## 项目概述

本项目旨在开发一个基于深度学习的热红外图像定位算法，用于无人机在GPS信号丢失或受到干扰条件下的定位。该算法能够在全天候及视觉受限环境中实现可靠的位置识别。

## 数据采集及预处理

### 2.1 数据采集

数据采集可以根据网络模型策略的不同而有所区别。具体来说，可以将数据采集分为两种形式：一种是为了重新训练或微调模型，另一种是直接使用预训练模型，以下是对两种情况下数据采集要求的详细介绍。

#### 2.1.1 数据采集用于重新训练或微调模型

如果采用重新训练或者微调的方式，需要对数据进行整个流程的处理，为提高数据集的质量，应满足以下条件：

1. **图像重叠度**：前后重叠应达到70%-80%，在飞行路径上，前一张图像与后一张图像之间应有70%-80%的内容重复；侧向重叠应为60%，相邻飞行路径之间，图像应有60%的重叠区域，最大化地面覆盖效率。飞行路线如图1所示。
2. **飞行速度**：速度应与无人机的相机快门速度匹配，以防止图像因飞行速度过快而出现模糊。
3. **覆盖内容**：飞行路线应尽量覆盖不同对象、光照条件、背景，以使模型能够适应不同环境下的特征提取及检索。
4. **飞行高度、时间、姿态**：如果需要验证在不同飞行时间和飞行姿态时算法的性能指标，应该计划在不同飞行高度（例如300m,600m,900m），不同飞行时间（上午、下午和晚上），以及飞行姿态（yaw, pitch, roll）进行飞行，以验证在不同空间分辨率，光照条件以及飞行姿态下的算法性能指标。

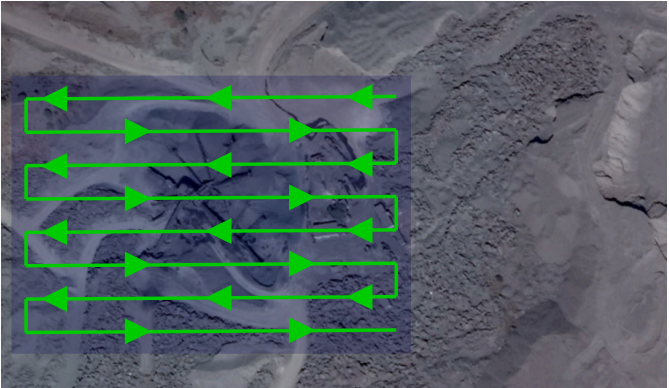


图1 飞行路线

#### 2.1.2 数据采集用于直接使用预训练模型

如果数据采集的目的是直接使用预训练模型，那么也不需要进行2.2中的预处理。这种情况下的数据采集条件应包括：

1. **飞行速度**：速度应与无人机的相机快门速度匹配，以防止图像因飞行速度过快而出现模糊。
2. **飞行高度**：飞行高度应尽量在300米以上，采集高度过低会导致图像覆盖地面面积较小，细节过载，覆盖区域有限，图像检索内容重复，降低定位的性能。如果需要验证在不同飞行高度时算法的性能指标，应该计划在不同飞行高度飞行，例如300m, 600m, 900m。验证在不同空间分辨率下的表现。
3. **不需要限制图像重叠度**。
4. **飞行时间和飞行姿态**：如果需要验证在不同飞行时间和飞行姿态时算法的性能指标，应该计划在不同的时间段（如上午、下午和晚上）进行飞行，以及记录无人机的各种姿态（偏航角yaw、俯仰角pitch和滚转角roll）。这样的覆盖可以确保算法在日照变化和不同光照条件下的性能得到准确评估。同时，通过测试不同的飞行姿态，可以评估算法在处理动态变化时的稳定性和准确性。

### 2.2 预处理

预处理是为后续网络模型提供高质量、精确对齐的热红外-卫星图像对，从而提高模型的优化效率和整体性能。需要指出的是，本节介绍的内容主要是模型采用重新训练或者微调的方式，如果直接用预训练模型则不需要下面的步骤，直接裁剪一定区域的卫星图像即可。本节具体步骤包括以下内容：

