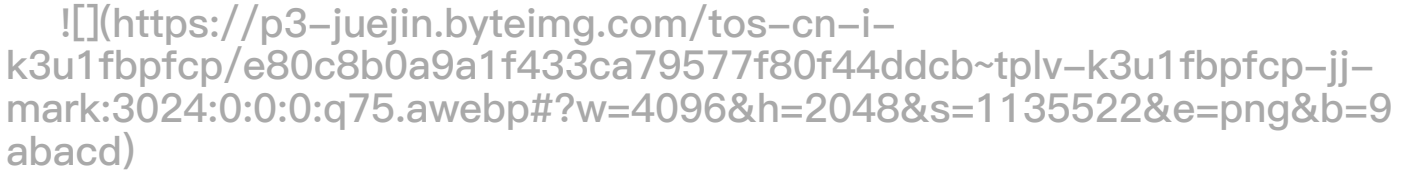
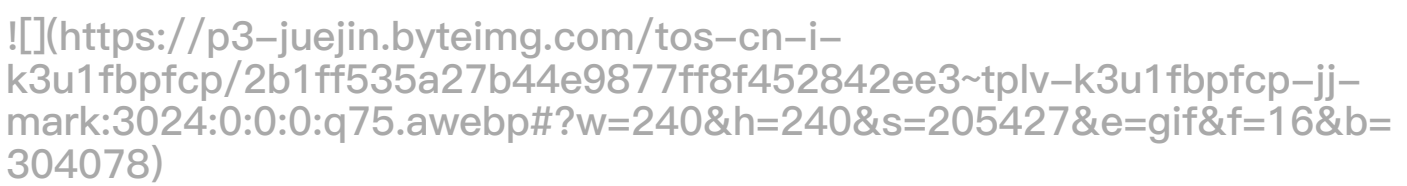
你的第一款开源视频分析框架

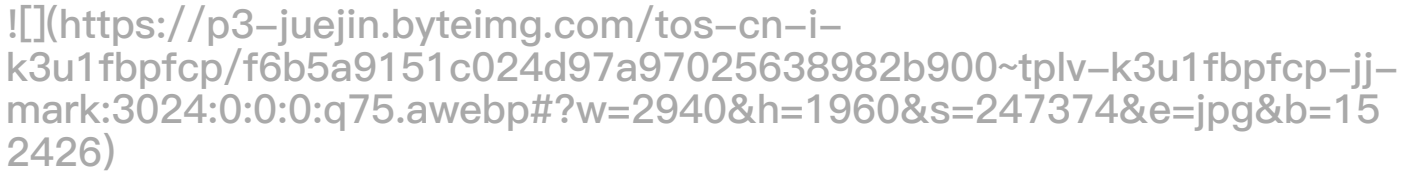
==========



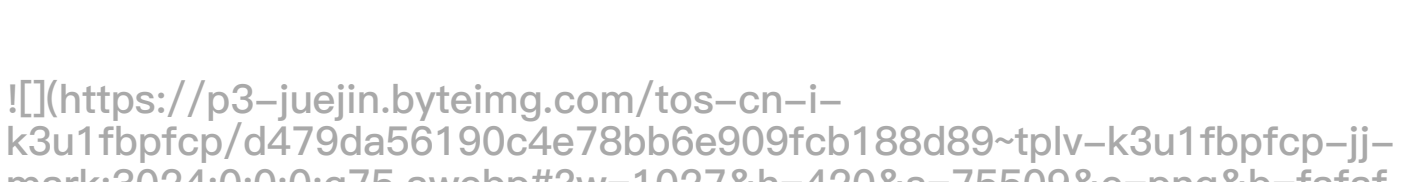
现在，刷视频已经成为我们生活中的一部分，而且很容易一看就停不下来。你有没有好奇过，它是如何在海量的视频里，找到让你感兴趣的视频？又是如何让你可以通过关键字，搜索到与之相关的视频内容的呢？这一切都离不开计算机对视频内容的分析和理解。



