融云 RONGCLOUD

X-Live RTC·进阶实战高手课

WebRTC

视频图像超分技术

融云 RONGCLOUD

X-Live RTC·进阶实战高手课

黄震坤

武汉大学计算机学院
国家多媒体软件工程技术研究中心博士

研究方向：
视频信息处理、视频压缩

融云视频编码算法专家

工作内容：
基于深度学习的视频压缩算法优化工作
WebRTC 视频编码和传输优化

RTC 服务的需求和现状

- 随着 5G 的发展应用，人们对音视频通信的品质要求不断提高
- 终端丰富多样，网络和使用场景复杂多变，用户带宽不足仍然存在
- 但在较低带宽下，获取更高清晰度的图像和视频需求更为迫切

高清视频 - 影响视频清晰度的主要因素

- 理论上对于同一种编码器而言，在分辨率不变的前提下，码率越大视频质量越高
- 从视觉方面来看，在特定编码器和分辨率的情况下，码率存在一个最优值

不同分辨率视频的推荐码率

	1280 x 720	1920 x 1080
极低码率	500 kbps	1 Mbps
低码率	1 Mbps	2 Mbps
中等码率	2 Mbps	4 Mbps
高码率	4 Mbps	8 Mbps
极高码率	8 Mbps	16 Mbps

RTC 常见编码方案

- **Google 开源 WebRTC**
  - 各行业可以基于网页或者客户端，快速进行音视频实时通信
  - 各大公司也基于 WebRTC 进行改进和应用

- **WebRTC 视频编码自带的是 H264、VP8、VP9 等编码器**
  - 对 1080P 视频进行压缩，视频设定为中等码率需要 4Mbps
  - 当用户带宽不足时，需要用联播（Simulcast）或者 SVC 来解决

存在的问题

- 无论是否使用 Simulcast 或 SVC
  - 由于带宽不足，终端用户只能传输低分辨率的视频
  - 但对于习惯了高清的用户，又非常希望能观看高清视频

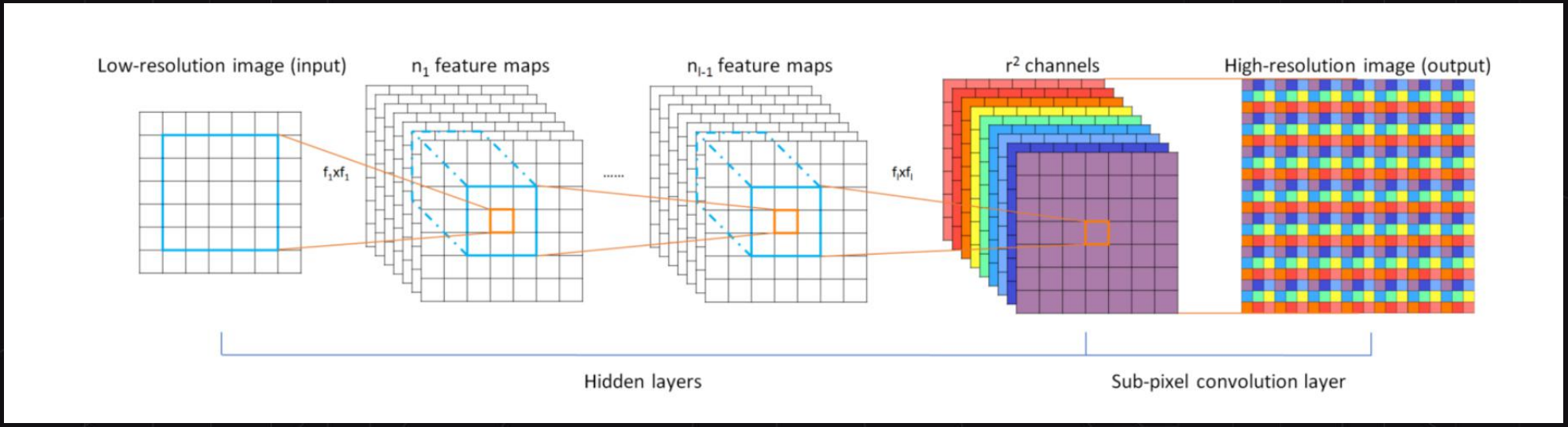
- 在不改变编码器的情况下，即仍然使用 H264 或 VP8、VP9，保证通用性
- 如何依然在终端上获得高清视频的体验，成为学术界和产业界需要共同研究和解决的课题

