白皮书

OpenVINO Toolkit

# OpenVINO™ + ResNet实现图像分类

本文我们将在前面文章的基础上，介绍OpenVINO™中最重要的组件推理引擎（Inference Engine, IE）SDK如何与应用开发集成，实现深度神经网络模型的部署，前面我们已经知道跟IE相关的对象是Core，这里我们首先介绍如何使用IE的Core实现与应用集成，给应用程序添加人工智能相关的功能。

## 推理引擎(IE)应用开发流程与相关函数介绍

通过OpenVINO™的推理引擎跟相关应用集成相关深度学习模型的应用基本流程如下:


图1

从图-1可以看到只需要七步就可以完成应用集成，实现深度学习模型的推理预测，各步骤中相关的API函数支持与作用解释如下:

### Step 1:

InferenceEngine::Core // IE对象

### Step 2:

Core.ReadNetwork(xml/onnx)输入的IR或者onnx格式文件，返回CNNNetwork对象

### Step 3:

InferenceEngine::InputsDataMap, InferenceEngine::InputInfo, // 模型输入信息

InferenceEngine::OutputsDataMap // 模型输出信息

使用上述两个相关输入与输出对象就可以设置输入的数据类型与精度，获取输入与输出层的名称。

### Step 4:

```
ExecutableNetwork  LoadNetwork (
const CNNNetwork &network,
const std::string &deviceName,
const std::map< std::string, std::string > &config={}
)
```

通过Core的LoadNetwork方法生成可执行的网络，如果你有多个设备，就可以创建多个可执行的网络。其参数解释如下:
network 参数表示step2加载得到CNNNetwork对象实例
deviceName表示模型计算所依赖的硬件资源，可以为CPU、GPU、 FPGA、 FPGA、MYRIAD
config默认为空

InferRequest InferenceEngine::ExecutableNetwork::CreateInferRequest()
表示从可执行网络创建推理请求。

### Step 5:

根据输入层的名称获取输入buffer数据缓冲区，然后把输入图像数据填到缓冲区，实现输入设置。其中根据输入层名称获取输入缓冲区的函数为如下:

```
Blob::Ptr  GetBlob (
const std::string &name // 输入层名称
)
```

注意：返回包含输入层维度信息，支持多个输入层数据设置！

### Step 6:

推理预测，直接调用推理请求的InferRequest.infer()方法即可，该方法无参数。

### Step 7:

调用InferRequest的GetBlob()方法，使用参数为输出层名称，就会得到网络的输出预测结果，根据输出层维度信息进行解析即可获取输出预测信息与显示。

## 图像分类与ResNet网络

图像分类是计算机视觉的关键任务之一，关于图像分类最知名的数据集是ImageNet，包含了自然场景下大量各种的图像数据，支持1000个类别的图像分类。OpenVINO™在模型库的public中有ResNet模型1000个分类的预训练模型支持，它们主要是:

- resnest-18-pytorch
- resnest-34-pytorch
- resnest-50-pytorch
- resnet-50-tf

其中18、34、50表示权重层，pytorch表示模型来自pytorch框架训练生成、tf表示tensorflow训练生成。ResNet系列网络的详细说明如下:

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| | | 3×3 max pool, stride 2 | | | | |
| conv2_x | 56×56 | [3×3, 64; 3×3, 64] ×2 | [3×3, 64; 3×3, 64] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 |
| conv3_x | 28×28 | [3×3, 128; 3×3, 128] ×2 | [3×3, 128; 3×3, 128] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×8 |
| conv4_x | 14×14 | [3×3, 256; 3×3, 256] ×2 | [3×3, 256; 3×3, 256] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×23 | [1×1, 256; 3×3, 256; 1×1, 1024] ×36 |
| conv5_x | 7×7 | [3×3, 512; 3×3, 512] ×2 | [3×3, 512; 3×3, 512] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

**图2（来自**
**《Deep Residual Learning for Image Recognition》论文）**

我们以ResNet18-pytorch的模型为例，基于Pytorch框架我们可以很轻松的把它转换为ONNX格式文件。然后使用Netron工具打开，可以看到网络的输入图示如下:

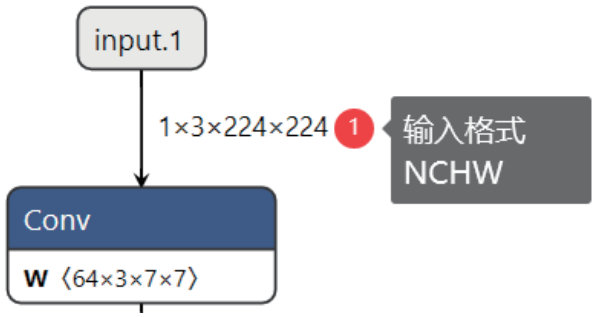


**图3**

查看网络的输出:

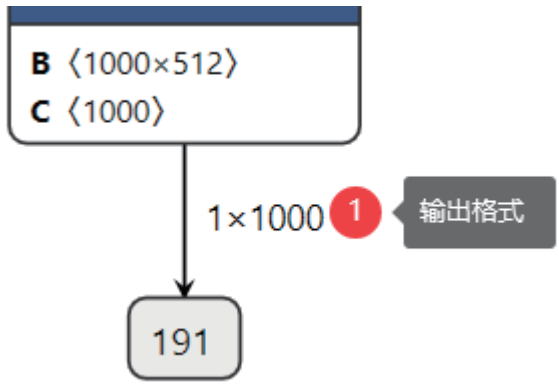


**图4**

这样我们很清楚的知道网络的输入与输出层名称，输入数据格式与输出数据格式，其中输入数据格式NCHW中的N表示图像数目，这里是1、C表示图像通道数，这里输入的是彩色图像，通道数为3、H与W分别表示图像的高与宽，均为224。在输出格式中1x1000中1表示图像数目、1000表示预测的1000个分类的置信度数据。

## 程序实现的基本流程与步骤

前面已经介绍了IE SDK相关函数，图像分类模型ResNet18的输入与输出格式信息。现在我们就可以借助IE SDK来完成一个完整的图像分类模型的应用部署了，根据前面提到的步骤各步的代码实现与解释如下:

### 1.初始化IE

```
InferenceEngine::Core ie;
```

### 2.加载ResNet18网络

```
InferenceEngine::CNNNetwork network = ie.ReadNetwork(onnx);
InferenceEngine::InputsDataMap inputs = network.getInputsInfo();
InferenceEngine::OutputsDataMap outputs = network.getOutputsInfo();
```

### 3.获取输入与输出名称、设置输入与输出数据格式

```
std::string input_name = "";
for (auto item : inputs) {
                input_name = item.first;
                auto input_data = item.second;
                input_data->setPrecision(Precision::FP32);
                input_data->setLayout(Layout::NCHW);
                input_data->getPreProcess().setColorFormat(ColorFormat::RGB);
                std::cout << "input name: " << input_name << std::endl;
}

std::string output_name = "";
for (auto item : outputs) {
                output_name = item.first;
                auto output_data = item.second;
                output_data->setPrecision(Precision::FP32);
                std::cout << "output name: " << output_name << std::endl;
}
```

### 4.获取推理请求对象实例

```
auto executable_network = ie.LoadNetwork(network, "CPU");
auto infer_request = executable_network.CreateInferRequest();
```

### 5.输入图像数据设置

```
auto input = infer_request.GetBlob(input_name);
size_t num_channels = input->getTensorDesc().getDims()[1];
size_t h = input->getTensorDesc().getDims()[2];
size_t w = input->getTensorDesc().getDims()[3];
size_t image_size = h*w;
cv::Mat blob_image;
cv::resize(src, blob_image, cv::Size(w, h));
cv::cvtColor(blob_image, blob_image, cv::COLOR_BGR2RGB);
blob_image.convertTo(blob_image, CV_32F);
blob_image = blob_image / 255.0;
cv::subtract(blob_image, cv::Scalar(0.485, 0.456, 0.406), blob_image);
cv::divide(blob_image, cv::Scalar(0.229, 0.224, 0.225), blob_image);
```

```
// HWC = 》NCHW
float* data = static_cast<float*>(input->buffer());
for (size_t row = 0; row < h; row++) {
                for (size_t col = 0; col < w; col++) {
                                for (size_t ch = 0; ch < num_channels; ch++) {
                                                data[image_size*ch + row*w + col] = blob_image.at<cv::Vec3f>(row, col)[ch];
                                }
                }
}
```

在输入数据部分OpenCV导入的图像三通道顺序是BGR，所以要转换为RGB，resize到224x224大小、像素值归一化为0~1之间、然后要减去均值(0.485, 0.456, 0.406)，除以方差(0.229, 0.224, 0.225)完成预处理之后再填充到Blob缓冲区中区。

## 6.推理

```
infer_request.Infer();
```

## 7.解析输出与显示结果

```
auto output = infer_request.GetBlob(output_name);
const float* probs = static_cast<PrecisionTrait<Precision::FP32>::value_type*>(output->buffer());
const SizeVector outputDims = output->getTensorDesc().getDims();
std::cout << outputDims[0] << "x" << outputDims[1] << std::endl;
float max = probs[0];
int max_index = 0;
for (int i = 1; i < outputDims[1]; i++) {
                if (max < probs[i]) {
                                max = probs[i];
                                max_index = i;
                }
}
cv::putText(src, labels[max_index], cv::Point(50, 50), cv::FONT_HERSHEY_SIMPLEX, 1.0, cv::Scalar(0, 0, 255), 2, 8);
cv::imshow("输入图像", src);
cv::waitKey(0);
```

解析部分代码首先通过输出层名称获取输出数据对象BLOB，然后根据输出格式1x1000，寻找最大值对应的index，根据索引index得到对应的分类标签，然后通过OpenCV图像输出分类结果。

## 运行结果


图5（来自ImageNet测试集）

这样我们就使用OpenVINO™的推理引擎相关的SDK函数支持成功部署ResNet18模型，并预测了一张输入图像。你可以能还想知道除了图像分类模型，OpenVINO™推理引擎在对象检测方面都有哪些应用，我们下次继续……

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。


intel