TengineFactory

发布 *0.0.1*

OPEN AI LAB

2021 年 03 月 25 日

简介

CHAPTER 1

介绍

1.1 关于

随着人工智能的普及，深度学习算法的越来越规整，一套可以低代码并且快速落地并且有定制化解决方案的框架就是一种趋势。为了缩短算法落地周期，降低算法落地门槛是一个必然的方向。

TengineFactory 是由 **OPEN AI LAB** 自主研发的一套快速，低代码的算法落地框架。我们致力于打造一个完全开源的易用的算法落地框架，通过配置 json 文件的方式，可以以最简洁的接口来获取你所需要的结果。此外，你也可以插入你自己特有的代码，完成算法的落地工程

1.2 轻量级

- 唯一依赖库Tengine-Lite，Tengine-Lite 是框架的核心推理框架。
- x86 平台：so 大小为 1.5M 左右。

1.3 易用性

- 针对通用的前处理，后处理框架内均已包含，无须再 code。
- 如模型拥有独有的前处理、后处理，插件化的方式，可以很快的新增你的算法。

1.4 致谢

Tengine Factory 参考一下项目：

- Tengine
- TengineKit
- MNN
- protobuffer

1.5 License

Apache 2.0

CHAPTER 2

架构设计

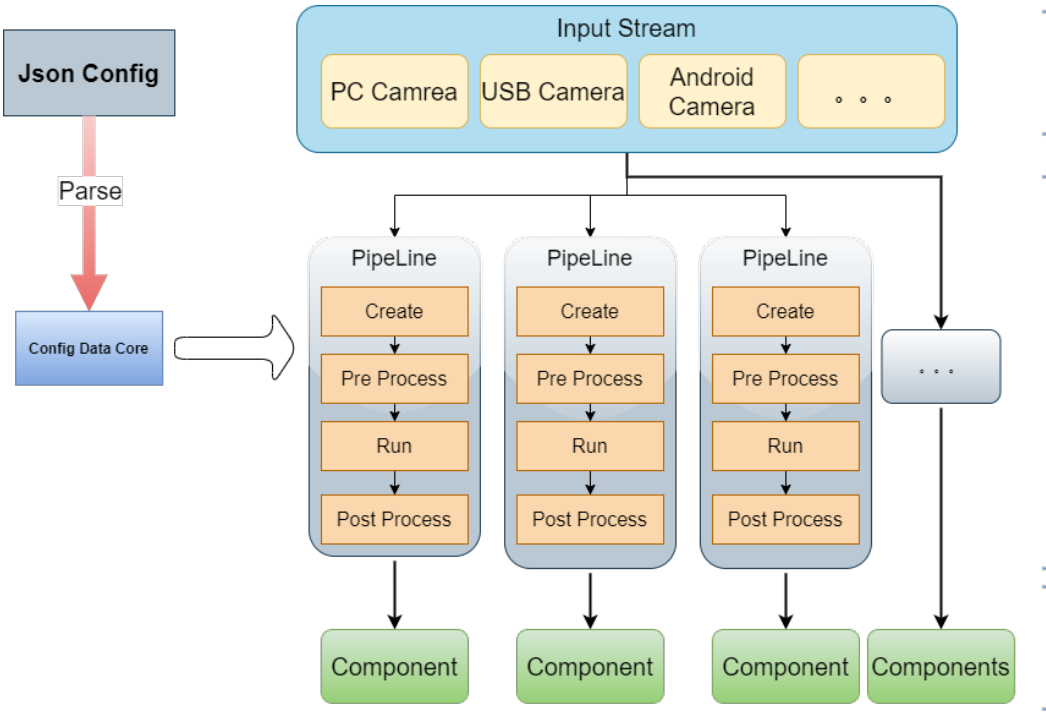
TengineFactory 采用 Json + Pipeline 的设计方式。

Json 文件用于配置算法的参数以及如何运行整体落地算法。TengineFactory 内部提供了多个前处理，后处理的方式，你可以通过配置选择你所需的前处理、后处理、输入输出，来完成整体算法的落地。

Pipeline 是管道方式运行算法的流，包含 create、preprocess、run、postprocess、destory，每个 pipeline 都继承 base，所以按照如此方式，方便插入个性化的 pipeline。我们也会补充各种统一化的算法的 pipeline 进去。

