

# 嵌入式人体姿态估计

张嘉伟 王佳 王天天

西安电子科技大学人工智能学院

**摘要** 嵌入式姿态估计是一项关键的计算机视觉任务，广泛应用于无人机、机器人、增强现实等领域。本项目旨在设计和优化一种适用于嵌入式系统的实时姿态估计算法。为了实现实时性能，算法需要在资源受限的嵌入式硬件平台上运行，并且具有较低的计算和存储需求。本项目在 OpenMV4 Pro 平台上进行实践，借助 TinyNeuralNetwork, TinyEngine 等开源库，部署了 lightweight-openpose, movenet 等用于姿态估计的深度学习模型。

## Embedded Human Pose Estimation

### Abstract

Embedded Pose Estimation is a critical task in computer vision, widely used in drones, robots, augmented reality, and other fields. This project aims to design and optimize a real-time pose estimation algorithm suitable for embedded systems. To achieve real-time performance, the algorithm needs to run on resource-constrained embedded hardware platforms and have low computational and storage requirements. This project is practiced on the OpenMV4 Pro platform, with the help of open-source libraries such as TinyNeuralNetwork and TinyEngine, and deploys deep learning models for pose estimation such as Lightweight OpenPose and Movenet.

## 1 引言

人体姿态估计是计算机视觉领域的一个重要研究方向，旨在从图像或视频中推断出人体的姿态信息。人体姿态包括关节的位置、角度和骨架结构，对于理解人体动作、行为分析、人机交互等应用具有重要意义。

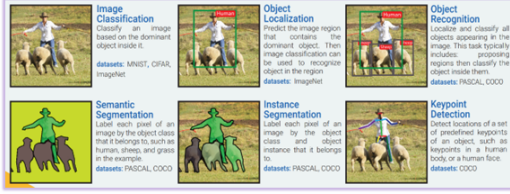


Fig. 1 姿态估计与其他计算机视觉任务的对比

姿态估计与图像分类、目标检测等其他计算机视觉任务存在一些关键的区别。首先，任务目标不同：图像分类是确定图像属于哪个预定义类别，目标检测是在图像中定位和标记物体的位置，而姿态估计是推断人体或物体的姿态信息，包括关节的位置、角度和骨架结构。其次，输出结果不同：图像分类输出类别标签，目标检测输出边界框和类别标签，而姿态估计输出关键点坐标或骨骼结构。此外，姿态估计更具挑战性，需要建模物体的内部结构和姿态变化，而图像分类和目标检测更注重整体特征和类别分类。姿态估计还需要复杂的数据标注，相对于简单的图像分类和目标检测标注。最后，姿态估计的主要应用领域包括人机交互、动作捕捉、虚拟现实和运动分析等。综合而言，虽然姿态估计与其他任务有区别，但不同任务的技术和方法可以相互融合，实现更复杂和全面的计算机视觉应用。

当涉及到人体姿态估计时，常见的两种方法是基于热图和基于回归。

基于热图的方法在人体姿态估计中被广泛使用。它的核心思想是通过在图像上生成一系列的热图（也称为关键点热图或关键点响应图），来表示人体关键点的位置。每个关键点在热图上都有一个对应的峰值，可以通过找到热图中的峰值位置来推断关键点的位置。通常，使用卷积神经网络（CNN）或类似的深度学习模型来生成这些热图，其中网络的输出是一个具有与关键点数量相同的通道数的热图张量。然后，根据峰值检测算法或阈值设定来确定关键点的位置。

另一种方法是基于回归的人体姿态估计。该方法直接回归出关键点的坐标值，而不是生成热图。在这种方法中，通常使用回归器，如卷积神经网络（CNN）或全连接网络，将输入图像映射到关键点的

