第 10 章 OpenCV 中的深度学习

10.1 使用 Caffe 与 tensorflow 图像分类

在前面的内容中，我们学习了如何使用 HOG 与 SVM 训练一个图像分类器实现对图像中对象有无的判断，本节我们将讨论如何在 OpenCV 中使用基于深度学习的图像识别模型实现。这些训练好的模型都为了 ILSVRC(ImageNet 大规模视觉识别挑战赛)。

什么是 ImageNet 与 ILSVRC

ImageNet 项目的目标是为研究工作提供大规模的图像数据集，它包含 1400 万张图像，超过 20000 个类别。其中 100 万张图像提供了矩形框标注，作为对象检测任务数据集。需要注意的是，它只是提供这些图像的 URL，需要你自己下载这些图像。

自从 2010 开始，ImageNet 大规模视觉识别挑战赛就是一项由 ImageNet 团队主导的年度竞赛，不同研究团队可以使用 ImageNet 数据集去评估他们的算法在计算机视觉任务-图像分类与对象检测上的表现。训练使用的数据集实际上 ImageNet 的一个子集，有 120 万张图像总计有 1000 个分类。

Alex 与他的团队采用深度学习的方法在 2012 年的比赛在之前的基础识别率提升 11%而赢得冠军，后续几年对象识别精度不断提高已经达到人脸的认知水平。

这些最佳模型的作者非常好，分享了他们的模型，后续的研究者也继续分享风格。所以在开源社区很容易就会发现这些最佳模型。

深度学习不是本课程的目标，在 OpenCV 中，我们将会简单的学习如何使用一些已经广为人知的深度学习模型，学习如何调用来自不同深度学习框架如 tensorflow、caffe 的模型。

OpenCV 深度神经网络模块

使用 OpenCV 深度神经网络模块，我们就可以调用来自不同深度学习框架训练好的模型，实现对输入图像的推理计算，预测分类、对象检测等计算机视觉任务。OpenCV 中使用深度神经网络模型主要有下面这些步骤

1. 加载网络，使用 readNet 函数
2. 把输入图像转换为张量，使用 blobFromImage
3. 设置网络的输入张量数据
4. 使用 forward 完成推理，获得输出
5. 解析/处理输出

下面让我们更加详细介绍一下每一步的主要函数

- **加载网络**

对不同的深度学习框架的模型，网络加载使用不同的函数完成。例如，tensorflow 模型通过 readNetFromTensorflow()函数读取，caffe 模型通过 readNetFromCaffe()读取。从 OpenCV3.4.1

开始，readNet()函数支持从参数与加载的网络自动判别模型对应的框架。
Tensorflow

```
Net cv::dnn::readNetFromTensorflow (
    const String &  model,
    const String &  config = String()
)
```

model 网络权重文件
config 网络定义文件

Caffe

```
Net cv::dnn::readNetFromCaffe(
    const String &  prototxt,
    const String &  caffeModel = String()
)
```

prototxt 网络定义文件
caffeModel 网络权重文件

注意：tensorflow 跟 caffe 两个参数顺序相反

- **转换图像为张量**

```
Mat cv::dnn::blobFromImage(
    InputArray  image,
    double  scalefactor = 1.0,
    const Size &  size = Size(),
    const Scalar &  mean = Scalar(),
    bool  swapRB = false,
    bool  crop = false,
    int  ddepth = CV_32F
)
```

从图像创建四维的张量数据，可以选择放缩、剪切、减去均值，交换 R 与 B 通道，值尺度化等操作。参数解释如下：
image 表示输入的单通道、三通道、四通道图像
scalefactor 像素值要乘的数值
size 表示大小
mean 表示减去的均值
swapRB 是否交换 R 与 B 通道
crop 是否剪切
ddepth 表示数据输出类型，默认浮点数

