白皮书

OpenVINO Toolkit

OpenVINO + SSD 实时目标检测

前面我们了解OpenVINO如何部署图像分类网络ResNet，本文我们将会学习OpenVINO中目标检测网络的部署与推理应用。说到目标检测网络，我们首先需要理解两个概念，目标检测与目标检测网络。

SSD目标检测模型

目标检测是计算机视觉核心任务之一，也是最常见与应用最广泛的视觉场景。OpenVINO已经提供了以下通用场景下的目标检测包括人脸检测、行人检测、物体检测、车辆检测、车牌检测等，一个图像目标检测显示示意图如下:

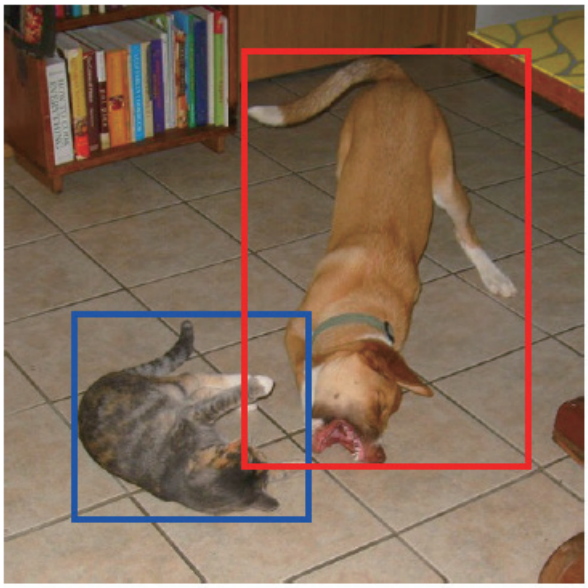


图1（来自SSD论文）

相比图像分类，目标检测多了每个对象位置信息，所以简单的认为目标检测=图像分类+Box位置信息。第一个深度学习相关的目标检测网络正是基于这样思想的RCNN模型，但是它的缺点是无法实时，所以2015年底有人提出了一个实时目标检测网络Single Shot MultiBox Detector缩写为SSD。它的模型结构如下:

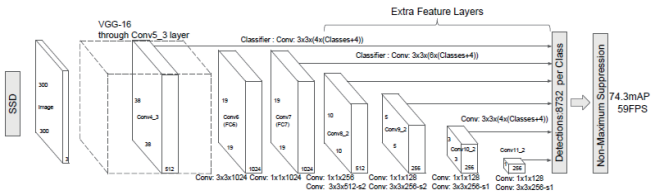


图2（来自SSD论文）

图2中SSD目标检测网络简单说可以分为三个部分:

- 骨干网络(backbone) 这里为VGG16
- 颈部Neck，构建多尺度特征
- 检测头–非最大抑制与输出

OpenVINO2021.04中模型库自带预训练的人脸检测网络有很多，分别针对不同的应用场景与输入分辨率。这里我们以face-detection-0202人脸检测模型为例说明，它是一个MobileNetV2作为基础网络的SSD目标检测模型，模型支持的输入图像大小与格式如下:

NCHW=1x3x384x384其中

N表示图像数目，这里为1

C表示输入图像通道数目，这里彩色图像为3

H表示图像高度

W表示图像宽度

期望的图像通道顺序：BGR

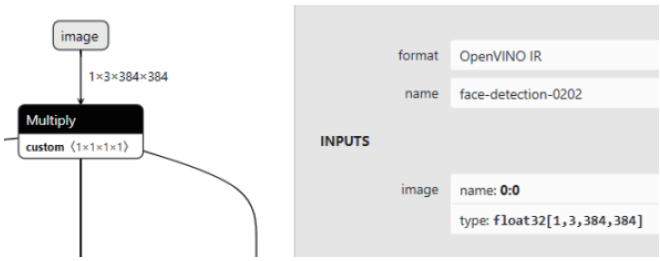


图3

模型推理计算得输出格式为:

1x1xNx7其中

N表示检测到的对象数目

7表示[image_id, label, conf, x_min, y_min, x_max, y_max]这七个值，其中

Image_id表示图像编号，这个输入的是一张图像，base为0

Label 表示标签，跟数据集的label_map文本文件相关，根据标签编号可以查找标签文本名

Conf 表示对象的置信度，取值范围在0~1之间，值越大表示置信程度越高

x_min, y_min, x_max, y_max四个值对象位置信息，分别是左上角与有下角的坐标

该模型的相关性能参数如下:

Metric	Value
AP (WIDER)	86.74%
GFlops	0.786
MParams	1.828
Source framework	PyTorch*

图4

从上面我们可以知道模型来自Pytorch训练生成。

OpenVINO基于SSD模型实时人脸检测

现在我们已经了解SSD模型的基本网络结构，OpenVINO自带SSD人脸检测模型face-detection-0202的输入与输出相关格式与参数细节信息，这里我们就基于该模型使用OpenVINO中的推理引擎模型实现一个实时人脸检测应用，根据之前的内容，我们首先需要加载模型，相关代码如下:

```
// 加载检测模型
auto network = ie.ReadNetwork(model_xml, model_bin);
```

