Iterative Coarse-to-Fine Image Colorization with GAN

Timo Streule
16-938-706
tstreule@ethz.ch

Thomas Rüegg
16-926-396
rueeggth@ethz.ch

Marc Styger
16-922-767
stygerma@ethz.ch

***Abstract*—The idea of colorizing images is a well-known problem in machine learning. Current research focuses on training neural networks where (discrete) convolution layers commonly play an important role. However, the colorization may be inconsistent across large images, as such convolutions have a limited range of "vision" and, thus, image regions will be colorized independently from other, distant image regions. In this work, we propose a new design to solve the problem of coloring high-resolution images by scaling and coloring input images "from coarse to fine" in an iterative manner. Starting with a pixelated version of the image, the result of each iteration serves as a color bias for the next larger pixelated version until we reach the final image size.**

***Index Terms*—Image colorization, GAN, cGAN, CNN.**

1. Introduction

THE problem of colorizing gray-scale images has been approached in various ways. For instance, one can rely on significant user interaction [1] but this significantly slows down the process due to the need of human labour. Other approaches pose the problem as a classification task [2, 3] or, conversely, use a regression loss as training objective [4].

Recently generative adversarial networks (GANs) have been used to tackle the colorization task with a lot of success. GANs are better at creating vibrant colors than previous methods [5]. We also based our approach on a GAN that is conditional (cGAN) and inspired by Isola et al. [6]. This allows us to bias the GAN on the gray-scale images we get as an input.

Advances in image colorization are not based on GANs anymore as GANs lack the ability to produce a variety of possible colorization. Royer et al. [7] introduce an autoregressive model for the task of image colorization. By embedding a gray-scale image with a deep convolutional neural net and using the result to bias convolutions of the autoregressive model they achieve highly saturated images. The results for objects in the image with no clearly determined color (e.g. a shirt or a book) are diverse as promised. Also Kumar et al. [8] achieve this property by their approach. They spatially downsample a given gray-scale image and restrict the color space. The low resolution image gets coarsely colorized with a conditional Axial Transformer [9]. In the following, they created a model to first upsample the color and another model to then upsample the image's space (i.e. pixel density). All three sub-models incorporate self-attention blocks and together provide a good system which outperformed previous state-of-the-art. This method is similar to ours in terms of the idea to downsample the gray-scale image first and colorizing the lower resolution picture which is in turns used to bias further computations.

Another alternative colorization method to GANs is given by CycleGANs [10]. Shanshan et al. [11] improved the original CycleGAN by introducing skip connection that should help to overcome the vanishing gradient problem and provide better color details. They outperformed other approaches when using datasets where all depicted objects were similar but got outperformed for more diverse datasets.

One drawback of some image colorization methods based on deep learning is the amount of required hardware resources. Especially when increasing the image resolution the process gets more resource intensive. This is mainly due to increased depth of the network. This depth is required to produce consistent colors without artifacts even if the resolution is very high. This is where our approach comes in. Our goal is to enable a relatively shallow model to colorize pictures on multiple scales and use the predictions from lower resolutions to improve on the predictions at higher resolutions. We call our method *coarse-to-fine (C2F)*. The result is a process with less hardware intensity at the cost of longer runtimes.

2. Models and Methods

2.1. Dataset and pre-processing

The image data is obtained from the popular *Microsoft COCO* dataset with over 328K images [12]. Since we were limited in time and, especially, computational power, we used a sample size of only 21'837 images with a resolution of 640x640 pixels. Out of those images, we u.a.r. have sampled 15'250 images (70%) for the *train* split, and 3'275 images (15%) each for the *test* and *validation* split.

In image data preprocessing, it is common practice to use various data augmentation techniques to artificially increase the size of the data set, achieve better generalization, and avoid overfitting. However, for the same reasons as above, namely available resources, we did not use these techniques, except that we used random horizontal flips.

