2022 Vol. 60, No. 7 / 1 March 2021 / Applied Optics Research Article

applied optics

Deep neural network for fringe pattern filtering and normalization

Alan Reyes-Figueroa,[1] Victor H. Flores,[1,2] and Mariano Rivera[1,*]

[1]Centro de Investigación en Matemáticas AC, 36023, Guanajuato, Gto., Mexico
[2]Departamento de Ingeniería Robótica, Universidad Politécnica del Bicentenario, 36283 Silao, Gto., Mexico
*Corresponding author: mrivera@cimat.mx

Received 28 October 2020; revised 4 January 2021; accepted 13 January 2021; posted 22 January 2021 (Doc. ID 413404); published 1 March 2021

We propose a new framework for processing fringe patterns (FPs). Our novel, to the best of our knowledge, approach builds upon the hypothesis that the denoising and normalization of FPs can be learned by a deep neural network if enough pairs of corrupted and ideal FPs are provided. The main contributions of this paper are the following: (1) we propose the use of the U-net neural network architecture for FP normalization tasks; (2) we propose a modification for the distribution of weights in the U-net, called here the V-net model, which is more convenient for reconstruction tasks, and we conduct extensive experimental evidence in which the V-net produces high-quality results for FP filtering and normalization; (3) we also propose two modifications of the V-net scheme, namely, a residual version called ResV-net and a fast operating version of the V-net, to evaluate potential improvements when modifying our proposal. We evaluate the performance of our methods in various scenarios: FPs corrupted with different degrees of noise, and corrupted with different noise distributions. We compare our methodology versus other state-of-the-art methods. The experimental results (on both synthetic and real data) demonstrate the capabilities and potential of this new paradigm for processing interferograms.

https://doi.org/10.1364/AO.413404

1. INTRODUCTION

Fringe pattern (FP) denoising–normalization consists of removing background illumination variations, normalizing amplitude, and filtering noise, which means transforming an FP corresponding to the mathematical model

$$x(p) = a(p) + b(p)\cos(\phi(p)) + \eta(p) \tag{1}$$

into the normalized FP modeled by

$$\hat{x}(p) = 1 + \cos(\phi(p)). \tag{2}$$

Here, x is the signal or irradiance, a the background illumination, b the fringe amplitude or modulation, ϕ the phase map, η an additive or correlated noise, and p the pixel position to indicate the spatial dependence of those quantities. Such a normalization can be represented by the transformation

$$\hat{x} = \mathcal{H}\{x\}. \tag{3}$$

FP normalization is a critical step in fringe analysis processing. There is a consensus that FP analysis can be seen as a pipeline that involves the following steps: denoising, normalization, phase extraction, and phase unwrapping, even though some of these steps are merged by some methods. In all these steps, background and modulation variations are considered as a source of error in phase estimation [1]. More precisely, in the simplest version of phase estimation [2], one introduces a set of $n \geq 2$ known phase steps δ_k, $k = 1, 2, \ldots, n$, in the modulating phase, and a set of signals,

$$x_k(p) = a_k(p) + b_k(p)\cos(\phi(p) + \delta_k) + \eta_k(p), \tag{4}$$

$1 \leq k \leq n$, is acquired.

The wrapped phase is estimated by

$$\hat{\phi}(p) = \arctan_2(x_s(p), x_c(p)), \tag{5}$$

where x_s and x_c denote the signals associated with the sine and cosine, respectively, of the phase multiplied by the modulation amplitude, i.e.,

$$x_s(p) = b(p)\sin\phi(p); \quad x_c(p) = b(p)\cos\phi(p). \tag{6}$$

Both terms are always computed as a combination of the phase shifted images x_k, $1 \leq k \leq n$.

It is well known that the quality in the approximation represented by the components (6) determines the error of the phase obtained by (5). Most phase estimation algorithms assume that the background a and modulation b do not change with the phase step. When this requirement is not satisfied, e.g., when there is a lack of illumination control, using Eq. (5) directly is conducive to big errors in phase estimation [2].

(a) (b) (c)

Fig. 1. Normalization of an FP with incomplete field of view: (a) data, (b) GFB (note the artifacts in regions with low-frequency fringes and near the border of the region of interest), and (c) proposal.

It is well known that algorithms for phase demodulation from a single FP with closed fringes [3–6] are particularly sensitive when processing noisy and denormalized FPs: deviations in the normalization $\hat{x}$ may produce failures in the phase estimation [2]. Moreover, according to Ref. [7], this problem also appears when analyzing pairs of FPs with a phase shift.

Due to the reasons above, it is important to develop robust methods to filter the signal x before the demodulation process.

References [8,9] present useful reviews of challenges related to FP analysis and a good list of proposed solutions. For example, the denoising, normalization, and phase extraction can be accomplished using two-dimensional windowed Fourier transform (WFT) [10], wavelet transform (WT), [11,12] or a Gabor filter bank (GFB)-based method [13]. As noted in Refs. [11,12], WFT and FT have limitations in dealing with FPs when phase discontinuities are present or the field of view (FOV) is not the full image. As we will show in this work, the same limitations apply for the GFB approach. These techniques estimate a transformation of the form (3), for a central pixel in an image neighborhood (image patch or weighted image window).

WFT, WT, and GFB methods rely upon the assumption that the neighborhood of a pixel of interest (image patch) has an almost constant frequency and phase, i.e., locally, the phase map is close to being a plane. The limitations of the mentioned methods occur at patches where the main assumption is violated, i.e., at phase discontinuities. Figure 1 shows a denoised–filtered FP computed with a GFB-based method and with our proposal. Further information on alternative strategies for FP normalization can be found in Ref. [9]. However, methods based on local spectral analysis (e.g., WFT, WT, and GFB) have shown to be very robust general methods for dealing with high noise levels [11–15]. In this work, we propose to implement such a transformation Eq. (3) with a deep neural network (DNN).

Neural networks (NNs) are known for being universal approximators [16]. Early attempts to use a NN for FP analysis are of limited application since they are based on simple multilayer schemes. In particular, in Ref. [17], a multilayer NN is trained for computing the phase and its gradient at the central pixel of an image patch. Instead, our proposal computes the normalization for the entire image patch. In addition, our work is based on a deep autoencoder NN that allows us to deal with noise and large illumination changes.

This paper investigates the V-net model performance in the task of normalizing FPs, and a comparison to other NN models is presented. Recently, we reported in Ref. [7] how the V-net performs in the task of normalizing FPs for phase step estimation. In that work, the V-net performance is favorably evaluated versus GFB and Hilbert–Huang transform (HHT) methods.

In this work, we present the details of the V-net. The paper is organized as follows: Section 2 presents a brief review of the theoretical foundations of NN autoencoders, and the architecture details of our U-net variants: V-net and residual V-net (ResV-net) models. In Section 3, the implementation details for the simulated data and the inference process for normalized FPs using the V-net models is described. Section 4 is devoted to applying the V-net models to normalize synthetic FPs and evaluate the method in several scenarios. Conclusions are given in Section 5.

2. METHOD

A. U-Net Model

Autoencoders were originally devised to compress (codify) and decompress (decodify) data vectors [18]. Autoencoders have two main components: (1) encoder $\mathcal{E}$ takes the data x in its original "spatial" dimension and produces a compressed vector y. This encoding can be expressed by

$$y = \mathcal{E}(x) \stackrel{\text{def}}{=} \varphi_1(W_1 x + b_1), \tag{7}$$

where x is the original data, W_1 the weights matrix, b_1 a bias term, y the encoded data, and φ_1 an activation function, e.g., rectified linear unit (ReLU), sigmoid, or softmax [19]. (2) Decoder $\mathcal{D}$ takes the compressed data y and computes a reconstruction $\hat{x}$ of the original data x, of the same dimension as x. This is expressed by

$$\hat{x} = \mathcal{D}(y) \stackrel{\text{def}}{=} \varphi_2(W_2 y + b_2), \tag{8}$$

where W_2 is a weights matrix, b_2 a bias term, and φ_2 the activation function.

In this work we propose to use a deep autoencoder, called the U-net [20], and its proposed variant, V-net, to perform FP denoising (FPD) and normalization. The U-net is a fully convolutional NN (CNN), and was initially designed for image classification (segmentation). That is, for each input image, it produces an image of labels of the same size as the input [21], unlike standard classification networks whose output is a single value. The loss function for the U-net is of the form

$$\underset{\theta}{\operatorname{argmin}} \sum_i \| f(x_i) - (\mathcal{D} \circ \mathcal{E}) x_i \|_M, \tag{9}$$

where $f(x_i)$ is a segmentation of x_i, $\mathcal{E}$ and $\mathcal{D}$ represent the encoder and decoder stages, respectively, θ is the vector of the autoencoder parameters, and $\| \cdot \|_M$ is a metric or divergence measure.

One can note important differences between classical autoencoders and the U-net: (1) U-net is a deep model, i.e., the number of layers in U-net is substantially larger than the number of layers in classical autoencoders. (2) U-net implements convolutional 2D filters so that the weights W of each layer are codified into an array of matrices (a 3D tensor) and produces a vector of processed images (a 3D tensor). On the other hand, classical autoencoders vectorize the input image, and therefore, the pixel's spatial relationships are lost. (3) The input of a decoder layer in U-net is the concatenation of the output tensor of the previous layer and output tensor of the symmetric encoder layer

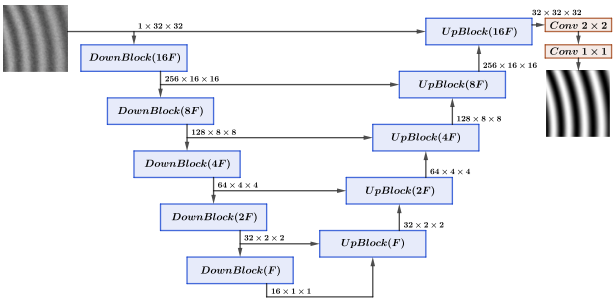

Fig. 2. V-net architecture. The number of filters per block is inversely distributed: while U-net is designed for image segmentation (classification), V-net (shown above) is for image reconstruction (regression).

