

- 文件和图像处理
- 文件和图像处理介绍
  - `sys.argv`
  - 命令行选项和参数解析
- 读写图像
  - 读取图像
- 读写图像
- 摄像机帧和视频文件读取
  - 相机帧读取
  - 访问捕获对象的属性
  - 相机帧保存
  - 读取视频文件
- 从IP摄像头读取数据
- 视频文件写入
  - 计算每秒帧数
  - 视频文件写入的注意事项
- 视频捕捉属性播放
  - 从视频捕获对象获取所有属性
  - 使用属性回放视频

- 小结
- 问题

# 文件和图像处理

在任何项目中，处理文件和图像都是关键点。很多项目用文件作为数据输入形式。项目在完成处理生成数据，这些数据也可以文件或图像的形式输出。在计算机视觉中，输入-处理-输出流具有特殊的相关性，因为这些类型的项目具有固有的特征，例如要处理的图像和由机器学习算法生成的模型。

本章介绍文件和图像处理。编写计算机视觉应用程序，要学习如何处理文件和图像，

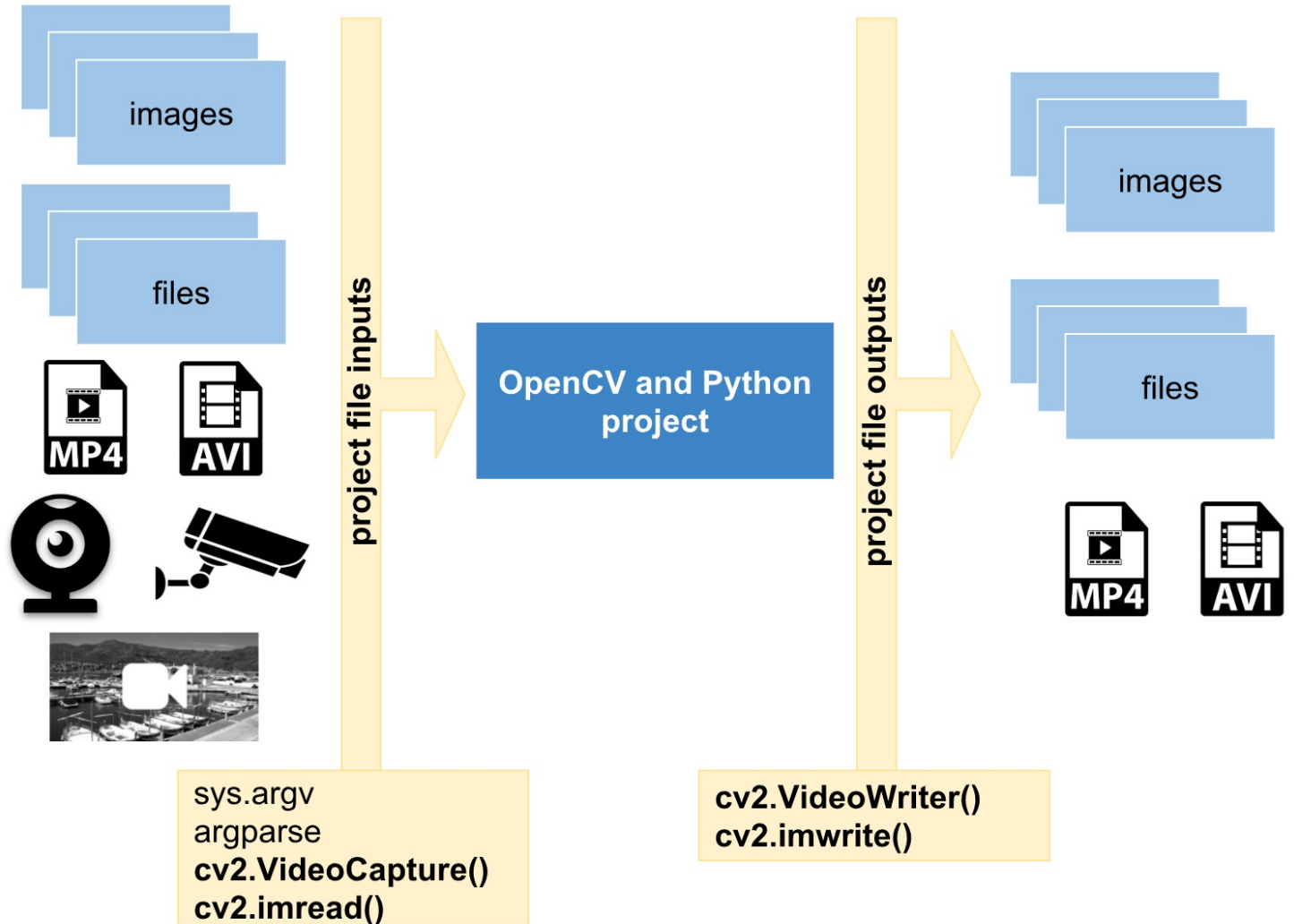
更具体地说，讨论以下主题

- 文件和图像处理的理论介绍
- 读/写图像
- 摄像机帧和视频文件读取

- 视频文件写操作
- 视频捕捉属性操作

# 文件和图像处理介绍

在深入处理文件和图像之前，概述本章内容，如下面



从上图可看到计算机视觉项目，例OpenCV和Python项目，要处理输入文件，例如文件和图像。此外，经过处理后，项目可输出文件，如图像和文件。本章说明如何处理这些需求，以及如何实现输入-处理-输出流。

