# 第3章

# 空间多尺度特征学习

现有语义解析算法在面对像素间长距离关联构建和多尺度表征学习的需求时，普遍承受着巨大的计算负担，这与实时性要求背道而驰。与现有网络普遍在通道维度上进行知识建模的方法不同，本章创新地从空间邻域解耦和耦合的角度出发，提出了一种新的多尺度表征学习算法，在大幅减少网络冗余和运算负担的同时，显著提高网络特征提取能力。本章首先定义并实现了“邻域解耦”（neighbor decoupling，ND）和“邻域耦合”（neighbor coupling，NC）操作，通过互补的缩略图采样和整合实现了互逆的无损分辨率转换。随后，基于ND/NC操作，本章进一步提出了应用于网络不同阶段的两种空间维度建模方法：局部特征感知与全局依赖构建方法（local capture and global builder，LCGB），以及空间多尺度特征提取网络（spacewise multiscale feature extractor，SMFE）。其中，LCGB在2.2节所提出方法的基础上，进一步减少了运算负担，并在提取初始特征的同时，构建了全局上下文关联。SMFE则实现了空间维度上的低冗余多尺度稠密特征提取。本章所提出的方法，模型容量大、运算量低。实验证明，本章方法减少了特征冗余，优化了资源开销，提高了多尺度特征提取效率，仅用本章算法替换现有网络对应结构，即可实现更优的语义解析性能。

## 3.1 概述

深度网络学习到的特征存在大量冗余，造成了参数及计算资源浪费。因此，减少特征提取过程中的网络冗余，充分利用网络参数学习重要特征，并节约计算量成为提高语义解析算法实时性的重要途径。针对语义解析对多尺度特征学习及像素间长距离依赖建模的需求，绝大多数现有深度网络模型通常直接对整个图像/特征图进行全通道的信息建模。然而，这些方法往往面临着3.1.1节所述的问题。为

此，本章提出了一种新的多尺度轻量学习体系，从一个新的角度来解决这些难题。具体研究内容与贡献详见3.1.2节。

### 3.1.1　拟解决的主要问题

本章拟解决的语义解析模型在特征提取方面的主要问题如下。

（1）**大尺度输入数据及特征的空间冗余性是导致深度网络计算瓶颈的关键因素。**输入图像与浅层特征由于相邻像素的高度相似性与关联性所导致的大量空间信息冗余，经具有局部感知特性卷积算子的传递、整合与抽象仍持续存在于各级特征表示中。因此大尺度图像的空间冗余最终会下沉至特征冗余，从而增加网络计算复杂度与模型存储需求。针对这一问题，现有方法通常以高效卷积设计[126-127]去除空间冗余或以复杂注意力机制[9]构建全局关联，却往往带来不可逆的信息损失或引入高昂的计算负担，从而影响算法的准确性或实时性。本章研究认为，无参数的空间表征可逆变换将有利于降低高维空间冗余表征学习的计算复杂度。受网页浏览器通过缩略图加载和预览大尺寸图片的启发，本章提出了邻域解耦与耦合概念：通过邻域解耦实现空间分组与去冗以获取特征缩略图，后通过邻域耦合实现空间恢复与聚合来生成无损高分辨率结果。

（2）**鉴于输入预处理中减少冗余和保留细节之间的矛盾，现有分割算法通常以不可逆的信息损失为代价来换取实时性能。**现有方法通常直接对整张输入在全部通道或分别在部分通道上进行建模。对输入图像全部通道进行建模的方法较为常见，这类方法[1,5,128]能较好建立多通道局部关联，但由于输入图像尺度较大，网络需要巨大的计算开销，影响其实时性；且在网络初级阶段，由于特征级别较弱，故输入的初始建模往往存在较大的特征冗余。而对输入图像分别在部分通道进行建模的方法最先由ResNeXt[2]提出，并由MobileNet[6]和ShuffleNet[8]发挥到极致。类似方法虽然尽可能压缩了网络参数量和计算量，却因通道间缺乏特征交流而造成不可逆的信息损失。本章借助缩略图思想，将输入图像在空间维度上解耦，抽取为多个缩略图，并分别在缩略图的全部通道上进行特征提取，以同时实现快速计算降维和完整信息建模。

（3）**在密集特征提取过程中，语义解析模型对多尺度表征的需求往往促使模型引入更多的计算负担。**由于语义解析场景中存在尺寸多样的物体和形状复杂的区域，现有的多尺度学习方案，如图像金字塔[32-33]和特征金字塔体系[34,125]，通

常利用多核并行卷积在同一输入上进行多尺度特征提取以获得复合感受野。但此类方法会导致大量特征冗余及网络参数浪费，从而削弱实时性能。此外，基于特征融合的多尺度学习方法往往通过拉取多个阶段的输出并引入额外的参数进行特征聚合。此类方法一方面由于多条学习路径的存在，造成了网络训练中回传梯度的冗余，加剧网络训练难度；另一方面额外的特征聚合模块引入了不必要的参数和计算负担，同样会降低网络实时性。本章依旧从空间分组角度出发，通过空间解耦生成相近且互补的缩略图，并利用不同大小的卷积在分组缩略图上进行并行特征提取，从而实现低计算成本下的单步多尺度学习。

### 3.1.2 研究内容及贡献

针对3.1.1节所提出的科学问题，本章研究内容及贡献如下。

（1）本章提出了通过对空间子集（缩略图）进行信息建模与表征学习的新思想与新体系，并针对缩略图采样和聚合操作定义了邻域解耦/耦合算子。

（2）本章在2.2节方法的基础上，借助缩略图的概念，进一步构建了局部特征感知与全局依赖算法，实现了快速且无损的输入图像快速降维，进而高效捕获具有全局感受野和特征关联的初始特征。

（3）本章提出了空间多尺度特征提取算法，通过邻域解耦/耦合算子和多尺寸卷积实现了低计算量、低冗余度、大感受野的多尺度表征学习。

## 3.2 空间邻域解耦-耦合算子

本章提出的邻域解耦（ND）操作和耦合（NC）操作是一对可逆算子，可通过进行缩略图采样与聚合来实现无损的分辨率变换。具体而言，ND将空间邻域内的像素解耦为若干类似但互补的低分辨率子特征图（本章形象地将其称作缩略图）。相反，其逆过程NC用于恢复分辨率及邻域内的像素相关性。

### 3.2.1 算子定义

本章定义的ND/NC算子及其操作效果示意图可见图3.1上部。由于NC是ND的逆运算，本节在此只对ND进行详细阐述。从图像角度来看，对于大小为 $C \times H \times W$ 特征图，ND将其在空间维度上划分为$r \times r$个网格，将每个网格内相对位置相同的像素分别组合，并保留像素原有通道，构成$r^2$个子图，每个子图大小则

变为$C \times \frac{H}{r} \times \frac{W}{r}$, 其中$C$为特征通道数，$r$表示邻域大小。算法2给出了ND的具体实现。

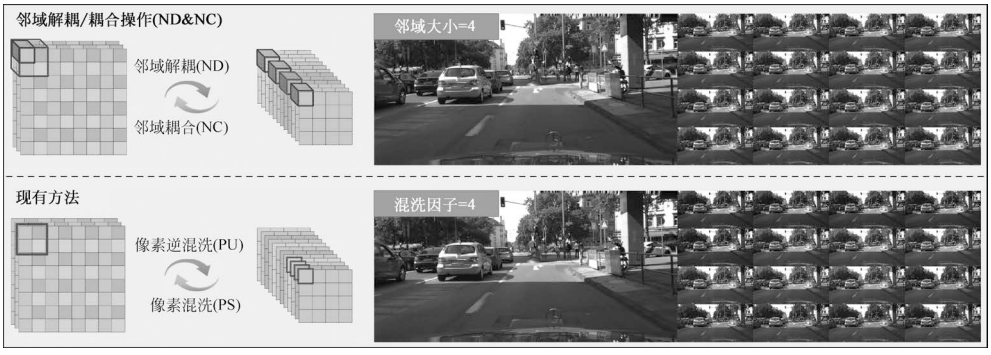



**图 3.1　邻域解耦-耦合算子及其操作效果示意图**

ND的数学计算过程如下所示。记输入为$\boldsymbol{X} \in \mathbb{R}^{[C,H,W]}$，按像素可表示为

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \cdots & x_{1w} \\ \vdots & \ddots & \vdots \\ x_{h1} & \cdots & x_{hw} \end{bmatrix} = (x_{ij})_{H \times W} \tag{3.1}$$