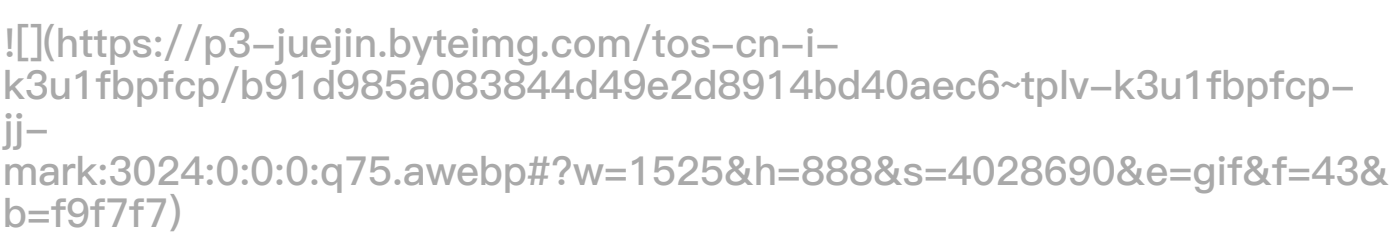
计算机是如何“看懂”海量视频的呢？**视频本质上是一系列连续的图像帧，按照一定的帧率播放，从而形成连续的动态效果**。因此，计算机分析视频的基本原理就是：解码（视频转图片）-> 分析/推理（AI 算法）-> 编码（结果呈现）



尽管这看起来就寥寥几步，但其中涉及许多技术细节和复杂的算法。比如，如何将训练好的 AI 图像算法模型，快速部署落地到实际应用场景中呢？对于没有接触过计算机视觉（Computer Vision，后简称 CV）的程序员，或是纯搞算法的算法工程师，要实现+落地 AI 视频分析相关功能可能会有点难度。然而，随着视频在日常生活中的普及和应用越来越广泛，处理和分析视频类数据的需求也在逐渐增加。



因此，今天 HelloGitHub 带来了一款开源的视频分析/结构化框架——VideoPipe，旨在让开发视频分析应用像使用 Django 写 Web 一样方便。**VideoPipe 独创的管道可视化显示，让每一步的处理状态都可以一目了然**。该框架能够轻松集成各种 CV 领域的模型，通过即插即用的方式轻松实现 AI 加持下的视频分析，适用于视频结构化、图片搜索、人脸识别、安防领域的行为分析（车牌识别、交通事故检测）等场景。



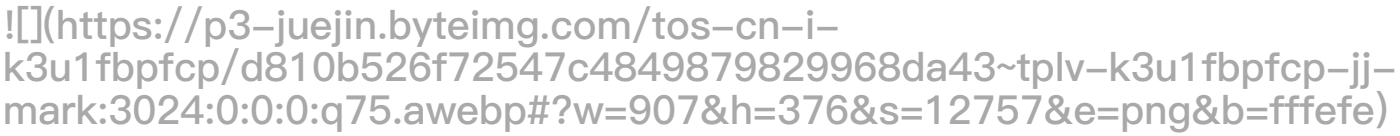
> GitHub 地址：[github.com/sherlockcho...](http://cxyroad.com/”https://github.com/sherlockchou86/VideoPipe”)

下面，让我们跟着该项目的作者（周智）一起来了解、上手 VideoPipe，然后深入其内部学习更多的技术细节。

一、介绍

VideoPipe 这是一个用于视频分析和结构化的框架，采用 C++ 编写、依赖少、易上手。它就像一个管道每个节点相互独立可自行搭配，用来构建不同类型的视频分析管道，适用于视频结构化、图片搜索、人脸识别、安防领域的行为分析（如交通事件检测）等场景。

你只需准备好模型并了解如何解析其输出即可，推理可以基于不同的后端实现，如 OpenCV::DNN（默认）、TensorRT、PaddleInference、ONNXRuntime 等，任何你喜欢的都可以。



