**项目详细方案**

## 一、项目执行思路介绍

本文选择论文《[Robust High-Resolution Video Matting with Temporal Guidance](https://arxiv.org/pdf/2108.11515.pdf" \t "_blank)》提出的人物分割模型RH-RVM作为算法基础框架。针对高清视频中人物分割问题RH-RVM算法利用帧间信息进行隐式学习，增加了分割结果在时序上的连贯性。但是RH-RVM依然是针对GPU设计的模型，无法在CPU中实现实时处理。因此本文对RH-RVM网络模型进行分析，将RH-RVM中的主干网络从movilenetv3-large更改为movilenetv3-large-0.75,使用数据集进行训练，训练指标达到，略低于论文指标。

部署时选择c++语言和[OpenVINO](https://link.zhihu.com/?target=https%3A//software.intel.com/en-us/openvino-toolkit" \t "_blank)推理框架进行推理工程搭建。[OpenVINO](https://link.zhihu.com/?target=https%3A//software.intel.com/en-us/openvino-toolkit" \t "_blank)是英特尔推出的一款全面的工具套件，最适用于英特尔处理器快速部署应用和解决方案。测试集分割效果如下：

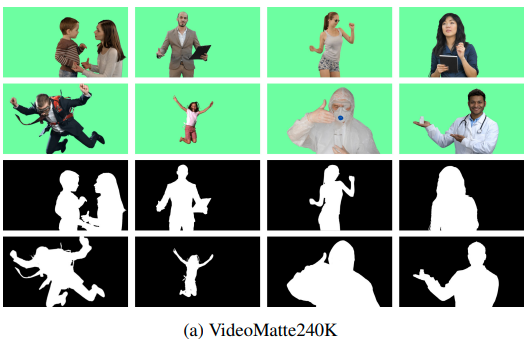
## 二、详细技术方案

### ２.１业务需求

本次任务需要实现一个不需要辅助性的trimap、CPU端可以实时进行人物分割的模型。本次数据集选择VideoMatte240K（VM）、Distinctions-646（D646）、Adobe Image Matting（AIM）。

**（１）VideoMatte240K**

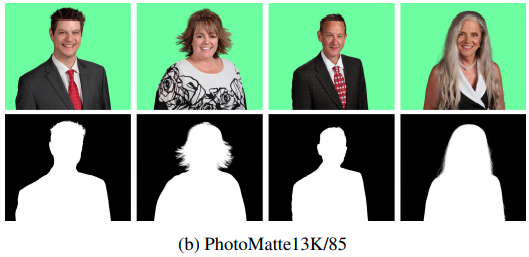
VideoMatte240K 中采集了 484 段高分辨率绿幕greenscreen 视频，并采用色度键抠图Chroma−key软件 Adobe After Effects 生成了 240709 张 alpha mattes 和 foregrounds 的视频帧。

[](https://aiuai.cn/uploads/2012/cc77d170441c7a65.png)其中，384 段视频的分辨率为 4K, 100 段视频的分辨率为 HD. 根据 479:5 的比例将视频划分为 tran 和 validataion 数据集.如图：

**（２）PhotoMatte13K/85**

PhotoMatte13K/85 中采集了 13665 张图像集合，这些图像是采用摄影棚质量的光照，在绿幕前的相机所拍摄的，并采用色度键抠图算法Chroma−key 和手工调整以及错误修正所提取的 mattes.

根据 13165:500 的比例划分为 train 和 validation 数据集.

[](https://aiuai.cn/uploads/2012/439d8d7bf947a84b.png)数据集图像分辨率平均为 2000x2500，包括了一些头发.