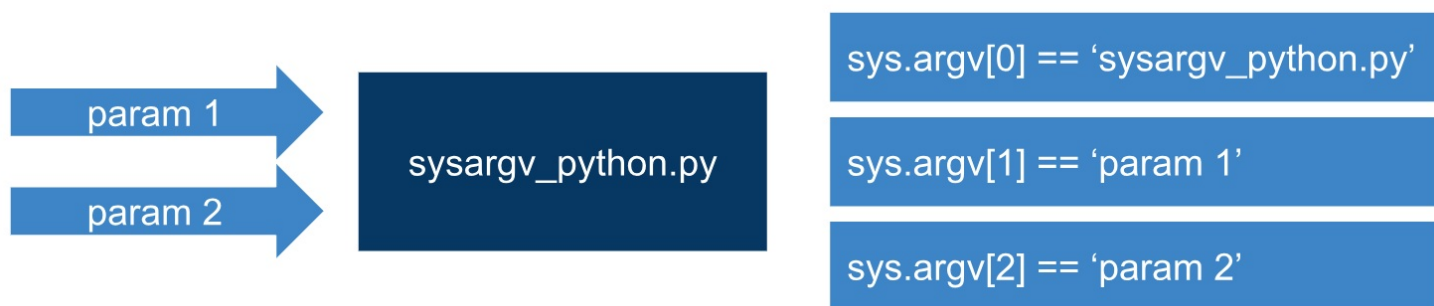
程序运行主要和必要步骤是处理命令行参数，这些参数提供给程序或脚本参数化信息。例如，编写两数相加的脚本，通常的方法是使用两个参数，这两个参数提供两数相加所需的信息。在计算机视觉项目中，图像和不同类型的文件通常作为命令行参数传递给脚本。

命令行参数是参数化程序执行的一种常见简单的方法。

## **sys. argv**

Python使用sys. argv处理命令行参数。当程序执行时，Python从命令行sys. argv列表获取所有的值。

该列表第一个元素`sys.argv[0]`是脚本的完整路径，或与操作系统相关的脚本名称。列表的第二个元素`sys.argv[1]`是脚本的第一个参数等。下面关系图中可看到这一点，`sysargv_python.py`脚本使用两个参数执行，



用`sysargv_python.py`脚本看`sys.argv`如何运行

```
# Import the required packages import sys
# We will print some information in connection
print("The name of the script being processed")
print("The number of arguments of the script")
print("The arguments of the script are: '{}'.")
```

如果我们在没有任何参数的情况下执行这个脚本，我们将看到以下信息

```
The name of the script being processed is: 'sys'
The number of arguments of the script is: '1'
The arguments of the script are: '['sysargv_pyt
```

另外，如果用一个参数执行这个脚本，例如  
sysargv\_python.py OpenCV，将获以下信息

```
The name of the script being processed is: 'sys'
The number of arguments of the script is: '2'
The arguments of the script are: '['sysargv_pyt
```

可以看到，列表第一个元素sysargv\_python.py  
(sys.argv[0])是脚本名。列表第二个元素  
sys.argv[1], OpenCV是脚本第一个参数。

argv[0]是脚本名，是否为完整路径名取决于操作系统。  
见<https://docs.python.org/3/library/sys>。  
更多信息

## 命令行选项和参数解析

当程序有复杂的参数或多个文件名,应考虑不直接处理`sys.argv`。或者用Python的`argparse`库,它以系统方式处理命令行参数,可用于编写用户友好的命令行程序。即Python在标准库中有`argparse`模块(<https://docs.python.org/3/library/argparse.html>),用于解析命令行参数。首先,确定程序需要什么参数,`argparse`将这些参数解析为`sys.argv`。此外,`argparse`生成帮助和使用消息,并在`arguments`无效时发出错误。

这里给出介绍这个模块的最小示例`argparse_minimum.py`,如下所示

```
# Import the required packages
import argparse

# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()

# The information about program arguments is st
# ArgumentParser parses arguments through the p
parser.parse_args()
```

在没有参数的情况下运行这个脚本，不会向stdout显示任何内容。但是，如果我们包含——help(或-h)选项，我们将获得脚本的使用消息

```
usage: argparse_minimal.py [-h] optional
```

```
arguments:
```

```
-h, --help show this help message and exit
```

例如，指定任何其他参数都会导致错误



```
argparse_minimal.py 6
```

```
usage: argparse_minimal.py [-h]
```

```
argparse_minimal.py: error: unrecognized
```

```
arguments: 6
```

因此必须使用-h参数调用这个脚本。这样，message消息将显示。没有定义参数，不允许有其他可能的输入。通过这种方式，argparse第二个示例是添加一个参数，如argparse\_positional\_arguments.py

```
# Import the required packages
import argparse

# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()

# We add a positional argument using add_argume
parser.add_argument("first_argument", help="thi

# The information about program arguments is st
# Then, it is used when the parser calls parse_
# ArgumentParser parses arguments through the p
args = parser.parse_args()

# We get and print the first argument of this s
print(args.first_argument)
```