通过上面的 VideoPipe 工作示意图，可以发现它提供了以下功能：

* 流读取/推送：支持主流的视频流协议，如 udp、rtsp、rtmp、文件。
* 视频解码/编码：支持基于 OpenCV/GStreamer 的视频和图片解/编码（支持硬件加速）。
* 基于深度学习的算法推理：支持基于深度学习算法的多级推理，例如目标检测、图像分类、特征提取。
* 目标跟踪：支持目标追踪，例如 IOU、SORT 跟踪算法等。
* 行为分析（BA）：支持基于跟踪的行为分析，例如越线、停车、违章等交通判断。
* 数据代理：支持将结构化数据（json/xml/自定义格式）以 kafka/Sokcet 等方式推送到云端、文件或其他
第三方平台。
* 录制：支持特定时间段的视频录制，特定帧的截图。
* 屏幕显示（OSD）：支持将模型输出结果绘制到帧上。

对比功能类似、耳熟能详的 DeepStream（英伟达）和 mxVision（华为）框架，**VideoPipe 更易于使用和调试、具备更好的可移植性，它完全由原生 C++ 编写，仅依赖于少量主流的第三方模块（如 OpenCV）**。同时提供了可视化管道，框架的运行状态会自动在屏幕上刷新，包括管道中每个连接点的 fps、缓存大小、延迟等信息，你可以根据这些运行信息快速定位处理时的瓶颈所在。

名称	是否开源	学习门槛	适用平台	性能	三方依赖
DeepStream	否	高	仅限英伟达	高	多
mxVision	否	高	仅限华为	高	多
VideoPipe	是	低	不限平台	中	少

二、快速上手

VideoPipe 对机器硬件没有要求，仅用 CPU 都可以运行，不需要额外的加速卡。而且项目中还提供了丰富的示例代码，下面让我们通过运行一个简单的「人脸识别」示例，快速上手该框架。

```
/*
* 名称：1-1-N sample
* 完整代码位于：samples/1-1-N_sample.cpp
* 功能说明：1个视频输入，1个视频分析任务（人脸检测和识别），2个输出（屏幕输出/RTMP推流输出）
```

```
* 注意：模型和视频文件需要自行准备
*/

int main() {
    VP_SET_LOG_INCLUDE_CODE_LOCATION(false);
    VP_SET_LOG_INCLUDE_THREAD_ID(false);
    VP_LOGGER_INIT();

    // 1、创建节点
    // 视频获取 Node
    auto file_src_0 =
std::make_shared<vp_nodes::vp_file_src_node>("file_src_0", 0,
"./test_video/10.mp4", 0.6);
    // 2、模型推理 Node
    // 一级推理：人脸检测
    auto yunet_face_detector_0 =
std::make_shared<vp_nodes::vp_yunet_face_detector_node>("yunet_face
_detector_0", "./models/face/face_detection_yunet_2022mar.onnx");
    // 二级推理：人脸识别
    auto sface_face_encoder_0 =
std::make_shared<vp_nodes::vp_sface_feature_encoder_node>("sface_fa
ce_encoder_0", "./models/face/face_recognition_sface_2021dec.onnx");
    // 3、OSD Node
    // 处理结果绘制到帧上
    auto osd_0 =
std::make_shared<vp_nodes::vp_face_osd_node_v2>("osd_0");
    // 屏幕展示
    auto screen_des_0 =
std::make_shared<vp_nodes::vp_screen_des_node>("screen_des_0", 0);
    // 推流展示
    auto rtmp_des_0 =
std::make_shared<vp_nodes::vp_rtmp_des_node>("rtmp_des_0", 0,
"rtmp://192.168.77.60/live/10000");

    // 构建管道，将节点的处理结果关联起来
    yunet_face_detector_0->attach_to({file_src_0});
    sface_face_encoder_0->attach_to({yunet_face_detector_0});
    osd_0->attach_to({sface_face_encoder_0});

    // 管道自动拆分，通过屏幕/推流输出结果
    screen_des_0->attach_to({osd_0});
    rtmp_des_0->attach_to({osd_0});

    // 启动管道
    file_src_0->start();

    // 可视化管道
    vp_utils::vp_analysis_board board({file_src_0});
```

```
    board.display();
}
```

