Image Exposure Correction with Deep Neural Network

ZhuoYi Cai
Stanford University
zycai@stanford.edu

Abstract

Image exposure correction attempts to adjust incorrect exposure images such that the underlying details in the under or over exposed regions are recovered. The existing post-processing methods are not effective, for example, when the data is missing from the overblown region of the raw image. In this study, we explored an image exposure correction methods using deep neural network. We will propose and conduct experiments on two network architectures, ExpoNet and ExpoNetDouble, which are built based on auto-encoder architecture with novel skip connectivity. We will also see and compare the results from the networks trained using different image representation-LAB vs RGB. Finally, we will discuss what we have learned from this project and potential enhancements to the proposed architecture.

1. Introduction

Exposure settings (exposure triangle) have been a critical part of photography. They determine the overall brightness of the final rendered images. Digital cameras have also encapsulated sophisticated technologies to automatically adjust exposure settings, such as shutter speed, f-number, and ISO value, in the attempt to achieve the best possible result. However, images with wide dynamic range could still contain over or underexposed regions. High ISO value in low light situation could easily result in noisy image regardless of the most advanced noise reduction techniques in the state of the art image signal processing pipeline.

In this project, we attempt to leverage image reconstruction capability from convolutional auto-encoder architecture and symmetric connections to perform image exposure correction. This proposed solution could potentially provide a powerful tool for noise reduction due to the compact representation of the auto-encoder bottleneck design. At the same time, it may effectively restore over or underexposure regions of the images which allows super high dynamic range representation of the images.

2. Related Works

Image restoration and correction using deep neural networks has been studied and explored in multiple different literatures [1][2][4][7]. In particular, both Multi-Scale Photo Exposure Correction [1] and pix2pix [7] network are variations of GAN with the generator using auto-encoder network architecture. They both reported promising results for the problems they are designed to solve respectively. However, due to the nature of the GAN network, their solutions are complex and hard to train. For instance, Multi-Scale Photo Exposure Correction network, which inspires this project, attempts to recover both under and over exposed image regions with multiple-resolution representation (Laplacian pyramid) of input images.

In this project, we are proposing a network which are simpler and easier to train while attempting to produce results which are comparable to the state of the art solutions as in [1]. The design of the proposed network is based on auto-encoder architecture and borrows ideas from Deep Reciprocating HDR Transformation (DRHT) [2] and convolutional auto-encoder with symmetric skip connections [9]. In addition, we attempt to enhance the network performance by introducing a new type of skip connectivity called bridge skip connection. We will show all the details of the proposed networks and discuss the observations from our experiments in this report.

3. Proposed Method

In this project, we studied two proposed network architectures – ExpoNet and ExpoNetDouble. ExpoNet is a single auto-encoder architecture, and ExpoNetDouble consists of two ExpoNet with additional bridge skip connectivity and input bypass. We also conducted experiments on different image representation, such as LAB and RGB and observed how image representation affect the network performance. In this session, we will discuss all the details of the implementation of the proposed architectures and experiments.

