# iVC Serminar

# 第二小组编程实践技术文档

成员：李瑶（组长），卞逸凡，蒋易恒，陈乔希，丁欣恒，陈逸博

## 文档索引

# 1 说明

该文档为 USTC-iVCSeminar"编程实践"部分第二小组的代码技术文档。本次"编程实践"着力于实现基于端到端的图像压缩编码任务，并以 JPEG 编码方法的性能作为基准进行测试和比较。

本技术文档包括以下几部分内容：

- 编解码器结构
- 训练过程
- 测试命令和结果
- 复杂度分析

以上全部内容将在之后的文档里详细介绍和展示，但不保证按照以上顺序分模块介绍。

我们的编解码器的结构 Baseline 来自于前人论文中的工作，本次 Seminar 中我们对多篇论文中的端到端网络架构进行了复现、修改、整合和创新。所有参考的论文和代码如下：

参考文献：

1) Ballé J, Laparra V, Simoncelli E P. End-to-end optimized image compression[J]. arXiv preprint arXiv:1611.01704, 2016.[DOI]

2) Ballé J, Minnen D, Singh S, et al. Variational image compression with a scale hyperprior[J]. arXiv preprint arXiv:1802.01436, 2018.[DOI]

3) Johnston N, Eban E, Gordon A, et al. Computationally efficient neural image compression[J]. arXiv preprint arXiv:1912.08771, 2019.[DOI]

4) Minnen D, Ballé J, Toderici G D. Joint autoregressive and hierarchical priors for learned image compression[J]. Advances in neural information processing systems, 2018, 31.[DOI]

参考代码：

- Pytorch Reimplementation for End-to-end Optimized Image Compression[Link]
- Compress AI[Link]

有关本次 Seminar 所有的代码工作已经全部上传至 Github 仓库——USTC-iVCSeminar-Team2。如有需要预训练模型和其他任何有关问题，请联系Esakak。

# 2 编解码器结构

我们编解码器结构的 Baseline 参考 Ballé J 于 2016 年的发表论文 [1] 中展示的一种通用的端到端图像编解码架构。

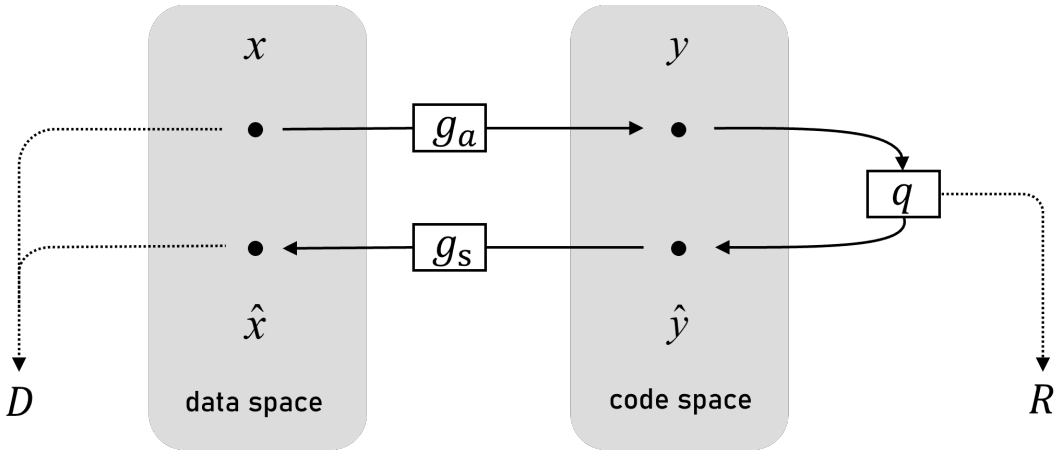



图 1: 端到端图像压缩通用架构

整个端到端图像压缩通用架构如图 1所示，其中 $x, y, \hat{x}, \hat{y}$ 分别表示原始图像、编码后特征、重构图像、量化后特征。方框 $g_a, g_s, q$ 分别表示编码器、解码器和量化器。大写字母 $R, D$ 分别表示码率 (Rate) 和失真 (Distortion)。

此处所示的网络结构只是一种通用架构，编码器、解码器和量化器等器件以及计算 $R, D$ 的方式都并非特指。基于以上的通用网络架构，本次 Seminar 中我们复现了论文 [1,2,4] 中提出的三种网络结构，分别记作：Factorized Model, Hyper Piror Model, Context Model。并在之后采用 [3] 中提出的一范数 GDN 代替前三篇论文中使用的二范数 GDN。最后我们分别针对 GDN 和数据预处理进行了尝试性的创新，分别采用余弦相似度来辅助 GDN 学习和提出一种可逆网络来转换图像至更易压缩的域中。

## 2.1 GDN

GDN(generalized divisive normalization) 是一种通用归一化模块，在神经网络卷积层后同时作为 normaliztion 和 activation 使用。其本身拥有可学习的参数，因此可以适应不同数据集、模型结构和参数设置。对于一张图片的第 $i$ 个通道，在传入 GDN 之前记作 $w_i(m,n)$，其中 $m, n$ 表示图片中像素的位置。那么经过 GDN 后得到的 $u_i(m,n)$ 由以下公式计算得到：

$$u_i(\,m,n\,) = \frac{w_i(\,m,n\,)}{(\,\beta_i \;+\; \Sigma_j\, \gamma_{ij}(w_j(\,m,n\,))^2\,)^{\frac{1}{2}}} \tag{1}$$

而对于 GDN 的逆向过程定义为 IGDN，并通过以下的计算得到：

$$\hat{w}_i(\,m,n\,) = \hat{u}_i(\,m,n\,) \cdot (\,\beta_i \;+\; \Sigma_j\, \gamma_{ij}(w_j(\,m,n\,))^2\,)^{\frac{1}{2}} \tag{2}$$

公式 (1),(2) 中的 $\gamma, \beta$ 均为 GDN 中可学习的参数，$\gamma$ 是一个 $C \times C$ 的参数矩阵，$\beta$ 是一个 $C$ 长度的参

数向量，其中 $C$ 代表输入图片的通道数目, 经过 GDN 不会改变 $C$。GDN 归一化计算的实现并不是像公式中逐个像素计算来进行的，而是通过一次 $2D$ 卷积操作直接得到所有通道的输出。

```
norm = F.conv2d(input=input_**2, weight=gamma, bias=beta)
norm = torch.sqrt(norm)

if self.inverse:
    return input_ * norm
else:
    return input_ / norm
```

