Extracting Vignetting and Grain Filter Effects from Photos

Abdelrahman Abdelhamed[1] Jonghwa Yim[2,3*] Abhijith Punnappurath[1]
Michael S. Brown[1] Jihwan Choe[2] Kihwan Kim[2]
[1]Samsung AI Center – Toronto [2]Samsung Electronics [3]NCSOFT
{a.abdelhamed, abhijith.p, michael.b1, jihwan.choe, kihwan23.kim}@samsung.com
jonhwayim@gmail.com

Abstract

Most smartphones support the use of real-time camera filters to impart visual effects to captured images. Currently, such filters come preinstalled on-device or need to be downloaded and installed before use (e.g., Instagram filters). Recent work [24] proposed a method to extract a camera filter directly from an example photo that has already had a filter applied. The work in [24] focused only on the color and tonal aspects of the underlying filter. In this paper, we introduce a method to extract two spatially varying effects commonly used by on-device camera filters—namely, image vignetting *and* image grain*. Specifically, we show how to extract the parameters for vignetting and image grain present in an example image and replicate these effects as an on-device filter. We use lightweight CNNs to estimate the filter parameters and employ efficient techniques—isotropic Gaussian filters and simplex noise—for regenerating the filters. Our design achieves a reasonable trade-off between efficiency and realism. We show that our method can extract vignetting and image grain filters from stylized photos and replicate the filters on captured images more faithfully, as compared to color and style transfer methods. Our method is significantly efficient and has been already deployed to millions of flagship smartphones.*

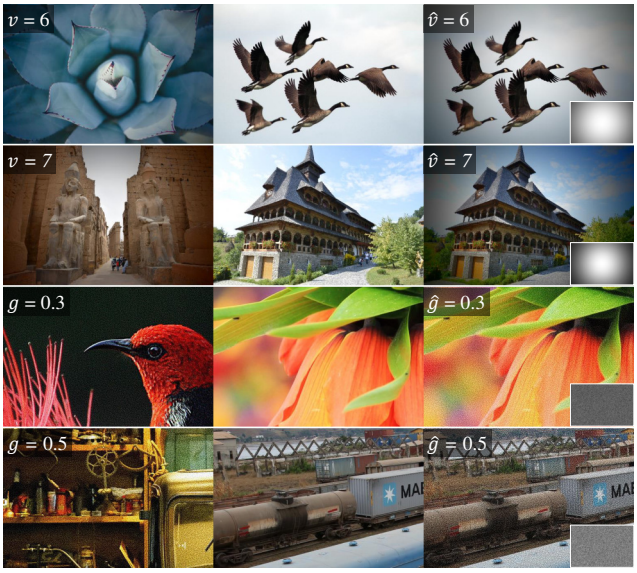


Figure 1. Four examples of extracting and transferring vignetting and image grain filters. Top two rows show transfer of vignetting filters with strength $v = 6, 7$. Bottom two rows show transfer of image grain filter with intensity $g = 0.3, 0.5$. Filter styles are estimated from filtered images (left) and then reapplied to content images (middle) to produce the filtered output images (right). The estimated filter parameter and the regenerated filter are shown on the output images. Images are better visualized while zooming in.

1. Introduction

With the increased usage of personal smartphones as cameras, most smartphones support camera filters that can be applied in real time to the camera's video stream. Such filters come preinstalled with the smartphone, or can be downloaded from a third party (e.g., Instagram, Snapseed, VSCO). The goal of this filter stylizing is to give the captured photos some desired visual effect. Most image stylizing filters modify the colors, tone, or contrast of the images. Common examples include changing the color temperature of the image or using color modifications to give the photo a nostalgic appearance of old film—for example, sepia. Recent work by Yim et al. [24] described a method to extract a color filter from an example photo that had been processed with an unknown filter. This provided an intuitive alternative to downloading a predefined filter. This idea is similar in nature to style transfer between images. Many methods address the problem of transferring styles between images. Color transfer methods (e.g., [17]) try to transfer the color distribution from one image to another such that the target image has a similar color distribution as the source image.

*This work was done while Jonghwa Yim was at Samsung Electronics.

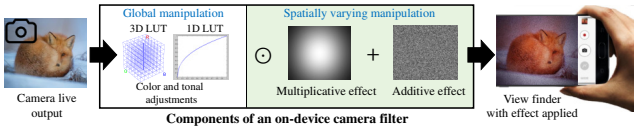


Figure 2. An illustration of an on-device camera filter integrated with the imaging pipeline. Our method is targeting the multiplicative (vignetting) and additive (image grain) effects only.

These types of methods basically focus on transferring color distributions only. Another line of work includes neural image style transfer methods (e.g., [5]) that aim at transferring general styles from one image to another. Such methods focus on transferring general styles, such as painting styles or weather styles, between images. A different approach to image stylizing is image-to-image translation (e.g., [6]). However, such approach is intended for transforming images between predefined image styles or modalities, such as translating sketches to real images, and not usually designed to handle different styles or different variations of the same style.