- **生成输出**

```
outputBlobs = net.forward([, outputBlobs[, outputName]])
```

该方法完成一次推理，根据输出层名称计算输出，参数解释
outputBlobs 包含所有输出层的 blob 对象
outputNames 所有输出层的名称

下面是一个使用模型识别熊猫的例子

使用 Caffe 模型

下面使用 Caffe 网络上训练的 googlenet 网络实现图像分类，首先看实现步骤

1. 加载网络与图像，把图像转换为 blob 对象，然后设置网络输入，forward 到输出层
2. 输出矩阵每个类别序号对应含有一个置信得分表示图像中是否含有该对象
3. Argmax 会从输出矩阵中找到最有可能的置信度预测
4. 输出的标签字符串从 lassification_classes_ILSVRC2012.txt 文件中获得

代码实现如下：

```
Mat frame = imread("D:/panda.jpg");
std::vector<std::string> classes;
ifstream ifs(classFile.c_str());
string line;
while (getline(ifs, line)) {
    classes.push_back(line);
}

Net net = readNetFromCaffe(protoFile, weightFile);

// 转换输入与推断
Mat blob;
blobFromImage(frame, blob, 1.0, Size(224, 224), Scalar(104, 117, 123), false, false);
net.setInput(blob);
Mat prob = net.forward();

// 解析输出
Point classIdPoint;
double confidence;
minMaxLoc(prob.reshape(1, 1), 0, &confidence, 0, &classIdPoint);
int classId = classIdPoint.x;

// 显示输出
string label = format("Predicted Class : %s, confidence : %.3f", (classes[classId].c_str()),
               confidence);
cout << label << endl;
putText(frame, label.c_str(), Point(10, 50), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0, 0, 255), 2, 8);
imshow("image classification", frame);
waitKey(0);
```

运行结果如下：

图 10.1-1

使用 tensorflow 模型

这个例子演示如何使用 tensorflow 版本的 googlenet 调用实现图像分类

Caffe Vs. Tensorflow

这个例子跟之前 caffe 版本 googlenet 调用的不同点主要有如下：

1.blobFromImage 函数中 swapRB 标志设置为真，这是因为 tensorflow 使用 RGB 格式而 Caffe 使用 BGR 格式。
2.分类标签的映射文件不同，tensorflow 使用 imagenet_comp_graph_label_strings.txt
3.均值参数不一样
4.没有配置文件，只有冻结模型可以使用，所有的信息都来自模型本身
实现的演示代码如下：

```
string weightFile = "D:/projects/opencv_tutorial/data/models/inception5h/tensorflow_inception_graph.
                    pb";
string classFile = "D:
                   /projects/opencv_tutorial/data/models/inception5h/imagenet_comp_graph_label_string
                   s.txt";
Mat frame = imread("D:/panda.jpg");
std::vector<std::string> classes;
ifstream ifs(classFile.c_str());
string line;
while (getline(ifs, line)) {
    classes.push_back(line);
}

Net net = readNetFromTensorflow(weightFile);

// 转换输入与推断
Mat blob;
blobFromImage(frame, blob, 1.0, Size(224, 224), Scalar(117, 117, 117), true, false);
net.setInput(blob);
Mat prob = net.forward();

// 解析输出
Point classIdPoint;
double confidence;
minMaxLoc(prob.reshape(1, 1), 0, &confidence, 0, &classIdPoint);
int classId = classIdPoint.x;

// 显示输出
string label = format("Predicted Class : %s, confidence : %.3f", (classes[classId].c_str()),
               confidence);
cout << label << endl;
putText(frame, label.c_str(), Point(10, 50), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0, 0, 255), 2, 8);
imshow("image classification", frame);
waitKey(0);
```

10.2 深度学习对象检测

本节我们将学习如何在 OpenCV 中使用深度学习对象检测模型。OpenCV 已经实现了对 Faster-RCNN、R-FCN、SSD、Mask-RCNN 这些对象检测模型的调用支持，而且还提供了脚本把 tensorflow 的这些模型转换为 OpenCV 格式，因此开发者可以首先训练这些模型，然后在 OpenCV 完成调用。