其中，$x_{ij} \in \mathbb{R}^C$表示具有全部通道信息的第$(i,j)$个像素。为便于描述，假设此处$H$和$W$都能被$r$整除（若无法整除，则进行边界补零至可整除状态），则$\boldsymbol{X}$可被重写为分块矩阵的形式，即

$$\boldsymbol{X} = \begin{bmatrix} P_{11} & \cdots & P_{1\frac{w}{r}} \\ \vdots & \ddots & \vdots \\ P_{\frac{h}{r}1} & \cdots & P_{\frac{h}{r}\frac{w}{r}} \end{bmatrix} = (\boldsymbol{P}_{ij})_{\frac{h}{r} \times \frac{w}{r}} \tag{3.2}$$

其中，$\boldsymbol{P}_{ij} \in \mathbb{R}^{[C,r,r]}$，并可进一步表示为

$$\boldsymbol{P}_{ij} = \begin{bmatrix} p_{11} & \cdots & p_{1r} \\ \vdots & \ddots & \vdots \\ p_{r1} & \cdots & p_{rr} \end{bmatrix} \tag{3.3}$$

**算法 2:** 邻域解耦算子实现方式

**输入** : $X \in \mathbb{R}^{[N,\ C,\ H,\ W]}$, 邻域大小 $r$
**输出** : $Y$, 邻域解耦的结果——缩略图
**函数定义:** `mod` 取余计算
`floor` 向下取整
`reshape` 调整矩阵形状
`permute` 多维矩阵转置

/* 按需求进行补零 */

padH $\leftarrow$ `mod`$(H, r)$ ; // 高度方向需填充 0 的数量
padW $\leftarrow$ `mod`$(W, r)$ ; // 宽度方向需填充 0 的数量

**if** padH!= 0 or padW != 0 **then**
  leftPad $\leftarrow$ `floor`(padW/2) ;
  rightPad $\leftarrow$ padW − leftPad ;
  topPad $\leftarrow$ `floor`(padH/2) ;
  botPad $\leftarrow$ padH − topPad ;
  fillPad(X, [leftPad, rightPad, topPad, botPad]);
**end**
$[N, C, H, W] \leftarrow X$.shape() ;
$X' \leftarrow$ `reshape`$(X, [N, C, H/r, r, W/r, r])$ ;
$X'' \leftarrow$ `permute`$(X', (0, 3, 5, 1, 2, 4))$ ;
$Y \leftarrow$ `reshape`$(X'', [N, r^2C, H/r, W/r])$ ; // 输出

而 ND 每遍历一次 $\boldsymbol{P}_{ij}$ 便从中抽取一个元素来形成 $\boldsymbol{X}$ 的一组子集，如下所示：

$$Y = \left\{ X\left(m,n\right) \mid I\left(m,n\right) = \left(P\left\langle m,n\right\rangle\right)_{\frac{H}{r}\times\frac{W}{r}},\ m,n \in [1,r] \right\} \tag{3.4}$$

其中，$P\left\langle m,n\right\rangle$ 表示在 $P$ 中第 $m,n$ 个位置的元素。

在物理意义上，ND 将原始输入解耦成 $r^2$ 个相似但互补的子特征图（缩略图），并将分辨率降低至原图的 $1/r$。由于缩略图的相似性，每个缩略图可以代表原始输入的一部分，并且 $r$ 越小，缩略图与原始输入就越相似。

### 3.2.2 与现有方法对比

“空间-深度”“深度-空间”转换至少有四种方式，其中像素混洗（pixel-shuffle）[19]和像素逆混洗（pixel-unshuffle）已经在很多任务中得到了广泛的应

用，并且已经作为标准函数集成到PyTorch中，本书第2章中应用的就是该操作。本节通过对比与ND具有相似功能的像素逆混洗操作来说明它们之间的差异。

在操作效果上，假设输入$\boldsymbol{X} \in \mathbb{R}^{[B,\ C,\ H,\ W]}$，ND对像素的操作可以形象化地表示为

$$\mathrm{ND} \triangleq \mathrm{rearrange}\left(\boldsymbol{X},' BC\left(Hr\right)\left(Wr\right) \rightarrow B\left(\boldsymbol{r}^2\boldsymbol{C}\right) HW'\right) \tag{3.5}$$

而像素逆混洗则应表示为

$$\mathrm{PU} \triangleq \mathrm{rearrange}\left(\boldsymbol{X},' BC\left(Hr\right)\left(Wr\right) \rightarrow B\left(\boldsymbol{C}\boldsymbol{r}^2\right) HW'\right) \tag{3.6}$$

虽然从直觉上看，这两组操作似乎只在周期性抽取像素的顺序上有所不同，但本章提出的ND操作允许在低分辨率空间中对像素进行分组，并赋予了每组独立的物理意义。正如图3.1中右边的对比示意图所示，ND将原始图像抽样为16张缩略图，这些缩略图与原始图像几乎完全相同，而像素逆混洗只是进行数学上的分组，其结果不具备可解释性。

对于一个大小为$r \times r$的邻域，可以假设该邻域内的每个元素都有类似的特征，或者说该邻域内的每个像素能够在各自相同的空间位置代表不同的特征。因此，分别对生成的缩略图进行特征提取能够节省大量的计算。相比于分组卷积在通道维度上对特征图进行分组的方法，ND/NC操作实现了对特征图在空间维度上的分组计算。这是像素混洗/逆混洗无法实现的。

此外，由于卷积具有良好的空间不变性和通道特异性，因此ND采样的缩略图在提取特征时几乎可近似于原始输入（如图3.1上部所示）。而由于此时分辨率降低为原始输入的$1/r$，故计算成本也被大大降低。然而，对于pixel-unshuffle来说，它在分组时打破了通道之间的关联性，并不能形成缩略图（如图3.1下部所示），更无法在后续的特征提取中具有与ND相同的能力。

## 3.3　初始特征的局部感知与全局建模

由于大尺度输入图像具有丰富的信息和极高的冗余度，故网络的输入预处理阶段对于提升网络实时性至关重要。基于以下三方面的考虑，本节在3.2节的基础上提出了一种输入及初始特征的局部感知与全局建模（LCGB）方法：① 为了满足实时性要求，对图像进行快速降采样是极其必要的，但丰富的边缘和细节信息不

应被忽略；② 足够的感受野对于提取初始特征是十分关键的，但会带来巨大的计算量，因而需要廉价的操作来释放计算负担；③ 图像整体的色调、亮度以及物体间关联可以为后续感知提供重要参照基础，因而全局感知与建模具有重要意义。

### 3.3.1 网络结构设计

图3.2详细绘制了LCGB网络结构示意图，其由三部分串联而成。首先，输入图像被ND解耦为大小为 $\left[3\times 16, \frac{H}{4}, \frac{W}{4}\right]$ 的缩略图（16组3通道特征图）。然后，通过四个级联的卷积组合来捕捉局部特征，每个卷积组合包括一个分组数为16的 $3\times 3$ 卷积，以提取每个组内的特征；及一个 $1\times 1$ 逐点卷积，来融合各组特征。最后，将所有特征图串联，并通过轻量化的自注意力机制建立全局依赖。该自注意力机制的实现细节如图3.2中下方所示。