Unlike most existing methods, we focus on spatially varying filter effects—in particular, vignetting and image grain. These types of filters do not necessarily change the color distribution of the image; instead they change pixel intensities in specific ways. A vignetting filter decreases pixel intensities as we move away from the center of the image and approach the corners of the image. An image grain filter applies an effect similar to film grain on old film cameras, which consists of random optical textures of the processed photographic film. See Figure 1 for examples.

An important aspect of our method is that it can be easily integrated with the camera imaging pipeline, as shown in Figure 2, which illustrates the main components of an on-device camera filter. Most previous methods focus on the color manipulation part, while our method focuses on the spatially varying multiplicative or additive filter effects, specifically vignetting and image grain.

Contribution We propose a method for extracting two common spatially varying filter effects from an image—vignetting and image grain. Our method works by detecting the existing filters in a user-supplied stylized image, estimating the filter parameters, and then constructing a new on-device filter. The extracted filter can then be selected by the user and applied to any input image or video feed in real time. Our method is efficient, requiring only a small amount of processing resources, which made it suitable for integration and deployment to millions of flagship smartphones.

2. Related Work

There have been many approaches to image style transfer, with some methods being specific to color filter transfer. However, there are few methods targeting filter style extraction, especially for spatially varying types of filters.

Color transfer A pioneering work in color transfer between images is [17], where the input image's color distribution is matched to a stylized one by applying simple scaling and shifting operations. The simplicity of this method makes it limited to simple color variations only. Other color transfer methods [16, 20, 21, 23] attempt to match colors based on clusters or objects; however, these methods are still limited to similar color clusters or objects in both images.

Neural style transfer With deep neural networks, many methods were proposed for image-to-image style transfer. Style transfer neural networks [5] enabled the transfer of colors, textures, and even shapes between images. Such methods usually target generic styles, such as painting styles, and are not specifically designed for transferring specific image filters, such as a vignetting filter.

Photo-realistic style transfer Instead of learning styles from distributions of images, follow-up approaches to neural style transfer focused on transferring photorealistic styles from a single style image [10, 19, 12, 25]. Such methods require large datasets for training, leading to a high computational cost. Even with the high processing time, they usually produce distorted or undesirable transfers of colors and textures.

Image-to-image translation Recent methods [6, 11, 28] addressed the problem of image-to-image translation or image domain transfer—that is, transferring image styles between different image domains, such as sketch to photo or aerial image to map image. However, most domain transfer approaches try to map between distributions of images, which makes their output images not explicitly reflect the style of a single image.

Filter style transfer Recently, a method for filter style transfer [24] proposed to estimate a filter from a single image and reapply the estimated filter to the input image. While this method is well suited to use on-device, it is focused only on color filters (white part in Figure 2) and does not address spatially varying filters (green part in Figure 2). Addressing such lack, we propose a new method that can handle the extraction of spatially varying filters—particularly, vignetting and grain—from photos.

3. Vignetting and Grain Filter Extraction

The goal of our method is to estimate the filter parameters from a filtered input or stylized image $\mathbf{y}$, use the estimated filter parameter $\hat{p}$ to regenerate the filter, and apply it to an input content image $\mathbf{x}$. We formulate this as follows:

$$\hat{p} = f\left(\mathbf{y}\right), \tag{1}$$

$$\tilde{\mathbf{x}} = h\left(\mathbf{x}, \hat{p}\right), \tag{2}$$

where $f(\cdot)$ is a function to estimate the filter parameter and $h(\cdot)$ is a function that regenerates the filter and applies it to the input content image $\mathbf{x}$ to produce the filtered image $\tilde{\mathbf{x}}$.