10.2.1 SSD 对象检测

我们将讨论基于 tensorflow 对象检测 API 训练的 SSD 对象检测，这些 SSD 对象检测模型支持 MobileNet 与 Inception 深度神经网络作为基础网络。使用 OpenCV DNN 模块调用对象检测模型实现对象检测步骤如下：

1. 加载模型与输入图像
2. forward 网络完成检测推理
3. 解析与显示检测结果

首先，读取 tensorflow 模型

```
cv::dnn::Net net = cv::dnn::readNetFromTensorflow(modelFile, configFile);
```

然后读取分类标签

```
ifstream ifs(classFile.c_str());
string line;
while (getline(ifs, line)) {
    classes.push_back(line);
}
```

对象检测

首先预处理输入图像，然后作为参数传给网络，使用 forward 方法完成网络推理，最后返回得到输出结果

```
cv::Mat inputBlob = cv::dnn::blobFromImage(frame, inScaleFactor, cv::Size(inWidth, inHeight),
                    meanVal, true, false);
net.setInput(inputBlob);
cv::Mat detection = net.forward("detection_out");
cv::Mat detectionMat(detection.size[2], detection.size[3], CV_32F, detection.ptr<float>());
```

解析与显示结果

循环所有的检测对象，绘制检测矩形框，显示检测对象的类别名称

```
// For every detected object
for (int i = 0; i < objects.rows; i++) {
    int classId = objects.at<float>(i, 1);
    float score = objects.at<float>(i, 2);

    // Recover original cordinates from normalized coordinates
    int x = static_cast<int>(objects.at<float>(i, 3) * img.cols);
    int y = static_cast<int>(objects.at<float>(i, 4) * img.rows);
    int w = static_cast<int>(objects.at<float>(i, 5) * img.cols - x);
    int h = static_cast<int>(objects.at<float>(i, 6) * img.rows - y);

    // Check if the detection is of good quality
    if (score > threshold) {
        int baseLine;
        cv::Size textSize = cv::getTextSize(text, cv::FONT_HERSHEY_SIMPLEX, 0.7, 1, &baseLine);
        // Use text size to create a black rectangle
        rectangle(img, Point(x,y-textSize.height-baseLine), Point(x+textSize.width,y+baseLine),
                  Scalar(0,0,0),-1);
        // Display text inside the rectangle
        putText(img, text, Point(x,y-5), FONT_HERSHEY_SIMPLEX, 0.7, Scalar(0,255,255), 1, LINE_AA);
        rectangle(img, Point(x,y), Point(x+w, y+h), Scalar(255,255,255), 2);
    }
}
```