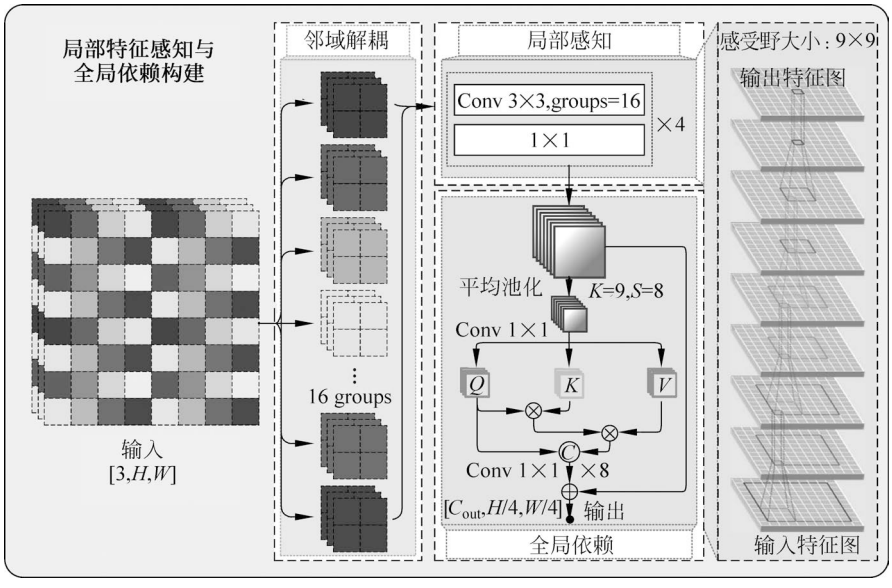



**图 3.2 局部特征感知与全局依赖构建网络结构示意图**

在建立全局依赖过程中为了避免产生过大的计算负担和内存占用，本节方法首先使用平均池化来减小特征图的大小，然后经过自注意力机制后插值回原始大小，并与输入相加。此处本章方法使用了核大小为 $9\times 9$ 步长为8的平均池化层。关于该参数设置的说明，详见3.3.2节的分析部分。

### 3.3.2　性能分析

**1. 设计原理**

继承3.2节的思想，并受网络浏览器使用缩略图对超分辨率图像进行快速加载和浏览的启发，本节使用ND操作将原始图像解耦为16组互补的1/4大小的缩略图，从而减少了每组缩略图内的冗余，同时允许所有像素能够参与后续处理以避免信息丢失。

在局部感知（LC）部分，正如图3.1中的例子所示，由于输入图像存在高度冗余，故ND解耦出的16组缩略图是相似的，因此原输入预处理过程可以分解为两个阶段，即先在每个缩略图中提取特征，然后对所有特征进行融合即可。为了保持与标准ResNet相同的层数，LCGB应用了四个级联的卷积组合，每个卷积组合由一个分组数为16的$3\times3$卷积和一个逐点卷积组成。

在全局依赖构建（GB）部分，LCGB使用平均池化层来进一步减少计算和内存占用。如图3.2的最右侧所示，LC输出映射到输入缩略图的感受野为$9\times9$，这表明在特征图中每隔9个像素就没有计算关联。因此，本节将平均池化层的核大小设置为9，步长设置为8，这样所有池化结果都会有所重叠，从而在不丢失原有关联性信息的情况下尽可能降低分辨率。在计算自注意力机制后，本方法将结果通过双线性插值扩大到1/4图像大小，并与LC的输出相加，从而对整个输出构建了全局依赖。

**2. 定量分析**

在计算效率方面，假设输入为彩色图像$[C_{\text{in}},H,W]$（$C_{\text{in}}=3$），输出为$[C_{\text{out}},H/4,W/4]$，LCGB卷积的计算量如式(3.7)～式(3.9)所示：

$$\text{FLOPs}_{\text{LC}}=\frac{\left(9C_{\text{in}}C_{\text{out}}+C_{\text{out}}^2\right)HW}{4} \tag{3.7}$$

$$\text{FLOPs}_{\text{GB}}=\frac{5C_{\text{out}}^2HW}{2048} \tag{3.8}$$

$$\text{FLOPs}_{\text{LCGB}}=\text{FLOPs}_{\text{LC}}+\text{FLOPs}_{\text{GB}} \tag{3.9}$$

感受野大小对初始特征提取至关重要。根据感受野的物理意义，同2.2.2节，本章也用原始输入图像中参与计算输出特征图中任一元素的像素数量（NPI）来量化感受野大小。除去GB获得的全局感受野，LC的感受野大小为$9\times9$（48个通道）。换算至原始输入图像，可得LC的NPI为$9\times9\times48=3888$，这说明仅LC就

获得了充足的感受野，具有提取有效的初始特征的能力。3.3.3节将通过与现有方法进行对比来进一步证明LCGB的优越性。

### 3.3.3 与现有方法对比

回顾2.2节中提到的典型网络输入预处理方法，LCGB与其结构对比示意图如图3.3所示。其中，图3.3(a)代表使用大卷积核级联池化层的单路径方法[1]，图3.3(b)代表使用并行小卷积和池化层的方法[14]，图3.3(c)代表单路径多个小卷积级联的方法[9]，图3.3(d)是本节所提出的方法。从结构上看，LCGB与其他方法的主要区别在于，本节方法不是简单地使用卷积，而是设计了精巧的输入预处理流程，通过无步长的ND生成信息无损的缩略图，分别对各个缩略图进行多分支特征提取，并构建了全局注意力机制。

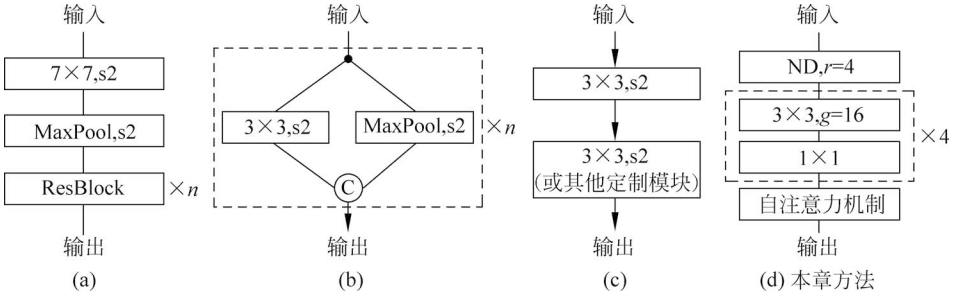



**图 3.3 局部特征感知与全局依赖构建方法与现有方法结构对比示意图**

LCGB与现有方法在感受野方面的定量对比见表3.1。与常用的ResNet-18网络结构相比，LCGB扩大了70%的感受野，而消耗的计算量却只有14.2%，这说明了LCGB在特征提取性能上的高效性。

**表 3.1 LCGB与现有方法在感受野方面的定量对比**

| 序　　号 | 特征图大小 | 代表性工作 | 参数设置 | 计算量 | NPI |
|---|---|---|---|---|---|
| (a) | [3,512,1024]<br>↓<br>[64,128,256] | ResNet-18[1] | $n$=2 | 6.065G | 5547 |
| (b) | | ERFNet[15] | $n$=2 | 0.414G | 147 |
| (c) | | HRNet[5] | — | 1.455G | 147 |
| (d) | | Ours(LC only) | — | 0.854G | 3888 |
| (d) | | Ours(LC+GB) | — | 0.859G | All |

## 3.4　高级特征的空间并行多尺度学习

语义解析需要充足的感受野和多尺度表征才能更好地完成密集预测任务。现有方法倾向于使用多个卷积核在通道维度上对相同的输入进行表征学习，这极有可能导致特征冗余。与现有方法不同，本节提出空间多尺度特征提取（SMFE）算法，通过在空间维度上进行多尺度特征提取，实现低运算量、低冗余、具有复合感受野的表征学习。

### 3.4.1　网络结构设计

空间多尺度特征提取（SMFE）算法采用解耦—变换—耦合的策略，结构示意图如图3.4所示。大小为$[C_{\text{in}}, H, W]$的输入首先被ND解耦为4组大小为$[C_{\text{in}}, H/2, W/2]$的缩略图；然后在每张缩略图上用不同的卷积核进行特征提取，得到4个形状为$[C_{\text{out}}, H/2, W/2]$的输出（此处卷积组合中卷积核大小分别为$\{1\times 1,\ 3\times 3,\ 1\times 5,\ 5\times 1\}$，选择此设置的相关分析详见表3.2）。最后，将所有通道级联，并使用一个$1\times 1$卷积进行特征融合。需要指出的是，如果该学习模块设定的步长为1，则$1\times 1$卷积将$4C_{\text{out}}$通道映射到$4C_{\text{out}}$，并用NC来整合缩略图并恢复分辨率。否则$1\times 1$卷积将$4C_{\text{out}}$通道映射到$C_{\text{out}}$，并直接输出结果。