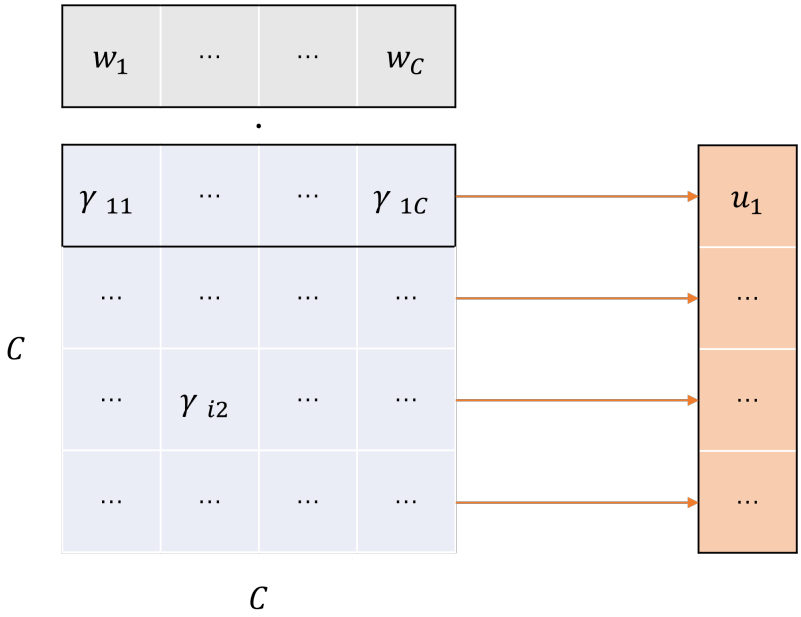



图 2: 卷积实现 GDN

其中 gamma 代表参数 $\gamma$, 是一个尺寸为 $(in\_channel = C, out\_channel = C, 1, 1)$ 的卷积核，表示接受通道数为 $C$ 的图片，在其每一个像素位置用 $C$ 个向量与图片在该位置的向量进行一次内积，从而输出通道数为 $C$ 的图片。具体的计算操作参照图 2。

需要注意的是在之后的实际代码实现中，为了降低运算消耗，我们使用了论文 [3] 中的一范数 GDN，两者 RD 曲线基本重合。一范数 GDN 的形式化描述为:

$$u_i(m,n) = \frac{w_i(m,n)}{\beta_i + \Sigma_j \gamma_{ij} \|w_j(m,n)\|} \tag{3}$$

## 2.2 Factorized Model

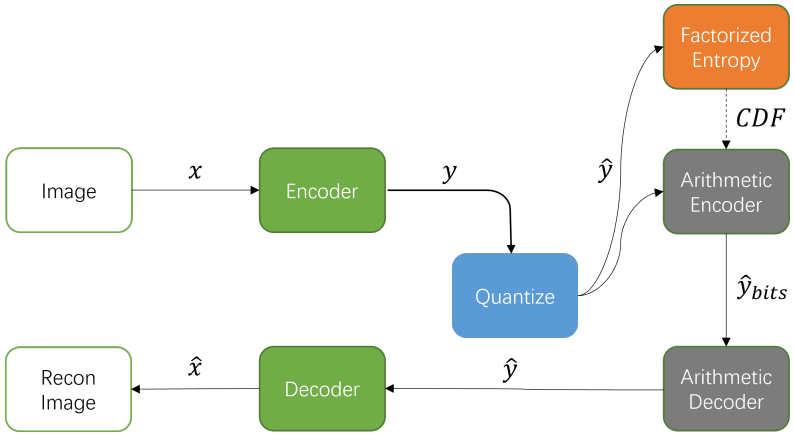



图 3: Factorized Model

Factorized Model 是论文 [1] 中提出的网络模型，其网络结构较为简单，由编码器、解码器、量化器和熵模型模块组成。如图 3，原始图像 $x$ 经过过编码器被编码到 $y$ 空间，经过量化后的 $\hat{y}$ 在 Factorized 的熵模型提供的 CDF 下通过算术编码被压缩成 $\hat{y}_{bits}$，解码后的 $\hat{y}$ 经过解码器得到重构图像 $\hat{x}$。

$Quantize$ 操作是指对 $y$ 的量化操作，在本次 Seminar 中，我们所有的模型都使用了论文 [1] 中提出的添加均匀分布噪声的方法来使得量化在训练过程中可回传梯度。量化操作可以形式化如下：

$$Q(y) = \begin{cases} round(y) & if\ train, \\ y + U(0,1) & else. \end{cases} \tag{4}$$

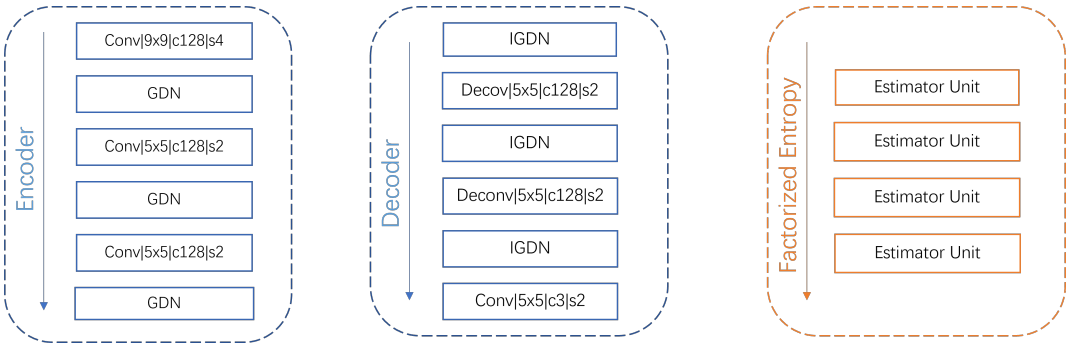



图 4: Factorized Model 中的模块

编码器和解码器均由 3 个卷积层和 3 个 GDN 交替级联形成，$[Conv|9X9|c128|s4]$ 表示卷积核尺寸为 9，输出通道数为 128，步长为 4 的 $2D$ 卷积。Factorized Entropy 由 4 个 bitEsitimator Units 级联而成，每一个 Unit 是一次对输入的累计分布函数的参数化拟合，Unit 含有可学习参数 $H, b, a$。其运算过程如下：

$$f_k(x) = g_k(\boldsymbol{H}^{(k)}x + \boldsymbol{b}^{(k)}) \qquad 1 \le k < K \tag{5}$$

