Deep neural networks for classifying complex features in diffraction images

Julian Zimmermann,[1,*] Bruno Langbehn,[2] Riccardo Cucini,[3] Michele Di Fraia,[3,4] Paola Finetti,[3] Aaron C. LaForge,[5]
Toshiyuki Nishiyama,[6] Yevheniy Ovcharenko,[2,7] Paolo Piseri,[8] Oksana Plekan,[3] Kevin C. Prince,[3,9]
Frank Stienkemeier,[5] Kiyoshi Ueda,[10] Carlo Callegari,[3,4] Thomas Möller,[2] and Daniela Rupp[1]

[1]*Max-Born-Institut für Nichtlineare Optik und Kurzzeitspektroskopie, 12489 Berlin, Germany*
[2]*Institut für Optik und Atomare Physik, Technische Universität Berlin, 10623 Berlin, Germany*
[3]*Elettra-Sincrotrone Trieste S.C.p.A., 34149 Trieste, Italy*
[4]*ISM-CNR, Istituto di Struttura della Materia, LD2 Unit, 34149 Trieste, Italy*
[5]*Institute of Physics, University of Freiburg, 79104 Freiburg, Germany*
[6]*Division of Physics and Astronomy, Graduate School of Science, Kyoto University, Kyoto 606-8502, Japan*
[7]*European XFEL GmbH, 22869 Schenefeld, Germany*
[8]*CIMAINA and Dipartimento di Fisica, University degli Studi di Milano, 20133 Milano, Italy*
[9]*Department of Chemistry and Biotechnology, Swinburne University of Technology, Victoria 3122, Australia*
[10]*Institute of Multidisciplinary Research for Advanced Materials, Tohoku University, Sendai 980-8577, Japan*

Intense short-wavelength pulses from free-electron lasers and high-harmonic-generation sources enable diffractive imaging of individual nanosized objects with a single x-ray laser shot. The enormous data sets with up to several million diffraction patterns present a severe problem for data analysis because of the high dimensionality of imaging data. Feature recognition and selection is a crucial step to reduce the dimensionality. Usually, custom-made algorithms are developed at a considerable effort to approximate the particular features connected to an individual specimen, but because they face different experimental conditions, these approaches do not generalize well. On the other hand, deep neural networks are the principal instrument for today's revolution in automated image recognition, a development that has not been adapted to its full potential for data analysis in science. We recently published [Langbehn *et al.*, Phys. Rev. Lett. **121**, 255301 (2018)] the application of a deep neural network as a feature extractor for wide-angle diffraction images of helium nanodroplets. Here we present the setup, our modifications, and the training process of the deep neural network for diffraction image classification and its systematic bench marking. We find that deep neural networks significantly outperform previous attempts for sorting and classifying complex diffraction patterns and are a significant improvement for the much-needed assistance during postprocessing of large amounts of experimental coherent diffraction imaging data.

I. INTRODUCTION

Coherent diffraction imaging (CDI) experiments of single particles in free flight have been proven to be a significant asset in the pursuit of understanding the structural composition of nanoscaled matter [1–6]. While traditional microscopy methods are able to image fixated, substrate-grown, or deposited individual particles [7–11], only CDI can combine high-resolution images with single particles in free flight in one experiment [12–14]. CDI became possible because of the recent advent of short-wavelength free-electron lasers (FELs) that produce coherent high-intensity x-ray pulses with

*julian.zimmermann@mbi-berlin.de

femtosecond duration with a single x-ray laser shot [15]. However, CDI also comes with its own set of new challenges.

One of the growing problems of CDI experiments is the sheer amount of recorded data that has to be analyzed. The LINAC Coherent Light Source (LCLS), for instance, has a repetition rate of 120 Hz with typical hit rates ranging from 1% to 30% [15–17], greatly depending on the performed experiment. The newly opened European XFEL will have an even higher maximum repetition rate of 27 000 Hz [18], which may add up to several million diffraction patterns in a single 12-h shift. The idea of using neural networks for classification of a large number of scattering patterns was born out of the significant difficulties of analyzing large data sets of clusters [19], in particular, metal clusters [20]. Moreover, the ability to analyze such data sets is sought after by the community in general [21]. For example, for the successful determination of three-dimensional (3D) structures from a CDI data set using the expansion-maximization-compression algorithm [21–23], it is necessary to sample the 3D Fourier space up to the Nyquist rate for the desired resolution, for all subspecies contained in the target under study. The achievable

resolution, as well as the chance for successful convergence of the algorithm, correlates directly with the number of diffraction patterns with a high signal-to-noise ratio [22]. Thus, huge data sets are taken and as a consequence of the sheer amount of data, it is getting increasingly complicated to distill the high-quality data subsets that are suitable for subsequent analytical steps.

The enormous success of neural networks in the regime of image processing and classification provides a unique way of facing the imminent data-analysis bottleneck and reduces the impending problem to a mere domain adaptation from data sets used throughout the industry to ones that are used in CDI research. This work aims to be a stepping stone toward this adaptation by providing an introduction to the theory of deep neural networks and analyzing how to best transfer and optimize these algorithms to the domain of scattering images. As a baseline, we train a widely used deep neural network architecture, a residual convolutional deep neural network [24], in a supervised manner with a training set of manually labeled data. We then adapt the neural network to the domain of diffraction images and improve on the baseline performance by addressing the following issues:

(1) Modification of the architecture to account for the specificities of diffraction images and thus optimization of the prediction capabilities.

(2) Determination of the appropriate size of the training data set in order to keep the manual work of a researcher to a moderate level.

(3) Mitigation of experimental artifacts, in particular noisy diffraction images.

Experience has shown that a researcher is able to relate diffraction patterns produced by similarly shaped particles of different sizes and orientations in context with each other. However, a programmatic description for a classification and sorting of these mostly similar patterns is almost impossible to achieve.

Figure 1 illustrates the case of two diffraction patterns captured from almost identical particles but under different orientations. Both patterns clearly show an elongated and bent streak, but the bending is differently pronounced and directed. If we wanted to handcraft an algorithm that detects this feature, we would need to describe it via some appropriate metric that must take into account the various grades of inflection, direction, brightness, and completeness of this feature within every image. Furthermore, we would need to redo it for every characteristic feature in a diffraction image for which we want to find similar ones.

In addition to that, poor signal-to-noise ratios, stray light, a beam stop, or central hole of multichannel plates or pnCCDs [25] and overall poor image quality can even further increase the difficulty to make an automatized classification of all images coherent [26–28].

Therefore, we need a robust classification routine that is insusceptible to the described artifacts, just as a researcher is, to tackle the upcoming data volume. Deep neural networks provide a way out of this situation, and we show in this paper that they outperform the current state-of-the-art classification and sorting routines.

Current state-of-the-art automatic classification routines for diffraction experiments employ so-called kernel
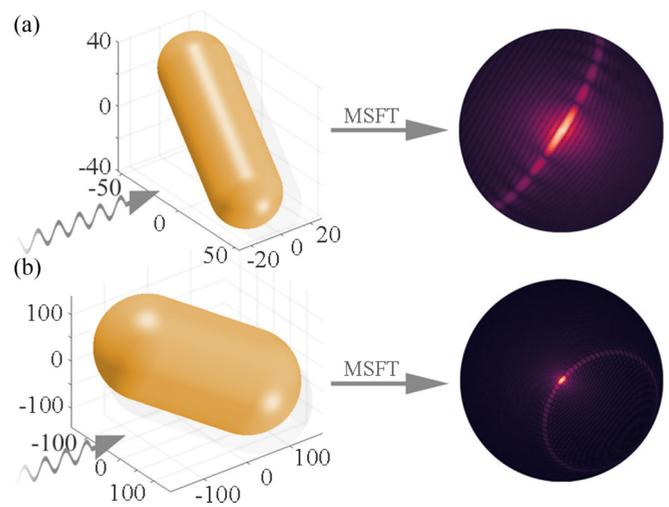
FIG. 1. Panels (a) and (b) show *capsule*-shaped particles whose orientation and size differs. The scattering images are calculated using a multislice Fourier transform (MSFT) algorithm that simulates a wide-angle x-ray scattering experiment which includes 3D information about the particle [6,20]. The two incoming beams (indicated by the arrow on the left-hand side) produce very different scattering images, yet the dominant feature, an elongated bent streak, is distinctly visible in both calculations. A handcrafted algorithm is typically not able to identify the similarity between the two scattering patterns and would classify these two images in two distinct classes, although they belong to the same *capsule* shape class. A deep neural network can learn these complicated similarities on its own when we provide a few manually selected diffraction patterns that contain this feature.

methods [27,29]. Bobkov *et al.* [27] trained a support-vector machine on a public small-angle x-ray scattering data set with an accuracy of 87%, but only on selected images (we will use this approach as a reference in Sec. IV). Yoon *et al.* [29] were able to achieve an accuracy of up to 90% using unsupervised spectral clustering on a nonpublic small-angle x-ray scattering data set.

Deep neural networks, on the other hand, have already been applied to a broad range of physics-related problems ranging from predicting topological ground states [30], distinguishing different topological phases of topological band insulators [31], enhancing the signal-to-noise at hadron colliders [32], differentiating between so-called known-physics background and new-physics signals at the Large Hadron Collider [33], and solving the Schrödinger equation [34,35]. Their ability to classify images has also been utilized in cryoelectron microscopy [36], medical imaging [37], and even for hit finding in serial x-ray crystallography [38]. We now have extended the use case of applying deep neural networks for classifying complex features within diffraction patterns. We show that deep neural networks outperform the current state-of-the-art classification and sorting routines, while being insusceptible to typical artifact features of diffraction measurements. Furthermore, a deeper analysis of the trained network shows that it can understand complex concepts of what constitutes a characteristic feature in a diffraction pattern.

The paper is organized as follows: In Sec. II, the data set is presented and a few experimental details are discussed. Section III provides the fundamental theory to understand the basics of neural networks; it has two subsections. Subsection III A covers the theory, algorithmic underpinnings of deep neural networks, and how to train these models, and Subsec. III B presents three common metrics to evaluate the quality of the neural network's predictions.

Section IV establishes our starting point, while the full benchmark report on the baseline neural network can be found in Appendix A. We introduce the chosen network architecture and provide baseline results on the data presented in Sec. II but also on a reference data set for which classification results are already published [39].

In Sec. V, we discuss solutions for the above stated issues of applying neural networks to diffraction data. In Subsec. V A, we discuss the choice of the activation function for the neural network and present a logarithmic activation function that enhances the prediction performance with diffraction image data. Subsection V B benchmarks the dependence of neural networks on training data size, asking essentially how much manually labeled data is needed for the neural network to give acceptable results, and Subsec. V C presents an approach to harden the neural network against very noisy data using a custom two-point cross-correlation map.

In Sec. VI, we then provide more profound insights into the output of the neural network by showing and discussing calculated heat maps that visualize the gradient flow within the neural network. These images directly correlate with what the neural network *sees*; they are created using an advanced visualization algorithm called GRADCAM++ [40].

Finally, we give a summary of the principal results and unique propositions of this paper and conclude with an outlook on further modifications as well as future directions.

II. THE DATA

Helium nanodroplets [41] were imaged using extreme ultraviolet (XUV) photon energies between 19 and 35 eV using the experimental setup of the LDM beamline [42,43] at the Free Electron Laser FERMI [44]. Scattering images were recorded with a multichannel-plate (MCP) detector combined with a phosphor screen, which was placed 65 mm downstream from the interaction region; this defines the maximum scattering angle of $30°$. Single-shot diffraction images in the XUV regime are in some respect a special case, as they cover large scattering angles and can contain 3D structural information [20], manifesting as complex and pronounced characteristic features, such as the bent streaks in Fig. 1. Out of 2×10^5 laser shots, about 38 000 images were obtained. The images were corrected for straylight background and the flat detector (see also Langbehn *et al.* [41]).

For the neural network training data set, we selected 7264 diffraction images randomly out of all recorded patterns. The size of the subset was chosen to be the maximum a researcher could classify manually within 1 week. From this subset, we manually identified 11 distinct but nonexclusive classes (see Fig. 2 for examples as well as a description and Table I for statistics about every class). We chose each of the diffraction patterns shown in Fig. 2 as a strong candidate for its class,

but it is important to note that almost all diffraction patterns belong to multiple classes since this is a multiclass labeling scenario. These patterns are therefore not always clearly distinguishable from each other and can exhibit multiple characteristics from different classes. For example, the Newton rings in Fig. 2(d) are superimposed on a concentric ring pattern that falls into the category *spherical/oblate*, but Newton rings can also occur in other classes, e.g., streak patterns. Furthermore, labeling all images is itself prone to systematic errors because the researcher has to learn to label [45]. This means that the labeling process itself is to some extent ill posed, as the researcher does not know the characteristics of a feature *a priori,* which results in a changing perception of features and classes along the labeling process and thus a systematically decreased consistency for every class.

We uploaded all available data alongside our assigned labels to the public CXI database (CXIDB [46]) under the public domain CC0 waiver.

III. BASIC THEORY

A. What is a deep neural network

We concentrate in this paper solely on deep feed-forward neural networks. They are a classification model consisting of a directed acyclic graph that defines a set of hierarchically structured nonlinear functions.

