

武汉.LuoJiaNET/SET 产业应用白皮书



引言：

随着大数据、人工智能等技术的发展，基于深度学习的遥感影像解译与监测技术表现出了一定的优势。由于遥感对地观测与智能处理的战略价值，国外在2020年将基于人工智能的遥感技术列为敏感技术并限制出口，形成新一轮封锁态势。各类国内外开源深度学习框架，虽然能适用于通用图像处理，但目前尚无顾及遥感大幅面、多通道、知识融合特性的深度学习框架和相应任务驱动的大规模遥感影像样本库。

武汉大学在国家自然科学基金委重大研究计划集成项目“大规模遥感影像样本库构建及开源遥感深度网络框架模型研究”（项目编号：92038301）的支持下，与华为昇腾AI团队合作，双方共享知识产权，共同研发了全球首个遥感影像智能解译专用深度学习框架LuoJiaNET和业界最大遥感影像样本库LuoJiaSET，并在华为昇思社区上线。为了进一步推进科研成果的产业落地，武汉大学与华为技术公司联合发布《大规模遥感样本库LuoJiaSET及遥感深度学习框架LuoJiaNET产业应用白皮书》，以期形成“全栈”自主可控的智能遥感生态。

本白皮书分为6部分，包括国内外技术的发展现状及趋势、LuoJiaSET样本库平台功能介绍、LuoJiaNET框架安装部署及遥感特性介绍、LuoJiaNET典型应用模型介绍、LuoJiaNET典型行业应用介绍、以及LuoJiaSET与LuoJiaNET的发展与展望。

白皮书亮点摘要：

1. 遥感智能化处理的云平台，仍有较大提升空间。

当前遥感影像样本存在分类体系不统一、样本数量不足、时空分布不均、标注效率不高等问题，需建立统一的遥感影像样本分类体系，研发支持众包协同的样本标注工具，建立完善的版权保护体系，形成分类统一、类型完备、具有自适应扩展能力的遥感影像样本库；同时主流开源深度学习框架与模型主要针对的是普通影像，即普通的小像幅室内/外影像；现有遥感能解译与监测系统大多由通用图像识别的深度神经网络改造，一般只考虑了影像二维空间的可见光图像特征，顾及遥感光谱特性、地学先验知识、数据与框架协同等重要因素，支持遥感地学特性的框架仍是空白。武汉大学与华为技术公司联合研发的 LuoJiaSETT 与 LuoJiaNET 构建了从底层硬件到样本框架、再到面向智能测图任务的上层应用模型的全链条自主可控智能化处理云平台。

2. LuoJiaSET 样本库可提供样本的统一规范化接口。

LuoJiaSET 是遥感领域满足 OGC 标准的大规模遥感影像样本库，其制定了支持全球范围的遥感影像样本分类标准、标注规范，建立涵盖不同遥感任务的统一分类体系，形成样本要素的采集要求、内容和流程规范，可支持多级别、多类型遥感影像样本库的采集、制作、管理、共享、应用。目前，形成了包含 82 个面向遥感能解译的样本数据集、总数超过 500 万张的样本库。LuoJiaSET 构建了 100 类+的细粒度地表覆盖数据集 LuoJia-FGLC(Fine Grain Land Cover,)、国际上最大范围的航空高光谱与高分辨率数据集 LuoJia-HSSR(High Spatial and Spectral Resolution)和第一个基于公开地理信息的省域大范围典型地表覆盖数据集 LuoJia-HBD4 (Hubei Dataset of 4 Land Cover Types)，有效支撑

了国家自然科学基金委“智能遥感处理大赛”等项目。

3. LuoJiaNET 框架充分融合遥感特性。

LuoJiaNET 由武汉大学 LuoJiaNET 框架团队与华为昇腾 AI 团队联合打造而成，是遥感领域首个自主可控的遥感专用机器学习框架。针对遥感数据像幅尺寸大、数据通道多、尺度变化大等特性，具备内存可扩展、尺度通道灵活创建、数据通道自主优选、框架与数据协同处理的特点。可兼容已有深度学习框架，并提供用户友好的、可拖拽的交互式网络结构搭建界面的方法。能屏蔽不同硬件设备间差异，同时管理多样化的遥感影像样本库，实现遥多源感影像样本的高效读写与存储管理。其与国产人工智能硬件昇腾 NPU 深度融合，使智能计算软硬件充分协同，形成融合探测机理与地学知识的统一计算图表达、编译优化、图算融合、自动混合并行的新一代遥感能解译框架，可进行遥感样本的自动提纯与增广，同时充分融合遥感地学知识。

4. LuoJiaNET 框架提供 5 大类遥感基础应用模型。

为了验证 LuoJiaNET 遥感专用深度学习框架的有效性，发挥其优势，构建了基于 LuoJiaNET 框架的典型应用解译模型，包括：(1) 场景分类模型；(2) 目标检测（侧重细粒度）；(3) 语义分割模型（含多光谱/高光谱）；(4) 多视三维生成模型；(5) 变化检测模型。系列模型可兼容华为昇腾 AI 硬件架构，并对其性能、效果与主流的框架 PyTorch、Tensorflow 等构建的模型进行对比分析。此外，提供了大幅面影像处理、遥感知识嵌入的核心解决方案。

5. LuoJiaSET 与 LuoJiaNET 框架为遥感行业应用提供支撑。

LuoJiaNET 与 LuoJiaSET 为上海数慧、武汉汉达瑞科技公司、航天宏图信息技术股份有限公司、珈和科技等行业单位提供遥感基础应用模型，共同构筑自

主可控开源遥感生态。在公测的一个月时间内，已有超 3000 人次的下载量，多家大型公司将 LuoJiaSET 与 LuoJiaNET 作为遥感人工智能基础软件平台。

目 录

1 概述	1
1.1 编写目的	1
1.2 项目背景	1
1.3 国内外发展现状及趋势	2
1.3.1 遥感影像样本数据集构建方法	2
1.3.2 深度学习框架与模型	6
1.3.3 顾及遥感数据及任务特性的深度神经网络优化理论与方法	10
1.3.4 遥感影像处理云平台	13
2. LuoJiaSET 样本库平台	15
2.1 样本共享服务平台基础功能	15
2.1.1 用户注册登录	15
2.1.2 数据集显示	16
2.1.3 数据集详情查询	18
2.2 样本共享服务平台查询下载流程	20
2.2.1 数据集包含类别查询	20
2.2.2 样本元数据信息查询	22
2.2.3 订单提交	23

2.2.4 管理员订单审核	24
2.2.5 订单文件下载	24
2.3 样本数据库.....	25
2.3.1 数据集数量查询	25
2.3.2 样本数量查询	25
3. LuoJiaNET 框架	27
3.1 安装部署	27
3.2 大幅面处理特性与功能.....	38
3.2.1 遥感大幅面影像四叉树索引功能	38
3.2.2 遥感大幅面影像专用 IR 功能	41
3.2.3 遥感大幅面影像算子分解功能	43
3.3 遥感先验知识并行计算.....	58
3.3.1 遥感先验知识提取	58
3.3.2 遥感先验知识自动并行	76
4. LuoJiaNET 典型应用模型	92
4.1 场景分类	92
4.1.1 任务简介	92
4.1.2 分类网络简介	92
4.1.3 测评结果	94

4.1.4 总结	97
4.2 目标检测	98
4.2.1 任务简介	98
4.2.2 目标检测网络简介	98
4.2.3 测评结果	100
4.2.4 总结	113
4.3 地物分类	113
4.3.1 高光谱地物分类（一）	113
4.3.2 高光谱地物分类（二）	128
4.3.3 高空间分辨率地物分类（一）	137
4.3.4 高空间分辨率地物分类（二）	141
4.4 变化检测	147
4.4.1 任务简介	147
4.4.2 变化检测网络简介	148
4.4.3 测评结果	151
4.5 多视三维	157
4.5.1 双目立体	157
4.5.2 多视密集匹配	163
4.6 知识嵌入	170

4.6.1 任务简介 170

4.6.2 知识嵌入网络简介 170

4.6.3 测评结果 171

4.6.4 总结 178

5. LuoJiaNET 的典型行业应用 179

5.1 上海数慧自然资源大脑解决方案 179

5.2 武汉汉达瑞遥感影像智能化处理平台 179

5.3 航天宏图城市土地变化检测平台 180

5.4 珑和科技地事通智慧农业 SaaS 服务平台 180

6. 进一步发展与展望 182

6.1 LuoJiaSET 发展及展望 182

6.2 LuoJiaNET 发展及展望 183

7. 致谢 184

1 概述

1.1 编写目的

白皮书介绍了遥感影像样本库 LuoJiaSET 及专用深度学习框架 LuoJiaNET 的整体使用方法；同时介绍 LuoJiaNET 基础上的五大类应用模型测试情况，包括遥感场景检索、目标检测、地物分类、变化检测、多视角三维重建等；此外，介绍了当前 LuoJiaSET 与 LuoJiaNET 在典型行业应用方面的成果。

预期读者包括机器学习系统设计开发人员、遥感影像样本库管理平台开发人员、深度学习模型开发人员等。

1.2 项目背景

随着大数据、人工智能等技术的发展，基于深度学习的遥感影像解译与监测技术表现出了一定的优势。但在实际应用中，遥感影像智能处理框架和信息服务能力相对滞后，仍未形成与人脸识别等类似的可广泛实用化的智能系统。无论是公开的遥感影像样本库，还是深度学习框架与模型，也尚不能满足空间稀疏表征与融合处理的需求。

当今人工智能时代，专用的遥感深度学习框架与模型是连通硬件、软件、应用场景的枢纽与关键。由于遥感对地观测与智能处理的战略价值，美国在 2020 年将基于人工智能的遥感技术列为敏感技术并限制出口，对我国形成新一轮封锁态势。鉴于目前我国尚无顾及遥感特性的深度学习框架，研究面向遥感应用、具备自主知识产权的专用深度学习框架模型技术，占领遥感人工智能生态链的制高点已显得尤为紧迫。

武汉大学面向国家科技重大需求，在国家自然科学基金重大研究计划项目

“大规模遥感影像样本库构建及开源遥感深度网络框架模型研究”（项目批准号：92038301）支持下，与华为昇腾 AI 团队合作，历时一年半，围绕空间信息稀疏表征与融合处理新理论、新技术与新成果综合集成和演示验证的实际效果评估需求：（1）提出了遥感影像数据集分类标准和标注规范，研发了互联网协同样本标注系统，构建了多种类、百万规模的标注数据和标准数据集及其发布平台 - LuoJiaSET。该样本库有不少于 500 万公开的、可扩展与精化遥感影像样本，包含第一个大规模细粒度地表覆盖样本集和大范围高光谱航空影像样本集；（2）针对遥感影像特点和应用需求，研发了遥感影像处理的深度神经网络开源架构、模型与网络优化方法，形成尺度通道灵活创建、数据通道自适应优选、多层级联合优化的遥感深度学习框架- LuoJiaNET。它是国际上首个针对遥感信息机器学习的专用框架，有效地解决了遥感影像大幅面、多通道数据的特征提取等问题。（3）提出了遥感数据及任务特性的深度神经网络优化方法，包括高效的网络结构自搜索、地学知识嵌入等。LuoJiaNET 不仅可以支持通用 GPU 的大幅面处理，还可以支持全国产昇腾 NPU 硬件，使遥感能智能信息提取的底座“全栈”自主可控。

1.3 国内外发展现状及趋势

1.3.1 遥感影像样本数据集构建方法

近年来，随着深度学习的应用，出现了遥感任务的样本数据集，包括场景分类样本集（如表 1 中的 UC-Merced、NWPU-RESISC45、AID 等）、地物目标检测（飞机、轮船等）样本集（如表 2 中的 DOTA、DIOR、xView 等）、单要素/多要素语义分割样本集（如表 3 中的 GID、Massachusetts）、变化检测样本集

(如表 4 中的 SCDN、SZTAKI) 等。从表 1-4 可看出, 遥感影像样本数量少, 地物要素分类和变化检测样本集严重不足, 且样本集**分类体系各异、采集方法不同、样本尺寸不一、影像分辨率多样**。遥感影像受传感器、季节变化等因素影响, 呈现“同物异谱、异物同谱”现象。现有大部分样本集并未提供成像时间、区域范围等信息。且大都是全色或 RGB 彩色图像, 缺少高光谱、红外、SAR 影像样本, 使得在解译与监测方法忽略了其它类型影像特性。同时, 缺少多视三维重建样本, 导致其未充分顾及多视角成像方式。

表 1. 场景分类常用公开遥感影像数据集

样本数据集	UC-Merced	RSD46-WHU	NWPU-RESISC45	AID	EuroSAT	PatternNet
发布时间	2010年	2017年	2017年	2017年	2017年	2017年
样本数量(张)	2,100	117000	31500	10000	27000	30400
类别数量	21	46	45	30	10	38
样本尺寸	256	256	256	600	64	256
样本分辨率(米)	0.3	0.5~2	0.2~30	0.5~8	10	0.062~4.693
影像来源	USGS National Map Urban Area Imagery影像库	Google Earth和天 地图	Google Earth	Google Earth	Sentinel-2	Google Earth

表 2. 目标检测常用公开遥感影像数据集

样本数据集	DOTA (v2.0)	DIOR	xView	UCAS-AOD	RSOD	VEDAI
发布时间	2020年	2019年	2018年	2015年	2017年	2015年
样本数量(张)	11067	23463	1413	910	976	1210
实例数量(个)	1488666	192472	1000000	6029	6950	3640
样本尺寸	800-20000	800	~3,000	1280	~1000	1024
实例标注方式	旋转边界框	水平边界框	水平边界框	水平边界框	水平边界框	旋转边界框
类别数量	18类	20类	60类	2类 (飞机和车辆) 4类 (飞机、操场、立交桥、油桶)		9类
影像来源	Google Earth、GF-2 和JL-1	Google Earth	WorldView-3	Google Earth	Google Earth和天地 图	Utah AGRC数据集
样本分辨率(米)	主要包括0.12、 0.26、0.5和1.0米等	0.5-30	0.3	未知	0.3-3	0.125

表 3. 常用的遥感影像语义分割公开数据集

样本数据集	GID	Zurich Summer	ISPRS-Vaihingen	ISPRS-Potsdam	Massachusetts Buildings	Indian Pine
发布时间	2020年	2015年	2012年	2012年	2013年	2015年
类别数量	15	8	6	6	2 (建筑物和其它)	16
样本数量 (张)	150	20	33	38	151	1
样本尺寸	6,800×7,200	1,000×1,150	2,500×2,500	6,000×6,000	1,500×1,500	145×145
分辨率 (米)	0.8~10	0.61	0.09	0.05	1	20
波段	RGB, NIR	RGB, NIR	IR,R,G,DSM,nDSM	IR,RGB,DSM,nDSM	RGB	224个波段
影像来源	GF-2	QuickBird	/	/	/	AVIRIS

表 4. 常用的遥感影像变化检测公开数据集

样本数据集	SZTAKI AirChange	Kunshan Data	Season-varing	ABCD	Mts-WH	SCDN
发布时间	2009年	2014年	2018年	2018年	2019年	2020年
类别数量	2	3	2	2	9	30
样本数量 (对)	13	1	16000	4253	1	4214
样本尺寸	952*640	800*800	256*256	160*160	7200*6000	512*512
分辨率 (米)	1.5	30	0.03~0.1	0.4	1	0.5~3
波段	RGB	6个波段	RGB	RGB	RGB,NIR	RGB

在分类体系方面，构建遥感影像样本库，首先要解决分类体系问题。然而，由于遥感任务的复杂性，不同样本集分类体系不尽相同。以土地覆盖分类为例，联合国粮农组织 (FAO)、美国地质调查局 (USGS)、欧洲环境署均提出各自的土地覆盖分类体系，其中 FAO 分类体系 (LCCS) 形成了国际标准 ISO 19144，中国据此制定了国家标准《地理信息分类系统》(GB/T 30322)。此外，还针对土地利用分类制定了《土地利用现状分类》(GB/T 21010-2017)，针对地理国情监测制定了《基础性地理国情监测内容与指标》(CH/T 9029-2019) 行业标准。但不同分类体系间在分类命名、类别层级、类别语义以及兼容性方面有较大差异，没有一种体系能完全满足所有需求，如土地覆盖分类体系适于地物要素分类样本，但缺少目标检测样本的类别描述 (如“飞机”、“车辆”等)。此外，再大量的样本库，都难做到分类完备，囊括所有的特征类别。不可预见的类别常会出现开集

问题（预测类别在样本库所含类别之外），导致解译体系无法灵活扩展，难以服务于大范围应用。

在**样本标注与构建**方面，遥感影像标注的专业性强，需具备地学知识和专业软件操作技术。首先，从**样本标注工具**角度，通用的标注工具（如 LabelMe、LabelImg）可用于小尺寸全色、RGB 影像的目标检测或语义分割标注，但大尺寸的遥感影像需切割成小尺寸进行标注，且不支持高光谱样本制作；同时，通用标注工具无法实现特定任务的标注，如遥感变化检测、全要素地物分类等；此外，地理信息软件（如 ArcGIS 和 QGIS）虽支持多光谱影像标注，但导出标签一般为矢量，输入模型前需转换格式。对于像素级标注，通常用专业软件（如 ENVI、ERDAS）人工采集，采用面向对象方法分割形成地物要素分类样本。其次，从**样本标注方法**角度，分人工与半自动标注。人工标注包括专业人员和众包标注方式，前者质量高但效率低，后者效率高但质量参差不齐。大规模普通影像样本（如 ImageNet）制作将两者结合，即众包标注后再经专业人员审核。在遥感领域，一方面样本标注需专业基础，另一方面众包标注工具缺乏，样本制作限于专业人员，并未发挥众包优势，且**缺乏样本版权保护**。因此亟需半自动标注方法，通过已训练模型（如目标检测模型）初步采集，再用标注工具完善版权等信息。此外，部分地物提取软件（如 EasyFeature 等）也可用于半自动样本制作，但缺乏样本采集的众包平台，导致成本高、效率低。

在**样本的地理空间采样策略**方面，合理的采样策略对构建大范围（区域及全球）遥感影像样本库至关重要，但目前已有样本集（特别是地物要素分类样本）是基于局部区域的少量影像构建，导致类别覆盖度不高、样本分布不均（包括时空分布不均和类内样本数量不均）。地理空间中某一类别要素的空间分布可能是

不均匀的，例如东南亚水系丰富，而中亚草原荒漠较多。均匀抽样会导致各要素的样本不均衡，使机器学习模型对少数类别错误分类。为此需研究顾及地貌景观类别的样本分布策略，以优化样本分布，保证稀有类别样本数量足够，每个地理空间区域有足够的样本量。

在**样本组织管理**方面，多数遥感影像样本集由遥感社区的研究人员制作，或遥感组织根据竞赛任务制作，其样本数量有限。通常以数据文件方式组织，存储在个人服务器、公共资源托管（如 GitHub 等）和云存储平台（如华为云、Google Drive 等），并提供链接供下载使用。虽然大部分数据集提供了描述信息，且部分综述文章介绍了各类样本集，但遥感领域仍没有统一平台，有效管理不同样本集并提供快速索引方式。此外，大多数遥感任务使用单一数据集训练模型，造成泛化性不佳等问题。以目标检测为例，不同样本集均有“飞机”目标，模型若能利用不同成像质量、季节和区域的样本训练，则会强化泛化能力。因此，迫切需要搭建开放平台，存储和管理多类型、多尺度、多级别的遥感影像样本库。

综上所述，当前遥感影像样本存在分类体系不统一、样本数量不足、时空分布不均、标注效率高等问题，不能满足多类型、多级别、多尺度的智能解译与监测需求，且公开样本集缺乏有效动态扩充机制和管理方式，不利于样本集综合开发利用和动态维护。因此，需建立统一的遥感影像样本分类体系，研发支持众包协同的样本标注工具，建立完善的版权保护体系，形成分类统一、类型完备、具有自适应扩展能力的遥感影像样本库，为遥感影像解译与监测提供数据基础。

1.3.2 深度学习框架与模型

自 2012 年 ImageNet 挑战赛以来，面向普通图像处理的深度神经网络框架和模型获得迅猛发展。在遥感领域，影像处理模型主要由普通影像预训练模型

迁移获得，并未从框架与模型集成遥感影像解译与动态监测所需特性。

目前开源深度学习框架种类繁多，项目组前期调研的国内外开源深度学习框架如图 1-1。国内中科院计算所推出了人脸识别深度学习框架 Dragon，清华大学发布了计图（Jittor），华为、旷世、一流科技等互联网企业相继开源了 MindSpore、MegEngine 以及 OneFlow 等框架。在国外，早期蒙特利尔理工学院开源了 Theano、伯克利大学研发了 Caffe，日本首选网络研发了当时领先的 Chainer 框架，Google、Facebook、Amazon 等先后开源了 Tensorflow、PyTorch、MxNet 等框架。虽然通用深度学习框架数目众多，但构建框架的核心技术正呈收敛态势。主要包括：控制流与数据流、以及操作符与张量；计算图优化与自动梯度计算；执行引擎、编程接口、部署运维及分布式训练等。

图 1-1 国内外主要开源深度学习框架对比

框架	时间	企业机构	支持数据类型	框架设计模式	移动端部署	亮点	多类型大幅遥感数据框架协同处理	遥感地学特性支持
Caffe1	2014	伯克利大学	图像	静态	√	张量与运算符绑定	×	×
Chainer	2015	首选网络	图像、文本	动态、分布式	×	元算子、即时定义与运算	×	×
MXNet	2015	亚马逊	图像、文本	静动态、分布式	√	轻量化、多编程语言交互	×	×
Tensorflow	2015	谷歌	图像、文本、语音	静动态、分布式	√	资源丰富、部署灵活	×	×
Caffe2/Pytorch	2016	脸书	图像、文本、语音	动态、分布式	√	使用简单、易于扩展	×	×
CNTK	2016	微软	图像、文本、语音	静态、分布式	×	语音领域性能高	×	×
PaddlePad dle	2016	百度	图像、文本、语音	静动态、分布式	√	预训练中文模型	×	×
MegEngine	2020	旷视	图像	静动态、分布式	√	物联网和视觉任务优化	×	×
MindSpore	2020	华为	图像、文本	静动态、分布式	√	框架与自研芯片的结合	×	×
Jittor	2020	清华大学	图像	统一计算图	×	元算子、统一计算图	×	×
OneFlow	2020	一流科技	图像、文本、语音	静动态、分布式	√	高效的分布式性能	×	×

在控制流与数据流方面，神经网络数据依赖关系表示为有向无环图，该图设计了表达式的求值先后关系，可并行执行。函数式编程能挖掘表达式间的数据依赖关系。随着并发处理需求增多，函数式编程的深度学习框架正占据主流。以 Tensorflow、PyTorch、MXNet 为代表的框架开始侧重计算图的函数

式求解方式，对完整模型一次性求解。在操作符与张量表达方面，传统深度学习框架，例如 Caffe 使用层（Layer）这种粗粒度结构的双向执行逻辑，在前向传播时，程序执行从零开始的递增循环；在反向传播时，程序逆向做递减循环。Tensorflow 将有向无环图的两个基本元素：操作符和张量分开表示。这种细粒度表示更加符合有向图计算思想，开发的灵活性更强。由于细粒度代码对编译器要求较高，多数框架也支持较粗粒度的操作符，例如卷积、池化、矩阵乘操作符等。因此，粗细结合的灵活算子是深度学习框架的发展趋势。此外，对张量计算的支持，也可通过 C++ 模板元编程提高效率。例如 TensorFlow 等框架使用 Eigen 库，MXNet 采用自研的 Mshadow 库。

在计算图优化方面，定义网络结构的有向无环图后，深度学习框架利用编译器技术对图优化重写。计算图优化包括编译器优化、无用代码与公共子表达式消除、操作符融合、类型/形状推导及内存优化等。这些优化方法在 Tensorflow、Pytorch、MXNet 等框架均有体现。在自动梯度计算方面，深度学习框架有两种构建方式，一种是静态图，例如 Caffe 和 Tensorflow，另一种是 Chainer 和 Pytorch 框架推出的动态图。静态图计算效率高、易优化，但灵活性、易用性不如动态图。无论基于静态图还是动态图的框架，自动逆拓扑序推导链式法则的反向传播计算图已成标配。用户只需描述前向传播，反向传播由框架推导完成。

在执行引擎、编程接口、分布式训练与迁移部署方面，Tensorflow/Pytorch 等通过协调 CPU 和 GPU 设备提高计算效率与资源利用率。框架底层基于 C++ 开发，同时提供 Python 等前端接口。从开发到部署遵从“离线训练、在线识别”原则。模型训练依赖分布式平台，例如 Hadoop 或

Spark 支撑，使用数据并行的策略扩大处理任务和规模。然而设备数量不断增加，会导致通信开销增长，出现模型效率损失等缺陷。因此，将计算矩阵分块的模型并行策略，及 GPU 接力训练的流水线并行策略受到了重视。目前，主流开源框架都能支持数据并行，但模型并行和流水线并行仍较困难。此外，在框架部署与运维方面，普遍使用 Docker 和 Kuberntess 结合，Tensorflow 也推出托管工具 Serving，便于线上部署与运维。同时，随着开放神经网络交换标准 (ONNX) 的推出，多框架切换也得以满足，Tensorflow、PyTorch 等框架都支持 ONNX 标准。

在基础模型设计方面，蒙特利尔大学率先在 GPU 上实现 AlexNet，并在 ImageNet 取得超过第二名 10% 的精度；牛津大学设计了 VGG-Net 使 top5 错误率降至 7.5%；谷歌推出的 GoogleNet，使识别精度大幅度提升；何恺明等人提出 ResNet，使 top5 错误率降低至 4.5%；康奈尔大学、清华大学与 Facebook FAIR 实验推出 DenseNet，使相同精度下计算量大幅降低；中国科学技术大学与 MSRA 开源了 HRNet，刷新了 MS-COCO 数据集三项纪录。在模型结构自动搜索 (NAS) 方面，数据驱动的搜索主要包含基于强化学习、演化计算与基于梯度的方法。基于强化学习方法通过代理模型指导网络搜索方向，常用代理模型有循环神经网络与近端策略优化等。基于演化计算方法将模型体系结构编码为字符序列，根据验证集的性能评估，执行交叉和变异操作，生成新的高性能结构。其中网络架构参数与模型参数分别基于演化与随机梯度下降方法迭代更新。基于强化学习和演化计算方法法具有出色性能，但对计算的要求很高。基于梯度的方法能显著提升效率，利用概率平滑使搜索空间可微，因而能够端对端优化。这些模型及搜索方法是针对小尺寸影像训练或搜索

的模型，未顾及大幅遥感影像“像素-目标-场景”多层次要素提取、变化发现等任务的特性。

通过对国内外开源深度学习框架和模型可以发现，为平衡计算性能和灵活性需求，主流深度学习框架都有其优缺点。然而，主流开源深度学习框架与模型主要针对的是普通影像，即普通的小像幅室内/外影像；现有遥感能解译与监测系统大多由通用图像识别的深度神经网络改造，一般只考虑了影像二维空间的可见光图像特征，顾及遥感光谱特性、地学先验知识、数据与框架协同等重要因素，支持遥感地学特性的框架仍是空白。

1.3.3 顾及遥感数据及任务特性的深度神经网络优化理论与方法

1.3.3.1 顾及遥感数据及遥感任务的高效网络结构自搜索优化理论

网络结构搜索方法最先在计算机视觉领域被提出。其流程为：定义搜索空间，通过搜索策略找出候选网络结构，对网络结构进行评估，根据反馈进行下一轮搜索。Baker 等人和 Zoph 等人于 2016 年分别提出基于强化学习的网络结构搜索方法，开拓了网络搜索方法的先河。该类方法在图像分类任务上基本可以击败同量级的网络，但需要花费大量的 GPU 计算资源和训练时间。此后，Real 等人引入进化算法解决网络结构搜索问题。相较于强化学习算法，进化算法搜索得更快，且能得到更小的模型。2018 年，Liu 等人提出了基于梯度的方法，将网络候选操作使用 *softmax* 函数进行混合，把搜索空间变成连续空间、目标函数成为可微函数，从而可以使用基于梯度的优化方法找寻最优结构，大大减少了计算资源消耗。此后，在加速手段方面，有层次化表示、权重共享、变现预测等。网络结构搜索在语义分割、模型压缩、数据增强等任务上都有应用。然而，现有的网络结

构搜索方法往往仅针对某一种任务类型，无法满足遥感目标识别的多任务需求。

在遥感领域，网络结构搜索方法的研究尚处于起步阶段。国内学者们目前刚开始探索该方法，尚未应用到遥感领域；国外有少量网络结构搜索方法在遥感领域的研究。具体地，Bahri 等人、Wei 等人在遥感影像场景分类问题上尝试了网络结构搜索方法；Dong 等人使用网络结构搜索解决极化合成孔径雷达影像分类问题；Zhang 等人将网络结构搜索应用于遥感影像语义分割。特别地，Wang 等人提出了一种任务驱动的网络结构搜索框架，支持自适应地处理场景分类加地物分类两种任务，该方法在面向多种任务的方向上做出了一定尝试，但其任务种类仍不足以满足遥感目标识别任务的多样性需求。亟需研究满足目标检索与场景分类、目标检测、地物要素分类、变化检测、多视三维重建等多种遥感解译与监测的网络结构搜索方法。

1.3.3.2 地学知识嵌入的遥感深度学习优化理论与方法

深度学习方法通过降低输出结果与标签的损失来优化网络参数。相比之下，人脑作为面向知识的高级智能系统，可以通过知识推理做出可靠性高、可解释性强的决策。因此，亟需研究地学先验知识的表达及嵌入，主要包括以下三个方面：

(1) 地学本体建模与地学知识图谱创建

作为对特定领域中概念及其相互关系的形式化表达，本体(Ontology)具有强大的表示、推理和共享知识的能力，因而被引入到地学领域以构建地学本体模型。知识图谱则由本体实例化而来，是由图数据结构表示的知识载体，描述了客观世界的事物与关系。知识图谱最早被用于介绍语义网搜索技术，随后出现了

DBpedia, YAGO 等大型通用知识图谱。知识图谱的创建分为自顶向下和自底向上的方法，具备图结构组织、抽象的概念与关系、实体间的关系连接等特征。目前地学领域尚无大规模可用的知识图谱，亟需开展面向遥感能智能解译和知识服务的地学本体建模和地学知识图谱创建工作。

(2) 基于地学知识图谱表示学习的遥感影像场景分类

基于地学知识图谱表示学习的遥感影像场景分类可将遥感影像包含的丰富语义信息融入学习过程，进而提高分类准确率。受语义空间中词向量的平移不变性特征的启发，Bordes 等人提出了 TransE 模型，TransE 模型训练参数少，易于计算而且能有效地对知识图谱包含的实体和关系进行表示，但缺乏对复杂关系的处理能力。在表示学习方法方面，Fan 等人提出了 DKRL 模型在利用实体的描述信息上进行了尝试。目前，针对地学领域知识图谱进行表示学习的探索仍处于萌芽期，亟需研究适用于地学领域的知识图谱构建和表示学习方法。

(3) 耦合地学知识图谱和深度语义分割网络的遥感影像语义分割

地学知识图谱基于知识建模、推理和共享，能够建立模拟人类感知过程的知识模型来实现遥感影像的智能解译。以地理对象图像分析方法（GEOBIA）为代表，基于地学知识图谱的遥感影像解译方法相继出现。上述方法与深度学习方法相比，分类精度较差。Alirezaie 等人提出了一种地学知识图谱推理协助深度语义分割网络的方法，增强可解释性的同时提高了分类精度。综上所述，结合知识驱动和数据驱动的优势，构建耦合地学知识图谱与深度学习的方法是实现遥感影像解译的必由之路。

1.3.4 遥感影像处理云平台

面向遥感大数据的云计算技术平台方面，诞生了美国谷歌地球引擎 GEE (Google Earth Engine)、美国航空航天局的 NEX (NASA Earth Exchange)、笛卡尔实验室的 Geoprocessing Platform、澳大利亚的地球科学数据立方体 (Geoscience Data Cube)、微软的行星计算机 (Planetary Computer)，以及中国科学院“地球大数据科学工程” (CASEarth) 的“地球大数据挖掘分析系统 (EarthDataMiner) 云服务”、航天宏图所开发 PIE-Engine、武汉大学与华为昇腾 AI 联合研发的 LuoJiaNET 等 (如图 1.3)。

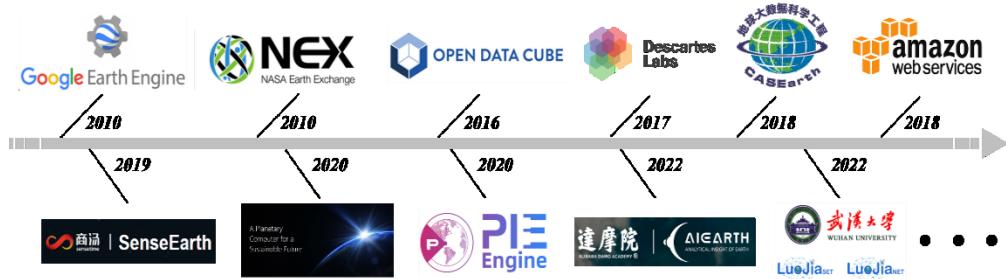


图 1.3 遥感云计算平台

遗憾的是，虽然现有遥感云计算平台已取得了较大进展，但在实际进行智能化分析和信息挖掘时，由于大规模遥感影像样本库的缺乏、深度学习智能处理框架相对滞后，未形成从底层硬件到样本框架、再到面向智能测图任务的上层应用模型的全链条自主可控生态。截止目前，现有平台仍存在以下局限：

(1) **底层 AI 硬件主要依赖国外进口。**现有通用人工智能硬件，主要依赖于美国英伟达 (NVIDIA) 公司生产的图形处理单元(GPU)。俄乌冲突期间，英伟达公司联合 AMD、Intel 公司对俄罗斯全面禁售 GPU 产品。此外，深度学习所依赖智能计算加速库 cudnn，不对外开源，卷积 conv 等算子也有显存容量限制，直接制约了大幅面遥感影像智能处理。武汉大学与华为技术有限公司联合研发了

遥感处理框架 LuoJiaNET，从底层硬件解决了上述问题，但在全链条的可控性方面，仍有待进一步增强。

(2) **遥感影像样本库仍需增强可控性。** 2019 年底，美国宇航局 (NASA) 和盖茨基金会等联合资助的 Radiant Earth Foundation 推出 Radiant ML Hub 平台，方便研究人员访问其发布的地球观测训练数据集。2020 年欧空局 (ESA) 立项了 AIREO 项目，旨在制定社区规范和最佳实践，以可发现、可访问、互操作、可重用的方式共享对地观测训练数据集。与 NASA、ESA 相比，我国现有遥感影像样本库分类不完备、传感器种类单一、时空分布零散、规模和扩展性有限，造成深度学习模型泛化能力不足，无法支撑大时空跨度海量遥感影像精准解译。

(3) **遥感深度学习框架不能自主可控。** 现有遥感深度学习处理主要依赖于国外的框架，例如 PyTorch、Tensorflow 等。一般只考虑了通用的图像处理模式，不能完全适应于多类型、多尺度、多级别的遥感影像测图处理任务。此外，PyTorch 框架转向支持自研 AI ASIC 硬件；Tensorflow 转向 JAX 框架，底层支持谷歌 TPU 硬件，致使深度学习框架可控性差，且无法满足遥感测图任务。

(4) **遥感解译缺乏自主知识产权模型。** 现有遥感云平台在执行解译任务时，所使用的模型通常由计算机视觉模型改造而来，例如 DeepLab 等地物分类模型。缺乏直接面向遥感测图任务、具备融合“场景-目标-像素”多层次的解译方法，不能直接输出地理信息系统制图综合所需的地物矢量要素。此外，在地物矢量入库方面，缺乏有效的地物要素识别、自动制图综合与人机协同提取机制，致使作业效率低下。

2. LuoJiaSET 样本库平台

2.1 样本共享服务平台基础功能

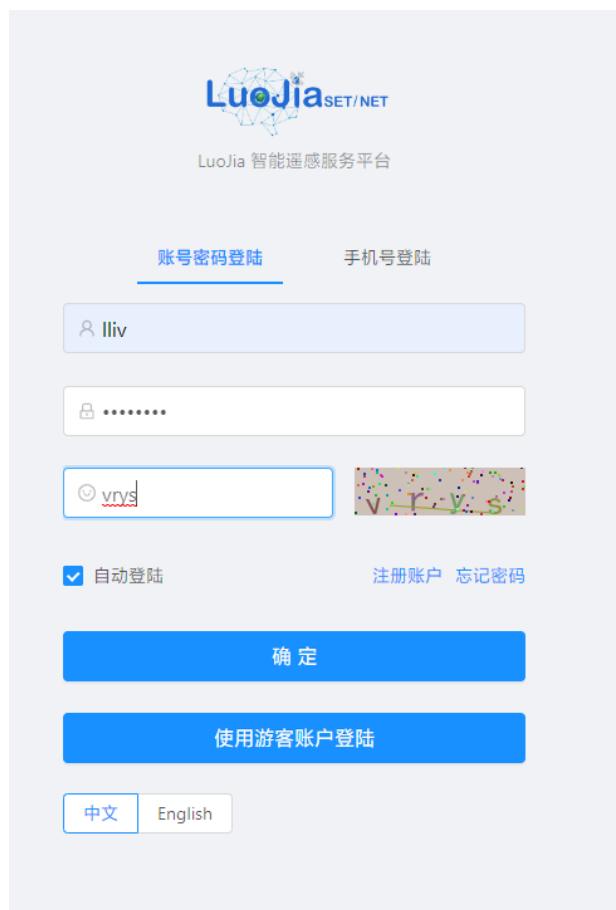
2.1.1 用户注册登录

(1) 平台新用户注册

测试方案：在已迁移的 LuoJiaSET 平台用户注册界面，进行用户注册。

测试结果：可成功注册登录，并在数据库中添加对应的用户字段。

注册用户：lliv

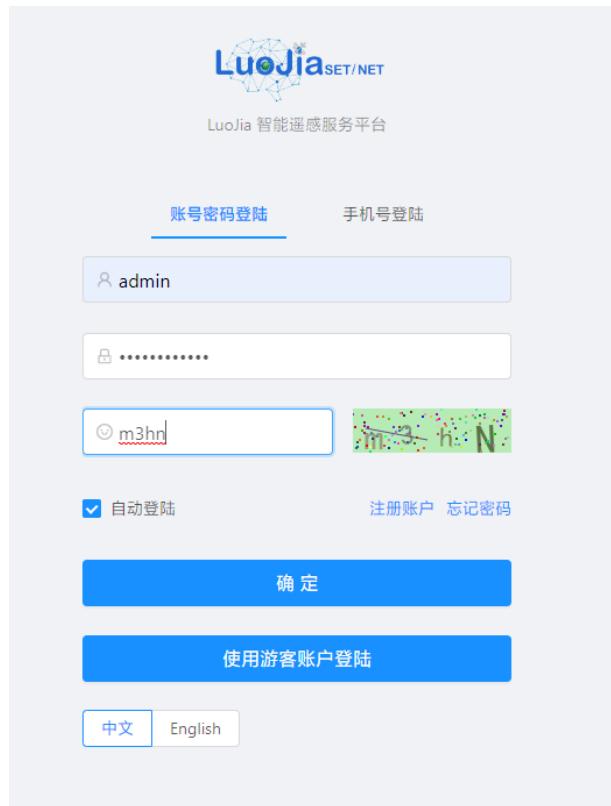


(2) 平台原有用户登录

测试方案：在原有 LuoJiaSET 平台注册的用户，在迁移后的平台登陆，查看是否通过用户数据库验证。

测试结果：经过测试，在原有 LuoJiaSet 平台注册的用户可以通过数据库验证，并成功登录。

测试用户：admin、demo



2.1.2 数据集显示

(1) 数据集总量：83 项

(2) 各任务类型数据集数量：

①场景分类：34项

Sortord : Relevancy Page View Update Time

Totally 34 items

			
RSD46-WHU	AID	PatternNet	RSSCN7
TaskType: Scene Classification	TaskType: Scene Classification	TaskType: Scene Classification	TaskType: Scene Classification
			
WHU-RS19	SIRI-WHU	NWPU-RESISC45	RSI-CB128
TaskType: Scene Classification	TaskType: Scene Classification	TaskType: Scene Classification	TaskType: Scene Classification

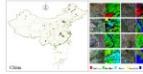
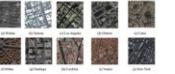
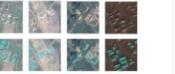
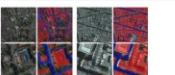
TaskType: Scene Classification

Scene Class...
Use computer vision and machine learning algorithms to extract meaningful tasks from images. This operation can be as simple as assigning a label to an image, or as advanced as interpreting the content of the image and returning a human-readable sentence.

②地物分类：13项

Sortord : Relevancy Page View Update Time

Totally 13 items

			
GID	WHU-Building-AID	WHU-Building-I	WHU-Building-II
TaskType: LC/LU Classification	TaskType: LC/LU Classification	TaskType: LC/LU Classification	TaskType: LC/LU Classification
			
waterExtraction	buildingExtraction	AISD	MBD
TaskType: LC/LU Classification	TaskType: LC/LU Classification	TaskType: LC/LU Classification	TaskType: LC/LU Classification

TaskType: LC/LU Classification

LC/LU Class...
The difference of spectral energy characteristics and structural characteristics of ground features are used to identify ground features.

③目标识别：26项

Sortord : Relevancy Page View Update Time

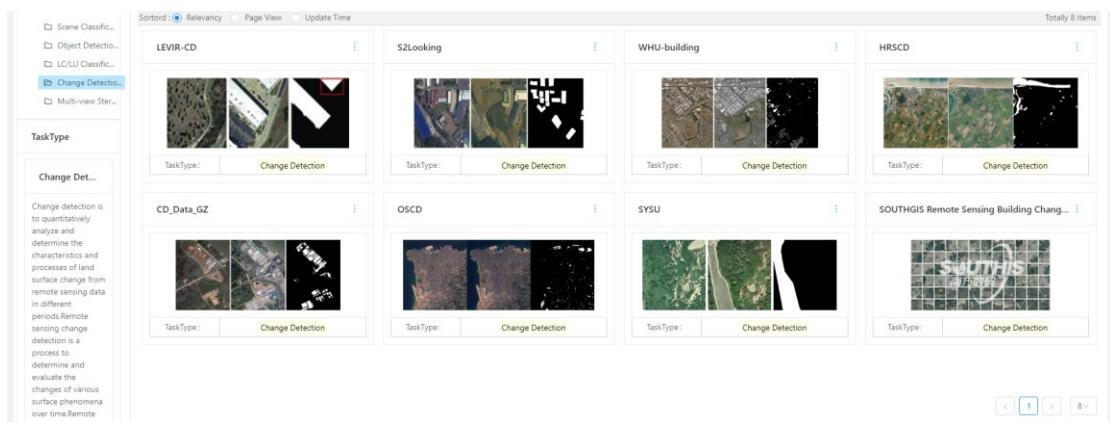
Totally 26 items

			
DOTA	RSOD	SSDD	DOTA
TaskType: Object Detection	TaskType: Object Detection	TaskType: Object Detection	TaskType: Object Detection
			
bridges_dataset	NWPU VHR-10 dataset	TGRS-HRRSD	DIOR
TaskType: Object Detection	TaskType: Object Detection	TaskType: Object Detection	TaskType: Object Detection

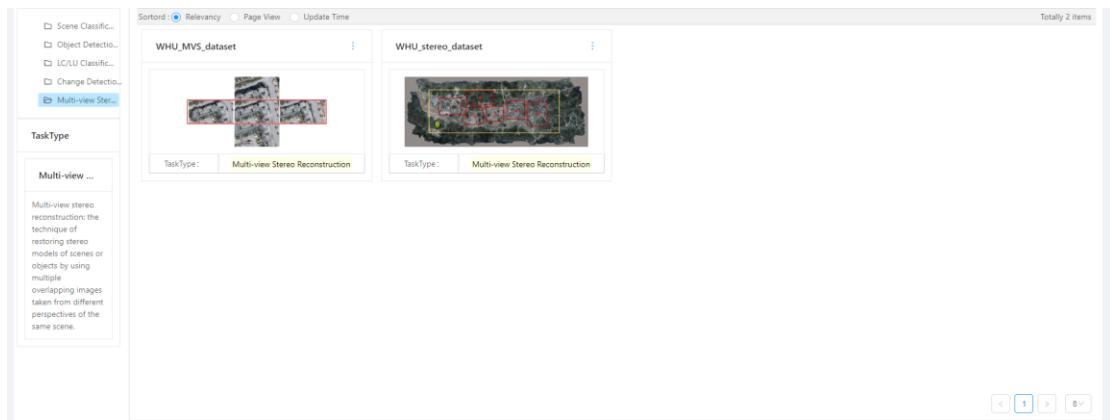
TaskType: Object Detection

Object Detectio...
Object detection refers to the process of detecting a special object (e.g. one type of object) from other objects (or other types of objects). It includes not only the recognition of two very similar objects, but also the recognition of one type of object with other types of objects.

