可选参数。第一个是normType，它指定要使用的距离测量。默认情况下为 `cv.NORM_L2`。对于SIFT，SURF等（也有 `cv.NORM_L1`）很有用。对于基于二进制字符串的描述符，例如ORB，BRIEF，BRISK等，应使用 `cv.NORM_HAMMING`，该函数使用汉明距离作为度量。如果ORB使用 `WTA_K == 3` 或 `4`，则应使用`cv.NORM_HAMMING2`。

第二个参数是布尔变量，即crossCheck，默认情况下为false。如果为true，则Matcher仅返回具有值(i, j)的那些匹配项，以使集合A中的第i个描述符具有集合B中的第j个描述符为最佳匹配，反之亦然。即两组中的两个特征应彼此匹配。它提供了一致的结果，并且是D.Lowe在SIFT论文中提出的比率测试的良好替代方案。

创建之后，两个重要的方法是`*BFMatcher.match*`()和`*BFMatcher.knnMatch*`()。第一个返回最佳匹配。第二种方法返回k个最佳匹配，其中k由用户指定。当我们需要对此做其他工作时，它可能会很有用。

就像我们使用`cv.drawKeypoints()`绘制关键点一样，`cv.drawMatches()`可以帮助我们绘制匹配项。它水平堆叠两张图像，并绘制从第一张图像到第二张图像的线，以显示最佳匹配。还有

`cv.drawMatchesKnn`绘制所有k个最佳匹配。如果 `k=2`，它将为每个关键点绘制两条匹配线。因此，如果要选择性地绘制，则必须通过掩码。

让我们来看一个SIFT和ORB的示例（两者都使用不同的距离测量）。

使用ORB描述符进行Brute-Force匹配

在这里，我们将看到一个有关如何在两个图像之间匹配特征的简单示例。在这种情况下，我有一个queryImage和trainImage。我们将尝试使用特征匹配在trainImage中找到queryImage。（图像是/samples/data/box.png和/samples/data/box_in_scene.png）

我们正在使用ORB描述符来匹配特征。因此，让我们从加载图像，查找描述符等开始。

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('box.png', cv.IMREAD_GRAYSCALE) # 索引图像
img2 = cv.imread('box_in_scene.png', cv.IMREAD_GRAYSCALE) # 训练图像
# 初始化ORB检测器
orb = cv.ORB_create()
# 基于ORB找到关键点和检测器
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)
```

接下来，我们创建一个距离测量值为**`cv.NORM_HAMMING`**的BFMatcher对象（因为我们使用的是ORB），并且启用了CrossCheck以获得更好的结果。然后，我们使用Matcher.match()方法来获取两个图像中的最佳匹配。我们按照距离的升序对它们进行排序，以使最佳匹配（低距离）排在前面。然后我们只抽出前10的匹配（只是为了提高可见度。您可以根据需要增加它）

```
# 创建BF匹配器的对象
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True) # 匹配描述符
matches = bf.match(des1, des2) # 根据距离排序
matches = sorted(matches, key = lambda x:x.distance) # 绘制前10的匹配项
img3 = cv.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(img3), plt.show()
```



将获得以下的结果：

什么是Matcher对象？

`matches = bf.match(des1,des2)` 行的结果是DMatch对象的列表。该DMatch对象具有以下属性：

- DMatch.distance-描述符之间的距离。越低越好。
- DMatch.trainIdx-火车描述符中的描述符索引
- DMatch.queryIdx-查询描述符中的描述符索引
- DMatch.imgIdx-火车图像的索引。

带有SIFT描述符和比例测试的Brute-Force匹配

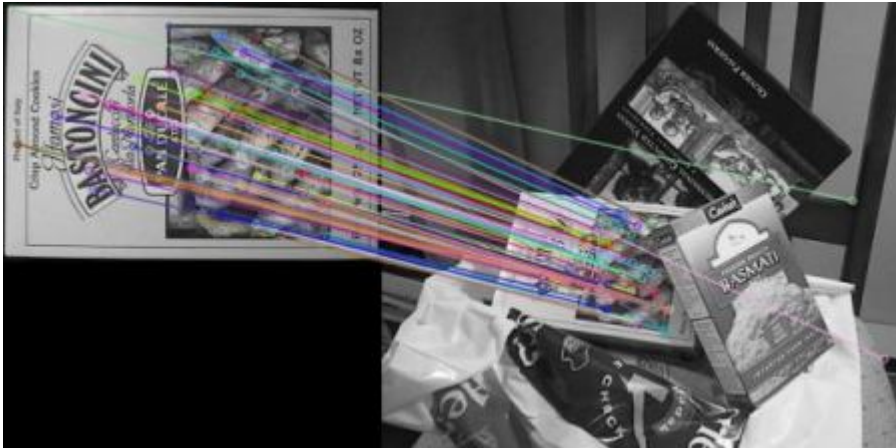
这次，我们将使用BFMatcher.knnMatch()获得k个最佳匹配。在此示例中，我们将 $k = 2$ ，以便可以应用D.Lowe在他的论文中阐述的比例测试。

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img1 = cv.imread('box.png', cv.IMREAD_GRAYSCALE) # 索引图像
img2 = cv.imread('box_in_scene.png', cv.IMREAD_GRAYSCALE) # 训练图像
# 初始化SIFT描述符
sift = cv.xfeatures2d.SIFT_create()
# 基于SIFT找到关键点和描述符
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
# 默认参数初始化BF匹配器
bf = cv.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
# 应用比例测试
good = []
for m, n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])
```

```
# cv.drawMatchesKnn将列表作为匹配项。  
img3 =  
cv.drawMatchesKnn(img1, kp1, img2, kp2, good, None, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,  
plt.imshow(img3), plt.show()
```

查看以下结果：



基于匹配器的FLANN

FLANN是近似最近邻的快速库。它包含一组算法，这些算法针对大型数据集中的快速最近邻搜索和高维特征进行了优化。对于大型数据集，它的运行速度比BFMatcher快。我们将看到第二个基于FLANN的匹配器示例。

对于基于FLANN的匹配器，我们需要传递两个字典，这些字典指定要使用的算法，其相关参数等。第一个是IndexParams。对于各种算法，要传递的信息在FLANN文档中进行了说明。概括来说，对于SIFT，SURF等算法，您可以通过以下操作：

```
FLANN_INDEX_KDTREE = 1  
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
```

使用ORB时，你可以参考下面。根据文档建议使用带注释的值，但在某些情况下未提供必需的参数。其他值也可以正常工作。

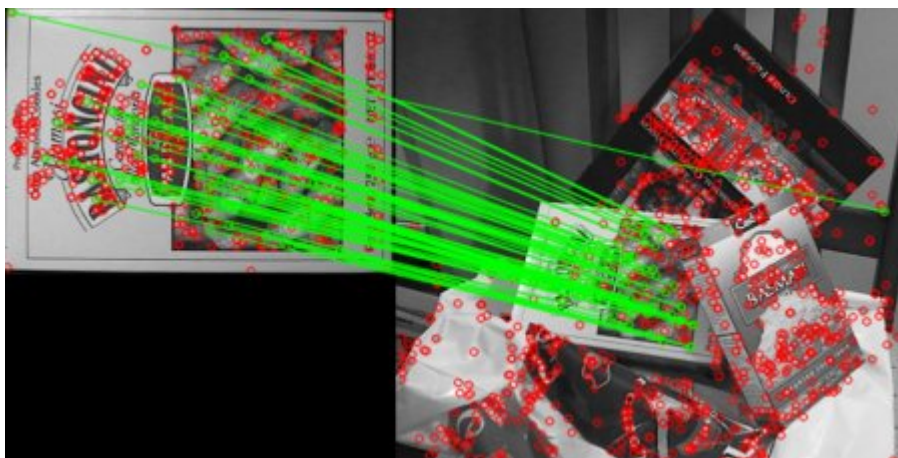
```
FLANN_INDEX_LSH = 6  
index_params= dict(algorithm = FLANN_INDEX_LSH,  
                    table_number = 6, # 12  
                    key_size = 12, # 20  
                    multi_probe_level = 1) #2
```

第二个字典是SearchParams。它指定索引中的树应递归遍历的次数。较高的值可提供更好的精度，但也需要更多时间。如果要更改值，请传递 `search_params = dict(checks = 100)`

有了这些信息，我们就很容易了。

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img1 = cv.imread('box.png', cv.IMREAD_GRAYSCALE) # 索引图像
img2 = cv.imread('box_in_scene.png', cv.IMREAD_GRAYSCALE) # 训练图像
# 初始化SIFT描述符
sift = cv.xfeatures2d.SIFT_create()
# 基于SIFT找到关键点和描述符
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
# FLANN的参数
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50) # 或传递一个空字典
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1, des2, k=2)
# 只需要绘制好匹配项，因此创建一个掩码
matchesMask = [[0,0] for i in range(len(matches))]
# 根据Lowe的论文进行比例测试
for i, (m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]
draw_params = dict(matchColor = (0,255,0),
                    singlePointColor = (255,0,0),
                    matchesMask = matchesMask,
                    flags = cv.DrawMatchesFlags_DEFAULT)
img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches, None, **draw_params)
plt.imshow(img3, ), plt.show()
```



查看以下结果：

附加资源

练习

特征匹配 + 单应性查找对象

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章节中，- 我们将把calib3d模块中的特征匹配和findHomography混合在一起，以在复杂图像中找到已知对象。

基础

那么我们在上一环节上做了什么？我们使用了queryImage，找到了其中的一些特征点，我们使用了另一个trainImage，也找到了该图像中的特征，并且找到了其中的最佳匹配。简而言之，我们在另一个混乱的图像中找到了对象某些部分的位置。此信息足以在trainImage上准确找到对象。

为此，我们可以使用calib3d模块中的函数，即**cv.findHomography**。如果我们从两个图像中传递点集，它将找到该对象的透视变换。然后，我们可以使用**cv.perspectiveTransform**查找对象。找到转换至少需要四个正确的点。

我们已经看到，匹配时可能会出现一些可能影响结果的错误。为了解决这个问题，算法使用 **RANSAC** 或 **LEAST_MEDIAN**（可以由标志决定）。因此，提供正确估计的良好匹配称为“内部点”，其余的称为“外部点”。**cv.findHomography()**返回指定内部和外部点的掩码。

让我们开始吧！！！！

代码

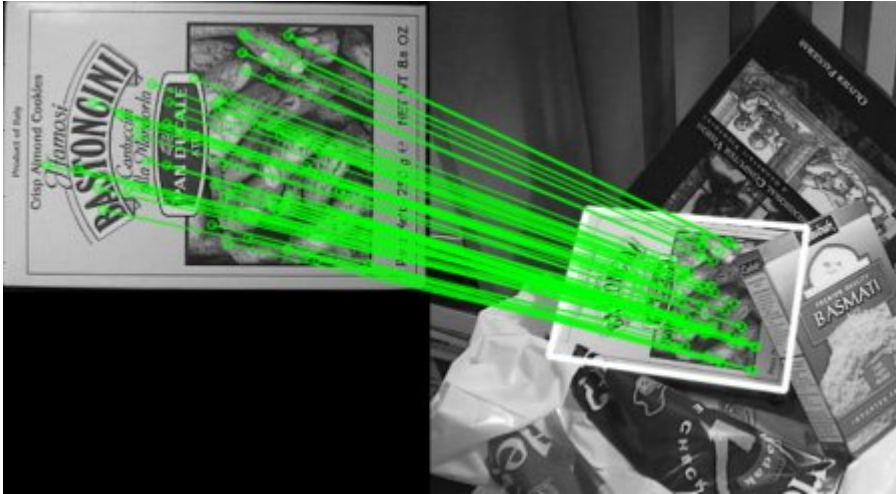
首先，像往常一样，让我们在图像中找到SIFT功能并应用比例测试以找到最佳匹配。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
MIN_MATCH_COUNT = 10
```



```
flags = 2)
img3 = cv.drawMatches(img1, kp1, img2, kp2, good, None, **draw_params)
plt.imshow(img3, 'gray'), plt.show()
```

请参阅下面的结果。对象在混乱的图像中标记为白色：



附加资源

练习

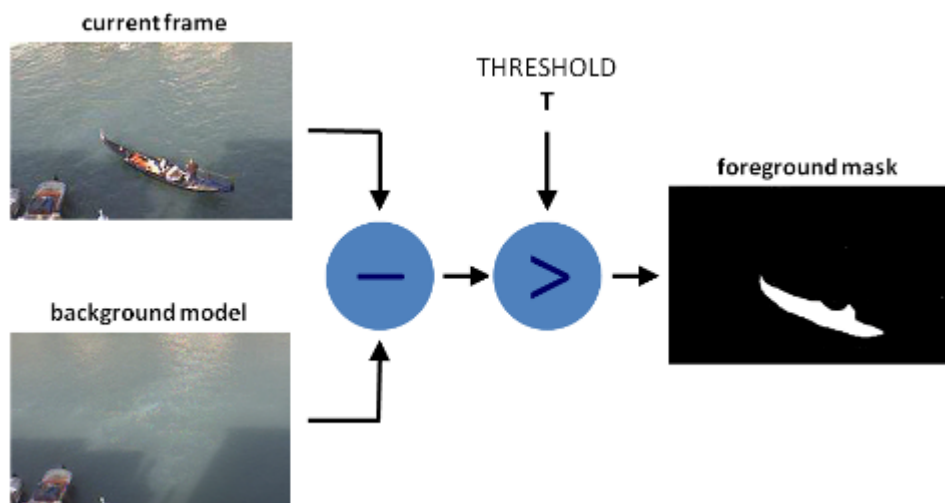
如何使用背景分离方法

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

- 背景分离 (BS) 是一种通过使用静态相机来生成前景掩码 (即包含属于场景中的移动对象像素的二进制图像) 的常用技术。
- 顾名思义, BS计算前景掩码, 在当前帧与背景模型之间执行减法运算, 其中包含场景的静态部分, 或者更一般而言, 考虑到所观察场景的特征, 可以将其视为背景的所有内容。



背景建模包括两个主要步骤：1. 背景初始化；2. 背景更新。

第一步，计算背景的初始模型，而在第二步中，更新模型以适应场景中可能的变化。

在本教程中，我们将学习如何使用OpenCV中的BS。

目标

在本教程中，您将学习如何：1. 使用**cv::VideoCapture**从视频或图像序列中读取数据；2. 通过使用**cv::BackgroundSubtractor**类创建和更新背景类；3. 通过使用**cv::imshow**获取并显示前景蒙版；

代码

在下面，您可以找到源代码。我们将让用户选择处理视频文件或图像序列。在此示例中，我们将使用**cv::BackgroundSubtractorMOG2**生成前景掩码。

结果和输入数据将显示在屏幕上。

```
from __future__ import print_function
import cv2 as cv
import argparse

parser = argparse.ArgumentParser(description='This program shows how to use
background subtraction methods provided by \
                                OpenCV. You can process both videos
and images.')
parser.add_argument('--input', type=str, help='Path to a video or a sequence of
image.', default='vtest.avi')
parser.add_argument('--algo', type=str, help='Background subtraction method (KNN,
MOG2).', default='MOG2')
args = parser.parse_args()
if args.algo == 'MOG2':
    backSub = cv.createBackgroundSubtractorMOG2()
else:
    backSub = cv.createBackgroundSubtractorKNN()
capture = cv.VideoCapture(cv.samples.findFileOrKeep(args.input))
if not capture.isOpened():
    print('Unable to open: ' + args.input)
    exit(0)
while True:
    ret, frame = capture.read()
    if frame is None:
        break

    fgMask = backSub.apply(frame)

    cv.rectangle(frame, (10, 2), (100, 20), (255, 255, 255), -1)
    cv.putText(frame, str(capture.get(cv.CAP_PROP_POS_FRAMES)), (15, 15),
               cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))

    cv.imshow('Frame', frame)
    cv.imshow('FG Mask', fgMask)

    keyboard = cv.waitKey(30)
    if keyboard == 'q' or keyboard == 27:
        break
```

解释

我们讨论上面代码的主要部分： - 一个**cv::BackgroundSubtractor**对象将用于生成前景掩码。在此示例中，使用了默认参数，但是也可以在create函数中声明特定的参数。

```
#创建背景分离对象
if args.algo == 'MOG2':
    backSub = cv.createBackgroundSubtractorMOG2()
else:
    backSub = cv.createBackgroundSubtractorKNN()
```

- 一个**cv::VideoCapture**对象用于读取输入视频或输入图像序列。

```
capture = cv.VideoCapture(cv.samples.findFileOrKeep(args.input))
if not capture.isOpened:
    print('Unable to open: ' + args.input)
    exit(0)
```

- 每帧都用于计算前景掩码和更新背景。如果要更改用于更新背景模型的学习率，可以通过将参数传递给apply方法来设置特定的学习率。

```
#更新背景模型
fgMask = backSub.apply(frame)
```

- 当前帧号可以从**cv::VideoCapture**对象中提取，并标记在当前帧的左上角。白色矩形用于突出显示黑色的帧编号。

```
#获取帧号并将其写入当前帧
cv.rectangle(frame, (10, 2), (100, 20), (255, 255, 255), -1)
cv.putText(frame, str(capture.get(cv.CAP_PROP_POS_FRAMES)), (15, 15),
           cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
```

- 我们准备显示当前的输入框和结果。

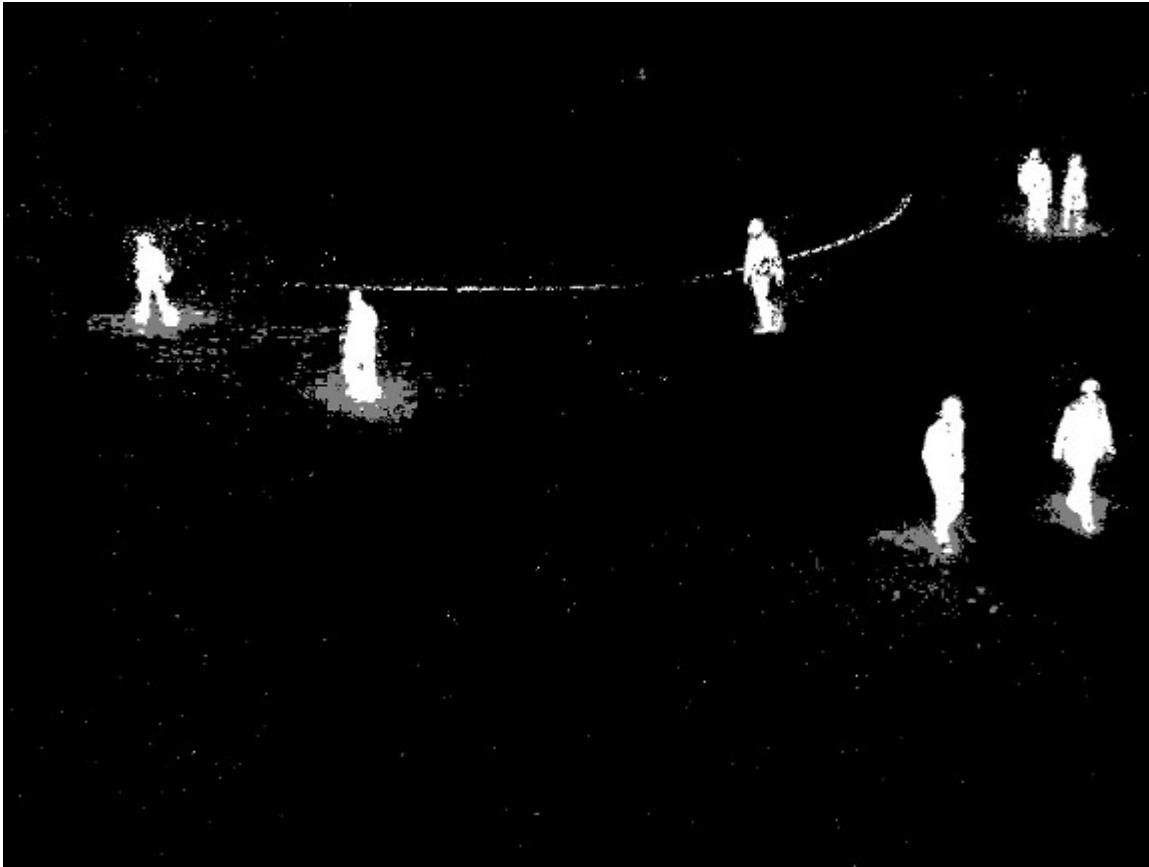
```
#展示当前帧和背景掩码
cv.imshow('Frame', frame)
cv.imshow('FG Mask', fgMask)
```

结果

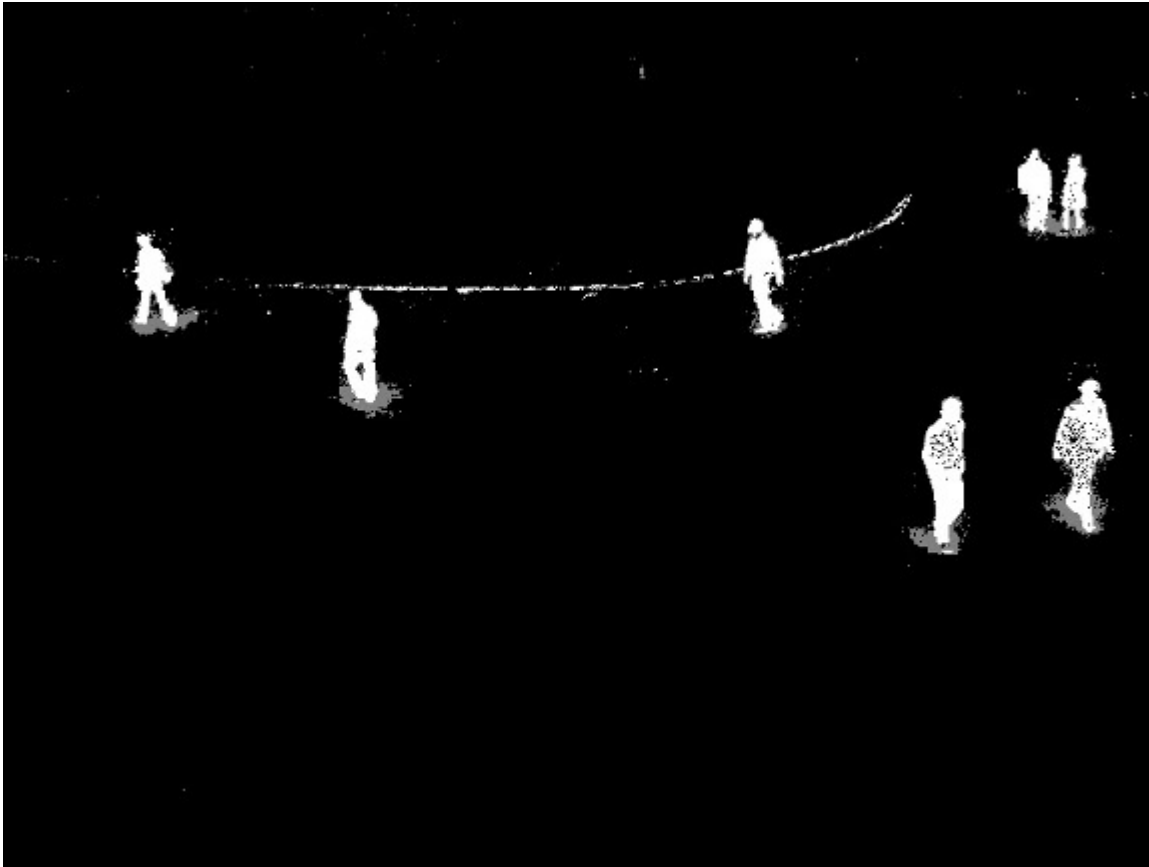
- 对于 `vtest.avi` 视频，适用以下框架：



MOG2方法的程序输出如下所示（检测到灰色区域有阴影）：



对于KNN方法，程序的输出将如下所示（检测到灰色区域的阴影）：



参考

- Background Models Challenge (BMC) website
- A Benchmark Dataset for Foreground/Background Extraction

Meanshift和Camshift

作者|OpenCV-Python Tutorials

编译|Vincent

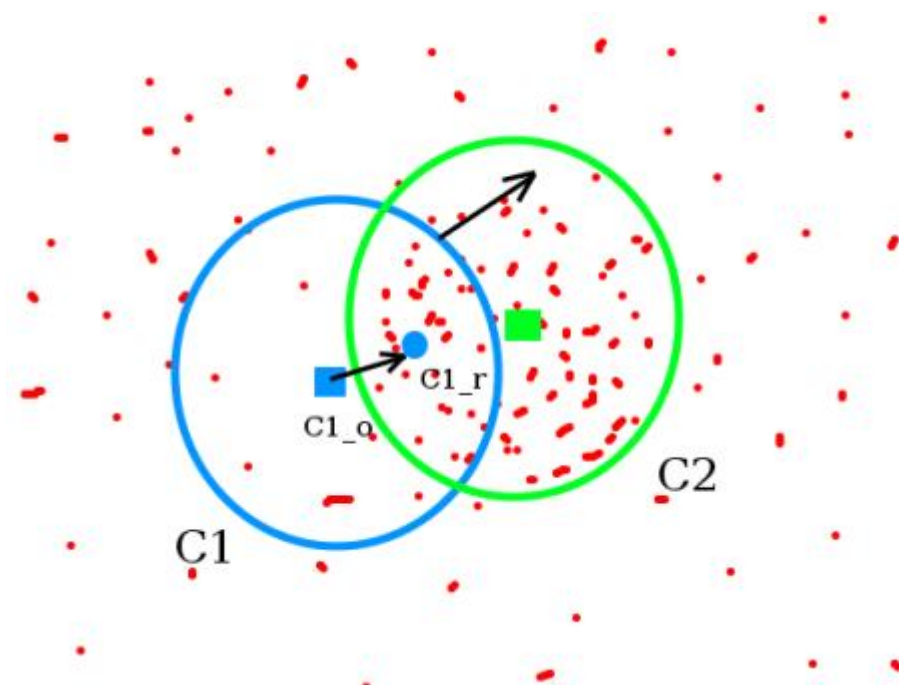
来源|OpenCV-Python Tutorials

学习目标

在本章中，- 我们将学习用于跟踪视频中对象的Meanshift和Camshift算法。

Meanshift

Meanshift背后的直觉很简单，假设你有点的集合。（它可以是像素分布，例如直方图反投影）。你会得到一个小窗口（可能是一个圆形），并且必须将该窗口移到最大像素密度（或最大点数）的区域。如下图所示：