坐标输出。网络的最后一层通常是具有与关键点数量相同的输出节点，每个节点对应一个关键点的坐标。

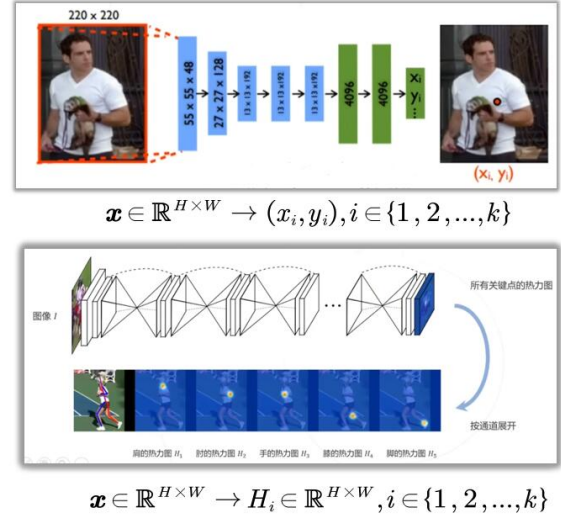


Fig. 2 热图与回归两种方法

这两种方法各有优势和适用场景。基于热图的方法可以提供对关键点位置的更准确的预测，因为每个关键点都在热图上有一个明确的峰值。此外，基于热图的方法还可以处理关键点的可见性问题，即使关键点在图像中不可见或部分遮挡，热图仍然可以提供有关位置的信息。基于回归的方法则更加直接和简单，直接从输入图像回归出关键点的坐标，而无需生成中间热图。这使得基于回归的方法在计算效率上通常更高，并且更适合于实时应用和资源受限的设备。

当计算机视觉领域的人体姿态估计算法迈向实际应用时，嵌入式系统的崛起成为一项重要的发展方向。传统的人体姿态估计算法主要依赖于高性能计算机和强大的计算资源，这限制了其在资源受限的嵌入式设备上的应用。然而，随着嵌入式系统的不断进步和性能提升，越来越多的应用场景对于在嵌入式设备上实现实时和准确的人体姿态估计提出了需求。

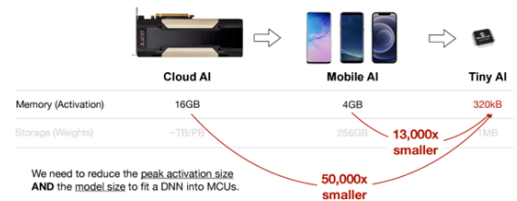


Fig. 3 不同设备下的部署条件

常用的嵌入式设备有 Raspberry pie, Jetson Nano 等。但这些嵌入式开发设备的成本仍然比较昂贵。基于此, 本项目旨在探索基于微处理器的姿态估计方案。微处理器是一种在嵌入式设备中广泛使用的中央处理器, 具有较小尺寸和较低功耗的特点。但其内存和储存空间非常有限。在这样的微处理器下进行深度学习模型的部署, 需要特殊的框架, 例如 TF-Lite for Micro。TF-Lite for Micro 是 Google 开发的 TensorFlow Lite 的一个版本, 专门针对资源受限的嵌入式设备而设计。它是一个轻量级的深度学习推理框架, 旨在在低功耗、低内存和计算资源有限的设备上执行深度神经网络模型。它通过优化的量化和压缩技术, 可以在保持模型准确度的同时显著减小模型的尺寸, 从而降低内存占用和计算要求。这使得它适合于资源受限的环境, 可以在边缘设备上实现实时的机器学习功能。

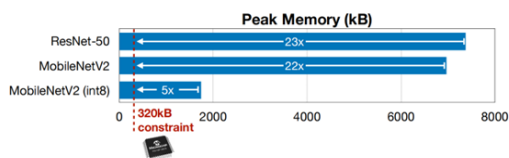


Fig. 4 微处理器内存和储存空间有限

## 2 实验方法

我们最终采取了如下的解决方案:

- 1) 在 PyTorch 框架下训练基于热图的深度学习模型。
- 2) 借助 TinyNeuralNetwork 开源库将 PyTorch 下的模型转换为 tflite 格式。
- 3) 使用 TinyEngine 库将 tflite 模型转换为 C 的实现。
- 4) 根据生成的 C 代码修改注册所需要的函数和算子。
- 5) 编写 MicroPython 的 C 扩展来实现后处理。
- 6) 编译整个项目, 生成固件文件。
- 7) 借助 OpenMVIDE 实现固件烧录。

这一过程中用到的第三方依赖如下:

### 一、PyTorch

PyTorch 是一个基于 Python 的开源机器学习框架, 专注于深度学习任务的开发和研究。它由 Facebook 的人工智能研究团队开发并于 2016 年发布, 得到了广泛的应用和社区支持。

PyTorch 的设计目标是提供简单、灵活和高效的接口, 使得构建、训练和部署深度神经网络模型变得更加容易。它以动态计算图为基础, 允许用户以类似于 Python 编程的方式定义和操作神经网络模型。这种动态图的设计方式使得 PyTorch 非常适

