系列文章

OpenVINO™ 工具套件

intel.

# Ubuntu20.04 环境下使用 OpenVINO 部署 BiSeNetV2 模型

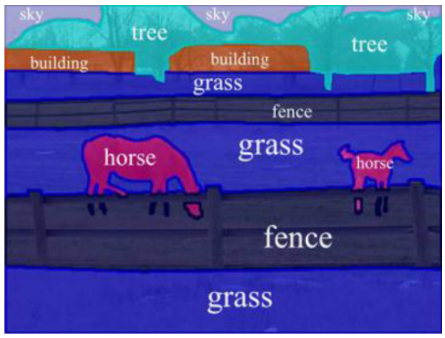
# 目录

# 1. 概述

语义分割 (Semantic Segmentation) 是计算机视觉对现实世界理解的基础，大到自动驾驶，小到个人应用只要细心观察都可以发现语义分割的应用场所无处不在， 其实语义分割相当于是 图像分割 + 对分割区域的理解。

图一：图像分割 [1]

图一可以看出图像分割就只负责分割出图像种不同的区域。




图二：语义像分割 [1]

与图一的图像分割相比，图二的语义分割明显更进一步， 不仅分割出了不同的"区域", 同时也理解不同"区域"所代表的事物，比如：马，草地，围栏，天空。

因此图像语义分割也称为图像语义标注，由于图像语义分割不仅要识别出对象还要标出每个对象的边界，所以相关的模型会具有像素级别 (Pixel Level) 的密集预测能力。

本篇文章的主要目的是展示如何把已有的百度飞桨 Paddle 模型一步一步的部署到 Intel 的 Movidius Myriad X VPU 上，同时本文也展示了一种思路：如何在浩瀚的 Internet 网络找到正确的信息，少走弯路，本文用到的包含有 Intel VPU 的设备是 OAK-D Camera，值得一提的是，相较于部署老模型，本文用到的模型是作者自己训练出来的相对较新的 BiSeNetV2 路面分割模型，BiSeNetV2 是一个在准确度和性能方面拥有较好平衡的模型，所有的代码都可以在第 6 部分找到链接。

## 1.1 部署模型落地的重要性

想必很多朋友都已经在电脑上训练过许多模型，但 AI 真正的用途除了部署在常规电脑上进行大规模的工业应用外，还在于与 IoT 这样的 edge devices 相结合且用于现实生活场景中来方便人们的生活，所以如果一个模型不能脱离 PC 且运行在低能耗的边缘设备上如树梅派，手机，Intel VPU, 那么它的实际价值会大打折扣。

## 1.2 这篇文章的着重点和非着重点

正如前面提到的，这篇文章的着重点在于一步一步演示怎样把已经训练好的百度飞桨 PaddleSeg 模型经过转 ONNX, 再转到 OpenVINO IR 模型，最后编译成体积更小的 Blob 模型，直至部署到包含有 Intel Movidius Myraid X VPU 的 OAK-D 相机上为止，在每一步我都会提供相应的官网 网址，一步一步的把读者推向正确的官网文档，减少走弯路。

图三 :OAK-D 以及 OAK-1 Cameras

再来讲一下这篇文章不讲什么，这篇文章不讲解怎样安装 Python, Anaconda, OpenVINO 这样的基础需求框架，以上几个产品的官方网站教程都做的非常详细，而且会实时更新，笔者不想让自己的文章过了几个月后 out of date, 我相信对于每个不同的技术 读相对应的官方文档可以省去很多麻烦，少走弯路，这篇文章更多的精力会用在讲解模型之间的转换，部署，以及排错。

## 1.3 Intel OpenVINO 简介

Intel OpenVINO( 以下简称 OV) 是 Intel 发布的一个综合性工具包，用于快速开发解决各种任务的应用程序和解决方案，

它包括人类视觉，自动语音识别，自然语言处理，推荐系统等。 该工具包基于最新一代人工神经网络，包括卷积神经网络 (CNN)、Recurrent Network 和基于注意力的网络，可跨英特尔® 硬件扩展计算机视觉和非视觉工作负载，从而最大限度地提高性能。

### 1.4 百度飞桨

百度飞桨 ( 以下简称 Paddle) 是百度旗下一个致力于让 AI 深度学习技术的创新与应用更加简单的工具集，其中包括 ,PaddleCV, PaddleSeg,PaddleClas 等工具帮助您快速的搭建起 AI 应用程序，最快落地 AI 模型项目，对于有过 Tensorflow, PyTorch 经验的朋友来说，Paddle 的角色和前面二者是一样的，都是高度集成的 AI 框架，目前 Paddle 有很活跃的开发者社区，有利于快速找到自己需要的答案和帮助。

