**基于华为自研AI芯片的计算机视觉深度学习算法研究与移植**

**开发指导说明书**



华为技术有限公司

目录

[**基于华为自研AI芯片的计算机视觉深度学习算法研究与移植……………………………….** 2](#_Toc39013293)

[1项目介绍 2](#_Toc39013294)

[2项目目标 2](#_Toc39013295)

[3预备知识 2](#_Toc39013296)

[4实验环境 2](#_Toc39013297)

[5运行示例 3](#_Toc39013298)

[6实验原理及流程 4](#_Toc39013299)

[6.1实验原理 4](#_Toc39013300)

[6.2实验流程 5](#_Toc39013301)

[7实验任务及步骤 5](#_Toc39013302)

[任务一 模板工程获取 5](#_Toc39013303)

[任务二 模型转换 6](#_Toc39013304)

[任务三 代码修改 7](#_Toc39013305)

[任务四 编译与运行工程 10](#_Toc39013306)

**基于华为自研AI芯片的计算机视觉深度学习算法研究与移植**

## 1项目介绍

本项目实现实现实时动作检测，用外设海康威视摄像头捕获实时动作视频，通过部署在Atlas200DK开发板上的模型对动作进行识别，再将带有识别结果的视频用Presenter Server在浏览器中播放出来。将本文档主要介绍基于华为自研AI芯片的计算机视觉深度学习算法研究与移植项目代码开发并部署在Atlas 200DK开发板上执行的方法。

## 2项目目标

将计算机视觉深度学习算法成功迁移到华为自研AI平台上

实现实时动作识别，并达到可观的效果

掌握将应用部署在Atlas 200 DK开发板上的操作。

## 3预备知识

具备一定的深度学习理论知识，对深度学习框架有一定了解。

具有C++和Python编程能力

## 4实验环境

实验环境需要从硬件和软件两个方面进行准备：

（1） 硬件配件准备环境：

使用Atlas 200 DK前，需自行购买相关配件，包含制作Atlas 200 DK启动系统的micro SD卡、读卡器，与Ubuntu虚拟机相连接的Type-C数据线，详细的配件信息如表4.1所示：

表4-1硬件配件清单

|  |  |  |
| --- | --- | --- |
| 配件名称 | 描述 | 推荐型号 |
| SD卡 | 用于制作Atlas 200 DK开发者板启动系统。 | 推荐使用经过测试的SD卡：  三星UHS-I U3 CLASS 10 64G  金士顿UHS-I U1 CLASS 10 64G |
| 读卡器 | 使用读卡器制作SD卡的场景。 | 支持USB3.0协议 |
| Type-C连接线 | 用于将开发板与Mind Studio所在服务器通过USB方式连接。 | 支持USB3.0的Type-C连接线 |

（2）软件部署环境：

已在Ubuntu机器上搭建好Mind Studio2.3.3版本的环境（参考文档：<https://support.huaweicloud.com/usermanual-mindstudioc73/atlasmindstudio_02_0004.html>）。

已配置部署好Atlas 200DK环境（版本20.0.0）（参考文档：<https://support.huaweicloud.com/usermanual-A200dk_3000/atlas200dk_02_0020.html>）。

## 5运行示例

开发者可以先从此链接<https://github.com/CHY7820/HuaweiAscend>下载相关代码和文档，其中包含部署指导书文件“基于华为自研AI芯片的计算机视觉深度学习算法研究与移植-部署指导.doc”。读者可以根据该指导书所述运行本案例的示例代码，观察实时动作识别的效果。

