Z-AI1.4 版检测器建模

文档版本:1.0

最后修改时间:2024-2-8

修改日志

目录

区分自定义建模与通用大模型 4

建模所需的通用条件 4

数据就是数据来源 4

GPU 5

建模工具 6

选择检测器样本 7

标注 11

标签 13

看懂标注数据信息 13

标注除错 14

尺度矫正 16

尺度分类标签 18

重叠标注 20

自动化标注 22

自动化标注的未来之路 23

列举 Z-AI 的检测器模型 23

列举 Image Matrix 自动标注工具 25

训练 26

OD6L/OD3L 27

训练 OD6L/OD3L 时在多路服务器把 CPU 性能挖掘出来 28

DNN-OD6L/DNN-OD3L(MMOD) 31

理解 DNN-OD 的推理流程 32

Test 标签的的作用 33

DNN-OD 的训练参数 34

YOLO-X 35

对 YOLO-X 建模实操的建议 38

关于 YOLO-X 对 Test 标签的支持说明 38

YOLO 体系=监控盒子技术祖坟 38

.AI_Set/.imgDataset(样本集) 39

.ImgMat, (图矩, ImageMatrix) 39
.OX(训练包) 39
命令行训练 40
TrainingTool 训练参数 40
训练小工具 41
服务器训练 42
测试模型 43
将 AI 模型合并到业务系统 44
需要具有可视化能力 44
需要具有 gpu 的算力调度能力 44
业务输出能力 44
从视频导入图片来建模 45
从视频大规模提取图片 45
在 Model Builder 导入视频图片 47

区分自定义建模与通用大模型

通用大模型,例如多态模型,大语言模型,SAM 模型,这些模型都是基于整个互联网的公共数据进行建模.这类建模工作第一步会是一个入口型的模型,入口的工作是给建筑 打地基,这类东西先大都在已经分类的数据基础上把数据全部规范化,例如小猫,小狗,汽车,自行车...这些来自整个互联网的数据整个都是一种建筑上的深入地基,然后再此基础一层一层的往上修建,这些修建工作是走的数据科学路线:用程序来代替人工的数据管理,从而达到数据挖掘目的.建筑修到 10 层,数据干净度会达到一个可以训练的水平,这会诞生出许多通用检测器,分类器,这类通用模型.这时候楼上建楼就变成可以实现的东西了:11 层可以是 gemini,可以是 chatgpt,也可以是 llama 和 SAM.而 10 层以下是全人类慢慢积累的成果,这一步是美国人在主导,从大数据到慢慢整理清晰的大数据再到通用模型.整个大模型都是这种发模式.

多态大模型是文字+语音+视觉的结合体,大模型的本质不会变:全人类的建筑工程.

我个人给大模型的稳定程度做过一次数字化假设,在未来 2 年中 11 层如果有 50%概率被推翻替代,那么 10 层以下就会有至少 20%概率被推翻替代.至于怎么推翻,我无法预测,未来的大模型会更加完善+可用性会更高,所有的建筑都是随着时代进步而推翻重构,最后变得更加适应人类居住.每一个时代都会自己的现代化,而现代化的定义总是在变化,同时每一次变化都会导致整个宏观经济学的重构.好比某些经济学家在 2010 出版的宏观经济学如果放在今天会完全对不上号.

自定义建模与大模型不同,大模型的数据都来自互联网,这些数据都是公有化的,自定义模型是私有化的.当公有模型做不到某些事情时,就需要考虑自定义模型路线.CV 领域的大模型是 SAM 模型,这种模型可以识别出狗,但无法识别出自养的狗.在另一方面,许多国家机关单位的监控画面,也是大模型无法识别的内容,例如许多军事化图片,大模型会包含所有在民间传播的照片和文字信息,只要数据在互联网的公共区域中,那么大模型都能涉猎到.

建模所需的通用条件

建模首选需要有数据,接下来是 GPU,最后是使用建模工具+花费时间标注和训练.

数据就是数据来源

持续不断的数据来源,会远远优于数据.数据会有一种特点,就是数据的形态,例如 10000 张二哈在狗窝的照片与 10 张并没有什么差别,我把这种形态定义成相同形态,如果老王和老李都养了二哈并且也提供了二哈的狗窝,那么老王和老李的二哈照片就是相似形态.对于 CV 的数据样本来说相似形态才是有用的,相同形态会多而无用.

数据来源优先取相似形态,如果数据来源都在街道的监控中,10 条街道各采集一张图会远远优于一个街道的采集 100 张图,哪怕是分时采集也是不如 10 条街道的相似形态的.

总结一下数据来源:持续不断并且具有相似形态的数据是最好的样本,其次是具有相似形态的样本,最糟糕的样本是大量相同形态的样本.

GPU

GPU 对整个人类来说是生产和使用 AI 的标准设备,先入为主,先下手为强,占山为王,这些都是刻画在人类基因中的东西.反人性是指善变,喜新厌旧,真实的喜新厌旧是因为旧事物反人性了,违背了基因的运作规律,因此不爱旧人,新事物取代了旧事物.

GPU 是由 NV 发起的异构计算芯片,从首发到今天,大约 30 年,从当初无人问津靠炒热点游戏经营再到天下无敌,这是因为 NV 做的事情很符合人性以及世界运行的规律.

GPU 编程非常讲究直观和便捷,同时它也非常讲究精度,例如训练中 loss 如果给出 0.001,这是个千分位的值,loss 的内部计算流程会精确到 1E-13 到 1E-18 左右,对于训练来说哪怕是一个极小误差也会导致 loss 计算出现问题.通常来说,建模都用专业 GPU,因为建模流程非常大,这对移植来说是极为困难,例如 NV 的 GPU 和国产车载用的 RK3588 运行同一个 GPU 计算程序结果不一样,这时候找精度问题会非常困难,硬件并行程序是个很大的技术栈,里面会有各种计算核和流水线,而模型运行,则可以跑在 CPU,APU,AIOT 这类设备,因为运行模型相对建模很小,更容易移植和优化,大不了重新写一个目标架构,其实很多目标架构都是重写.完整的 AI 方案,几乎都会提供多种目标架构来运行 GPU 程序.但就 GPU 的使用,主要是习惯,从这一层来看,NV 不可替代.当今能生产 GPU 的公司可以多达上百家,国内知名的就有寒武纪,景嘉微…但他们无法取代解决 GPU 科学计算那些人的使用习惯,他们只接口一个 py,然后开始宣传:我们可以对标 nv 的旗舰,py 这种东西都依赖于 c++做实现,高级接口无法满足 gpu 编程.他们的真正对手并不是 nv,而是人性的先入为主,他们无法改变人类使用 gpu 编程习惯.

Intel 已经开启了 cuda 傻瓜移植流程,我实测结果是,NV 的 gpu 程序可以直接移植到 intel 的 gpu,并且,不会丢失精度,这是一个叫 Advisor 的 sdk 开发包.同时我也给出 intel 体系的建模方案.我认为 intel 兼容 cuda 这种非常好,intel 并没有刻意要求我重写 gpu 流程,而是让我直接用他们的工具把程序转成 intel 专用.

作为一个 gpu 程序,我只想告诉大家,GPU 也需要 debug,也需要查阅大量 api 来下手,移植工作会有很多时间拿来写移植 api 的封装,相当于把 nv 提供的 api,用别的 gpu 来重写一个,如果一家厂商,不提供便捷 debug 方案,只让你通过 log,或则让你自己写算子,并且我一个开发者还需要花 2-4 倍于 nv 的价格买到他们的设备来编程,例如华为,我真的做不了他们家的 gpu 程序移植.或则更直接一点,我认为,log 和自写算子,是一种心理变态,违背人性.

说个题外话,我们生活环境中有太多的信息,大量的虚浮和营销信息,我在生活中,已经养成了只相信和接受所见+所经历的东西,未来什么样无法知道,今天 NV 就是全人类的 GPU 使用标准,人性决定了,未来的 GPU 发展不是硬件,而是兼容 NV 的编程方式.拥有支持者的一方,被编程者认可的 gpu,会是未来的取代 nv 的公司.现在 nv 是无敌的,不可超越.

回到主题上,建模所使用的 GPU 可以游戏 GPU,也可以是 A100,V100 这种服务器.

在我看来,今天买 gpu 和 20 年前买外资车,40 年前买日产彩电,没有区别,也不存在亏和赚,这是一个时代的消费.跟上时代就会消费.只要不是做 gpu 的倒货生意,符合区间价都能入,无需计较太多.

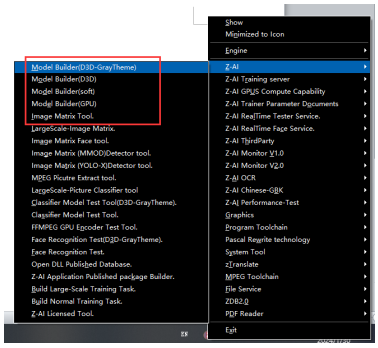
建模工具

建模工具可以从菜单去理解 Z-AI 体系的设计思路:建模以及一切与数据打交道都是一种生产,解决生产问题就是走程序化,把程序化体现在计算机层面就是做工具.当工具做到一定规模,也代表经历了许多生产层面的洗礼.

生产工艺是在生产过程中,用程序方式控制数据的流向,一旦做到了生产流水线,就可以计划生产了.由于每个项目环境差异,某些项目有许多人,某些项目是单兵作战,因此工具组合拳纷繁复杂.

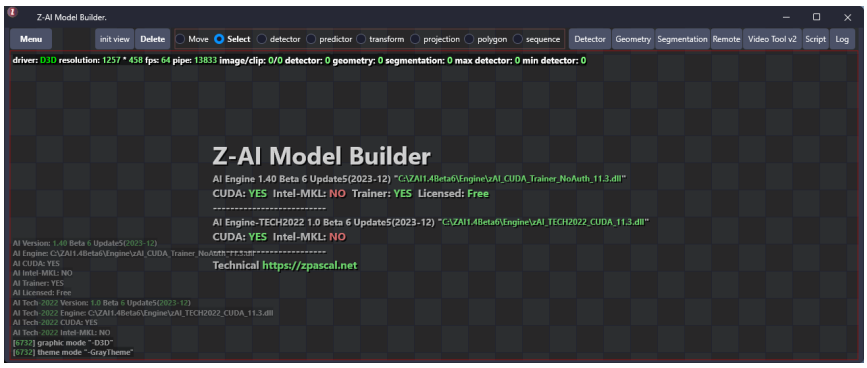
设计工具的初衷是跟随生产,在生产中,凡是耗时的工作,而程序可以减少耗时,这时候就可以给程序化工具,至于工具要做成什么样子,其实这不重要,主要是减少体力劳动.另一方面,如果程序无法减少耗时的工作,就需要让人感到不那么累,如果做到不反人性,也能有效生产,其次是排错,在数据化的生产工艺中会有出错的情况,如果程序化可以减少出错这会提升生产效率.上述都是 Z-AI 在生产工艺环节的设计思路.

红框是标注工具,标注主要是对已有数据做二次生产,形成可以被程序训练的样本数据.



Z-AI 标注工具核心就是 Model Builder,下图为它的启动界面

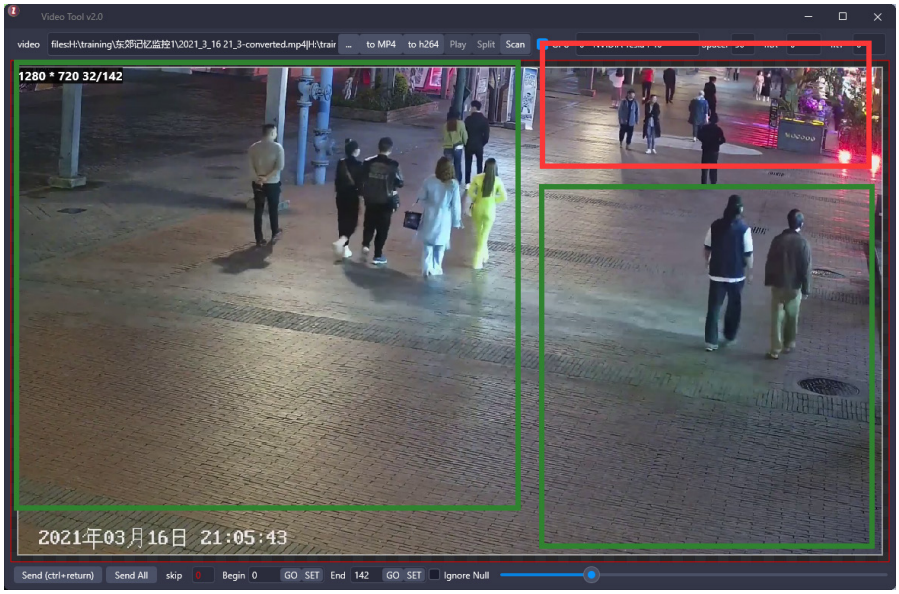
注意:在启动界面的版本号是不可参考的,也许编写文档用一个版本,而发行时又使用另一个版本



选择检测器样本

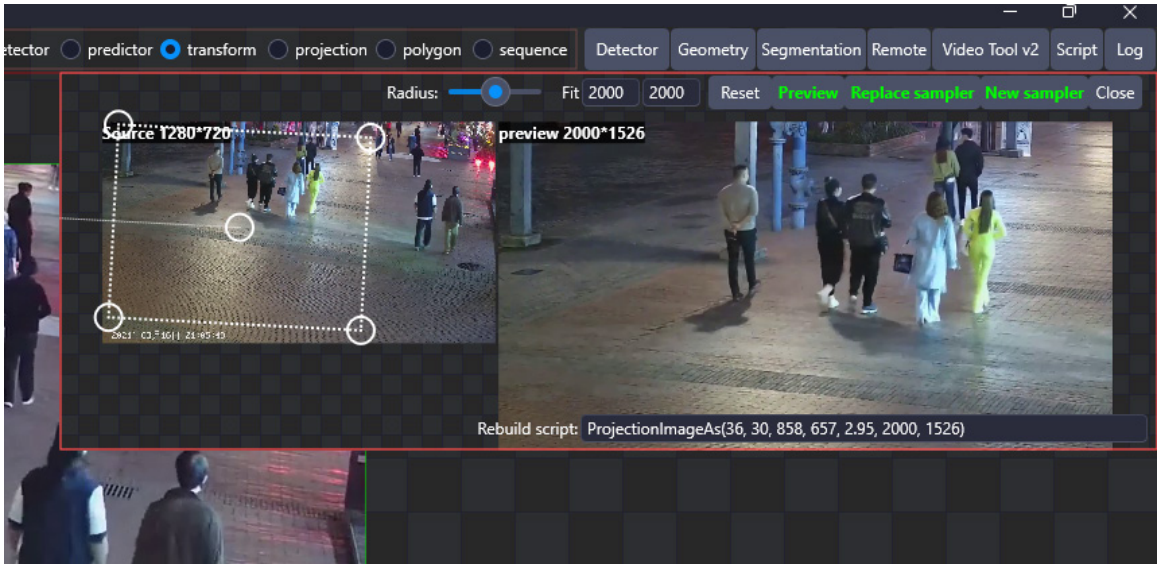
这里以一种挑战性非常高的行人模型来说明检测器样本

下列图片的分辨率是标准 720p,注意下面的红色+绿色区域

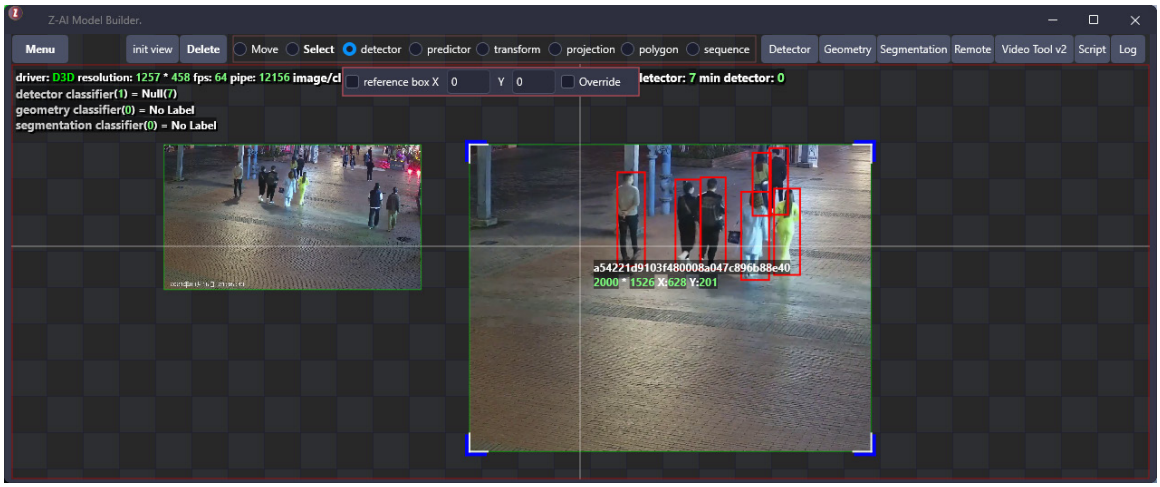


把红色区域在放大以后,箭头所指,要么是半身,要么是因为图片分辨率太低导致模糊,这些半身和模糊目标,对检测器算法来说会是一种模糊数据

而绿色区域在经过了 PS 或建模工具的处理以后可以得到相对更清晰的效果,新样本拥有周围环境,并且标也很清晰.同时新样本有一定的倾斜拨正效果,因为行人宽高比例大都为 1:3,这样更容易标注,倾斜的人会让框框很扁.



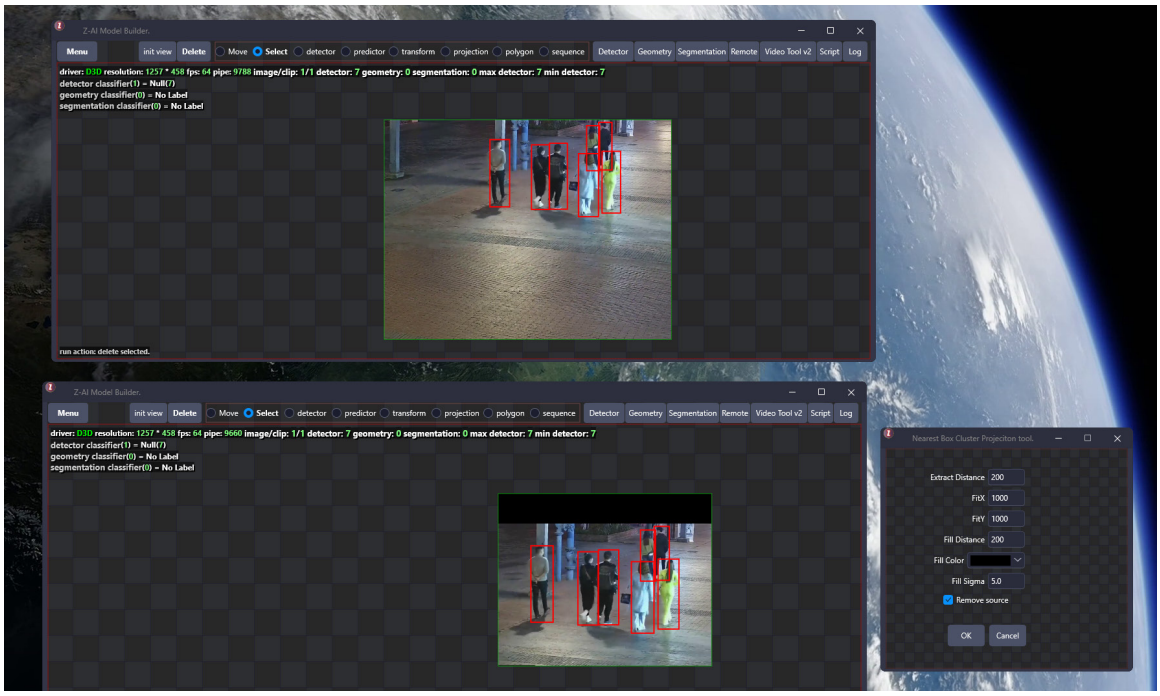
这时候,左边是原始的样本,右边是经过了处理并且直接标注新样本.



说一下重点:选择样本是根据建模目标决定的,如果建模目标是行人,当图片中的行人很模糊,或则行人数量很多,这时候要格外小心,模糊的行人会导致计算不准确,而数量很多的行人会导致后面标注非常累人,一张高清图上百人这类标注会非常痛苦,而远处的人往往十分模糊,这需要高清素材.