初始窗口以蓝色圆圈显示，名称为“C1”。其原始中心以蓝色矩形标记，名称为“C1_o”。但是，如果找到该窗口内点的质心，则会得到点“C1_r”（标记为蓝色小圆圈），它是窗口的真实质心。当然，它们不匹配。因此，移动窗口，使新窗口的圆与上一个质心匹配。再次找到新的质心。很可

能不会匹配。因此，再次移动它，并继续迭代，以使窗口的中心及其质心落在同一位置（或在很小的期望误差内）。因此，最终您获得的是一个具有最大像素分布的窗口。它带有一个绿色圆圈，名为“C2”。正如您在图像中看到的，它具有最大的点数。整个过程在下面的静态图像上演示：



Mean shift window
initialization

因此，我们通常会传递直方图反投影图像和初始目标位置。当对象移动时，显然该移动会反映在直方图反投影图像中。结果，meanshift算法将窗口移动到最大密度的新位置。

OpenCV中的Meanshift

要在OpenCV中使用meanshift，首先我们需要设置目标，找到其直方图，以便我们可以将目标反投影到每帧上以计算均值偏移。我们还需要提供窗口的初始位置。对于直方图，此处仅考虑色相。另外，为避免由于光线不足而产生错误的值，可以使用**cv.inRange**()函数丢弃光线不足的值。


```
import numpy as np
import cv2 as cv
import argparse

parser = argparse.ArgumentParser(description='This sample demonstrates the
meanshift algorithm. \

                                The example file can be downloaded
from: \

                                https://www.bogotobogo.com/python/
OpenCV_Python/images/mean_shift_tracking/slow_traffic_small.mp4')
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
cap = cv.VideoCapture(args.image)

# 视频的第一帧
ret, frame = cap.read()
# 设置窗口的初始位置
x, y, w, h = 300, 200, 100, 50 # simply hardcoded the values
track_window = (x, y, w, h)
# 设置初始ROI来追踪
roi = frame[y:y+h, x:x+w]
hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255., 255.)))
roi_hist = cv.calcHist([hsv_roi], [0], mask, [180], [0, 180])
cv.normalize(roi_hist, roi_hist, 0, 255, cv.NORM_MINMAX)
# 设置终止条件, 可以是10次迭代, 也可以至少移动1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
while(1):
    ret, frame = cap.read()
    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)
        # 应用meanshift来获取新位置
        ret, track_window = cv.meanShift(dst, track_window, term_crit)
        # 在图像上绘制
        x,y,w,h = track_window
        img2 = cv.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv.imshow('img2',img2)
        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
    else:
        break
```

我使用的视频中的三帧如下：



Camshift

您是否密切关注了最后结果？这儿存在一个问题。无论汽车离相机很近或非常近，我们的窗口始终具有相同的大小。这是不好的。我们需要根据目标的大小和旋转来调整窗口大小。该解决方案再次来自“OpenCV Labs”，它被称为Gary布拉德斯基（Gary Bradsky）在其1998年的论文“用于感知用户界面中的计算机视觉面部跟踪”中发表的CAMshift（连续自适应均值偏移）[26]。它首先

应用Meanshift。一旦Meanshift收敛，它将更新窗口的大小为 $s = 2 \times \sqrt{\frac{M_{00}}{256}}$ 。它还可以计算出最合适的椭圆的方向。再次将均值偏移应用于新的缩放搜索窗口和先前的窗口位置。该过程一直持续到达到要求的精度为止。



Mean shift window
initialization

OpenCV中的Camshift

它与meanshift相似，但是返回一个旋转的矩形（即我们的结果）和box参数（用于在下一次迭代中作为搜索窗口传递）。请参见下面的代码：

```
import numpy as np
import cv2 as cv
import argparse
parser = argparse.ArgumentParser(description='This sample demonstrates the
camshift algorithm. \
from: \
The example file can be downloaded
https://www.bogotobogo.com/python/
```

```
OpenCV_Python/images/mean_shift_tracking/slow_traffic_small.mp4')
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
cap = cv.VideoCapture(args.image)
# 获取视频第一帧
ret, frame = cap.read()
# 设置初始窗口
x, y, w, h = 300, 200, 100, 50 # simply hardcoded the values
track_window = (x, y, w, h)
# 设置追踪的ROI窗口
roi = frame[y:y+h, x:x+w]
hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255., 255.)))
roi_hist = cv.calcHist([hsv_roi], [0], mask, [180], [0, 180])
cv.normalize(roi_hist, roi_hist, 0, 255, cv.NORM_MINMAX)
# 设置终止条件, 可以是10次迭代, 有可以至少移动1个像素
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
while(1):
    ret, frame = cap.read()
    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)
        # 应用camshift 到新位置
        ret, track_window = cv.CamShift(dst, track_window, term_crit)
        # 在图像上画出来
        pts = cv.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv.polylines(frame, [pts], True, 255, 2)
        cv.imshow('img2', img2)
        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
    else:
        break
```



三帧的结果如下

附加资源

1. French Wikipedia page on Camshift: <http://fr.wikipedia.org/wiki/Camshift>. (The two animations are taken from there)

2. Bradski, G.R., “Real time face and object tracking as a component of a perceptual user interface,” Applications of Computer Vision, 1998. WACV ‘98. Proceedings., Fourth IEEE Workshop on , vol., no., pp.214,219, 19-21 Oct 1998

Exercises

1. OpenCV comes with a Python :<https://github.com/opencv/opencv/blob/master/samples/python/camshift.py> for an interactive demo of camshift. Use it, hack it, understand it.

关于

扫描下方二维码，关注公众号【深度学习与计算机视觉】，获取更多计算机视觉教程，资源。

扫描下方二维码，关注公众号【深度学习与计算机视觉】，发送关键字“**pytorchpdf**”到公众号后台，可获得最新pytorch中文官方文档 PDF 资源。



深度学习与计算机视觉

微信扫描二维码，关注我的公众号

扫描下方二维码或搜索微信ID “**mthler**”，加我微信，备注“视觉”，拉你进交流群



贝加尔 
阿尔巴尼亚

加我微信，备注“视觉”，拉你进交流群



扫一扫上面的二维码图案，加我微信

OpenCV 中文官方文档

<http://woshicver.com/>

Github 地址

<https://github.com/fendouai/OpenCVTutorials>

光流

作者|OpenCV-Python Tutorials

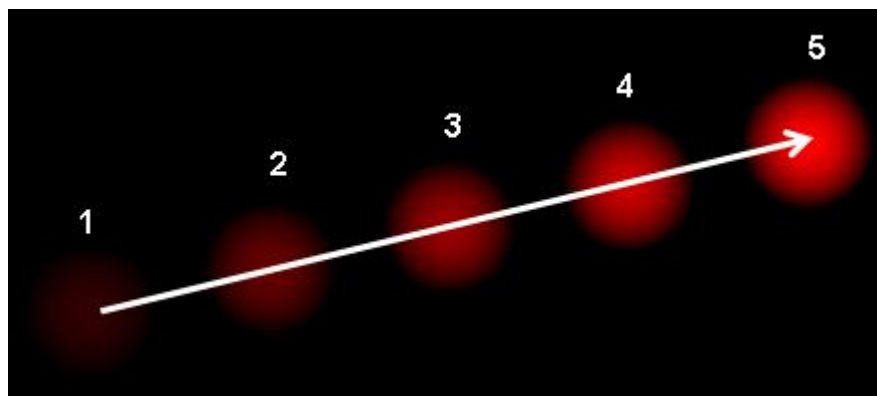
编译|Vincent

来源|OpenCV-Python Tutorials

在本章中， - 我们将了解光流的概念及其使用Lucas-Kanade方法的估计。 - 我们将使用 `cv.calcOpticalFlowPyrLK()` 之类的函数来跟踪视频中的特征点。 - 我们将使用 `cv.calcOpticalFlowFarneback()` 方法创建一个密集的光流场。

光流

光流是由物体或照相机的运动引起的两个连续帧之间图像物体的视运动的模式。它是2D向量场，其中每个向量都是位移向量，表示点从第一帧到第二帧的运动。考虑下面的图片（图片提供：Wikipedia关于Optical Flow的文章）。



它显示了一个球连续5帧运动。箭头显示其位移向量。光流在以下领域具有许多应用： - 运动的结构 - 视频压缩 - 视频稳定...

光流基于以下几个假设进行工作： 1. 在连续的帧之间，对象的像素强度不变。 2. 相邻像素具有相似的运动。

考虑第一帧中的像素 $I(x,y,t)$ （在此处添加新维度：时间。之前我们只处理图像，因此不需要时间）。它在 dt 时间之后拍摄的下一帧中按距离 (dx, dy) 移动。因此，由于这些像素相同且强度不变，因此可以说

$$I(x,y,t) = I(x+dx, y+dy, t+dt)$$

然后采用泰勒级数的右侧逼近，去掉常用项并除以dt得到下面的式子

$$f_x u + f_y v + f_t = 0$$

其中 $f_x = \frac{\partial f}{\partial x}$; $f_y = \frac{\partial f}{\partial y}$

$$u = \frac{dx}{dt}; v = \frac{dy}{dt}$$

上述方程式称为光流方程式。在其中，我们可以找到 f_x 和 f_y ，它们是图像渐变。同样， f_t 是随时间变化的梯度。但是 (u, v) 是未知的。我们不能用两个未知变量来求解这个方程。因此，提供了几种解决此问题的方法，其中一种是Lucas-Kanade。

Lucas-Kanade 方法

之前我们已经看到一个假设，即所有相邻像素将具有相似的运动。Lucas-Kanade方法在该点周围需要3x3色块。因此，所有9个点都具有相同的运动。我们可以找到这9点的 (f_x, f_y, f_t) 。所以现在我们的问题变成了求解带有两个未知变量的9个方程组的问题。用最小二乘拟合法可获得更好的解决方案。下面是最终的解决方案，它是两个方程式-两个未知变量问题，求解以获得解决答案。

$$\begin{bmatrix} u & v \end{bmatrix} = \begin{bmatrix} \sum_i \{f_{x_i}\}^2 & \sum_i \{f_{x_i} f_{y_i}\} \\ \sum_i \{f_{x_i} f_{y_i}\} & \sum_i \{f_{y_i}\}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i \{f_{x_i} f_{t_i}\} \\ -\sum_i \{f_{y_i} f_{t_i}\} \end{bmatrix}$$

(用哈里斯拐角检测器检查逆矩阵的相似性。这表示拐角是更好的跟踪点。) 因此，从用户的角度来看，这个想法很简单，我们给一些跟踪点，我们接收到这些光流矢量点。但是同样存在一些问题。到现在为止，我们只处理小动作，所以当大动作时它就失败了。为了解决这个问题，我们使用金字塔。当我们上金字塔时，较小的动作将被删除，较大的动作将变为较小的动作。因此，通过在此处应用Lucas-Kanade，我们可以获得与尺度一致的光流。

OpenCV中的Lucas-Kanade

OpenCV在单个函数`cv.calcOpticalFlowPyrLK()`中提供所有这些功能。在这里，我们创建一个简单的应用程序来跟踪视频中的某些点。为了确定点，我们使用`cv.goodFeaturesToTrack()`。我们采用第一帧，检测其中的一些Shi-Tomasi角点，然后使用Lucas-Kanade光流迭代地跟踪这些点。对于函数`cv.calcOpticalFlowPyrLK()`，我们传递前一帧，前一点和下一帧。它返回下一个点以及一些状态码，如果找到下一个点，状态码的值为1，否则为零。我们将这些下一个点迭代地传递为下一步中的上一个点。请参见下面的代码：

```
import numpy as np
import cv2 as cv
import argparse

parser = argparse.ArgumentParser(description='This sample demonstrates Lucas-
Kanade Optical Flow calculation. \

                                The example file can be downloaded
from: \

                                https://www.bogotobogo.com/python/
OpenCV_Python/images/mean_shift_tracking/slow_traffic_small.mp4')
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
cap = cv.VideoCapture(args.image)

# 用于ShiTomasi拐点检测的参数
feature_params = dict( maxCorners = 100,
                        qualityLevel = 0.3,
                        minDistance = 7,
                        blockSize = 7 )

# lucas kanade光流参数
lk_params = dict( winSize = (15,15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10,
0.03))

# 创建一些随机的颜色
color = np.random.randint(0,255,(100,3))

# 拍摄第一帧并在其中找到拐角
ret, old_frame = cap.read()
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_params)

# 创建用于作图的掩码图像
mask = np.zeros_like(old_frame)

while(1):
    ret, frame = cap.read()
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    # 计算光流
    p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,
**lk_params)

    # 选择良好点
    good_new = p1[st==1]
    good_old = p0[st==1]

    # 绘制跟踪
    for i,(new,old) in enumerate(zip(good_new, good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        mask = cv.line(mask, (a,b),(c,d), color[i].tolist(), 2)
        frame = cv.circle(frame, (a,b), 5, color[i].tolist(), -1)
    img = cv.add(frame, mask)
    cv.imshow('frame', img)
```

```
k = cv.waitKey(30) & 0xff
if k == 27:
    break
# 现在更新之前的帧和点
old_gray = frame_gray.copy()
p0 = good_new.reshape(-1,1,2)
```

(此代码不会检查下一个关键点正确性。因此，即使任何特征点在图像中消失了，光流也有可能找到下一个看起来可能与它接近的下一个点。因此，对于稳健的跟踪，实际上 应该以特定的时间间隔检测点。OpenCV样本附带了这样一个样本，该样本每5帧发现一次特征点，并且还 对光流点进行了后向检查，以仅选择良好的流点。请参阅代码 `samples/python/lk_track.py`)。

查看我们得到的结果：



OpenCV中的密集光流

Lucas-Kanade方法计算稀疏特征集的光流 (在我们的示例中为使用Shi-Tomasi算法检测到的角)。OpenCV提供了另一种算法来查找密集的光流。它计算帧中所有点的光通量。它基于Gunner Farneback的算法，在2003年Gunner Farneback的“基于多项式展开的两帧运动估计”中对此进行了解释。

下面的示例显示了如何使用上述算法找到密集的光流。我们得到一个带有光流矢量 (u,v) 的2通道阵列。我们找到了它们的大小和方向。我们对结果进行颜色编码，以实现更好的可视化。方向对应于图像的色相值。幅度对应于值平面。请参见下面的代码：

```
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(cv.samples.findFile("vtest.avi"))
ret, frame1 = cap.read()
prvs = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[...,1] = 255
while(1):
    ret, frame2 = cap.read()
    next = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)
    flow = cv.calcOpticalFlowFarneback(prvs, next, None, 0.5, 3, 15, 3, 5, 1.2, 0)
    mag, ang = cv.cartToPolar(flow[...,0], flow[...,1])
    hsv[...,0] = ang*180/np.pi/2
    hsv[...,2] = cv.normalize(mag, None, 0, 255, cv.NORM_MINMAX)
    bgr = cv.cvtColor(hsv, cv.COLOR_HSV2BGR)
    cv.imshow('frame2', bgr)
    k = cv.waitKey(30) & 0xff
    if k == 27:
        break
    elif k == ord('s'):
        cv.imwrite('opticalfb.png', frame2)
        cv.imwrite('opticalhsv.png', bgr)
    prvs = next
```



查看以下结果：

相机校准

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

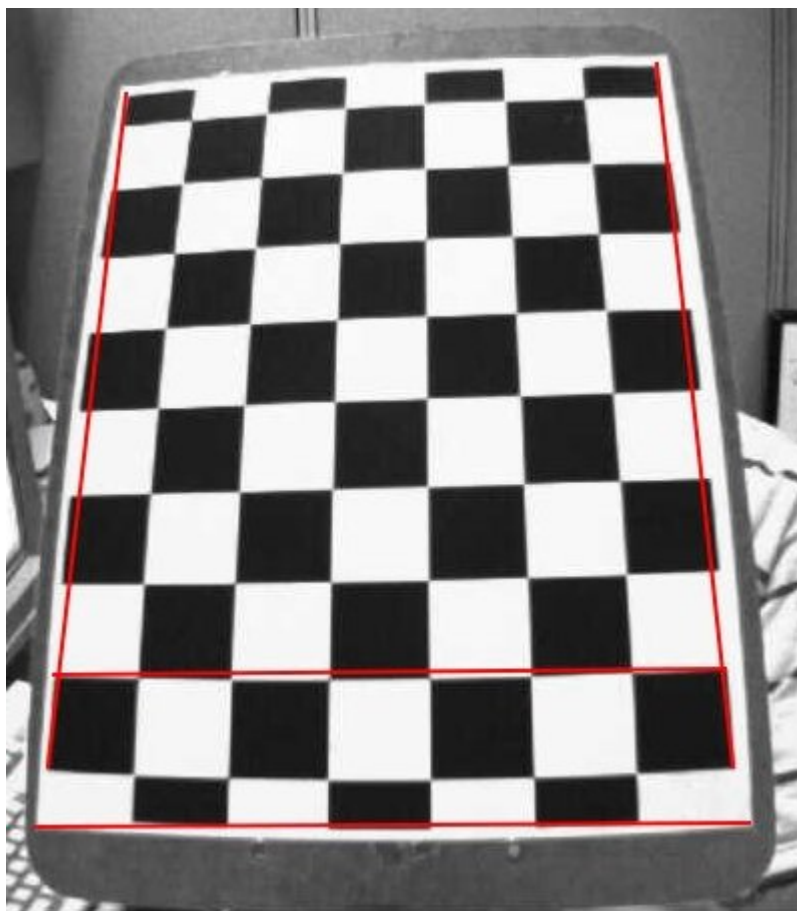
目标

在本节中，我们将学习 - 由相机引起的失真类型， - 如何找到相机的固有和非固有特性 - 如何根据这些特性使图像不失真

基础

一些针孔相机会给图像带来明显的失真。两种主要的变形是径向变形和切向变形。径向变形会导致直线出现弯曲。

距图像中心越远，径向畸变越大。例如，下面显示一个图像，其中棋盘的两个边缘用红线标记。但是，您会看到棋盘的边框不是直线，并且与红线不匹配。所有预期的直线都凸出。有关更多详细信息，请访问“失真（光学）”。



径向变形可以表示成如下：

$$x_{\text{distorted}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad y_{\text{distorted}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

同样，由于摄像镜头未完全平行于成像平面对齐，因此会发生切向畸变。因此，图像中的某些区域看起来可能比预期的要近。切向畸变的量可以表示为：

$$x_{\text{distorted}} = x + [2p_1xy + p_2(r^2 + 2x^2)] \quad y_{\text{distorted}} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

简而言之，我们需要找到五个参数，称为失真系数，公式如下：

$$\text{Distortion coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

除此之外，我们还需要其他一些信息，例如相机的内在和外在参数。内部参数特定于摄像机。它们包括诸如焦距(f_x , f_y)和光学中心(c_x , c_y)之类的信息。焦距和光学中心可用于创建相机矩阵，该相机矩阵可用于消除由于特定相机镜头而引起的畸变。相机矩阵对于特定相机而言是唯一的，因此一旦计算出，就可以在同一相机拍摄的其他图像上重复使用。它表示为3x3矩阵：

```
camera \; matrix = \left [ \begin{matrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{matrix} \right ]
```

外在参数对应于旋转和平移矢量，其将3D点的坐标平移为坐标系。

对于立体声应用，首先需要纠正这些失真。要找到这些参数，我们必须提供一些定义良好的图案的示例图像（例如国际象棋棋盘）。我们找到一些已经知道其相对位置的特定位置（例如棋盘上的四角）。我们知道现实世界空间中这些点的坐标，也知道图像中的坐标，因此我们可以求解失真系数。为了获得更好的结果，我们至少需要10个测试模式。

代码

如上所述，相机校准至少需要10个测试图案。OpenCV附带了一些国际象棋棋盘的图像（请参见 `samples / data / left01.jpg – left14.jpg`），因此我们将利用这些图像。考虑棋盘的图像。相机校准所需的重要输入数据是3D现实世界点集以及图像中这些点的相应2D坐标。可以从图像中轻松找到2D图像点。（这些图像点是国际象棋棋盘中两个黑色正方形相互接触的位置）

真实世界中的3D点如何处理？这些图像是从静态相机拍摄的，而国际象棋棋盘放置在不同的位置和方向。因此，我们需要知道(X,Y,Z)值。但是为简单起见，我们可以说棋盘在XY平面上保持静止（因此Z始终为0），并且照相机也相应地移动了。这种考虑有助于我们仅找到X，Y值。现在对于X，Y值，我们可以简单地将点传递为（0,0），（1,0），（2,0），...，这表示点的位置。在这种情况下，我们得到的结果将是棋盘正方形的大小比例。但是，如果我们知道正方形大小（例如30毫米），则可以将值传递为（0,0），（30,0），（60,0），...。因此，我们得到的结果以毫米为单位。（在这种情况下，我们不知道正方形的大小，因为我们没有拍摄那些图像，因此我们以正方形的大小进行传递）。

3D点称为**对象点**，而2D图像点称为**图像点**。

开始

因此，要在国际象棋棋盘中查找图案，我们可以使用函数**cv.findChessboardCorners()**。我们还需要传递所需的图案，例如8x8网格，5x5网格等。在此示例中，我们使用7x6网格。（通常，棋盘有8x8的正方形和7x7的内部角）。它返回角点和retval，如果获得图案，则为True。这些角将按顺序放置（从左到右，从上到下）

另外

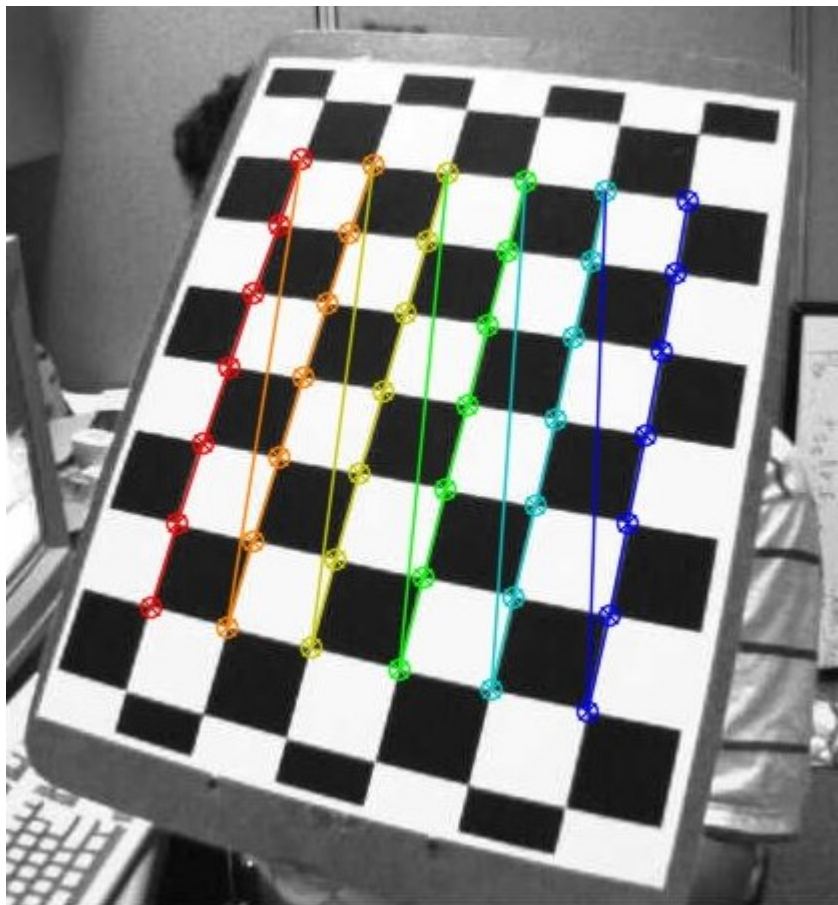
此功能可能无法在所有图像中找到所需的图案。因此，一个不错的选择是**编写**代码，使它启动相机并检查每帧所需的图案。获得图案后，找到角并将其存储在列表中。另外，在阅读下一帧之前请提供一些时间间隔，以便我们可以在不同方向上调整棋盘。继续此过程，直到获得所需数量的良好图案为止。即使在此处提供的示例中，我们也不确定给出的14张图像中有多少张是好的。

因此，我们必须**阅读**所有图像并仅拍摄好图像。除了棋盘，我们还可以使用圆形网格。在这种情况下，我们必须使用函数**cv.findCirclesGrid**()来找到模式。较少的图像足以使用圆形网格执行相机校准。

一旦找到拐角，就可以使用**cv.cornerSubPix**()来提高其精度。我们还可以使用**cv.drawChessboardCorners**()绘制图案。所有这些步骤都包含在以下代码中：

```
import numpy as np
import cv2 as cv
import glob
# 终止条件
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# 准备对象点， 如 (0,0,0), (1,0,0), (2,0,0) ...., (6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:, :2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
# 用于存储所有图像的对象点和图像点的数组。
objpoints = [] # 真实世界中的3d点
imgpoints = [] # 图像中的2d点
images = glob.glob('*.jpg')
for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # 找到棋盘角落
    ret, corners = cv.findChessboardCorners(gray, (7,6), None)
    # 如果找到，添加对象点，图像点（细化之后）
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
        # 绘制并显示拐角
        cv.drawChessboardCorners(img, (7,6), corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)
cv.destroyAllWindows()
```

