深 圳 大 学 实 验 报 告

课程名称：智能识别系统设计

实验项目名称：基于机器学习的目标检测实践

学院：计算机与软件学院

专业：软件工程

指导教师：沈琳琳、文嘉俊

报告人：洪子敬 学号：2022155033 班级：腾班

实验时间：2024 年 11 月 3 日至 11 月 30 日

实验报告提交时间：2024 年 11 月 26 日

教务处制

一、实验目的与要求：

1.实验目的：

1）了解 Fast-MTCNN 检测模型；

2）掌握 Fast-MTCNN 深度模型程序，并封装为 C++类以实现图像中的人脸检测；

3）了解仿射变换的原理，编制相关程序实现人脸归一化，以保证检测到的人脸图像可以获得很好对齐。

2.实验要求：

1）熟悉 OpenCV 编程环境；

2）熟练掌握至少一种深度学习网络结构；

3）熟悉 OpenCV 中对深度学习模块的调用；

二、方法、步骤：

以下问题二选一：

问题一：自由选择基于深度学习方法的检测模型对各种自然环境下人脸进行检测分析。

问题二：

(1) 根据实验素材（或从以下网页下载）把 Fast-MTCNN 模型和仿射对齐程序（附件中的 align_crop_example.cpp 文件）封装成 C++类；

https://gitee.com/xujun25/Fast-MTCNN

(2) 编写 OpenCV 程序实现深度学习模型接口的调用；

(3) 对多个图像展开人脸检测实验，并对实验预处理、参数调整、训练以及检测效果等的各个环节作必要分析。

三、实验过程及内容：

1. Fast-MCNN 检测模型

（1）模型简介

Fast-MCNN（Multi-task Cascaded Convolutional Networks，多任务级联卷积网络），是一种用于人脸检测和关键点定位的深度学习模型，旨在提高检测的速度和准确性。

（2）网络架构

Fast-MCNN 通常采用多阶段的卷积神经网络，主要分为以下三个阶段：

P-Net（Proposal Network）：

a. 首个网络阶段，负责生成候选人脸区域（人脸候选框）；
b. 对输入图像进行卷积操作，并使用滑动窗口的方法，生成多个候选框及其对应的置信度（人脸存在的概率）；
c. 使用非极大值抑制（NMS）来过滤掉重叠较大的候选框，最终输出较少的高置信度候选框。

R-Net（Refine Network）：

a. 第二个阶段，接收 P-Net 输出的候选框，进一步精炼这些框；
b. 通过更深的网络结构对候选框进行特征提取，输出更精确的边界框和人脸的关键点位置（如眼睛、鼻子和嘴巴）；
c. 同样使用 NMS 进行过滤，以提高检测准确性。

O-Net（Output Network）：

a. 第三阶段，进一步精细化 R-Net 的输出，提供更为准确的边界框和关键点位置；
b. 该网络不仅输出边界框，还输出人脸的五个关键点位置。

（3）多任务学习

Fast-MCNN 通过多任务学习的方式，在同一网络中同时进行人脸检测和关键点定位。这样的设计使得模型能够共享特征，从而提高效率和准确性。

（4）特点和优势

a. 快速性：Fast-MCNN 通过级联结构和卷积操作，使得检测速度较快，适合实时应用场景。

b. 高准确性：采用多级精炼的策略，能够在不同阶段提高检测的准确率，尤其是在复杂背景和多姿态的情况下。

c. 适应性强：该模型对不同尺寸、方向和光照条件下的人脸具有较好的适应能力，适合各种应用场景。

2. Fast-MCNN 深度模型 C++实现

（1）模型导入

由于实验已经给的 Fast-MCNN 的模型文件，包括预训练三个阶段的人脸检测模型的原型文件以及不同精度的 Caffe 权重文件，我们可以将其封装成 C++类 MTCNN；在主函数中新建类加载该模型文件即可。

