计算机工程
Computer Engineering
ISSN 1000-3428,CN 31-1289/TP

《计算机工程》网络首发论文

题目：基于 OpenGL ES 的移动端实时视频超分辨率显示
作者：陆晓华，王慈
DOI：10.19678/j.issn.1000-3428.0069470
网络首发日期：2024-08-19
引用格式：陆晓华，王慈．基于 OpenGL ES 的移动端实时视频超分辨率显示[J/OL]．计算机工程. https://doi.org/10.19678/j.issn.1000-3428.0069470

网络首发时间：2024-08-19 08:41:02
网络首发地址：https://link.cnki.net/urlid/31.1289.tp.20240816.1542.004

基于 OpenGL ES 的移动端实时视频超分辨率显示

陆晓华，王慈*
(华东师范大学通信与电子工程学院，上海 闵行 邮编：200241)

摘 要：当前主流的视频超分算法主要应用在服务器端或离线视频转换等业务场景中，当部署到移动端设备时，存在计算复杂、推理速度慢等问题。特别是在实时音视频通信（RTC）业务场景中，这些主流的超分算法虽然在图像质量上可以满足精度要求，但是在处理时间上很难达到性能要求，从而影响算法的实际应用效果。本文提出一种基于卷积神经网络改进和优化的实时视频超分技术 OGSR(Super resolution based on OpenGL ES)：首先采用分组卷积和通道混淆的方式，在基本不降低超分图像质量的条件下优化神经网络模型，成倍减小前向推理的计算量；其次使用 OpenGL ES 图形加速接口，将模型参数和通道数据布局成最快采样的纹理数据，上传显存用于 GPU 的并行计算；最后在 GPU 的着色器(Shader)中通过渲染像素坐标反向计算通道索引和模型参数索引实现超分算法核心模块，从而达到像素级别的细粒度并发。实验结果表明，对 QVGA(320*240)和 nHD(640*360)分辨率的视频帧进行三倍超分放大，在不同机型的移动手机上可以达到 15~30fps 的帧率，同时放大后图像质量基本与标准卷积神经网络模型的质量误差在 2%以内，满足实时业务场景的需求，性能提升显著。

关键词：视频超分；分组卷积；通道混淆；OpenGL ES；纹理
DOI: 10.19678/j.issn.1000-3428.0069470

Mobile real-time video super-resolution display based on OpenGL ES

LU Xiao-hua, WANG Ci*
(School of Communication & Electronic Engineering, East China Normal University, Shanghai 200241, China)

Abstract: The current mainstream video super-resolution algorithms are mainly applied in business scenarios such as server-side or offline video conversion. When deployed to mobile devices, there are problems such as complex computation and slow inference speed. Especially in real-time audio and video communication (RTC) business scenarios, although these mainstream super-resolution algorithms can meet the accuracy requirements in image quality, it is difficult to meet the performance requirements in processing time, which affects the practical application effect of the algorithms. This article proposes a real-time video super-resolution technology OGSR (Super resolution based on OpenGL ES) based on convolutional neural network improvement and optimization. Firstly, by using grouped convolution and channel obfuscation, the neural network model is optimized without significantly reducing the quality of the super-resolution image, which exponentially reduces the computational cost of forward inference; Next, use the OpenGL ES graphics acceleration interface to layout the model parameters and channel data into the fastest sampled texture data, and upload it to graphics memory for parallel computing on the GPU; Finally, in the shader of the GPU, the channel index and model parameter index are calculated in reverse by rendering pixel coordinates to achieve the core module of the super-resolution algorithm, thereby achieving fine-grained concurrency at the pixel level. The experimental results show that triple super-resolution amplification of QVGA (320 * 240) and nHD (640 * 360) resolution video frames can achieve a frame rate of 15-30fps on mobile phones of different models. At the same time, the quality error of the enlarged image is basically within 2% of the standard convolutional neural network model, which meets the requirements of real-time business scenarios and significantly improves performance.

Key words: Video Super-resolution; Grouped Convolution; Channel Shuffle; OpenGL ES; Texture

基金项目：上海市科技重大专项 (2023SHZDZX-0008)
*通信作者：cwang@cs.ecnu.edu.cn

1 引言

网络摄像头(IPC，全称 Internet Protocol Camera)类的设备形态主要有：电话手表、可视门铃、智能网络摄像头等。为了降低功耗，这类产品通常采用 QVGA(320*240)或者 nHD(640*360)等较低分辨率的摄像头。与此同时，这类设备的监控端应用主要是手机端 APP，手机屏幕通常超过 1280*720 像素。IPC 设备传送来的视频帧需要在移动设备上进行 2~3 倍放大后才能全屏显示。目前 Android 和 iOS 手机系统控件默认使用二次线性插值 (Biliner)图像放大算法，该算法速度快，但放大后的图像效果不太好，容易产生模糊、马赛克等效应。为了提升视频帧放大后的图像质量，三次线性插值[1] (Bicubic) 算法被引入进行视频帧放大，该算法可以提升视频帧图像的显示质量，但效果仍不尽如人意。

现在工业界主流的视频超分的实现方案是将超分算法放在云端边缘服务器进行：在云端边缘节点处接入算法服务器接收低分辨率的视频帧，通过算法服务器将视频帧进行解码、视频超分变换，再将放大后的视频帧重新编码、封装，通过网络传输给移动设备端，从而使得移动设备可以看到高分辨率的视频帧图像。采用这样的云算法服务器架构虽然简单，但是有很多缺陷：硬件成本高；通信延迟长、降低传输实时性和可靠性低；带宽成本高等问题。

本文从基于深度学习硬件平台优化的思路出发，提出了一种直接在手机端实现的实时视频超分方案—Super-resolution based on OpenGL ES（OGSR），对深度学习的视频超分神经网络框架进行优化改进，并利用移动芯片平台 GPU 硬件加速计算，实现实时视频超分。