3.1. Datasets

The proposed network is trained and evaluated using the

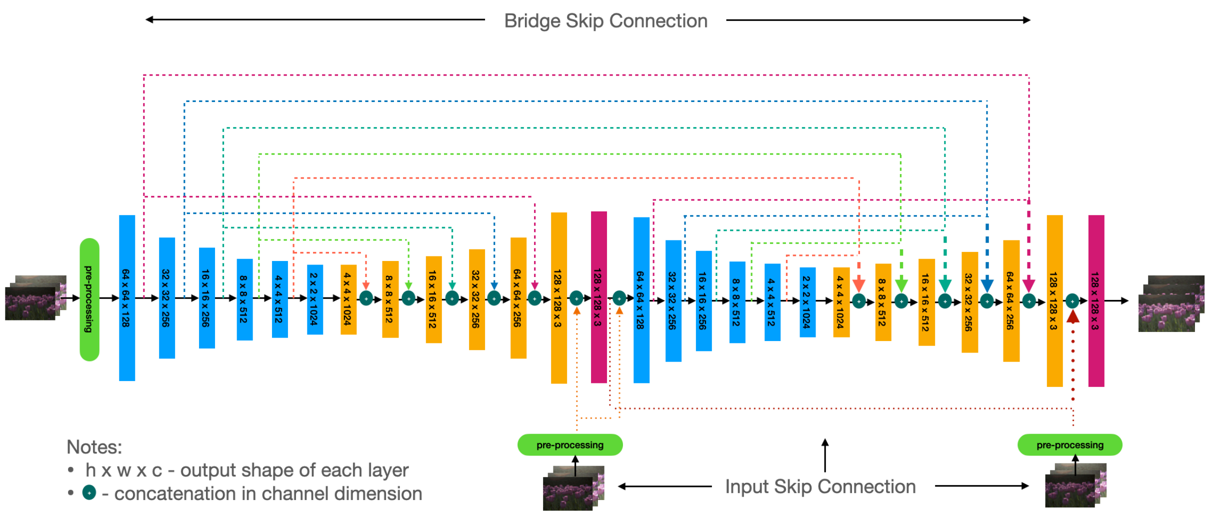


Figure 1 ExpoNetDouble with Skip Connections

same dataset from [1]. This dataset is rendered from the MIT-Adobe FiveK dataset [8], which has 5,000 raw-RGB images and engineered for exposure correction research purpose. Each raw RGB image is rendered via Adobe Camera Raw SDK with different digital EVs (−1.5, −1, +0, +1, and +1.5) to mimic underexposure, zero gain and overexposure errors. The ground truth images are generated via manually retouched by professionally photographer using ProPhoto RGB color space and converted to standard 8-bit sRGB color space. There are 24,330 8-bit sRGB images with different digital exposure settings. Training/validation/test datasets are split as the following, and they do not share any images in common. There are 17675 images in training set, 750 images in validation set and 5905 images in test set.

The images of the original data set have various sizes, mostly 1200x800. We first resized all the images to 128 (W) x128 (H) with bilinear interpolation via CV2 library and converted it to LAB and RGB image format from BGR format via tensorflow io library methods. The images are split into batches with 64 images per batch. 17664 images are used in training with 276 batches. 704 images are used for validation with 11 batches, and 5888 images are used for testing with 92 batches.

3.2. Architectures

3.2.1 ExpoNet

ExpoNet is built based on standard auto-encoder architecture. It consists of 6 down sampling convolutional layers (Kernal Size = 3; Stride = 2) with batch normalization and 5 up sampling de-convolutional layers. Leaky Relu is used as the activation functions for all the layers except the output layer which is represented as a purplish layer at the output in Figure 1. The activation of the output layer is sigmoid or tanh customized for RGB and LAB respectively. For regularization, L2 regularization with strength 0.2 is used for all layers and dropout with dropout rate 0.6 and 0.3 is used for first 3 and the last 2 up sampling layers respectively. Skip connection are also used to connect the corresponding down sampling and up sampling layers.

3.2.2 ExpoNetDouble

ExpoNetDouble consists of two back-to-back ExpoNet with additional skip connections, such as bridge skip connection and input skip connection as shown in Figure 1 to improve network training stability and network performance. We will see in the next section, there is mixed results observed between ExpoNetDouble and ExpoNet performance. And it is also observed that using LAB data format with this architecture is in general better over RGB as well but not without issues.

3.2.3 Skip Connection

The proposed networks are equipped with skip connections borrowing the ideas from [9]. The goal is to address two main problems in deep auto-encoder neural network for image reconstruction. First, with deep network, more layers of convolution and de-convolution result in a significant amount of details from the original image lost and corrupted. This phenomenon significant reduces image construction quality. We will discuss image defect in the last section. Second, deep networks, in general, suffer from vanishing gradients and are very difficult to train. This eventually contributes to the network performance degradation.