## 2. 面向的读者和需要的软件

### 2.1 面向的读者

本文面向的读者是具有一定编程能力和经验的开发人员 ,AI 模型开发员，熟悉 Python 语言，并使用 Anaconda, 已有训练好的模型，期待能部署到边缘设备上来投入实际生产中，对于 0 基础的读者也是一个很好的机会通过一篇文章一并了解以上的几个技术以及怎样综合使用这些技术，让它们成为您得心应手的工具来帮助您最快的实现 AI 部署。

### 2.2 需要的软件

Anaconda, Python( 创建 Anaconda 虚拟环境的时候会自带 ), OpenVINO, Paddle, PaddleSeg, Paddle2Onnx, mamba。

## 3. 安装 PaddlePaddle & PaddleSeg

在介绍完以上内容或，现在可以正式动工啦，由于本文用到的 BiSeNetV2 路面分割模型是用 PaddleSeg 训练的，所以需要先安装 PaddleSeg 的基础库 PaddlePaddle，然后再安装 PaddelSeg。

在安装 Paddle 组件之前，请确保您已经安装好了 Anaconda。

第一步：创建一个 conda 虚拟环境：

```
conda create -n "paddle" python=3.8.8 ipython
```

创建好环境后，别忘了激活环境：

```
conda activate paddle
```

第二步：安装 GPU 或者 CPU 版本的 PaddlePaddle:
至于是选择 GPU 还是 CPU 版本的 Paddle, 主要是根据您的硬件配置，如果您有 NVIDIA 最近几年的显卡例如 :RTX 1060, RTX 2070 等，那么请选择安装 GPU 版本。

首先安装 NVIDIA 的 cudnn

```
conda install cudnn
```

安装的时候也可以把 conda 换成 mamba, 从而得到更快的下载速度。


图四：使用 Mamba 安装 cudnn

这里快速介绍一下 mamba，它是 Conda 的 C++ 实现，相比于 Conda, 它提供多线程下载，这也就意味着它可以比 Conda 更好的利用网络资源，下载的更快，同时它也能更快的解析依赖项，估计用 Conda 多的朋友应该会遇到过，Conda 有时候查找依赖项非常慢 很耽误时间，Mamba 是您的好朋友，以下教程种再看到 conda 的命令，大家都可以自动替换成 mamba 来提高速度，让您的效率飞起来 ~！

安装 PaddlePaddle 的时候，Paddle 的官网是您的好朋友,(任何时候安装任何东西，找到官网一般就能获取最新的指南 ), 我以 Ubuntu 20.04 的系统为例 ( 如果您用其他的系统，那么请选择相应的选项 )。


图五：安装 PaddlePaddle

具体命令如下：

```
conda install paddlepaddle-gpu==2.1.2 cudatoolkit=11.2 -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/Paddle/ -c conda-forge
```

安装完底层框架之后，是时候安装 PaddleSeg 啦，同理，安装 PaddleSeg 的时候您的好朋友还是它的官网或者它的 Github 仓库。

```
pip install paddleseg
git clone https://github.com/PaddlePaddle/PaddleSeg
```

安装完 PaddleSeg 之后 我们就可以开始激动人心的步骤：导出已经训练好的模型 ~！

## 4. 模型转换

模型的转换分为 4 个步骤 :
（1）导出已经训练好的模型
（2）转换到 ONNX 模型
（3）通过 ONNX 模型转换到 OpenVINO 的 IR 模型
（4）最后编译 IR 模型成为 .blob 模型


图六 :AI 模型的转换过程

其中（3）和（4）都是可以在 Intel Movidius Myriad X VPU 上部署测试的。

### 4.1 导出已经训练好的模型

这里我以我的路面分割模型为例 ( 如果您有自己的模型 , 请替换掉下面例子中的模型 , 并且更新相应的配置即可 )，命令的格式如下 :

```
export CUDA_VISIBLE_DEVICES=0 # 设置 1 张可用的卡
$ python export.py --config /Github/PaddleSeg/configs/bisenet/bisenet_road_224.yml --model /models/paddle/road_seg_224_miou_9383_6000iter.pdparams
```