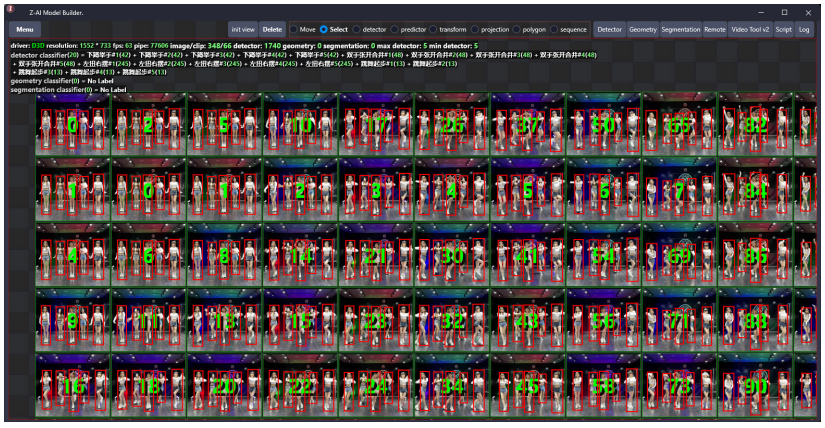
在 Z-AI 中会有许多先进算法用于优化样本,如下图

下图是临近算法根据标注的状态重新生成一张新样本,这时候上面的样本有更多的环境信息,下面的样本环境信息更少.但这对选择样本有什么影响呢?

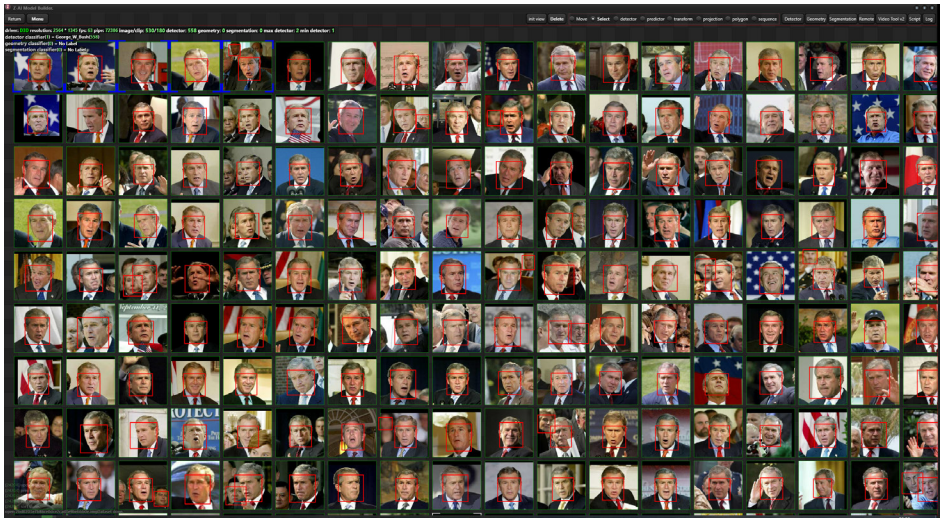


这里就必须说一下历史遗留问题,在早期机器视觉的发展中,检测器建模纯粹而直接,大都是滑动窗技术来匹配框框,建模的思路是直接把框框记录下来,然后走暴力的匹配.这种做法放在今天仍然有很多在项目使用.但今天是深度学习的时代,深度学习会需要框框以及框框周围的环境,简言来说,周围的环境越具有差异性,出来模型就会越具有通用性.环境对于高级模型来说是做通用性的关键,例如 YOLO,DNN-OD.

下图是在一个室内,对 5 个跳舞的女孩足帧进行标注,这些样本出来的模型因为缺少环境差异性,所以只能作为 demo 模型,不能作为通用模型.当然,下图这些数据都是用工具来生成的标注数据.



下图是一个小布什的通用检测器,每张样本目标周围环境差异,这种样本才是可以做成通用模型的样本. 当然,下图这些数据都是用工具来生成的标注.



一个重要问题,如果只标注一部分是不是可以建模检测器

下面这张图如果只标注一部分,检测器在训练时,会把没有标注的人当成是环境,从而导致计算出失误,最后训练出来的检测器也许将无法识别任何目标.**记住重点:环境会告诉检测器这里的不是你学习的目标,你给我记住,而框框则会告诉检测器这里是你的目标,你给我记住**.

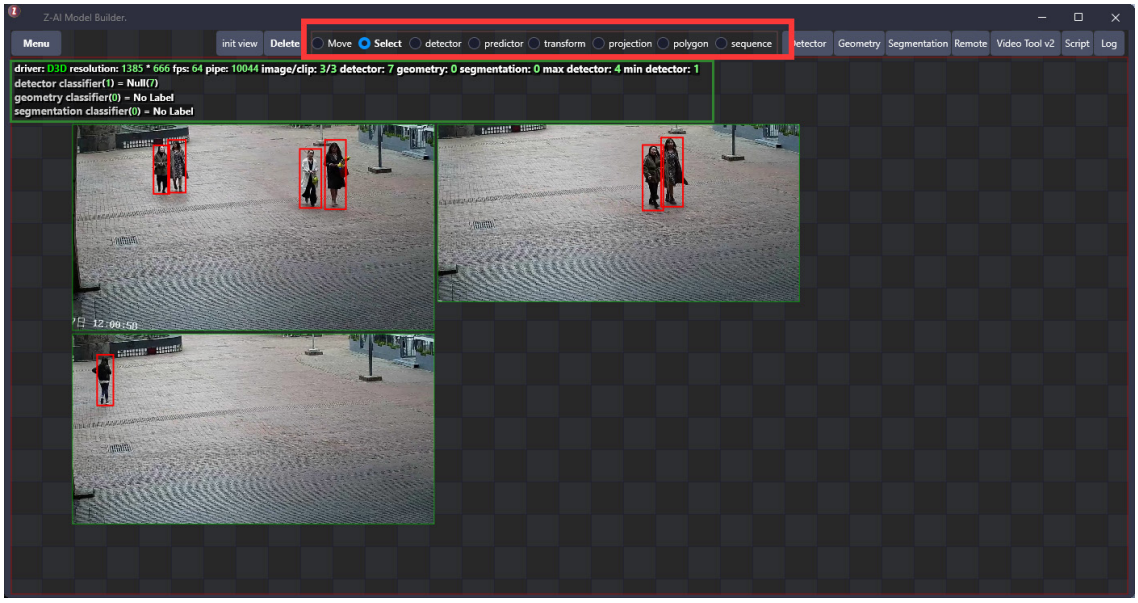
总结一下本小节:对于检测器样本来说,如果要使用 Z-AI 体系深度学习模型,例如 YOLO,那么在选择样本时,应该优先考虑环境迥异的样本,其次是,样本需要易于标注,同时也需要避免模糊不清的目标和环境.

本小结每一个环节都是重点,一定要反复阅读咀嚼,加深理解,做好样本选择是完成后面工作的先决条件.本小节非常非常重要!

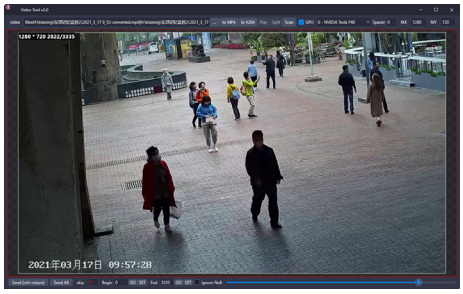
标注

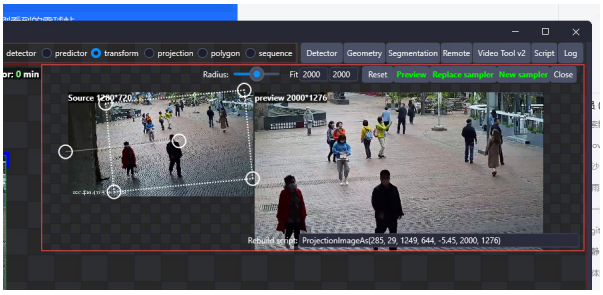
标注工具红框是操作切换,绿框是全局数据状态
检测器标注主要使用

- **Move:拖拉修改操作,可以修改标注框**
- **Select:自动化选择器,可以选择标注框**
- **Detector:检测器的框框标注模式**
- **Transform:图片投影变化器**



上面的样本都来自视频,使用视频工具慢慢找最佳样本,然后导进来,再用 Transform 工具做矫正+裁剪

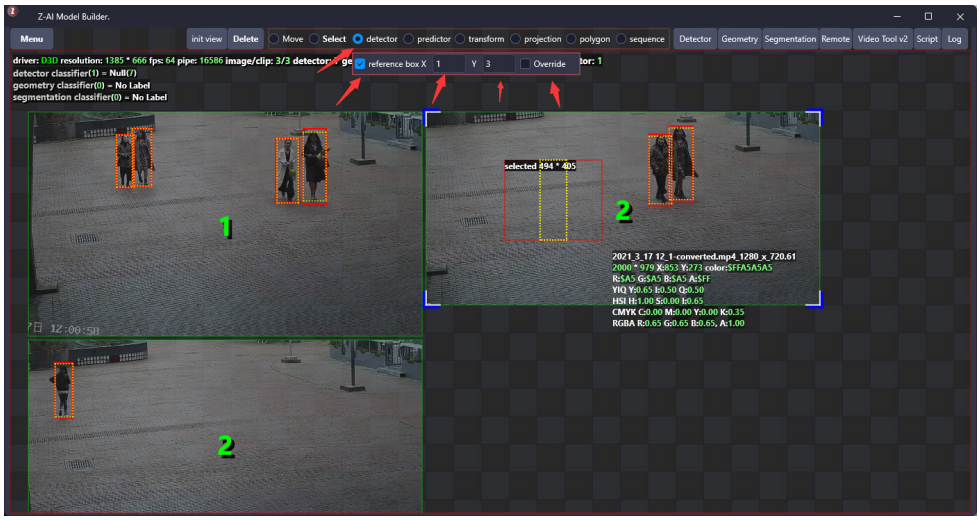




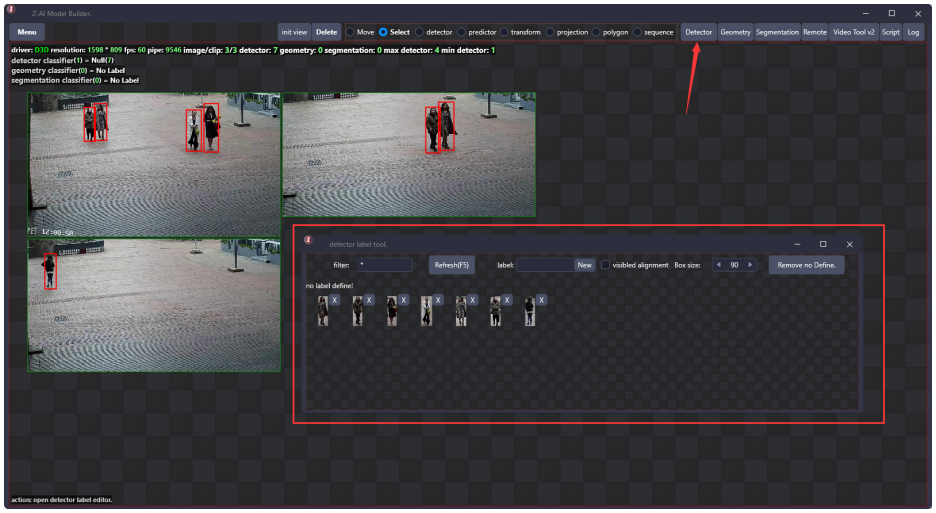
切换到检测器标注时下方会出现尺度工具:用于实时标注矫正,解决某些检测器模型对于 box 尺度的处理机制问题,例如 Z-AI 中的 DNN-OD 模型会使用 SVM 预先给尺度做分类,loss 计算时会发生不同尺度互相不兼容,例如 loss 将方框和长方会被视作两种检测器来计算,这将导致样本损失,例如原本 100 张样本,因为尺度差异,DNN-OD 被分割成 20 方框+80 长方框的样本组合,最后会导致检测器精确度下降.除 DNN-OD 之外,SVM-OD 也是走的类似机制.DNN-OD 与 SVM-OD 也是当今最主流人脸检测器模型,无论那家的人脸产品,人脸检测器环节都是这两个体系的变种衍生.

Z-AI 中的 YOLO-X 模型不受尺度影响,就算有 10 种尺度,YOLO-X 也可以完全兼容

YOLO 体系的底层网络走的是一种记忆体系+候选 Box 筛选路线, 样本中的许多数据会存放于一种记忆树,而 DNN-OD+SVM-OD 走的是雕刻路线,通过大量样本训练,会把相似性目标雕刻越加接近.简而言之,检测器使用非 YOLO 体系需要打开尺度参照工具,使用 YOLO 体系直接拉框就行.不需要管尺度问题.

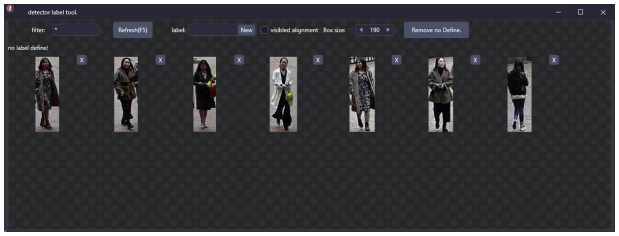


Detector 按钮把已经标注的检测器以列表方式呈现出来



标签

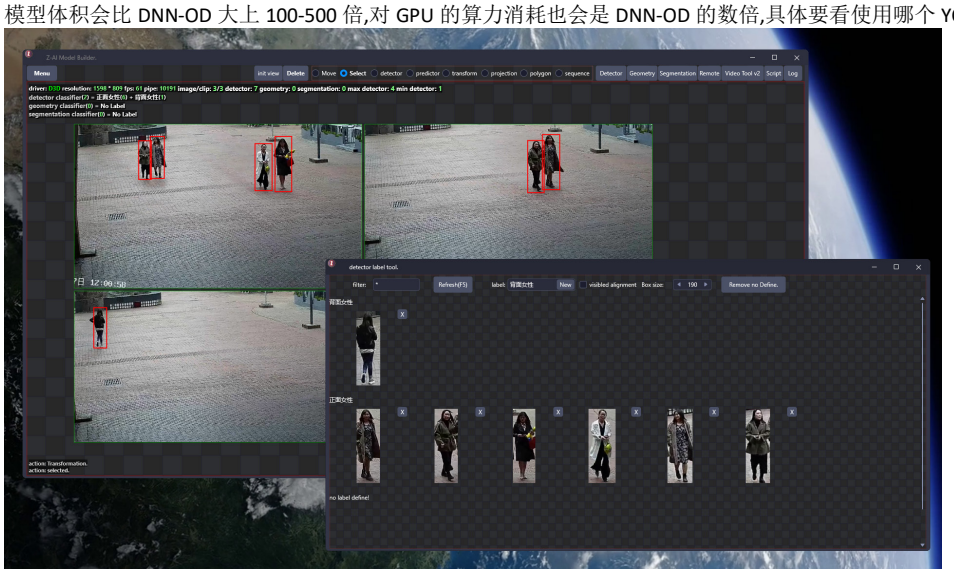
在检测器列表工具可以编辑标签



这里给了 2 个女性标签,正面+背面

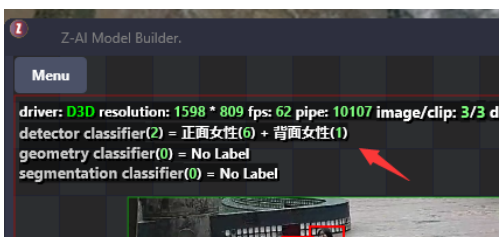
只要数据中有标签,那么检测器模型就会返回标签,DNN-OD 检测器会根据不同标签产生不同检测器,例如正面+背面会各是一套检测器,看起来这会是一个检测器,只要标签有多个那么检测器就会有多个,因为 DNN-OD 很偏雕刻,数据越多雕刻越精细.YOLO 体系检测器的标签是合并计算,理论上如果有多个标签对于 YOLO 来说是多种数据形态,而不是两种检测器,YOLO 体系更偏记忆.

歧义纠正:YOLO 体系从技术来说,它对检测目标的数据结构是以一种张量树的形式存在,这棵树是专门按 8*8,32*32 这种小块来存,具体模型根据不同的 YOLO 体系分支而定,通过不断的输入训练,树中张量会随时更新,时而高误差,时而低误差,这就是 loss,因为 8*8,32*32 这种小块数据的密集性超高,所以本文让大家用 YOLO 更偏记忆的方式来学习模型去理解 YOLO:相比 DNN-OD 来说 YOLO 的数据量更大,所以就理解成 YOLO 是在记忆中学习,在实际项目中 YOLO 的表现会更加聪明,与此同时 YOLO 的模型体积会比 DNN-OD 大上 100-500 倍,对 GPU 的算力消耗也会是 DNN-OD 的数倍,具体要看使用哪个 YOLO 分支.



看懂标注数据信息

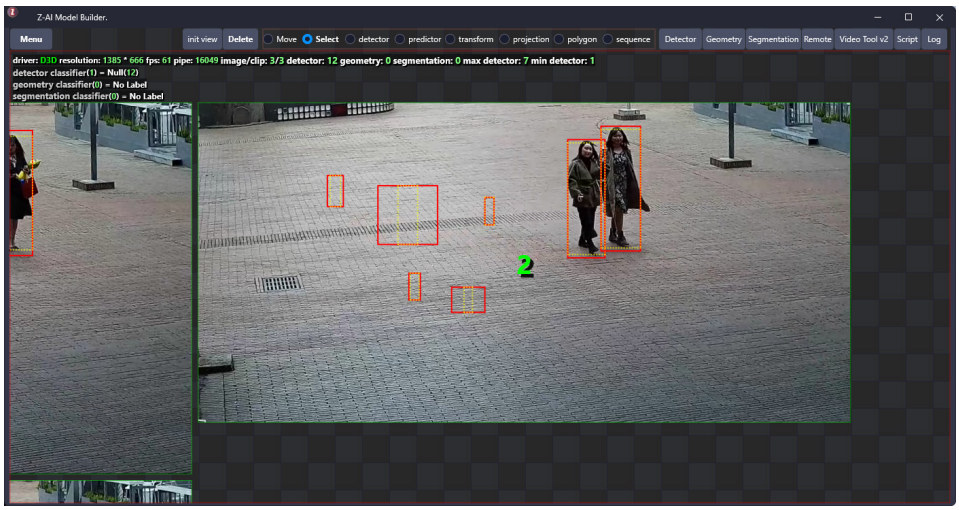
这里会出现摘要信息,如果放大一点,也会从 box 框框看到标签信息



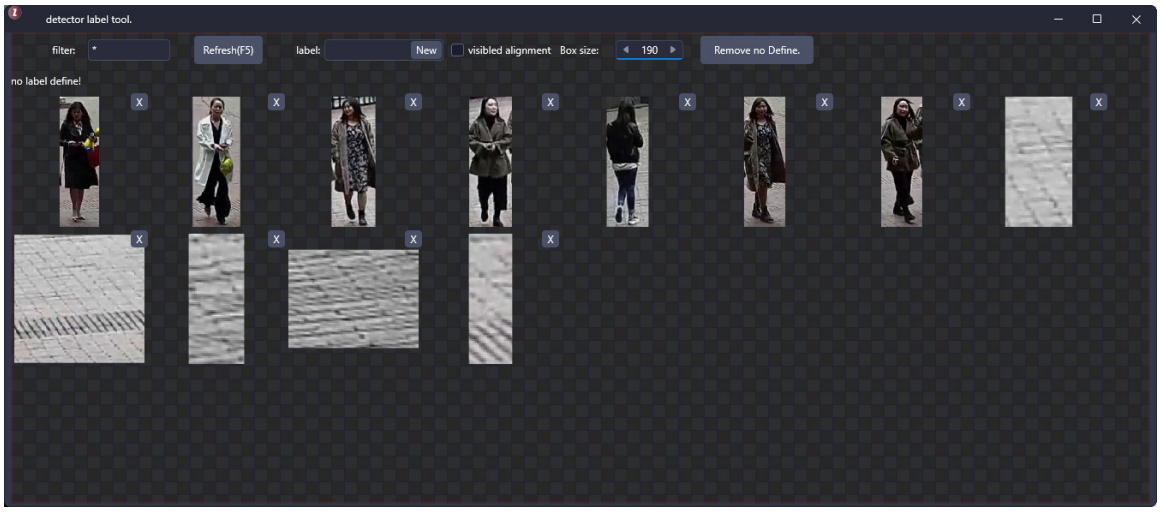
第一行,正面女性:119*326,这一张是标签数据,看检测器标签,主要看第一行
第二行,Sequence:正面女性,这一行是序列化数据,该数据属于序列型动作标签
第三行,Index:-1,这一行也是序列化数据,同样作用域序列型动作标签
在检测器列表工具修改的所有标签,都会默认一并修改序列行为数据,本文档以讲解检测器建模 为主,对于肢体和行为化建模会放到专门的文档中.