合研究、实验和迭代开发, 能够快速实现新的模型和算法。我们使用 PyTorch 来训练待部署的深度学习模型, 搭建整个 pipeline。方便迭代更新和快速实现。

### 二、TinyNeuralNetwork

TinyNeuralNetwork 是一个高效、易用的深度学习模型压缩框架。它包含模型结构搜索、剪枝、量化、模型转换等功能, 能将巨大的深度学习模型压缩数倍到数十倍, 目前在天猫精灵、海尔电视、优酷视频、人脸打卡机等场景中完成了落地, 为超千万的 IoT 设备提供了 AI 能力。我们使用 TinyNeuralNetwork 来将 torch 格式的模型一步转移为 tflite 的格式, 并进行量化。

### 三、Openmv

OpenMV 是一款基于嵌入式系统的计算机视觉平台, 专门设计用于开发和运行视觉应用。它采用了一种小型化、低功耗的硬件平台, 配备了高性能的图像传感器和处理器, 以及专门为计算机视觉任务而设计的软件开发工具。OpenMV 的硬件平台包括一个主控板和相机模块。主控板上搭载了微控制器, 例如 STM32 系列, 具有较强的计算能力和丰富的外设接口, 用于图像处理和计算。相机模块集成了高质量的图像传感器, 能够以高帧率捕捉图像, 并将其传输给主控板进行处理。OpenMV 还具有丰富的扩展接口, 可以连接各种传感器和执行器, 以实现更复杂的视觉应用。OpenMV 平台还提供了一套易于使用的软件开发工具, 包括基于 Python 的编程语言和专门为计算机视觉任务而设计的库。开发者可以使用 Python 语言编写图像处理和计算机视觉算法, 并通过 OpenMV IDE 进行代码编写、调试和上传。OpenMV 库提供了丰富的图像处理函数和算法, 包括图像滤波、特征提取、目标检测和跟踪等功能, 极大地简化了计算机视觉应用的开发过程。



Fig. 5 openmv 开发板

项目中使用的处理器为 ARM Cortex M7, 480MHz。整个设备有 1MB 的 RAM 和 2MB 的 flash。

#### 四、TinyEngine

TinyEngine 是一个内存高效的推断库。它通过根据整个网络拓扑而非逐层优化来适应内存调度，从而减少内存使用并加速推断过程。TinyEngine 超过了谷歌的 TF-Lite Micro、Arm 的 CMSIS-NN 以及 STMicroelectronics 的 X-CUBE-AI 等现有推断库。

TinyEngine 在微处理器上的地位就如平常设备中 PyTorch 的地位一样。其他的一些推断库，如 TF-Lite，虽然有更成熟的生态，但很难在如此低 RAM 的条件下执行。

### 3 实验克服的问题

我们在这里简要回忆并记录一下，整个开发过程中遇到的一些特殊情况：

- 采集出的图像颜色明显不对



Fig. 6 采集出的图像颜色不正确

后来发现，是因为所用型号为 openmv4p，而编译时的命令为 `make -j4 TARGET=OPENMV4 -C src`，应该使用 `make -j4 TARGET=OPENMV4P -C src`。同时，需要再多打一个 patch：`git apply ../openmvmp4p_person_detection.patch`