通过阅读上面的示例代码，可以发现 **VideoPipe 框架将视频分析/处理的步骤，抽象成了一个管道（pipe），每一步的处理都是管道中的一个节点（Node）**，处理流程如下：

1. 视频读取 Node：完成读取视频和解码的工作
2. 模型推理 Node：分为人脸检测和人脸识别两个模型
3. OSD Node：将模型输出的处理结果绘制到帧上
4. 构建管道：将上述节点依次连接，并将结果分成屏幕输出和推流输出，
5. 启动：启动程序，并展示管道的运行情况

代码运行后，会出现上面的 3 个画面。它们分别是管道运行状态图（状态自动刷新）、屏幕显示结果（GUI）、播放器显示结果（RTMP），至此就算上手 VideoPipe 了！

三、技术原理

接下来，将详细介绍 VideoPipe 框架实现的技术原理和细节，干货来啦！在深入了解 VideoPipe 框架技术细节之前，我们需要先弄清楚视频的整体处理流程。

3.1 视频结构化应用的核心环节

视频结构化是将非结构化数据（视频/图片）转换为结构化数据的过程。非结构化数据通常包括：视频、图像、音频、自然语言文本，而结构化数据主要包括诸如 JSON、XML 或数据库中的数据表等，这些数据可以直接由机器（程序）处理。具体到视频（含图片，下同）结构化的过程，主要涉及以下核心部分：

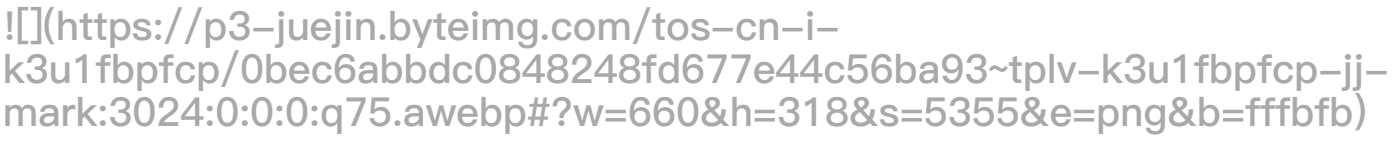
* 读取流：从网络或本地机器获取视频流。

* 解码：将字节流解码为帧，因为算法只能作用于图像。
* 推理：对图像进行深度学习推理，如检测、分类或特征提取。
* 跟踪：跟踪视频中的目标。
* 行为分析/逻辑处理：分析目标的轨迹、属性。
* OSD：在图像上显示结果，用于调试或得到直观效果。
* 消息代理：将结构化数据推送到外部，供业务平台使用。
* 编码：对包含结果的帧进行编码，以便传输、存储。
* 推送流：将字节流推送到外部或直接保存

上述每个环节对应 VideoPipe 中的一种插件类型，即代码中的 Node 对象。下面我们将逐一讲解 VideoPipe 的 Node、数据流、钩子的技术细节和实现。