$$f_K(x) = sigmoid(\boldsymbol{H}^{(K)}x + \boldsymbol{b}^{(K)}) \tag{6}$$

其中 $H^{(k)}$ 是矩阵，$b^{(k)}$ 是向量，$g_k$ 是定义如下的非线性计算：

$$g_k(x) = x + a^{(k)} \odot tanh(x)$$

## 2.3 Hyper Prior Model

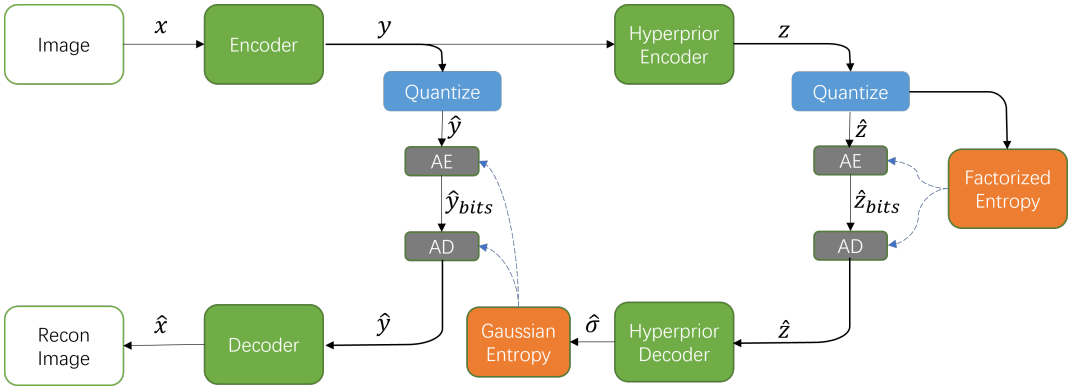



图 5: Hyper Prior Model

Hyper Prior Model 是论文 [2] 中提出的网络模型，其网络结构在 Factorized 的基础上添加了 Hyper 编码器和 Hyper 解码器以及新的熵模型 Gaussian Entropy Model。Hyper Prior Model 对原先的两个编码器进行了精简，新的 Hyper 编解码器使用 ReLU 作为新的激活函数，对 $y$ 的方差进行编码，作为一种 side information 进行传输。

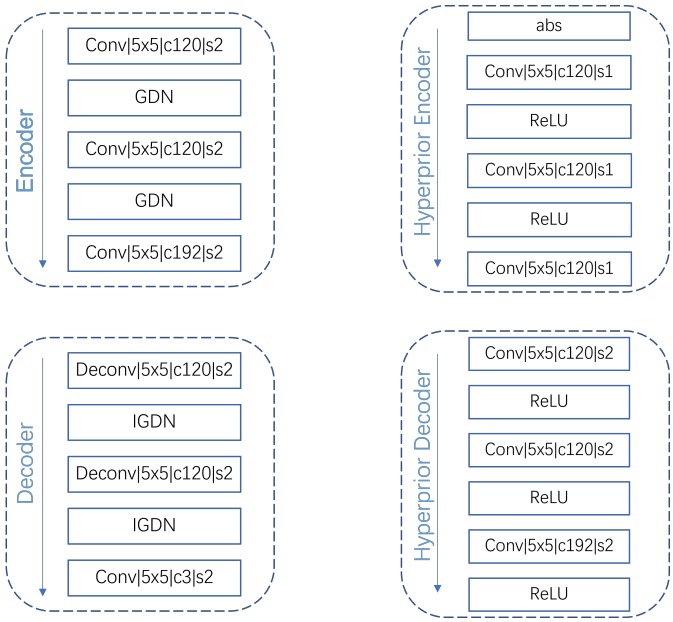



图 6: Hyper Prior Model 中的模块

Hyper Prior Model 中对 $y$ 和 $z$ 的编码采用不同的熵模型，其中对 $z$ 的编码采用 Factorized Entropy Model 来对 $z$ 的累计分布进行参数化拟合。而对 $y$ 的编码则是认为其服从正态分布 $N(0, \hat{\sigma})$，Hyper 层则用来从 $y$ 中提取每个位置对应的 $\hat{\sigma}$。

## 2.4 Context Model

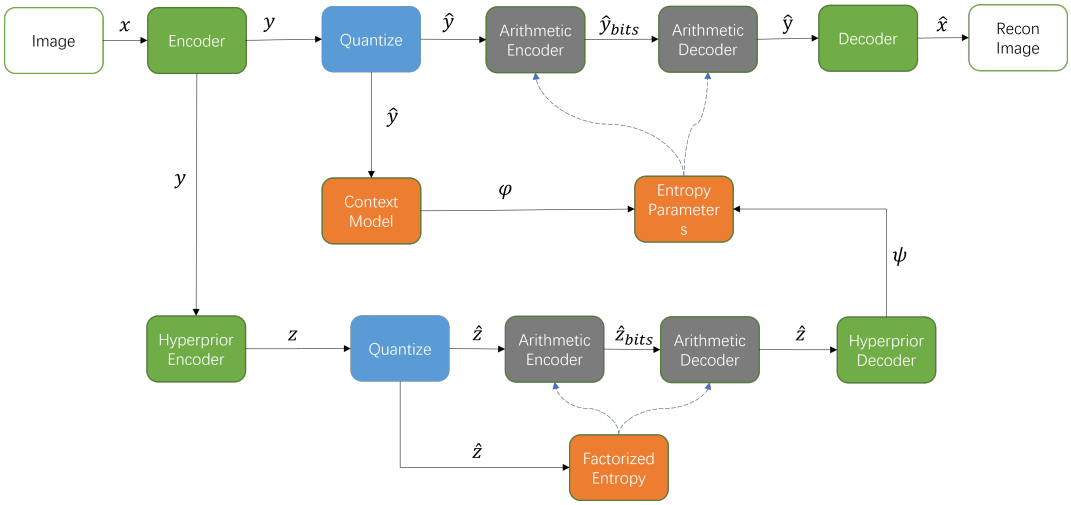



图 7: Context Model

Context Model 是论文 [4] 中提出的网络模型，其保留了 Hyper Prior Model 中的自编码器结构与超先验结构，在此基础上添加了基于自回归的上下文模型；Hyperprior 论文中只预测熵模型的 $\hat{\sigma}$ 参数，而该部分的网络结构增加了对于均值 $\mu$ 的预测，建立了 GMM 模型。网络结构中的自编码器结构学习量化后的隐变量 $\hat{y}$，超先验网络结合上下文模型学习基于隐变量 $y$ 的概率模型，这两部分生成的数据经熵参数网络后，得到预测的高斯模型参数。