添加了`add_argument()`方法。此方法用于指定程序将接受哪些命令行选项。在本例中，`first_argument`参数是必需的。此外，`argparse`模

块存储所有参数，将其名称与本例中添加的每个参数的名称`first_argument`匹配。因此为获得参数，执行`args.first_argument`。

运行脚本：

```
argparse_positional_arguments.py 5,
```

输出将是5。如果运行脚本时没有参数如：

```
argparse_positional_arguments.py
```

输出如下

```
usage: argparse_positional_arguments.py [-h] first_argument
argparse_positional_arguments.py: error:
the following arguments are
required: first_argument
```

用`-h`选项执行脚本，输出将如下所示

```
usage: argparse_positional_arguments.py [-h] first_argument positional arguments:
```

first\_argument this is the string text in connection with first\_argument optional arguments:

-h, --help show this help message and exit

缺省情况下，`argparse`把我们给的选项当作字符串。如果参数不是字符串，应建立`type`选项。我们将看到`argparse_sum_two_numbers.py`脚本添加了两个参数，这两个参数是`int`类型的

```
# Import the required packages
import argparse

# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()

# We add 'first_number' argument using add_argu
parser.add_argument("first_number", help="first

# We add 'second_number' argument using add_arg
parser.add_argument("second_number", help="seco

# The information about program arguments is st
# Then, it is used when the parser calls parse_
# ArgumentParser parses arguments through the p
args = parser.parse_args()
print("args: '{}'.format(args))

print("the sum is: '{}'.format(args.first_numb

# Additionally, the arguments can be stored in
args_dict = vars(parser.parse_args())
```

```
# We print this dictionary:
print("args_dict dictionary: '{}'.format(args_

# For example, to get the first argument using
print("first argument from the dictionary: '{}'
```

脚本在没有参数的情况下执行，输出如下

```
argparse_sum_two_numbers.py
usage: argparse_sum_two_numbers.py [-h]
first_number second_number
argparse_sum_two_numbers.py: error: the
following arguments are required:
first_number, second_number
```

此外，用-h选项执行脚本，输出如下

```
argparse_sum_two_numbers.py --help
usage: argparse_sum_two_numbers.py [-h]
first_number second_number

positional arguments:
first_number first number to be added
```

second\_number second number to be added

optional arguments:

-h, --help show this help message and exit

在前面的示例中，介绍了通过调用`vars()`函数在字典中存储参数的可能性

```
# Additionally, the arguments can be stor
```

```
# We print this dictionary:
```

```
print("args_dict dictionary: '{}'.format(args_
```

```
# For example, to get the first argument using
```

```
print("first argument from the dictionary:
```

例如，如果以`argparse_sum_two_numbers.py 5 10`的形式执行此脚本，输出将如下所示

```
args: 'Namespace(first_number=5,
```

```
second_number=10)' the sum is: '15'
```

```
args_dict dictionary: {'first_number': 5,
```

```
'second_number': 10}' first argument from  
the dictionary: '5'
```

这是对`sys.argv`和`argparse`的简介介绍。`argparse`的更多介绍可看

<https://docs.python.org/3/howto/argparse.html>。此外，它的文档非常详细和细致，有大量例子(<https://docs.python.org/3/library/argparse.html>)。在“图像读取和写入”一节中学习如何在OpenCV和Python程序中使用`argparse`读取和写入图像。

# 读写图像

在计算机视觉项目中，图像通常用作脚本中的命令行参数。接下来的部分中将看到如何读写图像。

## 读取图像



下面的例子是`argparse_load_image`。显示如何加载图像

```
# Import the required packages
import argparse
import cv2

# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()

# We add 'path_image' argument using add_argume
parser.add_argument("path_image", help="path to

# The information about program arguments is st
# Then, it is used when the parser calls parse_
# ArgumentParser parses arguments through the p
args = parser.parse_args()

# We can now load the input image from disk:
image = cv2.imread(args.path_image)

# Parse the argument and store it in a dictiona
args = vars(parser.parse_args())

# Now, we can also load the input image from di
image2 = cv2.imread(args["path_image"])
```

```
# Show the loaded image:
cv2.imshow("loaded image", image)
cv2.imshow("loaded image2", image2)

# Wait until a key is pressed:
cv2.waitKey(0)

# Destroy all windows:
cv2.destroyAllWindows()
```

本例所需的参数是`path_image`，包含所要加载图像的路径。图像的路径是字符串。位置参数中不应该包含任何类型，因为它是默认的字符串。

`path_image`和`args["path_image"]`两个参数包含图像的路径，两种从参数中获取值的方法，将用它们作为`cv2.imread()`函数的参数。

# 读写图像

## 在OpenCV中读写图像

一种常见的方法是加载一个图像，执行某种处理，最后输出这个处理过的图像(有关这三个步骤的详细说明，请参阅OpenCV中的第2章图像基础)。在这个意义上，处理后的图像可以保存到磁盘。在下面的示例中，将介绍这三个步骤(加载、处理和保存)。在这种情况下，处理步骤非常简单(将图像转换为灰度)。在下面的示例