3.2 Node

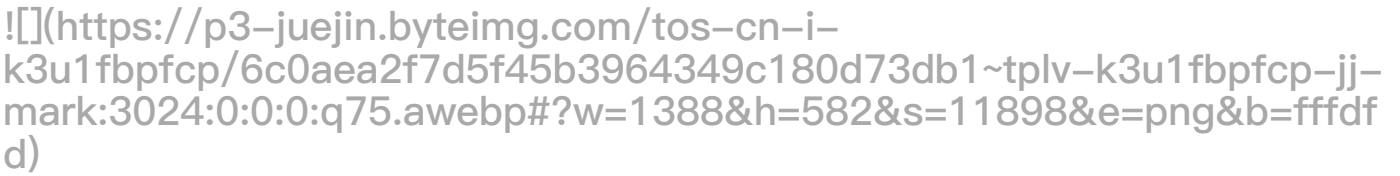
VideoPipe 中的每个 Node 负责一种任务（严格遵循单一职责原则），例如解码或推理。我们可以将许多节点串在一起构建成管道，并让视频数据流经整个管道。每个 Node 内部都有两个队列，一个用于缓存上游节点推送的数据，另一个用于缓存等待被推送到下游节点的数据。我们可以在两个队列之间编写逻辑代码，这是典型的生产者-消费者模式。



VideoPipe 中有三种类型的节点，分别是：

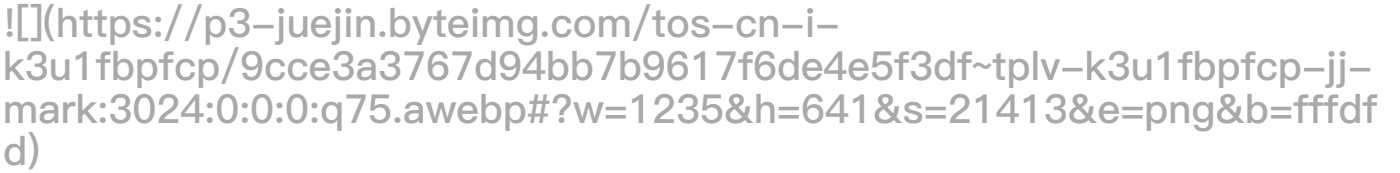
1. SRC节点：源节点，数据被创建的地方（内部只有一个队列，用于缓存被推送到下游节点的数据）。
2. MID节点：中间节点，数据将在此处理。
3. DES节点：目标节点，数据消失的地方（内部只有一个队列，用于缓存来自上游节点的数据）。

每个节点本身具有合并多个上游节点和拆分成多个下游节点的能力。注意，默认情况下节点在将数据从一个节点传输到另一个节点时使用浅拷贝和等值拷贝。如果您需要深拷贝或希望按通道索引传输数据（希望数据不混淆），则在分裂点添加一个 `vp_split_node` 类型节点。



3.3 数据流

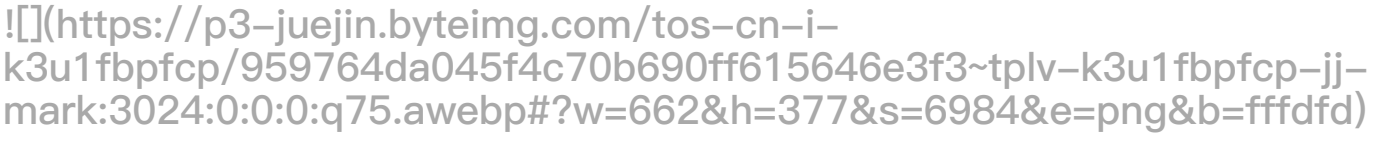
视频是一种重量级数据，因此频繁进行深拷贝会降低管道的性能。实际上，VideoPipe 中两个节点之间传递的数据默认使用智能指针，一旦数据由源节点创建，数据内容在整个管道中大多数时间不会被复制。但如果需要，我们可以指定深度拷贝模式，使用 `vp_split_node` 类型节点。