The most notable and probably most important step is the conversion from RGB to the CIELAB image space. The goal of CIELAB is to create a color space that approximates human vision. Its design based on perceptual uniformity makes it ideal for computer processing and,

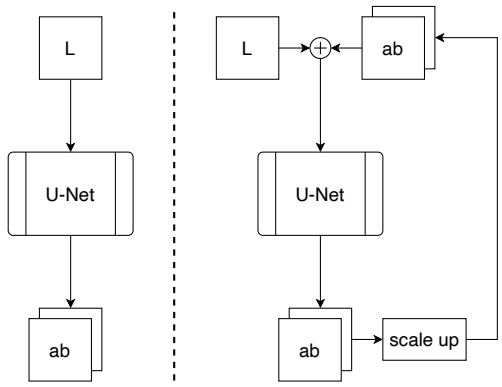


Figure 1: The base model generator on the *left* and a simplified view of the C2F generator on the *right*. Please note that the images are displayed in the CIELAB image space.

since the L channel corresponds to the grayscale levels of an image, a model only needs to predict channels a and b. Thanks to these properties, the CIELAB image space is very commonly used for image colorization tasks [2, 13], which is why we used it.

2.2. Objective

As general solution to image-to-image tasks in deep learning, the authors in [6] proposed a combination of two losses for the discriminator objective, which is how we implemented it in the code. The usage of a conditional generative adversarial network (cGAN) helps to solve the problem in an unsupervised manner and to produce good-looking colorful images that seem real. Combining this with the L1 loss, which makes it a *regression* task, introduces some supervision. The final objective is a λ-weighted sum of both objectives

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \qquad (1)$$

where the two losses are defined as $\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,y}[\log(1 - D(x, G(x, z))]$ and $\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]$. Note that G denotes the image generator and D the discriminator.

2.3. Network architectures

Two GAN architectures were implemented for this project. First, a *base* model is presented whose performance will serve as the basis for the second model. Then, our proposed *C2F* model is explained with its differences from the base model.

2.3.1. Base model. As it is in the nature of GANs, the base model consists of a generator and a discriminator [14]. Following the argument in [6], both the input and output structure (pixel-to-pixel) are roughly aligned. Adding skip connections that follow the shape of a "U-Net" [15] is therefore adequate. In our case, the *generator* is a "U-Net" with a pretrained ResNet18 [16] as a backbone. As shown on the left in Figure 1, the generator takes a gray-scale image (L channel) as input and generates the two additional channels a and b.

We implemented the *discriminator* by stacking blocks of strided 2D convolutions, batch normalization, and Leaky ReLU elements to decide whether the input image is fake or real. Note that, in comparison to a "vanilla" discriminator, the model outputs one number for every patch (receptive field). This choice seems reasonable because a "vanilla" discriminator cannot take care of local subtleties of an image [17].

2.3.2. C2F model. The *discriminator* is the same as that in the basic model. Although the *generator* is also very similar, there are two main differences. First, it takes three color channels instead of one gray-scale channel as input, which allows to additionally process colors from previous coloring steps. Second, as already indicated, the coloring consists of several coloring steps instead of just one as in the base model. The iterative *or* recursive behavior of the C2F model is shown on the right in Figure 1. Starting with the colorization of a pixelated version of the original input, the generated output is enlarged[1] and concatenated with the next larger pixelated version. This process is repeated until the original input size is reached, the output of which is then the final result. Since no coloring bias is available in the first iteration, a zero-filled array is created for channels a and b, respectively.

In contrast to the base model, we hope that this iterative behavior will help improve color consistency throughout the image and produce good results when viewed from a distance, but also when viewed up close. The first iteration(s) should act as a coarse color stroke, while each iteration refines the coloring taking into account the coloring bias from the previous iteration.

2.4. Optimization and inference

We use the same hyperparameters as proposed in [6, Section 3.3]. That is, we use minibatch SGD and apply the Adam solver [18], with a learning rate of 0.0002, and momentum parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$. The optimizing step is equivalent for both architectures and is done after image generation using the objective as described in Equation 1. At inference time, we run the generator net in exactly the same manner as during the training phase. In our experiments, we use a batch size of 32, with the images distributed across 8 GPUs.

2.4.1. Hyperparamters C2F. Our model has three different adjustable parameters that are worth mentioning. Tuning these parameters can improve the quality or performance of the model.

