Python OpenCV 计算机视觉（初译）

来源：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/SUMMARY.md

Python OpenCV 计算机视觉

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/README.md

> 原文：OpenCV Computer Vision with Python
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译
>
> 不要担心自己的形象，只关心如何实现目标。——《原则》，生活原则 2.3.c

- 在线阅读
- ApacheCN 面试求职交流群 724187166
- ApacheCN 学习资源

贡献指南

本项目需要校对，欢迎大家提交 Pull Request。

> 请您勇敢地去翻译和改进翻译。虽然我们追求卓越，但我们并不要求您做到十全十美，因此请不要担心因为翻译上犯错——在大部分情况下，我们的服务器已经记录所有的翻译，因此您不必担心会因为您的失误遭到无法挽回的破坏。（改编自维基百科）

联系方式

负责人

- 飞龙: 562826179

其他

- 在我们的 apachecn/apachecn-tf-zh github 上提 issue.
- 发邮件到 Email: `apachecn@163.com`.
- 在我们的 组织学习交流群 中联系群主/管理员即可.

赞助我们

微信	支付宝
	

零、前言

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/0.md

本书将向您展示如何使用 OpenCV 的 Python 绑定通过普通的网络摄像头或专用的深度传感器（例如 Microsoft Kinect）捕获视频，操纵图像并跟踪对象。 OpenCV 是一个开放源代码，跨平台的库，为计算机视觉实验和应用提供了构建基块。 它提供了用于捕获，处理和呈现图像数据的高级接口。 例如，它抽象了有关相机硬件和数组分配的详细信息。 OpenCV 在学术界和工业界都被广泛使用。

如今，计算机视觉可以通过网络摄像头，照相手机和诸如 Kinect 之类的游戏传感器在许多情况下吸引消费者。 不管是好是坏，人们都喜欢坐在相机上，作为开发人员，我们面临着捕获图像，改变其外观并从中提取信息的应用的需求。 OpenCV 的 Python 绑定可以帮助我们以高级语言和可与 NumPy 和 SciPy 等科学库互操作的标准化数据格式探索满足这些要求的解决方案。

尽管 OpenCV 是高级的并且可以互操作，但是对于新用户而言，并不一定很容易。 根据您的需求，OpenCV 的多功能性可能会以复杂的设置过程为代价，并且在如何将可用功能转换为经过组织和优化的应用代码方面还存在一些不确定性。 为了帮助您解决这些问题，我努力提供了一本简明的书，重点在于简洁的安装，简洁的应用设计以及对每个功能用途的简单理解。 希望您能从本书的项目中学习，扩大规模，并且仍然能够重用我们共同创建的开发环境和部分模块化代码。

特别是，在本书第一章的结尾，您可以拥有一个开发环境，该环境可以链接 Python，OpenCV，深度相机库（OpenNI，SensorKinect）和通用科学库（NumPy，SciPy）。 在五章之后，您可以拥有一个有趣的应用的多种变体，该应用可以在实时照相机源中操纵用户的脸部。 在此应用的后面，您将拥有一个小的可重用函数和类库，您可以在将来的计算机视觉项目中应用它们。 让我们更详细地看书的进展。

这本书涵盖的内容

第 1 章，“设置 OpenCV”让我们研究在 Windows，Mac 和 Ubuntu 上设置 Python，OpenCV 和相关库的步骤。 我们还将讨论 OpenCV 的社区，文档和官方代码示例。

第 2 章，“处理文件，相机和 GUI”有助于我们讨论 OpenCV 的 I/O 功能。 然后，使用面向对象的设计，我们编写了一个应用，该应用显示实时摄像机供稿，处理键盘输入以及编写视频和静态图像文件。

第 3 章，“过滤图像”帮助我们使用 OpenCV，NumPy 和 SciPy 编写图像过滤器。 过滤器效果包括线性颜色操纵，曲线颜色操纵，模糊，锐化和轮廓轮廓。 我们修改了我们的应用，以将其中的一些过滤器应用于实时摄像机供稿。

第 4 章，“使用 Haar 级联跟踪脸部”允许我们编写一个分层的脸部跟踪器，该跟踪器使用 OpenCV 定位图像中的脸部，眼睛，鼻子和嘴巴。 我们还编写了用于复制和调整图像区域大小的函数。 我们修改了我们的应用，以便它可以找到并处理相机供稿中的人脸。

第 5 章，“检测前景/背景区域和深度”有助于我们了解 OpenCV 可以从深度摄像机捕获的数据类型（在 OpenNI 和 SensorKinect 的支持

下）。 然后，我们编写使用此类数据将效果限制到前景区域的函数。 我们将此功能整合到我们的应用中，以便我们可以在操作面部区域之前进一步完善它们。

附录 A，“与 Pygame 集成”，让我们修改应用以使用 Pygame 代替 OpenCV 来处理某些 I/O 事件。 （Pygame 提供更多种事件处理功能。）

附录 B，“为自定义目标生成 Haar 级联”允许我们检查一组 OpenCV 工具，这些工具使我们能够为任何类型的对象或图案（不一定是面孔）构建跟踪器。

这本书需要什么

本书提供了所有相关软件的设置说明，包括包管理器，构建工具，Python，NumPy，SciPy，OpenCV，OpenNI 和 SensorKinect。 设置说明适用于 Windows XP 或更高版本，Mac OS 10.6（Snow Leopard）或更高版本以及 Ubuntu 12.04 或更高版本。 如果您愿意自己调整设置步骤，则其他类 Unix 的操作系统也应该工作。 您需要针对本书中所述的核心项目使用网络摄像头。 对于其他功能，项目的某些变体使用第二个网络摄像头，甚至使用与 OpenNI 兼容的深度摄像头，例如 Microsoft Kinect 或 Asus Xtion PRO。

这本书适合谁

本书非常适合计算机视觉新手并且喜欢通过应用开发学习的 Python 开发人员。 假设您以前有使用 Python 和命令行的经验。 对图像数据（例如，像素和颜色通道）的基本了解也将有所帮助。

约定

在本书中，您会发现许多可以区分不同类型信息的文本样式。 以下是这些样式的一些示例，并解释了其含义。

文本中的代码字如下所示：“Windows 上的两个可执行文件称为`ONopencv_createsamples.exe`和`ONopencv_traincascade.exe`。”

代码块设置如下：

```
negative
    desc.txt
    images
        negative 0.png
        negative 1.png
```

任何命令行输入或输出的编写方式如下：

```
> cd negative
> forfiles /m images\*.png /c "cmd /c echo
@relpath" > desc.txt
```

新术语和**重要词**以粗体显示。 您在屏幕上看到的单词，例如在菜单或对话框中，将以如下形式显示在文本中：“单击**下一步**按钮可将您移至下一个屏幕”。

注意

警告或重要提示会出现在这样的框中。

提示

提示和技巧如下所示。

一、设置 OpenCV

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/1.md

本章是设置 Python 2.7，OpenCV 和相关库的快速指南。 设置完成后，我们还将查看 OpenCV 的 Python 示例脚本和文档。

涵盖了以下相关库：

- **NumPy**：这是 OpenCV 的 Python 绑定的依赖项。 它提供数值计算功能，包括有效的数组。
- **SciPy**：这是与 NumPy 密切相关的科学计算库。 OpenCV 不需要它，但是它对于处理 OpenCV 映像中的数据很有用。
- **OpenNI**：这是 OpenCV 的可选依赖项。 它增加了对某些深度相机的支持，例如 Asus XtionPRO。
- **SensorKinect**：这是一个 OpenNI 插件，是 OpenCV 的可选依赖项。 它增加了对 Microsoft Kinect 深度相机的支持。

出于本书的目的，可以将 OpenNI 和 SensorKinect 视为可选的。 它们在第 5 章“分隔前景/背景区域深度”的整个过程中使用，但在其他各章或附录中未使用。

在撰写本文时，OpenCV 2.4.3 是最新版本。 在某些操作系统上，设置早期版本（2.3.1）更容易。 这些版本之间的差异不应影响我们将在本书中构建的项目。

OpenCV Wiki 中提供了一些其他信息，尤其是有关 OpenCV 的构建选项及其依赖项的信息。 但是，在撰写本文时，Wiki 尚未与 OpenCV 2.4.3 保持同步。

选择和使用正确的设置工具

我们可以自由选择各种设置工具，具体取决于我们的操作系统以及我们要执行的配置数量。 让我们来概述一下 Windows，Mac，Ubuntu 和其他类似 Unix 的系统的工具。

选择 Windows XP，Windows Vista，Windows 7 或 Windows 8

Windows 未预装 Python。 但是，安装向导可用于预编译的 Python，NumPy，SciPy 和 OpenCV。 或者，我们可以从源代码构建。 OpenCV 的构建系统使用 CMake 进行配置，使用 Visual Studio 或 MinGW 进行编译。

如果我们想支持包括 Kinect 在内的深度相机，我们应该首先安装 OpenNI 和 SensorKinect，它们可以作为预编译的二进制文件与安装向导一起使用。 然后，我们必须从源代码构建 OpenCV。

注意

OpenCV 的预编译版本不支持深度相机。

在 Windows 上，OpenCV 对 32 位 Python 的支持比对 64 位 Python 的支持更好。 即使我们是从源代码构建的，我也建议使用 32 位 Python。 幸运的是，32 位 Python 在 Windows 的 32 位或 64 位版本上都能正常工作。

注意

以下某些步骤涉及编辑系统的`Path`变量。 可以在**控制面板**的**环境变量**窗口中完成此任务。

在 Windows Vista/Windows 7/Windows 8 上，打开**开始**菜单，然后启动**控制面板**。 现在，转到**系统和安全性 | 系统 | 高级系统设置**。 单击**环境变量**按钮。

在 Windows XP 上，打开**开始**菜单，然后转到**控制面板 | 系统**。 选择**高级**标签。 单击**环境变量**按钮。

现在，在**系统变量**下，选择**路径**下一步**编辑**按钮。 按照指示进行更改。 要应用更改，请单击所有 **OK** 按钮（直到我们回到**控制面板**的主窗口中）。 然后，注销并重新登录（或者重新启动）。

使用二进制安装程序（不支持深度摄像头）

以下是将设置为 32 位 Python 2.7，NumPy 和 OpenCV 的步骤：

1. 从这个页面下载并安装 32 位 Python 2.7.3。
2. 从这个页面下载并安装 NumPy 1.6.2。
3. 从这个页面
4. 从这个页面下载 OpenCV 2.4.3 的自解压 ZIP。 运行自解压 ZIP，并在出现提示时输入任何目标文件夹，我们将其称为`<unzip_destination>`。 创建一个子文件夹`<unzip_destination>\opencv`。
5. 将`<unzip_destination>\opencv\build\python\2.7\cv2.pyd`复制到`C:\Python2.7\Lib\site-packages`（假设我们将 Python 2.7 安装到默认位置）。 现在，新安装的 Python 可以找到 OpenCV。
6. 如果我们希望默认情况下使用新的 Python 安装运行 Python 脚本，则需要最后一步。 编辑系统的`Path`变量并附加`;C:\Python2.7`（假设我们将 Python 2.7 安装到默认位置）。 删除所有以前的 Python 路径，例如`;C:\Python2.6`。 注销并重新登录（或者重新启动）。

使用 CMake 和编译器

Windows 不附带任何编译器或 CMake。 我们需要安装它们。 如果要支持包括 Kinect 在内的深度相机，还需要安装 OpenNI 和 SensorKinect。

假设我们已经从二进制文件（如前所述）或从源代码安装了 32 位 Python 2.7，NumPy 和 SciPy。 现在，我们可以继续安装编译器和 CMake，可以选择安装 OpenNI 和 SensorKinect，然后从源代码构建 OpenCV：

1. 从这个页面下载并安装 CMake 2.8.9。 运行安装程序时，选择**将 CMake 添加到所有用户的系统路径**或**将 CMake 添加到当前用户的系统路径**。

2. 下载并安装 Microsoft Visual Studio 2010，Microsoft Visual C++ Express 2010 或 MinGW。 请注意，OpenCV 2.4.3 不能使用较新的版本（Microsoft Visual Studio 2012 和 Microsoft Visual Studio Express 2012）进行编译。

   对于 Microsoft Visual Studio 2010，请使用购买的所有安装媒体。 在安装过程中，包括所有可选的 C++ 组件。 安装完成后重新启动。

对于 Microsoft Visual C++ Express 2010，请从这个页面获取安装程序。 安装完成后重新启动。

对于 MinGW，请从这个页面中获取安装程序。 运行安装程序时，请确保目标路径不包含空格，并且包括可选的 C++ 编译器。 编辑系统的`Path`变量，然后附加`;C:\MinGW\bin`（假定 MinGW 已安装到默认位置）。重新引导系统。

3. （可选）从这个页面下载并安装 OpenNI 1.5.4.0（32 位）。 另外，对于 64 位 Python，请使用这个页面（64 位）。

4. （可选）从这个页面下载并安装 SensorKinect 0.93（32 位）。 另外，对于 64 位 Python ，请使用这个页面（64 位）。

5. 从这个页面下载 OpenCV 2.4.3 的自解压 ZIP。 运行自解压 ZIP，并在出现提示时输入任何目标文件夹，我们将其称为`<unzip_destination>`。 创建一个子文件夹`<unzip_destination>\opencv`。

6. 打开命令提示符，然后在另一个文件夹中进行构建：

```
> mkdir<build_folder>
```

将目录更改为构建文件夹：

```
> cd <build_folder>
```

7. 现在，我们准备配置我们的构建。 要了解所有选项，我们可以阅读`<unzip_destination>\opencv\CMakeLists.txt`中的代码。 但是，出于本书的目的，我们只需要使用将为我们提供带有 Python 绑定的发行版本的选项，以及（可选）通过 OpenNI 和 SensorKinect 支持深度相机的选项即可。

对于 Visual Studio 2010 或 Visual C++ Express 2010，请运行：

```
> cmake -D:CMAKE_BUILD_TYPE=RELEASE -D:WITH_OPENNI=ON -G "Visual Studio
10" <unzip_destination>\opencv
```

或者，对于 MinGW，运行：

```
> cmake -D:CMAKE_BUILD_TYPE=RELEASE -D:WITH_OPENNI=ON -G
"MinGWMakefiles" <unzip_destination>\opencv
```

如果未安装 OpenNI，请省略`-D:WITH_OPENNI=ON`。 （在这种情况下，将不支持深度相机。）如果将 OpenNI 和 SensorKinect 安装在非默认位置，请修改命令以包括`-D:OPENNI_LIB_DIR=<openni_install_destination>\Lib -D:OPENNI_INCLUDE_DIR=<openni_install_destinati`。

CMake 可能报告它未能找到某些依赖项。 OpenCV 的许多依赖项是可选的。 因此，不要太在意。 如果构建无法完成或以后遇到问题，请尝试安装缺少的依赖项（通常以预构建的二进制文件形式提

供），然后从此步骤重新构建 OpenCV。

8. 配置好构建系统后，我们就可以进行编译了。

   对于 Visual Studio 或 Visual C++ Express，打开`<build_folder>/OpenCV.sln`。 选择`Release`配置并构建。 如果出现构建错误，请仔细检查是否已选择`Release`配置。

   或者，对于 MinGW，运行：

   ```
   > mingw32-make.
   ```

9. 将`<build_folder>\lib\Release\cv2.pyd`（来自 Visual Studio 版本）或`<build_folder>\lib\cv2.pyd`（来自 MinGW 版本）复制到`C:\Python2.7\Lib\site-packages`（假定将 Python 2.7 安装到默认位置）。 现在，Python 安装可以找到 OpenCV 的一部分。

10. 最后， 我们需要确保 Python 和其他进程可以找到其余的 OpenCV。 编辑系统的`Path`变量，并附加`;<build_folder>/bin/Release`（对于 Visual Studio 版本）或`;<build_folder>/bin`（对于 MinGW 版本）。 重新启动系统。

选择 Mac OS X Snow Leopard，Mac OS X Lion 或 Mac OS X Mountain Lion

Mac 的某些版本预装有 Python 2.7。 但是，Apple 预先定制了预装的 Python，以满足系统的内部需求。 通常，我们不应该在 Apple 的 Python 之上安装任何库。 如果这样做，我们的库可能会在系统更新期间中断，或者更糟的是，可能会与系统所需的预安装库冲突。 相反，我们应该安装标准 Python 2.7，然后在其顶部安装我们的库。

对于 Mac，有几种获取标准 Python 2.7，NumPy，SciPy 和 OpenCV 的可能方法。 所有方法最终都需要使用 Xcode Developer Tools 从源代码编译 OpenCV。 但是，根据方法的不同，第三方工具会以各种方式为我们自动完成此任务。 我们将研究使用 MacPorts 或 Homebrew 的方法。 这些工具可以完成 CMake 可以做的所有事情，此外，它们还可以帮助我们解决依赖关系并将开发库与系统库分离。

提示

我推荐 MacPorts，特别是如果您想通过 OpenNI 和 SensorKinect 编译具有深度摄像头支持的 OpenCV。 相关补丁和构建脚本（包括我维护的补丁）已为 MacPorts 准备就绪。 相比之下，Homebrew 当前不提供使用深度相机支持编译 OpenCV 的现成解决方案。

在继续之前，请确保正确设置 Xcode 开发人员工具：

1. 从 Mac App Store 或这个页面下载并安装 Xcode。 在安装过程中，如果可以选择安装**命令行工具**，请选择它。
2. 打开 Xcode 并接受许可协议。
3. 如果安装程序没有为我们提供安装**命令行工具**的选项，则最后一步是必需的。 转到 **Xcode | 首选项 | 下载**下一步**命令行工具**旁边的**安装**按钮。 等待安装完成并退出 Xcode。

现在，我们拥有任何方法所需的编译器。

将 MacPorts 与现成的包一起使用

我们可以使用 MacPorts 包管理器来帮助我们设置 Python 2.7，NumPy 和 OpenCV。 MacPorts 提供了终端命令，这些命令可自动执行各种**开源软件**（**OSS**）的下载，编译和安装过程。 MacPorts 还根据需要安装依赖项。 对于每个软件，依赖关系和构建秘籍都在称为 **Portfile** 的配置文件中定义。 MacPorts **存储库**是 Portfiles 的集合。

从已经设置了 Xcode 及其命令行工具的系统开始，以下步骤将为我们提供通过 MacPorts 进行的 OpenCV 安装：

1. 从这个页面下载并安装 MacPorts。
2. 如果我们想要支持 Kinect 深度相机，我们需要告诉 MacPorts 在哪里下载我编写的一些自定义 Portfile。 为此，请编辑`/opt/local/etc/macports/sources.conf`（假设 MacPorts 已安装到默认位置）。在`rsync://rsync.macports.org/release/ports/ [default]`行上方，添加以下行：

   ```
   http://nummist.com/opencv/ports.tar.gz
   ```

   保存文件。 现在，MacPorts 知道先在我的在线存储库中搜索端口文件，然后在默认的在线存储库中搜索。
3. 打开终端并运行以下命令以更新 MacPorts：

   ```
   $ sudo port selfupdate
   ```

   出现提示时，输入您的密码。
4. 现在（如果我们使用的是我的存储库），运行以下命令以安装带有 Python 2.7 绑定的 OpenCV 并支持包括 Kinect 在内的深度摄像机：

   ```
   $ sudo port install opencv +python27 +openni_sensorkinect
   ```

   或者（有或没有我的存储库），运行以下命令以安装具有 Python 2.7 绑定的 OpenCV 并支持除 Kinect 之外的深度相机：

   ```
   $ sudo port install opencv +python27 +openni
   ```

   依赖关系（包括 Python 2.7，NumPy，OpenNI 和（在第一个示例中）SensorKinect）也会自动安装。

   通过在命令中添加`+python27`，我们指定我们希望具有 Python 2.7 绑定的`opencv`变体（构建配置）。同样，`+openni_sensorkinect`通过 OpenNI 和 SensorKinect 指定了对深度摄像头的最大支持。 如果您不想使用深度相机，则可以省略`+openni_sensorkinect`；如果您不想使用与 OpenNI 兼容的深度相机，

而不能使用 Kinect，则可以将其替换为`+openni`。 要在安装之前查看可用变体的完整列表，我们可以输入：

```
$ port variants opencv
```

根据我们的定制需求，我们可以向`install`命令添加其他变体。 为了获得更大的灵活性，我们可以编写我们自己的变体（如下一节所述）。

5. 另外，运行以下命令来安装 SciPy：

```
$ sudo port install py27-scipy
```

6. Python 安装的可执行文件名为`python2.7`。 如果我们想将默认的`python`可执行文件链接到`python2.7`，我们也要运行：

```
$ sudo port install python_select
$ sudo port select python python27
```