④变化检测：8项



⑤多视三维：2项



2.1.3 数据集详情查询

测试方案：在数据集详情界面，查看各数据集的元数据是否正确显示，并查询各数据集各类别统计信息是否正确显示。

元数据信息包括：

- 1) 数据集文件大小
- 2) 数据集中包含样本数量
- 3) 数据集影像文件尺寸
- 4) 数据集影像文件影像类型
- 5) 数据集影像分辨率

6) 数据集影像波段数

7) 数据集影像文件类型格式

8) 数据集获取传感器

9) 数据集联系人姓名

10) 数据集联系人联系方式

11) 数据集联系人地址

12) 数据集联系人所属工作单位

测试结果：

可顺利对现有全部的数据集元数据信息访问：

DOTA:

The screenshot displays the LueJia dataset management platform interface. The top navigation bar includes links for '首页', '查询服务', '数据集', '订单列表', '资源', '关于我们', '个人页', '我的消息', '中文', 'English', and '欢迎您, Demo'.

The main content area shows the 'DOTA' dataset details. A red box highlights the '基本信息' (Basic Information) section, which contains the following data:

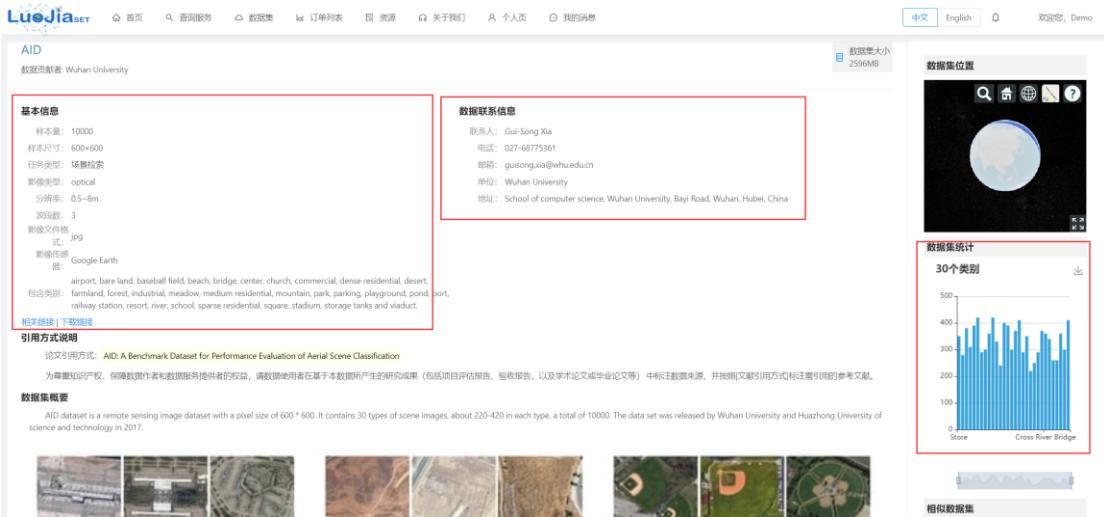
- 样本量: 11268
- 样本尺寸: 800 × 800 ~ 20000 × 20000
- 任务类型: 目标识别
- 影像类型: optical
- 分辨率: 高分
- 源像素: 3
- 影像文件格式: jpg
- 影像传感器: Google Earth.GF-2,aerial
- 鸟瞰类型: Google Earth.GF-2,aerial

A red box also highlights the '数据集联系信息' (Dataset Contact Information) section, which lists:

- 联系人: Gui-Song Xia
- 电话: 027-68775361
- 邮箱: guisong.xia@whu.edu.cn
- 单位: Wuhan University
- 地址: School of computer science, Wuhan University, Bayi Road, Wuhan, Hubei, China

Other sections visible include '数据集位置' (Dataset Location) with a globe icon, '数据集统计' (Dataset Statistics) showing a bar chart for 18 categories (e.g., Bridge, Basketball_Court), and '相似数据集' (Similar Datasets) with a search input field.

AID:



查询条件

* 任务类型
目标识别

数据集名称
DOTA_v1.5

影像类型
请选择样本影像类型

类别
请选择一个样本类别

- 飞机
- 小船
- 储存罐
- 棒球场
- 网球场
- 游泳池
- 田径场
- 港口

查询条件

* 任务类型
目标识别

数据集名称
DOTA_v1.5

影像类型
请选择样本影像类型

类别
请选择一个样本类别

- 桥
- 小型车辆
- 大型车辆
- 直升机
- 环形交叉路口
- 足球场
- 篮球场
- 集装箱起重机

其中可获取各个类别的详细数量：



类别名称	类别英文名称	数量
集装箱起重机	Container_Crane	156
港口	Harbor	8118
桥	Bridge	2541

飞机	Plane	10622
直升机	Helicopter	713
储存罐	Storage_Tank	8286
游泳池	Swimming_Pool	2757
网球场	Tennis_Court	3188
篮球场	Basketball_Court	672
环形交叉路口	Roundabout	622
田径场	Ground_Track_Field	476
大型车辆	Large_Vehicle	27357
棒球场	Baseball_Diamond	625
小型车辆	Small_Vehicle	169838
足球场	Soccerball_Field	487
小船	Ship	43738

2.2.2 样本元数据信息查询

在样本查询结果界面，点击某一条返回结果，查看样本详情弹出框中是否包含样本缩略图，以及样本 ID、类别、传感器、数据集、样本尺寸、分辨率、版本、空间范围等字段的元数据信息是否正确显示。如图所示：

Sample Detail



Sample Detail	
Sample ID	1
Classes	Small_Vehicle, Large_Vehicle, Ro undabout
Instrument	Google Earth
Datasets	DOTA
Sample size	1818 x 1162
Image Resolution	null
sample version	1.0
Min Lat/Lon	None
Max Lat/Lon	None

Sample Detail

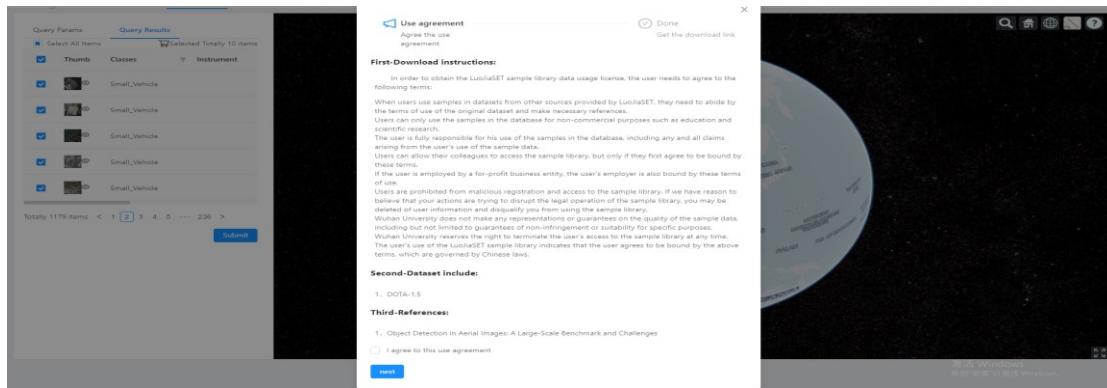


Sample Detail	
Sample ID	160178
Classes	Small_Vehicle, Truck, Hut/Tent, Shed, Building, Damaged_Buildin g, Construction_Site
Instrument	WorldView 3
Datasets	DIUx_xView
Sample size	3228 x 3320
Image Resolution	0.3m
sample version	1.0
Min Lat/Lon	[-23.48, 14.947]
Max Lat/Lon	[-23.47, 14.956]

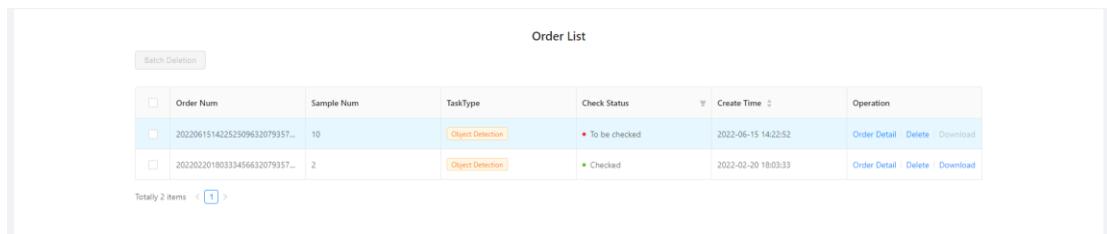
2.2.3 订单提交

以非游客身份登录共享服务平台，完成查询测试后，在样本查询结果界面，点击结果前的勾选框，选择一组样本后，点击提交查询，查看是否弹出使用协议对话框，且对话框中是否正确包含勾选的样本所在数据集信息及引用信息；完成上述操作后，点击订单列表菜单栏，查看是否生成了对应的样本下载订单。在迁移后的平台通过选择样本属性条件筛选，查询对应的样本集合：提交样本订单，

管理员查询并通过订单审核。通过样本查询，可顺利提交订单，并在个人订单列表中显示订单详情信息。

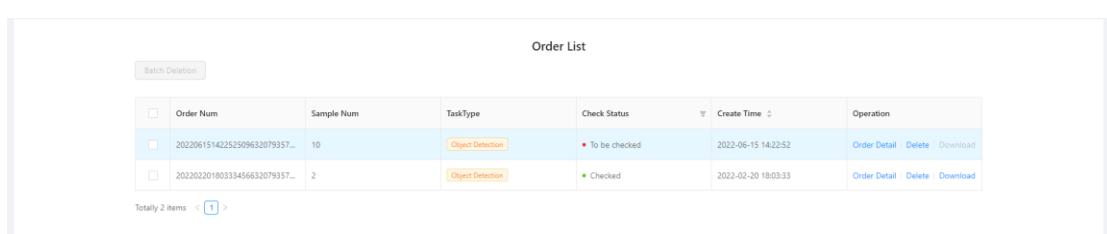


个人订单查看页面：



2.2.4 管理员订单审核

管理员登陆订单审核界面，通过点击审核按钮，通过订单审核，后台下载订单中的样本，并压缩上传至 OBS 服务器，并发送邮件为用户提供数据信息。如图所示：



2.2.5 订单文件下载

测试用户登录个人订单查看页面，点击下载按钮，可完成样本数据下载。结

果如下图：

2.3 样本数据库

2.3.1 数据集数量查询

统计 OBS 中各个任务类型的数据集数量，并与数据库中记录的数据集数量信息统计比较：

现迁移后的平台共包含数据集 83 个，其中：

任务类型	OBS 数据集数量	数据库数据集数量
场景识别	34	34
目标识别	26	26
变化检测	8	8
地物分类	13	13
多视三维	2	2

2.3.2 样本数量查询

在迁移后的平台样本查询界面对各个任务类型和各个类别的样本数量进行查询，核对查询结果是否与原有数据相对应。结果如下表：

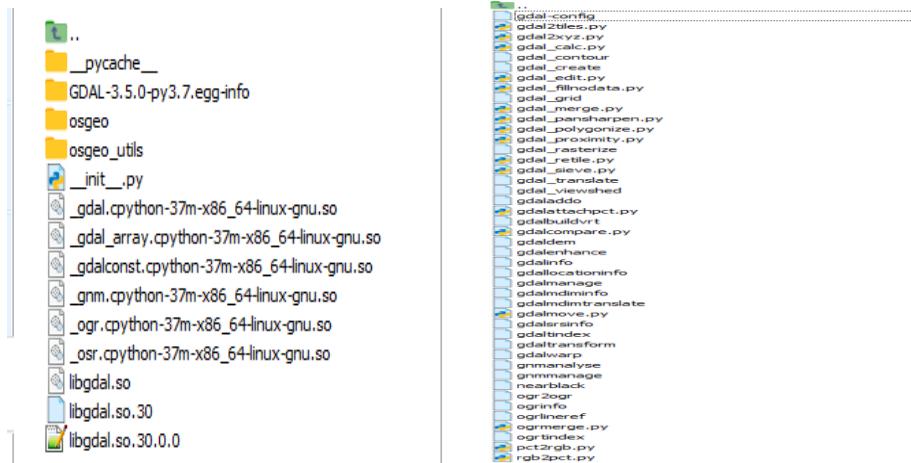
任务类型	样本影像数量
场景分类	2184116
目标检测	232721
地物分类	1915179
变化检测	41479
多视三维	16614

3. LuoJiaNET 框架

3.1 安装部署

(1) 多源遥感影像 I/O 库编译

执行命令: bash build_gdal.sh 编译获得如下 GDAL 文件:



在 GDAL_Linux/bin 或 GDAL_Win/bin 执行命令 ./gdalinfo --formats，即可查询多源 I/O 所支持的格式。经测试，所有第三方库（参见：<https://gitee.com/mizhangwhuer/projects>）均从源码编译，保证源码可控。可支持全色、多光谱、高光谱、SAR 等主要遥感影像类型的 I/O 操作（见 2.3.1 节调用情况，python 端

测 试 用 例 在
https://gitee.com/mindspore/luojianet/blob/master/tests/ut/python/dataset/test_index.py ; C++ 端 测 试 用 例 在
<https://gitee.com/mindspore/luojianet/tree/master/tests/ut/cpp/dataset>)

```

Supported Formats:
VRT -raster,multidimensional raster- (rw+v): Virtual Raster
DERIVED -raster- (ro): Derived datasets using VRT pixel functions
GTiff -raster- (rw+vs): GeoTIFF
COG -raster- (wv): Cloud optimized GeoTIFF generator
NITF -raster- (rw+vs): National Imagery Transmission Format
RPFTOC -raster- (rov): Raster Product Format TOC format
ECRGTOC -raster- (rov): ECRG TOC format
HFA -raster- (rw+v): Erdas Imagine Images (.img)
SAR_CEOS -raster- (rov): CEOS SAR Image
CEOS -raster- (rov): CEOS Image
JAXAPALSAR -raster- (rov): JAXA PALSAR Product Reader (Level 1.1/1.5)
GFF -raster- (rov): Ground-based SAR Applications Testbed File Format (.gff)
ELAS -raster- (rw+v): ELAS
ESRIC -raster- (rov): Esri Compact Cache
AIG -raster- (rov): Arc/Info Binary Grid
AAIGrid -raster- (rwv): Arc/Info ASCII Grid
GRASSASCIIGrid -raster- (rov): GRASS ASCII Grid
ISG -raster- (rov): International Service for the Geoid
SDTS -raster- (rov): SDTS Raster
DTED -raster- (rwv): DTED Elevation Raster
PNG -raster- (rwv): Portable Network Graphics
JPEG -raster- (rwv): JPEG JFIF
MEM -raster,multidimensional raster- (rw+): In Memory Raster
JDEM -raster- (rov): Japanese DEM (.mem)
GIF -raster- (rwv): Graphics Interchange Format (.gif)
BIGGIF -raster- (rov): Graphics Interchange Format (.gif)
ESAT -raster- (rov): Envisat Image Format
BSB -raster- (rov): Maptech BSB Nautical Charts
XPM -raster- (rwv): X11 PixMap Format
BMP -raster- (rw+v): MS Windows Device Independent Bitmap
DIMAP -raster- (rov): SPOT DIMAP
AirSAR -raster- (rov): AirSAR Polarimetric Image
RS2 -raster- (rovs): RadarSat 2 XML Product
SAFE -raster- (rov): Sentinel-1 SAR SAFE Product
PCIDSK -raster,vector- (rw+v): PCIDSK Database File
PCRaster -raster- (rw+): PCRaster Raster File
ILWIS -raster- (rw+v): ILWIS Raster Map
SGI -raster- (rw+v): SGI Image File Format 1.0
SRTMHT -raster- (rwv): SRTMHT File Format
Leveller -raster- (rw+v): Leveller heightfield
Terragen -raster- (rw+v): Terragen heightfield
ISIS3 -raster- (rw+v): USGS Astrogeology ISIS cube (Version 3)
ISIS2 -raster- (rw+v): USGS Astrogeology ISIS cube (Version 2)
PDS -raster- (rov): NASA Planetary Data System
PDS4 -raster,vector- (rw+v): NASA Planetary Data System 4
VICAR -raster,vector- (rw+v): MIPS VICAR file
TIL -raster- (rov): EarthWatch .TIL
ERS -raster- (rw+v): ERMapper .ers Labelled
LIB -raster- (rovs): NOAA Polar Orbiter Level 1b Data Set
FIT -raster- (rwv): FIT Image
GRIB -raster,multidimensional raster- (rwv): GRIdded Binary (.grb, .grb2)
RMF -raster- (rw+v): Raster Matrix Format
MSGN -raster- (rov): EUMETSAT Archive native (.nat)
RST -raster- (rw+v): Idrisi Raster A.1
GSAG -raster- (rwv): Golden Software ASCII Grid (.grd)
GSBG -raster- (rw+v): Golden Software Binary Grid (.grd)
GS7BG -raster- (rw+v): Golden Software 7 Binary Grid (.grd)
COSAR -raster- (rov): COSAR Annotated Binary Matrix (TerraSAR-X)
TSX -raster- (rov): TerraSAR-X Product
COASP -raster- (ro): DRDC COASP SAR Processor Raster
R -raster- (rwv): R Object Data Store
MAP -raster- (rov): OziExplorer .MAP
KMLSUPEROVERLAY -raster- (rwv): Kml Super Overlay
PDF -raster,vector- (w+): Geospatial PDF

```

(2) CPU/GPU/NPU 多平台编译流程

① CPU 下编译

安装 MinGW，执行命令：build.bat，编译过程如下图：

```
adding 'luojianet_ms/train/mind_ir_pb2.py'
adding 'luojianet_ms/train/model.py'
adding 'luojianet_ms/train/node_strategy_pb2.py'
adding 'luojianet_ms/train/print_pb2.py'
adding 'luojianet_ms/train/profiling_parallel_pb2.py'
adding 'luojianet_ms/train/serialization.py'
adding 'luojianet_ms/train/summary_pb2.py'
adding 'luojianet_ms/train/callback/_init_.py'
adding 'luojianet_ms/train/callback/_callback.py'
adding 'luojianet_ms/train/callback/_checkpoint.py'
adding 'luojianet_ms/train/callback/_dataset_graph.py'
adding 'luojianet_ms/train/callback/_f1_manager.py'
adding 'luojianet_ms/train/callback/_landscape.py'
adding 'luojianet_ms/train/callback/_loss_monitor.py'
adding 'luojianet_ms/train/callback/_lr_scheduler_callback.py'
adding 'luojianet_ms/train/callback/_summary_collector.py'
adding 'luojianet_ms/train/callback/_time_monitor.py'
adding 'luojianet_ms/train/summary/_init_.py'
adding 'luojianet_ms/train/summary/_lineage_adapter.py'
adding 'luojianet_ms/train/summary/_summary_adapter.py'
adding 'luojianet_ms/train/summary/_writer_pool.py'
adding 'luojianet_ms/train/summary/enum.py'
adding 'luojianet_ms/train/summary/summary_record.py'
adding 'luojianet_ms/train/summary/writer.py'
adding 'luojianet_ms/train/train_thor/_init_.py'
adding 'luojianet_ms/train/train_thor/convert_utils.py'
adding 'luojianet_ms/train/train_thor/dataset_helper.py'
adding 'luojianet_ms/train/train_thor/model_thor.py'
adding 'luojianet_ms-1.0.0.dist-info/METADATA'
adding 'luojianet_ms-1.0.0.dist-info/WHEEL'
adding 'luojianet_ms-1.0.0.dist-info/entry_points.txt'
adding 'luojianet_ms-1.0.0.dist-info/top_level.txt'
adding 'luojianet_ms-1.0.0.dist-info/RECORD'
removing build\bdist.win-amd64\wheel
Pack: - package: E:/LuoJiaNETMSV2_gitee/luojianet_new20220613/luojianet/build/luojianet_ms/luojianet_ms generated.
```

验证是否成功： `python -c "import`

`luojianet_ms;luojianet_ms.run_check()"`

输出如下，则说明安装成功：

```
E:\LuoJiaNETMSV2_gitee\luojianet_new20220613\luojianet\output>python -c "import lujianet_ms;luojianet_ms.run_check()"
LuoJiaNet version: 1.0.0
The result of multiplication calculation is correct, LuoJiaNet has been installed successfully!
```

② GPU 下编译

执行命令：`bash build.sh -e gpu -j 6`， 编译过程如下图：

```

[ 61%] Built target luojianet_fl_obj
[ 63%] Built target kernels-image
[ 63%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/runtime/device/gpu/gpu_launch_mul.cc.o
[ 63%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/runtime/device/gpu/gpu_memory_manager.cc.o
[ 63%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/backend/optimizer/graph_kernel/graph_kernel_helper.cc.o
[ 63%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/backend/optimizer/graph_kernel/graph_ops_splitter.cc.o
[ 63%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/backend/optimizer/graph_kernel/graph_kernel_splitter.cc.o
[ 75%] Built target _luojianet_ms_kernel_compiler_obj
[ 75%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/backend/optimizer/graph_kernel/value_graph_binder.cc.o
[ 75%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/backend/optimizer/pass/communication_op_fusion.cc.o
[ 75%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/runtime/device/bucket.cc.o
[ 75%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/runtime/device/kernel_runtime.cc.o
[ 75%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/runtime/device/kernel_runtime_manager.cc.o
[ 75%] Building CXX object luojianet_ms/ccsrc/CMakeFiles/gpu_cuda_lib.dir/runtime/device/launch_mul.cc.o
[ 75%] Linking CXX static library libgpu_cuda.lib.a
[ 91%] Built target gpu_cuda.lib
[ 91%] Built target luojianet_ms
[ 91%] Linking CXX shared module _c_expression.cpython-37m-x86_64-linux-gnu.so
[ 91%] Linking CXX shared library _c_mindrecord.cpython-37m-x86_64-linux-gnu.so
[ 91%] Linking CXX shared library libluojianet_ms.so
[ 92%] Built target _c_mindrecord
[ 92%] Built target luojianet_shared.lib
[ 92%] Linking CXX shared library _c_dataengine.cpython-37m-x86_64-linux-gnu.so
[ 92%] Built target _c_dataengine
[ 92%] Linking CXX executable cache_server
[ 92%] Linking CXX executable cache_admin
[ 92%] Built target cache_admin
[ 92%] Linking CXX executable ut_tests
[ 92%] Built target cache_server
[ 92%] Built target _c_expression
[100%] Built target ut_tests

```

```

adding 'luojianet_ms/train/train_thor/convert_utils.py'
adding 'luojianet_ms/train/train_thor/dataset_helper.py'
adding 'luojianet_ms/train/train_thor/model_thor.py'
adding 'luojianet_ms_gpu-1.0.0.dist-info/METADATA'
adding 'luojianet_ms_gpu-1.0.0.dist-info/WHEEL'
adding 'luojianet_ms_gpu-1.0.0.dist-info/entry_points.txt'
adding 'luojianet_ms_gpu-1.0.0.dist-info/top_level.txt'
adding 'luojianet_ms_gpu-1.0.0.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
/home/flash/anacondas3/lib/python3.7/site-packages/setuptools/command/install.py:37: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build and pip and other standards-based tools.
    setuptools.SetuptoolsDeprecationWarning,
CPack: - package: ./data02/LuoJiaNET_V2/luojianet/build/luojianet_ms/luojianet_ms generated.
success building lujianet_ms project!
----- LuoJiaNet: build end -----

```

验证是否成功： `python -c "import`

`luojianet_ms;luojianet_ms.run_check()"`

输出如下，说明安装成功：

```

(base) flash@node01:/data02/LuoJiaNET_V2/luojianet$ python -c "import lujianet_ms;luojianet_ms.run_check()"
[WARNING] ME(51132:140449739900736,MainProcess):2022-06-09-19:16:26.385.630 [luojianet_ms/run_check/_check_version.py:135] LuoJiaNet version 1.0.0 and cuda version 10.2.89 does not match, please refer to the installation guide for version matching information: https://www.lujianet\_ms.cn/install
[WARNING] ME(51132:140449739900736,MainProcess):2022-06-09-19:16:26.391.977 [luojianet_ms/run_check/_check_version.py:140] LuoJiaNet version 1.0.0 and nvcc(cuda bin) version 10.2 does not match, please refer to the installation guide for version matching information: https://www.lujianet\_ms.cn/install
LuoJiaNet version: 1.0.0
The result of multiplication calculation is correct, LuoJiaNet has been installed successfully!

```

③ NPU 下编译

执行命令：`bash build_npu.sh -e ascend -j 6`， 编译过程如下图：

```
[ 94%] Built target grpc_csharp_ext
[ 94%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/client/credentials_cc.cc.o
[ 94%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/client/insecure_credentials.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/client/xds_credentials.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/codegen/codegen_init.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/alarm.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/auth_property_iterator.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/channel_arguments.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/channel_filter.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/completion_queue_cc.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/core_codegen.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/resource_quota_cc.cc.o
[ 95%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/rpc_method.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/secure_auth_context.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/secure_channel_arguments.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/secure_create_auth_context.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/tls_certificate_provider.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/tls_credentials_options.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/tls_credentials_options_util.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/validate_service_config.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/common/version_cc.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/async_generic_service.cc.o
[ 96%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/channel_argument_option.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/create_default_thread_pool.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/dynamic_thread_pool.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/external_connection_acceptor_impl.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/health/default_health_check_service.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/health/health_check_service.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/health_check_service_server_builder_option.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/insecure_server_credentials.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/secure_server_credentials.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/server_builder.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/server_callback.cc.o
[ 97%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/server_cc.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/server_context.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/server_credentials.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/server_posix.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/server/xds_server_credentials.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/thread_manager/thread_manager.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/util/byte_buffer_cc.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/util/status.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/util/string_ref.cc.o
[ 98%] Building CXX object CMakeFiles/grpc++.dir/src/cpp/util/time_cc.cc.o
[ 98%] Linking CXX shared library libluojianet_ms_grpcpp.so
[ 99%] Built target grpcpp
```

验证是否成功：

```
import numpy as np

from luojianet_ms import Tensor

import luojianet_ms.ops as ops

import luojianet_ms.context as context

context.set_context(device_target="Ascend")

x = Tensor(np.ones([1,3,3,4]).astype(np.float32))

y = Tensor(np.ones([1,3,3,4]).astype(np.float32))

print(ops.add(x, y))
```

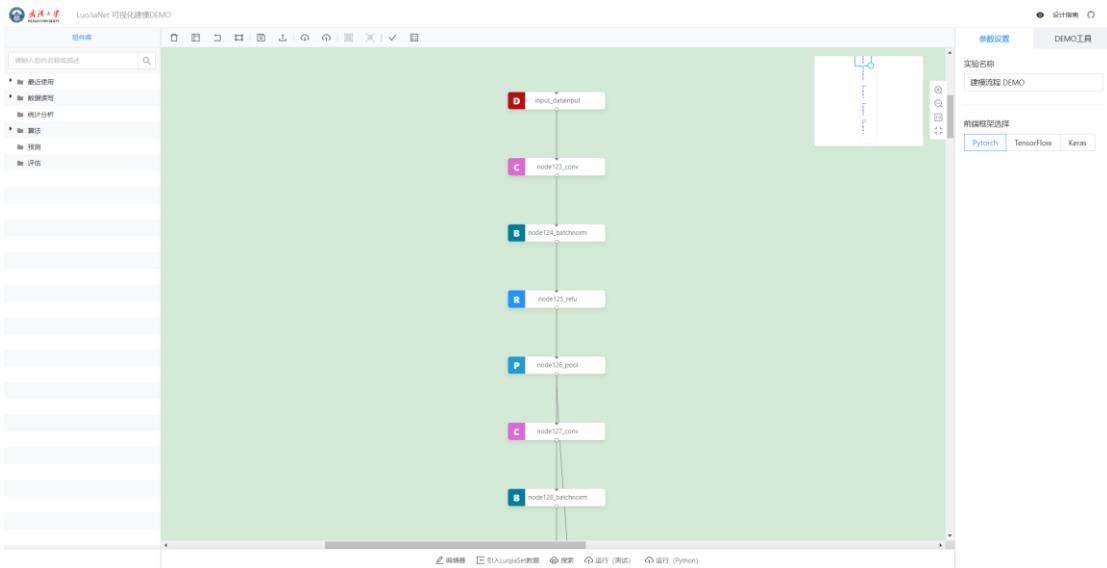
输出如下，说明安装成功：

```
[WARNING] CORE[23119,fffffb5a07010,python]:2022-06-10-20:15:44.511,844 [luojianet_ms/core/utils/ms_context.cc:128] CreateTensorPrintThread] MS_CONTEXT_DEVICEID = 0
[WARNING] DEBUG[23119,fffffb5a07010,python]:2022-06-10-20:15:44.519,602 [luojianet_ms/csrc/debugger/debugger.cc:96] Debugger] Not enabling debugger, Debugger does not support CPU.
[[[2, 2, 2, 2, 1]
[2, 2, 2, 2, 1]
[2, 2, 2, 2, 2, 1]
[[2, 2, 2, 2, 2, 1]
[2, 2, 2, 2, 2, 1]
[2, 2, 2, 2, 2, 1]]]
```

(3) 前端平台部署测试

① 模型构建

以 ResNet 模型为例进行测试，使用可视化建模拖拽式建模，建模图像图下图：



模型构建之后使用 Luojianet 后端将模型 Json 数据转换为 python 文件，转换后模型

代码如下：

```
import luojianet as nn
import luojianet.parametric_functions as PF
import luojianet.functions as F
from luojianet.logger import logger

# _weights_dict = dict()
# is_train = True
# def load_weights(weight_file):
#     import numpy as np
#     if weight_file == None:
#         return
#     try:
#         weights_dict = np.load(weight_file, allow_pickle=True).item()
#     except:
#         weights_dict = np.load(weight_file, allow_pickle=True, encoding='bytes').item()
#     return weights_dict

def LuoJiaNetModel(input_x, num_classes, num_layers, shortcut_type, test, tiny=False, weight_file = None):
    #global _weights_dict
    #_weights_dict = load_weights(weight_file)
    hidden = {}
    input      = input_x
    node17    = PF.convolution(inp=input, outmaps=64, kernel=(11, 11), stride=(4,4), pad=(2,2), group=1, with_bias = True, name='node17')
```

代码成功生成，模型构建流程成功。

② 模型训练

在 LuojiaNet 前端云平台上上传填写表单并上传 Resnet-18 模型文件，表单填写如下所示：

本地上传新模型

1 模型信息填写 2 模型上传 3 完成

模型名称: RESNET-18

模型类型: RESNET

模型任务类型: 场景检索

模型描述: TEST
4 / 900

【管理员】是否为公开模型?
 不公开 公开

提交

模型文件上传如下所示：

ResNet_AUTO.zip file uploaded successfully.

1 模型信息填写 2 模型上传 3 完成

模型代码上传 (*.zip)

Click or drag file to this area to upload
Support for a single upload.

ResNet_AUTO.zip

确认

流程完毕后，用户已成功将模型代码上传至数据库之中，可在训练订单界面进行训练订单创建任务，新建训练订单界面如下图所示，使用 Resnet-18 模型对 AID 数据库创建场景检索任务：

新建训练订单

X

模型选择 (模型名称)

RESNET-18

[查看选择模型详情](#)

数据集选择 [打开LuojiaSet](#)

场景检索 / AID



备注

请输入备注!

0 / 32

取消

确定

点击确定后，后台数据库中获得订单信息，等待管理员进行审核：

订单	模型ID	任务类型	数据集	备注	状态	创建时间	操作	管理员操作	申请者ID
<input type="checkbox"/> 066479ea-da07-4a2c-b63e-clead55d5c0e b63e-clead55d5c0e	a13b02be-9b17-4e20-a790-df6bc7add35e	场景检索	AID	-	等待审核	2022-06-17 15:25:32	-	日志下载 结束下载	c3ed894bc572865af2b8f 74817b1217

管理员通过审核后，显示“训练进行中”，则说明训练已在后台正常进行，此时前端页面如下图所示：

订单	模型ID	任务类型	数据集	备注	状态	创建时间	操作	管理员操作	申请者ID
<input type="checkbox"/> 066479ea-da07-4a2c-b63e-clead55d5c0e b63e-clead55d5c0e	a13b02be-9b17-4e20-a790-df6bc7add35e	场景检索	AID	-	训练进行中	2022-06-17 15:25:32	-	日志下载 结束下载	c3ed894bc572865af2b8f 74817b1217

后端 ModelArts 控制台任务显示如下图所示：

The screenshot shows the ModelArts control console interface. At the top, it displays the path '训练作业 / 066479ea-da07-4a2c-b63e-clead55d5c0e'. Below this, there's a navigation bar with tabs like '版本管理' and '版本对比'.

The main content area shows a detailed view of a task named '066479ea-da07-4a2c-b63e-clead55d5c0e | job699dee4b2'. It includes the following information:

- 配置信息:** 包含作业名称 (066479ea-da07-4a2c-b63e-clead55d5c0e | job699dee4b2), 状态 (运行中), 运行时间 (00:00:22), 配置文件 (V0001), 开始运行时间 (2022/06/17 15:30:17 GMT+08:00), 运行时长 (00:00:22), 资源池 (pool-ceee), 周期 (Ascend 8 * Ascend-910(32GB) | ARM: 192 核 720GB), 计算节点个数 (1), 日志输出位置 (/luojianet-frontend/order/066479ea-da07-4a2c-b63e-clead55d5c0e/log/), 和 NAS 挂载路径 (—).
- 运行参数:** 显示为 “--”.
- 训练输出位置:** /luojianet-frontend/order/066479ea-da07-4a2c-b63e-clead55d5c0e/output/
- 描述:** lujianet-fronted-TRAIN
- NAS 地址:** --

At the bottom right of the interface, there are buttons for '创建可视化作业' (Create Visualization Job), '创建模型' (Create Model), '创建新版训练作业' (Create New Version Training Job), '修改' (Modify), and '更多操作' (More Operations).

训练完毕后可在前端进行训练结果的 ckpt 等文件下载以及训练日志下载,具体如下图所示:

结果下载:

结果下载 X

model/net-10_1250.ckpt 文件大小:128.09MB	下载
model/net-6_1250.ckpt 文件大小:128.09MB	下载
model/net-7_1250.ckpt 文件大小:128.09MB	下载
model/net-8_1250.ckpt 文件大小:128.09MB	下载
model/net-9_1250.ckpt 文件大小:128.09MB	下载
model/net-graph.meta 文件大小:263.35KB	下载

复制所有下载链接至剪贴板

取消 确定

日志下载:

```
[WARNING] DEVICE(314,ffff8bf80420,python):2022-06-17-15:32:04.859.001
[luojianet_ms/ccsrc/runtime/device/ascend/ascend_stream_assign.cc:1294] ExistStreamSendAfterLastHcomNode] There is no hcom nodes of graph
1 in the root graph 1
[WARNING] DEVICE(316,ffff8b7e0420,python):2022-06-17-15:32:05.534.539
[luojianet_ms/ccsrc/runtime/device/ascend/ascend_stream_assign.cc:1294] ExistStreamSendAfterLastHcomNode] There is no hcom nodes of graph
1 in the root graph 1
[WARNING] DEVICE(312,fffffb8cc6420,python):2022-06-17-15:32:05.722.560
[luojianet_ms/ccsrc/runtime/device/ascend/ascend_stream_assign.cc:1294] ExistStreamSendAfterLastHcomNode] There is no hcom nodes of graph
1 in the root graph 1
Corrupt JPEG data: premature end of data segment
epoch: 1 step: 1250, loss is 2.8913733959198
epoch time: 143125.329 ms, per step time: 114.500 ms
epoch: 1 step: 1250, loss is 3.055614948272705
epoch time: 144040.629 ms, per step time: 115.233 ms
epoch: 1 step: 1250, loss is 2.8857975006103516
epoch time: 144724.041 ms, per step time: 115.779 ms
epoch: 1 step: 1250, loss is 2.935434341430664
epoch time: 145837.331 ms, per step time: 116.670 ms
epoch: 1 step: 1250, loss is 3.019761562347412
epoch time: 146169.192 ms, per step time: 116.935 ms
epoch: 1 step: 1250, loss is 3.0476739406585693
epoch time: 146650.256 ms, per step time: 117.320 ms
epoch: 1 step: 1250, loss is 3.042975902557373
epoch time: 146860.427 ms, per step time: 117.488 ms
epoch: 1 step: 1250, loss is 3.03090763092041
epoch time: 147225.739 ms, per step time: 117.781 ms
Corrupt JPEG data: premature end of data segment
epoch: 2 step: 1250, loss is 2.6974692344665527
epoch time: 35768.006 ms, per step time: 28.614 ms
epoch: 2 step: 1250, loss is 2.559579372406006
```

成功进行结果及日志下载后，说明模型训练成功。训练完成之后，用户可以根据此训练订单在推理界面新建推理任务。

③ 模型推理

选择第二步训练好的模型进行推理，模型训练后 CKPT 文件选择界面如下图所示：

选择训练订单

订单ID	模型ID	任务类型	数据集	备注	状态	CKPT选择 (仅可选择训练完成订单)
066479ea-da07-4a2c-b63e-c9ead55d5c0e	a13b05be-9b17-4e20-a790-d6fbca7add35e	场景检索	AID	-	● 训练完成	<button>CKPT选择</button>
1737cbfb-c483-4857-aec4-ae662d5c0fb3	4dc9bf53-98cf-4b0b-8bc9-1b2b9fc4f10d	场景检索	RSI-CB128	-	● 训练失败	<button>CKPT选择</button>
19293718-5a11-4bea-96c1-b7289f7a76a7	ed9e0a70-be47-495a-a472-256e2ca44b83	场景检索	AID	-	● 训练完成	<button>CKPT选择</button>
416ffd9b-8e7d-4061-9332-899c96150ecc	31b0a1f1-e96d-4281-a6a2-e5e9c7d72086	场景检索	AID	-	● 训练完成	<button>CKPT选择</button>
499491fb-71ab-4800-a39c-aee4328d4fd	4dc9bf53-98cf-4b0b-8bc9-1b2b9fc4f10d	场景检索	NaSC-TG2-RGB	-	● 等待审核	<button>CKPT选择</button>
5b824d3b-34f1-4ca9-8930-3d6e3d5c8c78	d8f1f992-9ce8-49c0-8e7b-f59877c2987	场景检索	RSI-CB256	-	● 等待审核	<button>CKPT选择</button>
67427ad3-caa5-4604-8150-a2545b9c6708	ade94800-4c92-4c3a-b10a-60baf69a7dea	场景检索	AID	test	● 训练完成	<button>CKPT选择</button>
708f2918-1c87-4906-a1aa-41617f6aedcd	4dc9bf53-98cf-4b0b-8bc9-1b2b9fc4f10d	场景检索	RSSCN7	-	● 训练完成	<button>CKPT选择</button>
7273555e-e8fe-458b-9ff7-0df0be4dcb2	4dc9bf53-98cf-4b0b-8bc9-1b2b9fc4f10d	场景检索	V-RSIR VGoogle	-	● 等待审核	<button>CKPT选择</button>

选择完成后自动生成推理任务表单，表单项如下图所示：

本地上传新模型

1 推理任务信息填写 2 数据上传 3 完成

已训练订单及CKPT文件选择

选择训练订单

* 订单ID
066479ea-da07-4a2c-b63e-c9ead55d5c0e

* CKPT文件路径
model/net-6_1250.ckpt

* 模型ID
a13b05be-9b17-4e20-a790-d6fbca7add35e

推理任务类型

备注

0 / 32

提交

推理任务初始信息完成后，用户可上传需要进行推理的数据，数据上传页面如下图所示：



如图，上传 input.png 作为测试图像，点击确定后任务推送至华为 ModelArts 进行推理任务，此时 ModelArts 控制台任务信息显示如下：

配置信息	日志	资源占用情况	
作业名称	7777494-clc9-4da3-b7c1-a1641f7f0b41 job324bfe5	健康地址	swccn-central-221.ovaiguang.com/modelarts-image_wdg/flusig_ms-ascend910...
状态	初始化	代码目录	/luojianet-frontend/model/a13b05be-9b17-4e20-a790-d6fb7ad535e/code/
运行版本	V0001	启动命令	/bin/bash /home/work/run_train.sh >3://luojianet-frontend/model/a13b05be-9b...
开始运行时间	--	训练数据源	/luojianet-frontend/pred/7777494-clc9-4da3-b7c1-a1641f7f0b41/input/
运行时间	00:00:00	运行参数	LUOJIANET_PRED_CKPT_PATH=>3:/luojianet-frontend/order/006479ea-da07-4a...
规格	Ascend 2 * Ascend-910(32GB) ARM: 48 核 128GB	训练输出位置	/luojianet-frontend/pred/7777494-clc9-4da3-b7c1-a1641f7f0b41/output/
计算节点个数	1	描述	luojianet-frontend-PRED-User Task
日志输出位置	/luojianet-frontend/pred/7777494-clc9-4da3-b7c1-a1641f7f0b41/log/	NAS 地址	--
NAS 挂载路径	--		

推理完成后可在前端下载，Resnet-18 执行场景检索后所得结果为 json 格式，数据信息如下图所示：

```

1  {"title": "Top-1", "class_num": 0, "class_name": "Airport", "class_prob": 0.1319580078125},
2  {"title": "Top-2", "class_num": 26, "class_name": "Square", "class_prob": 0.08355712890625},
3  {"title": "Top-3", "class_num": 12, "class_name": "Industrial", "class_prob": 0.07989501953125},
4  {"title": "Top-4", "class_num": 6, "class_name": "Church", "class_prob": 0.06744384765625},
5  {"title": "Top-5", "class_num": 2, "class_name": "BaseballField", "class_prob": 0.06549072265625},

```

获取 json 后，说明模型推理成功，至此，整个构建-训练-推理流程测试完毕。

3.2 大幅面处理特性与功能

3.2.1 遥感大幅面影像四叉树索引功能

(1) 模块的导入与使用

LuoJiaNET 安装测试成功后，即可在 python 训练脚本中直接导入遥感大幅面四叉树索引

功能模块，其使用方式如下：

```
from lujianet_ms.geobject import get_objects

# Get the minimum bounding rectangle data of one-specified class.

# param[in] device_num, the number of device for training (max = 8).

# param[in] device_id, the current device ID.

# param[in] image_path, big_input image path.

# param[in] label_path, big_input label path.

# param[in] n_classes, num classes of labels.

# param[in] ignore_label, pad value of ground features.

# param[in] seg_threshold, segmentation settings.

# param[in] block_size, basic processing unit for big_input data.

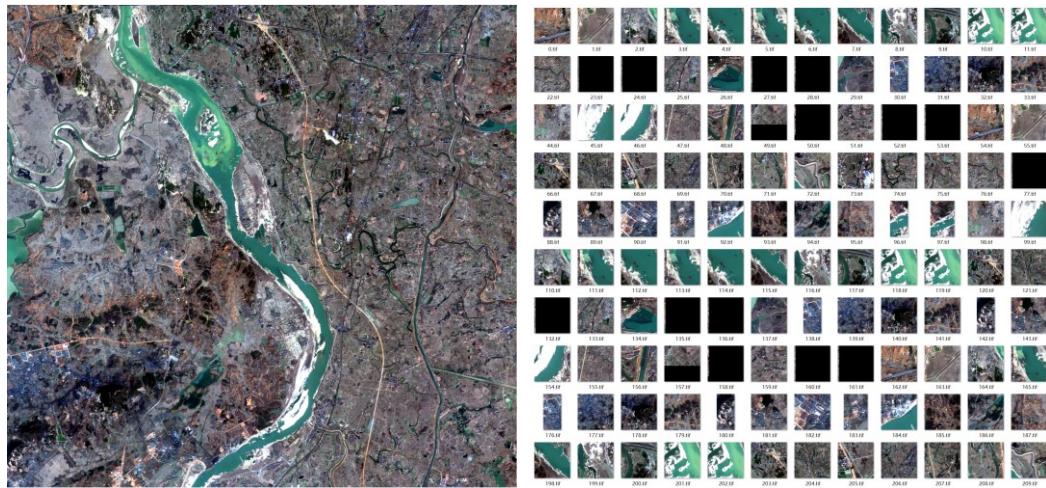
# param[in] max_searchsize, max output data size (max_searchsize x
max_searchsize).

# return out, image-label objects in Numpy dtype.

data_objects = get_objects(args.device_number, args.device_id, image_path,
label_path, args.num_classes, args.ignore_label, 150, 4096, 800)

image_objects, label_objects = data_objects[0], data_objects[1]
```

image_objects 和 label_objects 得到的大幅影像目标对象数据展示如下：

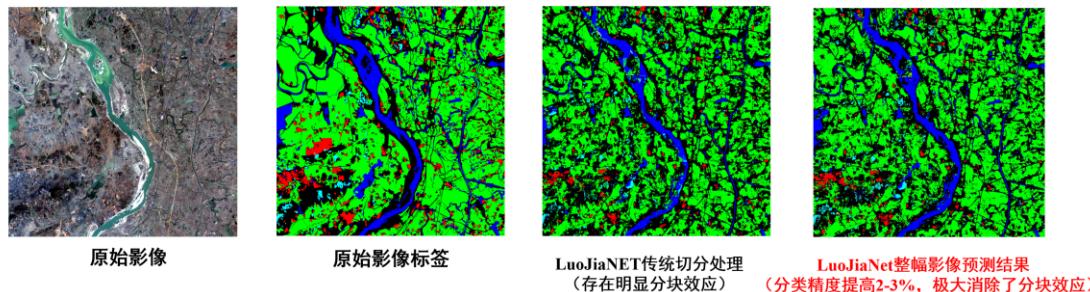


在得到当前大幅影像的所有目标对象数据之后，即可进行网络模型的数据加载和训练。

(2) 测试结果

随机抽取 GID 数据集的 120 张作为训练集，剩下 30 张作为测试集，在打乱顺序的情况下对整幅影像进行读取和训练，目标地物对象的尺寸大小统一为 800×800 ，使用 DeepLabV3+ 模型训练 300 个 epoch，采用总体精度 (OA)、平均交并比 (MIoU) 这两个指标进行评价，其测试结果的定量评估如下：