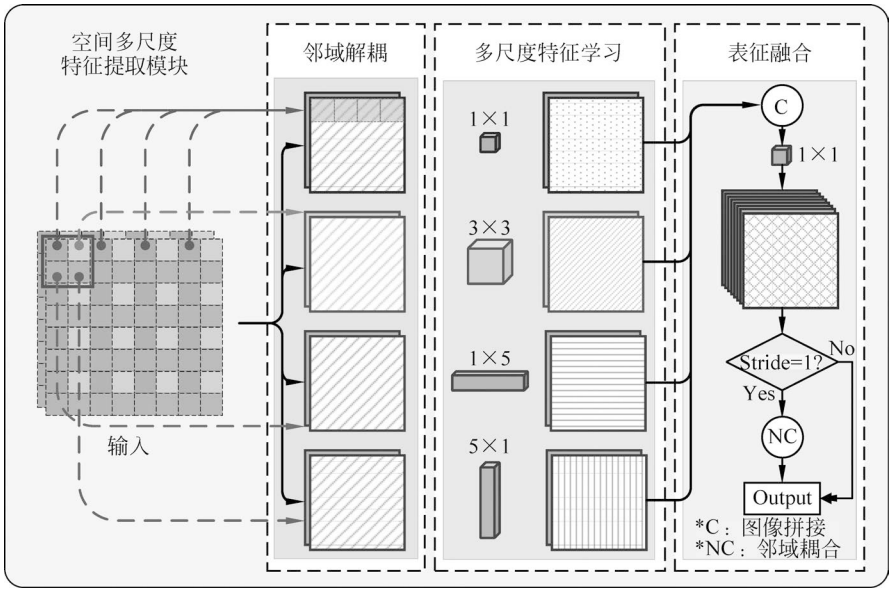



图 3.4　空间多尺度特征提取模块结构示意图

### 3.4.2 性能分析

SMFE在互补的缩略图上建立多尺度表征模型，在一定程度上避免了特征的冗余，大大减少了计算量。采用了多尺度和多形状的卷积核学习策略，有效提高了特征提取能力。

**1. 设计原理**

多分支结构已经被许多先前工作证明是有效的多尺度学习方式，其产生冗余的主要原因是对同一输入进行了多次特征提取。SMFE利用ND的空间解耦能力，生成了4组信息相似互补的子特征图（缩略图），每组缩略图都可以代表原始特征图信息，且分辨率大小只有原始输入一半。因此，当对每组缩略图进行卷积时，SMFE不仅可得到与全分辨率相似的结果，且能够节约大量计算成本。通过不同分支的多尺度学习和跨分支融合，SMFE便可获得丰富的多尺度特征。

对于多分支的卷积核参数设置，本节认为，场景解析的难点还集中于对细长的物体（如电线杆）、前景小而背景大的区域（如可以透过背景的长栅栏等）的感知效果较差。经研究发现，虽然对称卷积具有学习此类特征的能力，但非对称卷积的计算量和参数更少，具有对极端特征更敏感的优势。所以本章将多尺度卷积核与非对称卷积结合起来。SMFE中多尺寸多形状卷积核的学习效果如图3.5所示，可见非对称卷积使网络更容易提取具有极端尺寸的物体的特征，如电线杆、栅栏等。多尺度卷积则有利于提取同一物体在不同尺度上的特征。


**图 3.5 SMFE中多尺寸多形状卷积核学习效果示意图**

SMFE在完成多尺度特征提取后，通过逐点卷积进行跨分支的特征融合。多尺度特征提取的输出在空间上是相关的：如果步长为1，则应用NC将特征从深度耦合到空间以恢复空间关联，如果步长为2，则直接输出半分辨率结果。

此外，考虑一种输入分辨率非常小的极端情况，假设输入特征图大小只有2×2，则SMFE退化为深度卷积串联逐点卷积的模式，即MobileNet结构，此时多尺度特征的提取仍然有效。

**2. 定量分析**

为便于描述，假设SMFE的输入和输出有相同的通道数且步长为1，即该输入/输出的形状为$[C, H, W]$，则SMFE的计算量可由式(3.10)～式(3.12)得出：

$$\mathrm{FLOPs}_{\mathrm{learning}} = (1+9+5+5)\, C^2 \frac{H}{2}\frac{W}{2} = 5C^2HW \tag{3.10}$$

$$\mathrm{FLOPs}_{\mathrm{fusion}} = 4C \times 4C \times \frac{H}{2}\frac{W}{2} = 4C^2HW \tag{3.11}$$

$$\mathrm{FLOPs}_{\mathrm{SMFE}} = \mathrm{FLOPs}_{\mathrm{learning}} + \mathrm{FLOPs}_{\mathrm{fusion}} = 9C^2HW \tag{3.12}$$

相比而言，标准ResNet中的基础残差模块ResBlock的计算量如下式所示：

$$\mathrm{FLOPs}_{\mathrm{resnet}} = (9+9)\, C^2HW = 18C^2HW \tag{3.13}$$

对比式(3.12)和式(3.13)可得，SMFE比标准ResBlock节约了50%的计算量。

在特征提取能力方面，图3.6显示了SMFE在原始特征图上的等效多尺度核。与只有$5\times5$感受野的ResBlock相比，SMFE具有更复杂、更宽阔的感受野，并且在特征采样方面具有一定的稀疏性，从而能够提供更强的特征提取能力。

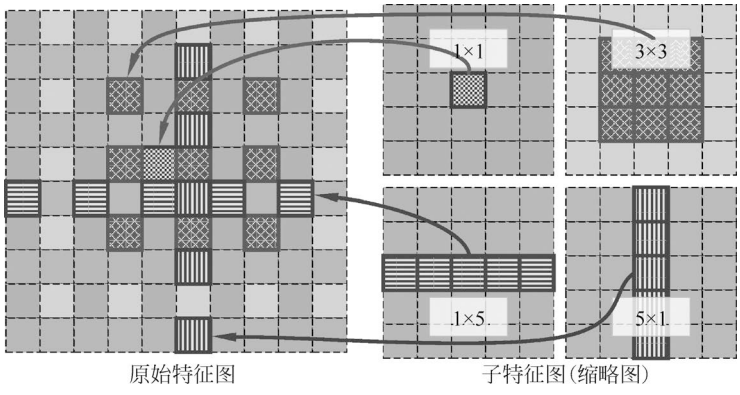



**图 3.6　SMFE在原始特征图上的等效多尺度核示意图**

### 3.4.3 与现有方法对比

现有方法大多在通道维度上采用分组卷积或多分支结构，遵循“分割—变换—合并”的方法实现多尺度学习与计算量削减，而本节所提出的方法是在空间维度上，按照“解耦—变换—耦合”的策略，在单步操作内实现多尺度并行学习。SMFE与典型多尺度学习方法在结构上的对比示意图详见图3.7，其中图3.7(a)代表在同一输入上应用多尺度卷积核的方法[34]，图3.7(b)代表将输入在通道维度分组并分别使用多尺度卷积核的方法[129]，图3.7(c)代表对输入进行金字塔池化的方

