X-ray Photon Fluctuation Spectroscopy Analysis using Convolutional Neural Networks

Sathya R. Chitturi
Department of Materials Science and Engineering
Stanford University / SLAC National Accelerator Laboratory
chitturi@slac.stanford.edu

Abstract

In this contribution, we apply Convolutional Neural Networks (CNNs) to study X-ray Photon Fluctuation Spectroscopy (XPFS) data. We use a DnCNN based architecture for background noise removal and show that it significantly outperforms the standard experimental baseline of manual thresholding on simulation data and appears to be qualitatively better on experimental data. We also find that this relative improvement increases with decreasing SNR. Furthermore, we show that the capacity to better denoise the background leads to a better estimate of the unknown photon ADU for the experiment. Finally, we also propose and develop a U-net with a novel photon conserving loss term which, according to preliminary results, appears outperform a standard U-net for the photon assignment task.

I. Introduction

A. Background

X-ray Photon Fluctuation Spectroscopy is a newly developed (at LCLS) coherent X-ray imaging technique used to probe ultrafast materials dynamics [1]–[3]. Data from this experiment comes in the form of 2D arrays which amounts to single-photons hitting a detector. Each frame represents a noisy measurement of what is known as the intrinsic photon map. The photon map is a discrete distribution over photon counts. More explicitly, each pixel records the number of photons that hit it during the measurement. The underlying probability distribution for the data is known to follow a negative binomial distribution [4], parameterized by scalars known as the contrast and the mean photon count. The physical insight from these measurements is the contrast and is extracted by accurately fitting the negative binomial distribution. Here it is worth noting that this measurement is an exquisitely sensitive statistical technique which requires tens of thousands of shots to obtain a good estimate for the contrast [4].

The overarching goal of the analysis pipeline is to recover photon maps from measurements – in essence, this problem is a denoising and deconvolution task. This problem is challenging for a number of reasons. One reason is that the non-linear blurring kernel is unknown. In fact, the data often has multiple unknown convolutional blurring kernels for the charge clouds; unfortunately, these charge smearing functions can range in size between 1 pixel and 25 pixels. In addition, for the experimental data, there are no ground truth labels and therefore any supervised data analysis pipelines will require a simulator for training data.

B. Motivation and Goals

In this project, we focus on two tasks related to XPFS analysis: the aforementioned direct photon map prediction and background denoising. Here, it is worth mentioning that the second task is useful for a number of reasons. The first is that an accurate denoising algorithm might be useful as a first stage procedure, before any deconvolution. The second is that it might help experimentalists better recover the true photon ADU from the experiment. This approach works by segregating pixels (droplets) corresponding to a one-photon events and fitting the experimental droplet histogram [3]. In cases where the charge smearing is high, this task is often very difficult as there are no primary peaks in the droplet ADU histogram. We approach both tasks using a supervised CNN pipeline based on simulated data. Here it is worth mentioning that although there are previous algorithms for both tasks, we choose to use a CNN/GPU pipeline due to the potential for faster and more accurate results.

II. Relevant Work

The forward simulation model that we use is based on an implementation given in [3]. For the photon map prediction task, there are other methods that have been previously developed such as Greedy Guess and Greedy Guess - Least Squares [3], [5], [6]. The main benefit of these algorithms are that they are unsupervised (although some hand-tuning is required to get them to perform well). The main limitation is that these algorithms are relatively slow and are ill-suited to data which exhibits higher noise and more charge smearing; crucially, they are very slow compared to the shot rate at LCLS. For the denoising task, the current experimental pipelines simply use a manual thresholding for the noise [3]. Although this method is extremely fast, it does lead to unphysically reduced signal.

In terms of the machine learning methodology, we leverage standard tools built in the literature for supervised computational imaging. These involve using the DnCNN [7] and U-net architectures [8] as well as using loss functions such as the Focal Loss [9].

III. Methods

In this work, the focused was on two problems: background denoising and photon map prediction. The methodology for these tasks are organized into the following sections below. Note, the methods were validated on simulation data for which a ground truth exists [3]. Where possible, these methods were also applied to real LCLS data from experiments.

