东南大學
SOUTHEAST UNIVERSITY

人工智能实验

小组报告

实验名称：聚类算法

组长姓名：白丰硕

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering College of Software Engineering

Southeast University

二０一九年一月

東南大學
SOUTHEAST UNIVERSITY

目录

小组分工

姓名	学号	承担工作
白丰硕	71116233	聚类理论知识总结，深度学习计算机视觉前沿知识调研，实验报告编写
叶绵和	71116236	聚类简介、聚类性能指标以及距离计算，原型聚类算法实现，实验报告编写
杨志成	71116218	密度聚类，密度聚类算法实现，实验报告编写
杨昱昊	71116216	层次聚类，层次聚类算法实现，实验报告编写
刘晓	71116111	数据集准备工作，密度聚类算法实现，实验报告编写
李雨桐	71116301	原型聚类，原型聚类算法实现，实验报告编写

Part1 聚类算法研究

一、 实验说明

本实验通过通过实现原型聚类，密度聚类，层次聚类的算法并调整相应的参数，完成了上述算法的测试，并且我们还额外进行了机器视觉方向的相关调研。

二、 性能度量

聚类性能度量亦称为“有效性指标”，可以作为评价聚类效果好坏的指标，也可以作为聚类过程中的优化目标。聚类的目标是：“簇内相似度高，簇间相似度低”，聚类性能度量要能很好地体现出这一目标。

聚类性能度量分为两类：外部指标和内部指标。外部指标将聚类结果和某个“参考模型”进行比较，内部指标则直接考察聚类结果而不利用任何参考模型。

外部指标

为了很好地描述待评价模型和参考模型之间的关关系，我们作如下定义：

对于数据集 $D = \{x_1, x_2, \ldots, x_m\}$,假定待评价模型给出的簇划分为 $C = \{c_1, c_2, \ldots, c_k\}$，参考模型给出的簇划分为 $C^* = \{c_1^*, c_2^*, \ldots, c_s^*\}$。相应的，令 λ 与 λ^* 分别表示 C 和 C^* 对应的簇标记向量，我们将样本两两配对考虑，对变量 a,b,c,d 定义如下

$$a = |SS|, SS = \{(x_i, x_j) \mid \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}$$

$$b = |SD|, SD = \{(x_i, x_j) \mid \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}$$

$$c = |DS|, DS = \{(x_i, x_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}$$

$$d = |DD|, DD = \{(x_i, x_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}$$

易知，m*(m-1)/2 = a+b+c+d

据此，我们可以定义如下的几个聚类性能度量外部指标

- Jaccard 系数

$$JC = \frac{a}{a + b + c}$$

- FM 指数

$$FMI = \sqrt{\frac{a}{a+b} * \frac{a}{a+c}}$$

- Rand 指数

$$RI = \frac{2(a+d)}{m(m-1)}$$

通常来讲，上述性能度量的范围在[0,1]区间内，值越大越好

内部指标

同样的我们做如下定义，对于聚类结果的簇划分$C = \{C_1, C_2, \ldots, C_k\}$，有：

$$avg(C) = \frac{2}{|C|(|C|-1)} \sum_{1 \le i < j \le |C|} dist(x_i, x_j)$$

$$diam(C) = max_{x_1 \le i < j \le |C|} dist(x_i, x_j)$$

$$d_{min}(C_i, C_j) = min_{x_i \in C_i, x_j \in C_j} dist(x_i, x_j)$$

$$d_{cen}(C_i, C_j) = dist(\mu_i, \mu_j)$$

$$\mu = \frac{1}{|C|} \sum_{1 \le i \le |C|} x_i,$$

对于如上定义，μ 代表簇内的中心点，avg(c)表示簇内样本间的平均距离，diam(C)代表簇内最远的两个样本间的距离，$d_{\min}(C_i, C_j)$代表两个簇内最接近的两个样本间的距离，$d_{cen}(C_i, C_j)$代表两个簇之间中心点的距离。