(so-called "skip links") (Fig. 2). The purpose of skip links can be understood in the context of residual nets [22]: they allow to construct the solution using both coarse and processed data.

As in standard convolutional DNNs, U-net follows the thumb rule for processing input tensors: the encoding layers increase the number of channels (filters) of the input tensor, and reduce its spatial dimension, while the decoding layers shrink the number of channels and extend the spatial dimensions of the processed tensors. Therefore, one improves the computational efficiency by applying a reduced number of filters on tensors with larger spatial dimension and a larger number of filters on spatial small-sized tensors. In the training stage, the filters are optimized for detecting the useful features that allow the U-net to correctly classify the pixels.

B. V-Net Model

FP normalization can be understood as a regression problem. Since classification (image segmentation) and regression (image filtering) may require different features, in this work, we propose a U-net modification designed to achieve image filtering instead of image segmentation. Despite the computational cost that it implies, our network applies a larger number of filters on the input tensor (original image) to capture local details and achieve precise reconstruction. Also, as opposed to the standard U-net, we reduce the number of filters as the layers are deeper on the encoder. Thus, the deepest layer in the encoder stage (bottom layer in Fig. 2) produces a tensor with the smallest dimension (spatial size and number of channels). These characteristics distinguish our architecture and provide our DNN with an advantage for the regression task. We call our improved model "V-net" because it uses tensors with few channels on deeper layers.

According to our knowledge, in the deep learning literature there are no previous studies about the distribution in the number of filters on CNNs. Recently, we reported in Refs. [7,23] that V-net filter distribution outperforms the classic U-net filter distribution. In this work, we present the details of our U-net variant, the so-called V-net. So, our approach can be seen as a contribution that produces better results for the task of FP normalization and can be used in other image processing tasks.

The V-net encoder is composed of a sequence of K encoding blocks (down-blocks), followed by the decoder, which consists of a sequence of K decoding blocks (up-blocks), and a tail

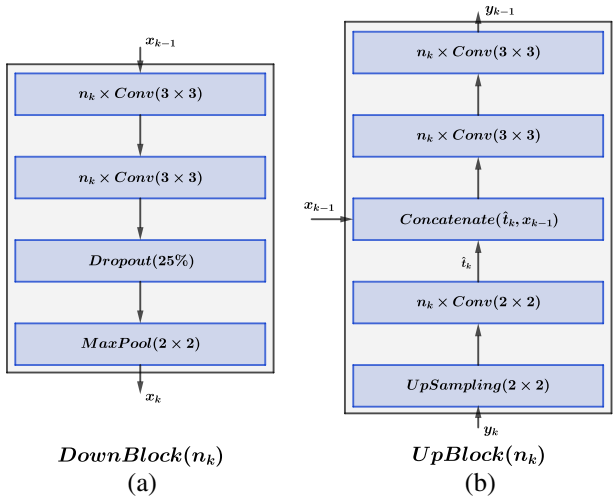

Fig. 3. V-net components: (a) down-block and (b) up-block.

Table 1. V-Net Down-Block (a, b, f, p) Architecture

Layer	Output Shape	# Params[a]
Conv2D	(None, a, b, f)	$3 \times 3 \times f$
Conv2D	(None, a, b, f)	$3 \times 3 \times f$
Dropout	(None, a, b, f)	0
MaxPooling2D	(None, $\lfloor \frac{a}{2} \rfloor, \lfloor \frac{b}{2} \rfloor, f$)	0

[a]Parameters: (None, a, b) = shape of input tensor, f = number of filters, p = dropout rate.

Table 2. V-Net Up-Block (a, b, f) Architecture

Layer	Output Shape	# Params[a]
UpSampling2D	(None, $2a, 2b, f$)	0
Conv2D	(None, $2a, 2b, 2f$)	$2 \times 2 \times f$
Concatenate	(None, $2a, 2b, 4f$)	0
Conv2D	(None, $2a, 2b, 2f$)	$3 \times 3 \times 2f$
Conv2D	(None, $2a, 2b, 2f$)	$3 \times 3 \times 2f$

[a]Parameters: (None, a, b) = shape of input tensor, f = number of filters.

(composed of the two last convolutional layers) (Fig. 2). The kth down-block, $k = 1, 2, \ldots, K$, starts by applying two convolutional layers with n_k channels of size 3×3. This number n_k determines the amount of output channels at each stage. A complete description of each encoding and decoding block architecture is illustrated in Fig. 3 and Tables 1 and 2.

In the following, x_k denotes the output tensor of the kth down-block, $k = 1, 2, \ldots, K$, as well as the second input of the $(k+1)$th up-block, $k = 1, 2, \ldots, K-1$. Similarly, y_k denotes the first input tensor of the kth up-block, $k = K, \ldots, 2, 1$, as well as the output for the $(k+1)$th up-block, $k = K-1, \ldots, 1, 0$.

The complete architecture of our V-net implementation is summarized in Table 3.

In our implementation, we set $K = 4$ (so we have four spatial-size levels). The number of channels n_k, $k = 1, 2, \ldots, K$, is determined by the number of channels n_{k-1} in the previous down-block following the rule $n_k = \frac{1}{2} n_{k-1}$. The same occurs with the number of channels for each up-block. In our model, the respective number of channels is set as

Table 3. V-Net Architecture

Block/Layer	Output Shape	# Params	Inputs
01. Input layer	(None, 32, 32, 1)	0	
02. DownBlock(32,32,256,0)	(None, 16, 16, 256)	592640	01
03. DownBlock(16,16,128,0.25)	(None, 8, 8, 128)	442624	02
04. DownBlock(8,8,64,0.25)	(None, 4, 4, 64)	110720	03
05. DownBlock(4,4,32,0.25)	(None, 2, 2, 32)	27712	04
06. UpBlock(2,2,32)	(None, 4, 4, 32)	592640	05, 04
07. UpBlock(4,4,64)	(None, 8, 8, 64)	592640	06, 03
08. UpBlock(8,8,128)	(None, 16, 16, 128)	592640	07, 02
09. UpBlock(16,16,256)	(None, 32, 32, 256)	592640	08, 01
10. Tail Conv2D	(None, 32, 32, 2)	578	09
11. Tail Conv2D	(None, 32, 32, 1)	3	10
Total params		3,721,669	
Trainable params		3,721,669	
Non-trainable params		0	

$n_k = 256, 128, 64, 32$, for $k = 1, 2, \ldots, 4$. Although more general configurations for the number of channels can be implemented, we have chosen as the global parameter the number $F = n_K = 32$ of channels in the last down-block (bottom level), and the other n_k's are determined by F as indicated in Table 3. Figure 2 indicates a deeper version with $K = 5$ blocks and $F = n_5 = 16$.

Finally, the training of the V-net can be written as

$$\underset{\Theta}{\operatorname{argmin}} \|Y - \hat{Y}\|_1 = \underset{\Theta}{\operatorname{argmin}} \|Y - (\mathcal{T} \circ \mathcal{D} \circ \mathcal{E})X_0\|_1, \quad \textbf{(10)}$$

where X_0 is the input tensor (stack of all input patches x_0), Y is the desired output (stack of all normalized FP patches y), and $\hat{Y} = (\mathcal{T} \circ \mathcal{D} \circ \mathcal{E})X_0$ is the output tensor (stack of all patch estimations $\hat{y}$) of the V-net. The operator $\mathcal{T}$ represents the tail stage (last two convolutional layers), added to produce a final refinement of the reconstruction. Here, Θ is the set of all model parameters (filter weights and bias). We use the L_1 norm as a loss function because it induces lower reconstruction errors.

C. Residual V-Net

We also propose another two U-net variants to explore whether there exist variations of our V-net model that produce even better FP normalizations.

The first variation is a residual version of the V-net. Residual architectures are common in the deep learning architecture design [22], and were introduced to tackle the well-known problem of "vanishing gradient" on deep networks and improve the training process. Residual learning models can be understood in the context of autoencoders. While in a typical autoencoder the encoding–decoding scheme is produced as in Eqs. (7) and (8) by $\hat{x} = (\mathcal{D} \circ \mathcal{E})(x)$, the estimation in the residual version of an autoencoder can be modeled by

$$\hat{x} = x \ominus (\mathcal{D} \circ \mathcal{E})(x) = x \ominus z, \quad \textbf{(11)}$$

where $z = (\mathcal{D} \circ \mathcal{E})(x)$.

Here, the original input signal can be understood as a summation:

$$x = \hat{x} \oplus z, \quad \textbf{(12)}$$

Table 4. ResV-Net ResUp-Block (a, b, f) Architecture

Layer	Output Shape	# Params[a]
UpSampling2D	(None, $2a$, $2b$, f)	0
Conv2D	(None, $2a$, $2b$, $2f$)	$2 \times 2 \times f$
Subtract	(None, $2a$, $2b$, $2f$)	0
Conv2D	(None, $2a$, $2b$, $2f$)	$3 \times 3 \times 2f$
Conv2D	(None, $2a$, $2b$, $2f$)	$3 \times 3 \times 2f$

[a]Parameters: (None, a, b) = shape of input tensor, f = number of filters.

where the $\hat{x}$ term refers to the clean (filtered and normalized) signal, plus a noise term z, which describes all the perturbation (background, modulation, and noise) added to the normalized FP, which is undesired. Hence, the residual scheme learns the noise z that one wants to remove, and can be interpreted as the inverse process to add noise. Residual NN architectures are common in the image processing community when the associated problem is related to restore and denoise some input signal x.