解决方案
-图像超分辨率技术

- 图像超分辨率技术（Super Resolution，SR）是计算机视觉中提高图像和视频分辨率的重要技术，可以将一幅低分辨率图像重建出高分辨率图像
- 通俗地说，就是当我们放大一张尺寸较小的图像时，会出现模糊的现象，超分辨率重建就是将原图中一个像素对应的内容，在放大之后的图中用更多的像素来表示，让图像尽可能地清晰

适合 WebRTC 传输的图像超分辨率技术

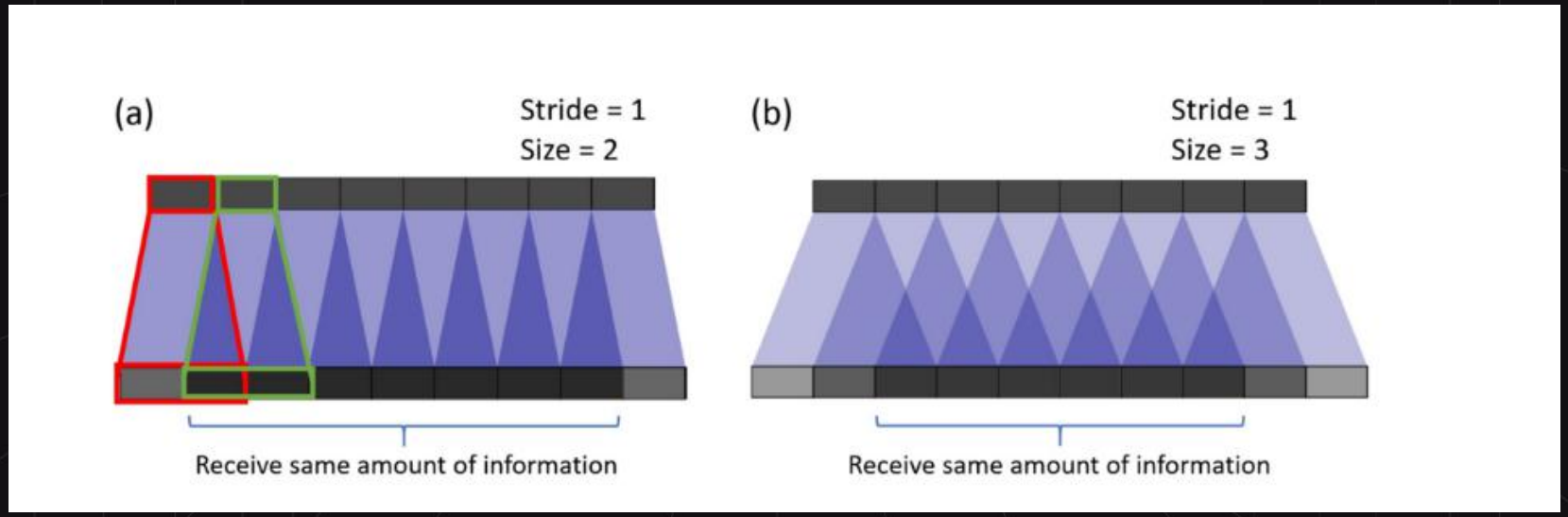
- 在 WebRTC 中，图像超分辨率算法需要同时满足实时超分和好效果的双重要求
- 通过对图像超分辨率技术的调研，参考了 2016 年发表在 CVPR 上的论文（Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network）的方案
- 简单来说，ESPCN 采用两个卷积层来提取特征，紧接着一个 sub-pixel convolution 进行上采样