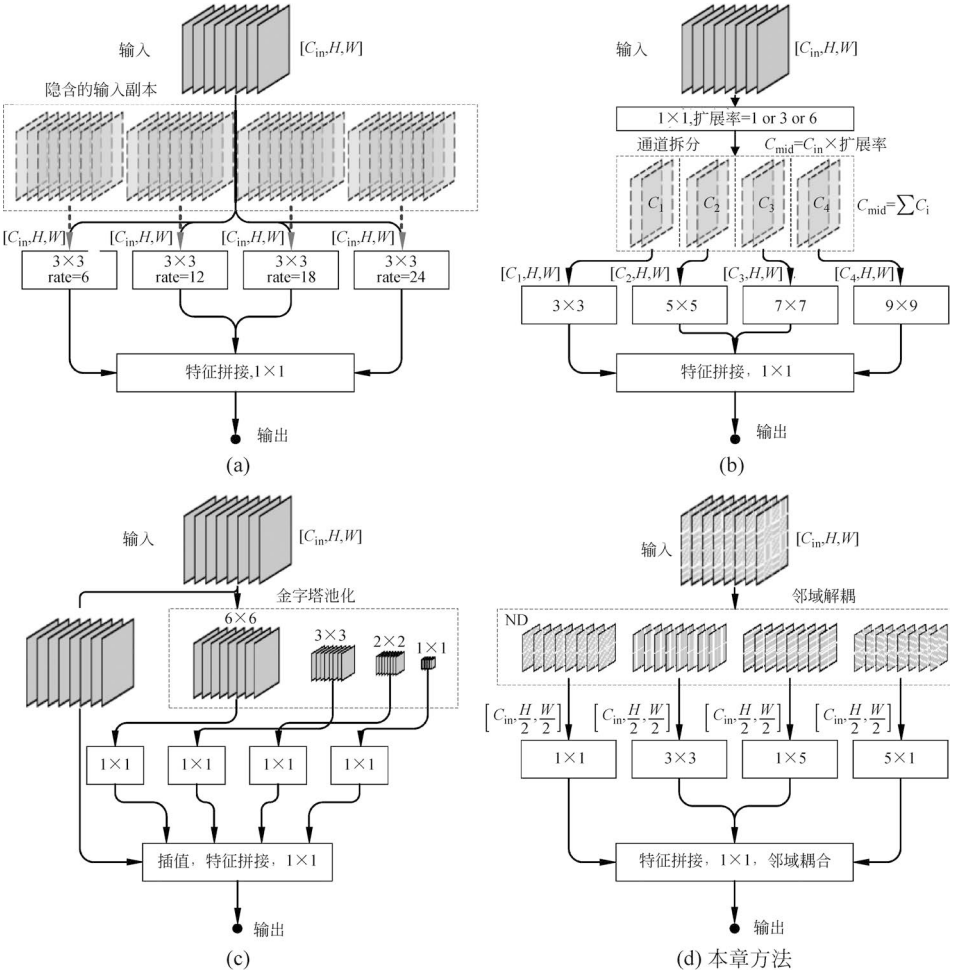



**图 3.7 SMFE与典型多尺度学习方法在结构上的对比示意图**

法[35]。相关定量比较结果详见表3.2。对于图3.7(a)和图3.7(b)所代表的方法，特征图大小是恒定的，每组卷积核生成的结果是最终输出在通道维度上的一个子集，而在图3.7(c)中，特征金字塔是通过将输入转化为多个尺度来构建的，不同尺度的特征表征也是最终输出在通道维度上的一个子集。相比之下，在本节所提出的方法中，每组的通道数是恒定的，而特征图是原始输入在空间维度上的一个互补子集。

**表 3.2　SMFE与典型多尺度学习方法的定量比较结果**

| 序号 | 尺　寸 | 代表性工作 | 参 数 设 置 | 计　算　量 |
|---|---|---|---|---|
| (a) | [256,64,128]<br>↓<br>[256,64,128] | ASPP[34] | 核大小=[1,3,3,3] | 17.20G |
| | | | 空洞率=[1,6,12,18] | |
| (b) | | MixNet[129] | 扩张率=6 | 7.01G |
| | | | 核大小=[3,5,7,9] | |
| (c) | | PPM[35] | 特征图大小=[1,2,3,6] | 2.70G |
| (d) | | SMFE(本节) | 核大小=[1,3,(1,5),(5,1)] | 4.84G |

## 3.5　实验结果与分析

本节首先构建了4种不同模型容量的实验模型，称为NDNet，并在 Cityscapes[107]数据集的验证集上进行消融研究以充分验证本章每节提出方法的有效性；然后使用大型数据集ImageNet[130]上的预训练模型，在Cityscapes[107]和CamVid[105]数据集上进行重新训练与评测，并与现有先进方法进行了对比；最后讨论了本章所提出方法的优势与局限性。

### 3.5.1　实验模型

参照ResNet[1]和高效网络DFNet[93]的结构，本节构建了一系列模型容量不同的网络，表3.3为NDNet网络结构详细参数表。

### 3.5.2　消融研究

本节使用NDNet-18模型在Cityscapes[107]验证集的测试结果来展示本章所提出方法的有效性。为了进行公平的比较，与2.5节相同，本节将标准的ResNet-18作为骨干网络，并将其中相应的结构替换为LCGB和SMFE。在高分辨率语义输出方面，本节使用了2.4节中提出的方法PRM。由于ND/NC贯穿于LCGB和SMFE

表 3.3　NDNet 网络结构详细参数表

| 网络阶段 | | 输出特征图尺寸 | | NDNet-18 | NDNet-34 | NDNet-DF1 | NDNet-DF2 |
|---|---|---|---|---|---|---|---|
| | | 分类 | 分割 | | | | |
| 输入 | | 224×224 | 512×1024 | 图像输入 | | | |
| 输入预处理 | | 56×56 | 128×256 | LCGB,64 | LCGB,64 | LCGB,64 | LCGB,64 |
| 阶段 3 | | 28×28 | 64×128 | [SMFE, 128]×2 | [SMFE, 128]×5 | [SMFE, 64]×3 | [SMFE, 64]×2 |
| | | | | | | | [SMFE, 128]×1 |
| 阶段 4 | | 14×14 | 32×64 | [SMFE, 256]×2 | [SMFE, 256]×6 | [SMFE, 128]×3 | [SMFE, 128]×10 |
| | | | | | | | [SMFE, 256]×1 |
| 阶段 5 | | 7×7 | 16×32 | [SMFE, 512]×2 | [SMFE, 512]×3 | [SMFE, 256]×3 | [SMFE, 256]×4 |
| | | | | | | [SMFE, 512]×1 | [SMFE, 512]×2 |
| 任务 | 分类 | 1×1 | — | 依次：全局池化, 1000 维全连接层, Softmax 函数 | | | |
| | 分割 | — | 512×1024 | PRM（来自2.4节）, 8 倍上采样 | | | |
| 参数量 | 分类 | — | — | 17.44M | 37.54M | 14.50M | 33.33M |
| | 分割 | — | — | 18.69M | 38.77M | 15.57M | 34.59M |
| 计算量 | 分类 | — | — | 1.03G | 1.99G | 0.57G | 1.18G |
| | 分割 | — | — | 13.39G | 21.86G | 7.37G | 14.33G |

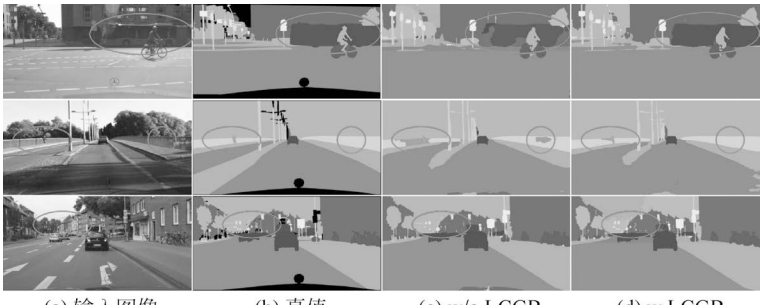
之中，且无法独立剥离，因而本节在讨论LCGB和SMFE的作用之后，再分别对LCGB和SMFE中的ND/NC进行消融实验。

本节训练并验证了7个网络模型，表3.4给出了NDNet各组成部分消融研究结果，具体比较了各个模型在参数量、计算量、内存占用峰值、mIoU及推理时间等指标上的不同性能，其中最后一行代表是NDNet-18。值得注意的是，为了保证公平性，内存消耗是在TensorRT优化后的模型上进行评估的，这有助于避免因各种代码实现而产生的内存差异。后文将进一步对该结果进行详细分析。

**1．初始特征的局部感知与全局建模的实验分析**

**1）功能分析**

表3.4显示LCGB大幅减少了计算量，同时将mIoU提高了约1%。然而，可能是由于多分支结构的原因，LCGB在推理速度上没有提升。图3.8中的消融LCGB前后网络分割效果样例说明了LCGB的优异表现：第一行显示了LCGB在扩大感受野方面的作用，第二行显示了具有长距离依赖性的上下文信息对分割结果的影响，第三行显示了LCGB在保存微小物体信息方面的优势。



(a) 输入图像　(b) 真值　(c) w/o LCGB　(d) w LCGB

**图 3.8　消融初始特征的局部感知与全局建模方法前后网络分割效果样例**

**2）各组成部分有效性研究**

LCGB由三部分组成：ND、LC和GB。本节分别研究了每一部分的作用，该实验中ND和LC被看作一个整体，因为它们共同实现了下采样和初始特征提取。表3.5展示了LCGB各组成部分有效性研究的实验结果，可以看出每个部分对最终结果都有积极的影响。