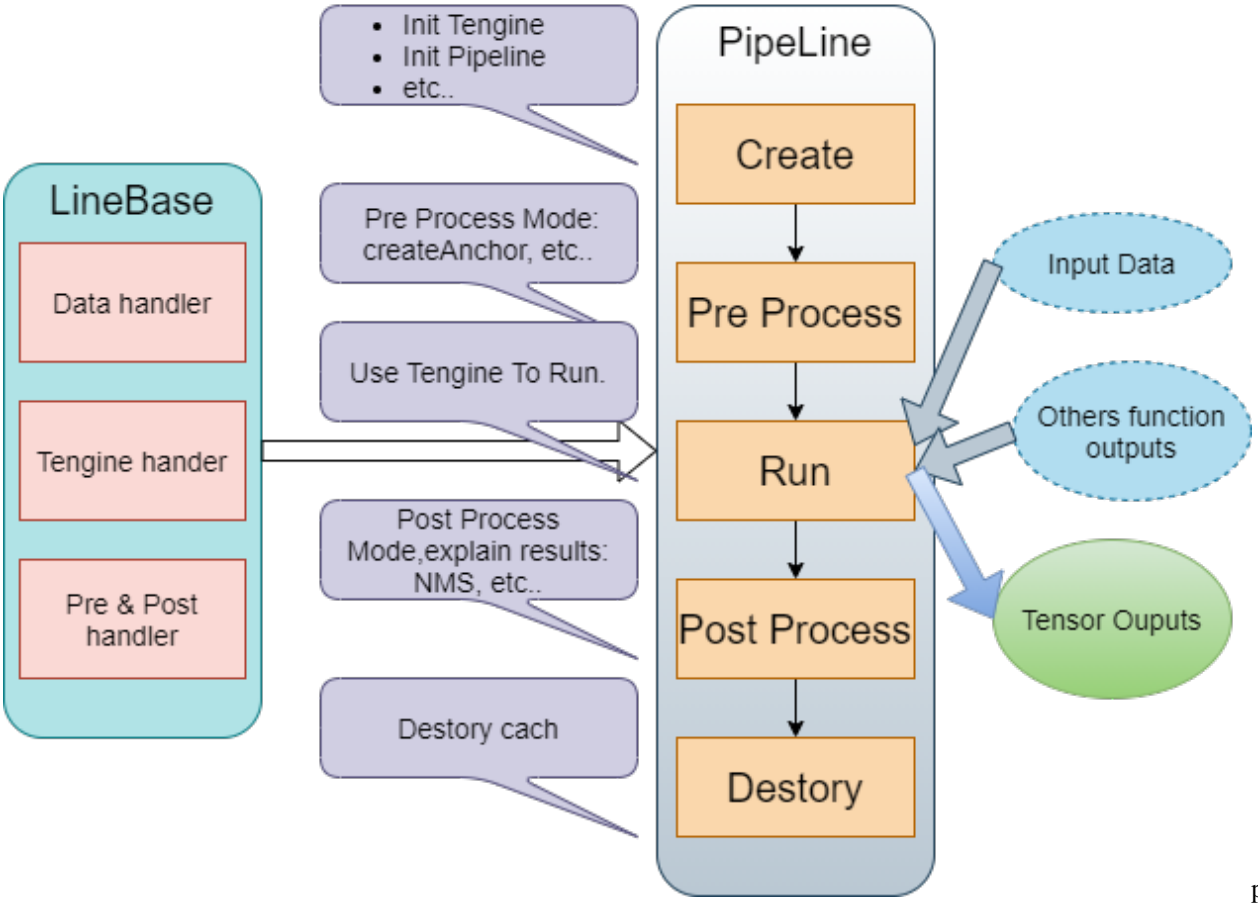
具体结构如下图：

2.1 Framework



framework

2.2 PipeLine



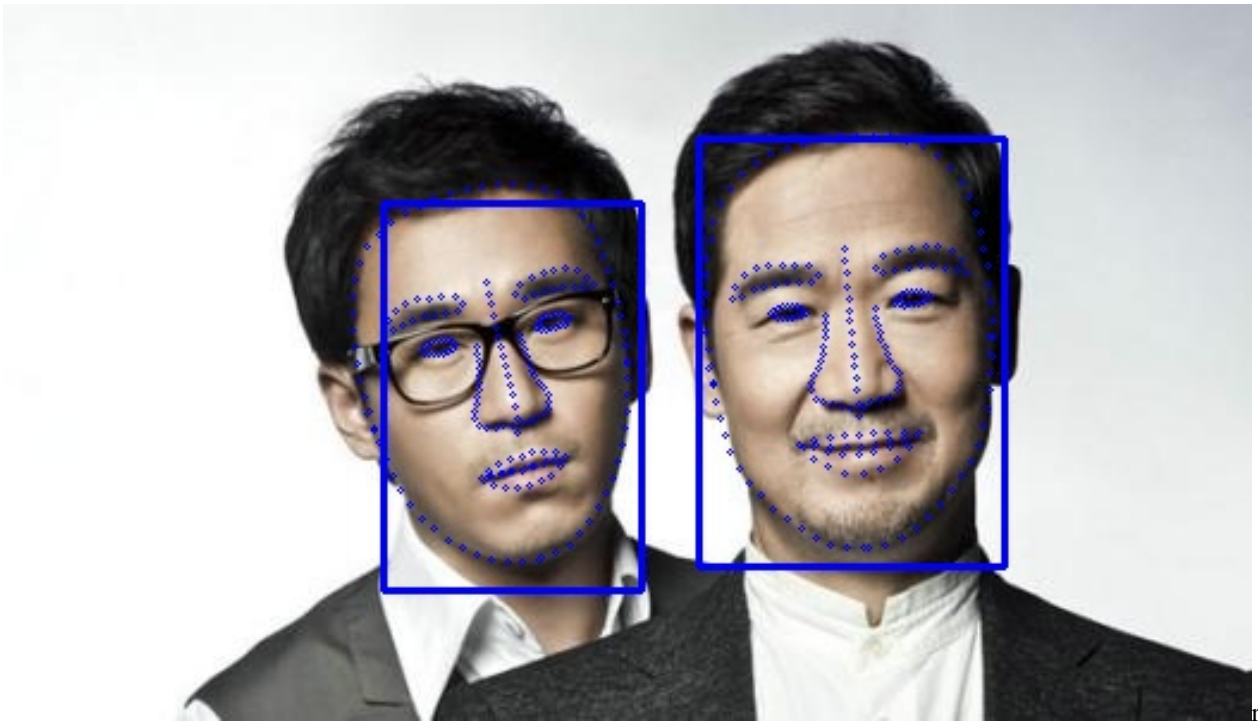
pipeline

CHAPTER 3

示例展示

3.1 人脸检测 & 关键点

原始图片：  origin

算法结果：  result

CHAPTER 4

编译 Tengine

4.1 环境要求

- opencv >= 3.4
- cmake >= 3.10
- protobuf >= 3.0
- gcc >= 4.9
- ndk >= 15c （如果需要使用 Android）

4.2 编译 Linux 版本

1. clone Tengine项目，`git clone https://github.com/OAID/Tengine.git`
2. 编译动态库。

```
cd Tengine
mkdir build && cd build
cmake .. && make -j4
make install
```

编译完成后 build/install/lib 目录会生成 `libtengine-lite.so` 文件，如下所示：

```
install
├── bin
│   ├── tm_benchmark
│   ├── tm_classification
│   └── tm_mobilenet_ssd
├── include
│   └── tengine_c_api.h
└── lib
    └── libtengine-lite.so
```

4.3 编译 Androidv7/v8 版本

下载 ndk，http://developer.android.com/ndk/downloads/index.html（可选）删除 debug 参数，缩小二进制体积。android.toolchain.cmake 这个文件可以从 $ANDROID_NDK/build/cmake 找到

```
# vim $ANDROID_NDK/build/cmake/android.toolchain.cmake
# 删除 "-g" 这行
list(APPEND ANDROID_COMPILER_FLAGS
  -g
  -DANDROID
```

配置环境：

```
export $ANDROID_NDK=<NDK-path>
```

Android v7 编译指令：

```
mkdir build-android-armv7
cd build-android-armv7
cmake -DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK/build/cmake/android.toolchain.cmake -
↪DANDROID_ABI="armeabi-v7a" -DANDROID_ARM_NEON=ON -DANDROID_PLATFORM=android-19 ..
make
make install
```

Android v8 编译指令：

```
mkdir build-android-aarch64
cd build-android-aarch64
cmake -DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK/build-android-aarch64/cmake/android.
↪toolchain.cmake -DANDROID_ABI="arm64-v8a" -DANDROID_PLATFORM=android-21 ..
make
make install
```

编译成功后，把 libtengine-lite.so 拷贝到<Tengine-Factory-path>/libs 下

CHAPTER 5

编译 TengineFactory

5.1 环境要求

- opencv >= 3.4
- cmake >= 3.10
- protobuf >= 3.0
- gcc >= 4.9
- ndk >= 15c （如果需要使用 Android）

5.2 编译 Linux 版本

1. clone TengineFactory项目，`git clone https://github.com/OAID/TengineFactory.git`
2. 编译动态库。

```
./build.sh
```

