报告	1	2	3	4	5	6	7	8	心得	总分
得分										

“信息技术导论”报告册

（2022）

姓　名：International River

班　级：软 2203

学　号：20222241075

课序号：01

大连理工大学

Dalian University of Technology

目录

报告一题目：嵌入式系统特点与实例分析

1　背景综述

嵌入式系统是指这样的一类计算机系统：作为整体的一个子系统，被嵌入至整个系统中，与系统中的其他软、硬件相配合，从而高效率地实现专门、特定功能。

嵌入式系统的核心是微处理器，目前这一领域几乎被持有 ARM 技术产权的 ARM 公司所垄断，其拥有知识产权核心的 32 位嵌入式处理器约占全球市场的 90%[1]。

1.1　特点

嵌入式系统使用目的决定，嵌入式系统应当具有以下的特性[2]：

（1）软硬件紧密结合：嵌入式系统是面向产品、面向应用的，应当根据需求对软、硬件同步进行特异化设计[3]，随着设计质量的提高，软、硬件可以更高质量地完成设计目标。

（2）小型化；随着市场需求与科技发展，许多电子设备正逐渐变得小型化、便捷化，这对承载嵌入式系统的硬件体积提出了要求。同时，为了提高运行效率、提升资源利用率，嵌入式操作系统与软件的小型化也势在必行[4]。

（3）高可靠性；

（4）高效率、高实时性；

（5）低功耗；

（6）低成本；

（7）高度集成化；

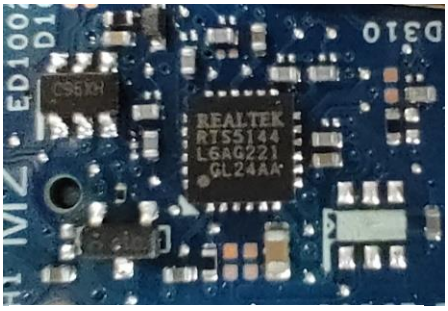

图 1-1　RTS5144 微处理器

以笔者的个人 PC 为例（见图 1-1），其中含有的 RTS5144 USB3.0 内存卡读卡器尺寸为 4.00mm×4.00mm，而 RTS5169 USB2.0 兼容读卡器尺寸为 7.00mm×7.00mm[4]，这种差异的原因在于，RTS5144 功能更加特化，且笔记本电脑对内嵌的微处理器的尺寸要求更加严格。

1.2　组成

硬件部分，以 MPU（微处理器）为核心，辅以电源模块、时钟、RAM、ROM 等外围电路，以及可能的外设设备（USB、显示屏、键盘等）；软件部分，应用软件是必要的，而嵌入式操作系统一般更多地用于较大型或者多任务的应用场合。[2]

2.　应用实例分析：智能家居检测控制系统

随网络技术进步与嵌入式系统的更广阔应用，智能家居行业正在蓬勃发展，并且更多地进入公众的视线。智能家居系统是一种综合的、复杂的、贴近生活的嵌入式系统，很适合作为典型应用实例进行分析。

2.1 需求

对家庭进行安全监护、环境优化管理、能源监视控制，以及家居智能控制[6]。每一项功能都需要核心智能主控制模块与其它模块协同实现，从而真正提高智能家居系统使用者的生活质量。

2.2 总体设计与组成

以主控模块为核心，经过通信模块与其他模块/用户形成联系，用户可以由主控模块控制智能家具，主控模块也可以接收传感器获取的物理、化学信号，分析温度、能量使用、家具状态等信息，从而根据设定对智能家具进行自动控制。

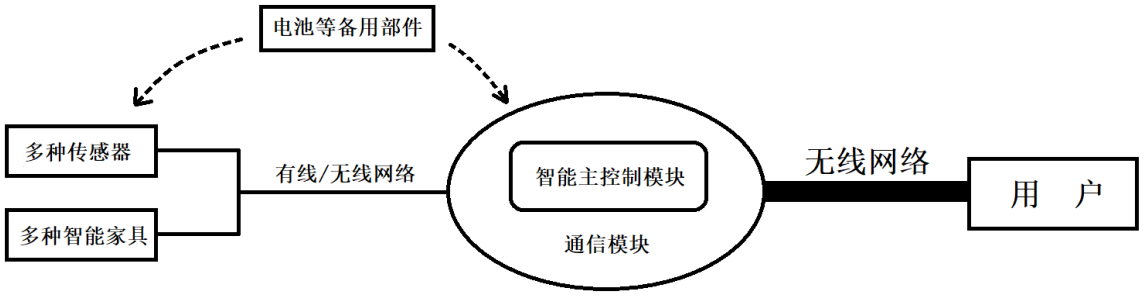

图 2-1 智能家居系统设计模式图

2.2.1 通信模块

智能家居间的网络通讯，需要稳定与低能耗。目前 ZigBee 已成为领先的无线标准，但一些主要的行业公司支持 Wi-Fi、ZWave、6LoWPAN2 等替代技术。[7]

2.2.2 软件部分

目前，大部分智能家居开发公司都开发了自己的软件系统，包括核心的多功能网关与手机 APP。以小米为例，其三大核心产品为小米手机、小米路由器和小米电视，其中小米路由器是整个生态圈的核心[8]。依托于这三种设备与内嵌的软件，小米目前已经实现了产品矩阵的构建。

2.2.3 其它模块

安全监护模块：包含燃气报警器、烟雾传感器等[6]。

环境优化模块：包括智能空调、暖气、加湿器以及对应的温度、湿度传感器等。

能源监控模块：有红外探测器等传感器实时感应人的位置，并且智能决策关闭某些电器以达到节能效果。

3. 总结

嵌入式系统是充满无限潜能与可能性的领域。事实上，早在 2002 年，分散式处理器

的生产总额中，就有 94%为嵌入式微处理器，PC 用处理器只占 6%。[9] 随着技术的迭代与需求的提升，嵌入式系统将在更多的领域发挥出巨大的作用，在通信、仪器仪表、汽车、船舶、制造工业、航空航天装备、军事装备乃至个人消费品中扮演不可或缺的角色。

参考文献

[1] 施乐平,杨征宇,马宪民,汤元会.ARM 嵌入式系统综述[J].中国测试,2012,38(S1):14-16.

[2] 马忠梅. ARM & Linux 嵌入式系统教程. 第 2 版. ed. 北京: 北京航空航天大学出版社, 2008.

[3] Wang, Z., Haberl, W., Herkersdorf, A., & Wechs, M. (n.d.). A Simulation Approach for Performance Validation during Embedded Systems Design. Leveraging Applications of Formal Methods, Verification and Validation, 385-399.

[4] 阳富民,李文海,涂刚. 嵌入式 linux 系统动态库小型化技术研究[J]. 华中科技大学学报（自然科学版）,2004,32(9):6-8. DOI:10.3321/j.issn:1671-4512.2004.09.003.

[5] Realtek Semiconductor Corp. RTS5169 USB 2.0 SD/MMC/MS/MSPRO/xD-Picture& Smart Card / SIM Card Reader DATASHEET. Rev1.2, 2009.

[6] 蒋建春 嵌入式系统原理及应用实例. 北京航空航天大学出版社, 2015.

[7] Balta-Ozkan, Nazmiye, et al. “The Development of Smart Homes Market in the UK.” Energy (Oxford), vol. 60, 2013, pp. 361–372.

[8] 严泽鼎.小米智能家居生态圈产品现状及发展研究[J].财讯,2018,0(13):180-180

[9] 魏庆福,郑文波.嵌入式系统的技术发展和我们的机遇. 自动化博览, vol. 20, no. z1, 2003, pp. 138–141.

报告二题目：三种平均时间复杂度为 O(nlogn)的排序算法

1 希尔排序

希尔排序是由其设计者 Donald Shell 于 1959 年公布的一种排序算法[1]，后由 Pratt 等人进行了少量修改[2]，使得算法的最高事件复杂度提高至 $O(nlog^2n)$，尽管如此，这仍然比最好的比较排序算法的 O(nlogn)差一些。

1.1 原理

希尔排序是插入排序的改进版本，实际极大改良了插入排序的平均时间复杂度。希尔排序是基于插入排序的以下两点性质而提出改进方法的：

1. 插入排序在对几乎排好序的数据操作时，效率高，可以达到线性效率；
2. 插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位。

因此，Shell 提出将整个序列分割成多个子序列分别进行插入排序，每个子序列的相邻元素下标相差增量 k_i，逐渐减小增量至最后 $k_i=1$。这样做的好处是显著增加了单个元素移动的步长，并在最后一步操作前使得数列非常接近于排序完成的结果。

1.2 伪代码

```
int n =  数组大小;
int a[n + 1];
将数组元素全部存至 a[1]到 a[n];
int j = 1;
while (1) {
    指定 kj;
    for (int i = 1; i <= kj; i++)
        对 a[i], a[i + kj], a[i + 2 * kj]...子列进行插入排序;
    if (kj == 1)
        break;
    j++;
}
```

1.3 代码实现

输入格式为：第一行一个数字 N，代表待排序数列的的元素数量，以下 N 行，每行一个数字，代表数列中的每一个元素。下文依旧如此。

默认使用 Shell 提出的增量序列，默认数列元素数量不大于 10^7，若要使其更大，需要更改 maxn 的值。

```
#include <bits/stdc++.h>
#define maxn 10000050
int arr[maxn];

int main() {
```

```
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    int temp, i, j, gap;
    for (gap = n / 2; gap >= 1; gap /= 2)
        for (i = gap; i < n; i++) {
            temp = arr[i];
            for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)
                arr[j + gap] = arr[j];
            arr[j + gap] = temp;
        }

    return 0;
}
```

1.4 改进

目前对希尔排序的改进主要集中在增量序列的选择上。

Shell 使用 N/2，N/4，...，1（反复除以 2）作为增量序列，此增量序列在最坏情况下时间复杂度为 $O(n^2)$。后来有学者提出更优的增量序列，如：

Hibbard 的增量[3]：1，3，7，...，$2^k - 1$ ；

Knuth 的增量[4]：1，4，13，...，$(3k - 1) / 2$；

目前表现最好的增量序列是 Sedgewick 给出的[5]：

Sedgewick 的增量：1，5，19，41，...此增量序列由两个序列的元素交错得到：

1，19，109，505，2161，……，$9(4^k – 2^k) + 1$

5，41，209，929，3905，……，$2^{k+2}(2^{k+2} - 3)+1$

另外，由斐波那契数列而得到的数列 1，9，34，182，836，…… (设斐波那契数列为 F(n)，则本数列为 n=floor[F(n+1)^(2Φ)]，其中 $\Phi = \frac{1+\sqrt{5}}{2}$)在大数组排序中表现优异。