1. **生成正射影像**：首先，将无人机搭载的热红外摄像头捕获的原始数据处理成正射影像。经过地理校正，实现与卫星地图相匹配的正射影像（Geo-TIFF文件）。
2. **获取卫星影像**：获取与正射影像相同空间区域的图像，同时保持空间分辨率一致，目前主要采用1m/pt, 2m/pt空间分辨率的卫星影像。
3. **裁剪、清洗与数据集划分**：从Geo-TIFF文件中精确裁剪出大小为512x512像素的图像块，采用32像素的步长进行滑动窗口裁剪，以最大化地利用原始图像，生成大量的样本。对边缘区域以及重建质量差的热红外图像进行数据清洗。然后，对其训练/测试/验证数据集进行划分，满足深度学习模型的训练测试需求。

#### 2.2.1 生成正射影像

利用开源项目OpenDroneMap (ODM) ，处理从无人机获取的图像数据，并将无人机拍摄的重叠图像拼接成高分辨率的正射影像。该过程中涉及以下步骤：

1. **图像匹配**： 特征提取和匹配，从每一张图像中提取特征点，将不同图像中的特征点进行匹配，以确定它们之间的几何关系。
2. **SfM（Structure from Motion）**：相机定位和三维重建，使用匹配的特征点来估计每张图像的相机位置和姿态，基于相机参数和图像内容，重建场景的三维模型。
3. **生成点云和网格模型**：从重建的三维数据中生成密集点云，然后将点云转换为三维网格模型。
4. **创建数字高程模型**：从点云中提取地面和非地面特征的高程数据。
5. **生成正射影像**：使用数字高程模型对每张图像进行纠正，消除建筑物和其他高出地面物体的视角偏差。然后，将纠正后的图像拼接和融合成一个连续的，无缝的正射影像。

#### 2.2.2 获取卫星影像

将上一节中生成的热红外正射影像导入QGIS。

点击“图层”->“添加图层”->“添加栅格图层”。

选择正射影像文件（如 odm\_orthophoto.tif）并点击“打开”。

添加谷歌地图。

在QGIS中，点击“插件”->“管理和安装插件”。

搜索并安装“QuickMapServices”插件。

安装完成后，点击“Web”->“QuickMapServices”->“OSM”->“Google”->“Google Satellite”。

将热红外正射影像与谷歌地图对齐，修改为相同的空间分辨率和坐标参考系。具体如图2所示，左图为生成的正射影像，右图为卫星地图。



图2 热红外正射图像与卫星图像

#### 2.2.3 裁剪、清洗与数据集划分

沿odm\_orthophoto.tif文件裁剪，像素大小为512×512，步长为32进行滑动窗口裁剪，坐标采用512×512子图中心点的位置坐标，并将EPSG:3857转换为UTM坐标。图像名称采用“@lat@lon@yaw@pitch@roll@alt...@.jpg”的格式。对边缘区域以及重建质量差的热红外图像进行数据清洗。

数据集划分如表1所示，名称data后第一个数字代表空间分辨率，1为1m/pt， 2为2m/pt；第二个数字用来区分拍摄的热红外图像的时间，这是因为不同时间拍摄的热红外图像会对结果产生显著影响，尤其是由于环境温度和太阳辐射强度的变化，这些变化会影响物体表面的温度，从而影响热红外图像中物体的热辐射。具体地，1为11：04时拍摄的数据；2为15：29时拍摄的数据；3为17：30时拍摄的数据。

表1 数据集划分

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 训练集 | 验证集 | 测试集 |
| total | 20656 | 2578 | 2578 |
| data\_1\_1 | 5840 | 729 | 729 |
| data\_1\_2 | 5871 | 733 | 733 |
| data\_1\_3 | 5923 | 740 | 740 |
| data\_2\_1 | 1011 | 126 | 126 |
| data\_2\_2 | 998 | 124 | 124 |
| data\_2\_3 | 1013 | 126 | 126 |

**生成.h5文件**

读取图像文件，对应生成.h5文件，该文件需要包含两个dataset的名称，需要设置为”image\_data”、”image\_name”，名称下对应的内容为两个numpy.array格式的数据。image\_data与image\_name需要一一对应。

## 训练与验证

### 3.1 网络模型

目前，选用了预训练的ResNet18模型作为特征提取的基础网络，为了提升训练效率并充分利用已有的模型资产，固定ResNet18中直至conv\_3之前所有层的参数，不进行进一步参数训练，只对后续层进行优化。这样做的目的是保留这些层在预训练过程中学到的通用特征，这些通用特征有利于增强模型的泛化能力，减少过拟合的风险。

