RMUL 2022 山东站

神经网络优化与运动预测算法分享

主办：深圳市大疆创新科技有限公司
RoboMaster 组委会

哈尔滨工业大学（威海）HERO战队

2022.02.14

分享流程

MEETING AGENDA

- 神经网络优化
- 运动预测模型
- 自由交流与问卷反馈

视觉分享之
目标检测神经网络

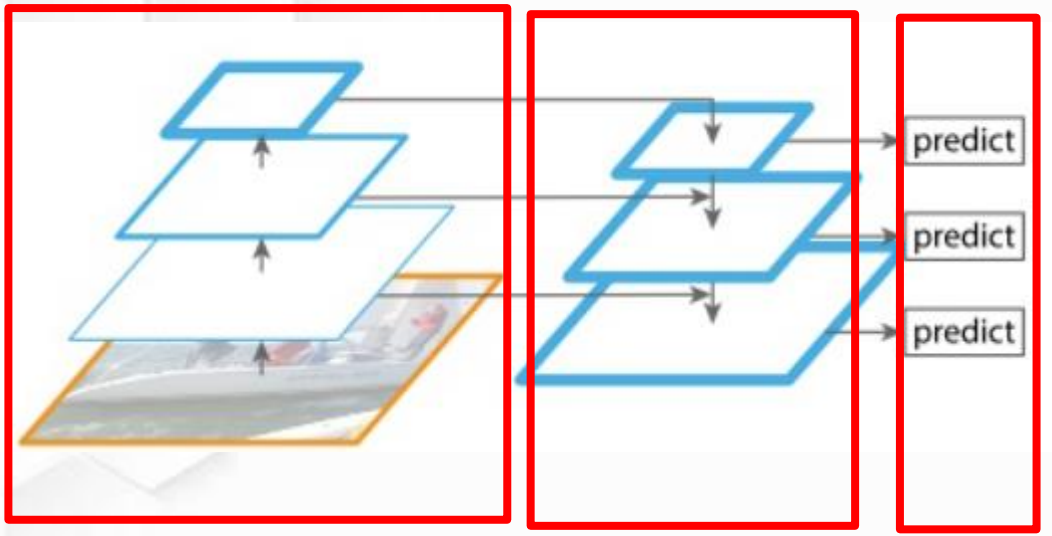
哈尔滨工业大学（威海）
HERO-鲁浩海

目录


01
PART 01
第一部分
神经网络的优势

目标检测网络基本结构

目标检测网络一般由下面三部分结构组成：



特征提取
backbone

特征融合
neck

预测输出
head

输出通常包括三部分：

- 预测出的目标位置
- 对于预测的正确性的置信度
- 对于预测出的目标归属类别

因此Loss函数也包含多个部分，一般总Loss由各部分Loss累加得到。

神经网络相对于传统视觉的优势：

- 不用临场调参，可以适应各个环境
- 鲁棒性高，精确率和召回率高（不容易误识别、漏识别）
- 可以识别一根灯条被遮挡、装甲板小半个在视野外等极端情况
- 可以识别灰色的装甲板