2 归并排序

1945 年， John von Neumann 发明了归并排序，这是典型的分治算法的应用。

2.1 原理

将待排序的数列不断地切分成若干个子列，直到每个子列只包含一个元素，此时所有子列均为有序子列。将子列两两合并，每合并一次会产生一个新的且更长的有序子列，重复这一步骤，直到最后只剩下一个数列，这就是完成排序的数列。

2.2 伪代码

以下是自顶向下的递归法，先分割再合并。

```
void mergesort(待排数列 a[], 暂存空间 save[], start, end) {
    if (start >= end)
        return;//已经切分完成
    int mid = (start + end) / 2;
    对 start 到 mid 使用 mergesort;
    对 mid + 1 到 end 使用 mergesort; /*自我迭代
    对上面两个排好序的子列进行合并*/;
}
```

以下是自下而上的迭代，优点是不需要分割，提高效率，同时减少内存占用，防止栈溢出错误。

```
void mergesort(待排数列 a[], 暂存空间 save[], start, end) {
    int step = 1;
    while (step < high) {
        for (int i = low; i < high; i += step * 2) { //重复执行约 end/(2*step)次
            定义长度不大于 2 * step 的子列的上下界，并防止越界
            int mid = min(i + step, high);
            对 min 到 mid 与 mid 到 end 进行有序归并
        }
        step *= 2;
    }
}
```

2.3 代码实现

默认数列元素数量不大于 10^7，若要使其更大，需要更改 maxn 的值。

以下是自顶向下的递归法，先分割再合并。

```
#include <bits/stdc++.h>
#define maxn 10000050
int a[maxn], save[maxn];

void mergesort(int arr[], int reg[], int start, int end) {
    if (start >= end)
        return;
    int len = end - start, mid = (len >> 1) + start;
    int start1 = start, end1 = mid;
    int start2 = mid + 1, end2 = end;
    mergesort(arr, reg, start1, end1);
    mergesort(arr, reg, start2, end2);
    int k = start;
    while (start1 <= end1 && start2 <= end2)
        reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[start2++];
    while (start1 <= end1)
        reg[k++] = arr[start1++];
    while (start2 <= end2)
        reg[k++] = arr[start2++];
    for (k = start; k <= end; k++)
```

```
        arr[k] = reg[k];
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    mergesort(a, save, 0, n - 1);
    return 0;
}
```

以下是自下而上的迭代，无需分割

```
#include <bits/stdc++.h>
#define maxn 10000050
int a[maxn], save[maxn];

void mergesort(int *arr, int start, int end) {
    int step = 1;
    while (step < end) {
        for (int i = start; i < end; i += step << 1) {
            int start1 = i, end2 = (i + (step << 1) - 1 ) <= end ? (i + (step << 1) - 1 ):end;
            int mid = i + step - 1 <= end ? i + step - 1 : end;
            int end1 = mid;
            int start2 = mid + 1;
            int k = start1;
            while (start1 <= end1 && start2 <= end2) {
                save[k++] = (arr[start1] < arr[start2]) ? arr[start1++] : arr[start2++];
            }
            while (start1 <= end1) {
                save[k++] = arr[start1++];
            }
            while (start2 <= end2) {
                save[k++] = arr[start2++];
            }
        }
        for (int i = start; i <= end; i++)
            arr[i] = save[i];
        step <<= 1;
    }
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    mergesort(a, 0, n - 1);
    return 0;
}
```

2.4 改进

在分割出的子序列长度足够小时，继续分割的效率反而不如简单的插入排序，因此可以在子序列足够小（如：小于等于 8）时，使用插入排序。

另外，快速归并排序与 Timsort 都更进一步采取了一个更高效率的策略[6][7]：对某些已经单调递增或是递减的子列，除对递减子列进行翻转，不做其它操作。

3 快速排序

此排序方法由 Hoare 在 1961 年提出[8]，是一种在大多数情况下效率极高的排序算法。

3.1 原理

形象地说，归并排序是先分割再有序归并，而快速排序是先有序分割再直接归并。

1.从数列中挑出一个元素，称为基准。

2.重新排序数列，比基准值小的元素放在基准前面，基准值大的放在基准后面（相同的数可以到任一边）。操作结束后，该基准就处于数列的中间位置。

3.重复地把小于基准值元素的子数列和大于基准值元素的子数列排序。

3.2 代码实现

默认数列元素数量不大于 10^7，若要使其更大，需要更改 maxn 的值。

以下是递归法，在某些数据下不排除堆栈过多造成段错误，并且速度相对较慢。

```
#include <bits/stdc++.h>
#define maxn 10000050
int a[maxn];

void quicksort(int arr[], const int left, const int right) {
    int low = left, high = right;
    if (low > high)
        return;
    int k = arr[low];
    while (arr[high] >= k && low < high)
        high--;
    arr[low] = arr[high];
    while (arr[low] <= k && low < high)
        low++;
    arr[high] = arr[low];
    arr[low] = k;
    quicksort(arr, left, low - 1);
    quicksort(arr, low + 1, right);
}

int main() {
    int n;
```

```
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    quicksort(a, 0, n - 1);
    return 0;
}
```

3.3 改进

为克服递归写法的缺点，可以使用结构体数组（二维数组也可，但不够直观）保存操作范围，进而运用循环模拟递归，由此可以实现更高效率的快速排序。基准值的选取也会影响到快速排序的效率：若是始终选取操作范围的第一个或是最后一个数，在面对比较有序的数列时快速排序效率会很低，选择操作范围中间值可以较好地避免这种情况。另外，与归并排序类似，在分割出的子序列长度足够小时，继续分割的效率反而不如简单的插入排序，因此可以在子序列足够小（如：小于等于 8）时，使用插入排序，以下是改进后的代码。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10000050;
int a[N] = {};

typedef struct {
    int *start, *end;
} Range;//定义 Range 结构体

Range r[N];//r 储存当前需要排序的范围

Range newrange(int *s, int *e) {
    Range r;
    r.start = s, r.end = e;
    return r;
}//返回值为 Range 结构体

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

void isort(int *start, int *end) {
    if (start == end)
        return;
    for (int i = 1; start + i <= end; i++) {
        int temp = *(start + i), j;
        for (j = i; temp < * (start + j - 1) && j > 0; j--)
            *(start + j) = *(start + j - 1);
        *(start + j) = temp;
```

```
    }
}

void qsort(int *start, int *end) {
    if (start >= end)
        return; //避免 len<=0 产生段错误（Segment Fault）
    int p = 0;
    /*p 每加 1 就是多一层迭代，每减 1 就是少一层迭代
    事实上并没有迭代，只是用很多组排序范围的改变模拟迭代，实质上是循环。
    循环的好处很明显：速度更快且不会栈溢出*/
    r[p++] = newrange(start, end - 1);
    while (p) {
        Range range = r[--p];//range 是下文的排序范围
        if (range.start >= range.end - 7) {
            isort(range.start, range.end);
            continue;
        }
        /*range 小于等于 8 时用插入排序：当 range 小于等于某个不大的数时，用插入排序比快排更快*/
        int *left = range.start, *right = range.end, benchmark = *(range.start + (range.end - range.start) / 2);
        /*基准值 benchmark 是整型格式的数据，之所以不使用指针，是因为指向的那个地址的值可能会变化，而基准值是不可以中途变化的*/
        do {
            while (*left < benchmark)
                ++left;
            while (*right > benchmark)
                --right;
            if (left <= right) {
                swap(left, right);
                ++left, --right;
            }
        } while (left <= right);
        if (range.start < right)
            r[p++] = newrange(range.start, right);
        if (range.end > left)
            r[p++] = newrange(left, range.end);
    }
}