图 8: Context Model 中的模块

图 8展示具体的网络层。编解码器与超先验编解码器的层结构与 hyperprior 论文相同，仅在通道数、卷

积核尺寸以及步长作调整。上下文模块通过一个 Masked 卷积层实现，对卷积核进行 Mask 操作，将未解码点的数值设为 0，使得预测当前解码点时的参数仅来源于之前已解码的像素点；在解码第一个像素点时，假设已解码点数值均为 0。

熵参数模块预测高斯分布的均值与标准差，传入参数为上下文模型的输出参数 $\phi$ 与超先验解码器的输出参数 $\psi$，将 $\phi$ 与 $\psi$ 拼接（concate）后传入熵参数模块，输出高斯模型的均值与标准差，因此将最后一层的通道数量设置为原通道数量的两倍，通过直接拆分通道的方式得到均值和标准差。

# 3 训练过程

我们的训练均部署在Bitahub提供的计算服务器上，采用多 GPU 并行训练的方式进行，硬件条件为 8 块 GTX 1080 Ti 显卡。我们使用 AdamW 作为模型训练的优化器，初始 learning rate 设为 2e-4，learning rate decay 设为 0.999，Batch size 设为 64。有损图像压缩编码需要注意 trade off between Rate and Distortion，所以我们使用了经过正则化的损失函数：

$$Loss = R + \lambda * D * 255^2$$

共训练并测算了四个码率点 {0.0067,0.0130,0.0250,0.0483} 我们使用 tensorboard 来对训练的过程进行可视化。

## 3.1 Factorized Model 的训练

作为我们的 baseline 模型，我们花费了一周的时间进行代码复现和训练工作。训练过程中，对于较低的两个码率点，我们设置通道数 $C = 128$，对于较高的两个码率点，则设置为 $C = 192$。$\lambda = 0.0250$ 时，训练过程的曲线如下：

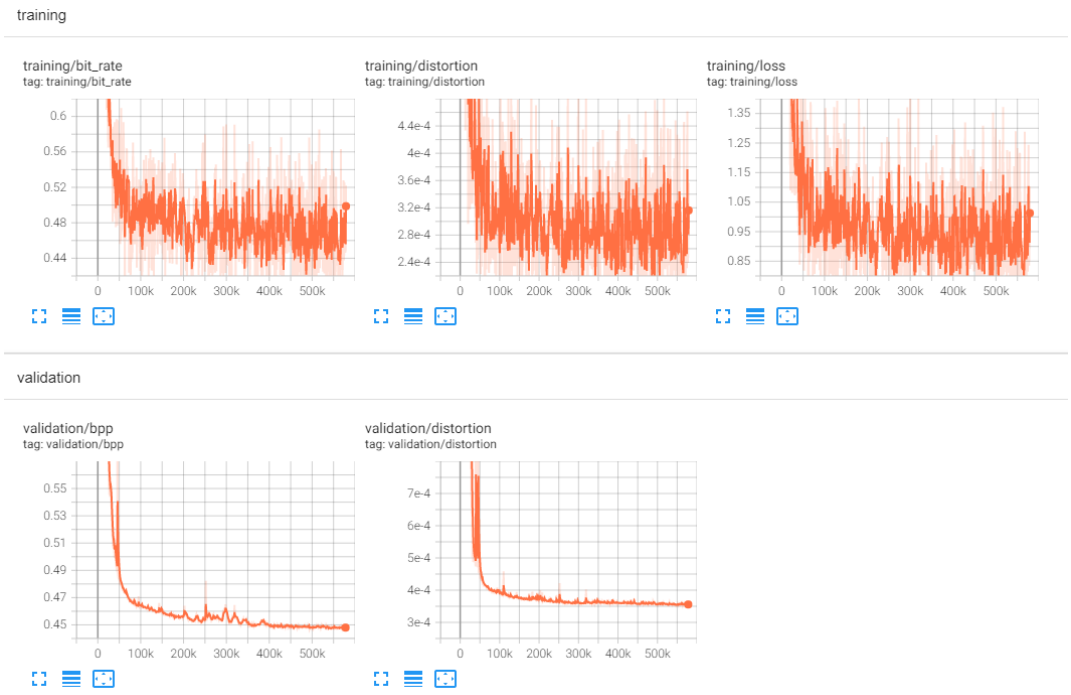



图 9: Factorized Model 训练曲线

## 3.2 Hyper Prior Model 的训练

训练过程中，对于较低的两个码率点，我们设置通道数 $N = 128, M = 192$，对于较高的两个码率点，则设置为 $N = 192, M = 320$。$\lambda = 0.0250$ 时，训练过程的曲线如下：

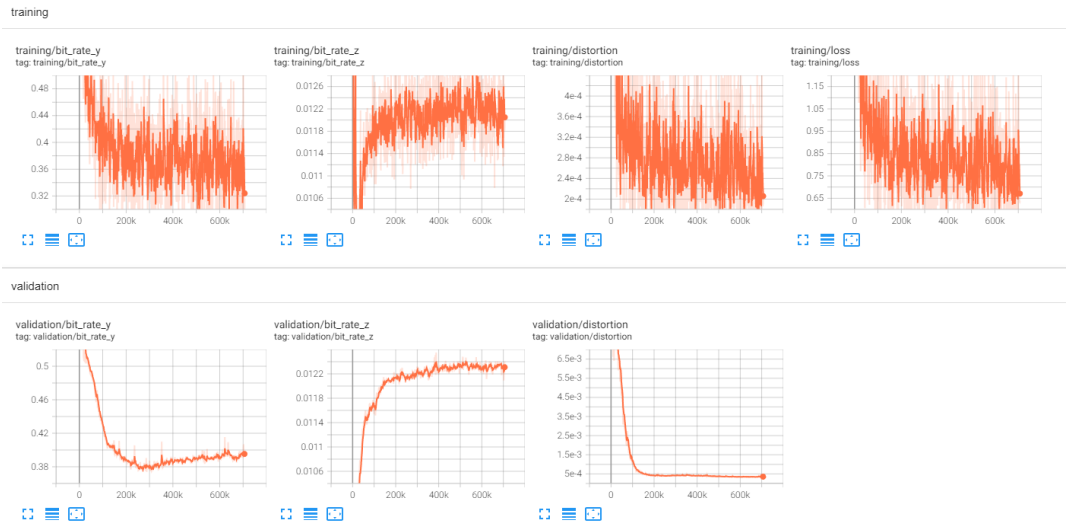



图 10: Hyper Prior Model 训练曲线

## 3.3 Context Model 的训练

检查 Vimeo 训练集中的图像发现，许多目录中连续的 7 张图像均是由单张图像进行增广得到的，各图像之间差异不大，因此对于每个目录，我们仅随机选取一张作为训练样本。对于每个样本，将其随机裁剪成 $256 \times 256$ 输入到网络。Context Model 的训练需要首先在 0.0483 这个码率点上进行 100 个 epoch 的预训练，随后在其基础上分别在 0.0483, 0.0250, 0.130, 0.0067 这 4 个码率点上进行训练。