`argparse_load_processing_save_image.py`中可以看到这一点

```
import argparse
import cv2

# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()

# Add 'path_image_input' argument using add_arg
parser.add_argument("path_image_input", help="p

# Add 'path_image_output' argument using add_ar
parser.add_argument("path_image_output", help="

# Parse the argument and store it in a dictiona
args = vars(parser.parse_args())

# We can load the input image from disk:
image_input = cv2.imread(args["path_image_input

# Show the loaded image:
cv2.imshow("loaded image", image_input)

# Process the input image (convert it to graysc
gray_image = cv2.cvtColor(image_input, cv2.COLO
```

```
# Show the processed image:
cv2.imshow("gray image", gray_image)

# Save the processed image to disk:
cv2.imwrite(args["path_image_output"], gray_image)

# Wait until a key is pressed:
cv2.waitKey(0)

# Destroy all windows:
cv2.destroyAllWindows()
```

在前面的示例中，有两个必需的参数。第一个是 `path_image_input`，包含想要加载的图像的路径。图像路径是字符串。位置参数中不应该包含任何类型，因为它是默认字符串。第二个是 `path_image_output`，包含想要保存结果的目录路径。例中有将彩色图像转换为灰度图像的操作。

```
# Process the input image (convert it to grayscale)
gray_image = cv2.cvtColor(image_input, cv2.COLOR_BGR2GRAY)
```

注意第二个参数`cv2.COLOR_BGR2GRAY`，加载图像是BGR彩色图像。若加载RGB彩色图像，希望转换为灰度，用`cv2.COLOR_RGB2GRAY`。

这是一个非常简单的处理。在以后的章节中，将展示更详细的处理算法。

# 摄像机帧和视频文件读取

一些项目中须捕获摄像机帧，例如用笔记本电脑的网络摄像头捕获帧。在OpenCV中，`cv2.VideoCapture`是用于从不同来源(如图像序列、视频文件和相机)捕获视频的类。这一节介绍一些捕获摄像机帧类的示例

## 相机帧读取

第一个例子是`read_camera`，展示如何从计算机上摄像机读取帧，参数是`index_camera`，指示要读取的摄像机的索引。如果已经将网络摄像头连接到计算机，索引为0。另外，如果有第二个相机，可以通过传递1来选择它。参数的类型是`int`。

第一步用`cv2.VideoCapture`是创建一个对象的工作。在本例中，对象是`capture`，调用构造函数

```
# We create a VideoCapture object to read from  
capture = cv2.VideoCapture(args.index_camera)
```

如果`index_camera`是0(第一个连接的相机)，相当于`cv2.VideoCapture(0)`。要检查是否正确地建立了连接，用`capture.isopen()`方法，如果不能建立连接，该方法返回`False`。如果捕获初始化正确，返回`True`。

调用`capture.read()`方法，摄像机一帧一帧地捕获镜头帧，它从摄像机返回帧。这个框架与OpenCV中



的图像具有相同的结构，可以以相同的方式使用它。例如，要将帧转换为灰度，请执行以下操作：

```
gray_frame = cv2.cvtColor(frame,  
cv2.COLOR_BGR2GRAY)
```

另外，`capture.read()` 返回一个 `bool`。此 `bool` 指示是否已正确地从捕获对象读取帧。

## 访问捕获对象的属性

可用 `capture.get(property_identifier)` 访问捕获对象的一些属性。在本例中，获取了一些属性，如帧宽度、帧高度和帧每秒 (fps)。如果调用不支持的属性，则返回值为 0

```
# Import the required packages
import cv2
import argparse

# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()

# We add 'index_camera' argument using add_argu
parser.add_argument("index_camera", help="index
args = parser.parse_args()

# We create a VideoCapture object to read from
capture = cv2.VideoCapture(args.index_camera)

# Get some properties of VideoCapture (frame wi
frame_width = capture.get(cv2.CAP_PROP_FRAME_WI
frame_height = capture.get(cv2.CAP_PROP_FRAME_H
fps = capture.get(cv2.CAP_PROP_FPS)

# Print these values:
print("CV_CAP_PROP_FRAME_WIDTH: '{}'.format(fr
print("CV_CAP_PROP_FRAME_HEIGHT : '{}'.format(
print("CAP_PROP_FPS : '{}'.format(fps))
```

```
# Check if camera opened successfully
if capture.isOpened() is False:
    print("Error opening the camera")

# Index to save current frame
frame_index = 0

# Read until video is completed
while capture.isOpened():
    # Capture frame-by-frame from the camera
    ret, frame = capture.read()

    if ret is True:
        # Display the captured frame:
        cv2.imshow('Input frame from the camera', frame)

        # Convert the frame captured from the camera to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Display the grayscale frame:
        cv2.imshow('Grayscale input camera', gray_frame)

        # Press c on keyboard to save current frame
        if cv2.waitKey(20) & 0xFF == ord('c'):
```

```
        frame_name = "camera_frame_{}.png".format(
            frame_index)
        gray_frame_name = "grayscale_camera_{}.png".format(
            frame_index)
        cv2.imwrite(frame_name, frame)
        cv2.imwrite(gray_frame_name, gray_frame)
        frame_index += 1

    # Press q on keyboard to exit the program
    if cv2.waitKey(20) & 0xFF == ord('q'):
        break

    # Break the loop
    else:
        break

# Release everything:
capture.release()
cv2.destroyAllWindows()
```