A. Background Denoising

1) Probabilistic Modelling of Background Noise: In order to model the background noise for the simulator, we isolated a corner of the detector which did not experience scattering. This corner essentially formed a "dark shot" from which we were able to build an empirical histogram of the noise. We used the empirical probability density as our noise generating model (Figure 1). Thus, our image formation model was the sum of the forward simulation model and the stochastic noise generation. Here, it is worth mentioning that the previous simulator noise model was gaussian which does not fit the empirical distribution well.

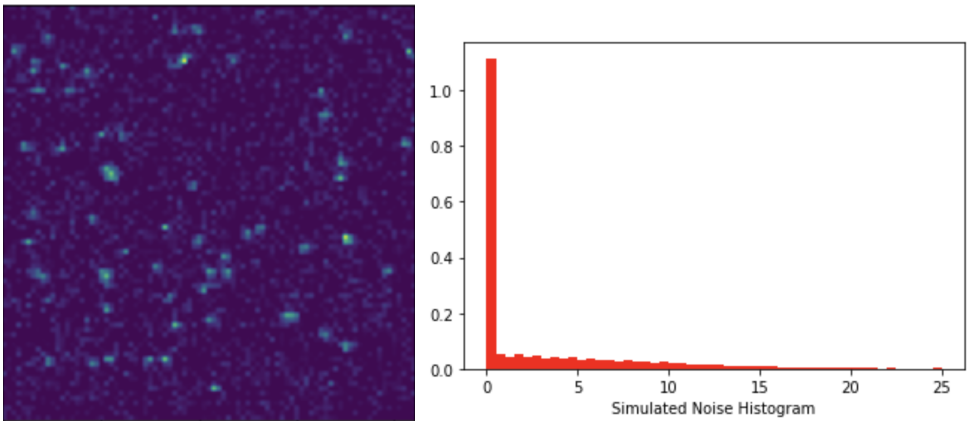

Fig. 1. Example of experimental data (left) and corresponding empirical noise histogram (right) obtained from using the left corner of the experimental data.

2) DnCNN Training and Deployment: We followed a standard DnCNN training procedure in which the model was trained on 10000 noisy simulation images and trained to predict the residual of the raw image and the denoised image – i.e. to predict the noise (Figure 2). In particular, we modified code from an off-the-shelf implementation [7], to fit our purpose; more details about the model can be found in Figure 6. At test time, we obtained the denoised image by subtracting the residual from the input. This analysis was conducted on both experimental data and simulated test data.

We quantitatively compared our simulation denoising results against those obtained by manually thresholding the data at a fixed ADU level. To ensure a fair comparison, we used a number of different sensible manual threshold

values (between 10 - 30 ADU) and used the average to compare against the machine learning. We report the improvement of the DnCNN as the ratio between the mean PSNR for the average manual thresholding result against the mean PSNR from the DnCNN pipeline. We also studied the effect of different signal strengths on the DnCNN denoising performance relative to manual thresholding. More specifically, we kept the noise distribution constant and scaled the intensity by 0.5, 1.0 and 2.0.

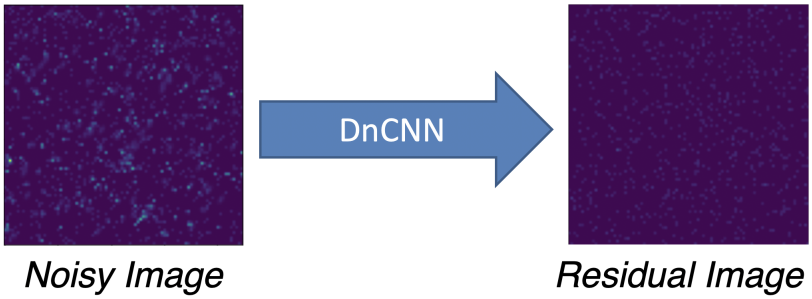

Fig. 2. Schematic of DnCNN denoising procedure.