2 神经网络框架优化

2.1 通道数据的改进

基于神经网络的超分模型有多种类别，单帧图像超分（SISR）对单张图像做超分放大处理[2]，早期基于卷积神经网络的算法主要有 SRCNN[3]；VDSR[4]；DRRN[5] 等，也有很多基于注意力机制和扩散生成的模型，例如 SRFormer[6]；SinSR[7]；Diff-Retinex[8] 等。视频属于多帧图像的超分（MISR），基于卷积神经网络的典型算法有 VESCPN[9]；EDVR[10]；TDAN[11]等，基于注意力机制的 VRT[12]、基于扩散生成的有 SATeCo[13]；MIA-VSR[14]等模型。这些单帧或者多帧超分模型主要从提升图像质量维度来考虑，较少关注模型部署落地的计算性能。近些年一些超分算法模型也关注在移动设备端的部署，会综合平衡图像质量和计算性能的平衡，例如：ECBSR[15]；QuickSRNet[16]等虽然会考虑移动设备计算性能，但依旧以图像质量维度为主。综上这些的超分算法，没有考虑到实时音视频传输(RTC)场景中至少达到 15fps 帧率的要求。本文提出的优化方案，重点解决模型部署后计算性能问题，对 QVGA(320*240)和 nHD(640*360)分辨率的视频帧进行三倍超分放大后可以达到 15~30fps 的帧率。

本文参考 ESPCN[17] 算法的网络结构，对通道数据和卷积方式进行优化改进。实时视频帧解码后的视频帧基本都是 YUV420 格式的，本文方案中仅对敏感的 Y 通道进行超分处理，这样可以降低超分计算量,同时 U 和 V 通道通过简单的双三次线性插值算法直接进行放大。对比卷积处理，插值运算计算量几乎可以忽略不计。最后将 Y 和放大后的 UV 通道格式转换，生成输出的 RGB 格式视频帧。

改进后的视频超分实现方案如图 1 所示，将 Y 通道超分放大的每一个处理过程看成一个处理组件。其中，Conv1 中的卷积核大小为 5*5，输入单通道 Y 数据，输出 36 张特征图，采用 tanh 激活函数。 Conv2 中的卷积核大小为 3*3，输出 18 张特征图，同样采用 tanh 激活函数。Conv3 中的卷积核大小为 3*3，输出 9 张特征图，sigmoid 激活函数。 相比于原始 ESPCN 网络中 RGB 三个通道数据都参与超分放大计算，本文改进的超分实现方案，整体参与卷积计算的通道数据量要减少近 2/3，这样可以减少整体计算耗时。

2.2 卷积处理的改进

在 ESPCN 网络模型中特征提取的卷积是前向推理过程中最耗时的操作，本文在 ESPCN 基础上使用轻量级神经网络来优化算法的特征提取部分。这里的轻重通常指模型的规模/参数量，常用的神经网络轻量化的技术有：网络剪枝、分组卷积、通道混淆、深度可分离卷积（包括逐通道卷积和逐点卷积）、共享权重等。基于卷积常用的轻量级网络主要有：MobileNetV2[18]; ShuffleNetV1[19]; ShuffleNetV2[20]; RepVGG[21]等，近年也有不少基于注意力机制或者其他方式的轻量级模型，如：Repvit[22]; SYENet[23]等。本文主要采用的核心思想有：

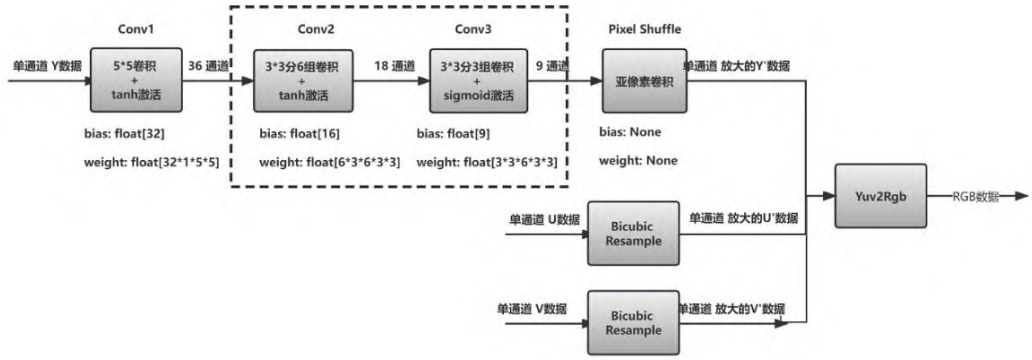

Figure 1　Optimized Super-resolution Scheme

图 1　改进的神经网络超分实现方案

（1）分组卷积

分组卷积被广泛用于各种轻量化模型中，用于减少运算量和参数量。参考 Roof-line 模型[24]使用计算量来评估性能，在如图 2 例子中，对于输入 12 路通道、输出 6 路通道数据进行标准卷积操作，那么所有卷积计算量为：

$$GLFOPs(standard)=W*H*(6*12*(K*K)) \quad (1)$$

其中 **W** 是图像宽度，**H** 是图像高度，**K** 是卷积核大小。

而将输入输出通道数据分成 3 组进行分组卷积操作，每组仅有 4 路输入通道，每组输出 2 路通道，总计的卷积计算量为：

$$GLFOPs(group)=3*(W*H*(2*4*(K*K))) \quad (2)$$

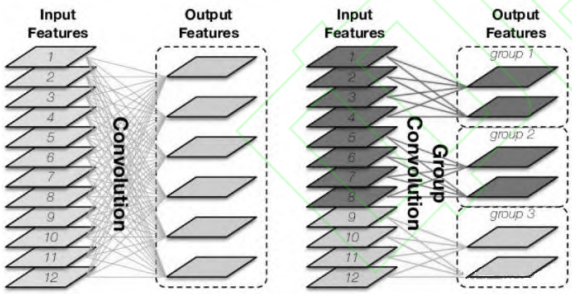

Figure 2　Standard Convolution and Group Convolution

图 2　标准卷积（左）和 分组卷积（右）对比

对比两种不同卷积方式的计算量，使用分组卷积方式的计算量仅有标准卷积方式的 1/3，这样显著降低卷积操作的计算量。但直接分组方式也有缺陷：缺失了组与组之间的通道关联性，需要通过其他方式来建立组之间通道的关联性。

