**Backgrounds**

The Multi-U-Branch Residual GAN (MUBRG) algorithm is a novel AI approach optimized for color image denoising, though other image transformation tasks may also be addressable. Proposed in 2023, it is currently still under development and subjected to updates. The newest test results on real noisy photos are presented at the end of this doc.

This algorithm works with square slide windows of images. A noisy image window can be expressed as Eq.1:

$$x = y + \eta \tag{1}$$

Where $y$ is the ground-truth of the image, $\eta$ is the noise, and $x$ is the resulted noisy image.

While training, the ground-truths are represented by $c_0 * w_0 * w_0$ clean image windows obtained from real photos taken at low ISO. While several color spaces are tested, this project adopts the RGB color space thanks to it outperforming other choices, thus $c_0 = 3$. The distribution of real camera noises is complicated and cannot be represented with any simple expression. To generate $x$ as inputs, the noise window $\eta$ can be generated by two approaches. The first approach is to randomly take $c_0 * w_0 * w_0$ windows from a pure camera noise image and augment it according to the value of each element in $y$ (i.e. the exposure of each pixel). The second approach is to randomly sample from the $c_0 * w_0 * w_0$ noise distribution in the frequency domain, then transform back to the content domain and augment it according to $y$. While the second approach gives a larger variety of noise patterns to better generalize the learning, the first approach is faster yet giving acceptable results.

The purpose of the denoising algorithm is to optimize Eq.2:

$$\hat{G} \rightarrow argmin_G\big(L_G(G(x), y)\big) \tag{2}$$

Where $G$ is the function represented by the generator network model, $\hat{G}$ is the function of this model with expected parameters after training, and $L_G(G(x), y) \in \mathbb{R}_{\geq 0}$ is a loss function that evaluates the similarity (higher similarity results in smaller value) of the model outputs $G(x) \in \mathbb{R}^3$ and the ground-truth $y \in \mathbb{R}^3_{[0,1]}$.

**Model structures**

The following neural network diagrams follow the color-coding shown in Figure 1, except else specified.

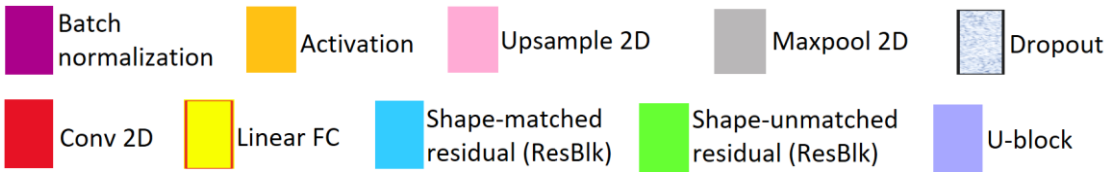



*Figure 1 Legend for Model Diagrams*

There are two versions of residual blocks (ResBlks) employed. When the output number of filters is the same as the input number of channels, the shape-matched ResBlks are used. Otherwise, the shape-unmatched ResBlks are used. They are as shown in Figures 2(a) and 2(b) respectively.