## 相机帧保存

前面的示例, 修改添加功能是比较容易的。将有趣的帧保存到硬盘, 例程read\_camera\_capture.py完成

这个功能。当按下键盘C键时，当前帧保存到磁盘。保存BGR和灰度帧。执行此功能的代码如下

```
# Press c on keyboard to save current frame
if cv2.waitKey(20) & 0xFF == ord('c'):
    frame_name = "camera_frame_{}.png".format(f
    gray_frame_name = "grayscale_camera_frame_{}
    cv2.imwrite(frame_name, frame)
    cv2.imwrite(gray_frame_name, gray_frame)
    frame_index += 1
```

ord('c')返回8位c字符值。cv2.waitKey()值是按位的，用&与0xFF运算，获取后8位，可对两个8位值进行比较。当按下C键时，两个帧起文件名，并将图像保存到硬盘。最后，增加frame\_index，以便为保存下一帧做好准备。read\_camera\_capture.py为完整代码。

## 读取视频文件

`cv2.VideoCapture`还允许读取视频文件。为读取视频文件，在创建`cv2.VideoCapture`对象时应提供视频文件的路径。

```
# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()
# We add 'video_path' argument using add_argume
parser.add_argument("video_path", help="path
args = parser.parse_args()
# Create a VideoCapture object. In this case, t
capture = cv2.VideoCapture(args.video_path)
```

Check out `read_video_file.py` to see the full example of how to read and display a video file using `cv2.VideoCapture`.

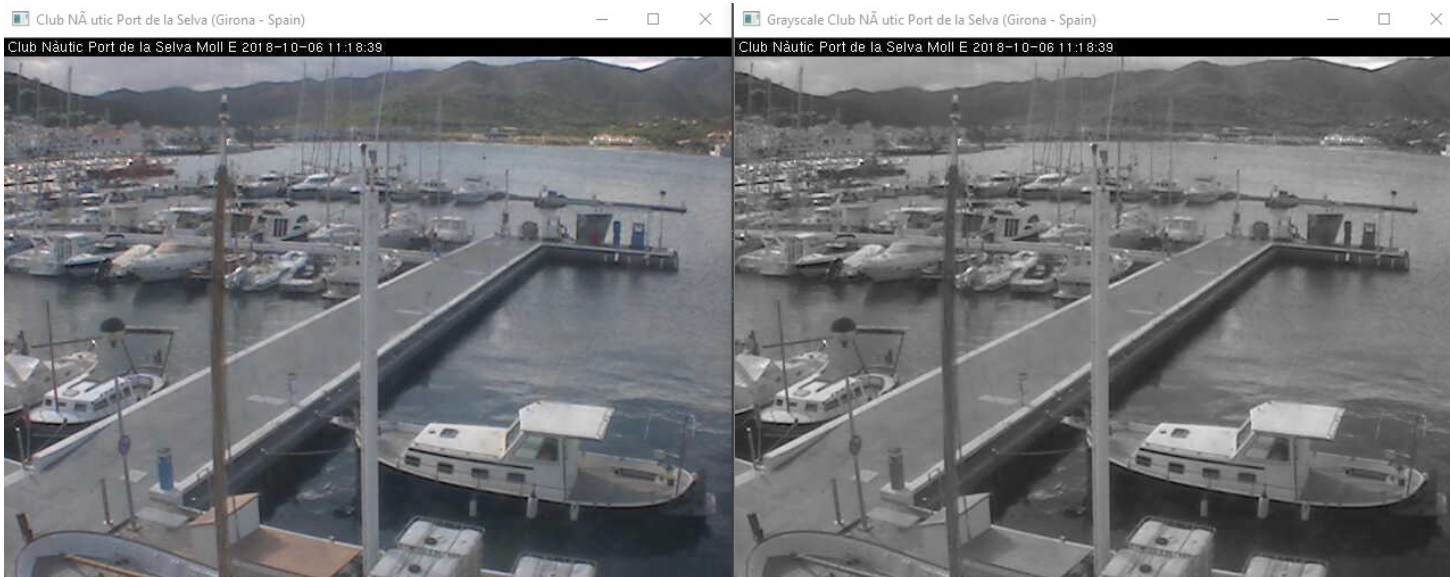
`read_video_file.py`

查看`read_video_file.py` 完整代码，看如何使用`cv2.VideoCapture`读取和显示视频文件。

# 从IP摄像头读取数据

`cv2.VideoCapture`也可从IP摄像头读取数据。在OpenCV中从IP摄像机中读取数据与从文件中读取数据非常相似。在这个意义上，只需`cv2.VideoCapture`参数改变。不需要在本地网络中使用IP摄像机就能试验此功能。有许多公共IP摄像机可以连接。例如，我们将连接到一个IP公共摄像头，它位于Club Nautic Port de la Selva Costa Brava Cap de Creus(西班牙赫罗纳)。该端口的网页位于<https://www.cnps.cat/>。您可以导航到网络摄像头部分(<https://www.cnps.cat/webcams/>)，以找到一些网络摄像头连接。

需要修改的是设`cv2.VideoCapture`参数。在本例中是<http://217.126.89.102:8010/axcgi/mjpg/video.cgi>。执行示例(`read_ip_camera.py`)，会得到下面屏幕截图，其中显示了从IP相机获得的BGR和灰度图像