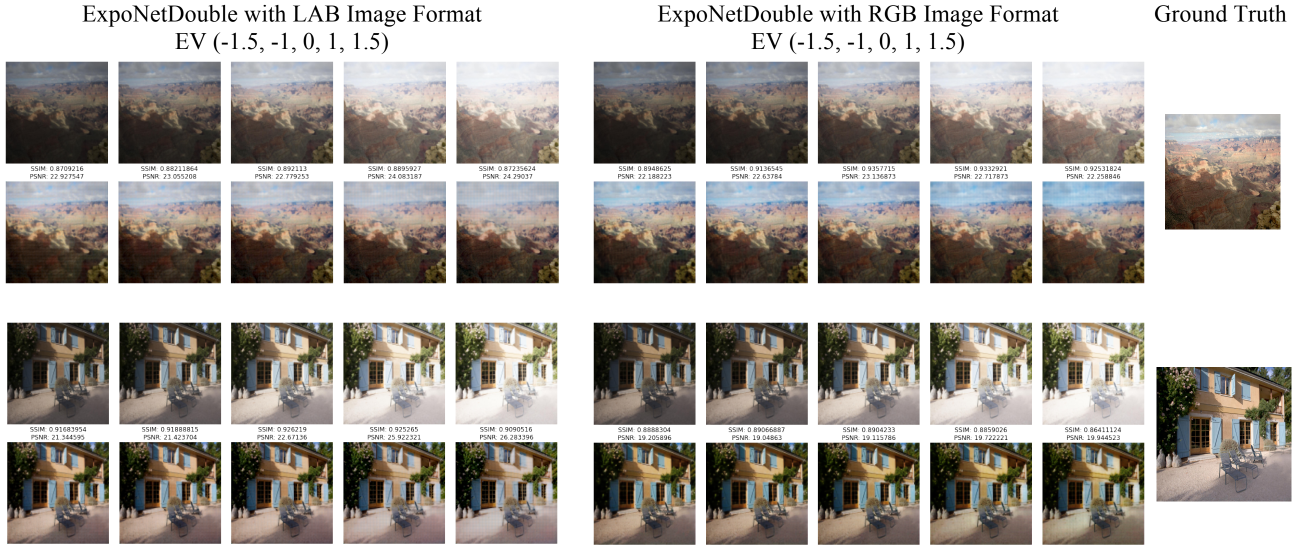


Figure 2 ExpoNetDouble Performance - LAB vs RGB Image Format

In order to address these issues, we added skip connections between every corresponding convolutional and de-convolutional layer within the network. In addition, we introduced a new skip connection called bridge skip connection in ExpoNetDouble to connect the output of the convolution layer of the first half of the network to the corresponding de-convolution layer of the second half of the network. We also experimented input bypass connection which concatenates the input image to the output of both decoder networks in channel dimension before the final output layer of the network as shown in Figure 1. As we will see in the next session, these network skip connections improve both training and image reconstruction performance.

3.2.4 Loss Function

All the networks built for this project uses L1 distance loss between the reconstructed and properly exposed ground truth images.

$$L_1 = \sum_{p=0}^{h*w*3-1} \|Y(p) - G(p)\| \tag{1}$$

where h and w denote the height and weight of the training/validation/test images. In our experiments, all images are resized to 128x128x3 (h x w x 3). p is the index of each pixel in the image. Y(.) and G(.) represent image reconstruction function and ground truth function respectively. The ground truth function represents the manual post-processing of the raw image data.

3.2.5 Metrics

The evaluation metrics used in this project are SSIM and PSNR. SSIM [10] provides a way to measure the similarity in perceived quality between the reconstructed and ground truth images while PSNR measures the reconstruction quality for the generated images.

4. Experiments and Results

4.1. Compute Environment

Colab compute environment is used for all the experiments and network training in this study. GPU and HIGH RAM (26GB) runtime setting are deployed. However, the GPU assignment is solely random based on Google algorithm.

4.2. LAB vs RGB Representation

RGB is the most common image representation. It consists of three channels - R (red), G (green) and B (blue). Like RGB, LAB color space is also a 3-dimensional representation with Lightness axis, a-axis (green to red), and b-axis (blue to yellow). Since the Lightness is designed to approximate human vision, LAB is commonly considered as a more accurate image representation than RGB. From Figure 2, we can see, in general, ExpoNetDouble trained with LAB format has significant better performance in most of the cases in terms of SSIM/PSNR. And the reconstructed images have more natural color. However, as it is shown in Figure 2, ExpoNetDouble with RGB format occasionally has better rating in SSIM, such as the "Canyon" images. This may be due to the ground truth image is more saturated in color. In fact, the reconstructed images from ExpoNetDouble trained with RGB format have relatively more saturated color.

4.3. Input Bypass/Skip Connection

Input skip connection as depicted in Figure 1 is to concatenate the input image to the output of both decoder

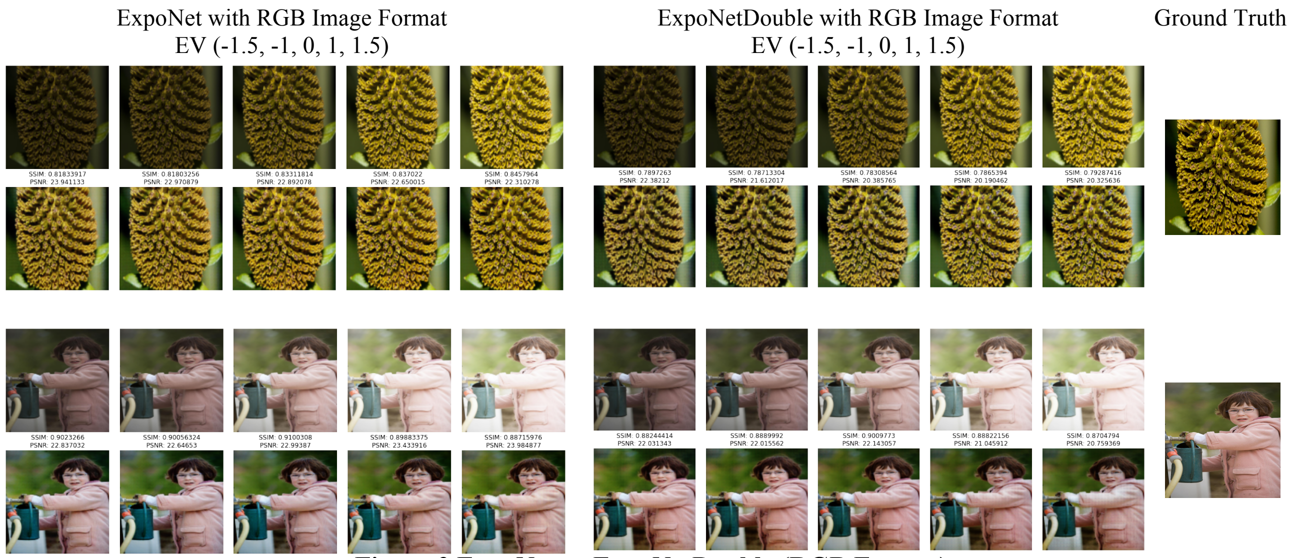


Figure 3 ExpoNet vs ExpoNetDouble (RGB Format)

networks in channel dimension before the final output layer of the network.