3) Estimation of Photon ADU: We tested the performance of manual thresholding against the DnCNN for the task of photon ADU prediction. In order to obtain an estimate of the photon ADU from the denoised image, we used a previously written algorithm [3] which seeks to sum the total intensity around a region determined to contain one photon. In this analysis, we qualitatively compare the droplet photon ADU histograms for simulated data where the ground truth is known.

B. Photon Map Prediction

The photon map prediction task involves predicting a discrete photon map which counts per-pixel photon counts from the raw photon ADU image. We had previously worked on this problem using a U-net formulation. In this method, we treated the problem as a semantic segmentation problem where each pixel contains a class (between 0 and 6) which corresponds to the number of photons which hit that pixel. This model was trained by minimizing $L(p_i, \hat{p}_i)$ with respect to the model parameters. The base model was modified from an off-the-shelf implementation in Keras [10]. More details about the base model can be found in Figure 7.

$$L(p_i, \hat{p}_i) = -\sum_{i=1}^{\mathrm{N}} p_i \cdot \log \hat{p}_i \tag{1}$$

We had determined that the task was challenging for high-resolution data due to the issues of spatial sparsity and heavy class imbalance. Therefore, the goal of this section was to explore methods to tackle these two issues. We tested a number of different methods to tackle class-imbalance and sparsity (described below). Our total training dataset size was 40000 and our results were validated on a testing set of 5000.

- Artificial Data: We add artificial data to the dataset (50 %) in order to over-represent rare classes. The intuition here is that we want the CNNs to see more examples with higher photon counts in order to recognize them better in the data.
- Sample Class Weighting: We weight each pixel by the reciprocal of the weight for the corresponding class. The maximum weight for any pixel was arbitrarily decided to be 10000.
- Focal Loss: Instead of using a Cross-Entropy Loss Function (CEL), we use a Focal Loss Function which essentially adds an annealing term to the CEL which down-weights pixels for which the model is already confident. This method has previously been shown to substantially improve class-imbalanced semantic segmentation tasks [9].
- Photon Conserving Loss: We developed an additional loss constraint to tackle the class-imbalance problem. This term essentially bins the data (sum pooling operation) and then calculates the mean squared loss on

this additional term. For this work, we considered a 2x and 3x binning operation. In the equation, $v^T = [0, 1, 2, 3, 4, 5, 6]$ and is used to obtain the number of photons per pixel from the corresponding softmax vector. The reason why it is termed Photon Conserving is because it seeks to enforce that the neural network will conserve the total number of photons after a sum pooling operation; this is the computational analogue of increasing the pixel size in hardware.

$$G(p_i, \hat{p}_i) = -\sum_{i=1}^{\mathrm{N}} p_i \cdot \log \hat{p}_i + ||\mathrm{Bin}(v^T p_i) - \mathrm{Bin}(v^T \hat{p}_i)||_F \tag{2}$$

Therefore, in this equation, the optimal solution is found my minimizing G with respect to the model parameters.

C. Simulation Dataset Description

For the background denoising task, we use the following directions to generate a training dataset of 10000 image / noise pairs. Each image was generated by sampling a random contrast value in the range [0.1, 0.9], $\bar{k}$ (average photon count) in the range [0.025, 0.50], and with a random distribution of charge cloud radii σ_g [0.25,0.35,0.45,0.55, 0.1, 0.1], which is based on the following probability distribution [0.15, 0.35, 0.35, 0.15, 0.0, 0.0]. The noise was generated by fitting a corresponding experimental histogram as described above.
For the photon map prediction task, we use the following directions to generate a training dataset of 40000 image / photon map pairs. Each image was generated by sampling a random contrast value in the range [0.1, 0.9], $\bar{k}$ (average photon count) in the range [0.1, 2.0], and with a random distribution of charge cloud radii σ_g [0.1,0.25,0.35,0.45,0.55,0.6], which is based on the following probability distribution [0.25, 0.15, 0.1, 0.3, 0.15, 0.05]. For more information on the simulator, please refer to [3].

IV. Results and Analysis

A. Background Denoising

We applied a DnCNN model to denoise simulated data for which we had explicitly modelled the noise to mimic real data. We find that we are able to obtain a denoised image with significantly better qualitative appearance relative to manual thresholding (Figure 3). Visually, the DnCNN results appears to closely match the ground truth simulation data.