据此，我们可以定义如下的几个聚类性能度量内部指标：

- DB 指数

$$DBI = \frac{1}{k} \sum_{i=1}^{k} max_{j \ne i} \left(\frac{avg(C_i) + avg(C_j)}{d_{cen}(\mu_i, \mu_j)} \right)$$

- Dunn 指数

$$DI = min_{1 \le i \le k} \left\{ min_{j \ne i} \left(\frac{d_{min}(C_i, C_j)}{max_{1 \le l \le k} diam(C_l)} \right) \right\}$$

DBI 用于衡量样本簇各自内部点的聚合度于样本簇之间的聚合度的大小，其值越小聚类效果

越好； DI 用于比较样本簇之间最小距离和样本簇之间的最大距离，主要是样本簇之间进行比较，其值越大聚类效果越好。

三、 距离计算

在聚类算法中，我们需要基于某种形式的距离来定义“相似度”。通常来讲，距离越大相似度越小。同时，在聚类算法中使用的相似度度量距离未必一定要满足普通距离度量的所有基本特性，如非负性、同一性、对称性、直递性等等。

我们对需要度量的属性有第一种划分：连续属性、离散属性。连续属性值定义域上有无穷多个可能的取值，无序在定义域上有有限个取值。

针对于属性是否有序，我们有第二种划分：有序属性、无序属性、混合属性。有序属性可以在属性值计算距离远近（如属性{1，2，3}可以直接比较距离远近），无序属性不能直接在属性值上计算距离远近（如属性{飞机，火车，轮船}不可以直接比较距离远近），混合属性是兼有有序属性和无序属性。

针对有序属性、无序属性、混合属性，我们分别有不同的距离计算方法：

- 有序属性

  给定样本 $x_i = (x_{i1}; x_{I2}; \ldots; x_{in})$ 和 $x_j = (x_{j1}; x_{j2}; \ldots; x_{jn})$ 最常用的是“闵科夫斯基距离“：

  $$\mathrm{dist}_{\mathrm{mk}}(x_i, x_j) = \left(\sum_{u=1}^{n} |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}}, p \geq 1$$

  特殊的，当 p=2 时，闵科夫斯基距离即欧式距离：

  $$\mathrm{dist}_{\mathrm{mk}}(x_i, x_j) = \sqrt{\sum_{u=1}^{n} |x_{iu} - x_{ju}|^2}$$

  当 p=1 时，闵科夫斯基距离即曼哈顿距离：

  $$\mathrm{dist}_{\mathrm{mk}}(x_i, x_j) = \sum_{u=1}^{n} |x_{iu} - x_{ju}|$$

- 无序属性

  对于无序属性可以采用 VDM 计算距离。令 $\mathrm{m}_{u,a}$ 表示在属性 u 上取值为 a 的样本数量，$\mathrm{m}_{u,a,i}$ 表示在第 i 个样本簇中在属性 u 上取值为 a 的样本数，k 表示为样本簇数。

  属性 u 上两个离散值 a 与 b 的 VDM 距离表示为：

  $$\mathrm{VDM}_p(a, b) = \sum_{i=1}^{k} \left| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right|$$

- 混合属性

将闵科夫斯基距离和 VDM 结合即可处理混合属性。假设有 n_c 个有序属性，n-n_c 个无序属性，不妨令有序属性排在无序属性之前，则

$$\mathrm{MinkovDM}_p(x_i, x_j) = \left(\sum_{u=1}^{n_c} |x_{iu} - x_{ju}|^p + \sum_{u=n_c+1}^{n} \mathrm{VDM}_p(x_{iu} - x_{ju})^{\frac{1}{p}}\right)$$

四、 算法及参数设置

原型聚类

➢ 算法一（基础）

- 算法简介：

k-means 算法是一种简单的迭代型聚类算法，采用距离作为相似性指标，从而发现给定数据集中的 K 个类，且每个类的中心是根据类中所有值的均值得到，每个类用聚类中心来描述。对于给定的一个包含 n 个 d 维数据点的数据集 X 以及要分得的类别 K,选取欧式距离作为相似度指标，聚类目标是使得各类的聚类平方和最小。