一张上面画有图案的图像如下所示：



校准

现在我们有目标点和图像点，现在可以进行校准了。我们可以使用函数`cv.calibrateCamera()`返回相机矩阵，失真系数，旋转和平移矢量等。

```
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
```

不失真

现在，我们可以拍摄图像并对其进行扭曲。OpenCV提供了两种方法来执行此操作。但是，首先，我们可以使用`cv.getOptimalNewCameraMatrix()`基于自由缩放参数来优化相机矩阵。如果缩放参数 $\alpha = 0$ ，则返回具有最少不需要像素的未失真图像。因此，它甚至可能会删除图像角落的一些像素。如果 $\alpha = 1$ ，则所有像素都保留有一些额外的黑色图像。此函数还返回可用于裁剪结果的图像ROI。

因此，我们拍摄一张新图像（在本例中为left12.jpg。这是本章的第一张图像）

```
img = cv.imread('left12.jpg')
h, w = img.shape[:2]
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
```

1. 使用cv.undistort()

这是最简单的方法。只需调用该函数并使用上面获得的ROI裁剪结果即可。

```
# undistort
dst = cv.undistort(img, mtx, dist, None, newcameramtx)
# 剪裁图像
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('calibresult.png', dst)
```

2. 使用remapping

该方式有点困难。首先，找到从扭曲图像到未扭曲图像的映射函数。然后使用重映射功能。

```
# undistort
mapx, mapy = cv.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h), 5)
dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)
# 剪裁图像
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('calibresult.png', dst)
```

尽管如此，两种方法都给出相同的结果。看到下面的结果：



您可以看到所有边缘都是笔直的。现在，您可以使用NumPy中的写入功能(`np.savez`，`np.savetxt`等)存储相机矩阵和失真系数，以备将来使用。

重投影误差

重投影误差可以很好地估计找到的参数的精确程度。重投影误差越接近零，我们发现的参数越准确。给定固有，失真，旋转和平移矩阵，我们必须首先使用`cv.projectPoints()`将对象点转换为图像点。然后，我们可以计算出通过变换得到的绝对值和拐角发现算法之间的绝对值范数。为了找到平均误差，我们计算为所有校准图像计算的误差的算术平均值。

```
mean_error = 0
for i in xrange(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error
print( "total error: {}".format(mean_error/len(objpoints)) )
```

附加资源

练习

1. 尝试使用圆形网格进行相机校准。

姿态估计

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本章中 - 我们将学习利用calib3d模块在图像中创建一些3D效果。

基础

这将是一小部分。在上一次相机校准的会话中，你发现了相机矩阵，失真系数等。给定图案图像，我们可以利用以上信息来计算其姿势或物体在空间中的位置，例如其旋转方式，对于平面物体，我们可以假设 $Z = 0$ ，这样，问题就变成了如何将相机放置在空间中以查看图案图像。因此，如果我们知道对象在空间中的位置，则可以在其中绘制一些2D图以模拟3D效果。让我们看看如何做。

我们的问题是，我们想在棋盘的第一个角上绘制3D坐标轴(X,Y,Z)。X轴为蓝色，Y轴为绿色，Z轴为红色。因此，实际上Z轴应该感觉像它垂直于我们的棋盘平面。

首先，让我们从先前的校准结果中加载相机矩阵和失真系数。

```
import numpy as np
import cv2 as cv
import glob
# 加载先前保存的数据
with np.load('B.npz') as X:
    mtx, dist, _, _ = [X[i] for i in ('mtx', 'dist', 'rvecs', 'tvecs')]
```

现在让我们创建一个函数，绘制，该函数将棋盘上的角（使用cv.findChessboardCorners()获得）和**轴点**绘制为3D轴。

```
def draw(img, corners, imgpts):
    corner = tuple(corners[0].ravel())
    img = cv.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0), 5)
```

```
img = cv.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0), 5)
img = cv.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255), 5)
return img
```

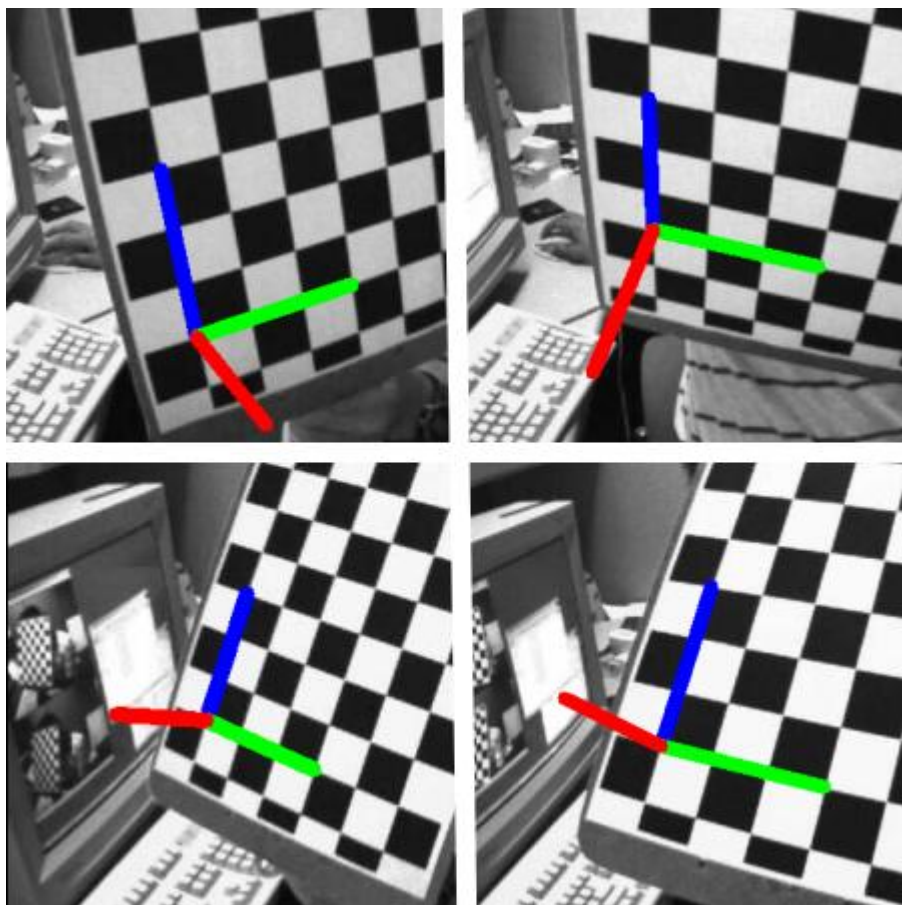
然后，与前面的情况一样，我们创建终止条件，对象点（棋盘上角的3D点）和轴点。轴点是3D空间中用于绘制轴的点。我们绘制长度为3的轴（由于我们根据该棋盘尺寸进行了校准，因此单位将以国际象棋正方形的尺寸为单位）。因此我们的X轴从(0,0,0)绘制为(3,0,0)，因此对于Y轴。对于Z轴，从(0,0,0)绘制为(0,0,-3)。负号表示它被拉向相机。

```
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
objp = np.zeros((6*7,3), np.float32)
objp[:, :2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
```

现在，像往常一样，我们加载每个图像。搜索7x6网格。如果找到，我们将使用子角像素对其进行优化。然后使用函数**cv.solvePnP Ransac**()计算旋转和平移。一旦有了这些变换矩阵，就可以使用它们将轴点投影到图像平面上。简而言之，我们在图像平面上找到与3D空间中(3,0,0),(0,3,0),(0,0,3)中的每一个相对应的点。一旦获得它们，就可以使用draw()函数从第一个角到这些点中的每个点绘制线条。完毕!!!

```
for fname in glob.glob('left*.jpg'):
    img = cv.imread(fname)
    gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCorners(gray, (7,6),None)
    if ret == True:
        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1),criteria)
        # 找到旋转和平移矢量。
        ret,rvecs, tvecs = cv.solvePnP(objp, corners2, mtx, dist)
        # 将3D点投影到图像平面
        imgpts, jac = cv.projectPoints(axis, rvecs, tvecs, mtx, dist)
        img = draw(img,corners2,imgpts)
        cv.imshow('img',img)
        k = cv.waitKey(0) & 0xFF
        if k == ord('s'):
            cv.imwrite(fname[:6]+'.png', img)
cv.destroyAllWindows()
```

请参阅下面的一些结果。请注意，每个轴长3个long单位。



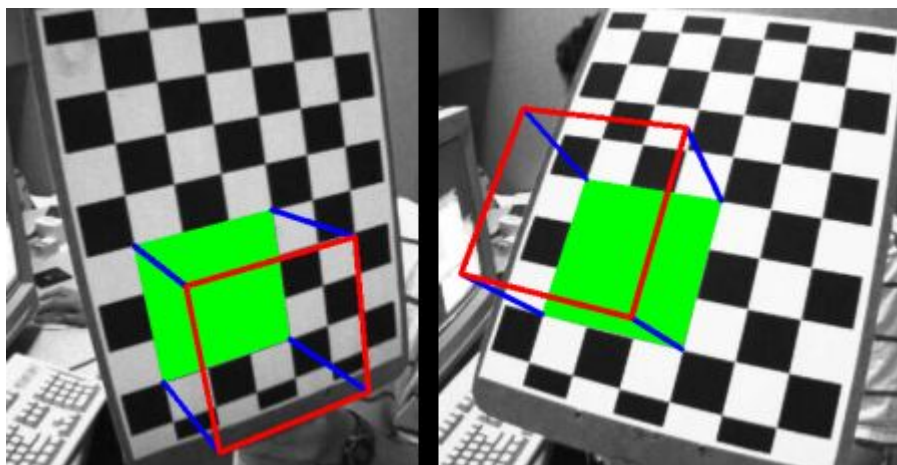
绘制立方体

如果要绘制立方体，请如下修改draw()函数和轴点。修改后的draw()函数：

```
def draw(img, corners, imgpts):
    imgpts = np.int32(imgpts).reshape(-1,2)
    # 用绿色绘制底层
    img = cv.drawContours(img, [imgpts[:4]],-1,(0,255,0),-3)
    # 用蓝色绘制高
    for i,j in zip(range(4),range(4,8)):
        img = cv.line(img, tuple(imgpts[i]), tuple(imgpts[j]),(255),3)
    # 用红色绘制顶层
    img = cv.drawContours(img, [imgpts[4:]],-1,(0,0,255),3)
    return img
```

修改的轴点。它们是3D空间中多维数据集的8个角：

```
axis = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],
                  [0,-3], [0,3,-3], [3,3,-3], [3,0,-3] ])  [0,
```

查看以下结果：

如果您对图形，增强现实等感兴趣，则可以使用OpenGL渲染更复杂的图形。

附加资源

练习

对极几何

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

在本节中 - 我们将学习多视图几何的基础知识 - 我们将了解什么是极点，极线，极线约束等。

基础概念

当我们使用针孔相机拍摄图像时，我们失去了重要信息，即图像深度。或者图像中的每个点距相机多远，因为它是3D到2D转换。因此，是否能够使用这些摄像机找到深度信息是一个重要的问题。答案是使用不止一台摄像机。在使用两台摄像机（两只眼睛）的情况下，我们的眼睛工作方式相似，这称为立体视觉。因此，让我们看看OpenCV在此字段中提供了什么。

（通过Gary Bradsky学习OpenCV在该领域有很多信息。）

在深入图像之前，让我们首先了解多视图几何中的一些基本概念。在本节中，我们将讨论对极几何。请参见下图，该图显示了使用两台摄像机拍摄同一场景的图像的基本设置。

如果仅使用左摄像机，则无法找到与图像中的点相对应的3D点，因为线上的每个点都投影到图像平面上的同一点。但也要考虑正确的形象。现在，直线OX上的不同点投射到右侧平面上的不同点(x')。因此，使用这两个图像，我们可以对正确的3D点进行三角剖分。这就是整个想法。

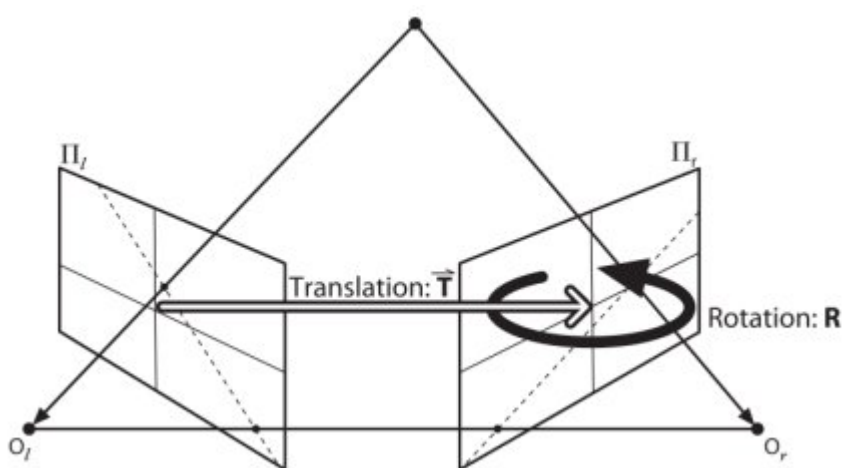
不同点的投影在右平面OX上形成一条线(line l')。我们称其为对应于该点的**Epiline**。这意味着，要在正确的图像上找到该点，请沿着该轮廓线搜索。它应该在这条线上的某处（以这种方式考虑，可以在其他图像中找到匹配点，而无需搜索整个图像，只需沿着Epiline搜索即可。因此，它可以提供更好的性能和准确性）。这称为对极约束。类似地，所有点在另一幅图像中将具有其对应的Epiline。该平面称为**对极面**。

O和O'是相机中心。从上面给出的设置中，您可以看到在点处的左侧图像上可以看到右侧摄像机O'的投影。它称为**极点**。极点是穿过相机中心和图像平面的线的交点。左摄像机的极点也同理。

在某些情况下，您将无法在图像中找到极点，它们可能位于图像外部（这意味着一个摄像机看不到另一个摄像机）。

所有的极线都通过其极点。因此，要找到中心线的位置，我们可以找到许多中心线并找到它们的交点。

因此，在节中，我们将重点放在寻找对极线和极线。但是要找到它们，我们需要另外两种成分，即**基础矩阵(F)和**基本矩阵(E)，基础矩阵包含有关平移和旋转的信息，这些信息在全局坐标中描述了第二个摄像头相对于第一个摄像头的位置。参见下图(图像由Gary Bradsky提供：Learning OpenCV):



但是我们会更喜欢在像素坐标中进行测量，对吧？基本矩阵除包含有关两个摄像头的内在信息之外，还包含与基本矩阵相同的信息，因此我们可以将两个摄像头的像素坐标关联起来。（如果我们使用的是校正后的图像，并用焦距除以标准化该点， $F=E$ ）。简而言之，基本矩阵F将一个图像中的点映射到另一图像中的线（上）。这是从两个图像的匹配点计算得出的。至少需要8个这样的点才能找到基本矩阵（使用8点算法时）。选择更多点并使用RANSAC将获得更可靠的结果。

代码

因此，首先我们需要在两个图像之间找到尽可能多的匹配项，以找到基本矩阵。为此，我们将SIFT描述符与基于FLANN的匹配器和比率测试结合使用。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img1 = cv.imread('myleft.jpg',0) #索引图像 # left image
img2 = cv.imread('myright.jpg',0) #训练图像 # right image
sift = cv.SIFT()
```

```
# 使用SIFT查找关键点和描述符
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
# FLANN 参数
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1, des2, k=2)
good = []
pts1 = []
pts2 = []
# 根据Lowe的论文进行比率测试
for i, (m, n) in enumerate(matches):
    if m.distance < 0.8*n.distance:
        good.append(m)
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)
```

现在，我们获得了两张图片的最佳匹配列表。让我们找到基本矩阵。

```
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
F, mask = cv.findFundamentalMat(pts1, pts2, cv.FM_LMEDS)
# 我们只选择内点
pts1 = pts1[mask.ravel() == 1]
pts2 = pts2[mask.ravel() == 1]
```

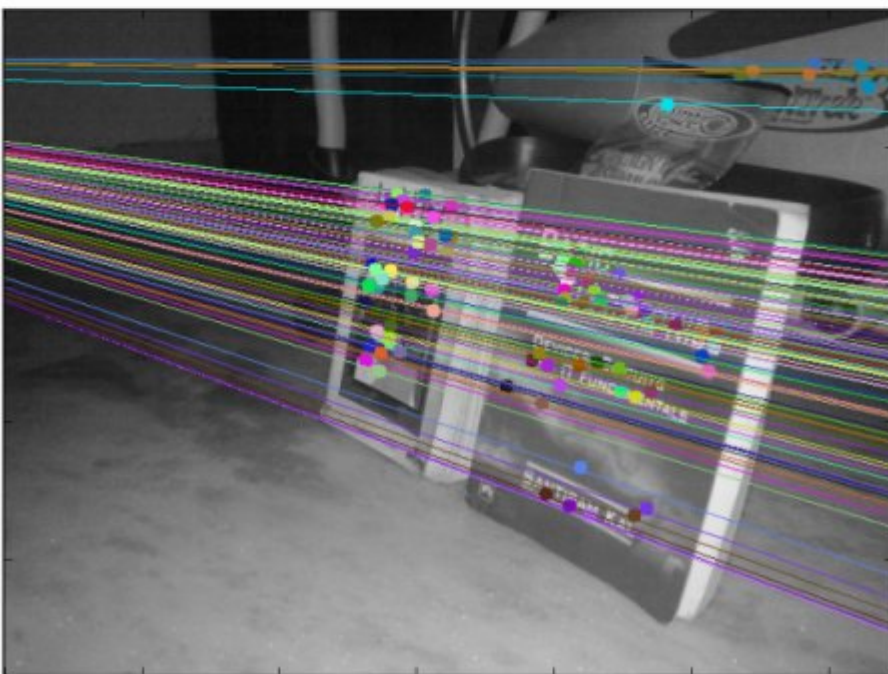
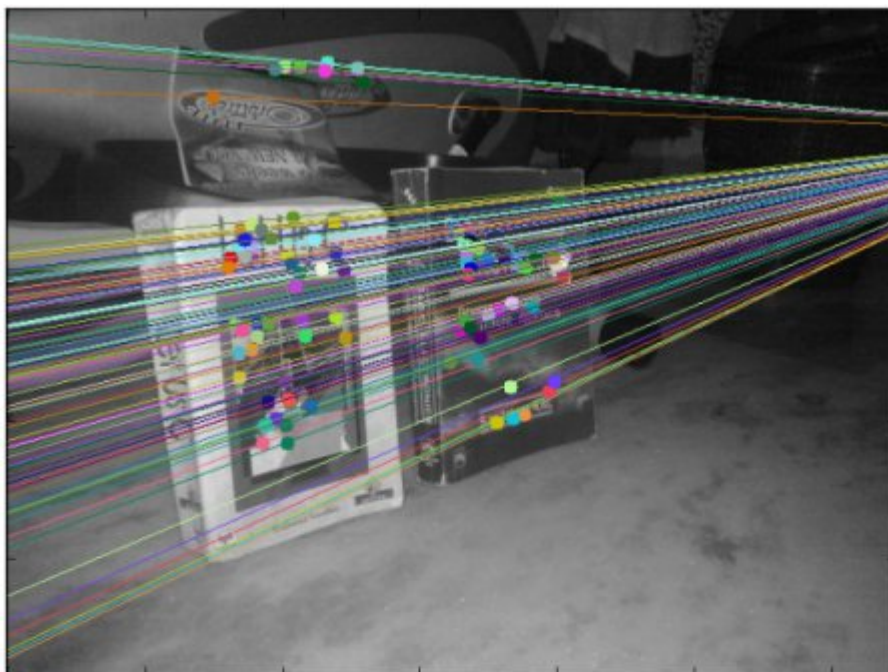
接下来，我们找到Epilines。在第二张图像上绘制与第一张图像中的点相对应的Epilines。因此，在这里提到正确的图像很重要。我们得到了一行线。因此，我们定义了一个新功能来在图像上绘制这些线条。

```
def drawlines(img1, img2, lines, pts1, pts2):
    ''' img1 - 我们在img2相应位置绘制极点生成的图像
        lines - 对应的极点 '''
    r, c = img1.shape
    img1 = cv.cvtColor(img1, cv.COLOR_GRAY2BGR)
    img2 = cv.cvtColor(img2, cv.COLOR_GRAY2BGR)
    for r, pt1, pt2 in zip(lines, pts1, pts2):
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0, y0 = map(int, [0, -r[2]/r[1] ])
        x1, y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
        img1 = cv.line(img1, (x0, y0), (x1, y1), color, 1)
        img1 = cv.circle(img1, tuple(pt1), 5, color, -1)
```

```
img2 = cv.circle(img2, tuple(pt2), 5, color, -1)
return img1, img2
```

现在，我们在两个图像中都找到了Epiline并将其绘制。

```
# 在右图（第二张图）中找到与点相对应的极点，然后在左图绘制极线
lines1 = cv.computeCorrespondEpilines(pts2.reshape(-1, 1, 2), 2, F)
lines1 = lines1.reshape(-1, 3)
img5, img6 = drawlines(img1, img2, lines1, pts1, pts2)
# 在左图（第一张图）中找到与点相对应的Epilines，然后在正确的图像上绘制极线
lines2 = cv.computeCorrespondEpilines(pts1.reshape(-1, 1, 2), 1, F)
lines2 = lines2.reshape(-1, 3)
img3, img4 = drawlines(img2, img1, lines2, pts2, pts1)
plt.subplot(121), plt.imshow(img5)
plt.subplot(122), plt.imshow(img3)
plt.show()
```



以下是我们得到的结果：

您可以在左侧图像中看到所有极点都收敛在右侧图像的外部。那个汇合点就是极点。为了获得更好的结果，应使用具有良好分辨率和许多非平面点的图像。

附加资源

练习

1. 一个重要的主题是相机的前移。然后，将在两个位置的相同位置看到极点，并且从固定点出现极点。请参阅此讨论。
2. 基本矩阵估计对匹配，离群值等的质量敏感。如果所有选定的匹配都位于同一平面上，则情况会变得更糟。检查此讨论。

7_4_立体图像的深度图

立体图像的深度图

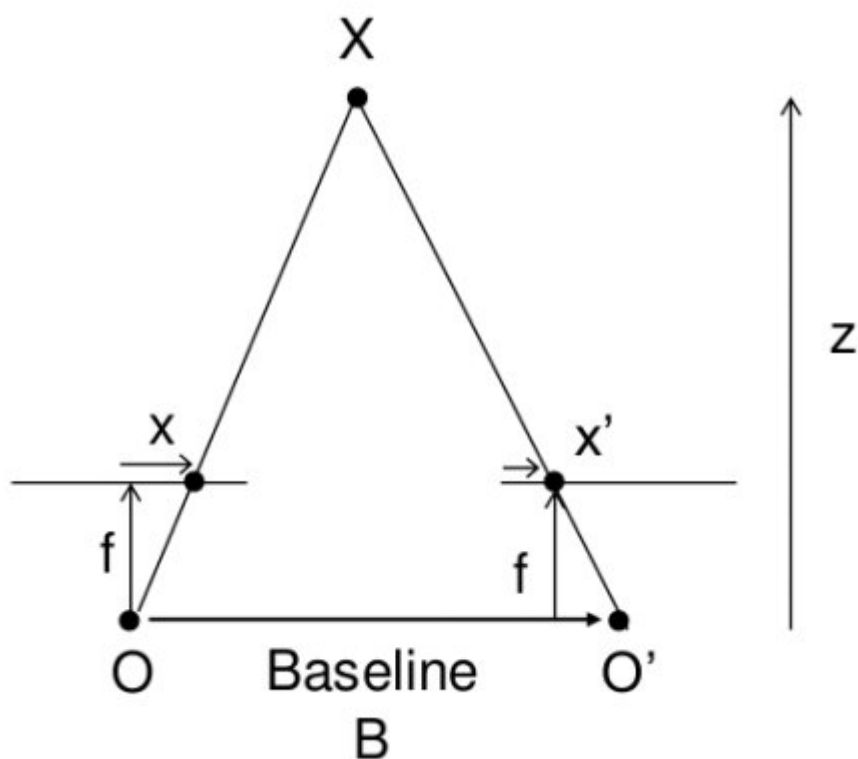
作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本节中，- 我们将学习根据立体图像创建深度图。

基础

在上一节中，我们看到了对极约束和其他相关术语等基本概念。我们还看到，如果我们有两个场景相同的图像，则可以通过直观的方式从中获取深度信息。下面是一张图片和一些简单的数学公式证明了这种想法。



上图包含等效三角形。编写它们的等式将产生以下结果：

$$\text{disparity} = x - x' = \frac{Bf}{Z}$$

x 和 x' 是图像平面中与场景点3D相对应的点与其相机中心之间的距离。 B 是两个摄像机之间的距离（我们知道）， f 是摄像机的焦距（已经知道）。简而言之，上述方程式表示场景中某个点的深度与相应图像点及其相机中心的距离差成反比。因此，利用此信息，我们可以得出图像中所有像素的深度。

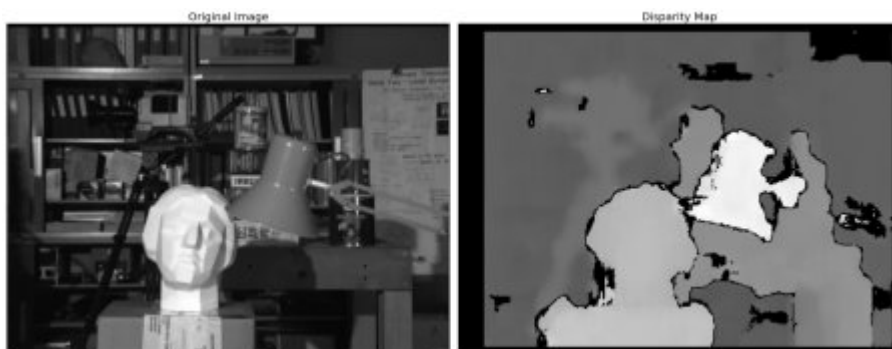
因此，它在两个图像之间找到了对应的匹配项。我们已经看到了Epiline约束如何使此操作更快，更准确。一旦找到匹配项，就会发现差异。让我们看看如何使用OpenCV做到这一点。

代码

下面的代码片段显示了创建视差图的简单过程。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
imgL = cv.imread('tsukuba_l.png',0)
imgR = cv.imread('tsukuba_r.png',0)
stereo = cv.StereoBM_create(numDisparities=16, blockSize=15)
disparity = stereo.compute(imgL,imgR)
plt.imshow(disparity,'gray')
plt.show()
```