标注除错

标注会是一件繁琐而易错的工作,一不小心手贱点了一下,这时就会产生一个标注框

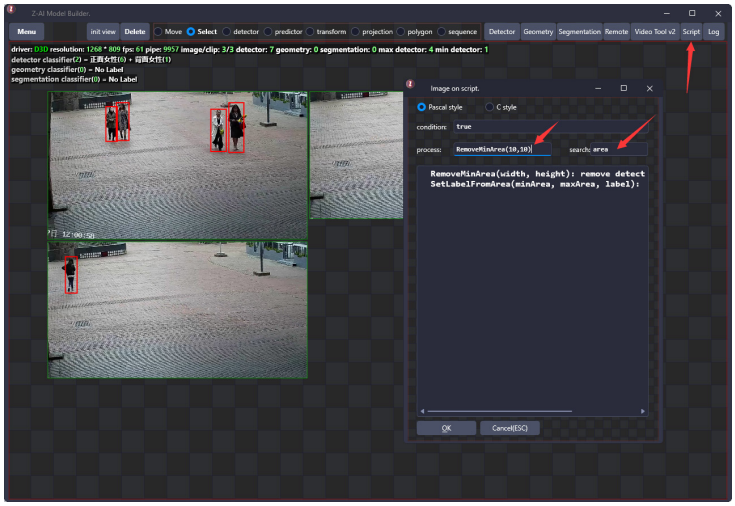


解决标错的日常方法是:点开 检测器列表工具 然后点 X 移除



当标注数量太多,这时候,从列表工具会很难找到错误
当数据数以千计时找错将会非常麻烦,在列表工具只能靠眼力找错

有一种错误标注是不小心点了一下鼠标,这时候,框框尺度大都是 1*1,2*2 这种极小的尺度,可以点开脚本工具,在 Search 栏输 area(面积),然后使用 RemoveMinArea(10,10)这个函数移除面积低于 10*10 的全部框框

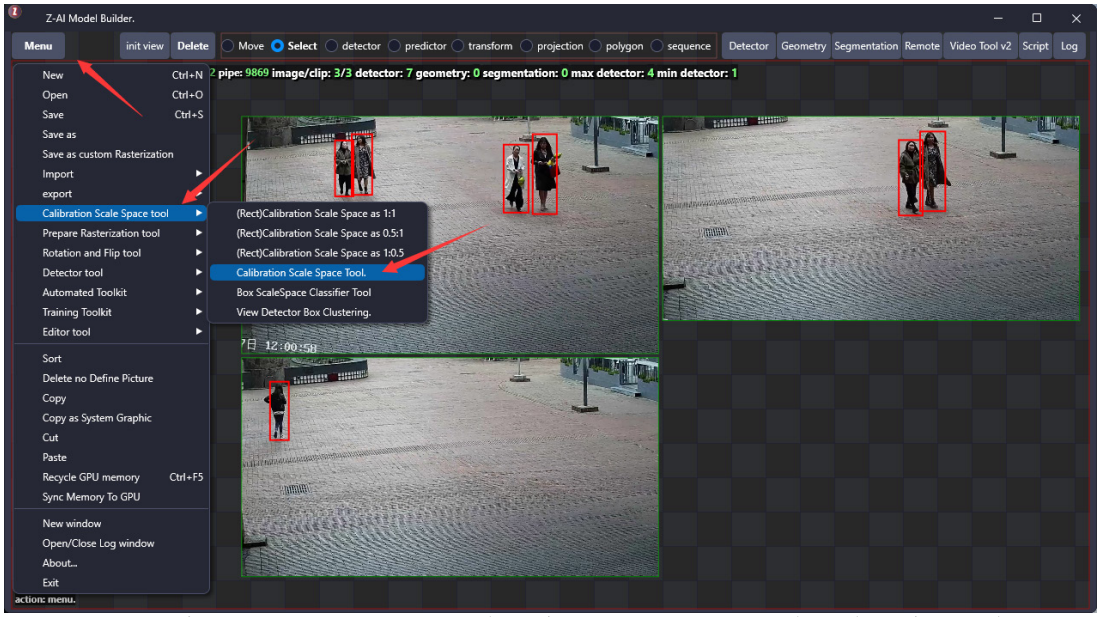


尺度矫正

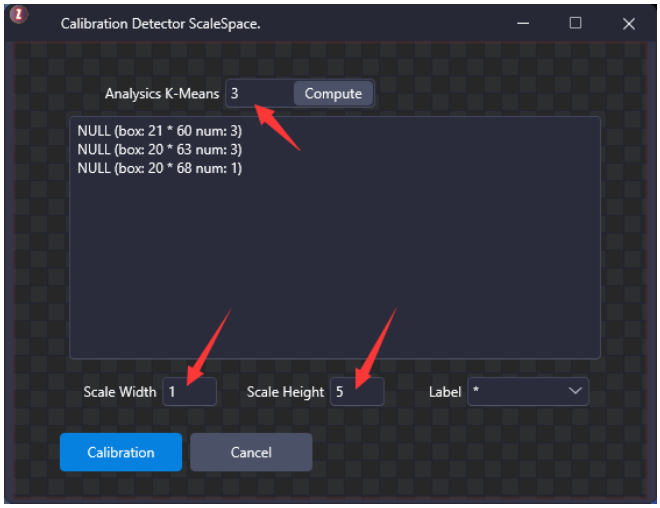
讲尺度之前需要先说概念,在数据层面 box size 是指框框的宽高数字,尺度是宽高比例,例如框框的宽高 100*300,那么尺度可以是 0.1:0.3,也可以是 1:3

在没有使用下列尺度参照工具时,标注的数据都会有尺度误差,尺度矫正工具是解决尺度误差的一种方案,如果使用 YOLO 检测器可以直接跳过本小节,而 DNN-OD,SVM-OD 这类检测器会对尺度非常敏感,尺度误差稍微一大就会影响训练的 Loss 精度.

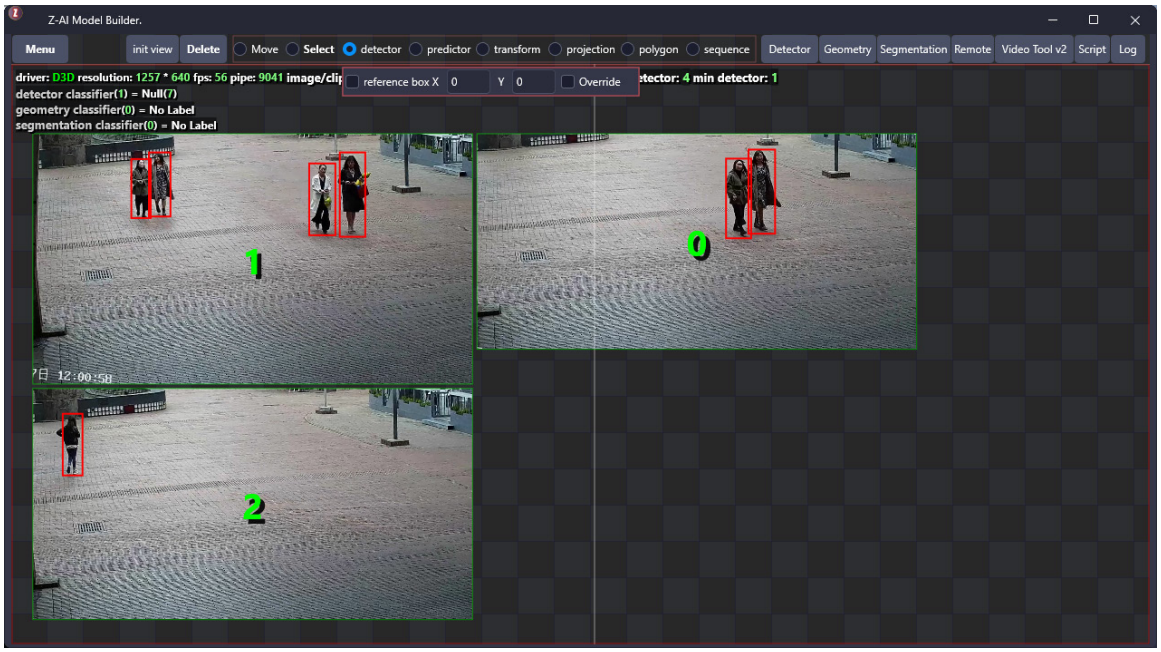
点开菜单打开尺度矫正工具



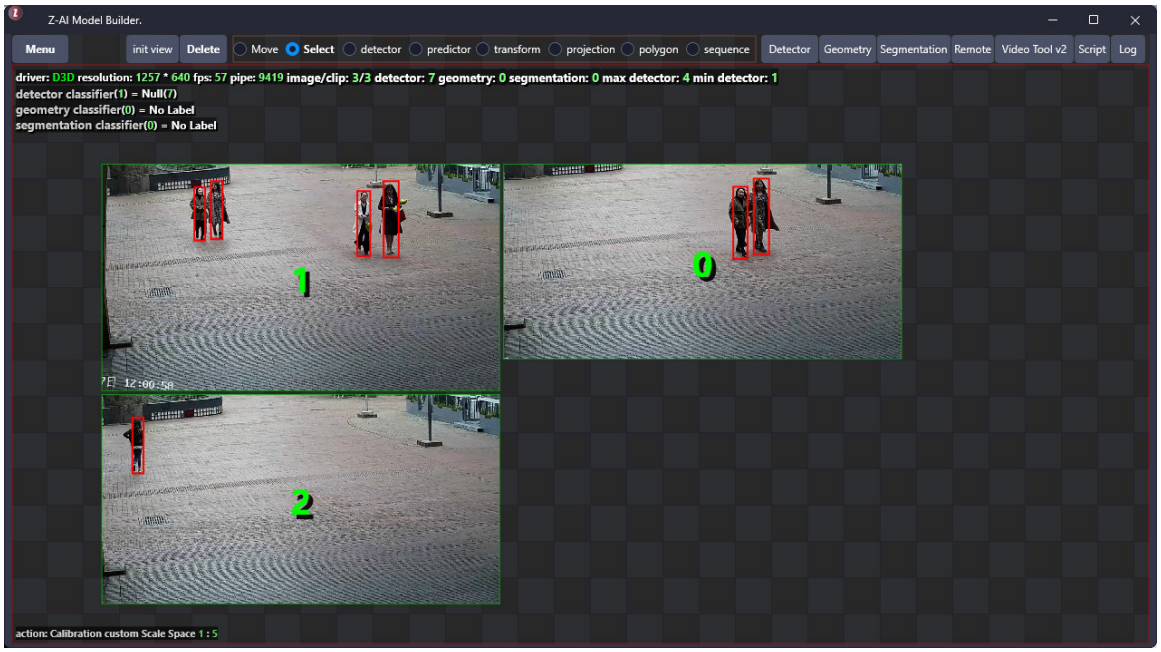
K-Means 是尺度分类器,这里分成了 3 个尺度类,我们可以看到,有 6 个尺度几乎都是 1:3 这时候,把尺度全部矫正成 1:5



尺度矫正前



尺度矫正后,这时候整个数据集的框框尺度都变成了 1:5



尺度矫正的准则需要是以小向大,1:3 -> 1:4,不可以 1:4 -> 1:3,尺度矫正算法会走择优方式,在实际工作中,以小向大矫正是走样最小的一种方式.

尺度分类标签

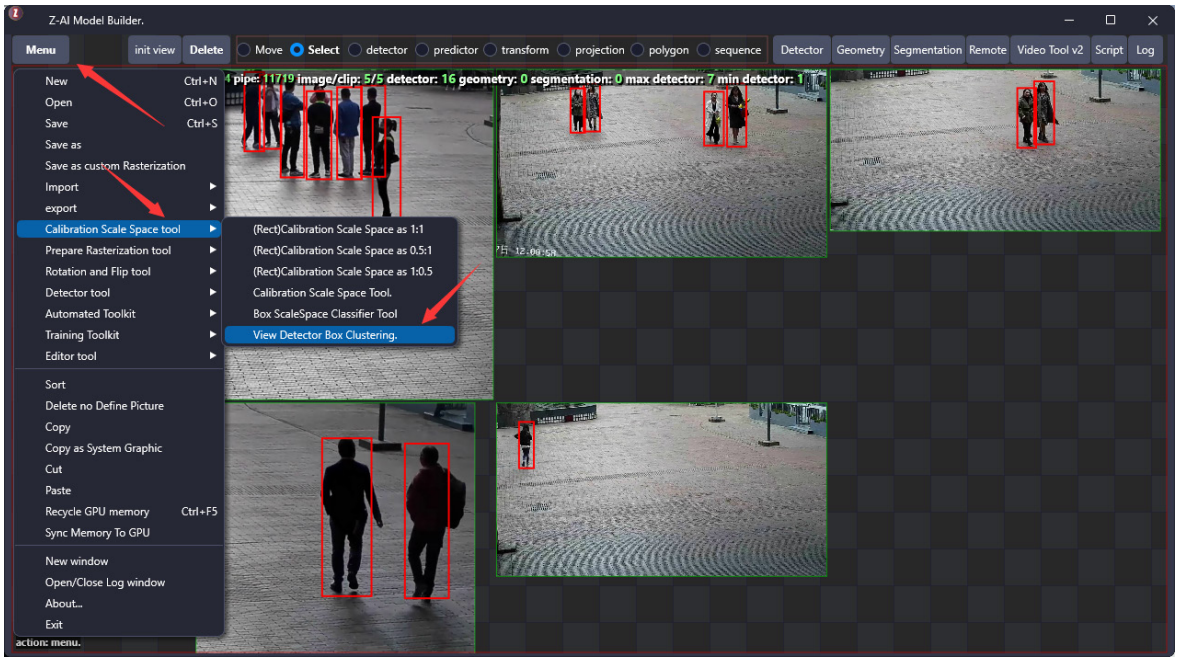
尺度分类标签是一种尺度数据层面的编辑,它的作用是根据尺度差异,从而给出不同的标签,通常来说,尺度差异不可直接由人类判断,需要使用机器计算尺度,然后做出尺度分类.

尺度分类主要用于对号流程化,例如流程走,检测器->分类器,当检测器找到目标,让检测器给出一个目标的分类标签,然后再让流程根据分类标签使用对号的分类器,这时候可以做出 2 级流程.

2 级流程会先使用检测器模型找目标,然后再用分类器模型对目标做分类,尺度分类标签是给 2 级流程组合使用的,因为分类器的模型都是单尺度适应性,一个分类器一次只能做一种尺度,而检测器可多尺度.2 级流程会让一个检测器模型对应多个分类器模型来工作.(2 级流程在 Z-AI 的监控体系中是自动化的流程,不需要编程)

做万物识别类的 AI 项目一定要多思考一下本小节的内容:检测器可以直接输出不同的标签,例如,检测器到人+垃圾桶+自行车,这类检测器训练会非常耗时,并且 Loss 精度也会难以控制,对于走简易分类流程的检测器来说,它对处理复杂标签并没有特别突出的能力,对检测器来说不同的标签可以等同于内部走多个检测器,除此之外,检测器对标签的判断没有候选能力.检测器工作定义:人和垃圾桶只用一种尺度+标签,自行车用另一种尺度和标签,不让检测器直接输出标签,而是输出尺度标签,最终的标签用 2 级流程来干.

首先打开检测器尺度分类可视化工具



这里是个小工具

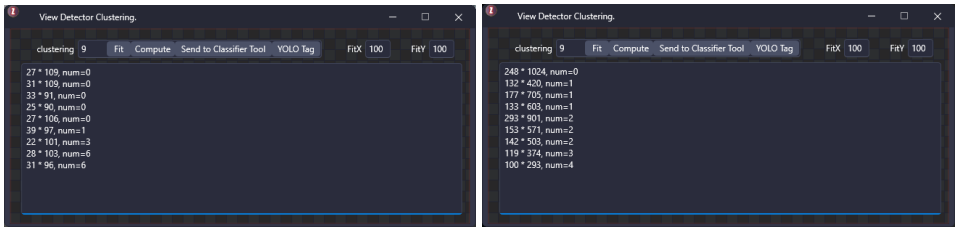
Fit:这是查看尺度拟合后的分类结果,该功能会根据 **fitX+fitY** 先做一次拟合,然后输出 **9** 个尺度分类,最后,再用尺度相似度去和 **9** 个尺度分类做匹配,从而得到一个可匹配的数量结果

Compute:这是查看 **Size** 分类结果,该功能是直接根据宽高做 **9** 个分类,然后匹配分类结果

Send to Classifier Tool:将 **Compute** 结果(**Size** 分类)发送至尺度标签编辑工具

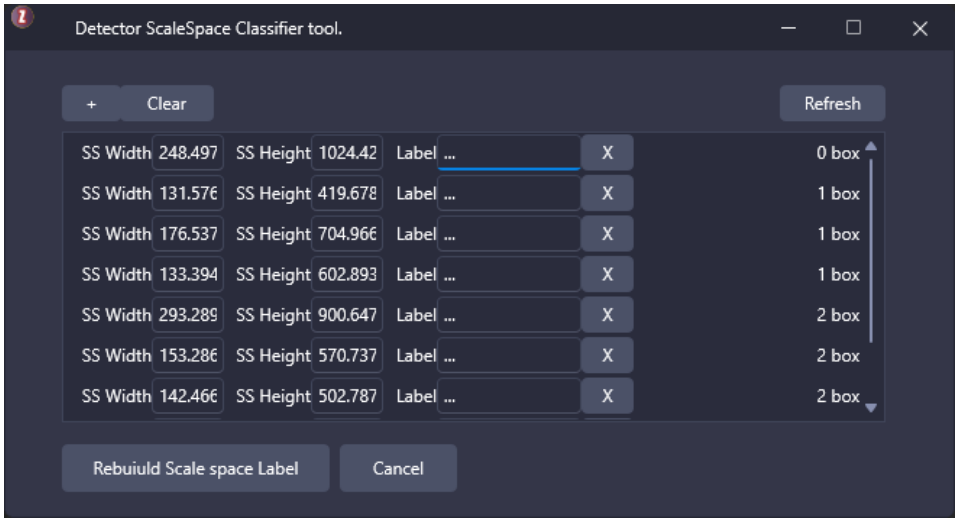
YOLO Tag:生成 **YOLO** 体系所使用的先验框数据

下图为 **Fit** 分类和 **Compute** 分类



Send to Classifier Tool,出现尺度分类标签的编辑工具,这里是给不同尺度的框框打上标签,当标签被打上以后,训练出检测器就会输出一个 **Box+Label**,而 **Label** 就是用 **2** 级流程使用的条件标签. 再说一次:不要直接用检测器来输出最终识别 **Label**,检测器的分类流程是简易化的,不可以直接用于实际项目.

只需要在 **Label** 里面填写对应的标签,然后点重建就会生成好.

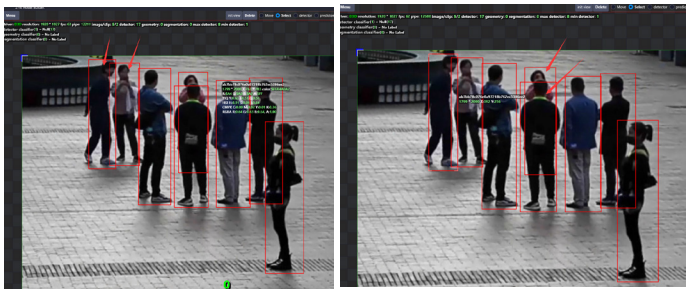


重叠标注

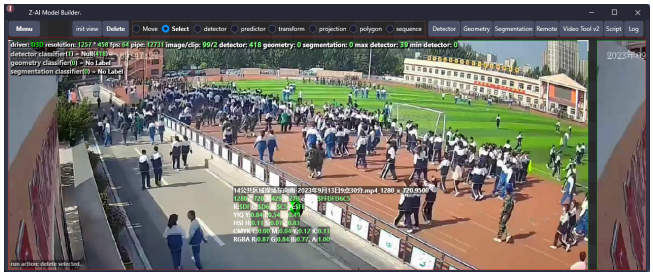
重叠检测能力是一项非常要的实用级功能,例如统计图像中的数量,人数.重叠检测是深度学习时代专有的产物,在早期形态学+数字化视觉时代重叠一直面对的问题是算力瓶颈(重叠是深度学习中针对张量的大规模 Loss 级计算,无论耗时,计算量,都很巨大).
在 Z-AI 训练体系中的所有支持重叠的检测器都会有类似 IoU 这种参数,它会包含在训练参数里面,IoU 是一种浮点数值在 0-1 之间.关于 IoU 的 pas 版计算公式在标准几何库如下.

```
function Compute_IoU(const r1, r2: TRectV2): TGeoFloat;
var
  R: TRectV2;
  f1, f2, f3: TGeoFloat;
begin
  if not Clip(r1, r2, R) then
      Exit(0);
  f1 := RectArea(r1);
  f2 := RectArea(r2);
  f3 := RectArea(R);
  Result := f3 / (f1 + f2 - f3);
end;
```

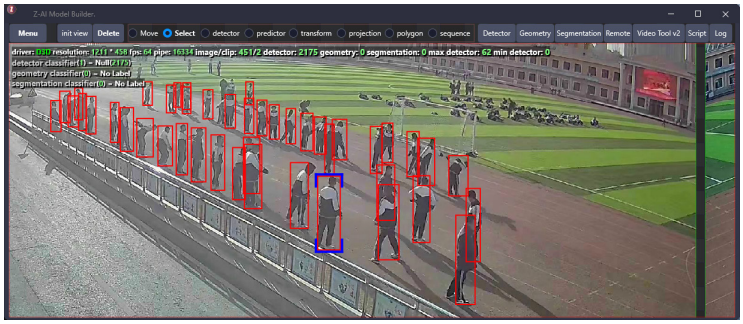
对上面 IoU 代码简言解释:当两个框框发生重叠,包括相交重叠(AB 框各重叠一部分),完全重叠(A 或 B 完全被另一个框覆盖),那么就会返回一个值,表示重叠的占比尺度,0 为不重叠,0.5 表示有一半是重叠,1.0 表示完全重叠.
下图中左边箭头所指在两张 IoU 大约 0.1-0.2 之间,右边箭头的 IoU 会在 0.7-0.9 之间,对于这些 IoU 的处理方式和限制,例如训练流程判定为重叠计算模式是 0.1,忽略重叠为 0.9,下图中的 IoU 是撰写文档时凭目测给的,准确的数值大家可以把 IoU 计算公式抄下来,然后自己写个程序来验证.