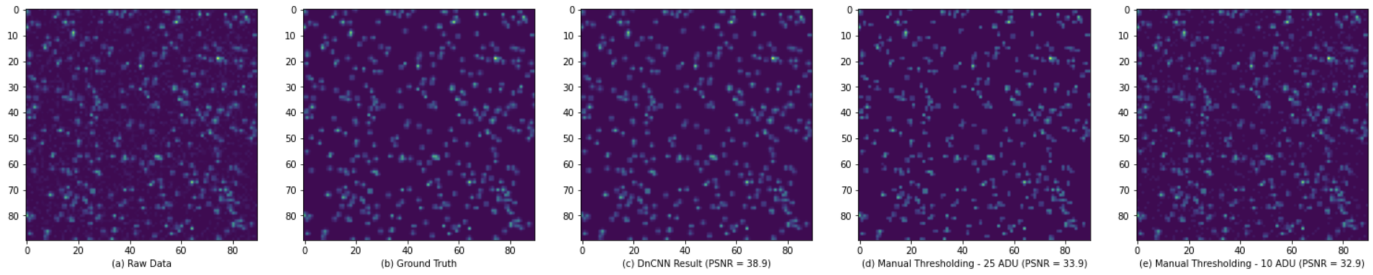

Fig. 3. Comparison of DnCNN (c) and Manual Thresholding (d, e) for the background denoising task. The raw image is shown in (a) and the ground truth is shown in (b). For the manual thresholding, we chose two plausible choices for the cut-off (10 and 25 ADU).

We also investigated the performance of the DnCNN on real experimental data which corresponds to the simulation forward model and the generative noise model. Although there is no ground truth, we see that the DnCNN better preserves peak shape and is more successful at separating signal from noise (Figure 4).

Finally, we used the denoised photon maps to calculate the photon droplet histogram in order to estimate the ADU of the incoming photons. We see that the denoised image gives a much closer approximation to the ground truth for the single photon ADU count (first peak) relative to manual thresholding (Figure 5). Interestingly, note that the peak width for the DnCNN changes for higher photon events while the peak width stays constant for manual thresholding. This will be explained further in the discussion section.

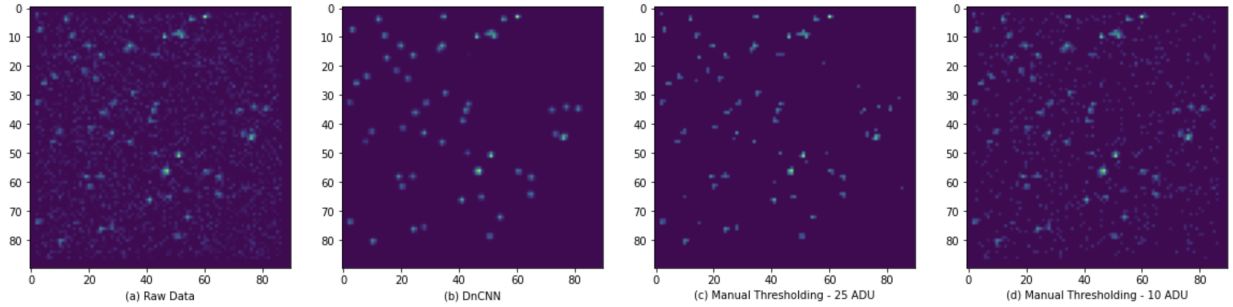

Fig. 4. Comparison of DnCNN (b) and Manual Thresholding (c, d) for the background denoising task. The raw image is shown in (a). Note, the manual thresholding sometimes gives unphysical peak shapes (c) and fails to remove all the noise (d).