下面的图像包含原始图像（左）及其视差图（右）。如你所见，结果受到高度噪声的污染。通过调整`numDisparities`和`blockSize`的值，可以获得更好的结果。



当你熟悉StereoBM时，会有一些参数，可能需要微调参数以获得更好，更平滑的结果。参数：

- `texture_threshold`：过滤出纹理不足以进行可靠匹配
- 区域斑点范围和大小：基于块的匹配器通常会在对象边界附近产生“斑点”，其中匹配窗口捕获一侧的前景和背景 在另一场景中，匹配器似乎还在桌子上投影的纹理中找到小的虚假匹配项。为了消除这些伪像，我们使用由`speckle_size`和`speckle_range`参数控制的散斑滤镜对视差图像进行后处理。`speckle_size`是将视差斑点排除为“斑点”的像素数。`speckle_range`控制必须将值差异视为同一对象的一部分的程度。
- 视差数量：滑动

窗口的像素数。它越大，可见深度的范围就越大，但是需要更多的计算。 - `min_disparity`：从开始搜索的左像素的x位置开始的偏移量。 - `uniqueness_ratio`：另一个后过滤步骤。如果最佳匹配视差不足够好于搜索范围中的所有其他视差，则将像素滤出。如果`texture_threshold`和斑点过滤仍在通过虚假匹配，则可以尝试进行调整。 - `prefilter_size`和`prefilter_cap`：预过滤阶段，可标准化图像亮度并增强纹理，以准备块匹配。通常，你不需要调整这些。

附加资源

- [Ros stereo img processing wiki page](#)

练习

1. OpenCV样本包含生成视差图及其3D重建的示例。查看OpenCV-Python示例代码 `stereo_match.py`

8_1_理解KNN

理解K近邻

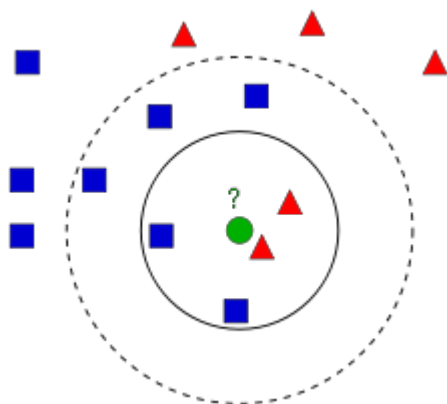
作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本章中，我们将了解k近邻 (kNN) 算法的原理。

理论

kNN是可用于监督学习的最简单的分类算法之一。这个想法是在特征空间中搜索测试数据的最近邻。我们将用下面的图片来研究它。



在图像中，有两个族，蓝色正方形和红色三角形。我们称每一种为**类**。他们的房屋显示在他们的城镇地图中，我们称之为特征空间。（你可以将要素空间视为投影所有数据的空间。例如，考虑一个2D坐标空间。每个数据都有两个要素，x和y坐标。你可以在2D坐标空间中表示此数据，对吧？现在假设如果有三个要素，则需要3D空间；现在考虑N个要素，需要N维空间，对吗？这个N维空间就是其要素空间。在我们的图像中，你可以将其视为2D情况。有两个功能）。

现在有一个新成员进入城镇并创建了一个新房屋，显示为绿色圆圈。他应该被添加到这些蓝色/红色家族之一中。我们称该过程为**分类**。我们所做的？由于我们正在处理kNN，因此让我们应用此算法。

一种方法是检查谁是他的最近邻。从图像中可以明显看出它是红色三角形家族。因此，他也被添加到了红色三角形中。此方法简称为“**最近邻**”，因为分类仅取决于最近邻。

但这是有问题的。红三角可能是最近的。但是，如果附近有很多蓝色方块怎么办？然后，蓝色方块在该地区的权重比红色三角更大。因此，仅检查最接近的一个是不够的。相反，我们检查一些k近邻的族。那么，无论谁占多数，新样本都属于那个族。在我们的图像中，让我们取 $k=3$ ，即3个最近族。他有两个红色和一个蓝色（有两个等距的蓝色，但是由于 $k=3$ ，我们只取其中一个），所以他又应该加入红色家族。但是，如果我们取 $k=7$ 怎么办？然后，他有5个蓝色族和2个红色族。太好了！！现在，他应该加入蓝色族。因此，所有这些都随k的值而变化。更有趣的是，如果 $k=4$ 怎么办？他有2个红色邻居和2个蓝色邻居。这是一个平滑！因此最好将k作为奇数。由于分类取决于k个最近的邻居，因此该方法称为****k近邻****。

同样，在kNN中，我们确实在考虑k个邻居，但我们对所有人都给予同等的重视，对吧？这公平吗？例如，以 $k=4$ 的情况为例。我们说这是平局。但是请注意，这两个红色族比其他两个蓝色族离他更近。因此，他更应该被添加到红色。那么我们如何用数学解释呢？我们根据每个家庭到新来者的距离来给他们一些权重。对于那些靠近他的人，权重增加，而那些远离他的人，权重减轻。然后，我们分别添加每个族的总权重。谁得到的总权重最高，新样本归为那一族。这称为****modified kNN****。

那么你在这里看到的一些重要内容是什么？- 你需要了解镇上所有房屋的信息，对吗？因为，我们必须检查新样本到所有现有房屋的距离，以找到最近的邻居。如果有许多房屋和家庭，则需要大量的内存，并且需要更多的时间进行计算。- 几乎没有时间进行任何形式的训练或准备。

现在让我们在OpenCV中看到它。

OpenCV中的kNN

就像上面一样，我们将在这里做一个简单的例子，有两个族（类）。然后在下一章中，我们将做一个更好的例子。

因此，在这里，我们将红色系列标记为****Class-0****（因此用0表示），将蓝色系列标记为****Class-1****（用1表示）。我们创建25个族或25个训练数据，并将它们标记为0类或1类。我们借助Numpy中的Random Number Generator来完成所有这些工作。

然后我们在Matplotlib的帮助下对其进行绘制。红色系列显示为红色三角形，蓝色系列显示为蓝色正方形。

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
# 包含(x,y)值的25个已知/训练数据的特征集
trainData = np.random.randint(0,100,(25,2)).astype(np.float32)
# 用数字0和1分别标记红色或蓝色
responses = np.random.randint(0,2,(25,1)).astype(np.float32)
# 取红色族并绘图
red = trainData[responses.ravel()==0]
plt.scatter(red[:,0],red[:,1],80,'r','^')
# 取蓝色族并绘图
blue = trainData[responses.ravel()==1]
plt.scatter(blue[:,0],blue[:,1],80,'b','s')
plt.show()
```

你会得到与我们的第一张图片相似的东西。由于你使用的是随机数生成器，因此每次运行代码都将获得不同的数据。

接下来启动kNN算法，并传递trainData和响应以训练kNN（它会构建搜索树）。

然后，我们将在OpenCV中的kNN的帮助下将一个新样本带入一个族并将其分类。在进入kNN之前，我们需要了解测试数据（新样本数据）上的知识。我们的数据应为浮点数组，其大小为 `number\ of\ testdata\times number\ of\ features`。然后我们找到新加入的最近邻。我们可以指定我们想要多少个邻居。它返回：

1. 给新样本的标签取决于我们之前看到的kNN理论。如果要使用“最近邻居”算法，只需指定k=1即可，其中k是邻居数。
2. k最近邻的标签。
3. 衡量新加入到每个最近邻的相应距离。

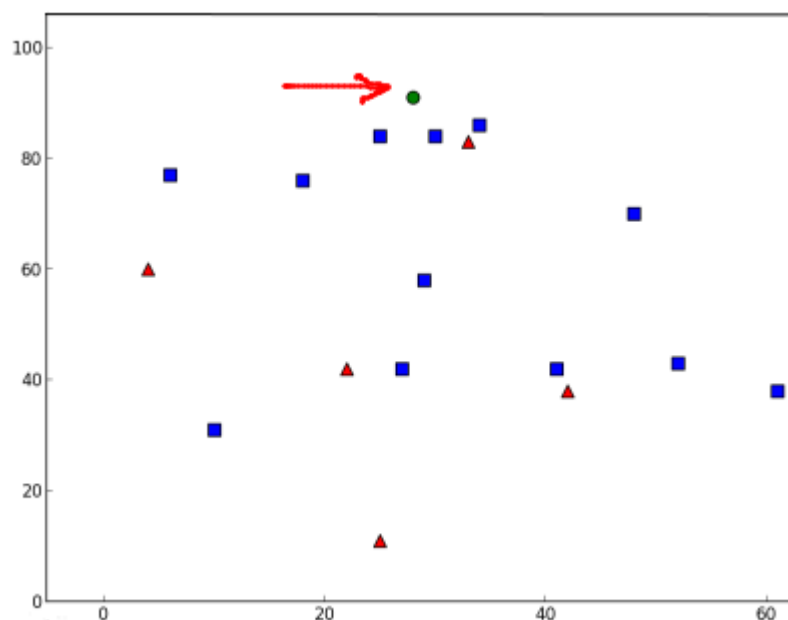
因此，让我们看看它是如何工作的。新样本被标记为绿色。

```
newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)
plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o')
knn = cv.ml.KNearest_create()
knn.train(trainData, cv.ml.ROW_SAMPLE, responses)
ret, results, neighbours, dist = knn.findNearest(newcomer, 3)
print( "result:  {}\n".format(results) )
print( "neighbours:  {}\n".format(neighbours) )
print( "distance:  {}\n".format(dist) )
plt.show()
```

我得到了如下的结果：

```
result:  [[ 1.]]
neighbours:  [[ 1.  1.  1.]]
distance:  [[ 53.  58.  61.]]
```

它说我们的新样本有3个近邻，全部来自Blue家族。因此，他被标记为蓝色家庭。从下面的图可以明显看出：



如果你有大量数据，则可以将其作为数组传递。还获得了相应的结果作为数组。

```
# 10个新加入样本
newcomers = np.random.randint(0,100,(10,2)).astype(np.float32)
ret, results,neighbours,dist = knn.findNearest(newcomer, 3)
# 结果包含10个标签
```

附加资源

1. NPTEL关于模式识别的注释，第11章

练习

8_2_使用OCR手写数据集运行KNN

使用OCR手写数据集运行KNN

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本章中 - 我们将使用我们在kNN上的知识来构建基本的OCR应用程序。 - 我们将尝试使用OpenCV自带的数字和字母数据集。

手写数字的OCR

我们的目标是构建一个可以读取手写数字的应用程序。为此，我们需要一些 `train_data` 和 `test_data`。OpenCV带有一个图片 `digits.png`（在文件夹 `opencv/samples/data/` 中），其中包含 5000 个手写数字（每个数字500个）。每个数字都是 `20x20` 的图像。因此，我们的第一步是将图像分割成 5000 个不同的数字。对于每个数字，我们将其展平为 400 像素的一行。那就是我们的训练集，即所有像素的强度值。这是我们可以创建的最简单的功能集。我们将每个数字的前 250 个样本用作 `train_data`，然后将 250 个样本用作 `test_data`。因此，让我们先准备它们。

```
import numpy as np
import cv2 as cv
img = cv.imread('digits.png')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# 现在我们将图像分割为5000个单元格，每个单元格为20x20
cells = [np.hsplit(row, 100) for row in np.vsplit(gray, 50)]
# 使其成为一个Numpy数组。它的大小将是 (50, 100, 20, 20)
x = np.array(cells)
# 现在我们准备train_data和test_data。
train = x[:, :50].reshape(-1, 400).astype(np.float32) # Size = (2500, 400)
test = x[:, 50:100].reshape(-1, 400).astype(np.float32) # Size = (2500, 400)
# 为训练和测试数据创建标签
k = np.arange(10)
train_labels = np.repeat(k, 250)[:, np.newaxis]
test_labels = train_labels.copy()
# 初始化kNN，训练数据，然后使用k = 1的测试数据对其进行测试
knn = cv.ml.KNearest_create()
knn.train(train, cv.ml.ROW_SAMPLE, train_labels)
ret, result, neighbours, dist = knn.findNearest(test, k=5)
```

```
# 现在，我们检查分类的准确性
#为此，将结果与test_labels进行比较，并检查哪个错误
matches = result==test_labels
correct = np.count_nonzero(matches)
accuracy = correct*100.0/result.size
print( accuracy )
```

因此，我们的基本OCR应用程序已准备就绪。这个特定的例子给我的准确性是91%。一种提高准确性的选择是添加更多数据进行训练，尤其是错误的数据。因此，与其每次启动应用程序时都找不到该培训数据，不如将其保存，以便下次我直接从文件中读取此数据并开始分类。您可以借助一些Numpy函数（例如np.savetxt，np.savez，np.load等）来完成此操作。请查看其文档以获取更多详细信息。

```
# 保存数据
np.savez('knn_data.npz',train=train, train_labels=train_labels)
# 现在加载数据
with np.load('knn_data.npz') as data:
    print( data.files )
    train = data['train']
    train_labels = data['train_labels']
```

在我的系统中，它需要大约 4.4 MB 的内存。由于我们使用强度值（uint8数据）作为特征，因此最好先将数据转换为 np.uint8，然后再将其保存。在这种情况下，仅占用 1.1 MB。然后在加载时，您可以转换回 float32。

英文字母的OCR

接下来，我们将对英语字母执行相同的操作，但是数据和功能集会稍有变化。在这里，OpenCV代替了图像，而在 `opencv/samples/cpp/` 文件夹中附带了一个数据文件 `letter-recognition.data`。如果打开它，您将看到20000行，乍一看可能看起来像垃圾。实际上，在每一行中，第一列是一个字母，这是我们的标签。接下来的16个数字是它的不同功能。这些功能是从UCI机器学习存储库获得的。您可以在此页面中找到这些功能的详细信息。现有20000个样本，因此我们将前10000个数据作为训练样本，其余10000个作为测试样本。我们应该将字母更改为ASCII字符，因为我们不能直接使用字母。

```
import cv2 as cv
import numpy as np
# 加载数据，转换器将字母转换为数字
data= np.loadtxt('letter-recognition.data', dtype= 'float32', delimiter = ',',
                 converters= {0: lambda ch: ord(ch)-ord('A')})
```

```
# 将数据分为两个, 每个10000个以进行训练和测试
train, test = np.vsplit(data, 2)
# 将火车数据和测试数据拆分为特征和响应
responses, trainData = np.hsplit(train, [1])
labels, testData = np.hsplit(test, [1])
# 初始化kNN, 分类, 测量准确性
knn = cv.ml.KNearest_create()
knn.train(trainData, cv.ml.ROW_SAMPLE, responses)
ret, result, neighbours, dist = knn.findNearest(testData, k=5)
correct = np.count_nonzero(result == labels)
accuracy = correct*100.0/10000
print( accuracy )
```

它给我的准确性为 93.22%。同样，如果要提高准确性，则可以迭代地在每个级别中添加错误数据。

附加资源

练习

8_3_理解SVM

理解SVM

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

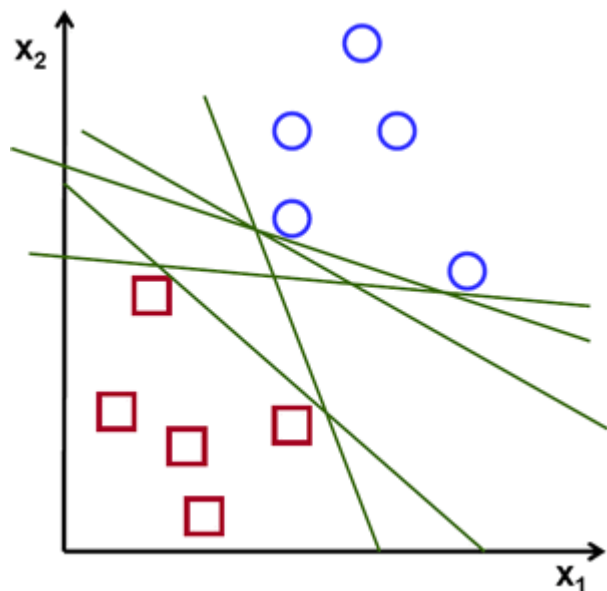
目标

在这一章中 - 我们将对SVM有一个直观的了解

理论

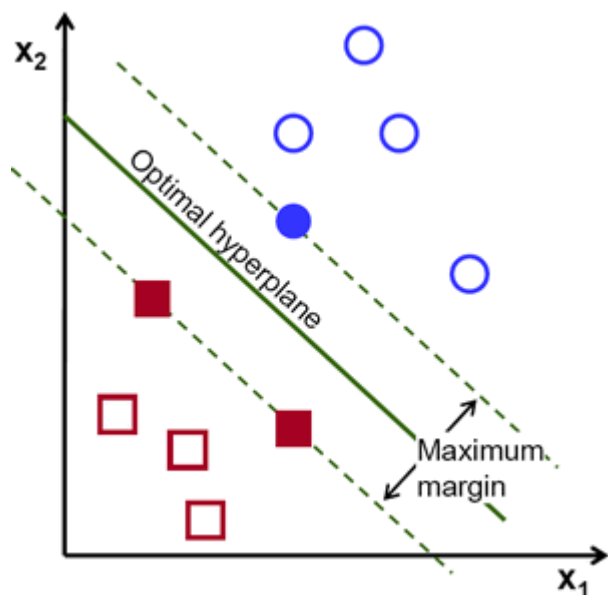
线性可分数据

考虑下面的图像，它具有两种数据类型，红色和蓝色。在kNN中，对于测试数据，我们用来测量其与所有训练样本的距离，并以最小的距离作为样本。测量所有距离都需要花费大量时间，并且需要大量内存来存储所有训练样本。但是考虑到图像中给出的数据，我们是否需要那么多？



考虑另一个想法。我们找到一条线 $f(x)=ax_1 + bx_2+c$ ，它将两条数据都分为两个区域。当我们得到一个新的test_data X时，只需将其替换为 $f(x)$ 即可。如果 $f(X)> 0$ ，则属于蓝色组，否则属于红色组。我们可以将此行称为“**决策边界**”。它非常简单且内存高效。可以将这些数据用直线（或高维超平面）一分为二的数据称为**线性可分离**数据。

因此，在上图中，你可以看到很多这样的行都是可能的。我们会选哪一个？非常直观地，我们可以说直线应该从所有点尽可能远地经过。为什么？因为传入的数据中可能会有噪音。此数据不应影响分类准确性。因此，走最远的分离线将提供更大的抗干扰能力。因此，SVM要做的是找到到训练样本的最小距离最大的直线（或超平面）。请参阅下面图像中穿过中心的粗线。



因此，要找到此决策边界，你需要训练数据。那么需要全部吗？并不用。仅接近相反组的那些就足够了。在我们的图像中，它们是一个蓝色填充的圆圈和两个红色填充的正方形。我们可以称其为**支撑向量**，通过它们的线称为**支撑平面**。它们足以找到我们的决策边界。我们不必担心所有数据。它有助于减少数据量。

接下来，找到了最能代表数据的前两个超平面。例如，蓝色数据由 $w^T x + b_0 > -1$ 表示，红色数据由 $w^T x + b_0 < -1$ 表示，其中 w 是**权重向量** ($w = [w_1, w_2, \dots, w_n]$)， x 是特征向量 ($x = [x_1, x_2, \dots, x_n]$)。 b_0 是**偏置**。权重矢量确定决策边界的方向，而偏置点确定其位置。现在，将决策边界定义为这些超平面之间的中间，因此表示为 $w^T x + b_0 = 0$ 。从支持向量到决策边界的最小距离由 $\text{distance}_{\text{support vectors}} = \frac{1}{\|w\|}$ 给出。间隔是此距离的两倍，因此我们需要最大化此间隔。也就是说，我们需要使用一些约束来最小化新函数 $L(w, b_0)$ ，这些约束可以表示如下：

$$\min_{w, b_0} L(w, b_0) = \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad t_i (w^T x + b_0) \geq 1 \quad \text{for all } i$$

其中 t_i 是每类的标签， $t_i \in [-1, 1]$ 。

非线性可分数据

考虑一些不能用直线分成两部分的数据。例如，考虑一维数据，其中'X'位于-3和+3，而'O'位于-1和+1。显然，它不是线性可分离的。但是有解决这些问题的方法。如果我们可以使用函数 $f(x)=x^2$ 映射此数据集，则在线性可分离的9处获得'X'，在1处获得'O'。

否则，我们可以将此一维数据转换为二维数据。我们可以使用 $f(x)=(x, x^2)$ 函数来映射此数据。然后，'X'变成(-3,9)和(3,9)，而'O'变成(-1,1)和(1,1)。这也是线性可分的。简而言之，低维空间中的非线性可分离数据更有可能在高维空间中变为线性可分离。

通常，可以将 d 维空间中的点映射到某个 D 维空间($D > d$)，以检查线性可分离性的可能性。有一个想法可以通过在低维输入（特征）空间中执行计算来帮助在高维（内核）空间中计算点积。我们可以用下面的例子来说明。

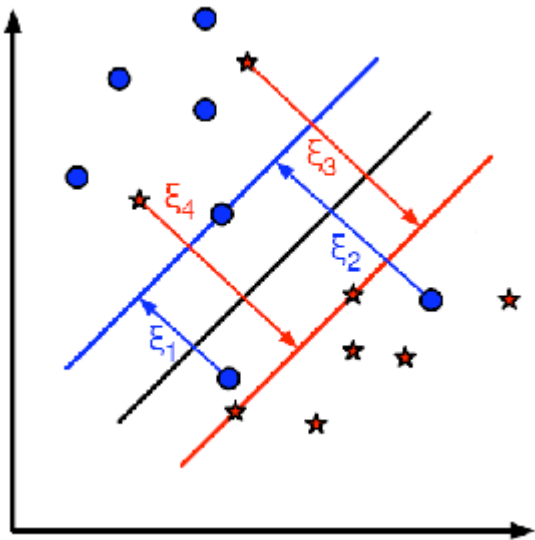
考虑二维空间中的两个点， $p=(p_1, p_2)$ 和 $q=(q_1, q_2)$ 。令 ϕ 为映射函数，它将二维点映射到三维空间，如下所示： $\phi(p)=(p_1^2, p_2^2, \sqrt{2}p_1p_2)$ $\phi(q)=(q_1^2, q_2^2, \sqrt{2}q_1q_2)$

让我们定义一个核函数 $K(p,q)$ ，该函数在两点之间做一个点积，如下所示：

$$\begin{aligned} K(p,q) &= \phi(p) \cdot \phi(q) = \phi(p)^T \phi(q) = (p_1^2, p_2^2, \sqrt{2} p_1 p_2) \cdot (q_1^2, q_2^2, \sqrt{2} q_1 q_2) \\ &= p_1^2 q_1^2 + p_2^2 q_2^2 + 2 p_1 p_2 q_1 q_2 = (p_1 q_1 + p_2 q_2)^2 = \phi(p) \cdot \phi(q) = (p \cdot q)^2 \end{aligned}$$

这意味着，可以使用二维空间中的平方点积来实现三维空间中的点积。这可以应用于更高维度的空间。因此，我们可以从较低尺寸本身计算较高尺寸的特征。一旦将它们映射，我们将获得更高的空间。

除了所有这些概念之外，还存在分类错误的问题。因此，仅找到具有最大间隔的决策边界是不够的。我们还需要考虑分类错误的问题。有时，可能会找到间隔较少但分类错误减少的决策边界。无论如何，我们需要修改我们的模型，以便它可以找到具有最大间隔但分类错误较少的决策边界。最小化标准修改为： $\min \|w\|^2 + C$ （分类错误的样本到其正确区域的距离）下图显示了此概念。对于训练数据的每个样本，定义一个新的参数 ξ_i 。它是从其相应的训练样本到其正确决策区域的距离。对于那些未分类错误的样本，它们落在相应的支撑平面上，因此它们的距离为零。



因此，新的优化函数为： $\min_{w, b_0} L(w, b_0) = \|w\|^2 + C \sum_i \xi_i$ subject to $y_i(w^T x_i + b_0) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

如何选择参数 C ？显然，这个问题的答案取决于训练数据的分布方式。尽管没有一般性的答案，但考虑以下规则是很有用的：- C 的值越大，解决方案的分类错误越少，但宽度也越小。考虑到在这种情况下，进行错误分类是错误的。由于优化的目的是最小化参数，因此几乎没有误分类的错误。- C 的值越小，解决方案的宽度就越大，分类误差也越大。在这种情况下，最小化对总和项的考虑不多，因此它更多地集中在寻找具有大间隔的超平面上。

附加资源¶

1. NPTEL notes on Statistical Pattern Recognition, Chapters 25-29.

练习¶

8_4_使用OCR手写数据集运行SVM

理解SVM

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本章中，我们将重新识别手写数据集，但是使用SVM而不是kNN。

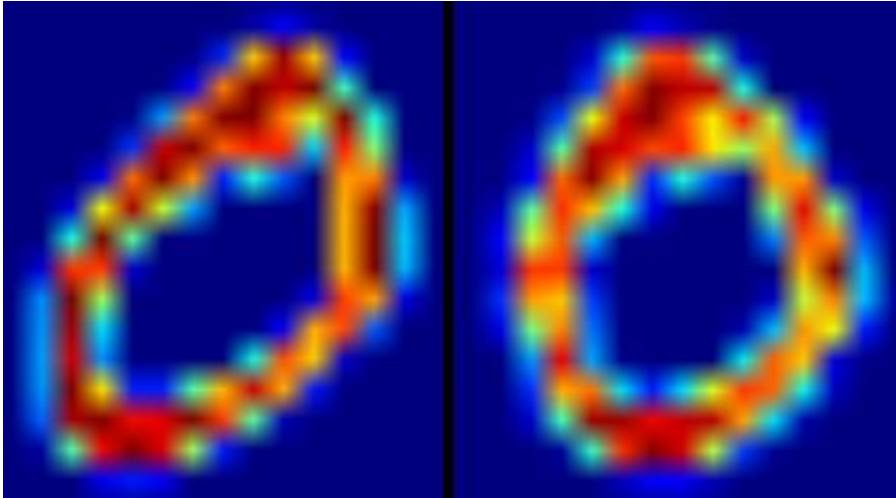
识别手写数字

在kNN中，我们直接使用像素强度作为特征向量。这次我们将使用定向梯度直方图(HOG)作为特征向量。

在这里，在找到HOG之前，我们使用其二阶矩对图像进行偏斜校正。因此，我们首先定义一个函数**deskew()**，该函数获取一个数字图像并将其校正。下面是deskew()函数：