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
        scanf("%d", (a + i));
    qsort(a, a + n);
    for (int i = 0; i < n; i++)
        printf("%d ", *(a + i));
    return 0;
}
```

4 总结：优势与不足

本文探讨的三种算法均为常用的排序算法，他们各有优点，又各有不足。

希尔排序的优点在于实现简单，且经过改进后最大时间复杂度为 $O(n\log^2 n)$，不会达到 $O(n^2)$，足够满足不太大的数据的排序，缺点是它在面对极大量数据时略差于归并排序等算法。

归并排序的优点在于其稳定性，即相同大小的两个数字在排序时不会改变先后顺序关系。同时，归并排序是一种时间复杂度恒为 O(nlogn)的排序算法，在面对一部分极端情况时表现优于另外两者。

快速排序的平均时间复杂度是 O(nlogn)，且 O(nlogn)记号中隐含的常数因子很小，比时间复杂度稳定等于 O(nlogn)的归并排序要小很多。所以，对绝大多数顺序性较弱的随机数列而言，快速排序总是优于归并排序。但是在极端情况下，快速排序的时间复杂度将会接近或达到 $O(n^2)$，然而这种情况在数据量大时极少见，因此快速排序依然是一种极为常用的优秀算法。

参考资料

[1] Shell, D. “A High-Speed Sorting Procedure.” Communications of the ACM, vol. 2, no. 7, 1959, pp. 30–32.

[2] PRATT, VAUGHAN R. Shellsort And Sorting Networks, Stanford University, Ann Arbor, 1972.

[3] Hibbard, Thomas. “An Empirical Study of Minimal Storage Sorting.” Communications of the ACM, vol. 6, no. 5, 1963, pp. 206–213.

[4] Knuth, Donald Ervin. The Art of Computer Programming. 2nd ed., Addison-Wesley Pub. Co., 1973.

[5] Sedgewick, Robert. “A New Upper Bound for Shellsort.” Journal of Algorithms, vol. 7, no. 2, 1986, pp. 159–173.

[6] Vignesh R, Tribikram Pradhan．Merge Sort Enhanced In Place Sorting Algorithm: 2016 International Conference on Advanced Communication Control and Computing Technologies, 2016,pp. 698-704

[7] Peter McIlroy. 1993. Optimistic sorting and information theoretic complexity. In Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms (SODA '93). Society for Industrial and Applied Mathematics, USA, 467–474.

[8] Hoare, C. “Algorithm 64: Quicksort.” Communications of the ACM, vol. 4, no. 7, 1961, p. 321.

报告三题目：机器学习实例分析——手写数字识别

1 实例简介

机器学习是一个相对新兴的研究方向，尽管早在 18 世纪，有关机器学习实现方法的许多理论已经建立，上世纪四五十年代，已经开始了机器学习的非纯理论性的研究，但机器学习的最新一波热潮，是从上世纪八十年代乃至本世纪才真正掀起的。[1]本文着眼于机器学习的最基础案例之一：手写数字识别，浅析其工作原理。

2 实例分析

2.1 神经网络的结构构建

手写数字图片是一个 28×28=784 像素的图片，每一个像素对应一个 0~1 的值，表示这个像素的激活值。最终输出应当是 10 个值为 0~1 的数据，对应识别出的数字的激活值。理想状态下，除了正确结果对应的激活值为 1，其余的激活值应当为 0。在输入层与输出层中，还有数个隐含层，为了叙述方便，选用两层分别具有 16 个“神经节”的隐含层。相邻两层间的每两个节点间都有一个连接，连接具有权重，权重值不限。[2]另外，可以在输入与第一层隐含层中加入偏置值，即，无论权重如何，输入必须到达某个固定的激活值才能将影响传递给下一层。传递到下一层的数据相加不一定能够保留在 0~1 的范围内，可以用 Sigmond 函数进行调整。

可以认为整个神经网络相当于一个有 784 个输入值和 10 个结果值的函数，其中所有的连接权重与偏置值是函数的参数。

2.2 代价函数

代价函数：对于一个固定所有权重值的函数而言，所有样例输入后，将得到的十个结果值与理想值作差并求平方的和的平均数称为代价。代价可以用来衡量神经网络的可靠性与有效性。代价越小，意味着对于目前输入的大量数据，网络的可靠性越好。对神经网络中权重参数的调整指向一个目的：使求出的代价最小。

2.3 梯度下降算法 [3]

为了使代价达到一个极小值，设一个函数，自变量为所有 n 个连接权重与偏置值，因变量为代价 ξ 。在 R^{n+1} 直角坐标系中建立函数图像，任意 n 个参数值在 n 维平面上都可以找到对应点。Z 轴垂直于 n 维平面，图像上每一点对应的 z 值代表 ξ 的值。很明显，为了使 ξ 尽可能逼近极小值，应当调整 n 个参数对应的点 A 向局部梯度最低的方向移动。可以求出 A 点的梯度，然后向梯度反方向移动。多次操作后，ξ 会逼近极小值。

2.4 反向传播算法[4]

反向传播算法是一种实现梯度下降的途径。

分析最后的十个输入值，对于每一个输入值，分析为了将其调整至期望值，应当将与其直接相关的 16 个参数如何调整。在调整权重值时，既要调整 16 个权重，也要综合调整前一层 16 个激活值。激活值可以通过调整第二隐含层前的权重与偏置值进行调整，而这 16 个激活值的期望调整幅度应当是输出层期望调整幅度反向传播的加权平均数。如此反向传播，每个输入样例引起不一样的改变，评估改变造成的代价 ξ 的改变，如此重复最终完成学习。

2.5 缺点与优化

将数万甚至更多样例投入神经网络并随着每次改动评估代价 ξ 在实际操作中是不现实的，因此可以使用每次少量样例进行模糊梯度下降。这与大数据时代的机器学习算法发展有相似之处。[5]

Sigmond 函数也是比较过时的将结果映射到固定区间的方式，研究发现用 ReLU，即线性整流函数，替换 Sigmond 函数，可以更容易地达到更好的训练效果，这可能是 ReLU 更多地模仿了生物界神经元的工作原理。

另外，本例中使用 MLP，即多层感知器进行机器学习，然而面对这样的案例，像素点间有紧密联系，应当使用 CNN，即卷积神经网络可以达到更好的实现效果。

参考资料

[1] 陈海虹 黄彪 刘峰 陈文国．机器学习原理及应用．成都：电子科技大学出版社，2017.

[2] [美]哈根（Hagan，M.T.）等著 神经网络设计. 2002.

[3] 周昀锴 “机器学习及其相关算法简介.” 科技传播, vol. 11, no. 6, 2019, pp. 153–165.

[4] Rumelhart, D.E., et al. “Learning Internal Representations by Error Propagation.” Readings in Cognitive Science, Elsevier Inc, 1988, pp. 399–421.

[5] 刘志强 “大数据下的机器学习算法探讨.” 中国新通信, vol. 20, no. 21, 2018, p. 183.

报告四题目：大数据智能化浅析

1 何为大数据智能化

大数据智能化是一类处理大量数据来获得有价值信息的技术，可以理解为数据从工具变成为资产的过程，从一个辅助的东西变成生产资料的过程。[1]这一领域从本世纪开始真正兴起。互联网时代催生了极为庞大的数据湖，因此可以说，数据智能是移动互联网时代的自然产物，也是未来很长一段发展阶段的核心所在。

2 大数据智能化的意义

在经济领域：2012 年 1 月,达沃斯世界经济论坛发布的大数据报告“Big data, big impact: new possibilitiesfor international development”将大数据列为和货币与黄金同等重要的新经济资产,[2]这正印证了大数据的潜在巨大价值以及发展大数据智能化的必要性。

在科研领域：大数据正引领数据密集型科学的到来，形成继实验科学、理论科学以及计算科学之后的第四科学范式。[3]

在社会科学领域：2012 年 5 月,联合国发布的 Big Data for Development: Challenges& Opportunities 白皮书指出，大数据是联合国和各国政府的一个历史性机遇，利用大数据进行决策，是提升国家治理能力，实现治理能力现代化的必然要求，可以帮助政府更好地参与经济社会的运行与发展。[4]

3 大数据智能化的特点

大数据智能化的特点由大数据的 4V 特性决定：数据体量大、数据类型多样、数据价值大、数据通量高。

3.1 重视相关关系

过往的数据分析中，人们尽力确保信息的精确性，并且假定数据间有某种因果关系，通过反复计算或实验寻找数据间的因果关系。而大数据环境下,数据的精确性难以保证,这与大数据价值大且价值密度低的特性是分不开的：数据总体完备性对价值获取异常重要。因此，大数据智能要寻找的是数据间的相关关系。[5]相关关系的相比于因果关系是一种较弱的关系，但在大数据环境下，相关关系是可以接受的，并且相关关系能够满足人类的众多决策需求。该方面的成功案例有 Google 公司的流感预测等（如图 1）。[6]

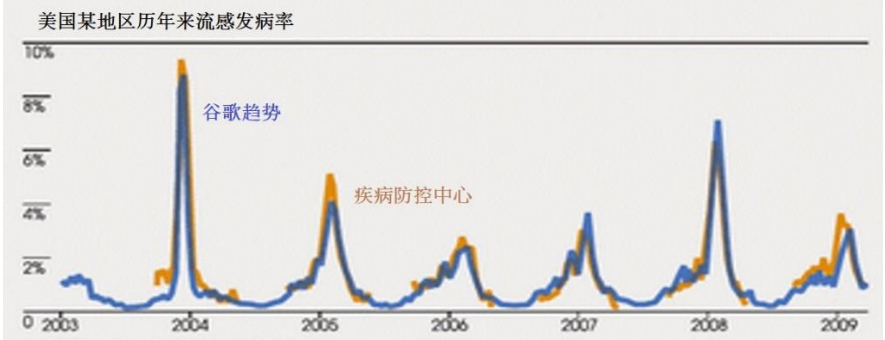

图 1 信息来自谷歌趋势与美国某地疾病防控中心

3.2 动态高速处理

这条特点与大数据的高通量紧密相关。人们在决策过程中的每一步行动，都会影响甚至改变事物的发展，并全程由大数据所反映。问题的描述以及求解的策略都需要进行及时调整，进而反馈到决策执行当中，因而大数据智能化需要有动态处理能力；需要高速处理能力，主要是由于数据量的庞大：如今的电商平台与短视频平台，日常处理 PB 级别的数据已成常态化。

4 机遇与挑战

目前的机器学习与大数据智能主要是规则下的智能学习或者是单纯地进行分类分析，而聚类分析是人类智能分析中较低层次的一种行为，要使机器学习具备更高层次的认知能力，还有很长的路要走。

张钹院士指出，人类在问题求解中具有天生的知识驱动能力、对全局整体的感知能力和对不确定性问题的处理优势。机器学习的优势在于高速计算能力，两者结合是未来发展的必然趋势。[7]

参考资料

[1] 行业前沿：互联网人必须读懂的”数据智能”, www.woshipm.com/ai/3702180.html, April 14, 2020.

[2] World Economic Forum. Big data, big impact: new pos-sibilities for international development [online], available:http://www3.weforum.org/docs/WEFTCMFSBigDataBig ImpactBriefing2012. pdf, April 12, 2019.

[3] Tolle K M, Tansley D S W, Hey A J G. The fourthparadigm: data-intensive scientific discovery.Proceedingsof the IEEE, 2011,99(8): 1334−1337.

[4] United Nations Global Pulse. Big data for develop-ment: opportunities and Challenges–White Paper[online], available: http://www.unglobalpulse. org/sites/default /files/BigDataforDevelopment-UNGlobalPulseJune2012.pdf, April 12, 2019.

[5] Liang Ji-Ye, Feng Chen-Jiao, Song Peng. A survey on cor-relation analysis of big data.Chinese Journal of Comput-ers, 2016,39(1): 1−18.