神经网络的“劣势”：

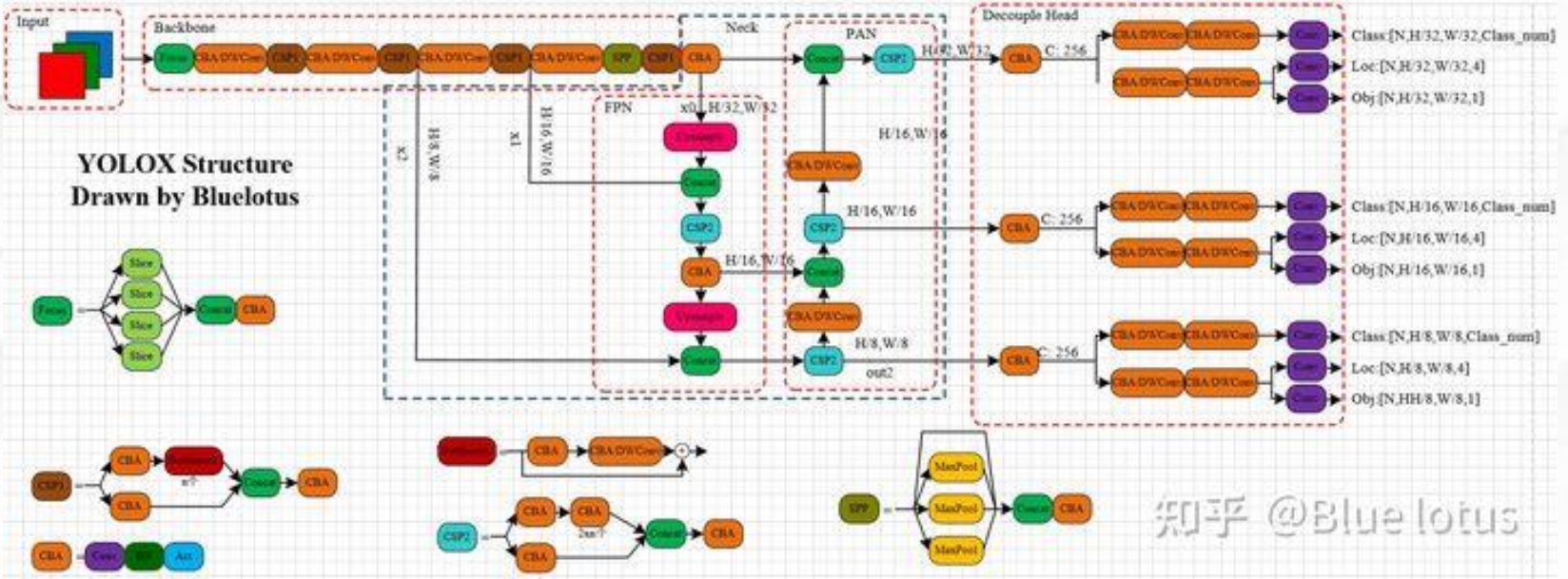
- 运行速度慢，对设备要求高?

现代小型目标检测神经网络模型大小最小可达980K(NanoDet, INT8),
在NUC11上使用iGPU轻松上100FPS


02
PART 02
第二部分
神经网络的优化与修正

YOLOX网络简介

我们的网络基于YOLOX基础改进，YOLOX的结构与特点如下：



YOLOX的新特性

- Decoupled Head
- Data Augmentation
- Anchor Free 与 Label Assignment

四点模型

常规目标检测的矩形框预测：

- x、y、w、h四个数据
- 准确度不够
- 不能表达倾斜情况、形变情况

四点模型：

- 预测矩形框的四个角点

改LOSS函数，改Head预测

优点

- 可以精准的表达矩形框的状态，包括旋转和形变

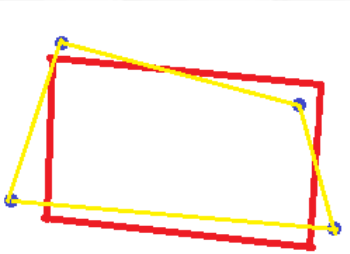
缺点

- 训练难度和模型设计难度直线上升，并且需要专门的标签

四点模型的LOSS

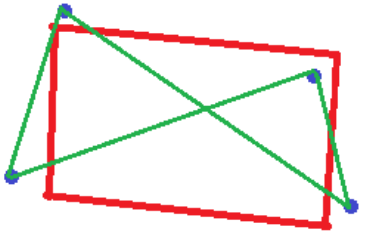
四点模型LOSS vs 普通矩形框LOSS

- 普通矩形框LOSS常见做法是使用真实标签和预测的矩形框的IOUloss作为定位回归



但是：

- 四点模型的IOU极难计算
- IOU大不代表预测的好，IOU小也不代表预测的不好



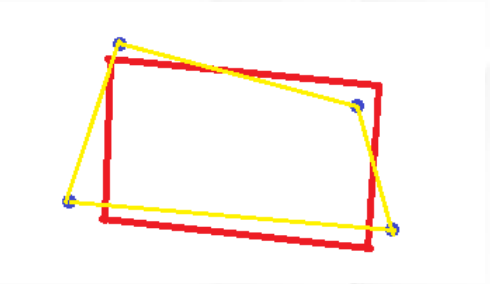
- 一般使用预测的四个点与标签四个点的距离绝对值之和作为损失函数（MSELoss）
- 因此，四点模型与目标检测相比更像一个回归问题。