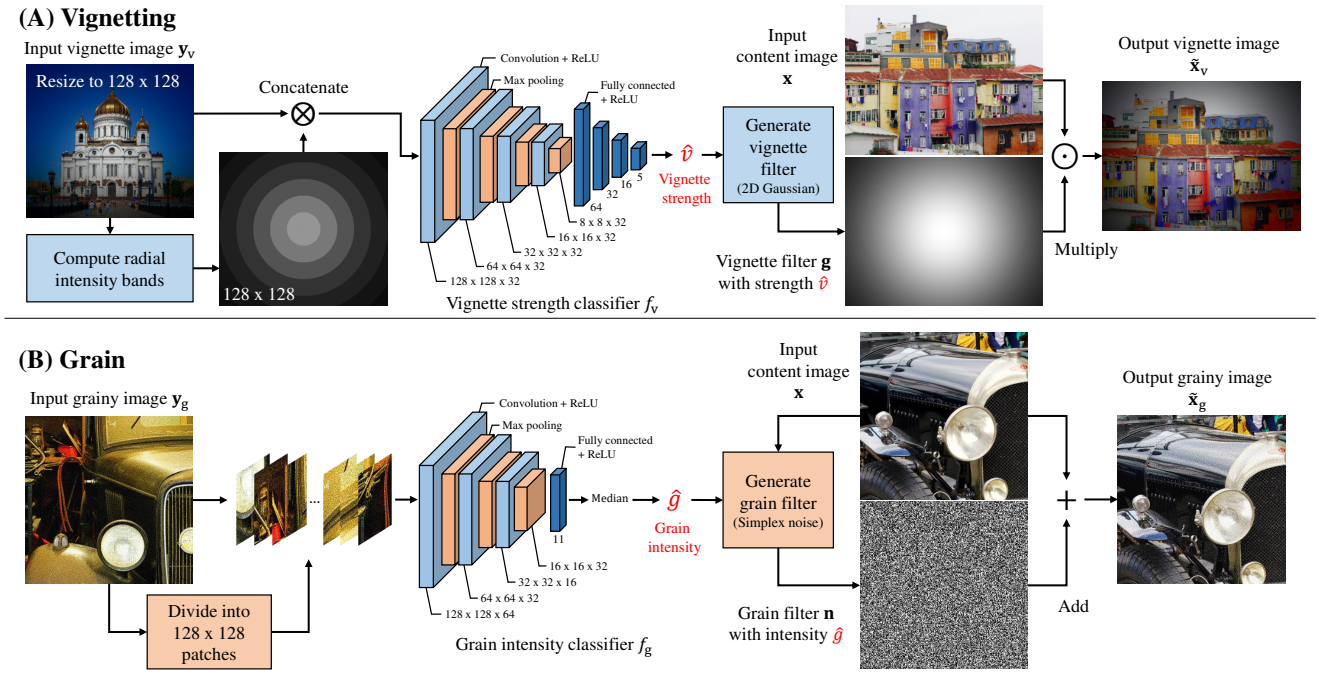


Figure 3. Our method for (A) *vignetting* and (B) *grain* filter extraction and transfer to other photos. For both filters, we use lightweight CNNs to classify vignetting filter strength $\hat{v}$ and grain filter intensity $\hat{g}$. Then, we regenerate the filters and apply them to the input content image. Images are best visualized while zooming in.

For efficiency, we treat the filter parameter estimation as a classification problem. We choose a set of parameters for each filter type and train a lightweight convolutional neural network (CNN) to predict the closest parameter value. With that formulation, we address two common types of spatial image filter styles: (1) *vignetting* and (2) *grain*.

3.1. Vignetting Filter Extraction

Vignetting model Image vignetting filters decrease pixel intensities as they move away from the center or approach the corners of the image. There are different models to simulate image vignetting (e.g., [26, 27, 14]). However, for efficiency, we adopt an isotropic 2D Gaussian filter to simulate vignetting. See Figure 4 for examples.

Assuming the input content image is $\mathbf{x} \in \mathbb{R}^{m \times n}$, with height m and width n, and the vignetting Gaussian filter is $\mathbf{g} \in \mathbb{R}^{m \times n}$ with values in the range $[0, 1]$, the output vignetted image $\tilde{\mathbf{x}}_v \in \mathbb{R}^{m \times n}$ would be generated as

$$\tilde{\mathbf{x}}_v = \mathbf{g} \odot \mathbf{x} \,, \tag{3}$$

$$\mathbf{g}_{i,j} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right), \tag{4}$$

where $\odot$ denotes element-wise multiplication and i and j are pixel coordinates where the point of origin $(0, 0)$ is at the center of the image.

Vignetting strength prediction To control *vignetting strength* (i.e., how much vignetting is in the image), we re-parameterize σ to define a discrete set of vignetting strengths $v \in \{0, \ldots, 8\}$ as

$$\sigma = (1 - 0.1\, v)\ z \,, \tag{5}$$

$$z = \max(m, n) \,. \tag{6}$$

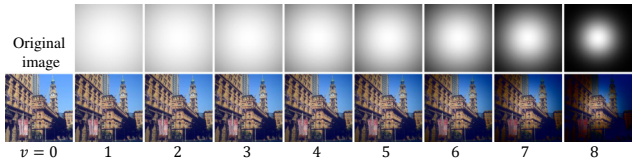


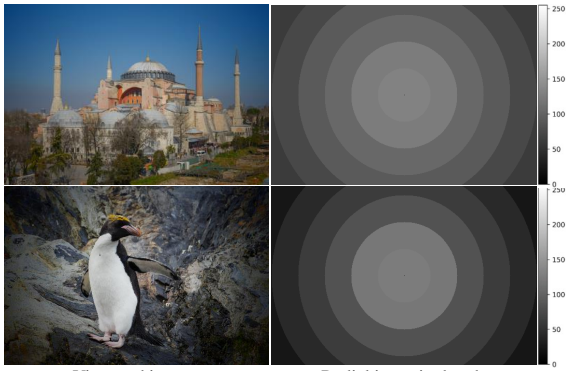
Figure 4. An example of synthesizing vignetted images with different vignetting strengths using Gaussian filters.

Figure 4 shows an example image and the result of applying the vignetting filter to the image with strengths 1–8. An image without vignetting has $v = 0$.