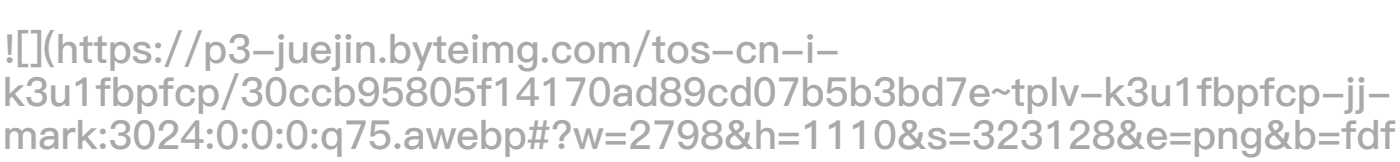
视频由连续的帧组成，因此 VideoPipe 逐帧处理这些帧，所以帧元数据中的帧索引也会连续增加。

3.4 钩子

钩子是一种机制，让主体在发生某些事件时通知检测者，VideoPipe 也支持钩子。管道触发回调函数 `std::function` 与外部代码通信，例如实时推送管道自身的 fps、延迟和其他状态信息。我们在编写回调函数内部代码时，不允许有阻塞出现，否则影响整个管道性能。



钩子有助于调试我们的应用程序，并快速找出整个管道中的瓶颈，VideoPipe 框架中自带的可视化工具 `vp_analysis_board` 就是依赖于钩子机制实现的。

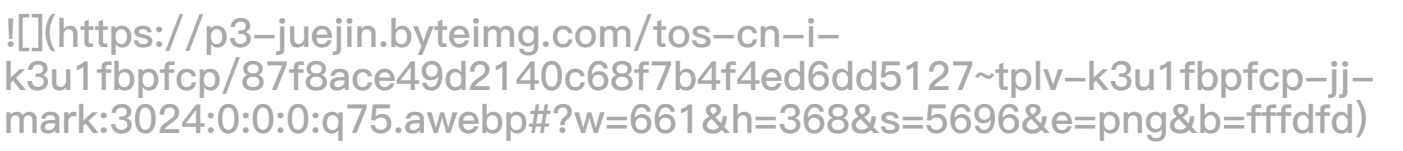


dfd)

3.5 如何实现新的 Node 类型

首先 `vp_node` 是 VideoPipe 中所有节点的基类，我们可以定义一个从 `vp_node` 派生的新节点类，并重写一些虚函数：

* `handle_frame_meta`：处理流经当前节点的帧数据。
* `handle_control_meta`：处理流经当前节点的控制指令数据。



帧数据指的是 VideoPipe 中的 `vp_frame_meta`，其中包含与帧相关的数据，如帧索引、数据缓冲区、原始宽度等等。控制指令数据指的是 VideoPipe 中的 `vp_control_meta`，其中包含与命令相关的数据，例如记录视频、记录图像等。并非所有流经当前节点的数据都应该被处理，只需要处理我们感兴趣的内容。

四、最后

目前，基于深度学习的视频分析技术的入门门槛还是比较高的，一些成熟的框架比如 DeepStream、mxVision 等，它们大多晦涩难懂、上手门槛高、对于新手不太友好。所以，我就花了两年的业余时间创建了 VideoPipe 视频分析框架，我的想法很简单就是想让**初学者能够快速了解视频分析相关技术栈，轻松地在自己机器上跑通一个人脸识别的应用**，让更多人掌握视频分析相关技术，同时搞清楚应该从哪里开始。

我深知这是一件道阻且长的事情，所以 VideoPipe 在诞生之初就是完全开源，我希望能够借助开源的力量让它“发光发热”，真正地做到降低开发视频分析应用的门槛，帮助更多的开发者进入到视频分析的领域。

> GitHub 地址：[github.com/sherlockcho...](http://cxyroad.com/”https://github.com/sherlockchou86/VideoPipe”)

最后，感谢「HelloStar 计划」提供的机会，能够让更多人了解 VideoPipe 框架。我作为开源生态的受益者，深知开源的力量和责任，此举也是希望 VideoPipe 项目能够成为一座连接对视频分析、结构化技术感兴趣的小伙伴的桥梁，能够和大家一起交流学习、共同进步、回馈开源社区！

原文链接: https://juejin.cn/post/7358687549848289331