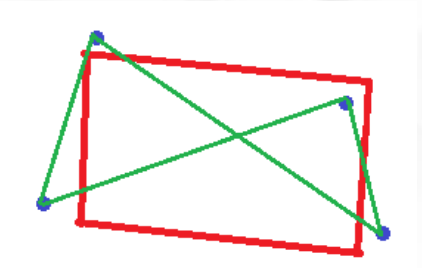
四点模型的IOU计算

然而，我们还是需要计算四点模型下的IOU

原因： **Label Assignment**

- Label Assignment的本质是为了更好的分配预测和标签，那么在分配过程中就需要拥有评估一个标签和一个预测是否匹配、匹配度有多少的一个衡量手段。一般来说，标签和这个预测所对应的感受野越近、预测的位置和标签越接近那么就越好。
- 因此在分配标签时，使用IOU作为分配依据无疑比使用MSE要合理的多。
- 但是还不够合理，所以在计算IOU时仍需要特殊处理。





能不能跳过Label Assignment呢?

Label Assignment意义重大

In NanoDet-Plus, we propose a novel label assignment strategy with a simple **assign guidance module (AGM)** and a **dynamic soft label assigner (DSLA)** to solve the optimal label assignment problem in lightweight model training. We also introduce a light feature pyramid called Ghost-PAN to enhance multi-layer feature fusion. These improvements boost previous NanoDet's detection accuracy by **7 mAP** on COCO dataset.

但是一个好的样本匹配算法可以天然缓解拥挤场景的检测问题（LLA、OTA 里使用动态样本匹配可以在 CrowdHuman 上提升 FCOS 将近 10 个点），缓解极端长宽比的物体的检测效果差的问题，以及极端大小目标正样本不均衡的问题。甚至可能可以缓解旋转物体检测效果不好的问题，这些问题本质上都是样本匹配的问题。在我们的认知中，样本匹配有 4 个因素十分重要：

1) **loss/quality/prediction aware：** 基于网络自身的预测来计算 anchor box 或者 anchor point 与 gt 的匹配关系，充分考虑到了不同结构/复杂度的模型可能会有不同行为，是一种真正的 dynamic 样本匹配。而 loss aware 后续也被发现对于 DeTR 和 DeFCN 这类端到端检测器至关重要。与之相对的，基于 IoU 阈值 /in Grid(YOLOv1)/in Box or Center(FCOS) 都属于依赖人为定义的几何先验做样本匹配，目前来看都属于次优方案。

2) **center prior :** 考虑到感受野的问题，以及大部分场景下，目标的质心都与目标的几何中心有一定的联系，将正样本限定在目标中心的一定区域内做 loss/quality aware 样本匹配能很好地解决收敛不稳定的问题。

3) **不同目标设定不同的正样本数量(dynamic k)**：我们不可能为同一场景下的西瓜和蚂蚁分配同样的正样本数，如果真是那样，那要么蚂蚁有很多低质量的正样本，要么西瓜仅仅只有一两个正样本。Dynamic k 的关键在于如何确定k，有些方法通过其他方式间接实现了动态 k，比如 ATSS、PAA，甚至 RetinaNet，同时，k的估计依然可以是 prediction aware 的，我们具体的做法是首先计算每个目标最接近的10个预测，然后把这个 10 个预测与 gt 的 iou 加起来求得最终的k，很简单有效，对 10 这个数字也不是很敏感，在 5~15 调整几乎没有影响。

4) **全局信息：**有些 anchor box/point 处于正样本之间的交界处、或者正负样本之间的交界处，这类 anchor box/point 的正负划分，甚至若为正，该是谁的正样本，都应充分考虑全局信息