- `shrink_size`: A scalar factor for resizing images. Assuming the input image has the resolution R, then the second to last iteration will take an image with a reduced resolution of $\hat{R} = {}^{R}/_{\texttt{shrink\_size}}$.
- `max_c2f_depth`: The maximal amount of iterations *or* recursion steps a forward pass will take.
- `min_ax_size`: The smallest allowed axis length of an image for doing a recursion step.

1. In subsubsection 2.4.1, we call the up-scaling factor "shrink size."

2.5. Implementation and source code

The complete code of the project can be found on GitHub[2]. Furthermore, the fully trained models can be downloaded from Polybox[3] together with the training logs. The code is implemented based on PyTorch[4] and the PyTorch-Lightning[5] framework, where the latter greatly simplifies the implementation of distributed training on multiple GPUs. Both, the base and the C2F model, can be called via the `main.py` function. You only need to pass the flag `--model=base` and `--model=c2f` respectively as argument.

2.6. Evaluation Metrics

The evaluation of the quality of these colorization methods is a non trivial task. Where other machine learning tasks can rely on MSE or MAE to give a good metric to test its methods, this will fail for image colorization tasks. The first problem is that these metrics do not work well when there is noise or a constant shift in the values present. The error will be quite high even though for a human the image will look fine. For this reason new methods have been developed such as peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM) [19] to better account for the human perception of quality. This works well in cases where there is degradation and when we want to quantify the loss in quality.

The task of colorizing an image however cannot be evaluated meaningfully by these methods. This is due to the inherent multi modality of the problem. For example if you want to colorize a black and white image of an apple you could make the apple either green, red or a mix of both. This means we have multiple plausible colors to pick from. So a picture might look plausible to a human observer but not to the computer who compares the image to the ground truth. Since we want a plausible colorization and not necessarily exactly the ground truth the only way to actually check for the quality is to ask a person which one looks better.

The GAN loss we use to actually train the network is unfortunately also not a good metric since this loss is not supposed to go to zero while training and is tailored to our specific generator and not to any other model.

3. Results

Throughout this section, we will demonstrate the results of the base model as well as several variations of our C2F. All images shall be compared to the ground truth in Figure 2a and were deducted from its gray-scale version Figure 2b. As long as nothing else is claimed, an image size of 1500x1000 pixels, a shrink size of 1.2 and maximum iteration number of 100 are to be assumed in each case.

2. https://github.com/tstreule/c2f_img_color
3. https://polybox.ethz.ch/index.php/s/uwF5Gml6rJjb0QY
4. https://pytorch.org/
5. https://www.pytorchlightning.ai/

(a) Ground truth. **(b)** Gray-scale version.

Figure 2: Ground truth and its gray-scale version [Src: 20].

(a) 1.1 **(b)** 1.5 **(c)** 2.0 **(d)** 3.0

Figure 3: Results at different shrink sizes.

3.1. Hyperparameters

Our model has different hyperparameters. In particular, we have evaluated the impact of the shrink size (`shrink_size`), the maximum number of iterations (`max_c2f_depth`) and minimal resolution (`min_ax_size`) on the quality of the colorization.

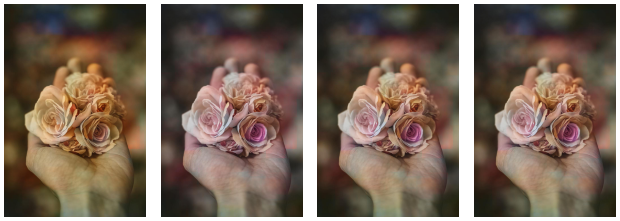
3.1.1. Shrink size. The shrink size has quite a big impact on the runtime, the smaller the shrink size the longer the runtime. This makes sense since this will lead to more iterations with higher resolutions.

The differences between the images here are quite subtle, but one thing that stands out, is that the small shrink sizes produce seemingly smoother colors compared to larger ones. At shrink size 3.0 there are some bright saturated spots that stand out and don't seem to match the surrounding. These spots disappear at the smaller shrink sizes.

3.1.2. Number of iterations. We tested the model at 0, 1, 10 and 30 iterations. As you can see in Figure 4, it seems like the colors converge over increasing iterations. There is a noticeable difference in hue between 4a and 4b but after that the differences are getting more subtle. To note here is, that the model cannot use the full 30 iterations because the picture can only be downsized to 32x32 pixels. So the true iteration count should be around 18 for the last image.