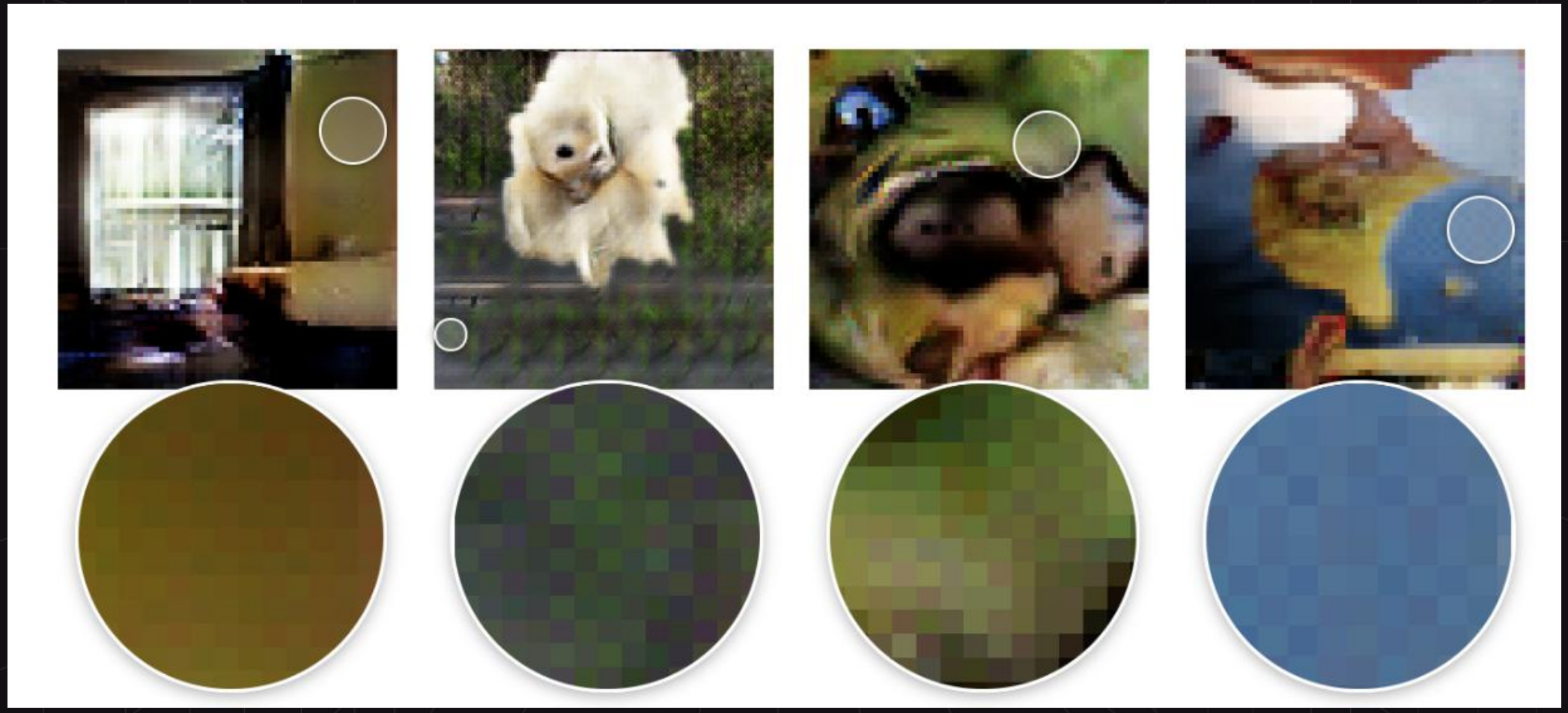
ESPCN 模型的分析

ESPCN 模型的创新点在于，提出了用 sub-pixel convolution 的方式代替反卷积进行上采样，解决了如下两个问题：

1. 反卷积在高分辨率空间，而在高分辨率进行反卷积计算量很大
2. 解决了反卷积的棋盘效应（checkerboard issue）

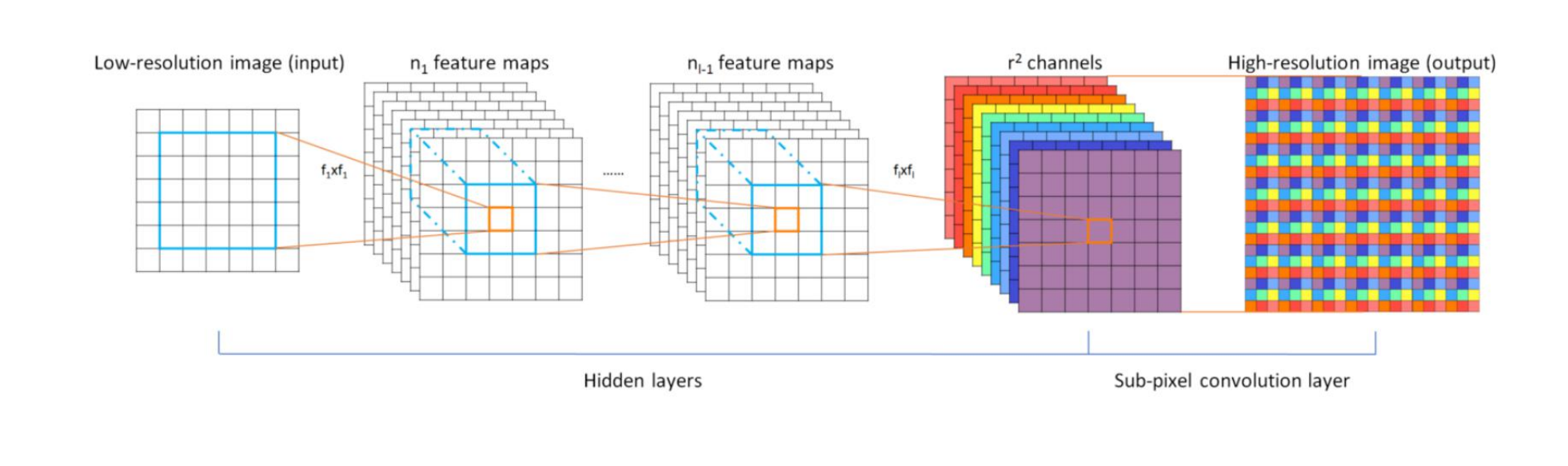


棋盘效应



Sub-pixel convolution

- Sub-pixel convolution 通过将深度转换成空间来进行工作
- 在低分辨率图像中的来自多个通道像素，被重新组织成一个单通道的高分辨率图像
- 如输入的图像为 5*5*4，通过 Sub-pixel convolution 可以转换成单通道的 10*10 超分辨图像



ESPCN 的实现

采用 tensorflow2.0 的 API， depth_to_space 即 tensorflow 实现的 Sub-pixel convolution 函数

```
inputs = keras.Input(shape=(None, None, 1))

conv1 = layers.Conv2D(128, 5, activation="tanh", padding="same")(inputs)

conv2 = layers.Conv2D(64, 3, activation="tanh", padding="same")(conv1)

conv3 = layers.Conv2D((upscale_factor*upscale_factor), 3, activation="sigmoid", padding="same")(conv2)

outputs = tf.nn.depth_to_space(conv3, upscale_factor, data_format='NHWC')

model = Model(inputs=inputs, outputs=outputs)
```

ESPCN 的主观效果



从左到右分别为：原始图像、bicubic 插值、ESPCN 模型

WebRTC 中的模型实现

将图像超分辨率技术应用于 WebRTC 的实现原理如下：

- 对原始高分辨率视频进行下采样生成低分辨率的视频
- 在带宽有限的网络中发送低分辨率的视频
- 采用超分辨率技术在接收终端对收到的低分辨率视频每一帧进行图像超分辨率

