|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| **1230-ATC EfficientNet (FP16) from Pytorch - Ascend310(1.1)\_谈锐** | | |  |
| **1230-ATC EfficientNet (FP16) from Pytorch - Ascend310(1.1)\_谈锐** | | |
| **文档版本** | **01** | |
| **发布日期** | **2020-12-15** | |
|  | | | | |
|  | 华为技术有限公司 | |  |  |

|  |
| --- |
| 版权所有 © 华为技术有限公司2020。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | <https://www.huawei.com> |
| 客户服务邮箱： | [support@huawei.com](mailto:support@huawei.com) |
| 客户服务电话： | 4008302118 |

目 录

[1 交付件基本信息 1](#_Toc58921272)

[2 输入输出数据 2](#_Toc58921273)

[3 使用要求 3](#_Toc58921274)

[4 快速上手 4](#_Toc58921275)

[5 Demo及应用sample 10](#_Toc58921276)

[6 模型推理性能 12](#_Toc58921277)

[7 配套环境 13](#_Toc58921278)

# 交付件基本信息

发布者（Publisher）：Huawei

应用领域（Application Domain）：Classification

版本（Version）：1.1

修改时间（Modified）：2020.09.30

大小（Size）：17.28M

框架（Framework）：

模型格式（Model Format）：om

精度（Precision）：FP16

处理器（Processor）：昇腾310

*应用级别（categories）：Official*

描述（Dsecription）：基于TorchVision官方densenet121模型，使用ATC工具转换出的，可以在Davinci芯片上运行的离线模型

# 输入输出数据

* 输入数据

| 输入数据 | 大小（宽 x 高，单位：像素） | 数据类型 | 数据排布格式 |
| --- | --- | --- | --- |
| input | batchsizes x 1 x 224 x 224 | RGB\_FP32 | NCHW |

* 输出数据

| 输出数据 | 大小（类别数） | 数据类型 | 数据排布格式 |
| --- | --- | --- | --- |
|  | 1 x 1000 | FLOAT32 | ND |

# 使用要求

昇腾的环境配置可采用裸机安装方式完成，也可以采用docker镜像方式加速用户的环境搭建，具体的安装方式请参见对应产品型号《快速部署指南》手册中的“安装运行环境”章节。

# 快速上手

单击页面右上角的，并选择合适的下载方式下载源码包。



模型转换。

1. ONNX模型。

本模型基于EfficientNet.onnx模型转换，获取EfficientNet.onnx的方法，主要介绍两种以供参考。如使用NPU训练的模型，可以根据本模型训练部分的指导导出onnx文件，否则需要用户根据自己的模型定义文件适当修改本节中的脚本，如import路径，模型文件名等。

* 1. 使用Pytorch导出ONNX模型。

完成训练后的EfficientNet pytroch模型可以导出ONNX模型，通过执行以下命令完成转换。

torch**.**onnx**.**export(model, dummy\_input, "EfficientNet.onnx", verbose**=True**, input\_names**=**input\_names, output\_names**=**output\_names)



* model为训练完成的pytorch EfficientNet模型。
* 可以加载自行训练的权重pth文件。
  1. 通过Ascend 910训练的pytorch EfficientNet 模型checkpoint会保存为pth.tar格式，通过如下脚本可以转换为ONNX模型，可根据自己保存的文件名等进行修改：

import torch   
#from efficientnet\_pytorch import EfficientNet   
from NPU.efficientnet\_pytorch import EfficientNet   
import torch.onnx   
   
from collections import OrderedDict   
   
def proc\_nodes\_module(checkpoint,AttrName):   
 new\_state\_dict = OrderedDict()   
 for k,v in checkpoint[AttrName].items():   
 if(k[0:7] == "module."):   
 name = k[7:]   
 else:   
 name = k[0:]   
   
 new\_state\_dict[name]=v   
 return new\_state\_dict   
   
def convert():   
 checkpoint = torch.load("./checkpoint.pth.140.ok.cpu", map\_location='cpu')   
 checkpoint['state\_dict'] = proc\_nodes\_module(checkpoint,'state\_dict')   
 model = EfficientNet.from\_name('efficientnet-b0')   
 model.set\_swish(memory\_efficient=False)   
 model.load\_state\_dict(checkpoint['state\_dict'])   
 model.eval()   
 #print(model)   
   
 input\_names = ["actual\_input\_1"]   
 output\_names = ["output1"]   
 dummy\_input = torch.randn(1, 3, 224, 224)   
 #dynamic\_axes = {'actual\_input\_1': {0: '-1'}, 'output1': {0: '-1'}}   
 torch.onnx.export(model, dummy\_input, "efficientnet\_tr.onnx", input\_names = input\_names, output\_names = output\_names, opset\_version=11)   
 #torch.onnx.export(model, dummy\_input, "efficientnet\_dynamic.onnx", input\_names = input\_names, output\_names = output\_names, dynamic\_axes = dynamic\_axes, opset\_version=11)   
   
if \_\_name\_\_ == "\_\_main\_\_":   
 convert()