结合最小二乘法和拉格朗日原理，聚类中心为对应类别中各数据点的平均值，同时为了使得算法收敛，在迭代过程中，应使最终的聚类中心尽可能的不变。

- 算法流程：

K-means 是一个反复迭代的过程，算法分为四个步骤：

1） 选取数据空间中的 K 个对象作为初始中心，每个对象代表一个聚类中心；

2） 对于样本中的数据对象，根据它们与这些聚类中心的欧氏距离，按距离最近的准则将它们分到距离它们最近的聚类中心（最相似）所对应的类；

3） 更新聚类中心：将每个类别中所有对象所对应的均值作为该类别的聚类中心，计算目标函数的值；

4） 判断聚类中心和目标函数的值是否发生改变，若不变，则输出结果，若改变，则返回步骤 2。

- 算法实现：

采用 Java 语言进行实现，代码节选如下：

```
private static void updateCenters(int k, Kmeans_data data) {//更新均值向量
        double[][] centers = data.centers;
        setDouble2Zero(centers, k, data.dim);
        int[] labels = data.labels;
        int[] centerCounts = data.centerCounts;
        for (int i = 0; i < data.dim; i++) {
            for (int j = 0; j < data.length; j++) {
                centers[labels[j]][i] += data.data[j][i];
            }
        }
        for (int i = 0; i < k; i++) {
```

```
            for (int j = 0; j < data.dim; j++) {
                centers[i][j] = centers[i][j] / centerCounts[i];
            }
        }
    }


    public static double dist(double[] pa, double[] pb, int dim) {
        double rv = 0;
        for (int i = 0; i < dim; i++) {
            double temp = pa[i] - pb[i];
            temp = temp * temp;
            rv += temp;
        }
        return Math.sqrt(rv);                    // 计算误差平方
    }


    public static Kmeans_result doKmeans(int k, Kmeans_data data, Kmeans_param param) {

        double[][] centers = new double[k][data.dim];
        data.centers = centers;
        int[] centerCounts = new int[k];
        data.centerCounts = centerCounts;
        Arrays.fill(centerCounts, 0);
        int[] labels = new int[data.length];
        data.labels = labels;
        double[][] oldCenters = new double[k][data.dim];


        if (param.initCenterMehtod == Kmeans_param.CENTER_RANDOM) {
            Random rn = new Random();                    //随机选择均值向量
            List<Integer> seeds = new LinkedList<Integer>();
            while (seeds.size() < k) {
                int randomInt = rn.nextInt(data.length);
                if (!seeds.contains(randomInt)) {
                    seeds.add(randomInt);
                }
            }
            Collections.sort(seeds);
            for (int i = 0; i < k; i++) {
                int m = seeds.remove(0);
                for (int j = 0; j < data.dim; j++) {
                    centers[i][j] = data.data[m][j];
```

```
                }
            }
        } else {
            for (int i = 0; i < k; i++) {
                for (int j = 0; j < data.dim; j++) {
                    centers[i][j] = data.data[i][j];
                }
            }
        }

        for (int i = 0; i < data.length; i++) {
            double minDist = dist(data.data[i], centers[0], data.dim);
            int label = 0;
            for (int j = 1; j < k; j++) {
                double tempDist = dist(data.data[i], centers[j], data.dim);
                if (tempDist < minDist) {
                    minDist = tempDist;
                    label = j;
                }
            }
            labels[i] = label;
            centerCounts[label]++;
        }
        updateCenters(k, data);
        copyCenters(oldCenters, centers, k, data.dim);

        int maxAttempts = param.attempts > 0 ? param.attempts :
Kmeans_param.MAX_ATTEMPTS;
        int attempts = 1;
        double criteria = param.criteria > 0 ? param.criteria : Kmeans_param.MIN_CRITERIA;
        double criteriaBreakCondition = 0;
        boolean[] flags = new boolean[k];

        iterate: while (attempts < maxAttempts) {
            for (int i = 0; i < k; i++) {
                flags[i] = false;
            }
            for (int i = 0; i < data.length; i++) {
                double minDist = dist(data.data[i], centers[0], data.dim);
                int label = 0;
                for (int j = 1; j < k; j++) {
                    double tempDist = dist(data.data[i], centers[j], data.dim);
                    if (tempDist < minDist) {
                        minDist = tempDist;
```

```
                    label = j;
                }
            }
            if (label != labels[i]) {
                int oldLabel = labels[i];
                labels[i] = label;
                centerCounts[oldLabel]--;
                centerCounts[label]++;
                flags[oldLabel] = true;
                flags[label] = true;
            }
        }
        updateCenters(k, data);
        attempts++;


        double maxDist = 0;
        for (int i = 0; i < k; i++) {
            if (flags[i]) {
                double tempDist = dist(centers[i], oldCenters[i], data.dim);
                if (maxDist < tempDist) {
                    maxDist = tempDist;
                }
                for (int j = 0; j < data.dim; j++) {
                    oldCenters[i][j] = centers[i][j];
                }
            }
        }
        if (maxDist < criteria) {
            criteriaBreakCondition = maxDist;
            break iterate;
        }
    }
```