对于特征的进一步聚合，使用NetVLAD层，NetVLAD层接收来自ResNet18的特征图。使其能够输出一个固定大小为4096的描述向量，用来可以度量两个图像之间的相似度，适合进行图像检索任务。

最后，为了实现高效的特征检索，还是采用faiss库实现快速地在大规模卫星图像数据库中找到与热红外查询图像最相似的图像，从而实现图像检索和定位任务。

### 3.2 在线数据增强

为了更好地覆盖模拟真实场景，采用以下数据增强技术：

几何变换：包括随机透视、旋转和平移变换，增强模型对于视角和位置，飞行姿态变化的鲁棒性。

尺度变换：通过随机缩放技术模拟无人机在不同高度的成像效果。

对比度增强：为了增强模糊的热红外特征，选择对热红外图像进行对比度增强，使用对比度因子为3 的线性缩放对比度调整。

具体如图3所示，图3展示了部分数据增强的结果。

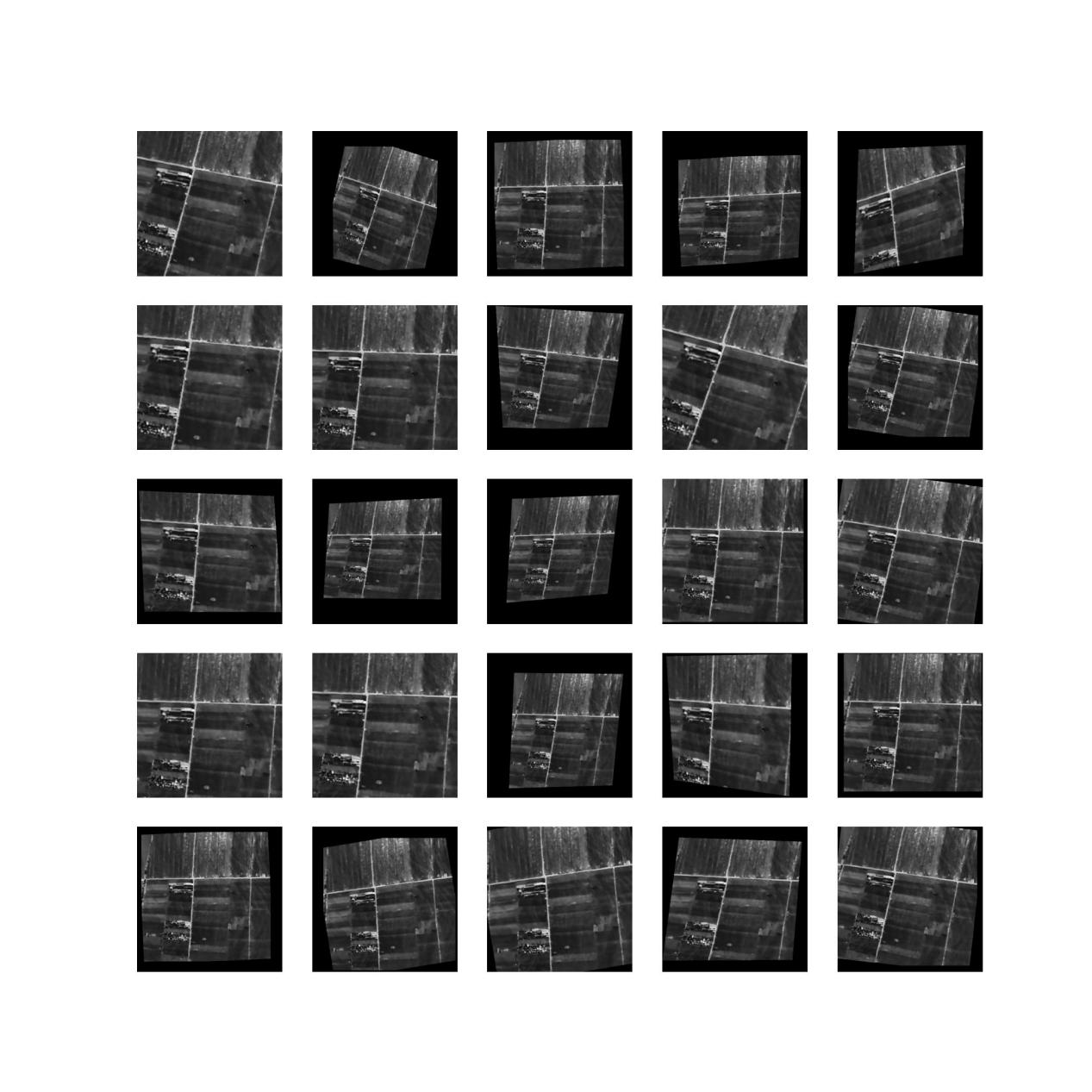


图3 在线数据增强

### 3.3 训练超参数设定

学习率设置为0.00004，以微调的形式训练40轮，采用warmup预测策略在训练初期逐渐提升学习率，防止训练初期由于学习率设置不当导致的不稳定。使用Adam优化器，线性衰减策略来逐步降低学习率。

此外，在指标计算时，设置正样本的最近邻搜索半径为60m。

### 3.4 测试集实验结果

在测试集上计算召回率，@后面的值为检索的位置前1/5/10/20个中含有正样本，该测试集不代表真实的场景，是从正射图像滑动窗口裁剪所组成的。

R@1: 99.5, R@5: 100.0, R@10: 100.0, R@20: 100.0

### 3.5 实际场景热红外图像前处理

在实际场景中，由于飞行姿态导致拍摄的图像与正射图像相比，存在几何畸变，因此我们在前处理时先根据无人机的飞行姿态计算变换前后点的位置，以此来进行透射变换，来近似重投影过程。

（1）**计算旋转矩阵** 根据俯仰角(pitch)，偏航角(yaw)，滚转角(roll)计算旋转矩阵。

（2）**定义原图像中的四个点** 定义图像四个角点的坐标，如 (0,0)，(width−1,0)，(width−1,height-1)，(0,height−1)。

（3）**计算目标图像的四个点** 通过旋转矩阵计算目标图像的四个点。将源点坐标中心化（可以考虑相机内参的影响），应用旋转矩阵，再恢复到原始位置。由于点是2D点，不使用旋转矩阵的最后一列。

（4）**计算透射矩阵** 使用 cv2.getPerspectiveTransform 通过原图像和目标图像的四个点计算透射矩阵。

（5）**应用透射矩阵** 使用 cv2.warpPerspective 将透射矩阵应用到原图像，以得到目标图像。