1. 使用ATC工具将ONNX模型转OM模型。

转换om模型：

* 不使用auto tune

export PATH=/usr/local/python3.7.5/bin:/usr/local/Ascend/ascend-toolkit/latest/atc/ccec\_compiler/bin:/usr/local/Ascend/ascend-toolkit/latest/atc/bin:$PATH   
export PYTHONPATH=/usr/local/Ascend/ascend-toolkit/latest/atc/python/site-packages/:$PYTHONPATH   
export LD\_LIBRARY\_PATH=/usr/local/Ascend/ascend-toolkit/latest/atc/lib64:$LD\_LIBRARY\_PATH   
export ASCEND\_OPP\_PATH=/usr/local/Ascend/ascend-toolkit/latest/opp   
# export DUMP\_GE\_GRAPH=2   
export SLOG\_PRINT\_TO\_STDOUT=1   
   
   
/usr/local/Ascend/ascend-toolkit/latest/x86\_64-linux\_gcc7.3.0/atc/bin/atc \   
--model= efficientnet\_npu.onnx \   
--framework=5 \   
--output=./efficientnet \   
--input\_format=NCHW \   
--input\_shape="actual\_input\_1:1,3,224,224" \   
--disable\_reuse\_memory=1 \   
--buffer\_optimize=off\_optimize \   
--log=debug \   
--soc\_version=Ascend310   
--insert\_op\_conf=aipp\_TorchVision.config

* 使用auto tune

export PATH=/usr/local/python3.7.5/bin:/usr/local/Ascend/ascend-toolkit/latest/atc/ccec\_compiler/bin:/usr/local/Ascend/ascend-toolkit/latest/atc/bin:$PATH   
export PYTHONPATH=/usr/local/Ascend/ascend-toolkit/latest/atc/python/site-packages/:/usr/local/Ascend/ascend-toolkit/latest/atc/python/site-packages/schedule\_search.egg:/usr/local/Ascend/ascend-toolkit/latest/atc/python/site-packages/auto\_tune.egg/auto\_tune:/usr/local/Ascend/ascend-toolkit/latest/opp/op\_impl/built-in/ai\_core/tbe/:$PYTHONPATH   
export LD\_LIBRARY\_PATH=/usr/local/Ascend/ascend-toolkit/latest/atc/lib64:/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64:$LD\_LIBRARY\_PATH   
export ASCEND\_OPP\_PATH=/usr/local/Ascend/ascend-toolkit/latest/opp   
# export DUMP\_GE\_GRAPH=2   
export SLOG\_PRINT\_TO\_STDOUT=1   
# export REPEAT\_TUNE=True   
   
/usr/local/Ascend/ascend-toolkit/latest/atc/bin/atc \   
--model=efficientnet\_npu.onnx \   
--framework=5 \   
--output=efficientnet \   
--input\_format=NCHW \   
--input\_shape="actual\_input\_1:1,3,224,224" \   
--enable\_small\_channel=1 \   
--log=info \   
--soc\_version=Ascend310 \   
--auto\_tune\_mode="RL,GA"   
# --input\_fp16\_nodes="actual\_input\_1"

环境变量：

* export DUMP\_GE\_GRAPH=2 ：保存ATC转换过程中生成的中间图，格式为txt和pbtxt，可以用netron等可视化软件打开查看。
* export SLOG\_PRINT\_TO\_STDOUT=1：打印转换日志。
* export REPEAT\_TUNE=True：AUTO\_TUNE功能不论算子是否命中知识库，都进行auto tune，参考auto\_tune\_mode参数说明。

参数说明：

* --model：为onnx模型文件。
* --framework：5代表onnx模型。
* --output：输出的om模型。
* --input\_format：输入数据的格式。
* --input\_shape：输入数据的shape。
* --actual\_input\_1根据onnx模型的输入shape进行改变。
* --enable\_small\_channel=1：对于densenet121等视觉模型四维数据卷积算子的特殊优化，可以提升性能，其他模型可能导致性能下降，不建议开启。
* --insert\_op\_conf=aipp\_TorchVision.config：AIPP插入节点，通过config文件配置算子信息，功能包括图片色域转换、裁剪、归一化，主要用于处理原图输入数据，常与DVPP配合使用，详见下文数据预处理。
* --auto\_tune\_mode：ATC工具提供的auto tune功能，可以帮助找到算子的最佳实现参数，分为强化学习RL和基因算法GA两种类型，对应不同的算子。“RL，GA”表示两种都开启，优化基于知识库，如果知识库中已经存在待优化算子的优化策略，则采用，否则根据设定的轮数进行训练。如果打开REPEAT\_TUNE环境变量，无论命中与否都会从零开始训练，ATC对视觉网络的优化已经比较成熟，建议不开启REPEAT\_TUNE。