Our implementation of the residual version of the V-net, here called "ResV-net," is the following: we have replaced all the concatenate layers in Table 3, by subtract layers. That is, instead of concatenating tensors $\hat{t}_k$ and x_{k-1} as

$$\tilde{t}_k(m, i, j) = \begin{cases} \hat{t}_k(m, i, j) & m = 1 : n_k \\ x_{k-1}(m - n_k + 1, i, j) & m = n_k : n_k + n_{k-1} \end{cases}, \quad \textbf{(13)}$$

where (m, i, j) indicates the layer m and the (x, y) spatial position of an input tensor, we replace them by

$$\tilde{t}_k(m, i, j) = x_{k-1}(m, i, j) \ominus \hat{t}_k(m, i, j), \quad m = 1, \ldots, n_k. \quad \textbf{(14)}$$

Hence, at each decoder stage (up-blocks) $k = K. \ldots, 2, 1$, the ResV-net model is designed to learn a tensor $\hat{t}_k$ that removes an amount of undesired noise from the input signal x. Observe that the residual layer introduction reduces the number of channels in each up-block from $n_k + n_{k-1}$ to n_k, so it reduces the number of parameters in the model. Table 4 describes the redefinition of the up-blocks for the ResV-net. Our implementation of the ResV-net is summarized in Table 5.

A second variation of the proposed V-net is a fast version of the original V-net. It will be described in Section 3.

Table 5. ResV-Net Architecture

Block/Layer	Output Shape	# Params	Inputs
01. Input layer	(None, 32, 32, 1)	0	
02. DownBlock(32,32,256,0)	(None, 16, 16, 256)	592640	01
03. DownBlock(16,16,128,0.25)	(None, 8, 8, 128)	442624	02
04. DownBlock(8,8,64,0.25)	(None, 4, 4, 64)	110720	03
05. DownBlock(4,4,32,0.25)	(None, 2, 2, 16)	27712	04
06. ResUpBlock(2,2,32)	(None, 4, 4, 32)	592640	05, 04
07. ResUpBlock(4,4,64)	(None, 8, 8, 64)	592640	06, 03
08. ResUpBlock(8,8,128)	(None, 16, 16, 128)	592640	07, 02
09. ResUpBlock(16,16,256)	(None, 32, 32, 256)	592640	08, 01
10. Tail Conv2D	(None, 32, 32, 2)	578	09
11. Tail Conv2D	(None, 32, 32, 1)	3	10
Total params		3,721,669	
Trainable params		3,721,669	
Non-trainable params		0	

3. IMPLEMENTATION DETAILS

A. Simulated Data

To quantitatively evaluate the performance of the proposed V-net-based normalization, we consider two datasets: the first consists of 46 generated pairs of FPs with size 1024×1024 pixels; the second consists of 180 generated FPs with size 320×320 pixels. In both cases, the corrupted FPs were generated according to the model in Eq. (1) and the normalized FPs (ground truth) according to the model in Eq. (2). The normally distributed random noise was generated with the Python Numpy package. On the other hand, the random smooth functions (illumination components and phase) were constructed using random numbers with uniform distribution generated with our implementation of a linear congruential generator with portable operating system interface (POSIX) parameters [24] to guarantee the FP generation replicability. In the following, we explain the smooth random surface generation procedure.

We generated the pseudo-random phase with a radial basis function with a Gaussian kernel [25]:

$$\phi(p) = \sum_{i=1}^{10} \alpha_i G(p; m_i, \sigma_\phi), \tag{15}$$

where we define

$$G(p; \mu, \sigma) \stackrel{\text{def}}{=} \exp\left(-\frac{1}{2\sigma^2}\|p - \mu\|^2\right), \tag{16}$$

with $m = [m_1, m_2, \ldots, m_{10}]^T$ the vector of the random kernel centers uniformly distributed into the FPs lattice (i.e., $m_i \in [0, 1023]^2$), and α the vector of random uniformly distributed Gaussian heights, with $\alpha_i \in [-180/\pi, 180/\pi]$. Similarly, we generated the illumination term with $a(p) = G(p, m_a; \sigma_a)$ and $b(p) = G(p, m_b; \sigma_b)$. In our data, we selected m_a and m_b uniformly distributed over the image domain (using our implementation of the POSIX algorithm), and we set $\sigma_\phi = 1024/6$, $\sigma_a = 1024/2$, and $\sigma_b = 1024$. Figure 4 depicts an example of the synthetic data used for training. Figs. 4(a) and 4(b) show the ideal and corrupted FPs, respectively, and 4(c) and 4(d) show a selected region. An example of a patch pair of size 32×32 used for training is depicted in 4(e) and 4(f).

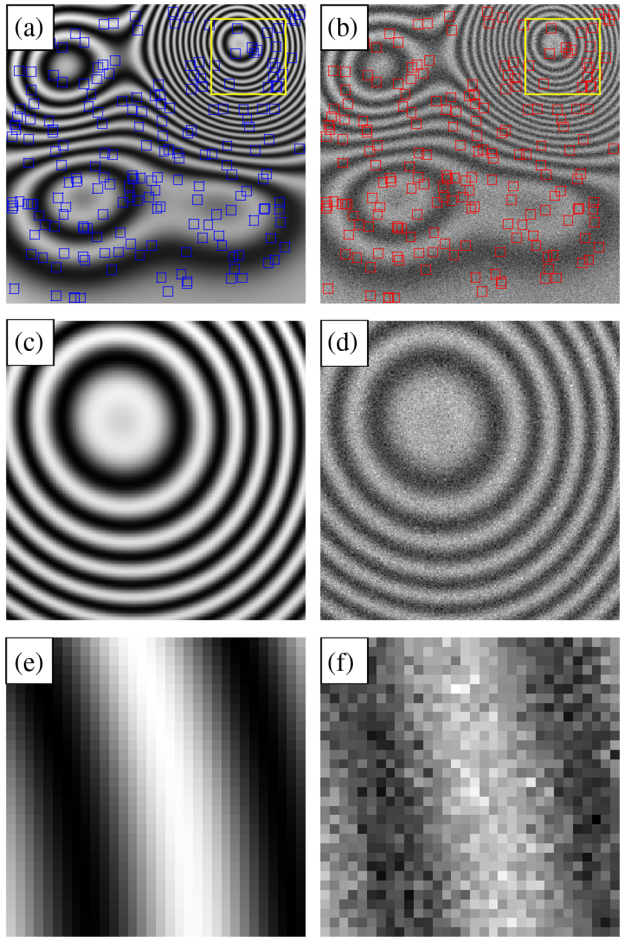

Fig. 4. Example of training data. (a) Synthetic normalized FP of 1024×1024 pixels with a selected region of interest of 256×256 pixels (yellow square), small random patches of 32×32 in blue; (b) the same FP corrupted with Gaussian noise, patches in red; (c), (d) regions of interest in (a), (b), respectively. Random patch pair used for training: (e) ground truth and (f) corrupted input.

In the Experiments section, we evaluate our method performance for different noise types, as Gaussian, salt–pepper, speckle, and combinations of them. We also evaluate the performance of

the models when some circular field of vision restriction mask (FOV) is added.

For the salt–pepper noise experiment, we randomly select the 25% of the pixels and saturate them to values zero and one in equal proportion.

In the case of FPs with speckle noise (correlated noise), we simulate images of an out-of-plane sensible electronic speckle pattern interferogram (ESPI). An interferogram produced by speckle is, indeed, affected by factors such as the point spread function (PSF) of the optical system and the sensor's characteristics (temperature, CCD pixel size), as well as the object's albedo/specular spatial ratio, shape, and displacement. Those variables make such a phenomenon complex to model. In this work we follow a simple model for generating simulation of out-of-plane sensible ESPI images. Since our simplified interferograms are used to evaluate all the denoising methods, we can neglect all the abovementioned sources of noise and focus on the most important property of the speckle: correlated noise (multiplicative noise). According to Ref. [26], the out-of-plane speckle interferogram can be modeled with

$$I_0(p) = a(p) + b(p)\cos(\eta_s(p)), \tag{17}$$

where $a(p) = I_i(p)^2 + I_r(p)^2$, and $b(p) = 2I_i(p)I_r(p)$, where I_i is the amplitude of the object wevefront, I_r is the amplitude of the reference, and $\eta_s(p)$ is random noise with a large variance (on the order of surface roughness, very large with respect to the wave longitude, λ, of the illumination light). Assuming independence of the time of the interferogram, a slight nonuniform deformation is introduced with displacements in the object's surface only out of the plane, $z(p)$; such a deformation introduces a phase shift $\phi(p) = 4\pi z(p)/\lambda$ into the random phase eta_s. Then the second registered speckle interferogram corresponds to

$$I_1(p) = a(p) + b(p)\cos(\phi(p) + \eta_s(p)). \tag{18}$$

Thus, an ESPI image can be computed with

$$\begin{aligned} |I_0(p) - I_1(p)| &= b(p)|\cos\eta_s(p) - \cos(\eta_s(p) + \phi(p)| \\ &= 2b(p)|\sin(\phi(p)/2)\sin(\eta_s(p) + \phi(p)/2)|. \end{aligned} \tag{19}$$

The important part in Eq. (19) is that one obtains a rectified sinusoidal FP multiplied by noise $\sin(r + \phi/2)$, where the low-frequency contribution of $\phi/2$ can be neglected. Then, we add Gaussian noise, η, with zero mean and standard deviation $\sigma_\eta = 2.5$ to make the ESPI image more realistic.

In the ESPI case, we also evaluated two more testing experiments with extreme noise conditions. For those experiments, we considered the second dataset of FPs with size 320×320. This FP dataset is generated according to model (19), in two scenarios:

(i) *high noise*: η_1 has uniform distribution in [0, 1000] (radians), and η_2 is a low Gaussian distribution (with zero mean and $\sigma_\eta = 2.5$, and modulation amplitude $b(p)$ generated as a Gaussian with maximal amplitude $b = 0.4$ (equivalent to 40% of the signal) (Fig. 5);

(ii) *extreme noise*: η_1 has uniform distribution in [0, 1000] (radians), and η_2 is a high Gaussian distribution (with zero mean and $\sigma_\eta = 2.5$, and modulation amplitude $b(p)$ generated as a Gaussian with maximal amplitude $b = 1.25$ (equivalent to 125% of the signal) (Fig. 6).