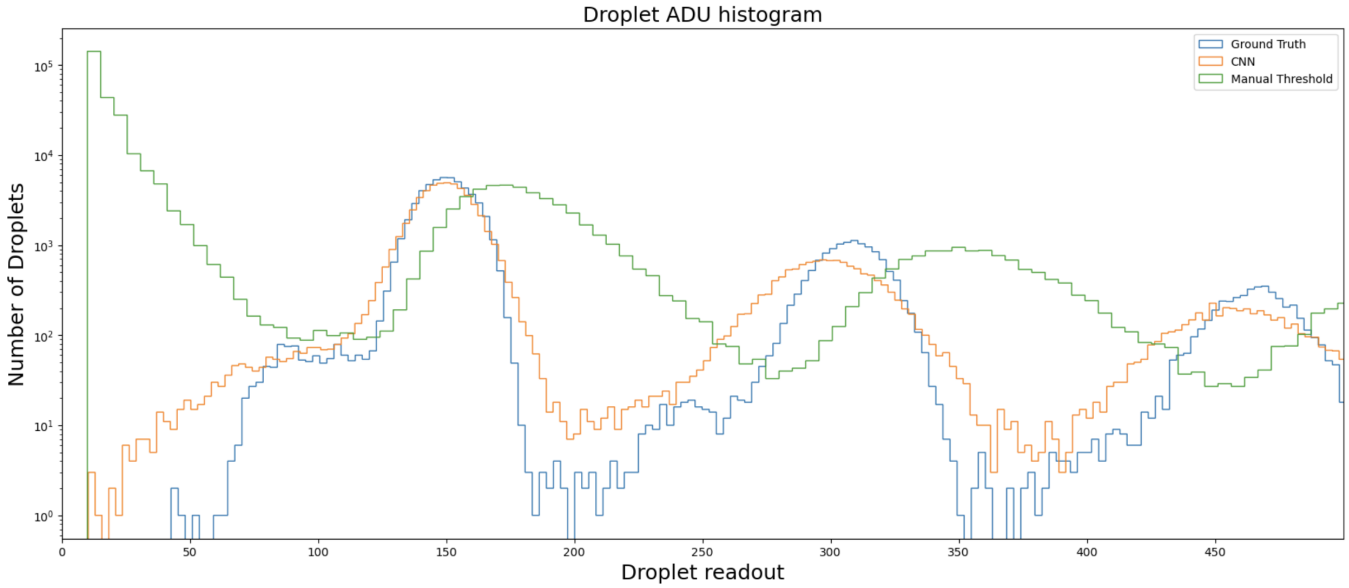

Fig. 5. Droplet photon histograms for denoised image via DnCNN and manual thresholding. These results are compared to a ground truth simulator histogram.

We also studied the performance of manual thresholding vs DnCNN as the overall signal strength was scaled by a constant (0.5, 1.0 and 2.0). In this formulation, we keep the overall noise level constant and generate it using the previously described empirical probability distribution. We quantify performance using the mean PSNR for the denoised images against the ground truth images. In addition, we also calculate the ratio of the PSNRs in order to analyze the relative performance of the two algorithms as a function of signal strength (Table I).

Signal Strength Factor	Mean PSNR (DnCNN)	Mean PSNR (Manual Thresholding)	Ratio
0.5	42.0	30.7	1.36
1.0	46.8	37.5	1.25
2.0	51.6	43.8	1.18

TABLE I
QUANTITATIVE COMPARISON OF MANUAL THRESHOLDING VS DNCNN DENOISING AS A FUNCTION OF SNR.

We find that the DnCNN outperforms manual thresholding for each signal strength and that furthermore, the relative performance increases with decreasing SNR.

Method	Macro F1 Score
U-Net	0.67
U-Net + Sample weighting (inverse class prevalence)	0.48
U-Net + Artificial Oversampling	0.67
U-Net + Focal Loss Function [4]	0.61
U-Net + Photon Conserving Loss Penalty (3x)	0.68
U-Net + Photon Conserving Loss Penalty (2x)	0.68

TABLE II
COMPARISON OF DIFFERENT METHODS TO TACKLE CLASS-IMBALANCE.

B. Photon Map Prediction

For the photon map prediction task, we compare a number of different methods known to be effective at combating sparsity and dataset class imbalance. For each method, the U-net architecture remains constant. We quantify the performance using the macro F1 score in order to better weight rare classes (Table II).

