实验：Haar Cascades 眼睛检测

实验概要

使用 Haar Cascades 检测面部独立部分

在之前的实验中，我们讨论了人脸检测及其重要性。我们还关注了 Cascades 以及如何将它们用于对象检测问题。另外，我们提及但没有详细介绍的一点，是 Haar Cascades 可以扩展为检测其他对象，而不仅仅是面部。如果您再看一下训练 Cascades 模型的方法，您会注意到这些步骤非常通用，因此可以扩展到任何对象检测问题。然而，这需要获取大量图像，然后根据对象的存在与否将它们分为两类，最后训练模型，这是一个计算量很大的过程。重要的是，考虑到此模型仅受过训练以检测脸部，因此我们不能使用相同的 haarcascade_frontalface_default.xml 文件来检测其他对象，**它仅适用于正面人脸检测。**OpenCV 的开源存储库 (https://gitee.com/mirrors/opencv/tree/master/data/haarcascades)，具有各种各样独立的 Cascades 模型，用于检测猫，正面，左眼，右眼，微笑等对象。本实验中，我们将看到如何使用这些级联来检测面部的一部分，而不是整个人脸。

其中，检测到诸如眼睛之类的面部部分可能很重要。假如您要构建一个非常基础的基于虹膜识别的生物特征安全软件 —— 基于人类的虹膜比指纹独特得多的事实。因此，虹膜识别是一种更好的生物统计安全手段。现在，假设您在办公室的正门装有高分辨率摄像头，则眼睛 Cascades 模型将能够在视频帧中存在的眼睛周围创建边界框。然后可以进一步处理检测到的眼睛区域，以进行虹膜识别，这样的场景在电影中很常见。

1. 要下载相关模型，我们将访问 OpenCV 的开源存储库 (https://gitee.com/mirrors/opencv/tree/master/data/haarcascades)，当前存储库中提供了用于检测眼睛，左眼，右眼，微笑等的模型。
2. 下载模型后，其余过程将与之前的人脸识别实验一样，首先读取输入图像。
3. 由于 Cascades 模型仅适用于灰度图像，因此必须将图像转换为灰度图像。
4. 加载级联分类器模型（XML文件）。
5. 唯一的新步骤是多尺度物体检测。

Haar Cascades 最方便的地方是您可以每次都使用相同的 OpenCV 函数。我们可以创建一个函数，该函数将完成所有的步骤并返回边界框列表。虽然，遵循的步骤保持不变，同时参数的重要性也大大提高。在执行了前面的实验之后，您知道通过更改 `minNeighbors` 最小邻居数和 `scale factor` 比例因子，可以改变 Cascades 的性能。具体而言，在我们试图检测较小的部分（例如:眼睛）的情况下，这些因素变得更加重要。计算机视觉工程师有责任确保已进行适当的实验,以提出可得出最佳结果的参数。在接下来的实验中，我们将看到如何检测图像中的眼睛，并且还将执行必要的实验评估以获取最佳参数。

实验目标

在本实验中，我们将使用 haarcascade_eye.xml 级联模型执行眼睛检测。我们将使用以下的图片：

1. 导入依赖库

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. 加载图像与模型

指定输入图像和 haarcascade_eye.xml 文件的路径：

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/eyes.jpeg"

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

3. 创建自定义函数

创建一个自定义函数 detectionUsingCascades 以执行所有步骤，创建自定义函数的目的是为了更灵活的应用 Haar Cascades 所提供的模型。之后，我们只需要更换输入图像和模型文件的路径，便能快速的使用 Haar Cascades 检测其他目标的模型用于检测新的图像。