（6）**中心裁剪** 将图像中心裁剪至512x512大小

（7）**调整对比度** 对热红外图像进行对比度增强

（8）**归一化** 均值：[0.485, 0.456, 0.406]；标准差：[0.229, 0.224, 0.225]



图4 效果图

图4中，左图为热红外相机拍摄的原始图像，右图为热红外图像经过前处理后的图像，中间图像为检索到的卫星图像。

## 部署

将算法部署到RK3588开发板上，整体的开发步骤主要包含3个部分：模型转换、模型评估和板端部署运行。

1. **模型转换**：在这一阶段，原始的深度学习模型会被转化为RKNN格式，以便在RKNPU平台上进行高效的推理。这一步骤包括：
2. 获取原始模型：获取Pytorch训练的深度学习模型，将其转换为通用框架ONNX。
3. 模型配置：在RKNN-Toolkit2中进行必要的配置，如均值，归一化，量化参数和目标平台等。
4. 模型加载：使用适当的加载接口将模型导入RKNN-Toolkit2，根据模型类型选择正确的加载方法。
5. 模型构建：通过rknn.build()接口构建RKNN模型，可选择是否进行量化，提高模型部署在硬件上的性能。
6. 模型导出：通过rknn.export\_rknn()接口将RKNN模型导出为一个文件（.rknn格式），用于后续部署。
7. **模型评估**：模型评估阶段用来量化和分析模型性能，包括精度，连板推理性能和内存占用等关键指标，评估的主要步骤包括以下操作：
8. 模型连板调试：模型连板调试阶段是通过使用Python连接RKNPU平台进行推理并验证模型，这个阶段涵盖了输入数据的前处理和输出结果的后处理，以确保模型在板上运行正确。
9. 精度评估：通过rknn.accuracy\_analysis()接口，比较量化模型与浮点模型之间的输出分析量化误差。
10. 性能评估：通过rknn.eval\_perf()接口，了解模型在平台上的推理性能，以便进一步优化模型结构，加快推理性能。
11. 内存评估：通过rknn.eval\_memory()接口，了解模型在板上的内存使用情况，进一步优化模型结构最小化内存占用。
12. **板端部署运行**：这个阶段涵盖了模型的实际部署和运行，通过包括以下步骤：
13. 模型初始化：加载RKNN模型到RKNPU平台，准确进行前处理。
14. 模型前处理：加载待推理数据到RKNPU平台，准备进行推理。
15. 模型推理：执行推理操作，将输入数据传递给模型并获取推理结果。
16. 模型后处理：取出推理结果进行后处理，后处理结构传给应用端。
17. 模型释放：在完成推理流程后，释放模型资源，以便其他任务使用RKNN模型。