具体训练好的模型以及训练模型的代码已经发布到了百度的 AI Studio 上面 : [AI 创造营 ] 基于 BiSeNetV2 的路面分割模型。[2]

如果想知道更多参数 , 您的好朋友还是 PaddleSeg 的官方 Github Repository: 模型导出。

**来讲解一下这条命令:**
--config 是用来指定模型配置参数的文件，在这个文件里它说明了您使用的模型叫什么名字 , 比如在我的例子中 , 使用的模型叫做 : BiSeNetV2, 您需要的分类有多少种 , 用了什么优化器 , 损失函数是什么 ,batch size 是多少等等都在这个文件里面。

**来看一下 bisenetv2_road_224.yml 文件的内容:**

```
_base_: '../_base_/cityscapes_1024x1024.yml'
train_dataset:
type: Cityscapes
transforms:
- type: RandomPaddingCrop
crop_size: [384, 384]
- type: RandomHorizontalFlip
prob: 0.3
- type: Resize
target_size: [224,224]
- type: Normalize
mode: train

val_dataset:
type: Cityscapes
transforms:
- type: Resize
target_size: [224,224]
- type: Normalize
mode: val

model:
type: BiSeNetV2
num_classes: 2

optimizer:
type: sgd
weight_decay: 0.0005

loss:
types:
- type: CrossEntropyLoss
- type: CrossEntropyLoss
- type: CrossEntropyLoss
- type: DiceLoss
- type: DiceLoss
coef: [1, 1, 1, 1, 1]
```

batch_size: 8
iters: 6000

lr_scheduler:
type: PolynomialDecay
learning_rate: 0.01
end_lr: 0.0001
decay_steps: 0.9

一个小窍门就是，参考 PaddleSeg 项目里已有的模板（例如您刚克隆的 PaddleSeg 代码下面的 configs/bisenet/bisenet_cityscapes_1024x1024_160k.yml）一级一级的追查回到最底层的模板，您就差不多可以知道在 xml 文件里有哪些参数可以指定的了，同时也参考自己在训练 AI 模型的时候代码里用到了哪些参数，基本上都是要在这个 config 文件里面反映出来的。

--model 指向的是您已经训练好的模型文件。

## 4.2 转模型到 ONNX: Paddle --> ONNX

模型导出后第一道转换现在开始了， Paddle 提供了转换工具 Paddle2onnx，我们先来安装它：

```
pip install paddle2onnx
```

是时候转化模型到 ONNX 啦：

```
paddle2onnx \  --model_dir inference \  --model_filename model.pdmodel \  --params_filename model.pdiparams \
--save_file road_seg.onnx \  --opset_version 11
--enable_onnx_checker True
```

这里 model_dir, model_filename, 以及 params_filename 和 Save_file 替换成自己的文件路径就好。

--model_dir 是转换后的模型要保存的目录
--enable_onnx_checker 把这个也启动，让转换程序帮我们检查模型

**我当时遇到的问题：**
Opset_version 的默认值是 9，当我在转 BiSeNetV2 的时候一开始并没有指定这个，而且出错了，经过研究，发现是因为 BiSeNetV2 的框架比较新，需要把这个 opset_version 调高到 11，更改到 11 后就好了，目前看到官网能稳定支持的是 11，但是也有看到过别人用 12 的，大家可以边试边用。

如果转换成功啦则会看到类似的如下信息：

```
2021-08-23 22:14:33 [INFO] ONNX model generated is valid.
2021-08-23 22:14:33 [INFO] ONNX model saved in /onnx_models/road_seg.onnx
```

## 4.3 转换 ONNX 模型到 OpenVINO IR 模型

铺垫了很久，终于来到了这一步。

先来快速介绍一下 OpenVINO 的 IR 模型，IR 的全称叫做 Intermediate Representation，IR 格式的模型是由 2 个文件组成的，它们分别是 .xml 和 .bin。

来到这一步之前请确保您的 Intel OpenVINO 安装成功啦，怎样安装 Intel OpenVINO 呢？您的好朋友又要出现了：Intel OpenVINO 官网安装教程，这里是 Intel OpenVINO 官方下载地址。