（2）通道混淆

为解决分组卷积中不同组之间特征断联的问题，在进行分组卷积时，通常可以将输入通道进行如图 3 所示的混淆处理。

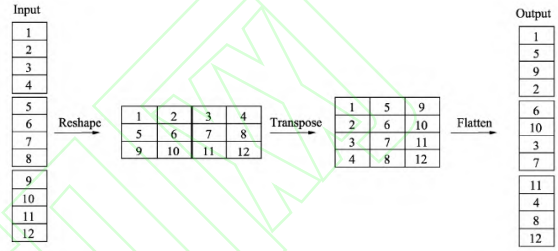

Figure 3　The Processing of Channel Shuffle

图 3　通道混淆处理过程

这里通道混排操作有三个步骤：

I. Reshape：输入通道按组分成二维矩阵

II. Transpose：输入通道矩阵转置

III.Flatten：置换后的通道矩阵重新按行展开

采用这样的混淆方式，每个组之间至少有一个通道关联，确保了原先所有组之间也至少建立起一对关联信息，从而建立起全局感受野。

2.3 卷积层优化

在本文实现方案中，特征提取的三层神经网络结构中，第一层网络依然使用标准卷积，第二层和第三层网络使用分组卷积+通道混淆来处理。具体方案如图 1 所示：

（1）Conv1 组件：采用 5*5 的标准卷积，输出通道为 36 个，方便后续进行分组。改进后的卷积计算量：

$$GLFOPs(Conv1)=W*H*(36*1*(5*5)) \quad (3)$$

（2）Conv2 组件：对于 36 个输入通道先进行混排，然后分成 6 组进行 3*3 卷积操作，每组卷积分别有 6 个输入通道和 3 个输出通道。经过改进后的卷积计算量：

$$GLFOPs(Conv2)=W*H*(3*6*(3*3)) \quad (4)$$

（3）Conv3 组件：对于 18 个输入通道先进行混排，然后分成 3 组进行 3*3 卷积操作，每组卷积分别有 6 个输入通道和 3 个输出通道。经过改进

后的卷积计算量：

GLFOPs(Conv3)=W*H*(3*6*(3*3)) （5）

这样改进后总的卷积计算量是：$GLFOPs = W * H * 1224$,而原先采用标准神经网络模型中，卷积计算量总计是：$GLFOPs = W * H * 6704$,优化后的网络模型，推理计算量只有原先的 1/5，并且根据后面的实验数据可以看到，使用分组卷积和通道混淆后，相比于 ESPCN 的标准卷积，图像帧的超分精度几乎没有下降。

3 OpenGL ES 的实现方案

网络模型在移动端部署时，成熟的推理框架在系统底层都使用 OpenCL 异构并行接口，将卷积计算部署在 GPU 中运行，常见的并行策略有：Kubeflow 异构算力调度[25]; S-NUCA 异构处理[26]; CPU 与 GPU 结合优化[27]等。但是 OpenCL 的并发主要依赖 SIMD 并行计算，其并发调度依赖于系统本身对于工作项的调用， 并发粒度比较粗。本文采用 OpenGL ES 图形处理接口，可以将并发粒度细化到像素级别，从而达到更好的并发性能。

3.1 渲染管线结构

在本文 OpenGL ES 的方案中，依据推理框架将每个组件实现相应的渲染管线。考虑到系统平台兼容性，本文使用 OpenGL ES 3.0 标准接口，简称 GLES3.0。在整个推理框架中，输入是原始小图的 YUV 数据，最终输出超分放大后的 RGB 数据。每个渲染管线中涉及的数据结构如图 4 所示：

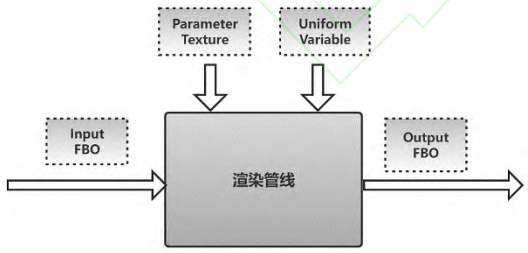

Figure 4 Data Structure of One Render Pipeline

图 4 单个渲染管线的数据结构图

（1） Input FBO:就是卷积计算的输入通道数据，该数据在渲染处理过程中是只读的，因此直接生成相应的纹理上传显存。

（2） Output FBO: 就是卷积输出通道数据，是前一个管线的输出通道数据或者原始小图 YUV 数据，这部分数据存储要支持可读可写。在 GLES3.0 中使用帧缓存对象 FBO（FrameBufferObject）来存储运算的通道数据，每个 FBO 都会绑定相应的渲染缓冲区 RBO（RenderBufferObject）和纹理（Texture）。

（3） Parameter Texture: 模型参数纹理，主要由卷积权重参数或者偏置参数组成，在渲染处理过程中是只读，因此可以直接将权重参数和偏置参数纹理生成相应的纹理上传到显存。

（4） Uniform Variable:是指卷积核大小、计算图宽高等配置信息，可以直接通过 Uniform 变量设置

3.2 浮点数据的处理

在 GLES3.0 的 Shader 中的浮点精度是按照 IEEE-754 标准来定义的，分别提供了三种精度的浮点数据：highp(高精度，32 位)；mediump(半精度，16 位，默认精度)；lowp(低精度，8 位)，其相应的表示值域范围表 1 所示：

Table 1 Range of float data type

表 1 浮点数据范围[28]

Qualifier	Floating Point Range	Floating Point Magnitude Range	Floating Point Precision	Integer Range	
				Signed	Unsigned
highp	As IEEE-754 $(-2^{126}, 2^{127})$	As IEEE-754 0.0,$(2^{-126}, 2^{127})$	As IEEE 754 relative: 2^{-24}	$[-2^{31}, 2^{31}-1]$	$[0, 2^{32}-1]$
mediump (minimum requirements)	$(-2^{14}, 2^{14})$	$(2^{-14}, 2^{14})$	Relative: 2^{-10}	$[-2^{15}, 2^{15}-1]$	$[0, 2^{16}-1]$
lowp (minimum requirements)	$(-2, 2)$	$(2^{-8}, 2)$	Absolute: 2^{-8} / 2^{-9} signed/unsigned	$[-2^{8}, 2^{8}-1]$	$[0, 2^{9}-1]$

