ALGORITHMS AND ARCHITECTURES FOR SOME PROBLEMS IN MULTIBEAM ELECTRON BEAM LITHOGRAPHY AND SEM METROLOGY

A Dissertation

by

NARENDRA CHAUDHARY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2019

Major Subject: Electrical Engineering

ABSTRACT

The original Moore's law has slowed down. It has become unfeasible to double the number of transistor per unit area on integrated circuits every 18 to 24 months. However, the continuous need for computation power is driving the semiconductor industry towards innovative solutions to reduce integrated circuit sizes. Multibeam mask writers and accurate scanning electron microscopy (SEM) metrology are two such innovative solutions. Multibeam mask writers enable next-generation integrated circuit fabrication technologies like extreme ultraviolet lithography (EUV). However, the digital communication capacity constraints limit the widespread adoption of multibeam mask writers. In the first part of this dissertation thesis, we present a study of multibeam systems and offer improvements to increase their communication capacity. We propose improvements to the communication datapath architecture, compression algorithms, and the decompression architecture to improve the communication capacity. In the second part of this thesis, we attempt to improve scanning electron microscopy (SEM) metrology using deep learning techniques. Poisson noise, edge effects, and instrument errors frequently corrupt SEM images. Significant improvements in SEM metrology will enable next-generation lithography. To attain metrology improvements, we first create simulated datasets of SEM images and then train multiple deep convolution neural networks on these datasets. Our deep convolution neural networks exhibit superior performance in comparison with previous techniques. Particularly, we demonstrate improvements to nanostructure roughness measurements like line edge roughness (LER), which determine the quality of fabrication processes. Overall, this thesis work attempts to improve the semiconductor manufacturing process using architectural and algorithmic improvements.

DEDICATION

To my parents and my family.

ACKNOWLEDGMENTS

I want to thank my advisor Dr. Serap A. Savari, for her encouragement, guidance, and support throughout my graduate studies. She guided me with patience, contributed to many research ideas, and carefully reviewed all my works, including the write-ups. I am grateful to have an advisor like her.

I want to thank my committee members, Dr. Shankar P. Bhattacharya, Dr. Paul Gratz, and Dr. Anxiao (Andrew) Jiang for their valuable suggestions and comments.

I want to thank Dr. Jeehong Yang for doing the groundwork on the data compression problem and for sharing his code. I want to thank Yao Luo for doing the groundwork on the SEM metrology problem. I also want to thank my colleagues Yao Luo and Sai S. Yeddulapalli, for their help with experimental work. I am grateful to N. Hayashi of Dai Nippon Printing, D. Lie and S. Mukhopadhyay for providing data to conduct my experiments. I also want to thank Dr. Uwe Behringer and Dr. T. R. Groves for their encouragement towards the research. Lastly, I want to thank my friends, colleagues, the department faculty, and staff who made my life at Texas A&M University memorable.

My special thanks go to my family - my mother, my father, and my brother. I am thankful to my family for their unconditional love and support throughout my life. It would not have been possible to reach here without them.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Serap A. Savari, serving as the advisor, Professor Shankar P. Bhattacharyya, and Professor Paul Gratz of the Department of Electrical and Computer Engineering and Professor Anxiao (Andrew) Jiang of the Department of Computer Science and Engineering.

The spectrum analysis code of chapter 3 was written by Sai S. Yeddulapalli. The data analyzed for Chapter 2 was provided by N. Hayashi of Dai Nippon Printing, D. Lie and S. Mukhopadhyay.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

My graduate studies were supported by a research assistantship from Dr. Serap A. Savari and a teaching assistantship from the Department of Electrical and Computer Engineering at Texas A&M University. This work was supported in part by the National Science Foundation grant ECCS-1201994. Also, this work would not have been possible without the support of GenISys, Inc., which was provided through an Agreement of Cooperation between GenISys, Inc. and Texas A&M Engineering Experiment Station at College Station.

NOMENCLATURE

IC	Integrated Circuit
SEM	Scanning Electron Microscopy
LER	Line Edge Roughness
LWR	Line Width Roughness
CNN	Convolutional Neural Network
EUV	Extreme Ultraviolet Lithography
DUV	Deep Ultraviolet Lithography
GPU	Graphical Processing Unit
VLSI	Very Large Scale Integrated Circuit
REBL	Reflective Electron Beam Lithography
PSD	Power Spectral Density
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
MSE	Mean Squared Error
MAE	Mean Absolute Error
ReLU	Rectified Linear Unit
PSNR	Peak Signal to Noise Ratio
AZC	All Zero Column
AZF	All Zero Frame
LSB	Least Significant Bit
R&D	Research and Development
AWGN	Additive White Gaussian Noise

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

Moore's law [10] is the trend of doubling the number of transistors per unit area on integrated circuits (IC). For the last five decades, this trend has given us an exponential increase in computation power. Integrated circuits today have tens of billions of transistors compared to a few hundred transistors at the time of Moore's writing in 1965 [10]. This enormous computing power at low cost has transformed almost every field and has changed the life of humankind. Recently, Moore's law has slowed down as major semiconductor companies struggle to fabricate smaller integrated circuits at the same pace. For instance, Intel Corporation started mass production of its 14 nm node integrated circuits at the end of the year 2014, and its 10 nm node integrated circuits will be in mass production in the second half of the year 2019. This recent node shrink took Intel Corporation almost five years compared to the original 18-24 months rate of Moore's law. This slowdown of Moore's law will have profound effects on all industries as future projections based on the continuous increase of computing power may not materialize. This slowdown is already showing an impact on the semiconductor industry in the increasing popularity of multiple chip designs (or chiplet designs) [11].

We can attribute this slowdown of Moore's law to fabrication technology challenges and cost increases. To understand the fabrication challenges, we take an analogy for integrated circuit design as a modern city design. A city contains layers of structures with different patterns such as underground railway networks, water pipeline networks, electricity line networks, road networks, etc. Similarly, a single integrated circuit design contains multiple layers, and layers have different geometric patterns. For instance, a metal layer might contain the metal wiring patterns which connect transistors on the integrated circuit. Thus in this analogy, the fabrication of ICs onto a silicon wafer compares to making copies of an entire city, layer by layer, on an empty piece of land. In reality, scales are even more extensive than a city. For example, we can compare a typical modern integrated circuit with approximately 510 mm^2 ($= 510$ trillion nm^2) area and 7.5 billion transistors to the earth's surface area of 510 trillion m^2 with living space for approximately 7.5

billion humans. Thus, in some ways, integrated circuit fabrication rivals that to the reconstruction of the entire earth's surface without a single broken highway. Today, the state-of-the-art integrated circuit fabrication process has at least three essential steps, 1) the pattern creation step; i.e., mask set fabrication using electron beam lithography, 2) the pattern replication step with optical lithography; i.e., the fabrication of multiple integrated circuits on silicon wafers by passing light through the masks, and 3) metrology; i.e., the measurement of the quality of the lithographic process using techniques such as scanning electron microscopy (SEM). The semiconductor industry needs innovative solutions for all three steps to enable future nodes and reduce fabrication costs.

The pattern replication step with optical lithography [12, 13] is the heart of integrated circuit fabrication, and significant advances in this technology have enabled Moore's law. Optical lithography transfers geometric patterns to the silicon wafer by passing light through a photomask. The current optical technology is also known as immersion deep ultraviolet lithography (DUV) [13], and it uses light at 193 nm wavelength. This technology has reached its limits. The industry hopes that the new optical technology extreme ultraviolet lithography (EUV) [12] will reduce the cost of the pattern replication step. EUV uses light at 13.5 nm wavelength, and it can theoretically make smaller circuits due to the shorter wavelength. Both 193 nm optical lithography and EUV lithography have the pressing issue of low yields, high photomask set cost, random roughness effects, and stochastic defects. Increasingly, these problems are becoming harder to solve at the smaller technology nodes.

The high cost of the pattern creation step or mask set fabrication is a contributing factor to the slowdown of Moore's law. The high cost is partly due to the increase in the number of mask layers and the decreasing throughput of mask writers. At the leading nodes, electron beam lithography fabricates the majority of photomasks. Electron beam (e-beam) lithography uses electrons instead of light to form the geometric patterns onto the silicon wafers. Electron beam lithography is a standard tool for creating photomasks because it is a pattern creation technology. In contrast, optical lithography is a pattern transfer or a pattern replication technology, and thus, it cannot generate patterns. Due to its pattern generation nature, electron beam lithography faces enormous data processing, data communication, and throughput challenges. The mask fabrication industry has been using the single beam variable-shaped beam (VSB) systems for years, but these systems are not equipped to handle future throughput and resolution requirements. To increase the throughput of electron beam lithography engineers have developed tools which use multiple electron beams (250000 to millions of beams in the future). The semiconductor industry is currently using these multibeam tools [9, 14, 15] for mask fabrication. To handle the future needs of this industry, the datapath architectures of these multibeam systems need to handle petabytes of data and communicate the data at terabits per second. The current datapath architecture of multibeam systems is ill-equipped to handle these requirements. In this thesis work, we propose parallel data compression and a parallel data decompression architecture to improve the communication datapath architecture of multibeam systems. Our approach increases the throughput of multibeam systems to alleviate the technology and cost concerns associated with the pattern creation step.

The third fabrication process step, metrology, needs significant improvements too as roughness, and other stochastic effects in the semiconductor lithographic process are impacting the limits of lithographic scaling [16, 17, 18]. It is necessary to accurately estimate printed nanostructure geometries for further improvements in the lithographic processes. Line edge roughness (LER) and line width roughness (LWR) are two of the standard parameters in the study of random effects, and they are often estimated by means of critical dimension scanning electron microscopy (CD-SEM) images. Low-dose SEM images are interesting because they reduce resist shrinkage and acquisition time, but they are corrupted by Poisson noise, edge effects, Gaussian blur, and other instrument errors. The estimation of edge geometry and roughness requires techniques to account for these artifacts [19, 20]. We believe that in the presence of enough realistic simulated data, supervised machine learning offers the prospect of high-accuracy estimates of the input from the outcome of complex physical processes without the constraints of modeling assumptions. In this thesis, we propose the use of deep supervised learning for the denoising of SEM images and rough edge detection. We concentrate here on Poisson noise and rough line images, but if there is sufficient realistic simulated data, the approach can be extended to the study of other detection and estimation problems such as the estimation of rough contours and the detection of microbridges and missing contact holes. We argue that this approach to SEM denoising preserves the fine details of edges and results in better estimates of LER and other roughness characteristics. The training of deep convolutional neural networks requires large datasets. There are currently no publicly available datasets of rough line SEM images, and we offer such a database to facilitate the future development of algorithms.

Our goal with this thesis work is to improve two of the three integrated circuit fabrication steps using information science algorithms and architectures. We use data compression and architecture based solutions to improve the multibeam mask writing systems, and we use the deep learning method to improve SEM metrology. Metrology is an essential part of the overall integrated circuit fabrication process. Accurate metrology improves both multibeam electron beam lithography and optical lithography by providing the critical process information and feedback. Thus, metrology also offers benefits to the pattern creation step and the pattern replication step. Overall, this thesis work attempts to improve the entire integrated circuit fabrication process flow. This thesis contains four chapters. In the remaining part of Chapter 1, we will explain the basics of various semiconductor fabrication processes. We will review the problem statements for multibeam systems and SEM metrology. In Chapter 2, we will discuss the details of the aperture array-based multibeam system, proposed data compression scheme, and parallel decompression architecture. We will also discuss the experimental results. In Chapter 3, we go into details of the simulated SEM dataset and discuss the proposed deep learning solutions for SEM metrology. We also present experimental results and various visualization techniques to understand the inner workings of our deep neural networks. Finally, we conclude the thesis report in Chapter 4.