## 6实验原理及流程

### 6.1实验原理

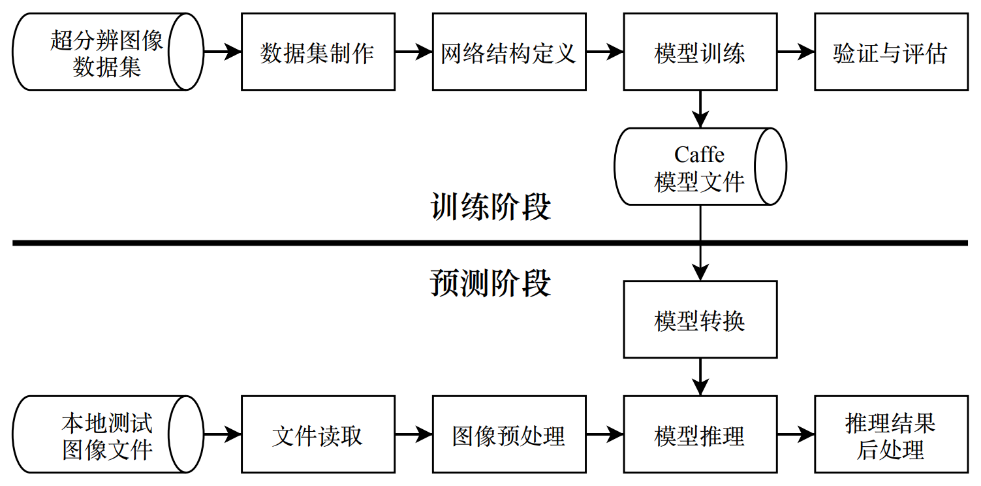


图6.1 超分辨率图像算法原理图

本实验分成两个阶段，分别是训练阶段和预测阶段，如图6.1所示。

训练阶段首先进行数据集的制作，使用MATLAB对数据集进行灰度化、下采样等操作，得到低分辨率与高分辨率的图像数据对，存储为hdf5格式文件，作为后续的训练数据；接着根据算法描述，进行Caffe网络结构的定义和训练参数的设置；随后进行网络的训练，得到能够完成图像超分辨率应用的模型文件；最后通过验证环节评估训练过程的质量，验证算法的有效性。由于训练阶段不是本实验的重点，因此这里不再详细介绍。我们在代码文件夹中也上传了训练完成得到的Caffe模型文件供读者使用。

预测阶段主要包括3个步骤。第一步是获取输入数据，系统读取本地图像数据，作为超分辨率图像网络的输入。第二步是在推理模块中执行预处理和网络的前向传播，得到放大后的图像数据。最后，对得到的数据进行后处理，再存储为本地文件，读者可将结果文件拷贝到主机查看。