编译后目录结构，如下所示：

```
TengineFactory
├── build
|   └── libTFactory.so
```

(下页继续)

(续上页)

```
└── include
    └── TFactoryComponent.hpp
    └── TFactoryProcess.hpp
```

5.3 编译 Androidv7/v8 版本

下载 ndk，http://developer.android.com/ndk/downloads/index.html（可选）删除 debug 参数，缩小二进制体积。android.toolchain.cmake 这个文件可以从 $ANDROID_NDK/build/cmake 找到

```
# vim $ANDROID_NDK/build/cmake/android.toolchain.cmake
# 删除 "-g" 这行
list(APPEND ANDROID_COMPILER_FLAGS
  -g
  -DANDROID
```

配置环境：

```
export $ANDROID_NDK=<NDK-path>
```

Android v7 编译指令：

```
mkdir build-android-armv7
cd build-android-armv7
cmake -DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK/build/cmake/android.toolchain.cmake -
↪DANDROID_ABI="armeabi-v7a" -DANDROID_ARM_NEON=ON -DANDROID_PLATFORM=android-19 ..
make
```

Android v8 编译指令：

```
mkdir build-android-aarch64
cd build-android-aarch64
cmake -DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK/build-android-aarch64/cmake/android.
↪toolchain.cmake -DANDROID_ABI="arm64-v8a" -DANDROID_PLATFORM=android-21 ..
make
```

CHAPTER 6

执行算法

```
/**
 * @brief Create Tengine Factory handler.
 * @return handler.
 */
static TFactoryProcess* create();
```

必须先进行 create 操作，api 都是基于这个 handler 的。建议只创建一次多次使用。

6.1 初始化

```
/**
 * @brief Initialize Tengine Factory.
 * @param jsonPath json config file path.
 * @return None.
 */
void init(const char* jsonPath);
```

6.2 运行

有两种方式进行输入的传入：

- 通过 json 配置，配置文件路径或者视频流。参数为第几张图片。

```
/**
 * @brief Run Tengine Factory by read json file sources.
 * @param index image file index, if is video you should not set the index.
 * @return None.
 */
void run(int index = -1);
```

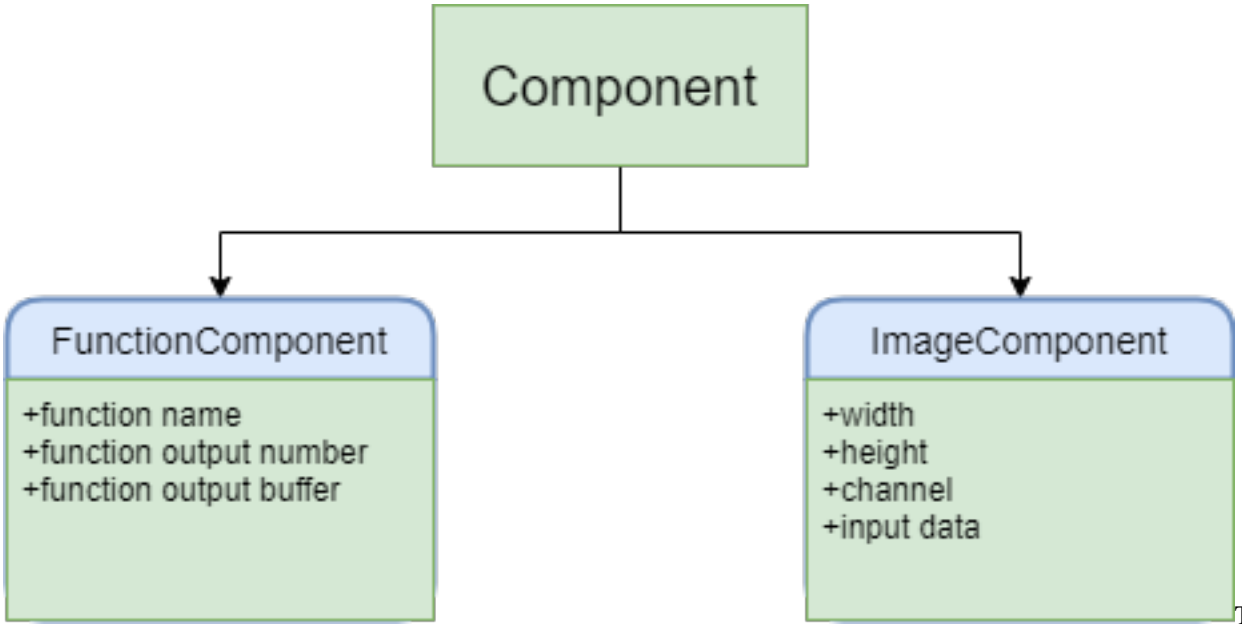
- 通过传入输入参数的方式。

```
/**
 * @brief Run Tengine Factory by bytes.
 * @param input_data image or video bytes.
 * @param input_data image or video width.
 * @param input_data image or video height.
 * @return None.
 */
void runWithData(uint8_t* input_data, int width, int height);
```