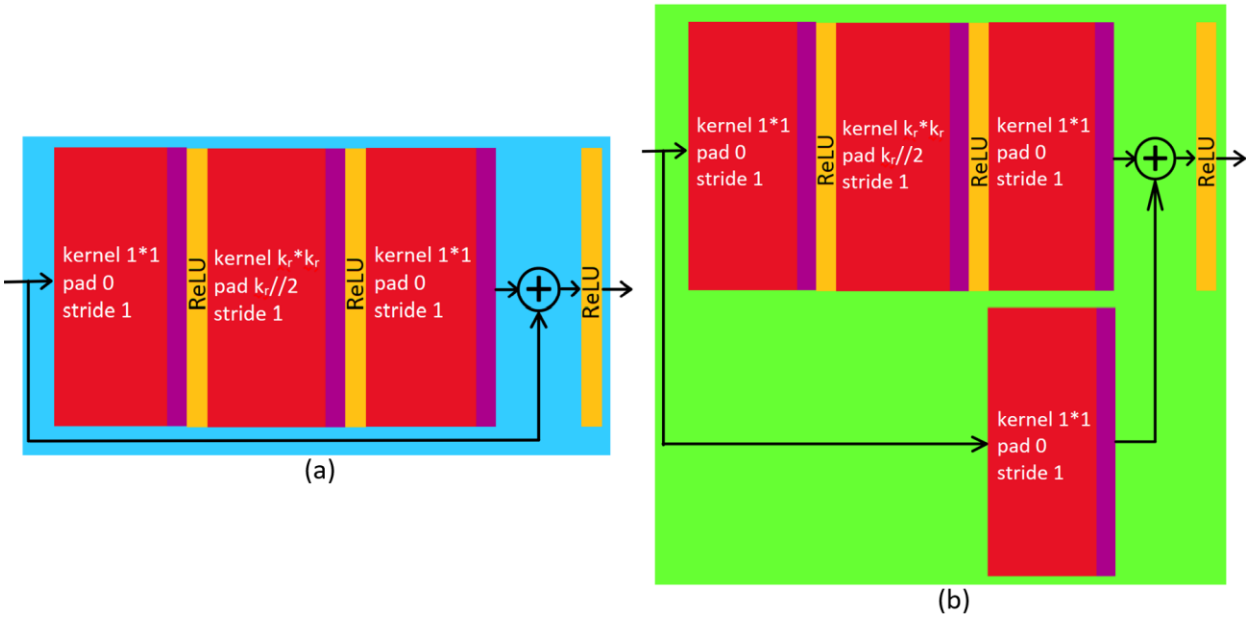



*Figure 2 Residual Blocks (ResBlks)*

The U-block shown in Figure 3 are constructed with variable numbers of residual blocks explained in Figures 2. The zigzag on the shape-matched ResBlks indicate a series of various such blocks. Note that the output of the U-block has twice the number of channels as the input.

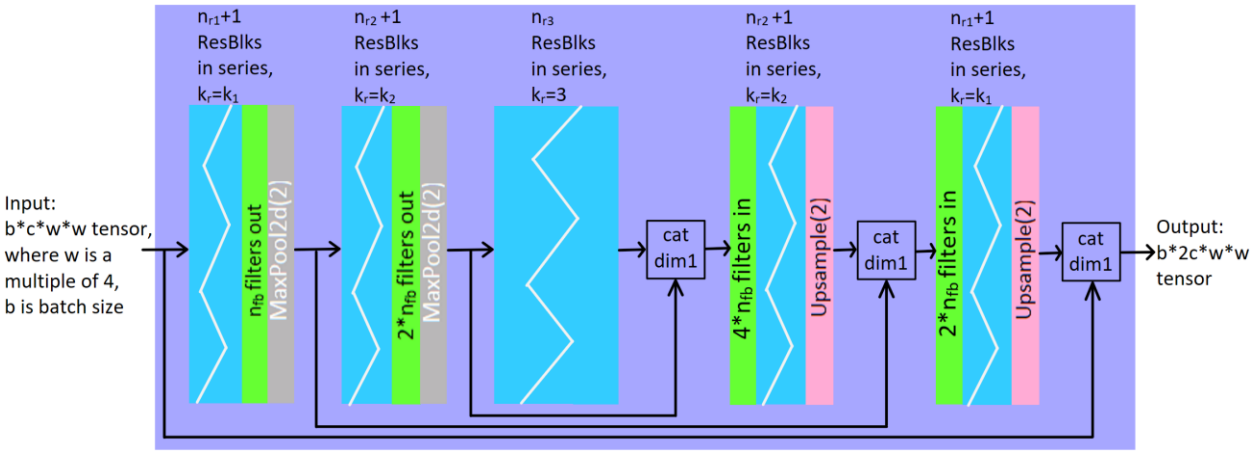



*Figure 3 U-block*

Then comes the full generator in Figure 4. Clearly, the input image is transformed into three types of feature maps through three branches before the fusing in a late stage.