其中

model_xml表示face-detection-0202模型文件

model_bin表示face-detection-0202权重文件

然后设置模型的输入与输出相关的格式，代码如下:

```
// 请求网络输入与输出信息
InferenceEngine::InputsDataMapinput_info(network.get-
InputsInfo());
InferenceEngine::OutputsDataMapoutput_info
(network.getOutputsInfo());
// 设置输入格式
for (auto &item : input_info) {
                auto input_data = item.second;
                input_data->setPrecision(Precision::U8);
                input_data->setLayout(Layout::NCHW);
}
printf("get it \n");

// 设置输出格式
for (auto &item : output_info) {
                auto output_data = item.second;
                output_data->setPrecision(Preci-
sion::FP32);
}
```

加载可执行网络，创建推理请求对象实例，代码如下:

```
// 创建可执行网络对象
auto executable_network = ie.LoadNetwork(network,
"CPU");

// 创建推理请求
auto infer_request = executable_network.CreateInferRe-
quest();
```

设置推理输入图像数据，转换为NCHW格式的blob数据，代码如下:

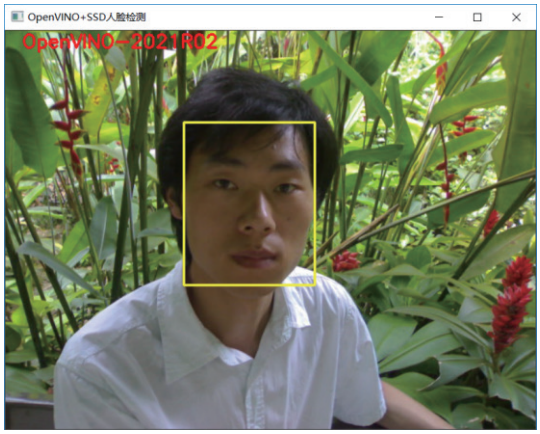
```
/** Getting input blob **/
auto input = infer_request.GetBlob(input_name);
size_tnum_channels = input->getTensorDesc().getDims()
[1];
size_t h = input->getTensorDesc().getDims()[2];
size_t w = input->getTensorDesc().getDims()[3];
```

```
size_timage_size = h*w;
Mat blob_image;
resize(src, blob_image, Size(w, h));

// NCHW
unsigned char* data = static_cast<unsigned char*>(input->buffer());
for (size_t row = 0; row < h; row++) {
    for (size_t col = 0; col < w; col++) {
        for (size_tch = 0; ch<num_channels; ch++) {
            data[image_size*ch + row*w + col] = blob_image.at<Vec3b>(row, col)[ch];
        }
    }
}
```

预测与解析输出结果，代码如下:

```
// 执行预测
infer_request.Infer();

// 处理输出结果
for (auto &item : output_info) {
    auto output_name = item.first;

    // 获取输出数据
    auto output = infer_request.GetBlob(output_name);
    const float* detection = static_cast<PrecisionTrait<Precision::FP32>::value_type*>(output->buffer());
    const SizeVectoroutputDims = output->getTensorDesc().getDims();
    const int maxProposalCount = outputDims[2];
    const int objectSize = outputDims[3];

    // 解析输出结果
    for (int curProposal = 0; curProposal<maxProposalCount; curProposal++) {
        float label = detection[curProposal * objectSize + 1];
        float confidence = detection[curProposal * objectSize + 2];
        float xmin = detection[curProposal * objectSize + 3] * image_width;
        float ymin = detection[curProposal * objectSize + 4] * image_height;
        float xmax = detection[curProposal * objectSize + 5] * image_width;
        float ymax = detection[curProposal * objectSize + 6] * image_height;
        if (confidence > 0.5) {
            printf("label id : %d\n", static_cast<int>(label));

            Rectrect;
            rect.x = static_cast<int>(xmin);
            rect.y = static_cast<int>(ymin);

            rect.width = static_cast<int>(xmax - xmin);

            rect.height = static_cast<int>(ymax - ymin);

            putText(src, "OpenVINO-2021R02", Point(20, 20), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0, 0, 255), 2, 8);

            rectangle(src, rect, Scalar(0, 255, 255), 2, 8, 0);
        }
        std::cout<< std::endl;
    }
}
imshow("OpenVINO+SSD人脸检测", src);
```

最终显示结果如下:



总结:

本文我们完成了OpenVINO人脸检测模型的推理调用演示，关键知识点在于模型的输入与输出格式，以及推理以后的模型输出数据的解析方式。到这里大家希望借助OpenVINO实现一个视频版本的人脸检测，没关系下一次我们将来完成这样的事情….

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。