将 MacPorts 与您自己的自定义包一起使用

通过一些额外的步骤，我们可以更改 MacPorts 编译 OpenCV 或任何其他软件的方式。 如前所述，MacPorts 的构建秘籍在称为 Portfiles 的配置文件中定义。 通过创建或编辑 Portfile，我们可以访问高度可配置的构建工具（例如 CMake），同时还可以受益于 MacPorts 的功能（例如依赖关系解析）。

假设我们已经安装了 MacPorts。 现在，我们可以配置 MacPorts 以使用我们编写的自定义 Portfile：

1. 在某个地方创建一个文件夹来保存我们的自定义 Portfile。 我们将此文件夹称为`<local_repository>`。
2. 编辑文件`/opt/local/etc/macports/sources.conf`（假设 MacPorts 已安装到默认位置）。
   在`rsync://rsync.macports.org/release/ports/ [default]`行上方，添加以下行：

   ```
   file://<local_repository>
   ```

   例如，如果`<local_repository>`为`/Users/Joe/Portfiles`，则添加：

   ```
   file:///Users/Joe/Portfiles
   ```

   注意三斜杠。

   保存文件。 现在，MacPorts 知道先在`<local_repository>`中搜索端口文件，然后在其默认的在线存储库中搜索。
3. 打开终端并更新 MacPorts，以确保我们具有默认存储库中的最新 Portfile：

```
$ sudo port selfupdate
```

4. 让我们以默认存储库的`opencv` Portfile 为例进行复制。 我们还应该复制目录结构，该结构确定 MacPorts 如何对包进行分类。

```
$ mkdir <local_repository>/graphics/
$ cp
/opt/local/var/macports/sources/rsync.macports.org/release/ports/graphic
s/opencv <local_repository>/graphics
```

另外，对于包含 Kinect 支持的示例，我们可以从这个页面下载我的在线存储库，解压缩并将其整个`graphics`文件夹复制到`<local_repository>`：

```
$ cp <unzip_destination>/graphics <local_repository>
```

5. 编辑`<local_repository>/graphics/opencv/Portfile`。 请注意，此文件指定 CMake 配置标志，依赖关系和变体。 有关端口文件编辑的详细信息，请转到这个页面。

要查看与 OpenCV 相关的 CMake 配置标志，我们需要查看其源代码。
从`http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.3/OpenCV-2.4.3.tar.bz2/downloa`下载源代码存档，将其解压缩到任何位置，然后阅读`<unzip_destination>/OpenCV-2.4.3/CMakeLists.txt`。

对 Portfile 进行任何编辑后，保存它。

6. 现在，我们需要在本地存储库中生成一个索引文件，以便 MacPorts 可以找到新的 Portfile：

```
$ cd <local_repository>
$ portindex
```

7. 从现在开始，我们可以像对待其他 MacPorts 包一样对待自定义`opencv`。 例如，我们可以按以下方式安装它：

```
$ sudo port install opencv +python27 +openni_sensorkinect
```

注意，本地存储库的 Portfile 优先于默认存储库的 Portfile，这是因为`/opt/local/etc/macports/sources.conf`中列出的顺序。

将 Homebrew 与现成的包一起使用（不支持深度相机）

Homebrew 是另一个可以帮助我们的包管理器。 通常，MacPorts 和 Homebrew 不应安装在同一台计算机上。

从已经安装了 Xcode 及其命令行工具的系统开始，以下步骤将通过 Homebrew 为我们提供 OpenCV 安装：

1. 打开终端并运行以下命令来安装 Homebrew：

```
$ ruby -e "$(curl -fsSkLraw.github.com/mxcl/homebrew/go)"
```

2. 与 MacPorts 不同，Homebrew 不会自动将其可执行文件放入`PATH`中。 为此，创建或编辑文件`~/.profile`，并在顶部添加以下行：

```
export PATH=/usr/local/bin:/usr/local/sbin:$PATH
```

保存文件并运行以下命令以刷新`PATH`：

```
$ source ~/.profile
```

请注意，由 Homebrew 安装的可执行文件现在优先于由系统安装的可执行文件。

3. 对于 Homebrew 的自我诊断报告，请运行：

```
$ brew doctor
```

遵循其提供的任何故障排除建议。

4. 现在，更新 Homebrew：

```
$ brew update
```

5. 运行以下命令以安装 Python 2.7：

```
$ brew install python
```

6. 现在，我们可以安装 NumPy。 Homebrew 对 Python 库包的选择是有限的，因此我们使用 Homebrew 的 Python 随附的名为 **PIP** 的单独的包管理工具：

```
$ pip install numpy
```

7. SciPy 包含一些 Fortran 代码，因此我们需要适当的编译器。 我们可以使用 Homebrew 来安装 gfortran 编译器：

```
$ brew install gfortran
```

现在，我们可以安装 SciPy：

```
$ pip install scipy
```

8. 要在 64 位系统上安装 OpenCV（自 2006 年底以来，所有 Mac 新硬件），请运行：

```
$ brew install opencv
```

或者，要在 32 位系统上安装 OpenCV，请运行：

```
$ brew install opencv --build32
```

提示

下载示例代码

您可以通过`http://www.packtpub.com`下载从帐户购买的所有 Packt 图书的示例代码文件。 如果您在其他地方购买了此书，则可以访问`http://www.packtpub.com/support`并注册以将文件直接通过电子邮件发送给您。

将 Homebrew 与您自己的自定义包一起使用

Homebrew 使编辑现有包定义变得容易：

```
$ brew edit opencv
```

包定义实际上是 Ruby 编程语言中的脚本。 可在 Homebrew Wiki 中找到有关编辑它们的提示。 脚本可以指定 Make 或 CMake 配置标志等。

要查看与 OpenCV 相关的 CMake 配置标志，我们需要查看其源代码。 从这个页面下载源代码存档，将其解压缩到任何位置，然后阅读`<unzip_destination>/OpenCV-2.4.3/CMakeLists.txt`。

对 Ruby 脚本进行任何编辑后，保存它。

定制包装可以视为正常包装。 例如，可以按以下方式安装：

```
$ brew install opencv
```

选择 Ubuntu 12.04 LTS 或 Ubuntu 12.10

Ubuntu 预先安装了 Python 2.7。 标准的 Ubuntu 存储库包含不支持深度摄像头的 OpenCV 2.3.1 包。 另外，可以使用 CMake 和 GCC 从源代码构建 OpenCV 2.4.3。 从源代码构建时，OpenCV 可以通过 OpenNI 和 SensorKinect 支持深度相机，这些相机可以作为预编译的二进制文件与安装脚本一起使用。

使用 Ubuntu 存储库（不支持深度相机）

我们可以使用 Apt 包管理器安装 OpenCV 2.3.1 及其依赖项：

1. 打开终端并运行此命令以更新 Apt：

```
$ sudo apt-get update
```

2. 现在，运行以下命令以使用 Python 绑定安装 NumPy，SciPy 和 OpenCV：

```
$ sudo apt-get install python-numpy
$ sudo apt-get install python-scipy
$ sudo apt-get install libopencv-*
$ sudo apt-get install python-opencv
```

   出现有关包安装的提示时，输入`Y`。

同样，我们可以使用 Ubuntu 软件中心，它是 Apt 的图形前端。

通过可自定义的现成脚本使用 CMake

Ubuntu 预先安装了 GCC 编译器。 但是，我们需要安装 CMake 构建系统。 我们还需要安装或重新安装其他各种库，其中一些库需要进行特殊配置以与 OpenCV 兼容。 因为依赖关系很复杂，所以我编写了一个脚本来下载，配置和构建 OpenCV 及其相关库，以便最终的 OpenCV 安装支持包括 Kinect 在内的深度摄像机：

1. 从这个页面下载我的安装脚本，并将其放在任何位置，例如`<script_folder>`。
2. （可选）编辑脚本以自定义 OpenCV 的构建配置。 若要查看与 OpenCV 相关的 CMake 配置标志，我们需要查看其源代码。 从这个页面下载源代码存档，将其解压缩到任何位置，然后读取`<unzip_destination>/OpenCV-2.4.3/CMakeLists.txt`。

   对脚本进行任何编辑后，将其保存。
3. 打开终端并运行此命令以更新 Apt：

```
$ sudo apt-get update
```

4. 将目录更改为`<script_folder>`：

```
$ cd <script_folder>
```

   设置脚本的权限，使其可执行：

```
$ chmod +x install_opencv_ubuntu.sh
```

   执行脚本：

```
$ ./install_opencv_ubuntu.sh
```

   出现提示时，输入密码。 出现有关包安装的提示时，输入`Y`。
5. 安装脚本创建一个文件夹`<script_folder>/opencv`，其中包含该脚本临时使用的下载和构建文件。 安装脚本终止后，可以安全地删除`<script_folder>/opencv`。 不过，首先，您可能想

在`<script_folder>/opencv/samples/python`和`<script_folder>/opencv/samples/python2`中查看 OpenCV 的 Python 示例。

选择其他类似 Unix 的系统

用于 Ubuntu 的方法（如前所述）可能适用于从 Ubuntu 12.04 LTS 或 Ubuntu 12.10 派生的任何 Linux 发行版，例如：

- Kubuntu 12.04 LTS 或 Kubuntu 12.10
- Xubuntu 12.04 LTS 或 Xubuntu 12.10
- Linux Mint 13 或 Linux Mint 14

在 Debian Linux 及其衍生版本上，Apt 包管理器的工作方式与在 Ubuntu 上相同，尽管可用包可能有所不同。

在 Gentoo Linux 及其衍生版本上，Portage 包管理器类似于 MacPorts（如前所述），尽管可用包可能有所不同。

在其他类似 Unix 的系统上，包管理器和可用包可能会有所不同。 请查阅包管理员的文档，并搜索名称中带有`opencv`的任何包。 请记住，OpenCV 及其 Python 绑定可能会分为多个包。

另外，查找由系统提供商，存储库维护者或社区发布的所有安装说明。 由于 OpenCV 使用摄像机驱动程序和媒体编解码器，因此在多媒体支持较差的系统上，使其所有功能正常工作可能会很棘手。 在某些情况下，可能需要重新配置或重新安装系统包才能兼容。

如果包可用于 OpenCV，请检查其版本号。 为了本书的目的，建议使用 OpenCV 2.3.1 或更高版本。 还要检查包是否提供 Python 绑定，以及它们是否通过 OpenNI 和 SensorKinect 提供深度相机支持。 最后，检查开发人员社区中是否有人报告使用包的成功或失败。

相反，如果我们想从源代码中进行 OpenCV 的自定义构建，则参考 Ubuntu 的安装脚本（先前已讨论过）并将其调整为适用于另一个系统上的包管理器和包可能会有帮助。

运行示例

运行一些示例脚本是测试 OpenCV 正确设置的好方法。 这些示例包含在 OpenCV 的源代码存档中。

在 Windows 上，我们应该已经下载并解压缩了 OpenCV 的自解压 ZIP。
在`<unzip_destination>/opencv/samples`中找到样本。

在包括 Mac 在内的类似 Unix 的系统上，请从这个页面下载源代码归档文件并将其解压缩到任何位置（如果我们尚未这样做的话）。 在`<unzip_destination>/OpenCV-2.4.3/samples`中找到样本。

一些示例脚本需要命令行参数。 但是，以下脚本（以及其他脚本）应该在没有任何参数的情况下运行：

- `python/camera.py`：这将显示网络摄像头供稿（假设已插入网络摄像头）。
- `python/drawing.py`：绘制一系列形状，如屏幕保护程序。
- `python2/hist.py`：显示照片。 按`A`，`B`，`C`，`D`或`E`查看照片的变化，以及颜色或灰度值的相应直方图。

- `python2/opt_flow.py`（Ubuntu 包中缺少）：这将显示一个网络摄像头源，其中叠加了可视化的光流（运动方向）。 例如，在网络摄像头上慢慢挥手以查看效果。 按`1`或`2`进行可视化显示。

要退出脚本，请按`Esc`（不是窗口的关闭按钮）。

如果遇到消息`ImportError: No module named cv2.cv`，那么我们正在从不了解 OpenCV 的 Python 安装中运行脚本。 有两种可能的解释：

- OpenCV 安装中的某些步骤可能失败或错过。 返回并查看步骤。
- 如果在计算机上安装了多个 Python，则可能是使用了错误的 Python 启动脚本。 例如，在 Mac 上，可能是为 MacPorts Python 安装了 OpenCV，但我们正在使用系统的 Python 运行脚本。 返回并查看有关编辑系统路径的安装步骤。 另外，尝试使用以下命令从命令行手动启动脚本：

  ```
  $ python python/camera.py
  ```

  您还可以使用以下命令：

  ```
  $ python2.7 python/camera.py
  ```

  作为选择其他 Python 安装的另一种可能的方法，请尝试编辑示例脚本以删除`#!`行。 这些行可能会将脚本与错误的 Python 安装（对于我们的特定设置）明确关联。

查找文档，帮助和更新

OpenCV 的文档在线发布在这个页面上。 该文档包含有关 OpenCV 的新 C++ API，其新的 Python API（基于 C++ API），其旧的 C API 和其旧的 Python API（基于 C API）的组合 API 参考。 查找类或函数时，请确保阅读有关新 Python API（`cv2`模块）而不是旧 Python API（`cv`模块）的部分。

注意

标题为 **OpenCV 2.1 Python 参考**的文档可能会显示在 Google 搜索`OpenCV Python API`的文档中。 避免使用此文档，因为它已过时并且仅涵盖旧的（类似于 C 的）Python API。

该文档也可以作为几个可下载的 PDF 文件获得：

- **API 参考**
- **教程** （这些教程使用 C++ 代码。有关教程代码的 Python 端口，参见 Abid RahmanK。 的资料库位于这个页面。）
- **用户指南（不完整）**

如果您在飞机上或无法访问互联网的其他地方编写代码，则肯定要保留文档的脱机副本。

如果文档似乎无法回答您的问题，请尝试与 OpenCV 社区联系。 在一些网站上，您会找到有用的人：

- **OpenCV 官方论坛**

- **DavidMillánEscrivá 的博客（本书的审阅者之一）**
- **Abid Rahman K. 博客（本书的审阅者之一）**
- **我这本书的网站**

最后，如果您是高级用户，想尝试使用来自最新（不稳定）的 OpenCV 源代码的新功能，错误修复和示例脚本，请查看项目的存储库。

总结

到现在为止，我们应该已经安装了 OpenCV，该安装可以完成本书中描述的项目所需的一切。 根据我们采用的方法，我们可能还会有一组工具和脚本，可用于重新配置和重建 OpenCV，以满足我们未来的需求。

我们知道在哪里可以找到 OpenCV 的 Python 示例。 这些示例涵盖的功能范围不同于本书的项目，但它们可作为额外的学习辅助工具。

二、处理文件，相机和 GUI

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/2.md

本章介绍 OpenCV 的 I/O 功能。 我们还将讨论项目概念以及该项目面向对象设计的开始，我们将在后续章节中充实它们。

通过研究 I/O 功能和设计模式，我们以制作三明治的方式构建项目：从外部开始。面包切片和涂抹或端点和胶粘，先于馅料或算法。 我们之所以选择这种方法，是因为计算机视觉是外向的，它考虑了计算机外部的真实世界，并且我们希望通过通用接口将所有后续的算法工作应用于真实世界。

注意

可以从我的网站下载本章的所有完成代码。

基本 I/O 脚本

所有 CV 应用都需要获取图像作为输入。 大多数还需要产生图像作为输出。 交互式 CV 应用可能需要将摄像机作为输入源，将窗口作为输出目标。 但是，其他可能的来源和目的地包括图像文件，视频文件，和原始字节。 例如，如果我们将过程图形合并到我们的应用中，原始字节可能是通过网络连接接收/发送的，或者可能是由算法生成的。 让我们看看每种可能性。

读取/写入图像文件

OpenCV 提供，，`imread()`和`imwrite()`函数，这些函数支持静态图像的各种文件格式。 支持的格式因系统而异，但应始终包括 BMP 格式。 通常，PNG，JPEG 和 TIFF 也应该是受支持的格式。 可以从一种文件格式加载图像并保存为另一种文件格式。 例如，让我们将图像从 PNG 转换为 JPEG：

```
import cv2

image = cv2.imread('MyPic.png')
cv2.imwrite('MyPic.jpg', image)
```

注意

我们使用的大多数 OpenCV 功能都在`cv2`模块中。 您可能会遇到其他 OpenCV 指南，它们依赖于`cv`或`cv2.cv`模块（它们是旧版）。 对于某些尚未在`cv2`中重新定义的常量，我们确实使用了`cv2.cv`。

默认情况下，即使文件使用灰度格式，`imread()`也会以 BGR 颜色格式返回图像。 **BGR**（**蓝绿红**）表示与 **RGB**（**红绿蓝**）相同的色彩空间，但字节顺序是相反的。

（可选）我们可以将`imread()`的模式指定为`CV_LOAD_IMAGE_COLOR`（BGR），`CV_LOAD_IMAGE_GRAYSCALE`（灰度）或`CV_LOAD_IMAGE_UNCHANGED`（BGR 或灰度，取决于文件的色彩空间）。 例如，让我们将 PNG 加载为灰度图像（过程中丢失任何颜色信息），然后将其另存为灰度 PNG 图像：

```
import cv2

grayImage = cv2.imread('MyPic.png',
```

```
cv2.CV_LOAD_IMAGE_GRAYSCALE)
cv2.imwrite('MyPicGray.png', grayImage)
```

无论哪种模式，`imread()`都会丢弃任何 alpha 通道（透明度）。 `imwrite()`函数要求图像为 BGR 或灰度格式，每个通道的位数可以由输出格式支持。 例如，`bmp`每个通道需要 8 位，而 PNG 允许每个通道 8 位或 16 位。

在图像和原始字节之间转换

从概念上讲，字节是从 0 到 255 的整数。在当今的实时图形应用中，像素通常由每个通道一个字节表示，尽管其他表示形式也是可能的。

OpenCV 图像是`numpy.array`类型的 2D 或 3D 数组。 8 位灰度图像是包含字节值的 2D 数组。 24 位 BGR 图像是 3D 数组，也包含字节值。 我们可以使用`image[0, 0]`或`image[0, 0, 0]`之类的表达式来访问这些值。 第一个索引是像素的`y`坐标或行，`0`是顶部。 第二个索引是像素的`x`坐标或列，`0`是最左侧。 第三个索引（如果适用）表示颜色通道。

例如，在左上角具有白色像素的 8 位灰度图像中，`image[0, 0]`为`255`。 对于左上角带有蓝色像素的 24 位 BGR 图像，`image[0, 0]`为`[255, 0, 0]`。

注意

作为使用类似`image[0, 0]`或`image[0, 0] = 128`的表达式的替代方法，我们可以使用类似`image.item((0, 0))`或`image.setitem((0, 0), 128)`的表达式。 后者的表达式对于单像素操作更有效。 但是，正如我们将

在后续章节中看到的那样，我们通常希望对图像的大片段而不是单个像素执行操作。

假设每通道图像有 8 位，我们可以将其转换为一维的标准 Python `bytearray`：

```
byteArray = bytearray(image)
```

相反，只要`bytearray`包含适当顺序的字节，我们就可以进行铸造然后对其进行整形，以获得作为图像的`numpy.array`类型：

```
grayImage =
numpy.array(grayByteArray).reshape(height, width)
bgrImage =
numpy.array(bgrByteArray).reshape(height, width,
3)
```

作为更完整的示例，让我们将包含随机字节的`bytearray`转换为灰度图像和 BGR 图像：

```
import cv2
import numpy
import os

# Make an array of 120,000 random bytes.
randomByteArray = bytearray(os.urandom(120000))
flatNumpyArray = numpy.array(randomByteArray)

# Convert the array to make a 400x300 grayscale
image.
grayImage = flatNumpyArray.reshape(300, 400)
cv2.imwrite('RandomGray.png', grayImage)
```

```
# Convert the array to make a 400x100 color
image.
bgrImage = flatNumpyArray.reshape(100, 400, 3)
cv2.imwrite('RandomColor.png', bgrImage)
```

运行此脚本之后，我们应该在脚本目录中有一对随机生成的图像`RandomGray.png`和`RandomColor.png`。

注意

在这里，我们使用 Python 的标准`os.urandom()`函数生成随机原始字节，然后将其转换为 Numpy 数组。 注意，也可以使用`numpy.random.randint(0, 256, 120000).reshape(300, 400)`之类的语句直接（更有效地）生成随机 Numpy 数组。 我们使用`os.urandom()`的唯一原因是帮助演示从原始字节的转换。

