Supplementary Materials for: "Content Authentication for Neural Imaging Pipelines: End-to-end Optimization of Photo Provenance in Complex Distribution Channels"

Paweł Korus[1,2] and Nasir Memon[1]
New York University[1], AGH University of Science and Technology[2]
http://kt.agh.edu.pl/~korus

1. Source Code

To facilitate further research in this direction, and enable reproduction of our results, our neural imaging toolbox is available at https://github.com/pkorus/neural-imaging.

2. Implementation Details

We implemented the entire acquisition and distribution pipeline in Python 3 and Tensorflow v. 1.11.0. In all experiments, we used the Adam optimizer with default settings. We proceeded in the following stages:

1. Training and validation of the NIPs.
2. Validation of the FAN for manipulation detection with no distribution channel between post-processing and forensic analysis.
3. Joint optimization of the FAN & NIP models with active distribution channel.

The NIP models trained in stage (1) were then used in (2) and (3) for NIP initialization to speed-up training.

2.1. NIP Architectures

We considered several NIP models with various levels of complexity. The simplest *INet* model was handcrafted to replicate the standard imaging pipeline. The architecture (Tab. 2) is a simple sequential concatenation of convolutional layers. Each operation was initialized with meaningful values (see Tab. 2) which significantly sped up convergence. (We also experimented with random initialization, but this led to slower training and occasional problems with color fidelity.) The last two layers were initialized with parameters from a simple 2-layer network trained to approximate gamma correction for a scalar input (4 hidden nodes + one output node).

The *UNet* model (Tab. 3) was the most complex of the considered models, but it delivered fairly quick convergence due to skip connections [10]. We adapted the implementation from a recent work by Chen et al. [4]. The configuration of skip connections is shown in detail in Tab. 3. All convolutional layers produce tensors of the same spatial dimensions, eliminating the need for cropping.

We also experimented with two recent networks used for joint demosaicing and denoising (*DNet* - Tab. 4) [7], and for general-purpose image processing (*CNet*) [5]. Overall, the results were unsuccessful. The *DNet* model was able to learn a high-fidelity photo development process, but converged very slowly due to colorization and sharpness artifacts. The trained model proved to be rather fragile, and quickly deteriorated during joint NIP and FAN optimization[1]. We were unable to obtain satisfactory photo development using the *CNet* model - the validation PSNR stopped improving around 25-27 dB.

All models were trained to replicate the output of a manually implemented imaging pipeline. (Detailed, per-camera validation performance measurements are shown in Tab. 1.) We used `rawkit` [3] wrappers over `libRAW` [8]. Demosaicing was performed using an adaptive algorithm by Menon et al. [9] from the `colour-demosaicing` package. We used learning rate of 10^{-4} and continued training until the average validation loss for the last 5 epochs changes by more than 10^{-4}. The maximal number of epochs was set to 50,000. Due to patch-based training, we did not perform any

[1]In more recent experiments with explicit regularization of the impact of the NIP's L_2 loss, we were able to improve *DNet*'s performance. However, the model still remains more fragile and reaching classification accuracy comparable to *UNet* leads to excessive artifacts. In addition to noise-like artifacts, the model looses edge sharpness. With a visually acceptable distortion, the model yielded accuracy between 70-80%. More detailed results will be available in an extended version of this work.

Table 1. Detailed validation performance statistics for all cameras and all NIPs.

Camera	INet		UNet		DNet	
	PSNR[1]	SSIM	PSNR[1]	SSIM	PSNR[1]	SSIM
Canon EOS 40D	42.7	0.987	43.6	0.990	44.5	0.992
Canon EOS 5D	42.4	0.987	44.8	0.992	48.4	0.997
Nikon D5100	43.7	0.989	45.3	0.990	48.1	0.996
Nikon D700	44.7	0.993	45.6	0.994	47.2	0.997
Nikon D7000	42.3	0.989	44.4	0.992	44.9	0.994
Nikon D750	42.7	0.990	44.8	0.994	45.5	0.996
Nikon D810	39.6	0.984	41.9	0.991	43.6	0.995
Nikon D90	44.6	0.993	44.4	0.991	47.7	0.997

[1] PSNR values in [dB]

global post-processing (e.g., histogram stretching). In each epoch, we randomly selected patches from full-resolution images, and fed them to the network in batches of 20 examples. We used input examples of size $64 \times 64 \times 4$ (RGGB) which are developed into $128 \times 128 \times 3$ (RGB) color patches. The final networks can work on arbitrary inputs without any changes. Fig. 1 shows an example full-resolution (12 Mpx) image developed with the standard (ab) and the neural pipelines (cde).