4.4. Bridge Skip Connection

Bridge skip connection in ExpoNetDouble is to connect the convolution layer of the first half of the network to the corresponding de-convolution layer of the second half of the network.

Table 1 lists the metrics obtained from the average of the test dataset. The "Base" represents the network with both input and bridge skip. Based on this observation, Input Skip Connection is closely correlated with bridge Skip connection. With both skip connectivity in place, the network performs slightly better in our experiment.

Test Metrics	Base	w/o Input Skip	w/o Input/Bridge Skip
SSIM	0.8518	0.8040	0.8451
PSNR	20.28	19.22	19.96

Table 1 Input and Bridge Skip Connection Observations

Another observation is that. The training stability seems better when both skip connections are present. As shown in Figure 4, it is consistently observed that there are spikes in Loss/SSIM/PSNR metrics during training at the early epochs. We haven't observed any of these spikes when the network equipped with both skip connections.

4.5. Small vs Double ExpoNet

ExpoNetDouble stacks two ExpoNet back-to-back with additional skip connections, such as bridge skip connection and input skip connection as shown in Figure 1. The idea is borrowed from [2] in the attempt to improve performance. Both versions of the network are trained with the same training/validation/test dataset with 100 epochs and RGB format. However, in terms of SSIM and PSNR, it turns out the simpler ExpoNet is superior than the more complex ExpoNetDouble as shown in Figure 3. We also observed that even though the simpler ExpoNet seems to perform better, the more complex ExpoNetDouble has better training stability and less likely has image defects. We will be highlighting a couple in the discussion section.

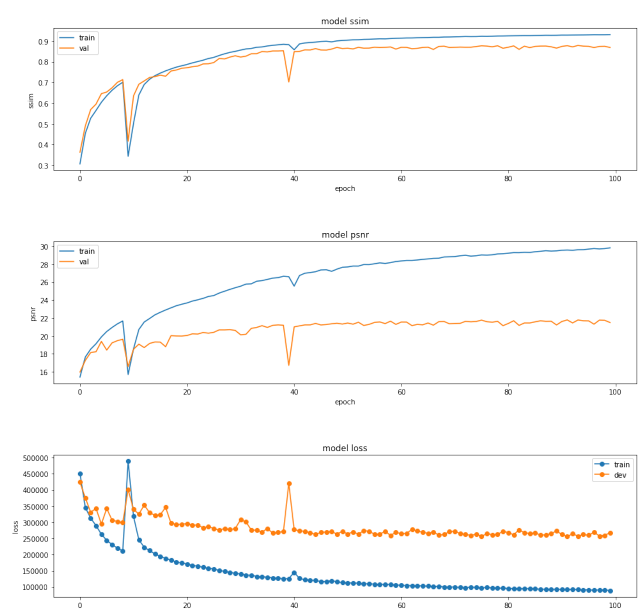


Figure 4 ExpoNetDouble without Input and Bridge skip connections

4.6. Over/Under exposure Image Correction

Figure 2 and 3 show the demo images from EV -1.5, -1, 0, 1, 1.5 adjustment and the corresponding reconstructed images from the network. The network trained with LAB format from Figure 2 seems to have more consistent results across different EV samples. The range of variation in SSIM is within +/- 0.02. However, since we didn't have time to conduct more testing with different types of images, such as portraits, landscape, flowers, etc., we cannot make a concrete conclusion at this time.

4.7. Noisy Image Correction

In this experiment, we added Gaussian noise (mean = 0, standard deviation = 1, scale = 10) to the input images and observed how this additional noise affects the network performance. It turns out the network's image reconstruction performance degrade along with the scale of the Gaussian noise. As shown in Figure 5, using ExpoNetDouble trained with RGB format, SSIM goes from 0.9009773 to 0.8243844 and PSNR goes from 22.143057 to 21.647007. The image quality degradation is approximately 8% in SSIM and 2.3% in PSNR. So, the network doesn't seem to correct external Gaussian noise effectively.

Figure 5 Noisy Input Images Example

5. Discussion

5.1. Result Comparison to Related works

In our experiments, the best result we observed from ExpoNetDouble with LAB is slightly worse than the one presented in [1]. However, our method is much simpler and still has plenty of room to be improved.