处理策略对比 (Backbone=ResNet-101)		GID 数据集 (6类地物)	
		Overall	
		mIoU	OA
基于LuoJiaNET传统影像切分处理 ($1 \times 3 \times 6800 \times 7200 \rightarrow N \times 3 \times 800 \times 800$)		13.01	24.03
基于LuoJiaNet整幅影像直接读取处理 ($1 \times 3 \times 6800 \times 7200 \rightarrow 1 \times 3 \times 6800 \times 7200$)		15.67	27.10



从结果看，与传统的切分方法相比，在模型训练中引入的遥感大幅面四叉树索引功能模块，能够将地物分类结果提高 2-3%，极大消除了地物分类任务中的“分块效应”。

3.2.2 遥感大幅面影像专用 IR 功能

(1) TVM 编译链接

TVM 源码基于 TVM0.8 Release 版，可以采用两种方式进行编译

1) 直接下载仓库内的 tvm 代码并编译。所有需要对源码修改的位置已经修改完毕，编译过程可参考 TVM 官网。

2) 基于自己的 TVM 版本进行修改。具体修改位置如下：

- python 部分，基础路径为 tvm/python/tvm:

- relay/op/nn/nn.py
- relay/op/nn/_nn.py
- relay/op/strategy/cuda.py
- relay/op/strategy/generic.py
- topi/cuda/conv2d.py
- contrib/cudnn.py
- relay/op/_tensor_grad.py

- C++部分，基础路径为 tvm/src:

- relay/op/nn/convolution.cc
- relay/op/nn/convolution.h
- runtime/contrib/cudnn/conv_backward.cc
- runtime/contrib/cudnn/cudnn_utils.h

(2) 推理和训练 FCN32s

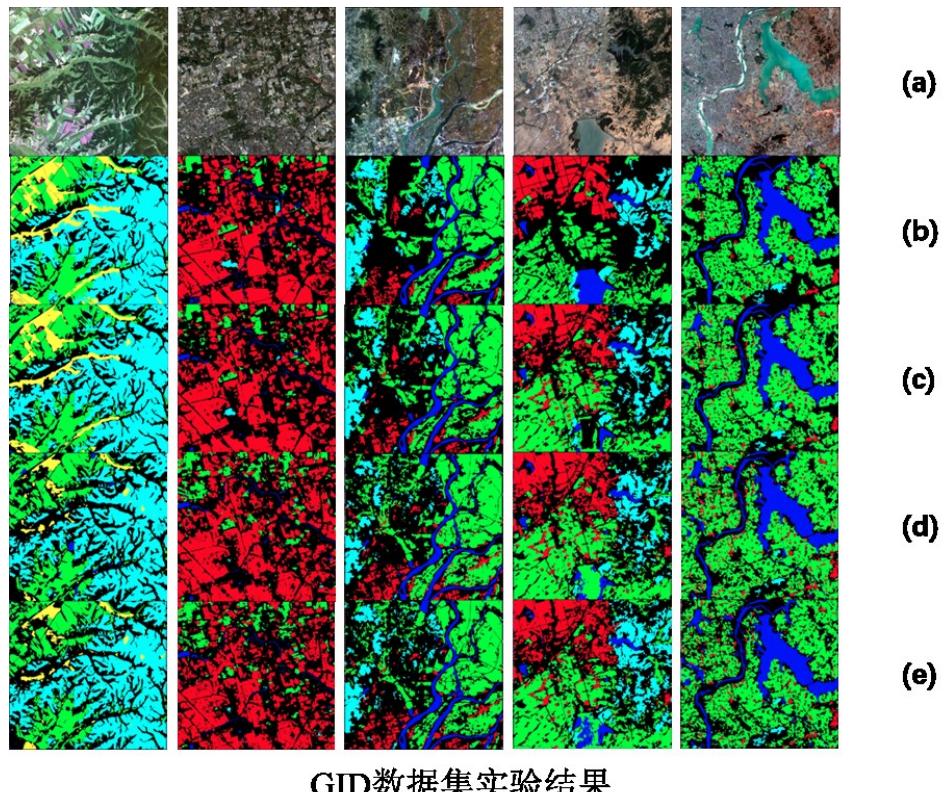
1) 对数据集 label 进行预处理，将 RGB 图像转成单通道图像

命令行输入 `python label.py`, 需要修改数据集目录

- 2) 根据 GPU 显存修改算子分解的块数 N, 保证能够正常计算
- 3) 训练 `python tvm_train_fcn_gid.py 1> log.txt 2>&!` 将输出写入文件方便记录
- 4) 根据训练好的模型在测试集上进行推理 `python tvm_infer_fcn_gid.py`

(2) 测试结果

随机抽取 120 张作为训练集, 剩下 30 张作为测试集, 在打乱顺序的情况下分别采用算子分解方法对整图进行训练, 裁剪成 1000×1000 大小后再进行训练, 以及使用最近滤波器降采样到 680×720 大小后再进行训练, 各训练 100 个 epoch。



(图a是原始图像, 图b是Ground Truth, 图c是算子分解方法的结果, 图d是裁剪方法的结果, e是降采样方法的结果)

采用像素精度 (PA)、平均像素精度 (MPA)、平均交并比 (MIoU)、加权平均交并比 (FWIoU), 测试结果的定量评估如下:

评价指标	训练方法		
	算子分解算法	裁剪	降采样
PA	86.3%	77.7%	77.8%
MPA	79.6%	71.9%	71.6%
MIoU	63.3%	48.5%	45.8%
FWIoU	76.7%	64.9%	65.2%

3.2.3 遥感大幅面影像算子分解功能

(1) FCN8s 算子分解调用方式如下：

(a) 网络结构定义

```
import luojianet_ms.nn as nn
from luojianet_ms.ops import operations as P

class FCN8s(nn.Module):
    def __init__(self, n_class):
        super().__init__()
        self.n_class = n_class
        self.conv1 = nn.SequentialCell(
            nn.Conv2d(in_channels=3, out_channels=64,
                     kernel_size=3, weight_init='xavier_uniform'),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64,
```

```
        kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(64),  
  
        nn.ReLU()  
    )  
  
    self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)  
  
    self.conv2 = nn.SequentialCell(  
  
        nn.Conv2d(in_channels=64, out_channels=128,  
  
            kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(128),  
  
        nn.ReLU(),  
  
        nn.Conv2d(in_channels=128, out_channels=128,  
  
            kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(128),  
  
        nn.ReLU()  
    )  
  
    self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)  
  
    self.conv3 = nn.SequentialCell(  
  
        nn.Conv2d(in_channels=128, out_channels=256,  
  
            kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(256),  
  
        nn.ReLU(),  
  
        nn.Conv2d(in_channels=256, out_channels=256,
```

```
        kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(256),  
  
        nn.ReLU(),  
  
        nn.Conv2d(in_channels=256, out_channels=256,  
  
                 kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(256),  
  
        nn.ReLU()  
  
)  
  
self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)  
  
self.conv4 = nn.SequentialCell(  
  
    nn.Conv2d(in_channels=256, out_channels=512,  
  
             kernel_size=3, weight_init='xavier_uniform'),  
  
    nn.BatchNorm2d(512),  
  
    nn.ReLU(),  
  
    nn.Conv2d(in_channels=512, out_channels=512,  
  
             kernel_size=3, weight_init='xavier_uniform'),  
  
    nn.BatchNorm2d(512),  
  
    nn.ReLU(),  
  
    nn.Conv2d(in_channels=512, out_channels=512,  
  
             kernel_size=3, weight_init='xavier_uniform'),  
  
    nn.BatchNorm2d(512),  
  
    nn.ReLU()
```

```
)  
  
    self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)  
  
    self.conv5 = nn.SequentialCell(  
  
        nn.Conv2d(in_channels=512, out_channels=512,  
  
                 kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(512),  
  
        nn.ReLU(),  
  
        nn.Conv2d(in_channels=512, out_channels=512,  
  
                 kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(512),  
  
        nn.ReLU(),  
  
        nn.Conv2d(in_channels=512, out_channels=512,  
  
                 kernel_size=3, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(512),  
  
        nn.ReLU()  
  
)  
  
    self.pool5 = nn.MaxPool2d(kernel_size=2, stride=2)  
  
    self.conv6 = nn.SequentialCell(  
  
        nn.Conv2d(in_channels=512, out_channels=4096,  
  
                 kernel_size=7, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(4096),  
  
        nn.ReLU(),
```

```
)  
  
    self.conv7 = nn.SequentialCell(  
  
        nn.Conv2d(in_channels=4096, out_channels=4096,  
  
            kernel_size=1, weight_init='xavier_uniform'),  
  
        nn.BatchNorm2d(4096),  
  
        nn.ReLU(),  
  
    )  
  
    self.score_fr = nn.Conv2d(in_channels=4096,  
  
        out_channels=self.n_class,  
  
            kernel_size=1,  
  
            weight_init='xavier_uniform')  
  
    self.upscore2 = nn.Conv2dTranspose(in_channels=self.n_class,  
  
        out_channels=self.n_class,  
  
            kernel_size=4, stride=2,  
  
            weight_init='xavier_uniform')  
  
    self.score_pool4 = nn.Conv2d(in_channels=512,  
  
        out_channels=self.n_class,  
  
            kernel_size=1,  
  
            weight_init='xavier_uniform')  
  
    self.upscore_pool4 = nn.Conv2dTranspose(in_channels=self.n_class,  
  
        out_channels=self.n_class,  
  
            kernel_size=4, stride=2,
```

```
weight_init='xavier_uniform')

    self.score_pool3 = nn.Conv2d(in_channels=256,
                               out_channels=self.n_class,
                               kernel_size=1,

weight_init='xavier_uniform')

    self.upscore8 = nn.Conv2dTranspose(in_channels=self.n_class,
                                   out_channels=self.n_class,
                                   kernel_size=16, stride=8,

weight_init='xavier_uniform')

    self.upscore_pool3 = nn.Conv2dTranspose(in_channels=self.n_class,
                                           out_channels=self.n_class,
                                           kernel_size=4, stride=2,

weight_init='xavier_uniform')

    self.score_pool2 = nn.Conv2d(in_channels=128,
                               out_channels=self.n_class,
                               kernel_size=1,

weight_init='xavier_uniform')

    self.upscore4 = nn.Conv2dTranspose(in_channels=self.n_class,
                                   out_channels=self.n_class,
                                   kernel_size=8, stride=4,

weight_init='xavier_uniform')

    self.shape = P.Shape()
```

```
    self.cast = P.Cast()

    self.add1 = P.Add()

    self.add2 = P.Add()

    self.add3 = P.Add()

def set_model_parallel_shard_strategy(self, device_num):

    self.conv2d_strategy = ((1, 1, 1, device_num), (1, 1, 1, 1))

    self.bn_strategy = ((1, 1, 1, device_num), (1,), (1,), (1,), (1,))

    self.relu_strategy = ((1, 1, 1, device_num),)

    self.maxpool_strategy = ((1, 1, 1, device_num),)

    self.add_strategy = ((1, 1, 1, device_num), (1, 1, 1, device_num))

    self.conv1.cell_list[0].conv2d.shard(self.conv2d_strategy)

    self.conv1.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv1.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv1.cell_list[2].relu.shard(self.relu_strategy)

    self.conv1.cell_list[3].conv2d.shard(self.conv2d_strategy)

    self.conv1.cell_list[4].bn_train.shard(self.bn_strategy)

    self.conv1.cell_list[4].bn_infer.shard(self.bn_strategy)

    self.conv1.cell_list[5].relu.shard(self.relu_strategy)

    self.pool1.max_pool.shard(self.maxpool_strategy)

    self.conv2.cell_list[0].conv2d.shard(self.conv2d_strategy)
```

```
    self.conv2.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv2.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv2.cell_list[2].relu.shard(self.relu_strategy)

    self.conv2.cell_list[3].conv2d.shard(self.conv2d_strategy)

    self.conv2.cell_list[4].bn_train.shard(self.bn_strategy)

    self.conv2.cell_list[4].bn_infer.shard(self.bn_strategy)

    self.conv2.cell_list[5].relu.shard(self.relu_strategy)

    self.pool2.max_pool.shard(self.maxpool_strategy)

    self.conv3.cell_list[0].conv2d.shard(self.conv2d_strategy)

    self.conv3.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv3.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv3.cell_list[2].relu.shard(self.relu_strategy)

    self.conv3.cell_list[3].conv2d.shard(self.conv2d_strategy)

    self.conv3.cell_list[4].bn_train.shard(self.bn_strategy)

    self.conv3.cell_list[4].bn_infer.shard(self.bn_strategy)

    self.conv3.cell_list[5].relu.shard(self.relu_strategy)

    self.conv3.cell_list[6].conv2d.shard(self.conv2d_strategy)

    self.conv3.cell_list[7].bn_train.shard(self.bn_strategy)

    self.conv3.cell_list[7].bn_infer.shard(self.bn_strategy)

    self.conv3.cell_list[8].relu.shard(self.relu_strategy)

    self.pool3.max_pool.shard(self.maxpool_strategy)

    self.conv4.cell_list[0].conv2d.shard(self.conv2d_strategy)
```

```
    self.conv4.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv4.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv4.cell_list[2].relu.shard(self.relu_strategy)

    self.conv4.cell_list[3].conv2d.shard(self.conv2d_strategy)

    self.conv4.cell_list[4].bn_train.shard(self.bn_strategy)

    self.conv4.cell_list[4].bn_infer.shard(self.bn_strategy)

    self.conv4.cell_list[5].relu.shard(self.relu_strategy)

    self.conv4.cell_list[6].conv2d.shard(self.conv2d_strategy)

    self.conv4.cell_list[7].bn_train.shard(self.bn_strategy)

    self.conv4.cell_list[7].bn_infer.shard(self.bn_strategy)

    self.conv4.cell_list[8].relu.shard(self.relu_strategy)

    self.pool4.max_pool.shard(self.maxpool_strategy)

    self.conv5.cell_list[0].conv2d.shard(self.conv2d_strategy)

    self.conv5.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv5.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv5.cell_list[2].relu.shard(self.relu_strategy)

    self.conv5.cell_list[3].conv2d.shard(self.conv2d_strategy)

    self.conv5.cell_list[4].bn_train.shard(self.bn_strategy)

    self.conv5.cell_list[4].bn_infer.shard(self.bn_strategy)

    self.conv5.cell_list[5].relu.shard(self.relu_strategy)

    self.conv5.cell_list[6].conv2d.shard(self.conv2d_strategy)

    self.conv5.cell_list[7].bn_train.shard(self.bn_strategy)
```

```
    self.conv5.cell_list[7].bn_infer.shard(self.bn_strategy)

    self.conv5.cell_list[8].relu.shard(self.relu_strategy)

    self.pool5.max_pool.shard(((1, 1, 1, device_num),))

    self.conv6.cell_list[0].conv2d.shard(self.conv2d_strategy)

    self.conv6.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv6.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv6.cell_list[2].relu.shard(self.relu_strategy)

    self.conv7.cell_list[0].conv2d.shard(self.conv2d_strategy)

    self.conv7.cell_list[1].bn_train.shard(self.bn_strategy)

    self.conv7.cell_list[1].bn_infer.shard(self.bn_strategy)

    self.conv7.cell_list[2].relu.shard(self.relu_strategy)

    self.score_fr.conv2d.shard(self.conv2d_strategy)

    self.upscore2.conv2d_transpose.shard(self.conv2d_strategy)

    self.score_pool4.conv2d.shard(self.conv2d_strategy)

    self.upscore_pool4.conv2d_transpose.shard(self.conv2d_strategy)

    self.score_pool3.conv2d.shard(self.conv2d_strategy)

    self.upscore8.conv2d_transpose.shard(self.conv2d_strategy)

    self.add1.shard(self.add_strategy)

    self.add2.shard(self.add_strategy)

    self.add3.shard(self.add_strategy)

    self.upscore_pool3.conv2d_transpose.shard(self.conv2d_strategy)
```

```
self.score_pool2.conv2d.shard(self.conv2d_strategy)

self.upscore4.conv2d_transpose.shard(self.conv2d_strategy)

def call(self, x):

    x1 = self.conv1(x)

    p1 = self.pool1(x1)

    x2 = self.conv2(p1)

    p2 = self.pool2(x2)

    x3 = self.conv3(p2)

    p3 = self.pool3(x3)

    x4 = self.conv4(p3)

    p4 = self.pool4(x4)

    x5 = self.conv5(p4)

    p5 = self.pool5(x5)

    x6 = self.conv6(p5)

    x7 = self.conv7(x6)

    sf = self.score_fr(x7)

    u2 = self.upscore2(sf)

    s4 = self.score_pool4(p4)

    f4 = self.add1(s4, u2)

    u4 = self.upscore_pool4(f4)
```

```
s3 = self.score_pool3(p3)

f3 = self.add2(s3, u4)

u3 = self.upscore_pool3(f3)

s2 = self.score_pool2(p2)

f2 = self.add3(s2, u3)

u2 = self.upscore4(f2)

return u2
```

(b) 数据加载并行切分：

```
if split_size is not None:

    if isinstance(split_size, int):

        num_h, num_w = split_size, split_size

    elif isinstance(split_size, tuple or list):

        num_h, num_w = split_size

column_img = "data"

column_lbl = "label"

column_img_slice = [column_img + str(x) for x in range(num_h * num_w)]

column_lbl_slice = [column_lbl + str(x) for x in range(num_h * num_w)]

slice_patches_op = vision.SlicePatches(num_h, num_w)
```

```
data_set = data_set.map(operations=slice_patches_op, input_columns=["data"],  
output_columns=column_img_slice, column_order=column_img_slice + ['label'] +  
['name'], num_parallel_workers=8)  
  
data_set = data_set.map(operations=slice_patches_op, input_columns=["label"],  
output_columns=column_lbl_slice, column_order=column_img_slice + column_lbl_slice  
+ ['name'], num_parallel_workers=8)
```

(2) FCN8s 多卡并行切分调用方式如下：

```
parallel_mode = ParallelMode.SEMI_AUTO_PARALLEL

dataset_strategy = (((1, 1, 1, rank_size), (1, 1, rank_size)))

context.reset_auto_parallel_context()

context.set_auto_parallel_context(parallel_mode=parallel_mode,
                                  gradients_mean=True, dataset_strategy=dataset_strategy,
                                  device_num=device_num, enable_alltoall=True)

if context.get_auto_parallel_context("parallel_mode") in
    [ParallelMode.SEMI_AUTO_PARALLEL, ParallelMode.AUTO_PARALLEL]:
    net.set_model_parallel_shard_strategy(device_num)
```

命令行中执行: `mpirun -n 8 python main_train.py`, 运行过程如下:

```
epoch: 428 step: 8, loss is 1.306698203086853
epoch: 428 step: 8, loss is 1.306698203086853
epoch time: 15650.861 ms, per step time: 1956.358 ms
epoch: 428 step: 8, loss is 1.306698203086853
epoch: 428 step: 8, loss is 1.306698203086853
epoch: 428 step: 8, loss is 1.306698203086853
epoch time: 15651.318 ms, per step time: 1956.415 ms
epoch time: 15651.270 ms, per step time: 1956.409 ms
epoch time: 15642.780 ms, per step time: 1955.348 ms
epoch time: 15650.847 ms, per step time: 1956.356 ms
epoch time: 15478.251 ms, per step time: 1934.781 ms
epoch: 429 step: 8, loss is 1.1542125940322876
epoch time: 15453.249 ms, per step time: 1931.656 ms
epoch: 429 step: 8, loss is 1.1542125940322876
epoch: 429 step: 8, loss is 1.1542125940322876
epoch time: 15453.364 ms, per step time: 1931.670 ms
epoch: 429 step: 8, loss is 1.1542125940322876
epoch: 429 step: 8, loss is 1.1542125940322876
epoch: 429 step: 8, loss is 1.1542125940322876
epoch time: 15454.561 ms, per step time: 1931.820 ms
epoch time: 15453.940 ms, per step time: 1931.742 ms
epoch: 429 step: 8, loss is 1.1542125940322876
epoch time: 15454.378 ms, per step time: 1931.797 ms
epoch time: 15453.898 ms, per step time: 1931.737 ms
epoch time: 15454.004 ms, per step time: 1931.751 ms
epoch time: 15646.457 ms, per step time: 1955.807 ms
epoch: 430 step: 8, loss is 1.2845218181610107
epoch time: 15652.722 ms, per step time: 1956.590 ms
epoch: 430 step: 8, loss is 1.2845218181610107
epoch time: 15653.276 ms, per step time: 1956.659 ms
epoch: 430 step: 8, loss is 1.2845218181610107
epoch time: 15653.267 ms, per step time: 1956.658 ms
epoch: 430 step: 8, loss is 1.2845218181610107
epoch time: 15653.315 ms, per step time: 1956.664 ms
epoch time: 15653.878 ms, per step time: 1956.735 ms
epoch time: 15653.846 ms, per step time: 1956.731 ms
epoch time: 15653.337 ms, per step time: 1956.667 ms
epoch time: 15548.024 ms, per step time: 1943.503 ms
epoch: 431 step: 8, loss is 1.2569628953933716
epoch time: 15576.583 ms, per step time: 1947.073 ms
epoch: 431 step: 8, loss is 1.2569628953933716
epoch: 431 step: 8, loss is 1.2569628953933716
epoch: 431 step: 8, loss is 1.2569628953933716
epoch time: 15576.622 ms, per step time: 1947.078 ms
epoch time: 15576.611 ms, per step time: 1947.076 ms
epoch: 431 step: 8, loss is 1.2569628953933716
epoch time: 15576.532 ms, per step time: 1947.066 ms
epoch: 431 step: 8, loss is 1.2569628953933716
epoch time: 15577.610 ms, per step time: 1947.201 ms
epoch: 431 step: 8, loss is 1.2569628953933716
epoch: 431 step: 8, loss is 1.2569628953933716
epoch time: 15577.116 ms, per step time: 1947.139 ms
epoch time: 15577.145 ms, per step time: 1947.143 ms
epoch time: 15588.108 ms, per step time: 1948.514 ms
```

(3) 运行结果

(a) 实现细节

在数据预处理时，以 120 张 6 个类别的 GID 数据为训练集，其余为验证集，验证模型精度。GPU 型号为 Nvidia Tesla V100，单卡显存 32Gb，8 卡并行处理。

在进行训练时，采用的损失函数为多类交叉熵 CE 损失，优化器选择为 SGD。初始学习率统一固定为 1.28，batch size 设置为 1，训练共 600 轮，每次均以原始像幅的 GID 数据作为输入。此外在训练中由于背景类是除上述 5 类地物的其他地物类别，因此背景类需要参与到网络训练过程，网络最终输出类别为 6，并参与最终的定量评价。

(b) 评价指标

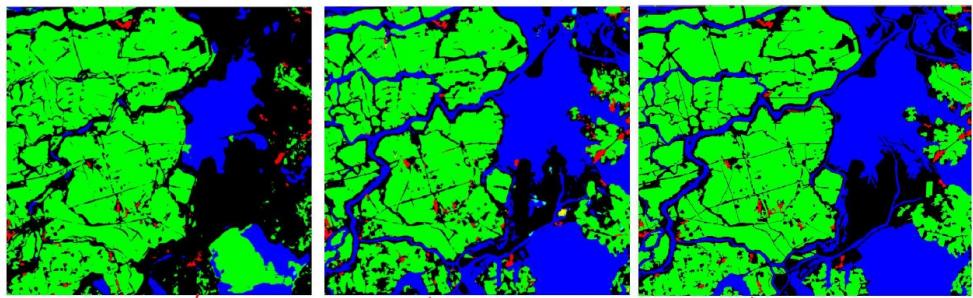
在实验中，采用的定量指标采用多个类别的平均交并比(mIoU)。

(c) 定量实验结果

表 1 FCN8s 精度结果

框架	mIoU
FCN8s (切分)	59.80
FCN8s (大幅面)	63.20

FCN8s 采用切分与原始大幅面处理方法，在 GID 分类数据集上的定量结果见表 1，从结果看出，采用大幅面算子等价分解处理的方法，以原始像幅的影像作为输入，其在验证集上的平均交并比为 63.20%，与传统切分策略相比，mIoU 提升约 4%，处理结果达到预期目标。下图为相应的可视化结果，采用大幅面整幅载入方法可保持较好的空间上下文信息，“分块效应”得以消除。



(a) 切分

(b) 大幅面

(c) 真值

大幅面影像算子分解测试结果

3.3 遥感先验知识并行计算

3.3.1 遥感先验知识提取

当前 LuoJiaNET 可支持 41 种遥感影像先验知识提取的方法，包括植被指数、水体指数、建筑物纹理与颜色特征、SAR 影像指数等多源遥感影像先验知识提取方法。列表如下：

表 3.1 植被指数类

序号	名称	公式	来源	备注
1	Normalized Difference Vegetation Index (NDVI)	$\text{NDVI} = \frac{(\text{NIR} - \text{R})}{(\text{NIR} + \text{R})}$	Deering D W. RANGELAND REFLECTANCE CHARACTERISTICS MEASURED BY AIRCRAFT AND SPACECRAFT SENSORS[M]. Texas A&M University, 1978.	输入影像：RGB+NIR (四波段)
2	Ratio Vegetation Index (RVI)	$\text{RVI} = \frac{\text{NIR}}{\text{R}}$	Jordan C F. Derivation of leaf-area index from quality of light on the forest floor[J]. Ecology, 1969, 50(4):	输入影像：RGB+NIR (四波段)

			663-666.	
3	Difference Vegetation Index (DVI)	$DVI = \frac{NIR - R}{R}$	Jordan C F. Derivation of leaf-area index from quality of light on the forest floor[J]. Ecology, 1969, 50(4): 663-666.	输入影像： RGB+NIR (四波段)
4	Enhanced Vegetation Index (EVI)	$EVI = 2.5 * \frac{(NIR - R) / (NIR + 6.0 * R - 7.5 * B + 1)}{(1 + atmospheric noise)}$	Liu H Q, Huete A. A feedback based modification of the NDVI to minimize canopy background and atmospheric noise[J]. IEEE transactions on geoscience and remote sensing, 1995, 33(2): 457-465.	输入影像： RGB+NIR (四波段)
5	Soil-adjusted Vegetation Index (SAVI)	$SAVI = \frac{(NIR - R) / (NIR + R + L) * (1 + L)}{L \in [0, 1]}$	Huete A R. A soil-adjusted vegetation index (SAVI)[J]. Remote sensing of environment, 1988, 25(3): 295-309.	输入影像： RGB+NIR (四波段)
6	Modified Adjusted Vegetation lindex (MSAVI)	$MSAVI = \frac{2 * NIR + 1 - \sqrt{((2 * NIR + 1)^2 + 8 * L) / 8}}{sqrt(((2 * NIR + 1)^2 + 8 * L) / 8)}$	Qi J, Chehbouni A, Huete A R, et al. A modified soil adjusted vegetation index[J]. Remote sensing of environment, 1994, 48(2): 119-126.	输入影像： RGB+NIR (四波段)

		$- 8 * (\text{NIR} - \text{R})) / 2$		
7	Transformed Vegetation Index (TVI)	$\text{TVI} = \text{sqrt} \left(\frac{(\text{NIR} - \text{R})}{(\text{NIR} + \text{R})} + 0.5 \right)$	Deering D W. Measuring "forage production" of grazing units from Landsat MSS data[C]//Proceedings of the Tenth International Symposium of Remote Sensing of the Environment. 1975: 1169-1198.	输入影像: RGB+NIR (四波段)
8	Wide Dynamic Range Vegetation Index (WDRVI)	$\text{WDRVI} = (\alpha * \text{NIR} - \text{R}) / (\alpha * \text{NIR} + \text{R})$ $\alpha \in [0.1, 0.2]$	Gitelson A A. Wide dynamic range vegetation index for remote quantification of biophysical characteristics of vegetation[J]. Journal of plant physiology, 2004, 161(2): 165-173.	输入影像: RGB+NIR (四波段)
9	Renormalized Difference Vegetation Index (RDVI)	$\text{RDVI} = \frac{\text{PAR absorbed by vegetation from bidirectional reflectance measurements[J].}}{\text{sqrt}(\text{NIR} + \text{R})}$	Roujean J L, Breon F M. Estimating PAR absorbed by vegetation from bidirectional reflectance measurements[J]. Remote sensing of Environment, 1995, 51(3): 375-384.	输入影像: RGB+NIR (四波段)
10	Optimization of OSAVI	$= \text{Rondeaux G, Steven M, Baret F.}$		输入影像:

	Soil-Adjusted Vegetation Indices (OSAVI)	$(NIR - R) / (NIR + R + \theta)$ const $\theta = 0.16$	Optimization of soil-adjusted vegetation indices[J]. Remote sensing of environment, 1996, 55(2): 95-107.	RGB+NIR (四波段)
--	------------------------------------------------	-----------------------------------------------------------	----------------------------------------------------------------------------------------------------------	------------------

表 3.2 水体指数类

11	Normalized Difference Water Index (NDWI)	$NDWI = (G - NIR) / (G + NIR)$	McFeeters S K. The use of the Normalized Difference Water Index (NDWI) in the delineation of open water features[J]. International journal of remote sensing, 1996, 17(7): 1425-1432.	输入影像： RGB+NIR (四波段)
12	Modified Normalized Difference Water Index (MNDWI)	$MNDWI = (G_{MIR_1} - NIR_{MIR_1}) / (G_{MIR_1} + NIR_{MIR_1})$	徐涵秋. 利用改进的归一化差异水体指数 (MNDWI) 提取水体信息的研究[D]., 2005.	多光谱 / 高光谱 按波段输入
13	Combined Index for Water	$CIWI = NDVI + NIR + C$	莫伟华, 孙涵, 钟仕全, 等. MODIS 水体指数模型 (CIWI) 研究及其应	输入影像： RGB+NIR

	identification (CIWI)	$C = 100$	用[J]. 遥感信息, 2007 (5): 16-21.	(四波段)
14	Enhanced Water index (EWI-y)	$EWI-y = (G - NIR - MIR_1) / (G + NIR + MIR_1)$	闫需, 张友静, 张元. 利用增强型水体指数 (EWI) 和 GIS 去噪音技术提取半干旱地区水系信息的研究[J]. 遥感信息, 2007 (6): 62-67.	多光谱 / 高光谱 按波段输入
15	Normalized Difference Pond Index (NDPI)	$NDPI = (MIR_1 - G) / (MIR_1 + G)$	Lacaux J P, Tourre Y M, Vignolles C, et al. Classification of ponds from high-spatial resolution remote sensing: Application to Rift Valley Fever epidemics in Senegal[J]. Remote Sensing of Environment, 2007, 106(1): 66-74.	多光谱 / 高光谱 按波段输入
16	New Water Index (NWI)	$NWI = (B - (NIR + MIR_1 + MIR_2)) / (B + (NIR + MIR_1 + MIR_2))$	丁凤. 一种基于遥感数据快速提取水体信息的新方法[J]. 遥感技术与应用, 2012, 24(2): 167-171.	多光谱 / 高光谱 按波段输入
17	Modified CIWI (MCIWI)	$MCIWI = NDVI + NDBI,$ $NDBI = (MIR - (MCIWI))$	杨宝钢, 陈昉, 罗孳孳. 基于 MODIS 的改进型组合水体指数 提取复杂水体信息的试	多光谱 / 高光谱 按波段输入

		NIR) / (MIR ₁ + NIR)	验[J]. 西南大学学报: 自然科学版, 2011, 33(1): 112-119.	
18	Gaussian Normalized Water Index (GNDWI)	$GNDWI = (NDWI - NDWI_{\bar{}}) / \delta$ NDWI ₁ 为所有像素 NDWI 的均值, δ 为所有像素的标准差	沈占锋, 夏列钢, 李均力, 等. 采用高斯归一化水体指数实现遥感影像河流的精确提取[J]. 中国图象图形学报, 2013, 18(4): 421-428.	多光谱/高光谱 按波段输入
19	False NDWI (FNDWI)	$FNDWI = (FG - NIR) / (FG + NIR)$ $FG = G + S * (C_{NIR} - NIR)$ $C_{NIR} = 40, S=1$ (默认值)	周艺, 谢光磊, 王世新, 等. 利用伪归一化差异水体指数组提取城镇周边[J]. 地球信息科学学报, 2014, 16(1): 102-107.	多光谱/高光谱 按波段输入
20	Simple Ratio of Water index (SRWI)	$SRWI = G / MIR_1$	王晴晴, 余明. 基于简单比值型水体指数 (SRWI) 的水体信息提取研究[J]. 福建师范大学学报: 自然科学版, 2014, 30(1): 39-44.	多光谱/高光谱 按波段输入
21	Automated Water Extraction Index	$AWEI_{Insh} = 4 * (G - MIR_1) - (0.25 * NIR + 2.75 *$	Feyisa G L, Meilby H, Fensholt R, et al. Automated Water Extraction Index: A new	多光谱/高光谱 按波段输入

	(AWEI)	MIR ₂); AWEIsh = B + 2.5 * G – 1.5 * (NIR + MIR ₁) – 0.25 * MIR ₂	technique for surface water mapping using Landsat imagery[J]. Remote Sensing of Environment, 2014, 140: 23-35.	
22	Enhanced Water Index (EWI-w)	EWI-w = (G - MIR ₁ + m) / ((G + MIR ₁) * (NDVI + n)) m = 0.1, n = 0.5	Wang S, Baig M H A, Zhang L, et al. A simple enhanced water index (EWI) for percent surface water estimation using Landsat data[J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2015, 8(1): 90-97.	多光谱/高光谱 按波段输入
23	Water Index 2015 (WI2015)	WI2015 = 1.7204 + 171 * G + 3 * R – 70 * NIR – 45 * MIR ₁ - 71 * MIR ₂	Fisher A, Flood N, Danaher T. Comparing Landsat water index methods for automated water classification in eastern Australia[J]. Remote Sensing of Environment, 2016, 175: 167-182.	多光谱/高光谱 按波段输入
24	Weighted	WNDWI = (G - α)	Guo Q, Pu R, Li J, et al. A	多光谱/高光谱 按波段输入

	Normalized Difference Water Index (WNDWI)	$* \text{NIR} - (1 - \alpha) * \text{MIR}_1 / (\text{G} + \alpha * \text{NIR} + (1 - \alpha) * \text{MIR}_1)$ $\alpha \in [0.45, 0.55]$	weighted normalized difference water index for water extraction using Landsat imagery[J]. International journal of remote sensing, 2017, 38(19): 5430-5445.	光谱 按波段输入
25	Multi-band Water Index (MBWI)	$\text{MBWI} = 2 * \text{G} - \text{R} - \text{NIR} - \text{MIR}_1 - \text{MIR}_2$	Wang X, Xie S, Zhang X, et al. A robust Multi-Band Water Index (MBWI) for automated extraction of surface water from Landsat 8 OLI imagery[J]. International Journal of Applied Earth Observation and Geoinformation, 2018, 68: 73-91.	多光谱 / 高光谱 按波段输入
26	Water Index 2019 (WI2019)	$\text{WI2019} = (1.75 * \text{G} - \text{R} - 1.08 * \text{MIR}_1) / (\text{G} + \text{MIR}_1)$	黄远林, 邓开元, 任超, 等. 一种新的水体指数及其稳定性研究[J]. 地球物理学进展, 2020, 35(3): 829-835.	多光谱 / 高光谱 按波段输入
27	Augmented Normalized Difference Water Index (ANDWI)	$\text{ANDWI} = (\text{B} + \text{G} + \text{R} - \text{NIR} - \text{MIR}_1) / (\text{B} + \text{G} + \text{R} + \text{NIR} + \text{MIR}_1)$	Rad A M, Kreitler J, Sadegh M. Augmented Normalized Difference Water Index (ANDWI) for water extraction from multi-spectral遥感图像[J]. International Journal of Remote Sensing, 2018, 39(18): 6453-6468.	多光谱 / 高光谱

	Difference Water Index (ANDWI)	$= \frac{MIR_2}{(B + G + R + NIR + MIR_1 + MIR_2)}$	Difference Water Index for improved surface water monitoring[J]. Environmental Modelling & Software, 2021, 140: 105030.	按波段输入
--	--------------------------------	-----------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------	-------

表 3.3 建筑物纹理、颜色特征

28	Normalized Difference Building Index (NDBI)	$NDBI = \frac{(MIR_1 - NIR)}{(MIR_1 + NIR)}$	Zha Y, Gao J, Ni S. Use of normalized difference built-up index in automatically mapping urban areas from TM imagery[J]. International journal of remote sensing, 2003, 24(3): 583-594.	多光谱 / 按波段输入
29	Morphological Building Index (MBI)	(1) $b(x) = \max_{1 \leq k \leq K} (\text{band}_k(x))$ (2) $W - TH(d, s) = b - \gamma_b^{re}(d, s)$ (3) $\begin{cases} MP_{W-TH}(d, s) = W - TH(d, s) \\ MP_{W-TH}(d, 0) = b \end{cases}$	Huang X, Zhang L, A RG multidirectional and B multiscale	RG 影像

		$DMP_{W-TH}(d, s) = MP_{W-TH}(d, s) - MP_{W-TH}(d, s) $ (4) 式中: $S_{min} \leq \Delta s \leq S_{max}$ $(5) MBI = \frac{\sum_{d,s} DMP_{W-TH}(d,s)}{D \times S}$ $S = ((S_{max} - S_{min}) / \Delta s) + 1$; D 为计算建筑物剖面时的方向数。	morphological index for automatic extraction from multispectral imagery[J]. Photogrammetric Engineering & Remote Sensing, 2011, 77(7): 721-732.	像
3 0	Local Binary Pattern (LBP)	$LBP(x_c, y_c) = \sum_{p=1}^8 s(I(p) - I(c)) * 2^p$ $s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$	Ojala T, Pietikainen M, Harwood D. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions[C]//Proceedings of 12th international conference on pattern recognition. IEEE, 1994, 1: 582-585.	影像

3 1	Gray Level Co- occurrence Matrix (GLCM)	(1) $ASM = \sum_{i=1}^k \sum_{j=1}^k (G(i,j))^2$ (2) $CON = \sum_{n=0}^{k-1} n^2 (\sum_{ i-j =n} G(i,j))$ (3) $IDM = \sum_{i=1}^k \sum_{j=1}^k \frac{G(i,j)}{1+(i-j)^2}$ (4) $ENT = - \sum_{i=1}^k \sum_{j=1}^k G(i,j) \log G(i,j)$ (5) $COR = \sum_{i=1}^k \sum_{j=1}^k \frac{(ij)G(i,j) - u_i u_j}{s_i s_j}$	Haralick R M, Shanmugam K, Dinstein I H. Textural features for image classification[J]. IEEE Transactions on systems, man, and cybernetics, 1973 (6): 610-621.	M, RG B 影像
3 2	Gabor filter	(1) $g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi \frac{x'}{\lambda} + \psi\right)\right)$ (复数) (2) $g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$ (实数) (3) $g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \sin\left(2\pi \frac{x'}{\lambda} + \psi\right)$ (虚数) $x' = x \cos \theta + y \sin \theta$ $y' = -x \sin \theta + y \cos \theta$	Dunn D, Higgins W E, Wakeley J. Texture segmentation using 2-D Gabor elementary functions[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1994, 16(2): 130-149.	RG B 影像
3 3	GIST	$G_i(x, y) = \text{cat}[f(x, y) * g_{\text{msh}}(x, y)], (x, y) \in P_i$	Friedman A. Framing pictures: The role of knowledge in automatized encoding	RG B 影像

			and memory for gist[J]. Journal of experimental psychology: General, 1979, 108(3): 316.	
3 4	Histogram of Oriented Gradient (HOG)	$I(x, y) = I(x, y)^{\text{gamma}}$ $G_x(x, y) = H(x + 1, y) - H(x - 1, y)$ $G_y(x, y) = H(x, y + 1) - H(x, y - 1)$ $G(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$ $\alpha(x, y) = \tan^{-1}\left(\frac{G_y(x, y)}{G_x(x, y)}\right)$	Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). ieee, 2005, 1: 886-893.	RG B 影 像
3 5	RGB (HSV) Histogram	$p_x(i) = p(x = i) = \frac{n_i}{n}, 0 \leq i < L$ $cdf_x(i) = \sum_{j=0}^i p_x(j)$ (直方图归一化)	Acharya T, Ray A K. Image processing: principles and applications[M]. John Wiley & Sons, 2005.	RG B 影 像

表 3.4 SAR 影像指数

36	Polarization Salinity Index (PSI)	$PSI = \frac{\sigma_{HV}^0 - \sigma_{HH}^0}{\sigma_{HV}^0 + \sigma_{HH}^0}$	刘全明,成秋明,王学,李相君. 河套灌区土壤盐渍化微波雷 达反演 [J]. 农业工程学 报,2016,32(16):109-114.	RADARSAT- 2, etc
37	Radar Forest Degradation Index (RFDI)	$RFDI = \frac{\sigma_{HH}^0 - \sigma_{HV}^0}{\sigma_{HH}^0 + \sigma_{HV}^0}$	Ningthoujam R K. Retrieving Forest Characteristics from High- Resolution Airborne S- Band Radar Data[D]. University of Leicester, 2016.	Sentinel-1, etc
38	Biomass Index (BMI)	$BMI = \frac{\sigma_{HH}^0 + \sigma_{VV}^0}{2}$	Pope K O, Rey-Benayas J M, Paris J F. Radar remote sensing of forest and wetland ecosystems in the Central American tropics[J]. Remote Sensing of Environment, 1994, 48(2): 205-219.	Sentinel-1, etc

			Bouvet A, Mermoz S, Le Toan T, et al. An above-ground biomass map of African savannahs and woodlands at 25 m resolution derived from ALOS PALSAR[J]. Remote sensing of environment, 2018, 206: 156-173.	
39	Volume Scattering Index (VSI)	$VSI = \frac{\sigma_{HV}^0}{\sigma_{HV}^0 + BMI}$		ALOS PALSAR, etc
40	Canopy Structure Index (CSI)	$CSI = \frac{\sigma_{VV}^0}{\sigma_{HH}^0 + \sigma_{VV}^0}$	Haldar D, Patnaik C, Mohan S, et al. Jute and tea discrimination through fusion of SAR and optical data[J]. Progress In Electromagnetics Research B, 2012, 39: 337-354.	ALOS PALSAR, etc
41	Radar Vegetation Index (RVI)	$RVI = \frac{8\sigma_{HV}^0}{\sigma_{HH}^0 + \sigma_{VV}^0 + 2\sigma_{HV}^0}$	Kim Y, Jackson T, Bindlish R, et al. Radar vegetation index for estimating the vegetation water content of rice and soybean[J]. IEEE	ALOS PALSAR, etc

			Geoscience and Remote Sensing Letters, 2011, 9(4): 564-568.	
--	--	--	-------------------------------------------------------------	--

(1) RGB 影像

Python 端使用方法:

```
def test_eager_LBP():
    img = cv2.imread("../data/dataset/test_rs_index/RGB/building.tif")
    img = vision.LBP(N=0)(img)
    img = cv2.imwrite("../data/dataset/test_rs_index/test_results/building_lbp.tif", img)
```

C++端使用方法:

```
class MindDataTestLBPop : public UT::CVOP::CVOpCommon {
protected:
    MindDataTestLBPop() : CVOpCommon() {}
    std::shared_ptr<Tensor> output_tensor_;
};

TEST_F(MindDataTestLBPop, TestOp1) {
    MS_LOG(INFO) << "Doing testLBP.";
    std::unique_ptr<LBPop> op(new LBPop(3));
    EXPECT_TRUE(op->OneToOne());
    std::string folder_path = "/test_luojianet/luojianet1/luojianet/building.tif";
    // prepare 3-channel image
    cv::Mat input_img = cv::imread(folder_path);
    // create new tensor to test conversion
    std::shared_ptr<Tensor> test_input;
    std::shared_ptr<CVTensor> input_cv_tensor;
    CVTensor::CreateFromMat(input_img, -3, &input_cv_tensor);
    test_input = std::dynamic_pointer_cast<Tensor>(input_cv_tensor);

    Status s = op->Compute(test_input, &output_tensor_);
    std::shared_ptr<CVTensor> output_cv = CVTensor::AsCVTensor(output_tensor_);
    cv::Mat output_img = output_cv->mat();
    cv::imwrite("/test_luojianet/luojianet1/luojianet/building_lbp.tif", output_img);
    MS_LOG(INFO) << "testLBP end.";
    //EXPECT_EQ(s, Status::OK());
}
```