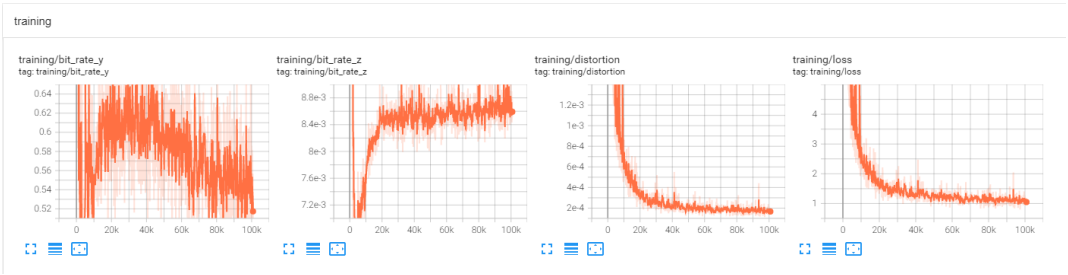



图 11: Context Model 的预训练曲线

$\lambda = 0.0250$ 时，训练过程的曲线如下：

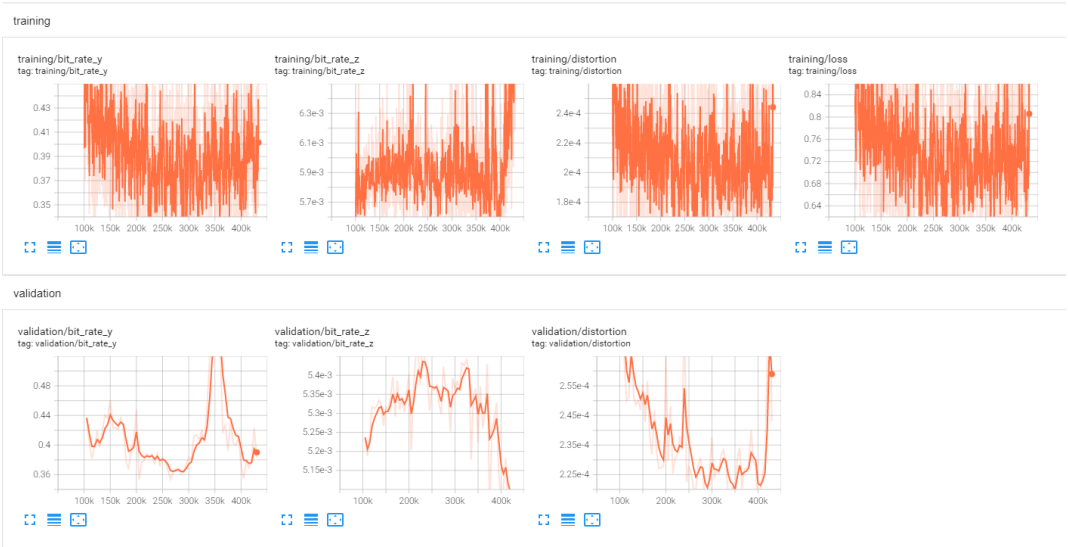



图 12: Context Model 的训练曲线

# 4 测试过程

## 4.1 测试命令

我们针对论文复现的代码可以通过以下链接获得，同样你也可以通过第1章节中提到的 Github Orgnization Link 整体获得。

- Factorized Model
- Hyper Prior Model By Ly
- Hyper Prior Model By Byf
- Context Model

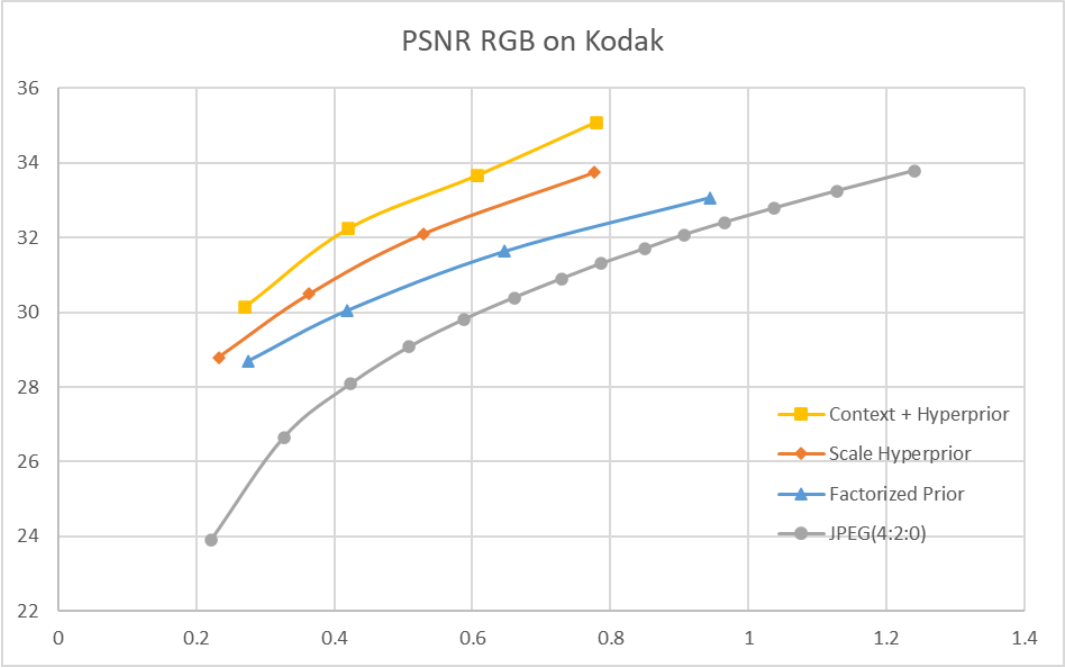
这 3 篇论文对应的代码和模型均可以使用如下命令进行测试，测试代码将输出 24 张 Kodak 测试图像对应的原始图与重建图的 PSNR, MS-SSIM, 编码的 bpp 以及编解码耗时等信息。

```
python test.py --test_dir <path to kodak set> --checkpoint_path <path to model file> --reco_dir <path to
    store reconstructed images>
```