重叠标注需要避免用左边这种零碎标注去跑右边这类大规模聚集的识别.
大规模重叠直接标注条件会很苛刻:分辨率要够大,够清晰,并且需要能看清全部目标,这种样本非常不好找,解决大规模重叠的标注办法是用堆一定数量的小规模重叠标注,并且标注的环境与大规模聚集有很多相似性,例如都是在白天,广阔地面,光照度,目标人物的衣着.
简单来说:使用 Transform 抠图,确保图片中的人清晰,并且尽量保留环境信息,然后重叠标注.
解决大规模聚集重叠识别,使用 YOLO 系以及 DNN-OD 系都可以胜任.如果样本中出现大量重叠标注,那么训练的耗时会更久,至于 Loss 结果,这要取决于样本和样本环境的丰富程度.训练中的超参数这些其实影响不大,最直接暴力做法是标注大规模重叠去识别大规模重叠.

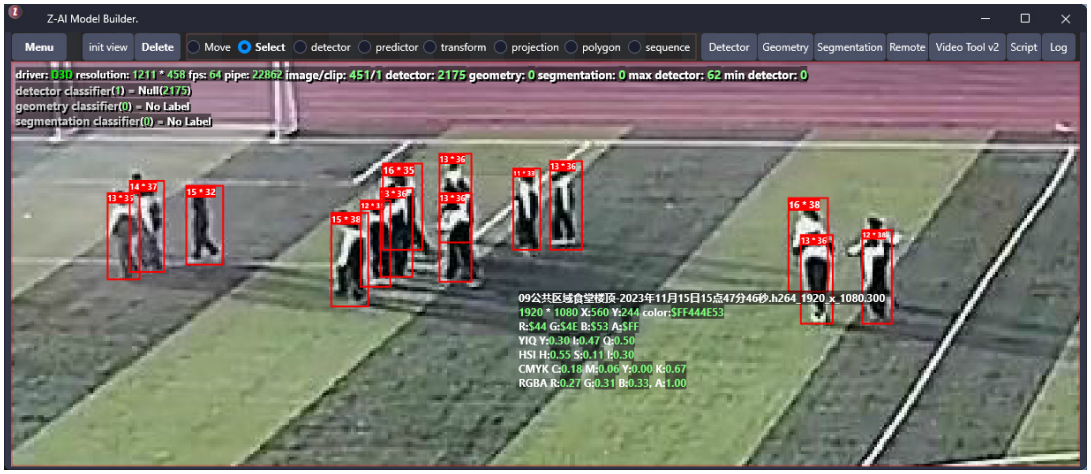


大规模重叠标注可以多找几张小规模群集的样本来标注,堆小规模标注到一定规模,可以完成大规模重叠检测,在下图中,远处许多目标非常模糊,几乎无法标注.因此下图标注不可用于样本,环境中已经包含了其它人,而其他人并没有标注.参考章节:选择检测器样本.



下图因为个体明确但图片细节很糟糕,在没有高清群集样本时,只能勉强使用,这会影响建模精度,但总比没有群集的样本强点.始终建议多在样本下功夫,建模是迭代行工作, 选择样本环节懒一下,可能导致后续工作前功尽弃.

通常高清群集样本非常不好找,非常模糊的人形与垃圾桶几乎没有差别.广场人群这类样本,用单反现场拍一些人,细节品质会远超监控,而当细节和环境到位相信可以做出非常通用大群集检测器模型.



自动化标注

自动化标注就是做通用模型,再用通用模型来完成标注.做通用模型的前提是数据,不需要跑去寻找别人建好的模型,建模和标注都不是问题,只需要有数据就能生产通用模型.

说自动化标注需要先明确自动化标注是用于数据科学领域的,前置工作如果解决了通用化模型等同于解决了未来繁琐的拉框标注,这可以作为一种加速生产的技术来 对待.当使用大数据构建检测器时(标注框规模大于 10000 个),YOLO 体系的兼容性会优于 DNN-OD 体系,兼容性是指样本中存在不规范的标注,例如十分模糊的目标,对于实际建模工作,很难让所有样本都清晰无误.最后,YOLO 检测器会输出许多目标,同时也会伴随许多错误的标注.对做数据工作来说,人工给自动化标注排错会是不可避免的,更多的工作会是从茫茫数据海中寻找更加优质的样本,如果以数值化来说,数据工作会由 80%优质样本+20%的标注组成,也就是说,你会花掉 80%的时间来寻找样本,当然,现在有大模型技术提高效率,当优质样本到位,标注可以用时间来推,一般来说,成品级模型数千个标注 3-5 天左右就能标注好.所以,要做自动化标注,就把主要精力放在样本的寻找上把.参考章节,选择检测器样本,非常重要章节,值得反复阅读.

短视频是局部数据的好源头,但是,短视频分辨率有限,在面对大规模群集识别的需求,它会显得非常无力,在另一方面,人类对于隐私一直存在述求,作为数据源的短视频在未来相信会有机器化的再加工规范,也就是,例如出现在视频中的车牌号码,电话号码,面部特征,都会被清洗掉(blur,mosaic).例如解决,识别摆摊,识别生活垃圾,识别穿衣打扮,这类需求,使用短视频的数据源是可以的.具体看怎么使用关键字去搜索短视频.

搜索引擎数据源,建模高清人群检测器模型时,通过搜索引擎可以找到一些无人机的航拍图,如果是国内街景大部分是非法偷拍,并且许多全高清素材需要付费才能下载,而免费的航拍图往往很难使用.简而言之,搜索引擎和关键字出来的样本如果足够多,那就可以做出通用模型,否则就只有放弃搜索引擎了.

Z-AI 的监控体系有大数据支持能力,当部署的监控头足够多,可以按监控命名来分时下载大数据.例如拉取含有街道关键字的监控头,并下载 19-21 点时段的监控视频,之后,它会拉取数百 GB,甚至数十 TB 的视频内容下来,然后,再使用视频图片工具体系来慢慢寻找样本.监控体系的数据大都是私有化的,公共场所的数据有保密性要求,不可以直接放到公网共享,随意传播可能被定罪.Z-AI 的监控体系需要服务器到位和监控基建完成才可以使用,一般建成以后,几乎可以不愁数据来源,做私有领域模型无问题,如果做通用模型,需要放飞数据来源渠道,光用监控是不够的,即使几百路监控,样本中的场景数据也会偏少.

社区数据源,例如在 github 上被千星点赞数据,meta,google 共享的数据,这类数据品质往往都非常高,几乎都是专业 CV 使用的数据源,经过了专业筛选+标注.同时,meta,google 的数据都非常巨大,100TB 上百个压缩包是家常便饭,例如大模型建模所使用的数据,这些数据几乎 100%都来自互联网.

主题数据源,在 BBC,社会类科教的视频节目中,自然,社会,人文,几乎都是 4k 级分辨率,这对于建模通用检测器是可以有帮助的.或则说,注重于视觉的现代化科教片可以作为样本数据来源来使用,因为拍科教片的行为就是会围绕主题从各种角度来拍,正好作样本使用.这需要花点时间来看片和筛选.

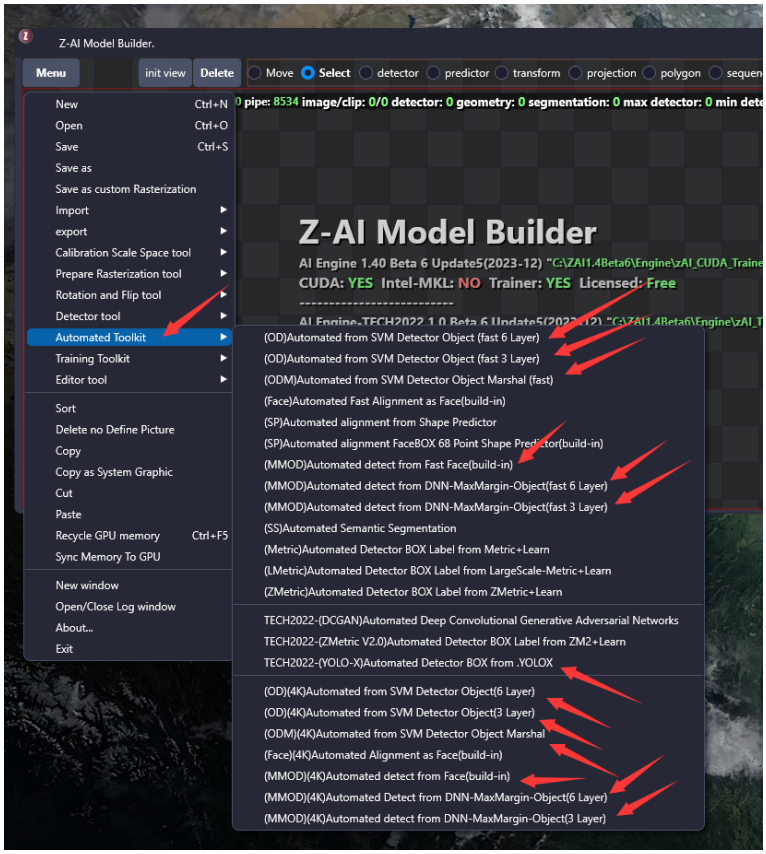
Edu 类数据源,这类数据源大都是一些学校的老师和学生在研究技术,自己弄的数据源,有很多是想非非而没有实用性的数据源,例如早期的自动驾驶探索,火警烟雾探索,在探索中,摸索解决方案,搜集数据,然后再放到 edu 网站去分享,这些数据,用现代化角度来看,几乎 90%都可以认定是业余数据源,并且历史悠久(可能是 20 年前的数据),数据的专业性只能是参考.

自动化标注的未来之路

在 Z-AI 未来的版本中,会确定引入大模型技术,例如 Meta 公司的 SAM 模型, Stability AI 公司的 SD 模型,这些大模型是造福全人类的基础建筑,因为这些大模型现在已知的数据规模高达 50TB,需要购买一些 gpu 和存储设备才能对它进行手术加工,这会是一件繁琐的工作,但对于未来提升工作效率,会很意义, 我不确定通用大模型可以解决自动化标注,但我会尝试,我能确定大模型可以解决数据筛选难题,当面对监控视频数据海,图片数据海,大模型可以帮助我们找到最合适的样本,这将非常有意义,意念到哪即可建模检测器到哪,这样的美好构想,对我们来说并不遥远.

列举 Z-AI 的检测器模型

在 Z-AI 体系中可以自动化标注的检测器模型种类非常多,红色箭头所指的地方是不同的检测器模型,只有熟悉了 Z-AI 的整个建模体系以后,才能驾驭这些功能,因为自动化标注需要模型,而做模型需要建模,建模又需要样本.本小节建议在了解完建模以后再来研究.



(OD)Automated from SVM Detector Object (fast 6 Layer)

SVM-OD,古典派的对象检测器,6 层模型,无预置处理,直接标注

(OD)Automated from SVM Detector Object (fast 3 Layer)

SVM-OD,古典派的对象检测器,3 层模型,无预置处理,直接标注

(ODM)Automated from SVM Detector Object Marshal (fast)

SVM-OD,古典派的对象检测器合集,古典派 OD 一次只能检测一种目标,合集是把多个模型打包成一个合集模型,从而实现一次检测多种目标,无预置处理,直接标注

(Face)Automated Fast Alignment as Face(build-in)

使用内置的通用人脸模型来做自动化标注,会先做人脸检测(OD),然后再做人脸对齐(SP)

(MMOD)Automated detect from Fast Face(build-in)

使用内置的通用人脸模型来做自动化标注,人脸检测使用 DNN-OD(可以检测出半脸这类重叠目标) ,然后再做人脸对齐(SP)

(MMOD)Automated detect from DNN-MaxMargin-Objec(fast 6 Layer)

使用内置的通用 DNN-OD 模型来做自动化标注, 6 层模型

(MMOD)Automated detect from DNN-MaxMargin-0bjetfast 3 Layer)

使用内置的通用 DNN-OD 模型来做自动化标注, 3 层模型

TECH2022- (YOLO XJAutomated Detector BOX from .YOLOx

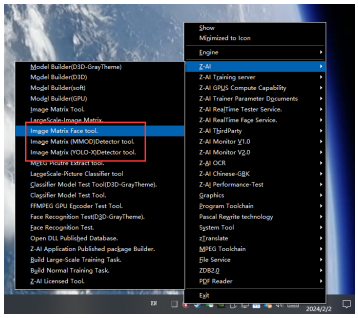
使用 YOLO-X 模型来做自动化标注

4k 预置化处理后,再做自动化标注, 更加耗时,显存+内存消耗会更大

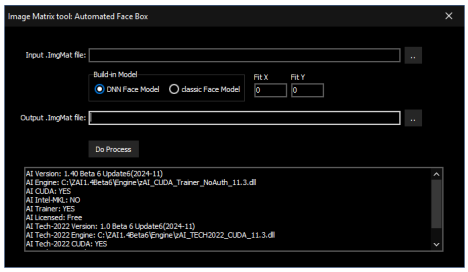
(OD)(4K)Automated from SVM Detector Object(6 Layer)
(OD)(4K)Automated from SVM Detector Object(3 Layer)
(ODM)(4K)Automated from SVM Detector Object Marshal
(Face)(4K)Automated Alignment as Face(build-in)
(MMOD)(4K)Automated detect from Face(build-in)
(MMOD)(4KJAutomated Detect from DNN-MaxMargin-Object(6 Layer)
(MMOD)(4KJAutomated detect from DNN-MaxMargin-Object(3 Layer)

列举 Image Matrix 自动标注工具

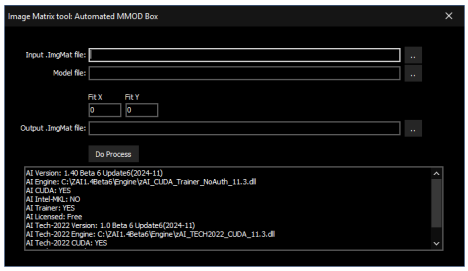
后缀为.ImgMat 是 Z-AI 体系的样本数据大群集包,通过菜单打开对应的自动标注工具



使用内置 face 模型的标注工具,直接标注群集包含有人脸的图片



DNN-OD 标注工具



YOLO-X 标注工具

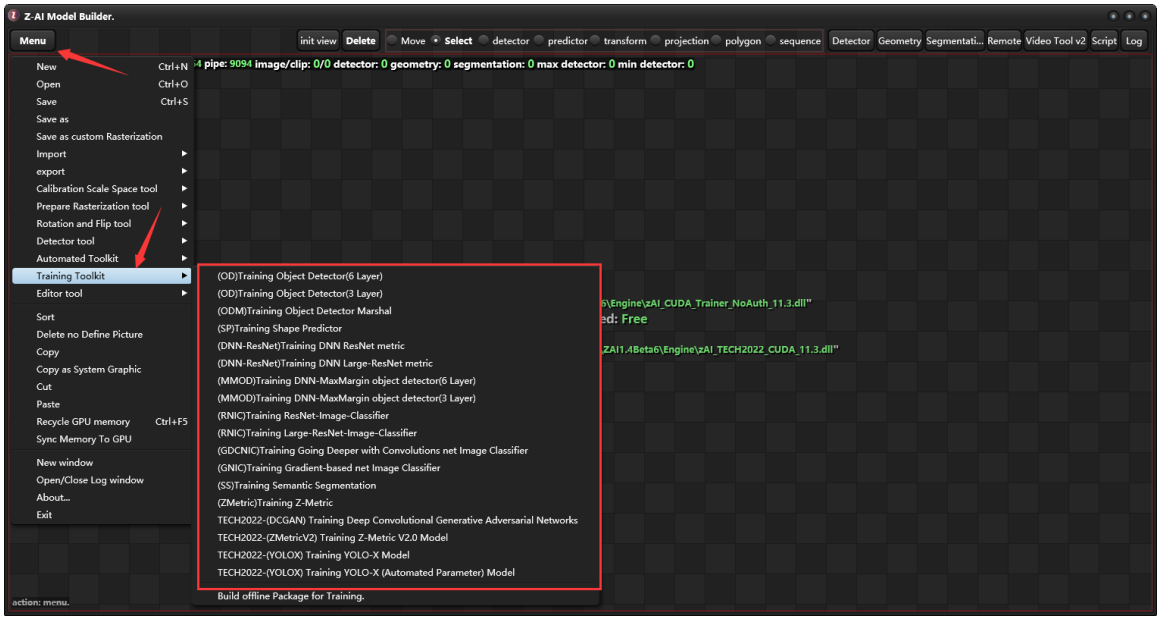
训练

完成标注后,训练模型是一件轻松而漫长的过程.

检测器训练体系分为传统训练(大都基于 CPU 的古典类 OD)和 GPU 训练(YOLO-X,DNN-OD),而在 GPU 训练中有两种分支模式,分别是 Normal(日常训练),Large-Scale(大数据训练).

检测器并不像分类器那样需要大量数据,通常来说,例如最常用的通用人脸样本库,图片数量在 5-8 千张,标注框在 10000 个左右.在实际建模中,具有场景差异的图片能达到 1000 张基本都可以算比较通用了.这时候,训练模型其硬件开销主要就是算力,对显存和内存要求很低.

下图红框中每一项菜单都是一个独立的模型



(OD),(ODM)开头的菜单项属于古典 CV 的检测器,放在现代,许多人脸机,包括在手机使用的人脸解锁,其实大都还是古典 CV 派的,因为单人上镜识别,并没有重叠和性能的需求,没有理由去使用 DNN-OD 和 YOLO-X.

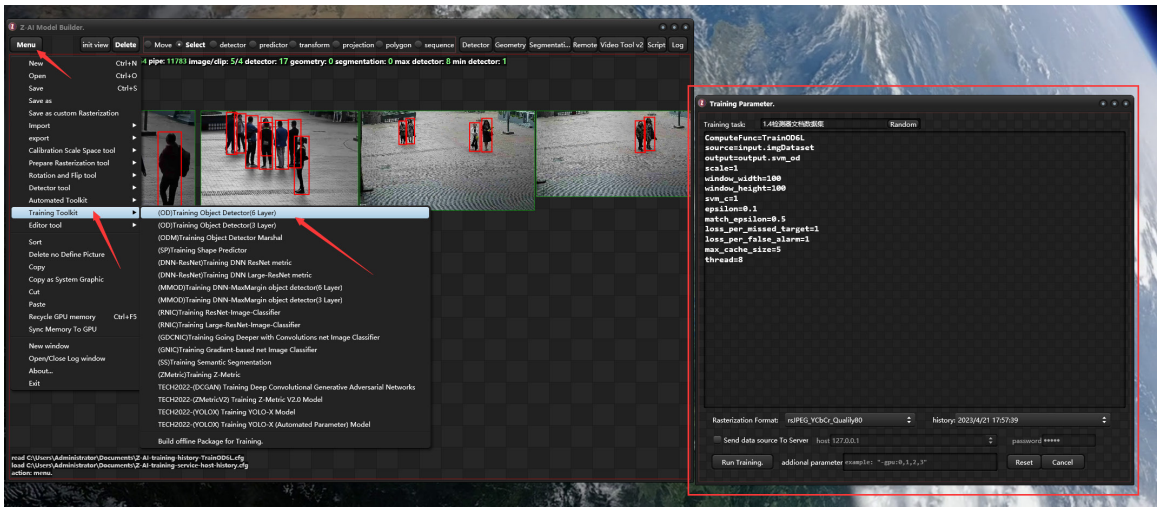
(MMOD),(YOLOX)开头的菜单是深度学习路线的检测器,其中 MMOD 对应本文的 DNN-OD,YOLOX 是 YOLO 体系的 X 版本,这是整个 YOLO 系列最前沿的版本.