A fundamental example can be constructed by arranging n nonlinear functions (z_1, z_2, ..., z_n) in a chainlike manner: $z_{\text{output}} = z_n(z_{n-1}(\ldots(z_2(z_1(\boldsymbol{x})))\ldots))$, where $\boldsymbol{x}$ is the input, which is in our case a diffraction image. The first function, $z_1(\boldsymbol{x})$, is called the input layer. We then pass the output of z_1 to z_2 and so on; this goes on until the last layer (z_n), which is called the output layer. The nomenclature is that all layers except the output layer (z_n) and the input layer (z_1) are called hidden layers.

For illustrative purposes, Fig. 3 shows a convolutional neural network. There, we schematically show the layer functions z_1, ..., z_n where every layer consists of two stages, a linear layer-specific operation on its inputs followed by a so-called activation function, which is always nonlinear. We address the choice of layer-specific operations in Sec. III A 1 and then introduce the activation functions in Sec. III A 2. In general, the layer-specific operation is always the name-giving component for the layer, so, for example, if we compute a two-dimensional (2D) convolution as the layer-specific operation on the input and then apply an activation function, we call the set of these two stages a *convolutional layer*. Figure 3 shows a neural network whose first layers are convolutional layers followed by a fully connected layer that produces the predictions.

1. Affine transformations

All common choices for layer-specific operations are affine transformations. They all introduce trainable weights; free parameters that are adjustable during the training process and are sometimes called neurons due to the intuition that in a fully connected layer they share some similarity to the dendrites, soma, and axon of a biological neuron [47]. These trainable weights are the name-giving components in a *neural* network.

(a) Exclusive superordinate classes

Spherical/Oblate — Prolate

Approximate particle shape

(b) Partially exclusive oblate subclasses

Round — Elliptical — Double Rings

Schematic representation of the expected feature

(c) Non-exclusive prolate subclasses

Bent — Streak

(d) Non-exlusive other subclasses

Asymmetric — Newton Rings — Layered
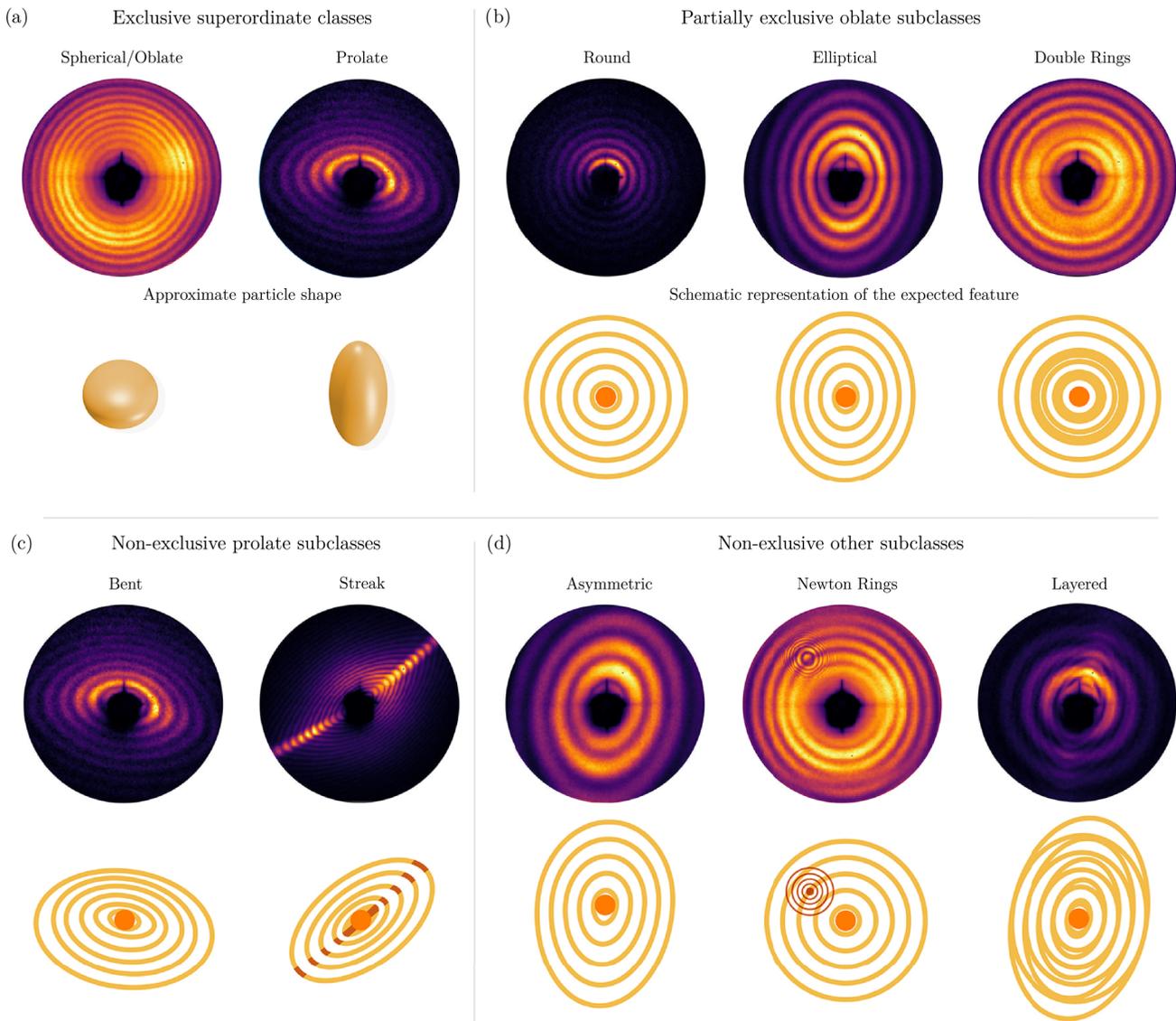
FIG. 2. Characteristic examples for all the classes assigned to the 7264 images by a researcher, except for the *empty* class. The top row of every class shows a representative diffraction pattern and the bottom row in panels (b)–(d) shows a stylized drawing of the characteristic feature of this class. The bottom row in panel (a) shows an illustration of the name-giving particle shape for the *spherical/oblate* and *prolate* classes. The shapes are derived from the analysis of the data in Langbehn *et al.* [41], and they serve as a form of superordinate classes. They are mutually exclusive to each other, and all diffraction patterns are part of one of these two classes. Also, both superordinate classes have subclasses. For example, panel (b) shows the *spherical/oblate* subclasses *round*, *elliptical,* and *double rings*. While a diffraction pattern can be part of the *round* and the *double rings* classes, it cannot be part of the *round* and the *elliptical* classes. For the *prolate* superordinate class, we find analog subclass rules, although there is no exclusivity rule as it was with the *round* and *elliptical* class. Therefore, an image belonging to *bent* can also be in the *streaks* class. Furthermore, all *spherical/oblate* and *prolate* patterns cannot only be part of their respective subclass but can also be part of one or more of the classes in the *nonexclusive other subclass* categories, shown in panel (d). These classes describe general features within the image which are to some extent independent of the particle shape. We derived the superordinate classes from these general features. These complicated interclass relationships demonstrate the capabilities of a researcher to interconnect mostly distinctive appearing features into a consistent description and ultimately leading to a valid physical interpretation. A hand-crafted algorithm could not account for these relationships normally, but now these interconnections can serve as an additional evaluation metric for the neural network. Since there is no diffraction pattern which belongs to the *spherical/oblate* and *prolate* class simultaneously, we can check if the neural network mislabeled a diffraction pattern according to these rules. We can then interpret this as a reliable indicator for a failed generalization of the network. The physics behind these patterns are quite complicated as well, but for a rigorous interpretation and analysis of these patterns, please see Langbehn *et al.* [41].

Now, the goal of training a neural network is to optimize all these weights for all layers, so that the predictions for all images in the training data match their accompanying original labels. The original labels are called ground truth and define the upper limit of how good of a network can fit a domain. No neural network is better than its training data. In this section, we briefly illustrate the affine transformations of the fully connected layer and the convolutional layer and

TABLE I. Statistics of the helium nanodroplets data set. Nonexclusive labels assigned by a researcher. One image can be in multiple classes. Total data-set size is 7264. Note that *spherical/oblate* as a class also contains *round* patterns; only *prolate* shapes are excluded from this class (see also caption of Fig. 2).

Class	No. of labels	Percentage (%) of the whole data set
Spherical/oblate	6589	90.7
Round	5792	79.7
Elliptical	796	11.0
Newton rings	460	6.3
Prolate	453	6.2
Bent	390	5.4
Asymmetric	367	5.1
Streak	242	3.3
Double rings	218	3.0
Layered	47	0.7
Empty	222	3.1

then explain in the next section the role of the activation function.

a. Fully connected layer. The name-giving operation for the fully connected layer is a matrix multiplication performed on a flattened input, for example, an $m \times n$ sized input image would be *flattened* into a $m \cdot n$ sized vector. Mathematically, this is a matrix multiplication between a matrix and a vector,

$$a_j = \sum_{k=1}^{m} x_k w_{kj}, \qquad (1)$$

where x is the flattened input and w is the weight matrix of a fully connected layer. Here, all input vector elements (e.g., the pixels of an image, now arranged in one large row x_k) contribute to all output matrix elements and are therefore *connected*. Furthermore, by convention is x_0 defined as 1 and $w_{0j} = b_j$, where b_j is a free and trainable bias parameter.

b. Convolutional layer. In a convolutional layer, the trainable weights are parameters of a kernel that slides over the inputs; this is visualized in Fig. 3. The general idea of a convolutional layer is to preserve the spatial correlations in the input image when going to a lower dimensional representation (the next layer). This is achieved by using a kernel with a spatial extent larger than 1 px. The kernel size is then also the extent to which one kernel can correlate different areas of an input and is called its local receptive field. Each kernel produces one output which is called a *feature map* or *filter*. Multiple feature maps from multiple kernels are grouped within one convolutional layer. For example, the first convolutional layer in Fig. 3 produces nine feature maps out of the input diffraction image and hence has nine kernels that get optimized during training. Since we usually only have in the input layer a two-dimensional diffraction image as input and a high number of feature maps for every subsequent convolutional layer as their inputs, we define the output of a convolutional layer with a four-dimensional kernel k that produces i feature maps of size $j \times k$:

$$a_{i,j,k} = \sum_{l,m,n} x_{l,j+m-1,k+n-1} k_{i,l,m,n}; \qquad (2)$$

here, the input x has l dimensions of size $j \times k$ and we slide a kernel of size $m \times n$ across all these l dimensions. In the given
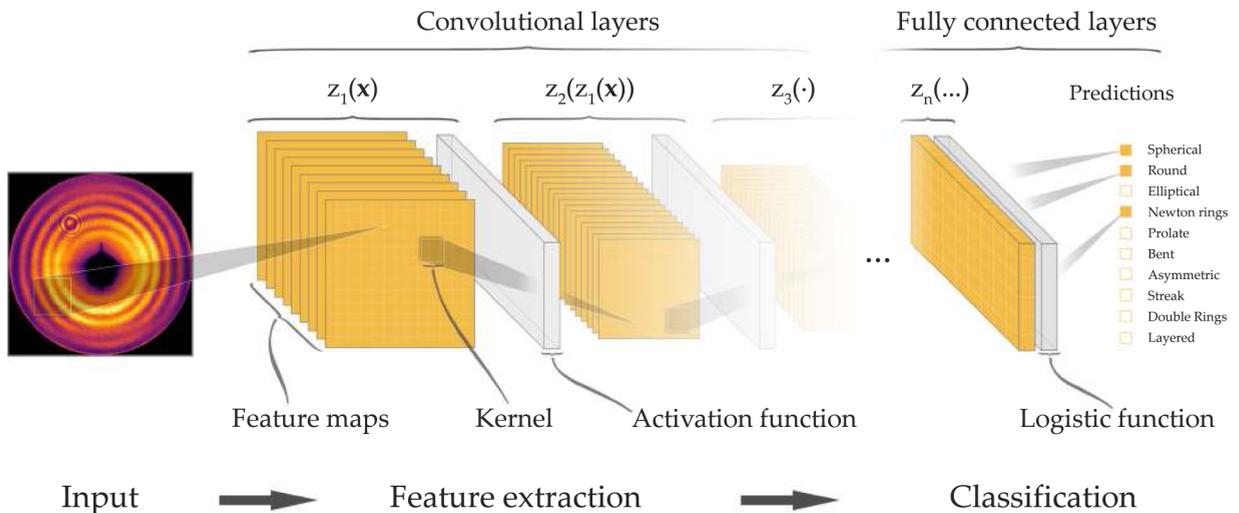
FIG. 3. Schematic visualization of a convolutional neural network. It shows the hierarchical structure of the network with the function hierarchy $z_1, \ldots, z_n$ above each layer. Depicted as input is a diffraction image, which is getting expanded by nine trainable convolutional kernel into nine feature maps. Note that only one kernel, producing the last feature map, is shown. The output of the first layer is then passed through multiple convolutional layer; this is the feature extraction part of the neural network. Ultimately, a fully connected layer with a logistic function as activation function produces the predictions. Every layer consists of two stages, also indicated by the brackets underneath $z_1, \ldots, z_n$. The first stage is an affine transformation and the second one is a nonlinear function, called an activation function. The operation that is used as affine transformation is then the name-giving component for the layer, e.g., a convolutional layer uses a convolution as affine transformation. The choice of the activation function is subject to empirical optimization with various choices possible. Section III A 1 describes the affine transformations in more detail and Sec. III A 2 covers the basics on activation functions.

example for the input layer, *l* is simply 1 and the summation is just across one input image, as shown in Fig. 3.