以上为 LBP 纹理计算的 Python 端和 C++ 端的调用方式。主要利用 OpenCV 读取 RGB 影像，调用 LBP 纹理计算，其中 N 表示 LBP 计算模式。0 表示原始 LBP 计算，1 表示圆形 LBP 计算，2 表示旋转不变 LBP，3 表示均匀模式 LBP (Uniform LBP)。该函数的输出为单通道的纹理特征。其余纹理、滤波、建筑物特征提取算子用法类似。

结果展示:

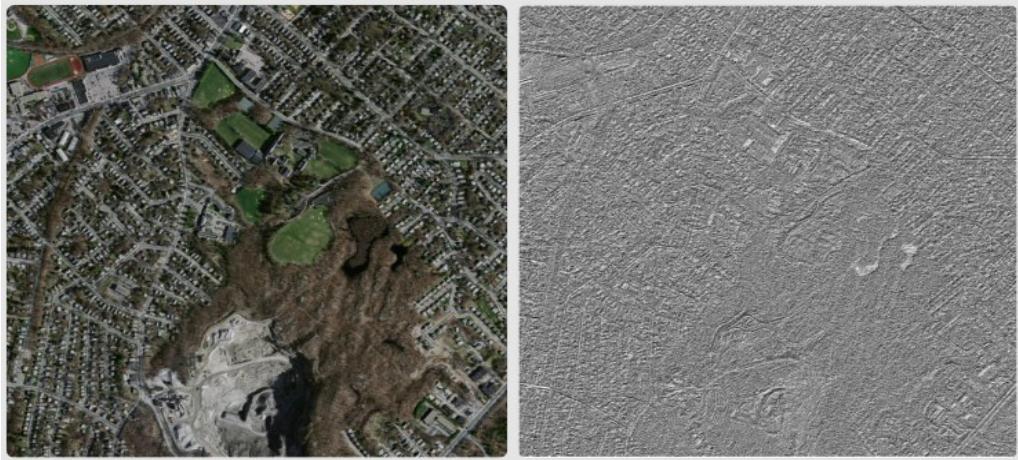


图 1 输入 RGB 影像，输出为单波段的 LBP 纹理特征（原始 LBP）

(2) 多光谱影像 (RGB+NIR)

Python 端使用方法：

```
def test_eager_NDVI():
    img = cv2.imread("../data/dataset/test_rs_index/MULTIBAND/temp.tif", cv2.IMREAD_UNCHANGED)
    img = vision.NDVI()(img)
    img = cv2.imwrite("../data/dataset/test_rs_index/test_results/temp_ndvi.tif", img)
```

C++端使用方法：

```
class MindDataTestNDVIOp : public UT::CVOP::CVOpsCommon {
protected:
    MindDataTestNDVIOp() : CVOpsCommon() {}

    std::shared_ptr<Tensor> output_tensor_;
};

TEST_F(MindDataTestNDVIOp, TestOp1) {
    MS_LOG(INFO) << "Doing testNDVI.";
    std::unique_ptr<NDVIOp> op(new NDVIOp());
    EXPECT_TRUE(op->OneToOne());

    std::string folder_path = "/test_luojianet/luojianet1/luojianet/temp.tif";
    // prepare 4 channel image
    cv::Mat input_img = cv::imread(folder_path, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
    // create new tensor to test conversion
    std::shared_ptr<Tensor> test_input;
    std::shared_ptr<CVTensor> input_cv_tensor;
    CVTensor::CreateFromMat(input_img, 3, &input_cv_tensor);
    test_input = std::dynamic_pointer_cast<Tensor>(input_cv_tensor);

    Status s = op->Compute(test_input, &output_tensor_);

    std::shared_ptr<CVTensor> output_cv = CVTensor::AsCVTensor(output_tensor_);
    cv::Mat output_img = output_cv->mat();
    cv::imwrite("/test_luojianet/luojianet1/luojianet/temp_c.tif", output_img);
    MS_LOG(INFO) << "testNDVI end.";
    //EXPECT_EQ(s, Status::OK());
}
```

以上为归一化植被指数(NDVI)的 Python 端和 C++端的调用方式。主要利用 OpenCV

读取 RGB+NIR 的多光谱影像，然后调用 NDVI 计算算子。该函数的输出为单通道的植被指
数特征。其余 RGB+NIR 的多光谱影像算子用法类似。

结果展示：

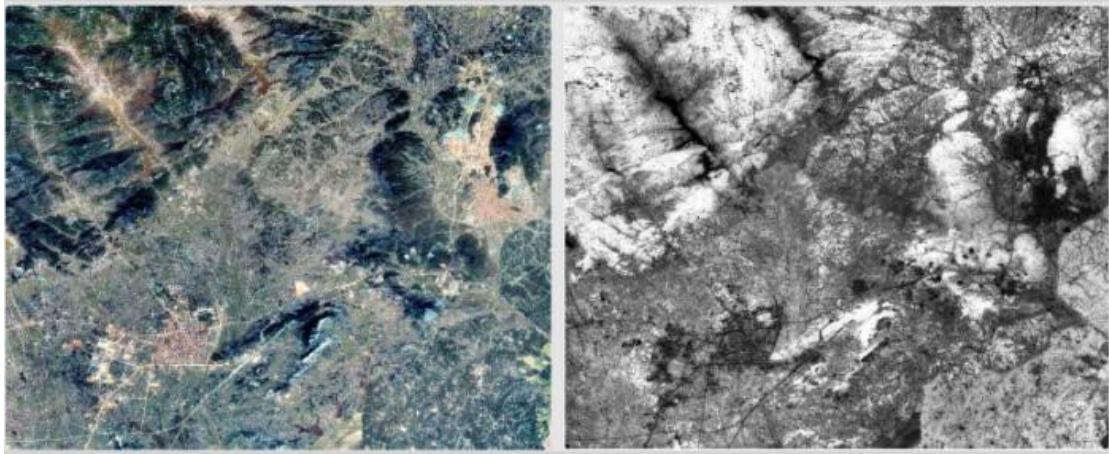


图 2 输入 RGB+NIR 的多光谱影像，输出为单波段的 NDVI 特征

(3) 高(多)光谱影像

Python 端使用方法：

```
def test_eager_ANDWI():
    band2 = cv2.imread("../data/dataset/test_rs_index/LANDSAT8/LC08_L1TP_122039_20131012_20170429_01_T1_B2.TIF",-1)
    band3 = cv2.imread("../data/dataset/test_rs_index/LANDSAT8/LC08_L1TP_122039_20131012_20170429_01_T1_B3.TIF",-1)
    band4 = cv2.imread("../data/dataset/test_rs_index/LANDSAT8/LC08_L1TP_122039_20131012_20170429_01_T1_B4.TIF",-1)
    band5 = cv2.imread("../data/dataset/test_rs_index/LANDSAT8/LC08_L1TP_122039_20131012_20170429_01_T1_B5.TIF",-1)
    band6 = cv2.imread("../data/dataset/test_rs_index/LANDSAT8/LC08_L1TP_122039_20131012_20170429_01_T1_B6.TIF",-1)
    band7 = cv2.imread("../data/dataset/test_rs_index/LANDSAT8/LC08_L1TP_122039_20131012_20170429_01_T1_B7.TIF",-1)
    img = cv2.merge([band2, band3, band4, band5, band6, band7])
    img = vision.ANDWI()(img)
    img = cv2.imwrite("../data/dataset/test_rs_index/test_results/land8_andwi.tif", img)
```

C++端使用方法：

```
class MindDataTestANDWIOp : public UT::CVOp::CVOpCommon {
protected:
    MindDataTestANDWIOp() : CVOpCommon() {}

    std::shared_ptr<Tensor> output_tensor_;
};

TEST_F(MindDataTestANDWIOp, TestOp1) {
    MS_LOG(INFO) << "Doing testANDWI";
    std::unique_ptr<ANDWIOP> op(new ANDWIOP());
    EXPECT_EQ(true, op->OneToOne());
}

std::string band2 = "/test_luojianet/luojianet/test_data/LC81220392013285LGN01/LC08_L1TP_122039_20131012_20170429_01_T1_B2.TIF";
std::string band3 = "/test_luojianet/luojianet/test_data/LC81220392013285LGN01/LC08_L1TP_122039_20131012_20170429_01_T1_B3.TIF";
std::string band4 = "/test_luojianet/luojianet/test_data/LC81220392013285LGN01/LC08_L1TP_122039_20131012_20170429_01_T1_B4.TIF";
std::string band5 = "/test_luojianet/luojianet/test_data/LC81220392013285LGN01/LC08_L1TP_122039_20131012_20170429_01_T1_B5.TIF";
std::string band6 = "/test_luojianet/luojianet/test_data/LC81220392013285LGN01/LC08_L1TP_122039_20131012_20170429_01_T1_B6.TIF";
std::string band7 = "/test_luojianet/luojianet/test_data/LC81220392013285LGN01/LC08_L1TP_122039_20131012_20170429_01_T1_B7.TIF";

cv::Mat input_band2 = cv::imread(band2, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
cv::Mat input_band3 = cv::imread(band3, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
cv::Mat input_band4 = cv::imread(band4, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
cv::Mat input_band5 = cv::imread(band5, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
cv::Mat input_band6 = cv::imread(band6, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
cv::Mat input_band7 = cv::imread(band7, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);

cv::Mat mergedc;
vector<cv::Mat> mergearc;
mergearc.push_back(input_band2);
mergearc.push_back(input_band3);
mergearc.push_back(input_band4);
mergearc.push_back(input_band5);
mergearc.push_back(input_band6);
mergearc.push_back(input_band7);
cv::merge(mergearc, mergedc);

// create new tensor to test conversion
std::shared_ptr<Tensor> test_input;
std::shared_ptr<CVTensor> input_cv_tensor;
CVTensor::CreateFromMat(mergedc, 3, &input_cv_tensor);
test_input = std::dynamic_pointer_cast<Tensor>(input_cv_tensor);

Status s = op->Compute(test_input, &output_tensor_);

std::shared_ptr<CVTensor> output_cv = CVTensor::AsCVTensor(output_tensor_);
cv::Mat output_img = output_cv->mat();
cv::imwrite("/test_luojianet/luojianet/temp_andwi.tif", output_img);
MS_LOG(INFO) << "testANDWI end.";
//EXPECT_EQ(s, Status::OK());
```

以上为增强的归一化水体指数 (ANDWI) 的 Python 端和 C++ 端的调用方式。主要利用 OpenCV 读取 Landsat8 的多光谱影像，然后调用 ANDWI 计算算子。该函数的输出为单通道的水体指数特征。其余的多光谱 (高光谱) 影像算子用法类似，高光谱可通过光谱特征选择、降维等操作获取相应波段作为输入即可。

结果展示：

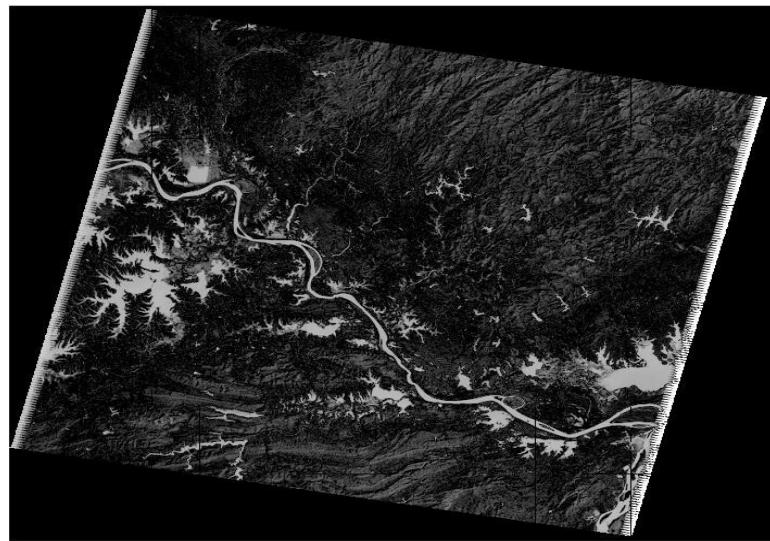


图 3 按波段输入多 (高) 光谱影像，输出为单波段的 ANDWI 特征

(3) SAR 影像

Python 端使用方法：

```
def test_eager_PSI():
    bandhh = cv2.imread("../data/dataset/test_rs_index/SAR/HHHV/sla-ew-grd-hh-20220117t144920-20220117t145019-041502-04ef79-001.tif",-1)
    bandhv = cv2.imread("../data/dataset/test_rs_index/SAR/HHHV/sla-ew-grd-hv-20220117t144920-20220117t145019-041502-04ef79-002.tif",-1)
    img = cv2.merge([bandhh, bandhv])
    img = vision.PSI()(img)
    img = cv2.imwrite("../data/dataset/test_rs_index/test_results/sar_PSI.tif", img)
```

C++ 端使用方法：

```

class MindDataTestPSIOP : public UT::CVOp::CVOpCommon {
protected:
    MindDataTestPSIOP() : CVOpCommon() {}

    std::shared_ptr<Tensor> output_tensor_;
};

TEST_F(MindDataTestPSIOP, TestOp) {
    MS_LOG(INFO) << "Doing test PSI.";
    std::unique_ptr<PSIOP> op(new PSIOP());
    EXPECT_TRUE(op->OneToOne());
    std::string bandhh ="/test_luojianet/luojianet1/luojianet/test_data/SAR/HHHV/sla-ew-grd-hh-20220117t144920-20220117t145019-041502-04ef79-001.tif";
    std::string bandhv ="/test_luojianet/luojianet1/luojianet/test_data/SAR/HHHV/sla-ew-grd-hv-20220117t144920-20220117t145019-041502-04ef79-002.tif";
    cv::Mat input_band1 = cv::imread(bandhh, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
    cv::Mat input_band2 = cv::imread(bandhv, cv::IMREAD_LOAD_GDAL | cv::IMREAD_COLOR | cv::IMREAD_ANYDEPTH);
    cv::Mat mergedst;
    vector<cv::Mat> mergesrc;
    mergesrc.push_back(input_band1);
    mergesrc.push_back(input_band2);
    cv::merge(mergesrc, mergedst);

    // create new tensor to test conversion
    std::shared_ptr<Tensor> test_input;
    std::shared_ptr<CTensor> input_cv_tensor;
    CTensor::CreateFromMat(mergedst, 3, &input_cv_tensor);
    test_input = std::dynamic_pointer_cast<Tensor>(input_cv_tensor);
    Status s = op->Compute(test_input, &output_tensor_);
    std::shared_ptr<CTensor> output_cv = CTensor::AsCTensor(output_tensor_);
    cv::Mat output_img = output_cv->mat();
    cv::imwrite("/test_luojianet/luojianet1/luojianet/temp_PSI.tif", output_img);
    MS_LOG(INFO) << "testPSI end.";
    //EXPECT_EQ(s, Status::OK());
}

```

以上为极化盐度指数 (PSI) 的 Python 端和 C++ 端的调用方式。主要利用 OpenCV 读取极化 SAR 的影像，然后调用 PSI 计算算子。该函数的输出为单通道的盐碱度特征。其余的极化 SAR 影像算子用法类似。

结果展示：

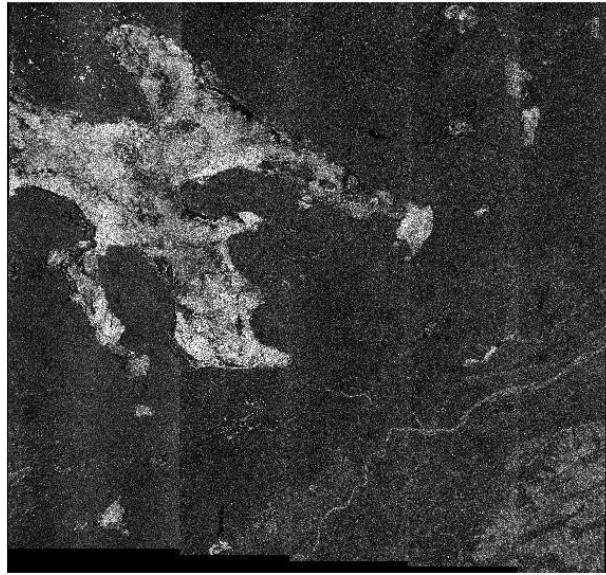


图 3 按波段输入极化 SAR 影像，输出为单波段的 PSI 特征

3.3.2 遥感先验知识自动并行

(1) 数据集 WHU-RS19 数据集

WHU-RS19 场景识别数据集影像来源于谷歌卫星影像上获取的遥感影像, 数据包含 19 个场景类。图像的输入尺寸为 600×600 像素, 共计 1005 张, 其中每个类别有 50 张。数据包含的主要区域为中国的城市地区。在实验验证时, 按照 3:1 的比例划分训练集与验证集, 包括训练集 804 张, 验证集 201 张。

(2) 实现细节

1) Python 端代码定义

在 `luojianet_ms/python/luojianet_ms/ops/operations/` 下, 到对应类型 `py` 文件, 在 `nn_ops.py` 定义了神经网络类型相关函数存储搬运函数:

```
class MemOP(Primitive):

    @prim_attr_register

    def __init__(self):
        """Initialize MemOP"""

        self.init_prim_io_names(inputs=['x'], outputs=['output'])
```

在 `luojianet_ms/python/luojianet_ms/ops/operations/_init_.py` 中对应的算子类别中添加算子名 `MemOP`。导入注册好的算子原语, 方便算子使用, 注意的是需要添加在对应的算子类别和 `_all_` 中两处地方添加算子名 `MemOP`。

2) C++端代码定义

在 `luojianet_ms/core/ops/*.h` 文件中声明算子类与接口函数文件 `mem_op.h` 与 `mem_op.cc`

`mem_op.h`

```
#ifndef LUOJIANET_MS_CORE_OPS_MEM_OP_H_
#define LUOJIANET_MS_CORE_OPS_MEM_OP_H_
```

```
#include <map>

#include <vector>

#include <string>

#include <memory>

#include "ops/primitive_c.h"

#include "ops/op_utils.h"

#include "abstract/abstract_value.h"

#include "utils/check_convert_utils.h"

namespace luojianet_ms {

namespace ops {

constexpr auto kNameMemOP = prim::kMemOP;

//constexpr auto kNameMemOP = "MemOP";

/// \brief Computes MemOP (Rectified Linear Unit activation function) of

input tensors element-wise.

/// Refer to Python API @ref luojianet_ms.ops.MemOP for more details.

class MS_CORE_API MemOP : public PrimitiveC {

public:

/// \brief Constructor.

MemOP() : PrimitiveC(kNameMemOP) { InitIOMap({ "x", "output" }); }

/// \brief Destructor.

~MemOP() = default;
```

```
MS_DECLARE_PARENT(MemOP, PrimitiveC);

/// \brief Init.

void Init() {}

};

} // namespace ops

} // namespace luojianet_ms

#endif // LUOJIANET_MS_CORE_OPS_MEM_OP_H_
```

mem_op.cc

```
#include "ops/mem_op.h"

#include <string>

#include <algorithm>

#include <map>

#include <set>

#include <vector>

#include "ops/op_utils.h"

#include "utils/check_convert_utils.h"

#include "abstract/primitive_infer_map.h"
```

```
namespace luojianet_ms {

namespace ops {

namespace {

abstract::ShapePtr InferShape(const PrimitivePtr &primitive, const
std::vector<AbstractBasePtr> &input_args) {

    MS_EXCEPTION_IF_NULL(primitive);

    auto prim_name = primitive->name();

    (void)CheckAndConvertUtils::CheckInteger("input numbers",
SizeToLong(input_args.size()), kGreaterEqual, 1, prim_name);

    (void)CheckAndConvertUtils::CheckArgs<abstract::AbstractTensor>(prim_name,
input_args, 0);

    auto x = input_args[0]->BuildShape();

    MS_EXCEPTION_IF_NULL(x);

    auto shape_element = x->cast<abstract::ShapePtr>();

    MS_EXCEPTION_IF_NULL(shape_element);

    return shape_element;

}

TypePtr InferType(const PrimitivePtr &prim, const
std::vector<AbstractBasePtr> &input_args) {

    MS_EXCEPTION_IF_NULL(prim);

    auto prim_name = prim->name();
```

```

(void)CheckAndConvertUtils::CheckInteger("input numbers",
                                         SizeToLong(input_args.size()), kEqual, 1, prim_name);

MS_EXCEPTION_IF_NULL(input_args[0]);

auto x_type = input_args[0]->BuildType();

(void)CheckAndConvertUtils::CheckTensorTypeValid("input_x", x_type,
                                                 common_valid_types, prim_name);

return x_type;

}

} // namespace

AbstractBasePtr MemOPIinfer(const abstract::AnalysisEnginePtr &, const
PrimitivePtr &primitive,

const std::vector<AbstractBasePtr> &input_args) {

auto type = InferType(primitive, input_args);

auto shape = InferShape(primitive, input_args);

return abstract::MakeAbstract(shape, type);

}

REGISTER_PRIMITIVE_EVAL_IMPL(MemOP, prim::kPrimMemOP, MemOPIinfer,
nullptr, true);

} // namespace ops

} // namespace luojianet_ms

```

在 `core/base/core_ops.h` 中添加:

```
constexpr auto kMemOP = "MemOP";  
  
inline const PrimitivePtr kPrimMemOP = std::make_shared<Primitive>(kMemOP);
```

core/ops/grad 中添加 mem_op_grad.h、mem_op_grad.cc:

mem_op_grad.h

```
#ifndef LUOJIANET_MS_CORE_OPS_MEME_OP_GRAD_H_
#define LUOJIANET_MS_CORE_OPS_MEME_OP_GRAD_H_

#include <map>

#include <vector>

#include <string>

#include <memory>

#include "ops/primitive_c.h"

#include "ops/op_utils.h"

#include "abstract/abstract_value.h"

#include "utils/check_convert_utils.h"

namespace luojianet_ms {

namespace ops {

constexpr auto kNameMemOPGrad = prim::kReLUGrad;

class MS_CORE_API MemOPGrad : public PrimitiveC {

public:

    MemOPGrad() : PrimitiveC(prim::kPrimMemOPGrad ->
```

```
{ InitOName({"x"}, {"output"}); }

~MemOPGrad() = default;

MS_DECLARE_PARENT(MemOPGrad, PrimitiveC);

void Init() {}

};

} // namespace ops

} // namespace luojianet_ms

#endif // LUOJIANET_MS_CORE_OPS_MEME_OP_GRAD_H_
```

mem_op_grad.cc

```
#include "ops/grad/mem_op_grad.h"

#include <string>

#include <algorithm>

#include <map>

#include <set>

#include <vector>

#include "abstract/param_validator.h"

#include "ops/op_utils.h"

#include "utils/check_convert_utils.h"

#include "abstract/primitive_infer_map.h"
```

```
namespace luojianet_ms {

namespace ops {

namespace {

abstract::ShapePtr InferShape(const PrimitivePtr &primitive, const
std::vector<AbstractBasePtr> &input_args) {

    MS_EXCEPTION_IF_NULL(primitive);

    auto prim_name = primitive->name();

    const int64_t input_num = 2;

    (void)CheckAndConvertUtils::CheckInteger("input number",
SizeToLong(input_args.size()), kEqual, input_num, prim_name);

    for (const auto &item : input_args) {

        MS_EXCEPTION_IF_NULL(item);

    }

    auto dout =
CheckAndConvertUtils::CheckArgs<abstract::AbstractTensor>(prim_name,
input_args, 0);

    auto out =
CheckAndConvertUtils::CheckArgs<abstract::AbstractTensor>(prim_name,
input_args, 1);

    abstract::CheckShapeSame(prim_name, out, dout);

    auto x = input_args[0]->BuildShape();
```

```
    MS_EXCEPTION_IF_NULL(x);

    auto shape_element = x->cast<abstract::ShapePtr>();

    MS_EXCEPTION_IF_NULL(shape_element);

    return shape_element;

}
```

```
TypePtr InferType(const PrimitivePtr &prim, const
std::vector<AbstractBasePtr> &input_args) {

    MS_EXCEPTION_IF_NULL(prim);

    auto prim_name = prim->name();
    const int64_t input_num = 2;
    (void)CheckAndConvertUtils::CheckInteger("input number",
SizeToLong(input_args.size()), kEqual, input_num, prim_name);
    MS_EXCEPTION_IF_NULL(input_args[0]);

    auto dout =
CheckAndConvertUtils::CheckArgs<abstract::AbstractTensor>(prim_name,
input_args, 0);

    auto out =
CheckAndConvertUtils::CheckArgs<abstract::AbstractTensor>(prim_name,
input_args, 1);

    (void)abstract::CheckDtypeSame(prim_name, out, dout);

    auto x_type = input_args[0]->BuildType();
```

```

MS_EXCEPTION_IF_NULL(x_type);

if (!x_type->isa<TensorType>()) {

    MS_EXCEPTION(TypeError) << "The " << prim_name << "'s "
                  << " input must be tensor type but got " <<
    x_type->ToString();

}

return x_type;

}

} // namespace

AbstractBasePtr MemOPGradInfer(const abstract::AnalysisEnginePtr &, const
PrimitivePtr &primitive,
const std::vector<AbstractBasePtr>
&input_args) {

auto type = InferType(primitive, input_args);

auto shape = InferShape(primitive, input_args);

return abstract::MakeAbstract(shape, type);

}

REGISTER_PRIMITIVE_EVAL_IMPL(MemOPGrad, prim::kPrimMemOPGrad,
MemOPGradInfer, nullptr, true);

} // namespace ops

} // namespace luojianet_ms

```

在 `core/base/core_ops.h` 中添加:

```
constexpr auto kMemOPGrad = "MemOPGrad";  
  
inline const PrimitivePtr kPrimMemOPGrad =  
  
    std::make_shared<Primitive>(kMemOPGrad);
```

gpu 端实现:

在 ccsrc\backend\kernel_compiler\gpu\nn 添加

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/nn/mem_op_grad_gpu_kernel.cc

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/nn/mem_op_grad_gpu_kernel.h

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/nn/mem_op_gpu_kernel.h

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/nn/mem_op_gpu_kernel.cc

backend/kernel_compiler/gpu/cuda_impl 添加:

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/cuda_impl/mem_op_impl.cuh

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/cuda_impl/mem_op_impl.cu

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/cuda_impl/mem_op_grad_impl.c

uh

luojianet_ms/ccsrc/backend/kernel_compiler/gpu/cuda_impl/mem_op_grad_impl.c

u

python 端 luojianet_ms/python/luojianet_ms/ops/_grad 添加反向:

luojianet_ms/python/luojianet_ms/ops/_grad/grad_nn_ops.py

```
@bprop_getters.register(P.MemOP)

def get_bprop_mem_op(self):

    """Grad definition for `ReLU` operation."""

    input_grad = G.MemOPGrad()

    def bprop(x, out, dout):
        dx = input_grad(dout, out)
        return (dx,)

    return bprop
```

luojianet_ms/python/luojianet_ms/ops/operations/_grad_ops.py

```
class MemOPGrad(Primitive):

    """Performs grad of MemOP operation."""

    @prim_attr_register

    def __init__(self):

        """Initialize MemOPGrad"""

        self.init_prim_io_names(inputs=['y_backprop', 'x'], outputs=['output'])
```

```
def __call__(self, y_backprop, x):
    raise NotImplementedError
```

并行计算添加：

luojianet_ms/ccsrc/frontend/parallel/auto_parallel/operator_costmodel.h

```
using TanhCost = SqrtCost;
using EluCost = SqrtCost;
using ReLUCost = SqrtCost;
using MemOPCost = SqrtCost;
using SigmoidCost = SqrtCost;
```

luojianet_ms/ccsrc/frontend/parallel/step_auto_parallel.cc

```
MEMOP
```

luojianet_ms/ccsrc/frontend/parallel/ops_info/ops_utils.h

```
constexpr char MEMOP[] = "MemOP";
```

luojianet_ms/ccsrc/frontend/parallel/ops_info/ops_info_head_files.h

```
#include "frontend/parallel/ops_info/mem_op_info.h"
```

luojianet_ms/ccsrc/frontend/parallel/dynamic_creator.h

```
REGISTER(MemOPIInfo);
```

luojianet_ms/ccsrc/frontend/parallel/ops_info/mem_op_info.h

luojianet_ms/ccsrc/frontend/parallel/ops_info/mem_op_info.cc

luojianet_ms/ccsrc/frontend/parallel/auto_parallel/rec_core/rec_parse_graph.h

```
{TANH, OperatorType::kRecReLU},  
  
{MEMOP, OperatorType::kRecReLU},
```

3) 在 LuoJiaNET 上采用 ResNET-50 网络。在进行训练时，每个样本大小保持原始比例，随机裁剪为 224x224 像素，以适应残差网络输入影像尺寸。采用的损失函数为 cross entropy 标签平滑损失，优化器选择为 SGD，初始学习率设置为 0.1，最大学习率为 0.1，采用 warm-up 训练策略，批次数设置为 64，训练时采用 fp32 浮点精度训练，共 1600 个轮次直至收敛。遥感先验知识采用 GLCM 灰度共生矩阵特征作为先验知识，与归一化后的影像堆叠为 4 通道的影像作为残差网络输入。训练环境为 Tesla V100，32Gb 显存，8 张显卡并行训练。在进行推理时，每个样本保持原始大小。

(3) 实验结果

实验对比三种结果，分别为基准网络（ResNET-50）、加入先验知识(+GLCM)的常规并行策略的网络（+AutoParallel）、加入 GLCM 与显存搬运（MemOP）后的分布式并行网络（+GLCM）。

表 1 精度结果对比

Methods	top1-acc	top5-acc
ResNET-50	0.865	0.985
+AutoParallel(+GLCM)	0.895	0.980
+AutoParallel(+GLCM+MemOP)	0.870	0.990

从表 1 能看到，融入 GLCM 遥感先验知识后，可以使 top-5 精度保持相当的情况下，

top-1 精度提升约 3%，验证了遥感先验知识融合的有效性；在增加 GLCM 同时采用显存搬运的并行策略，使 top-5 精度提升至 0.99，top-1 精度提升至 0.87。与基线网络 ResNET-50 相比，表明设计的遥感先验知识自动并行策略能显著提升网络精度，达到了预期目标。

4. LuoJiaNET 典型应用模型

4.1 场景分类

4.1.1 任务简介

(1) 使用 LuoJiaNET 完成经典深度学习场景分类网络 VGG、ResNet、SENet 的搭建与训练，将 LuoJiaNET 框架应用到遥感影像场景分类任务中；
(2) 将 LuoJiaNET 和主流的深度学习框架 PyTorch、TensorFlow 进行对比，测试 LuoJiaNET 在场景分类任务中的性能表现。

4.1.2 分类网络简介

➤ VGG

VGG 是 Oxford 的 Visual Geometry Group 的组提出。该网络是在 ILSVRC 2014 上的相关工作，主要工作是证明了增加网络的深度能够在一定程度上影响网络最终的性能。VGG 有两种结构，分别是 VGG16 和 VGG19，两者并没有本质上的区别，只是网络深度不一样。

VGG 网络是第一个在每个卷积层使用更小的 3×3 卷积核对图像进行卷积，并把这些小的卷积核排列起来作为一个卷积序列，仅使用多个 3×3 卷积核可以模仿较大卷积核那样对图像进行局部感知。因此其能在分类任务上取得较好的效果。

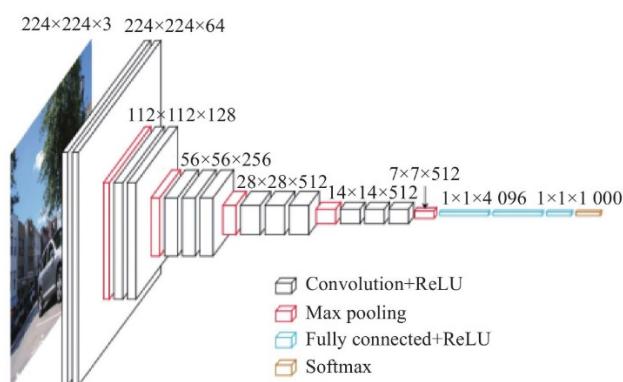


图 2 VGG 网络图

➤ ResNet

ResNet 网络是参考了 VGG19 网络，在其基础上进行了修改，并通过短路机制加入了残差单元，如图 3 所示。变化主要体现在 ResNet 直接使用 $\text{stride}=2$ 的卷积做下采样，并且用 global average pool 层替换了全连接层。ResNet 的一个重要设计原则是：当 feature map 大小降低一半时，feature map 的数量增加一倍，这保持了网络层的复杂度。从图 3 中可以看到，ResNet 相比普通网络每两层间增加了短路机制，这就形成了残差学习，其中虚线表示 feature map 数量发生了改变。图展示的 34-layer 的 ResNet，还可以构建更深的网络如 ResNet50, ResNet101, ResNet152。当网络更深时，其进行的是三层间的残差学习，三层卷积核分别是 1×1 , 3×3 和 1×1 ，一个值得注意的是隐含层的 feature map 数量是比较小的，并且是输出 feature map 数量的 $1/4$ 。

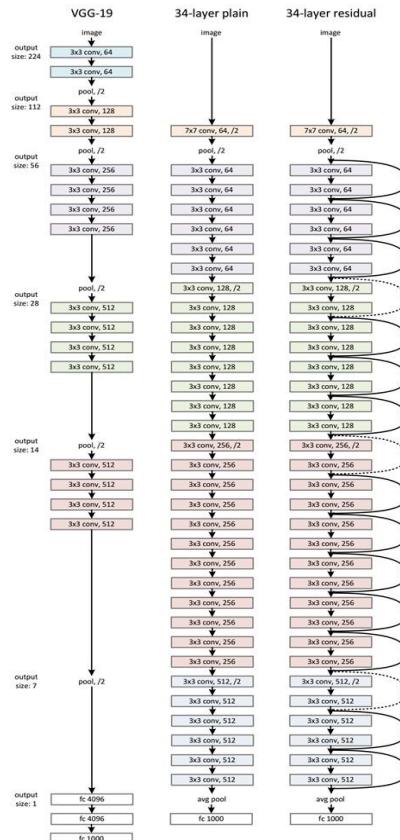


图 3 ResNet 网络图

➤ SENet

SENet 是一个非常经典的分类网络，它通过构建 SEblock 实现特征的通道选择，且 SEblock 能够嵌入到现有的任何全卷积网络中，实现特征在通道维度的筛选和增强。

SENet 主要由**特征提取模块、SEBlock 以及分类器**构成。SEBlock 即为 Squeeze-and-Excitation Block，该模块通过利用全连接层以及 sigmoid 生成每个通道的权重来对通道进行增强和筛选，输入的特征首先通过挤压和重采样操作，将特征图在空间维度 H, W 上聚集起来，产生一个通道描述符。这个描述符包含了通道特征反应的全局分布，使得全局信息得到充分的利用。接下来是一个激励操作，在这个过程中，每个通道的权重都是基于门控机制来学习的。分类器即为最终的全连接层，与 VGG、ResNet 等常见的网络一致。

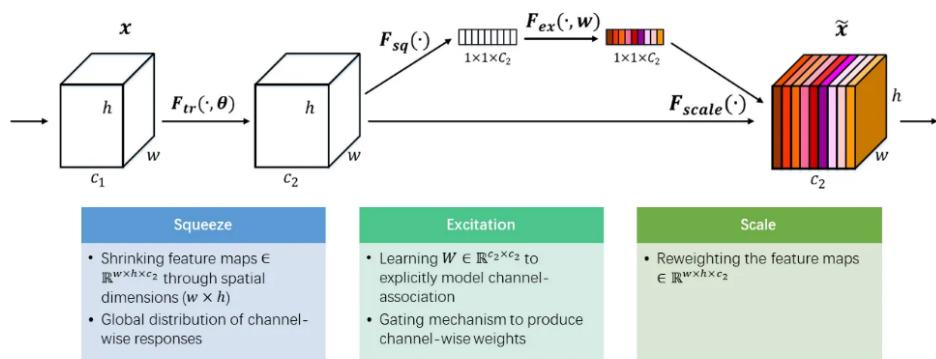


图 4 SEBlock 示意图

4.1.3 测评结果

(1) 数据集 PatternNet 数据集

PatternNet 由武汉大学于 2018 年发布，是一个用于遥感图像分类的大规模高分辨率遥感数据集。该数据集共包含有 38 个类，每个类有 800 张大小为 256×256 像素的图像。该数据集质量高，截止目前为止已有 215 个引用，超过了其余遥感影像分类数据集的引用量。我们从该数据集中每个类别随机选择 50 张作为测试集对模型精度进行验证和评价。

(2) 实现细节

我们分别在主流的深度学习框架 PyTorch 和我们的 LuoJiaNET 上复现了上面三种分类网络，在后文中分别简写为 VGG、ResNet、ResNet_SE。为了进行公平的对比，网络均在同一硬件上进行训练和测试。GPU 型号为 GeForce GTX 3090 (24G)、CPU 型号为 Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz。使用的 pytorch 框架为 v1.8.1，LuoJiaNET 版本为 v1.0.0，TensorFlow 版本为 v2.4.0。

在进行训练时，每个样本大小为 256×256 像素的小块，采用的损失函数为 cross entropy 损失，优化器选择为 SGD，初始学习率设置为 $1e-4$ ，最大学习率为 0.1，采用 warm-up 训练策略，避免过拟合，批次数设置为 8，训练共 200 个轮次，共计约 7.5 万次迭代。在进行推理时，每个样本保持原始大小。

(3) 评价指标

在实验中，采用的定量指标如下：

a. 准确率 (Precision)

统计分类的准确率，计算方式如下：

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

其中， TP 为预测正确的类别， FP 为预测错误的类别。

b. 召回率 (Recall)

统计分类结果占全部准确结果的百分比，计算方式如下：

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

其中， TP 表示预测正确的类别， FN 表示误检测的类别。

c. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要的显存占用。

d. 运行时间 (Time)

记录了模型在训练或者推理过程中处理一个样本所需要的时间。

(4) 实验结果

由于每个分类网络所包含的子集众多, 所以这里仅选择使用最广泛的网络进行对比测试。其中 LuojaNet 表示为 LuojaNET, TensorFlow 表示为 TF, pytorch 表示为 Pytorch。

表 2 精度结果对比

Methods	Precision	Recall	top1-acc	top5-acc
VGG16 (LuojaNET)	0.894	0.896	0.918	0.955
VGG16(TF)	0.891	0.890	0.902	0.950
VGG16 (PyTorch)	0.891	0.892	0.903	0.952
ResNet50 (LuojaNET)	0.941	0.960	0.967	0.979
ResNet50(TF)	0.937	0.961	0.965	0.977
ResNet50 (PyTorch)	0.940	0.959	0.965	0.977
ResNet50_SE (LuojaNET)	0.976	0.962	0.983	0.997
ResNet_SE(TF)	0.969	0.967	0.981	0.995
ResNet50_SE (PyTorch)	0.970	0.964	0.981	0.994

从表 1 能看到, 相比于 PyTorch 以及 TensorFlow 复现的网络, 使用 LuoJaNET 复现的网络在精度上有略微提升, 提升不明显的原因可能是所使用的数据集相对简单。

表 3 时间与显存占用对比

Methods	Training	Inference

	Memory (MB)	Time (ms)	Memory (MB)	Time (ms)
VGG16 (LuojiaNET)	5621	51.9	3517	24.9
VGG16(TF)	7523	58.3	3715	25.5
VGG16 (PyTorch)	6313	63.2	3673	26.3
ResNet50 (LuojiaNET)	12757	266.1	4285	93.9
ResNet50(TF)	15431	300.3	4703	90.3
ResNet50 (PyTorch)	13431	350.7	4610	94.2
ResNet50_SE (LuojiaNET)	18931	420.7	4539	143.5
ResNet50_SE(TF)	21345	507.8	6741	145.2
ResNet50_SE (PyTorch)	20257	532.5	6756	167.8

从表 2 可以看出，在训练时，模型越复杂（表现在网络的层数、是否使用注意力等方面）LuojiaNet 与其余两种框架下的结果差距越明显，且这个差距随模型复杂程度增大而增大，原因可能是 LuoJiaNet 在拟合网络参数时所保留的参数精度与另外两种不同，该原因也会影响网络的精度；在测试时，由于不存在梯度回传，所以差距较小；因此猜测训练时显存和时间差距主要由梯度回传修正网络模型参数时产生。

4.1.4 总结

基于 LuoJiaNET 框架，已经完成了遥感影像分类网络的典型应用。相比于主流深度学习框架 PyTorch、TensorFlow，LuoJiaNET 在训练过程中的显存占用低，在推理速度上也略优于另外两种。同时当使用相同的数据以及网络模型进行测试时，LuojiaNet 框架下的网

络精度略优于另外两种，造成这种现象的原因可能是 LuoJiaNet 在进行训练与推理时模型参数保存的精度较高。

4.2 目标检测

4.2.1 任务简介

- (1) 使用 LuoJiaNET 完成目标检测深度学习网络 Faster-RCNN、Mask-RCNN 的搭建、训练与推理实验，将 LuoJiaNET 框架应用到遥感影像目标检测任务中；
- (2) 测试 LuoJiaNET 在目标检测任务中的性能表现，并将 LuoJiaNET 和主流的深度学习框架 PyTorch 进行对比，测试在精度、训练时间、显存消耗方面的表现。

4.2.2 目标检测网络简介

➤ Faster-RCNN 目标检测网络

Faster-RCNN 是 Kaiming He 等人在 2016 年提出的端到端的两阶段目标检测算法，也是目前落地最成功的深度学习模型之一，是目标检测领域最经典的模型之一。Faster-RCNN 将目标检测任务分成了两个阶段，首先第一阶段，利用深度网络找出图像中可能存在物体的区域，产生 Region Proposal；第二阶段，对 Region Proposal 内的物体进行分类，并对第一阶段检测的 anchor 框进行回归。网络损失主要有三部分构成，包括 RPN 分类、回归损失，以及 Bounding Box Head 和 Classification Head 的损失。两阶段目标检测算法相比于 Yolo 系列等一阶段目标检测算法具有检测精度更高的优势，但是相对来说计算效率相对较低。

论文地址：<https://arxiv.org/abs/1506.01497>

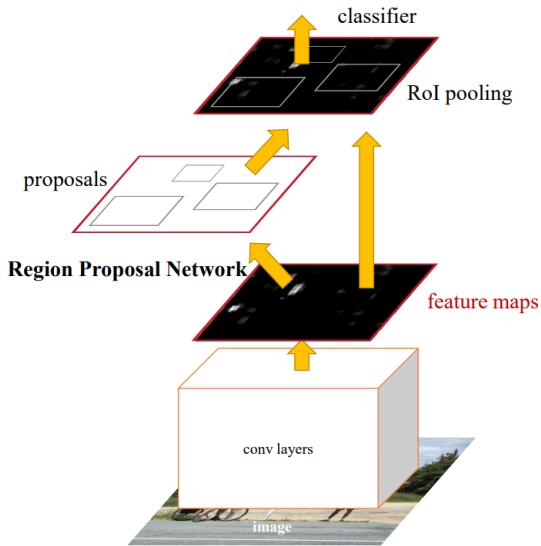


图 1 Faster-RCNN 目标检测网络结构图

➤ Mask-RCNN 实例分割网络

Mask-RCNN 是 Kaiming He 等人在 2018 年提出的端到端的两阶段实例分割算法，该方法是在 Faster-RCNN 的基础上添加了一个预测分割 mask 的分支以及对应的损失函数，并对 Faster-RCNN 做出了一些改进，实现了对图像中检测到的目标实例进行分割的目的。基于这个思想，可以借助该模型可以进一步实现姿态检测等任务。同时，实验证明，该方法通过利用多任务学习的思想，有助于提升目标检测的效果。该网络模型的主要结构如下图所示：

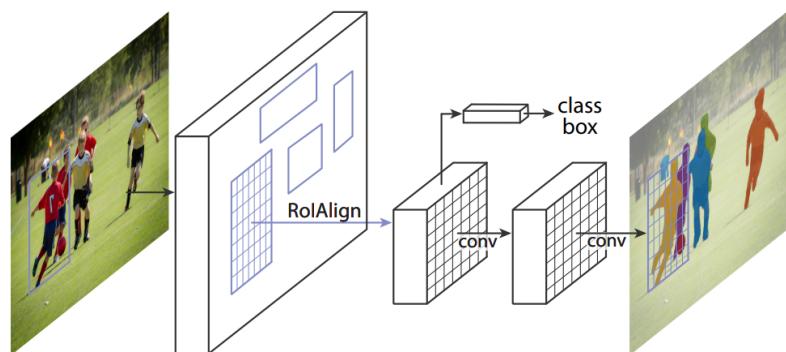


图 2 Mask-RCNN 目标检测网络结构图

本实验中，利用 Mask-RCNN 实现对旋转目标的检测，并与 pytorch 的结果进行对比。

论文地址如下：<https://arxiv.org/pdf/1703.06870.pdf>

4.2.3 测评结果

(1) 数据集：天智杯可见光图像飞机智能检测识别数据集

该数据集是慧眼“天智杯”人工智能挑战赛——可见光图像飞机智能检测识别赛道提供的训练和验证数据集。数据来源于国产自主产权系列卫星影像，图像内容主要是多种成像条件下的机场可见光图像，包含 611 幅图像，其中训练集 308 幅，验证集 122 幅，测试集 181 幅（不开放下载），包含 11 类目标，约 13000 个飞机样本。数据集影像的地面分辨率约为 0.5-1m。其中每组数据包含一幅飞机遥感图像，以及对应的飞机标签，即坐标和类别信息。图像为 4096x4096 尺寸的 png 格式。标签为 json 格式，以旋转框的方式对飞机进行标注。