（2）指定好模型以下参数：

a. factor：影响图像金字塔的缩放比例，较小的值可以提高检测的准确性，

但会增加计算时间；

b. minSize：最小检测人脸尺寸，如果图像中的人脸小于该尺寸，则不会被检测到，根据具体应用场景可以调整；

c. threshold：一个包含三个值的数组，分别对应三个网络的置信度阈值，调整这些值可以优化检测效果；其中 threshold[0]代表 P 网络的阈值、threshold[1]代表 R 网络的阈值，threshold[2]代表 O 网络的阈值，较低的阈值可能会检测到更多的面孔，但可能会增加误检率。

```
class MTCNN {
public:
    MTCNN(const string& proto_model_dir);
    vector<FaceInfo> Detect_mtcnn(const cv::Mat& img, const int min_size, const float* threshold, const float factor, const int stage);
    //protected:
    vector<FaceInfo> ProposalNet(const cv::Mat& img, int min_size, float threshold, float factor);
    vector<FaceInfo> NextStage(const cv::Mat& image, vector<FaceInfo>& pre_stage_res, int input_w, int input_h, int stage_num, const float threshold);
    void BBoxRegression(vector<FaceInfo>& bboxes);
    void BBoxPadSquare(vector<FaceInfo>& bboxes, int width, int height);
    void BBoxPad(vector<FaceInfo>& bboxes, int width, int height);
    void GenerateBBox(Mat* confidence, Mat* reg_box, float scale, float thresh);
    std::vector<FaceInfo> NMS(std::vector<FaceInfo>& bboxes, float thresh, char methodType);
    float IoU(float xmin, float ymin, float xmax, float ymax, float xmin_, float ymin_, float xmax_, float ymax_, bool is_iom = false);
    Mat getTformMatrix(float* std_points, float* feat_points);
    void cropFace(Mat src, MTCNN& fd, vector<FaceInfo> faceInfo);


    //    std::shared_ptr<dnn::Net> PNet_;
    //    std::shared_ptr<dnn::Net> ONet_;
    //    std::shared_ptr<dnn::Net> RNet_;
public:
    dnn::Net PNet_;
    dnn::Net RNet_;
    dnn::Net ONet_;

    std::vector<FaceInfo> candidate_boxes_;
    std::vector<FaceInfo> total_boxes_;
};
```

图 1. MTCNN 类封装代码图

（3）人脸检测前需要进行实验的预处理：

a. 图像的缩放：根据指定的缩放因子，通过 resize 函数调整图像大小；

b. 图像的复制：通过 clone 函数对输入图像进行复制，以便后续处理中不影响原始图像；

c. 处理计时：使用 opencv 函数 getTickCount，评估检测速度。

```
int main(int argc, char** argv)
{

    MTCNN detector("model");
    string name_list[] = {
        "D:/images/5.jpg",
        "D:/images/6.jpg",
    };
    float factor = 0.709f;
    float threshold[3] = { 0.8f, 0.7f, 0.7f };
    int minSize = 12;

    for (int n = 0; n < 2; ++n) {
        //图像读入
        cv::Mat image = cv::imread(name_list[n], 1);
        double scaleFactor = 0.5;
        Size newSize(static_cast<int>(image.cols * scaleFactor), static_cast<int>(image.rows * scaleFactor));
        resize(image, image, newSize);

        cv::Mat image_copy = image.clone(); // 复制图像
        //计时
        double t = (double)cv::getTickCount();
        //人脸检测
        vector<FaceInfo> faceInfo = detector.Detect_mtcnn(image, minSize, threshold, factor, 3);
        std::cout << name_list[n] << " time," << (double)(cv::getTickCount() - t) / cv::getTickFrequency() << "s"<< std::endl;
        //画框
        for (int i = 0; i < faceInfo.size(); i++) {
            int x = (int)faceInfo[i].bbox.xmin;
            int y = (int)faceInfo[i].bbox.ymin;
            int w = (int)(faceInfo[i].bbox.xmax - faceInfo[i].bbox.xmin + 1);
            int h = (int)(faceInfo[i].bbox.ymax - faceInfo[i].bbox.ymin + 1);
            cv::rectangle(image, cv::Rect(x, y, w, h), cv::Scalar(255, 0, 0), 2);
        }
        cv::imshow("image", image);
        cv::waitKey(0);
```

图 2. 进行人脸检测的主函数

3. 仿射变换

（1）基本原理

仿射变换是一种常见的几何变换操作,它可以将图像进行平移、旋转、缩放和倾斜等变换；主要特点是**保持直线**的直线性质,即变换后直线依然是直线。

仿射变换可以用一个 3x3 的变换矩阵来表示,该矩阵包含了平移、旋转、缩放和倾斜等参数。对于一个二维图像上的点 (x, y)，其经过仿射变换后的新坐标 (x', y') 可以表示为：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & tx \\ c & d & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