Intel OpenVINO 的安装包里 3 种安装选项分别是：
- 图像界面 GUI 安装
- 命令行安装
- 命令行安装安静模式

对于新手，推荐用 GUI 安装，清清楚楚 明明白白。

### 4.3.1 设置外部软件依赖

安装成功后，记得把 Install External Software Dependencies 这个部分的要求也跟完这一步是需要的。

### 4.3.2 激活 Intel OpenVINO 环境变量

**小提示：**接下来要使用 OV 就要设置好它的环境变量，官方教程要求把环境变量加载到您的 .bashrc 文件里，这样每次打开任何命令行接口都可以自动加载 OV 的环境变量，但是我在实际使用过程中发现了一个问题，安装完 OV 后，我的一部分程序开始报错，出现了一个和 Gstreamer 相关的错误信息，

经过研究发现原来 OV 的环境变量会和 Anaconda 的虚拟环境冲突，导致 GStreamer 出现问题。

其实解决方法也很简单，我们一般只会在模型转换的时候用到 OV，那么就不要把 OV 的环境变量设置到 .bashrc 文件里面，只需要在使用 OV 之前，在命令行里激活 OV 的环境变量就行。

激活环境变量的方法如下：

```
source /opt/intel/openvino_2021/bin/setupvars.sh
```

记住 /opt/intel/openvino_2021/bin 是默认的 OV 安装路径，如果您改变了路径，请记得也随之改变这里的路径。

### 4.3.3 配置模型优化器 Model Optimizer(MO)

相信我 同志们，我知道过程很长，但是曙光就在眼前啦 ~! 这个就是开始转 OpenVINO IR 模型前要调整的最后一步，坚持住 ~ !

MO 是一个基于 Python 的命令行工具，可以用来从其他流行的人工智能框架例如 Caffe, ONNX, TensorFlow 等 导入训练好的模型，没有用 MO 优化过的模型是不能用来在 OV 上做推理的。

在这一步可以只为您需要的环境比如 ONNX, 或者 Tensorflow 等做配置，但也可以一下配置好可以适用于各种人工智能框架的环境，我在这里选择了后者，毕竟路慢慢其修远 现在用 ONNX 之后也有可能用到任何其他网络。

那么第一步先 CD 到 MO 设置的文件夹里面：

```
cd /opt/intel/openvino_2021/deployment_tools/model_optimizer/install_prerequisites
```

然后运行以下命令来安装要求的依赖项：

```
sudo ./install_prerequisites.sh
```

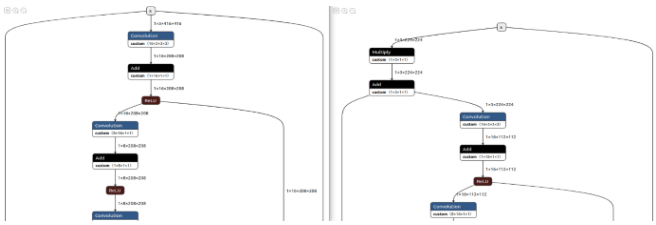
### 4.3.4 转 ONNX 模型到 IR 模式

```
cd /opt/intel/openvino_2021/deployment_tools/model_optimizer
python mo_onnx.py --input_model /inference/onnx_models/road_seg.onnx \
--output_dir /openvino/FP16 \
--input_shape [1,3,224,224] \
--data_type FP16 \
--scale_values [127.5,127.5,127.5] \
--mean_values [127.5,127.5,127.5]
```

由上面的命令行可以看出，在转换的时候您可以用 --data_type 来指定模型的精度，
--input_shape 来指定模型接受图像的形状。

最关键的两点我要拿出来讲，分别是 --mean_value(MV) 和 --scale_values(SV)。

在转到 IR 模型的时候如果指定了这 2 个参数，那么在之后用模型做推理的时候就可以不用做 Normalization，或许您可能觉得这个并没有什么用处，但是这 2 个选项省了我很多事情，它们会在模型内部直接生成 2 个操作来帮您做 Normalization。