6L/3L,分别表示 6 层和 3 层,其中 6 层是标准金子塔的下采样规格,3 层则是突出性能优先的金子塔下采样规格.对于 OD 和 DNN-OD 来说,检测都是不断的向下采样,然后滑动计算.

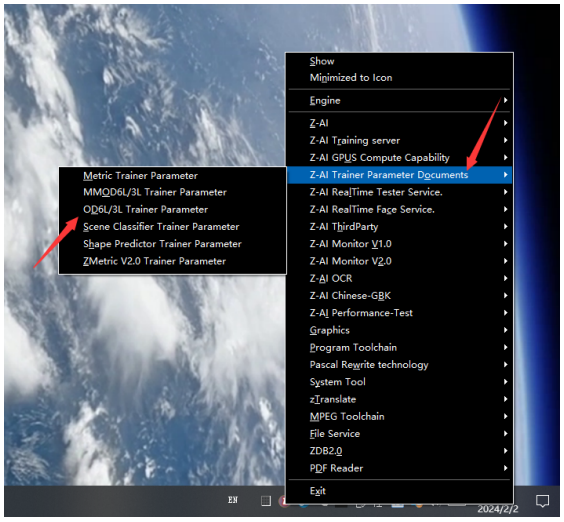
训练前应该准备两套数据样本库,第一个样本库是含有标注的样本库,第二个样本库是用于测试模型识别结果的样本库.如果走深度学习路线训练模型,建议准备两张 GPU 或则专门倒腾一台用于训练 GPU 服务器.

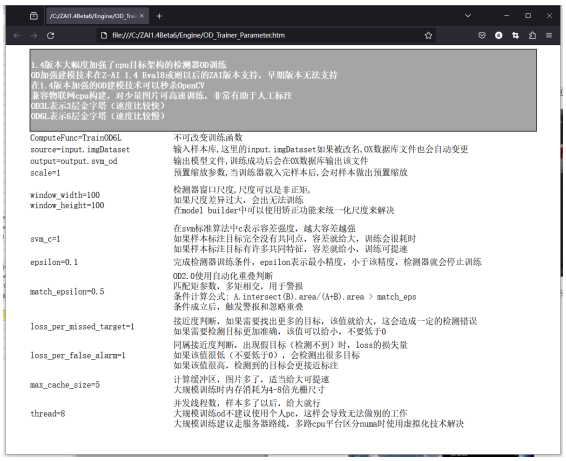
OD6L/OD3L

古典检测器在 Z-AI 体系有两种模型,OD6L/OD3L,可通过训练菜单打开,会出现训练参数窗口



了解这些参数的含义,可以通过 Z-AI 工具链菜单找到训练参数的详细说明文档





训练 OD6L/OD3L 需要标注尽量避免重叠,弱推理训练流程出现重叠标注会更加耗时和降识别精度.

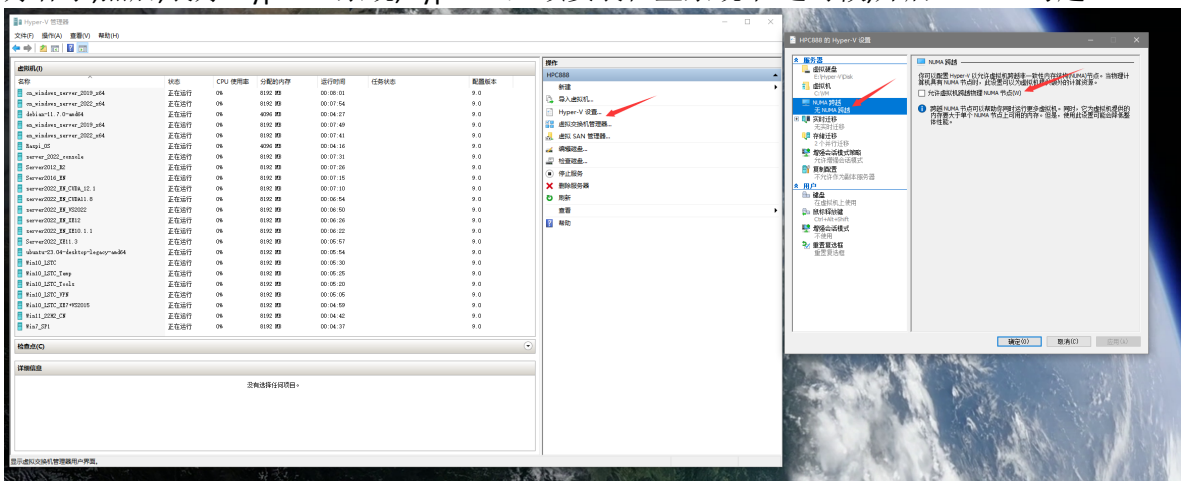
提示:OD6L/OD3L 不支持 Test 标签.

Z-AI 早期版本 OD6L/OD3L 模型都是简化的粗线条路线,在 1.4 版本,OD6L/OD3L 被大幅度加强,目的是让古典模型尽情的发挥作用,因为古典所以兼容,在 IOS/MIPS/Android/甚至国产,古典模型的移植性会非常好,古典模型的问题是建模会消耗 CPU,并且它的推理偏浅,无法和深度学习近乎无穷尽的推理方式相提并论.

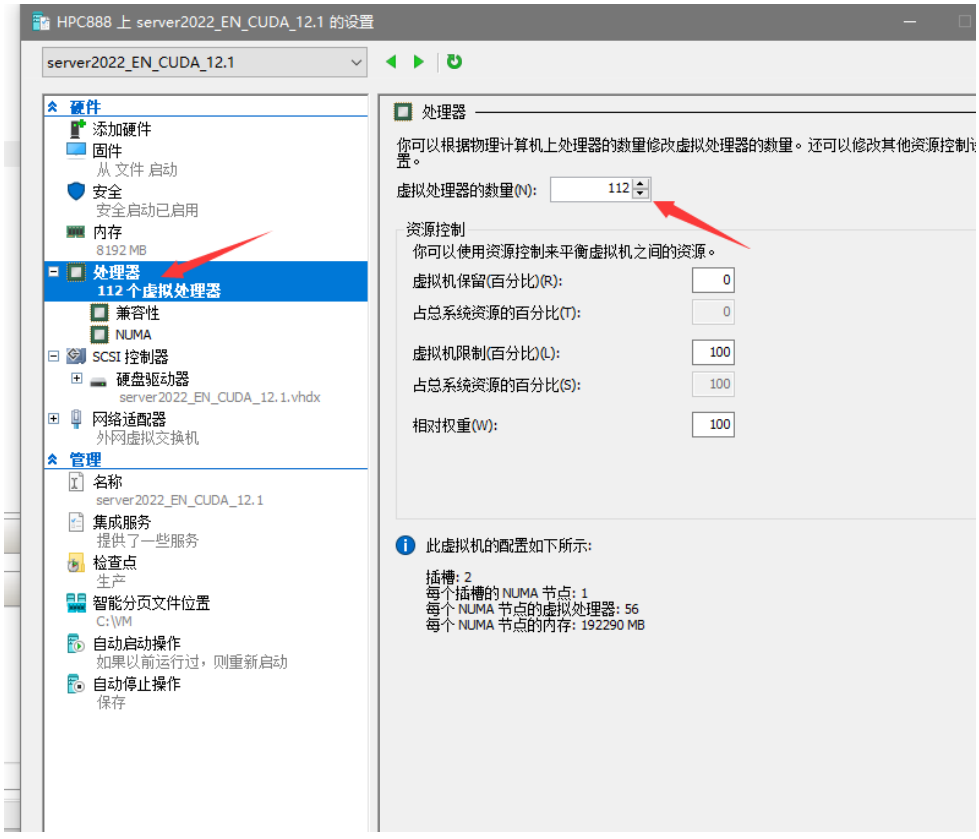
训练 OD6L/OD3L 时在多路服务器把 CPU 性能挖掘出来

OD6L/OD3L 主要使用 CPU 进行训练,当标注的图片多了,CPU 计算会非常慢,提速和优化就是充分发挥 CPU 的多核优势.下列介绍使用虚拟化技术来充分调动多核.

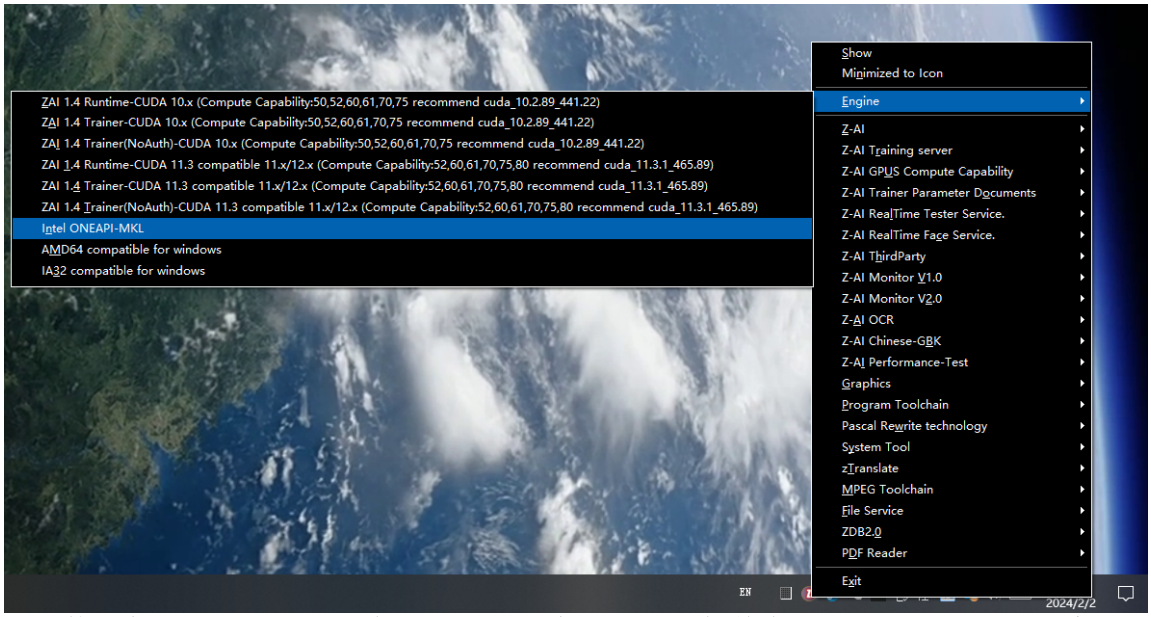
首先,服务器必须是多路,确保建模服务器是 2016/2019/2022 这类偏新的 Win 服务器系统,打好补丁,然后,装好 Hyper-V 系统,Hyper-V 必须安装在主系统中.这时候,开启 NUMA 跨越



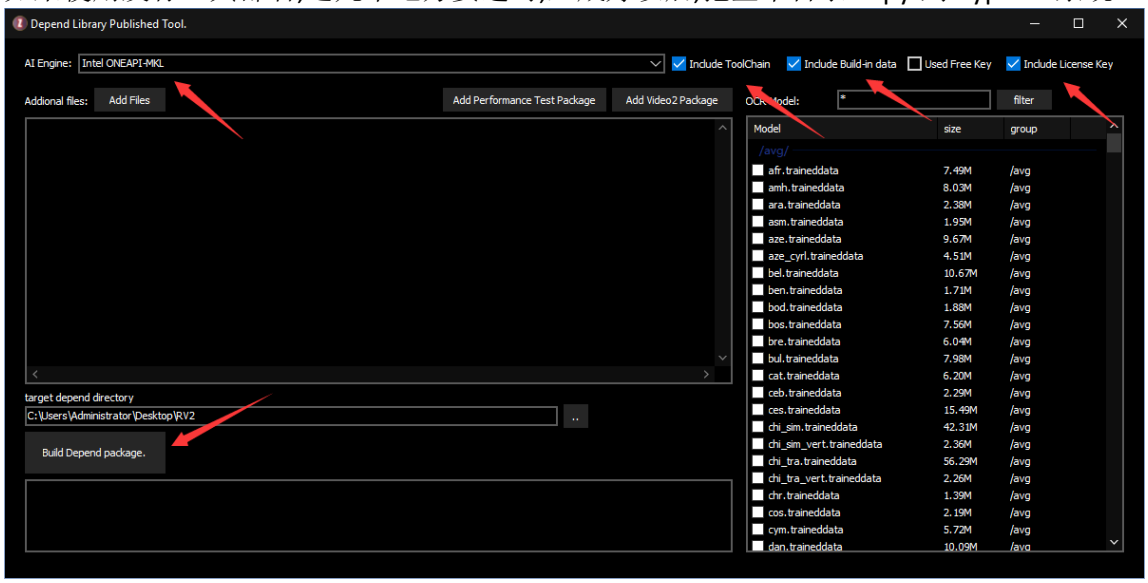
然后,在 Hyper-V 下安装一个虚拟化 Win 服务器,注意要包含桌面组件,然后把系统处理器的调成极限



然后安装一个 Z-AI 的副本,或则使用 Z-AI 发行工具生成一套可以部署的训练程序
如果安装的是 Z-AI 副本,那么,在工具链菜单中把 Z-AI 的计算引擎切换成 Intel-MKL

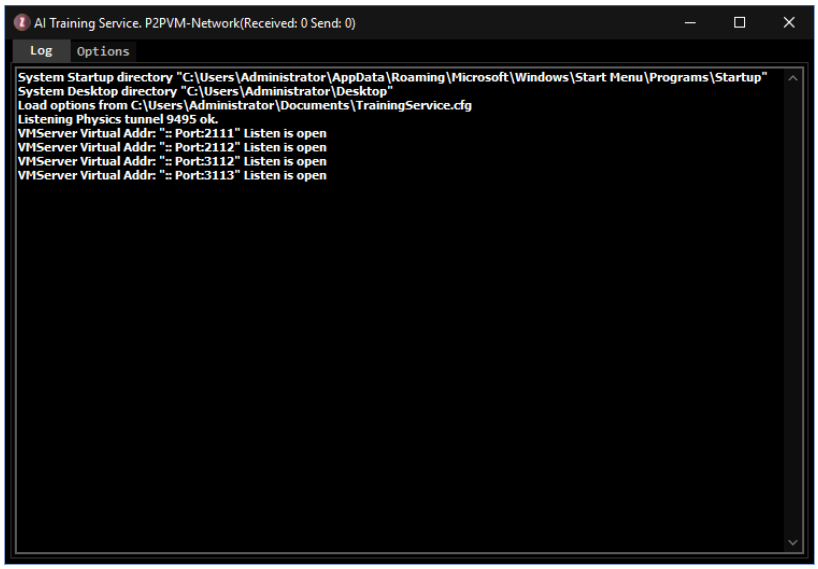


如果使用发行工具部署,这几个地方要选对,生成好以后,把整个目录 copy 到 hyper-v 系统

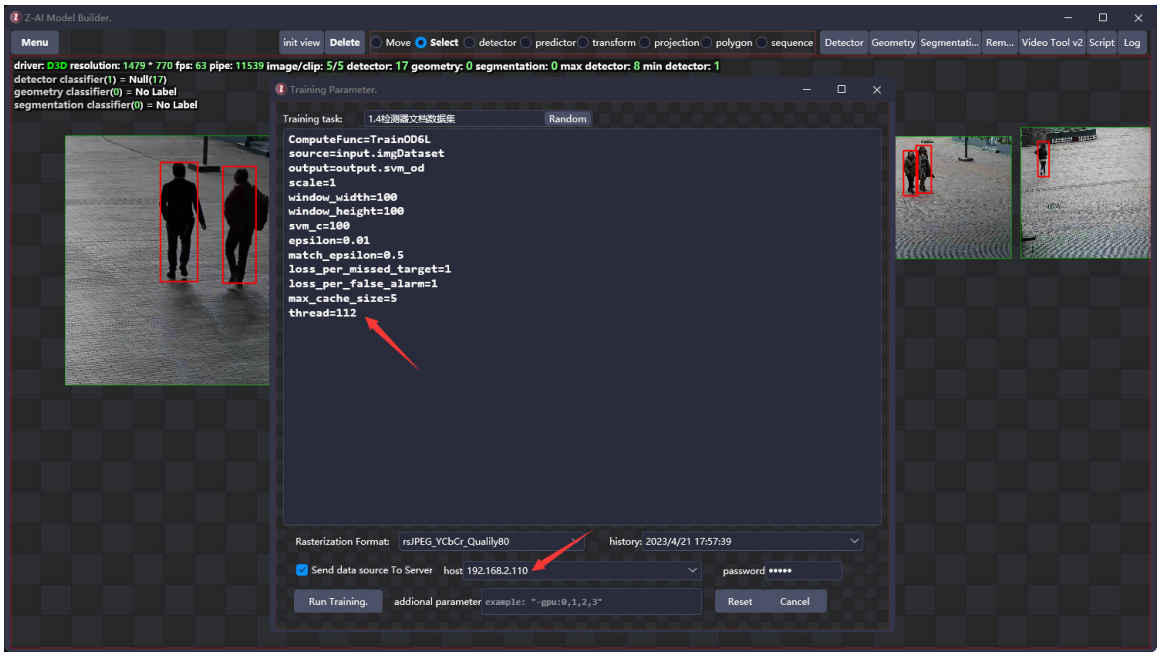


如果在新装系统安装和部署,需要运行一次 vc_redist.x64.exe,在部署包以及 Z-AI 根目录都可以找到 vc_redist.x64.exe

这时候,启动 Hyper-V 端的训练服务器,TrainingService.exe



最后一步是把训练样本发送至服务器,这时候,参数里面可以开出 CPU 极限



DNN-OD6L/DNN-OD3L(MMOD)

DNN-OD 体系的人脸模型一直被整个 AI 产业在使用,YOLO 体系的名声很大是因为 YOLO 建模方便,例如 YOLO 可以更自由的标注,而就最终的识别率来说,只要建模环节严谨一点,其实 DNN-OD 只会比 YOLO 略低,误差几乎可以忽略不计,最大差别就是标注,建模,训练这些环节.

讲解 DNN-OD 的训练先要说 GPU:

对于工作站,如果只安装一块 GPU 那么在训练时还要不要工作,这会是一个问题.GPU 并不像 CPU 那样可以自由控制 app 的算力开销,GPU 设计的起点就是让算力最大化,因此工作站一旦启动训练任务,显示器上的画面就会变慢无比,甚至连鼠标的移动就慢那么 0.x 秒,更不用说各种操作了.

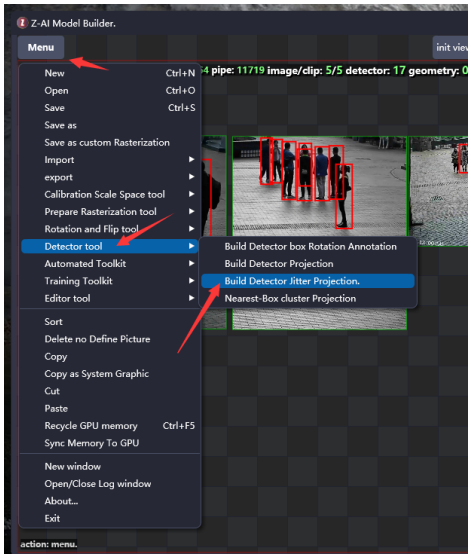
假如工作站安装安装 2 块 GPU,训练用一张,工作另一张,这样是可行的,并且这能做到工作与训练两边都不耽误,但也会有个问题,那就是内存带宽,和经济开销,在 GPU 的训练中,30G/s 的带宽消耗会是家常便饭,这时候,如果北桥带宽被用完,使用中也是会出现卡顿的,解决办法要么换更高级的 cpu 和内存(工作站的高端设备非常贵),要么就换服务器主板(服务器都是风冷,这会有散热噪音问题,这时候会需要分体水冷).除此之外,工作站因为 GPU 限制,即使 2 张 GPU 的高配,同时也只能开出 2 个 DNN-OD 的训练任务,当然,这是量产训练的需求,普通的训练,一次开一个 DNN-OD 训练任务已经足够,同时,2 张 GPU 工作站相比单 GPU 工作站这已经有质的提升了.这是一个工作站选择与权衡问题.