2.2. JPEG Codec Approximation

The architecture of the dJPEG model is shown in Tab. 5. The network was hand-crafted to replicate the operation of the standard JPEG codec with no chrominance sub-sampling (the 4:4:4 mode). We used standard quantization matrices from the IJG codec [1]. See the main body of the paper for approximation details. The input to the network is an image batch, and should be normalized to [0, 1].

2.3. The FAN Architecture

The FAN architecture (Tab. 6) is a fairly standard CNN model with an additional constrained convolution layer recommended for forensics applications [2]. In contrast to the study by Bayar and Stamm, who used only the green color channel, we take all color channels (RGB). We also used larger patches for better statistics - in all experiments, the input size is 128×128 px.

The network starts with a convolution layer constrained to learn a residual filter. We initialized the layer with the following residual filter (padded with zeros to 5×5):

$$\begin{bmatrix} -1, -2, -1 \\ -2, 12, -2 \\ -1, -2, -1 \end{bmatrix}. \quad (1)$$

We used leaky ReLUs instead of tanh for layer activation, and dispensed with batch normalization due to small network size and fast convergence without it.

2.4. Training Details

In our implementation, we use two Adam optimizers (with default settings) for: (1) updating the FAN (and in joint training also the NIP) based on the cross-entropy loss; (2) updating the NIP based on the image fidelity loss (L_2). The optimization steps are run in that order. To ensure comparable loss values, the L_2 loss was computed based on images normalized to the standard range [0,255]. Analogously to standalone NIP training, we feed raw image patches extracted from full-resolution images. In each epoch, the patches are chosen randomly, and fed in batches of 20. We start with learning rate of 10^{-4} and decrease it by 15% every 50 epochs.

References

[1] Independent JPEG Group. http://www.ijg.org/. visited 18 Apr 2017. ii

[2] B. Bayar and M. Stamm. Constrained convolutional neural networks: A new approach towards general purpose image manipulation detection. *IEEE Tran. Information Forensics and Security*, 13(11):2691–2706, 2018. ii

[3] P. Cameron and S. Whited. Rawkit. https://rawkit.readthedocs.io/en/latest/. Visited Nov 2018. i

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

[5] Q. Chen, J. Xu, and V. Koltun. Fast image processing with fully-convolutional networks. *arXiv*, 2017. arXiv:1709.00643v1. i

[6] D. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato. RAISE - a raw images dataset for digital image forensics. In *Proc. of ACM Multimedia Systems*, 2015. iii

[7] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand. Deep joint demosaicking and denoising. *ACM Tran. Graphics*, 35(6):1–12, nov 2016. i

[8] LibRaw LLC. libRAW. https://www.libraw.org/. Visited Nov 2018. i

[9] D. Menon, S. Andriani, and G. Calvagno. Demosaicing with directional filtering and a posteriori decision. *IEEE Tran. Image Processing*, 16(1):132–141, 2007. i

[10] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Int. Conf. Medical Image Computing and Computer-assisted Intervention*. Springer, 2015. i

(a) our implementation of the standard pipeline

(b) libRAW with default settings

(c) developed by the INet model

(d) developed by the UNet model

(e) developed by the DNet model

Figure 1. An example full-resolution (12.3 Mpx) image developed with standard pipelines (ab) and the considered NIPs (cde): image *r23beab04t* from the Nikon D90 camera (Raise dataset [6]). The full-size images are included as JPEGs (quality 85, 4:2:0) to limit PDF size. Close-up patches are included as uncompressed PNG files.

Table 2. The INet architecture: 321 trainable parameters

Operation	**Activation**	**Initialization**	**Function**	**Output size**
Input	-	-	RGGB feature maps	$N \times \frac{h}{2} \times \frac{w}{2} \times 4$
1×1 convolution	-	hand-crafted binary sample selection[1]	Reorganizes data for up-sampling	$N \times \frac{h}{2} \times \frac{w}{2} \times 12$
Depth to space	-	-	Up-sampling	$N \times h \times w \times 3$
5×5 convolution	-	zero-padded 3×3 bilinear kernel	Demosaicing	$N \times h \times w \times 3$
1×1 convolution	-	sample color conversion matrix	Color-space conversion (sRGB)	$N \times h \times w \times 3$
1×1 convolution	tanh	pre-trained model	Gamma correction[2]	$N \times h \times w \times 12$
1×1 convolution	-	pre-trained model	Gamma correction[2]	$N \times h \times w \times 3$
Clip to [0,1]	-	-	output RGB image	$N \times h \times w \times 3$