本文实现方案中所有权重数据和通道数据值和计算范围都在 $[-9.00001 * 10^{-2}, 1.0)$ 之间，因此本文 shader 代码中所有的浮点计算直接使用 mediap 半精度浮点即可。GLES3.0 支持浮点纹理数据格式，其中 GL_16F 指定使用 2 字节来存储 1 个像素半精度数据，因此在本文的纹理中直接使用 GL_16F 四字节来存储浮点数据。

3.3 模型参数在纹理中布局策略

模型参数包括卷积权重参数和偏置参数两种，都是只读数据，将模型参数作为纹理数据进行显存布局，然后在引擎初始化的时一次性上传到显存。

（1）卷积权重参数： 其数据量比较多，卷积中有 5*5 和和 3*3 两种系数矩阵。本文的纹理布局方案是同一个输入通道的卷积参数放在一行中，不同输出通道的卷积参数分别放在不同的行中，布局后的权重参数纹理宽高可以按照公式(6)进行计算：

$$\begin{cases} ParamTxrWidth = \mathrm{K} * \mathrm{InChannels} \\ ParamTxrHeight = G * \mathrm{K} * \mathrm{OutChannels} \end{cases} \quad (6)$$

其中 **InChannels** 是每组卷积输入通道数量；**OutChannels** 是每组卷积输出通道数量；**G** 是分组数量。

Conv1 组件中是采用标准卷积过程，输入 1 通道、输出 36 通道，每个卷积系数是 5*5，相应的卷积权重系数纹理按照图 5 所示进行布局，根据公式(6)计算，纹理的宽高是：5*180。

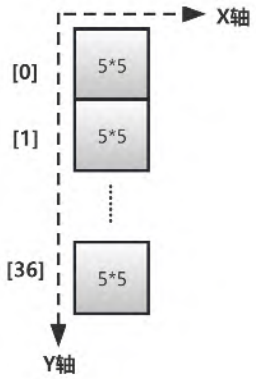

Figure 5 Texture Layout of Conv1 Weight Data

图 5 Conv1 卷积权重数据的纹理布局

在 Conv2 中有 36 路输入通道、18 路输出通道，分为 6 个组，每组 6 个输入通道和 3 个输出通道，卷积权重系数纹理按照图 6 的方式进行布局。根据公式(6)计算方式，卷积系数的纹理宽高为: 18*54。 同理，Conv3 组件中的卷积权重系数也是类似布局方式。

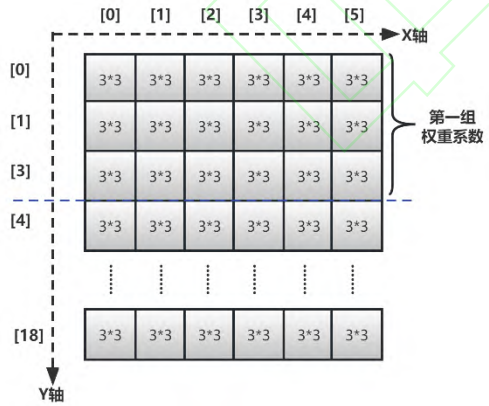

Figure 6 Texture Layout of Conv2 Weight Data

图 6 Conv2 卷积权重数据的纹理布局

（2） 偏置参数：每个卷积组件的偏置参数不同，最少的只有 9 个浮点数，最多的有 36 个浮点值，直接使用宽高为 $(OutChannel * 1)$的纹理图像来存储偏置参数值。

3.4 通道数据在纹理中布局策略

通道数据的显存布局类似模型参数数据布局，但是由于其每一个通道数据量较大。如果直接将通道数据按照所有一行或者一列排列，形成的纹理宽度或高度会超出 GPU 本身纹理大小限制，因此通道数据需要根据通道数量在水平和垂直两个方向上布局到一张纹理图中。

如图 7 所示，以输入分辨率为（320*240）视频帧为例，在 Conv1 组件中输出 36 个通道数据，如果直接将所有通道数据按列排列，纹理大小为：320*(36*240)=320*8640，而大部分的移动手机支持的纹理宽高达不到 8640 像素。

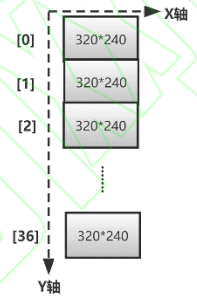

Figure 7 Texture Layout of Data in Vertical Direction

图 7 通道数据仅垂直方向进行纹理布局

考虑水平方向和垂直方向上布局的通道数乘积是总通道数，反推求两个方向上的通道数量，可以直接进行开方操作，即：

$$\begin{cases} layoutCols = \lceil \sqrt{totalChannels} \rceil \\ layoutRows = \lceil \sqrt{totalChannels} \rceil \end{cases} \quad (7)$$

例如：在 Conv2 的 36 个输出通道中，按照上述公式计算，可以形成 (6 行*6 列) 的纹理布局。这样可以计算出所有通道构成的纹理大小

$$\begin{cases} textureWidth = layoutRows * W \\ textureHeight = layoutCols * H \end{cases} \quad (8)$$

对于 320*240 大小的图像来说，存放 36 通道数据的纹理大小就是 1920*1440，这个大小完全满足各种中低端平台的 GPU 显存要求。这样 Conv1 组件的输出通道按照图 8 所示进行纹理布局。

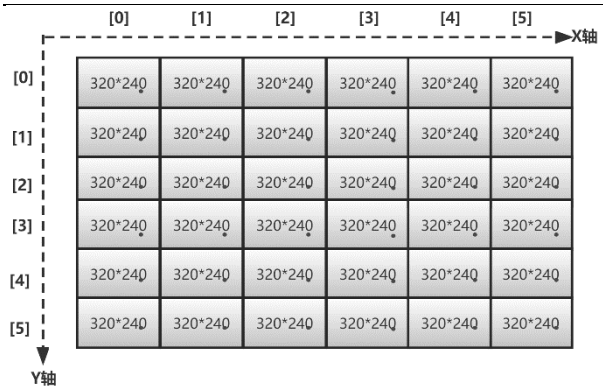

Figure 8 Texture Layout of Conv1 Output Channel Data

图 8 Conv1 输出通道数据的纹理布局

当公式(7)开方结果不为整数时，则向上取整，然后不断增加 1，直到可以分配水平和垂直方向布局。例如：Conv2 组件有 18 个输出通道，直接开方后是小数 4.2426, 向上取整为 5，仍不能被 18 整除，则增加 1 变为 6。对于 18 输出通道，可以布局为 3*6 的纹理布局，如图 9 所示。