四点模型的IOU计算

相比较常规矩形框预测，四点模型的IOU计算复杂度直线上升，需要考虑多种情况，这也是模型设计的难点。

首先，如何计算任意四边形面积？

简洁、优雅、高效

任意四边形面积选择向量叉乘法，利用pytorch提供的.det()函数实现批量计算

```
def _calc_area_fixnum(points, num=4):
    area = torch.zeros(*points.shape[:-2], dtype=torch.float, device=points.device)
    for i in range(num):
        area += torch.stack(
                (points[..., i, :],
                 points[..., i - 1, :]), dim=-2
                ).det()
    return area * 0.5
```

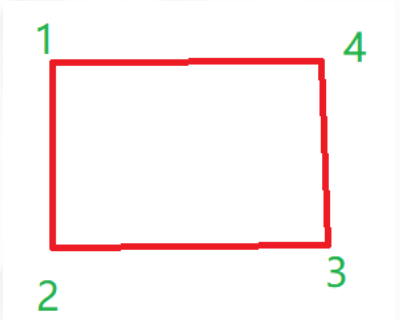
四点模型的IOU计算

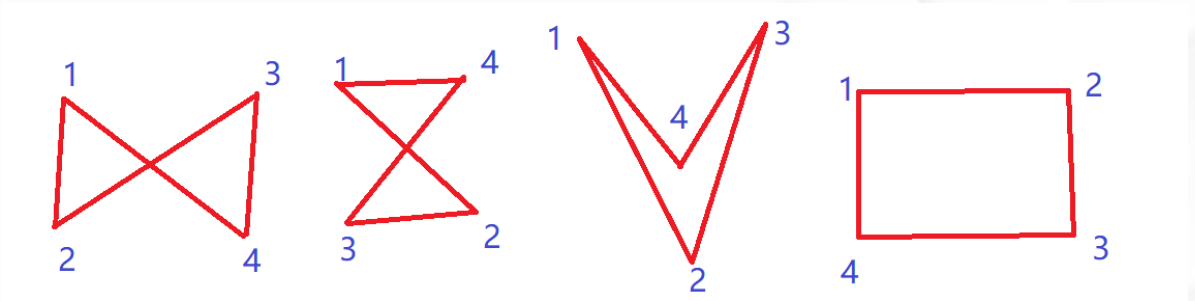
此外，如何计算两个任意四边形相交部分？

多边形裁剪Sutherland-Hodgeman算法（Github开源pytorch实现）

那我们能开始计算IOU了吗

由于预测出的矩形在网络初期形状具有极大的随机性，
因此计算IOU时需要考虑多种情况：





四点模型的IOU计算

假设给定标签都是逆时针顺序，并且不会出现凹四边形
预测出的结果可以是任意形状（凹四边形、角点交换）

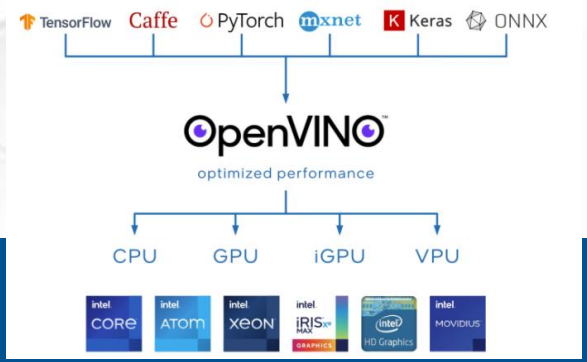
计算IOU的步骤

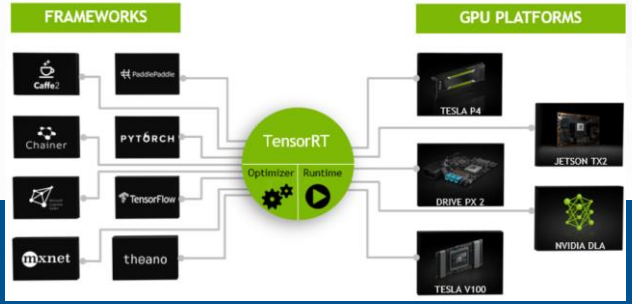
- 判断是否为凹四边形
- 判断角点顺序是否正确（是否对角线相交）并纠正
- 计算经过矫正的四边形IOU

```
mask = IOULoss._if_concave_quadrangle(pboxes)
pboxes[mask] = IOULoss._get_bounding_box(pboxes[mask])
mask = IOULoss._if_points_right_order(pboxes)