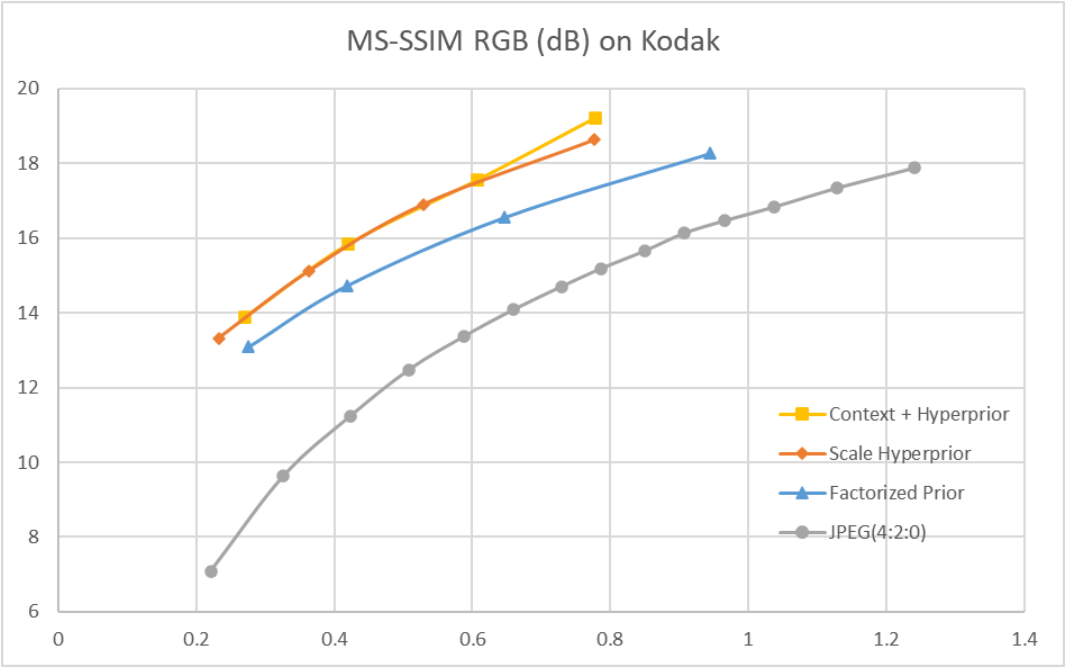
## 4.2 测试结果

### 4.2.1 RD-Curve

两种 Distortion 衡量下的 RD 曲线：




(a) PSNR 下的曲线




(b) MS-SSIM 下的曲线

图 13: RD 曲线

其中，JPEG 的编码是使用 PIL 库中的 *save* 方法进行的：

```
img.save('kodim01.jpg', quality=75, subsampling=2)
```

*quality* 分别取 5, 10, 15, ..., 100 这 20 个点，从而得到 JPEG 的 RD 曲线。比较测得的 RD-curve 和论文中展示的 RD-curve，发现在相同 bpp 下这两个模型的 PSNR 均和论文的结果有 1dB 左右的差距。

### 4.2.2 BD-rate

| Method | BD-rate (PSNR) | BD-rate (MS-SSIM) |
|---|---|---|
| Factorized Model | -31.78% | -44.73% |
| Hyper Prior Model | -46.98% | -54.41% |
| Context Model | -56.78% | -55.00% |

表 1: 不同指标上的 BD-rate

Factorized Model 和 Hyper Prior Model 相应的论文中均没有给出 BD-rate 的结果，Context Model 相应的论文中给出的 BD-rate (PSNR) 结果为 59.8%，训练的模型与其有 3% 的差距。这些性能上的差距可能是由训练集的差异以及训练策略的不同导致的。

### 4.2.3 主观图像质量分析

(a) Context Model (b) Hyper Prior Model (c) Factorized Model (d) JPEG

图 14: 主观质量比较。从左至右的 bpp 分别为：0.240, 0.210, 0.248, 0.254 。

在相近的 bpp 下，Context Model 有着最佳的视觉质量。注意到 Factorized Model 和 Hyper Prior Model 在栅栏密缝处以及灯塔的窗户位置有着较为明显的 artifacts，观察游泳圈及其周围的黄色丝带，也可比较出三者中 Context Model 的重建图像质量最佳。JPEG 在这一码率点下则有着非常严重的块状效应和失真。

# 5 复杂度分析

测试条件：

- Factorized Model, Hyper Prior Model: i5 8250U + MX150
- Context Model: i5 12600KF + RTX3070ti 8G

## 5.1 编码器时间

| lambda | Factorized Model | Hyper Prior Model | Context Model |
|---|---|---|---|
| 0.0067 | 0.33 | 0.98 | 0.25 |
| 0.0130 | 0.41 | 1.31 | 0.38 |
| 0.0250 | 0.52 | 1.90 | 0.29 |
| 0.0483 | 0.64 | 2.63 | 0.35 |

表 2: 编码器时间 (s)

## 5.2 解码器时间

| lambda | Factorized Model | Hyper Prior Model | Context Model |
|---|---|---|---|
| 0.0067 | 0.46 | 0.85 | 103.98 |
| 0.0130 | 0.53 | 1.19 | 168.63 |
| 0.0250 | 0.64 | 1.78 | 195.52 |
| 0.0483 | 0.76 | 2.51 | 252.63 |

表 3: 解码器时间 (s)

测试条件同上。可以看到，对于 Context Model 模型，即使使用了更强的计算资源，其解码器时间也远超于其余模型，这是由于其解码时使用了串行处理的自回归模块，从而不能进行并行计算，极大地提高了解码时间。

## 5.3 模型参数量与 MACs

| Method | Params | MACs |
|---|---|---|
| Factorized Model | 7.08e10% | 4.43e6% |
| Hyper Prior Model | 7.40e10% | 4.97e6% |
| Context Model | 1.62e11% | 1.21e7% |

表 4: 模型参数量与 MACs

# 6 模型拓展

我们不局限于复现他人的论文工作，在经过思考和讨论后，我们决定从以下两个方面开展我们的模型拓展工作。尽管效果欠佳，但是却是一次大胆创新的尝试。

## 6.1 利用可逆网络对模型做前置处理

### 6.1.1 动机

一般的端到端图像压缩网络都是在原始图像域上进行压缩，并没有做额外的前置处理。我们考虑使用可逆网络对原始图像进行无信息损失的前置处理，转换后的域称为转换域，然后在该转换域上训练端到端的图像压缩网络，希望网络能在转换域上有更好的压缩效果。