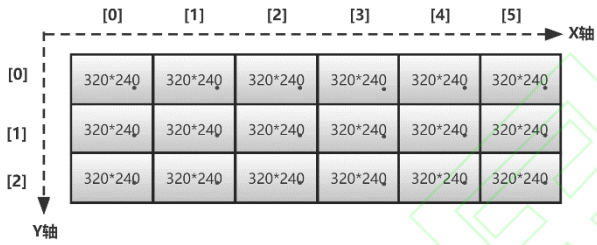

Figure 9 Texture Layout of Conv2 Output Channel Data

图 9 Conv2 输出通道数据的纹理布局

这样将通道数据在水平和垂直两个方向上布局到纹理中，可以满足手机硬件对于纹理的支持。

3.5 渲染并行化

在 OpenGL ES 渲染管线进行并行化处理时，具体是在像素着色器（Fragment-Shader）实现，并行粒度对应输出通道的单个像素，即输出通道的每个输出数据的计算都是并行进行的。每个渲染管线的并发度为：

$$Concurrency = (W*H*OutChannels) \quad (9)$$

以 320*240 分辨率的图像为例，在 Conv2 组件进行卷积计算是，总计有 1382400 个数据要进行并发的卷积和激活操作，完整的渲染算法如下：

算法 1 Conv2 组件渲染算法

```
输入: imgWidth: 图像宽度
      imgHeight: 图像高度
      outTxrCoord: 渲染像素点归一化坐标
输出: outColor: 渲染目标像素

//
// 计算绘制像素点在输出纹理中的位置
outX ← (outTxrCoord.x * (imgWidth * 6));
outY ← (outTxrCoord.y * (imgHeight * 3));
outX ← clamp(outX, 0, (imgWidth * 6 - 1));
outY ← clamp(outY, 0, (imgWidth * 6 - 1));
//
// 计算绘制像素点在输出纹理中的通道索引
outChnlIdx.x ← outX / imgWidth;
outChnlIdx.y ← outY / imgHeight;
outChnlIndex ← (outChnlIdx.y * 6 + outChnlIdx.x);
//
// 计算绘制像素点在一张 Image 中的坐标
centerX ← (outX % imgWidth);
centerY ← (outY % imgHeight);
//
// 累加初值为偏置值
sum ← biasParam[outChnlIndex];
//
// 根据输出通道索引，计算输出通道所在的组号（输出通道每 3 个一组）
groupIndex ← outChnlIndex / 3;
//
// 根据组号，计算对应输入通道的位置范围
beginChnlPos ← groupIndex * 6;
endChnlPos ← beginChnlPos + 6;
//
// 针对每一个输入通道，计算对应的输入数据位置和权重系数位置
for shuffleChnlPos ← beginChnlPos to (endChnlPos-1) do
    // 根据 shuffle 后的通道位置，反向计算原始输入通道
    row ← shuffleChnlPos / 6;
    col ← shuffleChnlPos % 6;
    inChnlIndex ← (col * 6 + row);
    // 输入通道的 3*3 卷积和累加
    cnnValue ← CNN(inChnlIndex);
    sum ← (cnnValue + sum);
end
//
// 激活函数处理结果存储到输出像素值中
outColor.r ← tanh(sum);
```

本文以 Conv2 组件的输出通道中一个坐标为(0.2688, 0.5667)的输出像素点为例，来说明整个计算过程。设为简化坐标系原点为左上角，该像素点的输出通道纹理值计算流程如图 10 所示：

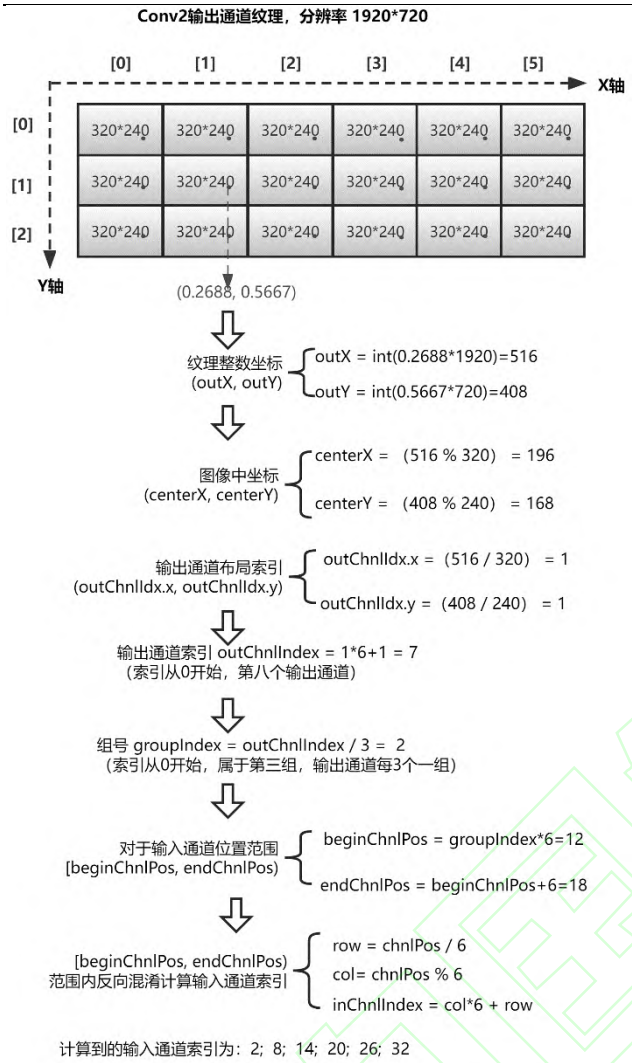

Figure 10 Render Algorithm of Conv2 Component

图 10 Conv2 组件渲染算法流程

通过上述的算法计算到相应的输入通道、卷积权重系数、输出位置后，可以直接在相应的像素位置进行采样，再进行正常的分组卷积。以 3*3 的卷积处理为例，假设每组输入通道数 N,那么某一路输出通道中某个像素点的核心卷积计算如下：

$$\sum_{k=0}^{N-1}\sum_{i=-1}^{1}\sum_{j=-1}^{1}(Weight[x+i,y+j]*InData[k,x+i,y+j]) \quad (10)$$