### 6.2实验流程

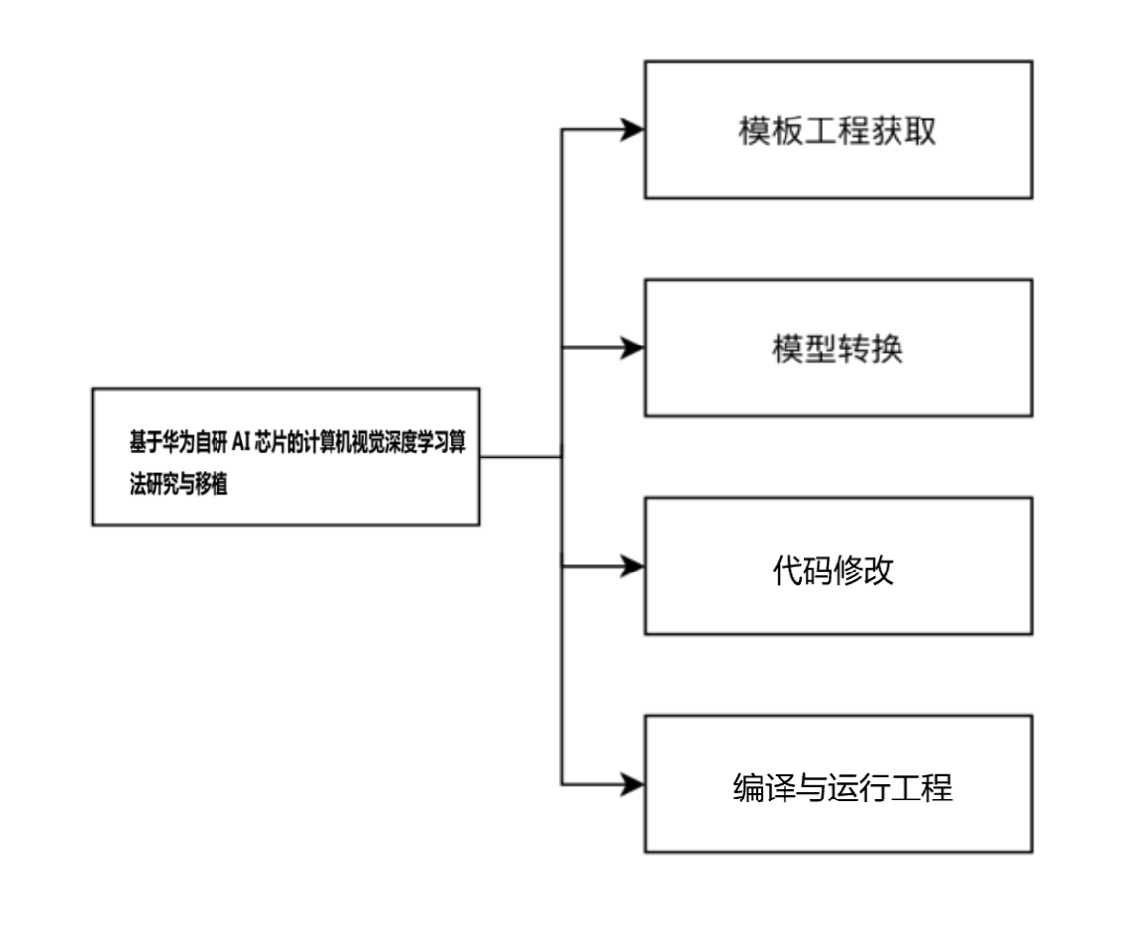


图 6.2 **基于华为自研AI芯片的计算机视觉深度学习算法研究与移植**

在本实验中，默认已完成硬件环境和软件环境的准备工作，在此基础上进行实时动作检测项目的实验操作。本实验需要在Ubuntu主机PC端完成基 于C++的应用程序代码的编写工作，网络模型的转换，最后在Atlas 200DK开发板上进行项目部署执行工作。本实验的开发流程图如图6.2所示。

本案例移植的源代码编写以GitHub上提供的源代码为例进行说明，实验任务及步骤将围绕图6.2所示四个方面分别展开介绍。

## 7实验任务及步骤

### 任务一 模板工程获取

本实验使用c++进行开发，并使用命令行操作进行应用的部署和使用，因此我们选用官方提供的通用分类网络应用案例作为接下来开发的模板工程。通用分类网络应用案例可在**<https://gitee.com/ascend/samples>**中进行下载。

参考该案例的README.md进行软件准备、部署、运行等步骤。确保环境配置无误，并能够得到正确的结果，即可进行下一步的开发。

### 任务二 模型转换

在完成超分辨模型的训练，得到全精度的算法模型之后，首先需要进行离线模型转换这一步骤，将Caffe模型转换为Ascend 310芯片支持的模型（Davinci架构模型），才可进一步将其部署在Atlas 200 DK开发板上。

在Mind Studio界面中，通过图形化的离线模型转换工具，可调用Matrix提供的模型管家接口将Caffe模型转换为Davinci架构模型，完成离线模型转换。图7.1展示了将训练完成的超分辨模型（以FSRCNN为例）转换为Davinci模型的参数配置。