### 6.1.2 网络结构

网络的主体结构如下图所示：

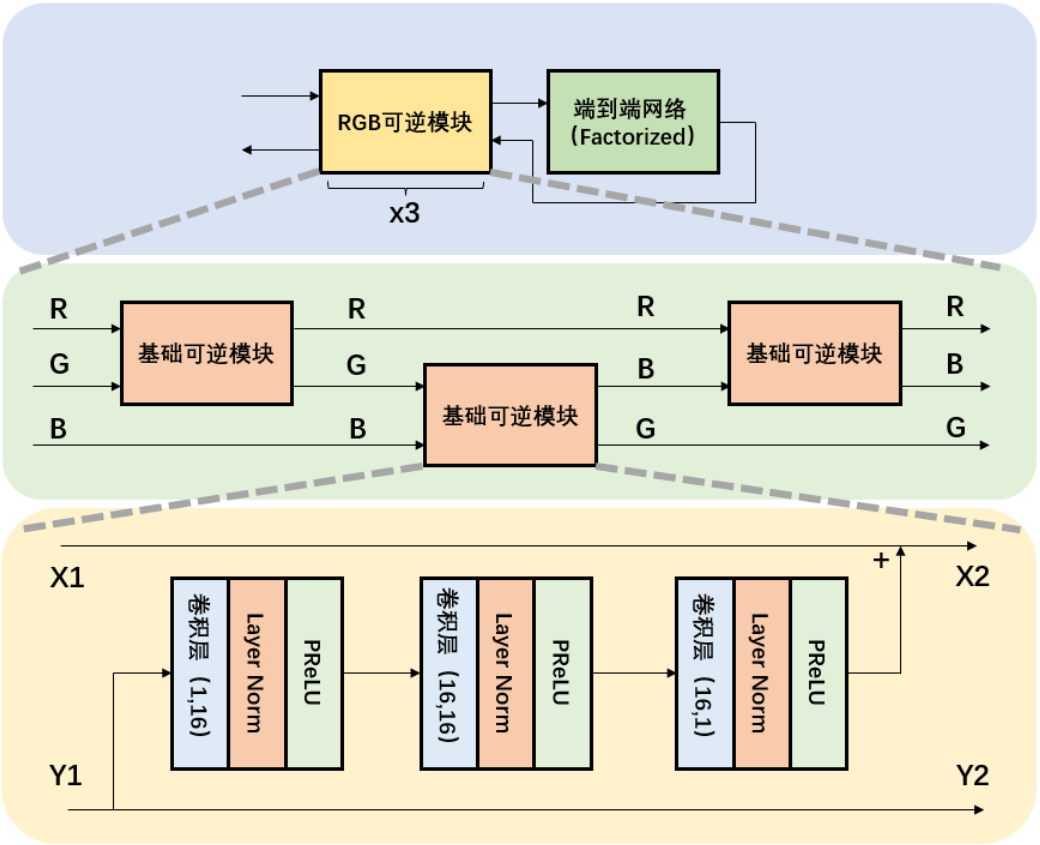



图 15: 可逆网络总体结构

网络由一个可逆网络和一个端到端网络构成，其中端到端网络选用的是 FactorizedPriorModel。可逆网络由 3 个 RGB 可逆模块组成，每个 RGB 可逆模块由 3 个基本可逆模块组成，每个基本可逆模块的非线性部

分由 3 组卷积层-Layer Norm 层-PReLU 层组成，每个卷积层的卷积核大小为 3x3，stride 为 1，通道数如图15所示。

### 6.1.3 训练结果

**（1）训练曲线**:

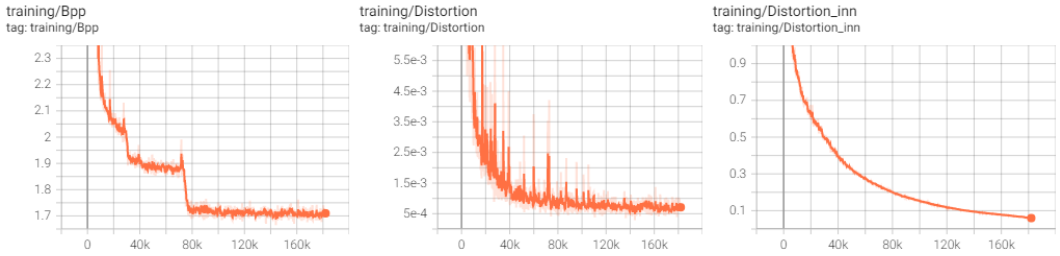



图 16: 可逆网络训练曲线

上图是从零开始训练的可逆网络加端到端图像压缩网络的训练曲线，可以看出 bpp 一直处于一个较高的水平，代表网络的压缩性能还有待提高。但从图像的失真度曲线可以得到比较有趣的结果，在原始图像域上的失真度达到了 6e-4 级别，但转换域上的失真度高达 0.05，理想的情况是转换域对比图像域有更高的失真容忍度。为了探究网络的真实情况，我们将预训练的可逆网络进行冻结，固定转换域，单独优化端到端的图像压缩网络，训练曲线如下：

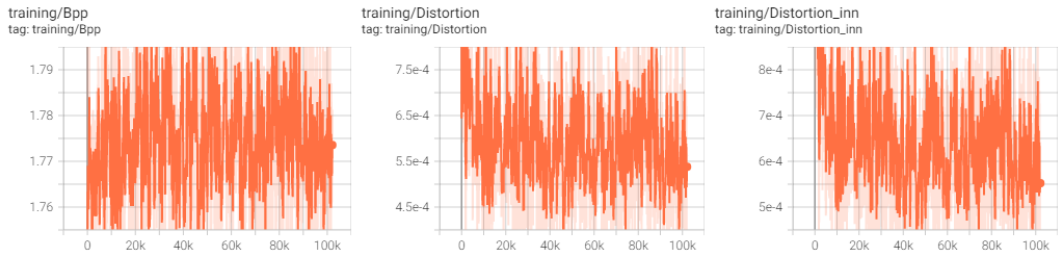



图 17: 固定前置网络后的训练曲线

可以看到，BPP 依然居高不下，原始图像域的失真度没有下降，但转换域上的失真度有了显著降低，到了 6e-4 的水平。这说明减小转换域的失真度并不能同时减小原始图像的失真，并且由于 bpp 不下降，转换域的图像实际上是更难压缩了。