```
def deskew(img):  
    m = cv.moments(img)  
    if abs(m['mu02']) < 1e-2:  
        return img.copy()  
    skew = m['mu11']/m['mu02']  
    M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])  
    img = cv.warpAffine(img, M, (SZ, SZ), flags=affine_flags)  
    return img
```

下图显示了应用于零图像的上偏移校正功能。左图像是原始图像，右图像是偏移校正后的图像。



接下来，我们必须找到每个单元格的HOG描述符。为此，我们找到了每个单元在X和Y方向上的Sobel导数。然后在每个像素处找到它们的大小和梯度方向。该梯度被量化为16个整数值。将此图像划分为四个子正方形。对于每个子正方形，计算权重大小方向的直方图（16个bin）。因此，每个子正方形为你提供了一个包含16个值的向量。（四个子正方形的）四个这样的向量共同为我们提供了一个包含64个值的特征向量。这是我们用于训练数据的特征向量。

```
def hog(img):
    gx = cv.Sobel(img, cv.CV_32F, 1, 0)
    gy = cv.Sobel(img, cv.CV_32F, 0, 1)
    mag, ang = cv.cartToPolar(gx, gy)
    bins = np.int32(bin_n*ang/(2*np.pi))    # quantizing binvalues in (0...16)
    bin_cells = bins[:10,:10], bins[10:,:10], bins[:10,10:], bins[10:,10:]
    mag_cells = mag[:10,:10], mag[10:,:10], mag[:10,10:], mag[10:,10:]
    hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells,
        mag_cells)]
    hist = np.hstack(hists)         # hist is a 64 bit vector
    return hist
```

最后，与前面的情况一样，我们首先将大数据集拆分为单个单元格。对于每个数字，保留250个单元用于训练数据，其余250个数据保留用于测试。完整的代码如下，你也可以从此处下载：

```
#!/usr/bin/env python
import cv2 as cv
import numpy as np
SZ=20
bin_n = 16 # Number of bins
affine_flags = cv.WARP_INVERSE_MAP|cv.INTER_LINEAR
def deskew(img):
    m = cv.moments(img)
    if abs(m['mu02']) < 1e-2:
        return img.copy()
```

```
skew = m['mu11']/m['mu02']
M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])
img = cv.warpAffine(img,M,(SZ, SZ),flags=affine_flags)
return img
def hog(img):
    gx = cv.Sobel(img, cv.CV_32F, 1, 0)
    gy = cv.Sobel(img, cv.CV_32F, 0, 1)
    mag, ang = cv.cartToPolar(gx, gy)
    bins = np.int32(bin_n*ang/(2*np.pi))    # quantizing binvalues in (0...16)
    bin_cells = bins[:10,:10], bins[10:,:10], bins[:10,10:], bins[10:,10:]
    mag_cells = mag[:10,:10], mag[10:,:10], mag[:10,10:], mag[10:,10:]
    hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells,
mag_cells)]
    hist = np.hstack(hists)         # hist is a 64 bit vector
    return hist
img = cv.imread('digits.png',0)
if img is None:
    raise Exception("we need the digits.png image from samples/data here !")
cells = [np.hsplit(row,100) for row in np.vsplit(img,50)]
# First half is trainData, remaining is testData
train_cells = [ i[:50] for i in cells ]
test_cells = [ i[50:] for i in cells]
deskewed = [list(map(deskew,row)) for row in train_cells]
hogdata = [list(map(hog,row)) for row in deskewed]
trainData = np.float32(hogdata).reshape(-1,64)
responses = np.repeat(np.arange(10),250)[: ,np.newaxis]
svm = cv.ml.SVM_create()
svm.setKernel(cv.ml.SVM_LINEAR)
svm.setType(cv.ml.SVM_C_SVC)
svm.setC(2.67)
svm.setGamma(5.383)
svm.train(trainData, cv.ml.ROW_SAMPLE, responses)
svm.save('svm_data.dat')
deskewed = [list(map(deskew,row)) for row in test_cells]
hogdata = [list(map(hog,row)) for row in deskewed]
testData = np.float32(hogdata).reshape(-1,bin_n*4)
result = svm.predict(testData)[1]
mask = result==responses
correct = np.count_nonzero(mask)
print(correct*100.0/result.size)
```

这种特殊的方法给我们近94%的准确性。你可以为SVM的各种参数尝试不同的值，以检查是否可以实现更高的精度。或者，你可以阅读有关此领域的技术论文并尝试实施它们。

附加资源

1. Histograms of Oriented Gradients Video : <https://www.youtube.com/watch?v=0Zib1YEE4LU>

练习

1. OpenCV示例包含digits.py，它对上述方法进行了一些改进以得到改进的结果。它还包含参考资料。检查并了解它。

理解K-Means聚类

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

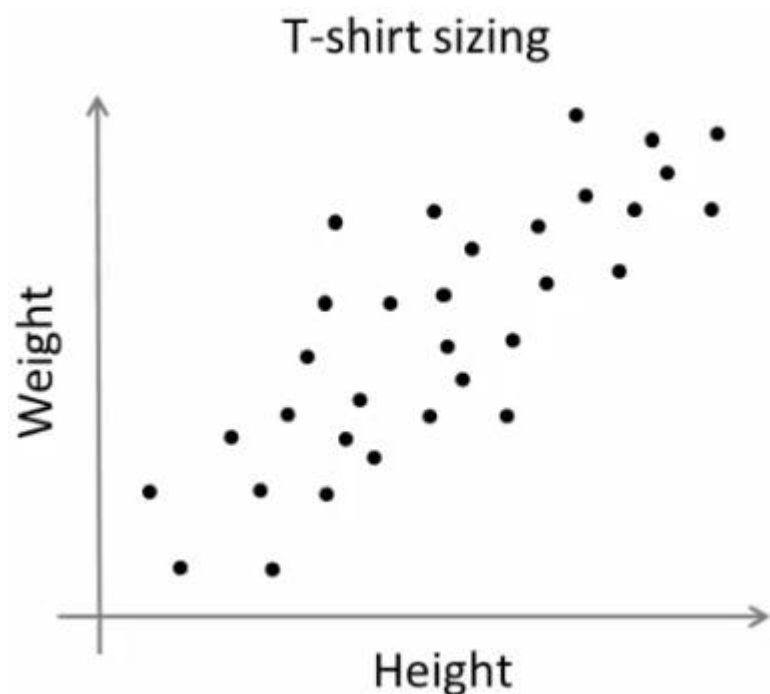
在本章中，我们将了解K-Means聚类的概念，其工作原理等。

理论

我们将用一个常用的例子来处理这个问题。

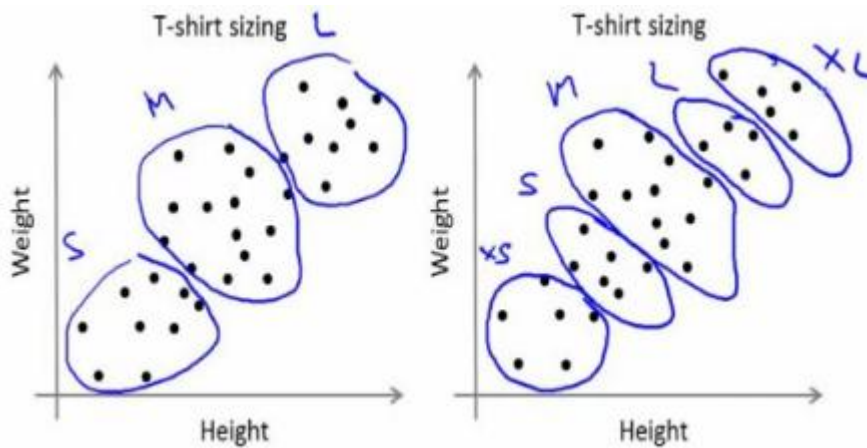
T-shirt尺寸问题

考虑一家公司，该公司将向市场发布新型号的T恤。显然，他们将不得不制造不同尺寸的模型，以满足各种规模的人们需求。因此，该公司会记录人们的身高和体重数据，并将其绘制到图形上，如下所示：



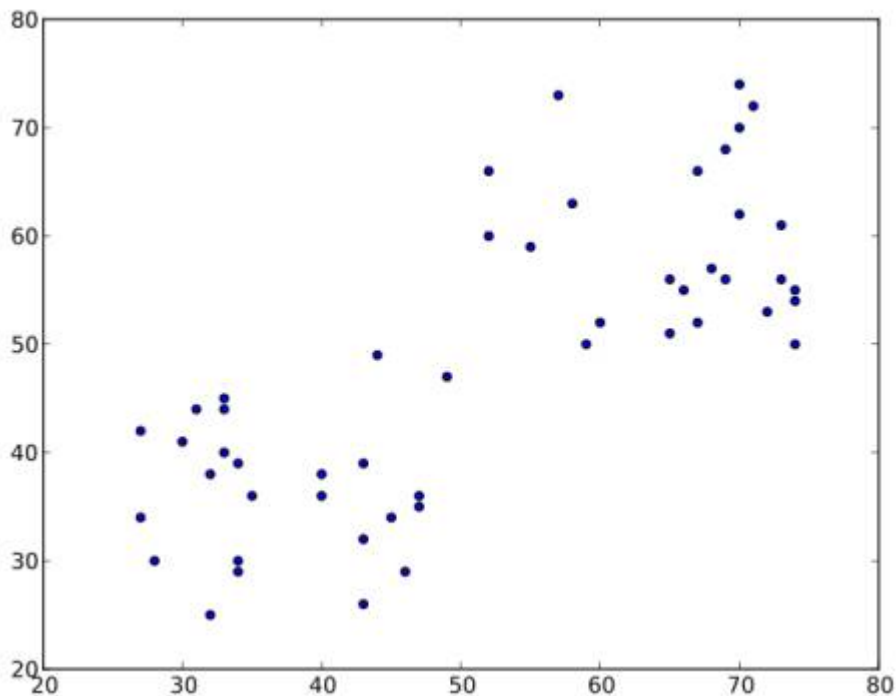
公司无法制作所有尺寸的T恤。取而代之的是，他们将人划分为小，中和大，并仅制造这三种适合所有人的模型。可以通过k均值聚类将人员分为三组，并且算法可以为我们提供最佳的3种大小，

这将满足所有人员的需求。如果不是这样，公司可以将人员分为更多的组，可能是五个，依此类推。查看下面的图片：



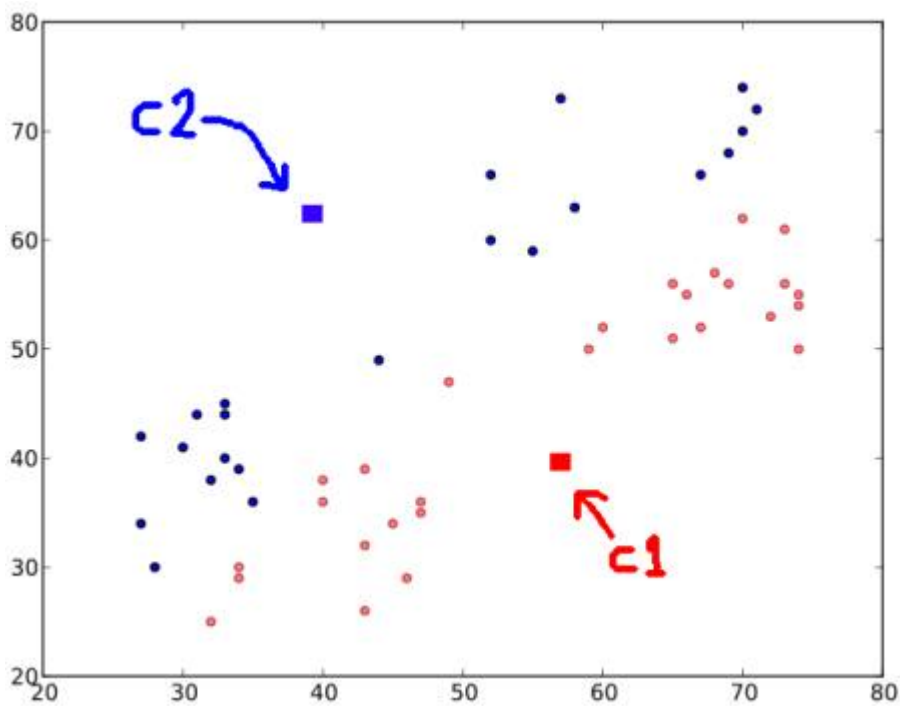
如何起作用？

该算法是一个迭代过程。我们将在图像的帮助下逐步解释它。考虑如下一组数据（您可以将其视为T恤问题）。我们需要将此数据分为两类。



步骤:1 -算法随机选择两个质心 C_1 和 C_2 （有时，将任何两个数据作为质心）。**步骤:2** -计算每个点到两个质心的距离。如果测试数据更接近 C_1 ，则该数据标记为“0”。如果它更靠近 C_2 ，则标记为“1”（如果存在更多质心，则标记为“2”，“3”等）。

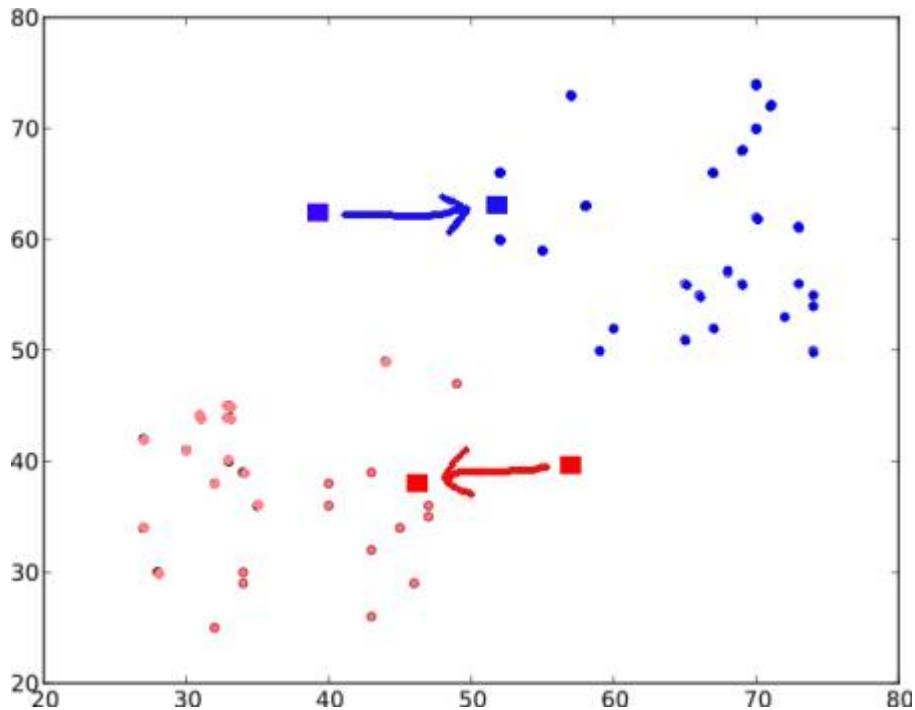
在我们的示例中，我们将为所有标记为红色的“0”和标记为蓝色的所有“1”上色。因此，经过以上操作，我们得到以下图像。



步骤:3 -接下来，我们分别计算所有蓝点和红点的平均值，这将成为我们的新质心。即C_1和C_2转移到新计算的质心。（请记住，显示的图像不是真实值，也不是真实比例，仅用于演示）。

再次，使用新的质心执行步骤2，并将标签数据设置为‘0’和‘1’。

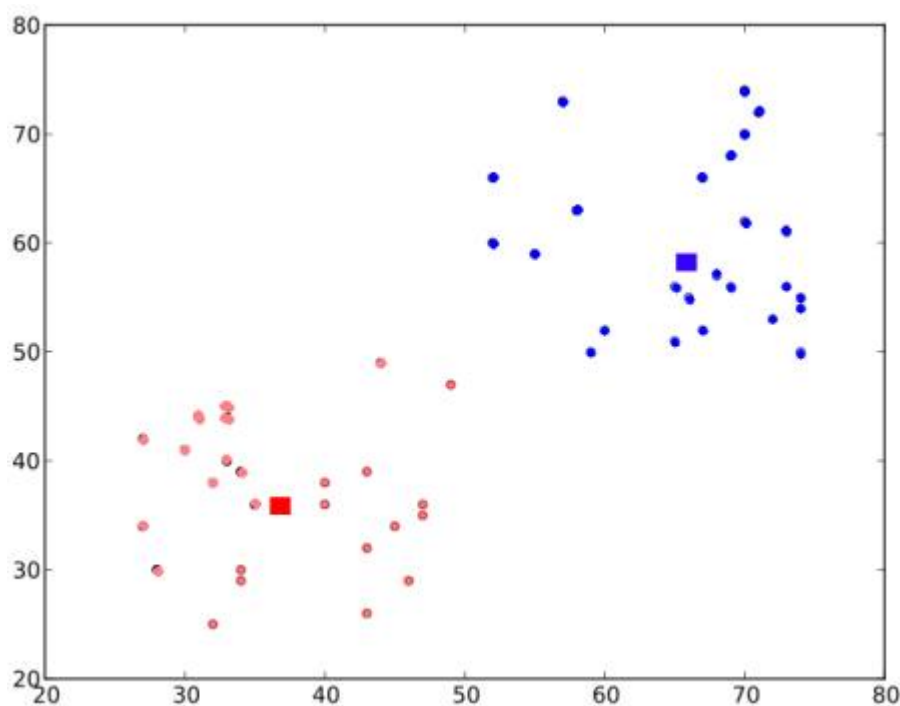
所以我们得到如下结果：



现在，迭代**步骤2**和**步骤3**，直到两个质心都收敛到固定点。（或者可以根据我们提供的标准（例如最大的迭代次数或达到特定的精度等）将其停止。）***这些点使测试数据与其对应质心之间的距离之和最小*。或者简单地说， $C_1 \leftrightarrow \text{Red_Points}$ 和 $C_2 \leftrightarrow \text{Blue_Points}$ 之间的距离之和最小。

minimize

$$\bigg[J = \sum_{\text{All: Red_Points}} \text{distance}(C_1, \text{Red_Point}) + \sum_{\text{All: Blue_Points}} \text{distance}(C_2, \text{Blue_Point}) \bigg]$$



最终结果如下所示：

因此，这仅仅是对K-Means聚类的直观理解。有关更多详细信息和数学解释，请阅读任何标准的机器学习教科书或查看其他资源中的链接。它只是K-Means聚类的宏观层面。此算法有很多修改，例如如何选择初始质心，如何加快迭代过程等。

附加资源

1. Machine Learning Course, Video lectures by Prof. Andrew Ng (Some of the images are taken from this)

练习s

8_6_OpenCV中的K均值

OpenCV中的K-Means聚类

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

- 了解如何在OpenCV中使用cv.kmeans()函数进行数据聚类

理解参数

输入参数

1. **sample** : 它应该是**np.float32**数据类型，并且每个功能都应该放在单个列中。
2. **nclusters(K)** : 结束条件所需的簇数
3. **criteria** : 这是迭代终止条件。满足此条件后，算法迭代将停止。实际上，它应该是3个参数的元组。它们是 `(type,max_iter,epsilon)` : a. 终止条件的类型。它具有3个标志，如下所示：
 - **cv.TERM_CRITERIA_EPS**-如果达到指定的精度epsilon，则停止算法迭代。
 - **cv.TERM_CRITERIA_MAX_ITER**-在指定的迭代次数max_iter之后停止算法。
 - **cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER**-当满足上述任何条件时，停止迭代。b. max_iter-一个整数，指定最大迭代次数。 c. epsilon-要求的精度 1. attempts : 该标志用于指定使用不同的初始标签执行算法的次数。该算法返回产生最佳紧密度的标签。该紧密性作为输出返回。 2. flags : 此标志用于指定初始中心的获取方式。通常，为此使用两个标志：**cv.KMEANS_PP_CENTERS****和****cv.KMEANS_RANDOM_CENTERS**。

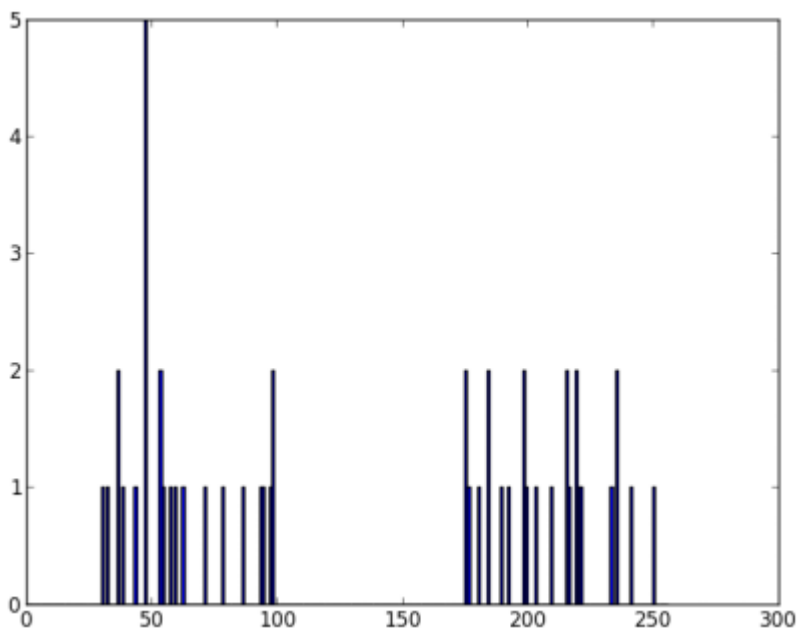
输出参数 1. 紧凑度：它是每个点到其相应中心的平方距离的总和。 2. 标签：这是标签数组（与上一篇文章中的“代码”相同），其中每个元素标记为“0”，“1”..... 3. 中心：这是群集中心的阵列。现在，我们将通过三个示例了解如何应用K-Means算法。

1. 单特征数据

考虑一下，你有一组仅具有一个特征（即一维）的数据。例如，我们可以解决我们的T恤问题，你只用身高来决定T恤的尺寸。因此，我们首先创建数据并将其绘制在Matplotlib中

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
x = np.random.randint(25,100,25)
y = np.random.randint(175,255,25)
z = np.hstack((x,y))
z = z.reshape((50,1))
z = np.float32(z)
plt.hist(z,256,[0,256]),plt.show()
```

因此，我们有了“z”，它是一个大小为50的数组，值的范围是0到255。我将“z”重塑为列向量。如果存在多个功能，它将更加有用。然后我制作了np.float32类型的数据。我们得到以下图像：



现在我们应用KMeans函数。在此之前，我们需要指定标准。我的标准是，每当运行10次算法迭代或达到epsilon = 1.0的精度时，就停止算法并返回答案。

```
# 定义终止标准 = ( type, max_iter = 10 , epsilon = 1.0 )
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
# 设置标志
flags = cv.KMEANS_RANDOM_CENTERS
```

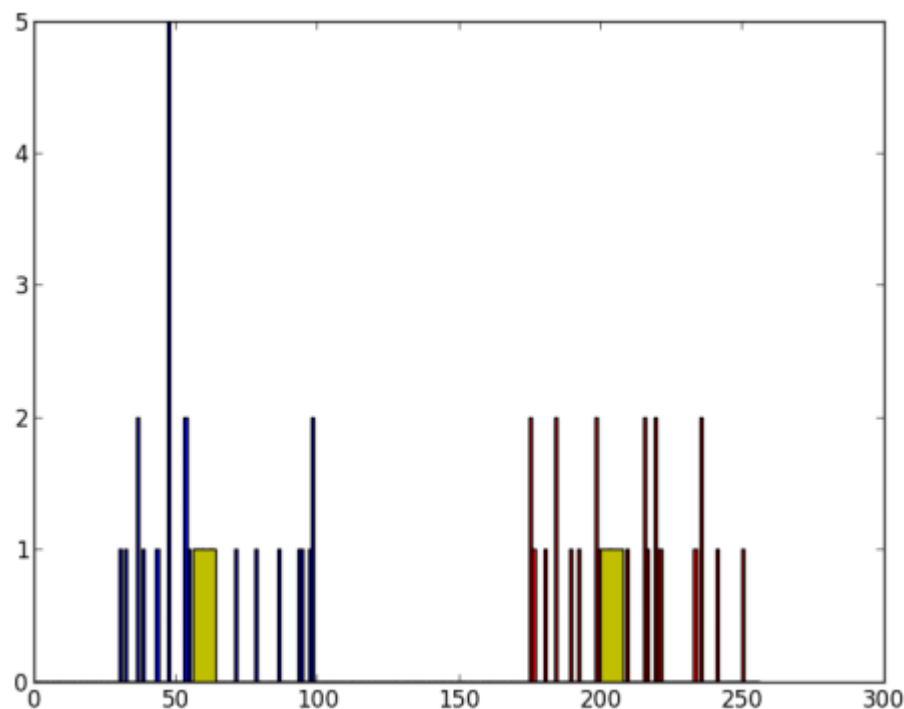
```
# 应用K均值
compactness, labels, centers = cv.kmeans(z, 2, None, criteria, 10, flags)
```