# ! if not in right order, then exchange the 'c', 'd'
pboxes[~mask] = pboxes[~mask][:, [0, 1, 3, 2]]
# ! if still not in right order
mask = IOULoss._if_points_right_order(pboxes)
# ! then continue exchange the 'b', 'd'
pboxes[~mask] = pboxes[~mask][:, [0, 2, 1, 3]]
clipped_res, res_points_num = IOULoss._clip(gboxes, pboxes)
area_i = torch.abs(IOULoss._calc_area(clipped_res, res_points_num))
return area_i / (torch.abs(IOULoss._calc_area_fixnum(pboxes)) +
                 IOULoss._calc_area_fixnum(gboxes) - area_i + 1e-16)
```


03
PART 03
第三部分
模型部署

模型部署工具





OpenVINO

Intel 闭源部署工具库，包括模型转化、量化优化、推理等多种工具。使用Open VINO在intel自家CPU、iGPU、VPU等设备上的推理速度无人能及

NCNN

ncnn 是一个为手机端极致优化的高性能神经网络前向计算框架。ncnn 从设计之初深刻考虑手机端的部署和使用。无第三方依赖，跨平台，手机端 cpu 的速度快于目前所有已知的开源框架。

TensorRT

TensorRT是一个高性能的深度学习推理（Inference）优化器，可以为深度学习应用提供低延迟、高吞吐率的部署推理。
使用TensorRT，在CPU或者GPU模式下其可提供10X乃至100X的加速。

OpenVINO使用步骤

导出模型

我们使用pytorch架构进行神经网络的训练。因此对于我们第一步是使用pytorch的API将网络导出成为ONNX结构的固定计算图模型。之后使用model optimizer进行模型优化并将模型文件转化成IR格式。

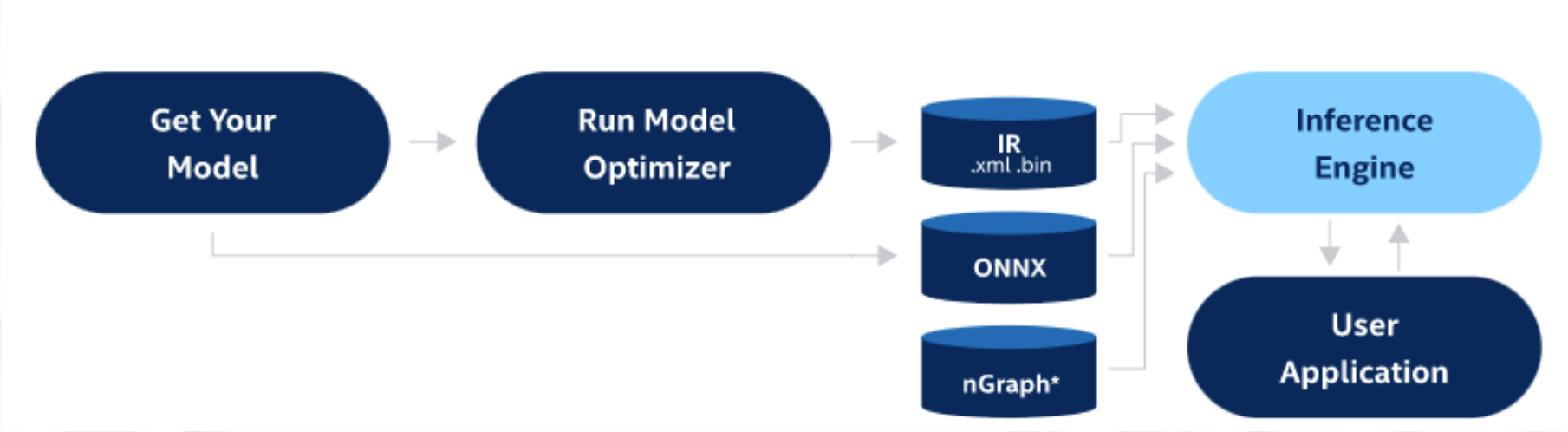
量化模型

对导出的模型进行后量化操作，可以使用POT工具（随openVINO一起发行），对导出后的IR格式文件进行INT 8 量化操作，其同时支持直接量化和损失感知量化两种方式。