**（2）图像可视化**：图像各通道的可视化结果如下：

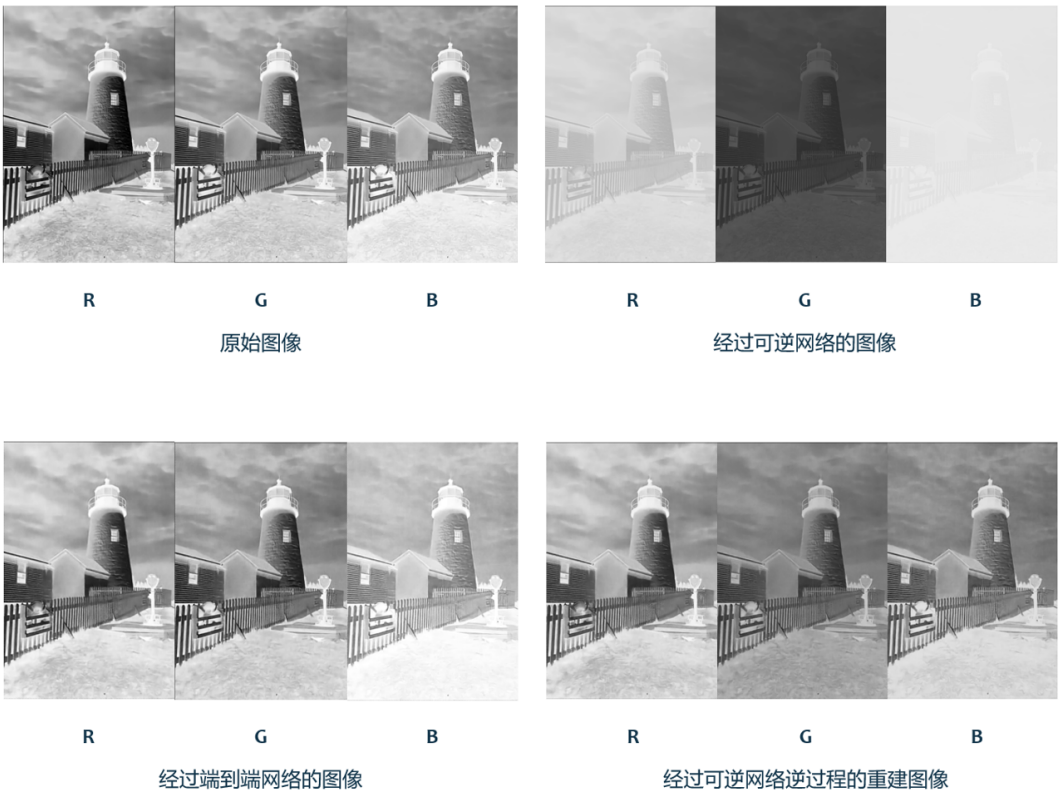



图 18: 网络中特征的可视化

## 6.2 参考 Attention 的思想改进 GDN

### 6.2.1 动机

GDN 本质上是对于某种生物现象的数学建模，原始的 GDN 实际上是一种较为简单的建模方式。通过对比发现，GDN 与 Attention 的思想非常类似，但 Attention 的建模更加复杂，弹性更强，因此我们考虑引入 Attention 的思想对 GDN 进行改进。

### 6.2.2 改进方法

如果直接使用常见的 Attention 建模方式会极大提高内存占用和计算时间，因此我们仅在原始 GDN 的基础上添加一个通道维度的余弦相似度，用该相似度来简单表征 Attention 中的相似度计算。

### 6.2.3 实验结果

我们尝试了两种添加余弦相似度的方法，一种是对一个 Batch 所有图像的通道计算一个总的余弦相似度矩阵，另一种是对每一张图像都单独维护一个余弦相似度矩阵，实验结果如下：

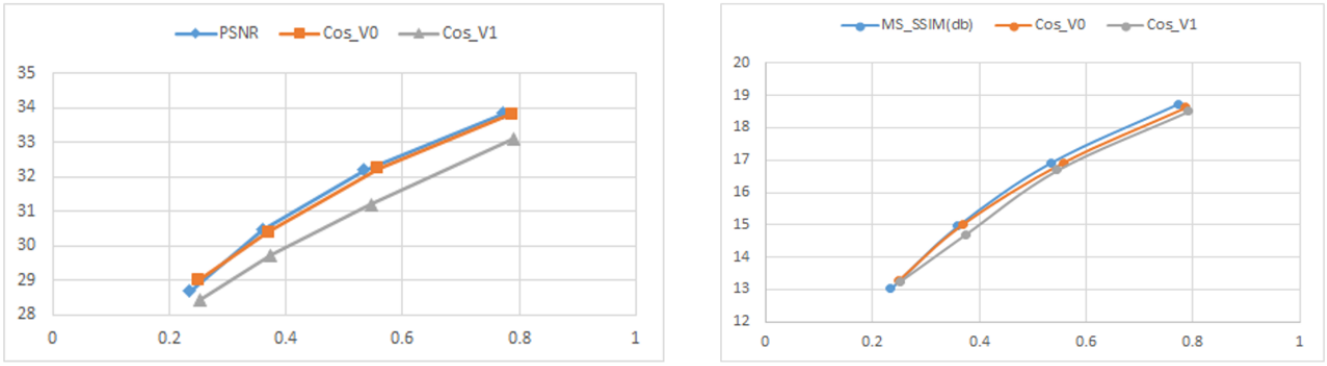



图 19: 加入余弦相似度后的 RD-Curve

可以看到的是，添加余弦相似度并不能提高网络的性能，反倒是有所下降。其中，对一个 Batch 所有图像维护一个余弦相似度矩阵的时候对于性能的几乎没有影响，但如果对每一张图片都维护一个余弦相似度矩阵的话网络的性能会有显著下降。分析认为，GDN 作为一种正则化方法，其性能主要与网络通道本身关系较大，而跟单张图像关系不大。我们对一整个 Batch 仅维护一个相似度矩阵的时候，因为一个 Batch 的图像具有多样性，该相似度矩阵是一个较为平均的表现，因而比较偏向于体现通道本身的性质，所以结果没有太大影响；但当我们对单张图像维护一个相似度矩阵的时候，图像本身的影响会非常大，并不能充分体现通道的性质，所以性能下降明显。