官网地址：<https://www.rsaicp.com/portal/contestDetail?id=2&tab=data>

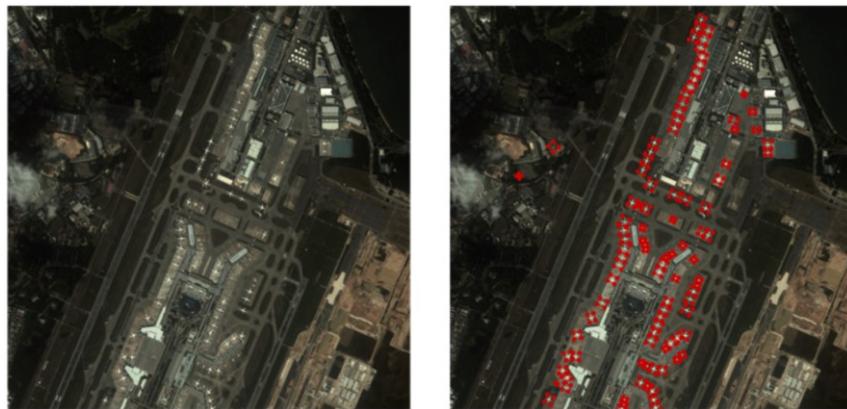


图 3 数据集图像示例

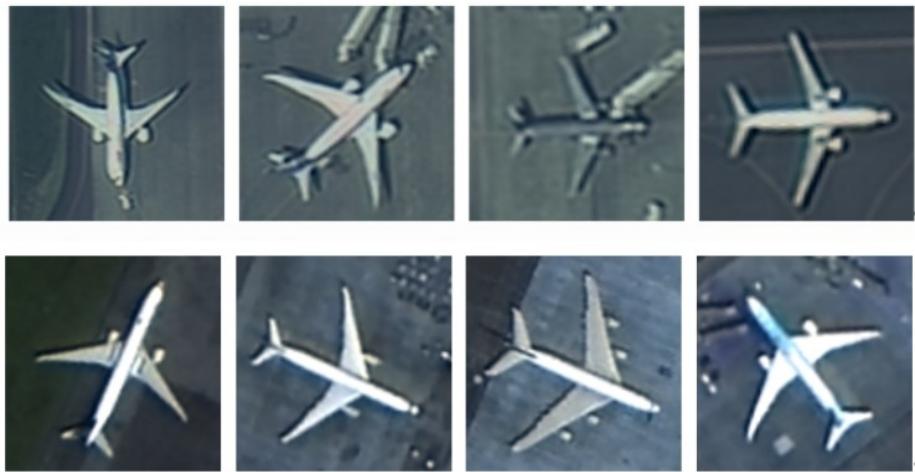


图 4 检测目标 (其中一类飞机) 示例

(2) 实现细节

PyTorch 版本: v1.10.1

LuoJiaNET: v1.0.0

GPU 型号: GeForce RTX 3090 (24G)

CPU 型号: Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz

数据集大小: 1024*1024 (4506 train) + 4096*4096 (122 val), 使用多尺度训练

优化器: Momentum

学习率: 0.02 (warmup + cosine decay)

批大小: 2

训练轮次: 50

补充说明:

目前广泛使用的目标检测网络都是基于现有的开源目标检测框架进行实现的, 常用的基于 Pytorch 的目标检测框架有 detectron2 (Facebook 团队开发维护), mmdetection(香港中文大学-商汤联合实验室开发维护)等。本项目实验中 Pytorch 下目标检测网络实现是基于 mmdetection 进行实现的, 具体内容可参考:

<https://zhuanlan.zhihu.com/p/96931265>。

(3) 评价指标

在本实验中，采用的定量评价指标如下所示：

a. 精度评价指标(基于混淆矩阵)

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

$$F1 = \frac{(Precision \times Recall \times 2)}{(Precision + Recall)} \quad (3)$$

$$mF1 = \frac{\sum_i F1_i}{N} \quad (N\text{表示总类别数}) \quad (4)$$

b. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要的显存占用。

c. 运行时间 (Time)

记录了模型在训练或者推理过程中处理一个样本所需要的时间。

(4) 实验结果

➤ Faster-RCNN

实验中，初步使用水平目标框，仅对飞机进行目标检测（不进行细粒度分类），下面是使用多尺度训练的结果。

表 4 Faster-RCNN 飞机目标检测精度结果对比

Methods	Precision	Recall	mF1
Faster-RCNN	92.05	91.44	91.75

(LuoJiaNET)			
Faster-RCNN (PyTorch)	89.04	91.27	90.15

从表 1 能看到，相比于 PyTorch 复现，在置信度为 0.7，IoU 为 0.5 的条件下，使用 LuoJiaNET 复现的结果在精确度以及 mF1 上具有一定优势，召回率上两者精度相当。综合来看，LuoJiaNet 在 mF1 数值上相比于 Pytorch 更好，其主要原因在于检测精度上的提高。

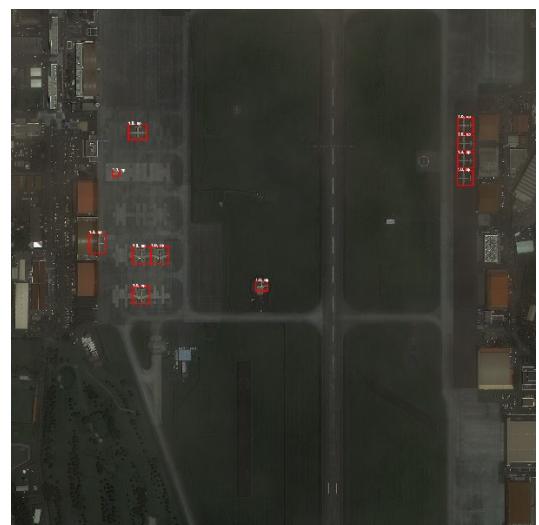
表 5 Faster-RCNN 飞机目标检测时间、显存占用对比

Methods	Training	Inference		
	Memory (MiB)	Time (s)	Memory (MiB)	Time (s)
Faster-RCNN (LuojiaNET)	16511	0.556	3513	2.64
Faster-RCNN (PyTorch)	16295	0.505	2963	5.46

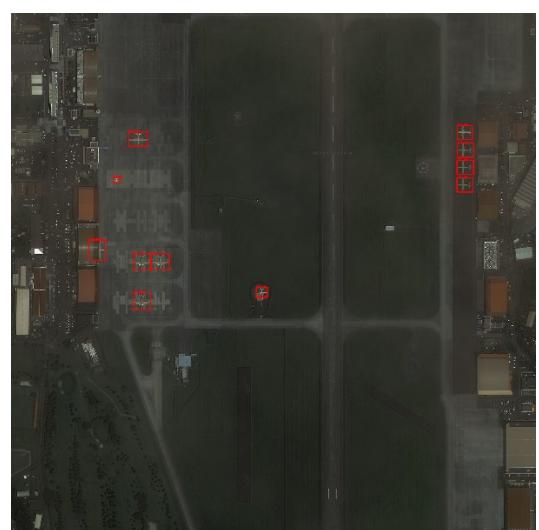
从表 2 可以看出，使用 LuoJiaNET 进行训练，在显存上相比 Pytorch 差距不大。单卡训练情况下，两者在处理时间上差异不大，LuoJiaNET 迭代相对时间略长。（注：由于对大幅面图像采取了切分再检测的策略以检测小目标，这里的时间仅计算了对裁剪后的图像进行前向推理的时间）推理占用显存上，LuoJiaNET 比 Pytorch 多 500M 左右，但是速度相对更快。



(a) Ground Truth



(b) Pytorch 检测结果



(c) LuoJiaNET 检测结果

图 5 Faster-RCNN 检测结果可视化

➤ **Mask-RCNN**

实验中，使用 Mask-RCNN 实现对旋转目标的检测，具体而言是通过将旋转目标框作为掩膜进行预测，并将掩膜的最小外接矩形作为最终的旋转目标检测结果。需要注意的是，这里仅对飞机进行目标检测（不进行细粒度分类）。

表 3 Mask-RCNN 飞机目标检测精度结果对比

Methods	Precision	Recall	mF1
HBB Result			
Mask-RCNN (LuoJiaNET)	93.51	91.00	92.24
OBB Result			
Mask-RCNN (LuoJiaNET)	92.79	90.30	91.53
Mask-RCNN (PyTorch)	89.72	91.49	90.59
Mask-RCNN (PyTorch)	89.03	90.79	89.90

从表 3 能看到，相比于 PyTorch 复现，在置信度为 0.7，IoU 为 0.5 的条件下，使用 LuoJiaNET 复现的结果在精确度以及 mF1 上具有一定优势，召回率上比 pytorch 结果略低。综合来看，LuoJiaNet 在 mF1 数值上相比于 Pytorch 更好，其主要原因在于检测精度上相比 pytorch 结果显著更好。同时，该结果与 Faster-RCNN 上的实验结论是一致的。

表 4 Mask-RCNN 飞机目标检测时间、显存占用对比

Methods	Training		Inference	
	Memory (MiB)	Time (s)	Memory (MiB)	Time (s)
Mask-RCNN (LuoJiaNET)	18559	1.901	5561	4.76
Mask-RCNN (PyTorch)	16121	0.601	2985	6.56

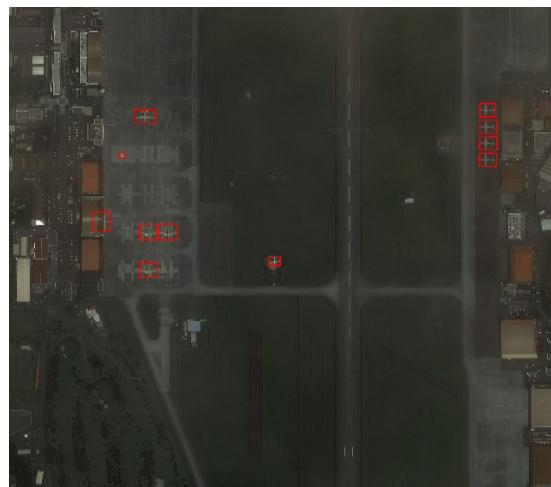
从表 4 可以看出，使用 LuoJiaNET 进行训练，Mask-RCNN 在显存上相比与 Pytorch 要多 2.5G 左右；同时，在处理时间上，单卡训练情况下，Mask-RNN 所需时间更长，原因是由于实现造成的差异，在训练数据标注的处理上与 mmdetection 有所不同（为了将同一个 batch 中的不同图像中实例数量处理成一致的以进行批迭代，增加了冗余的分割实例，具体而言将每张图的标注个数增加至 128 个实例，多余的标注使用掩膜进行标注）。推理占用显存上，LuoJiaNET 比 Pytorch 多 2.5G 左右，一部分可能是由于网络本身预测时的数据处理带来的（与训练阶段显存占用较多的原因一致），另一部分可能是由于框架本身造成的（在 Faster-RCNN 实验中，LuoJiaNET 相比 Pytorch 要占用的显存更多），但是 LuoJiaNET 在推理速度上相对 Pytorch 更快。



(a) Ground Truth



(b) Pytorch 检测结果



(c) LuoJiaNET 检测结果

图 6 Mask-RCNN 检测结果可视化

➤ **Mask-RCNN (DOTA_v2.0 数据集补充实验)**

实现细节：

PyTorch 版本：v1.10.1

LuoJiaNET：v1.0.0

GPU 型号：GeForce RTX 3090 (24G) * 2

CPU 型号：Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz

数据集大小：训练集 13135 张图像(1024*1024)，测试集为 DOTA2.0 的验证集

(593 张不同尺寸的图像)，使用多尺度训练

优化器：SGD

学习率：0.01 (warmup + step_decay)

批大小：2/GPU

训练轮次：30

该实验是为了比较 LuoJiaNet 与 PyTorch 之间在大规模数据集下的性能，以使结论更

具有说服力，具体实验结果如下：

表 5 Mask-RCNN 飞机目标检测精度结果对比(AP/mAP)

	LV	SP	HC	BR	PL	SH	SBF	BC	AP	CC	GTF	SV	HA	BD	TC	RA	ST	HP	mA P
HBB Result																			
luojiaN	66.8	61.0	54.4	40.8	87.0	82.4	61.6	62.6	11.3	6.19	70.7	46.5	71.9	67.7	93.9	58.9	59.0	0.0	55.7
et	6	1	0	9	4	2	0	5	4		0	1	7	8	4	5	2		4
Pytorch	66.6	58.7	53.4	42.4	85.8	81.6	63.5	60.7	20.7	1.46	69.0	46.0	72.8	63.6	93.2	60.7	58.6	0.0	55.5
	7	8	0	4	7	7	0	8	8		2	7	0	0	1	4	0		2

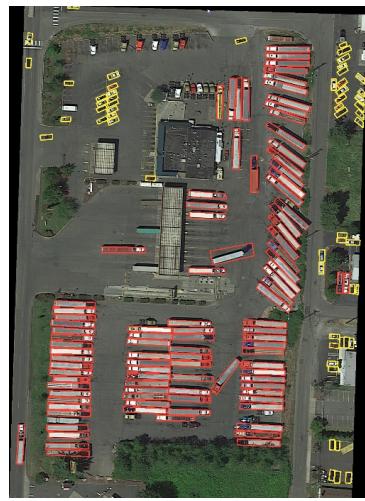
		OBB Result																	
		AP		AR@0.1		AR@0.5		AR@0.75		IoU=0.5		IoU=0.75		mAP		mAP _{0.5:0.95}		mAP _{0.5:0.95}	
		Recall		Precision		Recall		Precision		Recall		Precision		Recall		Precision		Recall	
luojiaN et	57.1 5	54.1 2	54.4 0	35.8 7	86.7 4	79.5 6	60.5 8	61.4 8	9.83 0.48	70.3 0	43.1 2	64.1 6	67.4 7	93.2 2	58.9 5	58.7 9	53.1 0.0	0.0 2	
Pytorch	57.2 6	54.1 6	56.3 1	39.4 3	85.8 7	78.4 3	62.6 2	59.2 7	20.7 8	0.89 0	69.4 6	42.3 2	66.0 7	66.0 4	92.8 7	58.3 4	60.1 7	0.0 0	53.9 0

从表 5 能看到，相比于 PyTorch 复现，置信度为 0.2(一般论文中测试 DOTA 数据集采用的置信度为 0.1 或 0.2)、IoU 为 0.5 的条件下，使用 LuoJiaNET 水平框 AP 结果在 12 类(总共 18 类)上的比 PyTorch 高，5 类上的结果比 PyTorch 低，mAP 比 PyTorch 高 0.22；旋转框 AP 结果在 8 类上比 PyTorch 高，9 类上的比 PyTorch 低，mAP 比 PyTorch 低 0.78。从整体结果上看，使用 luojanet 和 pytorch 的结果差异并不显著，可以近似认为两者检测结果精度接近。

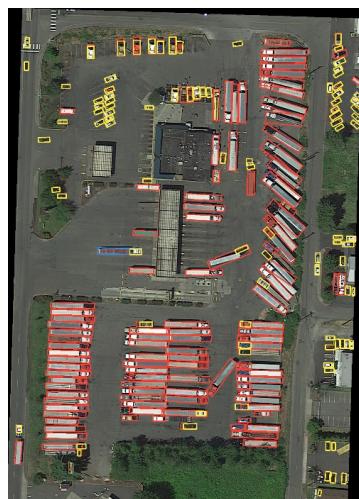
表 6 Mask-RCNN 目标检测时间、显存占用对比

Methods	Training		Inference	
	Memory (MiB)	Time (s)	Memory (MiB)	Time (s)
Mask-RCNN (LuoJiaNET)	10393/GPU	2.68	10681	5.23
Mask-RCNN (PyTorch)	7242/GPU	0.65	4027	5.63

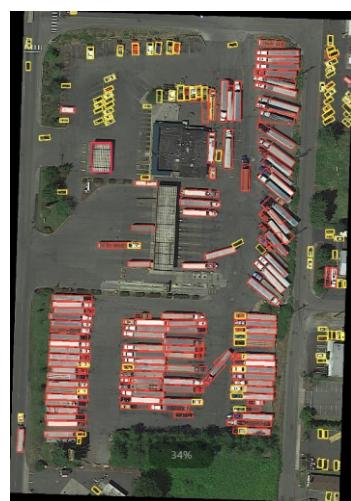
从表 6 的结果得到的结论与在天智杯数据集上得到的结论相近(注：由于对大幅面图像采取了切分再检测的策略以检测小目标，这里的时间仅计算了对裁剪后的图像进行前向推理的时间)。



(a) Ground Truth



(b) Pytorch 检测结果



(c) LuoJiaNET 检测结果

图 7 Mask-RCNN DOTA_v2.0 数据集检测结果可视化

➤ 细粒度目标检测模型

实现细节：

LuoJiaNET: v1.0.0

GPU 型号: GeForce RTX 3090 (24G)

CPU 型号: Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz

数据集大小： 训练集：根据天智杯训练集标注文件中给出的边界框数据从原图中裁剪出来的各类飞机图像(11类, 6793张) + 使用LuojiaNet训练得到的Mask-RCNN模型预测训练集时将背景错误识别为飞机的背景图像(1类, 134张, 有筛选), 共12类6927张。

优化器：Adam

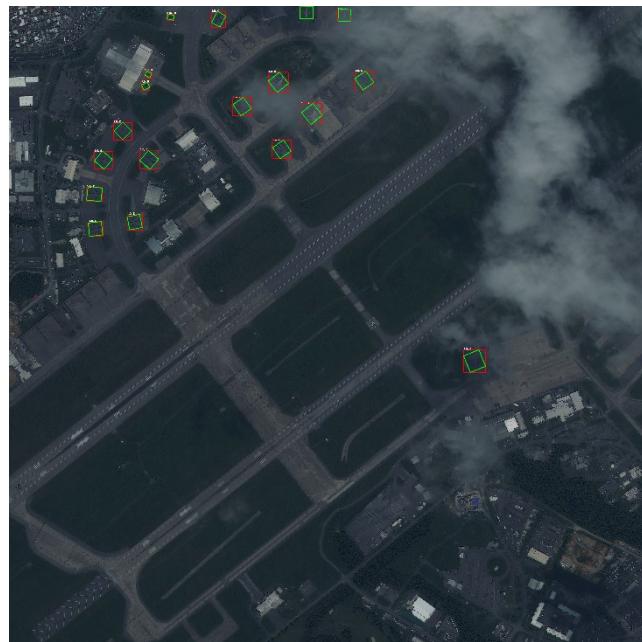
学习率: 0.0001 (cosine_decay)

批大小: 128

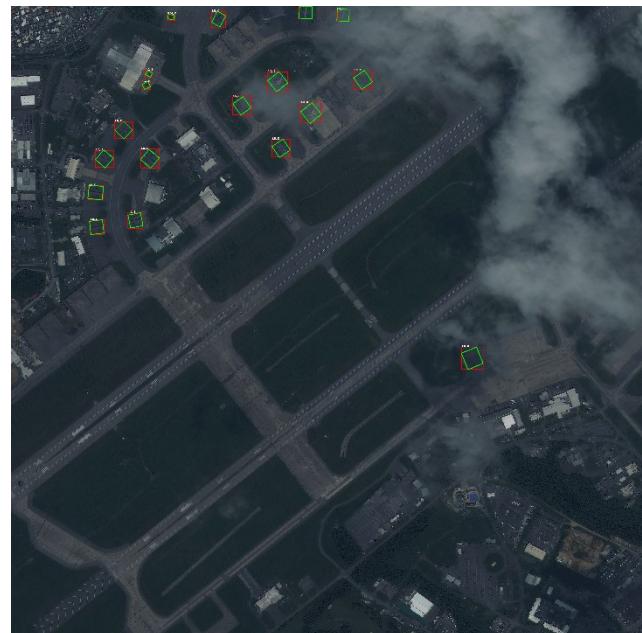
训练轮次: 300

表 7 Mask-RCNN + 细粒度分类结果(F1 / mF1)

luojiaNet	90.4	82.9	88.9	65.6	73.1	88.4	100.	95.6	89.2	91.9	78.3	85.8
	1	1	5	9	7	0	0	5	3	2	3	8



(a) Ground Truth



(b) LuoJiaNET 检测结果

图 8 Mask-RCNN + 细粒度分类检测结果可视化

4.2.4 总结

基于 LuoJiaNET 框架，已经完成 Faster-RCNN 以及 Mask-RCNN 的复现以及对比实验。发现两者在天智杯数据集上召回率大致相同，但是 LuoJiaNET 结果在检测精度上更高。综合来看，LuoJiaNET 的性能在该数据集上的结果更优，在天智杯数据集取得了相对 mmdetection (Pytorch) 更好的结果 mF1，在 DOTA_v2.0 数据集上两者性能差不多。在内存方面，LuoJiaNET 要比 PyTorch 消耗高，推理时间上则是 LuoJiaNET 稍快点。

4.3 地物分类

4.3.1 高光谱地物分类（一）

4.3.1.1 任务简介

- (1) 在 LuoJiaNET 遥感专用深度学习框架下实现高光谱语义分割网络 FreeNet 和 HRNet-3D，从而将 LuoJiaNET 框架应用于高光谱遥感影像地物分类中。
- (2) 进行深度学习主流框架 Tensorflow、Pytorch 和遥感专用深度学习框架 LuoJiaNET 的对比实验，以测试 LuoJiaNET 框架在高光谱语义分割中的性能。

4.3.1.2 高光谱地物分类网络介绍

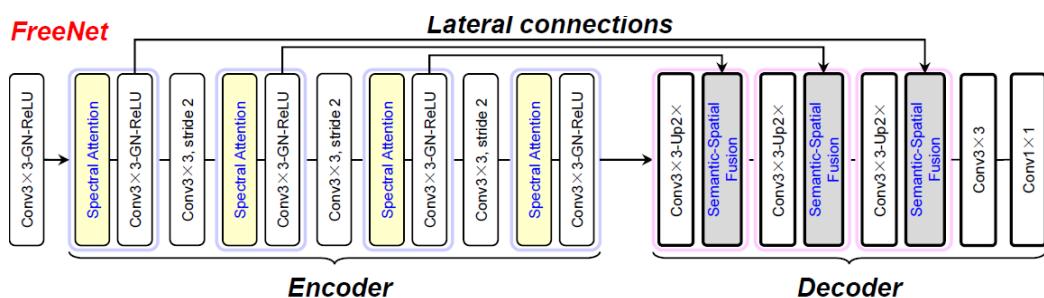


图 1 FreeNet 网络结构

FreeNet 网络[1]采用了语义分割网络中常用的编码-解码网络结构，总体结构如图 1 所示。主要由编码器、解码器、跳跃连接三部分组成，以实现“端到端”的语义分割。编码器用于从原始影像中抽象出特征，解码器用于从特征图中恢复出与原始影像大小相同的语义分割结果图，跳跃连接将编码器每一层的特征与解码器每一层的输出相结合，使得语义分割结果中能还原更多的影像细节。此外，为了充分挖掘高光谱影像中丰富的光谱信息，FreeNet 网络在编码器的每一个卷积层后加入了光谱注意力机制以充分挖掘影像各个波段之间的相互关系。

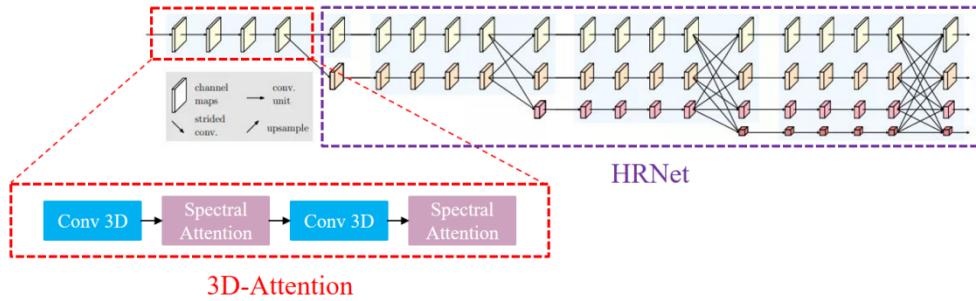


图 2 HRNet-3D 网络结构

HRNet 网络[2]最早为了解决姿态估计问题而提出，后被广泛应用于遥感影像的语义分割中，它着重于学习影像的高分辨率特征表示，现有的语义分割网络大多数采用从高分辨率到低分辨率，再由低分辨率恢复出高分辨率特征表示的结构，与此不同，HRNet 网络能够在整个过程中都保持高分辨率的表征，并通过逐步增加多分辨率的子网络，进行多次多尺度的融合，使得高分辨率的特征表示中能够含有更加丰富的信息，从而能够预测出更加准确的地物位置。为了适应高光谱影像波段数多的特点，HRNet-3D 网络在 HRNet 网络的基础之上加入了 3D 卷积层以及光谱注意力机制层，以充分挖掘影像的光谱信息，整体网络结构如

图 2 所示。

4.3.1.3 测评结果

(1) 实验数据

采用了 WHU-OHS 大范围高光谱数据集来进行实验，该数据集由覆盖了中国 42 个城市的 OHS 高光谱影像组成，每景影像空间分辨率为 10m，幅面大小约为 6000*6000，共 32 个波段，影像对应的标签共包含了 24 个地表覆盖类别。选取了数据集中的吉林、郑州两个城市的数据进行实验，如图 1 和图 2 所示。将影像和标签裁剪为了 256*256 的影像块以进行深度网络的训练和测试，以 7:3 的比例划分了训练集和测试集。

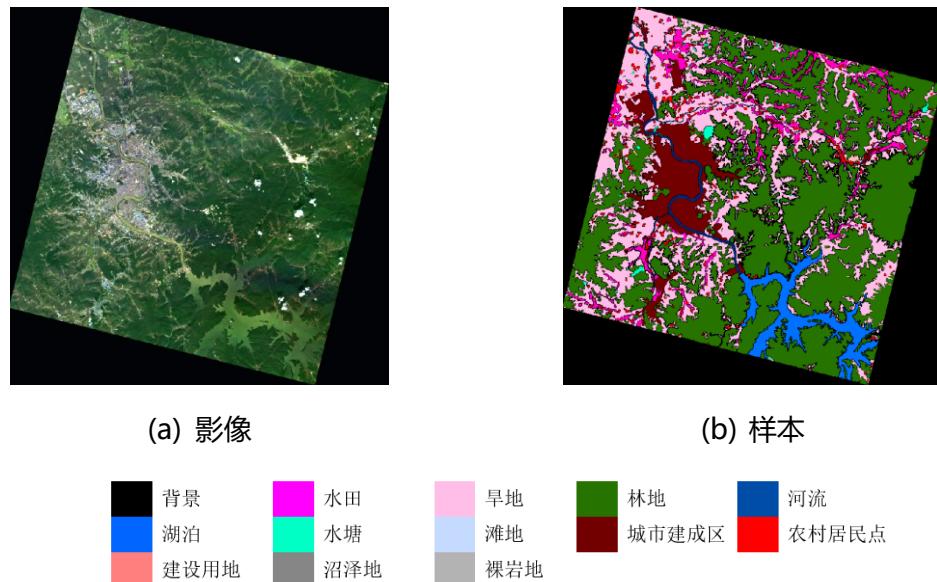


图 1 吉林 OHS 影像及样本

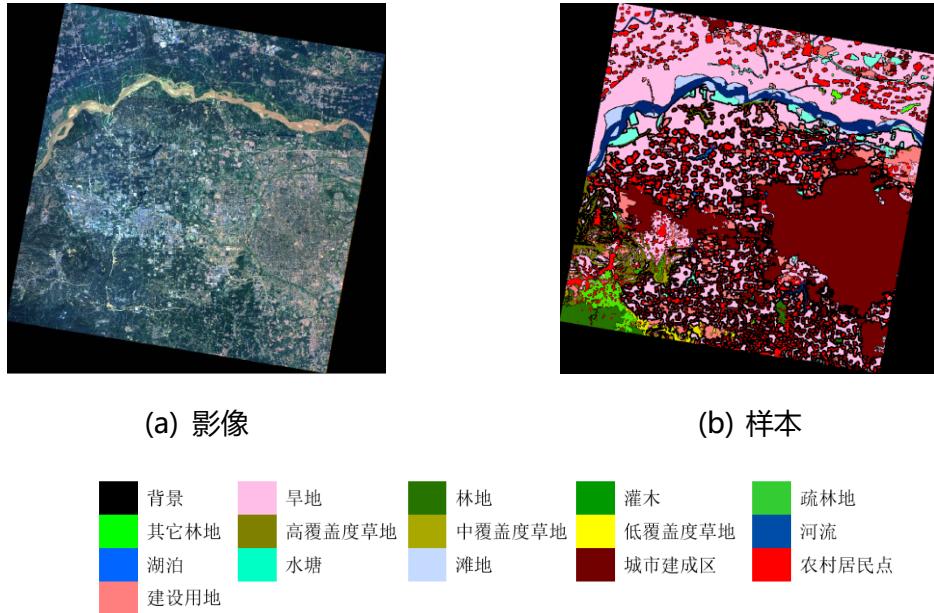


图 2 郑州 OHS 影像及样本

(2) 实验细节

实验分别在 Tensorflow、Pytorch 和 LuoJiaNET 框架下实现了 FreeNet 网络，采用的 Tensorflow 版本为 2.5.0、Pytorch 版本为 1.11.0、LuoJiaNET 版本为 1.0.0，采用的硬件设备为型号为 Inter(R) Core(TM) i7-10700 CPU @ 2.90GHz 的 CPU，以及型号为 NVIDIA GeForce RTX 3070 (8G) 的 GPU。

实验采用的损失函数为加权交叉熵损失函数，优化器选择为 Adam，batchsize 大小为 2，共训练了 100 个 epoch，初始学习率为 0.0001，并按照以下公式随着 epoch 的推进进行衰减（ lr_{init} 为初始学习率， $epoch$ 为当前的 epoch 数， num_epoch 为总 epoch 数）：

$$lr = lr_{init} \left(1 - \frac{epoch}{num_epoch}\right)^{0.9}$$

(3) 精度评价指标

采用了以下定量指标进行精度评价：

a. 总体精度 (OA)

指分类正确的样本数量占总体样本数量的比例：

$$OA = \frac{\sum_{i=1}^N C_{ii}}{\sum_{i=1}^N \sum_{j=1}^N C_{ij}}$$

其中 C 为混淆矩阵，矩阵的第 i 行第 j 列 C_{ij} 表示了真实样本为第 i 类，实际被预测为第 j 类的像素个数。

b. Kappa 系数

评价了分类结果和真实样本分布的一致性：

$$\begin{aligned} Kappa &= \frac{p_o - p_e}{1 - p_e} \\ p_o &= OA = \frac{\sum_{i=1}^N C_{ii}}{\sum_{i=1}^N \sum_{j=1}^N C_{ij}} \\ p_e &= \frac{\sum_{i=1}^N C_{i+} \times C_{+i}}{(\sum_{i=1}^N \sum_{j=1}^N C_{ij})^2} \end{aligned}$$

c. 生产者精度 (PA)

指每一类分类正确的样本个数占所有这一类样本个数的比例：

$$PA_i = \frac{C_{ii}}{\sum_{j=1}^N C_{ij}}$$

d. 用户精度 (UA)

指每一类分类正确的样本个数占所有被分为这一类的样本个数的比例：

$$UA_i = \frac{C_{ii}}{\sum_{j=1}^N C_{ji}}$$

e. F1 分数

生产者精度和用户精度的调和平均：

$$F1 = \frac{2 \times PA \times UA}{PA + UA}$$

f. 交并比 (IoU)

每一类实际提取的像素和真实的像素的交集和并集的比值：

$$IoU = \frac{Area(A \cap B)}{Area(A \cup B)}$$

其中 A 表示实际提取的像素的集合， B 表示真实的像素的集合。

除了测试模型的精度之外，还计算了模型在训练和测试过程中消耗的时间和所占用的显存，以测试不同框架的运行效率。

(4) 实验结果

➤ FreeNet 网络实验结果

得到的 FreeNet 在不同框架下的精度对比如表 1 和表 2 所示。

表 1 FreeNet 在不同框架下的测试精度对比（吉林）

WHU-OHS-Jilin	Tensorflow-FreeNet				Pytorch-FreeNet				LuoJiaNET-FreeNet				样本占比
	PA	UA	F1	IoU	PA	UA	F1	IoU	PA	UA	F1	IoU	
Paddy field	0.614	0.610	0.612	0.441	0.656	0.669	0.662	0.495	0.672	0.657	0.665	0.498	4.8%
Dry farm	0.829	0.770	0.798	0.664	0.791	0.811	0.801	0.668	0.832	0.778	0.804	0.672	15.7%
Woodland	0.935	0.989	0.962	0.926	0.965	0.984	0.974	0.950	0.957	0.989	0.973	0.947	60.2%
River canal	0.929	0.848	0.887	0.797	0.914	0.806	0.857	0.750	0.840	0.840	0.840	0.724	1.3%

Lake	0.998	0.943	0.970	0.941	0.998	0.937	0.967	0.936	0.997	0.958	0.977	0.955	11.4%	
Reservoir pond	0.505	0.374	0.430	0.274	0.240	0.373	0.292	0.171	0.307	0.791	0.443	0.284	0.6%	
Shoal	0.395	0.052	0.092	0.048	0.284	0.108	0.156	0.085	0.506	0.346	0.411	0.259	0.1%	
Urban built-up	0.888	0.987	0.935	0.878	0.918	0.975	0.945	0.896	0.944	0.922	0.933	0.874	3.9%	
Rural settlement	0.720	0.505	0.594	0.422	0.795	0.520	0.629	0.458	0.651	0.533	0.586	0.414	1.7%	
Other construction land	0.762	0.439	0.557	0.386	0.767	0.551	0.641	0.472	0.775	0.595	0.674	0.508	0.2%	
Marshland	0.320	0.124	0.179	0.098	0.400	0.173	0.241	0.137	0.491	0.185	0.269	0.155	0.1%	
Bare rock	0.186	0.879	0.307	0.181	0.077	0.868	0.141	0.076	0.360	0.714	0.479	0.315	0.1%	
OA	0.901				0.916				0.916					
Kappa	0.839				0.860				0.861					
mean F1	0.610				0.609				0.671					
mean IoU	0.505				0.508				0.550					

表 2 FreeNet 在不同框架下的测试精度对比 (郑州)

WHU-OHS-Zhengzhou	Tensorflow-FreeNet				Pytorch-FreeNet				LuoJiaNET-FreeNet				样本占比
	PA	UA	F1	IoU	PA	UA	F1	IoU	PA	UA	F1	IoU	
Dry farm	0.771	0.966	0.858	0.751	0.822	0.965	0.888	0.798	0.804	0.960	0.875	0.778	51.4%
Woodland	0.852	0.790	0.820	0.695	0.728	0.729	0.729	0.573	0.876	0.801	0.837	0.719	2.5%
Shrubbery	0.013	0.065	0.022	0.011	0.009	0.102	0.017	0.009	0.409	0.549	0.469	0.306	0.2%
Sparse woodland	0.298	0.290	0.294	0.172	0.381	0.421	0.400	0.250	0.491	0.553	0.520	0.352	1.6%
Other forest land	0.361	0.208	0.264	0.151	0.592	0.241	0.343	0.207	0.396	0.160	0.227	0.128	0.2%
High-covered grassland	0.261	0.127	0.170	0.093	0.182	0.095	0.124	0.066	0.566	0.153	0.241	0.137	1.2%
Medium-covered grassland	0.197	0.017	0.031	0.016	0.271	0.033	0.059	0.030	0.597	0.096	0.165	0.090	0.1%
Low-covered grassland	0.895	0.132	0.231	0.130	0.838	0.159	0.267	0.154	0.721	0.308	0.432	0.275	0.4%
River canal	0.943	0.841	0.889	0.800	0.921	0.894	0.907	0.830	0.929	0.851	0.888	0.799	6.4%
Lake	0.651	0.896	0.754	0.605	0.538	0.963	0.690	0.527	0.496	0.986	0.660	0.493	0.3%
Reservoir pond	0.700	0.369	0.484	0.319	0.654	0.414	0.507	0.340	0.830	0.490	0.616	0.445	2.1%
Shoal	0.754	0.463	0.574	0.402	0.798	0.558	0.656	0.488	0.713	0.659	0.685	0.520	3.4%

Urban built-up	0.963	0.977	0.970	0.942	0.958	0.990	0.974	0.948	0.951	0.992	0.971	0.943	16.7%
Rural settlement	0.831	0.741	0.783	0.644	0.850	0.757	0.801	0.668	0.839	0.774	0.806	0.674	10.1%
Other construction land	0.403	0.488	0.442	0.284	0.530	0.473	0.500	0.333	0.603	0.438	0.507	0.340	3.4%
OA	0.791			0.819			0.822						
Kappa	0.718			0.751			0.756						
mean F1	0.506			0.524			0.593						
mean IoU	0.401			0.415			0.467						

从表 1 和表 2 来看，Tensorflow 框架和 Pytorch 框架的效果相当，Pytorch 略好于 Tensorflow；而从 Pytorch 和 LuoJiaNET 框架下的精度对比来看，得到的总体精度指标 OA、Kappa 比较接近，这是因为每个测试区域中不同类别的样本量不均衡，这两个框架在样本数量较多的类别上得到的精度是比较接近的。而从每一类的精度上来看，在 LuoJiaNET 框架下大部分类别上(特别是样本较少的类别)都能够取得比在 Tensorflow 框架和 Pytorch 框架下更高的精度，因此得到的类别平均的 F1 分数和平均 IoU 也更高。更进一步地，对随着迭代的推进模型的训练集 loss 以及测试集平均 IoU 的变化进行了可视化分析，如图 3 和图 4 所示。

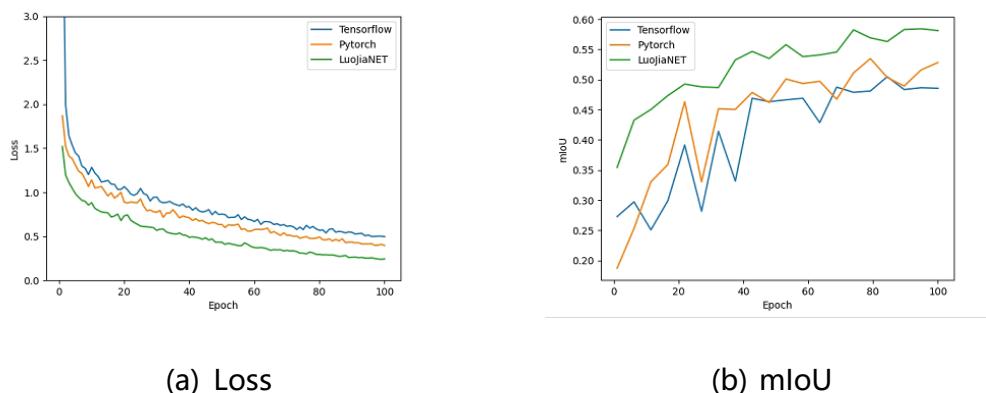


图 3 训练集 loss 和测试集 mIoU 随着迭代的变化 (吉林)

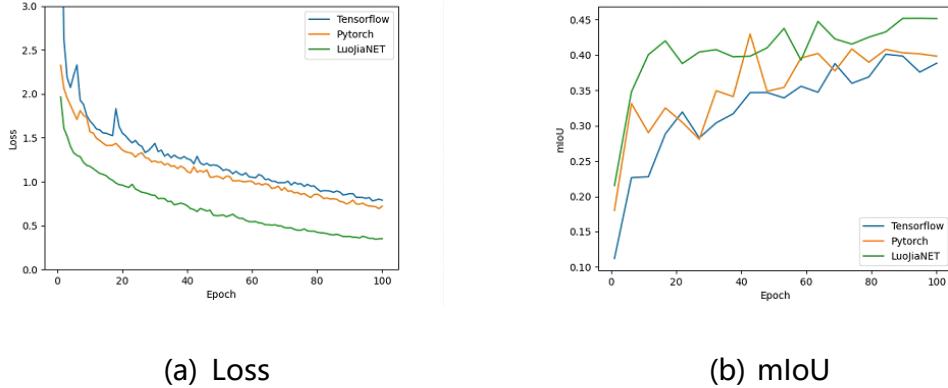


图 4 训练集 loss 和测试集 mIoU 随着迭代的变化 (郑州)

从图 3 和图 4 可以看出，FreeNet 网络在 LuoJiaNET 框架下进行训练的过程中的训练集 loss 值要明显比 Tensorflow 框架和 Pytorch 框架下更低，由于采用的 loss 是针对样本的比例进行了加权的交叉熵损失函数，因此这与 LuoJiaNET 框架下有着更高的各类别精度是比较一致的。此外，从测试集的平均 IoU 随着迭代的变化可以看出，LuoJiaNET 框架下的训练过程更加稳定。

图 5 展示了 FreeNet 网络在不同框架下得到的局部结果可视化对比，可以看出 LuoJiaNET 框架下得到的分类结果比其它框架更加准确。



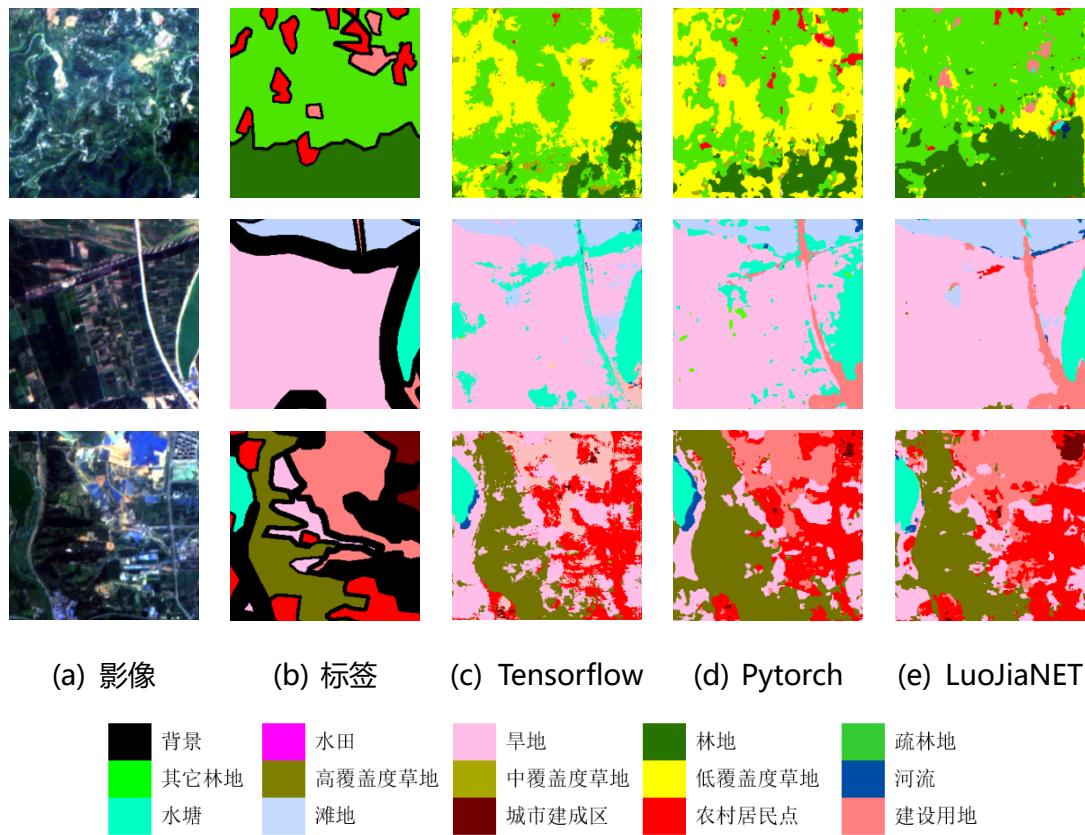


图 5 FreeNet 网络局部结果可视化对比

表3展示了不同框架下FreeNet的训练和测试的效率对比,其中训练时间为每个epoch所需要的时间,测试时间为整个测试过程所需要的时间。(额外测试了近期发布的LuoJiaNET最新版本v1.0.5的运行效率)

表 3 FreeNet 在不同框架下的运行效率对比

深度学习框架	训练时间	测试时间	训练显存占用	测试显存占用
Tensorflow	48.40s/epoch	15.09s	7352MB	7268MB
Pytorch	45.01s/epoch	5.98s	6436MB	5736MB
LuoJiaNET	171.48s/epoch	14.15s	7155MB	3999MB

LuoJiaNET_v1.0.	172.41s/epoch			
5	h	14.49s	6667MB	2965MB

从表 3 可以看出，LuoJiaNET 除了训练时间以外都优于 Tensorflow，但只在测试所占用的显存上优于 Pytorch，并没有体现出在运行效率上的优势，出现这种现象的可能原因在于 GPU 驱动和 CUDA 版本不匹配导致底层调度较慢，还有待进一步分析。此外，经测试，最新 v1.0.5 版本的 LuoJiaNET 能够显著降低训练和测试时的显存占用，但没有缩短训练和测试所需要的时间。