在实际项目中训练 DNN-OD 通过远程使用 GPU 服务器会是一间非常轻松愉快的事情,首先服务器在远程机房可以避免噪音,其次,单台服务器大多都会插上 4-8 张 GPU,训练时,超参数配置和标注优化会有一个摸索过程,怎样给超参数,以及怎样优化标注可以让 Loss 走的更好,这可以同时开多个任务来试错,例如任务 1 给 100*100 的最小检测尺度+AB 样本库,任务 2 给 80*80 最小检测尺度+CD 样本库,依次类推..先把任务开出来,然后,睡觉,第二天早起以后来看两个任务的训练结果,怎样的组合效果更好,开多任务训练主要让训练更加轻松,因为这使用是远程机房的 GPU 服务器,这太安静了,拿笔记本都可以提交 DNN-OD 的训练任务.

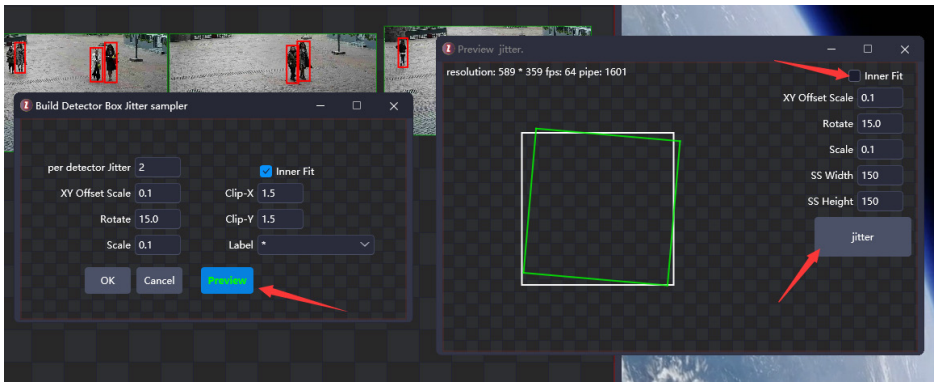
DNN-OD6L/DNN-OD3L,可以支持多 GPU 合算一个任务,但是传递开销一直都是问题,因为 DNN-OD 的网络架构设计原因,GPU 的算力消耗非常小,几乎是 YOLO 体系的 1/10,例如在训练任务使用-GPU:0,1,2,3 参数,一个任务同时使用 4 个 GPU 设备,这时候,每个 GPU 的算力开销大约只会 有 20-30%左右,并不会 100%,因为 DNN-OD 对 GPU 算力消耗太小了,每次往 GPU 发送一批数据计算,这时候 GPU 已经计算完成,而发送程序还在准备下一次发送.GPU 实际训练能力的体现是:每一个 DNN-OD 的训练任务使用单卡 GPU 训练是最快的,如果使用 2 个 GPU 训练的结果会是单卡的 80%-95%左右.针对该问题,我曾经尝试性做过线程优化:给 DNN-OD 开辟了 10 个数据预置线程,但结果仍然是没有任何效果,再后来,经过仔细分析,确定了问题所在:并不是预置数据延迟导致 gpu 跑不满,而是训练中算力开销太小,pci 总线响应跟不上导致.对硬件感兴趣的用户,可以试试在 pci4 的服务器跑 DNN-OD,对比 pci3,其中,pci4 跑多 GPU 的 DNN-OD 训练会比 pci3 提速 40-80%间.

理解 DNN-OD 的推理流程

首先打开建模工具,依次在菜单中找到 jitter 投影器,在 Z-AI 的标准 Demo 中也有许多关于 Jitter 算法演示程序,这些 Demo 按 jitter 关键搜一下都能找到



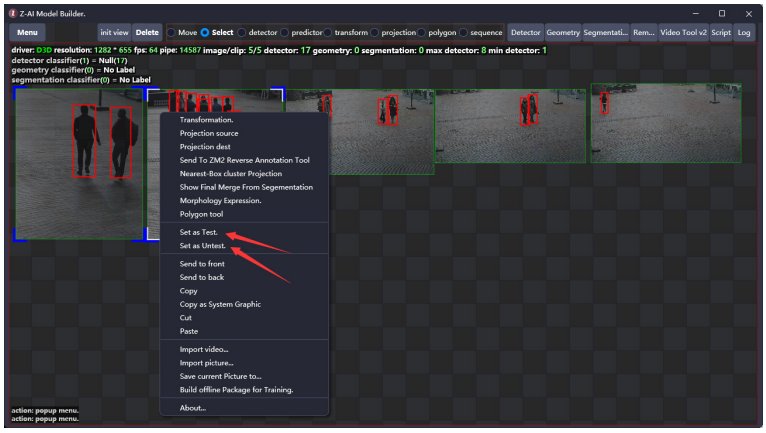
点开它,然后点 preview,会看到一个抖动的 box 随机乱跳



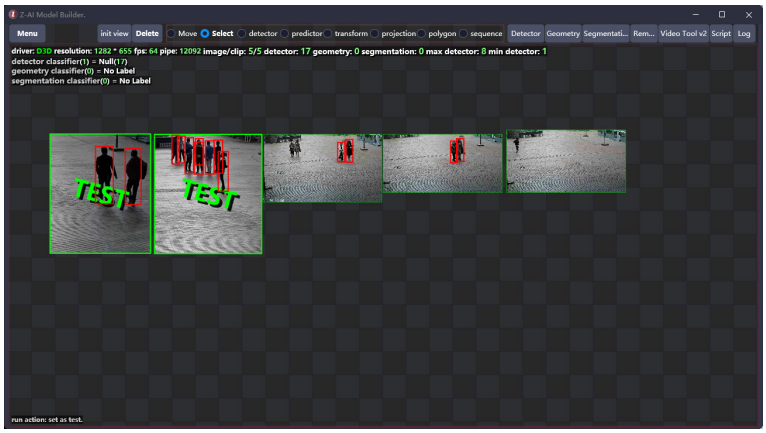
现在,回到本小节的主题上,DNN-OD 的训练流程,其实就是用 jitter 不断的去雕刻张量,Loss 这些东西都是数学上的方法,jitter 是一种 cv 形态的深度学习推理方法,通过不断的 jitter,然后把框框里面图抠出来,输入给网络做小块拆分,然后计算张量(拟合),这些张量在计算中,被 DNN-OD 框选中的目标会出现相似性和差异性(差异就是目标之外的场景,这些东西在 DNN 网络就是一堆数字),这时,当 jitter 的数量级达到一定规模,比如 1 亿次,现在,随便 jitter 怎么随机变换,都无法继续来雕刻张量,就会出现失效计算现象,这就是在训练流程中经常看到的失效步数,whitout step,在 DNN-OD 的训练流程中失效步数达到一定规模,就会开始自调内参,这种内参是学习率,这种学习率会影响张量的拟合粒度,当张量无法被雕刻导致失效步数增加从引发学习率变化而学习率影响张量拟合粒度,依此循环,最后,张量将被雕刻成最接近的检测目标的形态.这整个流程都是一种深度推理,这也是主流的 DNN 推理机制.

Test 标签的的作用

在建模工具中用弹出菜单,会看到 test 标签功能,test 是在一张图片中打上 test 记号,



有 test 记号的图片会有绿色的 TEST 大字效果



现在,需要回顾前面章节所讲到的因为图片模糊从而引发训练精度下降问题(参考,重叠标注)

Test 标签的作用是让正确度最高的图片样本被更加频繁的 jitter,例如样本库中有一些比较模糊样本以及清晰样本,这时候,jitter 被 dnn 流程执行 1 亿次,在其中就用 test 标签让清晰样本被抖动 9000 万,模糊样本只让它抖 1000 万,这样来干,模糊样本影响 loss 精度的难题也就解决了.

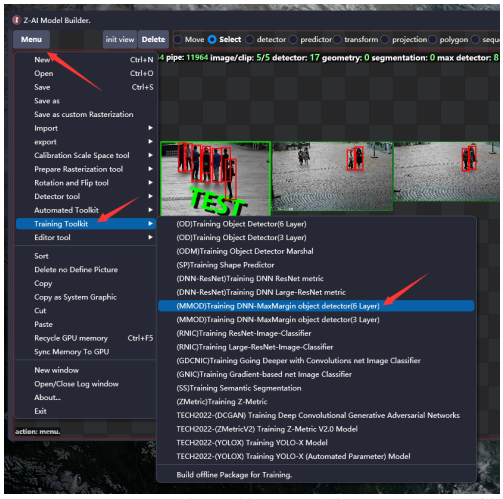
Test 标签主要用在样本繁多的数据集中,例如上千张样本,而这些样本又非常优劣迥异,那么就用 Test 标签给品质更好的样本打上,从而引导推理流程往更好 Loss 发展.

Test 标签小节,需要结合,重叠标注,DNN-OD 推理流程,反复阅读,理解后,会对于实际项目建模起到很大的推进作用,尤其在遇到推理几天下来,loss 始终不下降的情况.

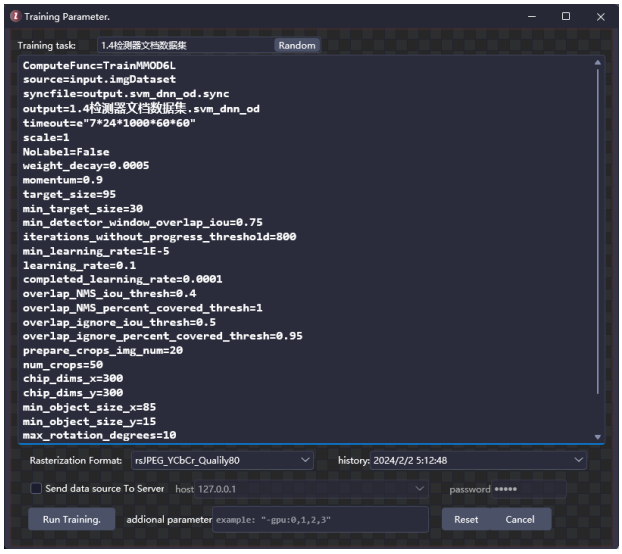
整个 AI 产业圈子使用的人脸模型,都有 Test 标签机制,从数千张样本基础上训练而来.

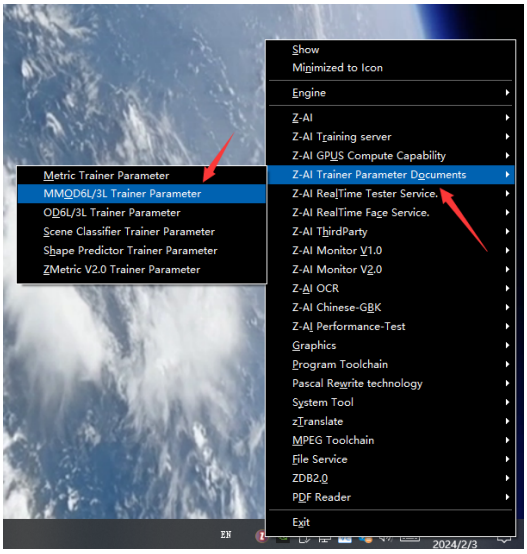
DNN-OD 的训练参数

DNN-OD 的训练器可通过以下菜单打开



DNN-OD 的训练参数非常繁多,这些参数功能和作用,都在工具链菜单指向的文档中,同时在 Z-AI 的标准 Demo 中也能找到关于 DNN-OD 的训练参数说明,看文档更直接.



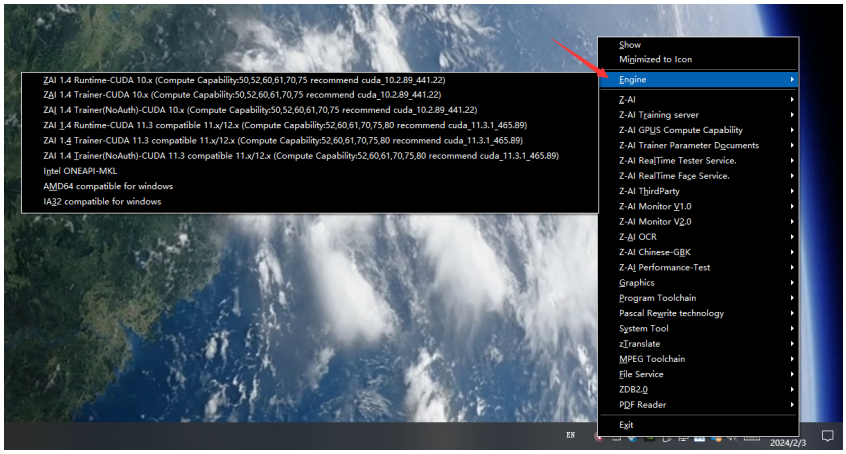


YOLO-X

训练 YOLO-X 所使用的 GPU 设备很偏单卡性能, YOLO-X 模型不支持多卡同时训练,但是 YOLO-X 模型可以支持多卡同时跑识别.如果需要大量 YOLO-X 建模,应该选择高性能单卡 GPU.

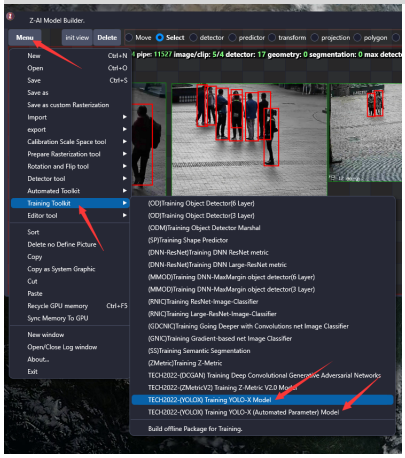
YOLO-X 单任务如果用多 GPU 训练,会出现显存平均,计算只能集中在单一 GPU 上,YOLO-X 的训练流程非常紧凑,目前在计算引擎只做到数据切分传递,流程是单一 GPU,修改计算引擎难度偏大,因为 YOLO-X 在 vs 环境的 build 大约在 15-20 分钟,程序架构是一个 cpp 承载全部 YOLO-X 泛型,修改 YOLO-X 只能把 cpp 的编译模型拆开,然后让编译提速在 2 分钟内,这个时候才能调试和改 gpu 程序.

训练 YOLO-X 模型需要使用 CUDA 类计算引擎,YOLO-X 算力消耗偏大,仅次于语义分割.日常训练耗时大约 2-5 天.

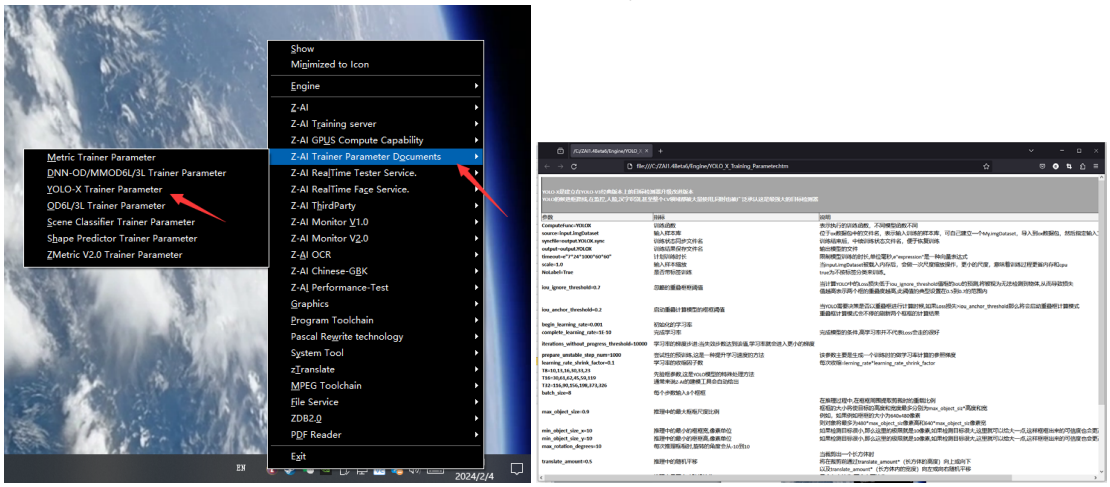


在标注工具启动 YOLO-X 训练,上面为原型参数,下面为自动化先验框.
自动化先验框会先生成 9 种尺度,然后排序,数量最多的走 8*8,数量最少的走 32*32,依次类推.因为 8*8 的细节信号更少,需要最多的框框来雕刻,32*32 往往是大图标注,这种标注的信号相对更丰富,细节更多.

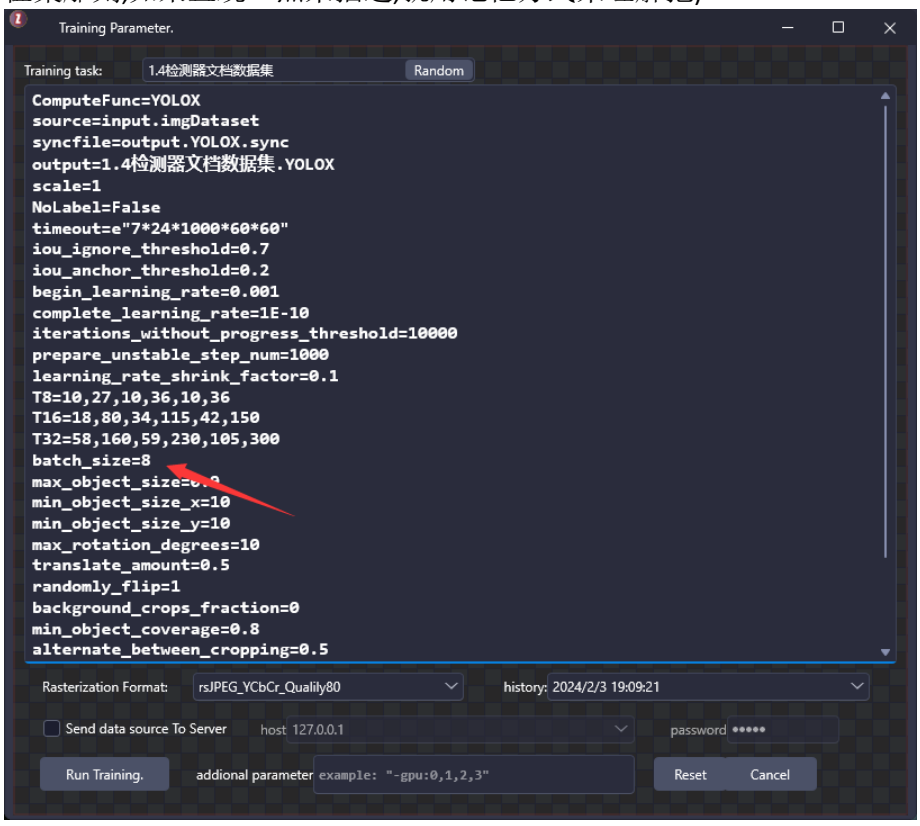
补充:YOLO-X 的先验框并不是根据绝对 size 生成的,这是梯度尺度,从 T8,T16,T32,一共三种尺度梯度,每种梯度各有三种以排序结果计算出的尺度.