We use a lightweight CNN $f_v(\cdot)$ to predict the vignetting strength $\hat{v}$ as follows:

$$\hat{v} = f_v(\mathbf{y}_v) \,, \tag{7}$$

where $\mathbf{y}_v$ is the input vignetted image. Since vignetting strengths below 5 are not visually strong, we train the network to predict strengths $v \in \{0, 5, 6, 7, 8\}$. The vignetting classifier $f_v(\cdot)$ is shown in Figure 3A. It contains four convolutional layers, and each layer contains 32 filters with 3×3 kernels, followed by three fully connected layers with



Vignetted images Radial intensity bands

Figure 5. Examples of (left) vignetted images and (right) their corresponding radial intensity bands computed using six bands.

64, 32, and 16 units. Each of the mentioned layers is followed by a rectified linear unit (ReLU) [13]. The last layer is fully connected with five units to predict the vignetting strength. Our vignetting classifier network consists of approximately 162 K parameters.

Radial intensity bands Using images only as input to the network is not sufficient to achieve high prediction accuracy of vignetting strength, especially when using a small network for the sake of efficiency. To boost the accuracy, we extract additional features from the images to help with the vignetting strength prediction. We compute the average pixel intensity in a set of ring-shaped areas around the image center. We replicate these average intensities in what we call the *radial intensity bands*, which have the same size as the vignetted image. The radial intensity bands are then concatenated to the vignetted image as input to the vignetting classification network. Figure 5 shows two examples of radial intensity bands. Once the vignetting strength $\hat{v}$ is estimated, a vignetting filter is generated using Equation 4, and then applied to the input content image $\mathbf{x}$ using Equation 3 to get the output vignetted image $\tilde{\mathbf{x}}_{\mathrm{v}}$. The whole process is depicted in Figure 3A.

3.2. Grain Filter Extraction

Film grain consists of random optical textures of processed photographic film due to the presence of small particles of metallic silver, or dye clouds, developed from silver halide interacting with light photons [7, 18, 3]. Film grain is more common in images captured with old film cameras. *Image grain* filters try to mimic the film grain effect on digital images. Camera apps and software programs, such as Photoshop, typically use random noise to emulate grain effects on digital images. See Figure 6 for examples.

Grain simulation There is a lack of datasets of paired grainy and non-grainy images. To alleviate the burden of collecting such a dataset, we generated synthetic grainy images from non-grainy ones. There are a number of models

$g = 0$ 0.2 0.4 0.6 0.8 1.0

Figure 6. An example of synthesizing grainy images with different grain intensities using simplex noise.

to simulate image grain (e.g., [15, 14]). We adopt the OpenSimplex noise model [22] to simulate the image grain effect with different intensities.

Grain intensity Assuming the input image is $\mathbf{x} \in \mathbb{R}^{m \times n}$ and the grain layer, generated as Simplex noise with standard deviation of 1, is $\mathbf{n} \in \mathbb{R}^{m \times n}$, the output grainy image $\tilde{\mathbf{x}}_{\mathrm{g}} \in \mathbb{R}^{m \times n}$ would be generated as

$$\tilde{\mathbf{x}}_{\mathrm{g}} = \mathrm{clip}\left(\mathbf{x} + g\,\mathbf{n}\right), \tag{8}$$

where $\mathrm{clip}(\cdot)$ indicates clipping the image values within the underlying intensity range (e.g., $[0, 255]$ or $[0.0, 1.0]$) and g represents *grain intensity*. Grain intensity indicates the magnitude of grain values added to the image. Figure 6 shows an example image and a number of synthesized grainy images using simplex noise with grain intensities in the range $[0, 1]$.

Grain intensity classification We used a lightweight CNN as our grain intensity classifier, as shown in Figure 3. To detect fine-scale grain intensities, we chose to train the network on 11 grain intensities in the range $[0, 0.5]$ with 0.05 steps. Images where predicted class is 0 are considered as not having any grain. The grain intensity classifier $f_{\mathrm{g}}(\cdot)$ is shown in Figure 3B. It contains three convolutional layers, with 64, 32, and 16 filters, with 3×3 kernels. Each of the mentioned layers is followed by a rectified linear unit (ReLU) [13]. The last layer is fully connected with 11 units to predict grain intensity. Our grain classifier network consists of approximately 59 K parameters. Once the grain intensity $\hat{g}$ is estimated, a grain filter is generated using OpenSimplex noise and then applied to the input content image $\mathbf{x}$ using Equation 8 to get the output grainy image $\tilde{\mathbf{x}}_{\mathrm{g}}$. The whole process is depicted in Figure 3B.

4. Results

We evaluate our method for vignetting and grain filter extraction on synthetically generated images and evaluate the ability to faithfully replicate such filters on other images. Then, we compare our method to existing methods, and evaluate our method on real-world images from the web.

4.1. Dataset and Setup

We used the DIV2K [2] dataset for training, validation, and testing. We used 700 images for training, 100 images for validation, and 100 images for testing. For training

Table 1. Vignetting strength prediction results.