We find that methods such as including sample weighting or changing the loss to the Focal loss function do not appear to work well relative to a control. However, we do see a slight improvement in the F1 macro score by training with the photon conserving loss term.

V. DISCUSSION

A. Background Denoising

For the background denoising task, it makes intuitive sense why a DnCNN approach might be more optimal than simple manual thresholding. The DnCNN explicitly learns to model the noise distribution, whereas the manual thresholding simply has a hard-cutoff. For instance, the DnCNN is likely cautious when denoising areas which appear to represent peaks (i.e. connected regions) while the manual thresholding will remove signal if it is less than the specified threshold. Here, it is worth mentioning that we considered other algorithms such as Weiner filtering as a baseline. However, we judged that such methods solve an optimization problem for each image and would therefore be ill-equipped to deal with the massive data volume from LCLS.

For the Droplet ADU histogram calculation, we noticed an interesting trend that the peak width increases for the DnCNN denoised images for high photon count events. We believe this is likely a consequence of the DnCNN being less likely to denoise regions with larger intensities as it might incorrectly reduce signal. Since we have a class-imbalanced dataset, it is likely that the DnCNN did not encounter as many droplets with higher photon counts and therefore its response is more conservative. On the other hand, since manual thresholding involves a hard cut-off, we would expect that higher order peaks should have similar widths.

Finally, for the PSNR analysis, it is clear that the DnCNN outperforms the manual thresholding (at least with the PSNR metric). In general, it seems that the DnCNN has relatively better performance for low SNR data. This is consistent with intuition. As the SNR decreases, manual thresholding should find it much harder to separate signal from noise. However, another important consideration is whether either algorithm incorrectly deletes full peaks, since this will ultimately affect the statistical interpretation of the data. Here, further analysis and a different metric will be needed in order to claim superiority of the DnCNN denoiser.

B. Photon Map Prediction

For the photon map prediction task, we employed a number of strategies aimed improving the prediction under sparsity and class-imbalance. While we found that our photon conserving regularization term did give the best performance, the performance was only slightly better than the control. The intuition behind the photon conserving loss was that it would give more signal in the downsampled space and therefore might allow us to better propagate gradients. Here it is worth emphasizing that it is possible that the result we obtain here is due to the issues of optimizing non-convex loss functions and that the improvement is not actually real. Further investigation is needed to quantify whether the photon conserving term leads to better performance. Additionally, one limitation that we experienced was that the dataset size exceeded the memory on the GPU. In order to train better models, we would likely need to exploit a multi-GPU architecture.

VI. Conclusions

In this work we have successfully developed a DnCNN based architecture for background denoising. This model appears to outperform the conventional experimental baseline of manual thresholding with regards to the PSNR metric. Development of such a tool will allow for better initial dataset processing as well as more accurate estimation of the true photon ADU. We have also investigated a number of different approaches to tackle class-imbalance and sparsity. Our photon conserving loss term appears to help in this task, relative to other conventional methods. However, much further analysis is required to validate this preliminary finding.

VII. Code Availability

All source code and data can be found here.

Acknowledgment

Thanks to Professor Gordon Wetzstein for his helpful advise and comments regarding this project. Thanks to Josh Turner, Nicolas Burdet and Professor Mike Dunne for supervision regarding XPFS. Thanks to Daniel Ratner and Youssef Nashed for supervision regarding machine learning.

References

[1] M. Seaberg, B. Holladay, J. Lee, M. Sikorski, A. Reid, S. Montoya, G. Dakovski, J. Koralek, G. Coslovich, S. Moeller *et al.*, "Nanosecond x-ray photon correlation spectroscopy on magnetic skyrmions," *Physical review letters*, vol. 119, no. 6, p. 067403, 2017.

[2] L. Shen, M. Seaberg, E. Blackburn, and J. Turner, "A snapshot review—fluctuations in quantum materials: from skyrmions to superconductivity," *MRS Advances*, vol. 6, no. 8, pp. 221–233, 2021.

