实验：Haar Cascades 人脸检测

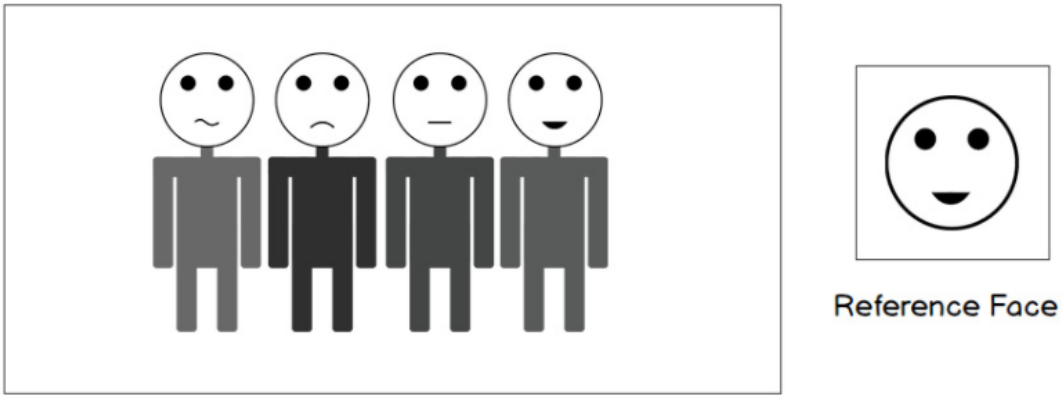
实验概要

人脸处理是人工智能领域的一个热点问题，因为利用计算机视觉算法可以从人脸中自动提取大量信息。人脸在视觉交流中扮演着重要的角色，因为从人脸中可以提取出大量的非语言信息，如：身份、意图、情感等。了解如何检测和跟踪图像和视频中的人脸，便能进一步执行各种过滤操作，例如，添加太阳镜滤镜，皮肤平滑，以及更多，以得出有趣和有意义的结果。无论是实现自拍时的自动对焦，还是在使用肖像模式拍摄照片时自动对焦，正面人脸检测都是构建这些应用程序的基础技术。

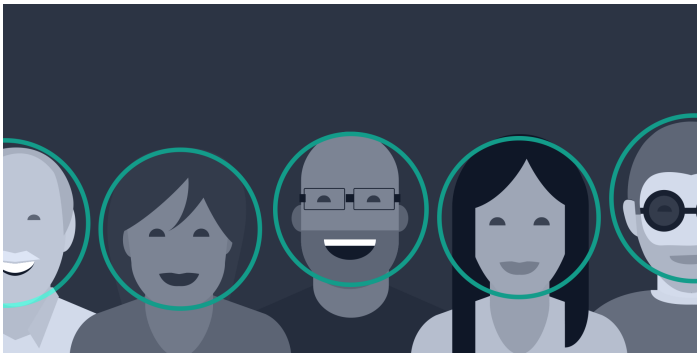
基于 Haar 特征的 Cascades（级联）分类器 - Haar Cascades

最直观简单实现人脸检测的方法，就是有一个窗口或块，将滑过输入图像，检测是否有人脸存在于该区域。

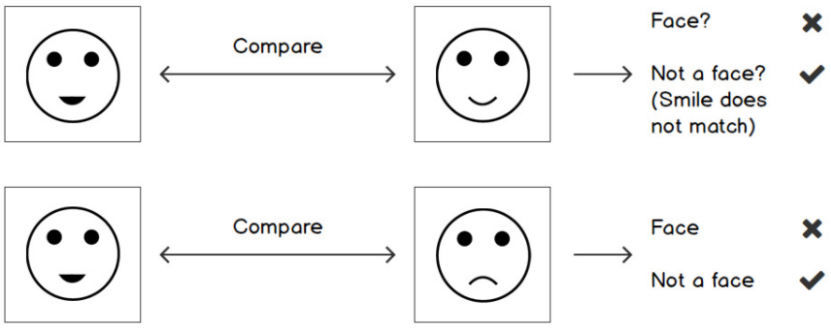
通过与样本人脸图像的比较，可以检测出该区域是否为人脸。



然而，正如以下这幅图像中的面孔是多种多样的。我们看一下用这种简单的面部检测的方法的问题：



如果我们要创建一个标准的脸，将其用作参考脸（Reference Face），用来检查图像中的一个区域是否存在人脸，这是很困难。不同的人，在肤色、面部结构、性别和其他面部特征上具有很大的差异。另一个问题是在比对图像的过程与比较数值不同 —— 你必须在两个图像中减去相应的像素值，然后看看这些值是否相似。假设有两张图像是相同的，但有不同的亮度。在这种情况下，像素值将非常不同。让我们考虑另一个场景，图像中的亮度是相同的，但比例（图像的大小）是不同的。处理这个问题的一个方法，是将两个图像调整为标准大小。加入考虑了所有的这些复杂性，通过使用基本图像处理步骤来实现良好的准确性，是非常困难的。