对应的时间复杂度是： $O(N*K*K)$

对于 Conv2 组件来说，每组有 6 路输入通道，3*3 卷积，每一个输出像素点的卷积需要进行 288 次的采样和乘法计算。

在分组卷积计算过程中，反向计算到输入通道索引后，可以直接根据分组情况来获取到分组权重数据，计算权重数据如图 11 所示。

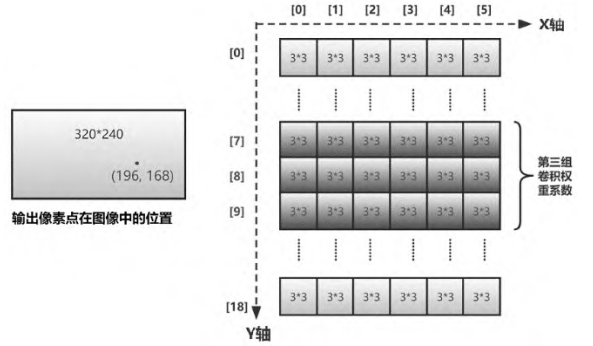

Figure 11 Output Pixel and Weight of Conv2 Component

图 11 Conv2 组件中计算的像素点和权重数据

同样，由输入通道索引也可以直接在输入纹理中找到参与卷积计算的输入通道数据，在本例子中参与计算的输入通道数据如图 12 所示。

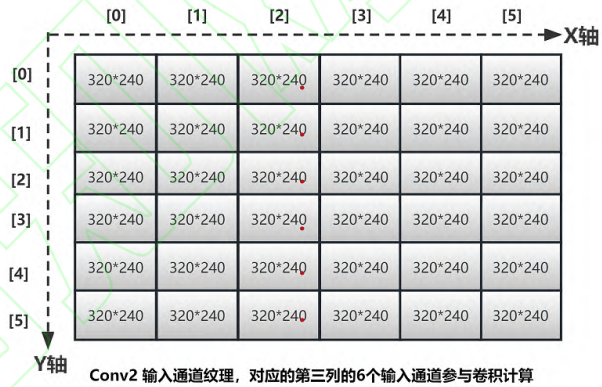

Figure 12 Input Channels of Calculate in Conv2

图 12 Conv2 组件中参与计算的输入通道

4 实验结果及分析

本章节将从不同维度的实验数据进行分析。

4.1 超分辨率精度分析

本文采用的 OGSR 超分算优先考虑推理计算性能，在保证性能前提下尽可能提升图像质量。在精度对比方面重点考虑两方面：一是与标准 ESPCN 对比，超分放大后的图像质量是否能保持质量没有明显下降；二是通过融合实验对比不同变量对于图像质量和性能的影响。

（1） OGSR 与 ESPCN 标准卷积的图像质量对比。本文采用采用开源的 CelebA 数据集，其中前面 16700 张大小图片作为训练数据集，后面 425 张大小图片作为测试数据集。在训练时两种网络框架都使用相同的轮次（epoch=100），确保训练条件相同。采用标准 ESPCN 模型和 OGSR 模型分别将测试集中的小图进行 3 倍超分放大，对比放大后图像与原始大图之间的 PSNR。从图 13 的的测试结果可以看到，在大量图片对比情况下，对

比两种方案与原图的峰值信噪比几乎是一致的,两者误差<2%，说明 OGSR 相对于标准 ESPCN 模型的图像质量，没有明显下降，达到预期要求。

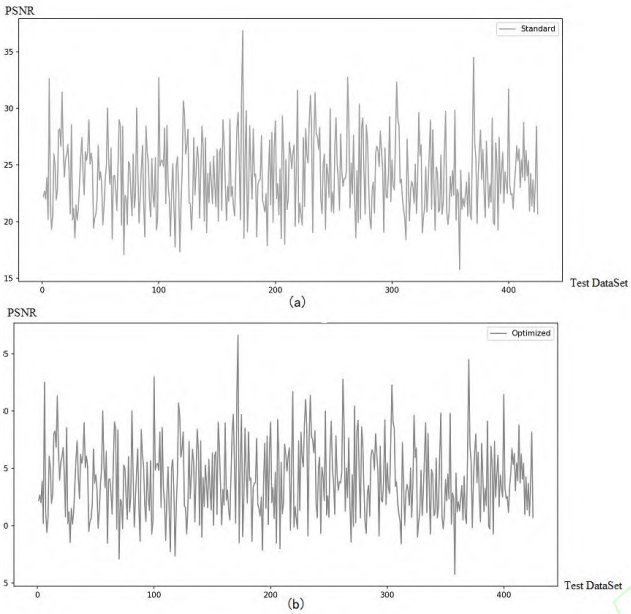

Figure 13　PSNR of standard and Optimized framework

图 13　性能对比 (a)标准超分框架 PSNR 和(b) 优化后框架 PSNR

（2）消融实验。为验证 OGSR 模型在质量和性能方面的平衡度最优性，对模型中的关键的两个卷积组件 Conv2 和 Conv3 模块进行消融实验。本文使用腾讯 NCNN 框架进行模型部署，采用不同的模型对同一张图片进行 3 倍超分放大，具体变量因素控制如下：