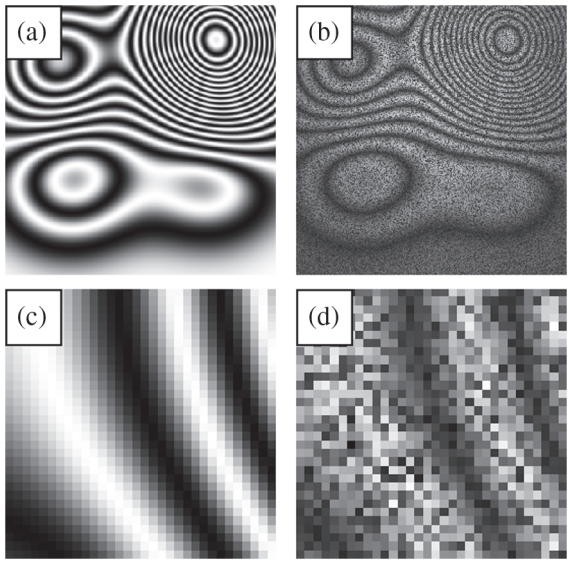

Fig. 5. Example of high-noise training data. (a) Synthetic normalized FP of 320×320 pixels; (b) same FP corrupted with speckle noise. A random patch pair of 32×32 used for training: (c) ground truth and (d) corrupted input.

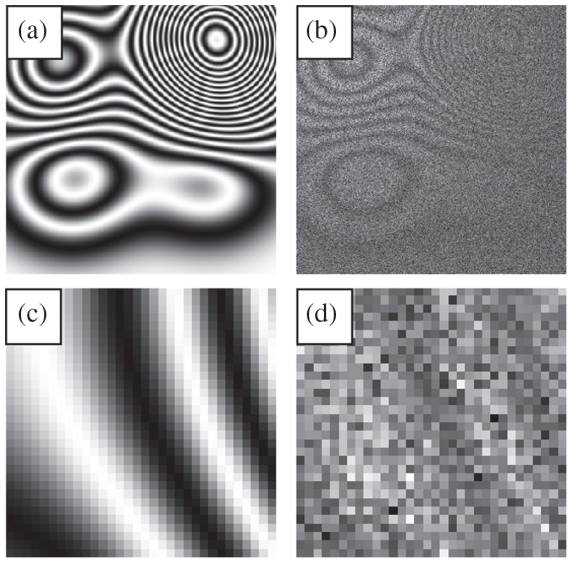

Fig. 6. Example of extreme-noise training data. Same legends as Fig. 5.

B. Training Data Sets

The training data generated from the first image dataset consists of 25,000 random patches of 32×32 pixels sampled from the first 30 generated FPs (the set of training images); 2500 of those patches were used for validation. In addition, the remaining 16 FPs were used as the test data set to measure and evaluate the performance of all compared models. We stacked the corrupted patches in the tensor $X_0 = [x_i]_{1=1,2,\_,25,000}$ and the

corresponding normalized patches form the desired output $Y = [y_i]_{1=1,2,\_,25,000}$. The input data for all our models are pairs of tensors X_0 and Y.

For the high-noise and extreme-noise experiments, the training data were generated from the second image dataset. This consists of 40,000 random patches of 32×32 pixels sampled from the first 150 generated FPs (the set of training images); 4000 of those patches were used for validation. In this case, the remaining 30 FPs were used as the test data set for evaluation purposes. Similar to the previous experiment, the tensors $X_0 = [x_i]_{1=1,2,\_,40,000}$ and corresponding normalized patches $Y = [y_i]_{1=1,2,\_,40,000}$ form the input data for all tested models.

In both cases, the patch size of 32 is a user-defined parameter. We chose the size as 32×32 by considering a maximum frequency close to 1.5 fringes per patch. Moreover, the V-net also requires a patch size divisible by $2^K = 2^5 = 32$, where K is the number of levels or blocks included in the model design.

C. Prediction of a Full FP from Reconstructed Patches

Recall that the V-net is designed to reconstruct small FP patches of 32×32 pixels. Thus, to reconstruct an entire FP, we generated a set of patches using a sliding window scheme, with a stride (pixels shift step) of $s_x = s_y = 4$ pixels in both horizontal and vertical directions in the 1024×1024 images, and $s_x = s_y = 2$ for the 32×32 datasets. All patches are fed to the V-net to compute their normalizations (Fig. 7). Each pixel in the entire reconstructed FP was computed as the average of the values in the same pixel position obtained from overlapped normalized patches. We preferred the mean because it is more efficiently computed than the median, and we did not find a significant difference if the median were used instead.

Again, pixel shifts s_x and s_y are user-defined parameters. We also tested pixel shifts equal to two and one, but the improvement in reconstruction was not significant, and the selection of lesser values of s_x and s_y only increases the computational cost and time (nearly four times for the two stride, and 16 times in case of the one stride). Higher values of s_x, s_y reduce the computational cost but can produce bad non-smooth reconstructions by introducing an undesired checker effect. Therefore, our choice of $s_x = s_y = 4$ responds to a trade-off between good reconstruction and computational cost.

For a 2D FP image, let κ_d be the number of patches computed over the dimensions $d = 1, 2$ (rows and columns). Then, κ_d is given by

$$\kappa_d = q_d + \varepsilon_d, \tag{20}$$

where

$$q_d = \left\lfloor \frac{H_d - h_d}{s_d} \right\rfloor + 1, \tag{21}$$

where H_d is the image size, h_d the patch size, s_d the stride step, and

$$\varepsilon_d = \begin{cases} 1 & H_d - (q_d - 1)s_d - h_d > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

In the case of the 1024×1024 dataset, we set $H_d = 1024$, $h_d = 32$, $s_d = 4$ for $d = 1, 2$. Then, the number of patches required to reconstruct a single FP is $\kappa_1 \times \kappa_2 = 249 \times 249 = 62,001$. This quantity is substantially larger than the number of patches in the training set, 25,000 patches. The expected number of patches per training FP was 833 (25,000/30). We used Monte Carlo simulations to estimate the covered area by the selected patches: on average, it was 53% of each entire FP. If we modify the number of training patches to 40,000, the average covered area would be 71% of each FP.

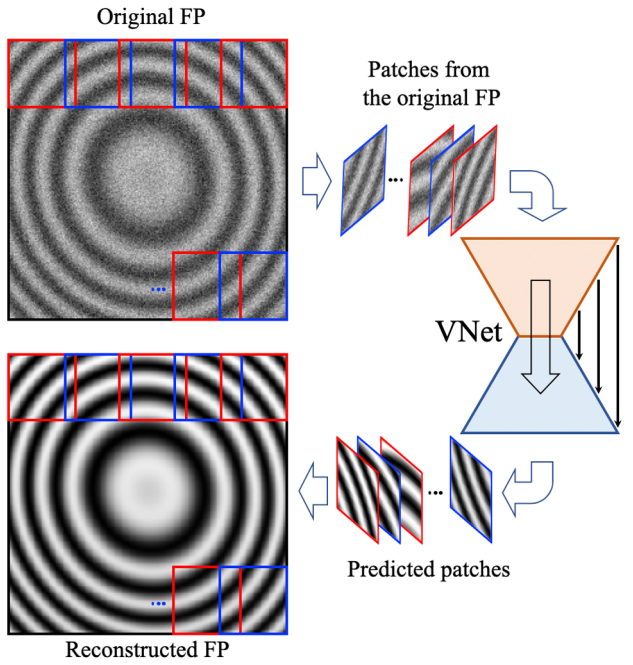

Fig. 7. FP normalization (inference). A set of overlapped patches that cover the entire FP to normalize is computed and used to feed our trained V-net model; the predicted patches are assembled to reconstruct the FP original. Pixels with multiple predictions (because of the patches' overlapping) are averaged for computing the normalized (reconstructed) FP.

D. Fast V-Net Implementation

Finally, a second modification of our V-net implementation is proposed. This variant preserves the same architecture as the V-net described in Fig. 2 and Table 3, but modifies the shape of the input tensor.

The modification is as follows: (1) after the V-net is trained, we directly clone our V-net model, with all hyper-parameters maintained the same, and all learned filters (or weight parameters) remaining equal. We just duplicate our model, now setting the input tensor shape as equal to the FP image dimensions (1024×1024 or 320×320, depending the experiment scenario). (2) Now we transfer all weights, layer by layer, from the original V-net to this new clone. The obtained clone version is what we call here the "fast V-net."

Now the reconstruction or inference process is done by passing the entire corrupted FP to the network, instead of the set of 32×32 covering patches (Fig. 8). That is, we generate the normalized FP using one patch only of the same size as the image: the entire FP. As a result, we obtain a similar quality of reconstruction, but now the computational time needed to process a single FP reduces to a fraction of the time consumed by the

Table 6. Fast V-Net Architecture (for Inference, Images of Size 320 × 320)

Block/Layer	Output Shape	# Params	Inputs
01. Input layer	(None,320,320,1)	0	
02. DownBlock(320,320,256,0)	(None,160,160,256)	592640	01
03. DownBlock(160,160,128,0.25)	(None,80,80,128)	442624	02
04. DownBlock(80,80,64,0.25)	(None,40,40,64)	110720	03
05. DownBlock(40,40,32,0.25)	(None,20,20,32)	27712	04
06. UpBlock(20,20,32)	(None,40,40,32)	592640	05, 04
07. UpBlock(40,40,64)	(None,80,80,64)	592640	06, 03
08. UpBlock(80,80,128)	(None,160,160,128)	592640	07, 02
09. UpBlock(160,160,256)	(None,320,320,256)	592640	08, 01
10. Tail Conv2D	(None,320,320,2)	578	09
11. Tail Conv2D	(None,320,320,1)	3	10
Total params		3,721,669	
Trainable params		3,721,669	
Non-trainable params		0	

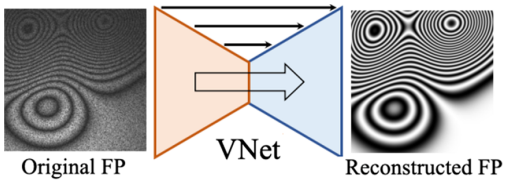

Fig. 8. One-patch fast V-net normalization (inference).