- 实验结果：

与课本上标注的结果进行对比，计算外部性能度量结果如下：

a=13,b=2,c=4,d=17,m=9 ;

JC=a/(a+b+c)=13/19 ;

FMI=a/√(〖(a+b)(a+c)〗^)=13/16 ;

RI=(2(a+d))/(m(m-1))=5/6

密度聚类

- 算法一（DBSCAN 算法）
  - 简介：DBSCAN 算法是一个经典的密度聚类算法，它基于一组“领域”参数来刻画样本分布的紧密程度。将“簇”定义为由密度可达关系导出的最大密度相连样本集合。目标是从数据集中找到满足连接性和最大性的聚类簇。实现方法是先任选数据集中的一个核心对象为 seed，根据给定的邻域参数找出所有的核心对象，然后以任一核心对象为出发点找到其密度可达的样本生成聚类簇。
  - 参数选择：邻域参数 ε，MinPts

层次聚类

- 算法一（AGNES 算法）
  - 简介：AGNES 是一种采用自底向上聚合策略的层次聚类算法。它先把数据集中的每个样本看作一个初始聚类簇，然后在算法运行的每一步找到距离最近的聚类簇合并，直到达到预设的聚类簇个数。当聚类簇的距离由最小距离、最大距离或平均距离决定时，AGNES 算法分别称为”单链接“、”全链接“或”均链接“。
  - 参数选择：预设的聚类簇个数 q

五、 实验结果

```
k=2
attempts=2
criteriaBreakCondition=0.0
The number of each classes are:
15 15

The labels of points is: 1  1  1  1  1  0  0  0  1  0  0  0  1  1  0  1  1  0  0  0  1  1  0  0  0  1  1  0  1  0
```

Part2 领域调研

71116233 白丰硕

CV 基础任务简介

- 图像分类

  对于给定的一张输入的图像，判断图像所属的类别。比如给一张有猫或狗的图像，判断图像是猫还是狗。但是不关注猫狗的位置和除了猫狗之外类别的物体。

- 目标定位

  在图像分类的基础上，希望通过分析获得某个物体在某张图像中具体的位置。并使用外框将目标包裹已体现出目标的位置。

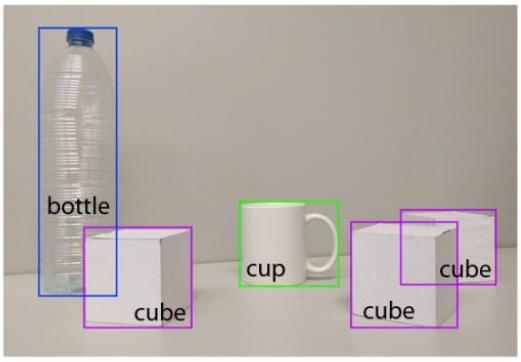
- 语义分割

  对图像进行像素级别的分类。顾名思义就是要对每一个像素都赋予一个类别的信息，在一副图像中，构成猫的像素的类别就是猫，构成狗的像素的类别就是狗。和图像分类的最大区别是语义分割关注的是更加具体，关注更多细节，一幅图像中可能会有多个目标物体，都是需要被分类的。

