需求分析：

主要对比对象为 GeoNet、PWC-Net、FlowNet2 及其衍生型。其中 GeoNet 速度较慢，且其推理过程中需要额外推理深度图和相机位姿，并不属于任务需求；PWC-Net+网络结构较复杂，但网络规模并不大，且推理速度较快，推理精度较高；FlowNet2 网络规模过大，推理速度也较慢，但其子网络 FlowNetS 若利用 PWC-Net 的训练方案，可以在保证速度快的同时得到精度较高的结果。任务需求为在 Jetson TX2（256 NVIDIA CUDA Cores）平台上得到 20~30FPS 的光流图，因此决定采用 PWC-Net+及 FlowNetS。

利用 Tensorflow 实现的 PWC-Net+在 GTX 1080（2560 NVIDIA CUDA Cores）上推理 1024x436 图片的速度约为 80ms，而 FlowNets 在 Tesla K80（4992 NVIDIA CUDA Cores）上推理 512x384 图片的速度约为 38ms，因此需对网络进行优化。目前主流的优化模型的方法有低秩分解、转移/紧凑卷积滤波器、知识蒸馏、剪枝、量化等，其中转移/紧凑卷积滤波器和知识蒸馏无法使用预训练模型，而重新训练模型至少需要数天时间，且无法保证结果满足任务需求，由于时间有限，暂不予考虑；对于剪枝，支持 tensorflow 的剪枝工具是基于 Keras 的，而目前没有 Keras 的预训练模型，因此优先选择量化，将 float32 量化为 int8 可以将模型大小压缩为原来的 1/4，速度提升 2~3 倍。量化主要可以通过 TensorflowLite 来完成，或通过 TensorRT 完成。

利用 TensorflowLite 模型优化：

①在模型运行过程中保存 Summary 并利用 TensorBoard 打开，观察网络结构，记录网络的输入输出 Tensor 名。②从.ckpt 文件读入网络结构和权值到 graph，并利用 tensorflow 中的 graph_util.convert_variables_to_constants 命令将 graph 冻结，并保存为.pb 文件。③将卷积操作与标准化操作合为一步。④由于 TensorFlowLite 不支持 4D 以上的 Tensor，因此将 PWC-Net 中的 Slice 节点作为输入。而 FlowNetS 的原始输入为两个 Constant Tensor，将其修改为 Placeholder 以用于输入。⑤将冻结的.pb 文件转换为.tflite 文件，该操作根据给定的输入输出删除其他无关的、用于训练的节点，同时设置其量化参数为 OPTIMIZE FOR LATENCY，最终得到的 PWC 模型大小为原始.pb 文件大小的四分之一，而 FlowNetS 大小与原始大小相仿。

TensorFlowLite 模型部署：

由于 PWC-Net 的前处理和后处理较为复杂，且经测试其 tflite 模型速度低于 FlowNetS 较多，因此先改写了 FlowNetS 的前处理和后处理部分进行 TX2 上测试。若要在 TX2 上安装 Tensorflow，需要安装 Jetpack，因此只安装 TensorFLow-Lite 的子模块 TF-Lite-Runtime 用于运行.tflite 模型。不同平台和不同量化模型运行时间及结果如下：

Device-Model	T_read(s)	T_infer(s)	T_write(s)

TX2-CPU-WeightQuantized	0.0751457214468	2.6958911418914	2.2689899818115
TX2-CPU-Unquantize	0.0751292705888	2.6265156269486	3.012796939087
TX2-CPU-TotalQuantized	0.0751359345644	27.485623131234	2.211787972569
I7-7700- WeightQuantized	0.0312428426269	4.717685235596	0.8126820678711
I7-7700-Unquantize	0.0312139987945	0.6717047607422	0.4996505279541
48 cores Xeon @ 2.20GHz - OriginProgram	0.0239009857178	1.07844996452	N/A