[6] Ginsberg J, Mohebbi M H, Patel R S, Brammer L, Smolin-ski M S, Brilliant L. Detecting influenza epidemics us-ing search engine query data.Nature, 2009,457(7232):1012−1014.

[7] Zhang B, Zhang L. Multi-granular computing in web age.In: Proceedings of the 14th International Workshop onRough Sets, Fuzzy Sets, Data Mining, and Granular-SoftComputing. Berlin, Heidelberg: Springer, 2013. 11−14.

报告五题目：自动驾驶现状浅析

1 概念分析

自动驾驶汽车搭载了先进的车载传感器、控制器、执行器等装置，网络通信技术上，采用了车内网、车际网和车载移动互联网等融合的方式。

智能网联汽车实现了车与人、车、路、后台等智能信息交换共享，能够综合感知复杂的行车环境、进行水平不低于甚至远高于普通驾驶人的智能决策、与驾驶人协同对汽车进行控制，最终实现安全、舒适、节能高效的驾驶。[1]长远来看，智能网联汽车将能够替代人在驾驶中的作用。

2 部分技术问题的解决方案

2.1 复杂交通环境感知

复杂环境感知主要包括基于单目视觉、基于立体视觉或是基于激光雷达的视觉感知，基于单传感器或多传感器融合的环境感知等。[2]

2.1.1 激光雷达感知

激光雷达可分为三类：[2]

1.单线激光雷达：一层扫描面，在一定范围内获取线性扫描点。响应时间短、可以快速扫描目标，但获取的数据量较少，无法详细描述目标信息。

2.多线局部激光雷达：扫描点密度更高，有多层扫描面，一定程度上能反映目标物的外形信息，但范围较窄，需要安装多个。

3.多线全视场激光雷达：全方位扫描周围环境，能获取丰富的点云信息，完整描述三维场景，但要求较高的实时处理能力。

扫描到需要的信息后，雷达信号经过点云聚类分析、可通行区域分析、障碍物识别、障碍物跟踪等步骤，得到周围环境的较全面感知信息。

2.1.2 车载摄像机感知

车载摄像机感知系统用于道路场景中的静态目标与动态目标检测，是感知系统的重要组成部分。以斯坦福大学研发的 Shelley 智能车为例，摄像头是其感知系统中不可或缺的极重要组成部分。[3]

摄像机获取的图像信息经过预处理、目标检测以及目标识别步骤，可以识别出交通标志、车道线、交通信号灯、行人、车辆等交通重要参与者或信息要素。

2.2 车辆导航定位

2.2.1 高精度地图

精度较低的电子导航地图，地图信息的完整性、准确性难以满足未来人们交通出行的需求。高精度地图能够辅助车辆实现高精度定位、解决特定情况下传感器失效的问题、弥补环境感知设备的不足。根据地图信息，可以给出最佳的行驶路径和合理的行驶策略，有效实现驾驶安全。因此，高精度地图将是未来智能出行关键因素之一。中国最早从 2010 年有高精度地图相关的研发报道[4]，2016 年高精度地图产业开始爆发。

2.2.2 精确定位技术

关于车辆高精度定位研究主要集中在卫星定位增强系统、超宽带无线定位技术、SLAM 定位等领域。

1.卫星基增强系统技术：卫星搭载卫星导航增强信号转发器，可以向用户播发星历误差、卫星钟差、电离层延迟等多种修正信息，实现对于原有卫星导航系统定位精度的改进。

2.超宽带无线定位技术。

3.SLAM 定位技术：在未知环境中移动时，根据传感器信息创建地图，利用创建出的地图进行定位。[5]

2.3 汽车路径规划

路径规划中主要建模方法有：路线图法、网格分解法等。

1.路线图法：先生成无碰路径网络，再利用搜索算法在网络中找出无碰路径序列。障碍物少时，效率较高。适合在条件已知的室内环境提取一些简单的障碍物几何特征。[6]

2.网格分解法：将物理空间划分一系列网格，划分简单、更新容易，但对障碍物的建模不准确。

3. 存在的问题

3.1 全天候感知

对计算机来说，晴天时的道路与雾天/黄昏时截然不同。施工区道路等的复杂状况又增加了新的挑战。

3.2 人机交互

半自动驾驶需要人的干预。车辆如何有效地通知一位正在阅读或小睡的驾车人，车辆如何确认驾驶人已准备好接手驾驶任务，这些都是亟待解决的问题，需要系统与认知学的发展。

3.3 事故责任划分

随着自动化程度的提升，驾驶人的担责比例下降，厂家的责任比例上升。车辆保险的结构及责任认定需要进行相应的调整。

3.4 潜在的伦理危机

车辆失控时，选择撞向外部行人以保障车主安全，还是撞向固定障碍物损害车主利益、保障行人安全。如果把某种选择简单地编入自动驾驶的行为规则中，将会引发潜在的社会伦理问题。[7]

参考资料

[1] 戴一凡,李克强. 智能网联汽车发展现状与趋势分析[J]. 汽车制造业,2015(18):14-17.

[2] 《中国公路学报》编辑部."中国汽车工程学术研究综述·2017." 中国公路学报 30.06(2017):1-197.

[3] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., . . . Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the Urban Challenge. Journal of Field Robotics, 25(8), 425-466.

[4] 四维图新发布 2010 年度地图产品[J]. 数字通信世界,2010(10):37.

[5] Jorge Fuentes-Pacheco; José Ruiz-Ascencio; Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. Artificial Intelligence Review. 43, 55–81. 2015.

[6] 陈慧岩. 无人驾驶汽车概论. 智能车辆先进技术丛书 Zhi Neng Che Liang Xian Jin Ji Shu Cong Shu. 北京: 北京理工大学出版社, 2014.

[7] RAHWANI. Moral Machine[EB/OL]. [2017-05-28] https://www.moralmachine.net/

报告六题目：GCC 编译器简介

1 诞生

1983 年底，为了引导 GNU 操作系统， Richard Stallman 向阿姆斯特丹编译器套件（自由大学编译器套件）的作者 Andrew S. Tanenbaum 请求允许将阿姆斯特丹编译器套件运用于 GNU；但是作者告知他该编译器仅对大学免费。因此，Richard Stallman 打算开发一个不同的编译器。[1]

GCC 于 1987 年 3 月 22 日在麻省理工学院的文件传输协议上发布。Richard Stallman 被列为作者。

GCC 编译器发布正值太阳微系统将其操作系统与其开发工具解绑，并提价单独出售。这使得许多客户购买或下载 GCC 而非供应商的工具。[2]

2 发展

只要遵守 GPL 许可条款，其他程序员可以自由地以 C 以外语言编写接口，开发自己的编译器分支。但是，多分叉在日后体现出低效和不便的特点。于是，FSF 对添加到官方版本 GCC 2.x（1992 年开始开发）中的内容进行了相当严格的控制。[3]

1997 年，一群不满 GCC 缓慢且封闭的创作环境者，组织了一个名为实验性/增强型 GNU 编译器系统（EGCS）的专案，他们集成了多种分叉版本的优势，进行了许多 C++ 的改进，添加了许多新的架构和操作系统变种。随着时间推移，EGCS 的发展明显更活跃，因此 FSF 正式停止他们对官方版本编译器的开发，并希望 EGCS 成为 GCC 的官方版本。

1999 年 4 月，EGCS 项目被任命为为 GCC 的维护者。

3 支持语言[4][5][6]

<table>
<tr><td>语言</td><td colspan="2">备注</td></tr>
<tr><td>C 语言</td><td colspan="2">gcc</td></tr>
<tr><td rowspan="3">C++</td><td>C++17</td><td></td></tr>
<tr><td>C++20</td><td>实验性部分支持</td></tr>
<tr><td>C++23</td><td>实验性部分支持</td></tr>
<tr><td>Objective-C</td><td colspan="2"></td></tr>
<tr><td>Fortran</td><td colspan="2">gfortran</td></tr>
<tr><td>Ada</td><td colspan="2">GNAT</td></tr>
<tr><td>Go</td><td colspan="2">gccgo</td></tr>
<tr><td>D</td><td colspan="2">gdc，从 9.1 版开始</td></tr>
<tr><td>Java</td><td colspan="2">gcj，GCC 7 以前的版本支持</td></tr>
<tr><td>Pascal</td><td colspan="2">第三方前端</td></tr>
<tr><td>Modula-2、3</td><td colspan="2">第三方前端</td></tr>
<tr><td>Mercury</td><td colspan="2">第三方前端</td></tr>
<tr><td>UPC</td><td colspan="2">实验性分支</td></tr>
</table>

Rust	实验性分支

4 支持芯片架构

CC 11.1 版本支持的常见处理器包括：[7]

处理器	备注
AArch64	
Alpha	
ARM	
AVR	
Blackfin	
eBPF	
Epiphany	GCC 4.8
H8/300	
HC12	
IA-32	x86
IA-64	
MIPS	
Motorola 68000	
MSP430	
Nvidia GPU	
PA-RISC	
PDP-11	
PowerPC	
R8C / M16C / M32C	
RISC-V	

5 贡献者

GNU 编译器套件是一项持续了数十年的伟大开源工程，期间做过贡献的人数不胜数。从第一个版本开始，其贡献者就不只是作者 Richard Stallman 一个人。在这项工程的官网上，记录的贡献者就达到了 500 人的规模，[8]更不必说世界各地的小型插件/拓展编写者和 bug 反馈者了。鉴于信息量大且本篇篇幅有限，官网列出的贡献者名单将作为附录一，置于报告册尾部。

参考资料

[1] Wall, Kurt, and William Von Hagen. The Definitive Guide to GCC. Berkeley, CA: Apress L. P, 2008.
[2] Salus, Peter H. Chapter 10. SUN and gcc. The Daemon, the Gnu and the Penguin. Groklaw. 2005.
[3] Henkel-Wallace, David, A new compiler project to merge the existing GCC forks, August 15, 1997
[4] GCC 7 Release Series: Changes, New Features, and Fixes. https://gcc.gnu.org/gcc-7/changes.html
[5] GCC Front Ends. https://gcc.gnu.org/frontends.html
[6] Open Source Security, Inc. Announces Funding of GCC Front-End for Rust. https://opensrcsec.com/open_source_security_announces_rust_gcc_funding
[7] Option Summary (Using the GNU Compiler Collection (GCC)). https://gcc.gnu.org/onlinedocs/gcc-10.2.0/gcc/Option-Summary.html#Option-Summary
[8] Contributors to GCC. https://gcc.gnu.org/onlinedocs/gcc-12.1.0/gcc/Contributors.html

报告七题目：人工智能与隐私安全：担忧与解决方案

1 人工智能正侵犯隐私安全

1.1 实例：DeepNude 软件