I. ESPCN 模型：Conv2 和 Conv3 组件对输入通道都直接采用标准卷积，不分组、不混淆。

II. Group-Conv23 模型：Conv2 组件和 Conv3 组件的输入通道都进行分组，但不混淆处理。

III. Shuffle-Conv2 模型：Conv2 组件的输入通道进行分组并且混淆处理，Conv3 输入通道直接标准卷积。

IV. Shuffle-Conv3 模型：Conv2 输入通道直接标准卷积，Conv3 组件的输入通道进行分组并且混淆处理。

V. OGSR 模型：Conv2 和 Conv3 两个组件的输入通道都进行分组和混淆处理。

Table 2 PSNR & SSIM & Cost of different models

表 2　不同模型的 PSN 和 SSIM、性能

不同模型	PSNR(db)	SSIM	推理耗时 (ms)
ESPCN	29.846	0.923	232
Group-Conv23	24.458	0.726	97
Shuffle-Conv2	27.230	0.874	118
Shuffle-Conv3	26.239	0.828	127
OGSR	29.1232	0.901	98

从表 2 的消融结果来，ESPCN 的超分质量最高，但是整个推理性能也是最耗时的；Group-Conv23 虽然推理性能很高，但是对应的超分质量也最差；Shuffle-Conv2 和 Shuffle-Conv3 是单个组件的输入道分组混淆，相比于纯分组卷积质量有所提升，但是性能也有明显下降；最后的 OGSR 模型在超分质量上最接近于 OGSR，同时推理性能也是最高的。

4.2 计算性能分析

目前已经有很多成熟的第三方框架支持神经网络模型在移动端进行部署和推理。常见的移动端推理框架有：Google 公司推出的轻量级的机器学习框架，专为移动设备设计的 TensorFlow Lite[29]；Apple 公司推出的 Core ML，可支持深度学习和机器学习模型的推理，但仅适用于 iOS 系统[30]；亚马逊推出开源的深度学习框架 MXNet[31]，支持移动设备推理和训练，并提供小型化模型设计和迁移学习等功能；而 Facebook 公司则在 PyTorch 基础上直接退出了 PyTorch Mobile，用来支持移动设备上的神经网络部署；国内的 Tencent 公司开源的 NCNN 推理框架是一个高性能、轻量级的深度学习推理框架。

在 IPC 类设备采集的视频帧通常都是 QVGA(320*240)和 nHD(640*360)分辨率为主，这里表 3 和表 4 分别是在 OPPO FindX 手机上对这两个分辨率进行三倍超分放大的性能数据,其中主流推理框架按照其官网提供标准方式调用，这里 OpenCL 算法部署方式，是直接使用 OpenCL 接口实现的异构并行算法加速。

Table 3 Performance for (320*240) resolution

表 3　(320*240)分辨率视频帧超分性能

推理框架	内存占用 (MB)	单帧平均耗时 (ms)
PyTorchMobile	89	203
TensorFlow Lite	68	102
腾讯 NCNN	54	98
OpenCL 实现算法部署	52	48
本文 OGSR 算法	47	34

Table 4 Performance for (640*360) resolution
表 4 (640*360)分辨率视频帧超分性能

推理框架	内存占用 (MB)	单帧平均耗时 (ms)
PyTorchMobile	312	312
TensorFlow Lite	237	204
腾讯 NCNN	214	187
OpenCL 实现算法部署	203	117
本文 OGSR 算法	196	63

从表 3 和表 4 的性能数据中可以看出，对比目前提供主流的移动端推理框架，本文采用的 OGSR 超分实现方案在性能上具有明显的优势。主要原因有两点：

（1）这些成熟框架在加载模型框架进行前向推理时，采用的时标准的模型计算图，每个计算节点依次顺序计算，特别是 Reshape 和 Transpose 操作需要较多的内存交换，这一块是比较耗时的。而在 OGSR 实现方案中，实际内存并不做转换，只是在通道数据采样时，反向计算所在位置，这样节省了所有 Reshape 和 Transpose 的操作

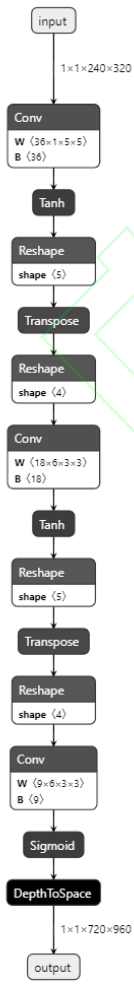

Figure 14 Computation Graph of Model
图 14 模型计算图

（2）主流的移动端推流框架，底层主要是采用 Open CL 的异构并发进行硬件加速优化。Open CL 虽然可以利用不同的工作组和工作项策略，分配不同的核进行高并发计算，但是属于粗粒度的并发处理。而本文的 OGSR 底层算法实现是基于像素着色器，使用 OpenGL ES 可以直接控制到像素级的并发度，从而也获得更好的性能。

4.3 兼容性验证

不同的手机芯片平台上对于 OpenGL ES 有不同的硬件支持力度。Android 系统从 2013 年发布的 4.3 版本开始就已经支持 OpenGL ES 3.0 标准，苹果公司的 iOS 系统从 2016 年开始就直接支持 OpenGL ES3.0.0 标准，因此本文基于 OpenGL ES 3.0 规范的 OGSR 推理框架可以在各个平台上得到支持。表 5 是使用几款不同芯片平台的手机对(320*240)分辨率进行 3 倍超分的性能，可以看出在不同的芯片平台中，本文的实现方案都可以达到本文 15FPS 的目标帧率。

Table 5 Performances on Different Mobile Phone
表 5 不同机型对于(320*240)分辨率超分性能

手机型号	芯片平台	GPU 型号	单帧平均耗时 (ms)
OPPO Find X	高通骁龙 845	高通 Adreno630	34
Redmi K60	高通骁龙 8+Gen	高通 Adreno730	37
华为 Meta 30	麒麟 990	Mali-G76	47
小米 10 Lite	高通骁龙 765G	高通 Adreno 620	42

5 结束语

本文提出了一种移动平台上的实时视频超分优化实现方案 OGSR，该方案先采用分组卷积和通道混淆的方式来降低神经网络模型中的计算量；然后使用 OpenGL ES 图形 API 接口进行神经网络模型部署，从而最大程度的利用 GPU 硬件资源提升计算并发度；最后在 GPU 执行的着色器代码中进行算法指令级优化，进一步提升算法性能。实验结果表明，对于 QVGA(320*240)分辨率的视频帧进行三倍超分，在中高端手机机型上可以达到 15~24fps 的帧率，满足实时性需求。

参考文献

[1] Keys R. Cubic convolution interpolation for digital image processing[J]. IEEE transactions on acoustics, speech, and signal processing, 1981, 29(6): 1153-1160.