1.1 Optical Lithography

The optical lithography process is used to achieve the task of making exact copies of large geometric patterns for integrated circuits. A photomask set contains the geometric patterns of the entire integrated circuit. Each layer of the integrated circuit has at least one photomask associated with it. The optical lithography process transfers these geometric patterns onto silicon wafers one

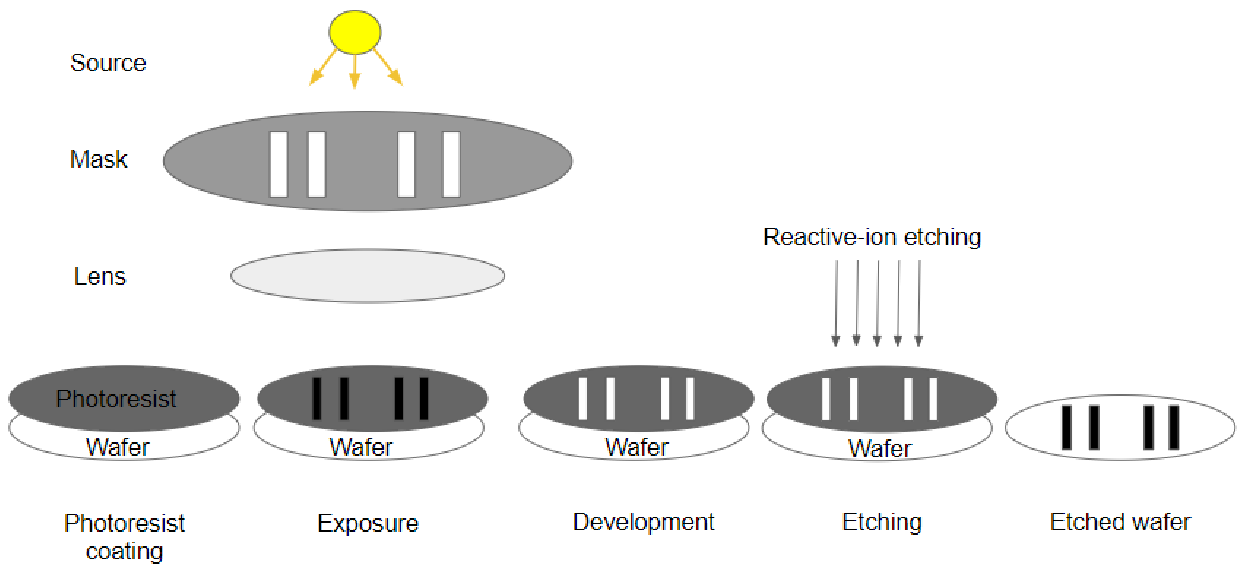


Figure 1.1: Conventional optical lithography.

layer at a time. Figure 1.1 shows the basic steps of optical lithography that are used to transfer geometric patterns of a layer onto the wafer. In the first step, silicon wafers are coated with photosensitive material or photoresist. Photoresist materials are the type of material whose chemical composition changes by exposure to light. In the second step, light passes through the photomask onto the photoresist for the exposure. The third step involves the usage of a photoresist developer to dissolve the exposed photoresist material. This way, the remaining photoresist material captures the geometric patterns of the photomask. After this, the wafer area not covered by photoresist is etched using the reactive ion etching process. Finally, the removal of the photoresist material gives an etched wafer with the geometric patterns of the photomask.

1.1.1 193nm Lithography and Multiple Patterning

Immersion DUV is the current optical lithography technology, and it uses the light at 193nm wavelength (λ). It consistently fabricates patterns below the wavelength of the source light. The resolution limit or minimum half-pitch (hp) of optical lithography depends on the wavelength and numerical aperture (NA) of the projection lens, as shown by the following equation:

$$hp\,(min) \propto \frac{\lambda}{NA}. \tag{1.1}$$

The resolution of 193nm lithography has improved over the years through multiple innovations. Two significant improvements were the use of high numerical aperture (NA) projection lenses and the immersion of the optical system inside high refractive index liquids. Nevertheless, the industry reached the limit of a 38 nm minimum half-pitch around the year 2008. The industry needed to use multiple patterning technology to fabricate structures at even lower half-pitch. Multiple patterning technology fabricates geometric structures of a layer in multiple stages using multiple photomasks. Figure 1.2 shows the desired geometric structures on a layer. Two separate photomasks contain these structures as shown in red and green colors. We can observe that the individual half-pitch remains greater than the minimum possible, but the combined pitch is lower. It is extremely difficult to do multiple patterning beyond quadruple patterning due to overlay and the nanostructure roughness problem, which significantly decrease the yield of the process.

1.1.2 Extreme Ultraviolet Lithography (EUV)

Extreme ultraviolet lithography (EUV) is a next-generation lithography technology. It has been in the development for over two decades. EUV lithography is different from traditional optical lithography in multiple ways. The first difference comes from the use of light at 13.5nm wavelength. Due to this shorter wavelength, it is expected to make smaller nanostructures and continue Moore's law. Secondly, it uses reflection optics instead of the refraction optics used in the traditional approach. EUV needs this reflection optics approach because most materials absorb the extreme ultraviolet wavelength light. EUV lithography faces multiple challenges of source power, pellicle (a mask protection film), nanostructure roughness, and stochastic defects. Stochastic defects are random defects that can occur at the time of exposure due to the dual nature of light. The roughness of nanostructures and stochastic defects can reduce the yield of the process and increase the cost of fabrication.

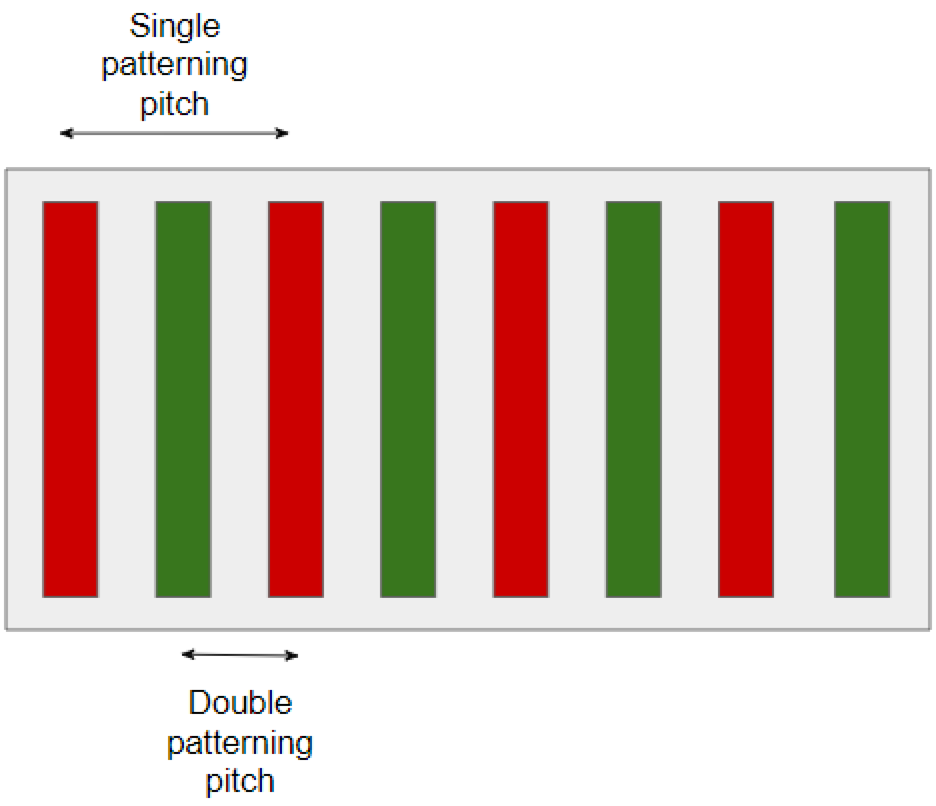


Figure 1.2: Multiple patterning.

1.2 Electron Beam Lithography

The electron beam lithography process is similar to the optical lithography process with some significant differences. First, electron beam lithography uses electrons instead of photons. This means silicon wafers are coated with the electron dose sensitive material instead of a photosensitive material. The more significant difference is that electron beam lithography does not need a photomask. Instead, a single or multiple beams of electrons scan the geometry of the wafer to provide the desired electron dose to form geometric patterns. This makes electron beam lithography a pattern creation technology. In contrast, optical lithography is a pattern transfer or a pattern replication technology. Optical lithography can only transfer or replicate the patterns already present on a photomask. In simple terms, optical lithography is somewhat similar to the old method of creating multiple images from a photo camera negative while electron beam lithography is similar to writing out each pixel on a digital screen to form a digital image. This is the reason electron beam lithography is a standard tool for creating photomasks. Electron beam lithography can also be used to fabricate circuits directly. Direct write systems are popular in R&D due to lax time constraints and high resolution. Electron beam mask writing tools come mainly in two forms, i.e., variable shaped beam systems and multibeam systems.

1.2.1 Variable Shaped Beam System

Variable shaped beam (VSB) mask writers use variable-sized rectangular and triangular electron beams to write patterns onto a substrate. The photomask file usually stores the geometric pattern data into polygon format, and these polygons are fractured into rectangular and triangular shots as shown in Figure 1.3. The VSB writer scans the beam only at the places where pattern features are to be written. This type of scanning is also known as vector scanning. The mask fabrication industry has extensively used these single beam variable-shaped beam (VSB) systems for years, but they are not equipped to handle future throughput and resolution requirements. This is because the VSB system's writing time depends on the complexity of mask shapes [21, 22] and on the number of shots, which are increasing at the newer nodes.

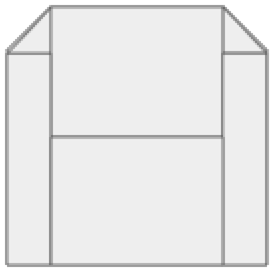


Figure 1.3: Rectangular and triangular VSB shots.

1.2.2 Multibeam Systems and the Datapath Problem

Multibeam systems use multiple electron beams to increase the throughput of electron beam lithography. There have been various attempts to make multibeam systems. The reflective electron beam lithography (REBL) [23, 24] system and the MAPPER [25] system were proposed for the direct fabrication of integrated circuits. Recently, the aperture array-based multibeam mask writers introduced by IMS Nanofabrication [9] and NuFlare Technology [14, 15] have been proposed to improve mask write times. These multibeam systems use a large number of smaller beams to write patterns for the mask layers. Multibeam systems use grayscale pixel-based writing where each beam provides a dose according to digital data. This dose is transferred to a single pixel or number of pixels [2, 26]. The multibeam systems face a throughput bottleneck described by Tennant's law [27] which can be described as follows: define areal throughput as the area of a wafer that can be printed per unit time using direct-write-like lithography technologies. Then

$$Areal\ throughput \propto resolution^5. \tag{1.2}$$

Besides the pixel throughput problem, multibeam systems also have to process large amounts of data and communicate grayscale data at high data rates. A recent paper [15] by NuFlare discusses the difficulties associated with data processing and communication. These data processing and communication requirements will continue increasing in the future in proportion to the square of the resolution. The current datapath architecture of multibeam systems is ill-equipped to handle these requirements. IMS and NuFlare have had some success in addressing throughput requirements by parallel processing [14, 15] using graphical processing units (GPUs) and by increasing the communication capacity [9, 15] of the datapath. However, as we explain in Chapter 2, these approaches have limitations. Our objective here is to improve the communication and computational efficiency of the "aperture array-based" multibeam mask writers. Improving data communication increases the throughput of the multibeam systems, which reduces the cost of integrated circuit fabrication and facilitates the future adoption of direct-write multibeam lithography systems.

1.3 Scanning Electron Microscopy and the Metrology Problem

The scanning electron microscope is a widely used electron microscope for critical dimension metrology. In this microscope, a focused beam of electrons scans the surface of the sample in a raster scan pattern to create an image of the sample at sub-micron resolutions. These electrons interact with the sample and produce signals which contain information about the sample surface geometry. The quality of images depends on the energy of the electron beam or the dose provided. Low-dose SEM images are potentially attractive because of relatively short acquisition times and resist shrinkage. However, to determine the edge geometry from such images requires techniques to account for Poisson noise, Gaussian blur, edge effects, and other instrument errors [19, 20]. One could use algorithms based on filtering [28] or physical model-based regression [29, 30]. However, the filtering based methods often require careful selection of the filter parameters to prevent changes in the edge geometry while the modeling assumptions constrain model-based regression. The estimation of edge geometries and roughness of nanostructures is critical to determine the yield of the IC fabrication process. Therefore, the algorithmic problems associated with the extraction of edge geometries in a high-volume manufacturing setting continue to be investigated.