2. *Activation functions*

Regardless of the affine transformation that is used, all layer-specific operations produce trainable weights which are passed through an activation function. This function is always nonlinear. We only address two activation functions here as they are the most common used by the community and the only ones we use: the sigmoid and the *LeakyRelu* functions. The first one is a logistic regression function used mostly at the outputs of neural networks, and the second one is a piecewise linear activation function used between layers for numerical reasons [48,49]. The sigmoid function is given as

$$h(x) = \frac{1}{1 + \exp(-x)} \tag{3}$$

and the *LeakyRelu* function is given as

$$h(x) = \begin{cases} x & \text{if } x \geqslant 0 \\ \gamma x & \text{if } x < 0, \end{cases} \tag{4}$$

where in both functions $x \in \boldsymbol{a}$ are the trainable weights of the affine transformation [the convolutional or the fully connected layer operation, i.e., the output of Eqs. (1) or (2)] and γ is the slope for the negative part in the *LeakyRelu* function and is called *leakage*.

In Fig. 3, the last activation function of the neural network, denoted by *Logistic function*, is a sigmoid function, because its output can be interpreted as the probability in a Bernoulli distribution, yielding a probability for how likely it is that a given event (an image in our case) is part of a class (in our case, the predefined classes from Table I). Sigmoid functions always give an output between 1 and 0. In our case, we have 11 distinct classes which are mutually nonexclusive, which means every image has a probability of being part of every class. Using a sigmoid function at the end of the neural network yields therefore 11 distinct Bernoulli distributions. The generalization from the single-case Bernoulli distribution to its multicase *n*-class distribution equivalent is called categorical distribution.

Interpreting the output of the neural network, as well as the original labels, as a categorical distribution is key to train the neural network because only then we can use statistical measures to evaluate the quality of the neural network's prediction, which allows us to optimize it iteratively.

However, because of the nonlinearity of all activation functions, optimizing a neural network is a nonconvex problem where no global extrema can be found with certainty. The general procedure is that of a forward pass and then a backward correction, meaning that we feed the neural network several images, take the network's prediction, and compare this prediction to the ground truth; this is the forward pass. Then, we calculate a loss function which is a metric for how bad or good the predictions were (see the next section), and correct the weights of the network in a way that would make it better equipped to predict the labels for the images it just saw. This correction step is starting at the end of the network using an algorithm called back-propagation, hence the name backward correction; see Sec. III A 4.

3. *The forward pass: Assess the network's predictions*

Optimizing a neural network always starts by feeding it multiple images and evaluate what the neural network made of it. For assessing the quality of the network's prediction, a so-called loss function is used. It is the defining metric that we seek to minimize during the training of the neural network. In every training step, we compare the output of the neural network to the real labels provided by the researcher and calculate the so-called loss. Lower loss values correspond to a higher prediction quality of the neural net.

Therefore, the goal during the training process is to adjust all weights and biases within the network so that the loss is minimal for all input training images. There are various possible loss functions which often serve a specific purpose. For classification tasks, such as the present case, primarily the *cross entropy* is used [50–53]. Cross entropy is a concept from information theory giving an estimate about the statistical distance between a *true* distribution *p* and an *unnatural* distribution *q*. In our case, *p* is the categorical distribution over the ground truth labels and *q* is the output of the neural network.

Cross entropy is calculated as the sum of the Shannon entropy [54] for the *true* distribution *p* and the Kullback-Leibler divergence [55] between *p* and *q*. The former is a measure of the total amount of information of *p*, and the latter is a typical distance measure between two probability distributions.

If the Kullback-Leibler divergence is zero, then the cross entropy is just the Shannon entropy of *p*, and we have $p = q$. Then, the predictions of the neural network are not distinguishable from the labels of all training images.

Cross entropy can be formally written as

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p\|q), \tag{5}$$

where $H(p)$ is the Shannon entropy of *p* and $D_{\mathrm{KL}}(p\|q)$ is the Kullback-Leibler divergence of *p* and *q* [56].

When using a sigmoid function as activation function on the output layer, the final loss function can be defined as

$$H(x_{\mathrm{out}}, x) = \sum_i^M x_i^{\mathrm{out}} - x_i^{\mathrm{out}} x_i + \log\left[1 + \exp\left(-x_i^{\mathrm{out}}\right)\right], \tag{6}$$

where *M* is the number of all images in the training data, x_i^{out} is the prediction for one image from the deep neural network, and x_i is the original label of the image, assigned by the researcher. Please see Appendix B for a complete derivation.

Using Eq. (6) as it is would require us to pass all images through the network for one training step, as the sum runs over all images. This is computationally intractable. Therefore, we use a variant of Eq. (6), where the sum runs only over a stochastically chosen subset of size *bs*, called a batch. The size of that batch is called batch size and is an important hyperparameter that needs to be chosen prior to training; see Sec. III A 5. One iteration step now involves only *bs* images from the data set, and we define an *epoch* as the number of iteration steps it takes the network during the training to see all images one time.

To summarize, minimizing the cross entropy is the goal during the training process in a neural network. The network

learns to link the user-defined labels to the provided images. All that is left to understand the basic training process of a neural network is a way to adjust the weights in all layers.

4. The backward correction: Gradient descent and back propagation

Optimizing the weights within the neural network so that they give minimal loss for all training images is done using two distinct algorithms: gradient descent and back propagation. In principal, gradient descent works by evaluating the gradient at some point and then moving a certain step size in the opposite direction. This is done iteratively until the gradient is smaller than some predefined threshold, which is the numerical equivalent of calculating the extrema of a function analytically.

The basic gradient descent step is given by

$$\boldsymbol{w}_{\tau+1} = \boldsymbol{w}_\tau - \eta \nabla_{\boldsymbol{w}_\tau} H(x_{\text{out}}, x), \qquad (7)$$

where η is the afore mentioned step size, called learning rate, $\nabla_{\boldsymbol{w}_\tau}$ is the gradient with regard to the weights at step τ, and $H(x_{\text{out}}, x)$ is the loss function from Eq. (6). With Eq. (7), we already could update the weights within the output layer of the neural network [$z_n(\cdot)$], since for the output layer we can calculate the numerical gradients, but we cannot do this for the layers that come before the output layer since we lack a way to include these. In order to propagate the gradient descent correction throughout the network, an algorithm called back propagation is used [57]:

First, we define the gradient of $H(x_{\text{out}}, x)$, with regard to the weights at the output of the deep neural network, using the chain rule:

$$\nabla_{\boldsymbol{w}_\tau} H(x_{\text{out}}, x) = \frac{\partial H(x_{\text{out}}, x)}{\partial w_{j\tau}^N} = \frac{\partial H(x_{\text{out}}, x)}{\partial h^N(a_j^N)} \frac{\partial h^N(a_j^N)}{\partial w_{j\tau}^N}, \quad (8)$$

where N denotes the layer depth of the output layer, $h^N(\cdot)$ is the used activation function in that layer, and a_j^N are the outputs of the layer-specific operation, as in Eqs. (1) and (2). Starting from there, we include the layer preceding the output layer [$z_{n-1}(z_n(\cdot))$], by making use of the chain rule again:

$$\frac{\partial H(x_{\text{out}}, x)}{\partial h^N(a_j^N)} = \frac{\partial H(x_{\text{out}}, x)}{\partial h^{N-1}(a_j^{N-1})} \frac{\partial h^{N-1}(a_j^{N-1})}{\partial h^N(a_j^N)}. \quad (9)$$

This can be iteratively repeated until the input layer [$z_1(\cdot)$] is included in the calculation. By making use of the chain rule until we reach the input layer, we can include all trainable weights of all layers into the correction term of the gradient descent algorithm. With this, we conclude the full optimization routine in Table II.

5. Training setup

Of significant importance is the way how the network is constructed. How deep should the network be and of what should it consist? For nomenclature, the combination of all used layers, the depth of the network, and the used activation functions is called an architecture.

We bench marked the performance of various architectural choices when used with diffraction images as input and

TABLE II. The iterative optimization routine for a deep feed-forward neural network.

1. **Forward pass**: Propagate *bs* images through the network.
2. **Evaluate the predictions**: At the output layer calculate the loss between the ground truth and the output of the deep neural network [Eq. (6)].
3. **Construct the backpropagation rule**: Include all gradients with regard to the weights of all layer according to Eq. (9).
4. **Backward correction**: Update all weights in the network using gradient descent; see Eq. (7).

provide the results in Appendix A, not in the main paper, due to its rather technical character. In short, all architectures are established through extensive empirical research. So far, not only the leading artificial intelligence (AI) research institutes, like the Massachusetts Institute of Technology (MIT) and the University of Toronto, but also large companies like Google, Facebook, and Microsoft have invested significant amounts of resources to establish well-working *out-of-the-box* solutions [50–52].

Building on this and after extensively bench marking the most common architectures on our own, we settled on an architecture called preactivated wide residual convolutional neural network in its 18-layer configuration, called *ResNet18* [24,58,59]. In essence, it is a convolutional neural network much like the example in Fig. 3 but it employs so-called residual skip connections which increase *accuracy* while decrease training time; see Appendix A for further details as well as comparisons with other architectures.

After settling on an architecture, training a neural network requires fine-tuning of multiple free parameters. Four of them are critical: The learning rate η, the batch size *bs,* and so-called regularization parameters of which we have two (which will be introduced at the end of this section).

We set the initial learning rate for the gradient descent algorithm to $\eta = 0.1$; see also Eq. (7). Throughout the training, we multiply η with 0.1 every 50 epochs; this increases the chance for the gradient descent algorithm to get numerically closer to a minimum in the loss function [58]. Furthermore, we use a batch size of 48 for all training procedures; see also the explanations for Eq. (6).

We split the manually classified part of the helium data set into training and evaluation subsets, where we shuffle the order of all images and then select 85% for the training set while the rest serves as an evaluation set.

We rescale all diffraction images to 224×224 px, which is necessary to fit the deep neural net on two Nvidia 1080Ti GPUs, each having 11 GB of memory. The image dimensions are chosen to be a compromise between file size and resolution. All features we are training the neural network on are still clearly visible and distinguishable after the rescaling.

Furthermore, we face the problem of having a comparatively small training set, consisting of only $\approx$6000 classified images, which could result in a phenomenon called overfitting, meaning the network memorizes the training set without learning to make any meaningful prediction from

it. Therefore, we employ two additional techniques called regularization and data augmentation:

(1) Regularization means adding a so-called penalty term to the loss function. There are two regularizations we use, L1 and L2 [60]. These penalty terms are dependent on the weights themselves and not on the labels, making the loss function explicitly dependent on the weights of the neural network. This dependency encourages the neural network to reduce the values of all weights according to the two penalty terms and ultimately find a sparser solution which in return helps to prevent overfitting. Formally, we add these two terms to the loss in Eq. (6):

$$H(x_{\text{out}}, x)_{\text{reg}} = H(x_{\text{out}}, x) + \alpha ||\boldsymbol{w}||_1 + \beta ||\boldsymbol{w}||_2, \qquad (10)$$

where $H(x_{\text{out}}, x)$ is the cross-entropy loss function, $||\boldsymbol{w}||_1$ and $||\boldsymbol{w}||_2$ are the L1 and the L2 norms applied on the sum of all trainable weight parameters, and α and β are so-called regularization coefficients. In our experiments, we set α and β to 1×10^{-5} during training. Using L1 and L2 regularization in combination is commonly referred to as elastic net regularization [60].

(2) Data augmentation means creating artificial input images by randomly applying image transformations on the original image like flipping the vertical or the horizontal axes and adjusting contrast or brightness values randomly. This greatly increases the robustness to overfitting and is used as a standard procedure when facing small training data sets [61,62].

We were able to train deep neural networks with a depth of up to 101 layers without overfitting using regularization and data augmentation; see Appendix A. In all experiments reported here, we chose a depth of 18 layers for the neural network, because of numerical, memory, and time reasons. We trained all deep neural networks variants for 200 epochs.