这为我们提供了紧凑性，标签和中心。在这种情况下，我得到的中心分别为60和207。标签的大小将与测试数据的大小相同，其中每个数据的质心都将标记为“0”，“1”，“2”等。现在，我们根据标签将数据分为不同的群集。

```
A = z[labels==0]
B = z[labels==1]
```

现在我们以红色绘制A，以蓝色绘制B，以黄色绘制其质心。

```
# 现在绘制用红色'A'，用蓝色绘制'B'，用黄色绘制中心
plt.hist(A, 256, [0, 256], color = 'r')
plt.hist(B, 256, [0, 256], color = 'b')
plt.hist(centers, 32, [0, 256], color = 'y')
plt.show()
```



得到了以下结果：

2. 多特征数据

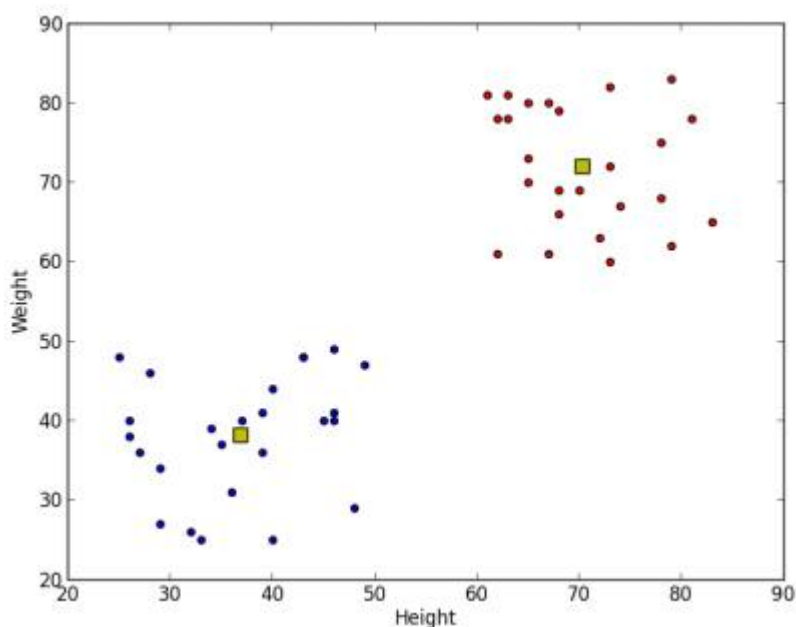
在前面的示例中，我们仅考虑了T恤问题的身高。在这里，我们将同时考虑身高和体重，即两个特征。请记住，在以前的情况下，我们将数据制作为单个列向量。每个特征排列在一列中，而每一行对应于一个输入测试样本。例如，在这种情况下，我们设置了一个大小为50x2的测试数据，即

50人的身高和体重。第一列对应于全部50个人的身高，第二列对应于他们的体重。第一行包含两个元素，其中第一个是第一人称的身高，第二个是他的体重。类似地，其余的行对应于其他人的身高和体重。查看下面的图片：

Features		
	Height	Weight
Person 1	H1	W1
Person 2	H2	W2
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
Person 50	H50	W50

现在，我直接转到代码：

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
X = np.random.randint(25, 50, (25, 2))
Y = np.random.randint(60, 85, (25, 2))
Z = np.vstack((X, Y))
# 将数据转换未 np.float32
Z = np.float32(Z)
# 定义停止标准，应用K均值
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
ret, label, center = cv.kmeans(Z, 2, None, criteria, 10, cv.KMEANS_RANDOM_CENTERS)
# 现在分离数据，Note the flatten()
A = Z[label.ravel() == 0]
B = Z[label.ravel() == 1]
# 绘制数据
plt.scatter(A[:, 0], A[:, 1])
plt.scatter(B[:, 0], B[:, 1], c = 'r')
plt.scatter(center[:, 0], center[:, 1], s = 80, c = 'y', marker = 's')
plt.xlabel('Height'), plt.ylabel('Weight')
plt.show()
```



我们得到如下结果：

3.颜色量化

颜色量化是减少图像中颜色数量的过程。这样做的原因之一是减少内存。有时，某些设备可能会受到限制，因此只能产生有限数量的颜色。同样在那些情况下，执行颜色量化。在这里，我们使用k均值聚类进行颜色量化。

这里没有新内容要解释。有3个特征，例如R,G,B。因此，我们需要将图像重塑为Mx3大小的数组（M是图像中的像素数）。在聚类之后，我们将质心值（也是R,G,B）应用于所有像素，以使生成的图像具有指定数量的颜色。再一次，我们需要将其重塑为原始图像的形状。下面是代码：

```
import numpy as np
import cv2 as cv
img = cv.imread('home.jpg')
Z = img.reshape((-1,3))
# 将数据转化为np.float32
Z = np.float32(Z)
# 定义终止标准 聚类数并应用k均值
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 8
ret,label,center=cv.kmeans(Z,K,None,criteria,10,cv.KMEANS_RANDOM_CENTERS)
# 现在将数据转化为uint8, 并绘制原图像
center = np.uint8(center)
res = center[label.flatten()]
res2 = res.reshape((img.shape))
cv.imshow('res2',res2)
cv.waitKey(0)
cv.destroyAllWindows()
```

我们可以看的K=8的结果



附加资源

练习

9_1_图像去噪

图像去噪

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本章中， - 你将学习用于去除图像中噪声的非局部均值去噪算法。 - 你将看到不同的函数，例如 `cv.fastNlMeansDenoising()`，`cv.fastNlMeansDenoisingColored()`等。

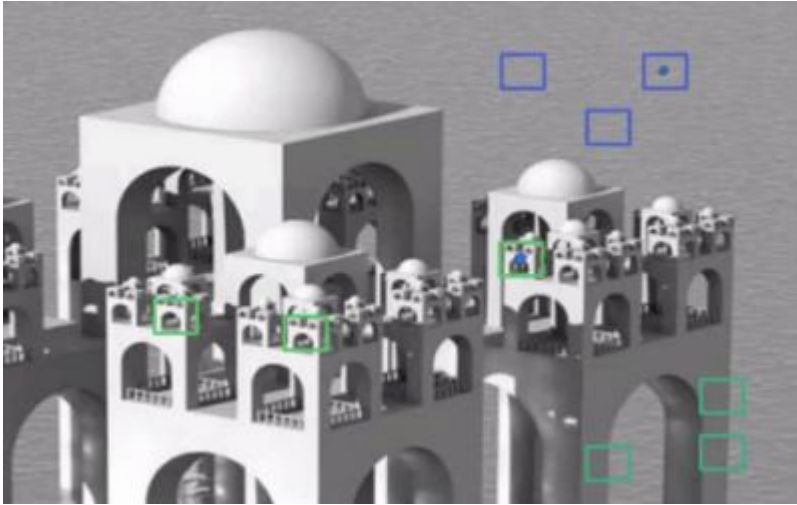
理论

在前面的章节中，我们已经看到了许多图像平滑技术，例如高斯模糊，中值模糊等，它们在某种程度上可以消除少量噪声。在这些技术中，我们在像素周围采取了一个较小的邻域，并进行了一些操作，例如高斯加权平均值，值的中位数等来替换中心元素。简而言之，在像素处去除噪声是其周围的局部现象。有噪声的性质。

通常认为噪声是零均值的随机变量。考虑一个有噪声的像素， $p = p_0 + n$ ，其中 p_0 是像素的真实值， n 是该像素中的噪声。你可以从不同的图像中获取大量相同的像素（例如 N ）并计算其平均值。理想情况下，由于噪声的平均值为零，因此应该得到 $p = p_0$ 。

你可以通过简单的设置自己进行验证。将静态相机固定在某个位置几秒钟。这将为你提供很多帧或同一场景的很多图像。然后编写一段代码，找到视频中所有帧的平均值（这对你现在应该太简单了）。比较最终结果和第一帧。你会看到噪声减少。不幸的是，这种简单的方法对摄像机和场景的运动并不稳健。通常，只有一张嘈杂的图像可用。

因此想法很简单，我们需要一组相似的图像来平均噪声。考虑图像中的一个小窗口（例如 5×5 窗口）。很有可能同一修补程序可能位于图像中的其他位置。有时在它周围的一个小社区中。一起使用这些相似的补丁并找到它们的平均值怎么办？对于那个特定的窗口，这很好。请参阅下面的示例图片：



图像中的蓝色补丁看起来很相似。绿色补丁看起来很相似。因此，我们获取一个像素，在其周围获取一个小窗口，在图像中搜索相似的窗口，对所有窗口求平均，然后用得到的结果替换该像素。此方法是“非本地均值消噪”。与我们之前看到的模糊技术相比，它花费了更多时间，但是效果非常好。更多信息和在线演示可在其他资源的第一个链接中找到。

对于彩色图像，图像将转换为CIELAB色彩空间，然后分别对L和AB分量进行降噪。

OpenCV中的图像去噪

OpenCV提供了此方法的四个变体。

1. **cv.fastNlMeansDenoising()**-处理单个灰度图像
2. **cv.fastNlMeansDenoisingColored()**-处理彩色图像。
3. **cv.fastNlMeansDenoisingMulti()**-处理在短时间内捕获的图像序列（灰度图像）
4. **cv.fastNlMeansDenoisingColoredMulti()**-与上面相同，但用于彩色图像。

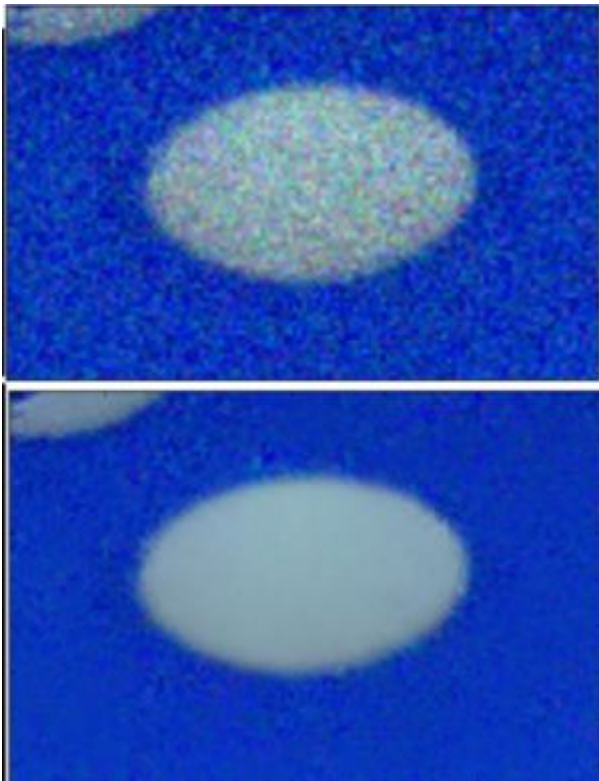
常用参数为：- h：决定滤波器强度的参数。较高的h值可以更好地消除噪点，但同时也可以消除图像细节。（可以设为10）- hForColorComponents：与h相同，但仅用于彩色图像。（通常与h相同）- templateWindowSize：应为奇数。（建议设为7）- searchWindowSize：应为奇数。（建议设为21）

请访问其他资源中的第一个链接，以获取有关这些参数的更多详细信息。我们将在此处演示2和3。剩下的留给你。

1. **cv.fastNlMeansDenoisingColored()** 如上所述，它用于消除彩色图像中的噪点。（噪声可能是高斯的）。请参阅以下示例：

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('die.png')
dst = cv.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
plt.subplot(121), plt.imshow(img)
plt.subplot(122), plt.imshow(dst)
plt.show()
```

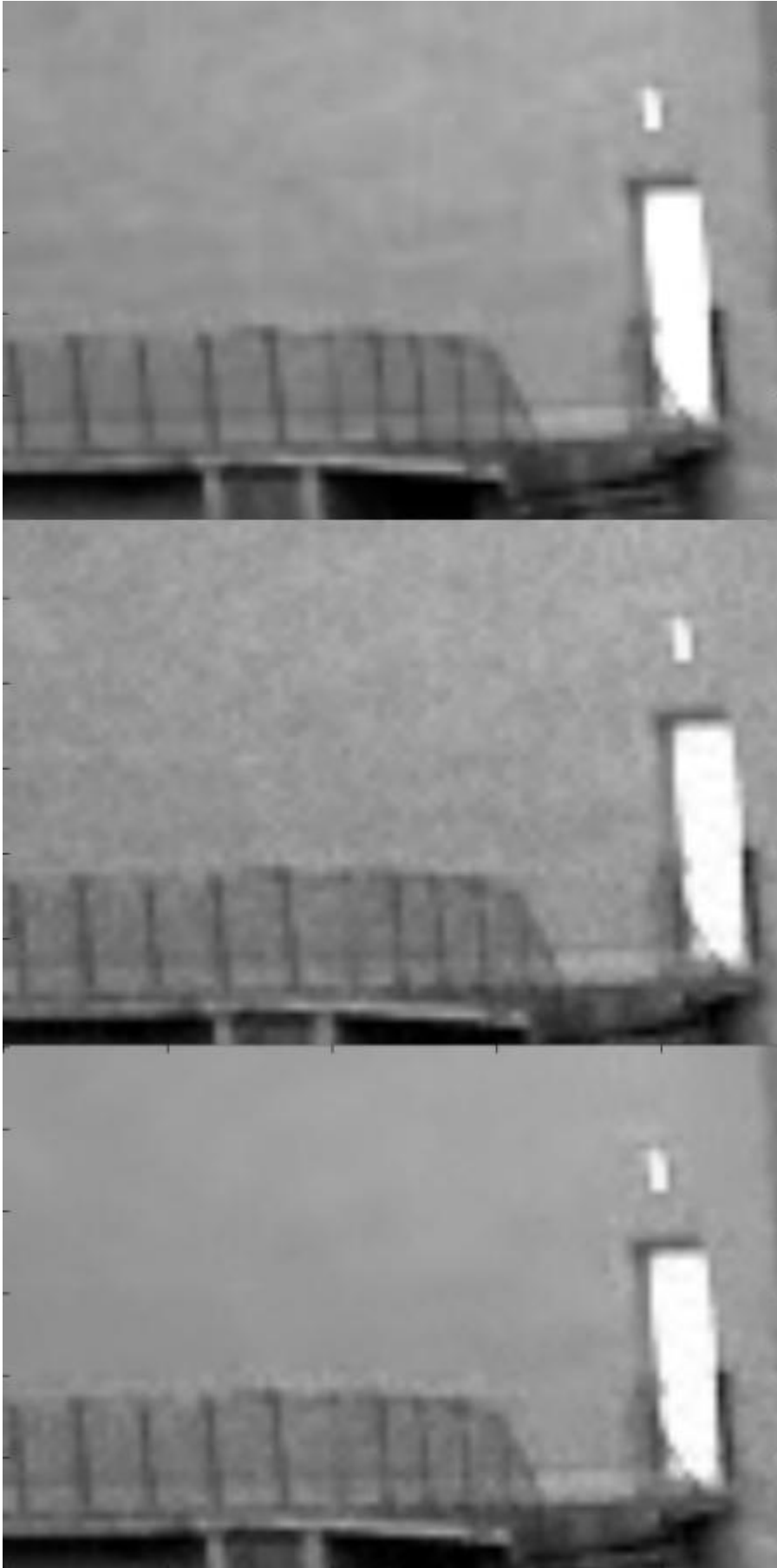
以下是结果的放大版本。我的输入图像的高斯噪声为 $\sigma=25$ 。查看结果：



1. **cv.fastNlMeansDenoisingMulti()** 现在，我们将对视频应用相同的方法。第一个参数是噪声帧列表。第二个参数 `imgToDenoiseIndex` 指定我们需要去噪的帧，为此，我们在输入列表中传递帧的索引。第三是 `temporalWindowSize`，它指定要用于降噪的附近帧的数量。应该很奇怪。在那种情况下，总共使用 `temporalWindowSize` 帧，其中中心帧是要被去噪的帧。例如，你传递了一个5帧的列表作为输入。令 `imgToDenoiseIndex = 2, temporalWindowSize = 3`。然后使用 `frame-1`，`frame-2` 和 `frame-3` 去噪 `frame-2`。让我们来看一个例子。

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
cap = cv.VideoCapture('vtest.avi')
# 创建5个帧的列表
```

```
img = [cap.read()[1] for i in xrange(5)]
# 将所有转化为灰度
gray = [cv.cvtColor(i, cv.COLOR_BGR2GRAY) for i in img]
# 将所有转化为float64
gray = [np.float64(i) for i in gray]
# 创建方差为25的噪声
noise = np.random.randn(*gray[1].shape)*10
# 在图像上添加噪声
noisy = [i+noise for i in gray]
# 转化为unit8
noisy = [np.uint8(np.clip(i,0,255)) for i in noisy]
# 对第三帧进行降噪
dst = cv.fastNlMeansDenoisingMulti(noisy, 2, 5, None, 4, 7, 35)
plt.subplot(131),plt.imshow(gray[2], 'gray')
plt.subplot(132),plt.imshow(noisy[2], 'gray')
plt.subplot(133),plt.imshow(dst, 'gray')
plt.show()
```



计算需要花费大量时间。结果，第一个图像是原始帧，第二个是噪声帧，第三个是去噪图像。

附加资源

1. http://www.ipol.im/pub/art/2011/bcm_nlm/ (它包含详细信息，在线演示等。强烈建议访问。我们的测试图像是从此链接生成的)
2. Online course at coursera (这里拍摄的第一张图片)

练习

9_2_图像修补

图像修补

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本章中，- 我们将学习如何通过一种称为“修复”的方法消除旧照片中的小噪音，笔画等。- 我们将看到OpenCV中的修复功能。

基础

你们大多数人家里都会有一些旧的旧化照片，上面有黑点，一些笔触等。你是否曾经想过将其还原？我们不能简单地在绘画工具中擦除它们，因为它将简单地用白色结构代替黑色结构，这是没有用的。在这些情况下，将使用一种称为图像修复的技术。基本思想很简单：用附近的像素替换那些不良区域，使其看起来和邻近的协调。考虑下面显示的图像（摘自Wikipedia）：



基于此目的设计了几种算法，OpenCV提供了其中两种。两者都可以通过相同的函数进行访问，**cv.inpaint()**

第一种算法基于Alexandru Telea在2004年发表的论文“基于快速行进方法的图像修补技术”。它基于快速行进方法。考虑图像中要修复的区域。算法从该区域的边界开始，并进入该区域内部，首先逐渐填充边界中的所有内容。在要修复的邻域上的像素周围需要一个小的邻域。该像素被附近所有已知像素的归一化加权总和所代替。权重的选择很重要。那些位于该点附近，边界法线附近的像素和那些位于边界轮廓线上的像素将获得更大的权重。修复像素后，将使用快速行进方法将

其移动到下一个最近的像素。FMM确保首先修复已知像素附近的那些像素，以便像手动启发式操作一样工作。通过使用标志**cv.INPAINT_TELEA**启用此算法。

第二种算法基于Bertalmio，Marcelo，Andrea L. Bertozzi和Guillermo Sapiro在2001年发表的论文“Navier-Stokes，流体动力学以及图像和视频修补”。该算法基于流体动力学并利用了偏微分方程。基本原理是启发式的。它首先沿着边缘从已知区域移动到未知区域（因为边缘是连续的）。它延续了等距线（线连接具有相同强度的点，就像轮廓线连接具有相同高程的点一样），同时在修复区域的边界匹配梯度矢量。为此，使用了一些流体动力学方法。获得它们后，将填充颜色以减少该区域的最小差异。通过使用标志**cv.INPAINT_NS**启用此算法。

代码

我们需要创建一个与输入图像大小相同的掩码，其中非零像素对应于要修复的区域。其他一切都很简单。我的图像因一些黑色笔画而旧化（我手动添加了）。我使用“绘画”工具创建了相应的笔触。

```
import numpy as np
import cv2 as cv
img = cv.imread('messi_2.jpg')
mask = cv.imread('mask2.png', 0)
dst = cv.inpaint(img, mask, 3, cv.INPAINT_TELEA)
cv.imshow('dst', dst)
cv.waitKey(0)
cv.destroyAllWindows()
```

请参阅下面的结果。第一张图片显示了降级的输入。第二个图像是掩码。第三个图像是第一个算法的结果，最后一个图像是第二个算法的结果。



附加资源 1. Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro. “Navier-stokes, fluid dynamics, and image and video inpainting.” In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1, pp. I-355. IEEE, 2001. 2. Telea, Alexandru. “An image inpainting technique based on the fast marching method.” Journal of graphics tools 9.1 (2004): 23-34.

练习

1. OpenCV一个有关修复的交互式示例，`samples/python/inpaint.py`，请尝试一下。
2. 几个月前，我观看了有关Content-Aware Fill的视频，Content-Aware Fill是Adobe Photoshop中使用的一种先进的修复技术。在进一步的搜索中，我发现GIMP中已经存在相同的技术，但名称不同，为“Resynthesizer”（你需要安装单独的插件）。我相信你会喜欢这项技术的。

9_3_高动态范围

高动态范围

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

目标

在本章中，我们将 - 了解如何根据曝光顺序生成和显示HDR图像。 - 使用曝光融合来合并曝光序列。

理论

高动态范围成像 (HDRI或HDR) 是一种用于成像和摄影的技术，可以比标准数字成像或摄影技术重现更大的动态亮度范围。虽然人眼可以适应各种光照条件，但是大多数成像设备每通道使用8位，因此我们仅限于256级。当我们拍摄现实世界的照片时，明亮的区域可能会曝光过度，而黑暗的区域可能会曝光不足，因此我们无法一次拍摄所有细节。HDR成像适用于每个通道使用8位以上 (通常为32位浮点值) 的图像，从而允许更大的动态范围。

获取HDR图像的方法有多种，但是最常见的一种方法是使用以不同曝光值拍摄的场景照片。要综合这些曝光，了解相机的响应功能以及估算算法的功能非常有用。合并HDR图像后，必须将其转换回8位才能在常规显示器上查看。此过程称为音调映射。当场景或摄像机的对象在两次拍摄之间移动时，还会增加其他复杂性，因为应记录并调整具有不同曝光度的图像。

在本教程中，我们展示了两种算法 (Debevec , Robertson) 来根据曝光序列生成和显示HDR图像，并演示了另一种称为曝光融合 (Mertens) 的方法，该方法可以生成低动态范围图像，并且不需要曝光时间数据。此外，我们估计相机响应函数 (CRF) 对于许多计算机视觉算法都具有重要价值。HDR流水线的每个步骤都可以使用不同的算法和参数来实现，因此请查看参考手册以了解所有内容。

曝光序列HDR

在本教程中，我们将查看以下场景，其中有4张曝光图像，曝光时间分别为15、2.5、1 / 4和1/30秒。(你可以从Wikipedia下载图像)



1. 将曝光图像加载到列表中

```
import cv2 as cv
import numpy as np
# 将曝光图像加载到列表中
img_fn = ["img0.jpg", "img1.jpg", "img2.jpg", "img3.jpg"]
img_list = [cv.imread(fn) for fn in img_fn]
exposure_times = np.array([15.0, 2.5, 0.25, 0.0333], dtype=np.float32)
```

2. 将曝光合成HDR图像 在此阶段，我们将曝光序列合并为一张HDR图像，显示了OpenCV中的两种可能性。第一种方法是Debevec，第二种方法是Robertson。请注意，HDR图像的类型为float32，而不是uint8，因为它包含所有曝光图像的完整动态范围。

```
# 将曝光合成HDR图像
merge_debevec = cv.createMergeDebevec()
hdr_debevec = merge_debevec.process(img_list, times=exposure_times.copy())
merge_robertson = cv.createMergeRobertson()
hdr_robertson = merge_robertson.process(img_list, times=exposure_times.copy())
```

3. 色调图HDR图像 我们将32位浮点HDR数据映射到[0..1]范围内。实际上，在某些情况下，该值可以大于1或小于0，因此请注意，我们稍后将必须裁剪数据以避免溢出。

```
# 色调图HDR图像
tonemap1 = cv.createTonemap(gamma=2.2)
res_debevec = tonemap1.process(hdr_debevec.copy())
```

4. 使用Mertens融合曝光 在这里，我们展示了一种替代算法，用于合并曝光图像，而我们不需要曝光时间。我们也不需要任何色调映射算法，因为Mertens算法已经为我们提供了[0..1]范围内的结果。

```
# 使用Mertens融合曝光
merge_mertens = cv.createMergeMertens()
res_mertens = merge_mertens.process(img_list)
```

5. 转为8-bit并保存 为了保存或显示结果，我们需要将数据转换为[0..255]范围内的8位整数。