[2] Huang J B, Singh A, Ahuja N. Single image super-resolution from transformed self-exemplars[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 5197-5206.

[3] Chao Dong, Chen Chang, He Kai-ming. Image super-resolution

using deep convolutional networks[C]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 1-5

[4] Kim J, Lee J K, Lee K M. Accurate image super-resolution using very deep convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 1646-1654.

[5] Kim J, Lee J K, Lee K M. Deeply-recursive convolutional network for image super-resolution[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 1637-1645.

[6] Zhou Y, Li Z, Guo C L, et al. Srformer: Permuted self-attention for single image super-resolution[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023: 12780-12791.

[7] Wang Y, Yang W, Chen X, et al. SinSR: Diffusion-Based Image Super-Resolution in a Single Step[J]. arXiv preprint arXiv:2311.14760, 2023.

[8] Yi X, Xu H, Zhang H, et al. Diff-retinex: Rethinking low-light image enhancement with a generative diffusion model[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023: 12302-12311.

[9] Caballero J, Ledig C, Aitken A, et al. Real-time video super-resolution with spatio-temporal networks and motion compensation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4778-4787.

[10] Wang X, Chan K C K, Yu K, et al. Edvr: Video restoration with enhanced deformable convolutional networks[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2019: 0-0.

[11] Tian Y, Zhang Y, Fu Y, et al. Tdan: Temporally-deformable alignment network for video super-resolution[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020: 3360-3369.

[12] Liang J, Cao J, Fan Y, et al. Vrt: A video restoration transformer[J]. IEEE Transactions on Image Processing, 2024.

[13] Chen Z, Long F, Qiu Z, et al. Learning Spatial Adaptation and Temporal Coherence in Diffusion Models for Video Super-Resolution[J]. arXiv preprint arXiv:2403.17000, 2024.

[14] Zhou X, Zhang L, Zhao X, et al. Video Super-Resolution Transformer with Masked Inter&Intra-Frame Attention[J]. arXiv preprint arXiv:2401.06312, 2024.

[15] Zhang X, Zeng H, Zhang L. Edge-oriented convolution block for real-time super resolution on mobile devices[C]//Proceedings of the 29th ACM International Conference on Multimedia. 2021: 4034-4043.

[16] Berger G, Dhingra M, Mercier A, et al. QuickSRNet: Plain Single-Image Super-Resolution Architecture for Faster Inference on Mobile Platforms[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023: 2186-2195.

[17] Shi W, Caballero J, Huszár F, et al. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 1874-1883.

[18] Sandler M, Howard A, Zhu M, et al. Mobilenetv2: Inverted residuals and linear bottlenecks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 4510-4520.

[19] Zhang X, Zhou X, Lin M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 6848-6856.

[20] Ma N, Zhang X, Zheng H T, et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 116-131.

[21] Ding X, Zhang X, Ma N, et al. Repvgg: Making vgg-style convnets great again[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021: 13733-13742.

[22] Wang A, Chen H, Lin Z, et al. Repvit: Revisiting mobile cnn from vit perspective[J]. arXiv preprint arXiv:2307.09283, 2023.

[23] Gou W, Yi Z, Xiang Y, et al. SYENet: A Simple Yet Effective Network for Multiple Low-Level Vision Tasks with Real-time Performance on Mobile Device[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023: 12182-12195.

[24] Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures[J]. Communications of the ACM, 2009, 52(4): 65-76.

[25] Sun Yi, Wang Huimei, Ming Ming, etc Research on Kubeflow Heterogeneous Computing Power Scheduling Strategy [J]. Computer Engineering, 2024,50 (2): 25-32.

[26] Zhou Yitao, Li Yang, Han Chao, et al. Task scheduling and thermal management system for S-NUCA heterogeneous processors [J]. Computer Engineering, 2024,50 (2): 196-205.

[27] Li Bo, Huang Dongqiang, Jia Jinfang, et al. Optimization of Heterogeneous Template Computing Based on CPU and GPU [J]. Computer Engineering, 2023,49 (4): 131-137.

[28] Munshi A, Ginsburg D, Shreiner D. OpenGL ES 2.0 programming guide[M]. Pearson Education, 2008.

[29] Li Shuangfeng TensorFlow Lite: End side machine learning framework [J] Computer Research and Development, 2020, 57 (9): 1839-1853.

[30] Li Lin, Ruopu Detailed explanation of Core ML framework and smart speaker HomePod [J] Computer and Network, 2017, 43 (12): 34-35.

[31] Gan Rundong, Shen Shuyin, Zhang Yuzhe Multidimensional linear data processing based on OpenCL kernel function in MXNet framework [J] Frontiers of Data and Computing Development, 2022, 4 (2): 29-38.

附中文参考文献：

[25] 孙毅,王会梅,鲜明,等.Kubeflow 异构算力调度策略研究[J].计算机工程,2024,50(2):25-32.

[26] 周义涛,李阳,韩超,等.适用于 S-NUCA 异构处理器的任务调度与热管理系统[J].计算机工程,2024,50(2):196-205.

[27] 李博,黄东强,贾金芳,等.基于 CPU 与 GPU 的异构模板计算优化研究[J].计算机工程,2023,49(4):131-137.

[29] 李双峰. TensorFlow Lite: 端侧机器学习框架[J]. 计算机研究与发展, 2020, 57(9): 1839-1853.

[30] 李林, 若朴. 详解 Core ML 框架及智能音箱 HomePod[J]. 计算机与网络, 2017, 43(12): 34-35.

[31] 甘润东, 沈舒尹, 张宇哲. MXNet 框架中基于 OpenCL 核函数的多维线性数据处理[J]. 数据与计算发展前沿, 2022, 4(2): 29-38