V-net. The reduction is about 100 times faster than the original V-net implementation. Table 6 summarizes the architecture of the fast V-net. It is identical to the architecture in Table 3, only that all output shapes are re-scaled according to the input shape (e.g., ×10 in the case of a 320 × 320 input image).

4. EXPERIMENTS

To evaluate the performance of the U-net and the proposed V-net models for the FP normalization task, we conducted three experiments. In the first one, we evaluated the U-net and V-net performance with respect to the noise level (assuming Gaussian noise). In the second experiment, we evaluated such models under different noise distributions: Gaussian, salt–pepper, speckle, and a combination of noise and the effect of incomplete FOV. Finally, the third experiment compares our proposals with methods of the state of the art, in normal scenarios, and when high-level and extreme-level noise is added to the FPs.

For all the evaluated networks, we set parameters equally for the training process. We used the ADAM algorithm [27] as the optimizer with a learning rate 1×10^{-4}, decay rate 1×10^{-3}, and batch size equal to 32, and we selected the best trained model over 300 epochs.

A. Performance Comparison of U-Net and V-Net for Different Noise Levels

In this experiment, we simulated noise levels as in the acquisition of typical interferometric FPs. We investigated the performance of the U-net and V-net models for seven levels of Gaussian noise, i.e., seven standard deviations σ for the noise η in Eq. (1). Such σ values are indicated in the first column in Table 7. For each trained model, we used a randomly generated training set and a randomly generated initial starting point for the models' parameters (weights). Table 7 reports the averaged mean absolute error (MAE) of the reconstructions over 10 different trained models.

Table 7. Summary of Synthetic Experiments (Full Images)[a]

Standard Deviation (σ with a, b Variable)	U-Net MAE ($\times 10^{-4}$)	V-Net MAE ($\times 10^{-4}$)
0.00	**2.142**	2.266
0.05	**2.016**	2.383
0.10	**2.416**	2.457
0.15	2.552	**2.539**
0.20	2.762	**2.620**
0.25	2.769	**2.726**
0.30	2.807	**2.702**

[a]FPs were generated using (1).

Table 7 shows that, in general, V-net performs better than U-net for denoising FPs corrupted with Gaussian noise. According to Fig. 9, the V-net model has a superior performance for higher standard deviation values ($\sigma > 0.15$ with a signal's dynamic range into the interval [0, 2]). Both models have a similar performance for σ close to 0.1. On the other hand, U-net produces better reconstructions for low noise levels ($\sigma \leq 0.05$). Figure 10 shows examples of FPs normalized with our method (noise with $\sigma = 0.15$). In general, V-net presents a lower error variance, which is understood as a better precision of the results.

B. Performance Comparison of U-Net and V-Net for Different Noise Types

The following experiment reports the U-net and V-net models' performance for the normalization of FPs under different noise distributions; in all cases, the illumination components (a, b) and the phase ϕ were generated according to the method presented in Section 5. The region of the interferogram was defined with a centered circular mask of diameter equal to 80% of the image size.

Table 8 and Fig. 11 report results for corrupted FPs under different scenarios: salt–pepper noise, speckle noise, Gaussian–speckle noise, and an incomplete FOV (region). Note that V-net

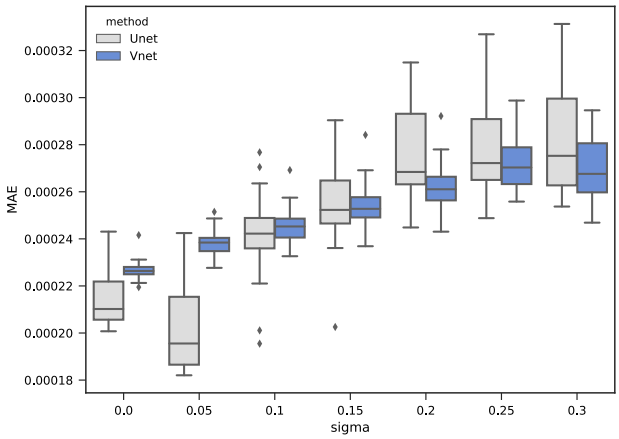

Fig. 9. Summary of experiments for normalizing 46 FPs corrupted with Gaussian noise and different levels of σ.

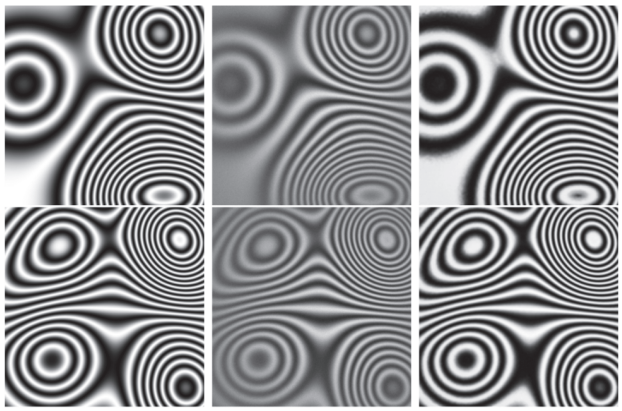

Fig. 10. Denoised–normalized FPs with V-net: (a) ground truth; (b) corrupted FPs, x; (c) reconstructions, $\hat{y}$.

Table 8. Summary of Synthetic Experiments (Patches)[a]

Noise (η with a, b variable)	U-net MAE ($\times 10^{-4}$)	V-net MAE ($\times 10^{-4}$)
$\eta = 0$ (no noise)	**2.142**	2.266
Salt–pepper	**2.416**	2.455
Speckle	2.552	**2.539**
Gaussian–speckle	2.762	**2.620**
Gaussian–speckle region	2.769	**2.726**

[a]Speckle FPs were generated using Eq. (19).

produces better results for speckle noise, Gaussian–speckle noise, and the interferogram with the region of interest. In contrast, U-net has a better performance when the task requires processing data with few intensity levels: as in the reconstruction of FP corrupted with salt–pepper noise (to remove a few data and to interpolate such pixels) or if only low-frequency illumination changes are present and noise is not a problem.

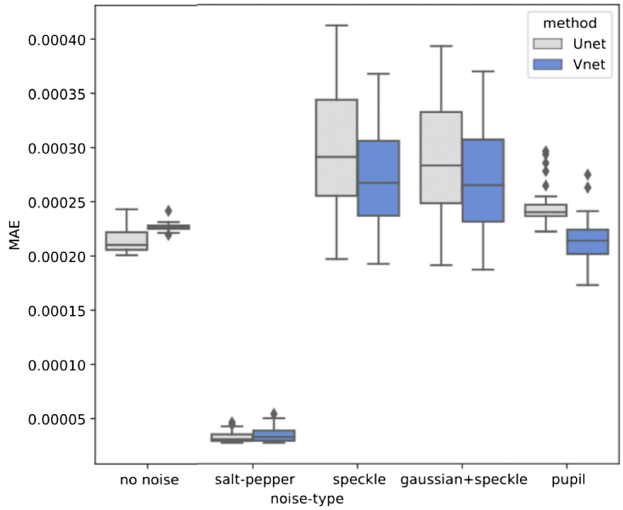

Fig. 11. Summary of experiments for normalizing 46 FPs corrupted with different noise distributions.

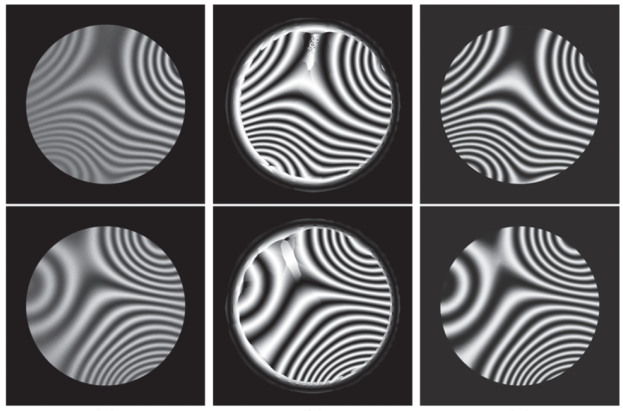

Fig. 12. Normalized FPs. (a) Data generated with Gaussian and speckle noise; (b) GFB based normalization; (c) our results.

C. Comparison Versus State-of-the-Art Methods

In this subsection, we evaluate the performance of the proposed models U-net, V-net, and ResV-net versus other methods of state of the art based on DNNs.

We compared our proposed models, with recently reported DNNs: the optical FPD CNN proposed in Ref. [28], deep CNN (DCNN) [29], and the application reported in Ref. [30] of the general purpose image denoising DNN (FFD) [31].

References [28,29] present favorable comparisons of their networks with respect to a filtering based on the WFT [11,32]. The authors argue that they chose WFT since it is one of the classical procedures with better performance for fringe denoising. We compare our proposals with the particular case of WFT: the GFB in Ref. [15] is reported the relationship between GFB and WFT. The results of our comparisons are consistent with those reported in Ref. [29]: the GFB method fails to reconstruct the FP in regions with phase discontinuities and low frequency. Figure 12 depicts evidence that supports this claim.