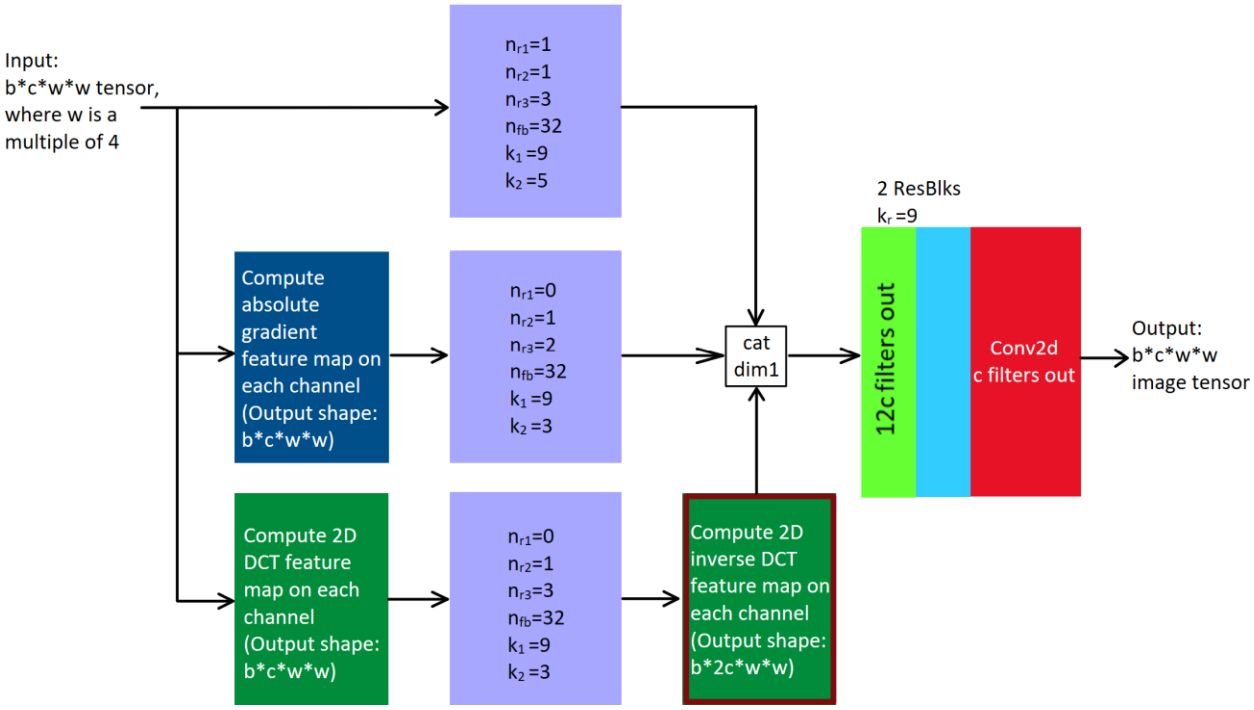



*Figure 4 Generator Model*

In Figures 2,3 and 4, $b$ is the batch size, $c = c_0$ and $w = w_0$. Given the low number of residual blocks assigned in Figure 4, this is a shallow version of such generator model. With merely less than 900k parameters, this model is highly efficient to train and evaluate with excellent performance, thanks to its novel structure. Due to the nature of residual networks, increasing the number of residual blocks used should further improve its performance, while pushing the demand for GPU resources and time.

In Figure 4, the absolute gradient $\nabla(x)$ of an image matrix $y$ with $c$ channels is computed per channel, as in Eq.3.

$$\nabla(x) = [\nabla(x_1), \nabla(x_2), \dots, \nabla(x_c)] \tag{3}$$

For each channel $n$ the gradient $\nabla(x_n)$ is computed as Eq.4.

$$\nabla(x_n) = \left((x_n \circledast \kappa_{sh})^{\circ 2} + (x_n \circledast \kappa_{sv})^{\circ 2}\right)^{\circ \frac{1}{2}} \tag{4}$$

While $\kappa_{s,h}$ and $\kappa_{s,v}$ are the horizontal and vertical Sobel gradient kernels respectively, indicated in Eq.5.

$$\kappa_{sh} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \kappa_{sv}^T \tag{5}$$

And $x_n \circledast \kappa$ represents the discrete convolution of kernel $\kappa$ on $x_n$.