读/写视频文件

OpenCV 提供支持各种视频文件格式的`VideoCapture`和`VideoWriter`类。 支持的格式因系统而异，但应始终包含 AVI。 通过其`read()`方法，可以轮询`VideoCapture`类以获取新帧，直到到达其视频文件的末尾。 每帧都是 BGR 格式的图像。 相反，可以将图像传递到`VideoWriter`类的`write()`方法，该方法将图像附加到`VideoWriter`中的文件。 让我们看一个示例，该示例从一个 AVI 文件读取帧并将其写入到另一个具有 YUV 编码的 AVI 文件中：

```
import cv2

videoCapture = cv2.VideoCapture('MyInputVid.avi')
fps = videoCapture.get(cv2.cv.CV_CAP_PROP_FPS)
```

```
size =
(int(videoCapture.get(cv2.cv.CV_CAP_PROP_FRAME_WI
DTH)),

 int(videoCapture.get(cv2.cv.CV_CAP_PROP_FRAME_HE
IGHT)))
videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi',
cv2.cv.CV_FOURCC('I','4','2','0'), fps, size)

success, frame = videoCapture.read()
while success: # Loop until there are no more
frames.
    videoWriter.write(frame)
    success, frame = videoCapture.read()
```

`VideoWriter`类的构造器的参数值得特别注意。 必须指定视频的文件名。 具有该名称的任何先前存在的文件都将被覆盖。 还必须指定视频编解码器。 可用的编解码器可能因系统而异。 选项包括：

- `cv2.cv.CV_FOURCC('I','4','2','0')`：这是未压缩的 YUV，4:2:0 色度被二次采样。 这种编码具有广泛的兼容性，但会产生大文件。 文件扩展名应为`avi`。
- `cv2.cv.CV_FOURCC('P','I','M','1')`：这是 MPEG-1。 文件扩展名应为`avi`。
- `cv2.cv.CV_FOURCC('M','J','P','G')`：这是动态 JPEG。 文件扩展名应为`avi`。
- `cv2.cv.CV_FOURCC('T','H','E','O')`：这是 Ogg-Vorbis。 文件扩展名应为`ogv`。
- `cv2.cv.CV_FOURCC('F','L','V','1')`：这是 Flash 视频。 文件扩展名应为`flv`。

还必须指定帧速率和帧大小。 由于我们正在从另一个视频复制，因此可以从`VideoCapture`类的`get()`方法读取这些属性。

捕捉相机帧

摄像机帧的流也由`VideoCapture`类表示。 但是，对于摄像机，我们通过传递摄像机的设备索引而不是视频的文件名来构造`VideoCapture`类。让我们考虑一个示例，该示例从摄像机捕获 10 秒的视频并将其写入 AVI 文件：

```
import cv2

cameraCapture = cv2.VideoCapture(0)
fps = 30 # an assumption
size =
(int(cameraCapture.get(cv2.cv.CV_CAP_PROP_FRAME_W
IDTH)),

 int(cameraCapture.get(cv2.cv.CV_CAP_PROP_FRAME_H
EIGHT)))
videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi',
cv2.cv.CV_FOURCC('I','4','2','0'), fps, size)

success, frame = cameraCapture.read()
numFramesRemaining = 10 * fps - 1
while success and numFramesRemaining > 0:
    videoWriter.write(frame)
    success, frame = cameraCapture.read()
    numFramesRemaining -= 1
```

不幸的是，`VideoCapture`类的`get()`方法无法返回相机帧频的准确值；它总是返回`0`。 为了为相机创建合适的`VideoWriter`类，我们必须对帧

速率进行假设（就像我们之前在代码中所做的那样），或者使用计时器对其进行测量。 后一种方法更好，我们将在本章后面介绍。

摄像机的数量及其顺序当然取决于系统。 不幸的是，OpenCV 没有提供任何查询摄像机数量或其属性的方法。 如果使用无效索引来构造`VideoCapture`类，则`VideoCapture`类将不会产生任何帧。 其`read()`方法将返回`(false, None)`。

当我们需要同步一组摄像机或多头摄像机（如立体摄像机或 Kinect）时，`read()`方法不合适。 然后，我们改用`grab()`和`retrieve()`方法。 对于一组相机：

```
success0 = cameraCapture0.grab()
success1 = cameraCapture1.grab()
if success0 and success1:
    frame0 = cameraCapture0.retrieve()
    frame1 = cameraCapture1.retrieve()
```

对于多头相机，我们必须指定头的索引作为`retrieve()`的参数：

```
success = multiHeadCameraCapture.grab()
if success:
    frame0 =
multiHeadCameraCapture.retrieve(channel = 0)
    frame1 =
multiHeadCameraCapture.retrieve(channel = 1)
```

我们将在第 5 章，“检测前景/背景区域和深度”中更详细地研究多头相机。

在窗口中显示摄像机帧

OpenCV 允许使用`namedWindow()`，`imshow()`和`destroyWindow()`函数创建，重绘和销毁命名窗口。 同样，任何窗口都可以通过`waitKey()`函数捕获键盘输入，并通过`setMouseCallback()`函数捕获鼠标输入。 让我们看一个显示实时摄像机输入帧的示例：

```
import cv2

clicked = False
def onMouse(event, x, y, flags, param):
    global clicked
    if event == cv2.cv.CV_EVENT_LBUTTONUP:
        clicked = True

cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('MyWindow')
cv2.setMouseCallback('MyWindow', onMouse)

print 'Showing camera feed. Click window or press
any key to stop.'
success, frame = cameraCapture.read()
while success and cv2.waitKey(1) == -1 and not
clicked:
    cv2.imshow('MyWindow', frame)
    success, frame = cameraCapture.read()

cv2.destroyWindow('MyWindow')
```

`waitKey()`的参数是等待键盘输入的毫秒数。 返回值可以是`-1`（意味着没有按下任何键）或 ASCII 键码，例如`Esc`的`27`。 有关 ASCII 键码的列表，请参见这个页面。 另外，请注意，Python 提供了标准函数`ord()`，该函数可以将字符转换为其 ASCII 键代码。 例如`ord('a') returns 97`。

提示

在某些系统上，`waitKey()`可能返回一个值，该值编码的不仅是 ASCII 键码。（众所周知，当 OpenCV 使用 GTK 作为其后端 GUI 库时，在 Linux 上会发生一个错误。）在所有系统上，我们都可以通过读取返回值的最后一个字节来确保仅提取 ASCII 键码，如下所示：

```
keycode = cv2.waitKey(1)
if keycode != -1:
    keycode &= 0xFF
```

OpenCV 的窗口功能和`waitKey()`是相互依赖的。仅当调用`waitKey()`时才更新 OpenCV 窗口，并且仅当 OpenCV 窗口具有焦点时`waitKey()`才捕获输入。

如代码示例所示，传递给`setMouseCallback()`的鼠标回调应采用五个参数。回调的`param`参数设置为`setMouseCallback()`的可选第三个参数。默认为`0`。回调的`event`参数是以下之一：

- `cv2.cv.CV_EVENT_MOUSEMOVE`：鼠标移动
- `cv2.cv.CV_EVENT_LBUTTONDOWN`：向下按钮按下
- `cv2.cv.CV_EVENT_RBUTTONDOWN`：向下按钮按下
- `cv2.cv.CV_EVENT_MBUTTONDOWN`：中间按钮按下
- `cv2.cv.CV_EVENT_LBUTTONUP`：向左按钮
- `cv2.cv.CV_EVENT_RBUTTONUP`：向上按钮
- `cv2.cv.CV_EVENT_MBUTTONUP`：中键向上
- `cv2.cv.CV_EVENT_LBUTTONDBLCLK`：双击鼠标左键
- `cv2.cv.CV_EVENT_RBUTTONDBLCLK`：右键单击双击
- `cv2.cv.CV_EVENT_MBUTTONDBLCLK`：双击中键

鼠标回调的`flags`参数可能是以下各项的按位组合：

- `cv2.cv.CV_EVENT_FLAG_LBUTTON`：按下左键
- `cv2.cv.CV_EVENT_FLAG_RBUTTON`：按下右键
- `cv2.cv.CV_EVENT_FLAG_MBUTTON`：按下中间键
- `cv2.cv.CV_EVENT_FLAG_CTRLKEY`：按下`Ctrl`键
- `cv2.cv.CV_EVENT_FLAG_SHIFTKEY`：按下`Shift`键
- `cv2.cv.CV_EVENT_FLAG_ALTKEY`：按下`Alt`键

不幸的是，OpenCV 不提供任何处理窗口事件的方法。 例如，当单击窗口的关闭按钮时，我们无法停止我们的应用。 由于 OpenCV 有限的事件处理和 GUI 功能，许多开发人员更喜欢将其与另一个应用框架集成。 在本章的后面，我们将设计一个抽象层，以帮助将 OpenCV 集成到任何应用框架中。

项目概念

OpenCV 通常是通过菜谱方法研究的，该菜谱方法涵盖许多算法，但与高级应用开发无关。 在某种程度上，这种方法是可以理解的，因为 OpenCV 的潜在应用是如此多样。 例如，我们可以在照片/视频编辑器，动作控制游戏，机器人的 AI 或心理学实验中使用它来记录参与者的眼球运动。 在这样的不同用例中，我们可以真正研究一组有用的抽象吗？

我相信我们可以并且越早开始创建抽象越好。 我们将围绕单个应用来组织对 OpenCV 的研究，但是，在每一步中，我们都将设计此应用的组件以使其可扩展和可重用。

我们将开发一个交互式应用，该应用可实时对摄像机输入进行面部跟踪和图像处理。 这类应用涵盖了 OpenCV 的广泛功能，并给我们提出了创建高效有效实现的挑战。 用户会立即发现缺陷，例如帧速率低或跟踪不准确。 为了获得最佳结果，我们将尝试使用常规成像和深度成像的几种方法。

具体来说，我们的应用将执行实时面部合并。 给定两个摄像机输入流（或可选地，预录制的视频输入），应用会将一个流中的人脸叠加在另一个流中的人脸之上。 将应用过滤器和变形以使混合场景具有统一的外观。 用户应具有进入另一环境和另一角色的现场表演的经验。 这种类型的用户体验在迪士尼乐园等游乐园中很受欢迎。

我们将调用我们的应用 Cameo。 **Cameo** 是（在珠宝中）人物的小肖像，（在电影中）是名人扮演的非常简短的角色。

面向对象的设计

Python 应用可以以纯粹的程序样式编写。 这通常是通过小型应用完成的，例如前面讨论过的基本 I/O 脚本。 但是，从现在开始，我们将使用面向对象的样式，因为它可以促进模块化和可扩展性。

从我们对 OpenCV 的 I/O 功能的概述中，我们知道所有图像都是相似的，无论它们的来源还是目的地。 无论我们如何获取图像流或将其作为输出发送到哪里，我们都可以将相同的特定于应用的逻辑应用于该流中的每个帧。 在像 Cameo 这样的使用多个 I/O 流的应用中，I/O 代码和应用程序代码的分离变得特别方便。

我们将创建名为`CaptureManager`和`WindowManager`的类作为 I/O 流的高级接口。 我们的应用代码可以使用`CaptureManager`读取新帧，并且可以

选择将每个帧分派到一个或多个输出，包括静止图像文件，视频文件和窗口（通过`WindowManager`类）。 `WindowManager`类允许我们的应用代码以面向对象的方式处理窗口和事件。

`CaptureManager`和`WindowManager`都是可扩展的。 我们可以实现不依赖 OpenCV 进行 I/O 的实现。 实际上，附录 A“与 Pygame 集成”使用了`WindowManager`子类。

抽象化视频流 – `manager.CaptureManager`

如我们所见，OpenCV 可以捕获，显示和记录来自视频文件或摄像机的图像流，但是每种情况都有一些特殊的注意事项。 我们的`CaptureManager`类抽象了一些差异，并提供了更高级别的接口，用于将图像从捕获流分发到一个或多个输出（静止图像文件，视频文件或窗口）。

`CaptureManager`类由`VideoCapture`类初始化，并具有`enterFrame()`和`exitFrame()`方法，通常应在应用主循环的每次迭代中调用它们。 在对`enterFrame()`的调用与对`exitFrame()`的调用之间，应用可以（任意次）设置`channel`属性并获得`frame`属性。 `channel`属性最初是`0`，只有多头相机使用其他值。 `frame`属性是与调用`enterFrame()`时当前通道状态相对应的图像。

`CaptureManager`类也具有可以随时调用的`writeImage()`， `startWritingVideo()`和`stopWritingVideo()`方法。 实际文件写入被推迟到`exitFrame()`为止。 同样在`exitFrame()`方法期间，`frame`属性可能会显示在窗口中，具体取决于应用代码是否提供`WindowManager`类作为`CaptureManager`构造器的参数，还是通过设置属性[`previewWindowManager`。

如果应用代码操纵`frame`，则操纵将反映在任何记录的文件和窗口中。`CaptureManager`类具有构造器参数和称为`shouldMirrorPreview`的属性，如果我们希望`frame`在窗口中而不是在已记录文件中进行镜像（水平翻转），则应为`True`。 通常，面对摄像机时，用户希望对实时摄像机源进行镜像。

回想一下`VideoWriter`类需要帧速率，但是 OpenCV 没有提供任何方法来获取摄像机的准确帧速率。 `CaptureManager`类通过使用帧计数器和 Python 的标准`time.time()`函数估计帧率来解决此限制。 这种方法不是万无一失的。 根据帧频波动和`time.time()`的系统相关实现，在某些情况下，估计的准确率可能仍然很差。 但是，如果我们要部署到未知的硬件，则比仅假设用户的摄像机具有特定的帧速率要好。

让我们创建一个名为`managers.py`的文件，其中将包含`CaptureManager`的实现。 事实证明，实现时间很长。 因此，我们将分几部分进行研究。 首先，让我们添加导入，构造器和属性，如下所示：

```
import cv2
import numpy
import time

class CaptureManager(object):

    def __init__(self, capture,
previewWindowManager = None,
                 shouldMirrorPreview = False):

        self.previewWindowManager =
previewWindowManager
        self.shouldMirrorPreview =
shouldMirrorPreview
```

```
        self._capture = capture
        self._channel = 0
        self._enteredFrame = False
        self._frame = None
        self._imageFilename = None
        self._videoFilename = None
        self._videoEncoding = None
        self._videoWriter = None

        self._startTime = None
        self._framesElapsed = long(0)
        self._fpsEstimate = None

    @property
    def channel(self):
        return self._channel

    @channel.setter
    def channel(self, value):
        if self._channel != value:
            self._channel = value
            self._frame = None

    @property
    def frame(self):
        if self._enteredFrame and self._frame is
None:
            _, self._frame =
self._capture.retrieve(channel = self.channel)
        return self._frame

    @property
    def isWritingImage (self):

        return self._imageFilename is not None
```

```
    @property
    def isWritingVideo(self):
        return self._videoFilename is not None
```

请注意，大多数成员变量都是非公开的，如变量名中的下划线前缀所表示，例如`self._enteredFrame`。 这些非公共变量与当前帧的状态以及任何文件写入操作有关。 如前所述，应用代码只需要配置一些东西，这些东西就可以作为构造器参数和可设置的公共属性来实现：相机通道，窗口管理器和镜像相机预览的选项。

注意

按照惯例，在 Python 中，以单个下划线为前缀的变量应被视为**受保护的**（只能在该类及其子类中访问），而以双下划线为前缀的变量应视为**私有**（仅在该类内访问）。

继续执行，让我们将`enterFrame()`和`exitFrame()`方法添加到`managers.py`中：

```
    def enterFrame(self):
        """Capture the next frame, if any."""

        # But first, check that any previous
frame was exited.
        assert not self._enteredFrame, \
            'previous enterFrame() had no
matching exitFrame()'

        if self._capture is not None:
            self._enteredFrame =
self._capture.grab()
```

```
    def exitFrame (self):
        """Draw to the window. Write to files.
Release the frame."""

        # Check whether any grabbed frame is
retrievable.
        # The getter may retrieve and cache the
frame.
        if self.frame is None:
            self._enteredFrame = False
            return

        # Update the FPS estimate and related
variables.
        if self._framesElapsed == 0:
            self._startTime = time.time()
        else:
            timeElapsed = time.time() -
self._startTime
            self._fpsEstimate =
 self._framesElapsed / timeElapsed
        self._framesElapsed += 1

        # Draw to the window, if any.
        if self.previewWindowManager is not None:
            if self.shouldMirrorPreview:
                mirroredFrame =
numpy.fliplr(self._frame).copy()

 self.previewWindowManager.show(mirroredFrame)
            else:

 self.previewWindowManager.show(self._frame)

        # Write to the image file, if any.
        if self.isWritingImage:
```

```
            cv2.imwrite(self._imageFilename,
self._frame)
            self._imageFilename = None

        # Write to the video file, if any.
        self._writeVideoFrame()

        # Release the frame.
        self._frame = None
        self._enteredFrame = False
```

请注意，`enterFrame()`的实现仅抓取（同步）帧，而从通道的实际检索被推迟到`frame`变量的后续读取。 `exitFrame()`的实现从当前通道获取图像，估计帧速率，通过窗口管理器（如果有）显示图像，并完成对将图像写入文件的所有未决请求。

其他几种方法也与文件写入有关。 为了完成我们的类实现，让我们将其余的文件写入方法添加到`managers.py`中：

```
    def writeImage(self, filename):
        """Write the next exited frame to an
image file."""
        self._imageFilename = filename

    def startWritingVideo(
            self, filename,
            encoding =
cv2.cv.CV_FOURCC('I','4','2','0')):
        """Start writing exited frames to a video
file."""
        self._videoFilename = filename
        self._videoEncoding = encoding

    def stopWritingVideo (self):
```

```
        """Stop writing exited frames to a video
file."""
        self._videoFilename = None
        self._videoEncoding = None
        self._videoWriter = None

def _writeVideoFrame(self):

        if not self.isWritingVideo:
            return

        if self._videoWriter is None:
            fps =
self._capture.get(cv2.cv.CV_CAP_PROP_FPS)
            if fps == 0.0:
                # The capture's FPS is unknown so
use an estimate.
                if self._framesElapsed < 20:
                    # Wait until more frames
elapse so that the
                    # estimate is more stable.
                    return
                else:
                    fps = self._fpsEstimate
            size = (int(self._capture.get(

 cv2.cv.CV_CAP_PROP_FRAME_WIDTH)),
                    int(self._capture.get(

 cv2.cv.CV_CAP_PROP_FRAME_HEIGHT)))
            self._videoWriter = cv2.VideoWriter(
                self._videoFilename,
self._videoEncoding,
                fps, size)

        self._videoWriter.write(self._frame)
```

公用方法`writeImage()`， `startWritingVideo()`和`stopWritingVideo()`仅记录用于文件写入操作的参数，而实际的写入操作则推迟到`exitFrame()`的下一次调用。 非公开方法`_writeVideoFrame()`以我们先前的脚本应该熟悉的方式创建或附加到视频文件。 （请参阅“读/写视频文件”部分。）但是，在帧速率未知的情况下，我们会在捕获会话开始时跳过一些帧，以便有时间构建帧速率的估计。

尽管我们当前的`CaptureManager`实现依赖于`VideoCapture`，但我们可以进行其他不使用 OpenCV 进行输入的实现。 例如，我们可以创建一个用套接字连接实例化的子类，该子类的字节流可以解析为图像流。 另外，我们可以创建一个子类，该子类使用第三方相机库，其硬件支持与 OpenCV 提供的硬件支持不同。 但是，对于 Cameo，我们当前的实现方式就足够了。

抽象化窗口和键盘 – `manager.WindowManager`

如我们所见，OpenCV 提供的功能可导致创建窗口，破坏窗口，显示图像以及进程事件。 这些函数不是窗口类的方法，而是需要窗口的名称作为参数传递。 由于此接口不是面向对象的，因此与 OpenCV 的常规样式不一致。 而且，它不太可能与我们最终想要代替 OpenCV 使用的其他窗口或事件处理接口兼容。

出于面向对象和适应性的考虑，我们使用`createWindow()`， `destroyWindow()`， `show()`和`processEvents()`方法将此功能抽象为`WindowManager`类。 作为属性，`WindowManager`类具有一个称为`keypressCallback`的功能对象，该对象将响应任何按键从`processEvents()`调用（如果不是`None`）。 `keypressCallback`对象必须采用单个参数，即 ASCII 键代码。

让我们将以下`WindowManager`的实现添加到`managers.py`中：

```
class WindowManager(object):

    def __init__(self, windowName,
keypressCallback = None):
        self.keypressCallback = keypressCallback

        self._windowName = windowName
        self._isWindowCreated = False

    @property
    def isWindowCreated(self):
        return self._isWindowCreated

    def createWindow (self):
        cv2.namedWindow(self._windowName)
        self._isWindowCreated = True

    def show(self, frame):
        cv2.imshow(self._windowName, frame)

    def destroyWindow (self):
        cv2.destroyWindow(self._windowName)
        self._isWindowCreated = False

    def processEvents (self):
        keycode = cv2.waitKey(1)
        if self.keypressCallback is not None and
keycode != -1:
            # Discard any non-ASCII info encoded
by GTK.
            keycode &= 0xFF
            self.keypressCallback(keycode)
```