2019 年 6 月，一键脱衣软件“DeepNude”受到大量的关注。据英国 Wired 杂志的统计数据，至少有 10.4 万名妇女受到深度伪造机器人“DeepNude”的攻击。大量的伪造色情信息被投放到各种网站上。[1]“DeepNude”软件使用深度学习和生成对抗网络生成受害者的裸体图片，其训练用的数据集建立在真实存在的裸照数据库上，它能够对图像中的人物进行识别，并且用隐私部位的图像取代图片中的衣物。用户不需要任何计算机知识，即可使用该软件：只需点击上传一张图片，程序就会在 30 秒内返回一张裸照。

1.2 更多的担忧

“DeepNude”软件引发的隐患是明显的、直接的：它本就是为了侵犯他人隐私而存在的，相比之下，有更多的对隐私的侵犯是在不知不觉间进行的，这正是人工智能安全隐患的核心，也是人们最大的担忧点。英国 Wired 杂志就曾报道过安装了语音识别技术的智能语音助手对其拥有者的监听。[2]当人工智能的载体越发与人类似后，人们有可能对其产生足够的信任，在与其主动“谈心”的时候将自己的隐私分享出去。[3]另外，随着时代发展，我们的生活中智能设备的介入越来越多。当一个智能设备参与生活到达一定比例后，它所收集的信息就足以进行精准的“画像识别”，分析得到一个人的生活的每一个方面，：手机、智能汽车、家庭机器人等，都有着这样的安全隐患。[4]

2 可能的解决方案

2.1 立法限制公司行为

鉴于目前的安全隐患主要来自于公司后台行为，立法限制公司行为是一条行之有效的道路。欧盟于 2016 年通过《一般数据保护法》，2018 年 5 月 25 日正式生效，欧盟多数成员国也陆续通过了该项法案。本法案对“数据控制者利用人工智能技术对个人信息进行完全或部分自动化处理的行为”进行了控制。这项法案增加了人工智能信息获取的透明度，拓展用户权利，并且禁止对个人的敏感隐私信息进行自动或半自动处理。

我国在这一方面还有较长的路要走：我国在历史上缺少隐私保护的概念，相比于上世纪上半叶就将隐私权纳入宪法保障的基本权利的美国，在我国，直到 2009 年《侵权责任法》出台，隐私权才真正被作为一项独立的具体人格权来保护。目前我国没有专门的《个人信息保护法》，而信息时代亟需这样的隐私保护法律，欧美国家的优秀经验值得我们借鉴。

2.2 对程序本身行为进行限制

加拿大渥太华省信息与隐私委员会前主席 Ann Cavoukian 女士于上世纪 90 年代提出“隐私设计”概念，主张在程序设计时就将隐私保护作为一项设计需求嵌入其中。[5] 这是一项十分具有长远目光的想法，但为了实现它，我们需要政府立法对科技公司、开发者进行必要的行为限制与专业的审核。欧盟《一般数据保护法》明确引入了隐私设计理论：首先确立隐私设计原则，其次确定隐私默认保护义务，最后规定了隐私影响评估义务，形成了一套完整的方法论。欧盟《一般数据保护法》的进步性举措有希望解决人工智能对隐私权的侵犯问题。[6]

参考资料

[1] 徐燕萍. "深度伪造"技术的伦理反思 ——以一键脱衣软件"Deepnude"为例[J]. 计算机时代,2021(11):118-121.

[2] Tim Moynihan. Alexa and Google Home Record What You Say. But What Happens to That Data?[N].Wired, November 5, 2016.

[3] Kaori Ishii. Comparative Legal Study on Privacy and Personal Data Protection for Robots Equipped with Artificial Intelligence: Looking at Functional and Technological Aspects [J].AI & Soc, 2017, (32) :7-20

[4] John Frank Weaver. Robots Are People Too: How Siri, Google Car, and Artificial Intelligence Will Force Us to Change Our Laws [M].Praeger, 2014.

[5] Ann Cavoukian. Privacy by Design [R]. Information and Privacy Commissioner, Ontario, Canada, 2013.

[6] 郑志峰.人工智能时代的隐私保护[J].法律科学(西北政法大学学报),2019,37(02):51-60.

报告八题目：超视觉技术

1 超视觉技术概念

超视觉技术泛指多种利用软件、硬件或是软硬件协同，从而实现在某一维度或是多个维度超越人类视觉感知边界的技术。超视觉技术的应用对象是客观世界的复杂多样的可成像数据。

对视觉能力的部分评价维度有：视场角、最高帧率、可视波长范围、动态范围等指标，人类在这些指标上的边界，主要来自于于人眼的物理极限和大脑的处理能力极限。比如，人眼的视场角不大，水平方向双眼视场角略大于 180 度，垂直方向约为 135 度，但能够集中注意力观察的视场角在四个方向上一般均不超过 30 度，即舒适视场角为水平不超过 60 度、垂直不超过 60 度[1]。

以上这些评价维度都是基于原始信息获取而提出的，事实上，视觉能力的更重要的方面是对于原始信息的分析能力。快速地从图像信息分析出物品种类、尺寸、材质，构建表面和拓扑结构等，显然对人类更加有用。

超视觉技术要做的，就是在这些维度上超越人类视觉。

2 实现方式

计算机外设可以进行拓展和改造，软件可以进行后期处理，因此为了实现超视觉，主要有两条路径，可以单选其一，也可以混合使用：提高原始信息的指标，进行后期算法处理。

3 技术实现与应用

3.1 图像分类

计算机观察图像并进行分类，例如在一幅图像中识别出多个物体并为它们分别打上标签，理想状况下，它能够准确地预测指定图像属于哪个特定类别。

深度卷积神经网络（CNN）为图像分类带来了一系列突破，对图像特征的识别级别可以通过堆叠层数（深度）来丰富。

然而，更深层次的神经网络更难训练，并且因为深度的提升带来的更多次权值乘积，存在梯度消失/爆炸（vanishing/exploding gradients）问题[2]。这个问题已经“用规范化初始化和中间归一化层很大程度地解决了”[3]，但是更深层次的神经网络依然难以训练。随着网络深度的增加，精度变得饱和，然后迅速下降，并且向深度合适的模型添加更多层会导致更高的训练误差[4]。

在 2016 年，何恺明、张祥雨、任少卿和孙剑发表论文 Deep Residual Learning for Image Recognition，提出残差网络 ResNet，大大提升了包括图像分类、目标检测和人脸识别在内的许多计算机视觉应用的性能。[3]

3.2 目标跟踪

目标跟踪就是在已知初始帧目标大小与位置的情况下，预测后续帧中该目标的大小与位置。这一技术目前广泛应用在监控、感知用户界面、增强现实、基于对象的视频压缩和驾驶员辅助等领域。

对运动的目标而言，高速运动、复杂场景、遮挡、形变、尺度变换等因素都是识别并跟踪目标的挑战性任务。

近年来，传统的光流法等追踪算法逐渐被淘汰，目前效果较好的算法的判别方法大多为相关滤波、深度学习或者是二者的结合。

3.2.1 相关滤波

研究人员认为早期的算法每帧只选取随机的几个样本的做法是可取的，但是采样不足也成了最重要的性能抑制因素，为了同时提高运算速度与准确度，核相关滤波（KCF）应运而生，它在运算速度上实现了巨大的突破[5]。后来，为了应对更多特定的需求，在 KCF 的基础上，又有许多改良方法被提出，包括能够更好地处理尺度变化的 DSST[6]、基于修补程序的用于解决遮挡情况和长时间跟踪的相关滤波算法[7]、以及为克服边界效应而提出的 SRDCF[8]。

3.2.2 深度学习

目标跟踪任务有其特殊性，只有初始帧数据可以利用，因为缺乏大量数据供神经网络学习；且神经网络擅长区分类间差异、忽视类内的区别，而目标跟踪恰恰要区分特定目标与背景、抑制同类目标，因此早期深度学习在目标跟踪方面并没有发挥出大的效用[9]。后来研究人员将 CNN 迁移到目标跟踪中来，基于深度学习的目标跟踪方法得到充分的发展。这一领域较热门的方法是将 CNN 与相关滤波结合，充分利用两者优点。代表案例有基于全卷积网络（FCN）的多层特征融合端到端跟踪[10]，以 KCF 为跟踪框架、以神经网络特定层的输出为特征提取信息的 HCF 算法[11]，多域卷积神经网络 MDNet[12]（取得 VOT2015 比赛第一名）、利用卷积神经网络提取目标特征结合 SRDCF 的 C-COT[13]（取得 VOT2016 比赛第一名）。

参考资料

[1] Robert H. Spector (1990). "Visual Fields". Clinical Methods: The History, Physical, and Laboratory Examinations. 3rd edition. Butterworths. ISBN 9780409900774.

[2] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," in IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157-166, March 1994.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[4] K. He and J. Sun, "Convolutional neural networks at constrained time cost," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5353-5360.

[5] J. F. Henriques, R. Caseiro, P. Martins and J. Batista, "High-Speed Tracking with Kernelized Correlation Filters," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 3, pp. 583-596, 1 March 2015.

[6] C. Liu, J. Gong, J. Zhu, J. Zhang and Y. Yan, "Correlation Filter With Motion Detection for Robust Tracking of Shape-Deformed Targets," in IEEE Access, vol. 8, pp. 89161-89170, 2020.

[7] Z. Qu, X. Lv, J. Liu, L. Jiang, L. Liang and W. Xie, "Long-term reliable visual tracking with UAVs," 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017, pp. 2000-2005.

[8] M. Danelljan, G. Häger, F. S. Khan and M. Felsberg, "Learning Spatially Regularized Correlation Filters for Visual Tracking," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 4310-4318.

[9] "目标跟踪简介," https://www.cnblogs.com/jjwu/p/8512730.html. 2018.

[10] Y. Kuai, G. Wen and D. Li, "Learning Fully Convolutional Network for Visual Tracking With Multi-Layer Feature Fusion," in IEEE Access, vol. 7, pp. 25915-25923, 2019.

[11] C. Ma, J. -B. Huang, X. Yang and M. -H. Yang, "Hierarchical Convolutional Features for Visual Tracking," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 3074-3082.

[12] H. K. Galoogahi, A. Fagg and S. Lucey, "Learning Background-Aware Correlation Filters for Visual Tracking," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1144-1152.

[13] Danelljan, Martin, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. "Beyond Correlation Filters: Learning Continuous Convolution Operators for Visual Tracking." Computer Vision – ECCV 2016, 2016, 472-88.