**2．高级特征的空间并行多尺度学习的实验分析**

**1）功能分析**

SMFE有效提升了准确性（2%↑）和推理速度。然而，SMFE对所有缩略图进

表 3.4　NDNet 各组成部分消融研究结果

| LCGB | SMFE | 参数量 | 计算量 | Δ 计算量节约↓‡ | 内存占用峰值† | | | | mIoU | 推理时间 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 推理引擎（Engine） | 中间过程（Context） | 合计 | Δ 内存占用减少↓‡ | | |
| — | — | 14.15M | 25.23G | — | 68M | 169M | 237M | — | 69.3% | 12.56ms |
| ✓ | — | 14.03M | 19.99G | 20.8%↓ | 70M | 146M | 216M | 8.9%↓ | 70.5% | 11.76ms |
| — | ✓ | 20.61M | 21.83G | 13.5%↓ | 72M | 156M | 228M | 3.8%↓ | 71.6% | 10.22ms |
| ✓ | — | **12.31M** | 16.80G | 33.4%↓ | 59M | 136M | 195M | 17.7%↓ | 71.2% | 11.78ms |
| — | ✓ | 18.81M | 18.63G | 26.2%↓ | 69M | 136M | 205M | 13.5%↓ | 72.3% | 10.02ms |
| ✓ | ✓ | 18.69M | **13.39G** | **46.9%↓** | 70M | **130M** | 200M | 15.6%↓ | **73.7%** | **9.3ms** |

注：† 内存占用峰值是在 TensorRT 优化后的模型上进行测量的，其中“推理引擎”（Engine）指的是计算流图和权重的内存占用，“中间过程”（Context）是指与模型的各模块实现和中间结果相关的内存消耗。

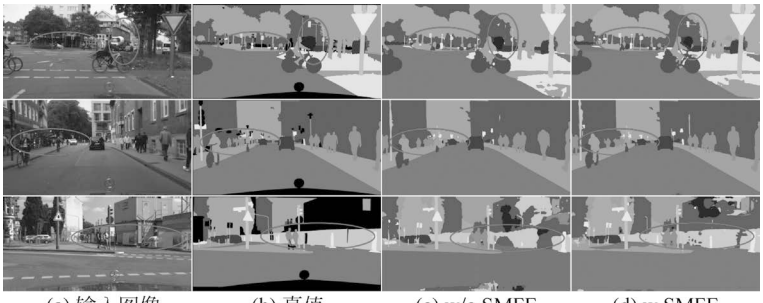
‡ Δ 是每个模型相对于基线（第一行）的减少比率。

表 3.5　初始特征的局部感知与全局建模方法各组成部分有效性研究

| [C3, s2] × 2 | ND+ LC | GB | 参　数　量 | 计　算　量 | mIoU |
|---|---|---|---|---|---|
| ✓ | — | — | 12.31M | 17.40G | 69.96% |
| — | ✓ | — | 12.29M | 16.79G | 70.11% |
| ✓ | — | ✓ | 12.32M | 17.40G | 70.89% |
| — | ✓ | ✓ | 12.31M | 16.80G | **71.22%** |

注：表中“C3, s2”代表步长为 2 的 3×3 卷积。

行的多尺度特征提取，增加了网络参数。图3.9中的消融SMFE前后网络分割效果样例，说明了SMFE在三方面对网络性能的提升，包括对细长物体（远处的电线杆）、不同尺度的同一物体（人和自行车）以及前景与背景比例非常小的区域（长长的栅栏）等精确的感知与定位。



(a) 输入图像　(b) 真值　(c) w/o SMFE　(d) w SMFE

**图 3.9　消融SMFE前后网络分割效果样例**

**2）参数设置研究**

SMFE的各个分支可做不同卷积核设置，本实验研究了一些典型的卷积核组合，其性能表现如表3.6所示。

**表 3.6　SMFE在不同卷积核设置下的性能对比**

| 设　置 | 参 数 量 | FLOPs | mIoU | 推 理 时 间 |
|---|---|---|---|---|
| 核大小=[3,3,3,3] | 27.06M | 23.46G | **73.02%** | 14.50ms |
| 空洞率=[1,2,3,5] | | | | |
| 核大小=[1,3,(1,3),(3,1)] | 16.74M | 17.42G | 71.32% | **9.90ms** |
| 空洞率=[1,1,(1,2),(2,1)] | | | | |
| 核大小=[1,3,(1,5),(5,1)] | 18.81M | 18.63G | 72.30% | 10.02ms |
| 空洞率=[1,1,1,1] | | | | |

由表3.6中结果可以看出：虽然使用不同空洞率的$3 \times 3$卷积可以获取较大的感受野以及最高的精度，但其计算开销是最大的，推理时间也是最长的。而表中第二行的设置所捕获的感受野虽与本章的相同，但其精度不高，速度优势也不明显，这可能是因为多空洞率设置造成了部分信息损失。综合考虑，本章采用了第三行所示的参数。

**3）与现有代表性方法的性能对比**

本节将SMFE与代表性多尺度学习方法进行了比较。本实验用ASPP[34]、Mix-

Conv[129]和PPM[35]分别替换了NDNet-18中的SMFE，实验结果见表3.7。可见，SMFE在提升准确率和推理时间方面具有明显优势，这也是本章关注的重点。尽管MixNet在参数数量上有优势，但它的实时性能比SMFE差。这是因为MixNet虽然使用大量的深度可分离卷积大幅减少了参数的数量，然而并行的多分支深度可分离卷积计算对GPU并不友好，阻碍了其实时性能。ASPP的精度与SMFE最接近，但由于ASPP以不同卷积核大小对同一输入进行了多次卷积，导致其计算成本极高，严重影响了整体算法实时性。

**表 3.7 SMFE与代表性多尺度学习方法性能对比**

| 方法名称 | 参数量 | FLOPs | mIoU | 推理时间 |
|---|---|---|---|---|
| ASPP[34] | 18.70M | 24.27G | 73.0% | 11.97ms |
| MixNet[129] | **7.50M** | 13.49G | 72.6% | 10.36ms |
| PPM[35] | 14.66M | 17.49G | 71.6% | 12.02ms |
| **SMFE(本研究)** | 18.69M | **13.39G** | **73.7%** | **9.3ms** |

**3. 空间邻域解耦-耦合算子的实验分析**

为了进一步探讨ND/NC与pixel-unshuffle/shuffle（简称PU/PS）对网络性能的影响，本实验将LCGB和SMFE中的所有ND/NC相应地替换为PU/PS，实验结果如表3.8所示。从表中可以看出，用PU/PS代替ND/NC后，网络的准确性有所下降。这也证实了3.2.2节的分析，即通过对ND采样的不同缩略图进行卷积可以近似于对原始输入进行多次特征提取，从而在预测准确率相近的情况下，极大地减小计算量。与此相反，在pixel-unshuffle之后对结果进行分组，会导致通道间的关系被打破。因此，每组的通道信息是不平衡的，对不同组的特征提取也是不平衡和不充分的。对表3.8结果分析可知，PU/PS对SMFE有极大的负面影响，这是由于SMFE中的多尺度核加剧了特征学习的不平衡性。因此，ND/NC赋予了LCGB和SMFE以更强的特征提取能力。

**表 3.8 空间邻域解耦-耦合算子消融实验结果**

| LCGB | | SMFE | | mIoU |
|---|---|---|---|---|
| ND/NC | PU/PS | ND/NC | PU/PS | |
| — | ✓ | ✓ | — | 72.8% |
| ✓ | — | — | ✓ | 72.3% |
| — | ✓ | — | ✓ | 72.0% |
| ✓ | — | ✓ | — | **73.7%** |

**4. 内存占用分析**

由于ND/NC仅改变了像素的排列方式，对内存占用不会产生影响，故而本节仅分析LCGB和SMFE对内存占用的影响。由表3.4可知，LCGB和SMFE的加入都促进了总内存消耗的减少。由于多个缩略图的存在使得计算流图更加复杂，LCGB略微提高了推理引擎的内存占用；但是，由于缩略图的尺寸较小，它也减少了中间结果的内存开销。同理，SMFE由于更多的参数和多分支结构增加了推理引擎的内存使用量，但是由于SMFE在半分辨率特征图上进行卷积计算，其中间结果的内存占用也得以降低，综合推理引擎和中间变量情况，SMFE在总体上仍降低了总的内存使用量。综合LCGB和SMFE，NDNet在推理引擎内存占用上获得了与基线相近的结果，但却大大降低了中间变量内存消耗，从而实现了最低的计算量以及最快的推理速度。这表明了本章所提出方法在减少内存用量和提高实时性能方面具有突出优势。