Input Image	Ground Truth	Ours	From [1]
		PSNR = 25.93 SSIM = 0.9001	PSNR = 25.68 SSIM = 0.931

Figure 6 Result Comparison to Related Work

5.2. Resolution Degradation and Image Defects

The reconstructed image quality from our networks is not as good as expected. We observed a couple issues from our experiments.

First, the image resolution of the reconstructed images is reduced, and there is a square pattern on top of the reconstructed images. This may be due to the interpolation for the image resizing in the pre-processing stage. See the example in Figure 7.

Pattern Defect | Color Defect

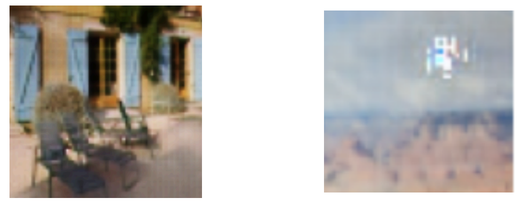


Figure 7 Image Defect Examples

The other defect we observed is the color defect. This occurs only in the highlight/overblown area of the image with the simpler ExpoNet trained with data in LAB format. It will need further investigation to find out the root cause. One example is shown in Figure 7.

5.3. Training with patches

One idea to solve the resolution degradation is to train the network with patches. But the challenge here is the color consistency among patches within one image. This could be another interesting experiment if the time is allowed.

5.4. Skip connections

The skip connections we experimented on are seen to have a certain impact on the network performance. However, the result is not as significant as expected. This will require further study before making a concrete conclusion.

6. Conclusion

Finally, in this project, the proposed networks provide a much simpler solution to the exposure correction problem. Even though there are a few issues observed, the network performance is very close to the more complex solution in [1][7]. And there are techniques we can further experiment to solve the observed issues. Thus, we demonstrated the possibility of a simpler solution. But it also requires further work to improve the model in order to achieve the state of the art result presented in other literatures.

References

[1] Afifi, M., Derpanis, K., Ommer, B., and Brown, M. "Learning Multi-Scale Photo Exposure Correction," CVPR 2021, arXiv:2003.11596

[2] Yang, X., Xu, K., Song, Y., Zhang, Q., Wei, X., and Lau, R. "Image Correction via Deep Reciprocating HDR Transformation," CVPR 2018, arXiv:1804.04371

[3] Schwartz, E., Giryes, R., and Bronstein, A., "DeepISP: Toward Learning an End-to-End Image Processing Pipeline," IEEE Transactions on Image Processing 28.2 (2019): 912-923 arXiv:1801.06724

[4] Chen C., Chen, Q., Xu, J., and Koltun, V., "Learning to see in the dark," CVPR, 2018, arXiv:1805.01934

[5] Müller, M. U., Ekhtiari, N., Almeida, R. M., and Rieke, C. "SUPER-RESOLUTION OF MULTISPECTRAL SATELLITE IMAGES USING CONVOLUTIONAL NEURAL NETWORKS," ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., V-1-2020, 33–40, https://doi.org/10.5194/isprs-annals-V-1-2020-33-2020, 2020