其中：a、b、c、d 表示缩放和旋转参数，tx、ty 表示平移参数；

平移：当 a = d = 1, b = c = 0 时，tx 和 ty 控制平移。

缩放：当 tx = ty = 0 时，a 和 d 控制 x 和 y 方向的缩放。

旋转：当 a = d, b = -c，tx = ty = 0 时，可以实现旋转。

倾斜：当 a ≠ d, b ≠ -c，tx = ty = 0 时，可以实现倾斜变换。

（2）人脸对齐

在前面检测到人脸并使用矩形框得到人脸区域后，由于各种因素，人脸可能会发生倾斜甚至不对齐的结果，因此下面我们进行人脸的对齐。

首先将检测到的人脸区域提取关键点（关键点为预先定义好的 std_points）存储在数组 facial_points 中；接着使用 getTformMatrix 函数得到一个仿射变换矩阵，这个矩阵可以将人脸变换到一个标准尺寸的图像中；然后使用 warpAffine 函数对原始图像进行仿射变换得到标准化人脸图像；最后将裁剪出的人脸图像进行归一化实现人脸的对齐。

```
void MTCNN::cropFace(Mat src, MTCNN& fd, vector<FaceInfo> faceInfo)
{
    if (faceInfo.empty()) {
        std::cerr << "No faces detected!" << std::endl;
        return;
    }
    // 假设输入图片中只有一张人脸
    for (int n = 0; n < faceInfo.size(); n++) {
        float facial_points[10];//坐标依次是：左眼X，左眼Y，右眼X，右眼Y，鼻尖X，鼻尖Y，左嘴角X，左嘴角Y，右嘴角X，右嘴角Y
        for (int i = 0; i < 5; i++) {
            facial_points[2 * i] = faceInfo[n].landmark[2 * i];
            facial_points[2 * i + 1] = faceInfo[n].landmark[2 * i + 1];
        }
        Mat tform = getTformMatrix(std_points, facial_points);
        Mat dstImage(112, 96, CV_8UC3);
        warpAffine(src, dstImage, tform, dstImage.size(), 1, 0, Scalar(0));

        Mat subfactor = 127.5 * Mat(112, 96, CV_32FC3, Scalar(1, 1, 1));
        dstImage.convertTo(dstImage, CV_32FC3);
        dstImage = dstImage - subfactor;
        dstImage = dstImage / 128;

        //展示图像
        //cv::imwrite("D:/images/2_cropped.jpg", cropped_face);
        cv::imshow("Cropped Image", dstImage);
        cv::waitKey(0);
    }
}
```

4. 人脸检测和对齐实验

由于本实验只给定了模型文件，但未给出数据集，无法使用带标签的验证集或者测试集对效果进行测试；现从网上选取两张人脸图片进行人脸检测和仿射对齐的效果测试，两张测试图像如下所示：

图 3. 测试图像 1

图 4. 测试图像 2

此外，不难看到我们选用的都是没加上噪声（模糊、遮挡以及光照条件变换等）的图像，这是**由于未知该模型的训练集上加的噪声类型，直接用噪声图片检测效果不可控，也不好调参；此外，去试异常也比较耗时**，于是实验选取的两张都是正常图片。