Open VINO 调用

将量化完成后的IR文件在C++推理程序中进行调用，实现加速推理。

OpenVINO使用步骤



Model optimizer

Model Optimizer使用指南：

Model Optimizer本身是一个Python脚本，其内部完成了对于onnx模型导入优化步骤。但是Model Optimizer支持很多集成操作，将数据预处理整合到模型推理过程，进一步加速推理。

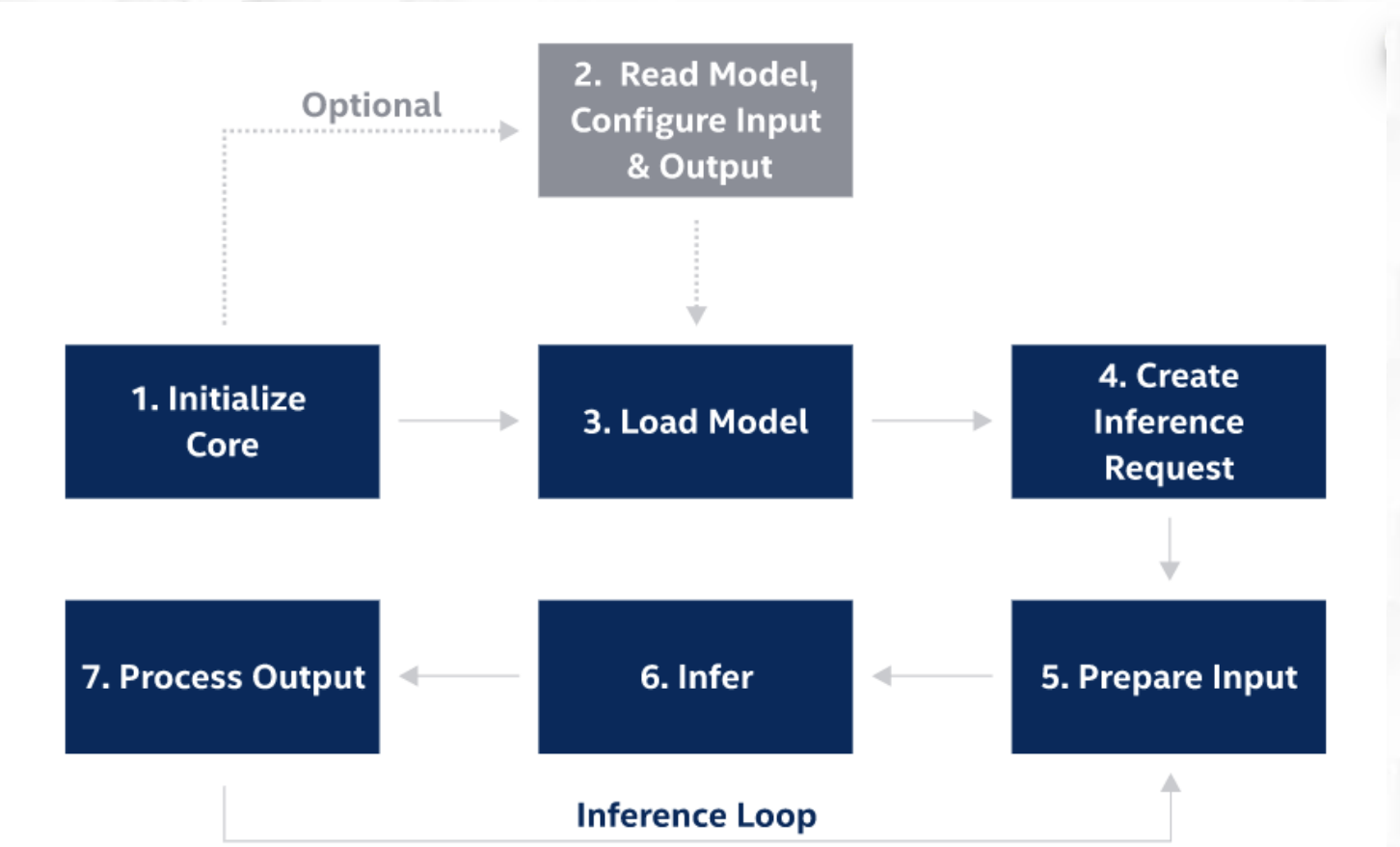
主要支持操作：

- 输入通道交换
- 输入数据归一化

对应参数：

- --reverse_input_channels
- --mean_values --scale_values

正确执行模型导出步骤后，我们就可以得到.bin、.xml、.map三个必要IR文件。在进行量化（也可以不量化）后，我们就可以开始推理。



视觉分享之
SINGER模型运动预测

哈尔滨工业大学（威海）
HERO-郭舒典

本赛季视觉面临的主要问题

- 大面积起伏路段所造成的场内对拼时车辆的抖动问题

英雄自瞄设计

主要考虑因素：

- 收敛速度
- 精度
- 稳定性

运动预测算法
弹道计算
决策系统

上个赛季 ： 传统的拟合算法

- 响应速度快
- 尽管有滤波器数据仍不够平滑，容易造成车辆抖动

↓

加入滤波器 ： 滞后

↓

本赛季 ： 机动目标追踪算法

Singer模型跟踪算法 ⟶

- 结合当前模型，对Singer模型进行修正
- 借鉴AEFK算法，在Singer模型中加入自适应项从而修正加速度协方差，增强适应性

目的：提高响应速度，减少车辆抖动

Singer模型的核心就是对加速度进行建模，认为加速度是指数自相关的零均值随机过程，同时Singer模型还根据其所建立的加速度模型，通过白噪声模型进行推导，建立了适用于卡尔曼滤波的状态方程。

即机动加速度自相关系数：

$$\mathrm{R}_{(r)} = E[a(t)a(\mathrm{t}+\tau)] = \delta_m^2 \times e^{-\alpha|\tau|}$$

白化后其状态方程如下：

$$\dot{X}(t) = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{a} \end{bmatrix} = AX(t) + \tilde{V}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -a \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \tilde{v} \end{bmatrix}$$