3.1.3. Minimal resolution. Our goal was to be able to properly colorize very high resolution images. Unfortunately, our model was not able to extrapolate the quality of smaller resolutions to the high resolution images.

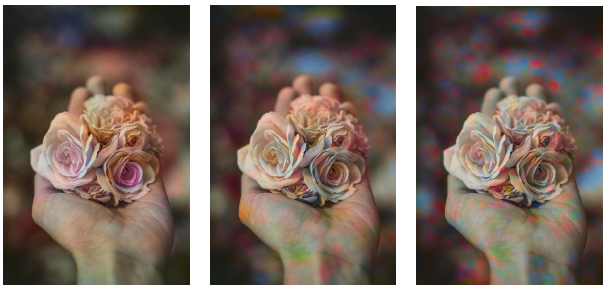
From the comparison in Figure 5 it is clearly visible on which image sizes the network was trained on. The



(a) 0 **(b)** 1 **(c)** 10 **(d)** 30

Figure 4: Results at different amounts of iterations.

(a) 187x125 **(b)** 375x250 **(c)** 750x500



(d) 1500x1000 **(e)** 3000x2000 **(f)** 6000x4000

Figure 5: Results at different resolutions.

image sizes below our "sweet spot" in Figure 5c are almost grayed out, whereas larger images progressively show more artifacts. Unfortunately, the network was not able to learn size invariance.

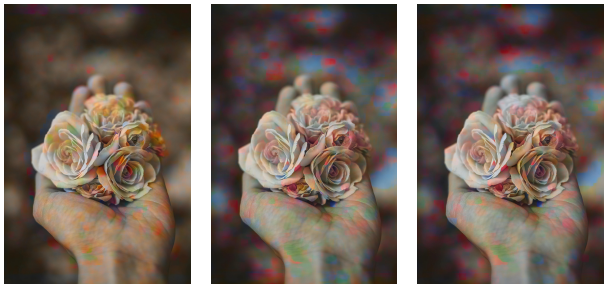
3.2. Comparison with Baseline

The most noticeable difference between the baseline and our model is the hue (see Figure 6). Our method tends to produce more blues and reds than the base model, which uses a more muted brownish color palette. Furthermore, our approach seems to produce more smoothed colors than the base model.

Also there is a difference in the artifacts generated by the two approaches. In comparison to the baseline model, the artifacts in our approach have a very box like shape. This might be due to the upsampling between iterations.

3.3. Quantitative evaluation

We evaluated the results of different models over different resolutions. The metrics captured in Table 1 are SSIM, PSNR and the time it took to colorize the image. What stands out about the SSIM and PSNR values is the fact, that the quality of the baseline stays fairly constant throughout the different resolutions but our approach



(a) Baseline **(b)** C2F 1.2 **(c)** C2F 2.0

Figure 6: Comparison between the baseline and C2F model with shrink sizes 1.2 and 2.0, respectively.

Resolution	**Base**	**C2F** (shrink size / #iterations)			
		1.2 / 1	1.2 / 10	2.0 / 1	2.0 / 10
750x500	0.771	0.823	0.810	0.835	0.830
1500x1000	0.755	0.766	0.754	0.727	0.736
3000x2000	0.729	0.647	0.672	0.663	0.681
6000x4000	0.760	0.565	0.578	0.541	0.559

(a) SSIM

Resolution	**Base**	**C2F** (shrink size / #iterations)			
		1.2 / 1	1.2 / 10	2.0 / 1	2.0 / 10
750x500	22.24	24.12	23.63	24.19	24.03
1500x1000	22.59	23.97	23.53	23.66	23.47
3000x2000	22.59	22.43	22.44	22.80	22.67
6000x4000	22.75	21.75	21.64	21.59	21.62

(b) PSNR

Resolution	**Base**	**C2F** (shrink size / #iterations)			
		1.2 / 1	1.2 / 10	2.0 / 1	2.0 / 10
750x500	1.6	3.0	5.8	2.4	2.6
1500x1000	6.0	11.4	21.0	8.8	9.4
3000x2000	23.8	43.2	79.1	34.4	35.9
6000x4000	115.1	203.3	339.7	160.5	166.5

(c) Time

Table 1: Comparison between baseline and C2F model with different hyperparamters.

seems to exhibit a noticeable drop of quality from low to high resolution images. One possible explanation to this is that the more saturated colors deviate more from the ground truth than the conservative brownish prediction of the baseline. For a more extensive comparison see Table 2.