6.3 获取输出

```
/**
 * @brief Get Tengine Factory Output.
 * @return memory of TFactoryComponent data.
 */
TFactoryComponent* getComponents();
```

对于每个图像的输入，都会输出一个 `TFactoryComponent`。



TFactoryComponen

通过下面的方式获取输入图像的宽、高、通道、以及数据。

```
    TFactory::TFactoryComponent *com = interProcess->getComponents();
    int image_w = com->width();
    int image_h = com->height();
    int channel = com->channel();
    uint8_t *input_data = com->buffer();
```

获取每个算法的输出：有两种方式：

- 传入算法的名称，也就是 json 文件里面配置的。

```
    /**
     * @brief get function component by functionName ,function name is configured in␣
↪the json file.
     * @param functionName function name.
     * @return function output.
     */
    FunctionComponent* componentOutput(std::string functionName);
```

- 获取所有的算法输出。

```
    /**
     * @brief get function components.
     * @return function outputs.
     */
    std::vector<FunctionComponent*> getComponentsOutput();
```

输出为 FunctionComponent 为一个数据结构。

```
struct FunctionComponent
{
    // function Name
    std::string functionName;
    // how many outputs in function
    int function_output_count;
    // outputs buffer
    std::vector<float*> output_buffers;
};
```

6.4 释放

```
/**
 * @brief Release Tengine Factory.
 * @return None.
 */
void release();
```

记得释放，否则会造成内存泄漏。

6.5 获取图像数量

```
/**
 * @brief Get image count in folder.
 * @return image count.
 */
int imageCount();
```

CHAPTER 7

配置

7.1 参数表格

7.2 示例配置

```
{
    "Function" : ["FaceDetect", "FaceLandmark"],
    "Thread" : 2,
    "Sources" : "./images/",
    "FaceDetect" : {
        "pipelineMode" : "Standard",
        "ModelPath" : "",
        "mean" : [127.0, 127.0, 127.0],
        "normal" : [0.0078125, 0.0078125, 0.0078125],
        "min_sizes" : [
            [10.0, 16.0, 24.0],
            [32.0, 48.0],
            [64.0, 96.0],
            [128.0, 192.0, 256.0]],
        "scales" : [
            [8.0, 4.0, 2.0, 1.0],
            [8.0, 4.0, 2.0, 1.0],
            [32.0, 16.0, 8.0, 4.0]],
```

(下页继续)

(续上页)

```
        "base_sizes" : [[16], [16], [16]],
        "ratios" : [2.5],
        "clip" : false,
        "input_w" : 160,
        "input_h" : 120,
        "input_type" : "RGB",
        "strides" : [8.0, 16.0, 32.0, 64.0],
        "score_threshold" : 0.6,
        "iou_threshold" : 0.3,
        "variance" : [0.1, 0.2],
        "preprocess" : "CreateAnchor",
        "postprocess" : "NMS",
        "TensorOuputString" : ["scores:score", "boxes:box"]
    },
    "FaceLandmark" : {
        "ModelPath" : "./model/landmark.tmfile",
        "pipelineMode" : "MutiInput",
        "mean" : [127.0, 127.0, 127.0],
        "normal" : [0.007874, 0.007874, 0.007874],
        "input_w" : 160,
        "input_h" : 160,
        "input_type" : "RGB",
        "input_stream" : ["FaceDetect:rect"],
        "output_stream" : ["points:212"]
    }
}
```

CHAPTER 8

贡献内容

8.1 反馈问题

- Github issues
- Email: Support@openailab.com
- QQGroup: 787519516 (TengineFactory)

8.2 提交你的代码

- 提交 PR，你需要说明你修改了什么。
  - Fork it!
  - 创建你的分支: git checkout -b my-new-feature
  - 提交你的修改: git add . && git commit -m ‘Add some feature’
  - 推送这个分支: git push origin my-new-feature
  - 提交请求

Thanks a lot!!

CHAPTER 9

ChangeLog

CHAPTER 10

FAQ