摘　要

在上一个实验中，我们训练出了可以用于深度估计的模型，但是，这并不能作为一个完整的项目。为了能让项目落地，我们将项目部署到 Atlas 200DK 开发板上，通过 Atlas 200 DK 开发板来实现深度估计推理实验，通过读取本地图像数据作为输入，对图像中的环境进行深度估计，并且将检测结果图片保存到文件中。

我们成功地将模型部署到了 Atlas 200DK 上，并完成了推理。我们沿用了上一次实验所用的模型，在其基础上进行了一些调整以服从模型转换的一些条件。如为了减少冗余问题，我们减少了卷积核数量并对卷积核进行了分治处理；为了解决算子不兼容问题，我们自行创建了对应的算子。

在模型转换部分，我们起初只试图用 onnx 作为 pytorch 和 om 的中间媒介，但是一直会出现算子报错，尝试许多方法后，仍无法解决。于是我们巧妙地转换了思路，又加了一个中间媒介 Caffe，并搭建自行改写的算子来解决模型转换中遇到的算子问题。最后我们编写好 acl 文件实现了模型的推理。

本次实验，我们掌握了如何在 Atlas 200DK 上进行简单地应用开发，通过读取本地图像数据作为输入，在 Atlas 200DK 上对图像中的事物进行深度估计，并将分类结果展示出来。

关键字：　深度估计　U-Net　Onnx　Caffe　om　acl 推理

目　录

1 模型构建与调整

本次实验使用的模型是以 U-Net 为基础，构建深度学习网络以实现单目视觉深度估计，并迁移了 DenseNet 和 ResNet 的思想，使用扩张卷积神经网络与残差学习模块对模型进行增强。并依据在 Atlas 200DK 中部署的要求，对模型进行一定的调整。

1.1 模型介绍

1.1.1 U-Net 网络结构

U-Net 网络主要由收缩路径和扩展路径两部分组成。收缩路径用于捕获图像中的信息，扩展路径用于区域的精确定位。收缩路径每一层包括 2 个 3×3 大小的卷积层、一个 ReLU 激活函数和一个 2×2 大小的最大池化层，扩展路径的每一层包含一步 2×2 大小的上采样、两个 3×3 大小的卷积层和一个 ReLU 激活函数。最后利用 1×1 的卷积核进行卷积运算，将维度映射为单通道的图像进行输出。[1] 其模型结构如下图 1.1.1所示：