不同终端的超分辨率优化

- 对于 GPU 性能较好的 PC 机，ESPCN 模型较小，能够对视频进行直接实时超分
- 这里可以采用 pytorch 模型来建模，模型如下所示：

```
self.feature_maps = nn.Sequential( nn.Conv2d(1, 64, (5, 5), (1, 1), (2, 2)),
                                   nn.Tanh(),
                                   nn.Conv2d(64, 32, (3, 3), (1, 1), (1, 1)),
                                   nn.Tanh(),)
self.sub_pixel = nn.Sequential(nn.Conv2d(32, 1 * (upscale_factor ** 2), (3, 3), (1, 1), (1, 1)),
                               nn.PixelShuffle(upscale_factor),    )
```

- 其中 PixelShuffle 就是 Sub-pixel convolution 在 pytorch 里面的实现方式

不同终端的超分辨率优化

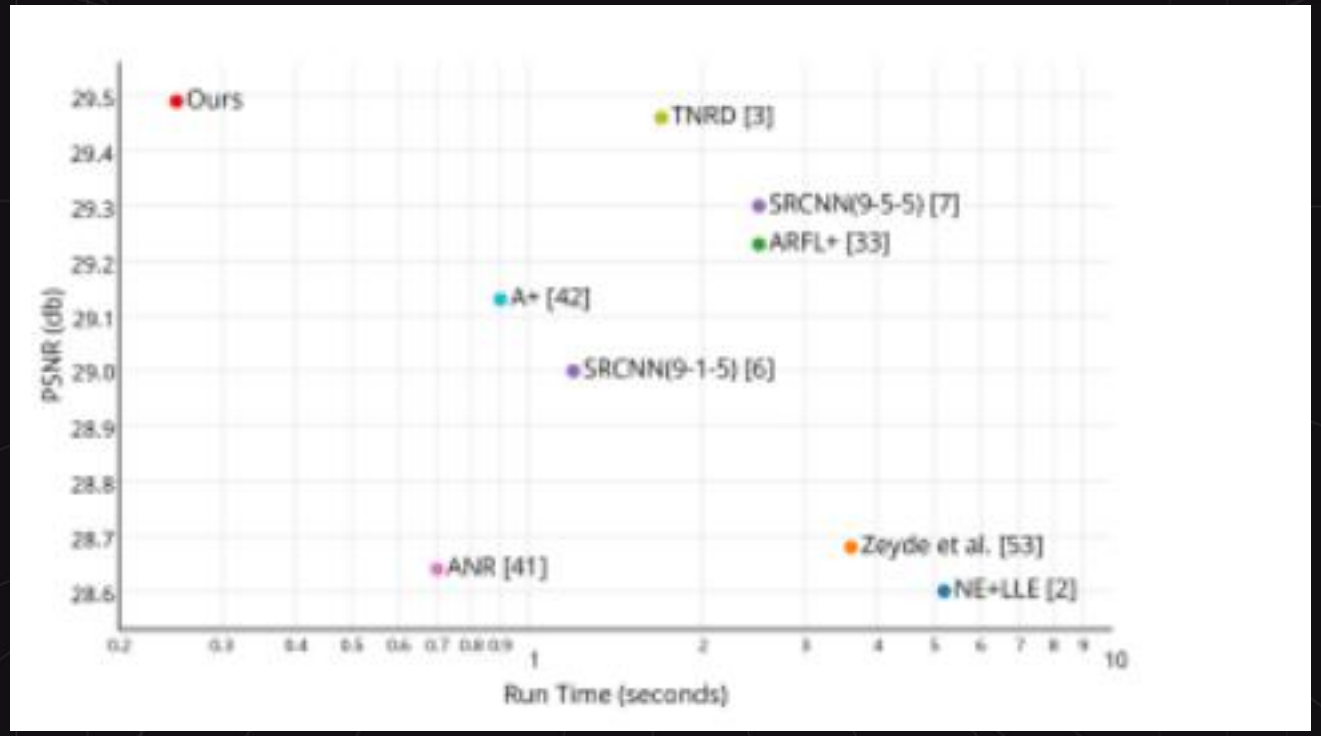
- 训练的数据一般可以采用常见的图像数据库，比如 PASCAL VOC、COCO 等
- 也可以根据行业特点采集相应数据集，让模型推理更有针对性