附录一　GNU 编译器套件官网登记贡献者

GNU 编译器套件官网登记贡献者，不包含对特定版本的致谢

更新至 2022 年 11 月前

Analog Devices	帮助实现了对复杂数据类型和迭代器的支持
John David Anglin	对 libstdc++-v3 和 HP-UX 端口进行了与线程相关的修复和改进
James van Artsdalen	编写了有效使用 Intel 80387 寄存器堆栈的代码
Abramo , Roberto Bagnara	为 SysV68 摩托罗拉 3300 Delta 系列移植
Alasdair Baird	修复了各种错误
Giovanni Bajo	用于分析大量复杂的 C++ 问题报告
Peter Barada	为改进新 ColdFire 内核的代码生成所做的工作
Gerald Baumgartner	将签名扩展添加到 C++ 前端
Godmar Back	对 Java 的改进
Scott Bambrough	帮助移植 Java 编译器
Wolfgang Bangerth	处理了大量错误报告
Jon Beniston	提供了 Java 的 Microsoft Windows 端口和 Lattice Mico32 的端口
Daniel Berlin	提供更好的 DWARF 2 支持、更快更好的优化、改进的别名分析，以及将 GCC 迁移到 Bugzilla
Geoff Berry	Java 对象序列化工作和各种补丁
David Binderman	针对多种架构针对 Fedora Rawhide 每周测试 GCC 主干快照
Laurynas Biveinis	负责内存管理工作和 DJGPP 端口修复
Uros Bizjak	实现用于 x87 数学内置函数，以及各种中间端和 i386 后端改进和错误修复
Eric Blake	帮助使 GCJ 和 libgcj 符合规范
Janne Blomqvist	对 GNU Fortran 的贡献
Hans-J. Boehm	垃圾收集器、IA-64 libffi 端口和其他 Java 工作
Segher Boessenkool	帮助维护 PowerPC 端口和指令组合器以及对中端的各种贡献
Neil Booth	负责 cpplib、lang hooks、debug hooks 和其他杂项清理工作
Steven Bosscher	将 GNU Fortran 前端集成到 GCC 中并为 tree-ssa 分支做出贡献
Eric Botcazou	修复后端错误
Per Bothner	通过指导委员会的指导以及对支持新语言的基础设施的各种改进；冷却前端实现，cpplib、fix-header、config.guess、libio 和过去的 C++ 库 (libg++) 维护者的初始实现；构思、设计和实施 GCJ 的大部分内容。
Devon Bowen	帮助将 GCC 移植到 Tahoe
Don Bowman	对 mips-vxworks 进行了贡献
James Bowman	贡献 FT32 端口
Dave Brolley	在 cpplib 和 Chill 方面进行了工作
Paul Brook	致力于 ARM 架构和维护 GNU Fortran
Robert Brown	实现了对 Encore 32000 系统的支持
Christian Bruel	对本地存储清除的改进
Herman A.J. ten Brugge	各种修复
Joerg Brunsmann	Java 编译器黑客攻击并帮助解决 GCJ 常见问题解答
Joe Buck	指导委员会从成立到 2013 年的指导
Iain Buclaw	对 D 前端的贡献

Craig Burley	领导 G77 Fortran 工作
Tobias Burnus	对 GNU Fortran 的贡献
Stephan Buys	为 libstdc++ 贡献了 Doxygen 注释
Paolo Carlini	进行 libstdc++ 工作：对 C++ 字符串、streambufs 和格式化 I/O 的大量效率改进，对令人沮丧的本地化问题进行艰苦的侦探工作，并跟上问题报告
John Carr	别名工作、SPARC 黑客攻击、基础设施改进、之前对指导委员会的贡献、循环优化等
Stephane Carrez	68HC11 和 68HC12 端口
Steve Chamberlain	对 Renesas SH 和 H8 处理器以及 PicoJava 处理器的支持，以及 GCJ 配置修复
Glenn Chambers	寻求 GCJ 常见问题解答方面的帮助
John-Marc Chandonia	各种 libgcj 补丁
Denis Chertykov	贡献并维护了 AVR 端口，这是第一个 8 位架构的 GCC 端口
Kito Cheng	在 RISC-V 端口上的工作，包括提出测试套件和维护
Scott Christley	对 Objective-C 的贡献
Eric Christopher	Java 移植帮助和清理工作
Branko Cibej	贡献了更多警告
GNU Classpath 项目	合并了运行代码
Nick Clifton	arm，mcore，fr30，v850，m32r，msp430 rx work，和其他随机黑客攻击
Michael Cook	为 libstdc++ 清理补丁以减少警告
R. Kelley Cook	使 GCC 可从只读目录构建以及其他杂项构建过程和文档清理
Ralf Corsepius	SH 测试和小错误修复
François-Xavier Coudert	对 GNU Fortran 进行了贡献
Stan Cox	负责 x86 端口的维护和供给以及应对大量的幕后黑客攻击
Alex Crain	为 3b1 提供了更改
Ian Dall	为 NS32k 端口进行重大改进
Paul Dale	为 m68k 后端添加 uClinux 平台支持做了工作
Palmer Dabbelt	维护 RISC-V 端口
Dario Dariol	贡献了四种示例程序，这些程序可以打印其源代码的副本
Russell Davidson	libstdc++ 中的 fstream 和 stringstream 修复
Bud Davis	在 G77 和 GNU Fortran 编译器方面进行的工作
Mo DeJong	GCJ 和 libgcj 的错误修复
Jerry DeLisle	对 GNU Fortran 进行了贡献
DJ Delorie	DJGPP 端口、构建和自由维护、各种错误修复以及 M32C、MeP、MSP430 和 RL78 端口
Arnaud Desitter	帮助调试 GNU Fortran
Gabriel Dos Reis	对 G++ 的贡献，GCC 诊断基础设施的贡献和维护，libstdc++-v3，包括 valarray<>，complex<>维护数字库（包括那个讨厌的<limits> :-) ，并保持
Ulrich Drepper	在 glibc 方面的工作、使用 glibc 测试 GCC、ISO C99 支持、CFG 转储支持等，以及对 C++ 运行时库的支持，包括针对各种 C 接口问题、贡献和维护 complex<>、健全性检查和支付，配置体系结构、libio 维护和早期数学工作
François Dumont	在 libstdc++-v3 方面的工作，特别是维护和改进 debug-mode 以及关联和无序容器
Zdenek Dvorak	用于新的循环展开器和各种修复
Michael Eager	在 Xilinx MicroBlaze 端口上的工作
Richard Earnshaw	在 ARM 的持续工作

David Edelsohn	通过指导委员会指导，进行 RS6000/PowerPC 端口的工作，帮助清理 Haifa 循环更改，徒手完成 libstdc++ 的整个 AIX 端口，并确保 GCC 正确地继续在 AIX 上工作
Kevin Ediger	libstdc++ 中 num_put::do_put 的浮点格式
Phil Edwards	负责 libstdc++ 的工作，包括配置黑客、文档维护者、网页的主要破坏者、偶尔的 iostream 错误修复，以及共享库符号版本控制方面的工作
Paul Eggert	对整个 GCC 的随机黑客攻击
Mark Elbrecht	各种 DJGPP 改进，以及 libstdc++ 配置支持语言环境和与 fstream 相关的修复
Vadim Egorov	为 libstdc++ 修复了字符串、streambufs 和 iostreams
Christian Ehrhardt	处理错误报告
Ben Elliston	帮助实现将 Objective-C 运行时移动到它自己的子目录中，以及在 autoconf 方面的工作
Revital Eres	PowerPC 750CL 端口
Marc Espie	OpenBSD 支持
Doug Evans	大部分全局优化框架、arc、m32r 和 SPARC 工作
Christopher Faylor	在 Cygwin 端口上的工作，以及关心和维护 gcc.gnu.org 盒子并为用户减少大量垃圾邮件
Fred Fish	BeOS 支持和 Ada 修复
Ivan Fontes Garcia	负责 GCJ FAQ 的葡萄牙语翻译
Peter Gerwinski	各种错误修复和 Pascal 前端
Kaveh R. Ghazi	通过指导委员会进行指导，做出了惊人的工作。 '-W -Wall -W* -Werror' 很有用，并在大量平台上测试 GCC（Kaveh 本人对 Rutgers 大学的 CAIP 中心表示感谢，感谢它为他提供了从 1980 年代后期到 2010 年从事自由软件工作的计算资源）
John Gilmore	向 FSF 捐款，指定用于改进 GNU Java
Judy Goldberg	对 c++ 的贡献
Torbjorn Granlund	各种修复和 c-torture 测试套件、乘法和除以常数优化、改进 long long 支持、改进叶函数寄存器分配，以及通过指导委员会进行指导
Jonny Grant	改进 collect2's - 帮助文档
Anthony Green	操作系统贡献、moxie 端口和 Java 前端工作
Stu Grossman	gdb hacking，允许 GCJ 开发人员调试 Java 代码
Michael K. Gschwind	为 PDP-11 贡献了端口
Richard Biener	持续的中端贡献和错误修复以及发布管理
Ron Guilmette	实现了 protoize 工具 unprotoize、对 DWARF 1 符号调试信息的支持，以及对 System V Release 4 的大部分支持。他还在 Intel 386 和 860 支持方面做了大量工作
Sumanth Gundapaneni	CR16 端口
Mostafa Hagog	Swing Modulo Scheduling (SMS) 和 post reload GCSE
Bruno Haible	改进了 EH 的运行时开销、新警告，进行了各种错误修复
Andrew Haley	惊人的 Java 编译器和库工作
Chris Hanson	协助 GCC 在 HP-UX 上为 9000 系列 300 工作
Michael Hayes	为使 c30/c40 端口正常工作，做了各种吃力不讨好的工作；进行了许多循环和展开的改进和修复
Dara Hazeghi	浏览了无数针对特定目标的错误报告
Kate Hedstrom	为 G77 人员提供了初始测试套件
Richard Henderson	SPARC、alpha、ia32 和 ia64 工作，loop opts，以及修复许多我们多年来忽略的老问题、流程重写和许多进一步的东西，如审查大量补丁

Aldy Hernandez	致力于 PowerPC 端口、SIMD 支持和各种修复
Nobuyuki Hikichi	来自东京软件研究协会。为 Sony NEWS 机器提供了支持
Kazu Hirata	负责 Renesas H8/300 端口的维护和修复以及各种修复
Katherine Holcomb	GNU Fortran 方面的工作
Manfred Hollstein	为保持 m88k 的活力做了持续工作；大量的测试和错误修复，尤其是 GCC 配置代码
Steve Holmgren	MachTen 补丁
Mat Hostetter	处理 TILE-Gx 和 TILEPro 端口
Jan Hubicka	x86 端口改进
Falk Hueffner	致力于 C 和优化错误报告
Bernardo Innocenti	m68k 工作，包括合并 ColdFire 改进和 uClinux 支持
Christian Iseli	修复了各种错误
Kamil Iskra	一般的 m68k 黑客攻击
Lee Iverson	进行随机修复和 MIPS 测试
Balaji V. Iyer	负责 Cilk+ 开发和合并