### 3.5.3　与当前先进方法的性能对比

本节首先在ImageNet[130]上进行预训练，并与基线模型进行对比；然后在Cityscape[107]和CamVid[105]数据集的测试集上进行实验，并与目前先进的实时语义解析算法进行对比。

**1. 模型预训练**

表3.9展示NDNet在ImageNet分类任务上的实验结果。可以看出，与原始网络（ResNet或DFNet）相比，NDNet在提高准确率的同时，大大降低了计算成本。由于NDNet对每个缩略图都单独提取了特征，所以NDNet中的参数量较基线更多。对于实时系统而言，存储设备的发展使得存储能力已不再是主要瓶颈，相比之下，计算能力的限制更为关键，由3.5.2节分析，NDNet在计算性能方面更有优势。因此，与标准的ResNet风格的网络相比，NDNet在实时性和准确性上都更有优势。

**表 3.9　NDNet在ImageNet分类任务上的实验结果**

| 网络名称 | 参数量 | 计算量 | 计算量削减比例 | Top-1 准确率 |
|---|---|---|---|---|
| ResNet-18[1] | 11.2M | 1.82G | — | 69.0% |
| **NDNet-18** | 17.4M | **1.03G** | **43.4% ↓** | **72.1%** |
| ResNet-34[1] | 21.3M | 3.67G | — | 73.6% |
| **NDNet-34** | 37.5M | **1.99G** | **45.8% ↓** | **76.4%** |

续表

| 网 络 名 称 | 参 数 量 | 计 算 量 | 计算量削减比例 | Top-1 准确率 |
|---|---|---|---|---|
| DF1[93] | 8.0M | 0.75G | — | 69.8% |
| **NDNet-DF1** | 14.5M | **0.57G** | **24.0% ↓** | **70.9%** |
| DF2[93] | 17.5M | 1.77G | — | 73.9% |
| **NDNet-DF2** | 33.3M | **1.18G** | **33.3%↓** | **75.4%** |

**2. 实时语义解析性能对比**

本实验在Cityscapes[107]测试集上对NDNet进行了性能评估，并与目前先进的实时语义解析算法进行对比，如图3.10所示。具体对比数据可见表3.10。

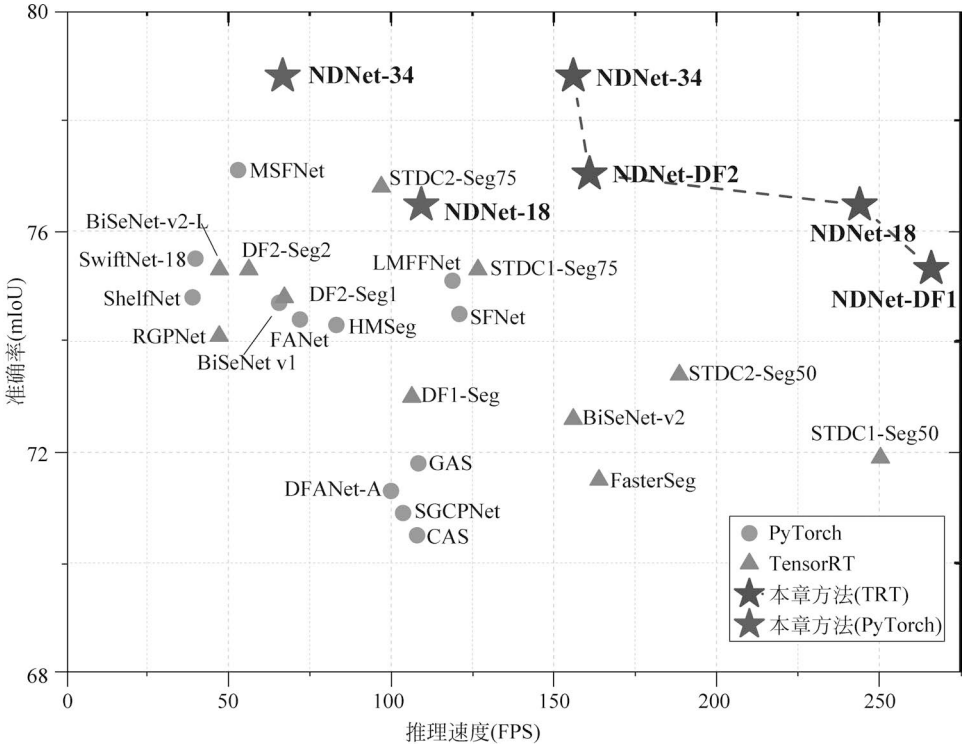



**图 3.10 Cityscapes上NDNet与先进方法在速度（FPS）与准确率（mIoU）上的对比**

由表3.10，NDNet在准确性和实时性方面都取得了优异的效果。为了公平比较，本节分别列出了使用PyTorch和TensorRT在两种常用的GPU上的推理速度。可以看出，NDNet比其他具有类似精度的方法快得多，例如，与STDC2-Seg75相

比（mIoU 76.8% vs 76.47%（本章方法）），NDNet-18在RTX 2080Ti/Titan X上分别快了2.52倍/1.68倍；与MSFNet相比（mIoU 77.1% vs 78.8%（本章方法）），NDNet-34在速度上快了1.26倍，同时获得了1.6%的准确率提升。

**表 3.10　NDNet与现有先进方法在Cityscapes测试集上的对比**

| 方法名称 | 输入分辨率 | 推理GPU型号 | mIoU(%) | 推理速度 (FPS) | |
|---|---|---|---|---|---|
| | | | | PyTorch | TensorRT |
| DFANet A[29] | 1024×1024 | Titan X | 71.3 | 100 | |
| DFANet B[29] | 1024×1024 | Titan X | 67.1 | 120 | |
| BiSeNet v1[28] | 768×1536 | Titan Xp | 74.7 | 65.6 | — |
| BiSeNet v2[39] | 512×1024 | GTX 1080Ti | 72.6 | — | 156 |
| BiSeNet v2-L[39] | 512×1024 | GTX 1080Ti | 75.3 | — | 47.3 |
| SwiftNet-18[114] | 1024×2048 | GTX 1080Ti | 75.5 | 39.9 | — |
| FANet[115] | 1024×2048 | Titan X | 74.4 | 72 | — |
| ShelfNet[116] | 1024×2048 | GTX 1080Ti | 74.8 | 36.9 | — |
| GAS[119] | 769×1537 | Titan Xp | 71.8 | 108.4 | — |
| CAS[120] | 768×1536 | GTX 1070 | 70.5 | 108 | — |
| HMSeg[122] | 768×1536 | GTX 1080Ti | 74.3 | 83.2 | — |
| SFNet[124] | 512×1024 | GTX 1080Ti | 74.5 | 121 | — |
| MSFNet[125] | 1024×2048 | RTX 2080Ti | 77.1 | 41 | — |
| SGCPNet[131] | 1024×2048 | GTX 1080Ti | 70.9 | 103.7 | — |
| LMFFNet[132] | 512×1024 | RTX 3090 | 75.1 | 118.9 | — |
| FasterSeg[94] | 1024×2048 | GTX 1080Ti | 71.5 | — | 163.9 |
| STDC1-Seg50[10] | 512×1024 | GTX 1080Ti | 71.9 | — | 250.4 |
| STDC2-Seg50[10] | 512×1024 | GTX 1080Ti | 73.4 | — | 188.6 |
| STDC1-Seg75[10] | 768×1536 | GTX 1080Ti | 75.3 | — | 126.7 |
| STDC2-Seg75[10] | 768×1536 | GTX 1080Ti | 76.8 | — | 97.0 |
| DF1-Seg[93] | 1024×2048 | GTX 1080Ti | 73.0 | — | 106.4 |
| DF2-Seg1[93] | 1024×2048 | GTX 1080Ti | 74.8 | — | 67.2 |
| DF2-Seg2[93] | 1024×2048 | GTX 1080Ti | 75.3 | — | 56.3 |
| RGPNet[117] | 1024×2048 | RTX 2080Ti | 74.1 | — | 47.2 |
| NDNet-18 | 512×1024 | TiTan X | 76.47 | 90.9 | 163 |
| | | RTX 2080Ti | | 109.3 | 244 |