图七：左边为未指定 MV 与 SV 的模型 VS 右边包含了 MV 和 SV 的模型

**如果转换成功，您将会看到如下输出：**

```
Model Optimizer arguments:
Common parameters:
- Path to the Input Model:
  /inference/onnx_models/road_seg.onnx
- Path for generated IR:    /openvino/FP16
- IR output name:           road_seg
- Log level:        ERROR
- Batch:  Not specified, inherited from the model
- Input layers:     Not specified, inherited from the model
- Output layers:    Not specified, inherited from the model
- Input shapes:     [1,3,224,224]
- Mean values:      [127.5,127.5,127.5]
```

```
- Scale values:     [127.5,127.5,127.5]
- Scale factor:     Not specified
- Precision of IR:  FP16
- Enable fusing:    True
- Enable grouped convolutions fusing:       True
- Move mean values to preprocess section:           None
- Reverse input channels:           False
ONNX specific parameters:
- Inference Engine found in:
/opt/intel/openvino_2021/python/python3.8/
openvino Inference Engine version:
2021.4.0-3839-cd81789d294-releases/2021/4 Model
Optimizer version:
2021.4.0-3839-cd81789d294-releases/2021/4
[ SUCCESS ] Generated IR version 10 model.
[ SUCCESS ] XML file: /openvino/FP16/road_seg.xml
[ SUCCESS ] BIN file: /openvino/FP16/road_seg.bin
[ SUCCESS ] Total execution time: 10.50 seconds.
[ SUCCESS ] Memory consumed: 142 MB.
```

至此，如果不追求更小体积模型的话，那么恭喜您 这个 IR 模型已经可以直接运行在 Intel 的 Movidius Myriad X VPU 上啦，在推理的性能上和编译后的模型没有区别，但是在读取模型的时候会比编译的后模型稍微慢一些，不过就只发生在程序初始化读取模型的时候。

## 4.4 验证转换后的 IR 模型

在继续下去之前我们应该先检验一下这个模型是否真的转换成功。

**在运行如下代码之前，请换一个命令行窗口，并且启动之前创建的 Anaconda 环境，这样做是为了确保 OV 的环境变量和 Conda 的不互相冲突，产生错误。**

**运行如下代码 ir.py 检测转换后的模型的正确性：**

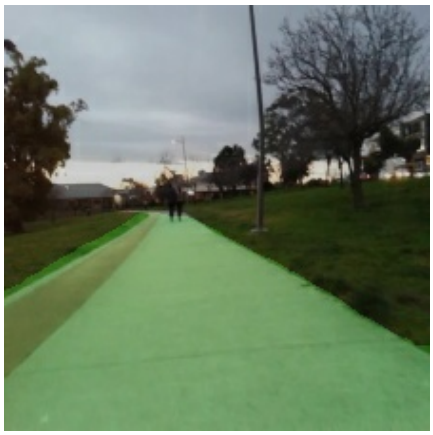
```
import numpy as np
from openvino inference_engine import IENetwork, IECore
import cv2
import paddleseg transforms as T
def get_net(model_xml, model_bin, device_name="MYRIAD"):
ie = IECore()
# Read IR
net = IENetwork(model=model_xml, weights=model_bin)
input_blob = next(iter(net.inputs))
exec_net = ie load_network(network=net, device_name=device_name)
del net
return exec_net, input_blob

def save_img(img, img_fn):
cv2.imwrite(img_fn, img)

model_xml = r'/openvino/ FP16/road_seg_half.xml'
model_bin = r'/openvino/ FP16/road_seg_half.bin'

transforms = [
T.Resize(target_size=(224,224))
]

# Run inference
img_fn = '/data/boxhill_079.jpeg'
img = cv2.imread(img_fn)

img, _ = T.Compose(transforms)(img)
# add an new axis in front
img_input = img[np.newaxis, :]
exec_net, input_blob = get_net(model_xml, model_bin)
result = exec_net.infer(inputs={input_blob: img_input})
img_segmentation = result['save_infer_model/scale_0.tmp_1']
img_segmentation = np.squeeze(img_segmentation)
class_colors = [[0,0,0], [0,255,0]]
class_colors = np.asarray(class_colors, dtype=np.uint8)
img_mask = class_colors[img_segmentation]
img_mask, _ = T.Compose(transforms)(img_mask)
img_overlayed = cv2.addWeighted(img, 0.8, img_mask, 0.2, 0.5)
img_overlayed = img_overlayed.transpose(1,2,0)
img_overlayed = cv2.cvtColor(img_overlayed, cv2.COLOR_RGB2BGR)
```

```
save_img(img_overlayed, 'demo.jpg')
```