（1）设定 factor 为 0.709，threshold 数组为{0.7f，0.6f，0.6f},minSize 为 12；

在给定参数组 1 情况下运行程序，可以得到：

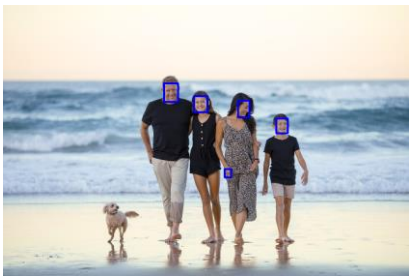

图 5. 参数组 1 下结果图像 1

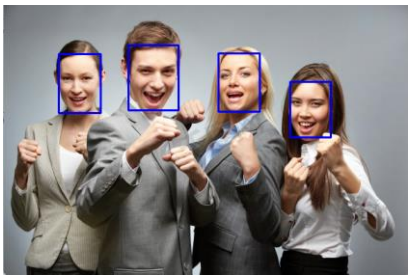

图 6. 参数组 1 下结果图像 2

观察图 5 可知，人脸基本检测完全，唯一出错的地方在于误判一只手为人脸并框出来了，可能是评判标准低了；而图 6 的人脸是全部检测出来了。

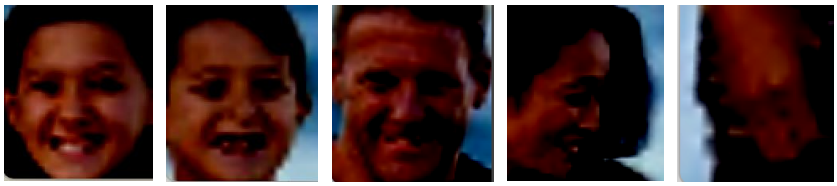

图 7. 图像 1 人脸参数组 1 下对齐结果

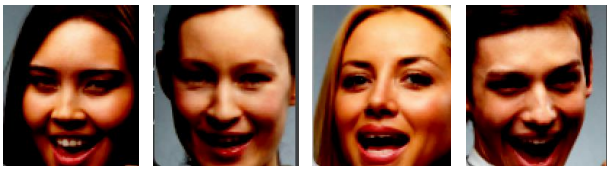

图 8. 图像 2 人脸参数组 1 下对齐结果

观察图 7、8 可知，在检测正确的人脸中，图像的对齐效果还是比较好的。

（2）设定 factor 为 0.709，threshold 数组为{0.6f，0.5f，0.5f},minSize 为 12；

在给定参数组 2 情况下运行程序，可以得到：

图 9. 参数组 2 下结果图像 1

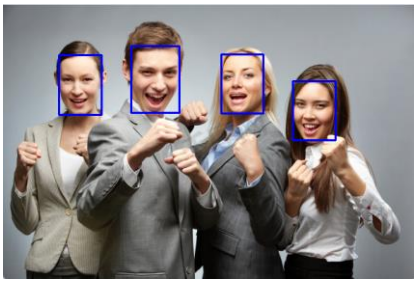

图 10. 参数组 2 下结果图像 2

观察图 9 可知，人脸基本检测完全，此时却没有发生误判了，这里可能因图像而异，毕竟这里是降低了阈值反而识别不到了，说明它是接近人脸关键点的干扰点；而图 10 的人脸同样是全部检测出来了。

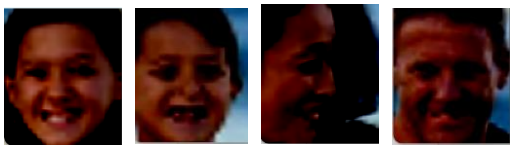

图 11. 图像 1 参数组 2 下人脸对齐结果

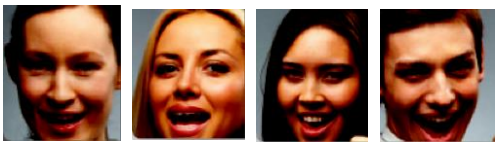

图 12. 图像 2 参数组 2 下人脸对齐结果

观察图 11、12 可知，图像的对齐效果同样比较好的。

（3）设定 factor 为 0.709，threshold 数组为{0.8f，0.7f，0.7f},minSize 为 12；

在给定参数组 3 情况下运行程序，可以得到：

图 13. 参数组 3 下结果图像 1

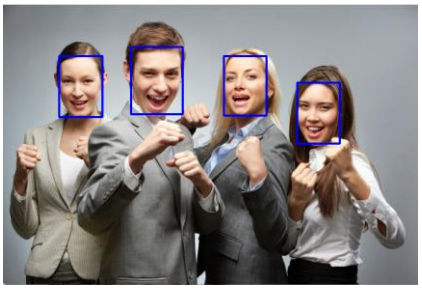

图 14. 参数组 3 下结果图像 2

观察图 13 可知，人脸基本检测完全，此时也没有发生误判了，这是符合理论的，升高阈值增加识别人脸的难度，自然而然就将非人脸图像筛查掉了；而图 14 的人脸同样是全部检测出来了。

图 15. 图像 1 参数组 3 下人脸对齐结果

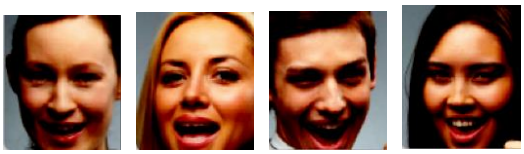

图 16. 图像 1 参数组 3 下人脸对齐结果

观察图 15、16 可知，图像的对齐效果同样比较好的。

（4）参数组 1、2、3 的结果分析与比较

参数组序	图片1耗时	图片2耗时
1	0.330383	0.31154
2	0.345023	0.513642
3	0.253605	0.224047

图 17. 参数组 1、2、3 下实验耗时统计表

观察耗时可知，参数组 2 虽然排除了干扰，但是耗时增多了，说明检测的人脸更多了，筛查耗时更多；而参数组 3 在提高识别精度的情况下，在两张图片的识别上耗时均减少了许多，说明此参数组 3 是这三组中的最佳选择。