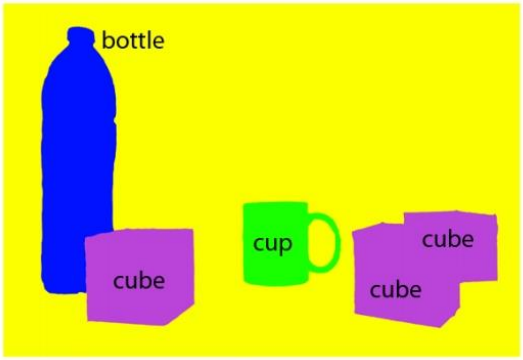
- 实例分割

  和语义分割相近，语义分割不区分相同类别不同实例，而实例分割中希望获得，图像中每一个不同的实例个体。两者的关系与 C++中类与对象的关系相近。[1]

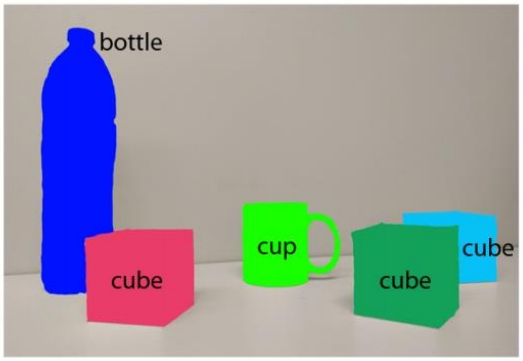
(a) Image classification



(b) Object localization



(c) Semantic segmentation



(d) Instance segmentation

语义分割

- 介绍：语义分割本质上是像素级别上的分类，属于同一类的像素都要被归为一类，因此语义分割是从像素级别来理解图像的。
- 数据集：
  - VOC2012

    这个数据集中一共有 20 个类别，一共含有 33043 张图片，79540 个物体

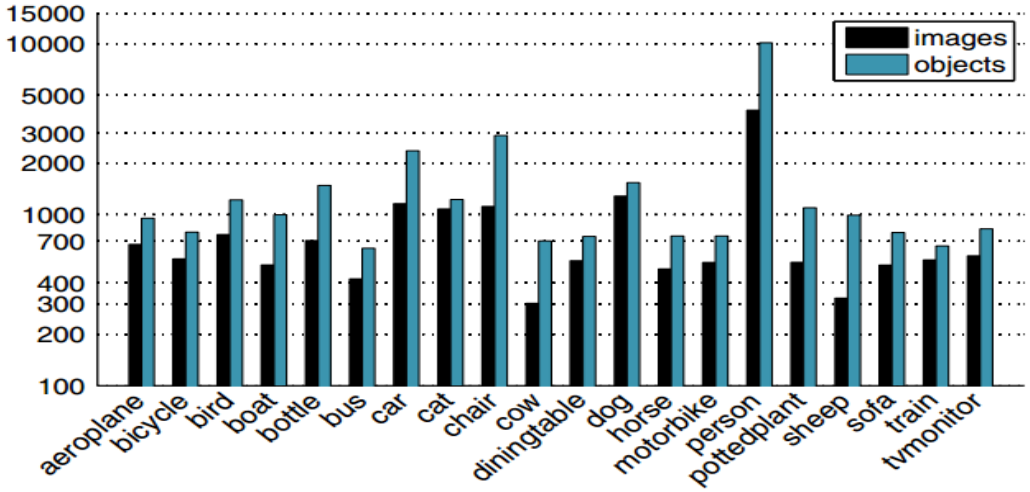
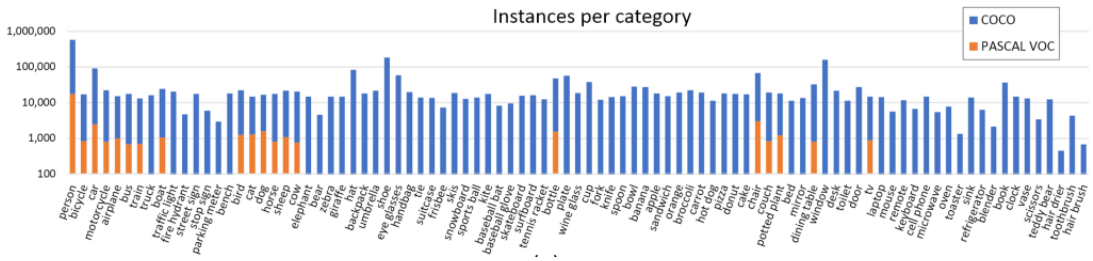