➤ HRNet-3D 网络实验结果

得到的 HRNet-3D 在不同框架下的精度对比如表 4 和表 5 所示。

表 4 HRNet-3D 在不同框架下的测试精度对比（吉林）

WHU-OHS-Jilin	Tensorflow-HRNet3D				Pytorch-HRNet3D				LuoJiaNET-HRNet3D				样本占比
	PA	UA	F1	IoU	PA	UA	F1	IoU	PA	UA	F1	IoU	
Classname													
Paddy field	0.490	0.793	0.606	0.434	0.508	0.786	0.617	0.446	0.618	0.703	0.658	0.490	4.8%
Dry farm	0.894	0.734	0.806	0.675	0.878	0.730	0.797	0.663	0.856	0.762	0.806	0.675	15.7%
Woodland	0.976	0.974	0.975	0.951	0.976	0.973	0.975	0.951	0.970	0.982	0.976	0.952	60.2%
River canal	0.630	0.965	0.762	0.616	0.795	0.945	0.863	0.759	0.897	0.908	0.903	0.823	1.3%
Lake	0.997	0.963	0.979	0.959	0.996	0.957	0.976	0.953	0.996	0.959	0.977	0.955	11.4%
Reservoir pond	0.232	0.526	0.322	0.192	0.188	0.563	0.282	0.164	0.193	0.504	0.280	0.162	0.6%
Shoal	0.322	0.784	0.456	0.296	0.106	0.601	0.180	0.099	0.517	0.697	0.594	0.422	0.1%
Urban built-up	0.838	0.920	0.877	0.781	0.766	0.917	0.834	0.716	0.929	0.975	0.952	0.908	3.9%
Rural settlement	0.372	0.699	0.485	0.320	0.437	0.708	0.540	0.370	0.473	0.614	0.535	0.365	1.7%

Other construction land	0.570	0.896	0.697	0.535	0.518	0.884	0.653	0.485	0.851	0.886	0.868	0.767	0.2%	
Marshland	0.021	1.000	0.041	0.021	0.034	0.963	0.065	0.034	0.029	0.444	0.054	0.028	0.1%	
Bare rock	0.573	1.000	0.729	0.573	0.668	1.000	0.801	0.668	0.666	1.000	0.799	0.666	0.1%	
OA	0.916				0.914				0.921					
Kappa	0.857				0.854				0.868					
mean F1	0.645				0.632				0.700					
mean IoU	0.529				0.526				0.601					

表 5 HRNet-3D 在不同框架下的测试精度对比 (郑州)

WHU-OHS-Zhengzhou	Tensorflow-HRNet3D				Pytorch-HRNet3D				LuoJiaNET-HRNet3D				样本占比
	PA	UA	F1	IoU	PA	UA	F1	IoU	PA	UA	F1	IoU	
Dry farm	0.939	0.858	0.897	0.813	0.955	0.842	0.895	0.810	0.890	0.923	0.906	0.829	51.4%
Woodland	0.069	0.717	0.126	0.067	0.057	0.531	0.104	0.055	0.771	0.817	0.793	0.657	2.5%
Shrubbery	0.092	0.153	0.115	0.061	0.066	0.125	0.086	0.045	0.204	0.351	0.258	0.148	0.2%
Sparse woodland	0.287	0.276	0.281	0.164	0.001	0.100	0.002	0.001	0.173	0.547	0.263	0.151	1.6%
Other forest land	0.171	0.079	0.108	0.057	0.193	0.071	0.104	0.055	0.136	0.234	0.172	0.094	0.2%
High-covered grassland	0.098	0.100	0.099	0.052	0.060	0.100	0.075	0.039	0.527	0.217	0.308	0.182	1.2%
Medium-covered grassland	0.258	0.070	0.110	0.058	0.245	0.305	0.272	0.157	0.480	0.141	0.218	0.122	0.1%
Low-covered grassland	0.399	0.683	0.503	0.336	0.092	0.603	0.159	0.087	0.675	0.589	0.629	0.459	0.4%
River canal	0.815	0.813	0.814	0.686	0.887	0.831	0.858	0.751	0.945	0.864	0.903	0.823	6.4%
Lake	0.010	0.242	0.020	0.010	0.390	0.917	0.547	0.377	0.126	0.920	0.222	0.125	0.3%
Reservoir pond	0.769	0.731	0.750	0.600	0.756	0.760	0.758	0.611	0.705	0.625	0.663	0.495	2.1%
Shoal	0.573	0.654	0.611	0.440	0.337	0.842	0.482	0.317	0.673	0.748	0.709	0.549	3.4%
Urban built-up	0.935	0.958	0.946	0.898	0.966	0.945	0.955	0.914	0.970	0.965	0.968	0.937	16.7%
Rural settlement	0.772	0.857	0.812	0.684	0.801	0.804	0.803	0.670	0.821	0.733	0.775	0.632	10.1%
Other construction land	0.330	0.492	0.395	0.246	0.365	0.510	0.425	0.270	0.425	0.457	0.441	0.283	3.4%
OA	0.826				0.834				0.847				
Kappa	0.739				0.746				0.780				

mean F1	0.439	0.435	0.548	
mean IoU	0.345	0.344	0.432	

HRNet-3D 网络在不同框架下精度的对比结果与 FreeNet 基本一致，在 LuoJiaNET 框架下能够得到明显更高的总体精度以及每一类的精度。此外，从不同网络的精度对比上看，HRNet-3D 网络要略好于 FreeNet 网络。

图 6 和图 7 对随着迭代的推进模型的训练集 loss 以及测试集平均 IoU 的变化进行了可视化分析。

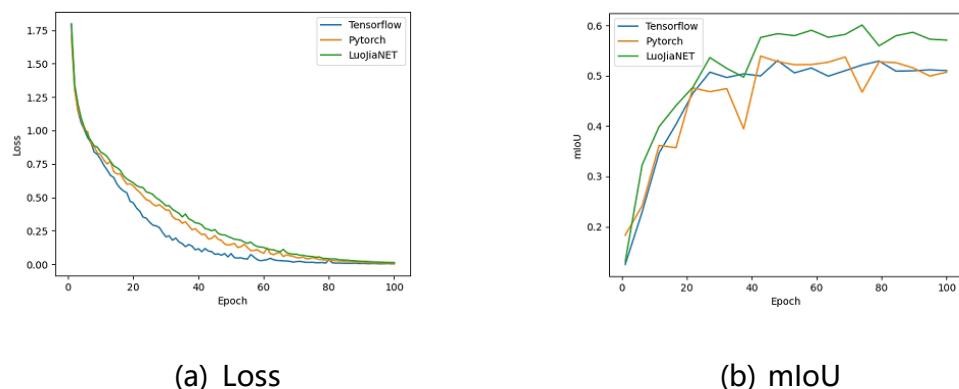


图 6 训练集 loss 和测试集 mIoU 随着迭代的变化 (吉林)

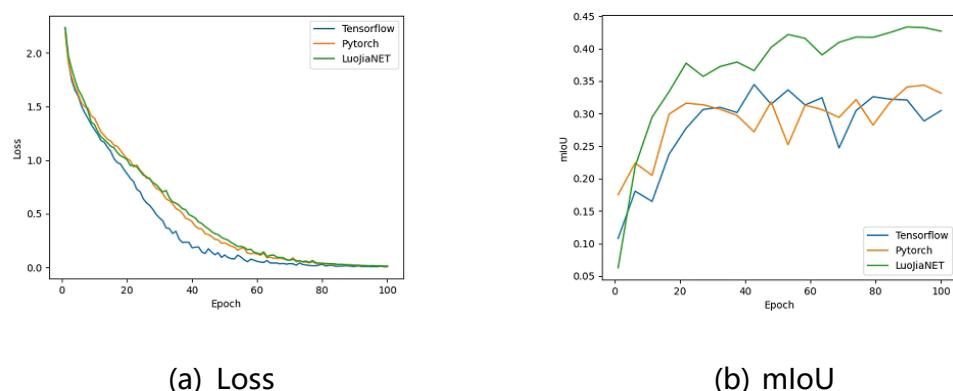


图 7 训练集 loss 和测试集 mIoU 随着迭代的变化 (郑州)

随着迭代的推进，HRNet-3D 在不同框架下的训练集 loss 最终能够收敛到比较接近的值，但是 LuoJiaNET 框架下能够达到的测试集 mIoU 明显更高，可见 LuoJiaNET 框架下所训练的网络具有更好的泛化能力。

图 8 展示了 HRNet-3D 网络在不同框架下得到的局部结果可视化对比，LuoJiaNET 框架下得到的地物分类结果更加准确。

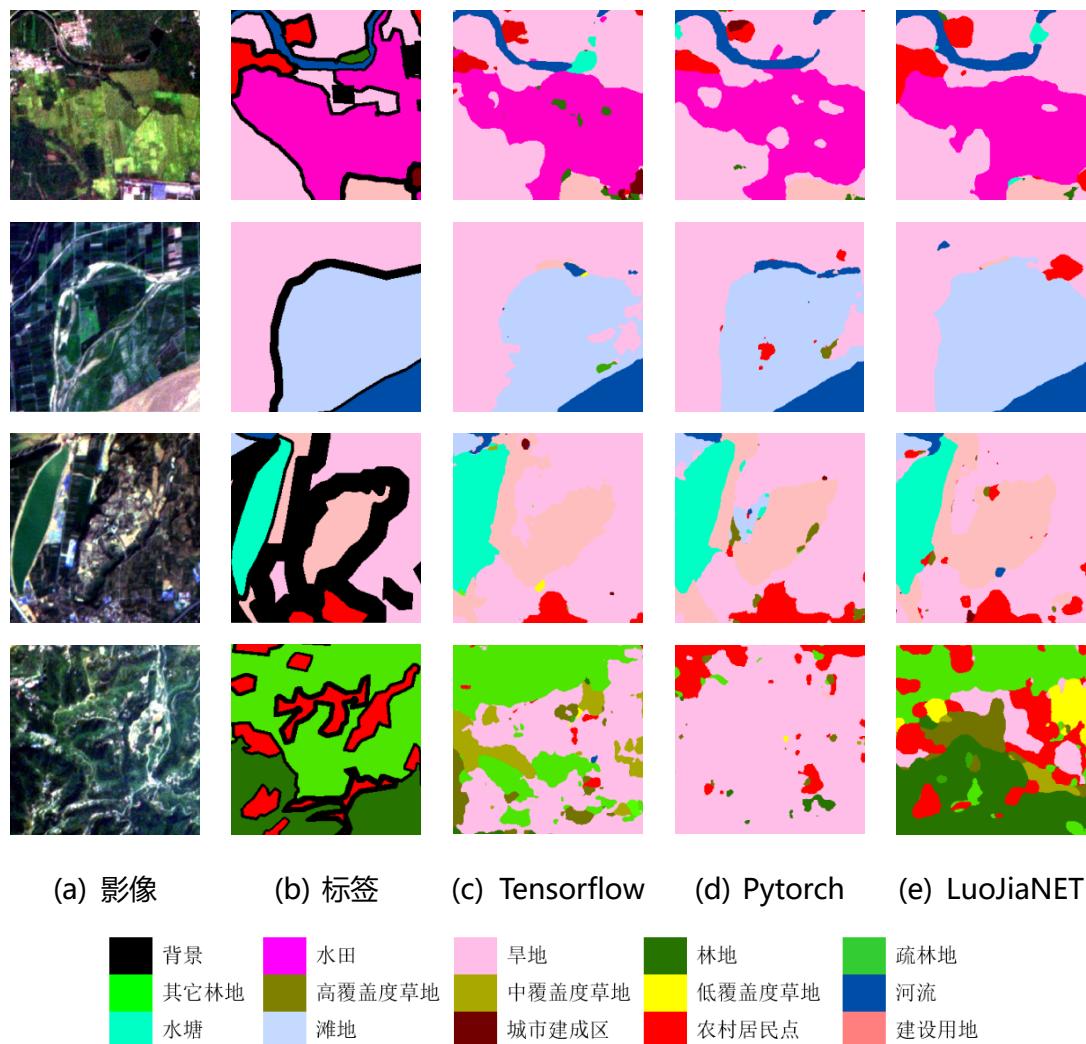


图 8 HRNet-3D 网络局部结果可视化对比

表 6 展示了不同框架下 HRNet-3D 网络的训练和测试的效率对比。(额外测试了近期发布的 LuoJiaNET 最新版本 v1.0.5 的运行效率)

表 6 HRNet-3D 在不同框架下的运行效率对比

深度学习框架	训练时间	测试时间	训练显存占用	测试显存占用
Tensorflow	193.54s/epoch	41.35s	7318MB	7270MB
Pytorch	200.14s/epoch	19.85s	5087MB	4781MB
LuoJiaNET	576.37s/epoch	41.52s	6129MB	3950MB
LuoJiaNET_v1.0.5	585.97s/epoch	41.23s	5610MB	2947MB

与 FreeNet 得到的结果较为一致，除了测试阶段的显存占用之外，并没有体现出 LuoJiaNET 框架在效率上的明显优势。此外，经测试，最新 v1.0.5 版本的 LuoJiaNET 能够显著降低训练和测试时的显存占用，但没有缩短训练和测试所需要的时间。

4.3.1.4 总结

实验基于 FreeNet 和 HRNet-3D 两种深度网络，在遥感专用深度学习框架 LuoJiaNET 下完成了高光谱遥感影像地物语义分割的应用，并且与深度学习主流框架 Tensorflow 和 Pytorch 进行了对比。实验结果表明，LuoJiaNET 框架能够得到比 Tensorflow 框架和 Pytorch 框架更加稳定的训练过程以及更高的语义分割精度，但是除了测试阶段能够占用更少的显存之外，没有体现出 LuoJiaNET 框架在效率上的优势，需要进一步分析其中的原因，可能这也是 LuoJiaNET 可以优化的地方。

4.3.2 高光谱地物分类 (二)

4.3.2.1 任务简介

- (1) 使用 LuoJiaNET 完成最新 SOTA 的高光谱深度分类网络 S³ANet 和经典高光谱分类网络 FreeNet 的搭建与训练，将 LuoJiaNET 框架应用到高光谱分类任务中；
- (2) 将 LuoJiaNET 和主流的深度学习框架 PyTorch 进行对比，测试 LuoJiaNET 在高光谱分类任务中的性能表现。

4.3.2.2 高光谱地物分类网络简介

➤ S³ANet 简介

S³ANet 网络是由 RSIDEA 研究组提出的高光谱深度分类模型，不同于传统高光谱分类网络仅关注于局部空谱特征，S³ANet 网络以端到端的方式基于全局空谱信息实现高光谱影像精细分类，具备更强的空谱特征提取能力（架构见图 1）。

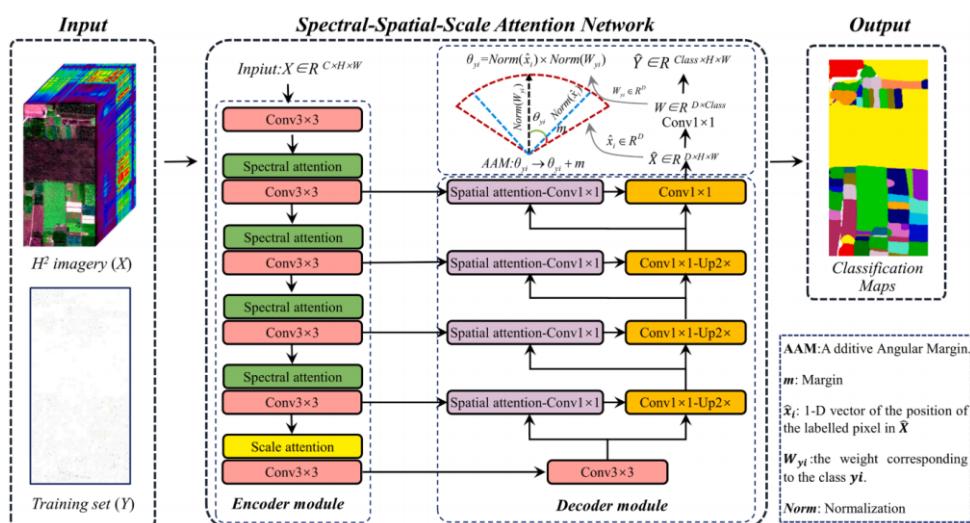


图 5 高光谱分类网络 S³ANet 结构图

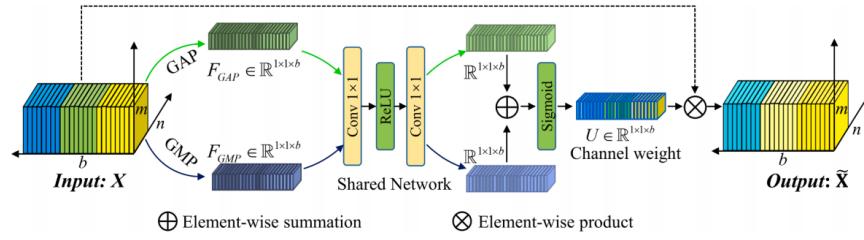


图 2 S³ANet 中光谱注意力模块图

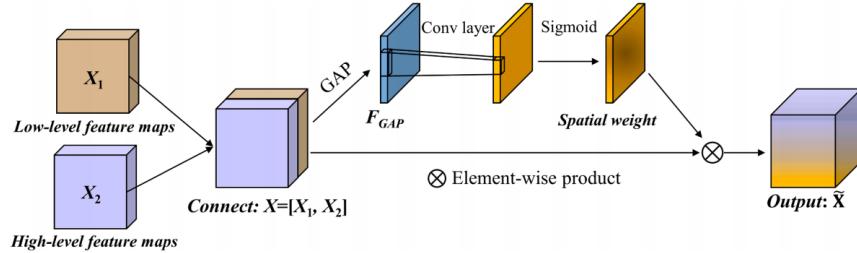


图 3 S³ANet 中空间注意力模块图

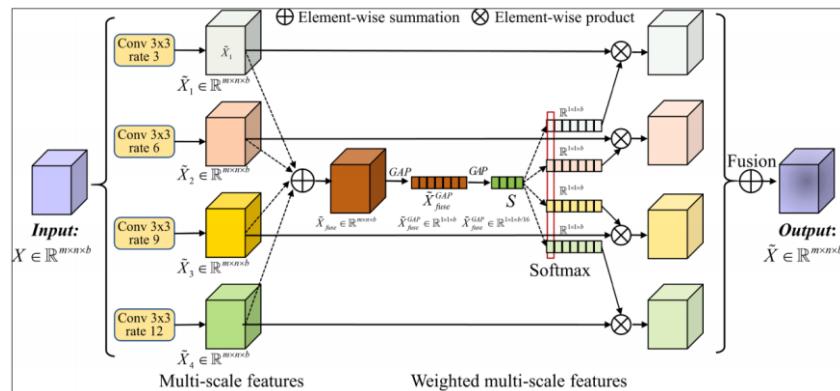


图 4 S³ANet 中尺度注意力模块图

为了更好地应对高光谱影像中的光谱异质性问题，缓解较大类内方差带来的影响，S³ANet 网络设计利用了光谱注意力模块、空间注意力模块和尺度注意力模块(架构见图 2, 3, 4)。编码器部分由级联的 3×3 卷积网络和空谱注意力模块构成，光谱注意力可以促使网络关注更具判别性的波段范围，3×3 卷积网络以较小的卷积核提取高层次的空间特征，二者相互结合提取更鲁棒的空谱特征。解码器部分，每层反卷积网络均与编码器输出特征图连接。借助空间注意力模块对低级空间信息进行加权处理，减少编码器输出特征图和解码器高级语义信息之间的鸿沟，使得网络在保持良好空间细节信息的同时，提升网络的语义特征提

取能力。鉴于高光谱影像中地物尺度不一，固定的卷积感受野会局限网络语义特征的提取，为此，在编码器和解码器的连接处，S³ANet 设计了尺度注意力模块将提取到的多尺度信息进行加权融合，而非 ASPP 中的等比例融合，实现网络对多尺度信息的捕获和特征提取。损失方面，S³ANet 网络在训练阶段除了交叉熵损失，还设计了边缘角度损失用于更好地应对光谱异质性问题。

➤ FreeNet 简介

FreeNet 是第一个做到端到端 patch-free 的分类架构，其架构如图 5 所示。编码器由一系列堆叠的 3×3 卷积和光谱注意力构成，解码器由 3×3 卷积和二倍上采样操作完成。编码器和解码器之间包含多个跨层连接，编码层输出经过空谱融合模块后与解码层相加，使得网络在解码时可以利用编码器的底层细节信息，兼顾高级语义特征和低级细节特征。

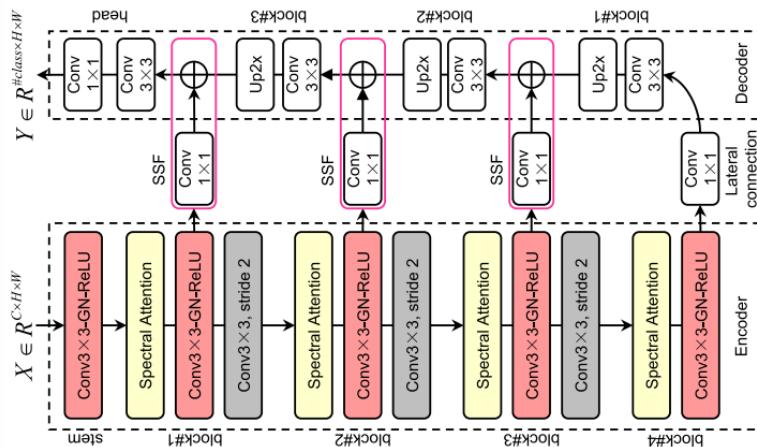


图 5 FreeNet 架构

4.3.2.3 测评结果

(1) 数据集

本次实验采用数据集为 RSIDEA 研究组发布的三套无人机高光谱数据集 WHU-Hi-

HongHu, WHU-Hi-HanChuan 和 WHU-Hi-JiaYu。所使用三套高光谱数据集影像和光谱信息展示见图 6，相关类别信息和采集信息见表 1。

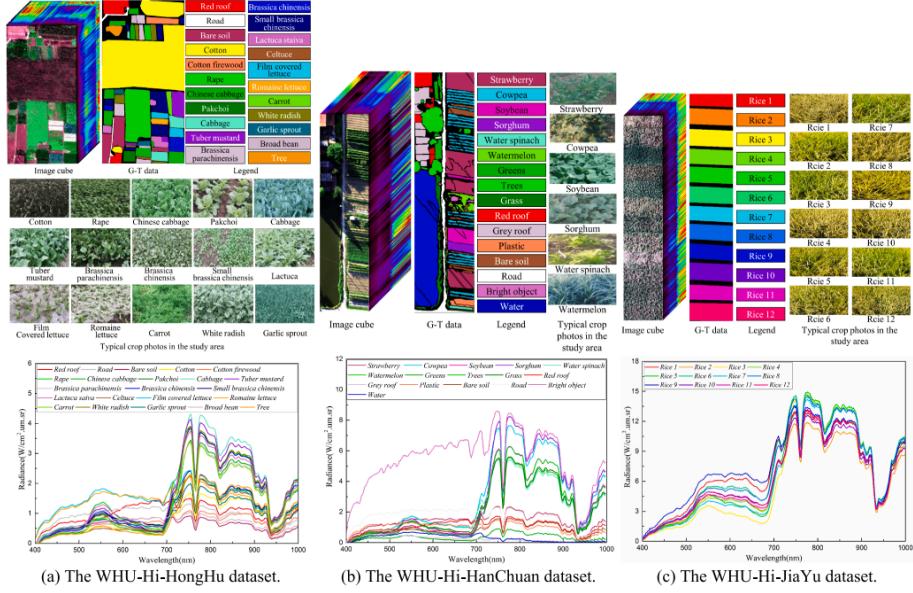


图 6 所使用三套高光谱数据集影像和光谱信息展示

表 1 所使用三套高光谱数据集影像类别信息和采集信息

Dataset	Focal length	Flight height	Spatial resolution	Number of bands	Image size	Number of classes	Label ratio	Location	Time
WHU-Hi-HongHu	17 mm	100 m	0.043 m	270	940 × 475	22	86.61%	City of Honghu	Nov. 2017
WHU-Hi-HanChuan	17 mm	250 m	0.109 m	274	1217 × 303	16	69.84%	City of HanChuan	Jun. 2016
WHU-Hi-JiaYu	17 mm	35 m	0.015 m	270	1935 × 370	12	82.30%	City of JiaYu	Sept. 2019

(2) 实现细节

使用的 PyTorch 版本是 v1.8.0，LuoJiaNET 版本为 v1.0.0。

我们分别在主流的深度学习框架 PyTorch 和我们的 LuoJiaNET 上复现了 S³ANet 和 FreeNet 网络。为了进行公平的对比，两网络均在同一硬件上进行训练和测试。GPU 型号为 Nividia Geoforce RTX 3090、CPU 型号为 Intel(R) Xeon(R) Silver 4210R CPU

@2.40GHz。

在进行训练时，S³ANet 损失函数为边缘角度损失及交叉熵损失，FreeNet 损失函数为交叉熵损失，优化器选择为 Adam。S³ANet 和 FreeNet 网络学习率为 0.001。训练时的样本构造采用分层采样算法。每个样本保持原始大小，不适用任何数据增强技巧。对于每个类别，本次实验选取 50 个样本点进行训练，其余样本点进行测试。

(3) 评价指标

a. 精度评价指标(基于混淆矩阵)

总体精度 Overall accuracy (OA)，平均精度 Average accuracy (AA)，Kappa 系数。

b. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要的显存占用。

c. 运行时间 (Time)

记录了模型在训练或者推理过程中处理网络一次反向传播和前向传播所需要的时间

(4) 定量实验结果

S³ANet 在三个数据集上的定量结果见表 2，FreeNet 的定量结果见表 3。加粗字体显示相对最优。精度方面，使用 LuojiaNet 框架训练的 S³ANet 和 FreeNet 相较于 Pytorch 框架，可以取得更优的 OA 和 Kappa 精度，但 AA 略低。显存方面，FreeNet 使用 LuojiaNet 框架占用训练和推理显存明显少于 Pytorch 框架，S³ANet 在两个框架上占用显存相似。时间方面，基于 Pytorch 的两个模型拥有更少的训练和推理时间，关于这点我们认为可能是由于以下几个原因导致。(1) 两种框架底层的梯度下降优化方式(动态图与静态图计算)不同有关；(2) 在 LuojiaNET 自定义单步训练流程需要继承 nn.TrainOneStepCell 类，然后再定义计算损失的 nn.Module 子类计算反向传播的损失，单步的网络反向传播方式不同，该部分可以考虑继续优化。(3) 在 LuojiaNET 自定义单步训练流程需要继承

nn.TrainOneStepCell 类，梯度反向传播过程中涉及到数据 cpu 和 gpu 之间的相互转换，

LuojiaNET 并没有像 Pytorch 一样将数据直接转换成.cuda()和.cpu()格式类型。

S³ANet 和 FreeNet 在三个数据集上的混淆矩阵见表 4。可以看出，LuojiaNet 最终得到的单类精度情况和 Pytorch 总体一致。

表 2 S³ANet 在三个数据集上的定量结果

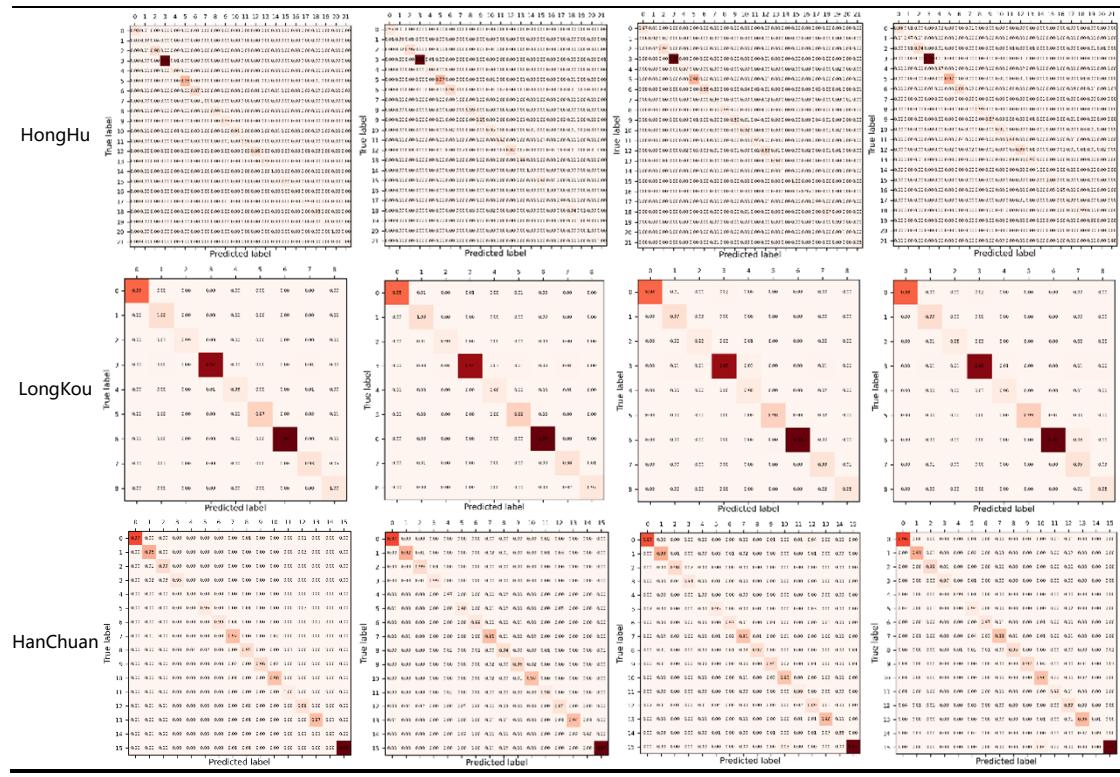
		OA	AA	Kappa	训练显存	推理显存	训练时间	推理时间
HongHu	Pytorch	96.43	97.20	95.51	8921M	3805M	0.520s	0.462s
	LuojiaNet	96.53	96.89	95.63	9459M	4409M	0.453s	0.746s
LongKou	Pytorch	97.70	98.26	96.99	5181M	2743M	0.532s	0.482s
	LuojiaNet	98.49	98.21	98.03	5361M	3001M	0.379s	0.612s
HanChau	Pytorch	96.28	96.70	95.32	6447M	3447M	0.537s	0.464s
	LuojiaNet	96.97	96.59	96.46	6857M	3513M	0.479s	0.724s

表 3 FreeNet 在三个数据集上的定量结果

		OA	AA	Kappa	训练显存	推理显存	训练时间	推理时间
HongHu	Pytorch	95.25	95.78	94.02	9635M	4291M	0.134s	0.012s
	LuojiaNet	95.68	95.56	94.57	8953M	3009M	2.462s	0.179s
LongKou	Pytorch	96.95	97.53	96.02	5513M	3021M	0.076s	0.016s
	LuojiaNet	97.31	97.43	96.48	4985M	2241M	2.063s	0.384s
HanChau	Pytorch	94.22	94.11	93.27	6723M	3839M	0.129s	0.013s
	LuojiaNet	94.67	94.28	93.79	6649M	3523M	2.165s	0.139s

表 4 S³ANet 和 FreeNet 在三个数据集上的单类精度

S ³ ANet		S ³ ANet		FreeNet		FreeNet	
LuojiaNet	Pytorch	Pytorch	LuojiaNet	LuojiaNet	Pytorch	Pytorch	



(5) 定性可视化

S^3 ANet 和 FreeNet 在三个高光谱分类数据集上的定性结果见图 7-9。从结果可以看出，Luojianet 输出的分类图具备更少的噪点和更平滑的边界，与定量结果展示的优越性一致。

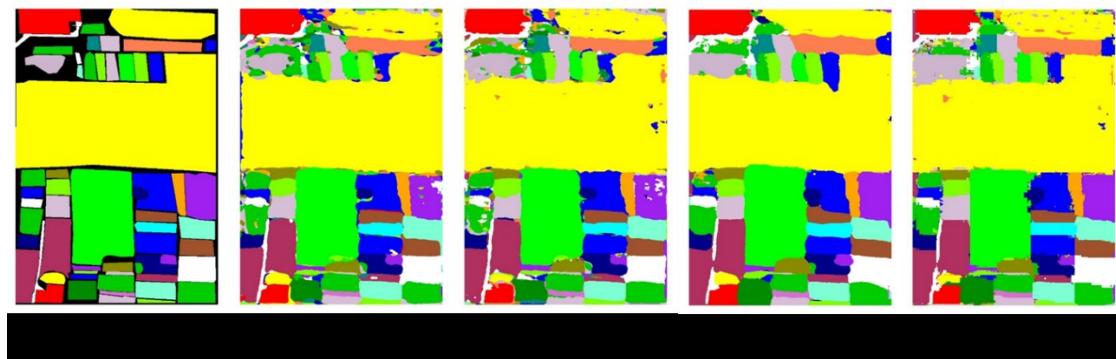


图 7 S^3 ANet 和 FreeNet 在洪湖数据集上的定性对比

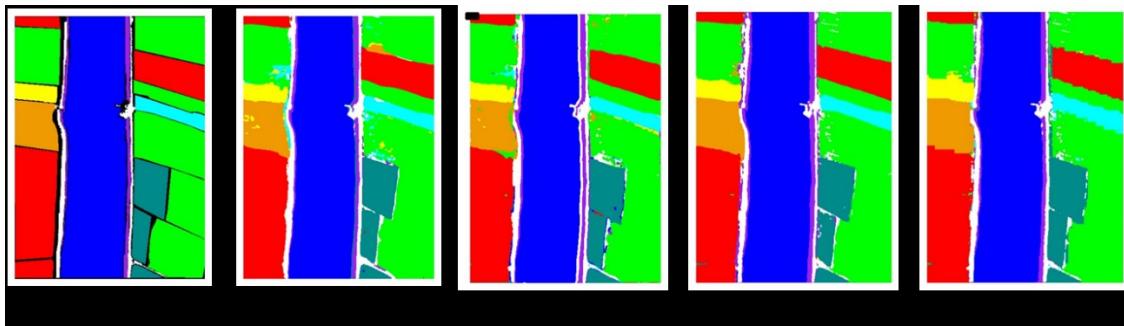


图 8 S³ANet 和 FreeNet 在龙口数据集上的定性对比

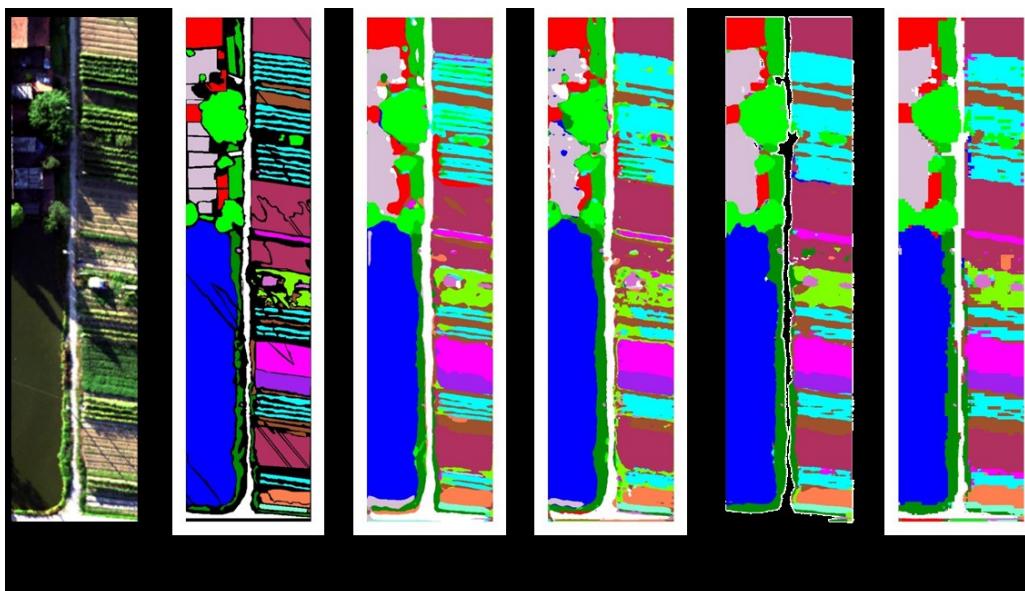


图 9 S³ANet 和 FreeNet 在汉川数据集上的定性对比

4.3.2.4 时间占用分析

从第三节的定量结果可以看出, 虽然 LuoJiaNet 在精度和显存占用方面略优于 Pytorch, 但在 FreeNet 的相关实验中, 训练和推理时间明显多于 Pytorch。为了找见具体原因, 我们将时间分析细化到每一层, 并分开展示了损失计算, 梯度计算和参数更新这些步骤。分析结果见表 5。所有测试均在相同条件下进行。

从结果可以看出, LuoJiaNet 占用时间明显较多的算子为 GN, AvgPool2d 和 interpolate 算子。此外, LuoJiaNet 在损失计算, 梯度计算和参数更新这三个步骤上消耗时间明显多于

Pytorch。其中，梯度计算消耗时间最为严重。

表 5 FreeNet 包含算子在三个数据集上训练时间的具体分析

	HongHu	HongHu	LongKou	LongKou	HanChuan	HanChuan
	LuoJianet	Pytorch	LuoJiaNet	Pytorch	LuoJianet	Pytorch
conv3x3	0.0004	0.0066	0.0004	0.0033	0.0009	0.0160
GN	0.0031	0.0021	0.0032	0.0011	0.0088	0.0021
Relu	0.0002	0.0004	0.0002	0.0002	0.0006	0.0005
SEBlock-AvgPool2d	0.0060	0.0003	0.0056	0.0002	0.0008	0.0001
SEBlock-Linear	0.0016	0.0003	0	0.0003	0.0001	0.0003
SEBBBlock-Multiply	0.0004	0.0005	0.0008	0.0002	0.0006	0.0002
Identity	0	0	0	0	0	0
Interpolate	0.0014	0.0002	0.0012	0.0001	0.0015	0.0002
Loss 计算	0.0025	0.0005	0.0025	0.0003	0.0025	0.0004
梯度计算	2.2011	0.0506	1.2084	0.0448	1.9027	0.0757
更新参数	0.0617	0.0060	0.0708	0.0043	0.0620	0.0035

4.3.2.5 总结

基于 LuoJiaNET 框架，已经完成了高光谱分类的典型应用。在高光谱分类任务中，LuoJiaNET 相比于主流深度学习框架 PyTorch，部分精度指标有一定提升，总体相差不大。在显存占用上，也略优于 PyTorch。不过，LuoJiaNET 需要更长的训练和推理时间。

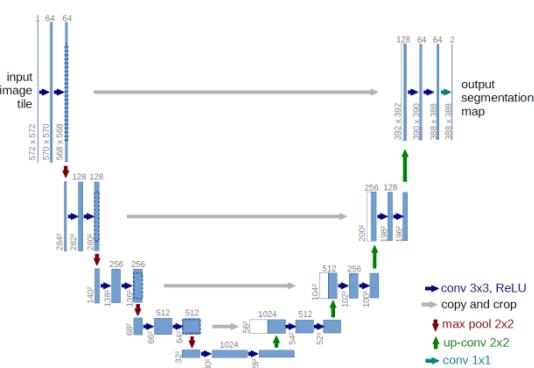
4.3.3 高空间分辨率地物分类 (一)

4.3.3.1 任务简介

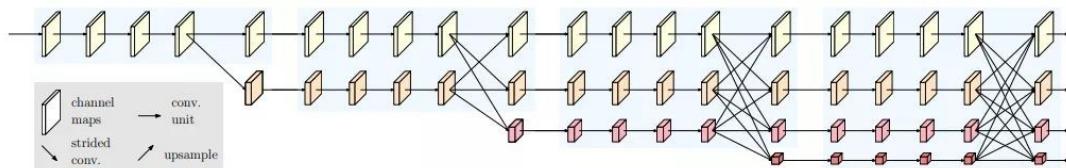
- (1) 使用 LuoJiaNET 完成经典深度学习语义分割网络 UNet 和 HRNet 的搭建与训练。
- (2) 将 LuoJiaNET 和主流的深度学习框架 PyTorch、Tensorflow 进行对比，测试 LuoJiaNET 在语义分割任务中的性能表现。

4.3.3.2 高分辨率地物分类网络简介

HRNet 和 UNet 分别代表了 CNN 在语义分割任务上的两大网络体系，一种是并行不同分辨率的卷积流并不断进行信息交互的高分辨率网络结构，一种是经过不断卷积池化提取特征后在恢复原始分辨率的编解码体系。



UNet 网络结构



HRNet 网络结构

4.3.3.3 测评结果

(1) GID 数据集

GID 数据集包括两个部分：大规模分类集和精细土地覆盖分类集。本次测试选用大规模分类集部分。该部分包括训练集 120 张，输入为 512*512，初始学习率 1e-1，并使用余弦退火策略，最大迭代次数为 90，并使用旋转、翻转、颜色抖动等数据增强方式。需要注意的是，继续训练网络更多的迭代次数仍可使得测验证集 30 张(6800*7200)。波段为 RGB 形式或 NIR+RGB(6800*7200*4)。分为 5 类(build-up、farmland、forest、meadow、water)。

下图为样本地理位置分布及图片示例。

(2) 实现细节

训练时，所有网络批次大小为 24，试精度上升。

(3) 评价指标

在实验中，采用的定量指标如下：

a. 平均交并比 (MIOU)

$$MIOU = \frac{1}{k+1} \sum_{i=0}^k \frac{TP}{TP + FP + FN}$$

b. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要的显存占用。

c. 帧率 (FPS)

每秒处理的图片数量。

(4) 实验结果

表 6 精度结果对比

	MIO	Build-	Farmlan	Forest	Meado	Water
	U	up	d		w	
UNet-Pytorch 9	62.8	63.34	63.28	41.04	65.96	80.85
UNet-Luojianet 6	63.6	62.80	65.28	45.67	64.57	79.97
UNet- Tensorflow 3	61.3	62.82	65.42	38.13	61.80	78.49
HRNet-Pytorch 4	61.3	62.47	64.78	40.51	58.38	80.55
HRNet- Luojianet 8	62.5	62.92	64.90	43.26	61.71	80.10
HRNet- Tensorflow 8	62.5	60.34	66.55	41.94	63.46	80.60

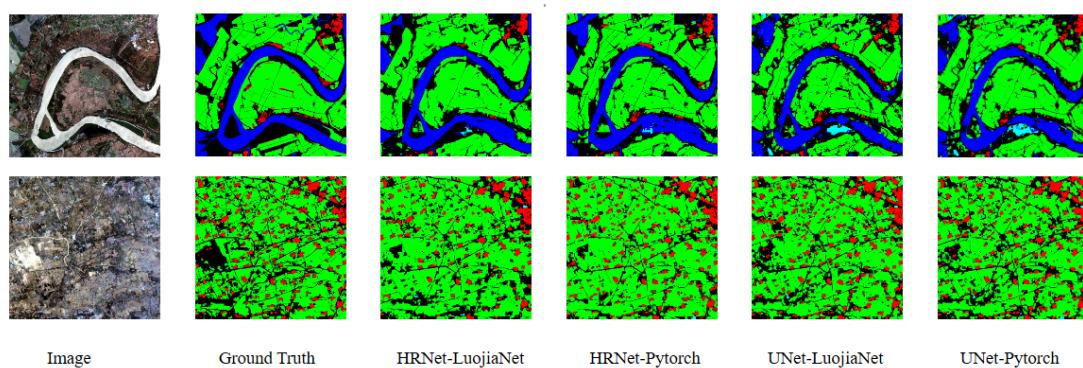
表 7 时间与显存占用对比

	UNet				HRNet			
	Training		Inference		Training		Inference	
	Memor	FP	Memor	FPS	Memor	FP	Memory	FPS
	y	S	y		y	S		

Pytorch	20694	38	7540	11 5	18550	30	5536	94
Luojianet	21996	24	9652	32	19936	15	5556	30
Tensorflo w	23641	31	9510	50	33191	11	7462	25

显存占用以及 FPS 为在 RTX3090、输入数据尺寸为(24,3,512,512)时测得。在精度上，LuoJiaNet 与 PyTorch 差别不大，有微弱优势。在效率上，LuoJiaNet 在内容和处理速度上均不如 PyTorch，但比 Tensorflow 的内存占用小，处理速度相当。

(5) 结果可视化



精度差别不大，预测结果未见明显区别。

4.3.3.4 总结

基于 LuoJiaNET 框架，已经完成了高分辨率遥感地物语义分割的典型应用。在高分辨率语义分割任务中，LuoJiaNET 相比于主流深度学习框架 PyTorch、Tensorflow，具有相当的精度，在效率上低于 Pytorch。

4.3.4 高空间分辨率地物分类（二）

4.3.4.1 任务简介

- (1) 使用 LuoJiaNET 完成/完善常用遥感语义分割深度学习网络 U-Net、DeepLabv3、DeepLabv3+ 的搭建、训练与推理实验，将 LuoJiaNET 框架应用到遥感语义分割任务中；
- (2) 将 LuoJiaNET 和主流的深度学习框架 PyTorch 进行对比，测试 LuoJiaNET 在遥感语义分割任务中的性能表现。