2. MULTIBEAM SYSTEM, DATAPATH IMPROVEMENT AND DATA COMPRESSION SCHEME [*]

The aperture array-based multibeam systems designed by NuFlare Technology [14, 31, 15] and IMS Nanofabrication [9, 32, 33, 34] use a 512×512 array of square beams arranged as a matrix of 512 rows and 512 columns. The square beams are created by passing a broad beam through an aperture plate system which consists of an aperture plate and a deflection plate. Figure 2.1 illustrates a smaller 4×4 array with an aperture plate to convert a broad electron beam into multiple small beams and a deflection plate which controls the direction of the beams. These smaller beams are projected on the silicon wafer to form the desired mask pattern.

2.1 Multibeam Blanker System

The blanker is a significant component of both the NuFlare Technology [14, 31, 15, 1] and the IMS Nanofabrication [9, 32, 33, 34] multibeam mask writing systems. The multibeam blanker system we consider is motivated by the 2016 NuFlare system [31, 15]. We focus on the NuFlare system in this thesis report because NuFlare disclosed more information than IMS about their circuit design and data communication system. We also discuss certain aspects of the IMS system in our publications [2]. We follow NuFlare's choice of 10-bit representations of the dose of a pixel as opposed to IMS' 4-bit representation with beam overlaps and point out that it is not straightforward to compare these representations since IMS uses 5 nm $\times$ 5 nm pixels and NuFlare uses 10 nm $\times$ 10 nm pixels. Furthermore, since NuFlare Technology has not disclosed all details about its multibeam mask writing systems we make various assumptions to explain our understanding of the multibeam blanker system; for example, we created Figures 2.2-2.5. The multibeam system consists of two plates; namely an aperture plate and a deflection plate [9, 32, 33, 34, 14, 31]. The aperture plate consists of an array of apertures which converts a broad beam into multiple beams.

[*] Part of the data reported in this chapter is reprinted with permission from N. Chaudhary and S. A. Savari, "Parallel compression/decompression-based datapath architecture for multibeam mask writers," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 16, no. 4, 043503, 2017. DOI: https://doi.org/10.1117/1.JMM.16.4.043503. Copyright [2017] by SPIE.

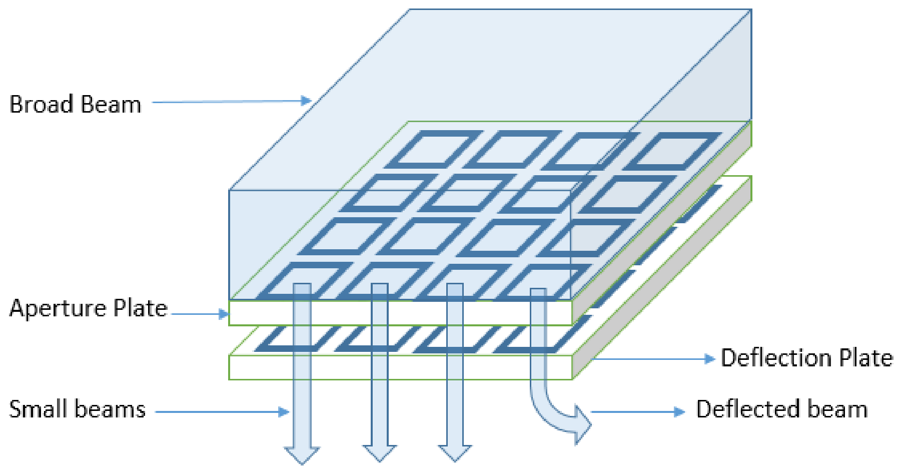


Figure 2.1: Aperture array-based multibeam system.

The deflection plate consists of an equal number of apertures at the same coordinates as the aperture plate. The deflection plate also contains a pair of electrodes at each aperture providing enough deflection voltage to deflect the beam passing through the aperture [31]. A grayscale dose control for each beam is provided by controlling the deflection voltage for a discrete amount of time by means of a control circuit for each beam [31]. The deflection control circuit of each beam is connected by a bus to the other control circuits to enable the communication of data within a row [31]. Each row's data is provided by a "side pad" digital circuit which could be a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC). The deflection plate is a stand-alone device or a chip packaged separately from the side pad. Figure 2.2 shows the deflection plate design for an array with sixteen beam apertures with four rows and four columns.

Figure 2.2 illustrates a control circuit associated with each individual beam. The control circuitry of each beam consists of two parts, namely a digital beam logic for the communication of 10-bit dose data with the ability to generate 1024 discrete time intervals and an analog circuit (mainly a differential amplifier) to drive the electrodes for the amount of time specified by the digital beam logic [31]. We do not consider the analog design of the amplifier circuit in this thesis report and focus instead on the digital logic needed for each beam. Figures 2.3 and 2.4 show one

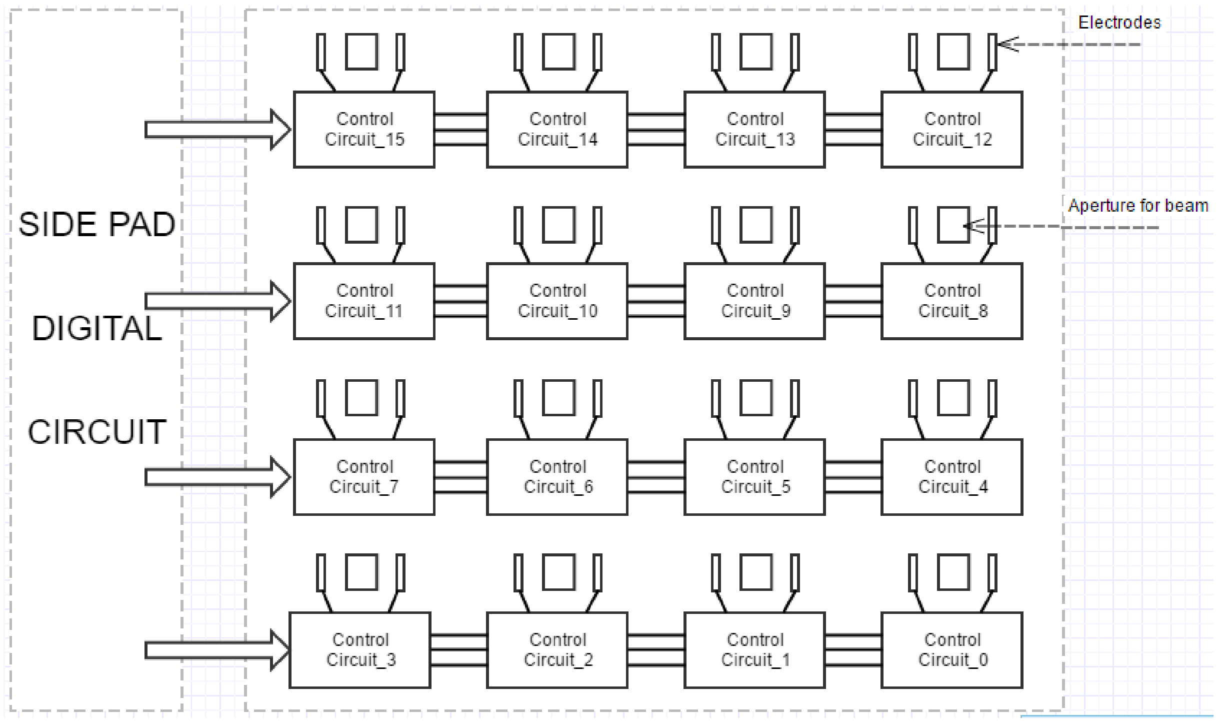


Figure 2.2: Deflection plate of a 4×4 beam array with side pad circuitry. Each beam has a control circuit connected to electrodes. The electrodes deflect the beams. Reprinted with permission from [1].

implementation of the beam logic. The digital beam logic consists of a shift register with a one-bit serial input to communicate 10-bit data and a 10-bit countdown and “stop at 0” counter to provide 1024 time intervals. Registers [35] are digital logic circuits which store data in the form of binary numbers. Shift registers [35] are the registers that load the input bit into the most significant bit (MSB) or least significant bit (LSB) of the register value and “shift” the stored binary number by one position. The counter starts from the 10-bit input dose value as the initial value and the value decreases by 1 at every clock cycle until the counter value reaches zero at which point the counter stops.

The beam logics require the supply voltage (VDD) and the ground (GND) as inputs. Besides the supply input (VDD, GND), a beam logic requires a clock signal input, a reset signal input, an unload signal input to load the counter with the shift register value as the initial value, and an

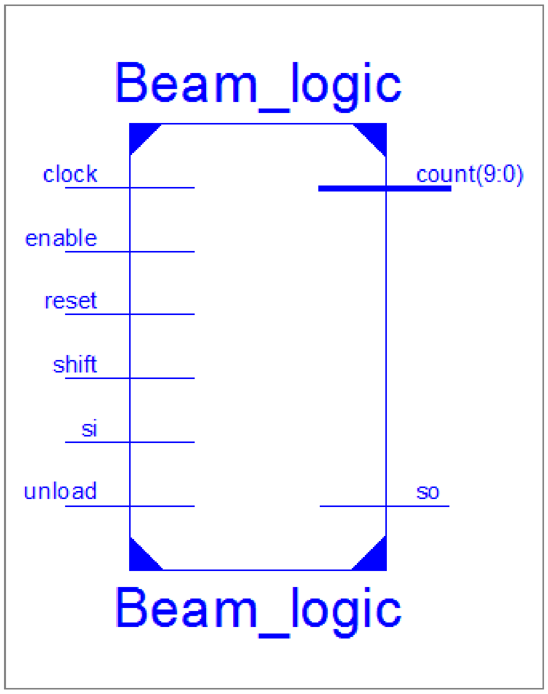


Figure 2.3: Digital beam logic associated with the control circuit of each beam. Reprinted with permission from [1].

enable (trigger) input signal to start the counter. These input signals should be synchronous and broadcast to all the control circuits of the beams, as simultaneous control of all the beams requires that all the counters start at the same clock cycle. Here a synchronous signal means the input signal (reset, unload, enable) values are sampled only at the positive edge of the clock signal. Let *si* and *so* respectively denote the serial input and the serial output to the shift register; they can be alternately implemented with a 10-bit parallel input and a 10-bit parallel output [31, 36]. An extra digital input shift signal is also provided to control the shift register, i.e., the data is moved in/out of the shift register only when the shift signal is high (1). The shift signal is common to all the control circuits in a row of the array.