### 4.1 模型转换

目前，直接将 Pytorch 模型转换为 RKNN 模型的支持有限。一般推荐的流程是先将 Pytorch 模型转换为 ONNX 模型，再通过 RKNN ToolKit2 将 ONNX 模型转换为 RKNN 模型。这是官方推荐且最稳定、广泛支持的方法。

在转换 ONNX 模型时，采用了版本为15的算子集，并固定了计算图的数据维度，以提高效率。随后，使用 onnx-simplifier 工具简化 ONNX 模型，移除冗余节点和复杂的计算图，从而优化模型的运行速度，减少内存占用，并提高模型的可读性。接着，使用 ONNXruntime 进行模型测试，对比简化前后的性能，如图 5 所示。

在从 ONNX 转换为 RKNN 的过程中，没有对模型进行量化，此时模型的网络层参数默认采用 Float16 精度。最后，导出 .rknn 文件，以便后续进行模型部署。

### 4.2 模型评估

#### 4.2.1 模型连板调试

将原始模型转换为 RKNN 模型后，可以将其连接到开发板进行推理，并对推理结果进行后处理以验证其正确性。由于 RKNN ToolKit2 需要通过 ADB 连接开发板，因此需要在开发板上启用 ADBD 服务。我们采用编译 Android 源代码的方法来创建 systemctl 启动服务。在开机后，通过运行 systemctl start adbd.service 来启动该服务，并使用 systemctl status adbd.service 查看服务运行状

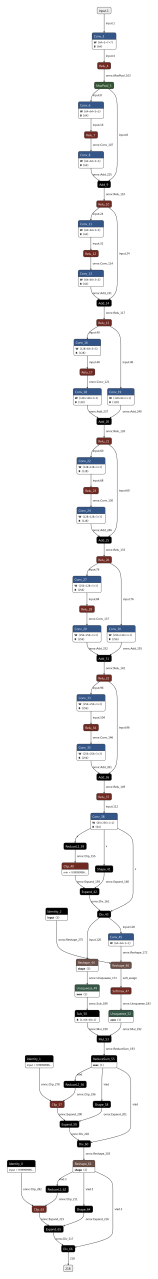
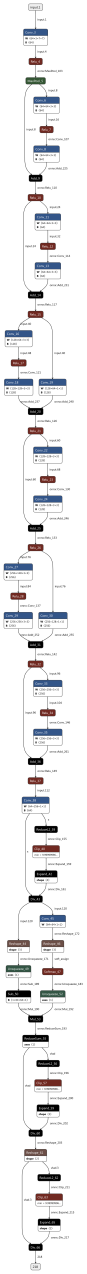
 

图 5 ONNX计算图对比

态。在 PC 端，需要安装 ADB 工具，并确保 PC 和开发板处于同一局域网内，或者通过 USB 连接。如果使用无线连接，可以使用 adb connect <ip>:5555 命令进行连接，并通过 adb devices 命令查看连接状态。确认连接成功后，即可运行连板推理程序，并验证其正确性。