**（３）Automatic Portrait Matting**

数据集是一个图像分割数据集，包含2000张图像和对应的Matting标注结果，大小为600×800，都是自拍图像。其中Matting用closed-form matting和KNN matting方法生成。下载地址：[Deep Automatic Portrait Matting (cuhk.edu.hk)](http://www.cse.cuhk.edu.hk/~leojia/projects/automatting/)

**（４）PPM-100**

PPM-100 是论文 MODNet (Github | Arxiv) 中提出的一个人像抠图基准，它包含了100张来自Flickr的人像图片，具有以下特点。精细标注 - 所有图像都被仔细标注并检查。 丰富多样 - 图像涵盖全身/半身人像和各种姿态。 高分辨率 - 图像的分辨率介于1080P和4K之间。 自然背景 - 所有图像都包含原始无替换的背景。 下载地址：https://github.com/PaddlePaddle/PaddleSeg/tree/release/2.3/contrib/Matting

**（５）matting\_human\_datasets**

数据集为目前已知最大的人像matting数据集，包含34427张图像和对应的matting结果图。 数据集由北京玩星汇聚科技有限公司高质量标注，使用该数据集所训练的人像软分割模型已商用。 数据集中的原始图片来源于Flickr、百度、淘宝。经过人脸检测和区域裁剪后生成了600\*800的半身人像。 https://github.com/aisegmentcn/matting\_human\_datasets PS：Matting比较粗糙，没有达到头发细致抠图；不过数据比较大，可以作为pretrained数据集使用。

### ２.２模型调研

训练框架选择Pytorch、详细训练过程如下：



### ２.３效果评估

模型部署时，将pytorch训练好的pth模型转为onnx格式，再使用[OpenVINO](https://link.zhihu.com/?target=https%3A//software.intel.com/en-us/openvino-toolkit" \t "_blank)的Mo工具转为xml和Bin格式的部署文件。

表1模型果评估结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试平台 |  | 前处理 | 后处理 | 推理时间 | 精度FG MSE |
| 11th Gen Intel(R) Core(TM) i7-11850H @ 2.50GHz |  | 5 ms | 25 ms | 12ms | 20.36 |