# 视频文件写入

在本节中，我们将看到如何使用`cv2.VideoWriter`写入视频文件。首先介绍一些概念，例如`fps`、编解码器和视频文件格式。

## 计算每秒帧数

在摄像机帧和视频文件部分，介绍了如何从`cv2.VideoCapture`对象获取属性。`fps`是计算机视觉项目的重要指标，表示每秒处理多少帧。`fps`数越高



越好。算法的每秒处理的帧数取决于要解决的具体问题。例如，如果算法跟踪和检测行人，那么15 fps可能就足够了。但如果目标是检测和跟踪高速公路上高速行驶的汽车，20-25 fps则是必要的。

因此，了解计算机视觉项目中计算fps是很重要的。下面例子中，`read_camera_fps.py`，修改`read_camera.py`输出fps的值。下面的代码显示了关键点

```
# Read until the video is completed, or 'q' is
while capture.isOpened():
# Capture frame-by-frame from the camera
    ret, frame = capture.read()

    if ret is True:
        # Calculate time before processing the fr
        processing_start = time.time()

        # All the processing should be included her
        # ...
        # ...
        # End of processing
        # Calculate time after processing the frame
        processing_end = time.time()

        #Calculate the difference
        processing_time_frame = processing_end - pr
        FPS = 1 / time_per_frame

        # Show the number of frames per second
        print("fps: {}".format(1.0 / processing_tim
# Break the loop
else:
    break
```

首先，在处理完成之前取时间

```
processing_start = time.time()
```

所有处理完成后，取时间

```
processing_end = time.time()
```

计算差值

```
processing_time_frame = processing_end -  
processing_start
```

最后，计算并打印fps的数量

```
print("fps: {}".format(1.0 /  
processing_time_frame))
```

## 视频文件写入的注意事项

视频编码是一种用于压缩和解压数字视频的软件。

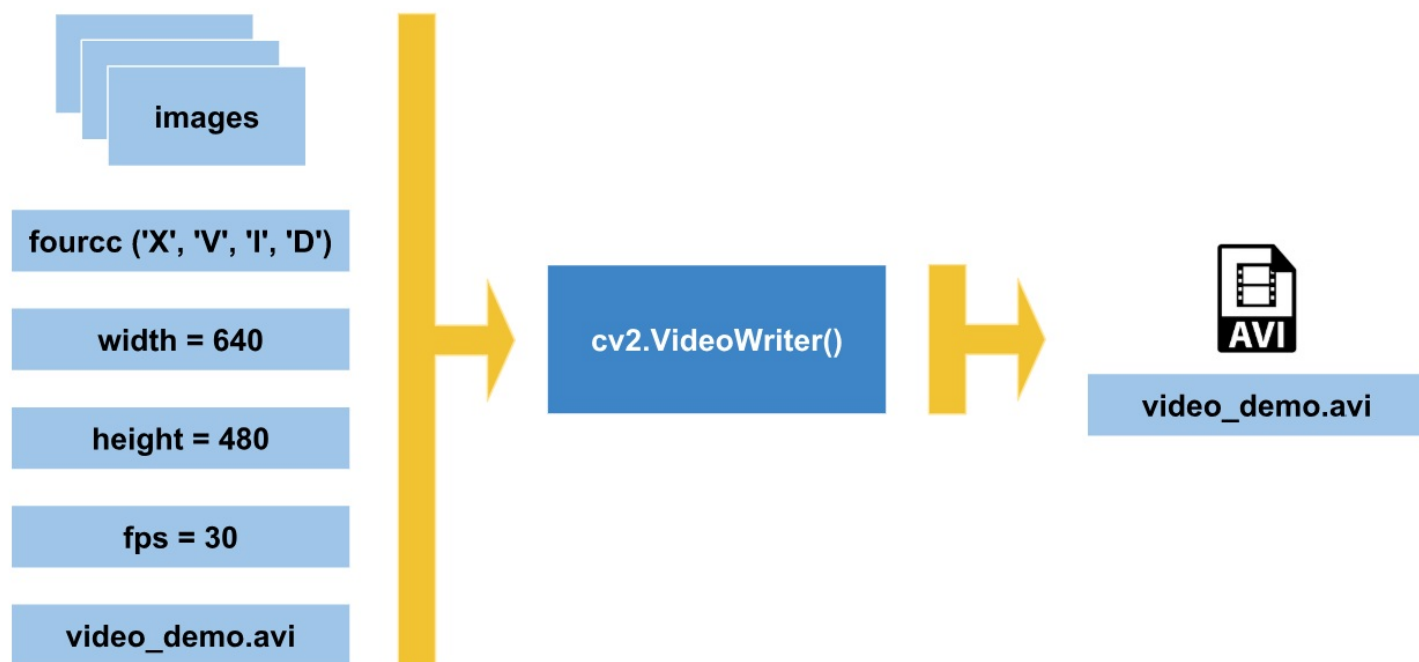
可用编解码器将未压缩视频转换为压缩视频，也可用编解码器将压缩视频解压。压缩视频格式通常遵

循标准规范，称为视频压缩规范或视频编码格式。OpenCV提供了FOURCC，这是一个4字节的编码，用于指定视频编解码器。FOURCC代表四字符代码。所有编码列表可在<http://www.fourcc.org/codecs.php>中看到。应该考虑到所支持的编解码器是平台相关的。这意味着要使用特定的编解码器，这个编解码器要安装在系统。典型的解码器有DIVX、XVID、X264和MJPG。