#### 4.2.2 性能及内存评估

在对模型的性能评估前，需要对硬件（包括CPU、NPU和DDR）进行定频操作。目前采用的配置如下：

CPU 频率：1200000 Hz

NPU 频率：800000000 Hz

DDR 频率：1596000000 Hz

在Python代码下处理单张图像的总时长(没有包括初始化，加载本地数据库等过程，工程仓库下有分析这一部分)约为180ms，其中具体分布如下：

前处理：约50ms

推理过程：约80ms

后处理：约50ms

在工程仓库中，我们记录了更全面的CPU性能分析，包括函数调用和原始调用次数、累计时间和内部时间等详细数据。此外，还记录了在上述频率下推理过程中每一层的耗时情况。

在整个过程中，Python代码占用的内存大约为940M，其中500M是加载本地数据库特征所占用的内存，推理过程占用大约34M，其他部分主要是Faiss特征检索所消耗的内存。这些详细的内存使用情况也在工程中有详细说明。

在C++代码下处理单张图像的总时长约为150ms，其中具体分布如下：

前处理：约25ms

推理过程：约75ms

后处理：约50ms

【C++其他的还没测】

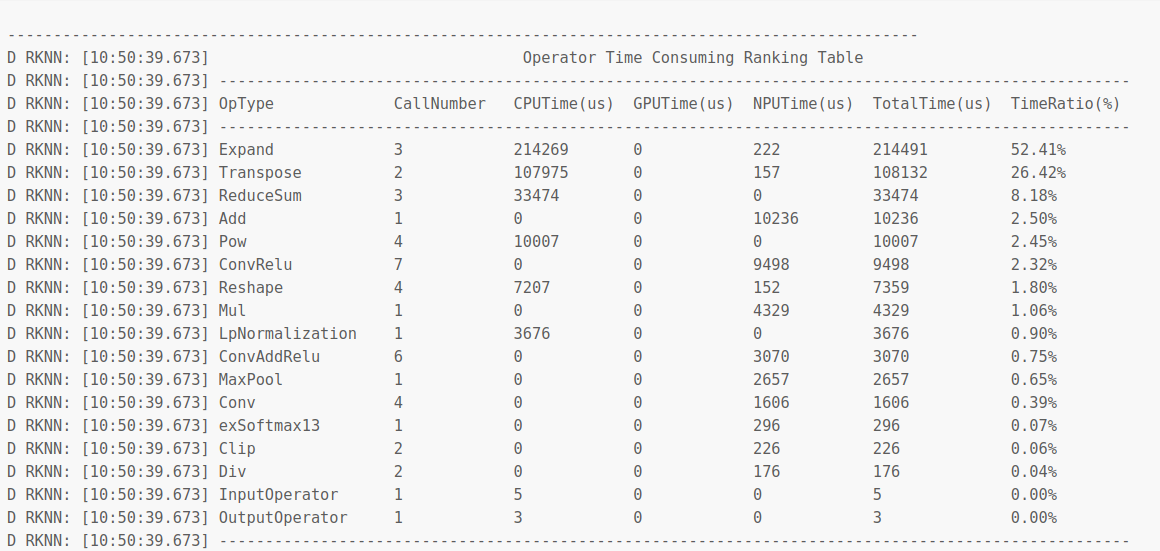


图6 各算子耗时情况

### 4.3 性能优化

此前，我们根据各算子耗时以及逐层耗时对其中的高耗时算子进行了优化。在优化前仅推理过程需要耗时400ms，下图6是按时间消耗排序列出的每个操作类型的详细信息，包括操作类型，调用次数，CPU时间，GPU时间，NPU时间，总时间以及时间比例。可以看出，Expand和Tranpose两个算子占用了78%，耗时300ms，改进后如图7所示。