我们当前的实现仅支持键盘事件，这对于 Cameo 足够了。 但是，我们也可以修改`WindowManager`以支持鼠标事件。 例如，可以将类的接口扩展为包括`mouseCallback`属性（和可选的构造器参数），但否则可以保持不变。 使用 OpenCV 以外的其他事件框架，我们可以通过添加回调属性以相同的方式支持其他事件类型。

附录 A“与 Pygame 集成”，显示了`WindowManager`子类，该子类是通过 Pygame 的窗口处理和事件框架而不是 OpenCV 来实现的。 通过正确处理退出事件（例如，当用户单击窗口的关闭按钮时），此实现在`WindowManager`基类上得到了改进。 潜在地，许多其他事件类型也可以通过 Pygame 处理。

应用所有内容 – `cameo.Cameo`

我们的应用以`Cameo`类表示，具有两种方法：`run()`和`onKeypress()`。 初始化时， `Cameo`类创建一个带有`onKeypress()`作为回调的`WindowManager`类，并使用摄像机和`WindowManager`类创建一个`CaptureManager`类。 调用`run()`时，应用执行一个主循环，在该循环中处理帧和事件。 作为事件处理的结果，可以调用`onKeypress()`。 空格键将截取屏幕快照，`TAB`导致屏幕录像（视频录制）开始/停止，`Esc`导致应用退出。

在与`managers.py`相同的目录中，我们创建一个名为`cameo.py`的文件，其中包含`Cameo`的以下实现：

```
import cv2
from managers import WindowManager,
CaptureManager

class Cameo(object):

    def __init__(self):
```

```
        self._windowManager =
WindowManager('Cameo',

 self.onKeypress)
        self._captureManager = CaptureManager(
            cv2.VideoCapture(0),
self._windowManager, True)

    def run(self):
        """Run the main loop."""
        self._windowManager.createWindow()
        while
self._windowManager.isWindowCreated:
            self._captureManager.enterFrame()
            frame = self._captureManager.frame

            # TODO: Filter the frame (Chapter 3).

            self._captureManager.exitFrame()
            self._windowManager.processEvents()

    def onKeypress (self, keycode):
        """Handle a keypress.

        space  -> Take a screenshot.
        tab    -> Start/stop recording a
screencast.
        escape -> Quit.

        """
        if keycode == 32: # space

 self._captureManager.writeImage('screenshot.png'
)
        elif keycode == 9: # tab
            if not
```

```
self._captureManager.isWritingVideo:

 self._captureManager.startWritingVideo(
                     'screencast.avi')
            else:

 self._captureManager.stopWritingVideo()
        elif keycode == 27: # escape
            self._windowManager.destroyWindow()

if __name__=="__main__":
    Cameo().run()
```

运行该应用时，请注意，实时摄像机的提要是镜像的，而屏幕截图和截屏不是。 这是预期的行为，因为我们在初始化`CaptureManager`类时将`shouldMirrorPreview`传递给`True`。

到目前为止，除了镜像它们以进行预览之外，我们不会以其他任何方式操作它们。 我们将在第 3 章，“过滤图像”中开始添加更多有趣的效果。

总结

到现在为止，我们应该有一个应用，它可以显示摄像机的供稿，监听键盘输入并（按命令）记录屏幕截图或截屏视频。 我们准备通过在每个帧的开始和结尾之间插入一些图像过滤代码（第 3 章，“过滤图像”）来扩展应用。 （可选）我们还准备集成 Open GV 支持的其他相机驱动程序或其他应用框架（附录 A，“与 Pygame 集成”）。

三、过滤图像

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/3.md

本章介绍了一些更改图像的技术。 我们的目标是实现艺术效果，类似于在图像编辑应用（例如 Photoshop 或 Gimp）中可以找到的过滤器。

在实现过滤器时，您可以尝试将其应用于任何 BGR 图像，然后保存或显示结果。 要充分欣赏每种效果，请在各种照明条件和拍摄对象下进行尝试。 在本章的最后，我们将把过滤器集成到 Cameo 应用中。

注意

可以从我的网站下载本章的所有完成代码。

创建模块

与`CaptureManager`和`WindowManager`类一样，我们的过滤器应可在 Cameo 外部重用。 因此，我们应该将过滤器分成各自的 Python 模块或文件。

让我们在与`cameo.py`相同的目录中创建一个名为`filters.py`的文件。 我们需要在`filters.py`中使用以下`import`语句：

```
import cv2
import numpy
import utils
```

我们还要在同一目录中创建一个名为`utils.py`的文件。 它应包含以下导入语句：

```
import cv2
import numpy
import scipy.interpolate
```

我们将在`filters.py`中添加过滤器函数和类，而`utils.py`中将使用更多通用数学函数。

通道混合 – 在 Technicolor 中查看

通道混合是重新映射颜色的简单技术。 目标像素处的颜色是（仅）相应源像素处的颜色的函数。 更具体地说，在目标像素处的每个通道的值是在源像素处的任何或所有通道的值的函数。 对于 BGR 图片，使用伪代码：

```
dst.b = funcB(src.b, src.g, src.r)
dst.g = funcG(src.b, src.g, src.r)
dst.r = funcR(src.b, src.g, src.r)
```

我们可以定义这些函数，但是我们可以。 潜在地，我们可以映射场景的颜色与摄影机或眼睛通常所映射的颜色有很大不同。

通道混合的一种用途是模拟 RGB 或 BGR 内部的其他一些较小的色彩空间。 通过为任意两个通道分配相等的值，我们可以折叠部分颜色空间，并产生一种印象，即我们的调色板仅基于两种颜色的光（相加混合）或两种墨水（相减地混合）。 这种类型的效果可以提供怀旧的价值，因为早期的彩色胶片和早期的数字图形的调色板比当今的数字图形更为有限。

作为示例，让我们发明一些概念色彩空间，这些色彩空间让人想起 1920 年代的 Technicolor 电影和 1980 年代的 CGA 图形。 所有这些概念性色彩空间都可以代表灰色，但不能代表 RGB 的整个色彩范围：

- **RC**（红色，青色）：请注意，红色和青色可以混合成灰色。 该色彩空间类似于 Technicolor 流程 2 和 CGA 调色板 3。
- **RGV**（红色，绿色，值）：请注意，红色和绿色不能混合成灰色。 因此，我们还需要指定值或白度。 此色彩空间类似于 Technicolor 流程 1。
- **CMV**（青色，洋红色，值）：请注意，青色和洋红色不能混合以产生灰色。 因此，我们还需要指定值或白度。 此颜色空间类似于 CGA 调色板 1。

以下是《The Toll of the Sea》（1922）（以 Technicolor 流程 2 拍摄的电影）的屏幕截图：

以下图像来自《Commander Keen：再见银河》（1991），该游戏支持 CGA 调色板 1。（有关彩色图像，请参见本书的电子版。）：

模拟 RC 色彩空间

RC 色彩空间易于在 BGR 中进行模拟。 蓝色和绿色可以混合成青色。 通过平均 B 和 G 通道并将结果存储在 B 和 G 中，我们有效地将这两个通道合为一个 C。为了支持这种效果，让我们向`filters.py`添加以下函数：

```
def recolorRC(src, dst):
    """Simulate conversion from BGR to RC (red,
cyan).

    The source and destination images must both
be in BGR format.

    Blues and greens are replaced with cyans.

    Pseudocode:
    dst.b = dst.g = 0.5 * (src.b + src.g)
    dst.r = src.r

    """
```

```
    b, g, r = cv2.split(src)
    cv2.addWeighted(b, 0.5, g, 0.5, 0, b)
    cv2.merge((b, b, r), dst)
```

此函数发生了三件事：

1. 使用`split()`，我们将源图像的通道提取为一维数组。 以这种格式放置数据后，我们可以编写清晰，简单的通道混合代码。
2. 使用`addWeighted()`，我们用 B 和 G 的平均值替换 B 通道的值。`addWeighted()`的参数（按顺序）是第一个源数组，一个权重应用于第一个源数组，第二个源数组， 应用于第二个源数组的权重，添加到结果中的常量以及一个目标数组。
3. 使用`merge()`，我们用修改后的通道替换目标图像中的值。 请注意，我们两次使用`b`作为参数，因为我们希望目标的 B 和 G 通道相等。

分割，修改和合并通道的类似步骤也可以应用到我们的其他色彩空间模拟中。

模拟 RGV 色彩空间

在 BGR 中，RGV 色彩空间难以模拟。 我们的直觉可能会说我们应该将所有 B 通道值都设置为`0`，因为 RGV 不能代表蓝色。 但是，此更改将是错误的，因为它将丢弃亮度的蓝色分量，从而将灰色和浅蓝色变成黄色。 相反，我们希望灰色保持灰色，而浅蓝色变为灰色。 为了获得此结果，我们应将 B 值减小到 B，G 和 R 的每个像素最小值。让我们在`filters.py`中实现以下效果：

```
def recolorRGV(src, dst):
    """Simulate conversion from BGR to RGV (red,
green, value).
```

```
    The source and destination images must both
be in BGR format.

    Blues are desaturated.

    Pseudocode:
    dst.b = min(src.b, src.g, src.r)
    dst.g = src.g
    dst.r = src.r

    """
    b, g, r = cv2.split(src)
    cv2.min(b, g, b)
    cv2.min(b, r, b)
    cv2.merge((b, g, r), dst)
```

`min()`函数计算前两个参数的每个元素的最小值，并将它们写入第三个参数。

模拟 CMV 颜色空间

模拟 CMV 颜色空间与模拟 RGV 非常相似，除了光谱的去饱和部分是黄色而不是蓝色。 要使黄色去饱和，应将 B 值增加到 B，G 和 R 的每个像素最大值。以下是可以添加到`filters.py`的实现：

```
def recolorCMV(src, dst):
    """Simulate conversion from BGR to CMV (cyan,
magenta, value).

    The source and destination images must both
be in BGR format.

    Yellows are desaturated.

```

```
    Pseudocode:
    dst.b = max(src.b, src.g, src.r)
    dst.g = src.g
    dst.r = src.r

    """
    b, g, r = cv2.split(src)
    cv2.max(b, g, b)
    cv2.max(b, r, b)
    cv2.merge((b, g, r), dst)
```

`max()`函数计算前两个参数的每个元素的最大值，并将它们写入第三个参数。

通过设计，前面的三个效果往往会产生严重的颜色失真，尤其是当源图像首先是彩色的时。 如果我们想制作微妙的效果，则将通道使用任意函数混合可能不是最好的方法。

曲线 – 弯曲色彩空间

曲线是另一种用于重新映射颜色的技术。 通道混合和曲线类似，只要目标像素处的颜色是相应源像素处的颜色（仅）即可。 但是，具体而言，通道混合和曲线是不同的方法。 使用曲线时，目标像素处的通道值是（仅）源像素处的相同通道值的函数。 此外，我们不直接定义函数； 相反，对于每个函数，我们定义一组控制点，从中插入函数。
对于 BGR 图片，使用伪代码：

```
dst.b = funcB(src.b) where funcB interpolates
pointsB
dst.g = funcG(src.g) where funcG interpolates
pointsG
```

```
dst.r = funcR(src.r) where funcR interpolates
pointsR
```

插值的类型可能在实现之间有所不同，尽管插值类型应避免在控制点处出现不连续的斜率，而产生曲线。 只要控制点数量足够，我们将使用**三次样条插值**。

绘制曲线

迈向基于曲线的过滤器的第一步是将控制点转换为函数。 大部分工作都是通过名为`interp1d()`的 SciPy 函数完成的，该函数接受两个数组（`x`和`y`坐标）并返回一个对点进行插值的函数。 作为`interp1d()`的可选参数，我们可以指定一种插值，原则上可以是`linear`，`nearest`，`zero`，`slinear`（球面线性），`quadratic`或[`cubic`，尽管并非所有选项都在当前版本的 SciPy 中实现。 另一个可选参数`bounds_error`可以设置为`False`，以允许外插和内插。

让我们编辑`utils.py`并添加一个将`interp1d()`包裹起来的函数，它的接口稍微简单一些：

```
def createCurveFunc(points):
    """Return a function derived from control
points."""
    if points is None:
        return None
    numPoints = len(points)
    if numPoints < 2:
        return None
    xs, ys = zip(*points)
    if numPoints < 4:
        kind = 'linear'
        # 'quadratic' is not implemented.
    else:
```

```
        kind = 'cubic'
    return scipy.interpolate.interp1d(xs, ys,
kind,

 bounds_error = False)
```

我们的函数采用两个`(x, y)`对数组，而不是两个单独的坐标数组，这可能是指定控制点更容易理解的方式。 必须对数组进行排序，以使`x`从一个索引增加到下一个索引。 通常，为获得自然效果，`y`值也应增加，并且第一个和最后一个控制点应为`(0, 0)`和`(255, 255)`，以便保留黑色和白色。 请注意，我们会将`x`视为通道的输入值，并将`y`视为相应的输出值。 例如，`(128, 160)`可使通道的中间色调变亮。

请注意，`cubic`插值至少需要四个控制点。 如果只有两个或三个控制点，我们将退回到`linear`插值，但是，为获得自然外观效果，应避免这种情况。

缓存和应用曲线

现在我们可以获得插值任意控制点的曲线的函数 。 但是，此函数可能很昂贵。 我们不希望每个通道每个像素运行一次（例如，如果应用于`640 x 480`视频的三个通道，则每帧运行 921,600 次）。 幸运的是，我们通常只处理 256 个可能的输入值（每个通道 8 位），并且可以廉价地预先计算并存储许多输出值。 然后，我们的每通道每像素成本只是对缓存的输出值的查找。

让我们编辑`utils.py`并添加函数以为给定函数创建一个查找数组，并将该查找数组应用于另一个数组（例如，图像）：

```
def createLookupArray(func, length = 256):
    """Return a lookup for whole-number inputs to
```

```
a function.

    The lookup values are clamped to [0, length -
1].

    """
    if func is None:
        return None
    lookupArray = numpy.empty(length)
    i = 0
    while i < length:
        func_i = func(i)
        lookupArray[i] = min(max(0, func_i),
length - 1)
        i += 1
    return lookupArray

def applyLookupArray(lookupArray, src, dst):
    """Map a source to a destination using a
lookup."""
    if lookupArray is None:
        return
    dst[:] = lookupArray[src]
```

请注意，`createLookupArray()`中的方法仅限于整数输入值，因为该输入值用作数组的索引。 `applyLookupArray()`函数通过使用源数组的值作为查找数组的索引来工作。 Python 的切片符号（`[:]`）用于将查找到的值复制到目标数组中。

让我们考虑另一个优化。 如果我们总是想连续应用两个或更多曲线怎么办？ 执行多次查找效率低下，并且可能导致精度损失。 我们可以通过在创建查找数组之前将两个曲线函数组合为一个函数来避免此问

题。 让我们再次编辑`utils.py`并添加以下函数，该函数返回两个给定函数的组合：

```
def createCompositeFunc(func0, func1):
    """Return a composite of two functions."""
    if func0 is None:
        return func1
    if func1 is None:
        return func0
    return lambda x: func0(func1(x))
```

`createCompositeFunc()`中的方法仅限于每个都带有单个参数的输入函数。 参数必须是兼容类型。 请注意，使用 Python 的`lambda`关键字创建匿名函数。

这是最终的优化问题。 如果我们想对图像的所有通道应用相同的曲线怎么办？ 在这种情况下，拆分和合并通道是浪费的，因为我们不需要区分通道。 我们只需要`applyLookupArray()`使用的一维索引。 让我们编辑`utils.py`以添加一个函数，该函数将一维接口返回到预先存在的给定数组，该数组可能是多维的：

```
def createFlatView(array):
    """Return a 1D view of an array of any
dimensionality."""
    flatView = array.view()
    flatView.shape = array.size
    return flatView
```

返回类型为`numpy.view`，其接口与`numpy.array`几乎相同，但是`numpy.view`仅拥有对数据的引用，而不是副本。

`createFlatView()`中的方法适用于具有任意数量通道的图像。 因此，当我们希望所有通道都相同时，它可以抽象出灰度图像和彩色图像之间的差异。

设计面向对象的曲线过滤器

由于我们为每个曲线缓存了一个查找数组，因此基于曲线的过滤器具有与之关联的数据。 因此，它们需要是类，而不仅仅是函数。 让我们制作一对曲线过滤器类，以及可以应用任何函数而不仅仅是曲线函数的相应高级类：

- `VFuncFilter`：这是一个用函数实例化的类，以后可以使用`apply()`将其应用于图像。 该函数适用于灰度图像的 **V（值）通道**或彩色图像的所有通道。
- `VcurveFilter`：这是`VFuncFilter`的子类。 而不是使用函数实例化，而是使用一组控制点实例化，这些控制点在内部用于创建曲线函数。
- `BGRFuncFilter`：这是一个用最多四个函数实例化的类，以后可以使用`apply()`将其应用于 BGR 图像。 这些函数之一适用于所有通道，其他三个函数分别适用于单个通道。 首先应用整体函数，然后再应用每通道函数。
- `BGRCurveFilter`：这是`BGRFuncFilter`的子类。 而不是用四个函数实例化，而是用四组控制点实例化，这些控制点在内部用于创建曲线函数。

此外，所有这些类都接受数字类型的构造器参数，例如`numpy.uint8`，每个通道 8 位。 此类型用于确定查找数组中应包含多少个条目。

让我们首先看一下`VFuncFilter`和`VcurveFilter`的实现，它们都可以添加到`filters.py`中：

```
class VFuncFilter(object):
    """A filter that applies a function to V (or
all of BGR)."""

    def __init__(self, vFunc = None, dtype =
numpy.uint8):
        length = numpy.iinfo(dtype).max + 1
        self._vLookupArray =
utils.createLookupArray(vFunc, length)

    def apply(self, src, dst):
        """Apply the filter with a BGR or gray
source/destination."""
        srcFlatView = utils.flatView(src)
        dstFlatView = utils.flatView(dst)

 utils.applyLookupArray(self._vLookupArray,
srcFlatView,
                               dstFlatView)

class VCurveFilter(VFuncFilter):
    """A filter that applies a curve to V (or all
of BGR)."""

    def __init__(self, vPoints, dtype =
numpy.uint8):
        VFuncFilter.__init__(self,
utils.createCurveFunc(vPoints),
                             dtype)
```

在这里，我们正在内部使用一些以前的函数：`createCurveFunc()`，`createLookupArray()`，`flatView()`和`applyLookupArray()`。 我们还使用`numpy.iinfo()`根据给定的数字类型确定相关的查找值范围。

现在，让我们看一下`BGRFuncFilter`和`BGRCurveFilter`的实现，它们也都可以添加到`filters.py`中：

```
class BGRFuncFilter(object):
    """A filter that applies different functions
to each of BGR."""

    def __init__(self, vFunc = None, bFunc =
None, gFunc = None,
                 rFunc = None, dtype =
numpy.uint8):
        length = numpy.iinfo(dtype).max + 1
        self._bLookupArray =
utils.createLookupArray(
            utils.createCompositeFunc(bFunc,
vFunc), length)
        self._gLookupArray =
utils.createLookupArray(
            utils.createCompositeFunc(gFunc,
vFunc), length)
        self._rLookupArray =
utils.createLookupArray(
            utils.createCompositeFunc(rFunc,
vFunc), length)

    def apply(self, src, dst):
        """Apply the filter with a BGR
source/destination."""
        b, g, r = cv2.split(src)
```

```
 utils.applyLookupArray(self._bLookupArray, b, b)

 utils.applyLookupArray(self._gLookupArray, g, g)

 utils.applyLookupArray(self._rLookupArray, r, r)
        cv2.merge([b, g, r], dst)

class BGRCurveFilter(BGRFuncFilter):
    """A filter that applies different curves to
each of BGR."""

    def __init__(self, vPoints = None, bPoints =
None,
                 gPoints = None, rPoints = None,
dtype = numpy.uint8):
        BGRFuncFilter.__init__(self,

utils.createCurveFunc(vPoints),

utils.createCurveFunc(bPoints),

utils.createCurveFunc(gPoints),

utils.createCurveFunc(rPoints), dtype)
```

同样，我们正在内部使用一些以前的函数：`createCurveFunc()`，`createCompositeFunc()`，`createLookupArray()`和`applyLookupArray()`。 我们还使用`iinfo()`，`split()`和`merge()`。

这四个类可以按原样使用，在实例化时将自定义函数或控制点作为参数传递。 或者，我们可以创建其他子类，这些子类对某些函数或控制点进行硬编码。 这样的子类可以实例化而无需任何参数。

模拟摄影胶片

曲线的常用用法是模拟数字前摄影中常用的调色板。 每种类型的胶卷都有自己独特的颜色（或灰色）表示法，但我们可以概括一些与数字传感器的区别。 胶片容易遭受细节损失和阴影饱和度的困扰，而数字趋向于遭受高光部分的这些缺陷。 而且，薄膜在光谱的不同部分上趋于具有不均匀的饱和度。 因此，每部影片都有*弹出*或跳出的某些颜色。

因此，当我们想到漂亮的电影照片时，我们可能会想到明亮的场景并呈现出某些主导色彩。 在另一个极端，我们可能还记得曝光不足的胶片的暗淡外观，而实验室技术人员的努力无法改善这种模糊的外观。

我们将使用曲线创建四个不同的胶片状过滤器。 他们的灵感来自三种胶片和一种处理技术：

- 柯达 Portra，专为肖像和婚礼而设计的胶卷系列
- 通用电影系列富士 Provia
- 富士 Velvia，针对风景优化的电影系列
- 交叉处理，一种非标准的胶片处理技术，有时用于在时装和乐队摄影中产生低劣的外观

每个电影模拟效果都是`BGRCurveFilter`的非常简单的子类。 我们只是重写构造器为每个通道指定一组控制点。 控制点的选择基于摄影师 Petteri Sulonen 的建议。 在这个页面查看他关于胶片状曲线的文章。

Portra，Provia 和 Velvia 效果应产生*外观正常的*图像。 除了前后比较之外，效果应该不明显。

模拟柯达 Portra

Portra 具有较宽的高光范围，倾向于暖色（琥珀色），而阴影较冷（更蓝）。 作为人像电影，它倾向于使人们的肤色更白皙。 而且，它会夸大某些常见的衣服颜色，例如乳白色（例如婚纱）和深蓝色（例如西装或牛仔裤）。 让我们将 Portra 过滤器的此实现添加到`filters.py`：

```
class BGRPortraCurveFilter(BGRCurveFilter):
    """A filter that applies Portra-like curves
to BGR."""

    def __init__(self, dtype = numpy.uint8):
        BGRCurveFilter.__init__(
            self,
            vPoints = [(0,0),(23,20),(157,173),
(255,255)],
            bPoints = [(0,0),(41,46),(231,228),
(255,255)],
            gPoints = [(0,0),(52,47),(189,196),
(255,255)],
            rPoints = [(0,0),(69,69),(213,218),
(255,255)],
            dtype = dtype)
```

模拟富士 Provia

普罗维亚（Provia）具有强烈的对比度，并且在大多数色调中略微凉爽（蓝色）。 天空，水，和阴影比太阳增强更多。 让我们将 Provia 过滤器的实现添加到`filters.py`：

```
class BGRProviaCurveFilter(BGRCurveFilter):
    """A filter that applies Provia-like curves
to BGR."""