另外，视频文件格式是一种用于存储数字视频数据的文件格式。典型的视频文件格式有AVI (*. AVI*)、MP4 (*. MP4*)、QuickTime (*. mov*)和Windows Media video (*. wmv*)。

最后，应该考虑到视频文件格式(例如\*.avi)和FOURCC(例如DIVX)之间的正确组合并不简单，可能要试验和使用这些值。在OpenCV中创建视频文件时，要考虑这些因素。

下图作了总结



该图总结了在OpenCV中使用`cv2.VideoWriter()`创建视频文件时应该考虑的主要事项。图中，是创建`video_demo.avi`视频，FOURCC值为XVID，视频文件格式为AVI (\*. AVI)，还有视频fps和每帧尺寸

下面例子`write_video_file.py`写视频文件，有助于发挥这些概念。本例的关键点进行了注释。所需的参数是视频文件名，例如`video_demo.avi`)

```
# We first create the ArgumentParser object
# The created object 'parser' will have the nec
# to parse the command-line arguments into data
parser = argparse.ArgumentParser()
# We add 'output_video_path' argument using add
parser.add_argument("output_video_path", help=
args = parser.parse_args())
```

We are going to take frames from the first camera that's connected to our computer.

Therefore, we create the object accordingly:

将从连接到电脑的第一个摄像头上取帧。创建对象

```
# Create a VideoCapture object and pass 0 as ar
capture = cv2.VideoCapture(0)
```

接下来，将从捕获对象获取一些属性，帧宽、帧高和帧速率。要用它们创建视频文件

```
# Get some properties of VideoCapture (frame
frame_width = capture.get(cv2.CAP_PROP_FRAME_WI
frame_height = capture.get(cv2.CAP_PROP_FRAME_H
fps = capture.get(cv2.CAP_PROP_FPS)
```

用FOURCC指定视频编解码器，是平台相关的。在本例中，编解码器定义为XVID

```
# FourCC is a 4-byte code used to specify the v
# In this case, define the codec XVID
fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I',
```

下面程序行也可以

```
# FourCC is a 4-byte code used to specify the v
# In this case, define the codec XVID
fourcc = cv2.VideoWriter_fourcc(*'XVID')
```

然后，创建cv2.VideoWriter对象，out\_gray。取输入摄像机相同的属性。最后一个参数是False，以便可以用灰度表示视频。如果建彩色视频，最后一个参数应该为True

```
# Create VideoWriter object. We use the same p
# Last argument is False to write the video in
out_gray = cv2.VideoWriter(args.output_vide
```

我们通过使用`capture.read()`从捕获对象获取逐帧输出。每一帧都转换成灰度并写入视频文件。显示帧，但不一定写视频。如果按下q，程序结束



```
# Read until video is completed or 'q' is press
while capture.isOpened():
    # Read the frame from the camera
    ret, frame = capture.read()
    if ret is True:
        # Convert the frame to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Write the grayscale frame to the video
        out_gray.write(gray_frame)
        # We show the frame (this is not necessary)
        # But we show it until 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
```

最后释放cv2.VideoCapture, cv2.VideoWriter对象,  
销毁创建的窗口

```
# Release everything:
capture.release()
out_gray.release()
cv2.destroyAllWindows()
```

完整代码在write\_video\_file.py文件。

# 视频捕捉属性播放

在之前例子中，我们看到了如何从cv2.VideoCapture对象中获取一些属性。本节中我们将看到如何获得所有的属性并了解它们是如何工作的，及用这些属性来加载视频文件并向后方向输出，即先显示视频的最后一帧，以此类推。

## 从视频捕获对象获取所有属性

read\_video\_file\_all\_properties.py脚本显示所有属性。其中一些属性只有在使用相机时才有效，而不是在使用视频文件时。在这些情况下，返回0值。此外，decode\_fourcc()函数将capture.get(cv2.CAP\_PROP\_FOURCC)返回值转换为

包含编解码器的int表示形式的字符串值。这个值转换成一个四字节字符，表示正确地输出编解码器。  
decode\_fourcc() 函数处理这个问题。

该函数的代码如下

```
def decode_fourcc(fourcc):  
    """Decodes the fourcc value to get the four cha  
    """  
  
    fourcc_int = int(fourcc)  
    # We print the int value of fourcc  
    print("int value of fourcc: '{}'.format(fo  
    # We can also perform this in one line:  
    # return "".join([chr((fourcc_int >> 8 * i)  
    fourcc_decode = ""  
    for i in range(4):  
        int_value = fourcc_int >> 8 * i & 0xFF  
        print("int_value: '{}'.format(int_valu  
        fourcc_decode += chr(int_value)  
    return fourcc_decode
```

它是如何工作的，下图表总结了主要步骤：

`capture.get(cv2.CAP_PROP_FOURCC)`

828601953

00110001011000110111011001100001

00110001-01100011-01110110-01100001

97-118-99-49

a-v-c-1

可以看到，第一步是获取

`capture.get(cv2.CAP_PROP_FOURCC)` 返回的值 `int` 表示形式，是一个字符串。然后，进行四次迭代，以获得每8位的数据，并将这8位转换为 `int`，最后，使用 `chr()` 函数将这些 `int` 值转换为 `char`。应该注意，只用一行代码中执行这个函数，如下所示

```
return "".join([chr((fourcc_int >> 8 * i) & 0xF
```