B. Evaluating a deep neural network

We use three metrics to assess the quality of the predictions from the neural network, *accuracy*, *precision*, and *recall*. We calculated these metrics every 2500 training iteration steps ($\approx$52 epochs) using the evaluation data set. Accuracy is formally defined as

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Condition Positives} + \text{Condition Negatives}},$$

where condition positives (negatives) is the real number of positives (negatives) in the data and true positives (negatives) is the correct overlap of the prediction from the model and the condition positives (negatives). An accuracy of 1 corresponds to a model that was able to predict all classes of all images correct. Therefore, accuracy is a good measure for evaluating the prediction capabilities of a model when true positives *and* true negatives are of importance. Predicting negative labels correct is in the case of the helium data set of particular interest because we want to estimate if the neural network was able to understand the complex interclass relationships imposed by the researcher. The network should realize that if, for example, one prediction is *spherical/oblate*, it cannot simultaneously be *prolate*. Therefore, the network has to produce a true negative for either one of these predictions. However, using

only accuracy as a metric has several downsides. The most important one is the decreased expressiveness of accuracy when working in a multiclass scenario. In order to understand this, we first introduce precision and recall, and then provide an example:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}},$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$

Precision, also called positive predictive value, is a measure for how reasonable the estimates of the model were when it labeled a class positive, and recall is a measure for how complete the model's positive estimates were.

For example, if the model would predict all training images in the helium data set to be *spherical/oblate* and nothing else (out of 7264 images, 6589 are indeed *spherical/oblate*) then accuracy would be 0.767, which translates to 77% of all labels correctly assigned. However, if the model estimated all images to be part of no class (setting every label to negative), then accuracy would be 0.801, because out of 79 904 possible labels (11 independent classes for 7264 images), 64 339 are negative. Therefore, we would have a useless model that still was able to predict 80% of all labels correct.

Using precision in these both examples would give 0.907 for the *spherical/oblate* example and 0.000 for the all-negative example. Precision is, therefore, a metric that quantifies how well the positive predictions were assigned. Since 91% of all images are indeed *spherical/oblate*, setting all labels positive in the *spherical/oblate* class can make sense, and precision also provides insight when the model makes no positive prediction at all which would be a useless model for our purpose. However, precision alone is not sufficient as a metric. At this point, we do not know if our model predicted almost every possible positive label correctly or if only a small fraction of all positive labels were assigned correctly. We therefore need an additional measure for the generalization capabilities of our model. For that reason, precision is always used in combination with recall. The recall for our first example is 0.423 and for the second one 0.000. Recall relies on ralse negatives instead of the false positives, used by precision, which provides a measure about the completeness of all positive predictions compared to all positive labels within our data. Recall states that our model only captured 42% of all possible positive labels in the *spherical/oblate* example, showing that generalization of the model would not be sufficient for a real-world application.

Therefore, a balanced interpretation of these three metrics is necessary to estimate the quality of the models tested here.

IV. BASELINE PERFORMANCE OF NEURAL NETWORKS WITH CDI DATA

In this section, we briefly report on what we call baseline results. We used the previously described ResNet [24] neural network architecture in its basic configuration with a depth of 18 layers, termed vanilla configuration or ResNet18 (see Sec. III A 5) and trained it with the helium diffraction data set as described in Sec. II as well as with a reference data

TABLE III. Overall evaluation metrics for the ResNet18 architecture (vanilla configuration) and both data sets. The table gives the max values during training for accuracy, precision, and recall. The training time after which the neural network achieved the highest accuracy score on the evaluation data set is labeled t_{max} and the time for training the full 200 epochs is labeled t_{full}. See also Appendix A for further details.

Architecture	ResNet18	
Data set	CXIDB	Helium
Accuracy	0.967	0.955
Precision	0.932	0.918
Recall	0.933	0.866
t_{max} [h]	0.278	0.231
t_{full} [h]	0.668	0.694

set from the literature [39]. This reference data set was made freely available on the CXIDB by Kassemeyer *et al.* [39]. It contains diffraction patterns of a number of prototypical diffraction imaging targets, namely the *Paramecium bursarium* chlorella virus (PBCV-1), bacteriophage T4, magnetosomes, and nanorice. For further experimental details, see Ref. [39].

We selected this data set because of a previous publication dealing with this data set [27] that describes, to our knowledge, the current state-of-the-art method for classification and sorting of diffraction images [27]. Bobkov *et al.* [27] trained a support-vector machine on the CXIDB data set and inferred the particle type directly from the diffraction images. Overall, they achieved an accuracy of up to 0.87, but only on selected high-quality images with a high confidence score of the support-vector machine above 0.75.

Table III shows the overall evaluation metrics as well as the training wall time. t_{max} is the time when the neural network achieved the highest *Accuracy* score on the evaluation data set, and t_{full} is the time for training 200 epochs. In practice, we achieved optimal convergence after training for 70 to 100 epochs.

We achieved an accuracy of 0.967 on not only a high-quality subset of the CXIDB data, like in Ref. [27], but on all available data (see Table III), using a vanilla ResNet18 architecture, proving that using a neural network significantly outperforms the current state-of-the-art approach given in Ref. [27].

In the case of the helium data set, we face a much more complicated multiclass learning problem (one image can belong to multiple classes compared to one image belongs to exactly one class as it is in the CXIDB data). However, we reach a comparable accuracy score of 0.955. Even more promising, precision and recall are very high for the helium and the CXIDB data set, proving that the neural network not only predicted the true positives with high confidence and reliability (high precision), it did so for almost all true positive labels in the evaluation data set (high recall).

In the next section, we show how to further improve on the baseline performance of neural networks with diffraction images as input data.

V. ADAPTING NEURAL NETWORKS FOR CDI DATA

Here, we describe our contribution for using neural networks in combination with diffraction images.

First, we show in Sec. V A that the performance of a neural network can be enhanced when using a special activation function after the input layer.

Second, in Sec. V B, we benchmark the performance of the neural network when using a smaller amount of training data. The idea is to provide an intuition about how much the prediction capabilities deteriorate when a smaller training data set is used. This is useful because so far a researcher still has to invest a lot of time preparing the training data set and, more general, minimizing the time spent looking through the raw data is the ultimate goal for using a neural network in the first place.

Third, in Sec. V C, we propose a data augmentation in the form of a custom two-point cross-correlation map that hardens the network against very noisy data. We show that when using this augmentation the network is more robust to noise from a uniform distribution added on top of the original diffraction image. This simulates the experimental scenario in which a very low signal-to-noise ratio is unavoidable, e.g., during CDI experiments with very limited photon flux [6] or very small scattering cross sections, as it is in the case with upcoming CDI experiments on single biomolecules [63,64].

A. The logarithmic activation function

One of the key additions of this paper is the proposed activation function, formally stated in Eq. (11). It is designed to account for the inherent property of diffraction images of scaling exponentially. More generally, the intensity distribution of scattered light on a flat detector follows two laws, depending on the scattering angle that is recorded. For very small angles (SAXS and USAXS experiments), the Guinier approximation is the dominant contribution to the recorded intensity, while for larger scattering angles (SAXS and WAXS experiments) Porod's law becomes dominant [65,66]. Where the scattering intensity in the Guinier approximation is proportional to $\approx \exp(-q^2)$, in Porod's law the intensity scales with $\approx q^{-d}$. q is the scattering vector (function of the scattering angle and of the wavelength in use) and d is the so-called Porod coefficient, which can vary significantly depending on the object from which the light was scattered [65].

In any case, the recorded detector intensity for diffraction images scale exponentially. For this reason, we propose a logarithmic activation function of the form

$$h(x) = \begin{cases} \alpha[\log(x + c_0) + c_1] & \text{if } x \geq 0 \\ -\alpha[\log(c_0 - x) + c_1] & \text{if } x < 0, \end{cases} \quad (11)$$

where $\alpha > 0$ is a tunable scaling parameter, $c_0 = \exp(-1)$, $c_1 = 1$, and x is the input.

We define c_0 and c_1 so that the activation function is antisymmetric around 0, which helps speed up training and avoids a bias shift for succeeding layers [67,68].

Since we are using a gradient-based optimization technique, we need to take care that the gradient can propagate throughout the whole network; otherwise it would lead to so-called *gradient flow* problems, which befalls deep

TABLE IV. Evaluation metrics for the ResNet18 network with and without the logarithmic activation function. We benchmark three values for α. Results are shown for both data sets and are the maximum value recorded during training. Bold numbers indicate the best scores across their respective category.

Architecture	ResNet18			
α	0.2	0.5	1.0	Unmodified
Data set		Helium		
Accuracy	**0.965**	0.960	0.959	0.955
Precision	**0.922**	0.920	**0.922**	0.918
Recall	**0.870**	**0.870**	0.868	0.867

TABLE V. Evaluation metrics of the ResNet18$_{\alpha=0.2}$ network with the logarithmic activation function and an α value of 0.2. Results are shown for the helium data sets and reflect the maximum achieved value reached throughout the training process, assessed on the evaluation data set. Bold numbers indicate the best scores across their respective category.

Architecture	ResNet18$_{\alpha=0.2}$			
Training set size	6174	4631	3088	1544
Data set		Helium		
Accuracy	**0.965**	0.915	0.829	0.797
Precision	**0.922**	0.821	0.740	0.673
Recall	**0.870**	0.771	0.679	0.593

architectures [48,69]. There are two possibilities for insufficient gradient flow: either the gradients are getting too small (*vanishing gradient*) or too large (*exploding gradient*) when propagating throughout the network. Both scenarios lead to numerical instabilities during training, making convergence for large architectures very hard or even impossible. The reason for this is the backpropagation algorithm which invokes the chain rule for calculating the gradients. Every gradient is therefore also a multiplicative factor for the gradient of a succeeding layer. For our case, the derivative of Eq. (11) with regard to x is given by

$$\frac{\partial h(x)}{\partial x} = \begin{cases} \frac{\alpha}{x+c_0} & \text{if } x \geqslant 0 \\ \frac{\alpha}{c_0-x} & \text{if } x < 0. \end{cases} \tag{12}$$

It shows that the gradient scales with x^{-1} with a discontinuity of size αc_0^{-1} at 0.

If we used this activation function for all activations throughout the network, the gradient would have an increased probability to vanish—or explode—the deeper the architecture gets. In addition to that, the discontinuity at $x = 0$ could lead to gradient jumps, which would further decrease numerical stability. Therefore, we use the logarithmic activation function only for the first convolutional layer and use a *LeakyRelu* activation with *leakage* of 0.2 on all hidden layers. This compromise still captures the exponential scale of the diffraction images but without losing numerical stability.

Since α is a tunable hyperparameter, we conduct experiments with three values for $\alpha \in [0.2, 0.5, 1.0]$ and evaluate its impact on the performance of the neural network.

In Table IV, we provide the evaluation metrics for ResNet18 used with the logarithmic activation function, trained with three different values for α. For comparison, we also provide the results of the unmodified ResNet18 labeled *unmodified*. The best-performing configuration is with an α value of 0.2, maxing out with an accuracy of 0.965. Therefore, providing a boost in accuracy of a full percentage point compared to the unmodified ResNet18. The lowest value for the maximum accuracy was reached without the logarithmic activation function, topping at 0.955. precision and recall both increase with the addition of the logarithmic activation function. These improvements all come without increasing training time or complexity of the model. The maximum achieved accuracy seems to be anticorrelated to α, with the ResNet18$_{\alpha=1.0}$ variant performing worst. We suspect that this

is related to the smaller size of the discontinuity of the derivative of $h(a_j)$ when choosing a small value for α; see Eq. (12).

However, choosing even smaller values for α did not improve the accuracy further, either because the benefit from the activation function plateaus there or because we reached the classification capacity of this ResNet layout.

These results show convincingly that the addition of the logarithmic activation function improves the overall performance and generalization of the deep neural network. This is expected because we imposed a form of feature engineering on the network, by exploiting a known characteristic of the data set. Therefore, without increasing the complexity, the depth or the training time, we showed that using the logarithmic activation improves all relevant evaluation metrics. For this reason, we use the logarithmic activation function with an α value of 0.2 as default for all following experiments.

B. Size of the training set

In this section, we evaluate the impact of the training set size on the evaluation metrics, and we trained the ResNet18$_{\alpha=0.2}$ with a varying amount of labeled images. The reason for this is to provide guidance for how many images are needed to be classified manually before the employment of a neural network is useful. We uniformly select images from the training set but kept the same evaluation data set described in Sec. III A 5. We decreased the size of the training set in three stages (to 75% $\equiv$ 4631, to 50% $\equiv$ 3088 images, to 25% $\equiv$ 1544 images).