Metric	Vignetting strength 0	5	6	7	8	Mean
Precision	0.97	0.82	0.83	0.96	1.0	0.92
Recall	0.98	0.85	0.80	0.95	1.0	0.92
F1 score	0.98	0.83	0.82	0.95	1.0	0.92

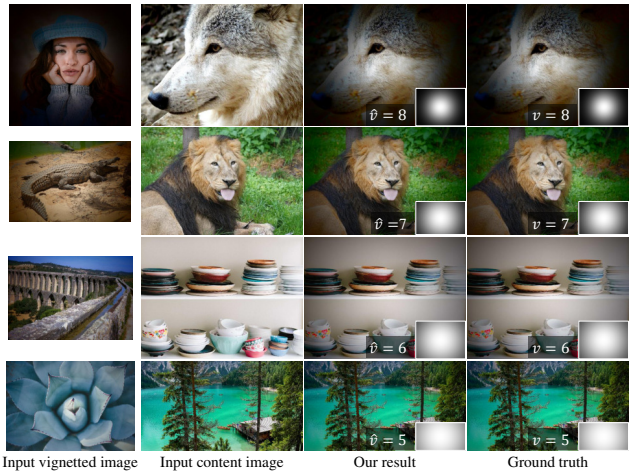


Figure 7. Examples of vignetting filter extraction and transfer using our method. Estimated and ground-truth vignetting strengths along with extracted vignetting filters are shown in the lower right corner of respective images.

the vignetting strength classifier, we generate synthetic vignetted images with vignetting strengths $v \in \{0, 5, 6, 7, 8\}$ using Equation 3. For training the grain intensity classifier, we generate synthetic grainy images with grain intensities $g \in [0, 0.5]$ with 0.05 steps, using Equation 8. We implement the networks in TensorFlow's [1] Keras [4] framework. We use a cross-entropy loss function and a learning rate of 10^{-4}. We train all classifiers for 2000 epochs.

4.2. Filter Extraction and Transfer

Vignetting extraction and transfer Our vignetting strength classifier achieves 92% accuracy. The precision, recall, and F1 scores are shown in Table 1. The performance of classifying vignetting strengths 5 and 6 is noticeably lower than other strengths because the visual difference between these two strengths is hardly noticeable. We have noticed this issue with vignetting strengths lower than 5 as well; that is why we omitted them from our training setup. Figure 7 shows examples of vignetting filter extraction and transfer using our method, along with estimated vignetting strengths and extracted vignetting filters.

Effect of radial bands Table 2 shows the effect of using different numbers of radial intensity bands on the testing accuracy of the vignetting strength classifier. Using four radial intensity bands yields the best accuracy, while not using any radial intensity bands yields low accuracy.

Table 2. Effect of using different numbers of radial intensity bands on testing accuracy of vignetting strength classifier.

Number of radial bands	0	2	4	6	8
Accuracy	0.54	0.74	**0.92**	0.65	0.68

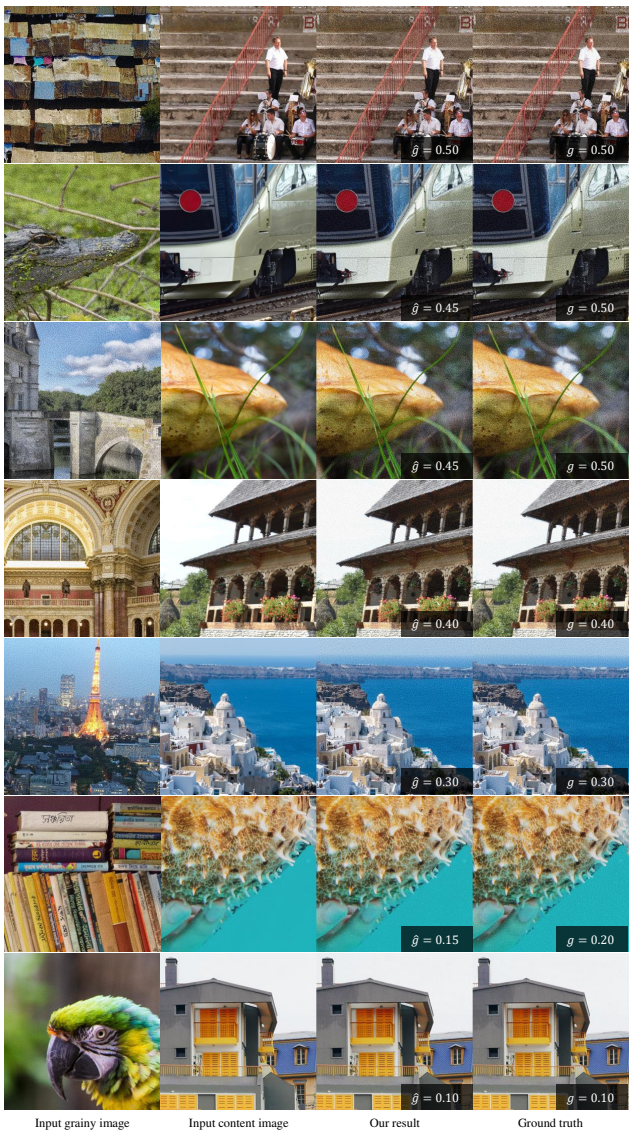


Figure 8. Examples of grain filter extraction and transfer using our method. Estimated and ground-truth grain intensities are shown in the lower right corner of respective images. Grain details are better visualized while zooming in.