In this experiment, we evaluated the performance of our models (U-net, V-net, and Res V-net) and the recent methods

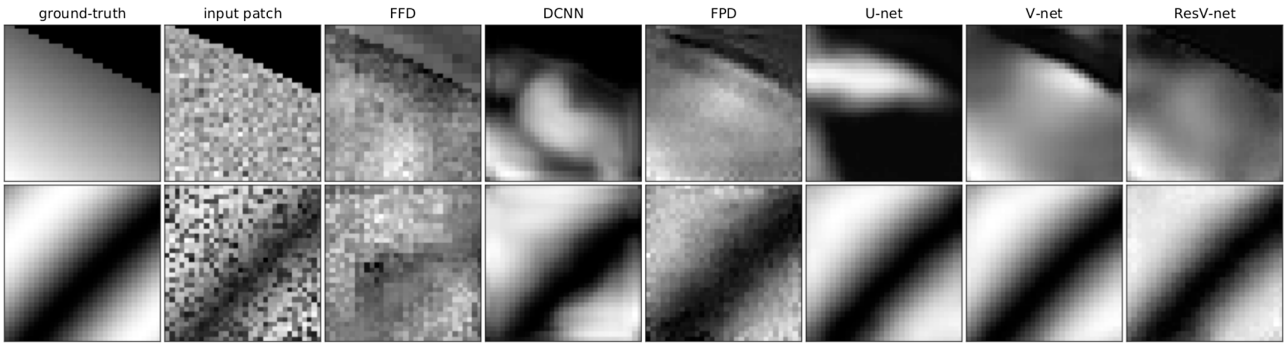

Fig. 13. Results of the compared deep neural network models: normalized patches.

Table 9. Averaged Errors over Training FPs (Accuracy)

Model	MSE x 10^{-2}	MAE x 10^{-1}	PSNR x 10^{1}
x (input)	6.644	1.410	1.182
FFD	5.008	1.938	1.302
DCNN	4.048	1.047	1.416
FPD	4.166	1.475	1.422
U-net	0.846	**0.587**	2.127
V-net	**0.764**	0.655	**2.147**
ResV-net	0.824	0.788	2.087

Table 10. Averaged Errors over Test FPs (Accuracy)

Model	MSE x 10^{-2}	MAE x 10^{-1}	PSNR x 10^{1}
x (input)	6.501	1.393	1.194
FFD	5.251	2.012	1.283
DCNN	3.797	1.004	1.434
FPD	3.983	1.506	1.426
U-net	0.807	**0.585**	**2.138**
V-net	**0.765**	0.681	**2.138**
ResV-net	0.812	0.787	2.090

reported in Refs. [28,30,31]. We evaluated all the methods in the task of normalizing FPs corrupted with Gaussian noise, speckle noise, illumination component variations, and incomplete FOV. The training was conducted using 25,000 patches for our models and 40,000 patches for the compared methods. The training set (patches) was randomly sampled over 30 of the total 46 FPs. The remaining 16 FPs constitute the test set. For all methods, we used the same training parameters as described at the beginning of this section.

Tables 9 and 10 summarize the experimental results. Table 9 shows the averaged mean square error (MSE), averaged MAE, and averaged peak-signal-to-noise ratio (PSNR) over the full reconstructions of the 30 FPs used to generate the training set. Since we used patches for training, the full FPs were never seen for the networks. Table 10 shows the averaged errors for the reconstructed 16 FPs used to generate the test set.

Finally, Figure 13 shows examples of reconstructed patches. Figure 14 shows examples of reconstructed full FPs. The procedure for reconstructing complete FPs was described in Section 3.C. From a visual inspection of Figures 13 and 14, one can note that the proposed networks produce better results.

D. High-Level and Extreme Noise

In this experiment, we evaluated again the performance of the U-net, V-net, ResV-net, and fast V-net models versus the methods reported in Refs. [28,30,31]. We evaluated all the methods in the task of normalizing FPs corrupted with high levels of Gaussian and also correlated noise.

We used the second dataset of 180 images of size 320 × 320 for comparison purposes. In all models, the training set was the same, and we maintained all hyper-parameters common to all models: ADAM optimizer, step size, or learning rate of α in the range of $[1 \times 10^{-4}, 5 \times 10^{-4}]$, decay rate $= 10^{-3}$, and batch size $= 32$. All models were trained (ours and the reported DCNN, FFD, and FPD) from zero using 40,000 training patches of size 32 × 32, taken from the first 150 images. The remaining 30 images were used for testing.

All times correspond to the same computational resources. For the GFB method, we used a Python convolutional CPU implementation on an i5 Intel 3.7 GHz and executed the process in one core. We used six filters with periods 7, 10, 15, 20, 30, and 40 and 10 frequencies, giving a 60 bank filter. For the HHT method, we used the MATLAB code provided in Ref. [33]. In this experiment, we set the HHT parameters as *method* $= 2$, *factor1* $= 1.5$, and *factor2* $= 0.2$. For all the NN models, the training process was done with a Python–Tensorflow–Keras implementation, on a GPU NVIDIA GeForce 1080 Ti, and the process was executed using NVIDIA 440 drivers and CUDA 10.2 installation. The testing process was a mixture between the i5 Intel (collecting and output of all covering patches) and the GPU (model inference). Table 11 summarizes the computational time required for all models, in both the training stage and the inference process.

Tables 12 and 13 summarize the experimental results. Table 12 shows the averaged MSE and averaged MAE over the full reconstructions of the 150 training and 30 testing FPs, for the speckle and high-level Gaussian noise scenario. Similarly, Table 13 summarizes the experimental results for the speckle and extreme-level Gaussian noise scenario.

Figures 15 and 16 show examples of reconstructed full FPs, in the scenario in which the FPs present speckle and extreme-level Gaussian noise. As one can observe, again the proposed models generate better normalized reconstructions. Please refer to Supplement 1 for more details on the level of noise included in the synthetic data. Figures S1–S6 correspond to high-level

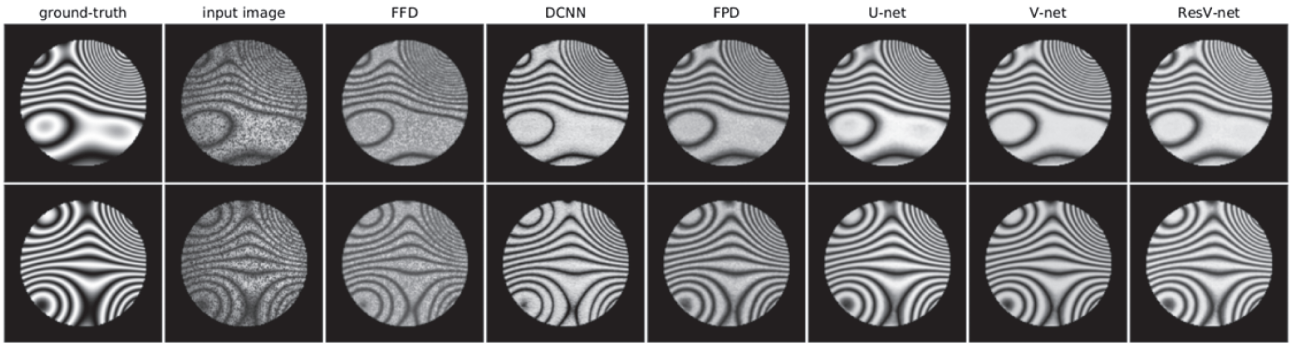

Fig. 14. Results of the compared deep neural network models: normalized complete FPs.

Table 11. Computation Cost (Time) of Evaluated Models

Model	Training	Inference (per image)	Inference Efficiency
FFD	40–50 min	2.2 s	27 img/min
DCNN	30–35 min	2.9 s	20 img/min
FPD	140–150 min	10.1 s	6 img/min
U-net	13–15 min	2.5 s	24 img/min
V-net	40–50 min	6.1 s	10 img/min
ResV-net	50–60 min	6.0 s	10 img/min
Fast V-net	—	0.024 s	2500 img/min
GFB	—	60 s	1.0 img/min
HHT	—	5.0 s	12 img/min

Table 12. Averaged Errors over Training and Testing FPs, High-Noise Level

Model	MSE Train	MSE Test	MAE Train	MAE Test
FFD	1723.73	1747.28	26.7057	26.9246
DCNN	252.45	253.71	9.1418	9.1461
FPD	129.10	145.03	5.2429	5.4363
U-net	156.37	159.84	7.4497	7.3633
V-net	**79.91**	**97.88**	**4.6833**	**4.8458**
ResV-net	86.54	104.92	4.8618	5.1471
Fast V-net	104.44	110.95	5.7322	5.7431
GFB	1642.57	1566.03	15.5754	15.095
HHT	1.13×10^9	1.12×10^9	22436	22282

Table 13. Averaged Errors over Training and Testing FPs, Extreme-Noise Level

Model	MSE Train	MSE Test	MAE Train	MAE Test
FFD	2215.80	2248.44	29.8887	30.0743
DCNN	1009.00	1038.59	18.6041	18.6464
FPD	7839.25	841.23	13.1928	13.6363
U-net	396.58	530.21	9.5679	10.4250
V-net	319.36	474.76	8.8652	10.0145
ResV-net	267.35	461.09	**8.1307**	9.8830
Fast V-net	**248.04**	**390.51**	8.2544	**9.6987**
GFB	3691.63	3748.59	27.698	27.937
HHT	1.13×10^9	1.12×10^9	22436	22282

noise, and Figs. S7–S12 correspond to the extreme-level scenario. Figures S13–S18 show a comparison among all methods in the extreme-level case.

E. Evaluation on Real FPs

U-net, V-net, ResV-net, and fast V-net can be used to process real FPs, even when they were trained with synthetic data. As an example, Fig. 17 depicts a one-shot real ESPI image, and Figs. 18 and 19 show the results of normalizing the left section of the real interferogram by using the evaluated methods. We use the same models that were trained with simulated data in this experiment. The performance of all the evaluated methods is consistent with the obtained results when processing synthetic FPs.

5. DISCUSSION AND CONCLUSION

A. Method Limitations

U-net, V-net, and ResV-net models base the FP reconstruction on processing image patches with local information. Despite this being an advantage in terms of computational efficiency, there are some limitations for solving FP analysis problems associated with global information. To illustrate this point, we trained a V-net to estimate the quadrature normalized FP. Mathematically, we trained a V-net to estimate an operator $\mathcal{H}$ that, given observations modeled by Eq. (1), produces normalized FPs according to

$$\hat{x}(p) = 1 + \sin(\phi(p)). \tag{23}$$

Figure 20 shows a reconstructed quadrature FP where the "global sign" problem is evident. The problem actually occurs at patch level. There are patches for which the network cannot infer the correct sign of the sine function (see the reconstructed patch in the last row in Fig. 21). However, this sign change does not appear in arbitrary orientations; it seems that there exists a principal axis in the orientation domain where the sign changes systematically appear.