```

```
    def __init__(self, dtype = numpy.uint8):
        BGRCurveFilter.__init__(
            self,
            bPoints = [(0,0),(35,25),(205,227),
(255,255)],
            gPoints = [(0,0),(27,21),(196,207),
(255,255)],
            rPoints = [(0,0),(59,54),(202,210),
(255,255)],
            dtype = dtype)
```

模拟富士 Velvia

Velvia 具有深阴影和鲜艳的色彩。 它通常可以在白天产生蔚蓝的天空，在日落时产生深红色的云。 效果很难模拟，但是这是我们可以添加到`filters.py`的尝试：

```
class BGRVelviaCurveFilter(BGRCurveFilter):
    """A filter that applies Velvia-like curves
to BGR."""

    def __init__(self, dtype = numpy.uint8):
        BGRCurveFilter.__init__(
            self,
            vPoints = [(0,0),(128,118),(221,215),
(255,255)],
            bPoints = [(0,0),(25,21),(122,153),
(165,206),(255,255)],
            gPoints = [(0,0),(25,21),(95,102),
(181,208),(255,255)],
            rPoints = [(0,0),(41,28),(183,209),
(255,255)],
            dtype = dtype)
```

模拟交叉处理

交叉处理会在阴影中产生强烈的蓝色或绿蓝色调，在高光部分产生强烈的黄色或绿黄色。 黑色和白色不一定要保留。 另外，对比度非常高。 交叉处理的照片看起来很不舒服。 人们看起来黄疸，而无生命的物体看起来很脏。 让我们编辑`filters.py`以添加以下交叉处理过滤器的实现：

```
class BGRCrossProcessCurveFilter(BGRCurveFilter):
    """A filter that applies cross-process-like
curves to BGR."""

    def __init__(self, dtype = numpy.uint8):
        BGRCurveFilter.__init__(
            self,
            bPoints = [(0,20),(255,235)],
            gPoints = [(0,0),(56,39),(208,226),
(255,255)],
            rPoints = [(0,0),(56,22),(211,255),
(255,255)],
            dtype = dtype)
```

突出显示边缘

边缘在人类和计算机视觉中都扮演着重要角色。 我们作为人类，仅通过查看背光轮廓或粗略草图就可以轻松识别许多对象类型及其姿势。 确实，当艺术强调边缘和姿势时，它似乎常常传达出原型的想法，例如罗丹的《思想家》或乔·舒斯特的《超人》。 软件也可以推断出边缘，姿势和原型。 我们将在后面的章节中讨论这类推理。

目前，我们对简单使用边缘来达到艺术效果感兴趣。 我们将使用粗体黑线跟踪图像的边缘。 效果应让人联想到用毡笔绘制的漫画书或其他

插图。

OpenCV 提供了许多边缘过滤器，包括`Laplacian()`，`Sobel()`和`Scharr()`。 这些过滤器应该将非边缘区域变成黑色，而将边缘区域变成白色或饱和色。 但是，它们易于将噪声误识别为边缘。 可以通过在尝试查找边缘之前对图像进行模糊处理来缓解此缺陷。 OpenCV 还提供了许多模糊过滤器，包括`blur()`（简单平均值），`medianBlur()`和`GaussianBlur()`。 边缘查找和模糊过滤器的参数有所不同，但始终包含`ksize`，这是一个奇数，代表过滤器核的宽度和高度（以像素为单位）。

注意

核是一组权重，这些权重应用于源图像中的区域以在目标图像中生成单个像素。 例如，`7`的`ksize`表示在生成每个目的地像素时考虑了`49 (7 x 7)`源像素。 我们可以将核视为一块磨砂玻璃，它在源图像上方移动，并让源光散射扩散。

为了模糊，让我们使用`medianBlur()`，它可以有效消除数字视频噪声，尤其是在彩色图像中。 对于边缘查找，让我们使用`Laplacian()`，它会产生粗体的边缘线，尤其是在灰度图像中。 应用`medianBlur()`之后，但应用`Laplacian()`之前，我们应该从 BGR 转换为灰度。

获得`Laplacian()`的结果后，我们可以将其反转为白色背景上的黑色边缘。 然后，我们可以对其进行归一化（使其值的范围为 0 到 1），然后将其与源图像的相乘以使边缘变暗。 让我们在`filters.py`中实现这种方法：

```
def strokeEdges(src, dst, blurKsize = 7,
edgeKsize = 5):
    if blurKsize >= 3:
        blurredSrc = cv2.medianBlur(src,
blurKsize)
        graySrc = cv2.cvtColor(blurredSrc,
cv2.COLOR_BGR2GRAY)
    else:
        graySrc = cv2.cvtColor(src,
cv2.COLOR_BGR2GRAY)
    cv2.Laplacian(graySrc, cv2.cv.CV_8U, graySrc,
ksize = edgeKsize)
    normalizedInverseAlpha = (1.0 / 255) * (255 -
graySrc)
    channels = cv2.split(src)
    for channel in channels:
        channel[:] = channel *
normalizedInverseAlpha
    cv2.merge(channels, dst)
```

请注意，我们允许将核大小指定为`strokeEdges()`的参数。 `blurKsize`自变量用作`medianBlur()`的`ksize`，而`edgeKsize`用作`Laplacian()`的`ksize`。使用网络摄像头，我发现`7`的`blurKsize`值和`5`的`edgeKsize`值看起来最好。 不幸的是，`medianBlur()`与像`7`这样的大型`ksize`一样昂贵。 如果在运行`strokeEdges()`时遇到性能问题，请尝试减小`blurKsize`的值。 要关闭模糊，请将其设置为小于`3`的值。

自定义核 – 令人费解

如我们所见，OpenCV 的许多预定义过滤器使用核。 请记住，核是一组权重，这些权重确定如何根据输入像素的邻域计算每个输出像素。

核的另一个术语是**卷积矩阵**。 它*混合*或卷积区域中的像素。 类似地，基于核的过滤器可以称为卷积过滤器。

OpenCV 提供了非常通用的函数`filter2D()`和，该函数应用了我们指定的任何核或卷积矩阵。 要了解如何使用此函数，让我们首先学习卷积矩阵的格式。 它是一个二维数组，具有奇数行和列。 中心元素对应于关注像素，其他元素对应于该像素的邻居。 每个元素都包含一个整数或浮点值，该值是应用于输入像素值的权重。 考虑以下示例：

```
kernel = numpy.array([[-1, -1, -1],
                      [-1,  9, -1],
                      [-1, -1, -1]])
```

在此，关注像素的权重为`9`，其相邻像素的权重为`-1`。 对于感兴趣的像素，输出颜色将是其输入颜色的九倍，减去所有八个相邻像素的输入颜色。 如果感兴趣的像素与其相邻像素已经有点不同，，这种差异就会加剧。 效果是，随着邻居之间的对比度增加，图像看起来*更加清晰*。

继续我们的示例，我们可以将此卷积矩阵应用于源图像和目标图像，如下所示：

```
cv2.filter2D(src, -1, kernel, dst)
```

第二个参数指定目标图像的每通道深度（例如，每通道 8 位的`cv2.CV_8U`）。 负值（此处使用）表示目标图像的深度与源图像的深度相同。

注意

对于彩色图像，请注意`filter2D()`将核均等地应用于每个通道。 要在不同的通道上使用不同的核，我们还必须使用`split()`和`merge()`函数，就像我们在较早的通道混合函数中所做的那样。 （请参阅“模拟 RC 色彩空间”部分。）

基于这个简单的示例，让我们向`filters.py`添加两个类。 一类`VConvolutionFilter`通常代表卷积过滤器。 子类`SharpenFilter`将专门代表我们的锐化过滤器。 让我们编辑`filters.py`来实现这两个新类，如下所示：

```
class VConvolutionFilter(object):
    """A filter that applies a convolution to V
(or all of BGR)."""

    def __init__(self, kernel):
        self._kernel = kernel

    def apply(self, src, dst):
        """Apply the filter with a BGR or gray
source/destination."""
        cv2.filter2D(src, -1, self._kernel, dst)

class SharpenFilter(VConvolutionFilter):
    """A sharpen filter with a 1-pixel radius."""

    def __init__(self):
        kernel = numpy.array([[-1, -1, -1],
                              [-1,  9, -1],
                              [-1, -1, -1]])
        VConvolutionFilter.__init__(self, kernel)
```

该模式与`VCurveFilter`类及其子类非常相似。 （请参阅“设计面向对象的曲线过滤器”部分。）

注意，权重总和为`1`。 每当我们要保持图像的整体亮度不变时，都应该是这种情况。 如果我们稍微修改锐化核，使其权重总和为`0`，则我们有一个边缘检测核，该边缘会将边缘变为白色，将非边缘变为黑色。 例如，让我们在`filters.py`中添加以下边缘检测过滤器：

```
class FindEdgesFilter(VConvolutionFilter):
    """An edge-finding filter with a 1-pixel
radius."""

    def __init__(self):
        kernel = numpy.array([[-1, -1, -1],
                              [-1,  8, -1],
                              [-1, -1, -1]])
        VConvolutionFilter.__init__(self, kernel)
```

接下来，让我们做一个模糊过滤器。 通常，对于模糊效果，权重之和应为`1`，并且在整个邻域中应为正。 例如，我们可以对邻域取一个简单的平均值，如下所示：

```
class BlurFilter(VConvolutionFilter):
    """A blur filter with a 2-pixel radius."""

    def __init__(self):
        kernel = numpy.array([[0.04, 0.04, 0.04,
0.04, 0.04],
                              [0.04, 0.04, 0.04,
0.04, 0.04],
                              [0.04, 0.04, 0.04,
0.04, 0.04],
                              [0.04, 0.04, 0.04,
```

```
0.04, 0.04],
                                [0.04, 0.04, 0.04,
0.04, 0.04]])
        VConvolutionFilter.__init__(self, kernel)
```

我们的锐化，边缘检测和模糊过滤器使用高度对称的核。 但是有时，对称性较低的核会产生有趣的效果。 让我们考虑一个在一侧模糊（权重为正）而在另一侧锐化（权重为负）的核。 它将产生凸纹或*浮雕*效果。 这是我们可以添加到`filters.py`的实现：

```
class EmbossFilter(VConvolutionFilter):
    """An emboss filter with a 1-pixel radius."""

    def __init__(self):
        kernel = numpy.array([[-2, -1, 0],
                              [-1,  1, 1],
                              [ 0,  1, 2]])
        VConvolutionFilter.__init__(self, kernel)
```

这套自定义卷积过滤器非常基础。 实际上，它比 OpenCV 的现成的过滤器集更基础。 但是，通过一些试验，您应该能够编写自己的核，从而产生独特的外观。

修改应用

现在，我们已经为几个过滤器提供了高级函数和类，将这些过滤器中的任何一个应用到 Cameo 中捕获的帧上都是微不足道的。 让我们编辑`cameo.py`，并在以下摘录中添加以粗体显示的行：

```
import cv2
import filters
from managers import WindowManager,
CaptureManager
```

```
class Cameo(object):

    def __init__(self):
        self._windowManager =
WindowManager('Cameo',

 self.onKeypress)
        self._captureManager = CaptureManager(
            cv2.VideoCapture(0),
self._windowManager, True)
        self._curveFilter =
filters.BGRPortraCurveFilter()

    def run(self):
        """Run the main loop."""
        self._windowManager.createWindow()
        while
self._windowManager.isWindowCreated:
            self._captureManager.enterFrame()
            frame = self._captureManager.frame

            # TODO: Track faces (Chapter 3).

            filters.strokeEdges(frame, frame)
            self._curveFilter.apply(frame, frame)

            self._captureManager.exitFrame()
            self._windowManager.processEvents()

    # ... The rest is the same as in Chapter 2.
```

在这里，我选择应用两种效果：抚摸边缘和模拟 Portra 胶片的颜色。随时修改代码以应用您喜欢的任何过滤器。

这是来自 Cameo 的屏幕截图，带有描边和类似 Portra 的颜色：

总结

在这一点上，我们应该有一个显示过滤后的相机提要的应用。 我们还应该有更多的过滤器实现，可以很容易地与我们当前使用的实现互换。 现在，我们准备分析每个框架，以便在下一章中找到要操纵的面孔。

四、使用 Haar 级联跟踪人脸

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/4.md

本章介绍一些 OpenCV 的跟踪功能，以及定义特定类型的可跟踪对象的数据文件。 具体来说，我们看一下 Haar 级联分类器，它们可以分析相邻图像区域之间的对比度，以确定给定图像或子图像是否与已知类型匹配。 我们考虑如何在层次结构中组合多个 Haar 级联分类器，以使一个分类器标识父区域（出于我们的目的，是面部），而其他分类器标识子区域（眼睛，鼻子和嘴巴）。

我们还绕过了谦虚但重要的矩形主题。 通过绘制，复制和调整矩形图像区域的大小，我们可以对正在跟踪的图像区域执行简单的操作。

在本章的最后，我们将把面部跟踪和矩形操作集成到 Cameo 中。 最后，我们将进行一些面对面的互动！

注意

可以从我的网站下载本章的所有完成代码。

概念化 Haar 级联

当我们谈论对对象进行分类并跟踪它们的位置时，我们究竟要精确指出什么？ 什么构成对象的可识别部分？

即使来自网络摄像头的摄影图像也可能包含许多细节，以使我们（人类）观看愉悦。 然而，关于照明，视角，观看距离，相机抖动和数字

噪声的变化，图像细节趋于不稳定。 此外，就分类而言，即使是物理细节上的实际差异也可能使我们不感兴趣。 我在学校里被教过，在显微镜下没有两个雪花看起来像。 幸运的是，作为一个加拿大孩子，我已经学会了如何在没有显微镜的情况下识别雪花，因为相似之处在体积上更加明显。

因此，一些抽象图像细节的方法可用于产生稳定的分类和跟踪结果。 抽象被称为**特征**，据说是从图像数据中提取的。 特征应该比像素少得多，尽管任何像素都可能影响多个特征。 可以基于图像相应特征之间的距离来评估两个图像之间的相似度。 例如，可以根据空间坐标或颜色坐标来定义距离。 类似 Haar 的特征是通常应用于实时面部跟踪的一种特征。 它们是 2001 年由 Paul Viola 和 Michael Jones 首次用于此目的的。每个类似 Haar 的特征都描述了相邻图像区域之间的对比度模式。 例如，边缘，顶点和细线均会产生鲜明的特征。 对于任何给定的图像，特征可以根据区域的大小而变化，这可以称为**窗口大小**。 尽管窗口大小不同，但只有两个比例不同的两个图像应该能够产生相似的特征。 因此，生成多个窗口大小的特征很有用。 这样的特征集合称为**级联**。 我们可以说 Haar 级联是尺度不变的，换句话说，它对尺度变化具有鲁棒性。 OpenCV 为规模不变的 Haar 级联提供了分类器和跟踪器，它期望采用特定的文件格式。 如在 OpenCV 中实现的 Haar 级联，对于旋转的变化并不稳健。 例如，上下颠倒的脸部不被视为与直立的脸部相似，并且侧面观察的脸部不被视为与从正面观看的脸部相似。 考虑到图像的多种转换以及多种窗口大小，更复杂，更耗费资源的实现可以提高 Haar 级联的旋转鲁棒性。 但是，我们将局限于 OpenCV 中的实现。

获取 Haar 级联数据

作为 OpenCV 设置的一部分，可能有一个名为`haarcascades`的目录。 它包含使用 OpenCV 随附的工具针对特定主题进行训练的级联。 目录的完整路径取决于您的系统和设置 OpenCV 的方法，如下所示：

- **从源归档文件构建：**`<unzip_destination>/data/haarcascades`
- **带有自解压 ZIP 的 Windows：**`<unzip_destination>/data/haarcascades`
- **带有 MacPorts 的 Mac：**`/opt/local/share/OpenCV/haarcascades`
- **带有 Homebrew 的 Mac：**不包括`haarcascades`文件；请参见`haarcascades`文件。 要获取它，请下载源存档
- **带有 APT 或软件中心的 Ubuntu：**不包括`haarcascades`文件； 要获取它，请下载源归档文件

提示

如果找不到`haarcascades`，请从这个页面 下载源归档文件（或从这个页面下载 Windows 自解压 ZIP） ，将其解压缩，然后寻找`<unzip_destination>/data/haarcascades`。

找到`haarcascades`后，在与`cameo.py`相同的文件夹中创建一个名为`cascades`的目录，并将以下文件从`haarcascades`复制到`cascades`：

```
haarcascade_frontalface_alt.xml
haarcascade_eye.xml
haarcascade_mcs_nose.xml
haarcascade_mcs_mouth.xml
```

顾名思义，这些级联用于跟踪脸部，眼睛，鼻子和嘴巴。 他们需要正面，正面查看主题。 稍后在构建高级跟踪器时将使用它们。 如果您

对如何生成这些数据集感到好奇，请参考附录 B，“为自定义目标生成 Haar 级联”。 有了足够的耐心和功能强大的计算机，您可以制作自己的瀑布，并针对各种类型的物体进行训练。

创建模块

我们应该继续保持专用代码与可重用代码之间的良好分离。 让我们为跟踪类及其助手创建新的模块。

应该在与`cameo.py`相同的目录中（并且等效地在`cascades`的父目录中）创建一个名为`trackers.py`的文件。 让我们将以下`import`语句放在`trackers.py`的开头：

```
import cv2
import rects
import utils
```

除了`trackers.py`和`cameo.py`之外，让我们制作另一个名为`rects.py`的文件，其中包含以下导入语句：

```
import cv2
```

我们的脸部追踪器和一张脸部定义将出现在`trackers.py`中，而各种帮助者将出现在`rects.py`和我们先前存在的`utils.py`文件中。

将人脸定义为矩形层次结构

在开始实现高级跟踪器之前，我们应该定义我们想要获得的跟踪结果的类型。 对于许多应用而言，估计对象在真实 3D 空间中的摆放方式非常重要。 但是，我们的应用是关于图像处理的。 因此，我们更关心 2D 图像空间。 人脸的正立正面图应占据图像中的大致矩形区域。