### ２.４参数统计

使用thop插件统计参数情况如下：

模型大小：14.7M

参数量：374万

计算量：1.24GFLOPs

### ２.５算法工程化

模型最终部署在因特尔显卡上，因此采用OpenVino（版本号2021.4.752）作为推理框架。

1. 模型转换：pytorch🡪onnx

|  |
| --- |
| * **理解RVM的隐向量的维度设置**   隐向量的输出在decoder.py的脚本里面实现了，如果想要转换成静态的ONNX文件，则必须弄清楚这几个rxi的通道数和维度。r1i,r2i,r3i,r4i的通道数，分别是16，20，40和64，所以在转换ONNX时也要按照这个通道数设置。那么隐向量的h和w具体怎么计算呢？其实和正常的网络结构输出的feature-map大小计算逻辑类似，除以步长即可。只是在RVM中，有个downsample\_ratio，图片进入模型后的第一步就是按比例进行resize，然后才是在resize后的张量上进行正常的特征抽取，所以，h和w的计算，需要集合downsample\_ratio和步长。计算逻辑为：  在假设下采样率为0.375，W=1280,H=720,那么:   * r1i的shape为(1,c1,Hx0.375/2,Wx0.375/2), 即(**1, 16, 135, 240**) * r2i的shape为(1,c2,Hx0.375/4,Wx0.375/4)，即(**1, 20, 68, 120**) * r3i的shape为(1,c3,Hx0.375/8,Wx0.375/8)，即(**1, 40, 34, 60**) * r4i的shape为(1,c4,Hx0.375/16,Wx0.375/16)，即(**1, 64, 17, 30**) * c1,...,c4的数值，跟模型结构有关， * **修改model.py跳过不必要的自定义op**   **def** **forward**(self, src, r1, r2, r3, r4,  downsample\_ratio: float **=** 0.375, *# 注意，需要一个默认值，直接导出成ONNX常量*  segmentation\_pass: bool **=** False):    **if** torch**.**onnx**.**is\_in\_onnx\_export():  *# 如果是为了导出静态的ONNX 可以不使用该自定义算子*  *# src\_sm = CustomOnnxResizeByFactorOp.apply(src, downsample\_ratio)*  src\_sm **=** self**.**\_interpolate(src, scale\_factor**=**downsample\_ratio)  修改后的源码实现长这样：  https://pic4.zhimg.com/80/v2-47e2fd6cf7ed7fe0d994493405878573_720w.jpg   * **导出成静态的ONNX**   现在，基本上把要准备好的都准备了，可以写个导出脚本了。export\_onnx\_static.py  **import** argparse  **import** torch  **from** model **import** MattingNetwork  **class** **StaticExporter**:  **def** \_\_init\_\_(self):  self**.**parse\_args()  self**.**init\_model()  self**.**export()  **def** **parse\_args**(self):  parser **=** argparse**.**ArgumentParser()  parser**.**add\_argument('--model-variant', type**=**str, required**=**True, choices**=**['mobilenetv3', 'resnet50'])  parser**.**add\_argument('--model-refiner', type**=**str, default**=**'deep\_guided\_filter',  choices**=**['deep\_guided\_filter', 'fast\_guided\_filter'])  parser**.**add\_argument('--checkpoint', type**=**str, required**=**False)  parser**.**add\_argument('--output', type**=**str, required**=**True)  self**.**args **=** parser**.**parse\_args()  **def** **init\_model**(self):  self**.**precision **=** torch**.**float32  self**.**device **=** torch**.**device("cpu")  self**.**model **=** MattingNetwork(  self**.**args**.**model\_variant,  self**.**args**.**model\_refiner  )**.**eval()**.**to(self**.**device, self**.**precision)  **if** self**.**args**.**checkpoint **is** **not** None:  self**.**model**.**load\_state\_dict(  torch**.**load(self**.**args**.**checkpoint, map\_location**=**self**.**device), strict**=**False)  **def** **export**(self):  **print**(self**.**args)  *# rec = (torch.zeros([1, 1, 1, 1]).to(self.args.device, self.precision),) \* 4*  src **=** torch**.**randn(1, 3, 720, 1280)**.**to(self**.**device, self**.**precision) *# h=720 w=1280*  r1i **=** torch**.**randn(1, 16, 135, 240)**.**to(self**.**device, self**.**precision)  r2i **=** torch**.**randn(1, 20, 68, 120)**.**to(self**.**device, self**.**precision)  r3i **=** torch**.**randn(1, 40, 34, 60)**.**to(self**.**device, self**.**precision)  r4i **=** torch**.**randn(1, 64, 17, 30)**.**to(self**.**device, self**.**precision)  *# 假设你的model的forward方法，已经指定downsample\_ratio的默认值为torch.tensor([0.375]).*  *# downsample\_ratio = torch.tensor([0.375]).to(self.args.device)*  torch**.**onnx**.**export(  self**.**model,  (src, r1i, r2i, r3i, r4i), *# 不需要导出downsample\_ratio，直接使用默认值*  self**.**args**.**output,  export\_params**=**True,  opset\_version**=**12,  do\_constant\_folding**=**True,  input\_names**=**['src', 'r1i', 'r2i', 'r3i', 'r4i'], *# 不需要导出downsample\_ratio，直接使用默认值*  output\_names**=**['fgr', 'pha', 'r1o', 'r2o', 'r3o', 'r4o'])  **if** \_\_name\_\_ **==** '\_\_main\_\_':  StaticExporter()   * **运行导出命令**   PYTHONPATH=. python3 ./export\_onnx\_static.py --model-variant mobilenetv3 --checkpoint weights/rvm\_mobilenetv3.pth --output weights/rvm\_mobilenetv3\_720x1280.onnx  这样就可以导出静态ONNX文件了，不会再报错。  ➜ weights ls -lh  total 288592  -rw-r--r--@ 1 yanjunqiu staff 15M Feb 1 18:49 rvm\_mobilenetv3.pth  -rw-r--r--@ 1 yanjunqiu staff 14M Feb 21 22:19 rvm\_mobilenetv3\_720x1280.onnx # 成功导出的静态ONNX  -rw-r--r--@ 1 yanjunqiu staff 103M Feb 1 18:52 rvm\_resnet50.pth |