Table V shows the performance of ResNet18$_{\alpha=0.2}$ when trained with data sets of different sizes. For the helium data set, the maximum achieved accuracy is dropping from 0.965 to 0.797 when using only 1544 images instead of the full 6174 images. Even more pronounced is the decline in precision and recall from 0.922 and 0.870 to 0.673 and 0.593 for the smallest training set size. The steeper decline rate for precision and recall, compared to accuracy, can be understood as the helium data set predominantly consists of negative ground truth labels (64 339 out of 79 904 labels) to which the neural networks resorts in the absence of sufficient training data. Precision and recall, on the other hand, provide only information about the positive prediction capabilities and their completeness and therefore decrease faster when a smaller training set size is used.
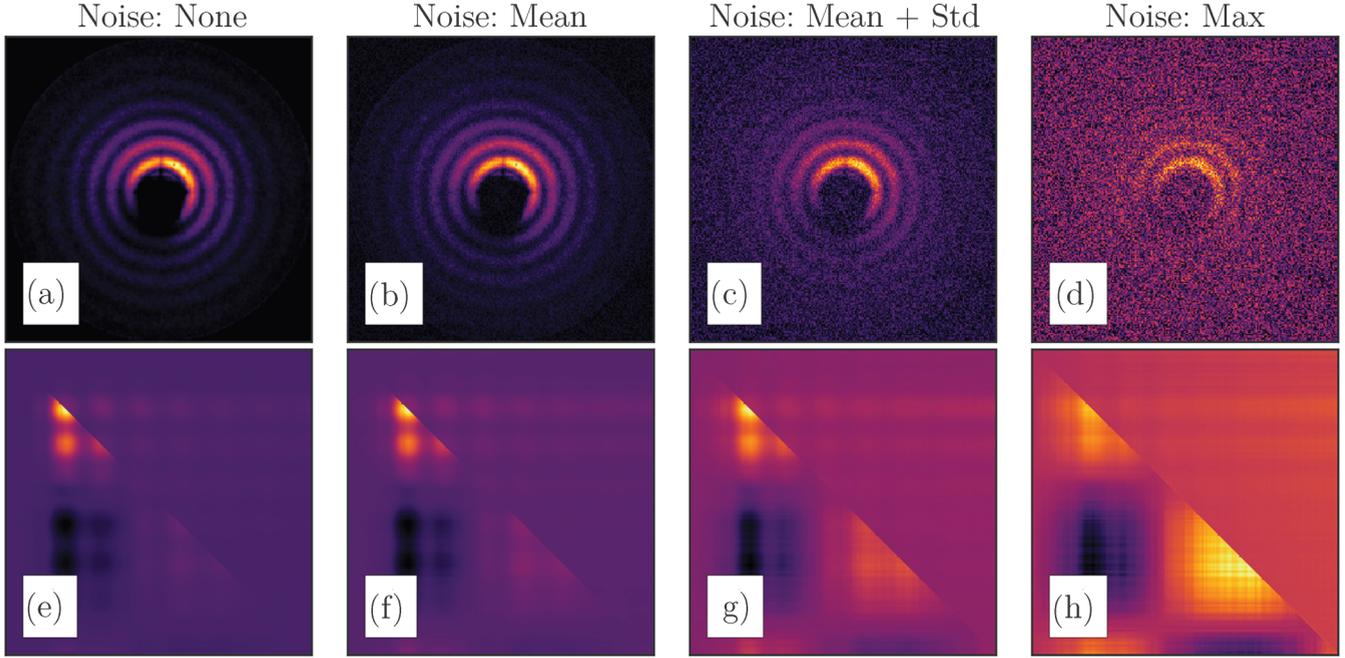
FIG. 4. Panels (a) to (d) showing the various stages of added noise to a standard scattering image. Panels (e) to (h) are the calculated correlation maps with the upper triangle of order $n = 8$ and lower triangle from the full CCF calculation.

This shows that the number of images is critical for the prediction capabilities of the neural network. The drastic decrease in training set size results in a much worse generalization of the model, detecting only those images that are very close to the ones from the training set, missing most from the evaluation set. The network has not learned the characteristics of a particular class to a point where it can transfer the gained knowledge to other images, which is the one critical property for which we employed a neural network in the first place.

Therefore, if time is limited, one may be well advised to concentrate efforts on preparing a sufficiently large, high-quality training data set while using, e.g., our here presented neural network approach in its standard configuration.

C. Using two-point cross-correlation maps to be more robust to noise

This section introduces an image augmentation based on the two-point cross-correlation function, which increases the resistance to noise. We prepare four training sets, each with an increasing amount of noise sampled from a uniform distribution, and analyze the noise dependence of the neural network.

One of the principal problems in CDI experiments, or imaging experiments in general, is recorded noise. Noise often leads to computational problems due to noise resistance being a known weak point for a significant fraction of predictive algorithms [28]. In particular, deep neural networks are known to be easily fooled by noise. When adding noise to an image, whose addition may be invisible to the human eye, a neural network can come to entirely different conclusions and this even with high confidence, for example, seeing a panda where there was a wolf [70,71]. Therefore, we propose an additional preprocessing step for the input images to increase the noise resistance of the neural network.

To quantify the quality of an image, the signal-to-noise ratio is often used. It is a measure for how much noise is present when compared to some information content, where low values indicate that information might be indistinguishable from noise. It has been shown that higher orders of the two-point cross-correlation function (CCF) can act as a frequency dependent noise filter and increase the quality of a reconstruction of a diffraction image even in the presence of recorded noise [72,73]. And since the CCF can be interpreted as an image [see Figs. 4(e) to 4(h)], we employ this method in a similar manner to optimize the use case with a convolutional deep neural network, expecting that the higher-order terms make the neural network more resistant to the presence of noise.

In general, the CCF is defined as

$$C_{i,j}(q_i, q_j, \Delta) = \int_{-\infty}^{\infty} I_i^*(q_i, \phi) I_j(q_j, \phi + \Delta) d\phi, \quad (13)$$

where Δ is the angular separation, ϕ is the angular coordinate, and (i, j) denotes the index of the two scattering vectors q_i and q_j. For discrete ϕ, and written as Fourier decomposition, Eq. (13) yields [72]:

$$C_{i,j}^n(q_i, q_j) = I_i^{n*}(q_i) I_j^n(q_j), \quad (14)$$

where n denotes the order of the CCF. I_i^n is given by

$$I_i^n(q_i) = \frac{1}{2\pi} \int_0^{2\pi} I(q_i, \phi) \exp(-in\phi) d\phi. \quad (15)$$

Since $C_{i,j} = C_{j,i}$, we can split the final correlation map into an upper and a lower triangle matrix. To maximize information, and to optimally use the local receptive fields of the convolutional layers, we merge the lower triangle from the full CCF calculation, Eq. (13), with $\Delta = 0$, and the upper

TABLE VI. Evaluation results when training a ResNet18$_{\alpha=0.2}$ on the original diffraction images and on CCF maps calculated from them. The results reflect the maximum value achieved throughout the training process, assessed on the evaluation data set. Bold numbers indicate the best scores across their respective category.

Architecture	ResNet18$_{\alpha=0.2}$							
Noise added	None		Mean		Mean + std.		Max.	
Input data	CCF maps	Diff. imgs.	CCF maps	Diff. imgs.	CCF maps	Diff. imgs.	CCF maps	Diff. imgs.
Data set				Helium				
Accuracy	0.950	**0.965**	0.948	**0.954**	**0.946**	0.944	**0.944**	0.926
Precision	0.901	**0.922**	0.897	**0.910**	**0.893**	0.887	**0.905**	0.865
Recall	0.838	**0.870**	0.833	**0.853**	**0.823**	0.815	**0.814**	0.808

triangle of order $n = 8$ from Eq. (14). Therefore, we combine a *plain* correlation map with a higher order map that is more resistant to noise; see Figs. 4(e) to 4(h) for a full example.

To test the robustness of this method, we use the ResNet18$_{\alpha=0.2}$ and train it with various preprocessed data sets.

From our original data set, we derive three additional data sets that only differ in the amount of noise added. We do this as follows: First, we calculate the mean, the standard deviation (std), and the maximum intensity values of each image in the original data set. From these values, we calculate the median instead of the mean (due to increased robustness against outliers), ending up with three statistical characteristics describing the intensity distribution throughout all diffraction images. With that, we define three continuous uniform distributions to sample noise from. A continuous uniform distribution is fully defined by upper and lower boundaries, a and b, respectively. The probability for a value to be drawn within these boundaries is equal and nonzero everywhere. For our three noise distributions, we always use a lower boundary of 0 and vary the upper boundary so that b is either the mean, the mean + the standard deviation, or the maximum of the intensity distribution on the images (the three statistical characteristics described above).

For example, for creating the maximum noise data set, we looped through every diffraction image and added noise sampled from the maximum noise distribution. We do this for all three noise distributions. From these three noise embedded data sets, as well as our original data set, we calculate the here proposed CCF maps. This leads to a total of eight data sets; for each of them we train a ResNet18$_{\alpha=0.2}$. An example of one image in all eight data sets is in Fig. 4.

The results for these eight data sets are given in Table VI. The performance of the neural network without added noise is much stronger when using the original diffraction images instead of the CCF maps. However, as soon as noise is added, the performance of the neural network trained on diffraction images deteriorates much faster as compared to the performance with CCF maps as input. When the upper boundary of the added noise excels the median values of mean + standard deviation, the neural network is performing better with the CCF maps instead of the original diffraction images. Especially with the noisiest data set, the differences in performance are significant. Precision is increased by 4 percentage points when using the CCF maps as input, showing that our data augmentation may serve as a helpful asset when dealing with very noisy data.

In general, it is a viable alternative to use the CCF maps as input to the convolutional deep neural network, which should be considered an option in the case of very noisy data where it provides a boost to classification results. The downside is that calculating the CCF for every image comes at an additional computational cost. It took us three full days to calculate the CCF maps for all 39 879 images of both data sets on an Intel 6700K quad-core machine using a multithreaded PYTHON script (also released on Github).

VI. WHAT THE NEURAL NETWORK SAW

Neural networks are often considered a black-box approach. We usually do not impose *a priori* knowledge on our model; the network learns this on its own. Although this is part of the reason why they are so successful, it also gives rise to doubts about the interpretability of their predictions. Some ways to interpret the processes of decision finding within a trained neural network have been presented in the literature [40,74–76]. In order to get a better understanding of why our deep neural network assigned images to certain classes, we calculated heat maps using the GRADCAM++ algorithm [40]. These heat maps are making visible where the network has looked for in a particular class, which we do by tracing back the gradient flow from the output layer to the last convolutional layer. The network's class-specific interest directly correlates with this gradient signal because, in essence, we simulate a training step using back propagation and interpolate the feature maps from the last convolutional layer. A full description of this process is given in Appendix D. The output of the GRADCAM++ algorithm provides contour maps whose amplitude is a normalized measure for how much the gradient would impose corrections on the weights if used during training. This gradient flow directly corresponds to what the network deemed the most relevant regions.

Figure 5 shows the GRADCAM++ results for the *streak* and *bent* classes using our best performing network: ResNet18$_{\alpha=0.2}$. We present results from these classes, because the distinct spatial characteristics are obvious to the human eye. Therefore, they are an ideal candidate to test if the neural network understood these characteristics. In each row of Fig. 2, a schematic sketch of the key feature together with five randomly selected images from this class are depicted.

The GRADCAM++ contour maps are overlaid on the image, in addition, the contour levels are also used as an α mask
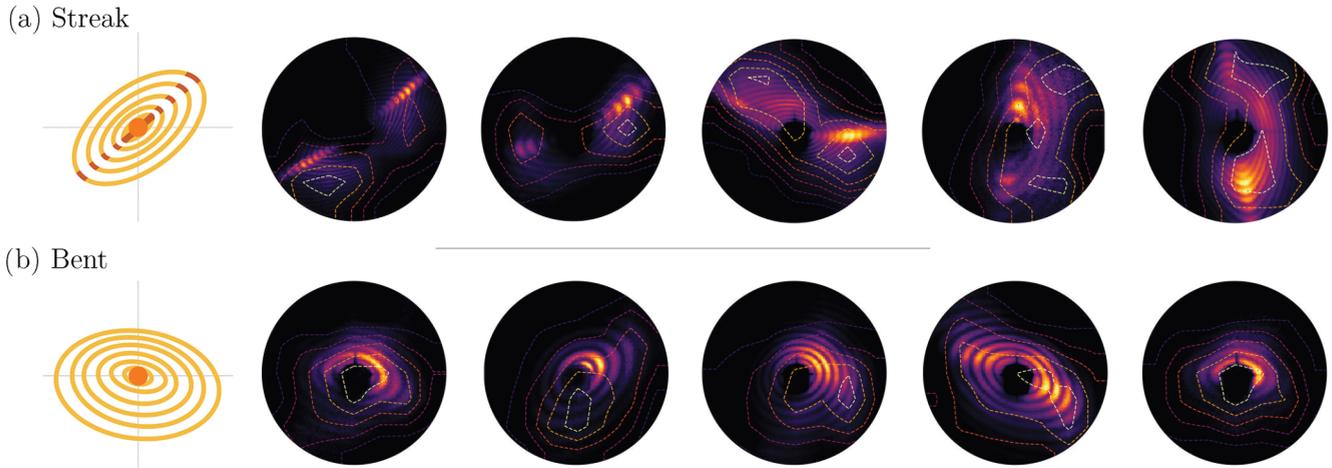
FIG. 5. Showing the GRADCAM++ results for two distinct classes from the helium data set. Panel (a) shows five randomly selected images from the *streak* class and panel (b) shows five images from the *bent* class. We chose these classes due to their distinct and distinguishable characteristic shapes which can easily be identified using the contour maps provided by the GRADCAM++ algorithm. For each class, we plot the schematic from Fig. 2 also at the beginning of each row. GRADCAM++ contour levels are plotted as dashed lines and used as transparency value for the images from which we calculated them. This way regions with strong gradients are also brighter.

for the diffraction image so that the brightest areas in each plot correspond to the ones with the highest gradient flow. In the case of the *streak* class, Fig. 5 clearly shows that the neural network was able to identify the dominant streak feature regardless of its orientation or size. Results on the *bent* class also show a strong correlation between the shape of the contour maps and the bent shape of the diffraction pattern.