In [3]:

```
# 这是一个自定义函数，负责使用 haarCascade 模型进行目标检测
# 该函数以图像文件和Cascades 模型文件的路径，
# 作为输入，并返回围绕检测到的对象实例的边框
def detectionUsingCascades(imageFile, cascadeFile):

    # Step 1 - 加载图片
    # 使用cv2.imread函数加载图像
    image = cv2.imread(imageFile)

    # Step 2 - 将图片从BGR转换为灰度
    # 这里要注意的一点是，如果图像已经是灰度图像，则不应再对其进行转换
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Step 3 - 加载 haarCascade模型
    # 如前所述，cv2.CascadeClassifier函数将XML文件的路径作为输入
    # 而该路径，正好是我们的自定义函数 detectionUsingCascades 的第二个输入参数
    haarCascade = cv2.CascadeClassifier(cascadeFile)

    # Step 4 - 执行多尺度检测
    # 这里，重要的是要注意 detectedObjects 只是一个检测到的对象的边框列表
    # 另外，注意我们已经静态指定了参数的值 - scaleFactor和minNeighbors（1.2, 2）
    # 您可以手动调整这些值以获得最佳结果(我们尝试的最佳结果是调整为1.1, 35)
    # 更完善的应用程序将显示不同参数值的一组结果，并让用户从多个结果中选择一个
    detectedObjects = haarCascade.detectMultiScale(gray, 1.2, 2)

    # Step 5 - 勾画检测边框
    # 使用cv2.rectangle函数绘制边界框：
    for bbox in detectedObjects:
        # 每个 bbox 都是一个矩形，表示被检测对象周围的边框
        x, y, w, h = bbox
        cv2.rectangle(image, (x, y), (x+w, y+h), (0, 0, 255), 3)

    # Step 6 - 输出显示
    # 显示带有在检测到的对象上绘制的边框的图像：

    #cv2.imshow("Object Detection", image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # 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
    # 完成转换后，显示图片
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.show()

    # Step 7 - 返回边框
    return detectedObjects
```

3. 运行自定义函数

由于在自定义函数之前，我们就已经加载了图像与模型的路径。因此，我们可以直接将 `inputImagePath` 和 `haarCascadePath` ，作为自定义函数 `detectionUsingCascades()` 的两个输入参数 ——

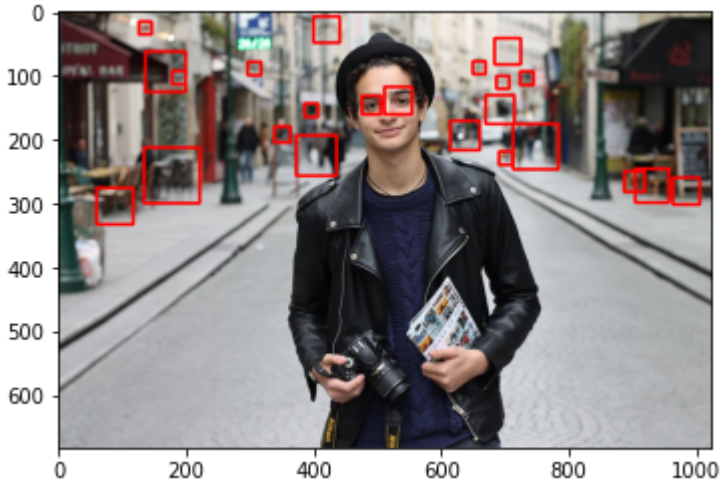
```
inputImagePath = "./data/eyes.jpeg"
haarCascadePath = "./data/haarcascade_eye.xml"
```