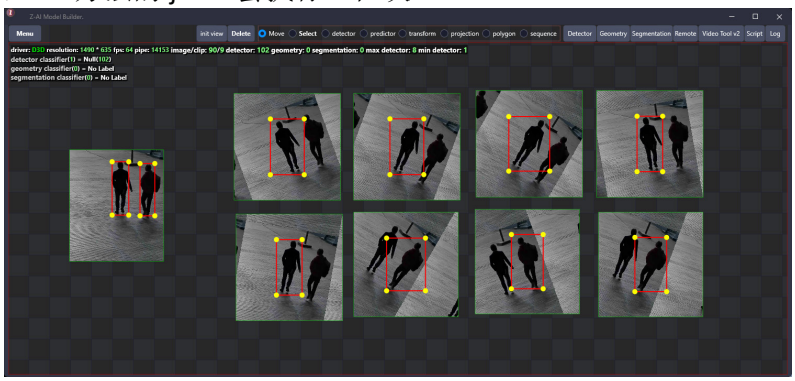
YOLO-X 的训练参数文档位于工具链指向的菜单中,这是一个 HTML 文档



因为 YOLO 体系会记忆许多目标特征,batch_size 是指每一个计算 step,会生成 8 张图,每张图都会有个标注框,这个时候框框是学习目标,框框周围是环境,YOLO 思路体系是把这些目标和环境全部用类似记忆的方法进行计算(从用严肃的技术来说,这不是增量记忆,应该说,超大特征集雕刻,如果直观一点来描述,就用记忆方式来理解把)



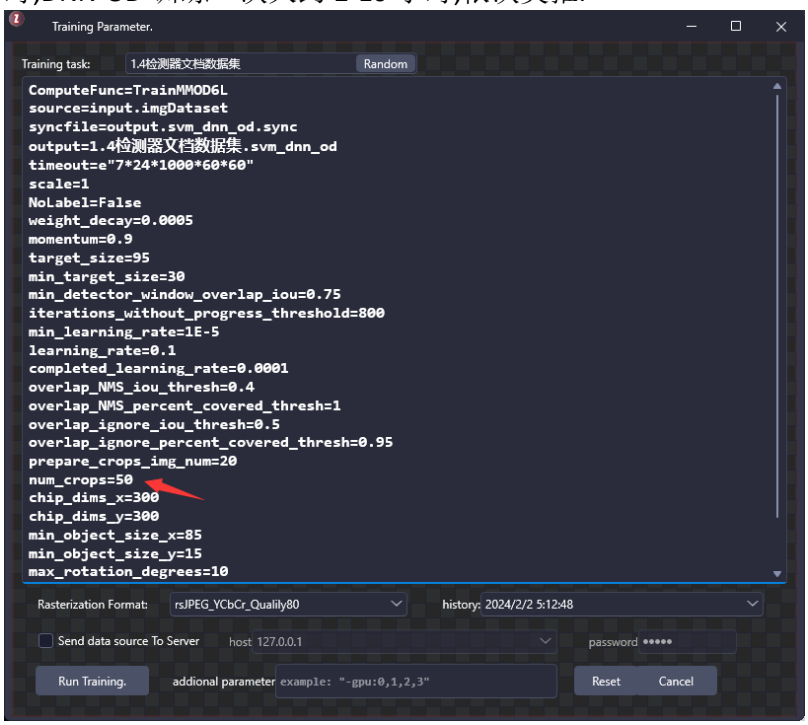
上面提到 batch_size=8,这个时候假设样本只有一张,就是左边的 2 框标注,那么 yolo-x 会在每一个计算 step 生成类似右边 8 的张 jitter 样本(下图用软件方式模拟),然后把整张图输入给 DNN 网络.这个时候 DNN 网络会进入一种拟合计算模式,例如 8*8,16*16,32*32 这类小图块,它们会计算相似和异同,这就是特征,通过不断的 jitter,这个时候,YOLO-X 的矩形特征集就会慢慢的形成对目标对象+目标环境的张量轮廓(这是一种数字化的结构体,类似长矩阵),这些 cv 方法的 jitter 会执行上亿次.



这时候,来对比一下 DNN-OD 的参数,DNN-OD 每个计算步数,会输入 50 张类似上面的抖动数据,而 YOLO-X 一次只会输入 8 张抖动数据.在实际训练中,YOLO-X 的 8 张图会消耗 7-9G 的显存,而 DNN-OD 的 50 张图会<3G 显存.这是 DNN 网络的差异,YOLO-X 内存会记忆大量的特征细节,而 DNN-OD 则是根据目标来精细雕刻.

在 GPU 算力开销层面,YOLO 体系大约消耗 DNN-OD 的 10 倍算力,同时每个步数也差不多是 DNN-OD 的 1/10,这也是本节开头,为什么建议做 YOLO 模型优先考虑购买高算力单卡的原因.3090/4090 会是训练 YOLO 模型的性价比 gpu.

YOLO 训练一次耗时大约会在 2-5 天,如果 GPU 算力提升 4 倍,那么时间可能缩短至 10-20 小时,DNN-OD 训练一次大约 2-10 小时,依次类推.



对 YOLO-X 建模实操的建议

YOLO 建模会需要时间摸索参数和标注,因为标注很轻松,参数修改也很方便,唯一的问题是训练一开就是 2 天,这时候,需要计划时间,例如,训练尝试性标注的样本,如果有 4 个 gpu,那么可以一次性开多个任务+配置多个样本库来尝试摸索,这会大大缩短切入时间.
摸索参数和标注也可以看错是一种,做解决方案之前的热身训练,真实项目中的检测器并不全是有样本和 GPU 就能训练,样本会需要人去清洗,提炼,标注,打 test 标签,然后,训练,所有的建模我都不建议直接一来就堆大,而是走摸索参数和标注,训练,测试,这一反复的流程.直到达标.
同一类目标,例如行人,汽车,在 YOLO-X 可以跨项目堆砌,例如,项目 1 采集的目标样本,只要类型相同,项目 2 可以沿用,沿用时将项目 2 的目标打上 Test 标签 .

关于 YOLO-X 对 Test 标签的支持说明

YOLO-X 模型训练机制与 DNN-OD 存在许多差异,YOLO-X 训练流程的前置工作会做一次非稳定训练,这也就是参数,prepare_unstable_step_num=1000,这一行表示用 1000 个计算 step,来验证样本的稳定性程度,它会粗线条的对样本标注的可拟合性做一次流程评估,生成 Loss 可参考的 jitter 数据蓝图,这种蓝图并不是条件处理,并不是让某些图片如果拟合不好直接不计算,而是开辟一种计划权重,例如在 15 万的 step 时,loss 权重和拟合粒度被权重影响,这种影响并不是宏观的,而是细微级,如果用最简单方式理解,YOLO-X 训练模型时,整个样本集并没有 test 标签,而非稳定性训练流程,大致会起到 1%-5%有 test 样本的影响.
如果给 YOLO-X 样本打上 test 标签,在训练之初的头 2 天也许看不出明显对 Loss 精度的影响,当训练达到第 4 天,甚至第 5 天,Step 走上 100 万,test 标签的影响就会体现出来了.这时候非稳定性训练也可以算无用武之地,**非稳定性训练就是让 Loss 走的更好,而 Test 标签则是人为的让 Loss 走的更好**.
在追求高精度的 YOLO-X 的 Loss 中,Z-AI 提供了中断训练机制:例如训练到第三天,直接人为的强制中断.

YOLO 体系=监控盒子技术祖坟

世界上几乎所有的 AI 盒子,一律使用 YOLO,因为 YOLO 建模的条件很容易满足,只需要有数据,就能建模,接下来的工作,就是把模型移植到别的硬件平台,然后,给出可应用的 web api 这类接口.

市面上常见的违法摆摊,电瓶车违规占道,生活垃圾乱堆,这类 AI 业务几乎都使用 YOLO 体系.具体的选择大致是,如果 AIOT 算力偏低,就选 YOLOv3,偏高的版本,如 gpu 服务器选 YOLOv5.而在技术移植和平台层面,AIOT 盒子大都会走 tensorflow-lite,onnx-lite 这两个框架,有钱公司会选择含有授权的 deepmind 框架.

使用 YOLO 体系骗钱的项目很多,很多项目负责人从开始就是想着如何少劳多得,对于真正的建模,Z-AI 认为必须从机理层面讲述原理,而不能让大家去追求结果,只有这样才能真正的做好建模.一心追求结果,而不顾方法,这是只懂皮毛,半灌水,未来必然会被咔嚓,这经不起时间洗礼.

.AI_Set/.imgDataset(样本集)

.AI_Set 与.imgDataset 都是样本集,数据内容基本相同,唯一的格式差异是.AI_Set 文件会包含几何数据,这些几何数据主要用于表达 Segmentatiion 光栅的结构,而.imgDataset 会把几何数据转换成 Segmentation 光栅.

在 Z-AI 的训练体系中,样本集数据以.imgDataset 为主,但这并不是直接给 .imgDataset+训练参数来执行训练,而是把.imgDataset+训练参数放到一个.OX 的文件包中,然后再训练.

在 Z-AI 的程序体系中 .AI_Set 与.imgDataset 是两种格式支持库,它们分别走不同的数据驱动库,而在 Model Builder 工具中 .AI_Set 与.imgDataset 是弱化区分,两种格式都可以被直接打开和编辑.

.AI_Set 与.imgDataset 在底层都使用 DFE 技术的数据结构引擎,DFE 特点是高速读写结构.

对训练检测器来说数据量通常都不会很大,因此建模数据会以单个.imgDataset 作为输入样本足够使用.

.ImgMat, (图矩, ImageMatrix)

.ImgMat 是.imgDataset 的样本群格式,翻译成中文就是简称图矩.

图矩主要解决分类器的离散问题,因为分类器种类繁多,数据量大,放到一个.imgDataset 往往很难维护和管理,因此需要无穷尽的.imgDataset 堆砌机制才能解决分类器的数据问题.图矩因此诞生.

在 Z-AI 训练体系,可以是.OX 包.ImgMat 包.imgDataset 的结构,也可以是.OX 包.imgDataset 结构.

Z-AI 的 Large-Scale 技术训练体系只支持.ImgMat 格式,Large-Scale 不支持.imgDataset

.OX(训练包)

.OX 是 ZDB1 技术体系的文件包,与.mpq 格式类似,主要用户存放文件.

Z-AI 的所有训练都是.OX 格式进行交换,数据机制是 IO,既输入一个.OX 包,然后待训练完成再输出一个.OX 包,输入的.OX 包可以包含.imgDataset 和.ImgMat,并且训练的超参数也会包在.OX 里面文件名默认是 param.txt

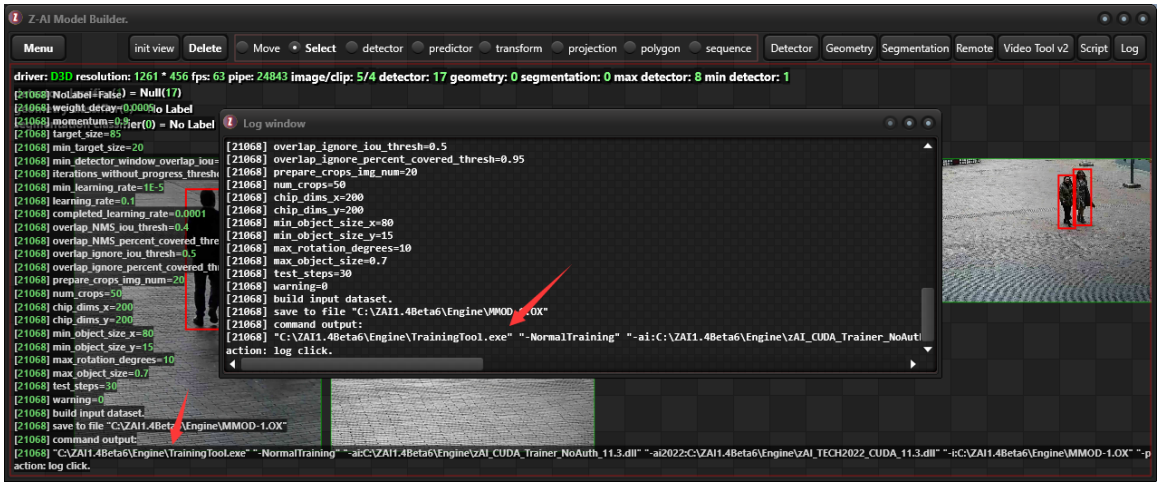
.OX 训练包在训练完成后会包含输入副本,这些副本大都是 input.imgDataset/input.ImgMat 这类文件命名,同时训练完成的模型也会包含在.OX 训练包.这样干的目的是方便管理追溯每次训练的数据.

简单来理解.OX 训练包:Z-AI 的所有训练每次都会只有一个.OX 文件输入,以及另一个.OX 文件输出.而繁琐复杂的各种训练细节数据,模型,同步文件,训练日志,Loss 值,excel 统计步数,都在.OX 包里面.而不是每次训练产生一大堆文件让用户自己去管理.

命令行训练

训练器自 1.2 开始就已经不再是 console window 的工作方式了,因为在训练中会需要有交互操作,例如人为中断,查看 Loss,散点分析这类功能.

Z-AI 所有的自动训练任务,例如在 Model Builder/ImgMat/TrainingService 这类 app 启动训练,都是用过 shell 命令方式启动 TrainingTool 达成,通常只要留意 Log,会看到 shell 传递细节



TrainingTool 训练参数

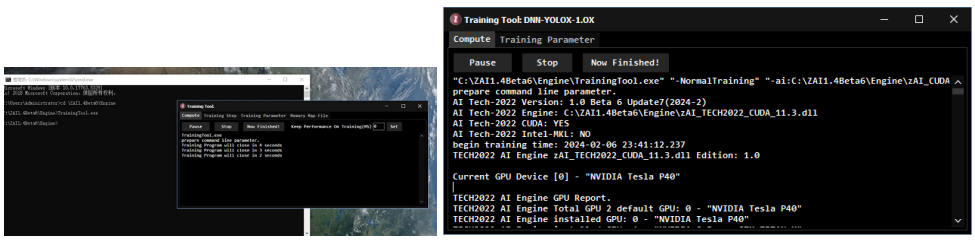
在早期版本训练流程都是直接在图矩工具,标注工具里面直接跑,当经历过几次升级以后,以独立的 TrainingTool 存在,通过命令行和做运行传递.

TrainingTool 可以使用指的 AI 计算引擎进行训练,甚至可以指定每一次训练的模型所属的产品体系,TrainingTool 的启动参数繁多且复杂,本文档主要针对建模检测器,这里不多介绍,了解 TrainingTool 启动参数建议直接看代码.

Z-AI 走的多计算引擎模式,例如训练 DNN-OD 使用传统引擎,训练 YOLO-X 会使用 Tech2022 引擎,内部的训练流程也走多引擎路线.

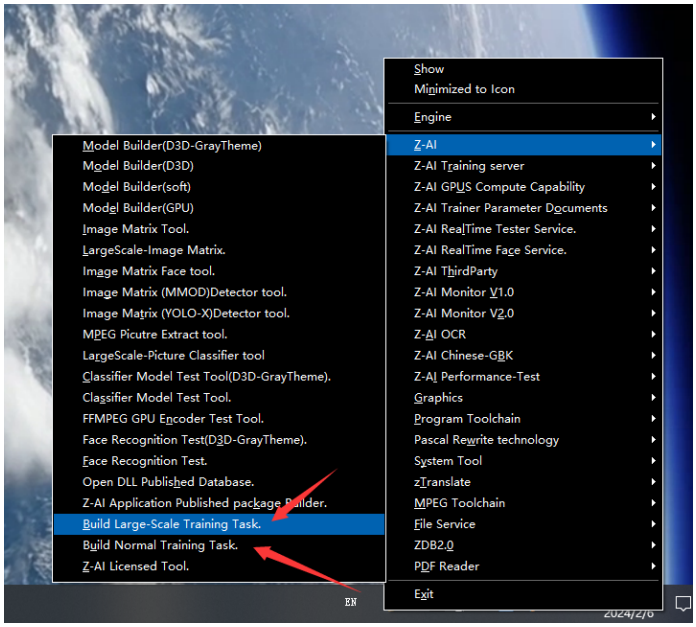
直接执行 TrainingTool 会弹出左边个小窗,不会做任何事,5 秒以后会自动关闭.而当给出训练数据这类参数以后,会出现右边小窗,这时候,它可以操作暂停,中断训练.

提示:对样本集选择不同的 gpu 进行训练,也是通过 TrainingTool 参数运行.



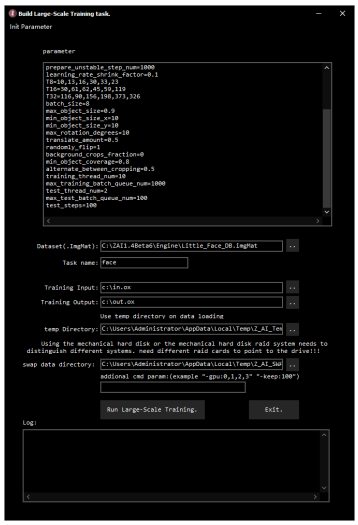
训练小工具

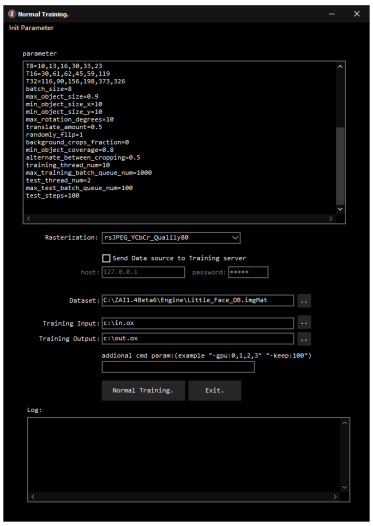
箭头所指是独立的训练小工具,



下列左图是 Large-scale 训练,只能是图矩数据集(.ImgMat),右图是 normal 训练,样本数据集可以是.AI_Set 也可以是.imgDataset.

训练小工具多用于一次性训练,与早期命令行不同的是小工具可以让 TrainingTool 启动参数自动化,相当于自己动手做训练包数据.这是一个设计上的问题,Z-AI 建模的训练体系算小具规模,用开放的方式设计模型训练体系,给出小工具是让训练可以有更多的自由启动方式,而不是死板的每次都从建模+标注工具去启动训练.

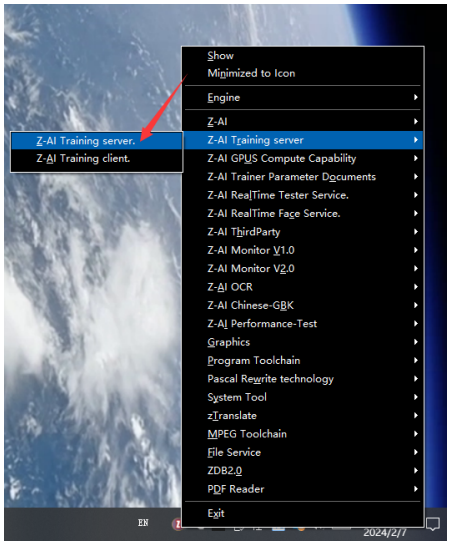


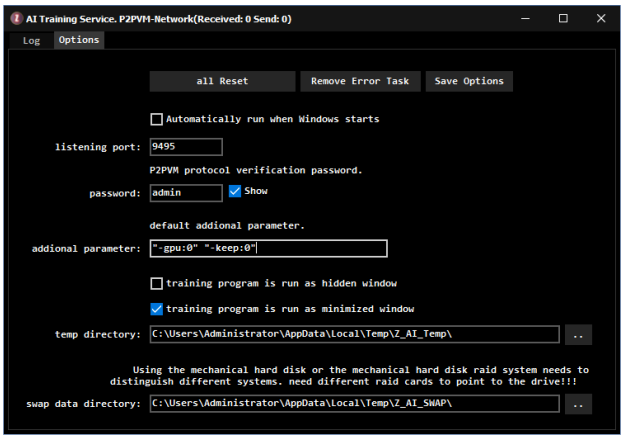


服务器训练

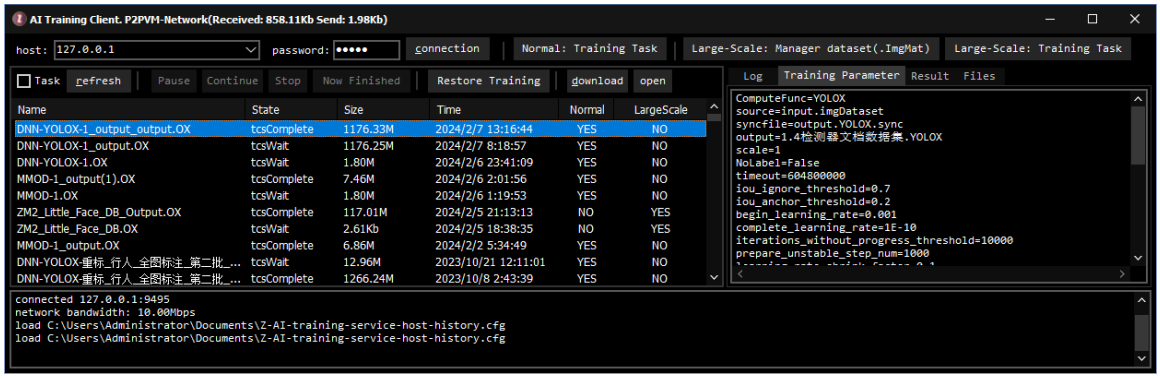
服务器训练会记录每一个模型的副本,这些副本包括,发送给服务器的副本,训练完成的副本,每次训练被中断的副本.

Z-AI 的服务器训练体系分为训练服务和训练客户端.可以通过工具链菜单启动,下列有图是训练服务器启动后的参数页.