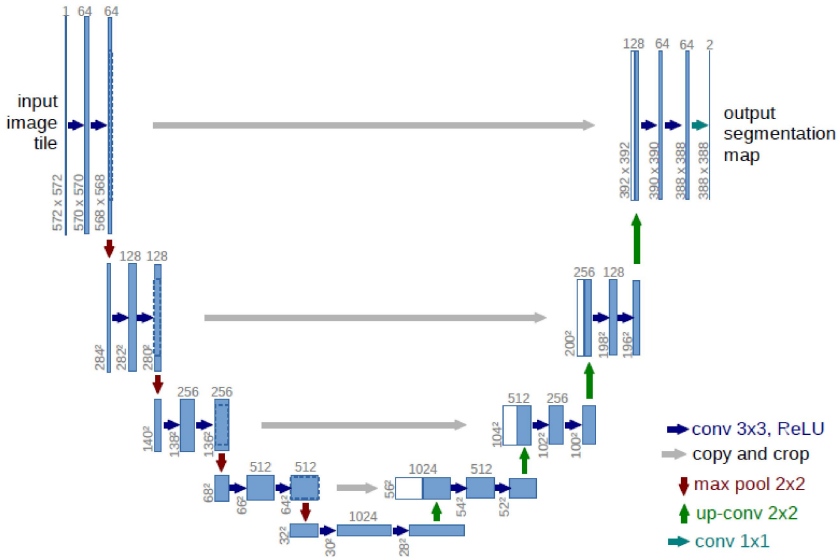


图 1: 模型结构

针对单目视觉深度估计部署任务的具体要求，对 U-Net 进行如下修改：

1) 原 U-net 网络使用了“valid”卷积方式，每次卷积后都会使得图像尺寸减小，为保持输入和输出图像的分辨率大小不变，本报告中采用“same”卷积方式。

2) 在输出层增加一个 Sigmoid 激活函数。Sigmoid 函数的作用是对每个神经元的输出进行标准化，将绝对值较大的输出值缓慢推向极值，可以使网络的预测结果更加稳定。另外,Sigmoid 函数采用最大熵模型，受噪声数据影响较小，因此可以增强网络的抗噪性能。

3) 为了解决部署时模型产生冗余的问题，我们选择缩小网络中的部分规模，方法是减小每层卷积的卷积核数量，并对卷积核进行分治处理。

1.1.2 扩张卷积神经网络

我们采用扩张卷积神经网络提取图像的深度信息来对 U-Net 进行增强。在特征提取的过程中，下采样操作常丢弃具有小尺寸的局部特征。而扩张卷积正是下采样层的替代方案，不仅增加了感受野，保留更多的局部信息，还保持了特征图的空间维度，实现了局部与全局的双重优化。[2]

扩张卷积在非零滤波器中插入零点对特征图进行采样，加快了感受野的动态速率，保持了特征图的空间维度而不增加计算复杂度。[3] 在卷积核大小为 3×3 的情况下，标准大小的卷积核的感受野大小为 3×3；扩张率为 n 的扩张卷积的感受野扩大为 (2n+1) × (2n+1)，在扩张卷积的过程中，卷积核大小保持不变，如图所示。

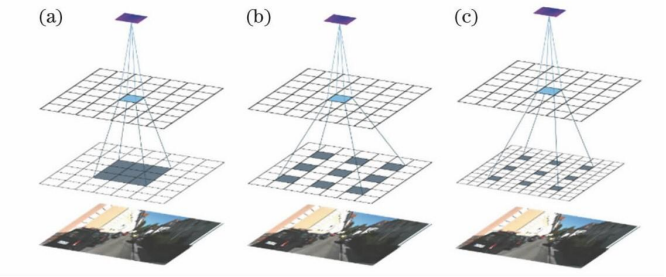


图 2: 扩展卷积

在实验中我们发现，带有扩张卷积神经网络的模块在进行模型转换和部署时，性能损失甚至相较于基础的 U-Net 网络架构更小。

1.1.3 残差学习模块

U-Net 模型在嵌入扩张卷积网络后，网络的参数量和网络层数增加导致了网络模型计算速度减慢、梯度爆炸和网络退化等问题。因此在嵌入扩张卷积机制的模型中引入残差结构，抑制梯度爆炸和网络退化的问题。模型的上下采样部分采用 BasicBlock 模块进一步优化卷积层，其结构如图所示。

输入特征首先经过一个 $3*3*C$ 的卷积层，通过激活函数后再经过一个 $3*3*C$ 的卷积层，然后与输入特征相加后再进入到下一层。[4] 模型的下采样中，输入特征通过 BasicBlock 模块后进入到 BottleNeck 模块中，其结构如图所示，BottleNeck 模块可以减少计算和参数量。