As shown in Figure 2.2 the beams are organized in the shape of a grid with rows and columns. We connect the control circuits of a row of beams in a beam array for the communication of the 10-bit dose data. The serial/parallel output of one control circuit is connected to the serial/parallel

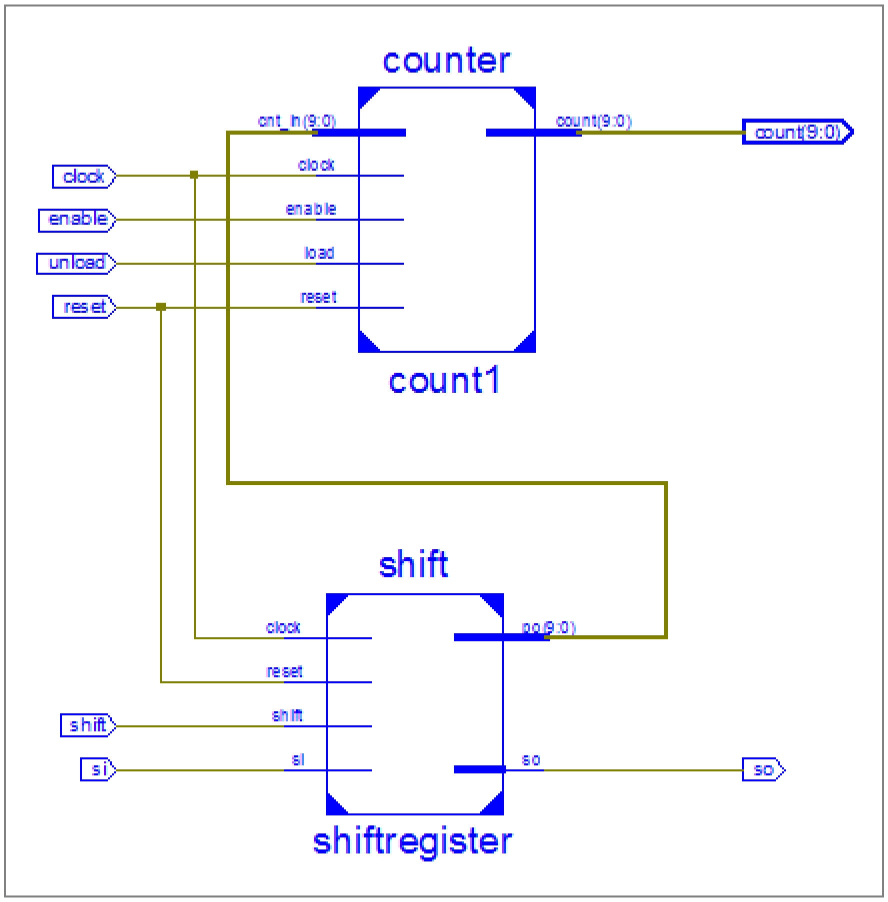


Figure 2.4: Expanded view of the digital beam logic associated with the control circuit of each beam. Reprinted with permission from [1].

input of the next control circuit [31, 36]. This allows the data to be shifted within a chain of beam logics. Figure 2.5 illustrates the connection of four beam logics in a row. This type of communication system removes the need for addressing each 10-bit dose value to a control logic. Let N be the number of beams in a row. When no compression is used $N * 10$ clock cycles are required to transfer the 10-bit data to each beam logic in a row. The clock, shift, unload, reset, enable are all common input signals to each beam logic block in a row. Each beam logic also has a separate 10-bit output count which is the counter value and serves as the timer for the beam deflection.

The beam array consists of multiple rows of beams in parallel [31, 36]. The clock, enable, unload and reset signals are common to all the rows in the beam array. Here, all the beam logic blocks receive the same set of signals to start all the counters at a given time instance [31]. The shift signals and the serial inputs for the data are different for each row of beams. Separate serial inputs and shift signals for each row of beams are required for a data transfer protocol without addressing.

2.2 Beam Arrays, Scanning Strategy and Compression Constraints

The IMS multibeam system uses a beam size of 20 nm or 10 nm with 4-bit dose control for each beam. It also has 5 nm $\times$ 5 nm pixels on the pixel grid at which beams write. In the IMS system the beams overlap to provide 241 or 61 gray levels for the 5 nm$\times$$5$ nm pixels depending on the beam size. The NuFlare multibeam system has a 10 nm beam size and 10 nm $\times$ 10 nm pixels on the pixel grid. Each beam has 10-bit dose control. In contrast with the IMS system, the beams do not overlap on the pixel grid to create the higher dose levels. Therefore each 10 nm $\times$ 10 nm pixel has 1024 gray levels assuming single-pass writing. For both systems, the distance between the individual beams in the X- and Y-directions is 160 nm, i.e., 16 pixels in the NuFlare multibeam system and 32 pixels in the IMS multibeam system.

To investigate the effect of parallelism we study arrays of varying dimensions. The beam arrays we consider have dimension $2^N \times (2^N - 1)$ beams; i.e., they form a matrix with 2^N rows and $2^N - 1$ columns, where N is an odd integer. The center to center distance between neighboring beams in

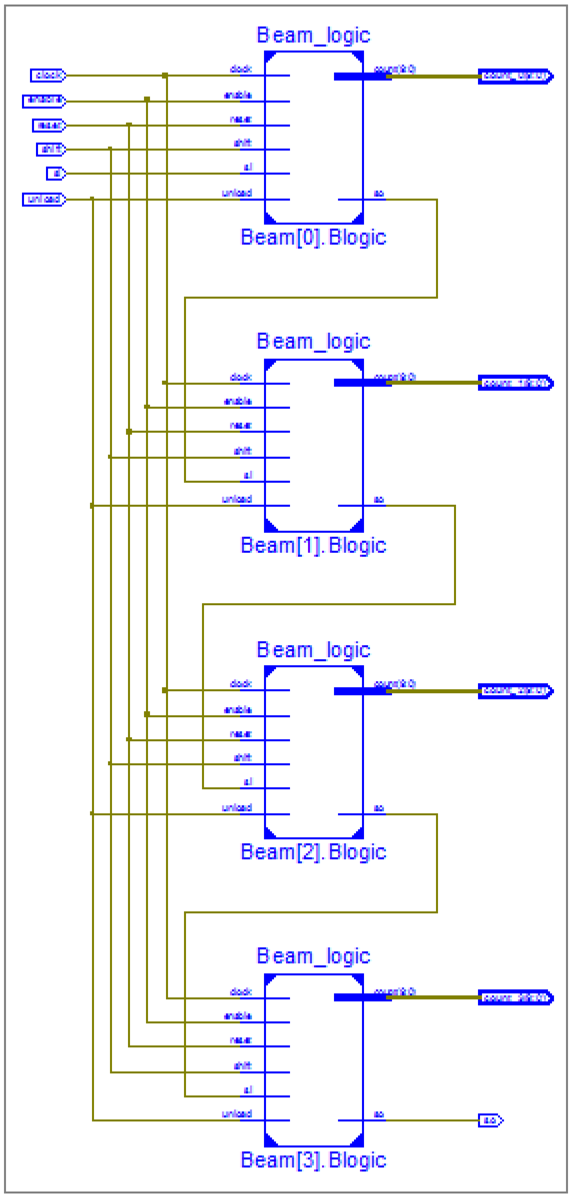


Figure 2.5: Row of beam logics connected in a chain by the *si* inputs and the *so* outputs. Reprinted with permission from [1].

a row or a column is $2^{0.5(N+1)}$ pixels for the IMS multibeam system and $2^{0.5(N-1)}$ for the NuFlare multibeam system; observe that this increases as N grows. Furthermore, the case where $N = 9$ corresponds to an array with 512×511 beams and a center to center distance of 32 pixels in the IMS multibeam system and 16 pixels in the NuFlare multibeam system. This case is closest to the real multibeam arrays of size 512×512.

The multibeam array system deflects the entire beam array as a unit. In the IMS case, while the distance between neighboring beams is $2^{0.5(N+1)}$ pixels the system as a unit must write every pixel. A raster scan might be a first guess for a scanning strategy, but this approach would either lead to pixels written by more than four beams in the 10 nm case or sixteen beams in the 20 nm case or to large shifts of the beam array. Therefore, we instead propose a zigzag scanning strategy described in Table 2.1 and depicted (for the first three steps) in Figure 2.6 in the case of an 8×7 beam array with 10 nm beams. This approach ensures that pixels are written four times for the case of a 10 nm beam and sixteen times for the case of 20 nm beams. Most of the time the beam array will move a distance of $0.5d$ pixels in the X-direction and +1 or -1 pixel in the Y-direction. This can be implemented by simultaneously controlling the stage and beam array so that the stage moves in the X-direction and the beam array is deflected in the Y-direction. Observe that the distribution of sequences of 4-bit dose values to each beam will depend on the scanning strategy.

Since the NuFlare system has no beam overlaps, its horizontal and vertical movement step sizes will differ from the zigzag scanning strategy of the IMS system. In Table 2.2 we propose a zigzag strategy for our family of multibeam arrays inspired by the NuFlare multibeam system.

We believe the zigzag scanning strategy requires less deflection of the multibeam array and covers the entire scanning area in fewer steps. A reduction in the deflection of the beam array could be desirable to reduce the settling time and increase the frequency of the beam array deflection. Furthermore, the use of a scanning strategy with predefined steps would help in reducing the beam array deflection data because it could be calculated in real time similar to raster scanning. It differs from those for the vector scanning strategies used in variable-shaped beam systems, where the coordinates of positions must be specified.

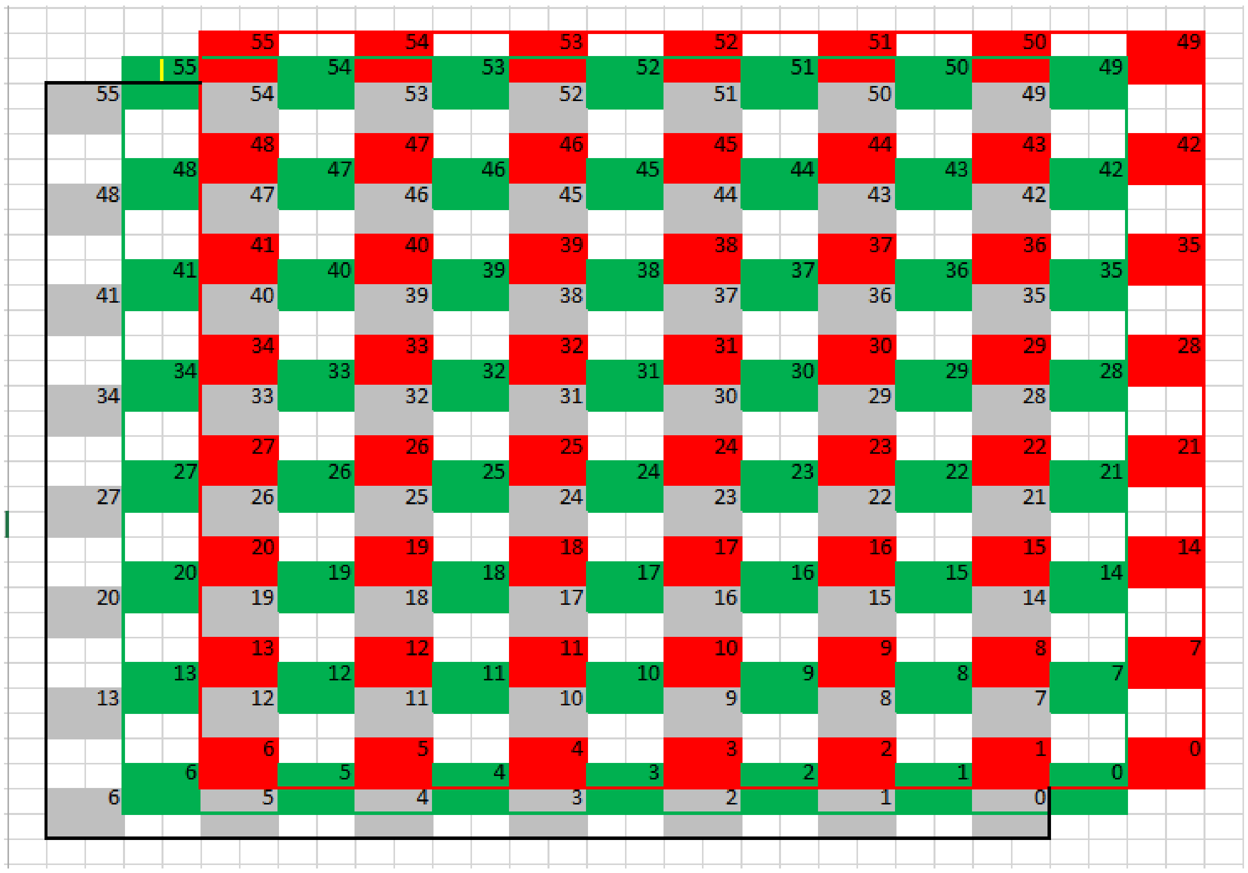


Figure 2.6: The scanning strategy for an 8 × 7 beam array on a 5 nm pixel grid. The numbers represent the positions of the 56 beams. Gray, green and red respectively represent steps 1, 2 and 3 of the beam array movement. Reprinted with permission from [2].

Table 2.1: Overview of scanning strategy for the IMS-inspired family of beam arrays. Reprinted with permission from [2].