其中：$E[\tilde{v}(t)\tilde{v}(\tau)] = 2\alpha\sigma_m^2\delta(t-\tau)$

于是在相距时间间隔为τ时的离散状态下，有状态方程：

$$X(k+1)=F(k)X(k)+V(k)\quad ，其中：F=e^{AT}=\begin{bmatrix}1 & T & \frac{\alpha T-1+e^{-\alpha T}}{\alpha^2}\\ 0 & 1 & \frac{1-e^{-\alpha T}}{\alpha}\\ 0 & 0 & e^{-\alpha T}\end{bmatrix}$$

离散时间过程噪声协方差：

$$Q=2\alpha\sigma_m^2\begin{bmatrix}q_{11} & q_{12} & q_{13}\\ q_{21} & q_{22} & q_{23}\\ q_{31} & q_{32} & q_{33}\end{bmatrix}$$

$$\begin{cases}q_{11}=\frac{1}{2\alpha^5}[1-e^{-2\alpha\tau}+2\alpha T+\frac{2\alpha^3T^3}{3}-2\alpha^2T^2-4\alpha Te^{-\alpha T}]\\ q_{12}=q_{21}=\frac{1}{2\alpha^4}[1+e^{-2\alpha\tau}-2\alpha T-2e^{-\alpha T}+\alpha^2T^2+2\alpha Te^{-\alpha T}]\\ q_{31=}q_{13}=\frac{1}{2\alpha^3}[1-e^{-2\alpha\tau}-2\alpha Te^{-\alpha T}]\\ q_{22}=\frac{1}{2\alpha^3}[-3-e^{-2\alpha T}+2\alpha T+4e^{-\alpha T}]\\ q_{32}=q_{23}=\frac{1}{2\alpha^2}[1+e^{-2\alpha T}-2e^{-\alpha T}]\\ q_{33}=\frac{1}{2\alpha}[1-e^{-2\alpha T}]\end{cases}$$