```
# 转化数据类型为8-bit并保存
res_debevec_8bit = np.clip(res_debevec*255, 0, 255).astype('uint8')
res_robertson_8bit = np.clip(res_robertson*255, 0, 255).astype('uint8')
res_mertens_8bit = np.clip(res_mertens*255, 0, 255).astype('uint8')
cv.imwrite("ldr_debevec.jpg", res_debevec_8bit)
cv.imwrite("ldr_robertson.jpg", res_robertson_8bit)
cv.imwrite("fusion_mertens.jpg", res_mertens_8bit)
```

结果

你可以看到不同的结果，但是请考虑到每种算法都有其他额外的参数，你应该将它们附加以达到期望的结果。最佳实践是尝试不同的方法，然后看看哪种方法最适合你的场景。

Debevec:



Robertson:



Mertenes融合

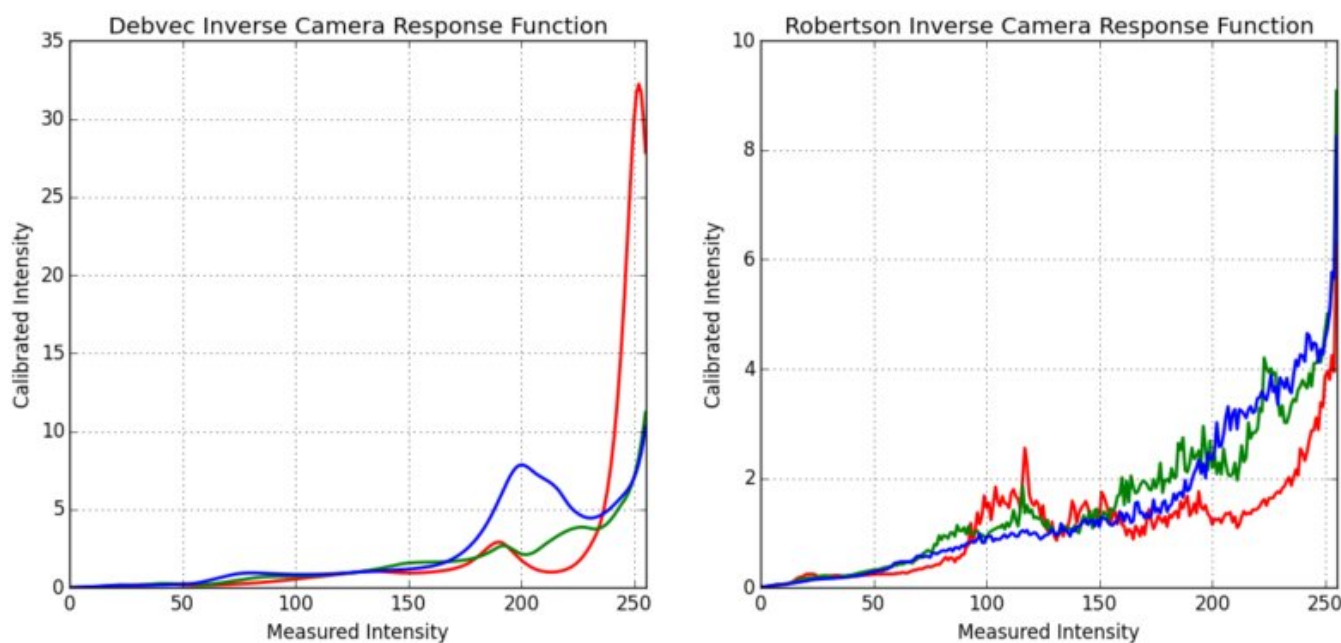


估计相机响应函数

摄像机响应功能 (CRF) 使我们可以将场景辐射度与测量强度值联系起来。CRF在某些计算机视觉算法 (包括HDR算法) 中非常重要。在这里，我们估计逆相机响应函数并将其用于HDR合并。

```
# 估计相机响应函数 (CRF)
cal_debevec = cv.createCalibrateDebevec()
crf_debevec = cal_debevec.process(img_list, times=exposure_times)
hdr_debevec = merge_debevec.process(img_list, times=exposure_times.copy(),
response=crf_debevec.copy())
cal_robertson = cv.createCalibrateRobertson()
crf_robertson = cal_robertson.process(img_list, times=exposure_times)
hdr_robertson = merge_robertson.process(img_list, times=exposure_times.copy(),
response=crf_robertson.copy())
```

相机响应功能由每个颜色通道的256长度向量表示。对于此序列，我们得到以下估计：



附加资源

1. Paul E Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In ACM SIGGRAPH 2008 classes, page 31. ACM, 2008. [48]
2. Mark A Robertson, Sean Borman, and Robert L Stevenson. Dynamic range improvement through multiple exposures. In Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on, volume 3, pages 159–163. IEEE, 1999. [182]
3. Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion. In Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on, pages 382–390. IEEE, 2007. [148]
4. Images from Wikipedia-HDR

练习

1. 尝试所有色调图算法：cv::TonemapDrago, cv::TonemapMantiuk和cv::TonemapReinhard
2. 尝试更改HDR校准和色调图方法中的参数。

级联分类器

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

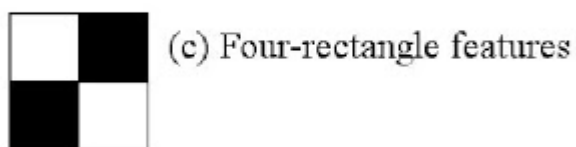
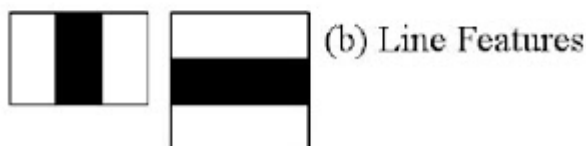
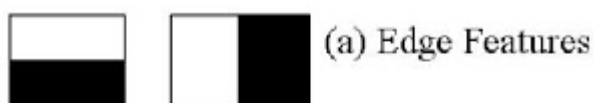
目标

在本教程中， - 我们将学习Haar级联对象检测的工作原理。 - 我们将使用基于Haar Feature的Cascade分类器了解人脸检测和眼睛检测的基础知识。 - 我们将使用**cv::CascadeClassifier**类来检测视频流中的对象。特别是，我们将使用以下函数： - **cv::CascadeClassifier::load**来加载.xml分类器文件。它可以是Haar或LBP分类器 - **cv::CascadeClassifier::detectMultiScale**来执行检测。

理论

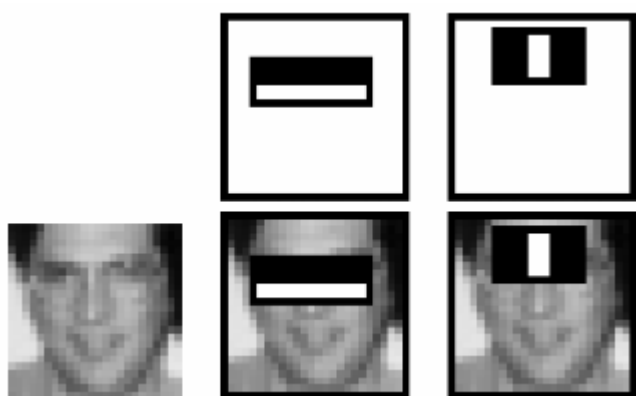
使用基于Haar特征的级联分类器的对象检测是Paul Viola和Michael Jones在其论文“使用简单特征的增强级联进行快速对象检测”中于2001年提出的一种有效的对象检测方法。这是一种基于机器学习的方法，其中从许多正负图像中训练级联函数。然后用于检测其他图像中的对象。

在这里，我们将进行人脸检测。最初，该算法需要大量正图像（面部图像）和负图像（无面部图像）来训练分类器。然后，我们需要从中提取特征。为此，使用下图所示的Haar功能。它们就像我们的卷积核一样。每个特征都是通过从黑色矩形下的像素总和中减去白色矩形下的像素总和而获得的单个值。



现在，每个内核的所有可能大小和位置都用于计算许多功能。（试想一下它产生多少计算？即使是一个24x24的窗口也会产生超过160000个特征）。对于每个特征计算，我们需要找到白色和黑色矩形下的像素总和。为了解决这个问题，他们引入了整体图像。无论你的图像有多大，它都会将给定像素的计算减少到仅涉及四个像素的操作。很好，不是吗？它使事情变得更快。

但是在我们计算的所有这些特征中，大多数都不相关。例如，考虑下图。第一行显示了两个良好的特征。选择的第一个特征似乎着眼于眼睛区域通常比鼻子和脸颊区域更暗的性质。选择的第二个特征依赖于眼睛比鼻梁更黑的属性。但是，将相同的窗口应用于脸颊或其他任何地方都是无关紧要的。那么，我们如何从16万多个功能中选择最佳特征？它是由**Adaboost**实现的。



为此，我们将所有特征应用于所有训练图像。对于每个特征，它会找到最佳的阈值，该阈值会将人脸分为正面和负面。显然，会出现错误或分类错误。我们选择错误率最低的特征，这意味着它们是对人脸和非人脸图像进行最准确分类的特征。（此过程并非如此简单。在开始时，每个图像的权重均相等。在每次分类后，错误分类的图像的权重都会增加。然后执行相同的过程。将计算新的错误率。还要计算新的权重。继续进行此过程，直到达到所需的精度或错误率或找到所需的功能数量为止。

最终分类器是这些弱分类器的加权和。之所以称为弱分类，是因为仅凭它不能对图像进行分类，而是与其他分类一起形成强分类器。该论文说，甚至200个功能都可以提供95%的准确度检测。他们的最终设置具有大约6000个功能。（想象一下，从160000多个功能减少到6000个功能。这是很大的收获）。

因此，现在你拍摄一张照片。取每个24x24窗口。向其应用6000个功能。检查是否有脸。哇..这不是效率低下又费时吗？是的。作者对此有一个很好的解决方案。

在图像中，大多数图像是非面部区域。因此，最好有一种简单的方法来检查窗口是否不是面部区域。如果不是，请一次性丢弃它，不要再次对其进行处理。相反，应将重点放在可能有脸的区域。这样，我们将花费更多时间检查可能的面部区域。

为此，他们引入了级联分类器的概念。不是将所有6000个功能部件应用到一个窗口中，而是将这些功能部件分组到不同的分类器阶段，并一一应用。（通常前几个阶段将包含很少的功能）。如果窗口在第一阶段失败，则将其丢弃。我们不考虑它的其余功能。如果通过，则应用功能的第二阶段并继续该过程。经过所有阶段的窗口是一个面部区域。这个计划怎么样！

作者的检测器具有6000多个特征，具有38个阶段，在前五个阶段具有1、10、25、25和50个特征。（上图中的两个功能实际上是从Adaboost获得的最佳两个功能）。根据作者的说法，每个子窗口平均评估了6000多个特征中的10个特征。

因此，这是Viola-Jones人脸检测工作原理的简单直观说明。阅读本文以获取更多详细信息，或查看其他资源部分中的参考资料。

OpenCV中的Haar-级联检测器

OpenCV提供了一种训练方法（请参阅**Cascade分类器训练**）或预先训练的模型，可以使用**cv::CascadeClassifier::load**方法读取。预训练的模型位于OpenCV安装的数据文件夹中，或在此处找到。

以下代码示例将使用预训练的Haar级联模型来检测图像中的面部和眼睛。首先，创建一个cv::CascadeClassifier并使用**cv::CascadeClassifier::load**方法加载必要的XML文件。然后，使用**cv::CascadeClassifier::detectMultiScale**方法完成检测，该方法返回检测到的脸部或眼睛的边界矩形。

本教程的代码如下所示。你也可以从这里下载

```
from __future__ import print_function
import cv2 as cv
import argparse
def detectAndDisplay(frame):
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    frame_gray = cv.equalizeHist(frame_gray)
    #-- 检测面部
    faces = face_cascade.detectMultiScale(frame_gray)
    for (x,y,w,h) in faces:
        center = (x + w//2, y + h//2)
        frame = cv.ellipse(frame, center, (w//2, h//2), 0, 0, 360, (255, 0, 255),
4)
        faceROI = frame_gray[y:y+h,x:x+w]
        #-- 在每张面部上检测眼睛
        eyes = eyes_cascade.detectMultiScale(faceROI)
        for (x2,y2,w2,h2) in eyes:
```

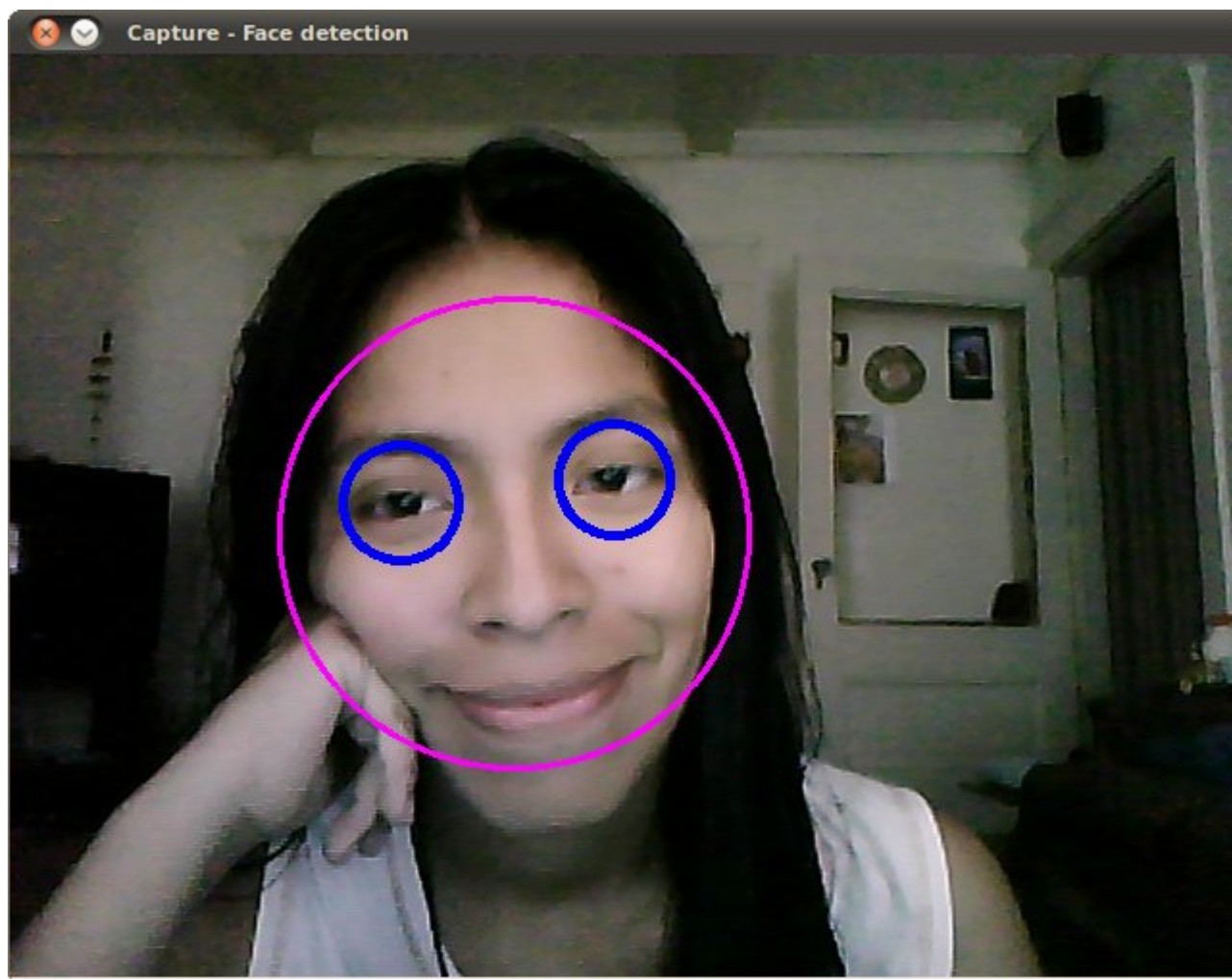


```
        eye_center = (x + x2 + w2//2, y + y2 + h2//2)
        radius = int(round((w2 + h2)*0.25))
        frame = cv.circle(frame, eye_center, radius, (255, 0, 0 ), 4)
    cv.imshow('Capture - Face detection', frame)
    parser = argparse.ArgumentParser(description='Code for Cascade Classifier tutorial.')
    parser.add_argument('--face_cascade', help='Path to face cascade.', default='data/haarcascades/haarcascade_frontalface_alt.xml')
    parser.add_argument('--eyes_cascade', help='Path to eyes cascade.', default='data/haarcascades/haarcascade_eye_tree_eyeglasses.xml')
    parser.add_argument('--camera', help='Camera divide number.', type=int, default=0)
    args = parser.parse_args()
    face_cascade_name = args.face_cascade
    eyes_cascade_name = args.eyes_cascade
    face_cascade = cv.CascadeClassifier()
    eyes_cascade = cv.CascadeClassifier()

    #-- 1. 加载级联
    if not face_cascade.load(cv.samples.findFile(face_cascade_name)):
        print('--(!)Error loading face cascade')
        exit(0)
    if not eyes_cascade.load(cv.samples.findFile(eyes_cascade_name)):
        print('--(!)Error loading eyes cascade')
        exit(0)
    camera_device = args.camera

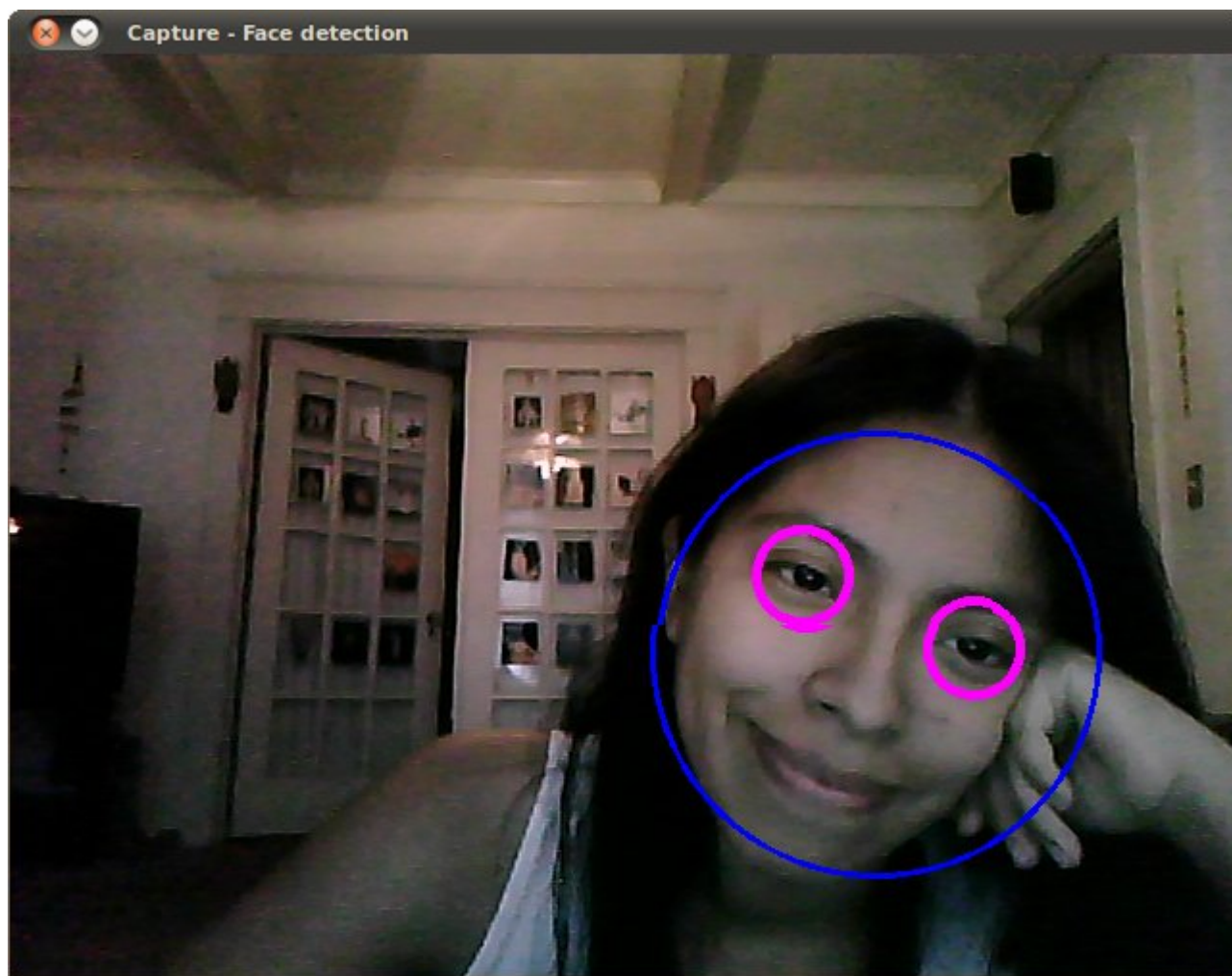
    #-- 2. 读取视频流
    cap = cv.VideoCapture(camera_device)
    if not cap.isOpened():
        print('--(!)Error opening video capture')
        exit(0)
    while True:
        ret, frame = cap.read()
        if frame is None:
            print('--(!) No captured frame -- Break!')
            break
        detectAndDisplay(frame)
        if cv.waitKey(10) == 27:
            break
```

结果 1. 这是运行上面的代码并将内置摄像头的视频流用作输入的结果：



请确保程序会找到文件*haarcascade_frontalface_alt.xml*和*haarcascade_eye_tree_eyeglasses.xml*的路径。它们位于*opencv/data/ haarcascades*中

1. 这是使用文件*lbpcascade_frontalface.xml* (经过LBP训练) 进行人脸检测的结果。对于眼睛，我们继续使用本教程中使用的文件。



附加资源

1. Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2):137–154, 2004. [228]
2. Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on, volume 1, pages I–900. IEEE, 2002. [129]
3. Video Lecture on Face Detection and Tracking
4. An interesting interview regarding Face Detection by Adam Harvey
5. OpenCV Face Detection: Visualized on Vimeo by Adam Harvey

级联分类器

作者|OpenCV-Python Tutorials 编译|Vincent 来源|OpenCV-Python Tutorials

简介

使用弱分类器的增强级联包括两个主要阶段：训练阶段和检测阶段。****对象检测教程****中介绍了使用基于HAAR或LBP模型的检测阶段。本文档概述了训练自己的弱分类器的级联所需的功能。当前指南将逐步完成所有不同阶段：收集训练数据，准备训练数据并执行实际模型训练。

为了支持本教程，将使用几个官方的OpenCV应用程序：opencv_createsamples，opencv_annotation，opencv_traincascade和opencv_visualisation。

重要的事项

- 如果您遇到任何提及旧的opencv_haartraining工具（*不推荐使用，仍在使用OpenCV1.x界面*）的教程，请忽略该教程并坚持使用opencv_traincascade工具。此工具是较新的版本，根据OpenCV 2.x和OpenCV 3.x API用C++编写。opencv_traincascade支持类似HAAR的小波特征[227]和LBP（局部二进制模式）[127]特征。与HAAR特征相比，LBP特征产生整数精度，产生浮点精度，因此LBP的训练和检测速度都比HAAR特征快几倍。关于LBP和HAAR的检测质量，主要取决于所使用的训练数据和选择的训练参数。可以训练基于LBP的分类器，该分类器将在训练时间的一定百分比内提供与基于HAAR的分类器几乎相同的质量。
- 来自OpenCV 2.x和OpenCV 3.x（**cv::CascadeClassifier**）的较新的层叠分类器检测接口支持使用新旧模型格式。如果由于某种原因而使用旧界面，则opencv_traincascade甚至可以旧格式保存（导出）经过训练的级联。然后至少可以在最稳定的界面中训练模型。
- ****opencv_traincascade****应用程序可以使用TBB进行多线程处理。要在多核模式下使用它，必须在启用TBB支持的情况下构建OpenCV。

准备训练数据

为了训练弱分类器的增强级联，我们需要一组正样本（包含您要检测的实际对象）和一组负样本（包含您不想检测的所有内容）。负样本集必须手动准备，而阳性样本集是使用opencv_createsamples应用程序创建的。

负样本 负样本取自任意图像，不包含要检测的对象。这些生成样本的负片图像应列在一个特殊的负片图像文件中，该文件每行包含一个图像路径（可以是绝对路径，也可以是相对路径）。注意，负样本和样本图像也称为背景样本或背景图像，在本文档中可以互换使用。

所描述的图像可能具有不同的尺寸。但是，每个图像都应等于或大于所需的训练窗口大小（与模型尺寸相对应，多数情况下是对象的平均大小），因为这些图像用于将给定的负像子采样为几个图像具有此训练窗口大小的样本。

负样本描述文件的示例：目录结构：