Array	Number of beams	Distance in pixels between the centers of two neighboring beams in the same row or column	Horizontal movement of array from one writing to the next within a "stripe"	Vertical movement of array from one writing to the next within a "stripe"
$2^N \times (2^N - 1)$, N odd	$2^N(2^N - 1)$	$d = \sqrt{2^{N+1}}$	$\frac{d}{2}$ or $\frac{d}{2} - 1$ pixels depending on the iteration	+1 or -1 pixels depending on position of array within zigzag
2 × 1	2	2	0 every 2nd iteration, +1 pixel for all other iterations	Progresses in the sequence $n,\ n+1,\ n+2,$ $n+1,\ n,\ n+1, \ldots$ for some n
8 × 7	56	4	+1 every 4th iteration, +2 pixels for all other iterations	Progresses in the sequence $n,\ n+1, \ldots,\ n+4,$ $n+3, \ldots,\ n+1,\ n$ $n+1,\ n+2,\ \ldots$ for some n
32 × 31	992	8	+3 every 8th iteration, +4 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ \ldots,\ n+8,$ $n+7,\ \ldots,\ n+1,\ n,$ $n+1,\ n+2,\ \ldots$ for some n
128 × 127	16256	16	+7 every 16th iteration, +8 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ \ldots,\ n+16,$ $n+15,\ \ldots,\ n+1,\ n,$ $n+1,\ n+2,\ \ldots$ for some n
512 × 511	261632	32	+15 every 32th iteration, +16 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ \ldots,\ n+32,$ $n+31,\ \ldots,\ n+1,\ n,$ $n+1,\ n+2,\ \ldots$ for some n

Table 2.2: Overview of scanning strategy for the NuFlare-inspired family of beam arrays. Reprinted with permission from [1].

Array	Number of beams	Distance in pixels between the centers of two neighboring beams in the same row or column	Horizontal movement of array from one writing to the next within a "stripe"	Vertical movement of array from one writing to the next within a "stripe"
$2^N \times (2^N - 1)$ N odd	$2^N(2^N - 1)$	$d = \sqrt{2^{N-1}}$	$2d$ or $(2d - 1)$ pixels depending on the iteration	+1 or -1 pixels depending on position of array within zigzag
8×7	56	2	+3 every 2nd iteration, +4 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ n+2,$ $n+1,\ n,\ n+1,\ \ldots$ for some n
32×31	992	4	+7 every 4th iteration, +8 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ \ldots,\ n+4,$ $n+3,\ \ldots,\ n+1,\ n,$ $n+1,\ n+2,\ \ldots$ for some n
128×127	16256	8	+15 every 8th iteration, +16 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ \ldots,\ n+8,$ $n+7,\ \ldots,\ n+1,\ n,$ $n+1,\ n+2,\ \ldots$ for some n
512×511	261632	16	+31 every 16th iteration, +32 pixels for all other iterations	Progresses in the sequence $n,\ n+1,\ \ldots,\ n+16,$ $n+15,\ \ldots,\ n+1,\ n,$ $n+1,\ n+2,\ \ldots$ for some n

For any choice of scanning strategy, after writing one set of beam shots the entire beam array moves to a new position. This imposes a synchronization constraint on the beam array as the data for all the beams should be available before the beam array moves to a new position. If we have a 512×512 beam array then the entire beam array would require 2621440 bits at one position as each beam needs 10-bit data and there are 262144 beams. The next set of 2621440 bits of 262144 beams will be needed when the beam array moves to a new position. We can consider the data of each set of 262144 beams as a frame of a grayscale "video" with dimension 512×512. Any compression scheme and decompression architecture should operate with these parameters. The decompression scheme should be able to decode the frames in their given sequence. The data inside the frames may be compressed in a variety of ways as long as the synchronization constraint is satisfied. The "video" analogy and constraints also allows the separate and parallel compression of each frame as long as the sequence of compressed frames doesn't change. Thousands of frames can be potentially compressed in parallel to reduce compression time. However, there are some differences between this communication problem and traditional "video" communication; the latter application can take advantage of inter/intra-frame dependencies. For our problem there are weaker intra-frame dependencies in the data because of the large distance between the beams in a beam array; for example in a 512×511 beam array the distance between successive beams in either the X- or Y-direction is 16 pixels. The inter-frame dependencies are mainly governed by the choice of scanning strategy. Our scanning strategy is described in Table 2.2. Observe that the distance a beam moves from one frame to the next in the X-direction is large, so there does not appear to be any appreciable inter-frame dependencies for our scanning strategy. However, a scanning strategy which more closely resembles raster scanning might provide inter-frame dependencies.

Simple design rules [37] require a large fraction of the pixels in a typical mask layer to have pixel value of zero; i.e., the data is sparse. Figure 2.7 illustrates a pattern without correction in which 50% of the pixel values are zeros. Complementary lithography and multiple patterning further contribute to the prevalence of sparse data, and our data compression scheme is designed to take advantage of the existing sparsity within mask data.

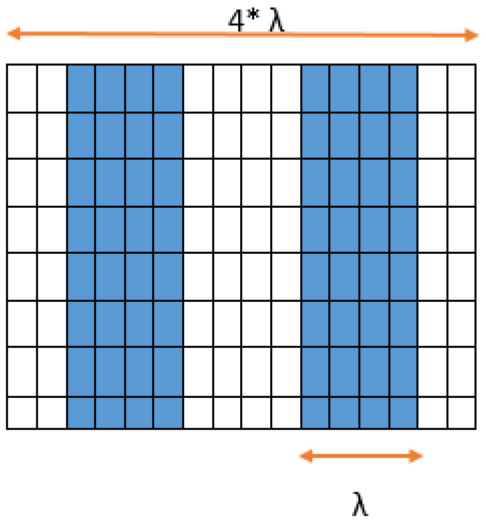


Figure 2.7: 50% sparsity with simple design rules. λ is equal to the half pitch. Reprinted with permission from [1].

2.3 The Datapath Problem and Data Compression

The largest beam array we study is a 512 $\times$ 511 beam array, and it has 261,632 beams. Since each beam requires a 10-bit shot dose data a total of 2,616,320 bits are used at each step of the beam array. If the beam array can be deflected at 1 MHz, i.e., if each step can be completed in 1 µs, then the beam array processes approximately 2.38 terabits of shot data per second. Data processing and uncompressed data transmission at this speed would not be an effective communication strategy.

The current datapath architecture of these multibeam systems shown in Figure 2.8(a) is ill equipped to handle throughput requirements. In the current datapath architecture the GDSII or OASIS formated data is converted into a tool-specific format and transferred to the mask writing tool. Data processing is next performed online to do rasterization, proximity effect corrections and other corrections which result in pixel data and beam deflection data. The data processing capability and datapath communication capacity affect the efficiency of this kind of datapath. IMS and NuFlare have had some success in addressing throughput requirements by parallel processing [14, 15] using graphical processing units (GPUs) and by increasing the communication capacity [9, 15] of the datapath. However, these approaches have limitations. In computer architecture Amdahl's law [38] states that the speedup or throughput gain for the execution of a task with increased resources is always limited by the fraction of the task that cannot benefit from the improvement. In the case of parallel processing the throughput gain is limited by the part which cannot be parallelized. Let $Speedup$ denote the throughput gain, $Fraction_{enhanced}$ be the proportion of the task benefiting from improvements and $Speedup_{enhanced}$ be the speedup in $Fraction_{enhanced}$ by parallelism or other improvements. Then

$$Speedup = \frac{1}{(1 - Fraction_{enhanced}) + \dfrac{Fraction_{enhanced}}{Speedup_{enhanced}}}. \tag{2.1}$$

Table 2.3: The effect of an increase in datapath resources in recent IMS mask writing tools [9]. Reprinted with permission from [1].

Tools	Datapath	Mask write time 100 mm × 130 mm
IMS MBMW-101 Beta	12 G	15 hours/mask
IMS MBMW-101 HVM	120 G	10 hours/mask

Observe from Table 2.3 that an increase in datapath speed by a factor of 10 improves the mask writing time only by a factor of 1.5. Part of the throughput bottleneck can be attributed to the time needed to write the pixel data. Our objective is to improve the communication and computational efficiency of the "aperture array-based" multibeam mask writers. In this work we introduce a datapath architecture (see Figure 2.8(b)) where the parallel decompression portion is inspired by the very large scale integrated circuit (VLSI) testing literature [39]; this architecture addresses the synchronization requirement among the decoders and shows how to reduce the number of clock cycles needed to transfer typical mask data. We also recommend that the data be compressed by

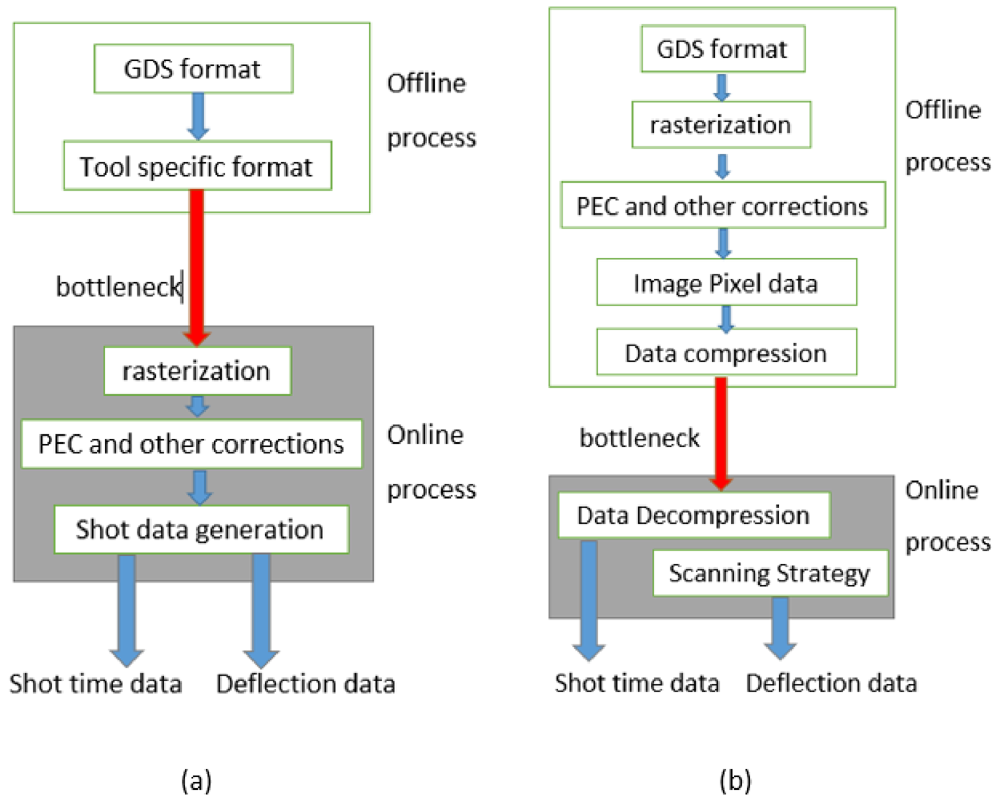


Figure 2.8: (a) Current datapath architecture. (b) Proposed datapath architecture. Reprinted with permission from [1].

custom-designed parallel compression algorithms to decrease the processing time, memory and file sizes. In addition to data decompression our proposed online processing includes the generation of deflection data from a predefined scanning strategy (see Figure 2.8(b)).

2.4 Previous Work

For direct-write electron beam lithography systems there have been multiple proposals of datapaths in which a major component of the data processing and the generation of pixel-based data is performed offline. Dai and Zakhor [40, 41] initiated the study of the communication and computation efficiency within datapaths by considering the hardware constraints of some architectures, and they proposed the Block C4 layout image compression algorithm. Yang and Savari [42, 43] proposed the Corner2 algorithm and custom run-length encoding schemes to reduce the hardware complexity and improve the decoding speeds, and Chaudhary, Luo, Savari and McCay [44] modified this algorithm for proximity-corrected grayscale data. Carroll *et al.* [24] implemented a variation of a Lempel-Ziv [45, 46] data compression algorithm in the Reflective Electron Beam Lithography (REBL) system [23]. We began to consider the impact of parallelism on aperture array-based multibeam mask writers in References [2] and [26]. Our inspiration in those papers was the IMS mask writer. While those papers proposed the use of multiple simple run-length decoders they did not examine how to coordinate the operation of the decoders to meet existing device constraints. Furthermore, the data transfer to and the data fan-out at the control circuitry of the blanking aperture array use metal transmission lines operating at low clock frequencies [15], and earlier papers did not consider the number of clock cycles in the communication of data.