- 为了增大数据库，采用图像随机裁剪（ Random Crop），该方法不仅能够对数据库进行扩容，而且可以弱化数据噪声与提升模型稳定性采用 Pytorch 对训练好的模型进行量化，通过 int8 量化，模型中的部分数据从 float 变成 int 型，采用 libtorch 集成到 WebRTC 中

➢ 实验结果显示，未量化之前 ESPCN 模型大小为 95KB，量化之后为 27KB

不同终端的超分辨率优化

- 个人的实验结果，见下图测试数据



更多内容，可以参考论文 “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network” CVPR2016

不同终端的超分辨率优化

手机端 GPU 算力不如 PC 平台，需要选择轻量级的模型来进行模型推理

对于视频超分算法的实时性主要取决以下两个方面：

- 模型的复杂度：ESPCN 作为只有两个卷积层和一个 sub-pixel 卷积层作为上采样层，足够简单
- 输入图像的分辨率和通道数：输入像素的总量

对于 ESPCN 模型而言，仅仅输入 Y 通道的；对于 UV，直接进行双三次插值（Bicubic interpolation）

不同终端的超分辨率优化

- 鉴于 tensorflow 框架的成熟性，采用 tensorflow 对 ESPCN 模型进行针对性部署
- 采用 tensorflow2.8 和对应的 tensorflowlite 进行实验
- Tensorflowlite的模型量化采用 EXPERIMENTAL_TFLITE_BUILTINS_ACTIVATIONS_INT16_WEIGHTS_INT8 的方式

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

```
converter.target_spec.supported_ops = [  tf.lite.OpsSet.EXPERIMENTAL_TFLITE_BUILTINS_ACTIVATIONS_INT16_WEIGHTS_INT8]
```

不同终端的超分辨率优化

以手机端 GPU 推理，即 TFLite GPU 的方式举例

- 测试手机为小米 Redmi K40，处理器是骁龙 870，属于中端手机

从前述的分析可知，影响推理速度因素主要是输入数据数量和模型的大小

```
conv1 = layers.Conv2D(128, 5, activation="tanh", padding="same")(inputs)
conv2 = layers.Conv2D(64, 3, activation="tanh", padding="same")(conv1)
```

- 上述的 tensorflow 实现中，滤波器的个数为 128，经过骁龙 870 的 GPU 测试，从 480*270 放大到 1920*1080，每帧大概 50 多毫秒，即 FPS 10 多帧。为了进一步达到实时，可以改为

```
conv1 = layers.Conv2D(64, 5, activation="tanh", padding="same")(inputs)
conv2 = layers.Conv2D(32, 3, activation="tanh", padding="same")(conv1)
```

即降低滤波器的数目，从而减少模型参数

- 实验结果：每帧大概 30 多毫秒，即 FPS 达到 30 多，RTC 场景下可以达到实时

Andriod 客户端的实验

采用 TfliteGpuDelegateOptionsV2Default 和 TfliteGpuDelegateV2Create 来创建采用 GPU 进行推理的模型