2）模型转换： onnx🡪OpenVino

|  |
| --- |
| openvino中用于将训练的模型转换成IR文件，可以使用mo.py脚本  **转换命令：python mo.py --input\_model=rvm\_mobilenetv3\_fp32.onnx --output\_dir=. --model\_name=rvm\_openvino**  该脚本位于/安装地址/intel/openvino/deployment\_tools/model\_optimizer/mo.py  上述文件需要安装openvino才会出现，安装方法详见网页https://blog.csdn.net/qq\_36556893/article/details/81385008  查看其帮助信息：  https://img-blog.csdnimg.cn/20191205165551846.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly96aGlrdW5odW8uYmxvZy5jc2RuLm5ldA==,size_16,color_FFFFFF,t_70  里面命令行参数较多，其中比较重要的参数为：  -input\_model: 为输入的训练的模型，如果使用的是pytorch训练的模型则应该为XXX.onnx  --output\_dir:为输出的转换后IR存放的路径  --log\_level：配置转换过程中输出的debug信息级别,支持的配置项为：  {CRITICAL,ERROR,WARN,WARNING,INFO,DEBUG,NOTSET}  --input\_shape：为整个深度学习网络的输入shape，有时网络的输入需要用户单独输入，比如caffe在的input layer,其输入可以使用--input\_shape指定，其输入的shape为四维，对于caffe其四维空间顺序为[N,C,H,W]，有多个输入需要设置时，需要使用逗号分开，例如：--input\_shape [1,1,240,320],[1,1,240,320]input\_shape需要和--input参数一起使用  --input：指定需要配置输入的哪个层的第几个参数，其格式为port1:node\_name1，后面也可以使用port1:node\_name1[shape1]，多个之间使用逗号分开  --input\_shape和--input结合使用用例：  --input 0:LeftCensus,0:RightCensus --input\_shape [1,1,240,320],[1,1,240,320] |

3）模型部署：C++

|  |
| --- |
| * **环境配置**   工程文件详见matting文件夹，编译工程需要安装并配置cmake和VS2015以上。文件夹中主要包含头文件、链接库、动态库以及主程序文件。生成工程时会自动配置上述文件。   * **生成工程**   双击build.sh等待生成VS工程   * **编译工程**   选择项目工程-》右键弹出控制菜单-》选择重新生成，等待编译完成。   * **可执行程序测试**   输出栏可以查看可执行文件地址，将可执行文件复制到Demo\_test文件夹中。  打开命令行工具cmd，切换到Demo\_test，输入 |

## 三、项目优势与劣势

优势：

模型选择最近2020年由论文《Real-Time High-Resolution Background Matting》提出的RH-RVM模型。主要考虑如下几点：

1. RH-RVM模型以实时、快速著称。
2. 模型不需要辅助性的trimap
3. 模型针对高清图像处理提出了DGF模块，提升图像清晰度。
4. 模型针对视频人像分割提出前后帧信息结合方法，实现时间信息融合。
5. 训练源码公开

劣势：

1）模型结构过于复杂无法在CPU实时处理

2）不利于工程化部署

3）训练源码虽然公开，但是复现时无法达到论文著称精度等

针对上述问题，本文从以下几点进行模型优化：

1. 增加数据。论除了论文中使用的VideoMatte240K、Distinctions-646、Adobe Image Matting三个数据，还增加了matting\_human\_datasets、PPM-100、Automatic Portrait Matting和PhotoMatte13K/85四个数据集。
2. 主干网络从movilenetv3-large更改为movilenetv3-large-0.75。movilenetv3-large的参数为5.4M，movilenetv3-large-0.75的模型参数为4M。模型参数下降1.4M，速度提升也很明显。
3. 训练中尝试不同ｔｒｉｃｋｓ，包括数据增广（翻转旋转或平移、亮度改变、颜色改变、随机形变、随机图像缩小）和Mixup。

## 四、项目指标

基于VideoMatte240K 测试集测效果如下图所示