Also, in Figure 4, for a 3-dimensional matrix $x$ representing the image with $c$ channels, the channel-wise 2D DCT (discrete cosine transform) is expressed in Eq.6:

$$\Gamma(x) = [\Gamma(\mathrm{x}_1), \Gamma(\mathrm{x}_2), \dots, \Gamma(\mathrm{x}_c)] \tag{6}$$

Similarly, for a 3-dimensional matrix $\Gamma(x)$ representing the DCT of an image $x$ with $c$ channels, the channel-wise 2D inverse DCT is expressed in Eq.7:

$$\Gamma^{-1}(\Gamma(x)) = [\Gamma^{-1}(\Gamma(\mathrm{x}_1)), \Gamma^{-1}(\Gamma(\mathrm{x}_2)), \dots, \Gamma^{-1}(\Gamma(\mathrm{x}_c))] = [\Gamma^{-1}(\Gamma(\mathrm{x})_1), \Gamma^{-1}(\Gamma(\mathrm{x})_2), \dots, \Gamma^{-1}(\Gamma(\mathrm{x})_c)] = x \quad (7)$$

The DCT of a $w * w$ matrix $x_n$, $\Gamma(x_n)$ is computed as Eq.8:

$$\Gamma(x_n)_{u,v} = \left(\frac{2}{w}\right) \sum_{i=1}^{w} \sum_{j=1}^{w} \left(\alpha_u \alpha_v \cos\left(\frac{\pi u}{2w}(2i-1)\right) \cos\left(\frac{\pi v}{2w}(2j-1)\right) x_{n,i,j}\right) \quad (8)$$

With

$$\alpha_\zeta = \begin{cases} \frac{1}{\sqrt{2}} & if\ \zeta = 1 \\ 1 & otherwise \end{cases} \quad (9)$$

Where $\Gamma(x_n)_{u,v}$ represents each element of $\Gamma(x_n)$ at row $u$ column $v$.

The inverse DCT of $\Gamma(x_n)$, $\Gamma^{-1}(\Gamma(x_n))$ is computed as Eq.10:

$$\Gamma^{-1}(\Gamma(x_n))_{u,v} = \left(\frac{2}{w}\right) \sum_{i=1}^{w} \sum_{j=1}^{w} \left(\alpha_i \alpha_j \cos\left(\frac{\pi i}{2w}(2u-1)\right) \cos\left(\frac{\pi j}{2w}(2v-1)\right) \Gamma(x_n)_{i,j}\right) = x_{n,u,v} \quad (10)$$

Note: A wavelet transform branch may be introduced in the future updates.

The discriminator structure is shown in Figure 5, with $b$ being batch size, $c = c_0$ and $w = w_0$.

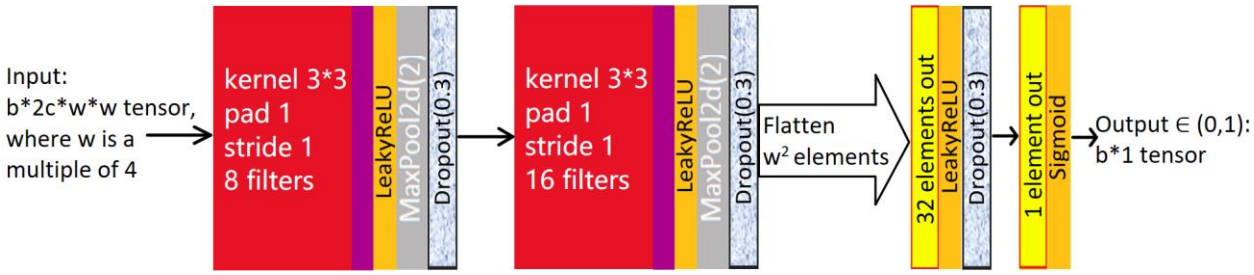



*Figure 5 Discriminator Model*

**Loss Functions for generator**

In this part, assume $c = c_0$ and $w = w_0$ holds. The total training loss is a weighted sum of 4 losses, including the content $L^2$ loss $L_{img}$, the gradient $L^2$ loss $L_{gr}$, the DCT loss $L_{dct}$, and the adversarial loss $L_{adv}$, as stated in Eq.11.

$$L_G = \omega_{img} L_{img} + \omega_{gr} L_{gr} + \omega_{dct} L_{dct} + \omega_{adv} L_{adv} \quad (11)$$

Where $\omega_{img}$, $\omega_{gr}$, $\omega_{dct}$ and $\omega_{adv}$ are the corresponding weighs of the 4 losses, respectively.