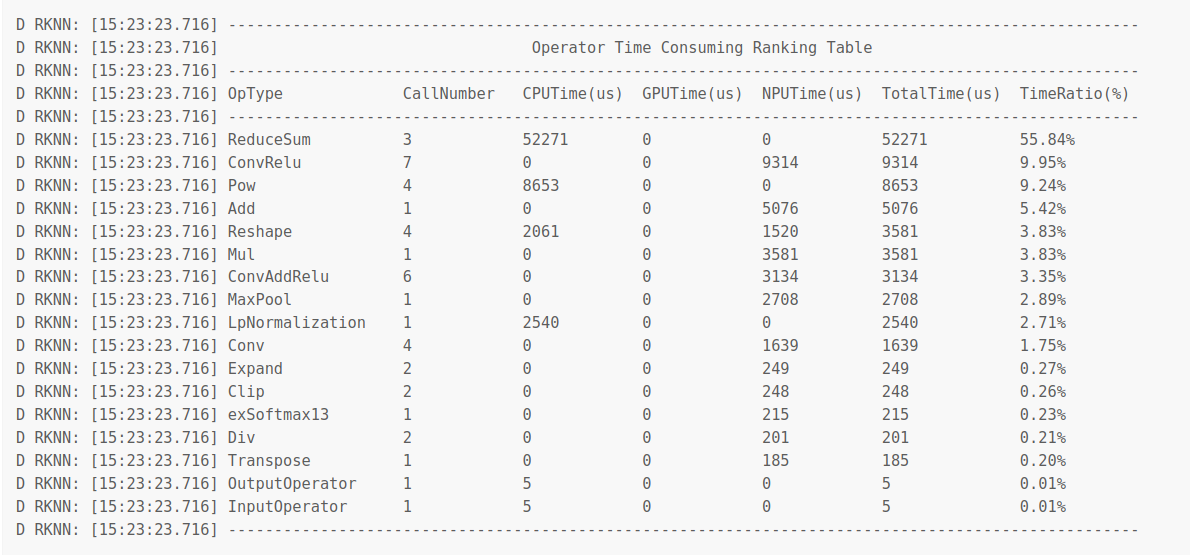


图7 各算子耗时情况

### 4.4 板端部署运行

工程仓库地址：https://github.com/Xjg-0216/UAV-VisionLoc-Deploy

#### 4.4.1 工程框架

该工程下包含python和c++两种部署代码，根目录下包括以下内容：

1. cpp : C/C++板端部署
2. python： python板端部署
3. data: 包含database本地数据库，queries图像
4. model: RKNN模型
5. .vscode: 配置文件

以下是C++和Python目录下的具体介绍

|-- cpp

| |-- 3rdparty 三方库，目前用到的有opencv, NPU的Runtime库

| |-- CMakeLists.txt

| |-- build 编译生成目录

| |-- build.sh 自动化构建脚本

| |-- faiss-demo 相似性搜索测试

| |-- include 头文件目录

| |-- install 可执行文件目录

| |-- run.sh 自动化执行脚本

| `-- src 源文件目录

`-- python

|-- eval.py 验证板端准确率（加载h5文件数据）

|-- main.py 部署运行

|-- memory 内存分析

|-- profile 性能分析

|-- py\_utils

`-- result

#### 4.4.2 配置文件

根目录.vscode文件夹用于存储与项目相关的配置文件。

tasks.json文件用于定义C++代码的生成任务，运行的同样是build.sh脚本，python则不需要生成任务；

launch.json文件用于配置调试器，可以用来调试python或者C++程序。定义C++调试时program为install文件夹中生成的可执行文件，args中需要包含rknn模型路径，图像路径和本地数据特征路径。preLaunchTask需要与tasks.json中的label一致。此外，还需配置python的调试器，默认执行当前文件即可。

c\_cpp\_properties.json用于定义 IntelliSense、编译器路径、包含路径和其他与 C/C++ 相关的配置。

#### 4.4.3 环境配置

在C++中，需要安装HDF5库以及Faiss库。以下是具体的安装方法和配置步骤：

HDF5库可以通过包管理器进行安装，使用以下命令安装HDF5开发库：

sudo apt update && sudo apt install libhdf5-dev

Faiss库可以采用conda进行安装，通过命令：  
 conda install -c conda-forge faiss-cpu进行安装，该命令会安装Faiss库的python接口，同时也包含了C++库文件。在CmakeLists.txt文件中需要链接这两个库的路径。

在Python环境中，可以参考项目中的requirements.txt文件来安装所需的依赖库。确保requirements.txt文件中包含了项目所需的所有Python库。

#### 4.4.4 构建和运行

在C++项目中，可以通过执行 ./build.sh 脚本来编译和构建可执行程序。然后，可以通过执行 ./run.sh 脚本来运行程序。run.sh 脚本可以命令行参数可修改模型、输入图像以及本地数据库特征H5文件的路径。

对于Python项目，可以直接运行 main.py 来实现部署和运行。使用 argparse 模块处理命令行参数选项，可以方便地修改RKNN模型路径、图像路径和本地数据特征路径等。