Lossy image compression Equations

Flora Quilichini

January 2021

1 Loss function

$$Loss = \underbrace{-log_2 Q([f(\mathbf{x})])}_{\text{number of bits}} + \underbrace{\beta}_{\text{tradeoff}} . \underbrace{d(\mathbf{x}, g([f(\mathbf{x})])}_{\text{distortion}} \tag{1}$$

With :
$f : \mathbb{R}^N \rightarrow \mathbb{R}^M$: encoder function
$g : \mathbb{R}^M \rightarrow \mathbb{R}^N$: decoder function
$Q : \mathbb{Z}^M \rightarrow [0, 1]$: discrete probabilistic distribution of symbols
$[.] : \mathbb{R}^M \rightarrow \mathbb{R}^M$: quantization (nearest integer rounding)
N : input space dimension
M : latent space dimension

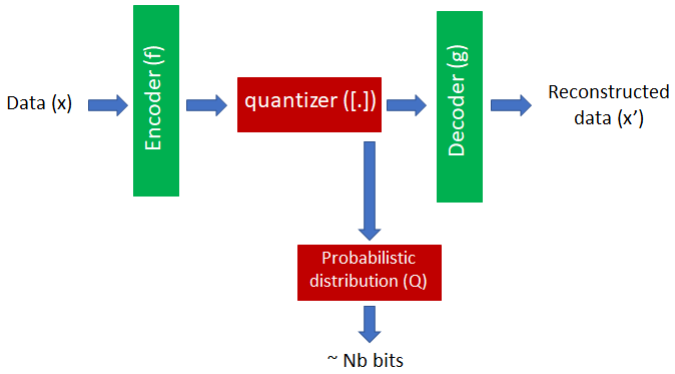

Figure 1: Simplified overview

Problem:
Q and $[.]$ are not differentiable, so we can not compute back-propagation gradient descent.

2 Differentiating loss function

2.1 Second term

We assume that d is differentiable.
$W = (W_f, W_g)$ where W_f and W_g are respectively the encoder and decoder weights

$$\frac{\partial d(\mathbf{x}, g([f(\mathbf{x})]))}{\partial W} = \frac{\partial d(\mathbf{x}, g([f(\mathbf{x})]))}{\partial g([f(\mathbf{x})])} * \frac{\partial g([f(\mathbf{x})])}{\partial W} \tag{2}$$

With :

$$\frac{\partial g([f(\mathbf{x})])}{\partial W} = (\frac{\partial g([f(\mathbf{x})])}{\partial W_f}, \frac{\partial g([f(\mathbf{x})])}{\partial W_g}) \tag{3}$$

$$\frac{\partial g([f(\mathbf{x})])}{\partial W_f} = \frac{\partial g([f(\mathbf{x})])}{\partial [f(\mathbf{x})]} * \frac{\partial [f(\mathbf{x})]}{\partial f(\mathbf{x})} * \frac{\partial f(\mathbf{x})}{\partial W_f} \tag{4}$$

d, g and f are differentiable, so in order to be able to compute gradient descent, we need to make $\frac{\partial [f(\mathbf{x})]}{\partial f(\mathbf{x})}$ differentiable. This is done by approximating the derivative of the quantization function (only for back-propagation).
The approximation used is : $\frac{\partial [f(\mathbf{x})]}{\partial f(\mathbf{x})} \sim 1$, where the quantization function is approximated by the linear function. This approximation is asymptotically correct when the quantization step is very small (tends to 0).
NB: Another approximation could be the convolution of quantization with a Gaussian.

2.2 First term

$$\frac{\partial log_2(Q([f(\mathbf{x})]))}{\partial W_f} = \frac{\partial log_2(Q([f(\mathbf{x})]))}{\partial [f(\mathbf{x})]} * \frac{\partial [f(\mathbf{x})]}{\partial W_f} \tag{5}$$

The derivative of $\frac{\partial [f(\mathbf{x})])]}{\partial W_f}$ is obtained in the same way as above.
Concerning the derivative of $\frac{\partial log_2(Q(\mathbf{z}))}{\partial \mathbf{z}}$, the "trick" is to compute an upper bound of the term for the derivation.

$$Q(\mathbf{z}) = \int_{[-0.5,0.5]^M} q(\mathbf{z}+\mathbf{u})\, \mathrm{d}\mathbf{u} \tag{6}$$

$$-log_2(Q(\mathbf{z})) \leq \int_{[-0.5,0.5]^M} -log_2(q(\mathbf{z}+\mathbf{u}))\, \mathrm{d}\mathbf{u} \tag{7}$$

Where q is a probability density, that we chose to be differentiable in $\mathbf{z}$.
In practice, q is a Gaussian scale mixture such as

$$log_2(q(\mathbf{z}+\mathbf{u})) = \sum_{i,j,k} log_2 \sum_s \pi_{ks} N(z_{k,i,j} + u_{k,i,j}; 0, \sigma_{ks}^2)$$

where i and j iterate over spatial positions, and k iterates over channels.
NB: For first step, we will take only one scale s.