[3] N. G. Burdet, V. Esposito, M. Seaberg, C. H. Yoon, and J. Turner, "Absolute contrast estimation for soft x-ray photon fluctuation spectroscopy using a variational droplet model," *Scientific Reports*, vol. 11, no. 1, pp. 1–9, 2021.

[4] J. W. Goodman, *Speckle phenomena in optics: theory and applications*. Roberts and Company Publishers, 2007.

[5] Y. Sun, J. Montana-Lopez, P. Fuoss, M. Sutton, and D. Zhu, "Accurate contrast determination for x-ray speckle visibility spectroscopy," *Journal of Synchrotron Radiation*, vol. 27, no. 4, pp. 999–1007, 2020.

[6] Y. Sun, V. Esposito, P. A. Hart, C. Hansson, H. Li, K. Nakahara, J. P. MacArthur, S. Nelson, T. Sato, S. Song *et al.*, "A contrast calibration protocol for x-ray speckle visibility spectroscopy," *Applied Sciences*, vol. 11, no. 21, p. 10041, 2021.

[7] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE transactions on image processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[8] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[10] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

VIII. Appendix

A. Machine Learning Model Architectures

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 90, 90, 1)]	0	
conv2d_7 (Conv2D)	(None, 90, 90, 64)	640	input_2[0][0]
activation_6 (Activation)	(None, 90, 90, 64)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 90, 90, 64)	36928	activation_6[0][0]
batch_normalization_5 (BatchNor	(None, 90, 90, 64)	256	conv2d_8[0][0]
activation_7 (Activation)	(None, 90, 90, 64)	0	batch_normalization_5[0][0]
conv2d_9 (Conv2D)	(None, 90, 90, 64)	36928	activation_7[0][0]
batch_normalization_6 (BatchNor	(None, 90, 90, 64)	256	conv2d_9[0][0]
activation_8 (Activation)	(None, 90, 90, 64)	0	batch_normalization_6[0][0]
conv2d_10 (Conv2D)	(None, 90, 90, 64)	36928	activation_8[0][0]
batch_normalization_7 (BatchNor	(None, 90, 90, 64)	256	conv2d_10[0][0]
activation_9 (Activation)	(None, 90, 90, 64)	0	batch_normalization_7[0][0]
conv2d_11 (Conv2D)	(None, 90, 90, 64)	36928	activation_9[0][0]
batch_normalization_8 (BatchNor	(None, 90, 90, 64)	256	conv2d_11[0][0]
activation_10 (Activation)	(None, 90, 90, 64)	0	batch_normalization_8[0][0]
conv2d_12 (Conv2D)	(None, 90, 90, 64)	36928	activation_10[0][0]
batch_normalization_9 (BatchNor	(None, 90, 90, 64)	256	conv2d_12[0][0]
activation_11 (Activation)	(None, 90, 90, 64)	0	batch_normalization_9[0][0]
conv2d_13 (Conv2D)	(None, 90, 90, 1)	577	activation_11[0][0]
subtract_1 (Subtract)	(None, 90, 90, 1)	0	input_2[0][0] conv2d_13[0][0]

Total params: 187,137
Trainable params: 186,497
Non-trainable params: 640

Fig. 6. DnCNN architecture used in this contribution.