Fig. 2 Summary of the main VOC2012 dataset. Training and validation images only. Histogram by class of the number of objects and images containing at least one object of the corresponding class. Note the log scale on the *vertical axis*. Best viewed in *colour* (Color figure online)

  - MSCOCO

    COCO 中的图片包含了自然图片以及生活中常见的目标图片，背景比较复杂，目标数量比较多，目标尺寸更小，因此 COCO 数据集上的任务更难，对于检测任务来说，现在衡量一个模型好坏的标准更加倾向于使用 COCO 数据集上的检测结果。其中包含 80 个类别，82,783 个训练图像、40,504 个验证图像以及 40,775 个测试图像。



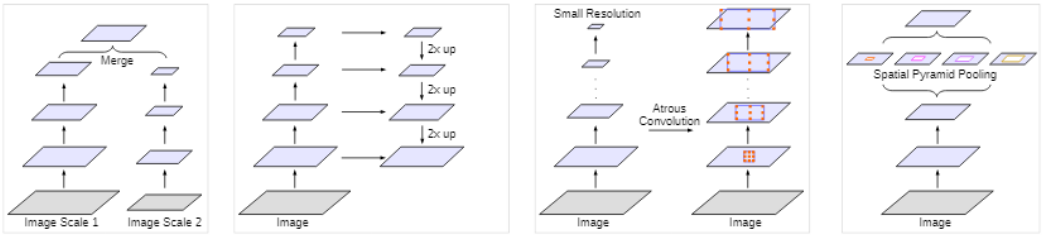
- 算法比较：

算法	VOC2012 测试分数
FCN	62.2
SegNet	59.9
Dilated Convolutions	71.3
DeepLab v2	79.7
RefineNet	84.2
PSPNet	85.4
DeepLab v3	85.7

语义分割中的挑战和解决方法

- 空间分辨率下降

在 CNN 中有一种类型的层叫池化层，主要是任务就是对图像进行下采样处理。而由于下采样处理虽然可以增加上层卷积核的感知野，但会丢弃部分的位置信息。但是语义分割中需要保留这部分信息。面对这个问题主要是提出了两种解决思路。第一种，使用编码器-解码器结构，在编码器中使用池化层，输入空间的维度会逐渐的降低，而空间维度及目标信息会在解码器的作用下逐渐恢复。通常会在编码器和解码器之间存在直接的信息连接，保留原始的输入数据来帮助解码器来更好地恢复目标的位置信息和空间维度。第二种方法，采用带孔卷积的结构来替代以往的常规的卷积层。这种卷积层，主要在卷积核的结构会有所不同，卷积核的结构中会有一些“空位置”，就像将一个原始的卷积核进行拉伸后，单个卷积核中的值得数目和大小不改变。输出空间维度大小是取决于输入空间的维度大小和卷积核大小（假设步长和填补方式是不变的情况下），调整卷积核的大小就可以使得输出空间的维度大小和输入空间的维度大小相同。第三种是在原始模型的顶端增加额外的模块，负责捕捉像素间长距离信息。第四中，使用空间金字塔池化方法，使用不同采样率的卷积核，来捕捉多尺度的物体。