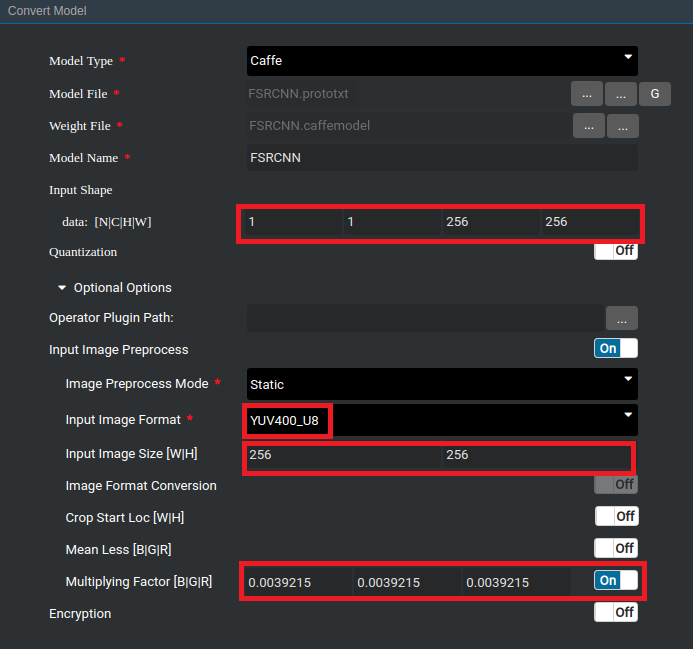


图7.1 FSRCNN离线模型转换

在模型转换的过程中需要注意以下几个参数的配置，在上图中也以红色框进行了标注。

Input Shape：以FSRCNN放大256×256的图像为例，NCHW需要设置为[1, 1, 256, 256]，其中C通道为1是由于以上训练得到的超分辨率模型都是针对灰度图像进行处理。另外，如果使用SRCNN模型，此处的H与W需要设置为768，即低分辨率图像经过bicubic放大后的尺寸。

Input Image Format：由于模型的输入是灰度图像，并且对于超分辨项目，不需要使用DVPP模块进行预处理，因此此处需要指定为格式为YUV400\_U8。

Input Image Size：与Input Shape中的HW相同。

Multiplying Factor：由于在模型训练时，输入数据的范围为[0, 1]，因此此处需将[0, 255]的数据乘上1/255进行归一化。

对于本项目中使用到的三种模型（SRCNN、FSRCNN以及ESPCN），离线模型转换的参数配置基本相同，唯一区别在于，转换SRCNN模型时，HW需要设置为输入图像尺寸的3倍，即低分辨率图像经过bicubic放大后的尺寸。

### 任务三 代码修改

完成以上步骤后，我们得到了所需要的网络模型。我们基于任务一获取的c++模板工程进行修改和补充，构建实时动作检测应用。接下来我们将对图像数据模块、推理模块以及后处理模块的更新和补充进行介绍。

图像数据模块

图像数据模块的作用是根据用户输入的参数，读取系统上的本地图像文件，作为超分辨应用的图像输入，该模块对应的完整代码请参考.cpp文件。

该模块中有两个关键点：（1）由于本项目所使用的超分辨网络的输入都是单通道的图片，因此我们需要以灰度形式读取数据，或者在读取彩色图像后将其转为灰度形式。（2）由于SRCNN网络模型的输入是低分辨率图像经bicubic放大后的结果，因此在读取图片后需要对其进行放大再送至推理模块。该部分的代码如清单7.1所示。

清单7.1 图像数据模块重要代码

// 使用OPENCV以灰度形式读取图像

cv::Mat mat = cv::imread(image\_path, CV\_LOAD\_IMAGE\_GRAYSCALE);

switch(model\_type) {

case 0: // SRCNN，若当前是SRCNN网络则对图像进行放大

{

cv::Mat mat\_bicubic;

cv::resize(mat, mat\_bicubic, cv::Size(0, 0), 3, 3, cv::INTER\_CUBIC);

mat = mat\_bicubic;

}

break;

case 1: // FSRCNN

case 2: // ESPCN

break;

