Fibocom 广和通
完 美 无 线 体 验

图像语义分割（deeplabv3）案例

----基于 SC171 开发套件 V3

文档版本：V1.0
更新时间：2025 年 3 月 27 日

适用型号

序列	文档版本	适用型号	更新说明
1	V1.0	SC171 开发套件第三代	NA

目录

1 引言

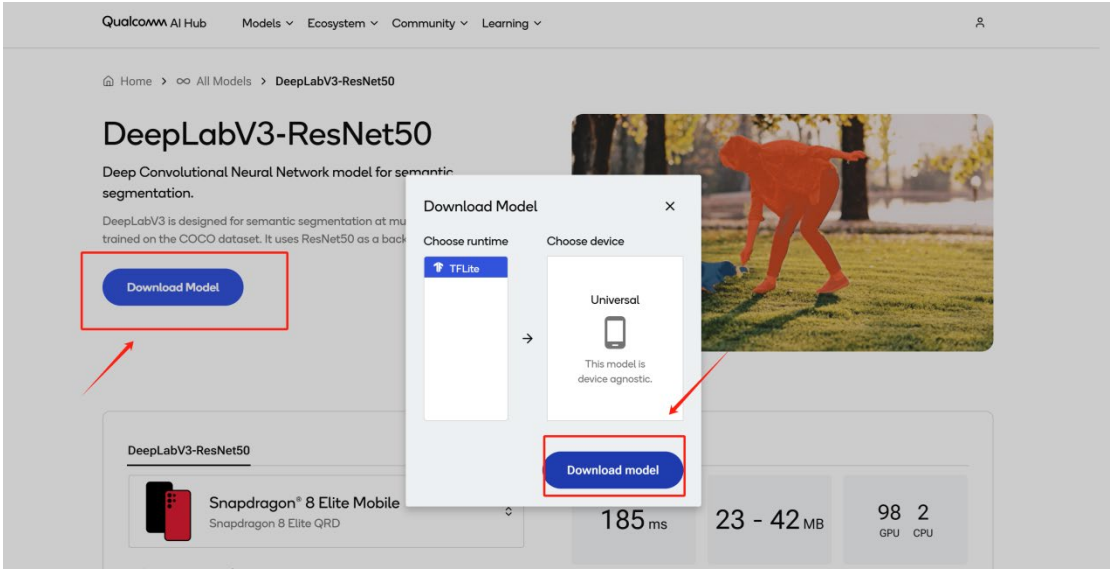
本指南以图像语义分割（DeepLabV3 模型）为例，详细演示如何在 SC171 开发套件 V3 上通过 Fibo AI Stack 实现高效的深度学习模型推理。

该案例使用的模型包含对 21 种类别的识别（详情见 label.txt）。若实践过程中遇到问题，请参考第 4 节“Q&A”或联系 Fibocom 技术支持团队获取帮助。

2 模型下载

点击进入模型链接 DeepLabV3-ResNet50 - Qualcomm AI Hub

选择 Download Model，可以选择三个模型类型下载，这里推荐选择 ONNX 格式



3 模型转化

详情请见《Fibo AI Stack 模型转化指南》，最后将转化好的 dlc 格式模型导入开发板中。

4 详细步骤

本案例所用到的工程源码见链接:
https://pan.baidu.com/s/1exlTnwZl0qLIbAEzTqfXYQ?pwd=h8zr

深圳市广和通无线股份有限公司 | 总机: +86 755-26733555 邮箱: market@fibocom.com | 传真: +86 755-26520841 官网: www.fibocom.com.cn

4.1 准备

1. 进入 Ubuntu 的 UI 界面，打开开发板终端，调用脚本配置环境参数：

```
cd /home/fibo/Fibo_AI_Stack/fiboaisdk_ubuntu_aarch64
. ./scripts/env_qualcomm.sh 68
```