4. Discussion

4.1. Quality

4.1.1. Color Palette. Looking at the results we can clearly see that our method tends to use a different color palette than the baseline model. The added feedback loop seems to have a saturating effect on the colors and does not stick to the more brownish colors like the baseline.

4.1.2. Color Consistency. The consistency of colors is a main concern especially on higher resolutions. In Figure 5 we can see quite well how the colors are the most consistent in Figure 5c and Figure 5d which correlates with the training resolution (640x640 pixels). At these resolutions

the colors look very smooth and compared to the baseline model (see Figure 6a) it produces more consistent colors. However at higher resolutions it does not work as well anymore and does not really improve upon the baseline in that regard.

While training the optimizer does a step after each batch has gone through all iterations. This means the model trains to create colors on exactly one scale. This leads to the predictions only really working on that one scale. A possible improvement to this could be to do an optimizer step after each iteration.

4.1.3. Artifacts. Our method seems to have a saturating effect on the colors. This is primarily a good thing. However, it also intensifies the artifacts and propagates them through the iterations. In comparison to the baseline these artifacts are more box shaped which might be caused by the upscaling between iterations. To reduce these artifacts one could add a refinement step between each iteration to remove artifacts and stop them from propagating through each iteration.

4.2. Performance

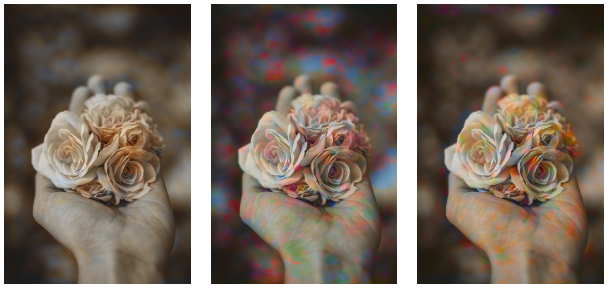
Performance wise our method requires more time to compute the image. The runtime is heavily dependent on the shrink size and max iterations (see Table 1c). This is due to the sequential nature of the iterative process. On the plus side, our model does not require any more memory than the baseline. This means any increase in iterations will just need more time but not require different hardware. So any improvements gained by our method do not come at an increased use of video memory.

5. Summary

The goal of this project was to improve the quality of colorization networks without the need for deeper networks which in turn would require much more powerful hardware.

We hoped our proposed method would help to give the model an understanding of the image on a course and fine level. This in turn would lead to more consistent colors because the model would be able to identify larger objects even when the resolution gets higher than the train images. In addition, we hoped the model would be able to extrapolate to higher resolutions than it was trained on. Unfortunately, this was not the case. And although there has been some improvements upon the baseline model on the consistency part, the high resolution images still have quite inconsistent colors.

Therefore, we thought of a few improvements that could help our approach. First of all, despite our efforts to keep the resource intensity of our model low it could help to create a deeper architecture in order to improve the colorization. Secondly, to counter the appearance of artifacts we propose joint bilateral filters that can handle the problem [21]. These filters would be adopted after each iteration. Furthermore, we currently have that the first iteration has its color input set to zeros. A more proper way of doing this could be to add another small network without feedback loop which generates the input for the first iteration of a second network with feedback loop. This way there is no lack of input in the first iteration.



(a) with **(b)** without (c2f) **(c)** without (baseline)

Figure 7: Introduction of intermediate supervision.

Intermediate Supervision: A promising method to improve the scale invariance of the network is to do optimizer steps after each iteration instead of only one step at the end. This way we force the network to learn colorization on all the different scales. We managed to adapt or code to implement this and have already started training the model. The model is not yet fully trained so the images are very desaturated but it looks like the artifacts are mostly gone and a lot less box like than compared to our method before and also when looking at the baseline (see Figure 7). Maybe with proper training this could yield very promising results.

References

[1] R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu, and A. A. Efros, "Real-time user-guided image colorization with learned deep priors," 2017.