Therefore, combining these metrics and the GRADCAM++ images, we think that the *streak* class feature identified by the neural network indeed corresponds to the one seen by the researcher. Also, the *bent* class contour maps from the network show a clear resemblance to the feature intended by the researcher, albeit not so strongly pronounced. Although the deep neural network learned these representations on its own, they align with the intentions of the researcher. This demonstrates that neural networks are capable of learning these complicated patterns on their own.

VII. SUMMARY AND OUTLOOK

In this paper, we give a general introduction on the capabilities of neural networks and provide results on the first domain adaption of neural networks for the use case of diffraction images as input data. The main additions of this paper are (i) an activation function that incorporates the intrinsic logarithmic intensity scaling of diffraction images, (ii) an evaluation on the impact of different training set sizes on the performance of a trained network, and (iii) the use of the pointwise cross-correlation function to improve the resistance against very noisy data. In addition, we provide a large benchmarking routine, utilizing multiple neural network architectures and layouts in Appendix A.

We have shown that even in the most basic configuration, convolutional deep neural networks outperform previously established sorting algorithms by a significant margin. More important, we improved on these baseline results by modifying the activation function for the first layer. For the case of very noisy data, often a problem in diffraction imaging experiments, we showed that two-point cross-correlation

maps as input data instead of the original diffraction images improve the robustness of the classification capabilities of the network. Our results set the stage for using deep learning techniques as feature extractors from diffraction imaging data sets. The ultimate goal will be establishing an unsupervised routine that can categorize and extract essential pieces of information of a large set of diffraction images on its own. We envision for the near future that the gained insights lead to multiple approaches regarding neural networks and diffraction data. For example, the MSFT algorithm used by Langbehn *et al.* [41] can be used as a generative module in an end-to-end unsupervised classification routine using large synthetic data sets as training data for a neural network. This approach can be extended to utilize these trained networks as an online-analysis tool during the experiments. Furthermore, we hope to develop an unsupervised approach that connects the recent research from generative adversarial network theory [77–80] and mutual information maximization [81] with the results of this paper. Such an approach would allow for finding characteristic classes of patterns within a data set without any *a priori* knowledge about the recorded data. All of the code, written in PYTHON 3.6+ and using the Tensorflow framework, is available at Github, free to use under the MIT License [82]. We hope the community uses and improves the code provided in this repository.

(a)

(b)

$$4 + 0 + 0 + 12 + 5 + 28 + 8 + 24 + 5 = 86$$
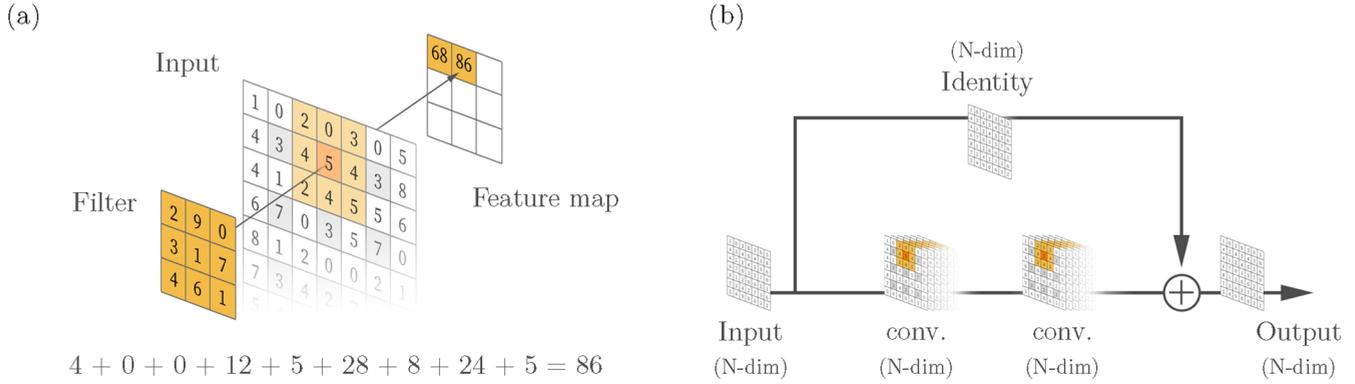
FIG. 6. Schematic for a convolutional operation inside a convolutional layer in panel (a) and for a classic skip connection found in the ResNet architecture in panel (b). Panel (a) illustrates the local receptive fields and shared weights concept. The convolutional filter has size 3×3 and stride 2 and is sliding over the input image of size 7×7, which produces an output, called feature map, of size 3×3. The stride is the distance the filter is moving in each step which is implied by the gray shading every two pixels in the input image. Using a local receptive field describes the inclusion of nearby pixels, and weight sharing means using the same filter weights for the whole input image. The calculation at the bottom is for the second entry in the feature map. (b) A classical skip connection is shown with two convolutional layers that approximate a sparse residue which gets added to the identity at the output.

APPENDIX A: ARCHITECTURAL DESIGN CHOICES

In this section, we describe and explain our choices for neural network architecture to establish as baseline performance when working with diffraction patterns, before the inclusion of our diffraction specific activation function; see Sec. V A in the main text. We present the theory and background on available architectures and provide results on two architectures with five depth layouts.

There are different layer styles from which we can build a neural network. Nomenclature is that a full arrangement of all layers is called architecture, or configuration, of the network.

For our tests, we use two different neural network architectures, a ResNet and a VGG- Net, both with multiple depth layouts. For the ResNet, we train and evaluate three depth variations (18, 50, and 101 layers), and for the VGG-Net, we train two variants (16 and 19 layers).

The structure of this section is as follows: First, we explain how a convolutional layer works in general. Second, we motivate the derivation of the VGG-Net from preceding architectures, and third, we show how the ResNet architecture can be explained by expanding the core ideas used in the VGG-Net. In the following section, we will then present the results for all the trained configurations.

Almost every architectural design is empirically derived [50–52] and consists of multiple combinations of only a few basic layer styles, namely the fully connected layer, a convolutional layer, a pooling operation, and a batch normalization operation. We discuss the pooling and batch normalization layer only in Appendix C, because of their minor role within the neural network. The reader is also referred to the exhaustive overviews by Schmidhuber [50] and LeCun *et al.* [51]. Since the convolutional layer serves as a fundamental basis for image analysis with neural networks, we explain it here in more detail.

The very basic idea of a convolutional layer is that nearby pixels in an input image are more strongly correlated than more distant pixels; this is called a local receptive field. Therefore, by calculating a convolution over an input image

with a trainable filter of size $>1 \times 1$ we can approximate these correlations.

In a convolutional layer, N filter, with size $M \times M$, slides over an input image and produces N convolved maps, called feature maps. One filter uses the same weights on all parts of the input image for producing one feature map; this is called weight sharing. Weight sharing reduces not only the complexity of the model but provides a bridge toward the convolution function in mathematics. With weight sharing, we can identify the filter within the convolutional layer as a kernel function from the mathematical convolution function. Figure 6(a) shows a schematic of a convolutional layer with one filter.

This exemplary filter with size 3×3 slides over an image of size 7×7 and produces a feature map of size 3×3. The feature map is smaller than the input image because the filter moves two pixels for each step. This step size is called *stride*.

Hereafter, we use the notation conv(a, b, c) for a convolutional layer with filter size a $\times$ a, number of filters b and stride c. The example from Fig. 6(a) could, therefore, be written as conv(3, 1, 2) and would result in nine trainable weight parameters plus one bias parameter (not shown in the figure).

This concept was introduced with the LeNet architecture by Lecun *et al.* [83], which is considered the seminal work in the field and the first deep convolutional neural network. After LeCun proposed the LeNet architecture, further research [84] led to the now *de facto* standard for plain convolutional networks, the VGG-Net. Simonyan and Zisserman [85] proposed the original architecture which consists of up to 19 weight layers, of which 16 are convolutional layers and 3 are fully connected ones.

It is easy to build, is easy to train, and provides in general good results [50,51]. For these reasons, we include two variations of it in our tests, namely versions D and E (nomenclature is from Ref. [85]). Table VII shows the details of the architecture, using the naming convention we introduced with the convolutional layer.

Simonyan and Zisserman [85] derived the VGG-Net directly from the LeNet by arguing that three convolutional

TABLE VII. The deep neural network architecture of the VGG variants D and E; conv(a, b, c) is a convolutional layer with filter size a × a, number of filters b, and stride c and max pooling(d, e) is a max pooling layer with filter size d × d and stride e. Note that we changed the fully connected layer of the original architecture to a convolutional layer.

Variant	D	E
Depth	16	19
Input	2 × conv(3, 64, 1)	
Pooling	max pooling(2, 2)	
Block 1	2 × conv(3, 128, 1)	
Pooling	max pooling(2, 2)	
Block 2	3 × conv(3, 256, 1)	4 × conv(3, 256, 1)
Pooling	max pooling(2, 2)	
Block 3	3 × conv(3, 512, 1)	4 × conv(3, 512, 1)
Pooling	max pooling(2, 2)	
Block 4	3 × conv(3, 512, 1)	4 × conv(3, 512, 1)
Pooling	max pooling(2, 2)	
Out block	2 × conv(7, 4096, 1), conv(1, N, 1)	

layers with filter size 3 and stride 1 (VGG-Net) achieve better results than only one filter with size 7 and stride 2 (LeNet), which equals to the same effective local receptive field size [85]. Three layers perform better than one because they have two additional nonlinear activation functions and reduced complexity (less weight parameter because of the smaller filter sizes), which forces the neural network not only to be more discriminative but also to find sparser solutions [85].

Building on the results achieved by the VGG-Net, it was shown that the depth of a deep neural network directly relates

to its classification capabilities [58,68,86]. This led to the introduction of the so-called residual skip connections which further exploit this *depth matters* concept [58,68]. These residual skip connections are the name-giving components for the ResNet architecture.

In principle, a ResNet still uses the VGG architectural layout but exchanges the convolutional blocks 1 to 4 with residual skip connections; compare Tables VII and VIII. This exchange drastically reduces the complexity of the whole network while increasing the number of layers.

The VGG architecture can be broken down into six blocks: one input block, one output block, and four convolutional blocks (see Table VII). Block 2 is the first block in which there are distinctions between VGG variants D and E.

The VGG-Net architecture proved that increasing the depth and decreasing the amount and size of the filters increases the accuracy, which ultimately gave rise to the *plain skip connections*: Blocks of few convolutional layers designed to replace the large amounts of filters in one layer for multiple layers with fewer, and smaller, filters. Two types exist: A classical and a bottleneck skip connection; both differ only in the amount of how much the depth is increased and the complexity decreased.

This addition has so far only modified the depth and complexity of the network and is called a *plain* network; see He *et al.* [58]. It performs reasonably well but not significantly better than VGG-Net. A residual skip connection differs from a plain skip connection only in adding the identity of its inputs to its outputs. This way all the convolutional layers in a skip connection learn only a *residual* of their input. This simple technique enables a ResNet to outperform all other convolutional deep neural network architectures [24,52]. Figure 6(b) exemplifies a classical residual skip connection. There is still an ongoing debate about why a residual neural

TABLE VIII. Used ResNet variants; see also 18-, 50-, and 101-layer layout in Ref. [58]. Note that we added the preactivated layer layout from Ref. [59]: conv(a, b, c) is a convolutional layer with filter size a × a, number of filters b, and stride c; max pooling(d, e) is a max pooling layer with filter size d × d and stride e; avg pooling is a global average pooling layer; and $fc(f)$ is a fully connected layer with output size f. Layers in bold have a stride of 2 during their first iteration, therefore reducing the dimension by a factor of 2.

Variant	Classic	Bottleneck	Bottleneck
Depth	18	50	101
Input	conv(7, 64, 2)		
Pooling	max pooling(3, 2)		
Block 1	$2 \times \begin{bmatrix} \textbf{conv}(3, 64, 1) \\ \text{conv}(3, 64, 1) \end{bmatrix}$	$3 \times \begin{bmatrix} \textbf{conv}(1, 64, 1) \\ \text{conv}(3, 64, 3) \\ \text{conv}(1, 256, 1) \end{bmatrix}$	$3 \times \begin{bmatrix} \textbf{conv}(1, 64, 1) \\ \text{conv}(3, 64, 3) \\ \text{conv}(1, 256, 1) \end{bmatrix}$
Block 2	$2 \times \begin{bmatrix} \textbf{conv}(3, 128, 1) \\ \text{conv}(3, 128, 1) \end{bmatrix}$	$4 \times \begin{bmatrix} \textbf{conv}(1, 128, 1) \\ \text{conv}(3, 128, 3) \\ \text{conv}(1, 512, 1) \end{bmatrix}$	$8 \times \begin{bmatrix} \textbf{conv}(1, 128, 1) \\ \text{conv}(3, 128, 3) \\ \text{conv}(1, 512, 1) \end{bmatrix}$
Block 3	$2 \times \begin{bmatrix} \textbf{conv}(3, 256, 1) \\ \text{conv}(3, 256, 1) \end{bmatrix}$	$6 \times \begin{bmatrix} \textbf{conv}(1, 256, 1) \\ \text{conv}(3, 256, 3) \\ \text{conv}(1, 1024, 1) \end{bmatrix}$	$36 \times \begin{bmatrix} \textbf{conv}(1, 256, 1) \\ \text{conv}(3, 256, 3) \\ \text{conv}(1, 1024, 1) \end{bmatrix}$
Block 4	$2 \times \begin{bmatrix} \textbf{conv}(3, 512, 1) \\ \text{conv}(3, 512, 1) \end{bmatrix}$	$3 \times \begin{bmatrix} \textbf{conv}(1, 512, 1) \\ \text{conv}(3, 512, 3) \\ \text{conv}(1, 2048, 1) \end{bmatrix}$	$3 \times \begin{bmatrix} \textbf{conv}(1, 512, 1) \\ \text{conv}(3, 512, 3) \\ \text{conv}(1, 2048, 1) \end{bmatrix}$
Output block	avg pooling, $fc(N)$		

TABLE IX. Overall evaluation metrics for all architectures and both data sets. The training time after which the neural network scored the highest accuracy score on the evaluation data set is labeled t_{max} and t_{full} is the time for training the full 200 epochs. The table gives the max values during training for accuracy, precision, and recall. Bold scores are the best results in their respective category.