4.3.4.2 高分辨率地物分类网络简介

➤ U-Net 简介

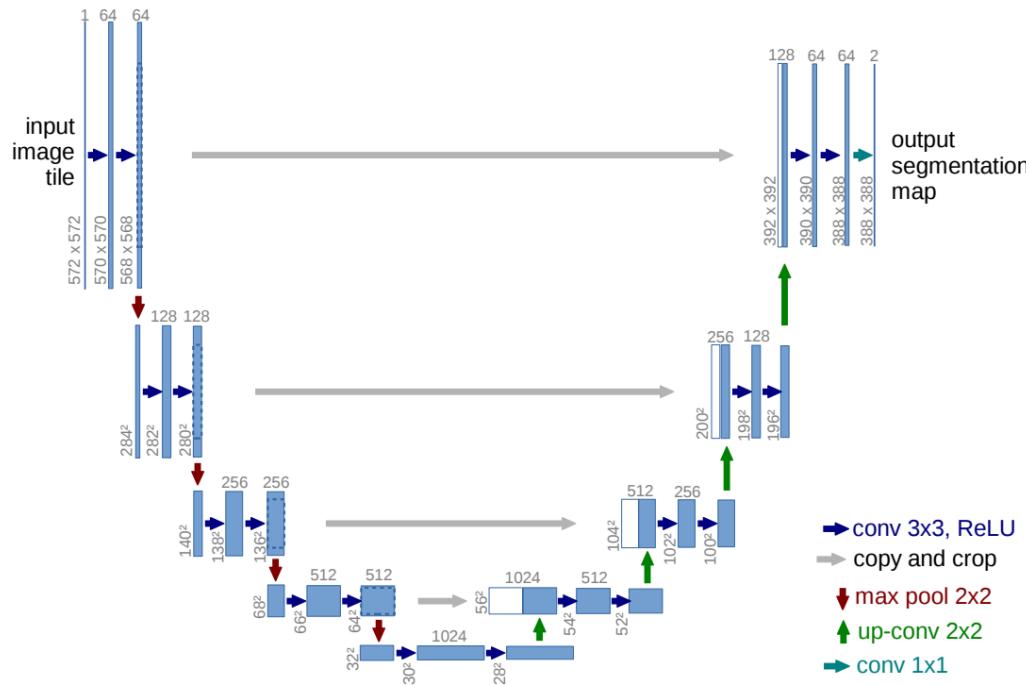


图 1 U-Net 结构图

U-Net 的 U 形结构如图 1 所示。是由卷积和 Max Pooling 构成的一系列降采样操作，压缩路径由 4 个 block 组成，每个 block 使用了 3 个有效卷积和 1 个 Max Pooling 降采样，每次降采样之后 Feature Map 的个数乘 2，因此有了图中所示的 Feature Map 尺寸变

化。最终得到了尺寸为的 Feature Map。同样由 4 个 block 组成，每个 block 开始之前通过反卷积或上采样将 Feature Map 的尺寸乘 2，同时将其个数减半（最后一层略有不同），然后和左侧对称的压缩路径的 Feature Map 合并，由于左侧压缩路径和右侧扩展路径的 Feature Map 的尺寸不一样，U-Net 是通过将压缩路径的 Feature Map 裁剪到和扩展路径相同尺寸的 Feature Map 进行归一化的（即图 1 中左侧虚线部分）。扩展路径的卷积操作依旧使用的是有效卷积操作，最终得到的 Feature Map 的尺寸。

➤ DeepLab 系列简介

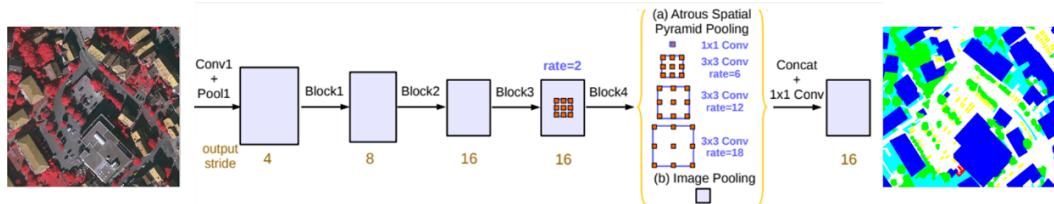


图 2 DeepLabv3 网络结构图

对于 DeepLabv3 网络结构，原始图像经过 encoder 特征提取，逐次下采样 4 次，在第 4 个 block 之后经过 ASPP 模块，经过不同的空洞卷积最后进行特征融合，其经过 1x1 的分类层后直接双线性插值到原始图片大小，输出分类结果。

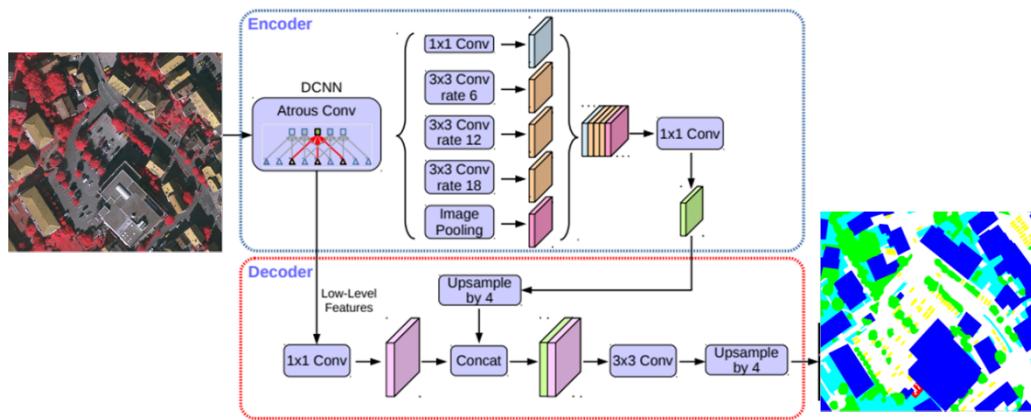


图 3 DeepLabv3+网络结构图

DeepLabv3+模型的整体架构如图 3 所示，它的 Encoder 的主体是带有空洞卷积的 DCNN，可以采用常用的分类网络如 ResNet，然后是带有空洞卷积的空间金字塔池化模块

(Atrous Spatial Pyramid Pooling, ASPP))，主要是为了引入多尺度信息；相比 DeepLabv3, v3+引入了 Decoder 模块，其将底层特征与高层特征进一步融合，提升分割边界准确度。

4.3.4.3 测评结果

(1) 数据集：ISPRS Vaihingen 数据集

ISPRS 提供了城市分类和三维建筑重建测试项目的两个最先进的机载图像数据集。该数据集采用了由高分辨率正交照片和相应的密集图像匹配技术产生的数字地表模型（DSM）。这两个数据集区域都涵盖了城市场景。Vaihingen 是一个相对较小的村庄，有许多独立的建筑和小的多层建筑；Postdam 是一个典型的历史城市，有着大的建筑群、狭窄的街道和密集的聚落结构。每个数据集已手动分类为 6 个最常见的土地覆盖类别，分别是不透水面、建筑物、低矮植被、树木、汽车、背景，其中背景类包括水体和与其他已定义类别不同的物体（例如容器、网球场、游泳池），这些物体通常属于城市场景中的不感兴趣的语义对象。

影像分辨率约为 0.1 米。该数据集包含 33 幅不同大小的遥感图像，每幅图像都是从一个更大的顶层正射影像图片提取的，图像选择的过程避免了出现没有数据的情况。顶层影像和 DSM 的空间分辨率为 9 cm。遥感图像格式为 8 位 TIFF 文件，由近红外、红色和绿色 3 个波段组成。DSM 是单波段的 TIFF 文件，灰度等级（对应于 DSM 高度）为 32 位浮点值编码，具体可视化图像如图 4 所示。

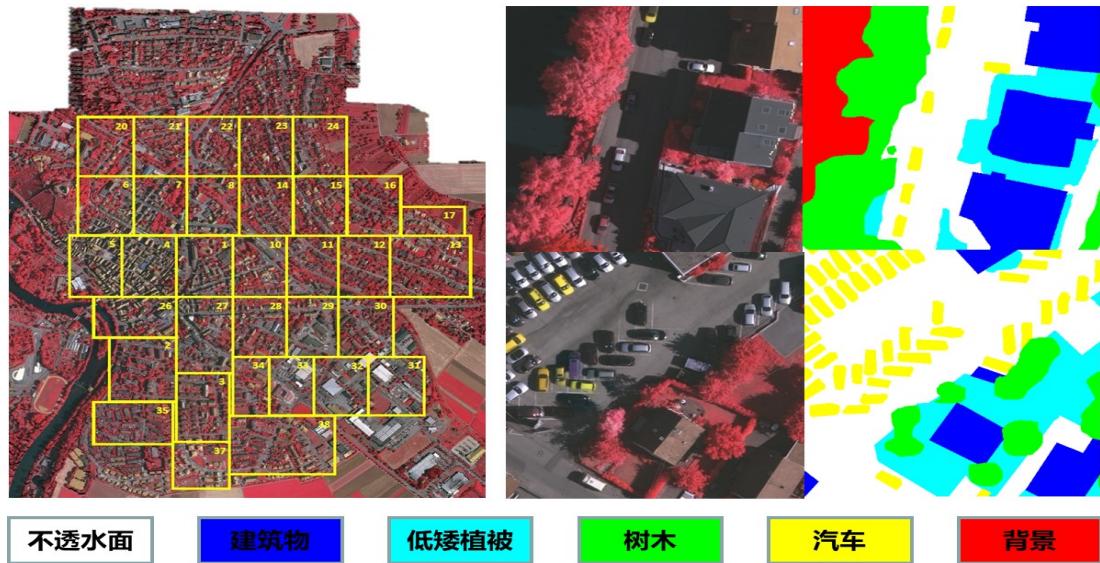


图 4 ISPRS Vaihingen 数据集

从数据集的地物空间分布来看，该数据集的主要挑战如下：

- (1) 地物类别不平衡；
- (2) 不同地物在影像中尺度差异大，光谱相似；
- (3) 目标背景复杂，建筑物阴影遮挡等问题。

(2) 实现细节

在数据预处理时，选用 Vaihingen 数据集中的编号为 1、11、13、15、17、21、23、26、28、3、30、32、34、37、5、7 的 RGB 图像作为训练集，以滑动窗口 512*512，步长 256 对图像裁切，选用 Vaihingen 数据集中的编号为 6、24、35、16、14、22、10、4、2、20、8、31、33、27、38、12、29 的 RGB 图像作为验证集，同样以滑动窗口 512*512，步长 256 对图像裁切。

本次实验使用的 PyTorch 版本是 v1.8.0，LuoJiaNET 版本为 v1.0.0。分别在主流的深度学习框架 PyTorch 和 LuoJiaNET 上复现 U-Net、DeepLabv3 和 DeepLabv3+ 网络。为了进行公平的对比，两网络均在同一硬件上进行训练和测试。GPU 型号为 Nvidia GeForce RTX 3090，CPU 型号为 Intel(R) Xeon(R) Silver 4210R CPU @2.40GHz。

在进行训练时，采用的损失函数为多类交叉熵 CE 损失，优化器选择为 Adam。学习率统一固定为 0.0001，batch size 设置为 8，训练共 200 轮，每轮 43 次迭代。数据增强采用随机水平旋转、垂直旋转、随机 90 度旋转、图像转置。验证时对 398 张 512*512 的图像直接推理验证。

此外在训练中由于背景类是除上述 5 类地物的其他地物类别，因此背景类需要参与到网络训练过程，网络最终输出类别为 6，并参与最终的定量评价。

(3) 评价指标

在实验中，采用的定量指标如下：

a. 精度评价指标(基于混淆矩阵)

各类别的交并比 (IoU)，总体精度 Overall accuracy (OA)，Kappa 系数与平均交并比 (mIoU)。

b. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要的显存占用。

c. 运行时间 (Time)

记录了模型在训练或者推理过程中处理网络一次反向传播和前向传播所需要的时间

(4) 定量实验结果

表 1 三种语义分割网络分类精度结果

方法	框架	IoU(%)						OA	Kappa	mIoU
		不透水面	建筑物	低矮植被	树木	汽车	背景			
Pytorch		78.21	84.27	67.00	75.44	54.78	11.78	86.10	81.61	61.91
U-Net										
LuojiaNET		78.06	83.25	65.20	73.89	59.39	22.80	85.47	80.79	63.77

	Pytorch	75.88	81.61	62.11	73.23	44.19	17.35	84.11	78.99	59.06
DeepLabv3	LuojiaNET	76.71	81.86	63.27	71.54	49.45	13.50	84.25	79.15	59.39
	Pytorch	75.61	82.08	59.45	71.29	41.08	18.98	83.51	78.17	58.09
DeepLabv3+	LuojiaNET	76.37	80.79	62.20	72.05	41.68	10.64	83.80	78.57	57.29

U-Net、DeepLabv3 和 DeepLabv3+ 在三个高光谱分类数据集上的定量结果见表 1，从结果看出，LuojiaNET 与 Pytorch 训练结果总体相差不大，且在部分指标上 LuojiaNET 可取得更高的精度。

表 2 三种语义分割网络运行在两种框架下的效率对比

方法	框架	训练显存(M)	训练时间(s)	测试显存(M)	测试时间(s)
U-Net	Pytorch	16491	0.21	2461	0.01
	LuojiaNET	15345	0.25	2235	0.02
DeepLabv3	Pytorch	12079	0.62	2517	0.02
	LuojiaNET	12273	0.70	1989	0.03
DeepLabv3+	Pytorch	7663	0.21	2289	0.02
	LuojiaNET	7514	0.38	1733	0.02

两种框架在运行效率指标下，LuojiaNET 在大多数网络架构中训练与测试显存占用更少，在训练时间上 LuojiaNET 的训练时间训练较慢，可能的原因如下：（1）两种框架底层的梯度下降优化方式（动态图与静态图计算）不同有关；（2）在 LuojiaNET 自定义单步训练流程需要继承 nn.TrainOneStepCell 类，然后再定义计算损失的 nn.Module 子类计算反向传播的损失，单步的网络反向传播方式不同，该部分可以考虑继续优化。（3）在 LuojiaNET 自定义单步训练流程需要继承 nn.TrainOneStepCell 类，梯度反向传播过程中涉及到数据 cpu 和 gpu 之

间的相互转换，LuoJiaNET 并没有像 Pytorch 一样将数据直接转换成.cuda()和.cpu()格式类型。

可能是以上三个原因导致训练时间。测试时间上相差不大。

(5) 定性可视化

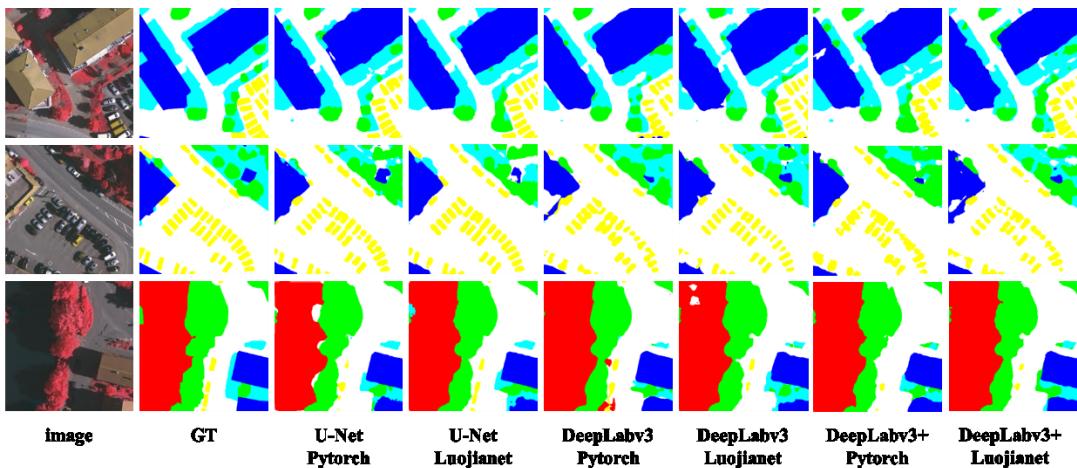


图 5 ISPRS Vaihingen 定性结果可视化

与 Pytorch 相比，LuoJiaNET 在分类边界表现更平滑，椒盐噪声较少。

4.3.4.4 总结

基于 LuoJiaNET 框架，完成高分辨率遥感语义分割任务典型应用。LuoJiaNET 相比于主流深度学习框架 Pytorch，部分精度指标有一定提升，总体相差不大，在运行效率方面，LuoJiaNET 在本报告大多数网络架构中训练与测试显存占用更少，测试时间相差不大，但训练时间不占优，后期可继续优化训练流程。

4.4 变化检测

4.4.1 任务简介

(1) 使用 LuoJiaNET 完成/完善全要素变化检测深度学习网络 DTCDSN、DSIFN 和单要素（建筑物）变化检测网络 BuildingChangeDetection（均已上传 gitee 模型库）的搭建、训练与推理实验，将 LuoJiaNET 框架应用到变化检测任务中；

(2) 将 LuoJiaNET 和主流的深度学习框架 PyTorch 进行对比，测试 LuoJiaNET 在变化检测任务中的性能表现。

4.4.2 变化检测网络简介

4.4.2.1 全要素变化检测网络

DTCDSN[1] (Dual Task Constrained Deep Siamese Convolutional Network) 网络是一个双任务约束的孪生卷积神经网络，它的整个网络包括一个变化检测模块和两个语义分割模块 SSN，引入了双注意力模块 (Dual Attention Module, DAM)，整个网络的输出是一个变化检测图和两时相分割结果图。其中变化检测网络是基于孪生神经网络的，其网络结构具有两个共享权重的编码器，以及一个解码器。采用 SE-ResNet 作为基础的编码器模块。为了利用全局上下文信息，引入了空间特征金字塔池化模块作为中心块，这样可以增大特征图的感受野，并嵌入不同尺度的上下文特征。解码器使用 D-LinkNet，其中每个变化检测块 (CD block) 都有 3 个输入。考虑到来自不同空间位置和不同通道的特征可能会相关，以及受到注意力机制的启发，在解码器部分加入了 DAM 注意力模块，因此可以提高特征提取的辨别能力。

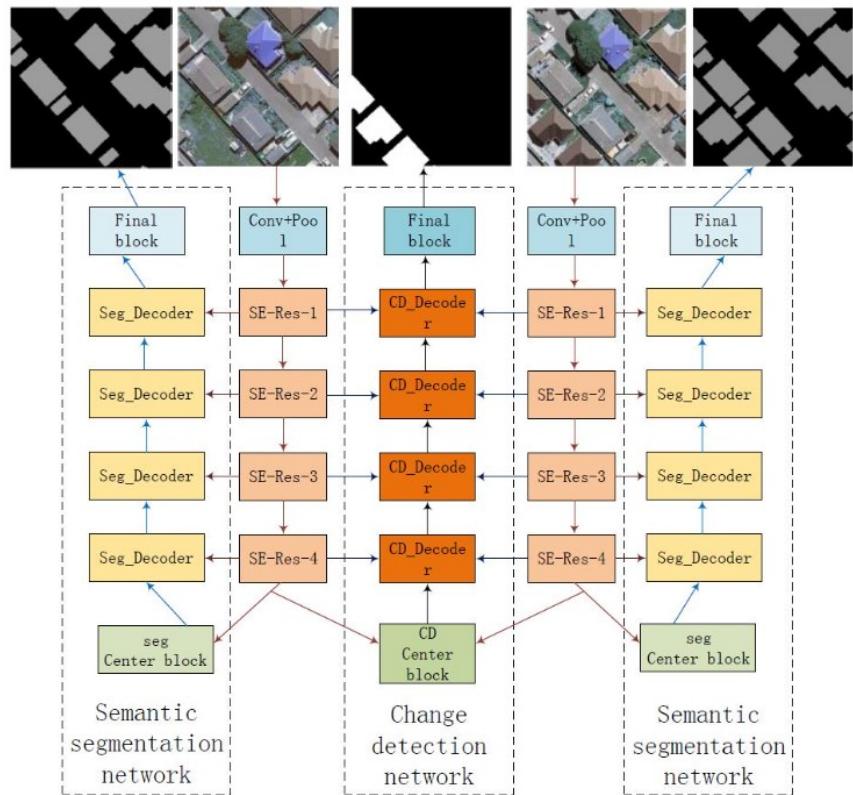


图 1 双任务约束的孪生卷积神经网络 DTCDSCN 结构图

DSIFN[2] (deeply supervised image fusion network) 包括深度特征提取网络 DFEN 和差异判别网络 DDN。首先通过完全卷积的双流架构 (DFEN) 提取具有高度代表性的双时图像深度特征，然后将提取的深度特征输入深度监督的差异判别网络 (DDN) 进行变化检测。为了提高输出变化图中对象的边界完整性和内部紧凑性，模型还增加了注意力模块将原始图像的多层深度特征与图像差异特征融合，通过直接向网络中间层引入变化图损失来进一步增强 DDN，并以端到端的方式训练整个网络。

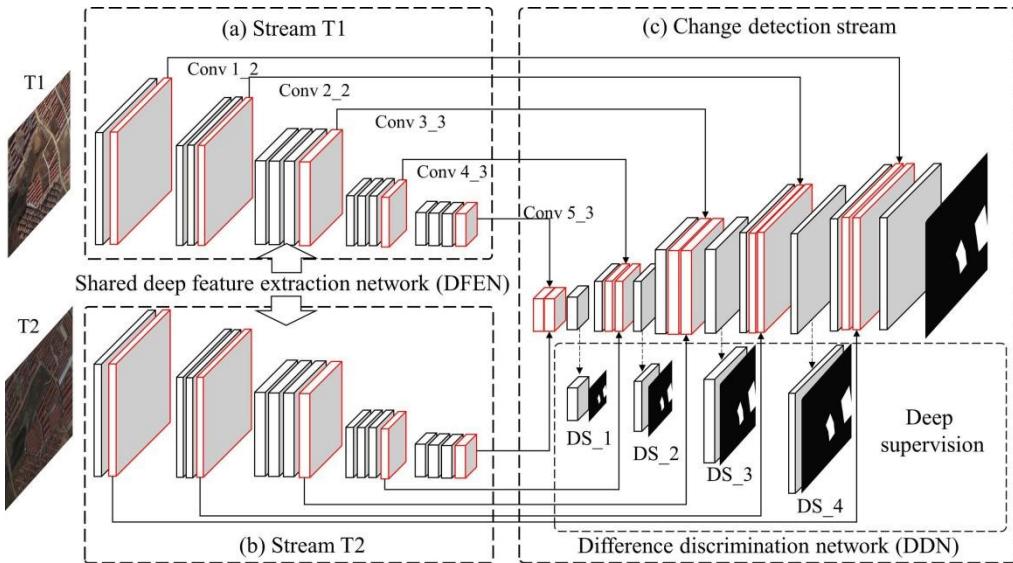


图 2 深度监督图像融合网络 DSIFN 结构图

4.4.2.2 单要素变化检测网络

BuildingCD 是一个基于建筑检测先验知识的端到端变化检测方法，是一种基于迁移学习思想的新型多任务网络，该网络主要包括建筑物提取分支和变化检测分支，并加入了对象级细化算法。该网络利用建筑提取数据集对建筑提取分支进行预训练，使网络的特征提取模块能够更好地提取建筑特征。因此，后续网络训练只需要少量的建筑物变化检测样本也可以达到优异的检测效果。并且为充分利用建筑提取分支的结果提高变化检测精度，提出一种对对象级细化算法。结合变化检测分支和建筑物提取分支的结果，选择变化面积大于预定义阈值的建筑掩膜作为最终变化检测结果，提高了变化检测结果的准确性和视觉效果。与其他多任务变化检测网络相比其优点在于：该网络借助高精度建筑掩膜可以充分利用来自建筑检测分支的先验信息，并可通过其对象级细化算法进一步提高变化检测结果。

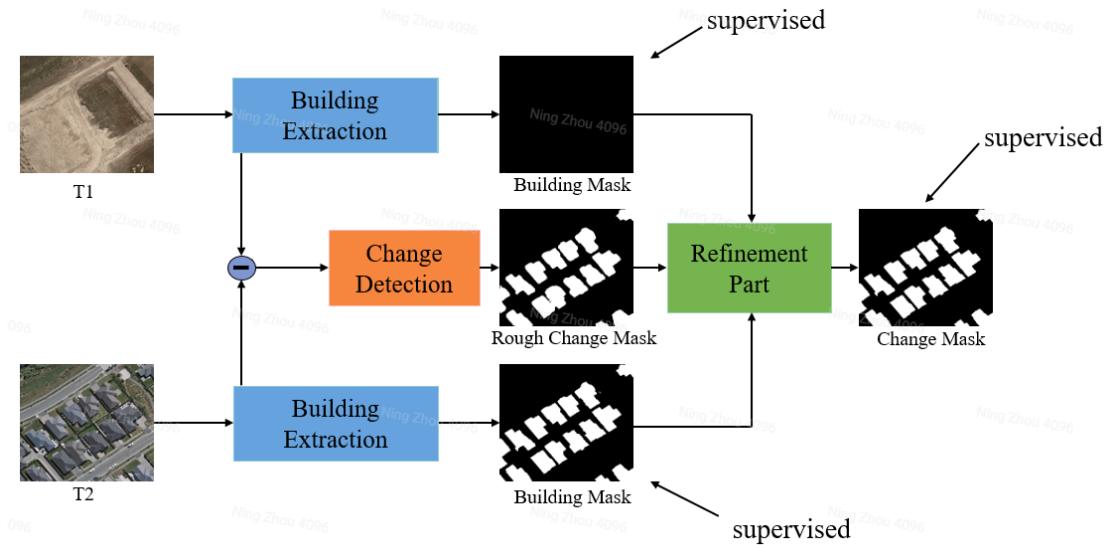


图 3 BuildingCD 网络结构图

4.4.3 测评结果

(1) 变化检测数据集：WHU-BCD 数据集

WHU-BCD 数据集是一个大规模建筑物变化检测数据集，整个数据集涵盖了 2011 年 2 月发生 6.3 级地震并在随后几年重建的区域。该数据集由 2012 年 4 月获得的航拍图像组成，包含 20.5 平方公里内的 12796 座建筑物（2016 年数据集中同一区域的 16077 座建筑物）。通过在地表手动选择 30 个 GCP，子数据集被地理校正为航空数据集，精度为 1.6 像素。每个样本包含一个前时期影像、后时期影像、前时期建筑物 mask，后时期建筑物 mask 以及发生变化的二值 label。为了方便深度学习的训练与测试，样本的尺寸已经被统一裁剪为 512*512 大小的图像，影像分辨率约为 0.1 米。

(2) 实现细节

PyTorch 版本：v1.6.0

LuoJiaNET：v1.0.0

GPU 型号：GeForce GTX 3080(10G)

CPU 型号: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz

数据集大小: 1500 张大小为 $512 \times 512 \times 3$ 的影像对

优化器: Adam

学习率: 0.001, batch_size: 5, epoch: 500

(3) 评价指标

在实验中, 采用的定量指标如下:

a. 精度评价指标(基于混淆矩阵)

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

$$\text{IoU} = \frac{TP}{TP + FP + FN} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$F1 = \frac{(Precision \times Recall \times 2)}{(Precision + Recall)} \quad (4)$$

b. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要的显存占用。

c. 运行时间 (Time)

记录了模型在训练或者推理过程中处理一个样本所需要的时间。

(4) 实验情况

➤ DTCDSN 网络

表 8 DTCDSN 精度结果对比

Methods	Precision	Recall	F1	IoU
DTCDSN (LuojiaNET)	88.210	97.381	0.89894	85.999
DTCDSN (PyTorch)	72.052	93.444	0.81365	81.203

从表 1 和图 4 可以看出, Luojianet 环境下, 模型各项指标均高于 pytorch 环境, 且均

达到 85%以上，从视觉效果上来看 luojanet 环境下的模型提取效果更优，pytorch 环境下的模型漏检高，且边界不完整，而在 luojanet 环境下该模型能够提取出更为完整、边界更准确的变化建筑。

表 9 DTCDSNC 时间与显存占用对比(batch_size=1)

Methods	Training		Inference	
	Memory		Memory	
	(MiB)	Time (s)	(MiB)	Time (s)
DTCDSNC(LuoJiaNET)	4372	82.3	2781	0.16
DTCDSNC (PyTorch)	5091	136.9	3522	0.211

使用 LuoJiaNET 框架进行训练，内存消耗比 PyTorch 略小，训练时间减少接近一半；在推理过程中，LuoJiaNET 推理速度稍快，内存占用较 Pytorch 也小。

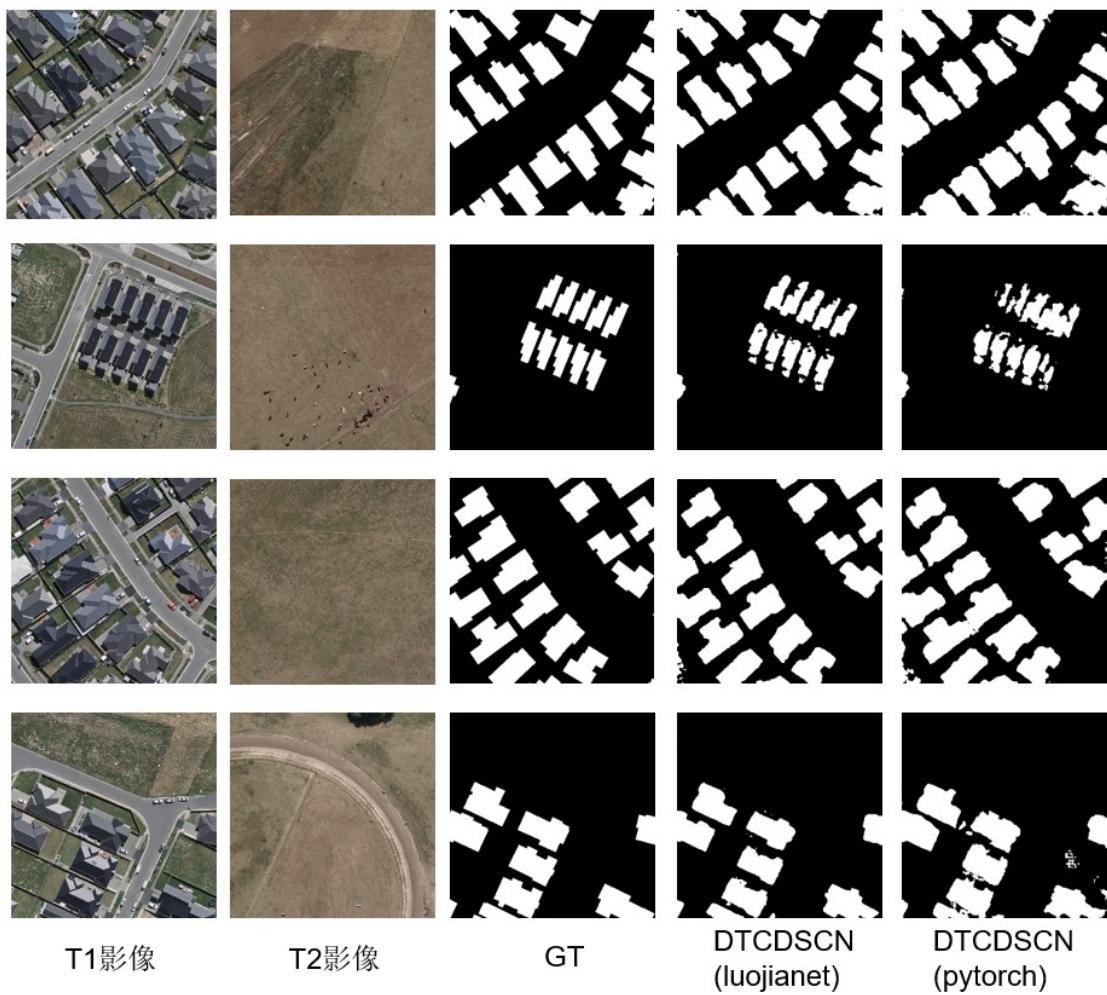


图 4 可视化结果

➤ DSIFN 网络

表 10 DSIFN 精度结果对比

Methods	Precision	Recall	F1	IoU
DSIFN (LuojiaNET)	95.896	89.910	0.9083	86.687
DSIFN (PyTorch)	96.552	80.517	0.8708	78.267

从表 3 和图 5 可以看出，Luojianet 环境下，模型虚检略高于 pytorch 环境，但 IoU 和 F1 远超 pytorch 环境，能够提取出更为完整、边界更准确的变化建筑，Luojianet 环境下的 DSIFN 整体提取效果要优于 pytorch 环境。

表 11 DSIFN 时间与显存占用对比(batch_size=1)

Methods	Training		Inference	
	Memory	Time (s)	Memory	Time (s)
	(MiB)		(MiB)	
DSIFN(LuoJiaNET)	4540	253.1	3459	0.103
DSIFN (PyTorch)	4664	425.65	2530	0.158

从表 4 可以看出，使用 LuoJiaNET 进行训练，内存消耗比 PyTorch 略小，训练时间减少接近一半；在推理过程中，LuoJiaNET 推理速度稍快，但是内存消耗较 Pytorch 占用更多(暂未更新到最新版 luojianet)。

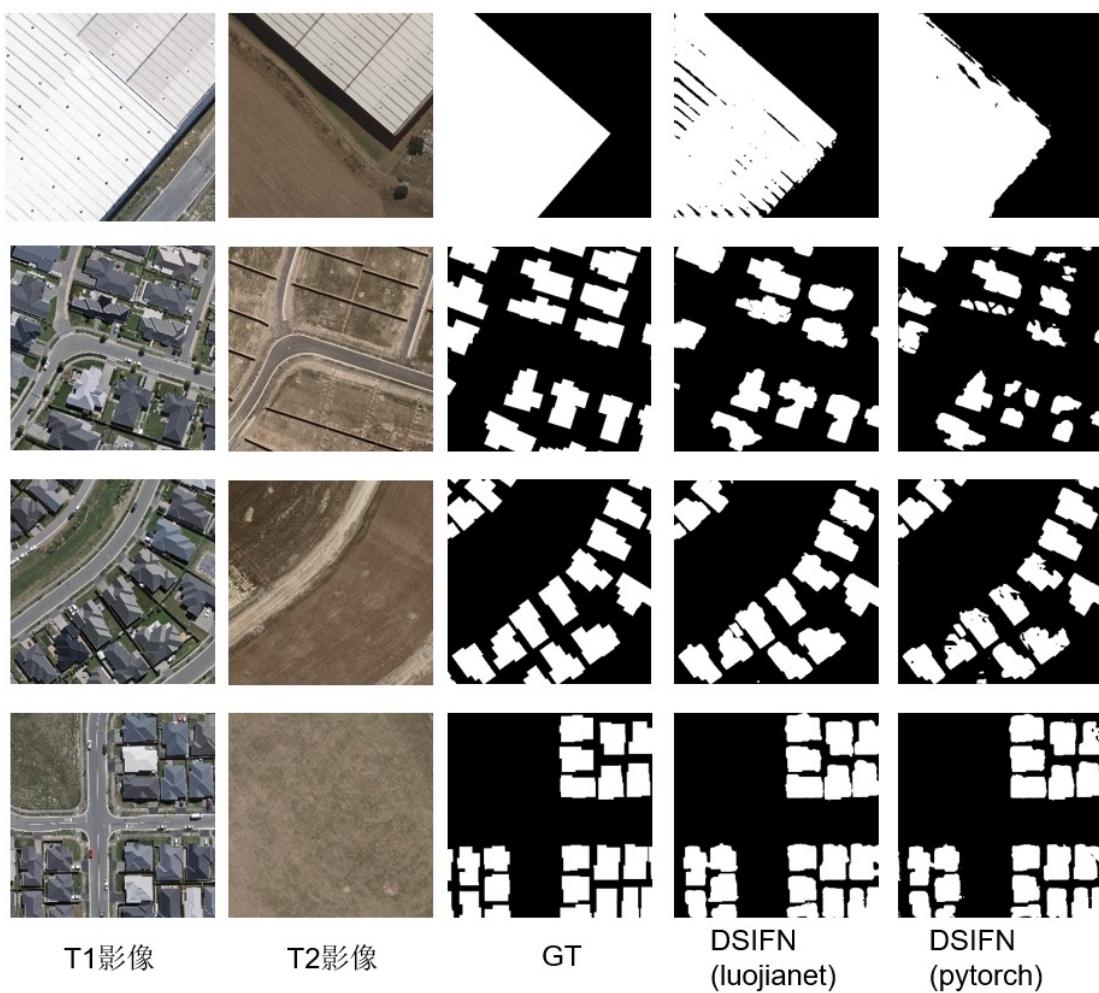


图 5 DSIFN 可视化结果

➤ BuildingCD 网络

表 5 BuildingCD 精度结果对比

Methods	Precision	F1	Recall
BuildingCD (LuoJiaNET)	87.030	0.85803	95.933
BuildingCD (PyTorch)	94.570	0.9374	92.930

从表 5 和图 6 可以看出，使用 LuoJiaNET 进行训练后的模型进行推理评估，模型召回率高于 pytorch 环境，且各项指标均达到 85% 以上，反映出 LuoJianet 环境下的该模型虚检稍高，但 luojianet 下的模型比 Pytorch 环境下的模型漏检更少，整体检测效果更优。

表 6 BuildingCD 时间与显存占用对比(batch_size=1)

Methods	Training		Inference	
	Memory (MiB)	Time (s)	Memory (MiB)	Time (s)
BuildingCD (LuoJiaNET)	5008	\	2781	\
BuildingCD (PyTorch)	5155	\	2436	\

从表 6 可以看出，使用 LuoJiaNET 进行训练，内存消耗比 PyTorch 略小；但在推理过程中，显存消耗较 Pytorch 占用更多，暂未更新 1.0.5 版本测试，更新后将继续进行测试。

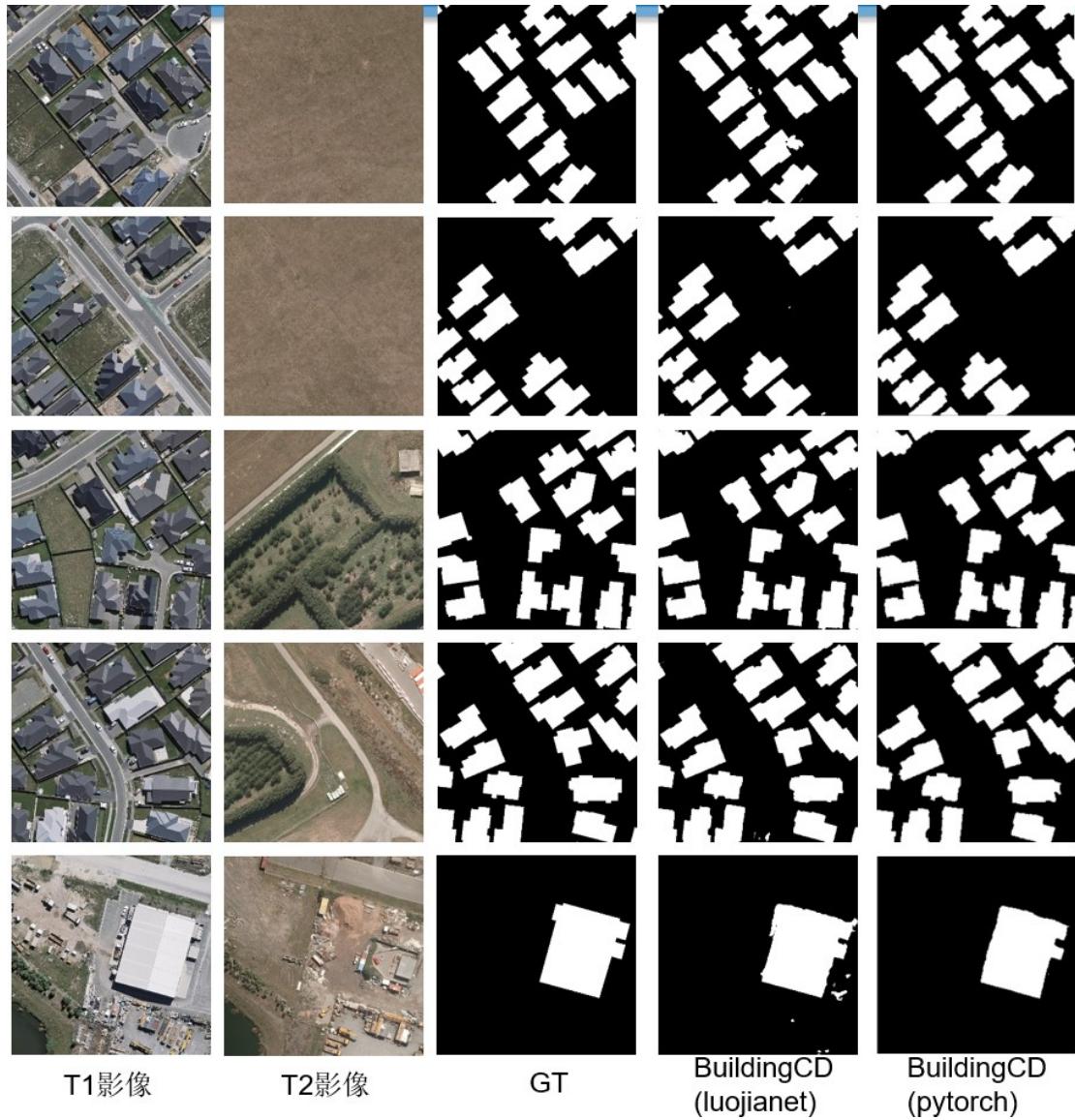


图 6 可视化结果

4.5 多视三维

4.5.1 双目立体

4.5.1.1 任务简介

- (1) 使用 LuoJiaNET 完成经典的深度学习双目立体匹配网络 GC-Net 的搭建与训练，将 LuoJiaNET 框架应用到立体匹配与三维重建任务中；
- (2) 基于搭建的 GC-Net 模型，将 LuoJiaNET 和主流的深度学习框架 PyTorch 以及

Tensorflow 进行对比，测试 LuoJiaNET 在立体匹配任务中的性能表现。

4.5.1.2 双目立体匹配网络 GC-Net 简介

GC-Net [1] (Geometry and Context Network)是一个经典的深度学习双目立体匹配模型，它以端到端的方式完成了双目密集匹配的过程，以纠正后的核线立体像对为输入，由网络学习特征映射得到视差图，无需任何手工设计特征和后处理就能达到子像素级别的立体匹配，大大减少工程设计的复杂性。

GC-Net 的组成部分包含**特征提取**模块、**代价体构建**模块、**代价体正则化**模块、**回归**模块四大部分组成。在特征提取模块中，网络通过共享权重的一系列 2D 卷积对左右核线影像对进行高维特征的提取。在代价体构建模块，将左右影像特征进行“错位叠合”，形成一个形状为 $W \times H \times D$ 代价体（其中，W、H 和 D 分别代表了影像的宽度、高度与最大视差）。其中，“错位叠合”的具体含义是，重复将右影像特征沿着视差方向移动，再与左影像特征进行串联，每次移动一个像素，直到达到最大视差处停止。在代价体正则化模块中，利用多尺度的 3D 卷积和 3D 转置卷积对代价体进行正则化，融合空间方向上和视差搜索方向上的特征信息，进行代价聚合。在回归模块，利用 soft argmin 运算从正则化后的代价体中得到归一化的概率体，以归一化的概率值作为权重，对搜索范围内的每个视差值 d 进行加权求和，从而回归出平面视差图。最后利用 L1 范式将网络得到的平面视差图与输入的真实视差图比较，比较结果用于迭代训练模型。GC-Net 网络以三维特征的形式同时考虑了图像平面特征和视差值，具有极高的鲁棒性和准确性，是目前大多数立体匹配方法的基础架构。

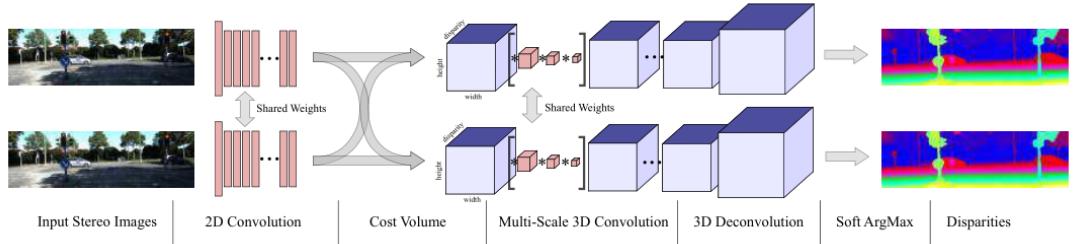


图 6 双目立体匹配网络 GC-Net 结构图

4.5.1.3 测评结果

(1) 数据集 LuoJiaSET-多视三维数据集

采用 LuoJiaSET-多视三维数据集中的 WHU-Stereo 子数据集。WHU-Stereo 是一个大规模合成航空影像双目立体匹配数据集。整个数据集包含训练集和测试集，其中训练集和测试集包含的样本数分别是 8316 组和 2618 组。每一个样本包含左影像、右影像以及左影像对应的视差图。为了便于深度学习的训练与测试，样本的尺寸已经被统一裁剪为 768×384 像素。影像数据的分辨率约为 10 cm。