[2] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," 2016.

[3] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," 2017.

[4] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification," *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, vol. 35, no. 4, pp. 110:1–110:11, 2016.

[5] K. Nazeri, E. Ng, and M. Ebrahimi, "Image colorization using generative adversarial networks," in *Articulated Motion and Deformable Objects*, F. J. Perales and J. Kittler, Eds. Cham: Springer International Publishing, 2018, pp. 85–94.

[6] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017.

[7] A. Royer, A. Kolesnikov, and C. H. Lampert, "Probabilistic image colorization," *arXiv preprint arXiv:1705.04258*, 2017.

[8] M. Kumar, D. Weissenborn, and N. Kalchbrenner, "Colorization transformer," *CoRR*, vol. abs/2102.04432, 2021. [Online]. Available: https://arxiv.org/abs/2102.04432

[9] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, "Axial attention in multidimensional

transformers," *CoRR*, vol. abs/1912.12180, 2019. [Online]. Available: http://arxiv.org/abs/1912.12180

[10] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[11] H. Shanshan, J. Xin, Q. Jiang, J. Li, S.-J. Lee, P. Wang, and S. Yao, "A fully-automatic image colorization scheme using improved cyclegan with skip connections," *Multimedia Tools and Applications*, vol. 80, pp. 1–28, 07 2021.

[12] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[13] M. Richart, J. Visca, and J. Baliosian, "Image colorization with neural networks," in *2017 Workshop of Computer Vision (WVC)*, 2017, pp. 55–60.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[15] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[17] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," 2016.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[19] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[20] Valeria Boltneva, "Flowers in hand," 2013, [Online; accessed January 13, 2022]. [Online]. Available: https://www.pexels.com/de-de/foto/selektiver-fokus-fotografie-der-person-die-rosa-und-lila-rosenblumen-halt-617965/

[21] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 664–672, 08 2004.

[22] Eftychia Syrimi, "House in the sea," 2013, [Online; accessed January 13, 2022]. [Online]. Available: https://www.pexels.com/de-de/foto/meer-schwarz-und-weiss-landschaft-natur-10736812/

6. Addendum

6.1. House on an Island

(a) shrink size: 2.0, max. iteration number: 1

(b) shrink size: 2.0, max. iteration number: 10

Figure 8: House on an Island; 480x640 pixels [Source: 22].

(a) shrink size: 1.2, max. iteration number: 1

(b) shrink size: 1.2, max. iteration number: 10

Figure 9: House on an Island, 3000x4000 pixels [Source: 22].

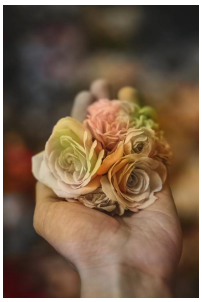
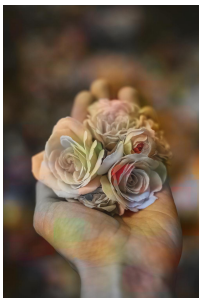
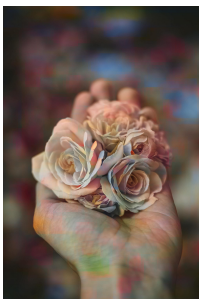
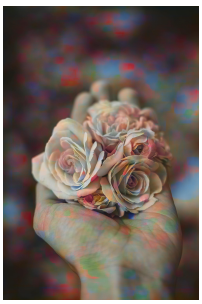
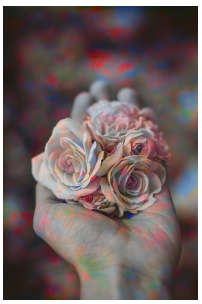
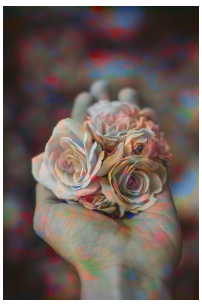
Resolution	**Base**	**C2F** (shrink size / #iterations)			
		1.2 / 1	1.2 / 10	2.0 / 1	2.0 / 10
750x500					
1500x1000					
3000x2000					
6000x4000					

Table 2: Comparison of different hyperparameters for our model and the baseline at different resolutions.