Model: "U-Net"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 90, 90, 1)]	0	
conv2d (Conv2D)	(None, 90, 90, 16)	160	input_1[0][0]
batch_normalization (BatchNorma	(None, 90, 90, 16)	64	conv2d[0][0]
activation (Activation)	(None, 90, 90, 16)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 90, 90, 16)	2320	activation[0][0]
batch_normalization_1 (BatchNor	(None, 90, 90, 16)	64	conv2d_1[0][0]
activation_1 (Activation)	(None, 90, 90, 16)	0	batch_normalization_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 30, 30, 16)	0	activation_1[0][0]
conv2d_2 (Conv2D)	(None, 30, 30, 32)	4640	max_pooling2d[0][0]
batch_normalization_2 (BatchNor	(None, 30, 30, 32)	128	conv2d_2[0][0]
activation_2 (Activation)	(None, 30, 30, 32)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 30, 30, 32)	9248	activation_2[0][0]
batch_normalization_3 (BatchNor	(None, 30, 30, 32)	128	conv2d_3[0][0]
activation_3 (Activation)	(None, 30, 30, 32)	0	batch_normalization_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 10, 10, 32)	0	activation_3[0][0]
conv2d_4 (Conv2D)	(None, 10, 10, 64)	18496	max_pooling2d_1[0][0]
batch_normalization_4 (BatchNor	(None, 10, 10, 64)	256	conv2d_4[0][0]
activation_4 (Activation)	(None, 10, 10, 64)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 10, 10, 64)	36928	activation_4[0][0]
batch_normalization_5 (BatchNor	(None, 10, 10, 64)	256	conv2d_5[0][0]
activation_5 (Activation)	(None, 10, 10, 64)	0	batch_normalization_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0	activation_5[0][0]
conv2d_6 (Conv2D)	(None, 5, 5, 128)	73856	max_pooling2d_2[0][0]
batch_normalization_6 (BatchNor	(None, 5, 5, 128)	512	conv2d_6[0][0]
activation_6 (Activation)	(None, 5, 5, 128)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 5, 5, 128)	147584	activation_6[0][0]
batch_normalization_7 (BatchNor	(None, 5, 5, 128)	512	conv2d_7[0][0]
activation_7 (Activation)	(None, 5, 5, 128)	0	batch_normalization_7[0][0]
conv2d_transpose (Conv2DTranspo	(None, 10, 10, 64)	73792	activation_7[0][0]
concatenate (Concatenate)	(None, 10, 10, 128)	0	conv2d_transpose[0][0] activation_5[0][0]
conv2d_8 (Conv2D)	(None, 10, 10, 64)	73792	concatenate[0][0]
batch_normalization_8 (BatchNor	(None, 10, 10, 64)	256	conv2d_8[0][0]
activation_8 (Activation)	(None, 10, 10, 64)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 10, 10, 64)	36928	activation_8[0][0]
batch_normalization_9 (BatchNor	(None, 10, 10, 64)	256	conv2d_9[0][0]
activation_9 (Activation)	(None, 10, 10, 64)	0	batch_normalization_9[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 30, 30, 32)	18464	activation_9[0][0]
concatenate_1 (Concatenate)	(None, 30, 30, 64)	0	conv2d_transpose_1[0][0] activation_3[0][0]
conv2d_10 (Conv2D)	(None, 30, 30, 32)	18464	concatenate_1[0][0]
batch_normalization_10 (BatchNo	(None, 30, 30, 32)	128	conv2d_10[0][0]
activation_10 (Activation)	(None, 30, 30, 32)	0	batch_normalization_10[0][0]
conv2d_11 (Conv2D)	(None, 30, 30, 32)	9248	activation_10[0][0]
batch_normalization_11 (BatchNo	(None, 30, 30, 32)	128	conv2d_11[0][0]
activation_11 (Activation)	(None, 30, 30, 32)	0	batch_normalization_11[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 90, 90, 16)	4624	activation_11[0][0]
concatenate_2 (Concatenate)	(None, 90, 90, 32)	0	conv2d_transpose_2[0][0] activation_1[0][0]
conv2d_12 (Conv2D)	(None, 90, 90, 16)	4624	concatenate_2[0][0]
batch_normalization_12 (BatchNo	(None, 90, 90, 16)	64	conv2d_12[0][0]
activation_12 (Activation)	(None, 90, 90, 16)	0	batch_normalization_12[0][0]
conv2d_13 (Conv2D)	(None, 90, 90, 16)	2320	activation_12[0][0]
batch_normalization_13 (BatchNo	(None, 90, 90, 16)	64	conv2d_13[0][0]
activation_13 (Activation)	(None, 90, 90, 16)	0	batch_normalization_13[0][0]
conv2d_14 (Conv2D)	(None, 90, 90, 7)	119	activation_13[0][0]

Total params: 538,423
Trainable params: 537,015
Non-trainable params: 1,408

Fig. 7. U-net architecture used in this contribution.