运行结果

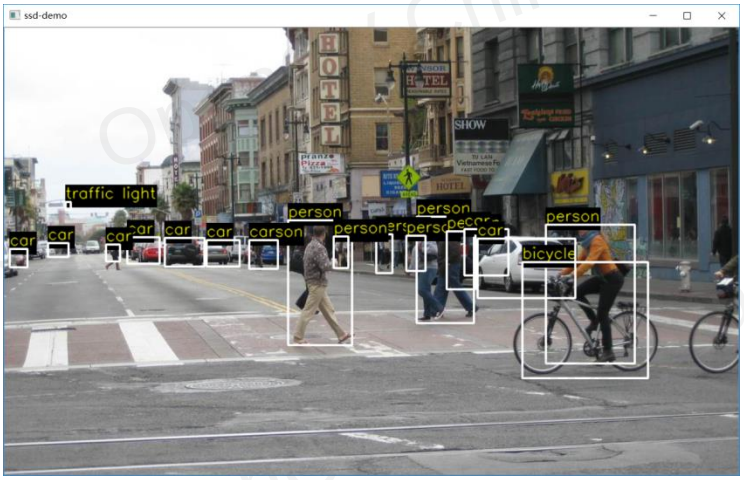

图 10.2.1-1

使用不同的 tensorflow 模型

如果你想在 OpenCV 中使用不同的 tensorflow 模型，直接看下面的链接即可：

1.浏览 tensorflow 模型库

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

2.下载本例中的模型文件

http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz

下载之后，解压缩的模型内部文件结构

```
ssd_mobilenet_v2_coco_2018_03_29
|— checkpoint
|— frozen_inference_graph.pb
|— model.ckpt.data-00000-of-00001
|— model.ckpt.index
|— model.ckpt.meta
|— pipeline.config
|— saved_model
|—— saved_model.pb
|—— variables
```

其中 forzen_inference_graph.pb 是 OpenCV 需要加载的模型权重文件。OpenCV 中使用模型，你还需要通过运行 tf_text_graph_ssd.py 该脚本生成*.pbtxt 配置文件

10.2.2 YOLO 对象检测

本节我们将学习在 OpenCV 如何使用 YOLOv3-另外一个最佳的对象检测模型

YOLOv3 是对象检测算法 **YOLO – You Only Look Once** 的最新版本，公开发布的模型支持视频与图像中 80 个种类的识别，最重要的是执行速度很快，精度与 SSD 对象检测算法相当。

为什么要在 OpenCV 中使用 YOLO

下面是一些在 OpenCV 中使用 YOLOv3 的原因

1. 容易在 OpenCV 应用中集成，如果你已经使用 OpenCV 做应用开发，现在想简单使用 YOLOv3，就无需编译与打包整个 darknet 框架。
2. OpenCV 实现的 CPU 版本调用比普通 CPU 计算快 9 倍，原因在于 OpenCV DNN 模块代码实现更加优化。例如 darknet 框架使用 OpenMP 单张图像推断需要 2 秒，而使用 OpenCV 只需要 0.22 秒
3. Python 语言支持，Darknet 框架是基于 C 实现的，官方是不支持 Python 语言的，但是基于 OpenCV，darknet 网络很容易完成 Python 调用。

YOLOv3 在在 Darknet 与 OpenCV 上的速度测试对比

下面的表数据显示了 YOLOv3 Darknet 与 OpenCV 在性能上的对比结果，在所有的测试对比中输入的图像都是 416x416 大小，不要惊讶，GPU 版本的 Darknet 的性能永远都是杰出表现，也不要惊讶 Darknet+OpenMP 版本的表现一直会比没有 OpenMP 的好，原因是 OpenMP 启用了多处理器。但是让人惊讶的是 OpenCV DNN 版本的 CPU 运行速度居然好过 Darknet+OpenMP。

OS	Framework	CPU/GPU	Time(ms)/Frame
Linux 16.04	Darknet	12x Intel Core i7-6850K CPU @ 3.60GHz	9370
Linux 16.04	Darknet + OpenMP	12x Intel Core i7-6850K CPU @ 3.60GHz	1942
Linux 16.04	OpenCV [CPU]	12x Intel Core i7-6850K CPU @ 3.60GHz	220
Linux 16.04	Darknet	NVIDIA GeForce 1080 Ti GPU	23
macOS	DarkNet	2.5 GHz Intel Core i7 CPU	7260
macOS	OpenCV [CPU]	2.5 GHz Intel Core i7 CPU	400

Table 1: Speed Test of YOLOv3 on Darknet vs OpenCV

图 10.2.2-1

OpenCV 使用 YOLOv3 对象检测

Step 1. 初始化参数

YOLOv3 算法会生成检测框作为预测输出，每个检测框都会关联到一个置信度得分。在第一阶段，所有置信度得分低于阈值的检测框将会被丢去而不进行下一步处理。

下一步就是对剩下的检测框进行非最大抑制处理去除重叠检测框。非最大抑制通过一个参数 nmsThreshold 来控制，你可以尝试不断改变该参数，看看随之变换的检测框数目。