Grain extraction and transfer Our grain intensity classifier achieves 95% accuracy. The precision, recall, and F1 scores are shown in Table 3. Figure 8 shows examples of grain filter extraction and transfer using our method, along with estimated grain intensities. Our method performs well in terms of predicting grain intensity; however, in few cases, the estimated grain intensity is off by ± 0.05, which is hardly noticeable.

Table 3. Grain intensity prediction results.

	Grain intensity											
Metric	0.0	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50	Mean
Precision	0.97	0.88	0.92	0.94	0.97	0.96	0.96	0.96	0.97	0.91	1.00	0.95
Recall	0.86	0.92	0.95	0.97	0.95	0.96	0.97	0.98	0.98	0.97	0.91	0.95
F1 score	0.91	0.90	0.94	0.95	0.96	0.96	0.97	0.97	0.97	0.94	0.95	0.95

4.3. Comparison to Style Transfer

In Figure 9, we compare our method against the conventional color transfer algorithm of Reinhard et al. [17], the recent filter style transfer (FST) approach of [24], and various deep learning-based style transfer methods, such as WCT [9], including photo-realistic methods, such as LinST [8], PhotoWCT [10], and WCT2 [25]. The first two examples show vignetting transfer while the last two examples are for image grain. The color transfer approach of [17] applies global color transformation and cannot model spatial variations. The FST method [24] tends to carefully manipulate the image colors while not handling spatial effects. Existing deep learning-based style transfer methods [8, 9, 10, 25] are also not well suited to transfer spatial effects, such as vignette and grain. It can be observed that they distort the colors (and sometimes even the structure) of the content image based on the filtered image, and fail to achieve the intended effect. In contrast, since our method is focusing on spatial filters, it faithfully transfers the spatially varying effects of vignetting and grain.

Runtime Table 4 shows a comparison of running time (in milliseconds) of filter style transfer per 1-megapixel image. All running times were computed on an Nvidia Tesla V100 GPU with 32 GB of RAM, except for the FST method [24], which was run on an Nvidia GTX 1080 Ti GPU, and Reinhard et al. [17], which was run on an Intel Core i7-8550U CPU. Our method is significantly faster than other methods due to two main factors. First, we use much lighter neural networks to estimate the filter parameters. Second, we use faster methods to regenerate the filter style effects—that is, we use an optimized implementation of simplex noise to generate the grain effect and use 2D isotropic Gaussian filters to generate the vignetting effect. In addition, our method currently runs on millions of smartphones within 1 second for initial filter extraction and within 2 milliseconds for filter application.

4.4. Integration with Filter Pipelines

On camera devices, it is often required to extract a filter from an image, pre-generate the filter effect, and then apply the filter effect on the captured images in real time [24]. Our method is perfectly suitable to be integrated in such image filtering pipelines. Figure 10 shows how well our vignetting and grain filter extraction methods can be integrated with color transfer methods, such as Reinhard et al. [17]. In Figure 10, the third and fourth columns show the result of transferring the vignetting effect only and the vignetting and grain effects together, respectively. The last column shows the result of transferring the three filter effects—that is, vignetting, grain, and colors. In combination with a simple and efficient method, such as Reinhard et al. [17], our method can faithfully transfer both global color and spatially varying effects (e.g., vignetting and grain).

Table 4. Comparison of running time of filter style transfer in milliseconds per 1-megapixel (1024×1024 pixels) image. Our method is significantly faster than other methods.

Method	Time (ms)	Device
Reinhard et al. [17]	254	Core i7-8550U
FST [24]	218	GTX 1080 Ti
LinST [8]	1021	Tesla V100
WCT [9]	4167	Tesla V100
PhotoWCT [10]	2532	Tesla V100
WCT2 [25]	5120	Tesla V100
Ours (vignetting)	7	Tesla V100
Ours (grain)	77	Tesla V100

Results on web images To further evaluate our method, we apply it on images downloaded from Flickr. Figure 11 shows some results along with the vignetting and grain parameters estimated from the style images. The estimated parameters reflect the amount of vignetting or grain found in the style images. In most cases, the estimated parameters visually match the actual effects in the style images.

5. Conclusion

In this paper, we proposed an efficient method for extracting spatially varying filters—namely, vignetting and image grain—from stylized images. Our method can detect whether these filters exist in the stylized image, estimate the filter parameters, and construct a new filter that can be applied to any input image or video. To estimate filter parameters (vignetting strength and grain intensity), we use lightweight neural networks that contain as few as 162K and 59K parameters. In addition, we adopt simple, yet effective, methods for regenerating the filters, advocating for a reasonable trade-off between efficiency and realism. Specifically, we used Gaussian filters and simplex noise to regenerate the vignetting and grain effects, respectively. Our small networks and efficient filter generation are computationally efficient and has been deployed to millions of flagship smartphones. We feel this is a great opportunity for other researchers to see deployed industry solutions. Code and data will be published.