在这样的区域内，眼睛，鼻子和嘴巴应该占据粗糙的矩形子区域。 让我们打开`trackers.py`并添加一个包含相关数据的类：

```
class Face(object):
    """Data on facial features: face, eyes, nose,
mouth."""

    def __init__(self):
        self.faceRect = None
        self.leftEyeRect = None
        self.rightEyeRect = None
        self.noseRect = None
        self.mouthRect = None
```

注意

每当我们的代码包含作为属性或函数参数的矩形时，我们都将假定其为`(x, y, w, h)`格式，单位为像素，左上角为`(x, y)`，右下角为[`(x+w, y+h)`。 OpenCV 有时使用兼容的表示形式，但并非总是如此。 因此，在向 OpenCV 发送矩形或从 OpenCV 接收矩形时，我们必须小心。 例如，有时 OpenCV 需要左上角和右下角作为坐标对。

跟踪，剪切和粘贴矩形

当我上小学时，我的手艺很差。 我经常不得不将未完成的手工艺品项目带回家，母亲自愿在那里为我完成这些项目，以便我可以花更多的时间在计算机上。 如果不考虑那些日子，我将永远不会剪切并粘贴一张纸或一个字节数组。

与工艺一样，如果我们首先绘制轮廓，则更容易发现图形程序中的错误。 出于调试目的，Cameo 将包括一个选项，可在`Face`表示的任何矩形周围绘制线条。 OpenCV 提供了用于绘图的`rectangle()`函数。 但是，其参数表示的矩形不同于`Face`。 为了方便起见，让我们在`rects.py`中添加以下`rectangle()`包装器：

```
def outlineRect(image, rect, color):
    if rect is None:
        return
    x, y, w, h = rect
    cv2.rectangle(image, (x, y), (x+w, y+h),
color)
```

在此，`color`通常应为 BGR 三元组（值的范围为 0 到 255）或灰度值（范围为 0 到 255），具体取决于图像的格式。

接下来，Cameo 必须支持将一个矩形的内容复制到另一个矩形。 我们可以使用 Python 的切片符号在图像内读取或写入矩形。 记住图像的第一个索引是`y`坐标或行，我们可以将矩形指定为`image[y:y+h, x:x+w]`。 对于复制，如果矩形的源和目标的大小不同，则会出现复杂问题。 当然，我们希望两个面孔的大小不同，因此我们必须解决这种情况。 OpenCV 提供`resize()`函数，使我们可以指定目标大小和插值方法。 结合切片和调整大小，我们可以将以下复制函数实现添加到`rects.py`中：

```
def copyRect(src, dst, srcRect, dstRect,
             interpolation = cv2.INTER_LINEAR):
    """Copy part of the source to part of the
destination."""

    x0, y0, w0, h0 = srcRect
```

```
    x1, y1, w1, h1 = dstRect

    # Resize the contents of the source sub-
rectangle.
    # Put the result in the destination sub-
rectangle.
    dst[y1:y1+h1, x1:x1+w1] = \
        cv2.resize(src[y0:y0+h0, x0:x0+w0], (w1,
h1),
                   interpolation = interpolation)
```

OpenCV 支持以下插值选项：

- `cv2.INTER_NEAREST`：这是最近邻插值，价格便宜，但会产生块状结果
- `cv2.INTER_LINEAR`：这是双线性插值（默认值），在实时应用的成本和质量之间提供了很好的折衷
- `cv2.INTER_AREA`：这是像素面积关系，在缩小时可能会在成本和质量之间提供更好的折衷，但在放大时会产生块状结果
- `cv2.INTER_CUBIC`：这是在`4 x 4`像素邻域上的双三次插值，是一种高成本，高质量的方法
- `cv2.INTER_LANCZOS4`：这是在`8 x 8`像素邻域上进行 Lanczos 插值的方法，是成本最高，质量最高的方法

如果我们要支持交换两个或更多矩形内容，则复制变得更加复杂。 考虑以下方法，这是错误的：

```
copyRect(image, image, rect0, rect1) # overwrite
rect1
copyRect(image, image, rect1, rect0) # copy from
rect1
```

```
# Oops! rect1 was already overwritten by the time
we copied from it!
```

相反，我们需要在覆盖任何内容之前将其中一个矩形复制到一个临时数组。 让我们编辑`rects.py`以添加以下函数，该函数交换单个源图像中两个或更多矩形的内容：

```
def swapRects(src, dst, rects,
              interpolation = cv2.INTER_LINEAR):
    """Copy the source with two or more sub-
rectangles swapped."""

    if dst is not src:
        dst[:] = src

    numRects = len(rects)
    if numRects < 2:
        return

    # Copy the contents of the last rectangle
into temporary storage.
    x, y, w, h = rects[numRects - 1]
    temp = src[y:y+h, x:x+w].copy()

    # Copy the contents of each rectangle into
the next.
    i = numRects - 2
    while i >= 0:
        copyRect(src, dst, rects[i], rects[i+1],
interpolation)
        i -= 1

    # Copy the temporarily stored content into
the first rectangle.

```

```
    copyRect(temp, dst, (0, 0, w, h), rects[0],
interpolation)
```

交换是圆形的，因此它可以支持任意数量的矩形。 每个矩形的内容都以下一个矩形为准，但最后一个矩形的内容以第一个矩形为准。

对于 Cameo，这种方法应该可以很好地为我们服务，但是仍然不是万无一失。 直觉可能告诉我们，以下代码应保持`image`不变：

```
swapRects(image, image, rect0, rect1)
swapRects(image, image, rect1, rect0)
```

但是，如果`rect0`和`rect1`重叠，则我们的直觉可能不正确。 如果看到奇怪的结果，请研究交换重叠矩形的可能性。

添加更多工具函数

上一章，我们为一些辅助函数创建了一个名为`utils`的模块。 几个额外的帮助程序函数将使我们更容易编写跟踪器。

首先，了解图像是灰度还是彩色可能很有用。 我们可以根据图像的维数来判断。 彩色图像是 3D 数组，而灰度图像的尺寸较小。 让我们在`utils.py`中添加以下函数以测试图像是否为灰度级：

```
def isGray(image):
    """Return True if the image has one channel
per pixel."""
    return image.ndim < 3
```

其次，了解图像的尺寸并将这些尺寸除以给定的因子可能会很有用。 图像（或其他数组）的高度和宽度分别是其`shape`属性中的前两个条

目。 让我们在`utils.py`中添加以下函数，以获取图像的尺寸除以一个值：

```
def widthHeightDividedBy(image, divisor):
    """Return an image's dimensions, divided by a
value."""
    h, w = image.shape[:2]
    return (w/divisor, h/divisor)
```

现在，让我们回到本章的主要主题跟踪上。

追踪人脸

使用 OpenCV 的 Haar 级联分类器的挑战不仅在于获得跟踪结果，而且还在于获得跟踪结果。 它以高帧频获得一系列有意义的跟踪结果。 我们可以强制执行的一种常识是，某些被跟踪的对象应该具有层次关系，其中一个相对于另一个。 例如，鼻子应该在脸中间。 通过尝试跟踪整个面部和面部的一部分，我们可以使应用代码能够执行更详细的操作并检查给定的跟踪结果有多好。 有鼻子的脸比没有脸的脸更好。 同时，我们可以支持某些优化，例如仅在某些位置查找具有特定大小的脸孔和鼻子。

我们将在名为`FaceTracker`的类中实现优化的分层跟踪器，该类提供了简单的接口。 `FaceTracker`可以使用与跟踪精度和性能之间的权衡有关的某些可选配置参数进行初始化。 在任何给定时间，`FaceTracker`的最新跟踪结果都存储在称为`faces`的属性中，该属性是`Face`实例的列表。 最初，此列表为空。 它通过`update()`方法刷新，该方法接受图像供跟踪器分析。 最后，出于调试目的，可以通过`drawDebugRects()`方法绘制`faces`的矩形，该方法接受图像作为绘制表面。 每帧，实时面部跟

踪应用都会调用`update()`，读取`faces`，甚至可能调用`drawDebugRects()`。

在内部，`FaceTracker`使用称为`CascadeClassifier`的 OpenCV 类。`CascadeClassifier`使用级联数据文件初始化，例如我们之前找到并复制的文件。 出于我们的目的，`CascadeClassifier`的重要方法是`detectMultiScale()`，它执行的跟踪可能对规模变化具有鲁棒性。`detectMultiScale()`的可能参数为：

- `image`：这是要分析的图像。 每个通道必须具有 8 位。
- `scaleFactor`：此缩放因子在两个连续的遍中将窗口大小分开。 较高的值可提高性能，但会降低比例变化的鲁棒性。
- `minNeighbors`：此值比比赛中所需的最小区域数小 1。 （一个匹配项可能会合并多个相邻区域。）
- `flags`：有几个标志，但并非所有组合都有效。 有效的独立标志和有效组合包括：
  - `cv2.cv.CV_HAAR_SCALE_IMAGE`：缩放每个窗口图像区域以匹配特征数据。 （默认方法是相反的：缩放特征数据以匹配窗口。）缩放图像可以对现代硬件进行某些优化。 该标志不得与其他标志结合使用。
  - `cv2.cv.CV_HAAR_DO_CANNY_PRUNING`：急切拒绝包含太多或太少边缘以致无法匹配对象类型的区域。 此标志不应与`cv2.cv.CV_HAAR_FIND_BIGGEST_OBJECT`结合使用。
  - `cv2.cv.CV_HAAR_FIND_BIGGEST_OBJECT`：最多接受一场比赛（最大一场）。
  - `cv2.cv.CV_HAAR_FIND_BIGGEST_OBJECT` | `cv2.cv.HAAR_DO_ROUGH_SEARCH`：最多最多接受一场比赛（最大一

场比赛），并跳过一些细化（缩小）这场比赛区域的步骤。 `minNeighbors`参数应大于`0`。

- `minSize`：代表要寻找的最小物体尺寸的一对像素尺寸。 较高的值可提高性能。
- `maxSize`：一对像素尺寸，代表要寻找的最大物体尺寸。 较低的值可以提高性能。

`detectMultiScale()`的返回值是一个匹配项列表，每个匹配项均以`[x, y, w, h]`格式表示为矩形。

同样，`FaceTracker`的初始化器接受`scaleFactor`，`minNeighbors`和`flags`作为参数。 给定的值将传递给`FaceTracker`内部进行的所有`detectMultiScale()`调用。 同样在初始化期间，`FaceTracker`使用脸部，眼睛，鼻子和嘴巴数据创建`CascadeClassifiers`。 让我们将初始化器的以下实现和`faces`属性添加到`trackers.py`中：

```
class FaceTracker(object):
    """A tracker for facial features: face, eyes,
nose, mouth."""

    def __init__(self, scaleFactor = 1.2,
minNeighbors = 2,
                 flags =
cv2.cv.CV_HAAR_SCALE_IMAGE):

        self.scaleFactor = scaleFactor
        self.minNeighbors = minNeighbors
        self.flags = flags

        self._faces = []

        self._faceClassifier =
```

```
cv2.CascadeClassifier(

 'cascades/haarcascade_frontalface_alt.xml')
        self._eyeClassifier =
cv2.CascadeClassifier(
            'cascades/haarcascade_eye.xml')
        self._noseClassifier =
cv2.CascadeClassifier(
            'cascades/haarcascade_mcs_nose.xml')
        self._mouthClassifier =
cv2.CascadeClassifier(
            'cascades/haarcascade_mcs_mouth.xml')

    @property
    def faces(self):
        """The tracked facial features."""
        return self._faces
```

`FaceTracker`的`update()`方法首先创建给定图像的均等灰度变体。 如 OpenCV 的`equalizeHist()`函数中所实现的均衡，可以标准化图像的亮度并增加其对比度。 均衡作为预处理步骤，使我们的跟踪器对光照变化更加鲁棒，而转换为灰度可提高性能。 接下来，我们将预处理后的图像输入到我们的面部分类器中。 对于每个匹配的矩形，我们在某些子区域中搜索左眼和右眼，鼻子和嘴巴。 最终，匹配的矩形和子矩形存储在`faces`中的`Face`实例中。 对于每种类型的跟踪，我们指定与图像大小成比例的最小对象大小。 我们对`FaceTracker`的实现应继续使用`update()`的以下代码：

```
    def update(self, image):
        """Update the tracked facial features."""

        self._faces = []
```

```
        if utils.isGray(image):
            image = cv2.equalizeHist(image)
        else:
            image = cv2.cvtColor(image,
cv2.cv.CV_BGR2GRAY)
            cv2.equalizeHist(image, image)

        minSize =
utils.widthHeightDividedBy(image, 8)

        faceRects =
self._faceClassifier.detectMultiScale(
            image, self.scaleFactor,
self.minNeighbors, self.flags,
            minSize)

        if faceRects is not None:
            for faceRect in faceRects:

                face = Face()
                face.faceRect = faceRect

                x, y, w, h = faceRect

                # Seek an eye in the upper-left
part of the face.
                searchRect = (x+w/7, y, w*2/7,
h/2)
                face.leftEyeRect =
self._detectOneObject(
                    self._eyeClassifier, image,
searchRect, 64)

                # Seek an eye in the upper-right
part of the face.
                searchRect = (x+w*4/7, y, w*2/7,
```

```
h/2)
                face.rightEyeRect =
self._detectOneObject(
                    self._eyeClassifier, image,
searchRect, 64)

                # Seek a nose in the middle part
of the face.
                searchRect = (x+w/4, y+h/4, w/2,
h/2)
                face.noseRect =
self._detectOneObject(
                    self._noseClassifier, image,
searchRect, 32)

                # Seek a mouth in the lower-
middle part of the face.
                searchRect = (x+w/6, y+h*2/3,
w*2/3, h/3)
                face.mouthRect =
self._detectOneObject(
                    self._mouthClassifier, image,
searchRect, 16)

                self._faces.append(face)
```

注意，`update()`依赖于本章前面的`utils.isGray()`和`utils.widthHeightDividedBy()`。 此外，它还依赖于私有助手方法`_detectOneObject()`，该方法被多次调用，以处理跟踪面部多个子部分的重复工作。 作为参数，`_detectOneObject()`需要分类器，图像，矩形和最小对象大小。 矩形是给定分类器应搜索的图像子区域。 例如，鼻子分类器应搜索面部中间。 限制搜索范围可以提高性能，并有助于消除误报。 在内部，`_detectOneObject()`通过在图像的切片上运行

分类器并返回第一个匹配项（如果没有匹配项，则返回`None`）来工作。 无论我们是否使用`cv2.cv.CV_HAAR_FIND_BIGGEST_OBJECT`标志，此方法都有效。 我们对`FaceTracker`的实现应继续使用`_detectOneObject()`的以下代码：

```
    def _detectOneObject(self, classifier, image,
rect,

 imageSizeToMinSizeRatio):

        x, y, w, h = rect

        minSize = utils.widthHeightDividedBy(
            image, imageSizeToMinSizeRatio)

        subImage = image[y:y+h, x:x+w]

        subRects = classifier.detectMultiScale(
            subImage, self.scaleFactor,
self.minNeighbors,
            self.flags, minSize)

        if len(subRects) == 0:
            return None

        subX, subY, subW, subH = subRects[0]
        return (x+subX, y+subY, subW, subH)
```

最后，`FaceTracker`应该提供基本的绘图功能，以便可以显示其跟踪结果以进行调试。 以下方法实现使用我们的`rects.outlineRect()`函数简单地定义颜色，遍历`Face`实例并绘制每个`Face`的矩形到给定图像：

```
def drawDebugRects(self, image):
        """Draw rectangles around the tracked
facial features."""

        if utils.isGray(image):
            faceColor = 255
            leftEyeColor = 255
            rightEyeColor = 255
            noseColor = 255
            mouthColor = 255
        else:
            faceColor = (255, 255, 255) # white
            leftEyeColor = (0, 0, 255) # red
            rightEyeColor = (0, 255, 255) #
yellow
            noseColor = (0, 255, 0) # green
            mouthColor = (255, 0, 0) # blue

        for face in self.faces:
            rects.outlineRect(image,
face.faceRect, faceColor)
            rects.outlineRect(image,
face.leftEyeRect, leftEyeColor)
            rects.outlineRect(image,
face.rightEyeRect,
                              rightEyeColor)
            rects.outlineRect(image,
face.noseRect, noseColor)
            rects.outlineRect(image,
face.mouthRect, mouthColor)
```

现在，我们有了一个高级跟踪器，该跟踪器隐藏了 Haar 级联分类器的详细信息，同时允许应用代码提供新图像，获取有关跟踪结果的数据并要求进行调试绘图。

修改应用

让我们看一下将面部跟踪和交换集成到 Cameo 中的两种方法。 第一种方法使用单个相机提要，并交换在此相机提要中找到的脸部矩形。 第二种方法使用两个摄像头提要，并将面矩形从一个摄像头提要复制到另一个摄像头。

现在，我们将限制自己只处理整个面孔，而不要处理诸如眼睛之类的子元素。 但是，例如，您可以修改代码以仅交换眼睛。 如果尝试这样做，请小心检查脸部的相关子矩形是否不是`None`。

在一个相机源中交换人脸

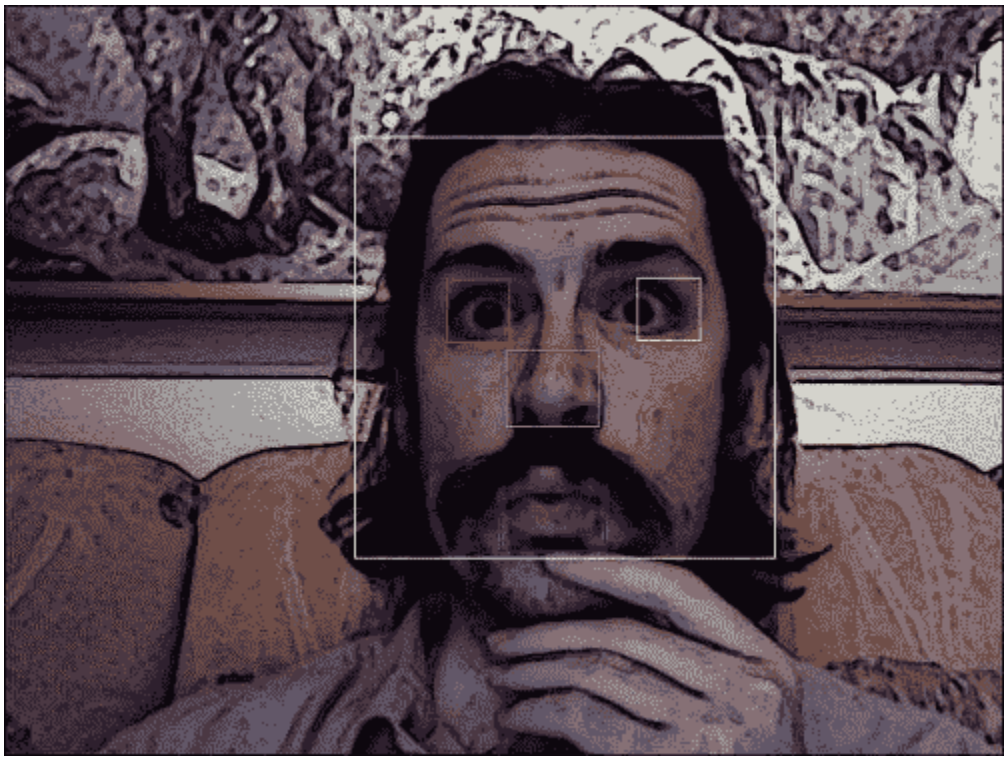
对于单相机版本，修改非常简单。 在初始化`Cameo`时，我们创建一个`FaceTracker`和一个布尔变量，指示是否应为`FaceTracker`绘制调试矩形。 响应`X`键，在`onKeypress()`中切换布尔值。 作为`run()`主循环的一部分，我们用当前帧更新`FaceTracker`。 然后，获取得到的`FaceFace`对象（在`faces`属性中），并使用`rects.swapRects()`交换它们的`faceRects`。 另外，根据布尔值，我们可能会绘制调试矩形，以反映任何交换之前面部元素的原始位置。

```
import cv2
import filters
from managers import WindowManager,
CaptureManager
import rects
from trackers import FaceTracker

class Cameo(object):

    def __init__(self):
        self._windowManager =
```