```
/img  
  img1.jpg  
  img2.jpg  
bg.txt
```

文件 bg.txt

```
img/img1.jpg  
img/img2.jpg
```

您的一组负窗口样本将用于模型训练，在这种情况下，当尝试查找您感兴趣的对象时，可以增强不需要查找的内容。

正样本

正样本由opencv_createsamples应用程序创建。提升过程使用它们来定义在尝试找到感兴趣的对象时模型应实际寻找的内容。该应用程序支持两种生成正样本数据集的方式。1. 您可以从单个正对象图像生成一堆正值。2. 您可以自己提供所有肯定的内容，仅使用该工具将其切出，调整大小并以opencv所需的二进制格式放置。尽管第一种方法对固定对象（例如非常刚性的徽标）效果不错，但对于刚性较差的对象，它往往很快就会失效。在这种情况下，我们建议使用第二种方法。网络上的许多教程甚至都指出，通过使用opencv_createsamples应用程序，可以生成100个真实对象图像，而不是1000个人工生成的正值。但是，如果您决定采用第一种方法，请记住以下几点：

- 请注意，在将其提供给上述应用程序之前，您需要多个正样本，因为它仅适用于透视变换。
- 如果您想要一个健壮的模型，请获取涵盖对象类中可能出现的各种变化的样本。例如，对于面孔，您应考虑不同的种族和年龄段，情绪以及胡须风格。当使用第二种方法时，这也适用。

第一种方法采用带有公司徽标的单个对象图像，并通过随机旋转对象，更改图像强度以及将图像放置在任意背景上，从给定的对象图像中创建大量正样本。随机性的数量和范围可以通过 `opencv_createsamples` 应用程序的命令行参数来控制。

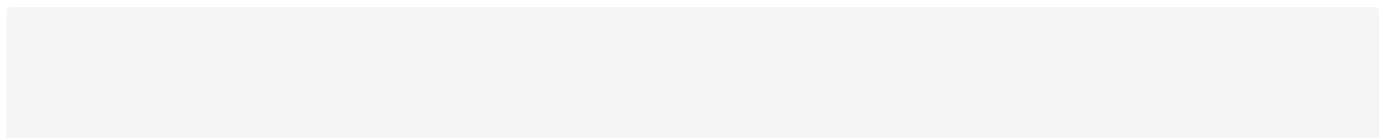
命令行参数： - `-vec <vec_file_name>`：包含用于训练的正样本的输出文件的名称。 - `-img <image_file_name>`：源对象图像（例如，公司徽标）。 - `-bg <background_file_name>`：背景描述文件；包含图像列表，这些图像用作对象的随机变形版本的背景。 - `-num <number_of_samples>`：要生成的正样本数。 - `-bgcolor <background_color>`：背景色（当前假定为灰度图像）；背景色表示透明色。由于可能存在压缩伪影，因此可以通过 `-bgthresh` 指定颜色容忍度。`bgcolor-bgthresh` 和 `bgcolor + bgthresh` 范围内的所有像素均被解释为透明的。 - `-bgthresh <background_color_threshold>` - `-inv`：如果指定，颜色将被反转。 - `-randinv`：如果指定，颜色将随机反转。 - `-maxidev <max_intensity_deviation>`：前景样本中像素的最大强度偏差。 - `-maxxangle <max_x_rotation_angle>`：朝向x轴的最大旋转角度，必须以弧度为单位。 - `-maxyangle <max_y_rotation_angle>`：朝向y轴的最大旋转角度，必须以弧度为单位。 - `-maxzangle <max_z_rotation_angle>`：朝向z轴的最大旋转角度，必须以弧度为单位。 - `-show`：有用的调试选项。如果指定，将显示每个样本。按Esc将继续示例创建过程，而不会显示每个示例。 - `-w <sample_width>`：输出样本的宽度（以像素为单位）。 - `-h <sample_height>`：输出样本的高度（以像素为单位）。

当以此方式运行 `opencv_createsamples` 时，将使用以下过程创建示例对象实例：给定的源图像围绕所有三个轴随机旋转。所选角度受 `-maxxangle`，`-maxyangle` 和 `-maxzangle` 限制。然后，像素在 `[bg_color-bg_color_threshold; bg_color + bg_color_threshold]` 范围内被设置为透明。白噪声被添加到前景的强度。如果指定了 `-inv` 键，则前景像素强度将反转。如果指定了 `-randinv` 键，则算法将随机选择是否应对该样本应用反演。最后，将获得的图像从背景描述文件放置到任意背景上，将其大小调整为由 `-w` 和 `-h` 指定的所需大小，并存储到由 `-vec` 命令行选项指定的 `vec` 文件中。

也可以从以前标记的图像的集合中获取正样本，这是构建鲁棒对象模型时的理想方式。该集合由类似于背景描述文件的文本文件描述。该文件的每一行都对应一个图像。该行的第一个元素是文件名，后跟对象注释的数量，后跟描述包围矩形（x，y，宽度，高度）的对象坐标的数字。

一个描述文件的示例：

目录结构：



```
/img
  img1.jpg
  img2.jpg info.dat
```

文件 info.dat

```
img/img1.jpg 1 140 100 45 45
img/img2.jpg 2 100 200 50 50 50 30 25 25
```

图像img1.jpg包含具有以下边界矩形坐标的单个对象实例：(140,100,45 , 45)。图像img2.jpg包含两个对象实例。

为了从此类集合创建正样本，应指定 `-info` 参数而不是 `-img`：

- `-info <collection_file_name>`：标记图像集合的描述文件。

请注意，在这种情况下，`-bg`、`-bgcolor`、`-bgthreshold`、`-inv`、`-randinv`、`-maxxangle`、`-maxyangle` 和 `-maxzangle` 等参数将被简单地忽略并且不再使用。在这种情况下，样本创建的方案如下。通过从原始图像中切出提供的边界框，从给定图像中获取对象实例。然后将它们调整为目标样本大小（由 `-w` 和 `-h` 定义），并存储在由 `-vec` 参数定义的输出 `vec` 文件中。没有应用任何失真，因此仅有的影响参数是 `-w`，`-h`，`-show` 和 `-num`。

也可以使用 `opencv_annotation` 工具完成手动创建 `-info` 文件的过程。这是一个开放源代码工具，用于在任何给定图像中直观地选择对象实例的关注区域。以下小节将详细讨论如何使用此应用程序。

额外事项 - `opencv_createsamples` 实用程序可用于检查存储在任何给定正样本文件中的样本。为此，仅应指定 `-vec`，`-w` 和 `-h` 参数。- 此处提供了 `vec` 文件的示例 `opencv/data/vec_files/trainingfaces_24-24.vec`。它可以用于训练具有以下窗口大小的面部检测器：`-w 24 -h 24`。

使用OpenCV中的集成标注工具 从OpenCV 3.x开始，社区一直在提供和维护开源注释工具，该工具用于生成 `-info` 文件。如果构建了OpenCV应用程序，则可以通过命令 `opencv_annotation` 访问该工具。

使用该工具非常简单。该工具接受几个必需参数和一些可选参数：`- --annotations`（必需）：注释 `txt` 文件的路径，您要在其中存储注释，然后将其传递给 `-info` 参数[example-/data/annotations.txt] - `--images`（必需）：包含带有对象的图像的文件夹的路径[示例-/data/testimages/] - `--maxWindowHeight`（可选）：如果输入图像的高度大于此处的给定分辨率，则调

整图像的大小以用于使用 `--resizeFactor` 可以简化注释。 - `--resizeFactor` (可选) : 使用 `--maxWindowHeight` 参数时用于调整输入图像大小的因子。

请注意，可选参数只能一起使用。可以使用的命令示例如下所示

```
opencv_annotation --annotations=/path/to/annotations/file.txt --images=/path/to/image/folder/
```

此命令将启动一个窗口，其中包含第一张图像和您的鼠标光标，这些窗口将用于注释。有关如何使用注释工具的视频，请参见此处。基本上，有几个按键可以触发一个动作。鼠标左键用于选择对象的第一个角，然后继续绘制直到您满意为止，并在单击鼠标第二次单击时停止。每次选择后，您有以下选择： - 按 `c`：确认注释，将注释变为绿色并确认已存储。 - 按 `d`：从注释列表中删除最后一个注释（易于删除错误的注释） - 按 `n`：继续进行操作下一张图片 - 按 `ESC` 键：这将退出注释软件

最后，您将获得一个可用的注释文件，该文件可以传递给 `opencv_createsamples` 的 `-info` 参数。

级联训练

下一步是根据预先准备的正负数据集对弱分类器的增强级联进行实际训练。

按用途分组的 `opencv_traincascade` 应用程序的命令行参数：

- 常用参数
- `-data <cascade_dir_name>`：应将经过训练的分类器存储在何处。此文件夹应事先手动创建。
- `-vec <vec_file_name>`：带有正样本的vec文件（由 `opencv_createsamples` 实用程序创建）。
- `-bg <background_file_name>`：背景描述文件。这是包含阴性样本图像的文件。
- `-numPos <number_of_positive_samples>`：在每个分类器阶段的训练中使用的阳性样本数。
- `-numNeg <number_of_negative_samples>`：在每个分类器阶段的训练中使用的负样本数。
- `-numStages <number_of_stages>`：要训练的级联级数。
- `-precalcValBufSize <precalculated_vals_buffer_size_in_Mb>`：用于预先计算的特征值的缓冲区大小（以Mb为单位）。分配的内存越多，训练过程就越快，但是请记住，`-precalcValBufSize` 和 `-precalcIdxBufSize` 的总和不应超过可用的系统内存。

- `-precalcIdxBufSize <precalculated_idx_buffer_size_in_Mb>` : 用于预先计算的特征索引的缓冲区大小 (以Mb为单位)。分配的内存越多, 训练过程就越快, 但是请记住, `-precalcValBufSize` 和 `-precalcIdxBufSize` 的总和不应超过可用的系统内存。
- `-baseFormatSave` : 对于类似Haar的功能, 此参数是实际的。如果指定, 级联将以旧格式保存。仅出于向后兼容的原因, 并且允许用户停留在旧的不赞成使用的界面上, 至少可以使用较新的界面训练模型, 才可以使用此功能。
- `-numThreads <最大线程数>` : 训练期间要使用的最大线程数。请注意, 实际使用的线程数可能会更少, 具体取决于您的计算机和编译选项。默认情况下, 如果您使用TBB支持构建了OpenCV, 则会选择最大可用线程, 这是此优化所必需的。
- `-acceptanceRatioBreakValue <break_value>` : 此参数用于确定模型应保持学习的精确度以及何时停止。良好的指导原则是进行不超过 $10e-5$ 的训练, 以确保模型不会对您的训练数据过度训练。默认情况下, 此值设置为-1以禁用此功能。
- **级联参数 :**
 - `-stageType <BOOST(default)>` : 阶段的类型。目前仅支持提升分类器作为阶段类型。
 - `-featureType <{{HAAR(default),LBP}}>` : 功能类型: HAAR-类似Haar的功能, LBP-本地二进制模式。
 - `-w <sampleWidth>` : 训练样本的宽度 (以像素为单位)。必须具有与训练样本创建期间使用的值完全相同的值 (`opencv_createsamples`实用程序)。
 - `-h <sampleHeight>` : 训练样本的高度 (以像素为单位)。必须具有与训练样本创建期间使用的值完全相同的值 (`opencv_createsamples`实用程序)。
- **提升分类器参数 :**
 - `-bt <{DAB, RAB, LB, GAB(default)}>` : 增强分类器的类型: DAB-离散AdaBoost, RAB-真实AdaBoost, LB-LogitBoost, GAB-温和AdaBoost。
 - `-minHitRate <min_hit_rate>` : 分类器每个阶段的最低期望命中率。总命中率可以估计为 $(\text{min_hit_rate}^{\text{number_of_stages}})$, [228]§4.1。
 - `-maxFalseAlarmRate <max_false_alarm_rate>` : 分类器每个阶段的最大期望误报率。总体误报率可以估计为 $(\text{max_false_alarm_rate}^{\text{number_of_stages}})$, [228]§4.1。
 - `-weightTrimRate` : 指定是否应使用修剪及其权重。不错的选择是0.95。
 - `-maxDepth <max_weak_tree>` : 弱树的最大深度。一个不错的选择是1, 这是树桩的情况。

- `-maxWeakCount <max_weak_tree_count>` : 每个级联阶段的弱树的最大计数。提升分类器 (阶段) 将具有如此多的弱树 ($\leq \text{maxWeakCount}$) , 以实现给定的`-maxFalseAlarmRate`。
- 类似Haar的特征参数 :
 - `-mode <BASIC(default)|CORE| ALL>` : 选择训练中使用的Haar功能集的类型。BASIC仅使用直立功能, 而ALL使用整套直立和45度旋转功能集。有关更多详细信息, 请参见 [129]。
- 本地二进制模式参数: 本地二进制模式没有参数。

opencv_traincascade应用程序完成工作后, 经过训练的级联将保存在 `-data` 文件夹中的 `cascade.xml` 文件中。此文件夹中的其他文件是为中断培训而创建的, 因此您可以在训练完成后将其删除。

训练完成后, 您可以测试级联分类器!

可视化级联分类器

有时, 可视化经过训练的级联可能很有用, 以查看其选择的功能以及其阶段的复杂程度。为此, OpenCV提供了一个opencv_visualisation应用程序。该应用程序具有以下命令:

- `--image (必需)`: 对象模型参考图像的路径。这应该是标注, 标注 `[-w, -h]` 传递给opencv_createsamples和opencv_traincascade应用程序。
- `--model (必需)`: 训练模型的路径, 该路径应该在opencv_traincascade应用程序的`-data`参数提供的文件夹中。
- `--data (可选)`: 如果提供了一个数据文件夹, 必须事先手动创建它, 那么将存储舞台输出和功能的视频。下面是一个示例命令:

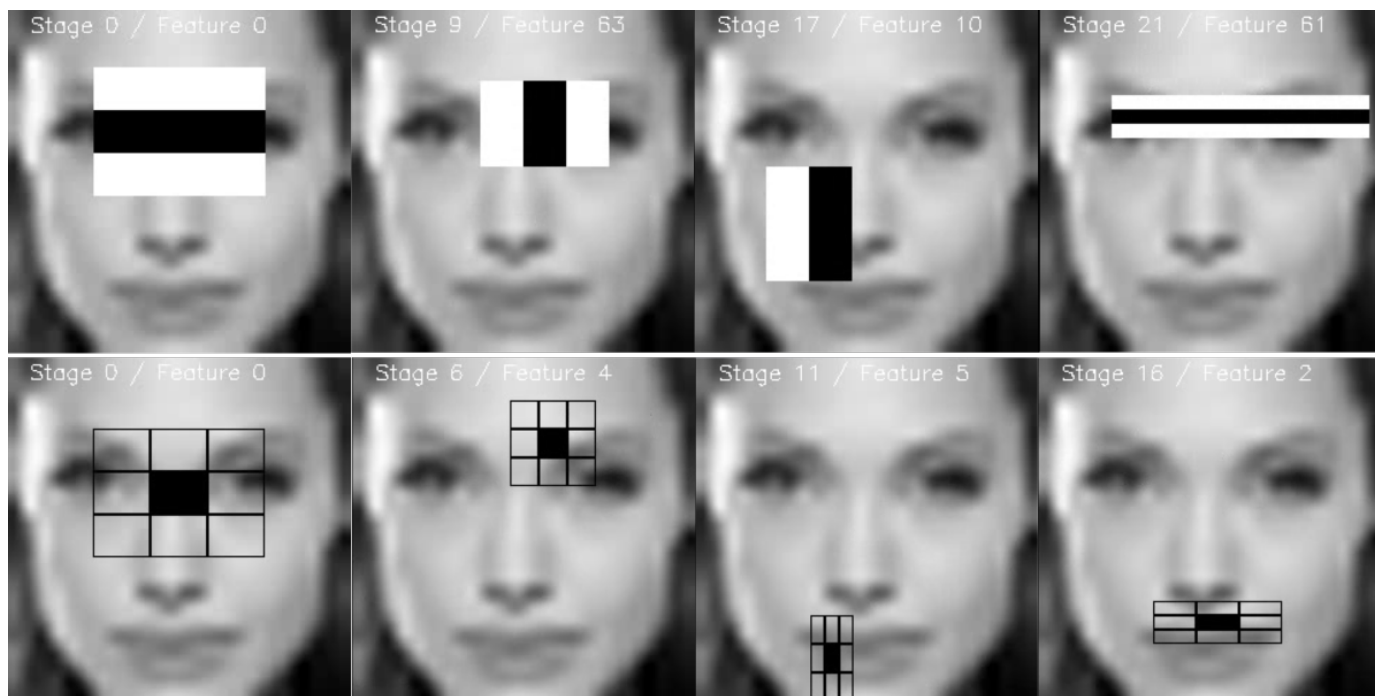
```
opencv_visualisation --image=/ data/object.png --model=/data/model.xml --data =/  
data/result/
```

当前可视化工具的某些限制 - 仅处理级联分类器模型, 使用opencv_traincascade工具进行培训, 其中包含树桩作为决策树[默认设置]。

- 提供的图像必须是带有原始模型尺寸的样本窗口, 并传递给`-image`参数。

HAAR / LBP人脸模型的示例在安吉丽娜·朱莉 (Angelina Jolie) 的给定窗口上运行, 该窗口具有与级联分类器文件相同的预处理 -> 24x24像素图像, 灰度转换和直方图均衡: 每个阶段都会制作一个视频, 每个特征都可可视化:

每个阶段都会制作一个视频，以显示每个功能：



每个阶段都作为图像存储，以供将来对功能进行验证：



这项工作是由StevenPuttemans为OpenCV 3 Blueprints创建的，但是Packt Publishing同意将其集成到OpenCV中。

OpenCV-Python Bindings 如何工作？

作者|OpenCV-Python Tutorials

编译|Vincent

来源|OpenCV-Python Tutorials

目标

了解： - 如何生成OpenCV-Python bindings？ - 如何将新的OpenCV模块扩展到Python？

OpenCV-Python bindings如何生成？

在OpenCV中，所有算法均以C++实现。但是这些算法可以从不同的语言（例如Python，Java等）中使用。绑定生成器使这成为可能。这些生成器在C++和Python之间建立了桥梁，使用户能够从Python调用C++函数。为了全面了解后台发生的事情，需要对Python / C API有充分的了解。在官方Python文档中可以找到一个有关将C++函数扩展到Python的简单示例[1]。因此，通过手动编写包装函数将OpenCV中的所有函数扩展到Python是一项耗时的任务。因此，OpenCV以更智能的方式进行操作。OpenCV使用位于 `modules/python/src2` 中的一些Python脚本，从C++头自动生成这些包装器函数。我们将调查他们的工作。

首先，`modules/python / CMakeFiles.txt` 是一个CMake脚本，用于检查要扩展到Python的模块。它将自动检查所有要扩展的模块并获取其头文件。这些头文件包含该特定模块的所有类，函数，常量等的列表。

其次，将这些头文件传递到Python脚本 `modules/python/src2/gen2.py`。这是Python Binding生成器脚本。它调用另一个Python脚本 `module/python/src2/hdr_parser.py`。这是标头解析器脚本。此标头解析器将完整的标头文件拆分为较小的Python列表。因此，这些列表包含有关特定函数，类等的所有详细信息。例如，将对一个函数进行解析以获取一个包含函数名称，返回类型，输入参数，参数类型等的列表。最终列表包含所有函数，枚举的详细信息，头文件中的structs，classs等。

但是标头解析器不会解析标头文件中的所有函数/类。开发人员必须指定应将哪些函数导出到Python。为此，在这些声明的开头添加了某些宏，这些宏使标头解析器可以标识要解析的函数。

这些宏由对特定功能进行编程的开发人员添加。简而言之，开发人员决定哪些功能应该扩展到Python，哪些不应该。这些宏的详细信息将在下一个会话中给出。

因此头解析器将返回已解析函数的最终大列表。我们的生成器脚本 (gen2.py) 将为标头解析器解析的所有函数/类/枚举/结构创建包装函数 (你可以在编译期间在 `build/modules/python/` 文件夹中以 `pyopencv_genic_*.h` 文件找到这些标头文件)。但是可能会有一些基本的OpenCV数据类型，例如 `Mat`，`Vec4i`，`Size`。它们需要手动扩展。例如，`Mat`类型应扩展为Numpy数组，`Size`应扩展为两个整数的元组，等等。类似地，可能会有一些复杂的结构/类/函数等需要手动扩展。所有这些手动包装函数都放在 `modules/python/src2/cv2.cpp` 中。

所以现在剩下的就是这些包装文件的编译了，这给了我们**cv2**模块。因此，当你调用函数时，例如在Python中说 `res = equalizeHist(img1,img2)`，你将传递两个numpy数组，并期望另一个numpy数组作为输出。因此，将这些numpy数组转换为 `cv::Mat`，然后在C++中调用 `**equalizeHist**()` 函数。最终结果将 `res` 转换回Numpy数组。简而言之，几乎所有操作都是在C++中完成的，这给了我们几乎与C++相同的速度。

因此，这是OpenCV-Python bindings生成方式的基本形式。

如何扩展新的模块到Python？

头解析器根据添加到函数声明中的一些包装宏来解析头文件。枚举常量不需要任何包装宏。它们会自动包装。但是其余的函数，类等需要包装宏。

使用 `CV_EXPORTS_W` 宏扩展功能。一个例子如下所示。

```
CV_EXPORTS_W void equalizeHist( InputArray src, OutputArray dst );
```

标头解析器可以理解诸如 `InputArray`，`OutputArray` 等关键字的输入和输出参数。但是有时，我们可能需要对输入和输出进行硬编码。为此，使用了 `CV_OUT`，`CV_IN_OUT` 等宏。

```
CV_EXPORTS_W void minEnclosingCircle( InputArray points,
                                       CV_OUT Point2f& center, CV_OUT float&
                                       radius );
```

对于大类，也使用 `CV_EXPORTS_W`。为了扩展类方法，使用 `CV_WRAP`。同样，`CV_PROP` 用于类字段。

```
class CV_EXPORTS_W CLAHE : public Algorithm
{
public:
    CV_WRAP virtual void apply(InputArray src, OutputArray dst) = 0;
    CV_WRAP virtual void setClipLimit(double clipLimit) = 0;
    CV_WRAP virtual double getClipLimit() const = 0;
}
```

可以使用 `CV_EXPORTS_AS` 扩展重载的函数。但是我们需要传递一个新名称，以便在Python中使用该名称调用每个函数。以下面的积分函数为例。提供了三个函数，因此每个函数在Python中都带有一个后缀。类似地，`CV_WRAP_AS` 可用于包装重载方法。

```
CV_EXPORTS_W void integral( InputArray src, OutputArray sum, int sdepth = -1 );
CV_EXPORTS_AS(integral2) void integral( InputArray src, OutputArray sum,
                                         OutputArray sqsum, int sdepth = -1, int
sqdepth = -1 );
CV_EXPORTS_AS(integral3) void integral( InputArray src, OutputArray sum,
                                         OutputArray sqsum, OutputArray tilted,
                                         int sdepth = -1, int sqdepth = -1 );
```

小类/结构使用 `CV_EXPORTS_W_SIMPLE` 进行扩展。这些结构按值传递给C++函数。示例包括 `KeyPoint`, `Match` 等。它们的方法由 `CV_WRAP` 扩展，而字段由 `CV_PROP_RW` 扩展。

```
class CV_EXPORTS_W_SIMPLE DMatch
{
public:
    CV_WRAP DMatch();
    CV_WRAP DMatch(int _queryIdx, int _trainIdx, float _distance);
    CV_WRAP DMatch(int _queryIdx, int _trainIdx, int _imgIdx, float _distance);
    CV_PROP_RW int queryIdx; // query descriptor index
    CV_PROP_RW int trainIdx; // train descriptor index
    CV_PROP_RW int imgIdx; // train image index
    CV_PROP_RW float distance;
};
```

可以使用 `CV_EXPORTS_W_MAP` 导出其他一些小类/结构，并将其导出到Python本机字典中。`Moments()` 就是一个例子。

```
class CV_EXPORTS_W_MAP Moments
{
public:
    CV_PROP_RW double m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;
    CV_PROP_RW double mu20, mu11, mu02, mu30, mu21, mu12, mu03;
```

```
CV_PROP_RW double nu20, nu11, nu02, nu30, nu21, nu12, nu03;  
};
```

因此，这些是OpenCV中可用的主要扩展宏。通常，开发人员必须将适当的宏放在适当的位置。其余的由生成器脚本完成。有时，在某些特殊情况下，生成器脚本无法创建包装。此类函数需要手动处理，为此，你需要编写自己的 `pyopencv_*.hpp` 扩展标头，并将其放入模块的 `misc / python` 子目录中。但是大多数时候，根据OpenCV编码指南编写的代码将由生成器脚本自动包装。

更高级的情况涉及为Python提供C++接口中不存在的其他功能，例如额外的方法，类型映射或提供默认参数。稍后，我们将以 `UMat` 数据类型为例。首先，要提供特定于Python的方法，`CV_WRAP_PHANTOM` 的用法与 `CV_WRAP` 相似，不同之处在于它以方法标头作为参数，并且你需要在自己的 `pyopencv_*.hpp` 扩展名中提供方法主体。`UMat::queue()` 和 `UMat::context()` 是此类幻象方法的示例，这些幻象方法在C++接口中不存在，但在Python端处理OpenCL功能时需要使用。其次，如果一个已经存在的数据类型可以映射到你的类，则最好使用 `CV_WRAP_MAPPABLE` 以源类型作为其参数来指示这种容量，而不是精心设计自己的绑定函数。从Mat映射的UMat就是这种情况。最后，如果需要默认参数，但本机C++接口中未提供，则可以在Python端将其作为 `CV_WRAP_DEFAULT` 的参数提供。按照下面的 `UMat::getMat` 示例：

```
class CV_EXPORTS_W UMat  
{  
public:  
    // 你需要提供 `static bool cv_mappable_to(const Ptr<Mat>& src, Ptr<UMat>& dst)`  
    CV_WRAP_MAPPABLE(Ptr<Mat>);  
    // returns the OpenCL queue used by OpenCV UMat.  
    // 你需要在资料夹代码中提供方法主体  
    CV_WRAP_PHANTOM(static void* queue());  
    // 你需要在资料夹代码中提供方法主体  
    CV_WRAP_PHANTOM(static void* context());  
    CV_WRAP_AS(get) Mat getMat(int flags CV_WRAP_DEFAULT(ACCESS_RW)) const;  
};
```

关于

扫描下方二维码，关注公众号【深度学习与计算机视觉】，获取更多计算机视觉教程，资源。

扫描下方二维码，关注公众号【深度学习与计算机视觉】，发送关键字“**pytorchpdf**”到公众号后台，可获得最新pytorch中文官方文档 PDF 资源。



深度学习与计算机视觉

微信扫描二维码，关注我的公众号

扫描下方二维码或搜索微信ID “mthler”，加我微信，备注“视觉”，拉你进交流群



贝加尔 
阿尔巴尼亚

加我微信，备注“视觉”，拉你进交流群



扫一扫上面的二维码图案，加我微信

OpenCV 中文官方文档

<http://woshicver.com/>

Github 地址

<https://github.com/fendouai/OpenCVTutorials>