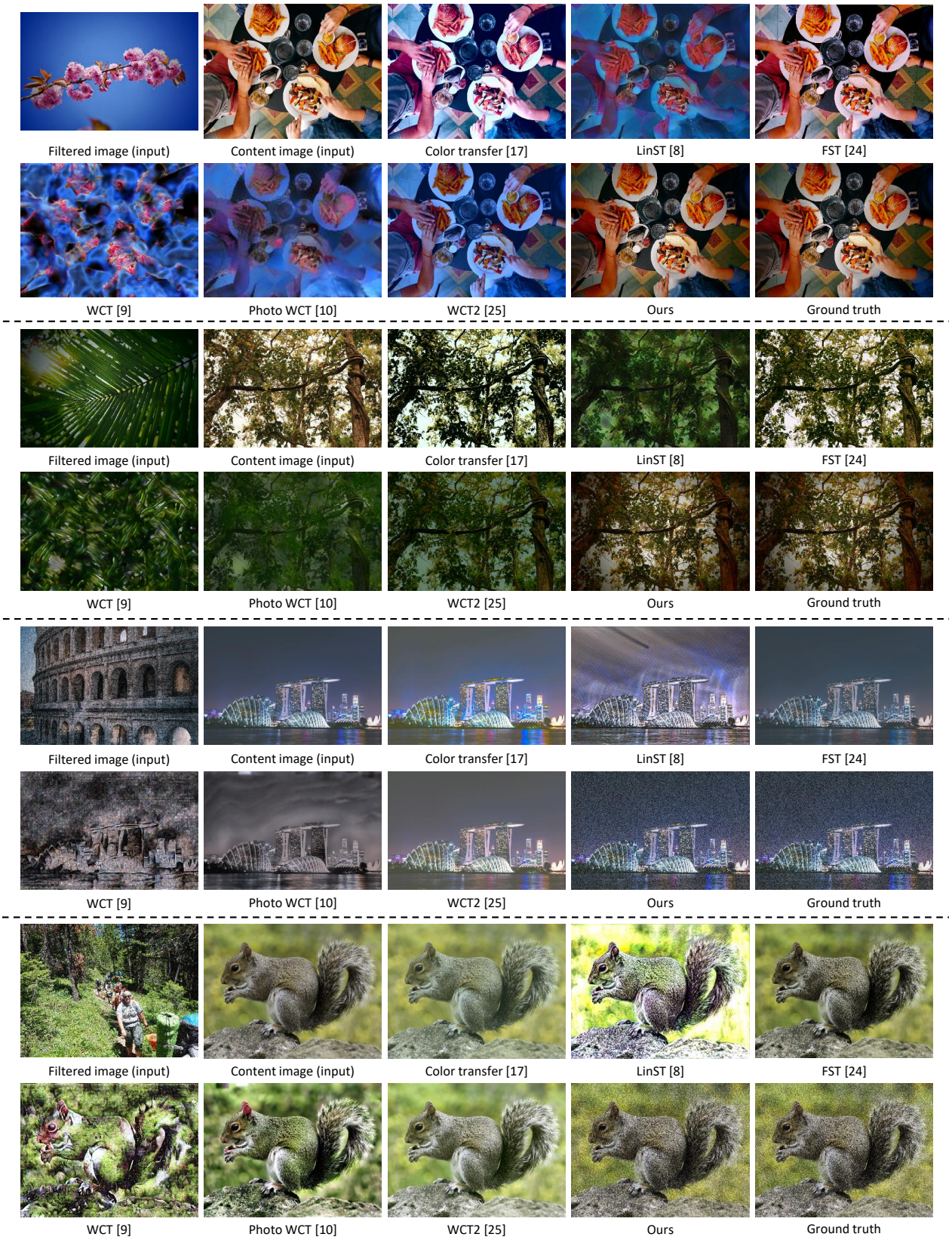
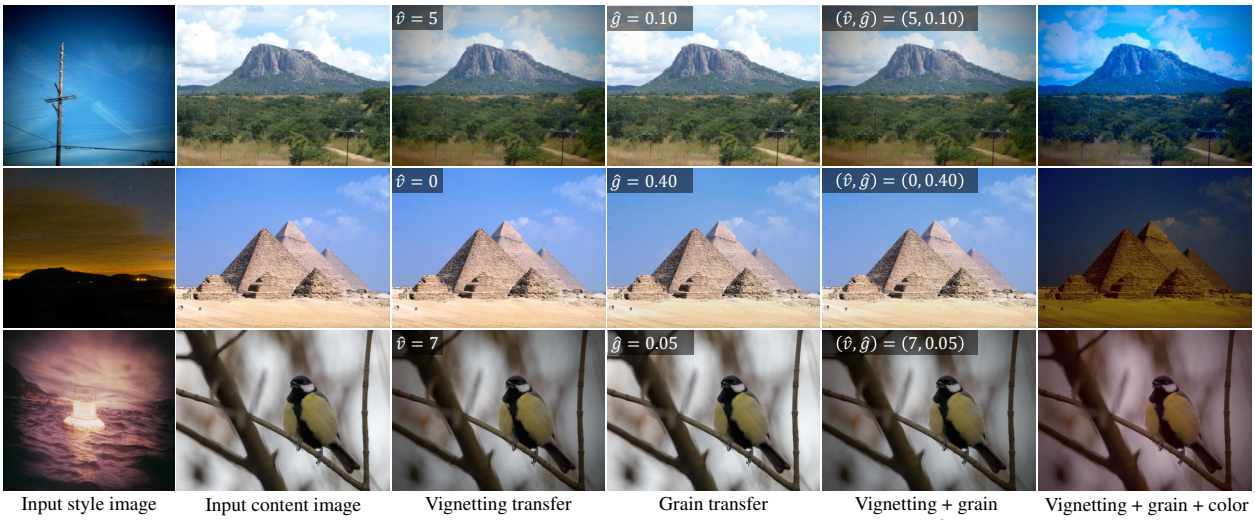