We also observed that U-net, V-net, and ResV-net models have limitations for filtering out noise in regions with very low frequencies and low contrast, i.e., in regions with low SNR. Those are regions where a visual inspection does not suggest a clear local dominant frequency. Moreover, we observed that ResV-net cannot completely remove noise, as shown in Section 4.C. An explanation for this behavior it that the ResV-net model assumes additive noise while our experimental data contain correlated noise (speckle).

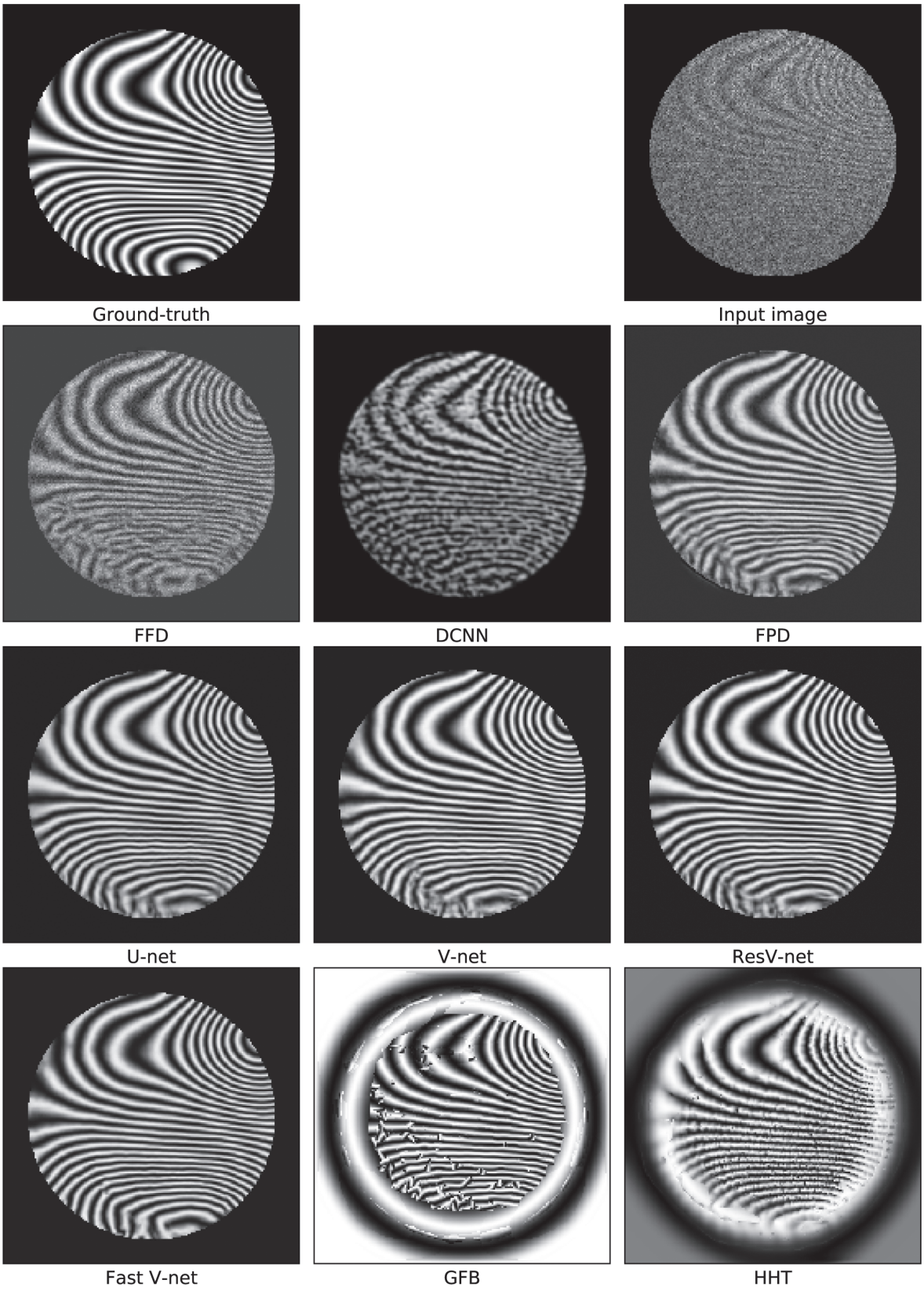

Fig. 15. Results of the compared methods: normalized FPs for speckle and extreme-level Gaussian noise.

On the other hand, the fast V-net model produces better results in extreme noise conditions, as noted in Table 13 and Fig. 16. It can normalize FPs with good results also in regions of high frequency, even in the boundary of the Nyquist phenomenon. Again, bad contrast and bad frequency regions bound the quality of the reconstruction.

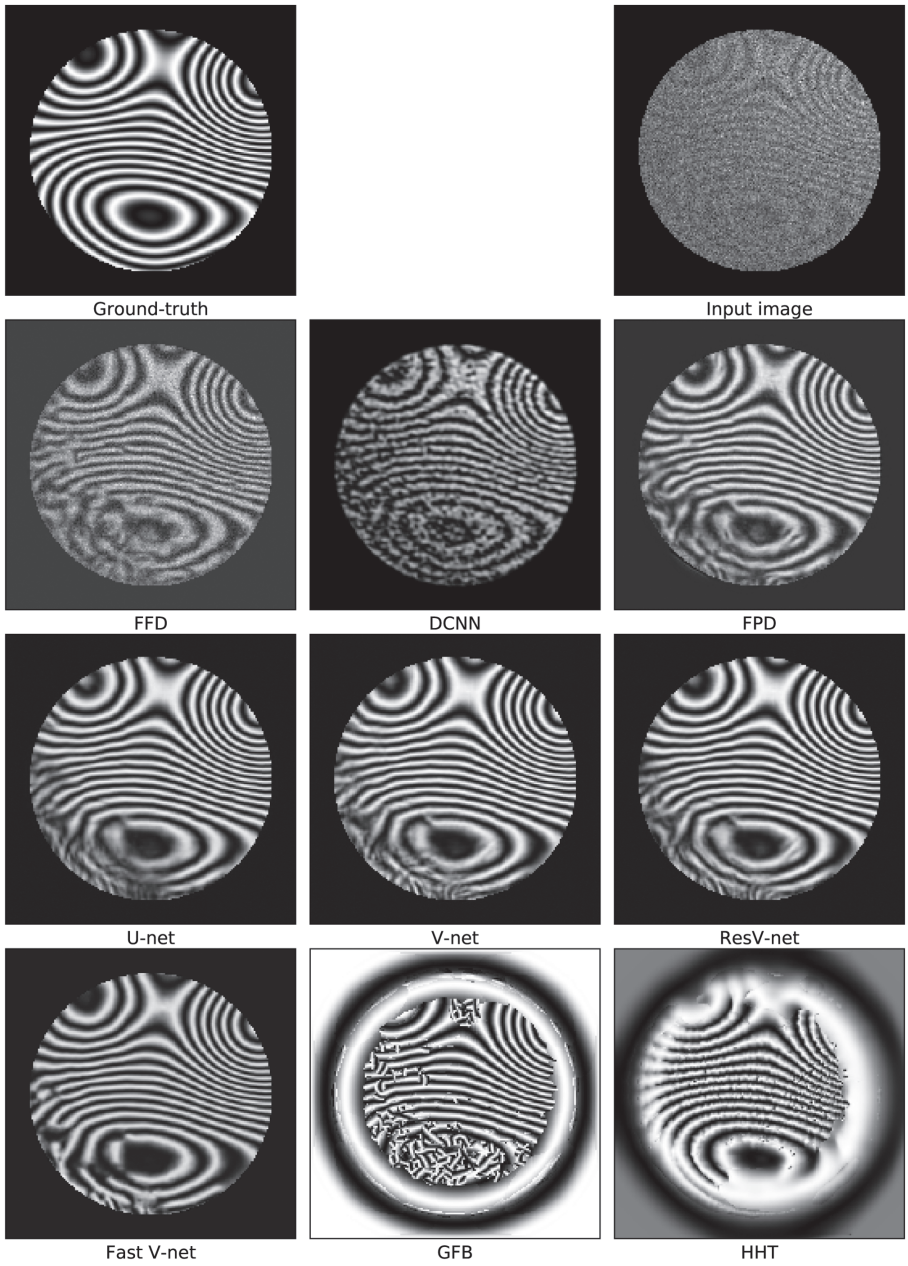

Fig. 16. Results of the compared methods: normalized FPs for speckle and extreme-level Gaussian noise.

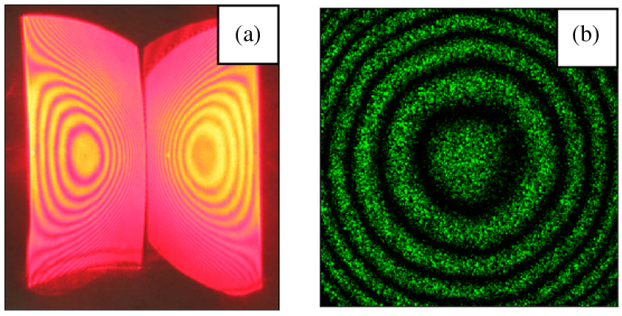

Fig. 17. Real ESPI images of a one-shot setup with a phase step equal π. (a) Defocused and (b) focused.

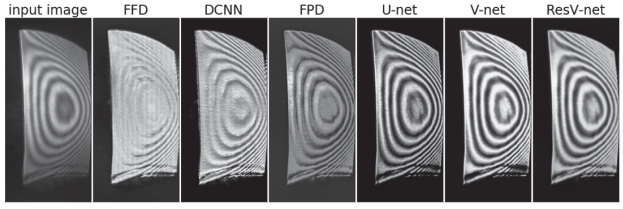

Fig. 18. Results of the compared models with real interferometric FP in Fig. 17(a).