开始推理验证。

1. 推理工具。

Benchmark工具为华为自研的模型推理工具，支持多种模型的离线推理，能够迅速统计出模型在Ascend310上的性能，支持真实数据和纯推理两种模式，配合后处理脚本，可以实现诸多模型的端到端过程。

工具地址：<https://gitee.com/ascend/cann-benchmark>

1. 数据预处理。
   1. 采用ImageNet 50000张图片的验证集，与torchvision官方一致，数据集获取及均值方差数值可以参 考 Torchvision： https://pytorch.org/docs/stable/torchvision/models.html
   2. 数据预处理方式分为二进制输入和JPEG图片输入两种，二进制输入需要先用脚本仿照TorchVision训练预处理方法处理数据，以获得最佳精度。JPEG输入可以直接读取原图，需要使用昇腾开发的DVPP模块和AIPP模块，因为解码、缩放等处理和TorchVison有一定区别，最终精度会下降 0.7% 左右
   3. 二进制输入

参考trochvision中的处理方法，对imagenet数据集通过缩放、均值方差手段归一化，输出为二进制文件。

import sys   
import os   
import torch   
import cv2   
from PIL import Image   
import numpy as np   
import torch.utils.data   
import torchvision.transforms as transforms   
from torch.autograd import Variable   
   
def efficientNet\_onnx(input\_path: str, output\_path: str):   
 img = cv2.imread(input\_path)   
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # 色域转换   
 pilimg = Image.fromarray(img)   
 normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],   
 std=[0.229, 0.224, 0.225])   
 val\_transformer = transforms.Compose([   
 transforms.Resize(256), #等比例缩放   
 transforms.CenterCrop(224), #中心裁剪   
 transforms.ToTensor(), #量化   
 normalize #均值方差归一化   
 ])   
   
 img\_tensor = val\_transformer(pilimg)   
 img\_tensor = Variable(torch.unsqueeze(img\_tensor, dim = 0).float(), requires\_grad = False)   
 img\_tensor.reshape(1, 3, 224, 224)   
 img\_numpy = img\_tensor.detach().cpu().numpy() if img\_tensor.requires\_grad else img\_tensor.cpu().numpy()   
   
 img\_name = input\_path.split('/')[-1]   
 bin\_name = img\_name.split('.')[0] + ".bin"   
 output\_fl = os.path.join(output\_path,bin\_name)   
 # save img\_tensor as binary file for om inference input   
 img\_numpy.tofile(output\_fl) #保存二进制数据   
   
if \_\_name\_\_ == "\_\_main\_\_":   
 pathSrcImgFD = sys.argv[1]   
 pathDstBinFD = sys.argv[2]   
 images = os.listdir(pathSrcImgFD)   
 for image\_name in images:   
 if not image\_name.endswith(".jpeg"):   
 continue   
 print("start to process image {}....".format(image\_name))   
 path\_image = os.path.join(pathSrcImgFD, image\_name)   
 efficientNet\_onnx(path\_image,pathDstBinFD)

* 1. JPEG图片输入

通过DVPP实现解码、缩放功能，输出YUV数据，再通过AIPP进行色域转换及裁剪，最终直接输入网络中进行推理，方便快捷。

DVPP使用方法详见昇腾DVPP使用指导，本模型使用Benchmark集成的DVPP。

AIPP需要配置aipp.config文件，在ATC转换的过程中插入AIPP算子，即可与DVPP处理后的数据无缝对接，AIPP参数配置详见ATC工具使用指导。

aipp\_TorchVision.config:

aipp\_op{   
 aipp\_mode:static   
 input\_format : YUV420SP\_U8   
   
 src\_image\_size\_w : 256   
 src\_image\_size\_h : 256   
   
 crop: true   
 load\_start\_pos\_h : 16   
 load\_start\_pos\_w : 16   
 crop\_size\_w : 224   
 crop\_size\_h: 224   
   
 csc\_switch : true   
 rbuv\_swap\_switch : true   
   
 min\_chn\_0 : 123.675   
 min\_chn\_1 : 116.28   
 min\_chn\_2 : 103.53   
 var\_reci\_chn\_0: 0.0171247538316637   
 var\_reci\_chn\_1: 0.0175070028011204   
 var\_reci\_chn\_2: 0.0174291938997821   
   
 matrix\_r0c0: 256   
 matrix\_r0c1: 0   
 matrix\_r0c2: 359   
 matrix\_r1c0: 256   
 matrix\_r1c1: -88   
 matrix\_r1c2: -183   
 matrix\_r2c0: 256   
 matrix\_r2c1: 454   
 matrix\_r2c2: 0   
 input\_bias\_0: 0   
 input\_bias\_1: 128   
 input\_bias\_2: 128   
}