默认的模型输入图像的大小为 416x416，也可以改为 320 可以运行得更快，或者改为 608 获得更高得检测精度支持。

初始化参数得代码如下：

```
// 初始化参数
float objectnessThreshold = 0.5;
float confThreshold = 0.5;
float nmsThreshold = 0.4;
int inpWidth = 416;
int inpHeight = 416;
vector<string> classes;
Mat frame = imread("D:/bird.jpg");
```

Step 2. 加载模型与分类标签

文件 coco.names 包含所有模型训练得所有对象标签，读取每个标签名称，然后加载 YOLOv3 模型，模型分为两个部分

- yolov3.weights 预训练权重文件
- yolov3.cfg 配置文件

```
// 加载网络
String config = MODEL_PATH + "yolov3.cfg";
String weights = MODEL_PATH + "yolov3.weights";
Net net = readNetFromDarknet(config, weights);
```

这里设置 OpenCV 计算设备为 CPU，计算后台为 OpenCV DNN，你也可以设置为 cv.dnn.DNN_TARGET_OPENCL，意思是使用 intel 得 GPU 加速，如果你没有 intel 的 GPU 硬件支持，会自动切换到 CPU 计算。

Step 3. 推理与处理输出

输入图像转换为 blob 之后，执行 forward 得到输出层数据，因为 YOLOv3 有多个输出层，在 OpenCV DNN 要想支持 forward 到那些输出层了，需要调用函数 getUnconnectedOutLayersNames ()获得所有的输出层，该函数返回的是输出层名称 vector

```
// 设置输入
Mat blob;
blobFromImage(frame, blob, 1.0 / 225.0, Size(416, 416), Scalar(0, 0, 0), true, false);
net.setInput(blob);

// infernece
vector<string> outNames = net.getUnconnectedOutLayersNames();
vector<Mat> outs;
net.forward(outs, outNames);
```

YOLOv3 的输出层处理比 SSD 稍微复杂点。执行推理，得到生成的检测框，经过非最大抑制之后，就可以绘制检测框，显示对象检测结果了，其中解析输出结果，是根据输出向量结构，YOLOv3 的输出是每个检测框生成一行向量数据，表示为 一个 vector 数据是各个分类标签的置信度 +5 个元素，这五个元素前面四个分别是 center_x、center_y、width 与 height，第五个是检测框对象的置信度得分。

```
// post-process
vector<int> classIds;
vector<float> confidences;
vector<Rect> boxes;

for (size_t i = 0; i < outs.size(); ++i) {
    // 解析输出
    float* data = (float*)outs[i].data;
    for (int j = 0; j < outs[i].rows; ++j, data += outs[i].cols) {
        float t = outs[i].at<float>(j, 4);
        if (t > objectnessThreshold) {
            Mat scores = outs[i].row(j).colRange(5, outs[i].cols);
            Point classIdPoint;
            double confidence;
            // Get the value and location of the maximum score
            minMaxLoc(scores, 0, &confidence, 0, &classIdPoint);
            if (confidence > confThreshold) {
                int centerX = (int)(data[0] * frame.cols);
                int centerY = (int)(data[1] * frame.rows);
                int width = (int)(data[2] * frame.cols);
                int height = (int)(data[3] * frame.rows);
                int left = centerX - width / 2;
                int top = centerY - height / 2;

                classIds.push_back(classIdPoint.x);
                confidences.push_back((float)confidence);
                boxes.push_back(Rect(left, top, width, height));
            }
        }
    }
}
```