Andreas Jaeger	GCC 的测试和基准测试以及各种错误修复
Martin Jambor	在过程间优化、开关转换过程和聚合标量替换方面的工作
Jakub Jelinek	SPARC 工作和兄弟调用优化，以及大量错误修复和测试用例，以及改进 Java 构建系统
Janis Johnson	ia64 测试和修复、质量改进旁路和网页维护
Kean Johnston	SCO OpenServer 支持和各种修复
Tim Josling	示例语言 treelang——最初基于 Richard Kenner 的“玩具”语言
Nicolai Josuttis	额外的 libstdc++ 文档
Klaus Kaempf	使 alpha-vms 成为可行目标做了持续工作
Steven G. Kargl	在 GNU Fortran 方面的工作
SRI 的 David Kashtan	将 GCC 改编为 VMS
Ryszard Kabatek	许多 libstdc++ 错误修复和字符串优化（尤其是成员函数），以及 auto_ptr 修复
Geoffrey Keating	正在进行的使 PPC 为 GNU/Linux 工作的工作和他的自动回归测试器
Brendan Kehoe	在 G++ 方面的持续工作，以及 libstdc++ 几乎每个部分的大量早期工作
Oliver M. Kellogg	来自德国宇航公司。为 MIL-STD-1750A 贡献了端口
Richard Kenner	来自纽约大学超级计算机研究实验室。编写了 AMD 29000、DEC Alpha、IBM RT PC 和 IBM RS/6000 的机器描述以及对指令属性的支持。他还进行了更改以更好地支持 RISC 处理器，包括对公共子表达式消除、强度降低、函数调用序列处理和条件代码支持的更改，此外还概括了帧指针消除和延迟槽调度的代码。Richard Kenner 多年来一直是 GCC 的首席维护者
Mumit Khan	对 Cygwin 和 Mingw32 端口的各种贡献和维护 Microsoft Windows 主机的二进制版本，以及对 Cygwin/Mingw32 的大量 libstdc++ 移植工作
Robin Kirkham	cpu32 支持
Mark Klein	PA 改进
Thomas Koenig	修复了各种错误
Bruce Korb	改进的 fixincludes 代码
Benjamin Kosnik	G++ 工作和领导和关于 libstdc++-v3 的工作
Maxim Kuvyrkov	对指令调度程序、Android 和 m68k/Coldfire 端口以及优化的贡献
Charles LaBrec	贡献了对 Integrated Solutions 68020 系统的支持
Asher Langton, Mike Kumbera	为 GNU Fortran 贡献了 Cray 指针支持，以及其他 GNU Fortran 改进

Jeff Law	通过指导委员会进行指导，协调整个 egcs 项目和 GCC 2.95，推出快照和发布，处理来自 GCC2 的合并，审查大量可能已被其他漏洞遗漏的补丁，以及应对随机但广泛的黑客攻击
Walter Lee	在 TILE-Gx 和 TILEPro 端口上的工作
Marc Lehmann	通过指导委员会指导并帮助分析和改进 x86 性能
Victor Leikehman	在 GNU Fortran 方面的工作
Ted Lemon	编写了部分 RTL 阅读器和打印机
Kriang Lerdsuwanakij	C++ 改进，包括模板作为模板参数支持，以及许多 C++ 修复
Warren Levy	在 libgcj（Java 运行时库）方面的大量工作以及在 Java 前端的随机工作
Alain Lichnewsky	将 GCC 移植到 MIPS CPU
Oskar Liljeblad	对 AWT 的黑客攻击以及他的许多 Java 错误报告和补丁
Robert Lipe	OpenServer 支持、新测试套件、测试
Chen Liqin	各种与 S+core 相关的修复/改进，以及维护 S+core 端口
Martin Liska	相同代码折叠、杀毒功能、HSA、一般错误修复以及运行 GCC 自动回归测试和报告大量错误方面的工作
Weiwen Liu	测试和各种错误修复
Manuel López-Ibáñez	改进-W 转换以及许多其他诊断修复和改进
Dave Love	在 Fortran 前端和运行时库方面的持续工作
Martin von Löwis	内部一致性检查基础设施、包括名称空间支持在内的各种 C++ 改进，以及对 libstdc++/编译器合并的大量帮助
HJ Lu	对指导委员会的贡献、许多 x86 错误报告、原型补丁以及保持 GNU/Linux 端口的工作
Greg McGary	随机修复和有界指针
Andrew MacLeod	在构建真正的 EH 系统、各种代码生成改进、全局优化器等方面的持续工作
Vladimir Makarov	破解了一些丑陋的 i960 问题，PowerPC 破解编译时性能的改进，指令调度领域的整体知识和方向，基于自动机的指令调度器的设计和实现以及集成和本地寄存器分配器的设计和实现
David Malcolm	改进 GCC 诊断、JIT、自测试和单元测试
Bob Manson	dejagnu 的幕后工作
John Marino	贡献了 DragonFly BSD 端口
Philip Martin	进行了大量 libstdc++ 字符串和向量迭代器修复和改进，以及字符串清理和测试套件
Michael Matz	在优势树发现、x86-64 端口、链接时优化框架和一般优化改进方面的工作
所有 Mauve 项目贡献者	Java 测试代码
Bryce McKinlay	对许多 GCJ 和 libgcj 的修复和改进
Adam Megacz	在 GCJ 的 Microsoft Windows 端口上的工作
Michael Meissner	负责 LRS 框架、ia32、m32r、v850、m88k、MIPS、powerpc、haifa、ECOFF 调试支持和其他各种黑客攻击
Jason Merrill	通过指导委员会进行指导并领导 G++ 的工作
Martin Michlmayr	使用整个 Debian 档案库在多个架构上测试 GCC
David Miller	通过指导委员会进行指导、大量 SPARC 工作、jump.cc 的改进以及与 Linux 内核开发人员的交互
Gary Miller	将 GCC 移植到 Charles River Data Systems 机器上
Alfred Minarik	libstdc++ 字符串和 ios 错误修复，并使整个 libstdc++ 测试套件名称空间兼容
Mark Mitchell	通过指导委员会进行指导，大量的 C++ 工作，循环外的加载/存储提升，别名分析改进，ISO Crestrict 支持，以及从 2000 年到 2011 年担任发布经理

Alan Modra	各种 GNU/Linux 位和测试
Toon Moene	通过指导委员会、Fortran 维护以及他为使我们使 Fortran 快速运行而进行的持续工作的指导
Jason Molenda	在 gcc.gnu.org（以前的 egcs.cygnus.com）机器上的所有服务的维护和提供方面提供了主要帮助——邮件、Web 服务、ftp 服务等。在草稿纸上和信封的背面完成所有这些工作会……很难
Catherine Moore	修复了我们发送给她的各种难看的问题，包括杀死 Alpha 和 PowerPC Linux 内核的海法错误
Mike Moreton	各种 Java 补丁
David Mosberger-Tang	各种 Alpha 改进，以及最初的 IA-64 端口
Stephen Moshier	贡献了浮点仿真器，它有助于交叉编译并允许支持大于 64 位的浮点数和 ISO C99 支持
Bill Moyer	在各种问题上的幕后工作
Philippe De Muyter	在 m68k 端口上的工作
Joseph S. Myers	感谢他在 PDP-11 端口、格式检查和 ISO C99 支持方面的工作，以及对文档的持续重视（和贡献）
Nathan Myers	在 libstdc++-v3 方面的工作：前三个快照的体系结构和作者身份，包括语言环境基础设施的实现、字符串、影子 C 标头和初始项目文档（设计、清单等）。后来，更多关于 MT 安全字符串和影子标头的工作
Felix Natter	有关移植 libstdc++ 的文档
Nathanael Nerode	清理配置/构建过程
NeXT, Inc.	捐赠了支持 Objective-C 语言的前端
Hans-Peter Nilsson	CRIS 和 MMIX 端口、搜索引擎设置的改进、各种文档修复和其他小修复
Geoff Noer	致力于让 cygwin 本机构建工作
Vegard Nossum	运行 GCC 的自动回归测试并报告大量错误
Diego Novillo	感谢他在 Tree SSA、OpenMP、SPEC 性能跟踪网页、GIMPLE 元组和各种修复方面的工作
David O'Brien	负责 FreeBSD/alpha、FreeBSD/AMD x86-64、FreeBSD/ARM、FreeBSD/PowerPC 和 FreeBSD/SPARC64 移植以及相关的基础设施改进
Alexandre Oliva	各种构建基础设施改进、脚本和令人惊叹的测试工作，包括保持 libtool 问题的理智和快乐
Stefan Olsson	在 mt_alloc 上的工作
Melissa O'Neill	各种 NeXT 修复
Rainer Orth	随机 MIPS 工作，包括改进 GCC 的 o32 ABI 支持、改进 dejagnu 的 MIPS 支持、Java 配置清理和移植工作，以及维护 IRIX、Solaris 2 和 Tru64 UNIX 端口
Steven Pemberton	贡献查询：这允许 GCC 确定浮点单元的各种属性并生成浮动.h 在旧版本的 GCC 中
Hartmut Penner	s390 端口
Paul Petersen	撰写了 Alliant FX/8 的机器说明
Alexandre Petit-Bianco	实现了大部分 Java 编译器并继续担任 Java 维护者
Matthias Pfaller	对 NS32k 端口做出了重大改进
Gerald Pfeifer	通过指导委员会进行指导，指出了我们需要解决的许多问题，维护网页，并负责一般的文档维护
Marek Polacek	在 C 前端、杀毒功能和一般错误修复方面的工作
Andrew Pinski	处理十几个错误报告
Ovidiu Predescu	感谢他在 Objective-C 前端和运行时库方面的工作

Jerry Quinn	对 C++ 格式化 I/O 的主要性能改进
Ken Raeburn	对检查器、MIPS 端口的各种改进和编译器中的各种清理
Rolf W. Rasmussen	对 AWT 的黑客攻击
David Reese	来自太阳微系统公司，为 Solaris on PowerPC 端口做出了贡献
John Regehr	运行 GCC 的自动回归测试并报告了大量错误
Volker Reichelt	运行 GCC 的自动回归测试并报告大量错误并跟上问题报告
Joern Rennecke	负责维护 sh 端口、循环、regmove 和重新加载黑客以及开发和维护 Epiphany 端口
Loren J. Rittle	改进了 libstdc++-v3，包括 FreeBSD 端口、线程修复、线程相关的配置更改、关键线程文档和真正棘手的 I/O 问题的解决方案，以及保持 GCC 在 FreeBSD 上正常工作和持续测试
Craig Rodrigues	处理了大量的错误报告
Ola Rönnerup	负责 mt_alloc 方面的工作
Gavin Romig-Koch	参与了很多 MIPS 幕后工作
David Ronis	启发并鼓励 Craig 以 texinfo 格式重写 G77 文档，他贡献了旧版本的第一遍翻译 g77-0.5.16/f/文件文件
Ken Rose	修复了 GCC 的延迟槽填充代码
Ira Rosen	感谢她对自动矢量化器的贡献
Paul Rubin	编写了大部分预处理器
Pétur Runólfsson	对 C++ 格式化 I/O 的主要性能改进和 C++ filebuf 中的大文件支持
Chip Salzenberg	libstdc++ 补丁和对语言环境、特征、Makefile、libio、libtool hackery 和“long long”支持的改进
Juha Sarlin	改进了 H8 代码生成器
Greg Satz	协助 GCC 在 HP-UX 上为 9000 系列 300 工作
Roger Sayle	改进了常量折叠和 GCC 的 RTL 优化器，并修复了许多错误
Bradley Schatz	在 GCJ FAQ 方面的工作
Peter Schauer	编写了允许在 Alpha 上进行调试的代码