}

推理模块

推理模块对应的文件为general\_inference.cpp，该模块的实现与图像分类样例中推理模块的实现类似，包括AI模型管家的初始化、离线模型的加载、调用ai\_model\_manager\_->Process()函数进行计算等，相同的部分这里不再重复阐述。不同之处在于，本样例删去了使用DVPP进行图像预处理的过程，这是因为在图像超分辨应用当中，往往不希望引入算法之外的图像缩放过程，以避免对超分辨率算法造成干扰。因此这个样例中删除了使用DVPP进行缩放的过程，这也要求输入图片的大小必须与模型转换时设置的大小相同。完成推理过程后，即可取出数据结果，发送至后处理模块。

后处理模块

在得到推理模块输出的结果后，我们需要对其进行后处理，这里主要包括三个方面：数值范围转换、ESPCN中的subpixel操作处理以及彩色图像处理。后处理模块对应的文件为general\_post.cpp。

1. 数值范围转换

由于推理模块的输出为浮点数据，且数据范围大部分在[0, 1]之间（这是由于训练时的标签也是[0, 1]之间的数据），而我们最终需要的是8位像素数据，因此我们需要进行数值范围转换，将浮点数据还原为[0, 255]的8位像素数据值，如清单7.2所示。

清单7.2 浮点数据还原为8位像素数据

//res存放浮点数据，转换后的结果存储至res\_uint8数组，size为数组大小

for (uint32\_t i = 0; i < size; i++) {

if (res[i] <= 0) {

res\_uint8[i] = 0;

} else if (res[i] >= 1) {

res\_uint8[i] = 255;

} else {

res\_uint8[i] = (uint8\_t)(round(res[i] \* 255));

}

}

1. subpixel操作

若当前处理的网络为ESPCN，我们则需要对输出的数据进行一次subpixel像素重排的操作。例如，若此时网络输出的数据为[1, 256, 256, 9]，经过subpixel操作（实际上是对一维数组进行数据重新排列的过程），可得到[1, 768, 768, 1]形状的图像数据，即为最后的高分辨率图像，该部分代码如清单7.3所示。

清单7.3 subpixel像素重排

output\_width = model\_width \* 3;

output\_height = model\_height \* 3;

uint32\_t idx1 = 0;

for (uint32\_t c = 0; c < 9; c++) {

for (uint32\_t h = 0; h < model\_height; h++) {

for (uint32\_t w = 0; w < model\_width; w++) {

uint32\_t idx2 = (h\*3+c%3) \* output\_width + (w\*3+c/3);

//与数值范围转换同时完成

if (res[idx1] <= 0) {

res\_uint8[idx2] = 0;

} else if (res[idx1] >= 1) {

res\_uint8[idx2] = 255;

} else {

res\_uint8[idx2] = (uint8\_t)(round(res[idx1] \* 255));

}

idx1++;

}

}

}

1. 彩色图像处理

通常，在图像超分辨率模型（包括本项目使用的三种模型）当中，输入图像与输出图像的通道数往往都为1，因此仅适用于单通道图像的放大。为了将这类单通道模型用于处理彩色图像，可采用图7.2的处理流程。