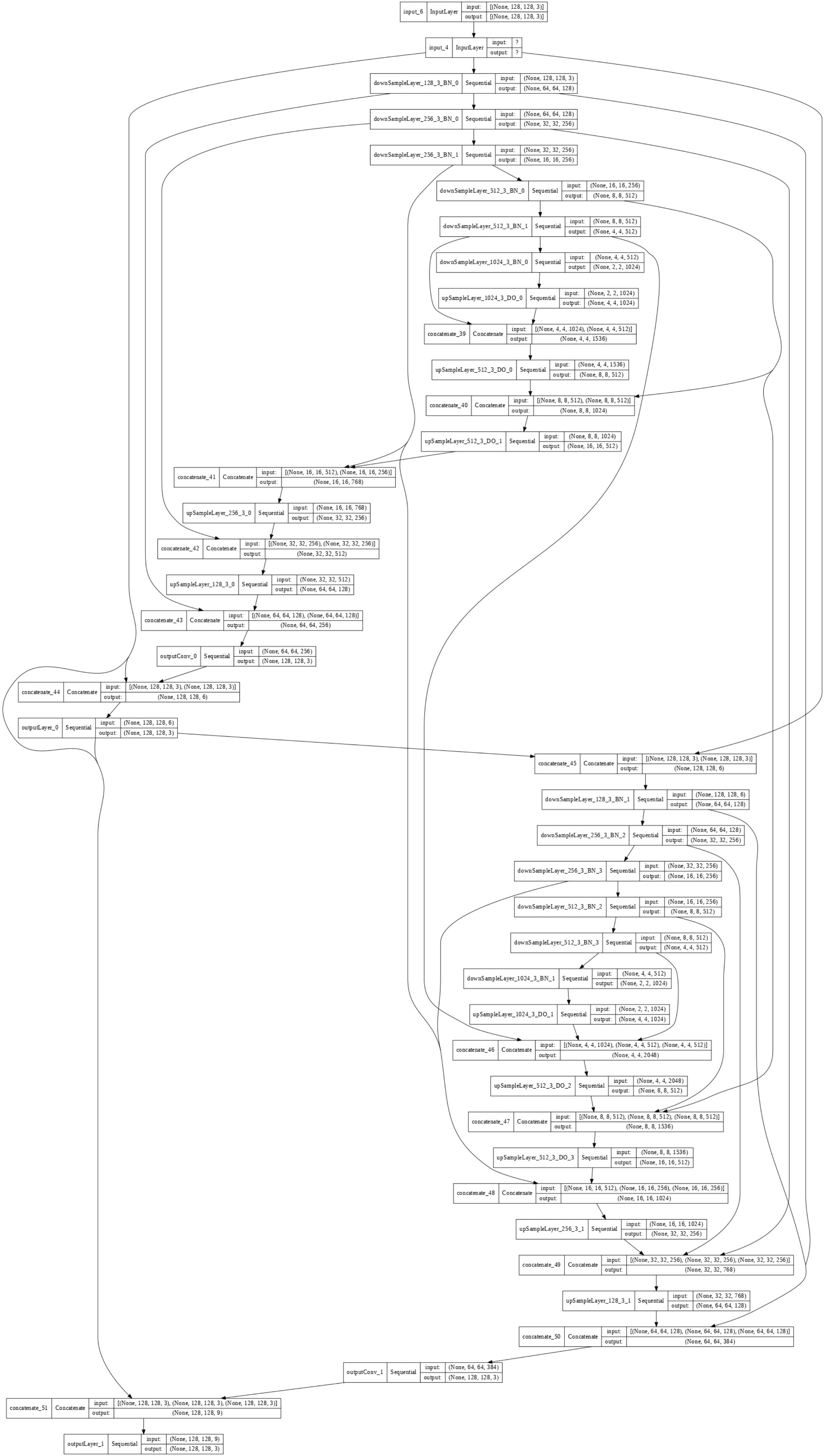
[6] Ronneberger, O., Fischer, P., Brox, T., "U-Net: Convolutional Networks for Biomedical Image Segmentation," MICCAI 2015, arXiv:1505.04597

[7] Isola, P., Zhu, J.-Y., Zhou, T., Efros, A., "Image-to-Image Translation with Conditional Adversarial Networks," CVPR 2017, arXiv:1611.07004

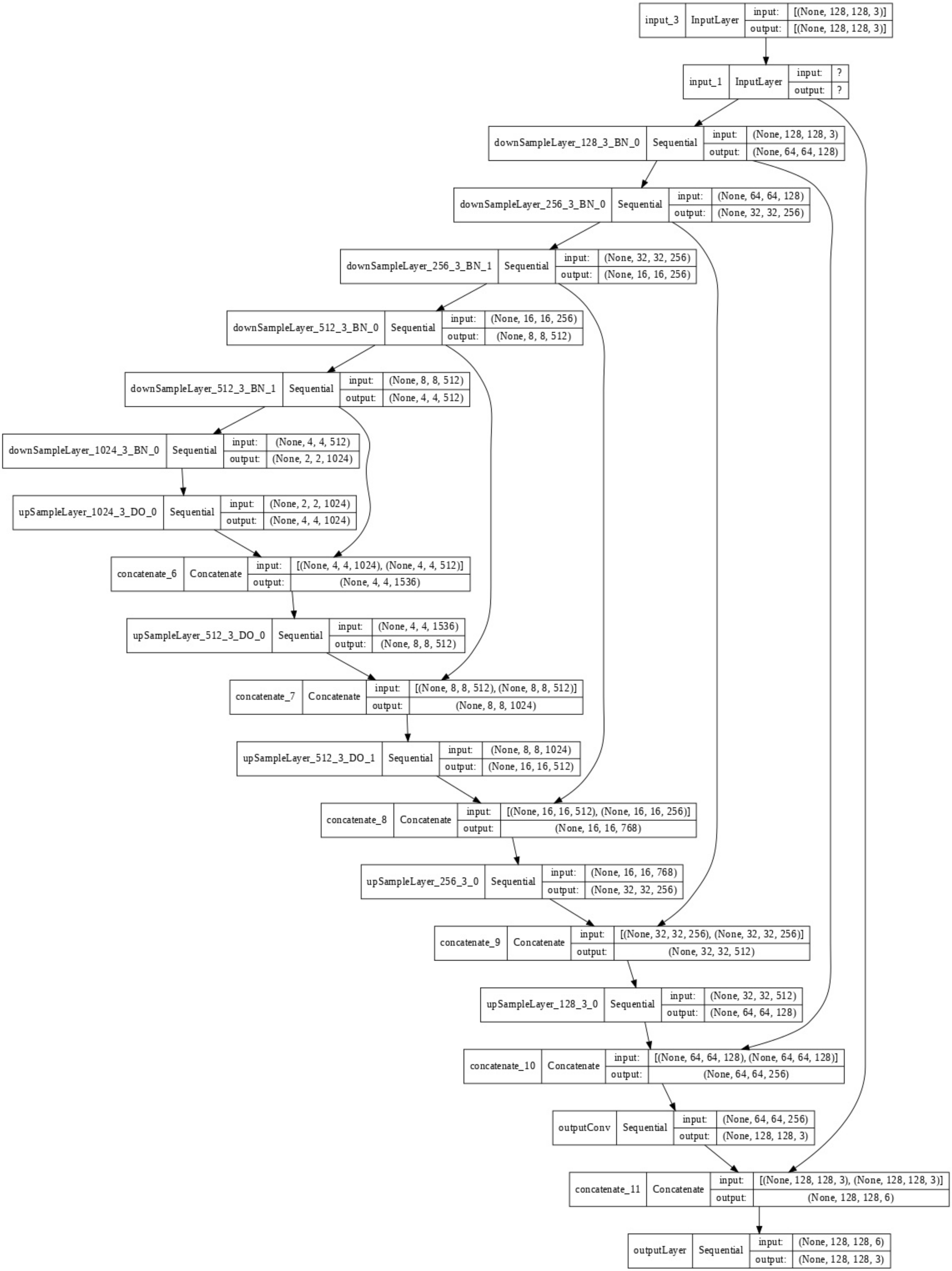
[8] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. "Learning photographic global tonal adjustment with a database of input / output image pairs," In CVPR, 2011.