2.5 A Compression Scheme and Decompression Architecture

To solve the datapath problem, we first discuss a simple run-length code for data compression and explain the decompression architecture for it. We will later propose a more effective scheme. As we mentioned earlier in Section 2.2, the majority of 10-bit symbols are zeros. In our problem we choose to represent a 10-bit zero symbol by a single-bit "1" and to encode the 10-bit nonzero symbols by the 11-bit string which concatenates the prefix "0" to the original 10-bit string in order to reduce data volumes and overall clock cycles. For example, two encodings of 10-bit symbols are illustrated below.

$$1101011010 - - - - - - - - - - - - > \underline{0}1101011010$$

$$0000000000 - - - - - - - - - - - - > \underline{1}$$

Observe that the maximum possible compression ratio of this simple scheme is 10, which occurs when all symbols are zero. The worst possible compression ratio is $10/11 = 0.91$, which occurs when all symbols are nonzero.

The data for multiple beams is arranged in a column by column fashion. If we take Figure 2.2 as a reference in the numbering convention of a beam array, then the data is arranged in a "frame" in the following sequence. Let U_i be the symbol or beam dose associated with beam number i. For a beam array with sixteen beams organized in four rows and four columns U_0 is the first symbol in a "frame" and U_{15} is the last symbol. After this data is written and the beam array moves a new "frame" starts. A similar strategy is applied to any size of beam array. The sequence of uncompressed symbols is given below.

$$U_0, U_4, U_8, U_{12}, U_1, U_5, U_9, U_{13}, U_2, U_6, U_{10}, U_{14}, U_3, U_7, U_{11}, U_{15}$$

To compare different communication schemes we compare the number of clock cycles they use on the hardware. We first look at the number of clock cycles for uncompressed data transmission. The serial communication of n 10-bit symbols $U_0, \cdots, U_{n-1}$ to n beams with a single bit wire will use $n * 10$ clock cycles. One could alternatively use $n * 10$ bit parallel communication wires to communicate data to all the beams, which should ideally take 1 clock cycle. The parallel communication will not always be helpful as $n * 10$ bits must be collected from a capacity constrained channel and stored before they are transmitted to the n beams. The process of collection and storage of data from a serial source may offset the advantage of parallel communication, and there will be a significant and undesirable increase in the hardware on the deflection plate. There can be many degrees of parallelism between these two extremes but for uncompressed data transfer all of them have implementation trade-offs.

We know from the previous discussion that compressed symbol C_i corresponding to an uncompressed symbol U_i will consist of either a single bit string or an eleven-bit string.

$$C_i = \begin{cases} 1, & \text{if } U_i = 0000000000 \\ \mathbf{0}U_i & \text{if } U_i \neq 0000000000 \end{cases} \tag{2.2}$$

The compressed string is communicated to the side pad digital circuit and after that it needs to be decompressed and transferred to the individual beams. An effective communication strategy should not only compress the data but should also reduce clock cycles by means of data decompression.

The compressed string $C_0, \cdots, C_{n-1}$ can be decoded in multiple ways. We first look at the case when a single decoder is used to decompress the data as shown in Figure 9(a). The function of the decoder is to collect the C_i symbols and generate the U_i symbols. When the input C_i symbol is eleven bits long the transmission through a one-bit wire to the decoder will take eleven clock cycles and the U_i symbol can be generated in the last ten of those eleven clock cycles. When the input C_i symbol is one bit long the transmission takes only one cycle and generation of the U_i takes the next ten cycles. Hence a single decoder always takes eleven clock cycles to generate each U_i from any C_i symbol. This shows that a single decoder decreases throughput by taking eleven clock cycles instead of the ten needed in the uncompressed case.

In the second case we have two decoders as shown in Figure 2.9(b). We pass every other C_i to each decoder. Since the C_i symbols are of variable lengths, we need some additional hardware to detect the symbol length. The detector in Figure 2.9(b) detects the length of C_i by checking the prefix and directs the entire symbol to the corresponding decoder. Consider the special case of two symbols $C_1 = 1$ and $C_2 = 1$; they can be decompressed in parallel as shown in Figure 2.9(c). The parallel decompression would only take thirteen cycle, i.e., two cycles for the communication of the compressed symbols to the decoders and eleven cycles for parallel decompression. Two decoders provide the opportunity for the parallel decompression of the $C_i = 1$ symbols. Any other combination of symbols will not reduce the number of clock cycles. We also assume that the number of decoders is known in advance.

We can extend and generalize this approach to multiple decoders. Multiple decoders provide increased opportunity for parallel decompression of the $C_i = 1$ symbols. This approach should increase the parallelism and communication efficiency for the uncompressed sparse data at the cost of increased hardware complexity. We can use the set of k decoders to decompress and transfer data to k nodes as shown in Figure 2.9(d). The compressed symbols C_i of length one bit can again be decompressed in parallel, while the C_i of length eleven bit are decompressed serially. The C_i of eleven bit length take eleven cycles and the C_i of one bit take one cycle plus 10 cycles every

Table 2.4: Simple compression scheme and clock cycles for decompression. Reprinted with permission from [1].

Symbol	Explanation	Prefix	Suffix	Cycles to Decode	Cycles for uncompressed data transfer
Z	0000000000	1	-	$1+$ (10 every column)	10
NZ	10-bit value	0	10-bit value	11	10

Table 2.5: Extended Compression Scheme for M(rows) $\times$ N(columns) array. Reprinted with permission from [1].

Symbol	Explanation	Prefix	Suffix	Cycles to Decode	Cycles for uncompressed data transfer
AZF	all zero frame	001	-	3	$M*N*10$
AZC	all zero column	000	-	$M+10+3$	$M*10$
Z	0000000000	1	-	$1+$ (10 every column)	10
NZ	10-bit value	01	10-bit value	12	10

column because of parallel decompression. The simple compression scheme is summarized in Table 2.4.

Figure 2.10 shows the logical architecture of the parallel decompression combined with the existing deflection plate architecture. We assume the deflection plate in Figure 2.10 consists of four columns and k rows. The set of k decoders are connected to the head column of the beam logic and they transmit the $U_1, \cdots, U_k$ symbols to the deflection plate. The decompressed symbols are shifted further to their desired beam logic by the chain of shift registers.

If the sole goal of the data compression scheme was the effective communication of data from the storage device to the side pad then a more complex compression scheme would better serve

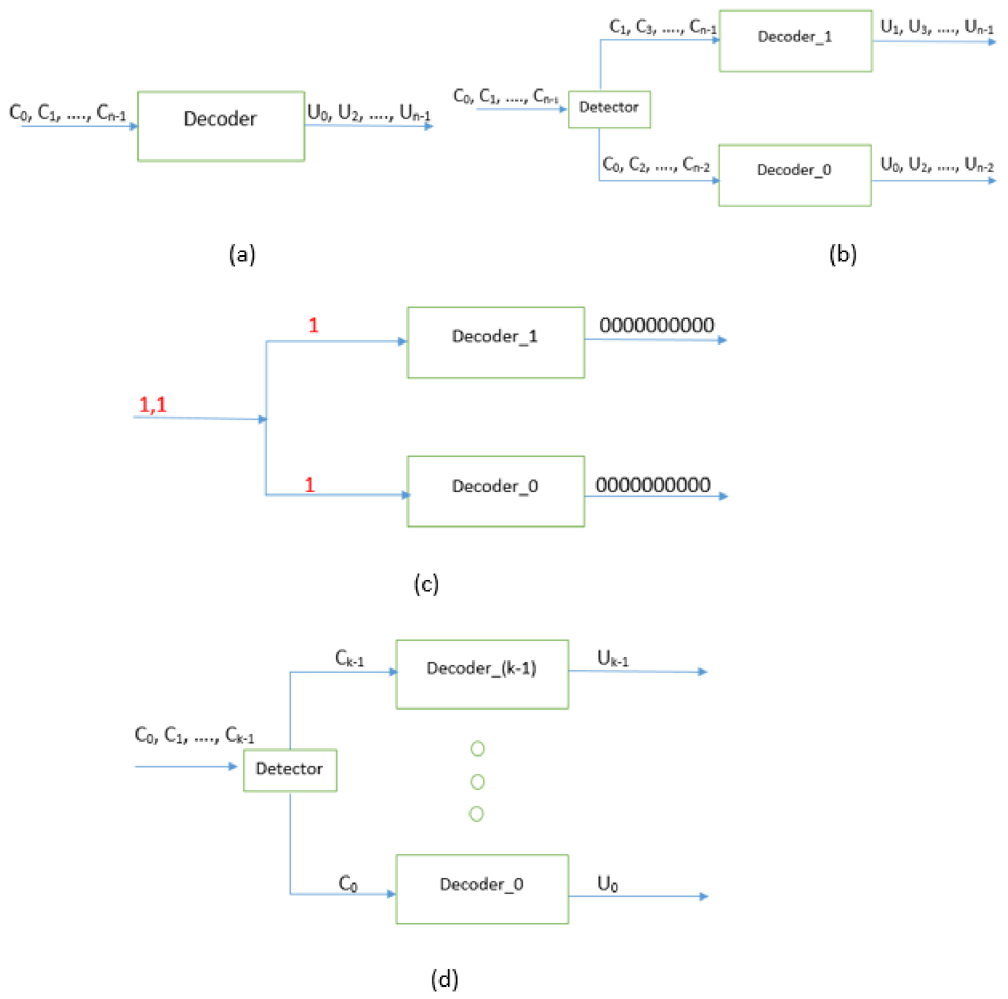


Figure 2.9: Multiple ways of decoding. (a) single decoder, (b) two decoders, (c) two decoders working in parallel when both compressed symbols are “1”, (d) k decoders for data decompression. Reprinted with permission from [1].

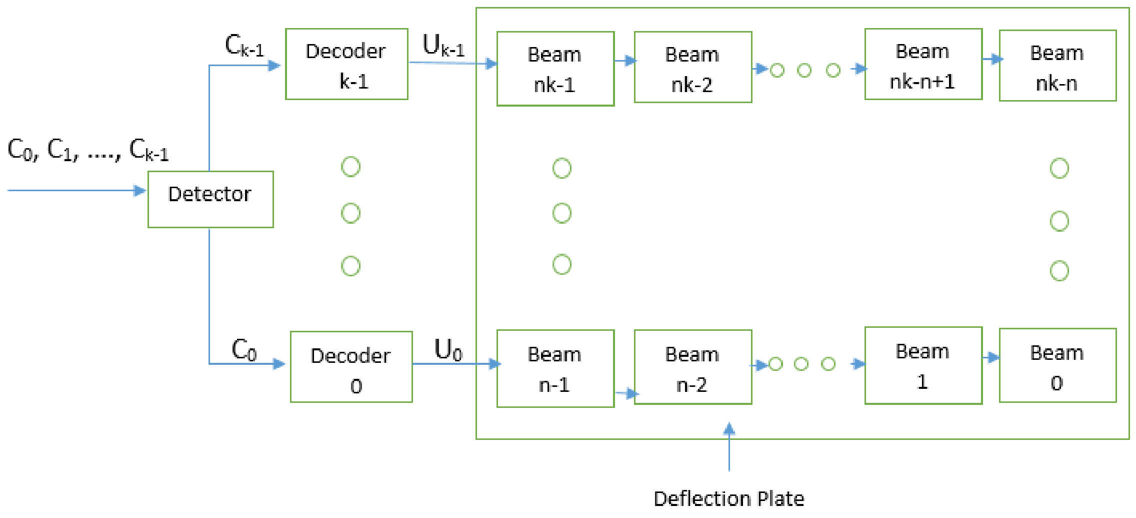


Figure 2.10: Logical decompression architecture with the deflection plate. Reprinted with permission from [1].