为了解决以上这些问题，Haar Cascades 应运而生。基于 Haar 特征的 Cascade 级联分类器是 Paul Viola 和 Michael Jone 在2001年的论文《Rapid Object Detection using a Boosted Cascade of Simple Features》(https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf)中提出的一种有效的物体检测方法。

本文提出了使用机器学习从图像中提取相关特征的方法，相关性是指可以通过特征在决定图像中是否存在目标对象时所起的作用来描述。作者还提出了一种称为 积分图 的技术，用于加快与图像中特征提取有关的计算。通过改变特征模板的大小和位置，可在图像子窗口中穷举出大量的特征。因此，此步骤的问题在于此过程将生成的功能数量众多。所以，著名的机器学习增强技术 AdaBoost 出现了。它可以在很大程度上减少特征的数量，并且只能从大量可能的特征中仅产生相关的特征。另外，我们还可以通过在视频中对其进行处理来进一步采取此操作。视频，简单而言是一个时间段内图像（称为帧）的幻灯片显示。例如：根据视频的质量/目的，它的范围从每秒 24 帧（fps）到 960 fps 不一。通过将 Haar Cascades 应用于每个帧，可以在视频中广泛使用它们实现特征提取。

像任何机器学习方法一样，生成此类级联的基本过程是基于数据的。这里的数据由两种图像组成。一组图像没有我们要检测的对象，而第二组图像中有对象。您拥有的图像越多，覆盖的对象越多，则级联效果越好。 如果您要构建正面人脸级联模型，则需要确保所有性别的图像数据的均衡性。 您还需要添加具有不同亮度的图像（即：在阴影或昏暗区域拍摄的图像与在明亮阳光下拍摄的图像）。为了获得更高的准确性，您还需要显示带有或不带有墨镜的面部图像 …… 依此类推。 其目的是要获得涵盖许多潜在可行参数的图像。

应用 Haar Cascades 实现人脸检测

了解了Haar Cascades背后的基本理论，让我们通过一个例子来了解我们如何使用 Haar Cascades 来检测图像中的人脸。

要获得相应的正面人脸检测模型，您需要下载 haarcascade_frontalface_default.xml 模型（在对大量有或没有人脸的图像进行训练后获得的预训练模型）。并且导入必要的库，分辨导入用于引用 OpenCV 库的 Python 封装器。以及流行的 NumPy 模块，该模块由于其高度优化的实现而广泛用于数值计算。

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

加载 Haar Cascade：

```
haarCascadeFace = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

在这里，我们使用了 `v2.CascadeClassifier` 函数，该函数仅使用一个输入参数：XML Haar Cascade 模型的文件名。这将返回一个 `CascadeClassifier` 对象。 `CascadeClassifier` 对象中有多种实现方法，但是我们仅关注 `detectMultiScale` 函数。