- 使用 openmvIDE 时，反复弹出“忙碌中”的中断同时摄像头即使变焦了也很糊，得不到清晰的成像。

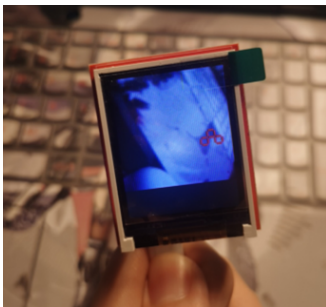


Fig. 7 得不到清晰的成像

在我们开发时，这是由于摄像头以及感光部件和下面的开发板接触不良导致的，拆下来重新装一次就好了。使用量化后的模型，在编译时会遇到：

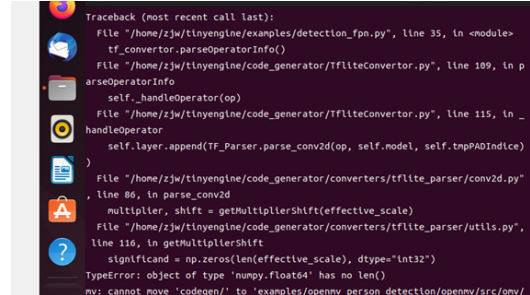


Fig. 8 量化设置不正确

这实际上是因为由于量化时设置的不正确，导致量化后的模型，例如卷积层每个通道的权重，都用的相同的一个量化系数：

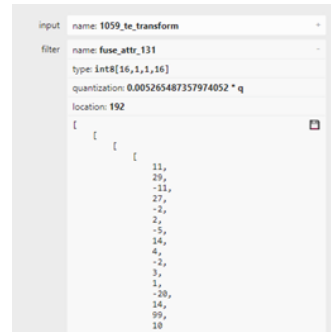


Fig. 9 卷积层每个通道的权重用的相同的量化系数

而 TinyEngine 的解释脚本中，默认每一个输入会是一个含多个参数的列表。只需在进行量化时，将量化器设置为：`config='asymmetric': True, 'per_tensor': False`

- 模型中的一些 tflite 算子在 TinyEngine 上没有部署

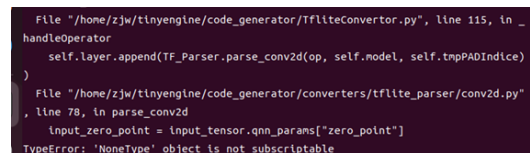


Fig. 10 一些 tflite 算子在 TinyEngine 上没有部署

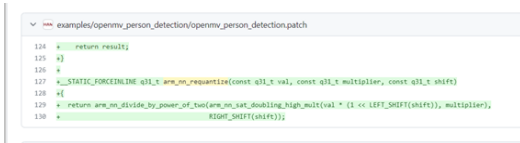
解决方案有，替换掉这个算子；以及使用



TinyEngine 另一个 branch 里所公开的, 支持更多算子的库。

- 在生成 C 代码时, `fp_requantize=false` 后, 没有定义 `arm_nn_requantize`

`arm_nn_requantize` 本是将 patch 打好后, 再 build 整个项目, 自然可以建立引用的一个函数。出现上述报错的原因可能是 patch 没有安装好, 或者之前 build 的那些依赖里是没有打 patch 的。一个通用的解决方法是将 `arm_nnsupportfunction.h` 直接 include 进 TinyEngine/Codegen/Incude 里的 `tinyengine_function.h` 中。这样无论从哪里开始编译, 都会把 `arm_nn_requantize` 引用进去。



```

124 // return result;
125 }
126 #
127 #define STATIC_FORCEINLINE __attribute__((static_inline))
128 #define __STATIC_FORCEINLINE __attribute__((static_inline))
129 #define __STATIC_FORCEINLINE __attribute__((static_inline))
130 #define __STATIC_FORCEINLINE __attribute__((static_inline))

```

Fig. 11 将 `arm_nnsupportfunction.h` 直接 include 进 TinyEngineCodegenIncude 里的 `tinyengine_function.h` 中

- TfLite 的模型所需的算子和编译时约定的算子有出入

此时可以寻找是哪个算子没有被注册, 无需重新写 patch, 可以在 `omv` 下的 `makefile` 文件里找到相应的位置, 直接修改; 同理也要修改 `ports/stm32/` 中的 `omv_config.mk` (应该是这个名字) 文件。

- 不支持一般的奇数卷积

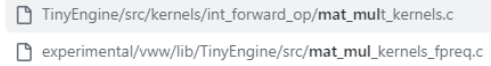
TinyEngine 并没有支持寻常的奇数卷积的实现, 因为他们不能被高效的优化掉。而且用的比较少。但例如 COCO2017 数据集, 其中是有 17 个关键点的。这样的话只能手动修改整个训练 pipeline, 将其改为 16, 或者删去其他的关键点。

- 编译不通过, 提示 xxx SDRAM overlap

这说明模型需要的 `peak_memory` 太大而不可承受, 考虑替换模型, 缩小通道数, 删去跳连等。

- 在注册矩阵乘法的部分时, 出现问题。

矩阵乘法在计算卷积时是被频繁调用的, 而 TinyEngine 的源代码里存在两种实现, 要注意分辨, 有时会引起困惑。



```

TinyEngine/src/kernels/int_forward_op/mat_mult_kernels.c
experimental/vwv/lib/TinyEngine/src/mat_mul_kernels_fpreq.c