Figure 9. Comparisons with the conventional color transfer algorithm of Reinhard et al. [17], the recent filter style transfer approach of [24], and various deep learning-based style transfer methods, such as LinST [8], WCT [9], PhotoWCT [10], and WCT2 [25].

Input style image (vignette + grain) | Input content image | Output image (vignette) | Output image (vignette + grain) | Output image (vignette + grain + color)

Figure 10. Examples of extraction and transfer of vignetting and grain filters from images in our testing set. We combine our method combined with Reinhard et al. [17] for color transfer (last column). Grain details are better visualized while zooming in.



Input style image | Input content image | Vignetting transfer | Grain transfer | Vignetting + grain transfer | Vignetting + grain + color transfer

Figure 11. Examples of extraction and transfer of combined vignetting and grain filters from web images from Flickr. We combine our method with Reinhard et al. [17] for color transfer. Grain details are better visualized when zoomed in.

References

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *CVPRW*, 2017.

[3] Hans I Bjelkhagen. Silver-halide materials. In *Silver-halide recording materials*, pages 13–92. Springer, 1995.

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

[5] Leon Gatys, Alexander Ecker, and Matthias Bethge. A neural algorithm of artistic style. *Journal of Vision*, 16(12):326–326, 2016.

[6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[7] Brian Keelan. *Handbook of image quality: Characterization and prediction*. CRC Press, 2002.

[8] Xueting Li, Sifei Liu, Jan Kautz, and Ming-Hsuan Yang. Learning linear transformations for fast arbitrary style transfer. In *CVPR*, 2019.

[9] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *NeurIPS*, 2017.

[10] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018.

[11] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *arXiv preprint arXiv:1703.00848*, 2017.

[12] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *CVPR*, 2017.

[13] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[14] Alasdair Newson, Noura Faraj, Bruno Galerne, and Julie Delon. Realistic film grain rendering. *Image Processing On Line*, 7:165–183, 2017.

[15] Alasdair Newson, Bruno Galerne, and Julie Delon. Stochastic modeling and resolution-free rendering of film grain. 2016.

[16] François Pitié, Anil C Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. *CVIU*, 107(1-2):123–137, 2007.

[17] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, 21(5):34–41, 2001.

[18] Nanette Salvaggio. *Basic photographic materials and processes*. Taylor & Francis, 2009.

[19] Lu Sheng, Ziyi Lin, Jing Shao, and Xiaogang Wang. Avatar-net: Multi-scale zero-shot style transfer by feature decoration. In *CVPR*, 2018.

[20] Yu-Wing Tai, Jiaya Jia, and Chi-Keung Tang. Local color transfer via probabilistic segmentation by expectation-maximization. In *CVPR*, 2005.

[21] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. In *Proceedings of the 29th annual conference on computer graphics and interactive techniques*, pages 277–280, 2002.

[22] Wikipedia. *OpenSimplex noise*, 2020 (accessed October 27, 2020).

[23] Xuezhong Xiao and Lizhuang Ma. Color transfer in correlated color space. In *Proceedings of the 2006 ACM international conference on virtual reality continuum and its applications*, pages 305–309, 2006.

[24] Jonghwa Yim, Jisung Yoo, Won-joon Do, Beomsu Kim, and Jihwan Choe. Filter style transfer between photos. In *ECCV*, 2020.

[25] Jaejun Yoo, Youngjung Uh, Sanghyuk Chun, Byeongkyu Kang, and Jung-Woo Ha. Photorealistic style transfer via wavelet transforms. In *ICCV*, 2019.

[26] Yuanjie Zheng, Stephen Lin, Chandra Kambhamettu, Jingyi Yu, and Sing Bing Kang. Single-image vignetting correction. *TPAMI*, 31(12):2243–2256, 2008.

[27] Yuanjie Zheng, Stephen Lin, Sing Bing Kang, Rui Xiao, James C Gee, and Chandra Kambhamettu. Single-image vignetting correction from gradient distribution symmetries. *TPAMI*, 35(6):1480–1494, 2012.

[28] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. *arXiv preprint arXiv:1711.11586*, 2017.