```
detectedObjects = cv2.CascadeClassifier.detectMultiScale(image, [scaleFactor,minNeighbors, flags, minSize, maxSize])
```

唯一必需的是 `image` ，该图像是要在其中检测面部的灰度输入图像。可选参数如下：

- `scaleFactor`： 决定每次迭代将图像调整大小的程度。这有助于检测输入图像中存在的不同大小的面部。
- `minNeighbors`： 用于确定检测目标时必须考虑的最小邻居的数量,默认为 `3` 。表明被检测出来的目标的旁边至少还需要有多少个类似的目标，其作用是避免模型将背景中的人脸也作为识别的对象。
- `flag`： 在新的分类器上面已经没有作用，主要是用于告诉分类器跳过平滑（无边缘区域）。
- `minSize` 和 `maxSize`： 指定要检测的脸部的最小和最大尺寸，超出此范围的人脸都会被忽略。

函数的输出是包含图像中存在的面的矩形（或边界框）列表。我们可以使用 OpenCV 的绘图函数 `cv2.rectangle` 在图像上绘制这些矩形。下图显示了需要遵循的步骤：

以下是使用 `scaleFactor = 1.2` 和 `minNeighbors = 9` 参数的正面人脸级联分类器来获得的效果：

将参数修改为 `scaleFactor = 1.2` 和 `minNeighbors = 5` 后，获得的结果下降了，因此，结果高度依赖于参数值。相关参数实现效果如下：

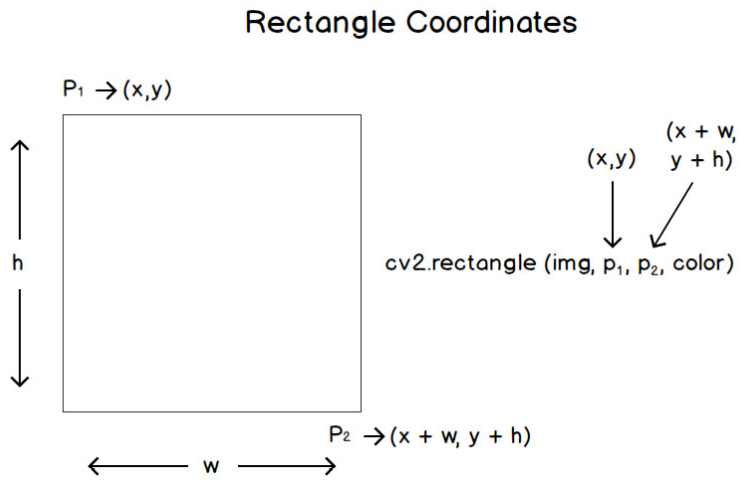
绘制识别边框

我们已经了解了使用 Haar Cascades 执行面部检测所需的函数，这里再复习一下之前先讨论一下关于边框的重要概念。

边框只不过是一个四个值的列表：x、y、w、h

`x` 和 `y` 表示边界框左上角的坐标（OpenCV 的原点在左上角）。 `w` 和 `h` 分别表示边界框的宽度和高度。

下图为 `cv2.rectangle（img，p1，p2，color）` 命令的示意图：



实验目标：

在本实验中，我们只会应用 **haarcascade_frontalface_default.xml** 模型进行正面人脸检测，这是一个大量存在人脸和不存在人脸的图像上训练后得到一个预训练模型。更多的预训练检测模型可以在这里下载 (https://github.com/opencv/opencv/tree/master/data/haarcascades)。

假如你需要找出一个班级的学生人数。一种方法是通过每小时去一趟教室，手动计算学生人数。我们也可以借助计算机视觉，将这一过程自动化。在本练习中，我们将使用 Haar Cascade 实现人脸检测。然后，通过计数检测到的面孔，我们将看到我们是否可以得到学生在班上的人数。观察应集中在发现的 `假阳性` 和 `假阴性` 。

在我们的场景中， `假阳性` 表示函数作为输出生成的边界框中没有人脸，而 `假阴性` 表示有人脸，但级联未能检测到它。

由于不需要大量的计算，因此它可以在边缘计算设备上很容易地进行。譬如：输入设备可以是一个非常简单的树莓派相机，考虑到学校的一节课通常持续一个小时，它可以每小时拍一张照片，例如下面这张图像：

1. 导入依赖库

本实验中，我们需要导入 CV2 与 Numpy 两个库。

同时，为了保证图像能在 Notebook 中正常显示，不能直接使用 OpenCV 自带的 `cv2.imshow` 命令，需要再导入matplotlib 库。

In [1]:

```
import cv2                          # 导入OpenCV
import numpy as np                  # 导入NumPy
import matplotlib.pyplot as plt     # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline
```

2. 指定输入图像路径

注意：确保可以根据图像的实际保存位置，指定路径，我们这里的路径是 `./data/face.jpg`

In [2]:

```
# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab5/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

inputImagePath = "./data/face.jpg" # 指定输入图像的路径
```

3. 指定 HaarCascade XML 文件的路径

请确保可以根据 HaarCascade XML 文件的实际保存位置，指定路径。我们这里的路径是 `./data/haarcascade_frontalface_default.xml`

In [3]:

```
haarCascadePath = "./data/haarcascade_frontalface_default.xml"
```

4. 加载图像

使用 cv2.imread() 函数，引用第 2 步中的图像路径 inputImagePath 加载图像

In [4]:

```
inputImage = cv2.imread(inputImagePath) # 加载输入图像
```

5. 将图像转换为灰度模式

使用 cv2.cvtColor() 函数，将图像从 BGR 模式转换为灰度模式

In [5]:

```
# 使用cv2将图像从BGR模式转换为灰度模式。
grayInputImage = cv2.cvtColor(inputImage, cv2.COLOR_BGR2GRAY)
```

6. 加载 HaarCascade 模型

In [6]:

```
haarCascade = cv2.CascadeClassifier(haarCascadePath)
```

7. 进行多尺寸人脸检测

使用 detectMultiScale() 函数，对输入图像执行多尺寸人脸检测

In [7]:

```
# 使用detectMultiScale()函数，进行多尺寸人脸检测
# 其中，“1.2”是scaleFactor，“1”是minNeighbors参数
detectedFaces = haarCascade.detectMultiScale(grayInputImage, 1.2, 1)
```

In [8]:

```
# 采用一个矩形边界框围绕在人脸四周，用来代表检测到的每一个人脸
# 这里，detectedFaces是覆盖图像中出现的面孔的边框列表。
# 因此，这里使用的 for 循环迭代每个检测到的人脸的边界框。
# 然后，我们使用cv2。矩形函数绘制边框。
# 为此,我们通过图像(inputImage)要绘制矩形,
# 矩形的左上角(x, y),矩形的右下角(x + w, y + h),
# 矩形的颜色红色=(0,0,255),和线厚度的矩形(3)作为参数。

for face in detectedFaces:
  x, y, w, h = face
  cv2.rectangle(inputImage, (x, y), (x+w, y+h), (0, 0, 255), 3)
```

8. 显示图像检测结果

- `cv2.imshow()` 函数：将在一个名为“Faces Detected”的弹出窗口中显示 `inputImage` 变量所指定的输入图片
- `cv2.waitKey()` 函数：显示图像的持续时间。我们这里使用了 `0` 指等待无限的时间，直到按下任意键
- `cv2.destroyAllWindows()` 函数：关闭所有窗口并退出程序

In [9]:

```
# cv2.imshow("Faces Detected", inputImage)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

注意： 以上代码只在本地运行 OpenCV 的时候启效，由于设计的架构不一致，在云计算端 Notebook 环境中执行会造成 Kernel 崩溃。

因此，使用 Matplotlib 取代 `cv2.imshow()` 函数将图像直接显示在 Notebook 中。代码如下 ——

In [10]:

```
# 借助matplotlib库来使检测结果在Notebook上显示
import matplotlib.pyplot as plt

# 使用cv2加载图片并进行处理
# 由于opencv默认以BGR格式加载，因此我们调整参数，以RGB格式显示
plt.imshow(cv2.cvtColor(inputImage, cv2.COLOR_BGR2RGB))

plt.show()
```

另外，你可以使用 `cv2.imwrite()` 函数保存图像，以便日后浏览。

我们可以看到，尽管我们能够检测出图像中的大多数面孔，我们仍然有两个 `假阳性` 和五个 `假阴性` 。

实验小结

在本实验中，您了解了基于 Haar 特征的 Cascades（级联）分类器的基本工作原理，如何在 Python 中调用 Haar Cascade 的正面人脸检测预训练模型对图像进行人脸识别。我们使用 `1.2` 的比例因子 `scaleFactor` 和 `3` 的最小邻居参数。建议您尝试对这些参数使用不同的值，并了解它们对获得的结果的影响，观测函数输出生成的边界框中的不同结果。

你可以看到，通过调节不同的参数值，以下图像没有误报。检测到的所有边界框都有一个面孔。但同时，这里检测到的面部数量也大大减少了：

再对比下面这张图像，已成功检测到图像中的所有面部，但是误报的数量已大大增加：

在计算机视觉中与检测、追踪相关的问题存在一个非常普遍的挑战，称为遮挡（Occlusion）。 遮挡是指某个对象被图像中的另一个对象部分或完全覆盖。譬如：您尝试自拍照，但有人来站在您面前，那么您的图像将被第二个人遮挡。

您可以和您的朋友一起拍摄两个图像，第一个图像中，你们的两张脸都应该清晰可见；而在第二个图像中，您的脸部被朋友的脸部分遮盖。请使用我们在本练习中使用的正面人脸检测 Haar Cascade 对这两个图像进行识别，并查看 Haar Cascade 是否可以检测到您被遮挡的脸。在实际工作中，应该始终对任何与检测相关的模型执行此类测试，以了解它们与现实场景的适应性。