`CAP_PROP_POS_FRAMES` 属性给出视频文件的当前帧，`CAP_PROP_POS_MSEC` 属性给出当前帧的时间戳。还可使用 `CAP_PROP_FPS` 属性获得 `fps` 的数量。

`CAP_PROP_FRAME_COUNT` 属性提供视频文件的总帧数

要获取和打印所有属性，请使用以下代码

# Get and print these values:

# Get and print these values:

```
print("CV_CAP_PROP_FRAME_WIDTH: '{}'.format(ca
print("CV_CAP_PROP_FRAME_HEIGHT : '{}'.format(
print("CAP_PROP_FPS : '{}'.format(capture.get(
print("CAP_PROP_POS_MSEC : '{}'.format(capture
print("CAP_PROP_POS_FRAMES : '{}'.format(captu
print("CAP_PROP_FOURCC : '{}'.format(decode_fo
print("CAP_PROP_FRAME_COUNT : '{}'.format(capt
print("CAP_PROP_MODE : '{}'.format(capture.get
print("CAP_PROP_BRIGHTNESS : '{}'.format(captu
print("CAP_PROP_CONTRAST : '{}'.format(capture
print("CAP_PROP_SATURATION : '{}'.format(captu
print("CAP_PROP_HUE : '{}'.format(capture.get(
print("CAP_PROP_GAIN : '{}'.format(capture.get
print("CAP_PROP_EXPOSURE : '{}'.format(capture
print("CAP_PROP_CONVERT_RGB : '{}'.format(capt
print("CAP_PROP_RECTIFICATION : '{}'.format(ca
print("CAP_PROP_ISO_SPEED : '{}'.format(captur
print("CAP_PROP_BUFFERSIZE : '{}'.format(captu
```

read\_video\_file\_all\_properties.py有完整代码

# 使用属性回放视频

为了解如何使用前面提到的属性，  
`read_video_file_backwards.py`，用其中的一些属性来加载视频并后方向输出，首先显示视频的最后一帧，以此类推。我们将使用以下属性

- `cv2.CAP_PROP_FRAME_COUNT`：此属性提供帧的总数
- `cv2.CAP_PROP_POS_FRAMES`：此属性提供当前帧

第一步是获得最后一帧的索引

```
# We get the index of the last frame of the vi  
frame_index = capture.get(cv2.CAP_PROP_FRAME_CO
```

将当前帧设置为读到这个位置

```
# We set the current frame position captu
```

这样，可以像往常一样读取帧

```
# Capture frame-by-frame from the video file re  
frame = capture.read()
```

最后，索引递减从视频文件中读取下一帧

```
# Decrement the index to read next frame  
frame_index = frame_index - 1
```

完整代码在`read_video_file_backwards.py`中。这个脚本可以很容易地修改为保存向后播放的结果视频，而不仅仅是显示它。问题部分提及

## 小结

这一章我们看到处理图像和文件是计算机视觉项目的关键要素。在这类项目中，一种常见的方法是加载图像，执行处理，输出处理后的图像。在本章中回顾了这个过程。视频流相关的

`cv2.VideoCapture`, `cv2.VideoWriter`都讲述。还说



明了cv2.VideoWriter类视频写处理。视频文件写处理，回顾了两个方面：视频编解码器，如DIVX，视频文件格式，如AVI。为了使用视频编解码器，OpenCV提供了四字节码FOURCC。典型的解码器是DIVX、XVID、X264和MJPG，典型的视频文件格式是AVI (. AVI)、MP4 (. MP4)、QuickTime (. mov)和Windows Media video (. wmv)。



也讨论了fps概念以及如何在程序中计算它，还研究了如何得到cv2.VideoCapture对象的所有性质，以及如何使用它们加载视频并向后输出，视频的最后一帧开始播放。如何处理命令行参数。Python用sys.argv处理命令行参数。当程序参数复杂或多个文件名时，应使用Python的argparse库。

下一章我们将学习如何用OpenCV库绘制基本和高级图形。OpenCV提供了画直线、圆、矩形、椭圆、文本和折线的函数。在计算机视觉项目中，绘制图像基本形状是常用的，以完成以下任务，

- 显示算法的一些中间结果，例如被检测对象的边界框
- 显示算法的最终结果，例如被检测对象的类，如汽车、猫或狗
- 显示一些调试信息，例如执行时间

下一章将对计算机视觉算法有很大的帮助。

# 问题

1. `sys.argv[1]`是什么？
2. 编写代码，添加int类型的`first_number`参数，并含有用`parser.add_argument()`添加的第一个帮助号。
3. 编写代码，将图像保存到磁盘，命名为.
4. 使用`cv2.VideoCapture()`创建捕获对象，从连接到计算机的第一个摄像头读取数据。
5. 使用`cv2.VideoCapture()`创建对象捕捉，从连接

到计算机的第一个摄像头读取，并打印  
`CAP_PROP_FRAME_WIDTH`属性。

6. 读取图像并将以同名加`_copy`的文件名保存到硬盘，如`logo_copy.png`。

7. 编写脚本

(`read_video_file_backwards_save_video.py`)，加载一个视频文件并向后播放，先播放视频最后帧，以此类推。