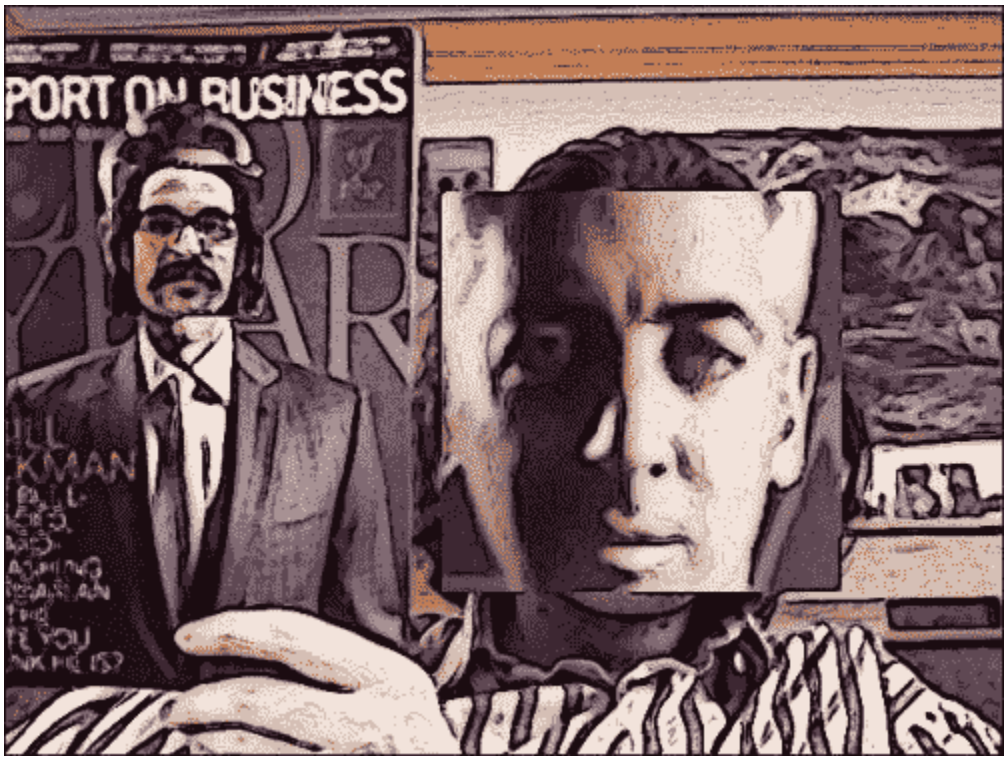
```
WindowManager('Cameo',

 self.onKeypress)
        self._captureManager = CaptureManager(
            cv2.VideoCapture(0),
self._windowManager, True)
        self._faceTracker = FaceTracker()
        self._shouldDrawDebugRects = False
        self._curveFilter =
filters.BGRPortraCurveFilter()

    def run(self):
        """Run the main loop."""
        self._windowManager.createWindow()
        while
self._windowManager.isWindowCreated:
            self._captureManager.enterFrame()
            frame = self._captureManager.frame

            self._faceTracker.update(frame)
            faces = self._faceTracker.faces
            rects.swapRects(frame, frame,
                            [face.faceRect for
face in faces])

            filters.strokeEdges(frame, frame)
            self._curveFilter.apply(frame, frame)

            if self._shouldDrawDebugRects:

 self._faceTracker.drawDebugRects(frame)

            self._captureManager.exitFrame()
            self._windowManager.processEvents()

    def onKeypress(self, keycode):
```

```
        """Handle a keypress.

        space  -> Take a screenshot.
        tab    -> Start/stop recording a
screencast.
        x      -> Start/stop drawing debug
rectangles around faces.
        escape -> Quit.

        """
        if keycode == 32: # space

 self._captureManager.writeImage('screenshot.png'
)
        elif keycode == 9: # tab
            if not
self._captureManager.isWritingVideo:

 self._captureManager.startWritingVideo(
                     'screencast.avi')
            else:

 self._captureManager.stopWritingVideo()
        elif keycode == 120: # x
            self._shouldDrawDebugRects = \
                not self._shouldDrawDebugRects
        elif keycode == 27: # escape
            self._windowManager.destroyWindow()

if __name__=="__main__":
   Cameo().run()
```

以下屏幕截图是来自 Cameo 的。 用户按下`x`后，脸部区域将被勾勒出轮廓：



以下屏幕截图来自 Cameo。 美国商人比尔·阿克曼（Bill Ackman）接管了作者的脸孔：



在相机源之间复制人脸

对于两机版，让我们创建一个新类`CameoDouble`，它是`Cameo`的子类。 在初始化时，`CameoDouble`调用`Cameo`的构造器，还创建第二个`CaptureManager`。 在`run()`的主循环中，`CameoDouble`从两个摄像机获取新帧，然后获取两个帧的面部跟踪结果。 使用`copyRect()`将脸部从一帧复制到另一帧。 然后，显示目标框架，可以选择在其上方绘制调试矩形。 我们可以在`cameo.py`中实现`CameoDouble`，如下所示：

注意

对于某些型号的 MacBook，插入外部网络摄像头后，OpenCV 在使用内置摄像头时会遇到问题。特别是，在等待内置摄像头提供框架时，应用可能会死锁。 如果遇到此问题，请使用两个外接摄像机，而不要使用内置摄像机。

```
class CameoDouble(Cameo):

    def __init__(self):
        Cameo.__init__(self)
        self._hiddenCaptureManager =
CaptureManager(
            cv2.VideoCapture(1))

    def run(self):
        """Run the main loop."""
        self._windowManager.createWindow()
        while
self._windowManager.isWindowCreated:
            self._captureManager.enterFrame()
```

```
 self._hiddenCaptureManager.enterFrame()
            frame = self._captureManager.frame
            hiddenFrame =
self._hiddenCaptureManager.frame

            self._faceTracker.update(hiddenFrame)
            hiddenFaces = self._faceTracker.faces
            self._faceTracker.update(frame)
            faces = self._faceTracker.faces

            i = 0
            while i < len(faces) and i <
len(hiddenFaces):
                rects.copyRect(
                    hiddenFrame, frame,
hiddenFaces[i].faceRect,
                    faces[i].faceRect)
                i += 1

            filters.strokeEdges(frame, frame)
            self._curveFilter.apply(frame, frame)

            if self._shouldDrawDebugRects:

 self._faceTracker.drawDebugRects(frame)

            self._captureManager.exitFrame()

 self._hiddenCaptureManager.exitFrame()
            self._windowManager.processEvents()
```

要运行`CameoDouble`而不是`Cameo`的，我们只需要修改`if __name__=="__main__"`块，如下所示：

```
if __name__=="__main__":
    #Cameo().run() # uncomment for single camera
    CameoDouble().run() # uncomment for double
camera
```

总结

现在，我们有两个版本的 Cameo。 一个版本可在单个相机供稿中跟踪面部，找到面部后，可通过复制和调整大小来交换它们。 另一种版本则跟踪两个相机源中的人脸，如果在每个源中找到人脸，则从一个源中复制人脸并调整其大小以替换另一个人脸。 此外，在这两个版本中，一个摄影机的提要均可见，并对其应用了效果。

这些版本的 Cameo 演示了我们在两章之前提出的基本功能。 用户可以将他或她的脸部移动到另一个身体上，并且可以对结果进行样式化以使其更统一。 但是，移植的脸仍然只是矩形切口。 到目前为止，还没有做出任何努力去掉矩形的非面部部分或对齐重叠的和下面的组件（例如眼睛）。 下一章将探讨一些更复杂的面部融合技术，尤其是使用深度视觉技术。

五、检测前景/背景区域和深度

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/5.md

本章介绍如何使用深度相机中的数据来识别前景和背景区域，这样我们就可以将效果限制为仅前景或背景。 作为前提条件，我们需要一个深度摄像头，例如 Microsoft Kinect，并且需要在支持深度摄像头的情况下构建 OpenCV。 有关构建说明，请参见第 1 章，“设置 OpenCV”。

创建模块

我们用于捕获和处理深度相机数据的代码可在`Cameo.py`外部重用。 因此，我们应该将其分成一个新模块。 让我们在与`Cameo.py`相同的目录中创建一个名为`depth.py`的文件。 我们需要在`depth.py`中使用以下`import`语句：

```
import numpy
```

我们还需要修改我们先前存在的`rects.py`文件，以便将复制操作限制为矩形的非矩形子区域。 为了支持我们将要进行的更改，让我们在`rects.py`中添加以下导入语句：

```
import numpy
import utils
```

最后，我们应用的新版本将使用与深度相关的功能。 因此，让我们在`Cameo.py`中添加以下`import`语句：

```
import depth
```

现在，让我们更深入地研究深度主题。

从深度相机捕获帧

回到第 2 章，“处理文件，摄像机和 GUI”，我们讨论了计算机可以具有多个视频捕获设备，每个设备可以具有多个通道的概念。 假设给定的设备是立体相机。 每个通道可能对应于不同的镜头和传感器。 同样，每个通道可能对应于不同种类的数据，例如正常彩色图像与深度图。 使用 OpenCV 的`VideoCapture`类或包装器`CaptureManager`时，我们可以在初始化时选择一个设备，并且可以从该设备的每一帧读取一个或多个通道。 每个设备和通道均由整数标识。 不幸的是，设备和通道的编号是不直观的。 OpenCV 的 C++ 版本为某些设备和通道的标识符定义了一些常量。 但是，这些常量未在 Python 版本中定义。 为纠正这种情况，让我们在`depth.py`中添加以下定义：

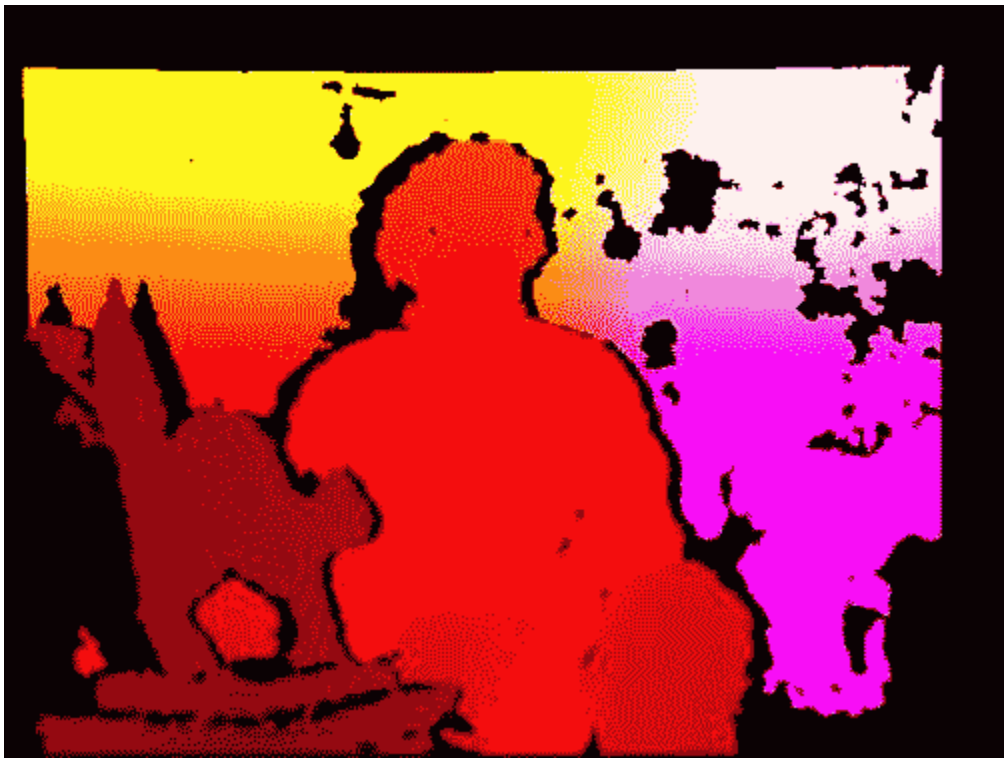
```
# Devices.
CV_CAP_OPENNI = 900 # OpenNI (for Microsoft
Kinect)
CV_CAP_OPENNI_ASUS = 910 # OpenNI (for Asus
Xtion)
# Channels of an OpenNI-compatible depth
generator.
CV_CAP_OPENNI_DEPTH_MAP = 0 # Depth values in mm
(CV_16UC1)
CV_CAP_OPENNI_POINT_CLOUD_MAP = 1 # XYZ in meters
(CV_32FC3)
CV_CAP_OPENNI_DISPARITY_MAP = 2 # Disparity in
pixels (CV_8UC1)
CV_CAP_OPENNI_DISPARITY_MAP_32F = 3 # Disparity
in pixels (CV_32FC1)
```

```
CV_CAP_OPENNI_VALID_DEPTH_MASK = 4 # CV_8UC1
# Channels of an OpenNI-compatible RGB image
generator.
CV_CAP_OPENNI_BGR_IMAGE = 5
CV_CAP_OPENNI_GRAY_IMAGE = 6
```

与深度有关的通道需要一些解释，如以下列表所示：

- **深度图**是灰度图像，其中每个像素值是从相机到表面的估计距离。具体而言，来自`CV_CAP_OPENNI_DEPTH_MAP`通道的图像给出的距离为浮点数毫米。
- **点云图**是彩色图像，其中每种颜色对应于一个空间尺寸（`x`，`y`或`z`）。 具体而言，从相机的角度来看，`CV_CAP_OPENNI_POINT_CLOUD_MAP`通道会生成 BGR 图像，其中 B 为`x`（蓝色为右），G 为`y`（绿色为上），R 为`z`（红色为深）。值以米为单位。
- **视差图**是灰度图像，其中每个像素值是表面的**立体视差**。 为了概念化立体视差，我们假设我们叠加了一个场景的两个图像，这些图像是从不同的角度拍摄的。 结果就像看到双重图像。 对于场景中任何一对孪生对象上的点，我们都可以以像素为单位测量距离。该测量是立体差异。 与远处的物体相比，附近的物体表现出更大的立体视差。 因此，附近的物体在视差图中显得更亮。
- **有效深度掩码**显示给定像素处的深度信息是有效的（由非零值表示）还是无效的（由零值表示）。 例如，如果深度相机依靠红外照明器（红外闪光灯），则深度信息在被该光遮挡（阴影）的区域中无效。

以下屏幕截图显示了一个坐在猫雕塑后面的人的点云图：



以下屏幕截图显示了一个坐在猫雕塑后面的人的视差图：

以下屏幕截图显示了一个坐在猫雕塑后面的人的有效深度面具：

从视差图创建遮罩

出于`Cameo`和的目的，我们对视差图和有效的深度遮罩感兴趣。 它们可以帮助我们完善对面部区域的估计。

使用我们的`FaceTracker`函数和正常彩色图像，我们可以获得面部区域的矩形估计。 通过分析相应的视差图中的矩形区域，我们可以知道矩形中的某些像素离群值-太近或太远而不能真正成为人脸的一部分。 我们可以优化面部区域以排除这些异常值。 但是，我们仅应在有效深度掩码所指示的数据有效的地方应用此测试。

让我们编写一个函数来生成一个遮罩，该遮罩的值对于面部矩形的拒绝区域为`0`，对于接受区域为`1`。 此函数应使用视差图，有效的深度遮罩和矩形作为参数。 我们可以在`depth.py`中实现它，如下所示：

```
def createMedianMask(disparityMap,
validDepthMask, rect = None):
    """Return a mask selecting the median layer,
plus shadows."""
    if rect is not None:
        x, y, w, h = rect
        disparityMap = disparityMap[y:y+h, x:x+w]
        validDepthMask = validDepthMask[y:y+h,
x:x+w]
    median = numpy.median(disparityMap)
    return numpy.where((validDepthMask == 0) | \
                       (abs(disparityMap -
median) < 12),
                       1.0, 0.0)
```

为了识别视差图中的离群值，我们首先使用`numpy.median()`来找到中值，它以数组作为参数。 如果数组的长度为奇数，则`median()`返回如果对数组进行排序，则该值将位于数组的中间。 如果数组的长度为偶数，则`median()`返回将最接近数组中间排序的两个值的平均值。

为了基于每个像素的布尔运算生成遮罩，我们将`numpy.where()`与三个参数一起使用。 作为第一个参数，`where()`接受一个数组，该数组的元素将评估为真还是假。 返回类似尺寸的输出数组。 只要输入数组中的元素为`true`，`where()`函数的第二个参数都将分配给输出数组中的相应元素。 相反，无论输入数组中的哪个元素是`false`，`where()`函数的第三个参数都将分配给输出数组中的相应元素。

当像素的有效视差值与中位数视差值相差 12 或更多时，我们的实现会将像素视为离群值。 我只是通过实验选择了值 12。 以后根据您使用特定相机设置运行`Cameo`时遇到的结果，随时调整此值。

带遮罩的复制操作

作为上一章的一部分，我们将`copyRect()`编写为一种复制操作，将其自身限制为源图像和目标图像的给定矩形。 现在，我们想对该复制操作应用更多限制。 我们要使用与源矩形具有相同尺寸的给定遮罩。 我们将仅复制源矩形中掩码值不为零的那些像素。 其他像素应保留目标图像中的旧值。 可以使用我们最近了解的`numpy.where()`函数来简洁地表达具有条件数组和两个可能的输出值数组的逻辑。

让我们打开`rects.py`并编辑`copyRect()`以添加一个新参数`mask`。 该参数可能是`None`，在这种情况下，我们将退回到`copy`操作的旧实现。 否则，我们接下来确保`mask`和图像具有相同数量的通道。 我们假设`mask`具有一个通道，但是图像可能具有三个通道（BGR）。 我们可以使用`numpy.array`的`repeat()`和`reshape()`方法向`mask`添加重复通道。 最后，我们使用`where()`执行`copy`操作。 完整的实现如下：

```
def copyRect(src, dst, srcRect, dstRect, mask =
None,
             interpolation = cv2.INTER_LINEAR):
    """Copy part of the source to part of the
destination."""

    x0, y0, w0, h0 = srcRect
    x1, y1, w1, h1 = dstRect

    # Resize the contents of the source sub-
rectangle.
    # Put the result in the destination sub-
rectangle.
    if mask is None:
        dst[y1:y1+h1, x1:x1+w1] = \
            cv2.resize(src[y0:y0+h0, x0:x0+w0],
```

```
(w1, h1),
                        interpolation =
interpolation)
    else:
        if not utils.isGray(src):
            # Convert the mask to 3 channels,
like the image.
            mask = mask.repeat(3).reshape(h0, w0,
3)
        # Perform the copy, with the mask
applied.
        dst[y1:y1+h1, x1:x1+w1] = \
            numpy.where(cv2.resize(mask, (w1,
h1),
                                   interpolation
= \

cv2.INTER_NEAREST),
                        cv2.resize(src[y0:y0+h0,
x0:x0+w0], (w1, h1),
                                   interpolation
= interpolation),
                        dst[y1:y1+h1, x1:x1+w1])
```

我们还需要修改`swapRects()`函数，该函数使用`copyRect()`执行矩形区域列表的循环交换。 对`swapRects()`的修改非常简单。 我们只需要添加一个新参数`masks`，它是一个掩码的列表，其元素将传递给相应的`copyRect()`调用。 如果给定的`masks`为`None`，则将`None`传递给每个`copyRect()`调用。 以下是完整的实现：

```
def swapRects(src, dst, rects, masks = None,
              interpolation = cv2.INTER_LINEAR):
    """Copy the source with two or more sub-
rectangles swapped."""
```

```
    if dst is not src:
        dst[:] = src

    numRects = len(rects)
    if numRects < 2:
        return

    if masks is None:
        masks = [None] * numRects

    # Copy the contents of the last rectangle
into temporary storage.
    x, y, w, h = rects[numRects - 1]
    temp = src[y:y+h, x:x+w].copy()

    # Copy the contents of each rectangle into
the next.
    i = numRects - 2
    while i >= 0:
        copyRect(src, dst, rects[i], rects[i+1],
masks[i],
                 interpolation)
        i -= 1

    # Copy the temporarily stored content into
the first rectangle.
    copyRect(temp, dst, (0, 0, w, h), rects[0],
masks[numRects - 1],
             interpolation)
```