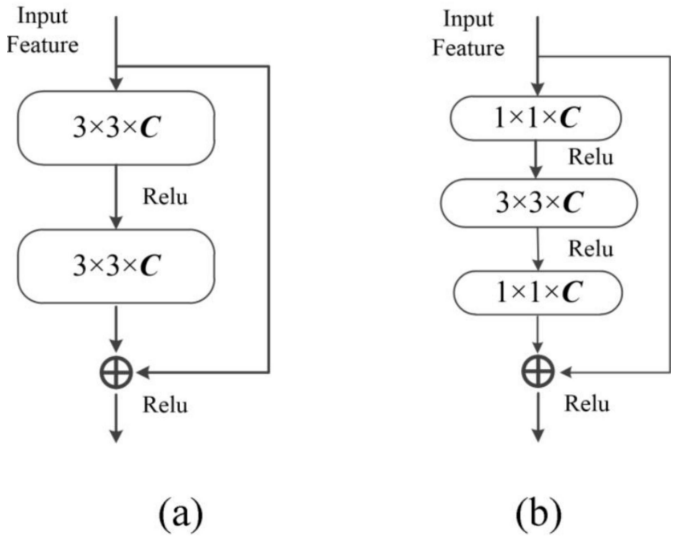


图 3: BasicBlock 和 BottleNeck 模块

实验表明，使用残差学习模块可以一定程度上改善模型转换后对部分图像性能大幅降低的问题，但是会产生更多的算子不兼容，需要自行创建对应算子。

经过调整后，我们的模型框架如下图所示：

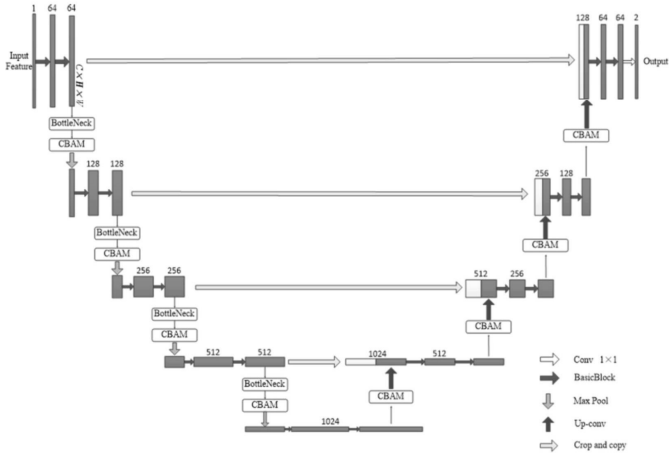


图 4: 调整后的模型框架

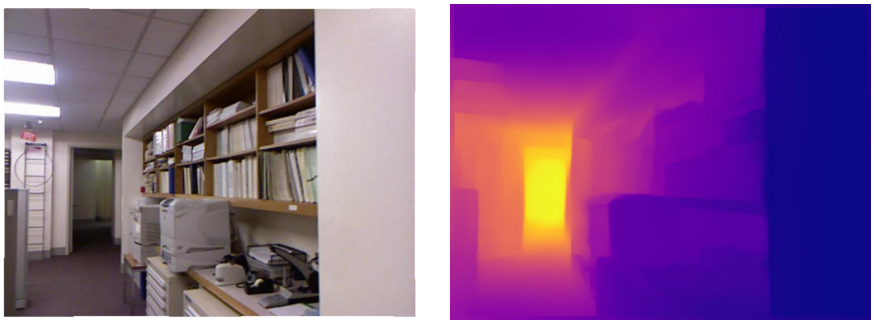
1.2 模型训练

1.2.1 训练配置

本模型所使用的框架是 Pytorch，使用了华为云平台中 ModelArts 中规格为 GPU：1*P100(16GB) | CPU：8 核 64GB 的 Notebook 开发环境进行模型的训练。模型训练所使用的数据集是常用于深度估计等 CV 任务的 KITTI 和 NYU-Depth V2 中的训练集。模型训练总共经过了 30 个 epoch，batch-size 为 10。为了更大程度上减少模型在转换后和部署到开发板上时的损失，在训练中我们使用了三种损失对其进行优化：1. 结构相似性损失 (SSIM).2.L1-Loss.3. 深度平滑损失 (Depth smoothness loss)。训练中使用的优化器是 Adam，初始学习率为 0.0001，并在 75% 的迭代结束后减少至一半。所用到的残差学习模块维度与 ResNet-50 中的一致。