the purpose. Data compression is typically used to reduce the data transmission bottleneck from capacity constrained channels. The data communication from a large storage device to the side pad digital circuit of an FPGA or ASIC is also capacity constrained. Our simple compression scheme will mitigate the data communication problem. We next extend the compression scheme as shown in Table 5 to improve the compression ratio and decrease clock cycles. We introduce two extra symbols, namely, an all zero frame (AZF) with prefix 001 and an all zero column (AZC) with prefix 000. The all zero frame signifies that all the pixel values in the video frame have zero value which occurs when the $M * N$ beam shots all have zero dose value. In this case, the shot data does not need to be communicated to the deflection plate as the array should simply move to the next position. In uncompressed data transfer $M * N * 10$ clock cycles are needed but the AZF symbol instructs the beam array to move to the next position. The AZF symbol can be detected in 3 clock cycles. The AZC symbol signifies that all the pixels in a column of a frame have zero value. In this case, all the zero pixels can be generated in parallel according to our previous discussion of the parallel architecture. It takes 3 clock cycles to decode the AZC symbol and $M + 10$ clock cycles for the parallel generation of zero symbols for a column using the parallel architecture. The

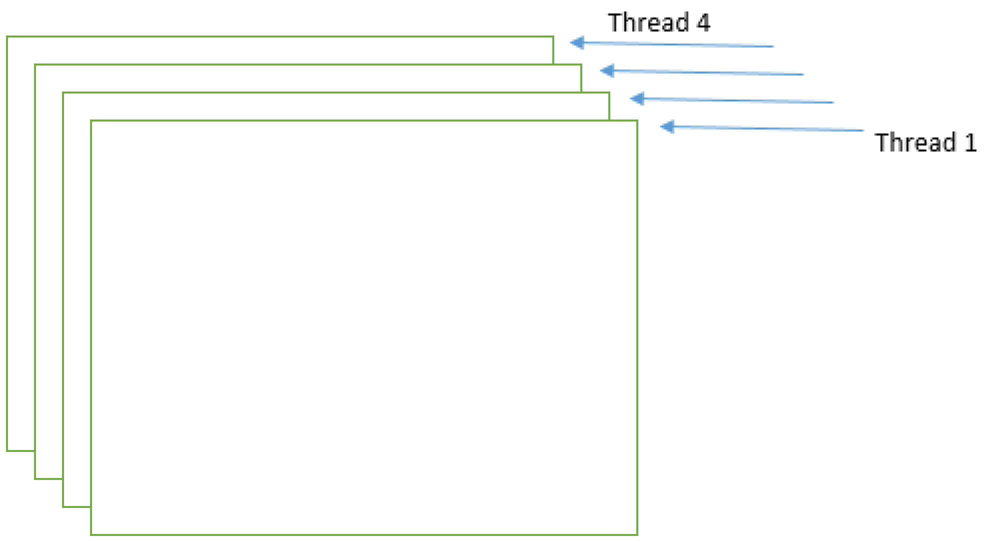


Figure 2.11: Compression of four frames in parallel on four different threads of execution. Reprinted with permission from [1].

zero symbol (Z) has the same representation and processing as in the simple compression scheme. The nonzero symbol (NZ) now uses 12 bits with a prefix of 01 and 12 cycles to decode. A similar parallel architecture with minor changes can be applied to the extended compression scheme.

The compression of data should happen offline as the compression algorithm for the extended scheme can run on thousands of processors. According to our "video" analogy each frame can be compressed on a separate thread of execution. Since there is no dependency between the frames in this compression algorithm, the parallel compression of data should provide nearly linear speedup (execution time inversely proportional to the number of processors). Figure 2.11 shows the compression of four frames in parallel on four different threads of execution.

2.6 Experiments and Results

We performed our experiments on two sets of data using the extended compression scheme. One image is an inverse lithography technology (ILT) mask motif pattern of contact holes with a smallest element of 80 nm. This pattern was enlarged by repeating it to generate a layout image of dimension 30017×33300 pixels with each pixel having dimensions of 10 nm $\times$ 10 nm. We also produce results for 23 layers of an image compression block (ICB) based on the FREEPDK45 45 nm library with a smallest element of 60 nm. The image size for the image compression block (ICB) data is 91209×90970 pixels with each pixel having dimensions of 10 nm $\times$ 10 nm. We used the electron beam proximity effect correction algorithm of GenISys, Inc., BEAMER_v4.6.2_x64, to generate proximity corrected images with 256 shot dose levels. Each pixel's shot dose level was multiplied by four to create 1024 dose levels. This does not change the nature of the data for our compression algorithm as the nonzero/zero doses remain nonzero/zero.

The images were divided into stripes and frames according to the scanning strategy of Table 2.2. Each frame was subsequently compressed according to the extended compression scheme. Two separate implementations of algorithms were done. In one implementation the frame generation and the data compression were done serially. In the other implementation the frame generation and the data compression were done in parallel. The implementations of the frame creation and compression algorithms are in C++, and the parallelization was done by the OpenMP library. The computation of the number of cycles to decode is implemented in C++. The experiments were performed on Intel i7-2600 CPU processors at 3.40GHz with 8 GB of RAM on a Windows7 Enterprise operating system. The processor contains four cores.

Table 2.6 shows the results for the compression ratios and the speedups. The compression speedup reported is the ratio of the execution times of the serial and parallel compression algorithms without write to memory. The decompression speedup reported is the ratio of the computed uncompressed transfer cycles to the computed decode cycles. The uncompressed data consists of the original image data and the extra zero padding for the frames at the edge of the image. The definition of the compression ratio is as follows

$$Uncompressed\,data = Original\,data\ +\ Zero\,padding\,data\,for\,edge\,frames, \tag{2.3}$$

Table 2.6: Data compression ratios and speedups with parallel decompression architecture and parallel compression speedup. Reprinted with permission from [1].

Data type and beam array ($M \times N$)	Uncompressed (MB)	Compressed (MB)	Compression ratio	Decompression Speedup	Compression Speedup (8 Threads)
23 layers ICB (128×127)	230471.5	3141.4	73.4	24.9	3.2
23 layers ICB (512×511)	267885.6	4372.3	61.3	22.4	2.4
2 ILT layers (128×127)	2500.4	262.6	9.5	6.6	3.1
2 ILT layers (512×511)	3730.2	309.1	12.1	7.6	2.0

$$Compression\ ratio = \frac{Uncompressed\ data\ size}{Compressed\ data\ size}. \tag{2.4}$$

Figure 2.12 shows a plot of the compression speedup with respect to the number of threads. We can see that the compression speedup increases with more threads. It is not a linear speedup in the number of threads probably due to the frame generation part of the algorithm and the memory accesses. The speedup of a single thread is not exactly equal to one as the serial and parallel codes were different. The compression ratio and decompression throughput are higher for the ICB data as that data is more sparse. In the ILT case the throughput and compression ratios are higher for the 512×511 beam array compared to the 128×127 beam array while the reverse occurs for the ICB data. Observe that the 512×511 beam array has a larger frame size and column size than the 128×127 beam array, and the probability of the AZF and AZC symbols are lower for a larger beam array when the sparsity is lower. The non-uniformity in decompression clock cycles per frame could be a concern for some datapath systems. We report the clock cycles per frame results in Figure 2.13 and Figure 2.14. Figure 2.13 shows the histogram of clock cycles that every frame takes for Layer 1 and Layer 2 of the ILT data with a 128×127 beam array. Figure 2.14

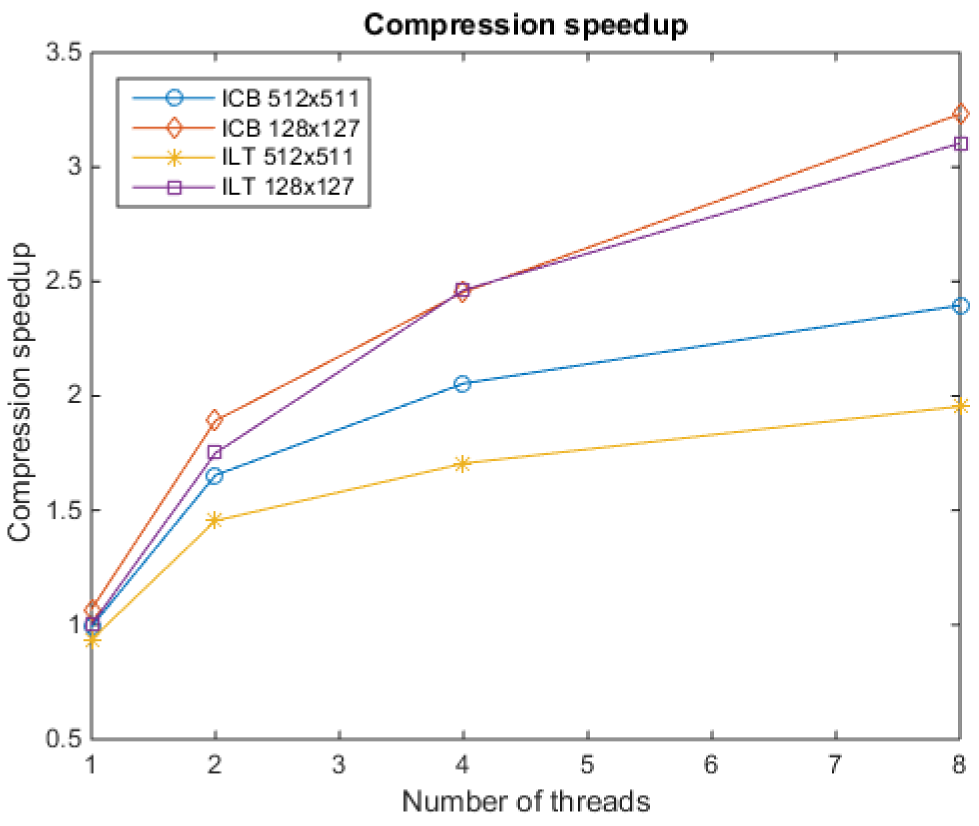


Figure 2.12: Compression speedup with respect to the number of threads in the quadcore machine. Reprinted with permission from [1].

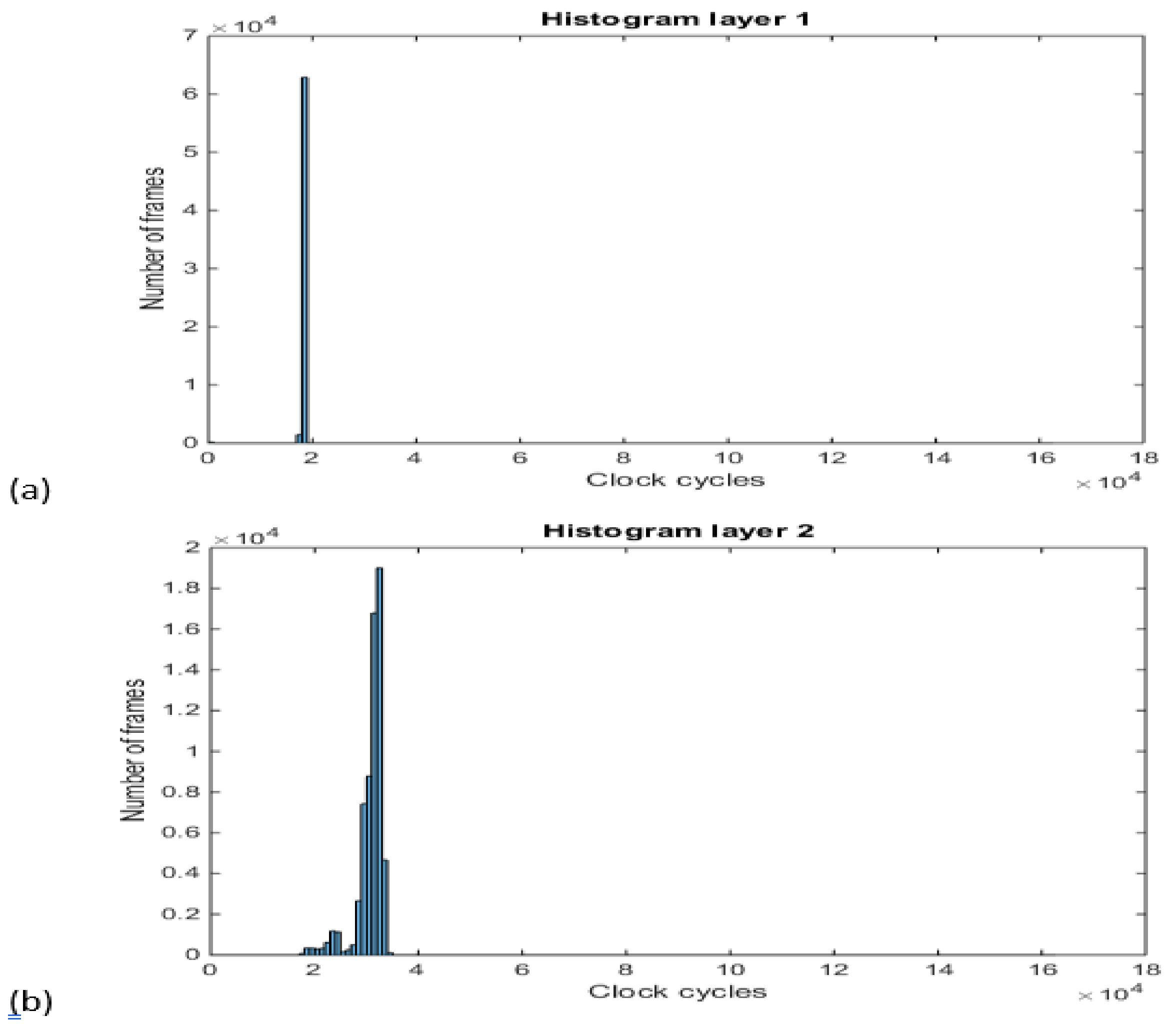


Figure 2.13: Histograms of clock cycles in the ILT data with a 128×127 beam array. (a) Layer 1. (b) Layer 2. Reprinted with permission from [1].