William Schelter	完成了 Intel 80386 支持方面的大部分工作
Tobias Schlüter	在 GNU Fortran 方面的工作
Bernd Schmidt	负责各种代码生成改进和重新加载过程中的主要工作，担任 GCC 2.95.3 的发布经理，并致力于 Blackfin 和 C6X 端口
Peter Schmid	对 libstdc++ 的不断测试——尤其是应用程序测试，超越了发布标准的要求——以及 libstdc++ 头文件调整
Jason Schroeder	jcf-dump 补丁
Andreas Schwab	在 m68k 端口上的工作
Lars Segerlund	GNU Fortran 方面的工作
Dodji Seketeli	修复了许多 C++ 错误并改进了调试信息
Tim Shen	在<regex>上进行的主要工作
Joel Sherrill	通过指导委员会、RTEMS 贡献和 RTEMS 测试的指导
Nathan Sidwell	许多 C++ 修复/改进
Jeffrey Siegal	帮助 RMS 完成 GCC 的原始设计、一些处理解析树和 RTL 数据结构的代码、常量折叠并帮助处理原始的 VAX 和 m68k 端口
Kenny Simpson	由于 LWG 的缺陷报告，他提示 libstdc++ 修复（从而使 GCC 与 ISO 的更新保持一致）
Franz Sirl	致力于使 GNU/Linux 的 PPC 端口稳定
Andrey Slepuhin	各种 AIX 黑客攻击

Trevor Smigiel	贡献了 SPU 端口
Christopher Smith	为 Convex 机器做了移植
Danny Smith	在 Mingw（和 Cygwin）端口上的重大努力。2010 年 8 月从 GCC 维护职位退休，指导了两名新的维护人员担任该职位
Randy Smith	完成了 Sun FPA 支持
Ed Smith-Rowland	感谢他在 libstdc++-v3、特殊函数、<random>、以及对 C++11 功能的各种改进方面的持续工作
Scott Snyder	队列、迭代器、istream 和字符串修复以及 libstdc++ 测试套件条目。还为 G77 提供补丁以添加对 INTEGER*1、INTEGER*2 和的基本支持 LOGICAL*1
Zdenek Sojka	运行 GCC 的自动回归测试并报告大量错误
Arseny Solokha	运行 GCC 的自动回归测试并报告大量错误
Jayant Sonar	贡献了 CR16 端口
Brad Spencer	对 GLIBCPP_FORCE_NEW 技术的贡献
Richard Stallman	编写了最初的 GCC 并启动了 GNU 项目
Jan Stein	来自 Chalmers 计算机学会，提供了对 Genix 的支持，以及 32000 机器描述的一部分
Gerhard Steinmetz	运行 GCC 的自动回归测试并报告大量错误
Nigel Stephens	各种 mips16 相关修复/改进
Jonathan Stone	为 Pyramid 计算机编写了机器描述
Graham Stott	各种基础设施改进
John Stracke	Java HTTP 协议修复
Mike Stump	Elxsi 端口、多年来对 G++ 的贡献以及最近他对 vxworks 的贡献
Jeff Sturm	提供 Java 移植帮助、错误修复和鼓励
Zhendong Su	运行 GCC 的自动回归测试并报告了大量错误
Chengnian Sun	运行 GCC 的自动回归测试并报告了大量错误
Shigeya Suzuki	为 bsdi 平台修复了问题
Ian Lance Taylor	对 Go 前端的攻陷，最初的 mips16 和 mips64 支持，一般配置黑客，修复包含等
Holger Teutsch	为 Clipper CPU 提供支持
Gary Thomas	为使 PPC 为 GNU/Linux 工作做了持续工作
Paul Thomas	GNU Fortran 方面的工作
Philipp Thomas	对整个编译器的随机错误修复
Jason Thorpe	NetBSD 上 libstdc++ 的线程支持
Kresten Krab Thorup	编写了对 Objective-C 语言的运行时支持和出色的 Java 字节码解释器
Michael Tiemann	随机错误修复、第一个指令调度程序、初始 C++ 支持、功能集成、NS32k、SPARC 和 M88k 机器描述工作、延迟槽调度
Andreas Tobler	将 libgcj 移植到 Darwin
Teemu Torma	线程安全异常处理支持
Leonard Tower	编写了部分解析器、RTL 生成器和 RTL 定义，以及 VAX 机器描述
Daniel Towner, Hariharan Sandanagobalane	贡献并维护了 picoChip 端口
Tom Tromey	提供国际化支持以及他的许多 Java 贡献和 libgcj 维护工作
Lassi Tuura	改进 config.guess 以确定 HP 处理器类型
Petter Urkedal	libstdc++ CXXFLAGS、数学和算法修复
Andy Vaught	负责 GNU Fortran 前端的设计和初步实现
Brent Verner	使用 libstdc++ cshadow 文件及其相关的配置步骤
Todd Vierling	对 NetBSD 端口的贡献

Andrew Waterman	贡献了 RISC-V 端口并维护它
Jonathan Wakely	贡献了 libstdc++ Doxygen 注释和 XHTML 指南并维护了 libstdc++
Dean Wakerley	为 GCC 3.0 及时将安装文档从 HTML 转换为 texinfo
Krister Walfridsson	随机错误修复
Feng Wang	对 GNU Fortran 的贡献
Stephen M. Webb	花时间和精力使 libstdc++ 影子文件与棘手的 Solaris 8+ 标头一起工作，并推动构建时标头树；驱动<regex>工作
John Wehle	对 x86 代码生成器的各种改进，相关的基础设施改进，以帮助 x86 代码生成、值范围传播等工作，WE32k 端口
Ulrich Weigand	在 s390 端口上的工作
Janus Weil	对 GNU Fortran 的贡献
Zack Weinberg	在 cpplib 和各种其他错误修复方面的主要工作
Matt Welsh	寻求 GCJ 中 Linux 线程支持的帮助
Urban Widmark	帮助修复 java.io
Mark Wielaard	新 Java 库代码和他与 Classpath 集成的工作
Dale Wiles	帮助将 GCC 移植到 Tahoe
Bob Wilson	来自 Tensilica 公司，负责 Xtensa 端口
Jim Wilson	通过指导委员会进行指导，解决了其他人不想处理的各个地方的难题，进行强度降低和其他循环优化
Paul Woegerer, Tal Agmon	CRX 端口
Carlo Wood	进行各种修复
Tom Wood	在 m88k 端口上的工作
Chung-Ju Wu	在 Andes NDS32 端口上的工作
Canqun Yang	在 GNU Fortran 方面的工作
Masanobu Yuhara	来自富士通实验室，实现了 Tron 架构（特别是 Gmicro）的机器描述
Kevin Zachmann	帮助将 GCC 移植到 Tahoe
Ayal Zaks	Swing 模数调度 (SMS)
Qirun Zhang	运行 GCC 的自动回归测试并报告了大量错误
Xiaoqiang Zhang	在 GNU Fortran 方面的工作
Gilles Zunino	帮助将 Java 移植到 Irix

Bernard Banner Romain Berrendonner Geert Bosch Emmanuel Briot Joel Brobecker Ben Brosgol Vincent Celier Arnaud Charlet Chien Chieng Cyrille Comar Cyrille Crozes Robert Dewar Gary Dismukes Robert Duff Ed Falis Ramon Fernandez	Sam Figueroa Vasiliy Fofanov Michael Friess Franco Gasperoni Ted Giering Matthew Gingell Laurent Guerby Jerome Guitton Olivier Hainque Jerome Hugues Hristian Kirtchev Jerome Lambourg Bruno Leclerc Albert Lee Sean McNeil Javier Miranda	Laurent Nana Pascal Obry Dong-Ik Oh Laurent Pautet Brett Porter Thomas Quinot Nicolas Roche Pat Rogers Jose Ruiz Douglas Rupp Sergey Rybin Gail Schenker Ed Schonberg Nicolas Setton Samuel Tardieu	对 GCC 的 Ada 前端 GNAT 的贡献

Michael Abd-El-Malek Thomas Arend Bonzo Armstrong Steven Ashe Chris Baldwin David Billinghurst Jim Blandy Stephane Bortzmeyer Horst von Brand Frank Braun Rodney Brown Sidney Cadot Bradford Castalia Robert Clark Jonathan Corbet Ralph Doncaster Richard Emberson Levente Farkas Graham Fawcett Mark Fernyhough	Robert A. French Jörgen Freyh Mark K. Gardner Charles-Antoine Gauthier Yung Shing Gene David Gilbert Simon Gornall Fred Gray John Griffin Patrik Hagglund Phil Hargett Amancio Hasty Takafumi Hayashi Bryan W. Headley Kevin B. Hendricks Joep Jansen Christian Joensson Michel Kern David Kidd	Tobias Kuipers Anand Krishnaswamy A. O. V. Le Blanc llewelly Damon Love Brad Lucier Matthias Klose Martin Knoblauch Rick Lutowski Jesse Macnish Stefan Morrell Anon A. Mous Matthias Mueller Pekka Nikander Rick Niles Jon Olson Magnus Persson Chris Pollard Richard Polton	Derk Reefman David Rees Paul Reilly Tom Reilly Torsten Rueger Danny Sadinoff Marc Schifer Erik Schnetter Wayne K. Schroll David Schuler Vin Shelton Tim Souder Adam Sulmicki Bill Thorson George Talbot Pedro A. M. Vazquez Gregory Warnes Ian Watson David E. Young	测试人员

课程总结

1. 收获与体会

（1）是否达到了解专业特色、知识体系的目标（ B ）

A. 完全达到　B. 基本达到　C. 勉强达到　D. 未达到

（2）对主讲教授的授课内容是否感兴趣（ A ）

A. 很感兴趣　B. 较感兴趣　C. 一般　D.完全没兴趣

（3）是否愿意对其中某一个方向深入学习，并加入创新实验室或研究团队（ B ）

A. 有明确的兴趣方向　B. 比较有兴趣，但还不明确

C. 只想先专注课内学习　D. 没什么兴趣

（4）最感兴趣的三个专题（ B ）（ E ）（ H ）

A. 专题一　B. 专题二　C. 专题三　D. 专题四

E. 专题五　F. 专题六　G. 专题七　H. 专题八

（5）个人心得体会（100-300 字）

大致了解了计算机学科的一些热门研究方向（研究目标、代表性问题、前沿发展），并对其中的部分方向产生了兴趣。会在之后查阅相关资料，考虑未来的个人发展方向。

2. 对本课程的建议（100-300 字）