B. Conclusion

The proposed normalization–denoising method for FPs is based on the deep learning paradigm. Under this paradigm, one trains a NN (universal approximator) that estimates an appropriated

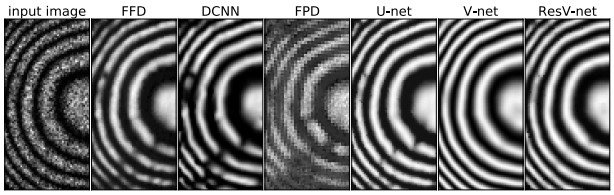

Fig. 19. Results of the compared models with real interferometric FP in Fig. 17(b).

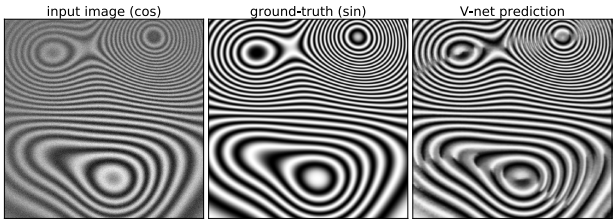

Fig. 20. Computation of the normalized signal in quadrature. Note the global sign problem.

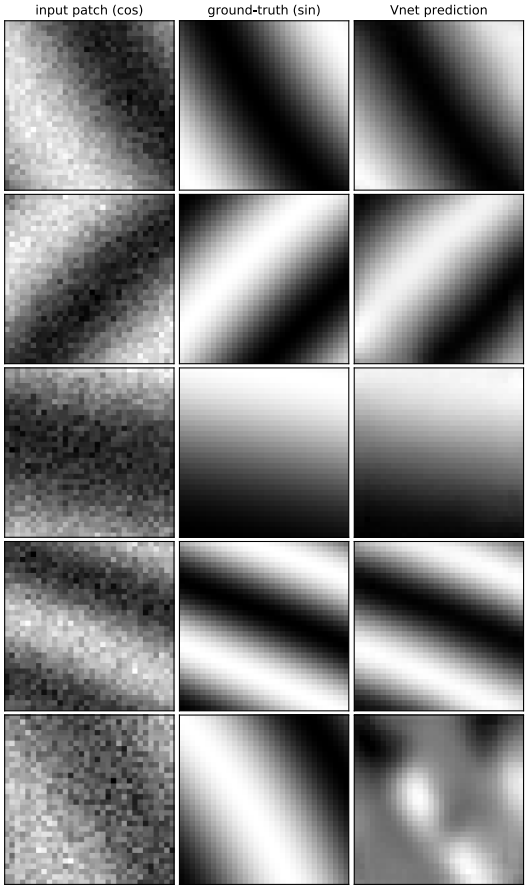

Fig. 21. Computation of the normalized signal in quadrature at patch level. Note the sign problem in the patch in last row.

transformation between observations (corrupted inputs) and outputs. In particular, our solution builds upon a deep autoencoder that produces results with very small errors. Our results show that the proposed U-net and V-net schemes can be applied to real FP images, in which the image's corruption irregularities correspond to high noise levels, illumination component variations, or an incomplete FOV (Figs. 12 and 14). Our models can process real FPs, even if we train the networks with simulated data.

We observed that DNNs can compute reconstructions with fewer errors than GFBs near the boundary of the FOV (Fig. 12). We found that V-net produces higher quality reconstruction for FPs for higher levels of noise and fringes of incomplete FOV than U-net and the compared methods. In our opinion, the reason is that the V-net filter distribution across the layers is designed to retain more details, as opposed to U-net, which is designed to segment images.

We believe that the evaluated methods are a new research branch for developing more sophisticated FP analysis methods that, based on DNNs, can compute solutions in a front-to-end strategy. It could be interesting to design and implement specific deep network architectures for solving challenging problems in FP analysis, e.g., the phase recovery from a single interferogram with closed fringes and quadrature FP computation.

Funding. Nvidia (Nvidia Academic Program); Consejo Nacional de Ciencia y Tecnología (CONACYT, Mexico) (Grant A1-S-43858, Postdoctoral Grant VHF, PhD Scholarship Grant ARF).

Disclosures. MR: Nvidia Academic Program (F).

Supplemental document. See Supplement 1 for supporting content.

REFERENCES

1. J. A. Quiroga and M. Servín, "Isotropic n-dimensional fringe pattern normalization," Opt. Commun. **224**, 221–227 (2003).
2. J. A. Quiroga, J. A. Gómez-Pedrero, and A. García-Botella, "Algorithm for fringe pattern normalization," Opt. Commun. **197**, 43–51 (2001).
3. J. L. Marroquín, M. Servín, and R. Rodriguez-Vera, "Adaptive quadrature filters and the recovery of phase from fringe pattern images," J. Opt. Soc. Am. A **14**, 1742–1753 (1997).
4. M. Servín, J. L. Marroquín, D. Malacara, and F. Cuevas, "Phase unwrapping with a regularized phase-tracking system," Appl. Opt. **37**, 1917–1923 (1998).
5. M. Servín, J. L. Marroquín, and F. J. Cuevas, "Fringe-follower regularized phase tracker for demodulation of closed-fringe interferograms," J. Opt. Soc. Am. A **18**, 689–695 (2001).
6. M. Rivera, "Robust phase demodulation of interferograms with open or closed fringes," J. Opt. Soc. Am. A **22**, 1170–1175 (2005).
7. V. H. Flores, A. Reyes-Figueroa, C. Carrillo-Delgado, and M. Rivera, "Two-step phase shifting algorithms: where are we?" Opt. Laser Technol. **126**, 106105 (2020).
8. S. S. Gorthi and P. Rastogi, "Fringe projection techniques: whither we are?" Opt. Lasers Eng. **48**, 133–140 (2010).
9. R. Juarez-Salazar, F. Guerrero-Sanchez, and C. Robledo-Sanchez, "Theory and algorithms of an efficient fringe analysis technology for automatic measurement applications," Appl. Opt. **54**, 5364–5374 (2015).
10. Q. Kemao, "Two-dimensional windowed Fourier transform for fringe pattern analysis: principles, applications and implementations," Opt. Lasers Eng. **45**, 304–317 (2007).
11. L. Huang, Q. Kemao, B. Pan, and A. K. Asundi, "Comparison of Fourier transform, windowed Fourier transform, and wavelet transform methods for phase extraction from a single fringe pattern in fringe projection profilometry," Opt. Lasers Eng. **48**, 141–148 (2010).
12. Z. Zhang, Z. Jing, Z. Wang, and D. Kuang, "Comparison of Fourier transform, windowed Fourier transform, and wavelet transform methods for phase calculation at discontinuities in fringe projection profilometry," Opt. Lasers Eng. **50**, 1152–1160 (2012).

13. M. Rivera, O. Dalmau, A. Gonzalez, and F. Hernandez-Lopez, "Two-step fringe pattern analysis with a Gabor filter bank," Opt. Lasers Eng. **85**, 29–37 (2016).
14. O. Dalmau, M. Rivera, and A. Gonzalez, "Phase shift estimation in interferograms with unknown phase step," Opt. Commun. **372**, 37–43 (2016).
15. M. Rivera, "Robust fringe pattern analysis method for transient phenomena," Opt. Lasers Eng. **108**, 19–27 (2018).
16. V. Kůrková, "Kolmogorov's theorem and multilayer neural networks," Neural Netw. **5**, 501–506 (1992).
17. F. Cuevas, M. Servin, O. Stavroudis, and R. Rodriguez-Vera, "Multilayer neural network applied to phase and depth recovery from fringe patterns," Opt. Commun. **181**, 239–259 (2000).
18. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science **313**, 504–507 (2006).
19. V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *27th International Conference on Machine Learning (ICML)* (2010), pp. 807–814.
20. O. Ronneberger, P. Fischer, and T. Brox, "U-net: convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds. (Springer, 2015), pp. 234–241.
21. J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR) (2015), pp. 3431–3440.
22. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.
23. O. I. Renteria-Vidales, J. C. Cuevas-Tello, A. Reyes-Figueroa, and M. Rivera, "ModuleNet: a convolutional neural network for stereo vision," in *Mexican Conference on Pattern Recognition (MCPR)*, Lecture Notes in Computer Science 12088 (Springer, 2020), 219–228.
24. A. Rotenberg, "A new pseudo-random number generator," J. ACM **7**, 75–77 (1960).
25. D. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," Complex Syst. **2**, 321–355 (1988).
26. R. Jones and C. Wykes, *Holographic and Speckle Interferometry*, 2nd ed. (Cambridge University, 1987).
27. D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," *3rd International Conference for Learning Representations*, San Diego, California, USA (2014).
28. B. Lin, S. Fu, C. Zhang, F. Wang, and Y. Li, "Optical fringe patterns filtering based on multi-stage convolution neural network," Opt. Lasers Eng. **126**, 105853 (2020).
29. K. Yan, Y. Yu, C. Huang, L. Sui, K. Qian, and A. Asundi, "Fringe pattern denoising based on deep learning," Opt. Commun. **437**, 148–152 (2019).
30. F. Hao, C. Tang, M. Xu, and Z. Lei, "Batch denoising of ESPI fringe patterns based on convolutional neural network," Appl. Opt. **58**, 3338–3346 (2019).
31. K. Zhang, W. Zuo, and L. Zhang, "FFDNet: toward a fast and flexible solution for CNN-based image denoising," IEEE Trans. Image Process. **27**, 4608–4622 (2018).
32. Q. Kemao, "Windowed Fourier transform for fringe pattern analysis," Appl. Opt. **43**, 2695–2702 (2004).
33. M. Trusiak, M. Wielgus, and K. Patorski, "Advanced processing of optical fringe patterns by automated selective reconstruction and enhanced fast empirical mode decomposition," Opt. Lasers Eng. **52**, 230–240 (2014).