2. 导入所需的库，在终端输入以下命令：

```
pip3 install numpy pillow
```

3. 需要的参数：

- dlc_path：模型文件路径（.dlc 文件）
- img_path：输入图片路径
- 模型输入、输出张量名称与类型

4. 将本案例中的 test2.py、utils2.py 与 api_infer.py 放置在同一路径下

4.2 Python sample 代码

1. 主程序内容：

```python
# 导入必要的库和模块
from api_infer import *  # 导入SNPE推理框架的Python接口
from utils2 import (preprocess_image, postprocess_21class)  # 导入自定义工具函数：图像预处理和21类分割后处理
from PIL import Image    # PIL图像处理库

#配置区域
dlc_path = "/home/fibo/Fibo_AI_Stack/SS/deeplabv3_resnet50.dlc"# 定义模型文件路径
img_path = "/home/fibo/Fibo_AI_Stack/SS/5.jpg"# 定义输入图片路径
output_path = "/home/fibo/Fibo_AI_Stack/SS/deep_output5.jpg"# 定义输出图片保存路径

#初始化SNPE推理上下文
snpe_ort = SnpeContext(dlc_path, [], Runtime.GPU, PerfProfile.BALANCED, LogLevel.INFO)
assert snpe_ort.Initialize() == 0

#图像前处理
img = Image.open(img_path)
img.show()
processed_data, model_input, meta_data = preprocess_image(img_path)
meta_data["original_image_path"] = img_path  # 在元数据中记录原始图像路径，供后续结果叠加使用
# 执行推理，获取输出字典
output_names = []
outputs = snpe_ort.Execute(output_names, model_input)

#结果后处理
result = postprocess_21class(
    model_output=outputs,
    meta_data=meta_data,
    alpha=0.7,              # 前景70%不透明度
    edge_thickness=1,        # 边缘粗细
    background_class=0       # 背景类别ID
)

# 保存并显示结果
result.save(output_path)
result.show()

#释放资源
assert snpe_ort.Release() == 0
```

模型前处理

模型推理

模型后处理

2. utils1.py 文件内容：

调用本地文件和库以及前处理函数内容：

图像预处理函数：

参数：

image_path:输入图像路径

target_size:目标尺寸（高度，宽度）

返回：

batched:处理后的图像数组 (1,H,W,3)float32

model_input:模型输入字典 {"image":batched}

meta_data:包含原始尺寸、填充量等元信息

处理流程：

1. 加载图像并确保 RGB 格式
2. 按比例缩放（保持长宽比）
3. 填充至目标尺寸
4. 归一化像素值到[0,1]

```python
import cv2
import numpy as np
from PIL import Image
from typing import Tuple, Dict, List
import matplotlib.pyplot as plt

def preprocess_image(  前处理函数
    image_path: str,
    target_size: Tuple[int, int] = (520, 520)
) -> Tuple[np.ndarray, Dict[str, np.ndarray], dict]:
    # 1. 加载图像并转换为RGB格式（确保3通道）
    with Image.open(image_path) as img:
        orig_w, orig_h = img.size  # 获取原始宽高
        img_np = np.array(img.convert("RGB"))  # 转换为RGB格式的numpy数组
    # 2. 计算缩放比例（保持长宽比）
    target_h, target_w = target_size
    scale = min(target_w / orig_w, target_h / orig_h)  # 计算最小缩放比例
    new_w, new_h = int(orig_w * scale), int(orig_h * scale)  # 计算新尺寸
    # 3. 使用双线性插值缩放图像，先归一化到0-1范围，再进行缩放
    resized = cv2.resize(img_np.astype(np.float32) / 255.0,  (new_w, new_h),interpolation=cv2.INTER_LINEAR)
    # 4. 计算填充量（居中填充）
    pad_w = (target_w - new_w) // 2  # 宽度方向填充量
    pad_h = (target_h - new_h) // 2  # 高度方向填充量
    # 5. 应用填充（上下左右对称填充）
    padded = np.pad(
        resized,
        [(pad_h, target_h - new_h - pad_h),  # 高度方向（上填充，下填充）
         (pad_w, target_w - new_w - pad_w),  # 宽度方向（左填充，右填充）
         (0, 0)],  # 通道方向不填充
        mode='constant',  # 常数填充
        constant_values=0  # 填充值为0（黑色）
    )
    # 6. 添加批次维度（模型通常需要batch维度）
    batched = np.expand_dims(padded, axis=0).astype(np.float32)
    # 7. 收集预处理元数据（用于后处理时恢复）
    meta_data = {
        "original_size": (orig_w, orig_h),  # 原始图像尺寸
        "scale": scale,  # 缩放比例
        "padding": (pad_w, pad_h),  # 应用的填充量
        "original_image_path": image_path  # 原始图像路径
    }
    return batched, {"image": batched}, meta_data
```

后处理函数内容：

结果后处理函数：

参数：

深圳市广和通无线股份有限公司 | 总机:+86 755-26733555 传真:+86 755-26520841
邮箱:market@fibocom.com 官网:www.fibocom.com.cn

model_output:模型输出字典（需包含'mask'键）
meta_data:预处理保存的元数据
alpha:前景透明度（0-1）
edge_thickness:边缘线粗细
background_class:背景类别 ID

返回:PIL.Image:处理后的 RGB 图像

处理流程:

1. 将模型输出转换为分割掩码
2. 去除填充并还原原始尺寸
3. 为不同类别应用不同颜色
4. 生成带透明度的叠加结果

后处理函数

```python
def postprocess_21class(model_output: Dict[str, List],meta_data: dict,alpha: float = 0.6,edge_thickness: int = 1,background_class: int = 0) -> Image.Image:
    # 1. 转换模型输出为520x520数组
    mask = np.array(model_output["mask"]).reshape(520, 520)
    # 2. 去除预处理时添加的填充区域
    pad_w, pad_h = meta_data["padding"]
    unpadded = mask[pad_h:520-pad_h, pad_w:520-pad_w]
    # 3. 还原到原始尺寸（使用最近邻插值保持类别ID）
    orig_w, orig_h = meta_data["original_size"]
    resized_mask = cv2.resize(unpadded.astype(np.uint8),(orig_w, orig_h),interpolation=cv2.INTER_NEAREST )
    # 4. 加载原始图像（确保RGB格式）
    original_img = np.array(Image.open(meta_data["original_image_path"]).convert("RGB"))
    # 5. 创建透明叠加层（RGBA格式）
    overlay = Image.new("RGBA", (orig_w, orig_h), (0, 0, 0, 0))
    overlay_np = np.array(overlay)
    # 6. 使用matplotlib的tab20颜色映射（21个类别）
    colors = (plt.get_cmap("tab20", 21).colors * 255).astype(np.uint8)[:, :3]
    # 7. 为每个前景类别上色（跳过背景类）
    for class_id in range(21):
        if class_id == background_class:
            continue
        # 7.1 生成当前类别的二值掩码
        class_mask = (resized_mask == class_id)
        # 7.2 应用颜色和透明度
        overlay_np[class_mask, :3] = colors[class_id]  # RGB通道
        overlay_np[class_mask, 3] = int(alpha * 255)   # Alpha通道（透明度）
        # 7.3 边缘强化处理（可选）
        if edge_thickness > 0:
            # 查找轮廓
            contours, _ = cv2.findContours(class_mask.astype(np.uint8),cv2.RETR_EXTERNAL,  cv2.CHAIN_APPROX_SIMPLE )
            # 绘制白色边缘
            cv2.drawContours(
                overlay_np,
                contours,
                -1,  # 绘制所有轮廓
                color=(255, 255, 255, 255),  # 白色边缘（完全不透明）
                thickness=edge_thickness  # 边缘粗细
            )
    # 8. 图像合成：将分割结果叠加到原图上
    result = Image.alpha_composite(Image.fromarray(original_img).convert("RGBA"), Image.fromarray(overlay_np) )
    # 返回RGB格式结果（兼容性更好）
    return result.convert("RGB")
```

4.3 模型的运行推理

调用 python 主程序执行文件进行推理，在开发板终端输入以下命令：
python3 <Python 文件地址>
如：
(python3 /home/fibo/Fibo_AI_Stack/SS/test2.py)

输入内容，如图所示

深圳市广和通无线股份有限公司 | 总机: +86 755-26733555 传真: +86 755-26520841
邮箱: market@fibocom.com 官网: www.fibocom.com.cn

深圳市广和通无线股份有限公司
总机: +86 755-26733555
邮箱: market@fibocom.com
传真: +86 755-26520841
官网: www.fibocom.com.cn

输出结果，执行成功，结果如图所示：

深圳市广和通无线股份有限公司
总机: +86 755-26733555
邮箱: market@fibocom.com
传真: +86 755-26520841
官网: www.fibocom.com.cn

5 Q&A

5.1 无法找到文件

可能原因：
1. 模型、标签等文件路径配置错误。
2. 模型文件未正确下载或解压。
3. 无权限访问文件。

解决办法：
1. 修改模型的存放路径。
2. 确保模型文件已正确下载并解压到指定路径。
3. 确保运行的 Python 文件处于正确路径下。
4. 检查文件权限，确保当前用户有权限访问模型文件。

5.2 环境配置失败

可能原因：
1. 环境变量未正确设置。
2. 缺少必要的 Python 包。

解决办法：
1. 确保已正确执行环境配置脚本：
```
cd /home/fibo/Fibo_AI_Stack/fiboaisdk_ubuntu_aarch64
. ./scripts/env_qualcomm.sh 68
```
2.检查环境变量是否已正确设置，确保路径中没有错误。
3.安装必要的 Python 包。

5.3 推理结果不准确

可能原因：
1. 输入图像不符合模型要求。
2. 模型未正确加载或配置。

解决办法：
1. 确保输入图像格式正确使用，可以输出输入数据来检查格式是否符合模型要求
2. 检查模型，输入输出 Tensor 名称等信息正确。
3. 检查输入图像的预处理步骤，确保图像数据格式符合模型要求。

深圳市广和通无线股份有限公司 | 总机: +86 755-26733555 邮箱: market@fibocom.com 传真: +86 755-26520841 官网: www.fibocom.com.cn