[9] Xiao-Jiao Mao, Chunhua Shen, Yu-Bin Yang "Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections" In CVPR, 2016.

[10] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to struc- tural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.



input_6 InputLayer input: [(None, 128, 128, 3)] output: [(None, 128, 128, 3)]
input_4 InputLayer input: ? output: ?
downSampleLayer_128_3_BN_0 Sequential input: (None, 128, 128, 3) output: (None, 64, 64, 128)
downSampleLayer_256_3_BN_0 Sequential input: (None, 64, 64, 128) output: (None, 32, 32, 256)
downSampleLayer_256_3_BN_1 Sequential input: (None, 32, 32, 256) output: (None, 16, 16, 256)
downSampleLayer_512_3_BN_0 Sequential input: (None, 16, 16, 256) output: (None, 8, 8, 512)
downSampleLayer_512_3_BN_1 Sequential input: (None, 8, 8, 512) output: (None, 4, 4, 512)
downSampleLayer_1024_3_BN_0 Sequential input: (None, 4, 4, 512) output: (None, 2, 2, 1024)
upSampleLayer_1024_3_DO_0 Sequential input: (None, 2, 2, 1024) output: (None, 4, 4, 1024)
concatenate_39 Concatenate input: [(None, 4, 4, 1024), (None, 4, 4, 512)] output: (None, 4, 4, 1536)
upSampleLayer_512_3_DO_0 Sequential input: (None, 4, 4, 1536) output: (None, 8, 8, 512)
concatenate_40 Concatenate input: [(None, 8, 8, 512), (None, 8, 8, 512)] output: (None, 8, 8, 1024)
upSampleLayer_512_3_DO_1 Sequential input: (None, 8, 8, 1024) output: (None, 16, 16, 512)
concatenate_41 Concatenate input: [(None, 16, 16, 512), (None, 16, 16, 256)] output: (None, 16, 16, 768)
upSampleLayer_256_3_0 Sequential input: (None, 16, 16, 768) output: (None, 32, 32, 256)
concatenate_42 Concatenate input: [(None, 32, 32, 256), (None, 32, 32, 256)] output: (None, 32, 32, 512)
upSampleLayer_128_3_0 Sequential input: (None, 32, 32, 512) output: (None, 64, 64, 128)
concatenate_43 Concatenate input: [(None, 64, 64, 128), (None, 64, 64, 128)] output: (None, 64, 64, 256)
outputConv_0 Sequential input: (None, 64, 64, 256) output: (None, 128, 128, 3)
concatenate_44 Concatenate input: [(None, 128, 128, 3), (None, 128, 128, 3)] output: (None, 128, 128, 6)
outputLayer_0 Sequential input: (None, 128, 128, 6) output: (None, 128, 128, 3)
concatenate_45 Concatenate input: [(None, 128, 128, 3), (None, 128, 128, 3)] output: (None, 128, 128, 6)
downSampleLayer_128_3_BN_1 Sequential input: (None, 128, 128, 6) output: (None, 64, 64, 128)
downSampleLayer_256_3_BN_2 Sequential input: (None, 64, 64, 128) output: (None, 32, 32, 256)
downSampleLayer_256_3_BN_3 Sequential input: (None, 32, 32, 256) output: (None, 16, 16, 256)
downSampleLayer_512_3_BN_2 Sequential input: (None, 16, 16, 256) output: (None, 8, 8, 512)
downSampleLayer_512_3_BN_3 Sequential input: (None, 8, 8, 512) output: (None, 4, 4, 512)
downSampleLayer_1024_3_BN_1 Sequential input: (None, 4, 4, 512) output: (None, 2, 2, 1024)
upSampleLayer_1024_3_DO_1 Sequential input: (None, 2, 2, 1024) output: (None, 4, 4, 1024)
concatenate_46 Concatenate input: [(None, 4, 4, 1024), (None, 4, 4, 512), (None, 4, 4, 512)] output: (None, 4, 4, 2048)
upSampleLayer_512_3_DO_2 Sequential input: (None, 4, 4, 2048) output: (None, 8, 8, 512)
concatenate_47 Concatenate input: [(None, 8, 8, 512), (None, 8, 8, 512), (None, 8, 8, 512)] output: (None, 8, 8, 1536)
upSampleLayer_512_3_DO_3 Sequential input: (None, 8, 8, 1536) output: (None, 16, 16, 512)
concatenate_48 Concatenate input: [(None, 16, 16, 512), (None, 16, 16, 256), (None, 16, 16, 256)] output: (None, 16, 16, 1024)
upSampleLayer_256_3_1 Sequential input: (None, 16, 16, 1024) output: (None, 32, 32, 256)
concatenate_49 Concatenate input: [(None, 32, 32, 256), (None, 32, 32, 256), (None, 32, 32, 256)] output: (None, 32, 32, 768)
upSampleLayer_128_3_1 Sequential input: (None, 32, 32, 768) output: (None, 64, 64, 128)
concatenate_50 Concatenate input: [(None, 64, 64, 128), (None, 64, 64, 128), (None, 64, 64, 128)] output: (None, 64, 64, 384)
outputConv_1 Sequential input: (None, 64, 64, 384) output: (None, 128, 128, 3)
concatenate_51 Concatenate input: [(None, 128, 128, 3), (None, 128, 128, 3), (None, 128, 128, 3)] output: (None, 128, 128, 9)
outputLayer_1 Sequential input: (None, 128, 128, 9) output: (None, 128, 128, 3)