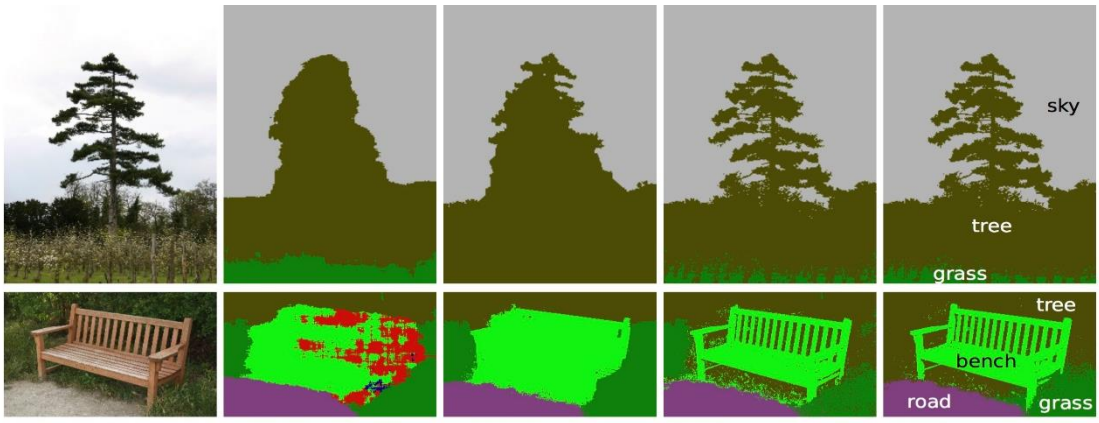
(a) Image Pyramid (b) Encoder-Decoder (c) Deeper w. Atrous Convolution (d) Spatial Pyramid Pooling

Figure 2. Alternative architectures to capture multi-scale context.

[5]

- CRF

条件随机场方法主要是用于语义分割处理后，对于分割效果的进一步优化和改善。当信息是与时间或者空间的前后有关系的时候就需要考虑 CRF。其主要的原理是，训练特征函数的权重，对于训练集，采用极大似然估计法计算权重参数，通过对条件概率的对数似然函数求导，令其解为零，从而得到解或者近似解。[2]



(a) Image (b) Unary classifiers (c) Robust P^n CRF (d) Fully connected CRF, MCMC inference, 36 hrs (e) Fully connected CRF, our approach, 0.2 seconds

- 物体的多尺度

  在一幅图像中同一类物体可能会出现多个尺度的实例，就需要模型对不同尺度的物体都可以有相应的识别处理。在计算机视觉中，解决这个问题的方法一般是采用将一张图片缩放成不同版本，汇总这些特征后用来预测结果。但是这个方法会增加计算机特征的时间，而且需要更多的存储空间。在 DeepLabv2 中采用 ASPP (atrous spatial pyramid pooling)模块。对 SPP 进行改善，对于一张给定输入的图像，使用不同分辨率的带孔卷积层进行处理采样，相当于使用多种比例来捕捉图像中的物体。[4]

- 空间定位精度

  由于 CNN 对于分类任务要求空间变换不变性，这个会影响 DCNN 的目标空间位置的定位准确度。在 DeepLabv2 中采用在计算最终分类结果的时候，使用跨层连接的方式，将前面的提取出的特征融合在一起，通过全连接的 CRF 来细化模型对图像中物体与物体边界地方的识别能力。[4]

参考文献

[1] 张皓. 计算机视觉分享. https://blog.csdn.net/u011707542/article/details/79151978

[2] Sasank Chilamkurthy. A 2017 Guide to Semantic Segmentation with Deep Learning. http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review

[3] Liang-Chieh Chen, George Papandreou∗, Iasonas Kokkinos, Kevin Murphy and Alan L. Yuille. DeepLabv1: Semantic image segmentation with deep convolutional nets and fully connected CRFs. In ICLR, 2015

[4] Liang-Chieh Chen, George Papandreou,Iasonas Kokkinos ,Kevin Murphy, and Alan L. Yuille. DeepLabv2:DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. In TPAMI, 2017

[5] Liang-Chieh Chen,George Papandreou,Florian Schroff,Hartwig Adam.DeepLabv3:Rethinking Atrous Convolution for Semantic Image Segmentation. arXiv: 1706.05587, 2017