下图为训练客户端,每个训练任务都是各自的文件数据包,这些任务总是输入是个独立任务,训练完成后输出又是另一个独立任务.因此使用服务器训练总是发给服务器训练以后,过段时间再回来看结果.训练客户端的中继训练是一键式的,具体细节参考 AI 超算系统使用指南

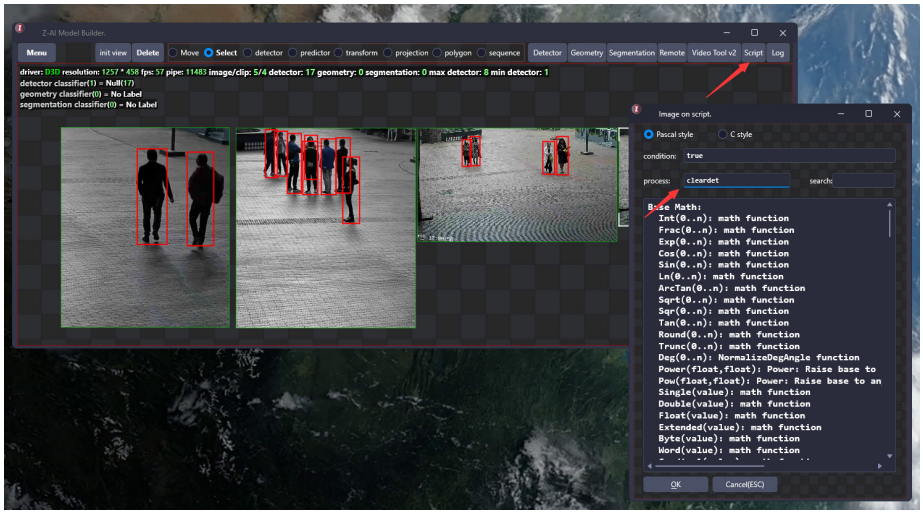


当通过 Model Builder 这类工具往服务器发送任务时,会在临时目录(%temp%)先生成一个样本库,然后启动训练客户端,这时候客户端会自动上传到服务器并训练,当上传完成后,再删除临时样本库.

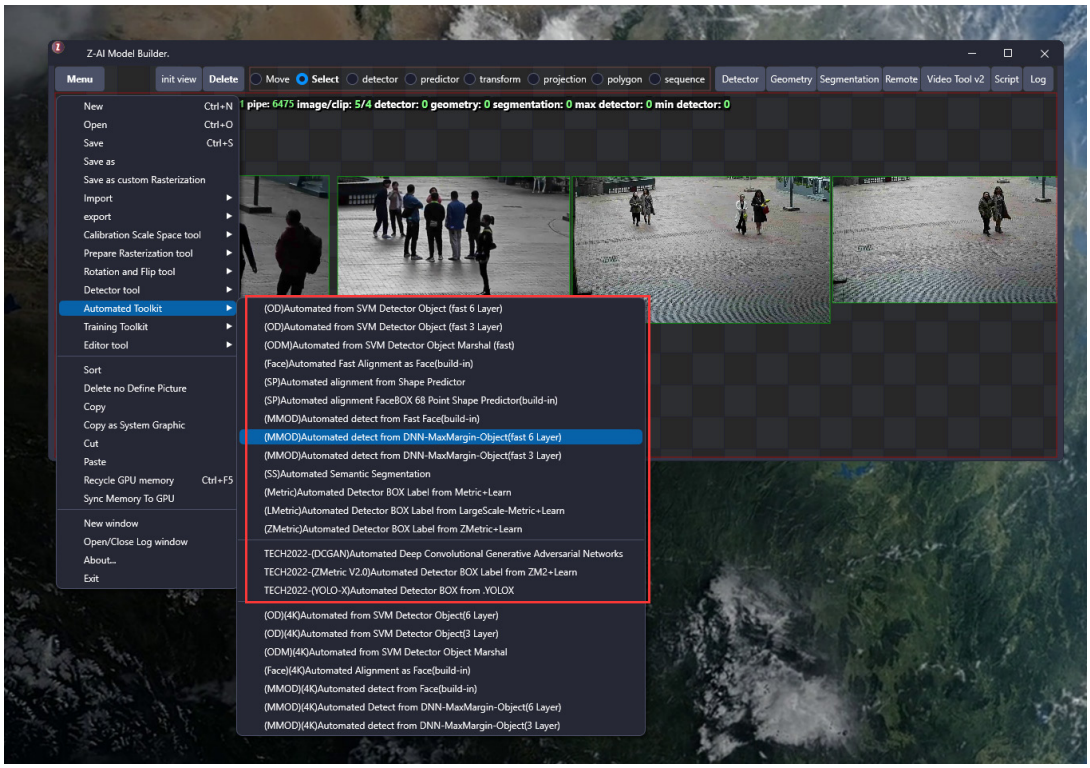
测试模型

最直观的检测器测试是在 Model Builder 标注工具使用自动标注

首先打开样本库,然后点开脚本,执行 cleardet,把框框圈都清理了



然后选择自动标注,通常来说,测试样本应该使用与源样本不同的样本+源样本,这样可以测试出差异性,通常来说,训练模型中的 loss=原样本的测试准确度,这是使用独立样本测试原因.



将 AI 模型合并到业务系统

通常我把业务系统称之为 AI Runtime,AI 本身它只是一个技术点,在真实项目中,需要有一种应用技术点的运行载体,这种载体就是 AI Runtime.
AI Runtime 更像是一种框架,在真实的 AI 项目中,我总结了一下

需要具有可视化能力

让人可以直观看和感受,在交付环节,可以提升认可度,另一方面,可视化会很方便传播,录点视频截截图,圈子就会知道你在做 AI,传播是一种现代化的社交方式,先传播再社交免走弯路.
在可视化环节,Z-AI 有 DrawEngine 和 soft Rasterization,几何库这类技术体系,这对画图来非常重要.没有这些技术体系,是无法做可视化开发的,画图中会有大量的时间,空间,坐标系,数据体系的要求,直接用 opengl,dx,或则 fmx 裸绘是行不通的,这会过于消耗时间,可视化很注重计算,并不是画出来.而计算需要有计算函数工具箱,项目执行期间,从零起步做视觉是行不通的.

需要具有 gpu 的算力调度能力

因为 Gpu 是 AI 项目中的一个投资点,所有的 AI 项目都会要求 GPU 设备需要物尽其用,可以花 5 万买台服务器跑业务就没有必要花 10 万买两台.
例如在 AI 监控系统,每一路监控每秒 25-50 帧不等(根据参数调节),100 路监控可以用廉价的低配服务器也可以用高配服务器,如果使用低配就会涉及 gpu 优化,算力调节这些要求,最后让服务器跑出极限(内存,北桥,cpu,gpu,这几样东西的占用率达到 90%以上就是极限).而高配服务器则可以让系统集成更省心,直接扔给服务器就行了,不用考虑优化,例如闲置的大模型服务器之类.
Z-AI 各种模型对算力要求都不一样,某些对北桥带宽高,某些对 gpu 算力要求高,某些对 cpu 要求高,极限挖掘大都会使用脚本来干:针对服务器配置,定做计算脚本,这会比直接跑去改流程代码更有效率.

业务输出能力

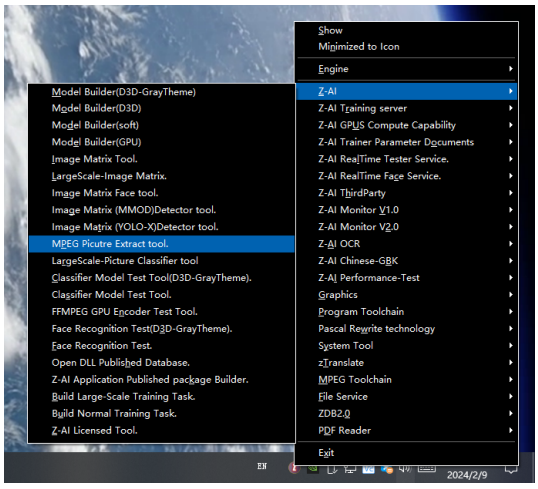
AI Runtime 面向业务输出,基本都是走的服务器化,存数据库,http-post,这类路线,所有的 AI 识别结果放在数据库中,同时也会有一些程序化的流程,例如识别到目标,拿 http 通知一下,例如用户 app 层定期对 AI 识别数据库统计一下,挖掘点东西.
业务输出能力,其实就是把 AI 识别给数据化一下,很多用户喜欢把整个项目全部仍出来,而真实的工作划分是不可以这样干的,必须区分 AI 的工作,以及业务层的工作,因为业务层有变化属性,不可以把 AI 与业务混淆,人的思路变化会导致设计变化,这种变化将会导致时间延迟,而长时间的项目履行会发生许多外部事件,这会发生许多难以预料的事情.总之做 AI 业务的关键点是把识别结果被数据化.
关于产品的变化,人都是不断的思考,学习,接触,提升的,产品的变化其实会一直遵守人的变化,作者对此的解决办法是走的想象力路线:针对可能出现的变化,给出不同的 AI 业务模型,这类模型例如,监控系统模型,手机 app 模型,iot app 模型,这类模型是在听取了众多的思路后,从普适性角度慢慢摸索出来的,主要拿给用户自己换皮,改代码用,但这不是直接跑业务来变现,大约会是 80-90%完成度的成品,需要用户自己去做一下外围的开发.如果没有业务模型,那就需要走定做开发路线,这是企业范围.在多数时候,会建议 app 层由用户自己解决,作者只负责 AI 数据化业务.

从视频导入图片来建模

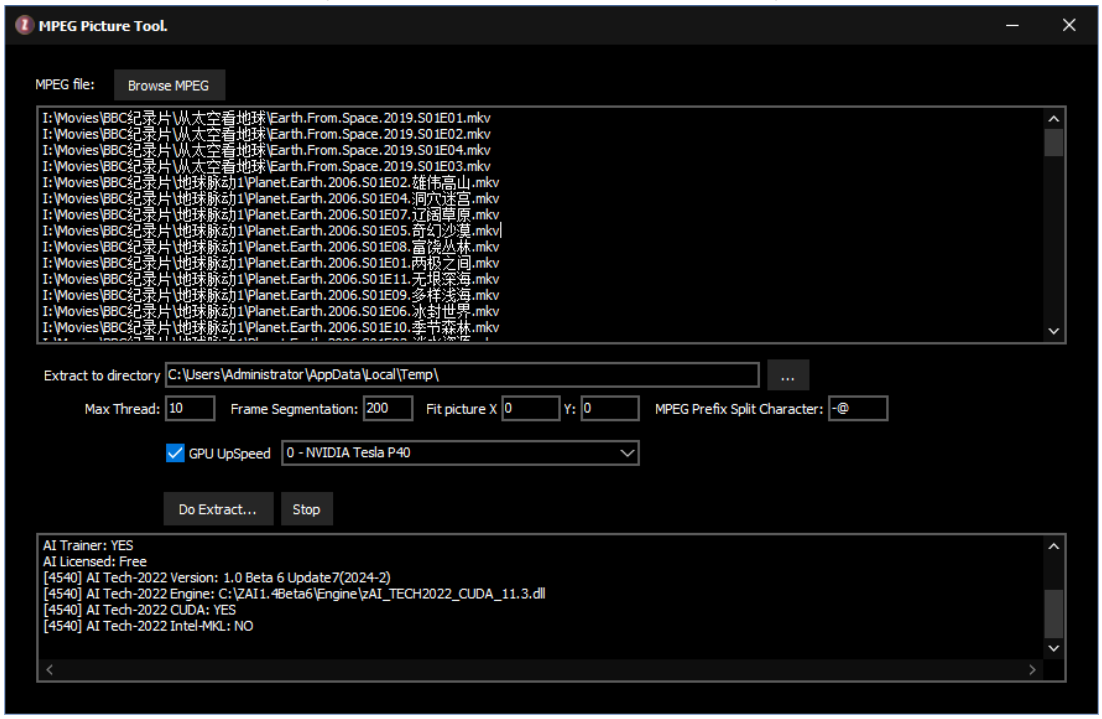
导入图片需要区分局部视频导入和大规模导入,局部导入就是从数量不多的短视频或剪辑导图片,大规模导入是针对从 BBC,科教片,监控大数据这类庞大的原始数据中先提取图片,然后再导入图矩数据库.

从视频大规模提取图片

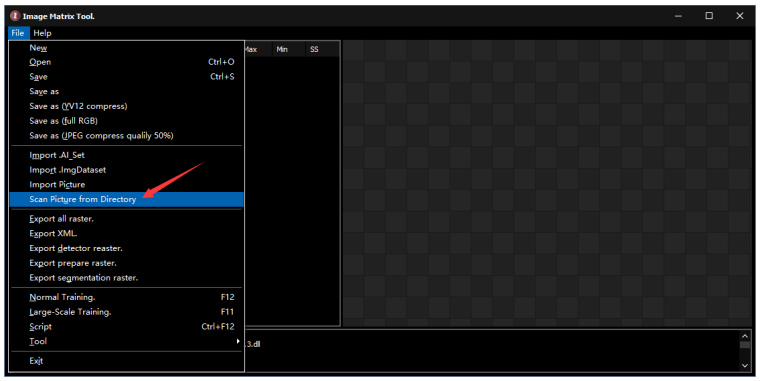
首先打开工具链菜单指向视频图片提取工具



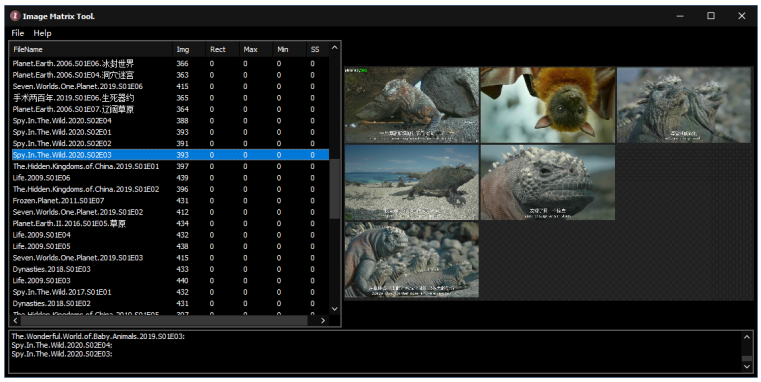
提取工具可以使用 GPU 加速,它会根据视频前缀或视频名把图片提取到对应的子目录.
提取工具一次只能使用一个 gpu 加速,如果要多 gpu,就开多个提取工具指定不同的 gpu.
提取工具可以指定帧间隔,下图为每 200 帧间隔提取一张图片,并且不拟合图片尺寸.



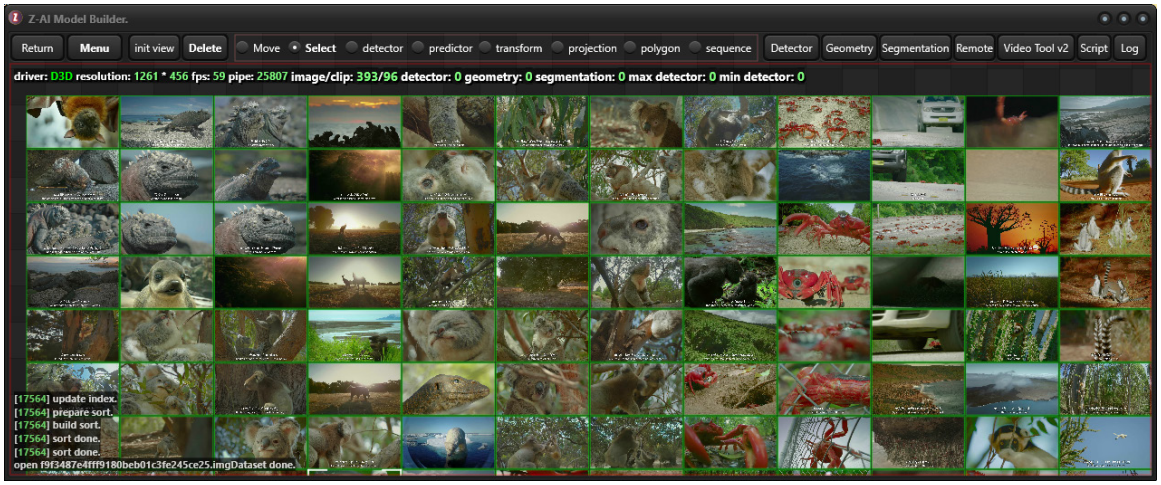
然后再用图矩导入目录的功能



图片提取出来以后,可以直接图矩里面双击打开 Model Builder,



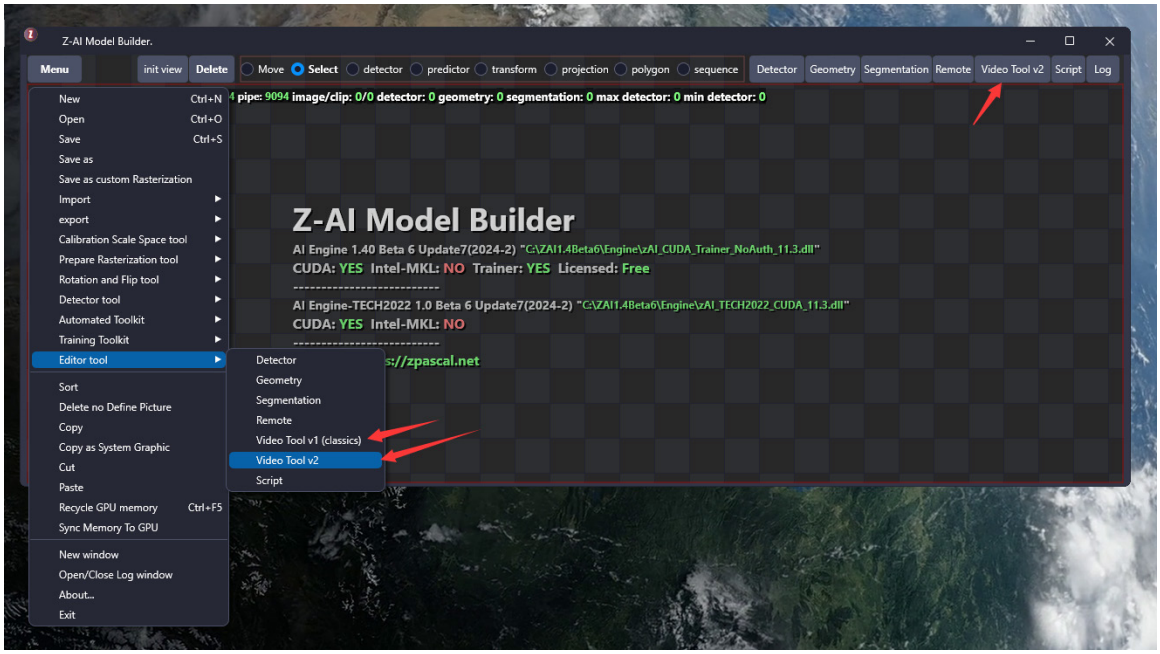
然后在 Model Builder 用过 ctrl+c/ctrl+v 把需要的样本图片考到自己的样本库里面



如此反复操作,也可以直接使用系统资源管理器的图片浏览功能 ctrl+c 图片.

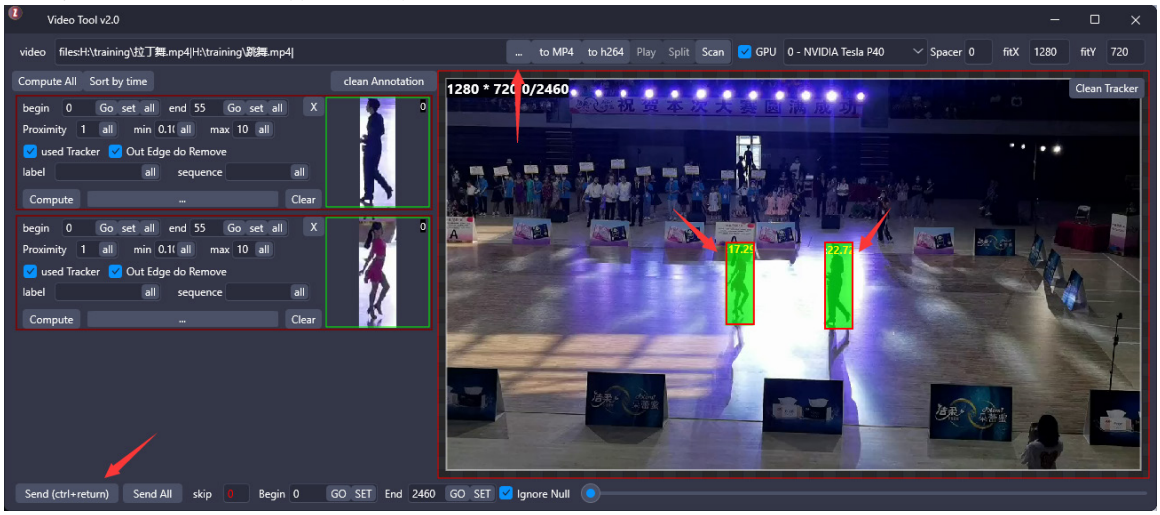
在 Model Builder 导入视频图片

在标注工具有 2 个视频工具版本,Video Tool V1(classic)只能单标 Tracker, Video Tool V2 可以多标 Tracker,同时 V2 支持指定 gpu 解码和热键传递.



在 V2 工具可以一次选择多个视频,然后导入进来,ctrl+return 传递.在 Model Builder 的主力视频导入工具也是 V2.

在 V2 工具拉框为 Tracker 功能,多标功能是做肢体跟踪,行为跟踪,运动跟踪的建模功能,在目标分类器和行为识别会做详细介绍.



全文完.

By.qq600585