[1] we disabled optimization of this filter to speed up convergence
[2] adapted from a 2-layer network trained separately to approximate gamma correction

Table 3. The UNet architecture: 7,760,268 trainable parameters

Operation	**Activation**	**Input**	**Output**	**Output size**
Input	-	-	x	$N \times h/2 \times w/2 \times 4$
3×3 convolution	leaky ReLU	x	$c_{1,1}$	$N \times h/2 \times w/2 \times 32$
3×3 convolution	leaky ReLU	$c_{1,1}$	$c_{1,2}$	$N \times h/2 \times w/2 \times 32$
2×2 max pooling	-	$c_{1,2}$	p_1	$N \times h/4 \times w/4 \times 32$
3×3 convolution	leaky ReLU	p_1	$c_{2,1}$	$N \times h/4 \times w/4 \times 64$
3×3 convolution	leaky ReLU	$c_{2,1}$	$c_{2,2}$	$N \times h/4 \times w/4 \times 64$
2×2 max pooling	-	$c_{2,2}$	p_2	$N \times h/8 \times w/8 \times 32$
3×3 convolution	leaky ReLU	p_2	$c_{3,1}$	$N \times h/8 \times w/8 \times 128$
3×3 convolution	leaky ReLU	$c_{3,1}$	$c_{3,2}$	$N \times h/8 \times w/8 \times 128$
2×2 max pooling	-	$c_{3,2}$	p_3	$N \times h/16 \times w/16 \times 128$
3×3 convolution	leaky ReLU	p_3	$c_{4,1}$	$N \times h/16 \times w/16 \times 256$
3×3 convolution	leaky ReLU	$c_{4,1}$	$c_{4,2}$	$N \times h/16 \times w/16 \times 256$
2×2 max pooling	-	$c_{4,2}$	p_4	$N \times h/32 \times w/32 \times 256$
3×3 convolution	leaky ReLU	p_4	$c_{5,1}$	$N \times h/32 \times w/32 \times 512$
3×3 convolution	leaky ReLU	$c_{5,1}$	$c_{5,2}$	$N \times h/32 \times w/32 \times 512$
2×2 strided convolution	-	$c_{5,2}$	s_5	$N \times h/16 \times w/16 \times 256$
3×3 convolution	leaky ReLU	$s_5 \mid c_{4,2}$	$c_{6,1}$	$N \times h/16 \times w/16 \times 256$
3×3 convolution	leaky ReLU	$c_{6,1}$	$c_{6,2}$	$N \times h/16 \times w/16 \times 256$
2×2 strided convolution	-	$c_{6,2}$	s_6	$N \times h/8 \times w/8 \times 128$
3×3 convolution	leaky ReLU	$s_6 \mid c_{3,2}$	$c_{7,1}$	$N \times h/8 \times w/8 \times 128$
3×3 convolution	leaky ReLU	$c_{7,1}$	$c_{7,2}$	$N \times h/8 \times w/8 \times 128$
2×2 strided convolution	-	$c_{7,2}$	s_7	$N \times h/4 \times w/4 \times 64$
3×3 convolution	leaky ReLU	$s_7 \mid c_{2,2}$	$c_{8,1}$	$N \times h/4 \times w/4 \times 64$
3×3 convolution	leaky ReLU	$c_{8,1}$	$c_{8,2}$	$N \times h/4 \times w/4 \times 64$
2×2 strided convolution	-	$c_{7,2}$	s_8	$N \times h/2 \times w/2 \times 32$
3×3 convolution	leaky ReLU	$s_8 \mid c_{1,2}$	$c_{9,1}$	$N \times h/2 \times w/2 \times 32$
3×3 convolution	leaky ReLU	$c_{9,1}$	$c_{9,2}$	$N \times h/2 \times w/2 \times 32$
1×1 convolution	-	$c_{9,2}$	c_{10}	$N \times h/2 \times w/2 \times 12$
Depth to space	-	c_{10}	y_{rgb}	$N \times h \times w \times 3$
Clip to [0,1]	-	y_{rgb}	y	$N \times h \times w \times 3$