1.2.2 训练结果

我们将训练出来的模型加载入 demo 中。向 demo 输入原始图像，即可得到深度估计的结果，下面就是一个深度估计的样例：



原图　　深度图

图 5: 深度估计效果

我们可以发现，尽管为了满足部署的一些条件，我们对模型进行了一些调整，但调整后的模型依旧可以达到不错的深度估计效果。

2 模型部署

进行部署需要的环境和设备为为 VirtualBox 虚拟机下的 Ubuntu 18.04 Desktop（X86 架构），操作系统为 Ubuntu X86 架构的服务器或者 PC 机，用于连接开发板；Atlas200DK 的合设环境，即 Atlas 200 DK 即作为开发环境又作为运行环境。Atlas 200DK 合设环境搭建可以通过 dd 镜像的方式来将环境烧录到 SD 卡中。

2.1 模型转换

我们训练出来的是 pytorch 的模型，而 pytorch 是无法在 Atlas 200DK 上进行推理的。为了满足模型在开发板上的部署要求,我们选择的模型转换路线为:Pytorch→onnx→caffe→om。此前我们尝试过 Pytorch→onnx→om 的路线，但是总是有算子不识别的报错。我们也根据网上的教程调整过版本，但是最终都无法解决这个问题。于是我们更改了转换路线，不直接将 Pytorch 转为 caffe，或直接将 onnx 转为 om，都是因为在这两次转换中存在无法改写完善的算子，导致模型转换报错。引入中间转换步骤后，[5] 可以使用搭建好的模型转换工具或是自行改写算子来对这部分的算子的转换进行一些过渡，从而避免报错的出现。此外，从 caffe 转换为 om 可以更好地根据华为官方的转换教程进行。

2.1.1 Pytorch 转为 Onnx

Pytorch 转为 onnx 的过程中，我们使用了 torch.onnx.export 函数:

```
state_dict = torch.load(args.model_dir)
new_state_dict = OrderedDict()
for k, v in state_dict.items():
        name = k[7:] # remove "module."
        new_state_dict[name] = v
Model = LDRN(args)
if args.cuda and torch.cuda.is_available():
    Model = Model.cuda()
assert (args.model_dir != ''), "Expected pretrained model directory"
Model.load_state_dict(new_state_dict)
Model.eval()
input_names = ["raw_picture"]
output_names = ["depth_picture"]
x = torch.randn((16, 3, 352, 704))
torch.onnx.export(Model,x,'./model1.onnx',input_names=input_names,
output_names=output_names, dynamic_axes={'input_0':[], 'output_0':[]},
    opset_version=11)
```

为了方便 onnx 进一步转为 caffe 模型，且合并零散的算子对模型结构进一步优化，我们选择了 *onnx_simplifier* 工具。在使用此工具的过程中，出现了模型的进阶图优化 (如合并扩张卷积层) 无法实现的情况。为此，我们将模型架构与问题需求致信该工具的开发者 (*Github* : *Daquexian*)，并在其指导下附加安装了 *onnx_optimizer* 并调用、改写其中的 pass 实现了针对本模型的优化。

2.1.2 Onnx 转为 Caffe

onnx 转为 caffe 的过程中，我们使用了 onnx2caffe 工具:

```
onnx_to_caffe.trans_net(detect_model, input, name)
onnx_to_caffe.save_prototxt('{}.prototxt'.format(name)) # 生成prototxt
onnx_to_caffe.save_caffemodel('{}.caffemodel'.format(name))# 生成caffemodel
```

在使用该工具对模型进行转换时，由于本模型中含有较多扩张卷积、残差连接和用于深度估计的 depthwise 部分，会产生层损失甚至转换失败的部分。针对模型特点，我们在 caffe 使用 depthwise 层的 prototxt 中添加 engine: CAFFE 来避免 GPU 神经网络库的错误。此外，在所有 group > 1 的扩张卷积层中均添加该参数来避免层损失和连接失效。