表 1 各模型在不同平台上的运行时间

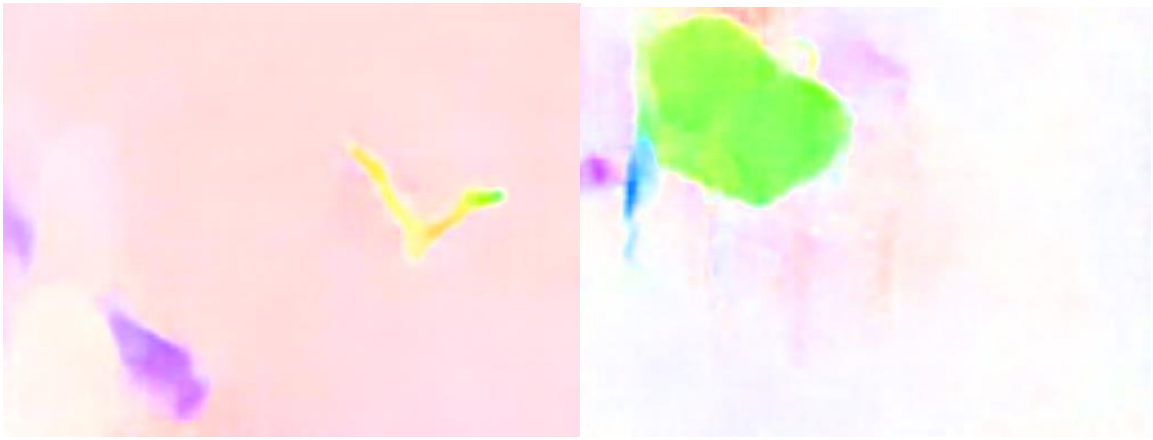

图 1 不进行量化的.tflite 结果

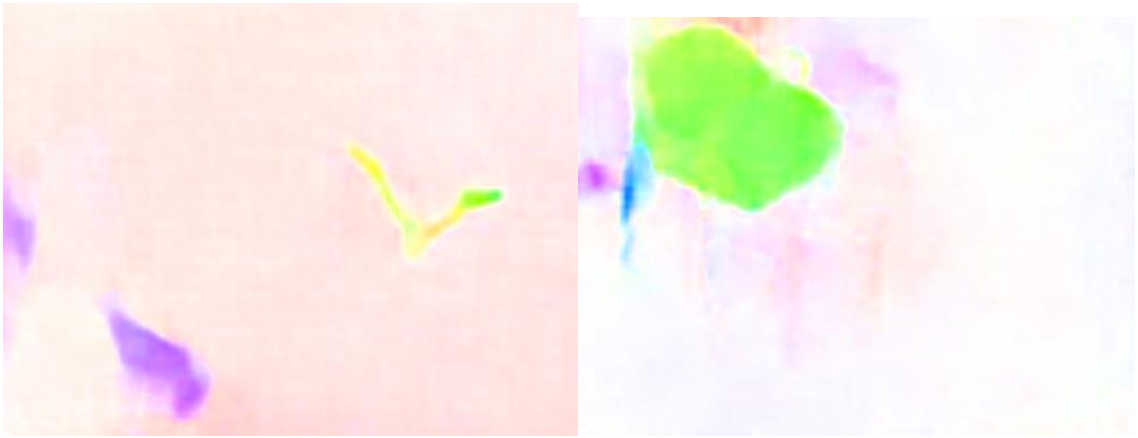

图 2 量化后的.tflite 结果

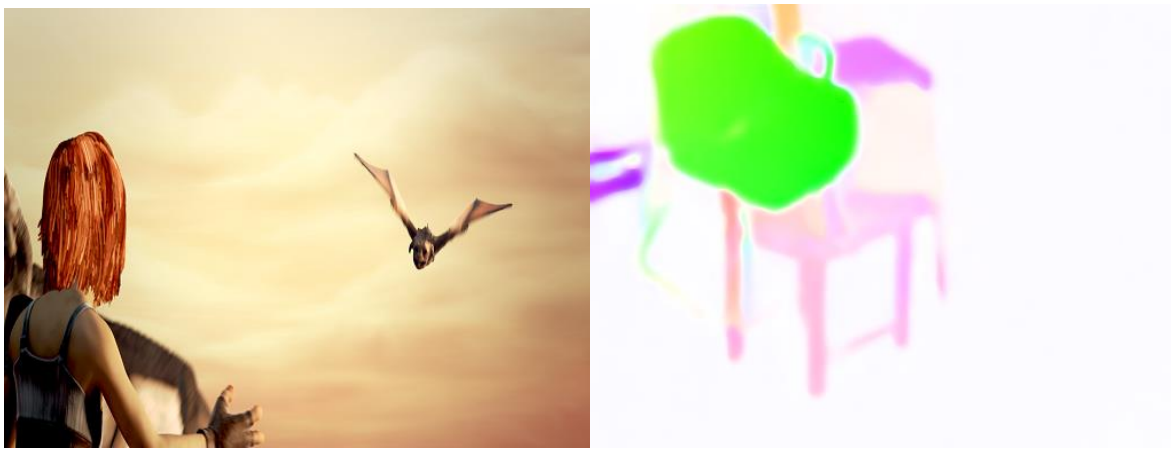

图 3 原始图像

转换成 Tensorflow-Lite 模型后，在 TX2 平台上进行权值量化并不能加速推理速度，甚至可能减慢了推理速度，但是却加快了写入图片的速度，而若将所有数学模型都进行量化，则会大大减慢推理速度，这是因为该操作会在每一层的前后都加入量化及反量化的操作，增大了计算负担。而对比 Tensorflow，TensorflowLite 对 ARM 架构下的指令集进行了优化，因此速度会稍快，但都达不到 20fps 的要求。之后尝试使用 16 位浮点数量化、GPU 加速，但官方文档只给出了 IOS 及 Android 的 GPU 代理接口，因此还需要进一步的研究。

TensorRT 加速：

TensorRT 消除了未使用的输出层以避免不必要的计算。同时对卷积，偏置和 ReLU 层进行融合以形成单个层。它还会优化卷积核选择，根据整型或浮点型优化数据矩阵等等操作来降低延迟，提高计算量和效率。这些图优化操作不会更改计算图中的基础计算：相反，它们会对计算图进行重新构建，使其可以更快，更有效地进行推理。所以 TensorRT 不需要依赖深度学习框架，而是可以直接优化训练完的模型文件。

TensorRT 不仅通过消除和融合层来优化图，而且将训练完的模型进行解析，然后与 TensorRT 中对应的层进行一一映射，把模型转化为 TensorRT 层组合的模型，然后对于模型中使用 NVIDIA GPU 的层进行优化和部署加速。尝试使用 TensorRT 优化 FlowNetS 的过程：①把从 TensorflowLite 优化过程中得到的冻结图转化为 uff 文件格式用于在 TX2 上使用。②在 TX2 上进行环境部署，包括安装 TensorRT 及其依赖库包括 Cuda, Cudnn 等（目前正在进行）③在 TX2 上使用 TensorRT 优化 FlowNetS 并运行。

工作计划：

之后主要工作方向有两个：①为网络增加 GPU 代理节点，利用 GPU 加速 tflite 模型，或利用 TensorRT 优化模型。②重新训练网络进行知识蒸馏或利用预训练模型对网络进行裁剪。