```

Fig. 12 矩阵乘法出现两种实现

- 有些时候, 我不想要训练好的模型的其他一些监督头, 但训练时要用到。

在使用 TinyNeuralNetwork 时, 这是一个非常有趣的问题。我们在量化感知训练时, 是操纵了 `qat_model`, 插入伪量化结点和伪解量化结点来训练。然后最后用一行 `qat_model = quantizer.convert(qat_model)` 来将伪量化变成真的量化。这一步以后得到的 `qat_model`, 我们不能像寻常的调用之前的 `torch` 对象一样来处理它。常规来看下一步我们就会用 `TFLiteConverter()` 来将模型转为 `tfLite`。当模型按照 `tfLite` 定义后, 我们很难对其结构进行操纵了。所以一个方法是, 当得到 `qat_model` 后, 我们直接将不想要的那些部分替换为 `nn.Identity()`, 然后利用 `partial()` 重写 `qat_model` 的 `forward()` 函数。

## 4 实验结果与分析

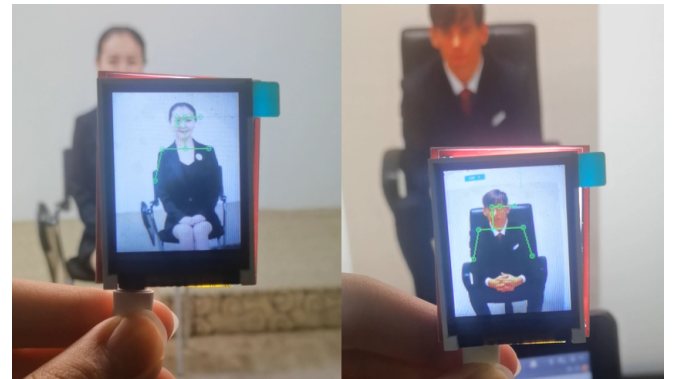


Fig. 13 实验结果 demo

在我们的实践中, 我们尝试了两种不同的方法: Movenet 和 Lightweight OpenPose。

首先, 我们尝试使用 Movenet。Movenet 是一种轻量级的实时人体姿态估计模型, 由 Google AI 开发, 主要用于移动和嵌入式设备。它以高效、实时和准确为目标, 通过优化网络结构和计算过程, 实现在资源受限的设备上进行高效的人体姿态估计。尽管我们投入了大量的时间和精力, 但是在写报告时, Movenet 的实践仍未能成功。我们遇到了许多挑战和问题, 包括模型转换、算子支持、硬件资源限制

等。由于这些问题，我们无法在嵌入式设备上成功部署和运行 Movenet。我们只能提供在 TensorFlow Lite 环境下的推断效果。

其次，我们尝试使用 Lightweight OpenPose。Lightweight OpenPose 是一种轻量级的人体姿态估计模型，其设计目标是在保证精度的同时，降低模型的复杂度和计算量，从而适应资源受限的设备。我们使用的 Lightweight OpenPose 模型由王世龙学长提供。在实践中，我们发现我们对热图的后处理较为粗糙，处理效果并不理想。我们认识到，热图的后处理是一个关键步骤，它直接影响到姿态估计的精度和效果。为了改进后处理效果，我们可能需要进一步研究和优化后处理的方法。

总体来说，我们的实践过程充满了挑战，但也让我们获得了宝贵的经验。虽然我们在实践中遇到了许多困难，但我们也学到了许多。我们明白，要在嵌入式设备上实现实时姿态估计，需要面对和解决许多挑战，包括模型设计、计算优化、硬件资源限制等。我们将在未来的工作中，继续研究和探索更有效的方法和技术，以实现更好的实时姿态估计

效果。以下是其 tflite 下推断的效果：



Fig. 14 tflite 下推断的效果

## 5 实验总结

这是在人工智能学院就读三年来做过的最有挑战性的作业，这个作业需要我谨慎的思考一些之前从来没有考虑过的问题。比如 `cv2.imread()` 它读取图片的索引顺序。比如 flops, MAC 等这些之前只是 ShuffleNetV2 里讲出来的一些概念。比如之前，`nn.ReLU()` 到底 inplace 不 inplace，基本是一个不怎么被我关心的问题。

感谢王世龙学长的大力帮助，我们给他添了不少麻烦。