Architecture	ResNet			VGG	
Depth	18	50	101	16	19
Data set			Helium		
Accuracy	0.955	0.958	**0.964**	0.958	0.959
Precision	0.918	0.917	**0.925**	0.923	0.920
Recall	0.866	0.864	**0.878**	0.867	0.867
t_{max} [h]	**0.231**	0.605	0.940	3.271	6.615
t_{full} [h]	**0.694**	1.814	2.820	6.541	6.726
Data set			CXIDB		
Accuracy	0.967	0.973	**0.978**	0.970	0.970
Precision	0.932	0.937	**0.949**	0.944	0.943
Recall	0.933	0.937	**0.941**	0.904	0.904
t_{max} [h]	**0.278**	1.205	1.093	4.374	4.480
t_{full} [h]	**0.668**	1.807	2.623	6.562	6.720

TABLE X. Per-class accuracy, precision, and recall values for the best performing ResNet configuration with 101 layers. Samples are the number of images whose ground truth label is positive in the evaluation data set. Results are shown for both data sets and reflect the maximum achieved value reached throughout the training process, assessed on the evaluation data set.

Class	Accuracy	Precision	Recall	Samples
Oblate	0.9681	0.9770	0.9965	988
Spherical	0.9166	0.9247	0.9849	869
Elliptical	0.9231	0.8054	0.4836	119
Newton rings	0.9352	0.6325	0.2282	69
Prolate	0.9690	0.9274	0.6777	68
Bent	0.9657	0.8161	0.6487	59
Asymmetric	0.9458	0.6044	0.2207	55
Streak	0.9898	0.9372	0.9876	36
Double rings	0.9768	0.7708	0.6788	33
Layered	0.9896	0.9062	0.6170	7
Empty	0.9904	0.9537	0.9763	32

network performs so well [58,68,87]. Research has shown that ResNets find sparser solutions faster because of their layout and that they behave like ensembles of shallower networks with information flow only activated on 10 to 34 layers even when the neural network has a depth of 101 layers [58,68,87].

However, besides empirical success, one of the critical advantages of ResNets is that reaching training convergence is not getting significantly harder when increasing the depth of the neural network, which is usually the case with other architectures. Therefore, the training of very deep residual neural networks is no more difficult than training shallow plain neural networks [52,88].

For these reasons, we train three variants, with 18, 50, and 101 layers, of a further optimized version of the classical ResNet, called preactivated ResNet [24] (see Table VIII for implementation details).

Table IX shows the overall evaluation metrics on the helium and the CXIDB data set. Table X shows the per-class evaluation metrics for the helium data set, which are not needed for the CXIDB data set because predictions on the helium data set are a multiclass problem whereas predictions on the CXIDB data are single class. Single-class—*one-hot*—problems have identical overall and per-class-evaluation metrics. We trained all models as described in Sec. III A 5 in the main text.

Table IX shows the overall evaluation metrics as well as the training wall time. t_{max} is the time when the neural network achieved the highest accuracy score on the evaluation data set, and t_{full} is the time for training 200 epochs. However, in practice, we achieved optimal convergence after training for 70 to 100 epochs. After this, the network showed overfitting.

Both VGG models took significantly longer to train than the ResNet variants, needing between 6.5 and 6.7 h for 200 epochs on both data sets, whereas training ResNet101 took only 2.8 and 2.6 h respectively. Furthermore, the maximum

accuracy of both VGG networks is more than half a percentage point below the maximum of ResNet101: 0.959 compared to 0.964 for the helium data and 0.970 versus 0.978 for the CXIDB data. Also, when increasing the depth from 16 to 19, accuracy did not change much, precision even decreased slightly, and recall remained unchanged.

On the other hand, increasing complexity within the ResNet architecture helped to boost the accuracy from 0.955 (CXIDB data: 0.973) with ResNet18 to 0.964 (CXIDB data: 0.978) with ResNet101.

For the multiclass results in Table X, we chose the ResNet101 layout, as it is our best performing configuration. For classes *oblate*, *spherical*, *streak,* and *empty,* a precision of 0.9247 to 0.9770 and a recall of at least 0.9763 show that the majority of all predictions in these classes were correct and virtually no image was missed.

For the classes *prolate*, *bent*, *double rings,* and *layered*, the ResNet reached a good precision, but a recall score of ≈ 0.65 shows that it missed almost a third of all available images, indicating we failed to generalize the network for these classes.

For *elliptical*, *Newton rings,* and *asymmetric* images, the recall of 0.2207 to 0.4836 shows that these images were a lot harder to find, observable in the relatively low-precision scores for those classes. *Elliptical* is the only class of these three where precision is high enough for using the neural network as a predictor. For the *Newton rings* and *asymmetric* class, with precision scores around 0.6, the neural network is effectively guessing.

The performance of all variants clearly shows the general good classification capabilities of a convolutional deep neural networks in the use case of diffraction patterns. Even the lowest performing neural network can outperform previous classification approaches by a large margin; compare with Ref. [27]. In particular, the results of ResNet18 are compelling; it is small, easy to train, and has relatively low complexity. Although having only a fraction of trainable parameters, it performed almost always on par with the much more complex

VGG architectures and all this while taking only 0.2 h for reaching the maximum accuracy during training.

Therefore, we chose the ResNet18 layout as the default configuration for all the following experiments; it is an ideal compromise among complexity, training time, and classification accuracy.

APPENDIX B: DERIVATION OF THE BINARY CROSS ENTROPY

Here, we give an derivation for the binary cross entropy [Eq. (6) in the main text]. We start with the most general form of the cross entropy given by

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p\|q), \tag{B1}$$

where $H(p)$ is the Shannon entropy of p and $D_{\mathrm{KL}}(p\|q)$ is the Kullback-Leibler divergence of p and q [56]. This is equivalent to

$$H(p, q) = -\sum_i p_i \log q_i, \tag{B2}$$

where p_i and q_i are two probability distributions over the same set of events. p_i is the "correct" distribution, and q_i is the approximation of p_i from the deep neural network. Since we are using a Bernoulli distribution as our probabilistic model, there are only two outcomes that one event (k) can have: $k \in \{0, 1\}$. The probability for both outcomes of one event and of both distributions can be written as

$$p(x) = \begin{cases} y(x) & \text{if } k = 1 \\ 1 - y(x) & \text{if } k = 0 \end{cases},$$

$$q(x) = \begin{cases} \hat{y}(x) & \text{if } k = 1 \\ 1 - \hat{y}(x) & \text{if } k = 0 \end{cases}.$$

x is some event, y is the ground truth label, and $\hat{y}$ is the approximate probability assigned by the deep neural network. Since we are using a sigmoid function at the output of our deep neural network, we can simplify Eq. (B2). Using

$$\hat{y}(x)_{\mathrm{sigmoid}} = \frac{1}{1 + \exp{(-x)}},$$

we can write

$$H(p, q) = -\sum_i^2 p_i \log q_i,$$

$$= -y(x) \log{[\hat{y}(x)]} - [1 - y(x)] \log{[1 - \hat{y}(x)]}$$

$$= -y(x) \log{\left[\frac{1}{1 + \exp{(-x)}} \right]} - [1 - y(x)]$$

$$\times \log{\left[1 - \frac{1}{1 + \exp{(-x)}} \right]}$$

$$\vdots$$

$$= x - xy(x) + \log{[1 + \exp{(-x)}]},$$

where x is an event (e.g., the activation in the output layer of the deep neural network) and y is the real label of this event.

APPENDIX C: FURTHER BUILDING BLOCKS OF DEEP NEURAL NETWORKS

This section describes the pooling layer and the batch normalization layer in more detail. Since these components are not critical for the neural network, their explanation is given here.

1. Pooling

There are two commonly used variants of pooling layers, the max pool and the average pool. The idea is to reduce the dimensionality of the output from a preceding layer [dim $\boldsymbol{x} = (N \times X)$] by letting a filter with size $a \times a$ slide over parts of the image with step size b, called stride, and let them perform a down-sample operation.

A max pool filter only takes the maximum value, and an avg pool filter averages over all values, within its perceptive field [83,84]; this process is equivalent to a convolutional operation but instead of a matrix multiplication with a convolutional kernel the pooling operation is carried out.

2. Batch normalization

Every layer within a deep neural network is to some point modeling the probability distribution given to it by its preceding layer. It is a hierarchical regression problem, which becomes harder if one layer changes key characteristics of the modeled probability distribution (e.g., the mean, variance, or kurtosis). This shift is then further multiplied in every succeeding layer and is therefore dependent on the depth of the network. This phenomenon is called a covariate shift [89]. Although this problem is solved in a deep neural network via domain adaptation, the costs of a covariate shift are usually much longer training times and reduced accuracy [90].

For this reason, a batch normalization layer (bn) is used to shift the mean of the mini-batch input to zero and to set the variance to one. This significantly reduces the amount of training time and increases accuracy [91]. This consists of four steps after which a normalized minibatch is returned:

(1) Calculated the minibatch mean:

$$\mu_{\mathrm{mb}} = \frac{1}{m} \sum_{i=1}^m x_i.$$

(2) Calculated the minibatch variance:

$$\sigma_{\mathrm{mb}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathrm{mb}})^2.$$

(3) Normalize:

$$\hat{x}_i = \frac{x_i - \mu_{\mathrm{mb}}}{\sqrt{\sigma_{\mathrm{mb}}^2 + \epsilon}}.$$

(4) Scale and shift according to adjustable parameter:

$$y_i = \gamma \hat{x}_i + \beta,$$

where y_i is the normalized output of input x_i and γ and β are adjustable parameters.

APPENDIX D: GRADCAM++

In Sec. VI of the main text, we show what the neural network deemed the most relevant areas within an input image. We calculated these so-called heat maps with an algorithm called GRADCAM++. The main idea is based on CAM [75] and GRADCAM [74] and allows for a very intuitive explanation for the decisions made by a convolutional deep neural network [40].

The core principle is that the output of a convolutional deep neural network can be expressed as a linear combination of the globally average pooled feature maps of the last convolutional layer:

$$Y^c = \sum_k w_k^c \sum_i \sum_j A_{ij}^k,$$

where A_{ij}^k is one feature map of all k maps from the last convolutional layer and w_k^c are the weights for a particular class prediction c of feature map k. Y^c is the predicted probability that the input image belongs to this certain class c. In the GRADCAM++ formalism, the weights can be calculated:

$$w_k^c = \sum_i \sum_j a_{ij}^{kc} \, h_{\mathrm{LR}} \left(\frac{\partial Y^c}{\partial A_{ij}^k} \right), \tag{D1}$$

where a_{ij}^{kc} are the gradient weights and h_{LR} is the *LeakyRelu* activation function from Eq. (4) in the main text. a_{ij}^{kc} depends only on A_{ij}^k and Y^c via

$$a_{ij}^{kc} = \frac{\frac{\partial^2 Y^c}{\left(\partial A_{ij}^k \right)^2}}{2 \frac{\partial^2 Y^c}{\left(\partial A_{ij}^k \right)^2} + \sum_a \sum_b A_{ab}^k \frac{\partial^3 Y^c}{\left(\partial A_{ij}^k \right)^3}}.$$

The final heat map, often called the saliency map, can then be obtained:

$$L_{ij}^c = h_{\mathrm{LR}} \left(\sum_k w^c A_{ij}^k \right). \tag{D2}$$

So the algorithm propagates an image forward through the network, then calculates the gradients until the last convolutional layer, and, using Eqs. (D1) and (D2), obtains a heat map of the areas within the input image that shows the gradient flow from the convolutional layer.

[1] M. M. Seibert, T. Ekeberg, F. R. N. C. Maia, M. Svenda, J. Andreasson, O. Jönsson, D. Odić, B. Iwan, A. Rocker, D. Westphal *et al.*, Nature (London) **470**, 78 (2011).