The content $L^2$ loss $L_{img}$ and the gradient $L^2$ loss $L_{gr}$ are expressed separately in Eq.12 and Eq.13.

$$L_{img}(G(x), y) = \frac{1}{w^2 c} \sum_{n=1}^{c} \sum_{i=1}^{w} \sum_{j=1}^{w} \left(G(x)_{n,i,j} - y_{n,i,j}\right)^2 \quad (12)$$

$$L_{gr}(G(x), y) = \frac{1}{w^2 c} \sum_{n=1}^{c} \sum_{i=1}^{w} \sum_{j=1}^{w} \left(\nabla(G(x))_{n,i,j} - \nabla(y)_{n,i,j}\right)^2 \quad (13)$$

Due to the nature of camera noise distribution, noises with different frequencies have varying amplitudes. Therefore, when evaluating the DCT loss

$$L_{dct}(G(x), y) = M_{L_{dct}}(\Gamma\big(G(x)\big) - \Gamma(y)) \tag{14}$$

Differences in coefficients for different frequencies should be handled differently. By experiments, $M_{L_{dct}}$ can be the sum of two terms, in which one term sums the absolute of each value with weighs and another term sums only positive values with weighs, so that the model is enhanced to preserve contrast while attenuating the majority of mid to low-frequency chromatic noise.

The adversarial loss is provided by the discriminator model during the simultaneous training of both models. The generator is intended to generate outputs that makes the discriminator to predict them as 0 (real), thus the loss is as stated in Eq.15.

$$L_{adv} = -\log\big(1 - D(G(x), y)\big) \tag{15}$$

**Loss function for discriminator**

The discriminator is trained simultaneously with the generator. Its purpose is to classify between generated and real images. During each update, three forms of inputs (image windows) are involved for training, while the labels are always the clean ground-truth image windows. The first form of inputs are identical to the ground-truth $y$, where the discriminator is supposed to classify such inputs as real (0). The second form of inputs are the noisy image windows, which are also used as inputs to train the generator, where the discriminator should classify them as fake (1). The third form of inputs are the outputs of the generator $G(x)$, which should be classified as fake (1). The expression is shown in Eq.16.

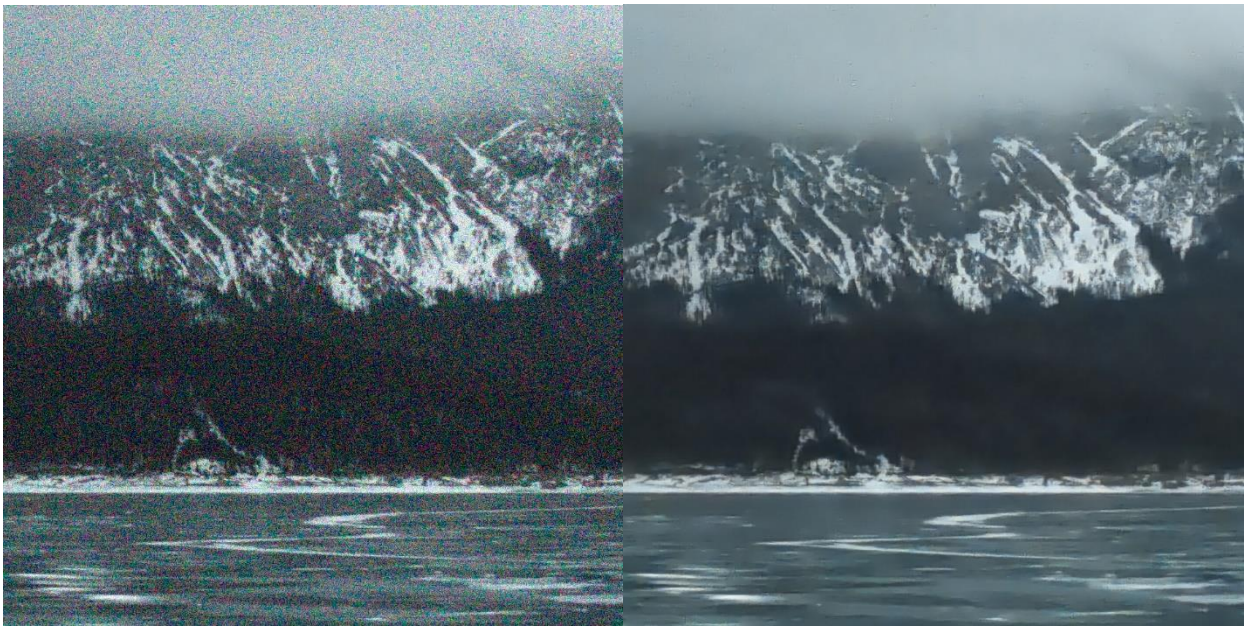
$$L_D = -\log\big(1 - D(y, y)\big) - \log\big(D(x, y)\big) - \log\big(D(G(x), y)\big) \tag{16}$$