剩下 vector 是每个类别的置信度得分。检测框内的对象类别属于得分最高的类别。检测框本身的置信度得分如果小于指定的阈值则被丢弃，不会进一步处理，所有大于阈值的检测框会进行非最大抑制处理之后得到最终的输出检测框。

```
// 非最大抑制
vector<int> indices;
NMSBoxes(boxes, confidences, confThreshold, nmsThreshold, indices);

// 显示输出
for (size_t i = 0; i < indices.size(); ++i) {
    int idx = indices[i];
    Rect box = boxes[idx];
    rectangle(frame, Point(box.x, box.y), Point(box.x + box.width, box.y + box.height), Scalar(255,
              178, 50), 3);

    // 获取分类标签与得分
    string label = format("%.2f", confidences[idx]);
    if (!classes.empty()) {
        label = classes[classIds[idx]] + ":" + label;
    }

    // 显示对象标签
    int baseLine;
    Size labelSize = getTextSize(label, FONT_HERSHEY_SIMPLEX, 0.5, 1, &baseLine);
    box.y = max(box.y, labelSize.height);
    rectangle(frame, Point(box.x, box.y - round(1.5*labelSize.height)),
              Point(box.x + round(1.5*labelSize.width), box.y + baseLine), Scalar(255, 255, 255),
                    FILLED);
    putText(frame, label, Point(box.x, box.y), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0, 0, 0), 1);
}
imshow("YOLOv3-Object Detection Demo", frame);
```

运行结果如下：

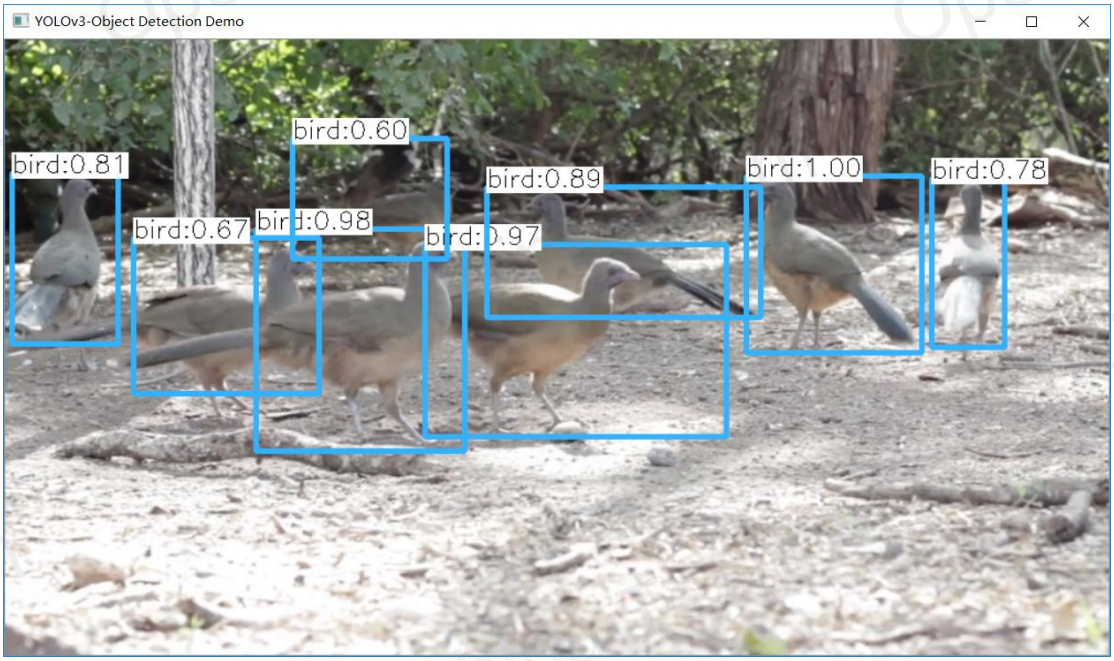

图 10.2.2-2

10.3 基于深度学习的人脸检测

OpenCV 中深度学习人脸检测模型在 OpenCV3.3 中开始提供，它是 SSD 对象检测网络基于 ResNet-10 作为基础网络实现。模型训练主要是是基于网络图像完成。OpenCV 提供了两个人脸检测模型