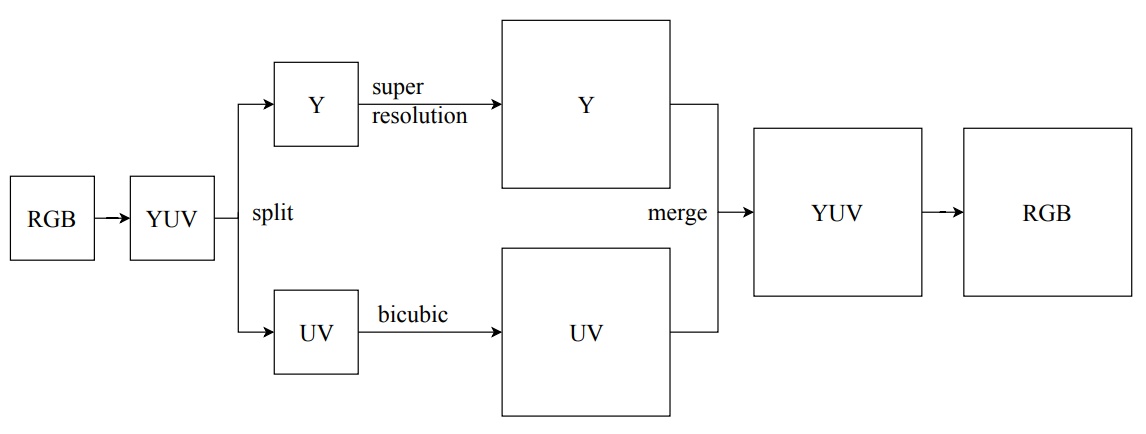


图7.2 超分辨网络彩色图像处理流程

首先，将RGB格式的图像转换为YUV格式，然后将通道进行分离。对于亮度通道Y，使用图像超分辨率算法进行放大，得到一张高分辨率灰度图像，而对于色度通道UV只需使用bicubic插值算法进行处理，这是由于人眼对亮度信号的空间分辨率大于对色度信号的空间分辨率，因此对于相对不敏感的色度通道，使用插值算法即可满足视觉要求。最后，将三个通道的高分辨率数据进行合并，再转换回RGB格式，即可完成可使用单通道模型进行彩色图像的超分辨率应用过程。该部分代码如清单7.4所示。

清单7.4 彩色图像处理

// 构造cv::Mat实例，存储的是已经放大的灰度通道的超分辨率图像

cv::Mat mat\_out\_y(height, width, CV\_8U, result);

// 读取BGR彩色图像

cv::Mat mat = cv::imread(file\_path, CV\_LOAD\_IMAGE\_COLOR);

// 转换为YUV格式，并进行bicubic放大

cv::Mat mat\_ycrcb, mat\_out\_ycrcb;

cv::cvtColor(mat, mat\_ycrcb, cv::COLOR\_BGR2YCrCb);

cv::resize(mat\_ycrcb, mat\_out\_ycrcb, cv::Size(0, 0), 3, 3, cv::INTER\_CUBIC);

// 使用超分辨率网络的输出替换Y通道的数据

vector<cv::Mat> channels;

cv::split(mat\_out\_ycrcb, channels);

channels[0] = mat\_out\_y;

cv::merge(channels, mat\_out\_ycrcb);

// 将YUV格式数据重新转换为BGR格式，并进行保存

cv::Mat mat\_out\_bgr;

cv::cvtColor(mat\_out\_ycrcb, mat\_out\_bgr, cv::COLOR\_YCrCb2BGR);

cv::imwrite(output\_name, mat\_out\_bgr);

### 任务四 编译与运行工程

本应用包含部署和运行两个部分，都是在命令行界面中进行操作。

我们在指导书文件“超分辨率图像算法-部署指导.doc”中详细提供了运行本案例部署和运行步骤、脚本使用方法与各参数的意义供读者阅读与实验。

参考“部署指导”的实验步骤后，我们能够得到超分辨率图像算法生成的高分辨率图像，运行过程和结果如图7.3、7.4所示。

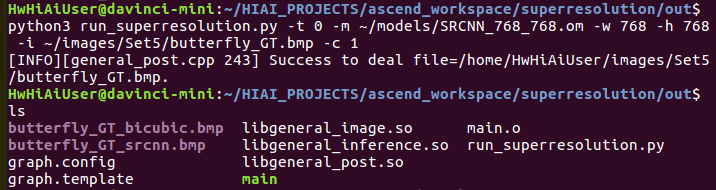


图7.3 应用运行过程

LR bicubic SRCNN

图7.4 运行结果