续表

| 方法名称 | 输入分辨率 | 推理GPU型号 | mIoU(%) | 推理速度 (FPS) | |
|---|---|---|---|---|---|
| | | | | PyTorch | TensorRT |
| NDNet-34 | 512×1024 | TiTan X | **78.80** | 52.6 | 103 |
| | | RTX 2080Ti | | 66.7 | 156 |
| NDNet-DF1 | 512×1024 | TiTan X | 75.32 | 80.0 | 189 |
| | | RTX 2080Ti | | 105.2 | **266** |
| NDNet-DF2 | 512×1024 | TiTan X | 77.03 | 48.3 | 119 |
| | | RTX2080Ti | | 62.5 | 161 |

NDNet与现有先进方法在CamVid[105]测试集上的对比结果如表3.11所示，其中MSFNet* 表示在768 × 512分辨率下的测试结果，而MSFNet** 的分辨率则为1024 × 768；其他所有方法的评测指标都是在960 × 720分辨率下获取的。该结果同样说明了本章方法在权衡语义解析任务准确率与推理速度方面的优越性。总体而言，本章所提出的方法达到了目前实时语义解析领域先进性能。

**表 3.11 NDNet与现有先进方法在CamVid测试集上的对比**

| 方法名称 | 输入分辨率 | 推理GPU型号 | mIoU(%) | 推理速度 (FPS) | |
|---|---|---|---|---|---|
| | | | | PyTorch | TensorRT |
| DFANet A[29] | 720×960 | Titan X | 64.7 | 120 | — |
| DFANet B[29] | 720×960 | Titan X | 59.3 | 160 | — |
| BiSeNet v1[28] | 720×960 | Titan Xp | 65.6 | 175 | — |
| FANet[115] | 720×960 | Titan X | 69.0 | 154 | — |
| RGPNet[117] | 720×960 | RTX 2080Ti | 66.9 | 90.2 | — |
| DCNet[118] | 720×960 | RTX 2080Ti | 66.2 | 166 | — |
| GAS[119] | 720×960 | Titan Xp | 72.8 | 153.1 | — |
| CAS[120] | 720×960 | GTX 1070 | 71.8 | 169 | — |
| DSANet[123] | 720×960 | GTX 1070 | 69.9 | 75.3 | — |
| SFNet[124] | 720×960 | GTX 1080Ti | 73.8 | 35.5 | — |
| MSFNet[125]* | 512×768 | RTX 2080Ti | 72.7 | 160 | — |
| MSFNet[125]** | 768×1024 | RTX 2080Ti | 75.4 | 91 | — |
| BiSeNet v2[39] | 720×960 | GTX 1080Ti | 72.4 | — | 124.5 |
| BiSeNet v2-L[39] | 720×960 | GTX 1080Ti | 73.2 | — | 32.7 |
| STDC1-Seg50[10] | 720×960 | GTX 1080Ti | 73.0 | — | **197.6** |

续表

| 方 法 名 称 | 输入分辨率 | 推理GPU型号 | mIoU(%) | 推理速度(FPS) | |
|---|---|---|---|---|---|
| | | | | PyTorch | TensorRT |
| STDC2-Seg50[10] | 720×960 | GTX 1080Ti | 73.9 | — | 152.2 |
| NDNet-18 | 720×960 | RTX 2080Ti | 76.0 | 76.9 | 175 |
| NDNet-34 | 720×960 | RTX 2080Ti | **77.9** | 52.7 | 108 |

### 3.5.4　算法优势与局限性分析

为了更好地展示NDNet的优越性和局限性，本节在Cityscapes[107]验证集上分别对推理速度最快的NDNet-18和准确率最高的NDNet-34进行了探究，并选择了图3.11所示的8个样例进行进一步分析讨论。

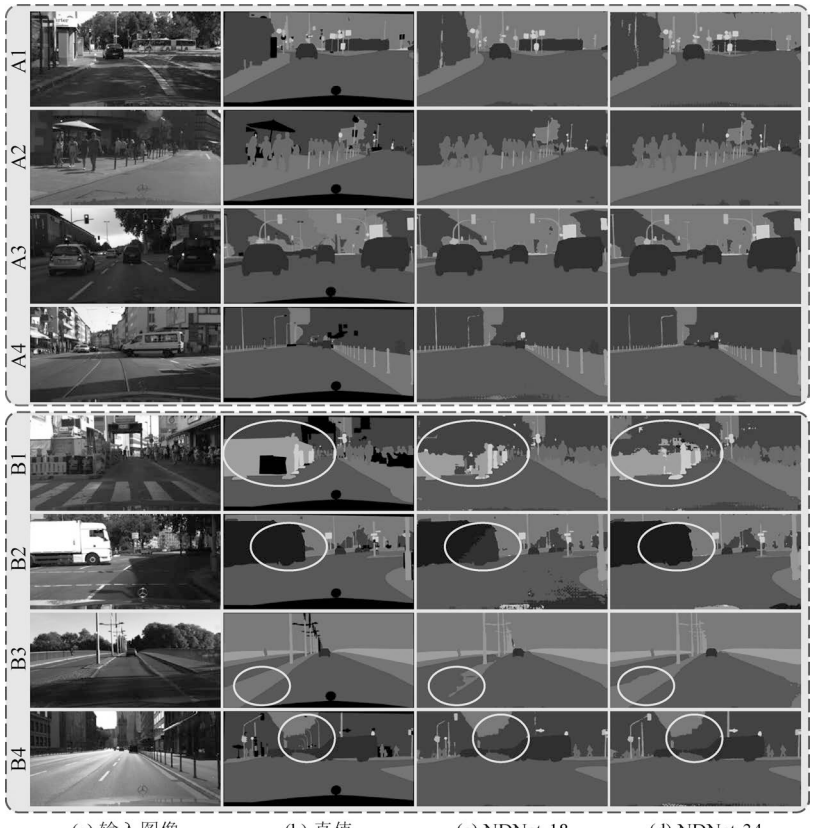



**图 3.11　NDNet算法优势与局限性分析样例**

由上至下，本节在A组图中分别展示了NDNet在准确的边界定位、微小物体识别、多尺度目标感知、细长物体分割以及避免语义混淆等方面的能力。

而在B组图中则展示了一些失败案例。其中，图B1展示了一个在半透明背景（弱语义）干扰下的失败案例。图B2和图B3展示了不同感受野对分割结果的影响。具有大感受野的深层网络对大型物体及大面积干扰下的区域感知具有更强的预测能力。图B4显示了NDNet在预测远端电线杆上的不足。其原因在于两方面。一方面是远处的电线杆被建筑物的语义所淹没。另一方面是在特征提取过程中，电线杆在缩略图中的有效像素太少，导致了信息丢失。

针对图B1~B3所暴露的在多层级特征聚合时语义吞噬与感受野不足等方面的问题，本书将在第4章通过对知识表征问题的研究来进行优化。图B4则指出了邻域解耦方法的固有问题，而如何在缩略图生成和并行特征提取过程中处理像素关联以减少细小物体信息损失，则成为NDNet的未来工作之一。

## 3.6 本章小结

面对场景语义解析任务在大规模输入预处理、多尺度学习等方面挑战，本章提出了一种基于邻域解耦-耦合的新的空间维度表征学习方案。本章首先定义了邻域解耦（ND）及其逆运算邻域耦合（NC）；随后提出了用于大规模输入预处理的局部感知和全局依赖构建（LCGB）方法，和用于轻量级表征学习的空间多尺度特征并行提取（SMFE）网络。大量实验论证了本章所提方法的有效性和优越性。本章内容的未来工作包括使用轻量级操作进一步提高计算效率，特别是合并相邻的“解耦-耦合”操作，以及通过神经网络架构搜索（NAS）算法，设计合理的搜索空间与搜索机制，进一步确定适用于不同阶段的网络参数和更优的整体网络架构等。

面对多层级特征聚合时语义吞噬与感受野不足等问题，本书将在下一章通过研究知识表征方法，充分利用高维先验知识与多层级多尺度特征，实现全图感受野与特征高效融合。