- 半精度的 16 浮点数模型，基于 Caffe 框架实现
- 八位量化模型，基于 tensorflow 实现

下面是基于两个模型实现人脸检测的 OpenCV 程序演示部分。

代码解释：

首先加载图像或者视频文件，然后把图像或者视频帧转换位 300x300 大小的 blob 对象，调

用 forward 函数实现网络推理预测，输出的是一个 4-D 的矩阵，其中

- 第三维开始是每个检测框的数据，即人脸检测结果
- 第四维包含的是每个检测框位置信息跟它的置信度得分，

输出的检测框坐标位置被归一化在 0~1 之间，得到输出的检测框位置信息，必须乘以图像的实际宽高，才最终转得到正确的像素坐标位置。

```
Mat frame = imread("D:/images/lena.jpg");
int frameHeight = frame.rows;
int frameWidth = frame.cols;
cv::Mat inputBlob = cv::dnn::blobFromImage(frame, 1.0, cv::Size(300, 300),
                    Scalar(104.0, 177.0, 123.0), false, false);

net.setInput(inputBlob, "data");
cv::Mat detection = net.forward("detection_out");
cv::Mat detectionMat(detection.size[2], detection.size[3], CV_32F, detection.ptr<float>());
for (int i = 0; i < detectionMat.rows; i++) {
    float confidence = detectionMat.at<float>(i, 2);

    if (confidence > 0.5) {
        int x1 = static_cast<int>(detectionMat.at<float>(i, 3) * frameWidth);
        int y1 = static_cast<int>(detectionMat.at<float>(i, 4) * frameHeight);
        int x2 = static_cast<int>(detectionMat.at<float>(i, 5) * frameWidth);
        int y2 = static_cast<int>(detectionMat.at<float>(i, 6) * frameHeight);

        cv::rectangle(frame, cv::Point(x1, y1), cv::Point(x2, y2), cv::Scalar(0, 255, 0),
                      frameHeight / 150, 8);
    }
}
imshow("DNN Face-Detection Demo", frame);
```