input_3 | InputLayer | input: [(None, 128, 128, 3)] | output: [(None, 128, 128, 3)]
input_1 | InputLayer | input: ? | output: ?
downSampleLayer_128_3_BN_0 | Sequential | input: (None, 128, 128, 3) | output: (None, 64, 64, 128)
downSampleLayer_256_3_BN_0 | Sequential | input: (None, 64, 64, 128) | output: (None, 32, 32, 256)
downSampleLayer_256_3_BN_1 | Sequential | input: (None, 32, 32, 256) | output: (None, 16, 16, 256)
downSampleLayer_512_3_BN_0 | Sequential | input: (None, 16, 16, 256) | output: (None, 8, 8, 512)
downSampleLayer_512_3_BN_1 | Sequential | input: (None, 8, 8, 512) | output: (None, 4, 4, 512)
downSampleLayer_1024_3_BN_0 | Sequential | input: (None, 4, 4, 512) | output: (None, 2, 2, 1024)
upSampleLayer_1024_3_DO_0 | Sequential | input: (None, 2, 2, 1024) | output: (None, 4, 4, 1024)
concatenate_6 | Concatenate | input: [(None, 4, 4, 1024), (None, 4, 4, 512)] | output: (None, 4, 4, 1536)
upSampleLayer_512_3_DO_0 | Sequential | input: (None, 4, 4, 1536) | output: (None, 8, 8, 512)
concatenate_7 | Concatenate | input: [(None, 8, 8, 512), (None, 8, 8, 512)] | output: (None, 8, 8, 1024)
upSampleLayer_512_3_DO_1 | Sequential | input: (None, 8, 8, 1024) | output: (None, 16, 16, 512)
concatenate_8 | Concatenate | input: [(None, 16, 16, 512), (None, 16, 16, 256)] | output: (None, 16, 16, 768)
upSampleLayer_256_3_0 | Sequential | input: (None, 16, 16, 768) | output: (None, 32, 32, 256)
concatenate_9 | Concatenate | input: [(None, 32, 32, 256), (None, 32, 32, 256)] | output: (None, 32, 32, 512)
upSampleLayer_128_3_0 | Sequential | input: (None, 32, 32, 512) | output: (None, 64, 64, 128)
concatenate_10 | Concatenate | input: [(None, 64, 64, 128), (None, 64, 64, 128)] | output: (None, 64, 64, 256)
outputConv | Sequential | input: (None, 64, 64, 256) | output: (None, 128, 128, 3)
concatenate_11 | Concatenate | input: [(None, 128, 128, 3), (None, 128, 128, 3)] | output: (None, 128, 128, 6)
outputLayer | Sequential | input: (None, 128, 128, 6) | output: (None, 128, 128, 3)