请注意，`copyRect()`中的掩码和`swapRects()`中的掩码都默认为`None`。因此，这些函数的新版本与`Cameo`的先前版本向后兼容。

修改应用

对于`Cameo`的深度相机版本，让我们创建一个新类`CameoDepth`作为`Cameo`的子类。 初始化时， `CameoDepth`类会创建一个使用深度相机设备的`CaptureManager`类（对于 Microsoft Kinect 是`CV_CAP_OPENNI`还是对于 Asus Xtion 是`CV_CAP_OPENNI_ASUS`，这取决于我们的设置）。 在`run()`的主循环中，`CameoDepth`函数在每帧中获取视差图，有效的深度遮罩和正常的彩色图像。 正常彩色图像用于估计面部矩形，而视差图和有效深度遮罩用于使用`createMedianMask()`细化面部区域的估计。 使用`copyRect()`交换普通彩色图像中的脸部，并应用这些脸部的相应遮罩。 然后，显示目标框架，可以选择在其上方绘制调试矩形。 我们可以在`cameo.py`中实现`CameoDepth`，如下所示：

```
class CameoDepth(Cameo):
    def __init__(self):
        self._windowManager =
WindowManager('Cameo',

self.onKeypress)
        device = depth.CV_CAP_OPENNI # uncomment
for Microsoft Kinect
        #device = depth.CV_CAP_OPENNI_ASUS #
uncomment for Asus Xtion
        self._captureManager = CaptureManager(
            cv2.VideoCapture(device),
self._windowManager, True)
        self._faceTracker = FaceTracker()
        self._shouldDrawDebugRects = False
        self._curveFilter =
filters.BGRPortraCurveFilter()
    def run(self):
        """Run the main loop."""
```

```
        self._windowManager.createWindow()
        while
self._windowManager.isWindowCreated:
            self._captureManager.enterFrame()
            self._captureManager.channel = \
                depth.CV_CAP_OPENNI_DISPARITY_MAP
            disparityMap =
self._captureManager.frame
            self._captureManager.channel = \

depth.CV_CAP_OPENNI_VALID_DEPTH_MASK
            validDepthMask =
self._captureManager.frame
            self._captureManager.channel = \
                depth.CV_CAP_OPENNI_BGR_IMAGE
            frame = self._captureManager.frame
            self._faceTracker.update(frame)
            faces = self._faceTracker.faces
            masks = [
                depth.createMedianMask(
                    disparityMap, validDepthMask,
face.faceRect) \
                for face in faces
            ]
            rects.swapRects(frame, frame,
                            [face.faceRect for
face in faces], masks)
            filters.strokeEdges(frame, frame)
            self._curveFilter.apply(frame, frame)
            if self._shouldDrawDebugRects:

self._faceTracker.drawDebugRects(frame)
            self._captureManager.exitFrame()
            self._windowManager.processEvents()
```

要运行`CameoDepth`函数而不是`Cameo`或`CameoDouble`函数，我们只需要修改`if __name__=="__main__"`块，如下所示：

```
if __name__=="__main__":
    #Cameo().run() # uncomment for single camera
    #CameoDouble().run() # uncomment for double
camera
    CameoDepth().run() # uncomment for depth
camera
```

以下是显示正在运行的`CameoDepth`类的屏幕截图。 请注意，我们的遮罩会按预期为复制区域提供一些不规则的边缘。 在脸部的左侧和右侧（与背景相遇的位置）比在顶部和底部（与相似深度的头发和颈部区域相遇）更成功：

总结

现在，我们有了一个使用深度相机，面部跟踪，复制操作，遮罩和图像过滤器的应用。 通过开发此应用，我们获得了利用 OpenCV，NumPy 和其他库的功能的实践。 我们还练习了将此功能包装在高级，可重用且面向对象的设计中。

恭喜你！ 您现在具备使用 OpenCV 在 Python 中开发计算机视觉应用的技能。 尽管如此，总会有更多的东西要学习和做！ 如果您喜欢使用 NumPy 和 OpenCV，请从 Packt Publishing 查看以下其他标题：

- 《NumPy Cookbook》，伊万·伊德里斯
- 《OpenCV 2 计算机视觉应用设计手册》，RobertLaganière，该手册在台式机上使用 OpenCV 的 C++ API
- 《通过实用的计算机视觉项目掌握 OpenCV》，由多位作者撰写，其将 OpenCV 的 C++ API 用于多个平台
- 即将出版的书《OpenCV for iOS How-to》，使用针对 iPhone 和 iPad 的 OpenCV 的 C++ API
- 我即将出版的《OpenCV Android 应用编程》，使用了用于 Android 的 OpenCV 的 Java API

至此，我们结束了 OpenCV 的 Python 绑定之旅。 希望您能够将本书及其代码库用作奖励计算机视觉工作的起点。 让我知道您接下来要学习或发展的内容！

附录 A：与 Pygame 集成

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/6.md

本附录显示了如何在 OpenCV 应用中设置 Pygame 库以及如何使用 Pygame 进行窗口管理。 此外，附录还概述了 Pygame 的其他功能以及一些学习 Pygame 的资源。

注意

本章的所有完成代码都可以从我的网站下载。

安装 Pygame

假设我们已经根据第 1 章，“设置 OpenCV”中描述的方法之一设置了 Python。 根据我们现有的设置，我们可以通过以下方式之一安装 Pygame：

- 带有 32 位 Python 的 Windows：从以下位置下载并安装 Pygame 1.9.1。
- 带有 64 位 Python 的 Windows：从以下位置下载并安装 Pygame 1.9.2 预览版。
- 带有 Macports 的 Mac：打开“终端”并运行以下命令：

```
$ sudo port install py27-game
```

- 带有 Homebrew 的 Mac：打开终端并运行以下命令来安装 Pygame 的依赖项，然后安装 Pygame 本身

```
$ brew install sdl sdl_image sdl_mixer sdl_ttf
smpeg portmidi
$ /usr/local/share/python/pip install \
> hg+http://bitbucket.org/pygame/pygame
```

- Ubuntu 及其衍生版本：打开“终端”并运行以下命令：

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

- 其他类似 Unix 的系统：Pygame 在许多系统的标准存储库中可用。 典型的包名称包括`pygame`，`pygame27`，`py-game`，`py27-game`，`python-pygame`，和`python27-pygame`。

现在，Pygame 应该可以使用了。

文档和教程

Pygame 的 API 文档和一些教程可以在以下网址在线找到。

Al Sweigart 的《使用 Python 和 Pygame 制作游戏》是一本烹饪手册，用于在 Pygame 1.9.1 \中重新创建几个经典游戏。 免费的电子版本可从以下网站在线获得。或在以下网站下载 PDF 文件。

派生`Manager.WindowManager`

如第 2 章，“处理照相机，文件和 GUI”中所述，我们的面向对象设计使我们可以轻松地将 OpenCV 的 HighGUI 窗口管理器切换为另一个

窗口管理器，例如 Pygame。 为此，我们只需要继承我们的`managers.WindowManager`类的子类，并覆盖四个方法：`createWindow()`，`show()`，`destroyWindow()`和`processEvents()`。 另外，我们需要导入一些新的依赖项。

要继续，我们需要第 2 章“处理照相机，文件和 GUI”中的`managers.py`文件，和第 4 章“用 Haar 级联跟踪人脸”中的`utils.py`文件。 在`utils.py`中，我们只需要一个函数`isGray()`，我们在第 4 章，“用 Haar 级联跟踪人脸”中实现。 让我们编辑`managers.py`以添加以下导入：

```
import pygame
import utils
```

同样在`managers.py`中，在执行`WindowManager`之后的某个位置，我们想添加名为`PygameWindowManager`的新子类：

```
class PygameWindowManager(WindowManager):
    def createWindow(self):
        pygame.display.init()

pygame.display.set_caption(self._windowName)
        self._isWindowCreated = True
    def show(self, frame):
        # Find the frame's dimensions in (w, h)
format.
        frameSize = frame.shape[1::-1]
        # Convert the frame to RGB, which Pygame
requires.
        if utils.isGray(frame):
            conversionType = cv2.COLOR_GRAY2RGB
        else:
```

```
            conversionType = cv2.COLOR_BGR2RGB
        rgbFrame = cv2.cvtColor(frame,
conversionType)
        # Convert the frame to Pygame's Surface
type.
        pygameFrame = pygame.image.frombuffer(
            rgbFrame.tostring(), frameSize,
'RGB')
        # Resize the window to match the frame.
        displaySurface =
pygame.display.set_mode(frameSize)
        # Blit and display the frame.
        displaySurface.blit(pygameFrame, (0, 0))
        pygame.display.flip()
    def destroyWindow(self):
        pygame.display.quit()
        self._isWindowCreated = False
    def processEvents(self):
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN and \
                    self.keypressCallback is not
None:
                self.keypressCallback(event.key)
            elif event.type == pygame.QUIT:
                self.destroyWindow()
                return
```

注意，我们使用了两个 Pygame 模块：`pygame.display` 和`pygame.event`。

调用`pygame.display.init()`创建一个窗口，调用`pygame.display.quit()`销毁一个窗口。 重复调用`display.init()`无效，因为 Pygame 仅适用于单窗口应用。 Pygame 窗口的绘图表面类型为`pygame.Surface`。 为了获得对该`Surface`的引用，我们可以调

用`pygame.display.get_surface()`或`pygame.display.set_mode()`。 后一个函数在返回实体之前修改`Surface`实体的属性。 一个`Surface`实体具有一个`blit()`方法，该方法将另一个`Surface`和一个坐标对作为参数，其中后一个`Surface`应被“变白”（绘制）到第一个上。 完成当前帧的窗口`Surface`的更新后，我们应该通过调用`pygame.display.flip()`来显示它。

通过调用`pygame.event.get()`可以轮询诸如`keypresses`之类的事件，该函数将返回自上次调用以来发生的所有事件的列表。 每个事件的类型均为`pygame.event.Event`，并具有属性 type，它指示事件的类别，例如`pygame.KEYDOWN`表示按键，pygame.QUIT 表示窗口的**关闭**按钮被点击。 取决于`type`的值，`Event`实体可能具有其他属性，例如，`KEYDOWN`事件的`key`（ASCII 键代码）。

相对于使用 HighGUI 的基本`WindowManager`而言，`PygameWindowManager`通过在每帧 OpenCV 的图像格式和 Pygame 的`Surface`格式之间进行转换而产生一些间接费用。 但是，`PygameWindowManager`提供正常的窗口关闭行为，而基础`WindowManager`不提供。

修改应用

让我们将`cameo.py`文件修改为使用`PygameWindowManager`而不是`WindowManager`。 在`cameo.py`中找到以下行：

```
from managers import WindowManager,
CaptureManager
```

将其替换为：

```
from managers import PygameWindowManager as
WindowManager, \
                       CaptureManager
```

就这样！ 现在，`cameo.py`使用一个 Pygame 窗口，当单击标准“关闭”按钮时，该窗口应该关闭。

Pygame 的进一步使用

我们仅使用了`pygame.display`和`pygame.event`模块的一些基本功能。 Pygame 提供了更多功能，包括：

- 绘制 2D 几何
- 绘制文字
- 管理可绘制 AI 实体（精灵）的分组
- 捕获与窗口，键盘，鼠标和操纵杆/游戏手柄相关的各种输入事件
- 创建自定义事件
- 播放和合成声音和音乐

例如，Pygame 可能是使用计算机视觉的游戏的合适后端，而 HighGUI 则不是。

总结

到现在为止，我们应该有一个应用，该应用使用 OpenCV 捕获（并可能操纵）图像，同时使用 Pygame 显示图像和捕获事件。 从这个基本的集成示例开始，您可能想扩展`PygameWindowManager`来包装其他 Pygame 功能，或者您想创建另一个`WindowManager`子类来包装另一个库。

附录 B：为自定义目标生成 Haar 级联

> 原文：https://gitee.com/apachecn/apachecn-cv-zh/blob/master/docs/opencv-cv-py/7.md

本附录显示了如何生成 Haar 级联 XML 文件，如第 4 章“使用 Haar 级联跟踪面部”中所使用的文件。 通过生成自己的级联文件，我们可以潜在地跟踪任何模式或对象，而不仅仅是面部。 但是，好的结果可能不会很快出现。 我们必须仔细收集图像，配置脚本参数，执行实际测试并进行迭代。 可能涉及大量的人工时间和处理时间。

收集正面和负面的训练图像

你知道抽认卡的教学法吗？ 这是一种向幼儿教授单词和识别技巧的方法。 老师给全班同学展示了一系列图片，并说了以下内容：

> “这是牛。Mo！这是马。Neigh！”

级联文件的生成方式类似于抽认卡教学法。 要学习如何识别母牛，计算机需要预先识别为母牛的**正训练图像**和预先识别为“非母牛”的**负训练图像**。 作为训练师，我们的第一步是收集这两套图像。

在确定要使用多少个正面训练图像时，我们需要考虑用户查看目标的各种方式。 理想，最简单的情况是目标是始终在平坦表面上的 2D 图案。 在这种情况下，一个正面的训练图像可能就足够了。 但是，在其他情况下，可能需要数百甚至数千张训练图像。 假设目标是您所在国家的国旗。 当在文档上打印时，标志的外观可能可预测，但是当在顺风飘扬的织物上打印时，标志的外观变化很大。 诸如人脸之类的自然 3D 目标的外观范围可能更大。 理想情况下，我们的一组正面训练图像应代表我们

的相机可能捕获的许多变化。 可选地，我们的任何正面训练图像都可以包含目标的多个实例。

对于我们的负面训练集，我们需要大量图像，这些图像不包含目标的任何实例，但确实包含相机可能捕获的其他内容。 例如，如果一面旗帜是我们的目标，那么我们的负面训练集可能包括各种天气情况下的天空照片。 （天空不是旗帜，但经常在旗帜后面看到。）不过不要假设太多。 如果相机的环境无法预测，并且目标出现在许多设置中，请使用各种各样的负面训练图像。 考虑构建一套通用的环境图像，您可以在多个训练方案中重复使用这些图像。、

查找训练可执行文件

为了使级联训练尽可能自动化，OpenCV 提供了两个可执行文件。 它们的名称和位置取决于操作系统和 OpenCV 的特定设置，如以下两节所述。

在 Windows 上

Windows 上的两个可执行文件称为`ONopencv_createsamples.exe`和`ONopencv_traincascade.exe`。 它们不是预建的。 而是，仅当您从源代码编译 OpenCV 时，它们才存在。 根据您在第 1 章“设置 OpenCV”中选择的编译方法，它们的父文件夹是以下文件夹之一：

- MinGW：`<unzip_destination>\bin`
- Visual Studio 或 Visual C++ Express：`<unzip_destination>\bin\Release`

如果要将可执行文件的文件夹添加到系统的`Path`变量中，请参考第 1 章，“设置 OpenCV”的“在 Windows XP，Windows Vista，Windows 7 和

Windows 8 上进行选择”部分的信息框中的说明。 否则，请注意可执行文件的完整路径，因为我们将需要在运行它们时使用它。

在 Mac，Ubuntu 和其他类似 Unix 的系统上

Mac，Ubuntu 和其他类似 Unix 的系统上的两个可执行文件称为`opencv_createsamples`和`opencv_traincascade`。 它们的父文件夹是以下文件夹之一，具体取决于您的系统和在第 1 章“设置 OpenCV”中选择的方法：

- 带有 MacPorts 的 Mac：`/opt/local/bin`
- 带有 Homebrew 的 Mac：`/opt/local/bin`或`/opt/local/sbin`
- 具有 Apt 的 Ubuntu：`/usr/bin`
- 使用我的自定义安装脚本的 Ubuntu：`/usr/local/bin`
- 其他类 Unix 系统：`/usr/bin`和`/usr/local/bin`

除 Mac 带有 Homebrew 的情况外，默认情况下，可执行文件的文件夹应位于`PATH`中。 对于 Homebrew，如果要将相关文件夹添加到`PATH`，请参阅第 1 章，“设置 OpenCV”的“使用 Homebrew 和现成的包（不支持深度摄像头）”第二部分中的说明。 否则，请注意可执行文件的完整路径，因为我们需要在运行它们时使用它。

创建训练集和级联

此后，我们将这两个可执行文件称为`<opencv_createsamples>`和`<opencv_traincascade>`。 切记替换适合您的系统和设置的路径和文件名。

这些可执行文件具有某些数据文件作为输入和输出。 以下是生成这些数据文件的典型方法：

1. 手动创建一个描述负面训练图像集的文本文件。 我们将此文件称为`<negative_description>`。
2. 手动创建一个描述正面训练图像集的文本文件。 我们将此文件称为`<positive_description>`。
3. 以`<negative_description>`和`<positive_description>`作为参数运行`<opencv_createsamples>`。 该可执行文件将创建一个描述训练数据的二进制文件。 我们将后一个文件称为`<binary_description>`。
4. 以`<binary_description>`作为参数运行`<opencv_traincascade>`。 该可执行文件创建二进制级联文件，我们将其称为`<cascade>`。

我们可以选择`<negative_description>`，`<positive_description>`，`<binary_description>`和`<cascade>`的实际名称和路径。

现在，让我们详细了解三个步骤。

创建`<negative_description>`

`<negative_description>`是一个文本文件，列出了所有负面训练图像的相对路径。 路径应由换行符分隔。 例如，假设我们具有以下目录结构，其中`<negative_description>`是`negative/desc.txt`：

```
negative
    desc.txt
    images
        negative 0.png
        negative 1.png
```

然后，`negative/desc.txt`的内容可能如下：

```
"images/negative 0.png"
"images/negative 1.png"
```

对于少量图像，我们可以手动编写这样的文件。 对于大量图像，我们应该改用命令行来查找与特定模式匹配的相对路径，并将这些匹配输出到文件中。 继续我们的示例，我们可以通过在 Windows 的“命令提示符”中运行以下命令来生成`negative/desc.txt`：

```
> cd negative
> forfiles /m images\*.png /c "cmd /c echo
@relpath" > desc.txt
```

请注意，在这种情况下，相对路径的格式为`.\images\negative 0.png`，这是可以接受的。

另外，在类似 Unix 的外壳中，例如 Mac 或 Ubuntu 上的 Terminal，我们可以运行以下命令：

```
$ cd negative
$ find images/*.png | sed -e "s/^/\"/g;s/$/\"/g" >
desc.txt
```

创建`<positive_description>`

如果我们有多个正面训练图像，则需要使用`<positive_description>`。 否则，请继续下一节。 `<positive_description>`是一个文本文件，列出了所有积极训练图像的相对路径。 在每个路径之后，`<positive_description>`还包含一系列数字，这些数字指示在图像中找到了多少个目标实例，以及哪些子矩形包含了这些目标实例。 对于每个子矩形，数字按以下顺序排列：x，y，宽度和高度。 考虑以下示例：

```
"images/positive 0.png"  1  120 160 40 40
"images/positive 1.png"  2  200 120 40 60  80 60 20
20
```

在此，`images/positive 0.png`包含子矩形中目标的一个实例，该子矩形的左上角为`(120, 160)`，右下角为`(160, 200)`。 同时，`images/positive 1.png`包含目标的两个实例。 一个实例位于子矩形中，该子矩形的左上角为`(200, 120)`，而其右下角为`(240, 180)`。 另一个实例位于子矩形中，该子矩形的左上角为`(80, 60)`，右下角为`(100, 80)`。

要创建这样的文件，我们可以以与`<negative_description>`相同的方式开始生成图像路径列表。 然后，我们必须基于对图像的专家（人类）分析，手动添加有关目标实例的数据。

通过运行`<opencv_createsamples>`创建`<binary_description>`

假设我们有多个正面训练图像，因此，我们创建了`<positive_description>`，现在可以通过运行以下命令来生成`<binary_description>`：

```
$ <opencv_createsamples> -vec <binary_description>
-info <positive_description> -bg
<negative_description>
```

另外，如果我们有一个正面的训练图像，我们将其称为`<positive_image>`，则应改为运行以下命令：

```
$ <opencv_createsamples> -vec <binary_description>
-image <positive_image> -bg <negative_description>
```

对于其他`<opencv_createsamples>`标志（可选），请参见官方文档。

通过运行`<opencv_traincascade>`创建`<cascade>`

最后，我们可以通过运行以下命令生成`<cascade>`：

```
$ <opencv_traincascade> -data <cascade> -vec
<binary_description> -bg <negative_description>
```

有关`<opencv_traincascade>`的其他（可选）标志的信息，请参见官方文档。

提示

发声

为了好运，在运行`<opencv_traincascade>`时发出模仿声音。 例如，说“Moo！”如果正面训练图像是母牛。

测试和改进`<cascade>`

`<cascade>`是与 OpenCV 的`CascadeClassifier`类的构造器兼容的 XML 文件。 对于如何使用`CascadeClassifier`的示例，请参考第 4 章“用 Haar 级联跟踪人脸”的`FaceTracker`实现。 。 通过复制和修改`FaceTracker`和`Cameo`，您应该能够创建一个简单的测试应用，该应用在跟踪的自定义目标实例周围绘制矩形。

也许在您第一次尝试级联训练时，您将不会获得可靠的跟踪结果。 要提高训练效果，请执行以下操作：

- 考虑使分类问题更具体。 例如，`bald, shaven, male face without glasses`级联可能比普通的`face`级联更容易训练。 稍后，随着结果的改善，您可以尝试再次扩大问题范围。

- 收集更多的训练图像，更多！
- 确保`<negative_description>`包含*所有*负面训练图像，*仅包含*负面训练图像。
- 确保`<positive_description>`包含*所有*正面训练图像，*仅包含*正面训练图像。
- 确保`<positive_description>`中指定的子矩形正确。
- 查看并尝试使用`<opencv_createsamples>`和`<opencv_traincascade>`的可选标志。 这些标志在这个页面的官方文档中进行了描述。

祝你好运，寻找图像！

总结

我们已经讨论了用于生成与 OpenCV 的`CascadeClassifier`兼容的级联文件的数据和可执行文件。 现在，您可以开始收集您喜欢的事物的图像并为其训练分类器！