以下代码即：使用 `./data/haarcascade_eye.xml` 模型来检测 `./data/eyes.jpeg` 图像。

您可以在这里将模型更换为任意一个 Haar Cascades 模型，将图像更换为任意一张您喜欢的图像。

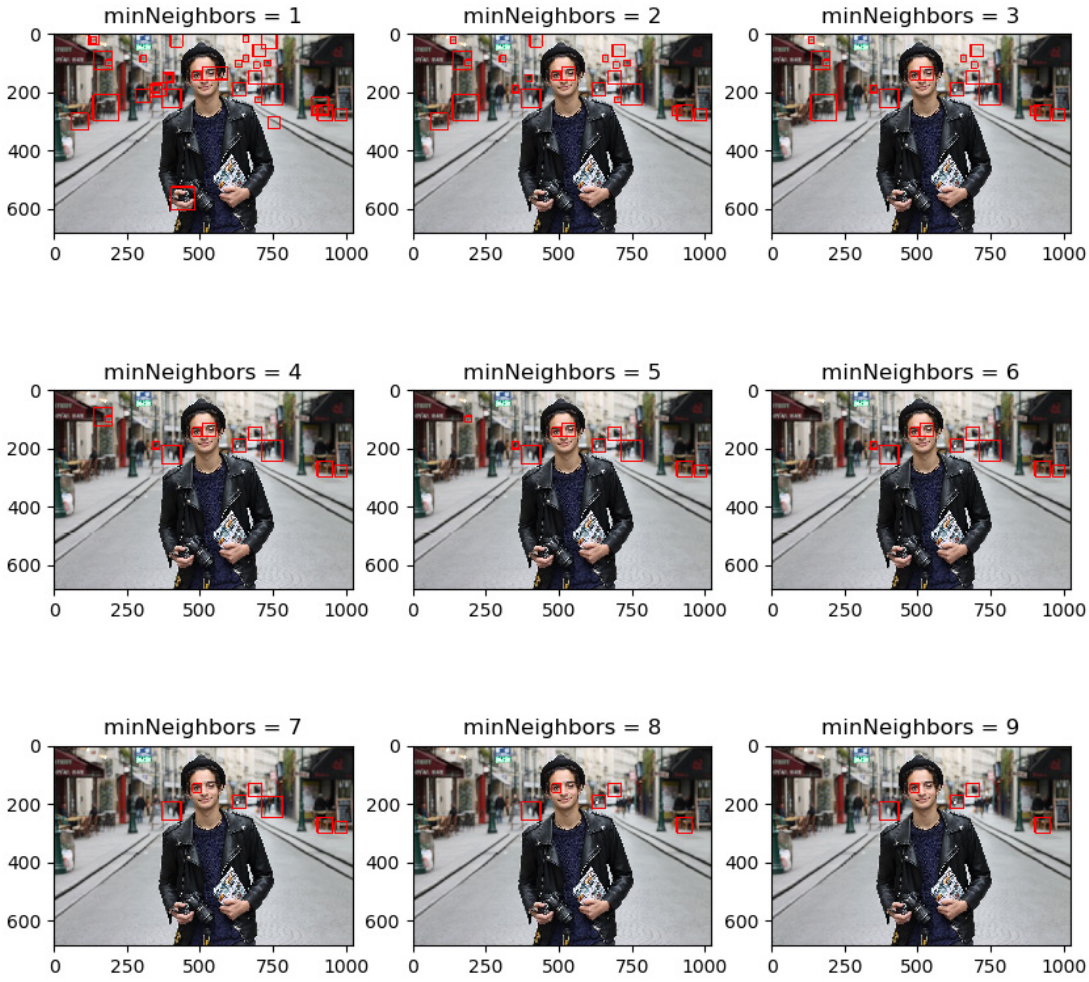
In [4]:

```
eyeDetection = detectionUsingCascades(inputImagePath, haarCascadePath)
```



4. 调试参数

如您所见，尽管已经检测到眼睛，但是在前面的结果中有许多误报。后面我们还需要进行必要的实验以改善结果。以下为更改 `minNeighbors` 参数后的结果：



实验小结

实验小结

在本实验中，您掌握了如何使用更多的 Haar Cascades 模型实现对图像中人脸其他部位的检测。另外，您了解了使用自定义函数，在只需要替换模型与图像路径两个参数的前提下，实现任意 Haar Cascades 检测模型在任意一张图片上的推理。

让我们总结一下 Haar Cascades 缺点。 Haar Cascades 非常适合在边缘执行的操作，这些 Cascades 模型与英伟达 Jetson 等边缘设备配合使用，可以以很高的精度和非常快的速度提供结果，对于视频，它们也具有很高的 Frames/Per Ssecond。在这种情况下，为什么行业和研究人员转向基于深度学习的对象检测解决方案？ 深度学习模型需要更长的时间进行训练，需要更多的数据量，并且在进行训练时（有时甚至在推理过程中）会消耗大量内存。

原因之一，是深度学习模型提供的更高的准确性。