**Test on real noisy photos**

This model is proved to well preserve details while removing almost all the noise and untrue color patterns. In the below pictures, the left side is part of the noisy photo, and the right side is the denoised output by the trained generator.

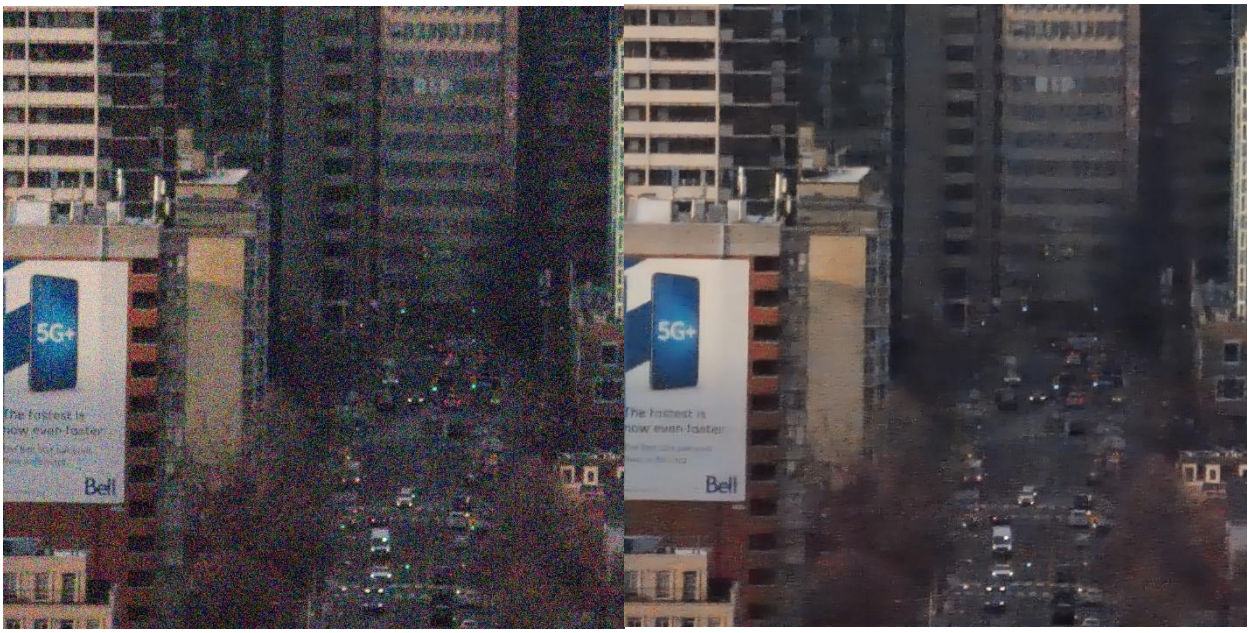
Panasonic ZS110, ISO640


Panasonic ZS110, ISO4000



Panasonic ZS110, ISO4000

Panasonic ZS110, ISO12800




Panasonic ZS110, ISO12800

Panasonic ZS110, ISO12800