（5）设定 factor 为 0.709，threshold 数组为{0.7f，0.6f，0.6f},minSize 为 20；

在给定参数组 4 情况下运行程序，可以得到：

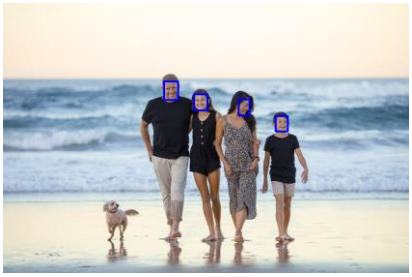

图 18. 参数组 4 下结果图像 1

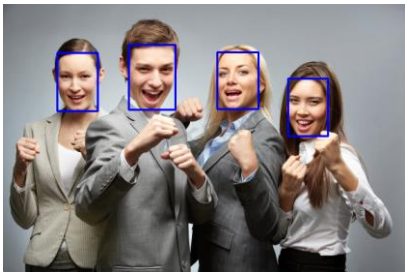

图 19. 参数组 4 下结果图像 2

观察图 18 可知，人脸基本检测完全，此时也没有发生误判了，相比参数组 1 的情况筛除了误分点，这是因为增大了 minSize，检测到的手部不够大而被筛除了；而图 19 的人脸同样是全部检测出来了。

图 20. 图像 1 参数组 4 下人脸对齐结果

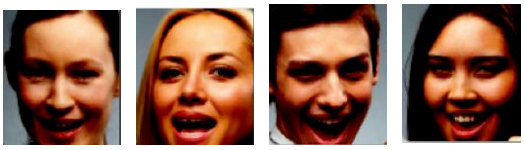

图 21. 图像 2 参数组 4 下人脸对齐结果

观察图 20、21 可知，图像的对齐效果同样比较好的。

（6）设定 factor 为 0.709，threshold 数组为{0.7f，0.6f，0.6f},minSize 为 30；

在给定参数组 5 情况下运行程序，可以得到：

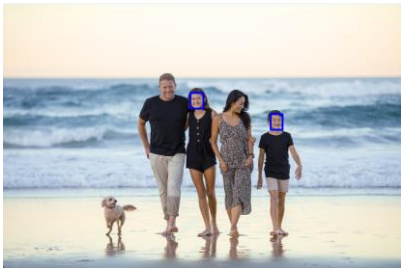

图 22. 参数组 5 下结果图像 1

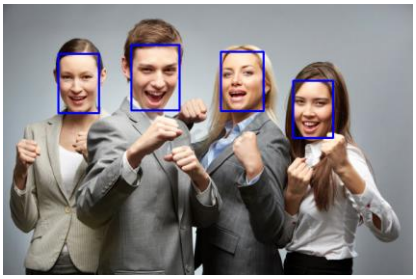

图 23. 参数组 5 下结果图像 2

观察图 22 可知，只检测到两个人脸，这是因为 minSize 设置过大了，导致有些小一点的人脸被筛选掉了；而图 19 的人脸同样是全部检测出来了。

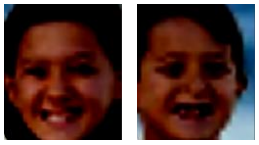

图 24. 图像 1 参数组 5 下人脸对齐结果

图 25. 图像 2 参数组 5 下人脸对齐结果

观察图 24、25 可知，在检测出的人脸中，对齐效果同样比较好的。

（7）参数组 1、4、5 的结果分析和比较

参数组序	图片1耗时	图片2耗时
1	0. 330383	0. 31154
4	0. 173654	0. 186074
5	0. 107316	0. 121661

图 26. 参数组 1、4、5 下实验耗时统计表

观察耗时可知，参数组 5 虽然最少，但是效果差，是通过减少识别到的人脸数来达到的，非我们想要的；而参数组 4 准确地识别到了所有人脸，耗时也大大减少，是这三个参数组中的最佳选择。

四、实验结论：

本次实验了解了 Fast-MTCNN 检测模型和仿射变换原理，并通过给的代码成功将 Fast-MTCNN 深度模型程序封装为 C++类并在主函数中实例化该对象，进行人脸检测并利用仿射变换实现归一化，形成完整的目标检测程序。

最后也是在条件有限下成功地使用两张人脸图片进行了参数的设置，在控制变量条件下粗略地测试得到 minSize 较好的值为 20，threshold 为{0.8f，0.7f，0.7f}，当然实际遇到数据集还得继续调整，但结果人脸识别和对齐地结果也足以说明程序设计的成功。

本次实验圆满结束。

五、实验体会：

本次实验进行 Fast-MTCNN 的实验，是我了解的除了 yolo 深度模型以及前个实验的 Haar 检测器之外的另一种目标检测模型，在参数调好的情况下，整体的效果还是不错的；本实验唯一不足的就是没给数据集，无法进行模型的训练、验证和测试，也无法找到比较好的参数范围，无法深入了解该模型。

指导教师批阅意见：

成绩评定：

指导教师签字：沈琳琳、文嘉俊

2024 年 12 月 4 日

备注：