3 Network architecture

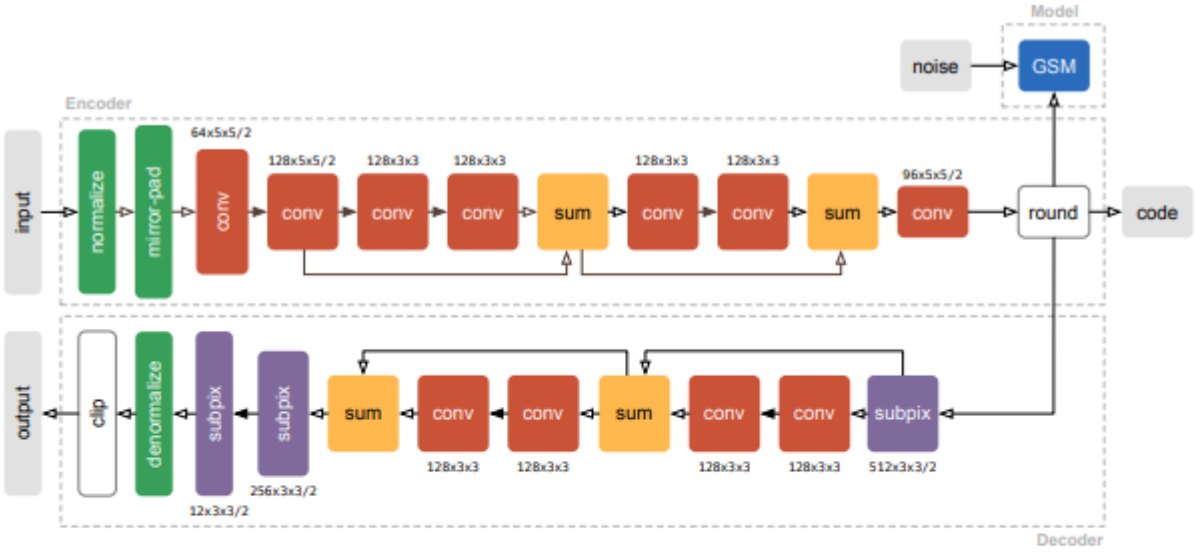

Figure 2: Illustration of the compressive autoencoder architecture used in this paper. Inspired by the work of Shi et al. (2016), most convolutions are performed in a downsampled space to speed up computation, and upsampling is performed using sub-pixel convolutions (convolutions followed by reshaping/reshuffling of the coefficients). To reduce clutter, only two residual blocks of the encoder and the decoder are shown. Convolutions followed by leaky rectifications are indicated by solid arrows, while transparent arrows indicate absence of additional nonlinearities. As a model for the distributions of quantized coefficients we use Gaussian scale mixtures. The notation $C \times K \times K$ refers to $K \times K$ convolutions with C filters. The number following the slash indicates stride in the case of convolutions, and upsampling factors in the case of sub-pixel convolutions.

4 Training and Test

- Optimizer: Adam
- Batch size : 32
- Training Images: 434 high quality images from flickr.com, downsampled below 1536*1536 pixels, and cropped to 128*128.
- Test images: Kodak PhotoCD dataset of 24 images of 768*512 px
- Learning rate: Initially set to 10^{-4} and goes to 10^{-5}
- d (metric for distortion) : mean squared error

The training is performed in an incremental fashion, with the use of a binary mask $\mathbf{m}$

$$-log_2(q([f(\mathbf{x}) \otimes \mathbf{m}] + \mathbf{u})) + \beta * d(\mathbf{x}, g([f(\mathbf{x}) \otimes \mathbf{m}])) \tag{8}$$

Initially, all but 2 coefficients of the mask are set to 0. The network is trained until performance improvements reach below a threshold, and then another coefficient is set to 1. Until the mask is not complete, the learning rate ranges from 10^{-4} to 10^{-5}. Once the mask is complete, and the training done for a fixed value of β, then a fine-tuned scale parameter $\boldsymbol{\lambda}$ is introduced in order to be able to control bit rate without storing lots of different neural network models.

5 Controlling bit rate

To have finer control over bit rate, and to avoid the storage of a big number of neural network models, the authors introduced an additional parameter $\lambda \in \mathbb{R}^M$. This parameter is used to control the length of the quantization step, and so the sharpness (quality) of the quantization. Both λ and β enable bit rate variations. The new loss becomes

$$-log_2(q([f(\mathbf{x}) \otimes \boldsymbol{\lambda}] + \mathbf{u})) + \beta * d(\mathbf{x}, g([f(\mathbf{x}) \otimes \boldsymbol{\lambda}]/\boldsymbol{\lambda})) \tag{9}$$

NB : For first step, we will take $\boldsymbol{\lambda} = [1..1]$ NB : The learning rate changes from previously. It is initially set to 10^{-3}, and then is continuously decreased by a factor $\tau^k/(\tau + t)^k$ where t is the number of updates performed, $k = 0.8$ and $\tau = 1000$

References

[1] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders, 2017.