All leaky ReLUs have $\alpha = 0.2$
$\mid$ denotes concatenation along the feature dimension

Table 4. The DNet architecture: 493,976 trainable parameters

Operation	**Activation**	**Input**	**Output**	**Output size**
Input	-	-	c_0	$N \times h/2 \times w/2 \times 4$
Repeat for $i = 1, 2, \ldots, 14$ {				
3×3 convolution + BN	ReLU	c_{i-1}	$\hat{c}_i$	$N \times h/2 - 2 \times w/2 - 2 \times 64$
Padding (reflection)	-	$\hat{c}_i$	c_i	$N \times h/2 \times w/2 \times 64$
}				
3×3 convolution + BN	ReLU	c_{14}	$\hat{c}_{15}$	$N \times h/2 - 2 \times w/2 - 2 \times 12$
Padding (reflection)	-	$\hat{c}_{15}$	c_{15}	$N \times h/2 \times w/2 \times 12$
Depth to space	-	c_{15}	f_{conv}	$N \times h \times w \times 3$
1×1 convolution	-	c_0	c_{16}	$N \times h/2 \times w/2 \times 12$
Depth to space	-	c_{16}	f_{bayer}	$N \times h \times w \times 3$
3×3 convolution	ReLU	$f_{\text{conv}} \mid f_{\text{bayer}}$	$\hat{c}_{17}$	$N \times h - 2 \times w - 2 \times 64$
Padding (reflection)	-	$\hat{c}_{17}$	c_{17}	$N \times h \times w \times 64$
1×1 convolution	-	c_{17}	y_{rgb}	$N \times h \times w \times 3$
Clip to [0,1]	-	y_{rgb}	y	$N \times h \times w \times 3$

$\mid$ denotes concatenation along the feature dimension

Table 5. The dJPEG architecture for JPEG codec approximation

Operation	**JPEG Function**	**Output size**
Input	-	$N \times h \times w \times 3$
1×1 convolution	RGB $\rightarrow$ YCbCr	$N \times h \times w \times 3$
Space to depth & reshapes	Isolate 8×8 px blocks	$3N \times 8 \times 8 \times B$
Transpose & reshape	-	$3BN \times 8 \times 8$
$2 \times$ matrix multiplication	Forward 2D DCT	$3BN \times 8 \times 8$
Element-wise matrix division	Divide by quantization matrices	$3BN \times 8 \times 8$
Rounding / approximate rounding	Quantization	$3BN \times 8 \times 8$
Element-wise matrix multiplication	Multiply by quantization matrices	$3BN \times 8 \times 8$
$2 \times$ matrix multiplication	Inverse 2D DCT	$3BN \times 8 \times 8$
Transpose & reshape	-	$3N \times 8 \times 8 \times B$
Depth to space & reshapes	Re-assemble 8×8 px blocks	$N \times h \times w \times 3$
1×1 convolution	YCbCr $\rightarrow$ RGB	$N \times h \times w \times 3$

Table 6. The FAN architecture: 1,341,990 trainable parameters

Operation	**Activation**	**Initialization**	**Comment**	**Output size**
Input	-	-	RGB input	$N \times h \times w \times 3$
5×5 convolution	-	Standard residual filter[1]	Constrained convolution	$N \times h \times w \times 3$
5×5 convolution	leaky ReLU	MSRA	-	$N \times h \times w \times 32$
2×2 max pool	-	-	-	$N \times h/2 \times w/2 \times 32$
5×5 convolution	leaky ReLU	MSRA	-	$N \times h/2 \times w/2 \times 64$
2×2 max pool	-	-	-	$N \times h/4 \times w/4 \times 64$
5×5 convolution	leaky ReLU	MSRA	-	$N \times h/4 \times w/4 \times 128$
2×2 max pool	-	-	-	$N \times h/8 \times w/8 \times 128$
5×5 convolution	leaky ReLU	MSRA	-	$N \times h/8 \times w/8 \times 256$
2×2 max pool	-	-	-	$N \times h/16 \times w/16 \times 256$
1×1 convolution	leaky ReLU	MSRA	-	$N \times h/16 \times w/16 \times 256$
global average pooling	-	-	-	$N \times 256$
fully connected	leaky ReLU	MSRA	-	$N \times 512$
fully connected	leaky ReLU	MSRA	-	$N \times 128$
fully connected	Softmax	MSRA	Class probabilities	$N \times 5$