(2) 实现细节

我们分别在主流的深度学习框架 PyTorch、Tensorflow 以及我们的 LuoJiaNET 上复现了 GC-Net 网络，在后文中分别简写为 GC-Net (PyTorch)、GC-Net (Tensorflow) 与 GC-Net (LuoJiaNET)。其中使用的 Pytorch 版本是 V1.8.0，使用的 Tensorflow 版本是 V1.13.1，使用的 LuoJiaNET 版本是 V1.0.0。为了进行公平的对比，三个网络均在同一硬件上进行训练和测试，所有超参数和网络架构均保持一致。GPU 型号为 GeForce GTX 1080Ti (11G)、CPU 型号为 Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz，使用的 CUDA 的版本是 V10.1。在进行训练时，最大视差设置为 160，每个样本被随机裁剪为 512×256 像素的小块，

采用的损失函数为 L1 损失，优化器选择为 RMSProp，学习率固定为 0.001，批次数设置为 1，训练共 20 个轮次，共计约 17 万次迭代。推理时的参数设置与训练过程保持一致，最大视差设置为 160，样本大小为 512×256 像素。

(3) 评价指标

在实验中，采用的定量指标如下：

a. 平均绝对误差 (Mean Absolute Error, MAE)

统计模型预测视差图与视差图真值之间绝对差异的平均值，计算方式如下：

$$\frac{1}{m} \sum_{i=1}^m |y_i - f(x_i)| \quad (1)$$

其中， y_i 为视差真值，而 $f(x)$ 为视差估计值，其中 m 为影像中有效像素的数量， x 代表了影像上的像素。

b. n 像素精度占比 (< n -pixel)

统计模型预测视差图与视差图真值之间的绝对差异小于 n 个像素的像素数量占比，计算方式如下：

$$\frac{\sum_{i=1}^m [|y_i - f(x_i)| < n]}{\sum_{i=1}^m 1} \quad (2)$$

其中，[] 为艾佛森括号，当括号内的条件满足时为 1，否则为 0.

c. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要占用的峰值显存。

d. 运行时间 (Time)

记录了模型在训练或者推理过程中处理一个样本所需要的时间。

(4) 实验结果

表 12 精度结果对比

Methods	MAE(pixel)	<1-	<2-pixel(%)	<3-pixel(%)
		pixel(%)		
GC-Net (LuoJiaNET)	0.176	98.12	98.98	99.31
GC-Net (PyTorch)	0.202	97.88	98.83	99.20
GC-Net(Tensorflow)	0.235	96.87	98.16	98.67

从表 1 可知,相比于 PyTorch 和 Tensorflow 构建的模型,使用 LuoJiaNET 构建的 GC-Net 网络在精度上均有所提升。与 PyTorch 构建的模型相比,在平均绝对误差 (MAE) 指标上提升约 12%,在一像素 (<1-pixel)、两像素 (<2-pixel)、三像素 (<3-pixel) 精度占比指标上分别提升约 0.24%、0.15%和 0.11%。相比于 Tensorflow 模型, LuoJiaNET 构建的 GC-Net 网络在精度上也有优势,在平均绝对误差 (MAE) 指标上提升约 25%,在一像素(<1-pixel)、两像素(<2-pixel)、三像素(<3-pixel) 精度占比指标上分别提升约 1.25%、0.82%和 0.64%。

表 2 时间与显存占用对比

Methods	Training		Inference	
	Memory (MiB)	Time (s)	Memory	Time (s)
			(MiB)	
GC-Net (LuoJiaNET)	7587M	1.63s	4515M	0.28s

GC-Net (PyTorch)	8695M	2.75s	3085M	0.45s
GC-Net(Tensorflow)	8671M	0.83s	4575M	0.27s

从表 2 可以看出，使用 LuoJiaNET 进行训练时，内存消耗比 PyTorch 以及 Tensorflow 略小，在本模型上减少约 12%；在本模型上训练时，训练时间相比于 PyTorch 减少约 33%，相比于 Tensorflow 的训练时间增加约 1 倍。在推理过程中，显存占用较 PyTorch 要略高，与 Tensorflow 的占用基本持平；LuoJiaNet 构建的模型的推理时间与 Tensorflow 基本持平，相比于 PyTorch，推理时间减少约 37%。

(5) 结果可视化

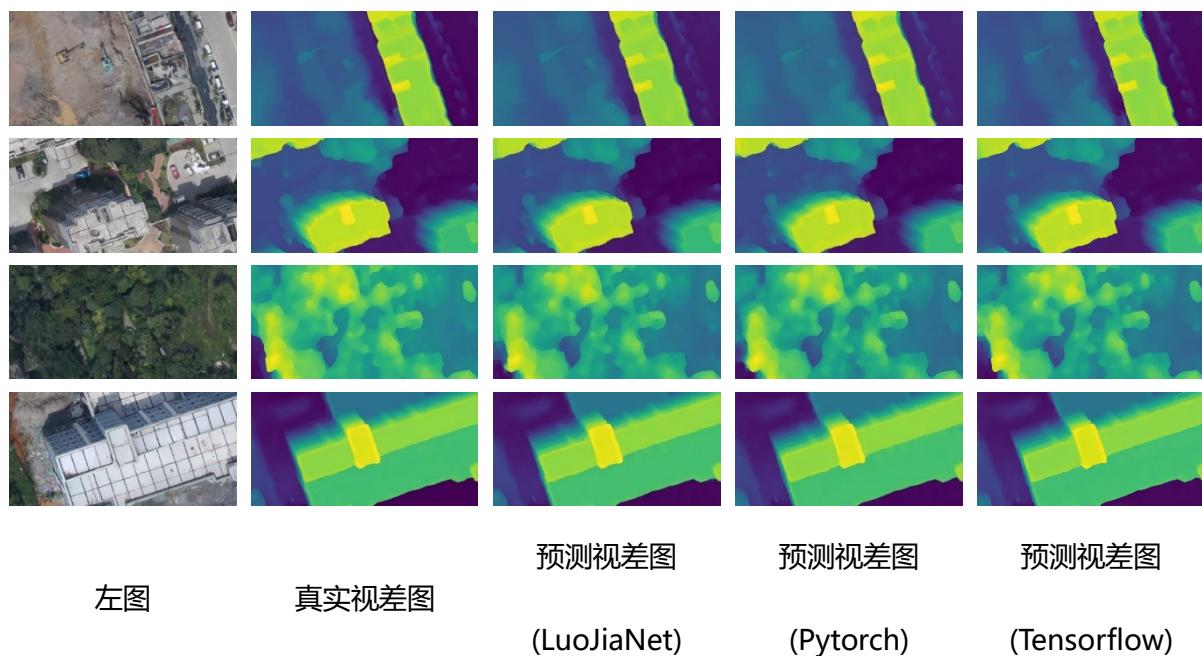


图 2. GC-Net 模型推理的视差图可视化结果。从上到下，分别是不同的样本，样本编号分别为 009_53/006003、009_53/009010、009_59/005006 与 012_38/011008。从左到右，分别是左影像、视差图真值、GC-Net (LuoJiaNET) 模型的预测结果、GC-Net (PyTorch) 的预测结果和 GC-Net (Tensorflow) 模型的预测结果。

4.5.1.4 总结

基于 LuoJiaNET 框架，实现了立体匹配网络，完成了在三维重建任务上的典型应用。在立体匹配与三维重建任务中，基于复现的双目立体匹配网络 GC-Net，在 WHU-Stereo 数据集中的实验证明了：LuoJiaNET 相比于主流深度学习框架 PyTorch，在结果精度上，提高约 10%~20% 左右；在训练和推理的时间占用上，降低约 30% 左右；在训练的显存占用上，减少约 10% 左右，但是在推理时，显存占用较 PyTorch 略高。相比于静态图模型的 Tensorflow，在结果精度上提高约 20%~30%，LuoJiaNET 在训练时的显存占用略高，训练时间增加约 1 倍，在推理时，二者的显存占用和时间占用基本持平。

4.5.2 多视密集匹配

4.5.2.1 任务简介

(1) 使用 LuoJiaNET 完成经典的深度学习多视密集匹配网络 MVSNet 的搭建与训练，将 LuoJiaNET 框架应用到多视立体匹配与三维重建任务中；
(2) 基于搭建的 MVSNet 模型，将 LuoJiaNET 和主流的深度学习框架 PyTorch、TensorFlow 进行对比，测试 LuoJiaNET 在多视密集匹配任务中的性能表现。

4.5.2.2 多视密集匹配网络 MVSNet 简介

MVSNet[1](Multi-view Stereo Network) 是一种经典的端到端深度学习多视密集匹配模型，用于对具有多视角影像的场景的密集匹配，不借助于核线影像和视差值，而直接从原始多视影像出发，以一张中心影像和几张辅助影像，以及与影像对应的相机位姿参数作为输入，得到中心影像对应的深度图。

MVSNet 组成部分包括特征检测模块、代价体构建模块、代价体规则化模块、回归模块、深度精化模块五个部分组成。在**特征检测模块**中，通过一个由八个卷积层构成的 2D CNN 模块提取输入的 N 张影像的深层特征表示，最终得到下采样四倍后的 N 个 32 通道的影像特征图。在**代价体构建模块**，利用相机参数将特征图构建为以深度为第三维的 3D 代价体。利用相机参数将特征图构建为以深度为第三维的 3D 代价体。所有特征图转换为在不同深度 d 处的一系列互相平行的平面 F_i 。第 i 张特征图到深度 d 处的平面的坐标映射转换关系由 3×3 的单应矩阵 $H_i(d)$ 表示：

$$H_i(d) = K_i \cdot R_i \cdot \left(I - \frac{(t_1 - t_i) \cdot n_1^T}{d} \right) \cdot R_1^T \cdot K_1^T$$

每张特征图在不同深度 d 处得到一张平面特征，给定多个 d 值可得到多个特征图对应的特征体 V_i 。对于 N 张不同视角的输入影像，以方差的形式计算相似性测度，计算代价体 C 。即：

$$C = M(V_1, \dots, V_N) = \frac{\sum_{i=1}^N (V_i - \bar{V}_i)^2}{N}$$

代价体的计算过程遵循等权原则，即所有输入影像具有同等重要的地位。

在**代价体规则化模块**中，通过一个 4 层的 3D U-Net 进行规则化。3D U-Net 结构引入了多尺度的卷积操作，通过编码器-解码器结构聚合不同尺度的空间方向上和深度搜索方向上的特征信息，进行代价聚合。在 3D 卷积的最后一层，输出为 1 通道的 3D 特征代价体。在**深度回归模块**中，沿着深度方向应用 softmax 函数将规则化后的代价体转换为概率体，概率体中的每个值表示对应像素点的深度为当前高深度图 D_i 对应深度值的概率，以归一化的概率值作为权重，对搜索范围内的每个深度值 d 进行加权求和，得到连续的深度预测值。

深度精化模块采用 4 个卷积层组成的残差结构，用于利用影像的特征信息，对得到的初始深度图进行边缘精化。训练 MVSNet 网络采用的 Loss 为以中心影像对应的地面真实深度值与网络估计的深度值之间绝对差值的平均值。当地面真实深度图存在黑洞时，只考虑具有有效

标签的像素点。MVSNet 将多视成像几何显式地编码进深度学习网络中，从而在几何约束条件的支持下，端到端的实现多视影像间的密集匹配任务，是现有通用的多视密集匹配方法的基础和核心架构。

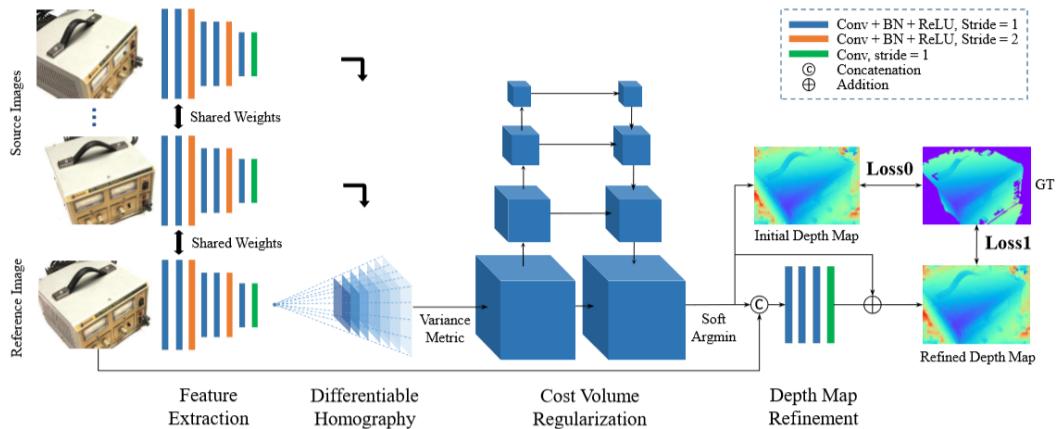


图 1 MVSNet 网络结构示意图

4.5.2.3 测评结果

(1) 数据集 LuojiaSET-多视三维数据集

采用 LuojiaSET-多视三维数据集中的 WHU-MVS 数据集。WHU-MVS 是一个大规模合成航空影像多视密集匹配数据集。用于模型训练的数据有 45 组，测试集有 17 组，每组包括 80 对五视匹配数据样本，每一个样本包含五个相邻视角、彼此具有一定重叠度的影像，以及每张影像对应的相机参数文件和真实深度图。为了便于深度学习的训练与测试，样本的尺寸已经被统一裁剪为 768×384 像素。影像数据的分辨率约为 10 cm。在训练和测试中，每张影像的深度采样数量固定为 200 个采样平面，训练视角数量固定为 3.

(2) 实现细节

参与实验的深度学习框架有 PyTorch、Tensorflow 和 LuojiaNET，这三个框架上实现

的 MVSNet 网络分别表示为 MVSNet (PyTorch)、MVSNet (Tensorflow)与 MVSNet (LuoJiaNET)。其中使用的 PyTorch 版本是 V1.1.0，使用的 Tensorflow 版本是 V1.13.1，使用的 LuoJiaNET 版本是 V1.0.0。三个网络均在同一硬件上进行训练和测试，所有超参数和网络架构均保持一致。GPU 型号为 GeForce GTX 1080Ti (11G)、CPU 型号为 Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz，使用的 CUDA 的版本是 V10.1。

在训练阶段，采用的损失函数为 Smooth L1 损失，优化器选择为 RMSProp，学习率固定为 0.001，批次数设置为 1，训练共 30 个轮次，每个轮次有 3600 对样本，共计约 11 万次迭代。在训练和推理阶段，所有超参数保持一致：参与匹配的影像视角数量固定为 N=3，每张影像的大小为 738×384 像素，深度采样数量为 D=200，每张影像的最大和最小深度搜索范围采用该影像实际覆盖到的深度范围，MVSNet 网络输出深度图的分辨率是原始输入大小的 1/16，利用双线性上采样函数将结果采样为真实深度图大小，在同一个标准下进行精度评定。

(3) 评价指标

在实验中，采用的定量指标如下：

a. 平均绝对误差 (Mean Absolute Error, MAE)

表示深度估计值与地面真实深度值间的绝对差值的平均值，为剔除极大粗差对距离测度的影响，仅对偏差在 100 个采样间隔范围内的误差求均值：

$$\frac{1}{m} \sum_{i=1}^m |y_i - f(x_i)| \quad (1)$$

其中， y_i 为深度真值，而 $f(x_i)$ 为深度估计值，其中 m 为影像中有效像素的数量， x 代表了影像上的像素。

b. 固定间隔精度占比 (n)

统计模型预测深度图与真实深度图之间的绝对差异小于采样间隔 n 的像素数量占所有像素点的百分比，计算方式如下：

$$\frac{\sum_{i=1}^m [|y_i - f(x_i)| < n]}{\sum_{i=1}^m 1} \quad (2)$$

其中，[]为艾佛森括号，当括号内的条件满足时为 1，否则为 0.

c. 显存占用 (Memory)

记录了模型在训练或者推理过程中所需要占用的峰值显存。

d. 运行时间 (Time)

记录了模型在训练或者推理过程中处理一个样本所需要的时间。

(4) 实验结果

表 13 精度结果对比

Methods	MAE(m)	< 0.1m (%)	<0.3m(%)	<0.6m(%)
MVSNet (LuoJiaNET)	0.218	56.31	86.00	93.46
MVSNet (PyTorch)	0.195	54.79	88.51	95.03
MVSNet(TensorFlow)	0.182	59.63	91.53	95.67

从表 1 可知，在 LuoJiaNET、PyTorch、Tensorflow 三个框架上构建的 MVSNet 模型，以 Tensorflow 的精度最高，在各个指标上相比其余二者均有明显优势。与 PyTorch 相比，使用 LuoJiaNET 构建的 MVSNet 网络仅在 < 0.1m (%) 的精度上稍有优势。总体提升不明显。

表 2 时间与显存占用对比

Methods	Training		Inference	
	Memory (MiB)	Time (s)	Memory (MiB)	Time (s)
MVSNet (LuoJiaNET)	11173M	1.5s	10661M	0.63s
MVSNet (PyTorch)	5901M	0.57s	2629M	0.13s
MVSNet(TensorFlow)	9255M	0.78s	3115M	0.34s

从表 2 可以看出，使用 LuoJiaNET 进行训练时，内存占用和时间消耗均比 PyTorch 以及 Tensorflow 高：其中显存占用比 PyTorch 高 1.9 倍，比 TensorFlow 高约 1.2 倍；时间消耗比 PyTorch 高 2.6 倍，比 TensorFlow 高约 1.9 倍。使用 LuoJiaNET 进行推理时，LuoJiaNET 的显存占用和时间消耗最高，PyTorch 的最低，其中时间消耗比 PyTorch 高约 5 倍，比 TensorFlow 高约 1.9 倍。

问题分析：

在本模型的实验中，使用 LuoJiaNET 构建的 MVSNet 模型在训练和推理的显存和时间占用上均表现较差，因此对 MVSNet 模型进行逐模块拆分分析问题所在。

MVSNet 主要由特征检测模块、代价体构建模块、代价体聚合模块、回归模块、深度精化模块五个部分组成。其中前三个模块是网络的核心结构，我们单独统计了这三个子模块各自的峰值显存占用和时间消耗情况，如表 3 所示。

表 3 推理阶段 3 个主要子模块峰值显存与时间占用情况

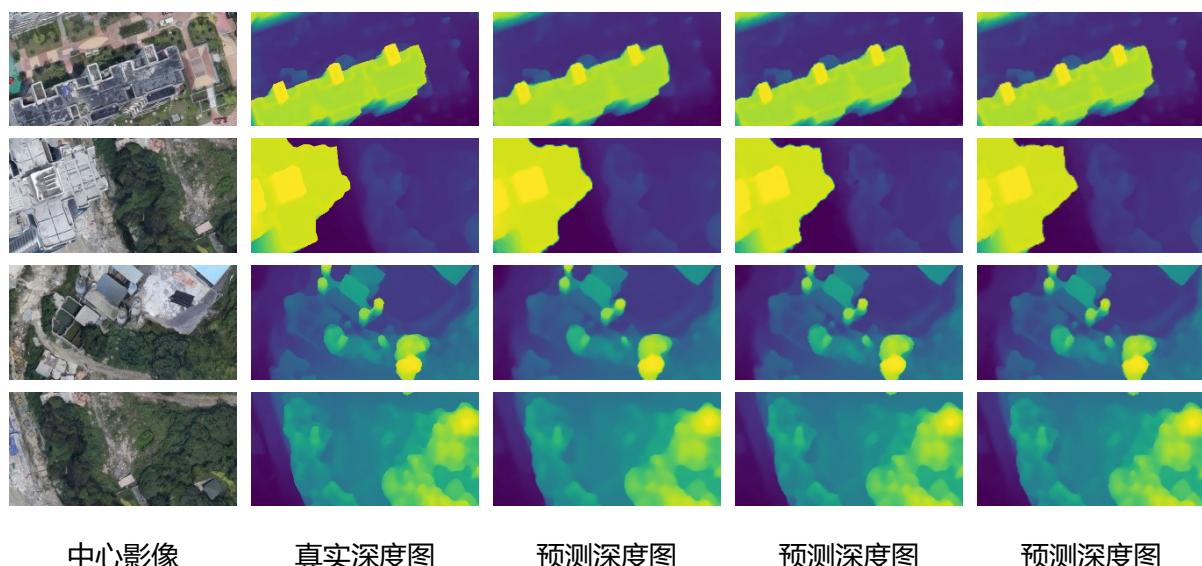
模块 Modules	显存 Memory (MiB)	时间 Times(s)*

2D 特征提取模块	2467 M	0.014 s
代价体构建模块 (with homo. Warping)	10661 M	0.607 s
3D 代价体聚合模块	5539 M	0.086 s

*运行时间是取 100 个样本的平均时间。

与表 2 对比之后可以看出，MVSNet(LuoJiaNET)的时间和显存峰值均出现在代价体构建模块。该模块主要用于多视特征几何对齐，给定一系列深度采样平面，该模块通过单应映射变换将一组相邻影像的 2D 特征图投影变换至中心影像视角下，并计算多个不同视角特征图间的方差作为代价值。其中涉及到**矩阵相乘、矩阵相除、矩阵求逆、规则格网重采样等操作**，目前在 LuoJiaNET 里的模型实现这一过程采用的方式较 PyTorch 更加复杂，因此影响了模型的总体性能。该部分具有很大的提升空间，随着 LuoJiaNET 对一些算子的支持更加全面，我们对一些函数和运算进行精简，该问题有望被很好地优化。

(5) 结果可视化



(LuoJiaNet) (Pytorch) (Tensorflow)

图 3. MVSNet 模型推理的深度图可视化结果。从上到下，分别是不同的样本，样本编号分别为 009_53/000008、009_56/001009、009_59/000005 与 009_59/001005。从左到右，分别是中心影像、深度图真值、MVS-Net (LuoJiaNET)模型的预测结果、MVS-Net (PyTorch)模型的预测结果和 MVS-Net (Tensorflow)模型的预测结果。

4.5.2.4 总结

在立体匹配与三维重建典型应用任务中，分别基于 LuoJiaNET、PyTorch、Tensorflow 框架复现了经典的多视密集匹配网络 MVSNet。在该模型上，LuoJiaNET 相比其余两者未表现出优势，甚至在显存和时间占用上明显更高。经分析主要是一些几何运算模块的实现复杂，降低了模型性能，随着 LuoJiaNET 对更多算子提供支持，该问题或许可以进一步被优化。

4.6 知识嵌入

4.6.1 任务简介

- (1) 专用框架中遥感解译知识图谱的构建方法，构建包含地理空间实体及关系的遥感解译知识图谱，建立遥感解译知识库；
- (2) 研究专用框架下经验知识引导与知识规则推理的方法，建立专用框架下面向遥感智能解译任务的知识耦合模型。

4.6.2 知识嵌入网络简介

知识嵌入的遥感影像解译网络结构如图 1 所示。利用该网络对遥感影像进行解译时，首

先基于遥感影像的地理坐标、成像时间等元数据信息，查询时空知识图谱，获取对应的坡度坡向数据，将其与遥感影像一同输入深度网络进行学习。输出预测结果后，一方面，将预测结果划分为一定数量的推理单元，调用时空知识图谱的解译常识，应用本体推理规则对预测结果进行修正。另一方面，再次查询时空知识图谱，获取对应的众源地理数据、定量遥感产品等先验知识，结合置信度规则将其用于预测结果的修正。经过上述时空知识图谱可信推理，即得到遥感影像最终的解译结果。

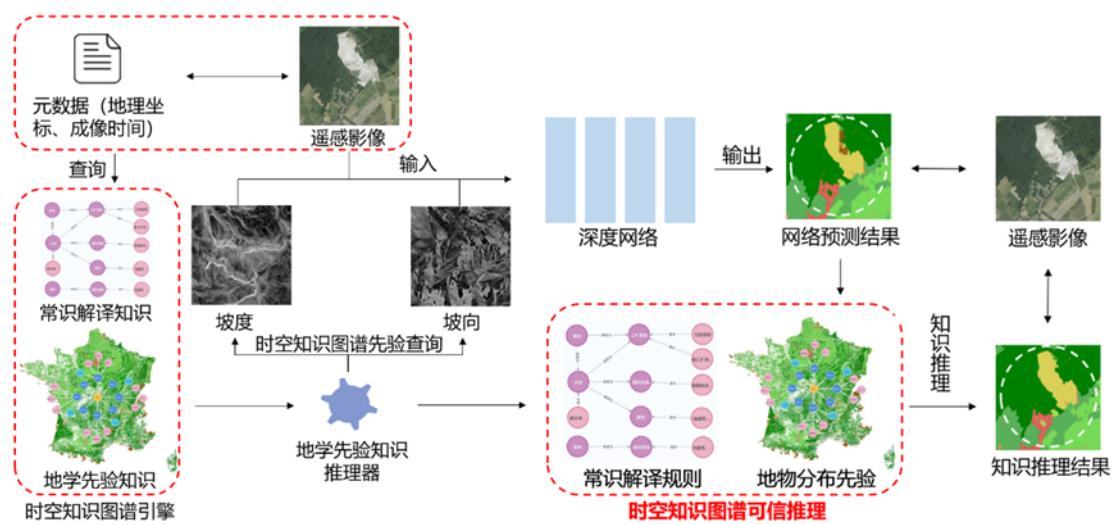


图 1 知识嵌入的遥感影像解译网络

4.6.3 测评结果

(1) 语义分割数据集：France Urban Atlas 数据集

France Urban Atlas 数据集是一个自主构建的大规模高分辨率大幅面遥感影像语义分割数据集。数据集共包含法国范围内 321 幅幅面为 10000×10000 、分辨率为 0.5 米的遥感影像。如图 2 所示，训练集 1、验证集、测试集由一定区域以内的 201 幅影像按 6:2:2 的比例不重叠地划分而成，训练集 2 由上述区域以外的 120 幅影像组成。语义标签采用开源的 Urban Atlas 数据，并将类别合并至 12 类。

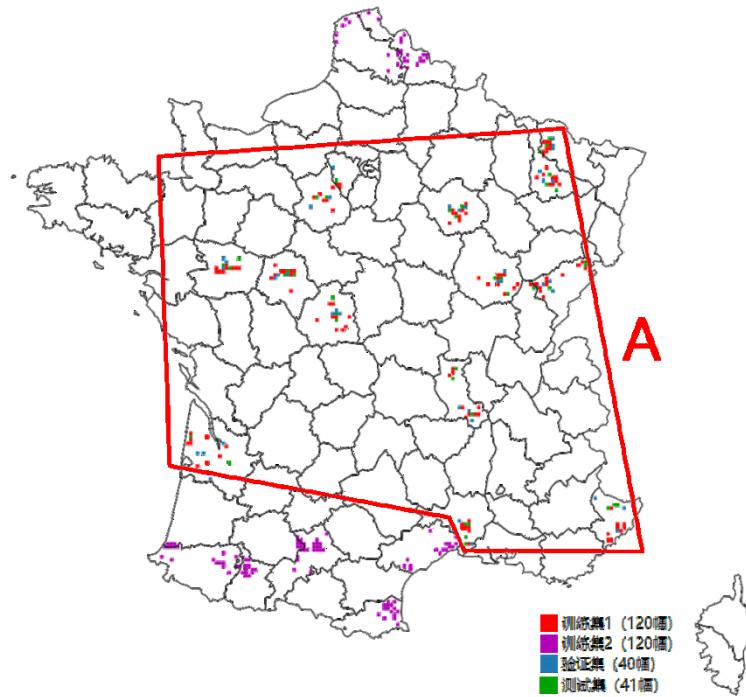


图 2 数据集在法国的地理分布

(2) 评价指标

在实验中，采用的定量指标如下：

a. 交并比 (IoU, Intersection over Union)

统计单个类别对应真实值和预测值的交集和并集之比，计算方式如下：

$$\text{IoU} = \frac{TP}{TP + FP + FN} \quad (1)$$

其中， TP 为预测正确的正类， FP 为预测错误的正类， FN 为预测错误的负类。

b. 均交并比 (mIoU, mean Intersection over Union)

统计所有类别交并比的平均值，计算方式如下：

$$\text{mIoU} = \frac{1}{k+1} \sum_{i=0}^k \text{IoU}_i \quad (2)$$

(3) 实验结果

影像对应的数字高程模型是一种自然先验知识，在将其应用于遥感影像解译时，通过对其进行计算，生成坡度、坡向，作为遥感影像的附加通道输入网络进行训练。在遥感影像解

译模型中嵌入自然先验知识的实验条件设置见表 1，实验的定量结果见表 2。由表 2 可知，

嵌入自然先验知识的实验(2)相比无知识嵌入的实验(1)在 mIoU 上提升约 3%。

表 1 时空先验知识嵌入实验条件设置

与工矿用地

规则 相邻的裸地

✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

1.2 修正为工矿

用地

将被建筑、

基础设施、

规则 工矿用地包

✓ ✓ ✓ ✓ ✓ ✓ ✓

1.3 围的水体修

正为相应类

别

将被水体包

规则

围的裸地修

✓ ✓ ✓ ✓ ✓ ✓

1.4 正为水体

将 OSM 数

场 据中属于工

规则

景 矿用地的像

✓ ✓ ✓ ✓ ✓

2.1

先 素修改为工

验 矿用地

知 将 OSM 数

规则

识 据中属于工

✓ ✓ ✓ ✓

2.2

业用地、商

	业用地、零		
	售用地的像		
	素修改为基		
	础设施		
将 OSM 数			
据中属于道			
规则			
路的像素修		✓	✓
2.3 改为基础设		✓	
施			
将 OSM 数			
据中属于交			
规则			
通设施的像		✓	✓
2.4 素修改为基			
础设施			
将 OSM 数			
据中属于水			
规则			
体的湿地、		✓	
2.5 园地修改为			
水体			

表 2 时空先验知识嵌入实验定量结果

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
建筑	0.6871	0.7057	0.7057	0.7057	0.7057	0.7057	0.7057	0.7067	0.7072	0.7073	0.7073
基础	0.5300	0.5524	0.5524	0.5524	0.5527	0.5527	0.5520	0.5795	0.5902	0.6030	0.6030
设施											
工矿	0.2354	0.2538	0.2538	0.2712	0.2713	0.2713	0.2854	0.2971	0.2956	0.2980	0.2980
用地											
城市	0.2444	0.3024	0.3197	0.3197	0.3197	0.3197	0.3198	0.3204	0.3205	0.3195	0.3195
绿地											
耕地	0.5058	0.5211	0.5211	0.5211	0.5211	0.5211	0.5207	0.5218	0.5215	0.5228	0.5228
园地	0.3704	0.3831	0.3823	0.3825	0.3825						
牧场	0.3991	0.4287	0.4287	0.4287	0.4287	0.4287	0.4285	0.4291	0.4291	0.4318	0.4318
森林	0.7716	0.7683	0.7702	0.7702	0.7702	0.7702	0.7702	0.7757	0.7758	0.7758	0.7758
灌木	0.5015	0.3136	0.3136	0.3136	0.3136	0.3136	0.3137	0.3138	0.3139	0.3139	0.3139
裸地	0.1930	0.2962	0.2962	0.3033	0.3033	0.3036	0.3036	0.3039	0.3039	0.3040	0.3040
湿地	0.5447	0.7486	0.7486	0.7486	0.7486	0.7486	0.7486	0.7486	0.7492	0.7492	0.7607
水体	0.7135	0.7955	0.7955	0.7955	0.7961	0.7965	0.7941	0.7948	0.7953	0.7955	0.7971
mIoU	0.4747	0.5058	0.5074	0.5094	0.5095	0.5096	0.5105	0.5145	0.5154	0.5169	0.5180

专家先验知识的知识图谱实现如图 3 所示。对网络预测结果进行连通域划分，形成推理单元；生成推理单元的面积向量、类别向量，计算出推理单元之间的邻接矩阵、邻接边长矩阵；根据以上结果，计算出推理单元满足的条件；将每个推理单元的类别及其满足的条件输入知识图谱进行知识推理，生成知识推理结果。

在遥感影像解译模型中嵌入专家先验知识的实验条件设置见表 1, 实验的定量结果见表 2。观察实验结果发现, 嵌入了 4 种专家先验知识的实验结果(6)相比实验(2)提高了约 0.4%。

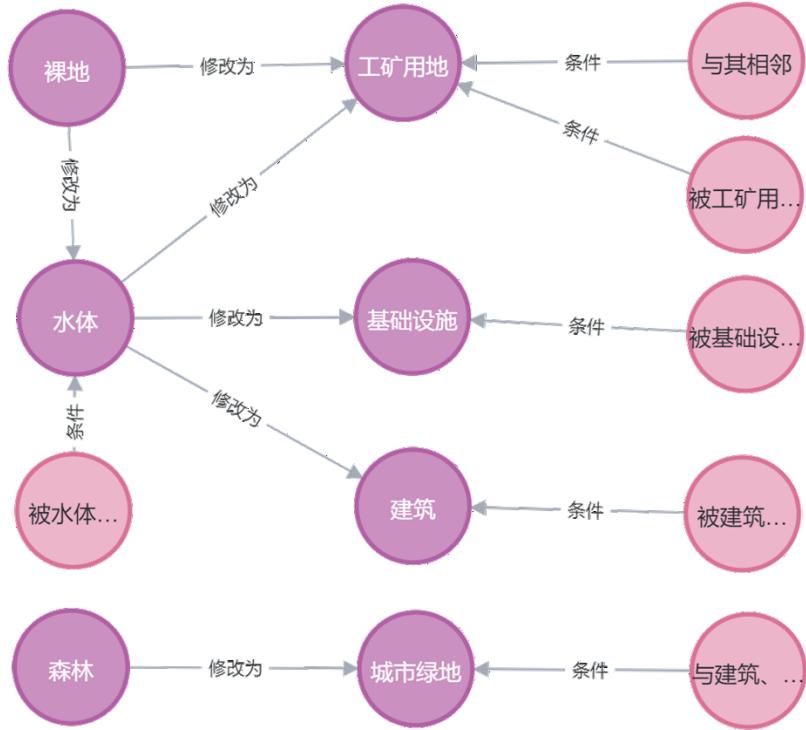


图 3 常识解译规则的知识图谱实现

在遥感影像解译模型中嵌入场景先验知识的实验条件设置见表 1, 实验的定量结果见表 2。由于 OSM 数据中工矿用地的分布较准确, 工业用地、商业用地、零售用地等类别部分弥补了深度网络难以把握基础设施整体性的问题, 规则 2.1、2.2 带来了一定的精度提升。观察实验结果发现, 应用 4 种专家先验知识及 5 种场景先验知识的实验(11)相比作为基准的实验(1)在 mIoU 上提高了 4.33%。

由统计结果总结出裸地的叶面积指数分布较为集中, 是一种明显的场景先验知识, 可以用于区分裸地和其他类别。使用叶面积指数取值作为限制条件, 将叶面积指数高于 9 的裸地像素的类别修改为网络预测概率第二高的类别, 应用这一场景先验知识进行修正。实验结果的 mIoU 有所上升, 达到了 0.5269, 主要是由于灌木、裸地两个类别的分类精度提升较为明显。

深度学习结果、深度学习+自然先验结果、深度学习+自然先验+专家先验结果、深度学习+自然先验+专家先验+场景先验结果的定性对比如图 4 所示。第一行示例中，工矿用地内的裸地被修正为工矿用地，城市内的森林被修正为城市绿地；第二行示例中，引入坡度坡向可以防止将坡度相对较大的类别识别为水体，大量在深度学习结果中被误分类为水体的单元被纠正，利用专家先验知识修正后城市内的森林被修正为城市绿地。第三行示例表明，场景先验知识的规则 2.3 实现了对断续道路的修正。

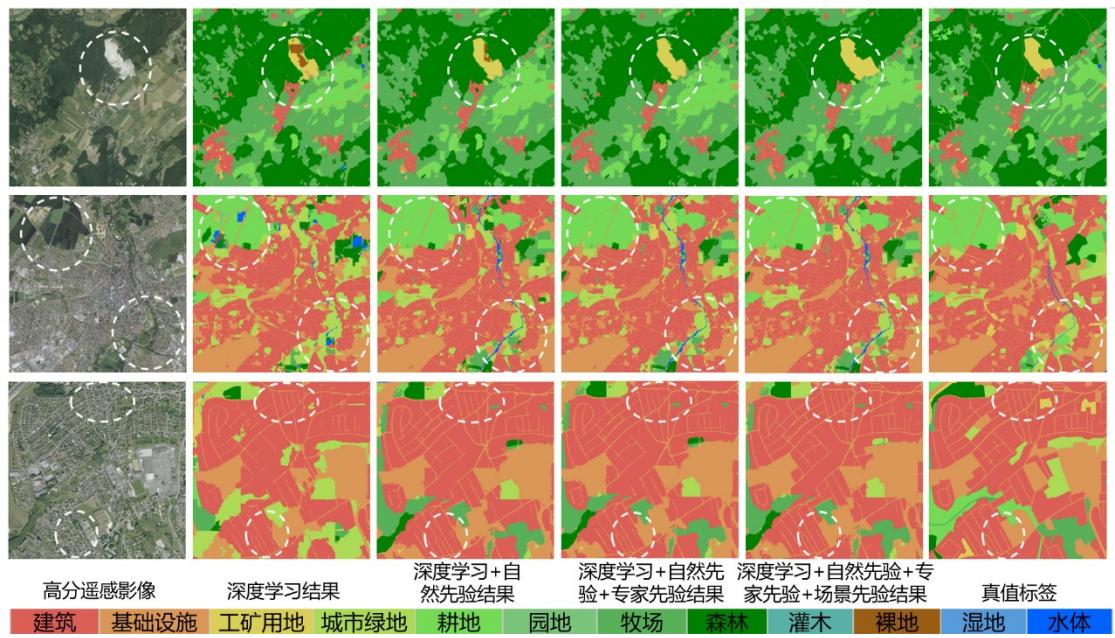


图 4 时空先验知识嵌入实验定性结果

4.6.4 总结

在所构建的法国区域大幅面高分遥感影像数据集上进行的对比实验结果表明，本方法通过嵌入自然先验、专家先验、场景先验等多种类型的经验知识，在 mIoU 这一定量指标上相比无知识嵌入的深度网络模型提升了 4%以上。

5. LuoJiaNET 的典型行业应用

5.1 上海数慧自然资源大脑解决方案

上海数慧构建了自然资源大脑，提供了“一张网--一张图--一大脑”的解决方案。其中“一大脑”的空间智能识别采用了LuoJiaNET“全栈”解决方案，硬件平台以华为昇腾910服务器为基础，软件平台采用LuoJiaNET构建了地物分类的识别系统。

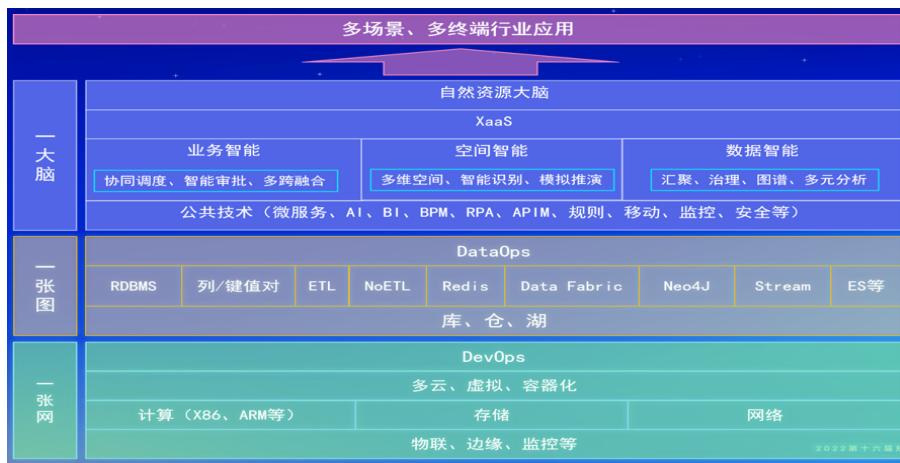


图 5.1 上海数慧“自然资源大脑”解决方案

5.2 武汉汉达瑞遥感影像智能化处理平台

汉达瑞科技有限公司以LuoJiaSET与LuoJiaNET作为基础底座，结合昇腾AI硬件，研发了遥感人工智能解译平台，包含样本制作与管理、模型训练与要素自动提取等功能。可支持全要素遥感解译，道路、房屋、水系、林地、云雪专题信息提取，同时提供全要素自动变化检测与建筑物变化检测功能，以云平台人机交互方式实现多要素的协同处理。



图 5.2 武汉汉达瑞科技有限公司遥感人工智能平台

5.3 航天宏图城市土地变化检测平台

航天宏图公司以国产 AI 芯片、遥感影像样本库 LuoJiaSET 和遥感深度学习框架 LuoJiaNET，构建了智能变化检测平台，可处理海量军用、民用、商用遥感数据与变化检测样本，同时采用国产 AI 芯片提升遥感影像处理速度。



图 5.3 航天宏图基于 LuoJiaNET 的土地利用变化监测平台

5.4 珈和科技地事通智慧农业 SaaS 服务平台

武汉珈和科技公司以 LuoJiaNET 作为遥感时空大数据 SaaS 平台基础软件，结合昇腾 AI 软硬件，研发了地事通 SaaS 服务平台，包含农田非粮化监测、农情病虫害监测、智慧农田服务、种植险监测等应用，解决行业在农业监测方面的痛点问题。



图 5.4 珉和科技地事通农业遥感 SaaS 平台

6. 进一步发展与展望

截止目前，境内外已有超 30 余家媒体专题报道 LuoJiaSET 与 LuoJiaNET 形成的初步成果，在公测的一个月时间内，已有超 3000 人次的下载量，多家大型公司将 LuoJiaSET 与 LuoJiaNET 作为遥感人工智能基础软件平台。



6.1 LuoJiaSET 发展及展望

- 在线标注工具的实用化完善
- 发挥联盟的作用，快速扩充样本集，2022 年达到 1000 万样本的规模
- 建立全球网格下的样本组织和管理，OPEN-GLOBAL_LANDCOVER_MAPPING?
- 开发可进化的样本库的开源工具集
 - 样本的**精化** - 自动的样本提纯、修正系统
 - 样本的**转换** - 自动的样本转换（实现传感器成像特性、地理环境、物候等特性的遥感图像转换与模拟）
 - 样本的**生成** - 自动的跨模态样本生成与仿真系统（例如：语言描述-遥感样本的生成系统）

6.2 LuoJiaNET 发展及展望

- 针对（大规模）**点云数据**的深度学习优化框架，算子补齐，例如：体素处理、稀疏卷积、超点分割
- **高维数据**（高光谱、多通道）处理进一步优化，目前的通道优选方法优点单薄
- **时序数据与雷达数据处理？**
- 大幅面影像的**图神经网络**处理优化+**知识嵌入方法**
- 建立 OGC 的模型标准。目前 NASA、ESA 已在参与讨论交换标准，但仍未立项
- 构建更大范围的智能遥感开源社区，促进遥感产业长远发展

7. 致谢

本白皮书制作过程中，在中国科学院院士龚健雅教授、武汉大学胡翔云教授指导下，由武汉大学张觅副研究员（LuoJiaNET 技术负责人）执笔统筹，得到了武汉大学姜良存副研究员（LuoJiaSET 技术负责人）、测绘遥感信息工程国家重点实验室孙开敏教授（场景分类）、眭海刚教授（变化检测）、王密教授、钟燕飞教授、潘俊教授的支持，武汉大学遥感信息工程学院黄昕教授、李家艺副教授（地物分类）、王心宇副研究员（高光谱地物分类）、季顺平教授（多视三维）、李彦胜副教授（知识嵌入）的支持，武汉大学计算机学院夏桂松教授、刘菊华副教授（目标检测）、袁梦霆教授（遥感 IR）的支持。同时得到 LuoJiaNET 框架团队博士研究生张展、杨炳楠、赵元昕、腾昊天，硕士研究生周桓、王斌、谭静懿，本科生刘青瑀、杨青林、余可、刘思琪、杨元钊，科研助理李大宇的支持；LuoJiaNET 应用模型验证得到博士生李王斌、赵恒伟、涂理林、张震、杜卓童、史玮玥、刘瑾、陈蔚，硕士生宣文杰、贺海斌、段坤仁、潘洋、李静涛、吴敏郎、钟振宇、周宁、高建，本科生王宇的支持；LuoJiaSET 样本库团队乐鹏教授、博士生许越、曹志鹏、刘帅旗等的支持。武汉理工大学熊盛武教授、硕士生周云飞、党伟冲、李锐在众智算子方面的支持；华为昇腾 AI 团队田昆阳、丁来平、张国稳、苏腾、杨振章、张晓达、梁成辉、江振光、刘军主、张丰伟、李咏、曹建、范翱翔等在双方共享知识产权、联合申报专利方面的支持。武汉人工智能计算中心王景俊、董长杰、邓凯、杨佳妮、段欣宇在华为云平台接口上的支持。

衷心感谢上述机构和人员在白皮书制作方面的鼎力支持！