2.1.3 Caffe 转为 om

caffe 转为 om 的过程中，我们需要对一部分不被支持的算子根据模型实际要求进行改写，以避免 ATC 工具转换模型失败。此处以问题最多的 resize 算子为例陈述算子改写过程：1. 通过官方文档《Caffe 算子边界》确定” .om” 模型支持的 Caffe 算子边界限制，如果模型中的 resize 参数超出限制，则将其缩小。2. 通过 API 获取 caffe 模型中对应中间层的算子参数与输出，如” .om” 制定了算子默认值，则修改。3. 根据中间层的输出及算子的具体作用，使用” .om” 所支持的算子将 resize 算子表示为其他形式 (如下表示代码)，并替换至原 caffe 模型中。4. 根据使用的算子选择对应的 ATC 版本与指令。

```
for i in range(len(model.graph.node)):
n = model.graph.node[i]
if n.op_type == "Resize":
model.graph.initializer.append(
caffe.helper.make_tensor('scales{}'.format(i), caffe.TensorProto.FLOAT,
    [4], [1, 1, 2, 2])
)
newnode = caffe.helper.make_node(
'Resize',
name=n.name,
inputs=ReplaceScales(n.input, 'scales{}'.format(i)),
outputs=n.output,
coordinate_transformation_mode='asymmetric',
cubic_coeff_a=-0.75,
mode='nearest',
nearest_mode='floor'
)
model.graph.node.remove(model.graph.node[i])
model.graph.node.insert(i, newnode)
```

接下来我们在通过虚拟机连接开发板，通过 atc 指令实现模型的转换，atc 指令如下:

```
atc --model=DepthNet.prototxt --weight=DepthNet.caffemodel --framework=0
--output=DepthNet_yuv --soc_version=Ascend310 --insert_op_conf=DepthNet.cfg
```

对修改后的 caffe 模型进行转换后，为了确定转换是否成功，我们可以在输入和对应预处理方法均相同的情况下利用官方 dump 工具对原 Pytorch 模型、caffe 模型和 om 模型的输出特征进行余弦距离的比对，以确定模型精度是否能用于部署。

余弦距离	Pytorch	Caffe	om
Pytorch	1	0.984	0.957
Caffe	0.984	1	0.972
om	0.957	0.972	1

通过评估我们可以发现，模型转换的精度损失在可接受范围内，此时我们可以认为该 om 模型的性能能够较好地还原 Pytorch 模型，因此可以用于开发板的部署。

2.2 模型推理

将模型转换好之后，现在需要编写 ACL 推理文件对转换为 om 格式的模型进行推理，并将推理结果存入 Atalas 200DK 开发板。我们使用官方提供的在 AscendCL 的基础上使用 CPython 封装得到的 Python API 库，pyACL，对本模型进行推理，推理的流程如下所示。每步中使用的 pyACL 中 API 为：

- 资源初始化和 device 管理（用于初始化系统内部资源，固定的调用流程，运行管理资源申请的功能封装在了函数 Depth.__init__resource 中）：

```
1       acl.init/acl.rt.set_device
```

- 模型初始化（加载模型文件，构建模型输出内存，加载本地 om 模型文件到内存中，并创建并获取模型的描述信息，此函数功能封装在 Model.init__resource；根据模型的描述信息，获取模型的每路输出数据在设备上所需的空间大小，ACL 库内置数据类型说明：aclmdlDataset 主要用于描述模型推理时的输入数据或输出数据，模型可能存在多个输入、多个输出，每个输入或输出的内存地址、内存大小用 aclDataBuffer 类型的数据来描述）：

```
1       acl.mdl.load_from_file/acl.mdl.create_desc/acl.mdl.get_desc
```

- Host 数据传输至 Device（读取本地图像数据并进行预处理，将读取到的图像数据拷贝至设备侧申请的内存空间中，为接下来构建模型输入数据做好准备）：

```
1       acl.rt.memcpy
```