当前模型

从本质上来讲，该算法是一个具有自适应非零加速度的Singer模型。与Singer模型算法中机动加速度均匀分布不同，该算法采用修正瑞利分布：分布随均值变化而变化，方差由均值决定。实现了闭环自适应追踪。

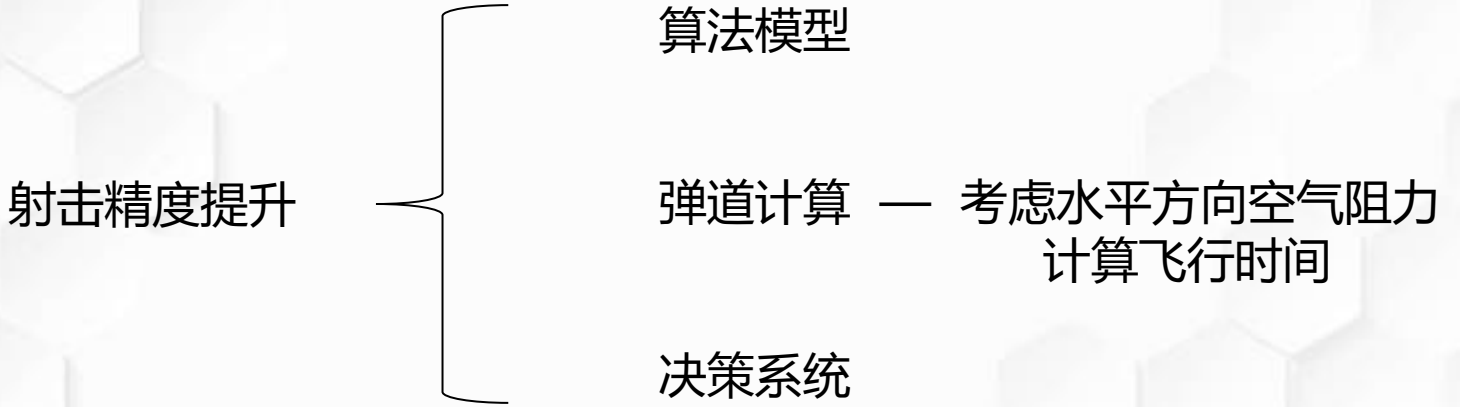
状态方程：

$$X(k+1) = F(k)X(k)+G(k)\bar{a} + V(k)$$

$$G(k) = \begin{bmatrix} \frac{1}{\alpha}(-T + \frac{\alpha T^2}{2} + \frac{1-e^{-\alpha T}}{\alpha}) \\ T - \frac{1-e^{-\alpha T}}{\alpha} \\ 1 - e^{-\alpha T} \end{bmatrix}$$

$$\sigma_a^2 = \frac{4-\pi}{\pi}[a_{max} - \bar{a}(k)]^2$$

$$\bar{a}(k) = \hat{\ddot{x}}(k|k-1)$$


射击精度提升
算法模型
弹道计算
考虑水平方向空气阻力
计算飞行时间
决策系统


否
开始
1. 时间戳
2. 连续数帧存在装甲板
3. 所识别位置在相关可信阈值范围内
是
（新目标）重置该滤波器
（旧目标）Singer滤波器
1. 电控实时性判断
2. 计算结果在可信阈值内
是
计算轨迹，发弹
否

感谢聆听

有奖问卷反馈↑

微信公众号↑

哈工大威海HERO战队

微博扫一扫，加入我们

微博官号

哈工大威海_HERO战队
UID: 208915710

B站官号

HERO YES!
群号：1130432908

扫一扫二维码，加入群聊。

QQ

QQ宣传群

哈工大威海HERO战队
QQ：481839650

点击卡片更换背景

QQ官号