1. 使用Benchmark工具进行推理。

* 二进制输入

./benchmark.x86\_64 -model\_type=vision -batch\_size=1 -device\_id=0 -input\_text\_path=./BinaryImageNet.info -input\_width=224 -input\_height=224 -om\_path=./efficientnet.om -useDvpp=False

BinaryImageNet.Info为处理后的数据集信息，参考demo中样例

* 图片输入

./benchmark.x86\_64 -model\_type=vision -batch\_size=1 -device\_id=0 -input\_text\_path=./ImageNet.info -input\_width=256 -input\_height=256 -om\_path=./efficientnet.om -useDvpp=False

ImageNet.info为图片信息，参考demo中样例，注意这里的input\_height\input\_weight与AIPP节点输入一致，值为256因为AIPP中做了裁剪。

训练后的输出默认在当前目录result下，调用vision\_metric\_ImageNet.py脚本与label比对，可以获得Accuracy数据，结果保存在result.json中，val\_label保存数据集标签，vision\_metric.py为评测脚本，参考demo

python3.7 vision\_metric\_ImageNet.py result/dumpOutput/ ./val\_label.txt ./efficientnet\_result.json

Benchmark的更详细使用方法请参考benchmark工具Readme

vision\_metric.py中分类相关代码：

gt = img\_gt\_dict[img\_name]   
if (n\_labels == 1000):   
 realLabel = int(gt)   
elif (n\_labels == 1001):   
 realLabel = int(gt) + 1   
else:   
 realLabel = int(gt)

----结束

# Demo及应用sample

Demo文件

Demo展示了如何用TorchVision预训练的densenet121模型在ImageNet 5万张图片验证集上进行推理与精度评测，包含文件目录如下：

* Pth转换om脚本，pth.tar转换om脚本
* ATC转换脚本autu\_tune.sh
* benchmark二进制文件
* 数据集信息ImageNet.info及二进制数据集信息BinaryImageNet.info
* 二进制数据集预处理脚本PytorchTransfer
* 数据集标签val\_label.txt
* 精度评测脚本vision\_metric\_ImageNet.py
* ReadMe
* aipp.cofig

应用Sample

本模型已适配MindX SDK sample，所需文件demo中已提供<https://www.huaweicloud.com/ascend/apps/applicationDetails/60172215>

自定义数据集推理

用户使用自定义数据集推理可以参考ImagenNet，需要准备的包括数据集图片信息或者处理后的二进制数据集信息（如ImageNet.info或BinaryImageNet.info），数据集的标签信息val\_label.txt。

更换数据集需要先进行迁移学习，参考训练部分相关指导，推理时数据集预处理的需要与训练的预处理尽可能保持一致，以便提高精度。

如使用benchmark进行推理，生成的数据用vision\_metric.py脚本评测时需要修改分类类别数目。

vision\_metric.py中分类相关代码：

gt = img\_gt\_dict[img\_name]   
if (n\_labels == 1000):   
 realLabel = int(gt)   
elif (n\_labels == 1001):   
 realLabel = int(gt) + 1   
else:   
 realLabel = int(gt)

# 模型推理性能

调用ACL接口推理计算，性能参考下列数据，该数据为基于已有知识库auto\_tune后的模型数据。

| Framework | Num of NPU Cards | Batch Size | Throughput | Atlas NPU Model | Server | Container | Precision | Dataset | Accuracy | Ascend AI Processor | NPU Version |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pytorch | 1 | 1 | 209.336  images/sec | Atlas 300-3010 | Atlas 800-3010 | NA | fp16 | ImageNet2012 | top1：71.83%  top5：90.54% | Ascend 310 | Atlas 300-3010-32GB |
| 16 | 20.692 images/sec |

# 配套环境

| Network | Network Type | Framework | Atlas NPU Model | Server | Ascend AI Processor | NPU Version |
| --- | --- | --- | --- | --- | --- | --- |
| Densenet121 | Classification | Pytorch | Atlas 300-3010 | Atlas 800-3010 | Ascend 310 | Atlas 300-3010-32GB |