- 模型推理（根据已经加载到内存中，要进行推理的模型 ID、构建好的模型推理输入数据，调用 ACL 库中模型推理接口进行模型推理。模型推理功能函数封装在了 Model.execute 中）：

```
1       acl.mdl.execute
```

- device 和 host 上的内存管理(模型推理结果解析的功能函数封装在 Depth.post__process 中)：

```
acl.rt.malloc/acl.rt.malloc_host
```

- 数据后处理：

```
acl.mdl.get_dataset_buffer/acl.mdl.get_dataset_num_buffers
```

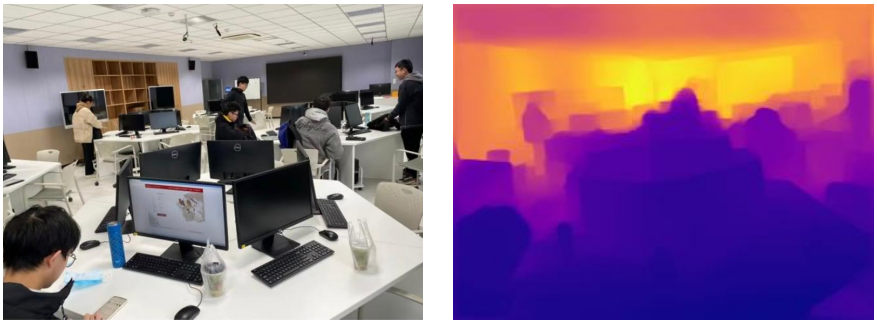
- Device 数据传输至 Host：

```
acl.rt.memcpy
```

- 释放资源 (推理接收后要卸载模型，并释放与模型推理相关的资源，此功能函数封装在 Model.___del___ 中；释放存放图像数据的内存空间的功能函数封装在 AclImage.destroy 中；运行管理资源释放以及 ACL 去初始化的功能函数封装在 Depth.___del___ 中)：

```
acl.rt.free/acl.rt.free_host/acl.rt.reset_device/acl.finalize
```

以上就是部署的全部流程了，部署到开发板后也成功对图像进行了深度估计，图??为其中的一组示例。我们将操作的过程制作成了演示视频，并且已在课上展示。



原图　　深度图

图 6: 深度估计示例

3 总结

我们成功地将模型部署到了 Atlas 200DK 上，并完成了推理。我们沿用了上一次实验所用的模型，在其基础上进行了一些调整以服从模型转换的一些条件。如为了减少冗余问题，我们减少了卷积核数量并对卷积核进行了分治处理；为了解决算子不兼容问题，我们自行创建了对应的算子。

在模型转换部分，我们起初只试图用 onnx 作为 pytorch 和 om 的中间媒介，但是一直会出现算子报错，尝试许多方法后，仍无法解决。于是我们巧妙地转换了思路，又加了一个中间媒介 Caffe，并搭建自行改写的算子来解决模型转换中遇到的算子问题。最后我们编写好 acl 文件实现了模型的推理。

参考文献

[1] 肖东, 韩晨, and 范文强, “基于 u-net 和 resnet 的图像缺陷检测,” 计算机与数字工程, vol. 50, no. 8, pp. 1791–1794, 2022.

[2] 李承珊 *et al.*, “基于深度卷积神经网络的图像语义分割研究,” Ph.D. dissertation, 北京: 中国科学院光电技术研究所, 2019.

[3] 李阳, 陈秀万, 王媛, 刘茂林 *et al.*, “基于深度学习的单目图像深度估计的研究进展,” 激光与光电子学进展, vol. 56, no. 19, p. 190001, 2019.

[4] Z. Li, M. Jia, X. Yang, and M. Xu, “Blood vessel segmentation of retinal image based on dense-u-net network,” *Micromachines*, vol. 12, no. 12, p. 1478, 2021.

[5] M. YukiNagato Siyuan Yang, “Convert pytorch to caffe by onnx,” https://github.com/MTLab/onnx2caffe.