[2] N. D. Loh, C. Y. Hampton, A. V. Martin, D. Starodub, R. G. Sierra, A. Barty, A. Aquila, J. Schulz, L. Lomb, J. Steinbrener *et al.*, Nature (London) **486**, 513 (2012).

[3] C. Bostedt, M. Adolph, E. Eremina, M. Hoener, D. Rupp, S. Schorb, H. Thomas, A. R. B. de Castro, and T. Möller, J. Phys. B: At. Mol. Opt. Phys. **43**, 194011 (2010).

[4] L. F. Gomez, K. R. Ferguson, J. P. Cryan, C. Bacellar, R. M. P. Tanyag, C. Jones, S. Schorb, D. Anielski, A. Belkacem, C. Bernando *et al.*, Science **345**, 906 (2014).

[5] H. N. Chapman and K. A. Nugent, Nat. Photon. **4**, 833 (2010).

[6] D. Rupp, N. Monserud, B. Langbehn, M. Sauppe, J. Zimmermann, Y. Ovcharenko, T. Möller, F. Frassetto, L. Poletto, A. Trabattoni *et al.*, Nat. Commun. **8**, 493 (2017).

[7] Z. Y. Li, N. P. Young, M. Di Vece, S. Palomba, R. E. Palmer, A. L. Bleloch, B. C. Curley, R. L. Johnston, J. Jiang, and J. Yuan, Nature (London) **451**, 46 (2008).

[8] J. Farges, M. F. de Feraudy, B. Raoult, and G. Torchet, J. Chem. Phys. **84**, 3491 (1986).

[9] D. E. Clemmer and M. F. Jarrold, J. Mass Spectrom. **32**, 577 (1997).

[10] O. Kostko, B. Huber, M. Moseler, and B. von Issendorff, Phys. Rev. Lett. **98**, 043401 (2007).

[11] A. Sakdinawat and D. Attwood, Nat. Photon. **4**, 840 (2010).

[12] C. Bostedt, H. N. Chapman, J. T. Costello, J. R. C. López-Urrutia, S. Düsterer, S. W. Epp, J. Feldhaus, A. Föhlisch, M. Meyer, T. Möller *et al.*, Nucl. Instr. Methods Phys. Res., Sect. A **601**, 108 (2009).

[13] T. Gorkhover, M. Adolph, D. Rupp, S. Schorb, S. W. Epp, B. Erk, L. Foucar, R. Hartmann, N. Kimmel, K.-U. Kühnel *et al.*, Phys. Rev. Lett. **108**, 245005 (2012).

[14] C. Bostedt, T. Gorkhover, D. Rupp, M. Thomas, and T. Möller, in *Synchrotron Light Sources and Free-Electron Lasers*, edited by E. Jaeschke, S. Khan, R. J. Schneider, and B. J. Hastings (Springer International, Cham, 2016), 1st ed., pp. 1–38.

[15] P. Emma, R. Akre, J. Arthur, R. Bionta, C. Bostedt, J. Bozek, A. Brachmann, P. Bucksbaum, R. Coffee, F.-J. Decker *et al.*, Nat. Photon. **4**, 641 (2010).

[16] C. Bostedt, S. Boutet, D. M. Fritz, Z. Huang, H. J. Lee, H. T. Lemke, A. Robert, W. F. Schlotter, J. J. Turner, and G. J. Williams, Rev. Mod. Phys. **88**, 015007 (2016).

[17] G. D. Calvey, A. M. Katz, C. B. Schaffer, and L. Pollack, Struct. Dyn. (Melville, N.Y.) **3**, 054301 (2016).

[18] E. A. Schneidmiller, Photon beam properties at the European XFEL, Tech. Rep. (XFEL, Hamburg, 2011).

[19] D. Rupp, M. Adolph, L. Flückiger, T. Gorkhover, J. P. Müller, M. Müller, M. Sauppe, D. Wolter, S. Schorb, R. Treusch *et al.*, J. Chem. Phys. **141**, 044306 (2014).

[20] I. Barke, H. Hartmann, D. Rupp, L. Flückiger, M. Sauppe, M. Adolph, S. Schorb, C. Bostedt, R. Treusch, C. Peltz *et al.*, Nat. Commun. **6**, 6187 (2015).

[21] I. V. Lundholm, J. A. Sellberg, T. Ekeberg, M. F. Hantke, K. Okamoto, G. van der Schot, J. Andreasson, A. Barty, J. Bielecki, P. Bruza *et al.*, IUCrJ **5**, 531 (2018).

[22] J. Flamant, N. Le Bihan, A. V. Martin, and J. H. Manton, Phys. Rev. E **93**, 053302 (2016).

[23] T. Ekeberg, M. Svenda, C. Abergel, F. R. Maia, V. Seltzer, J.-M. Claverie, M. Hantke, O. Jönsson, C. Nettelblad, G. van der Schot *et al.*, Phys. Rev. Lett. **114**, 098102 (2015).

[24] K. He, X. Zhang, S. Ren, and J. Sun, arXiv:1603.05027.

[25] N. Meidinger, R. Andritschke, R. Hartmann, S. Herrmann, P. Holl, G. Lutz, and L. Strüder, Nucl. Instr. Methods Phys. Res., Sect. A **565**, 251 (2006).

[26] R. P. Kurta, M. Altarelli, and I. A. Vartanyants, in *Advances in Chemical Physics*, edited by S. A. Rice and A. R. Dinner (John Wiley & Sons, Hoboken, NJ, 2016), pp. 1–39.

[27] S. A. Bobkov, A. B. Teslyuk, R. P. Kurta, O. Y. Gorobtsov, O. M. Yefanov, V. A. Ilyin, R. A. Senin, and I. A. Vartanyants, J. Synchrotron Radiat. **22**, 1345 (2015).

[28] A. Atla, R. Tada, V. Sheng, and N. Singireddy, in *Journal of Computing Sciences in Colleges*, Vol. 26 (Consortium for Computing Sciences in Colleges, 2011), pp. 96–103.

[29] C. H. Yoon, P. Schwander, C. Abergel, I. Andersson, J. Andreasson, A. Aquila, S. Bajt, M. Barthelmess, A. Barty, M. J. Bogan *et al.*, Opt. Express **19**, 16542 (2011).

[30] D.-L. Deng, X. Li, and S. Das Sarma, Phys. Rev. B **96**, 195145 (2017).

[31] P. Zhang, H. Shen, and H. Zhai, Phys. Rev. Lett. **120**, 066401 (2018).

[32] R. D. Field, Y. Kanev, M. Tayebnejad, and P. A. Griffin, Phys. Rev. D **53**, 2296 (1996).

[33] W. Bhimji, S. A. Farrell, T. Kurth, M. Paganini, Prabhat, and E. Racah, arXiv:1711.03573.

[34] K. Mills, M. Spanner, and I. Tamblyn, Phys. Rev. A **96**, 042113 (2017).

[35] S. Manzhos, K. Yamashita, and T. Carrington, Chem. Phys. Lett. **474**, 217 (2009).

[36] Y. Zhu, Q. Ouyang, and Y. Mao, BMC Bioinformat. **18**, 348 (2017).

[37] Z. Gao, L. Wang, L. Zhou, and J. Zhang, IEEE J. Biomed. Heal. Informat. **21**, 416 (2017).

[38] T. W. Ke, A. S. Brewster, S. X. Yu, D. Ushizima, C. Yang, and N. K. Sauter, J. Synchrotron Radiat. **25**, 655 (2018).

[39] S. Kassemeyer, J. Steinbrener, L. Lomb, E. Hartmann, A. Aquila, A. Barty, A. V. Martin, C. Y. Hampton, S. Bajt, M. Barthelmess *et al.*, Opt. Express **20**, 4149 (2012); https://www.cxidb.org/id-10.html

[40] A. Chattopadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, arXiv:1710.11063.

[41] B. Langbehn, K. Sander, Y. Ovcharenko, C. Peltz, A. Clark, M. Coreno, R. Cucini, M. Drabbels, P. Finetti, M. Di Fraia *et al.*, Phys. Rev. Lett. **121**, 255301 (2018).

[42] V. Lyamayev, Y. Ovcharenko, R. Katzy, M. Devetta, L. Bruder, A. LaForge, M. Mudrich, U. Person, F. Stienkemeier, M. Krikunova *et al.*, J. Phys. B: At. Mol. Opt. Phys. **46**, 164007 (2013).

[43] C. Svetina, C. Grazioli, N. Mahne, L. Raimondi, C. Fava, M. Zangrando, S. Gerusina, M. Alagia, L. Avaldi, G. Cautero *et al.*, J. Synchrotron Radiat. **22**, 538 (2015).

[44] E. Allaria, R. Appio, L. Badano, W. Barletta, S. Bassanese, S. Biedron, A. Borga, E. Busetto, D. Castronovo, P. Cinquegrana *et al.*, Nat. Photon. **6**, 699 (2012).

[45] B. Frénay and A. Kabán, in *Proceedings of the European Symposium on Artificial Neural Networks*, Computational Intelligence and Machine Learning, Bruges, Belgium (2014).

[46] F. R. N. C. Maia, Nat. Methods **9**, 854 (2012); http://cxidb.org/id-94.html

[47] M. A. Arbib, *Brains, Machines, and Mathematics* (Springer-Verlag, Berlin, 1987), p. 202.

[48] V. Nair and G. E. Hinton, in *Proceedings of the 27th International Conference on Machine Learning* (Omnipress, Madison, WI, 2010), pp. 807–814.

[49] A. L. Maas, A. Y. Hannun, and A. Y. Ng, in *Proceedings of the 30th International Conference on Machine Learning*, Vol. 30 (2013), p. 3.

[50] J. Schmidhuber, Neural Networks **61**, 85 (2014).

[51] Y. LeCun, Y. Bengio, and G. Hinton, Nature (London) **521**, 436 (2015).

[52] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, arXiv:1602.07261.

[53] G. Hinton, O. Vinyals, and J. Dean, arXiv:1503.02531.

[54] C. E. Shannon, Bell Syst. Tech. J. **27**, 379 (1948).

[55] S. Kullback and R. A. Leibler, Ann. Math. Stat. **22**, 79 (1951).

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, 2016), p. 775.

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Nature (London) **323**, 533 (1986).

[58] K. He, X. Zhang, S. Ren, and J. Sun, arXiv:1512.03385.

[59] S. Zagoruyko and N. Komodakis, arXiv:1605.07146.

[60] H. Zou, H. Zou, and T. Hastie, J. R. Stat. Soc., Ser. B **67**, 301 (2005).

[61] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, arXiv:1207.0580.

[62] L. Perez and J. Wang, arXiv:1712.04621.

[63] S. Ikeda and H. Kono, Opt. Express **20**, 3375 (2012).

[64] T. Shintake, Phys. Rev. E **78**, 041906 (2008).

[65] B. Hammouda, J. Appl. Crystallogr. **43**, 716 (2010).

[66] S. K. Sinha, E. B. Sirota, S. Garoff, and H. B. Stanley, Phys. Rev. B **38**, 2297 (1988).

[67] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, arXiv:1511.07289.

[68] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, arXiv:1409.4842.

[69] J. F. Kolen and S. C. Kremer, *A Field Guide to Dynamical Recurrent Networks* (IEEE, 2001).

[70] A. Nguyen, J. Yosinski, and J. Clune, arXiv:1412.1897.

[71] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, arXiv:1610.08401.

[72] R. P. Kurta, J. J. Donatelli, C. H. Yoon, P. Berntsen, J. Bielecki, B. J. Daurer, H. DeMirci, P. Fromme, M. F. Hantke, F. R. N. C. Maia *et al.*, Phys. Rev. Lett. **119**, 158102 (2017).

[73] J. J. Donatelli, P. H. Zwart, and J. A. Sethian, Proc. Natl. Acad. Sci. USA **112**, 10286 (2015).

[74] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, in *Proceedings of the 2017 IEEE International Conference on Computer Vision* (IEEE, 2017), pp. 618–626.

[75] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, arXiv:1512.04150.

[76] K. Li, Z. Wu, K.-C. Peng, J. Ernst, and Y. Fu, arXiv:1802.10171.

[77] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, arXiv:1805.08318.

[78] T. Miyato and M. Koyama, arXiv:1802.05637.

[79] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, arXiv:1802.05957.

[80] I. Goodfellow, arXiv:1701.00160.

[81] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, arXiv:1606.03657.

[82] https://github.com/julian-carpenter/airynet

[83] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Proc. IEEE **86**, 2278 (1998).

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, Vol. 1 (Curran Associates Inc., 2012), pp. 1097–1105.

[85] K. Simonyan and A. Zisserman, arXiv:1409.1556.

[86] R. Eldan and O. Shamir, arXiv:1512.03965.

[87] A. Veit, M. Wilber, and S. Belongie, arXiv:1605.06431.

[88] S. Li, J. Jiao, Y. Han, and T. Weissman, arXiv:1611.01186.

[89] H. Shimodaira, J. Stat. Plan. Inference **90**, 227 (2000).

[90] J. Jiang, *A Literature Survey on Domain Adaptation of Statistical Classifiers* (2008).

[91] S. Ioffe and C. Szegedy, arXiv:1502.03167.