在运行程序之前请记住先把您的带有 Intel Movidius Myriad X VPU 的设备链接上电脑，我这里用的是 OAK-D Camera。

**请替换掉代码中的图片和 IR 模型路径。**

如果成功 那么应该能够看到这个路面分割模型会自动识别出路面部分并且在原有图像的路面上加上一层半透明的绿色层，如下图：



图八：路面分割 mask 图像

至此，恭喜您不仅把模型转到了 Intel OpenVINO 的 IR 格式，而且已经成功部署到了 Intel 的 VPU 上面了。

代码里面的 device_name='MYRIAD'就是指定模型的部署方向，这里除了 MYRIAD 还可以是 CPU 等。

## 4.5 编译 IR 模型为 Blob 格式

最后的最后，如果您精益求精，不仅要部署到 VPU 上面，还想要进一步减小模型体积，那么可以用 Intel OpenVINO 来对 IR 模型进行编译，具体命令如下：

```
(base) winstonfan@wf-x:/opt/intel/openvino_2021/deployment_tools/tools/compile_tool$ ./compile_tool -m /openvino/FP16/road_seg.xml \
-d MYRIAD \
-o /openvino/compiled_u8.blob \
-ip U8 \
-il NCHW
Inference Engine:
IE version ..... 2021.4.0
Build ..... 2021.4.0-3839-cd81789d294-releases/2021/4
Network inputs:
 x : U8 / NCHW
Network outputs:
save_infer_model/scale_0.tmp_1 : I32 / CHW
[Warning][VPU][Config] Deprecated option was used : VPU_MYRIAD_PLATFORM
Done. LoadNetwork time elapsed: 20295 ms
```

**小提示：如果不知道各个参数的含义，可以使用 -h 参数来获取参数说明。**

## 4.6 检验 Blob 模型

其实验证编译后的 Blob 模型的思路和 IR 模型是一样的，唯一需要替换的代码就是在调入神经网络时候的代码。

**代码如下 blob_infer_img.py:**

```
import time
import cv2
import numpy as np
from openvino.inference_engine import IECore
import paddleseg.transforms as T

def get_net(model_blob, device_name='MYRIAD'):
ie = IECore()
exec_net = ie.import_network(model_blob, device_name = device_name)
input_blob = next(iter(exec_net.inputs))
return exec_net, input_blob

def save_img(img, img_fn):
cv2.imwrite(img_fn, img)

# Run inference
img_fn = '/data/boxhill_079.jpeg'
img = cv2.imread(img_fn)

transforms = [
```

```
T.Resize(target_size=(224,224))
]

model_blob = r'/openvino/blob/ model_uint8.blob'

img, _ = T.Compose(transforms)(img)
# add an new axis in front
img_input = img[np.newaxis, :]
t1 = time.time()
exec_net, input_blob = get_net(model_blob)
result = exec_net.infer(inputs={input_blob: img_input})
print(' time used : {}'.format(time.time() - t1))
img_segmentation = result['save_infer_model/scale_0.tmp_1']
# img_segmentation is int32
img_segmentation = np.squeeze(img_segmentation)
class_colors = [[0,0,0], [0,255,0]]
class_colors = np.asarray(class_colors, dtype=np.uint8)
img_mask = class_colors[img_segmentation]
img_mask, _ = T.Compose(transforms)(img_mask)
img_overlayed = cv2.addWeighted(img, 0.8, img_mask, 0.2, 0.5)
img_overlayed = img_overlayed.transpose(1,2,0)
img_overlayed = cv2.cvtColor(img_overlayed, cv2.COLOR_RGB2BGR)
save_img(img_overlayed, "demo2.jpg")
```

至此 , 整个流程结束。恭喜大家成功的把模型落地 并且部署到了边缘设备上，期待看到你们各个精彩的应用啦 !

本文对应的源代码 Github 仓：[https://github.com/franva/Intel-OpenVINO-Paddle](https://github.com/franva/Intel-OpenVINO-Paddle), 欢迎大家提出宝贵意见。

## 5. 总结

本文一开始先介绍了图像分割和语义分割，阐述了部署模型到边缘设备的重要性，快速介绍了 Intel 的 OpenVINO 以及百度的 PaddlePaddle 框架，然后以一个训练好的百度飞桨 Paddle 模型为例开始，一步一步带着大家把模型转换到了 OpenVINO 的格式，最后编译优化，直到部署到 Intel 的 Movidius Myriad X VPU 上面，对于不同的模型，只需要适量的改动，便可以快速独立的开发属于自己的 AI 应用程序。

## 6. 学术引用

[1]: 多伦多大学，Semantic Segmentation。
[2]: 百度 [AI 创造营 ] 基于 BiSeNetV2 的路面分割模型。

如欲了解更多 OpenVINO™ 开发资料，
请扫描下方二维码，我们会把最新资料及时推送给您。