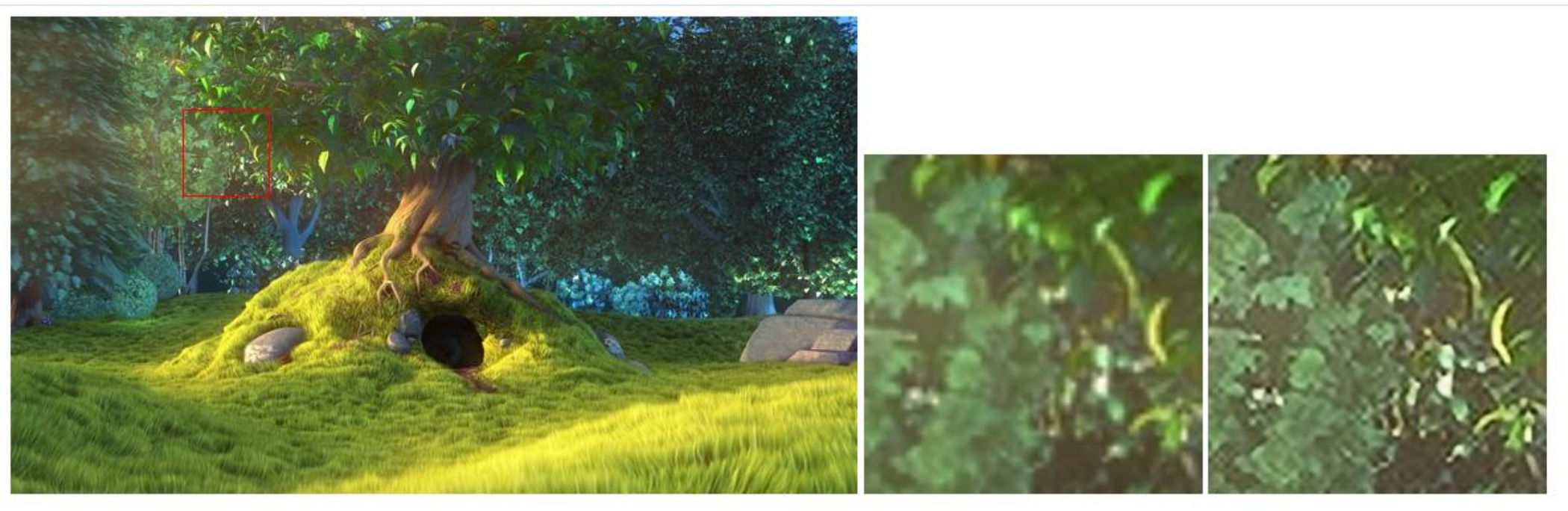
```
// Create the interpreter options
options_ = TfLiteInterpreterOptionsCreate();


// Choose CPU or GPU
if (use_gpu) {
  TfLiteGpuDelegateOptionsV2 options = TfLiteGpuDelegateOptionsV2Default();
  options.experimental_flags = TFLITE_GPU_EXPERIMENTAL_FLAGS_NONE;
  delegate_ = TfLiteGpuDelegateV2Create(&options);


  //delegate_ = TfLiteGpuDelegateV2Create(/*default options=*/nullptr);
  TfLiteInterpreterOptionsAddDelegate(options_, delegate_);
} else {
  TfLiteInterpreterOptionsSetNumThreads(options_, kThreadNum);
}
```

Andriod 客户端的实验

- 程序运行方式是通过 Java 的 JNI 调用 C++ 来完成。
- 可以用 tensorflowlite 超分的 demo 进行验证

```
final long startTime = SystemClock.uptimeMillis();
int[] superResRGB = doSuperResolution(lowResRGB);
final long processingTimeMs = SystemClock.uptimeMillis() - startTime;
if (superResRGB == null) {
  showToast( str: "Super resolution failed!");
  return;
}
```

ESPCN 的最终效果



总结与展望

总结：

- 介绍了适用于 WebRTC 传输的一种图像超分算法（ESPCN）
- 对模型的训练、量化、算力优化进行了探讨

展望：

- 相对于单个图像，视频超分算法，采用视频的时域特征能进一步提高超分效果
- 最新的模型可以利用 transformer 进行超分，但是算力要求高，适用性比较差
- 优化 transformer 模型推理效率是未来 RTC 中超分应用的重要研究方向

融云 RONGCLOUD

X-Live RTC·进阶实战高手课

感谢聆听！

融云官方公众号

融云【RTC 进阶实战高手课】六群

系列直播课讨论群