运行结果如下：

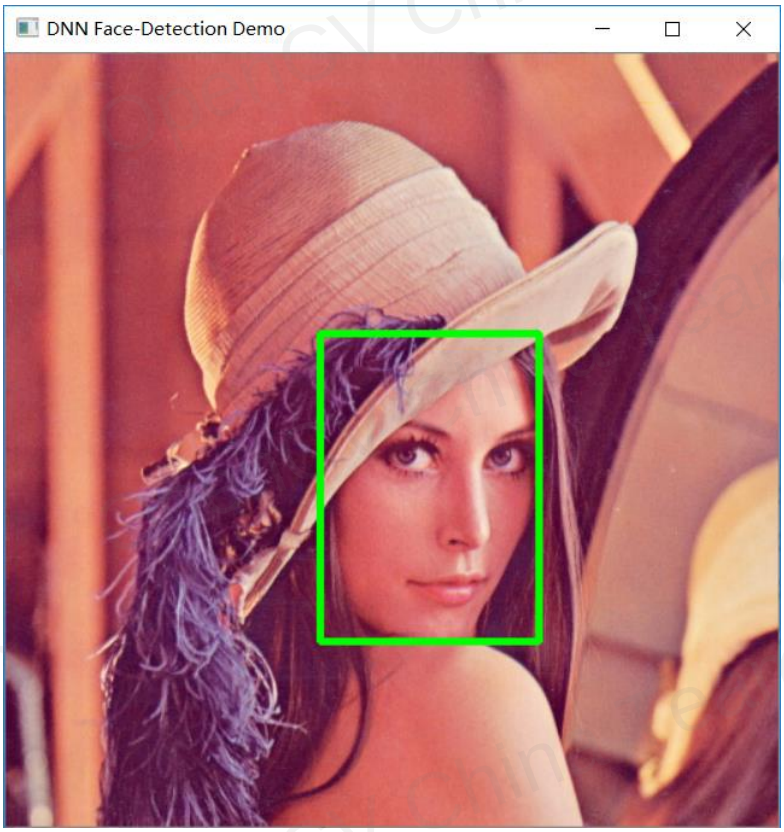

图 10.3-1

10.4 人体姿态评估

使用 OpenPose 实现人体姿态评估，OpenPose 是一个实时多人姿态检测开源框架，这里 OpenCV 通过调用 OpenPose 预训练的模型，实现人体姿态评估。这些模型主要是基于 Caffe 框架完成训练的，Caffe 模型一般有两个文件

- .prototxt 文件是模型定义文件
- .weights 文件是模型权重文件

首先需要加载模型文件，代码如下：

```
const int POSE_PAIRS[14][2] = {
    { 0,1 },{ 1,2 },{ 2,3 },
    { 3,4 },{ 1,5 },{ 5,6 },
    { 6,7 },{ 1,14 },{ 14,8 },{ 8,9 },
    { 9,10 },{ 14,11 },{ 11,12 },{ 12,13 }
};
string protoFile = MODEL_PATH + "mpi.prototxt";
string weightsFile = MODEL_PATH + "pose_iter_160000.caffemodel";
int nPoints = 15;
Net net = readNetFromCaffe(protoFile, weightsFile);
```

然后对输入图像进行 blob 转换，设置模型输入数据之后，调用 forward 方法完成推理，得到输出层结果，代码如下：

```
int inWidth = img.cols;
int inHeight = img.rows;
Size netInputSize = Size(368, 368);
Mat inpBlob = blobFromImage(img, 1.0 / 255, netInputSize, Scalar(0, 0, 0), false, false);
net.setInput(inpBlob);

// Forward pass
Mat output = net.forward();
int H = output.size[2];
int W = output.size[3];
```

对输出结果进行解析，解析步得到姿态评估的关键点坐标与点对，然后进行绘制，代码实现如下：

```
vector<Point> points(nPoints);
for (int n = 0; n < nPoints; n++) {
    // Probability map of corresponding body's part.
    Mat probMap(H, W, CV_32F, output.ptr(0, n));

    Point2f p(-1, -1);
    Point maxLoc;
    double prob;
    minMaxLoc(probMap, 0, &prob, 0, &maxLoc);
    if (prob > thresh) {
        p = maxLoc;
        p.x *= (float)frameWidth / W;
        p.y *= (float)frameHeight / H;

        circle(frameCopy, cv::Point((int)p.x, (int)p.y), 8, Scalar(0, 255, 255), -1);
        cv::putText(frameCopy, cv::format("%d", n), cv::Point((int)p.x, (int)p.y), cv::
                    FONT_HERSHEY_COMPLEX, 1, cv::Scalar(0, 0, 255), 2);

    }
    points[n] = p;
}

int nPairs = sizeof(POSE_PAIRS) / sizeof(POSE_PAIRS[0]);
for (int n = 0; n < nPairs; n++) {
    // lookup 2 connected body/hand parts
    Point2f partA = points[POSE_PAIRS[n][0]];
    Point2f partB = points[POSE_PAIRS[n][1]];

    if (partA.x <= 0 || partA.y <= 0 || partB.x <= 0 || partB.y <= 0)
        continue;

    line(imgOrig, partA, partB, Scalar(0, 255, 255), 8);
    circle(imgOrig, partA, 8, Scalar(0, 0, 255), -1);
    circle(imgOrig, partB, 8, Scalar(0, 0, 255), -1);
}
imshow("lines", imgOrig);
imshow("keypoints", frameCopy);
```

运行结果如下：

关键点　　关键点+连线

图 10.4-1