shows the plot of clock cycles taken by each frame in one stripe of Layer 2 of the ILT data with a 128×127 beam array. Layer 1 has 99.6% sparsity and Layer 2 has 92.2% sparsity. We can see that even for Layer 2 the maximum number of clock cycles taken by any frame stays well below the maximum possible cycles in the uncompressed data communication case ($128 \times 127 \times 10$ cycles) or the compressed data communication case ($128 \times 127 \times 12$ cycles).

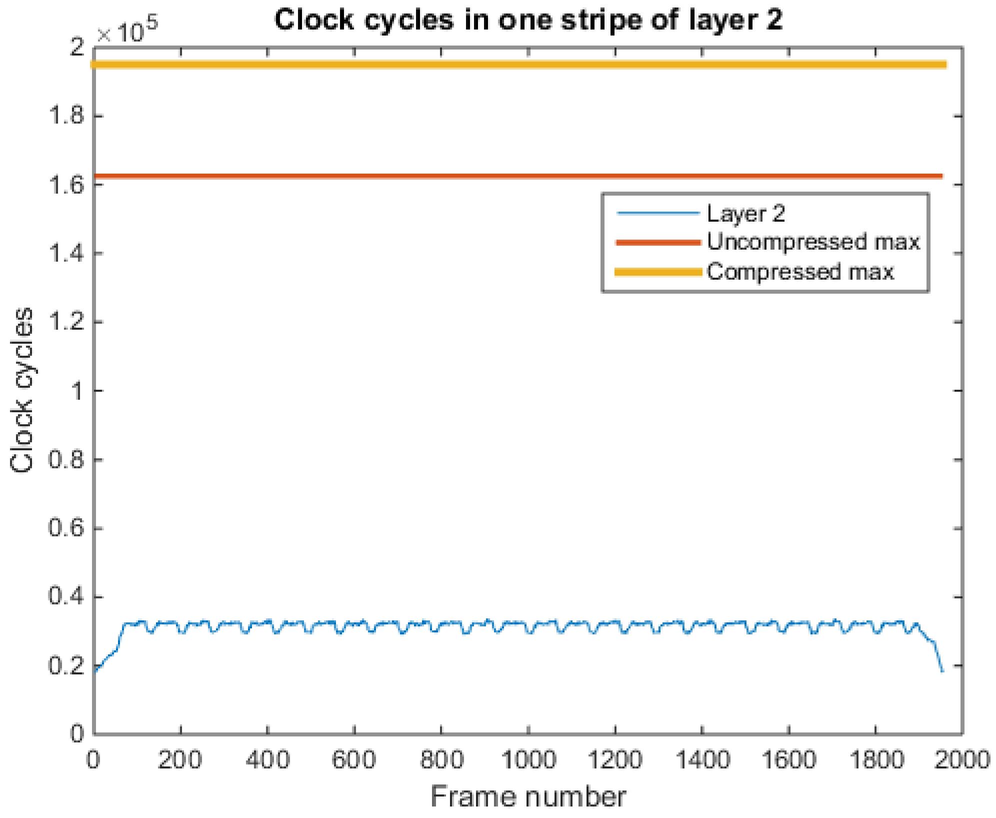


Figure 2.14: Clock cycles taken by the frames in one stripe of Layer 2 of the ILT data with a 128×127 beam array. Reprinted with permission from [1].

3. LINE ROUGHNESS ESTIMATION AND POISSON DENOISING IN SEM IMAGES USING DEEP LEARNING [*]

The semiconductor industry has demanding requirements for the accuracy and repeatability of nanostructure dimension measurements. The roughness of nanostructures, like lines, is a vital measurement to determine the quality of the lithography process. Scanning electron microscopy is the primary process to collect nanostructure geometry measurements. In SEM microscopy, the interaction of an electron beam with nanostructures forms a SEM image. A high energy or high dose electron beam produces low noise SEM images, but it increases acquisition time and damages the nanostructures. Conversely, low dose SEM images reduce nanostructure shrinkage and acquisition time, but the estimation of edges becomes difficult due to high Poisson noise. Additionally, the edges of nanostructures are brighter, and the region of increased brightness can be some nanometers in extent. The edge location in this bright region is unknown. The edges are brighter because secondary electrons have high escape probability at the edges. Figure 3.1 shows an example of a noisy SEM image of a line. Novel approaches are needed to estimate line edge roughness (LER) and line width roughness (LWR) from these types of images.

There have been significant advances in machine learning in the last decade. Deep convolutional neural networks [47, 48] can solve natural image classification problems [49, 48] and achieve superhuman performance on games [50] because they are effective at learning complex nonlinear models [48, 51]. For natural images corrupted by Gaussian noise, the state of the art in image denoising is attained by deep neural networks [7, 52]. Since semiconductor metrology has important inverse problems, deep supervised learning has the potential to advance the discipline. In this chapter, we propose the use of deep supervised learning for the estimation of line edge roughness (LER) and line width roughness (LWR) in low-dose scanning electron microscope (SEM) images.

[*]Part of the data reported in this chapter is reprinted with permission from N. Chaudhary, S. A. Savari, and S. S. Yeddulapalli, "Line roughness estimation and Poisson denoising in scanning electron microscope images using deep learning," *Journal of Micro/ Nanolithography, MEMS, and MOEMS*, vol. 18, no. 2, 024001, 2019. DOI: https://doi.org/10.1117/1.JMM.18.2.024001, Copyright [2019] by SPIE.

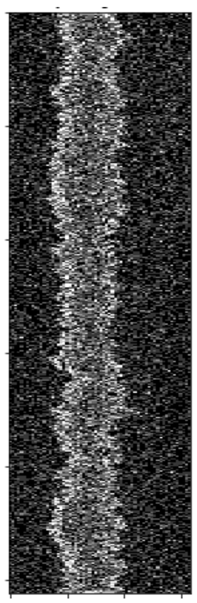


Figure 3.1: A noisy SEM image of dimension 64×1024. The image has one line with two edges. The aspect ratio of the image has been scaled for a better view. Reprinted with permission from [3].

We simulate supervised learning datasets of rough line SEM images constructed by means of the Thorsos method [53] and the ARTIMAGEN [54, 55] library developed by the National Institute of Standards and Technology. We also devise multiple separate deep convolutional neural networks called SEMNet, EDGENet, LineNet1, and LineNet2, each of which has 17 convolutional layers, 16 batch normalization layers and 16 dropout layers. SEMNet performs the Poisson denoising of SEM images. EDGENet directly estimates edge geometries from noisy SEM images. The LineNet1 and LineNet2 networks perform simultaneous denoising and edge estimation from single-line and multiple-line SEM images. Furthermore, we use multiple visualization tools to improve our understanding of the LineNet1 CNN and motivate the study of variations of LineNet1 with fewer neural network layers.

3.1 Deep Convolutional Neural Networks

Supervised machine learning is a branch of artificial intelligence which is well known for tackling classification problems. More generally, it is an approach to the problem of fitting parametric functions to sets of input/output data pairs. Let us assume x^i is an input vector/scalar and y^i is the corresponding output vector/scalar. Suppose that the true relationship between the output and the input can be described by a function $f^*(x)$, i.e.,

$$y^i = f^*(x^i). \tag{3.1}$$

Machine learning algorithms try to approximate the function $f^*(x)$ based on some loss criterion. For example, consider linear regression [56] with mean squared error (MSE) as the loss; here the algorithm searches for a linear function that minimizes the mean squared error over the training data. In particular, given a weight matrix W and a bias vector b the corresponding parametric linear function is

$$f(x) = Wx + b, \tag{3.2}$$

and the mean squared error (MSE) for N input/output data points is defined by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y^i - f(x^i))^2. \tag{3.3}$$

The solution to the linear regression problem minimizes the preceding loss over all of the data points (x^i, y^i) and the parameters W and b; i.e.,

$$f^*(x) \approx g^*(x) = \min_{\text{MSE } (W,\ b)} f(x). \tag{3.4}$$

Many machine learning approaches can be understood as generalizations of the previous problem. Numerous complex systems have a nonlinear relationship between input/output pairs; hence we wish to extend the types of estimates in (3.2) to parametric nonlinear functions. The study of feedforward neural networks has established that in various practical problems it is useful to focus on the composition of simple nonlinear functions. The rectified linear unit (ReLU) [47] is a widely

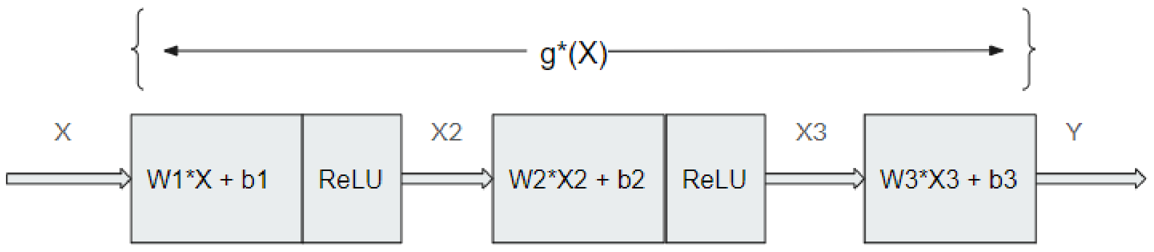


Figure 3.2: A three-layer neural network. Reprinted with permission from [4].

used nonlinearity which is defined by the following half wave rectifier:

$$y = \max(0, x) \qquad \text{(Rectified linear unit (ReLU))}. \tag{3.5}$$

Figure 3.2 illustrates a three-layer neural network which builds on the preceding ideas. Define weight matrices $W_1,\ W_2,\ W_3$ and bias vectors $b_1,\ b_2,\ b_3$. We can define a neural network with rectified linear unit (ReLU) nonlinearity to have the following form [57]:

$$f_{\text{NN}}(x) = W_3 \max(0, W_2 \max(0, W_1 x + b_1) + b_2) + b_3. \tag{3.6}$$

Observe that this function is the composition of three simpler nonlinearities or *layers*; i.e.,

$$f_{\text{NN}}(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))). \tag{3.7}$$

A layer is said to be *fully connected* if all the elements of its weight matrix and bias vector are allowed to vary.

General feedforward neural networks are parametric nonlinear functions which are the compositions of simple nonlinear functions. We focus here on ReLU nonlinearities, but alternate nonlinearities like the tanh function have also been implemented in neural network architectures. Assuming the three layers in Figure 3.2 are fully connected, the counterpart to equation (3.4) for that neural network function is

$$f^*(x) \approx g^*(x) = \min_{\text{MSE } (W1,\ W2,\ W3,\ b1,\ b2,\ b3)} f_{\text{NN}}(x). \tag{3.8}$$

Our focus will be on neural networks, but we mention that there are alternate approaches to machine learning such as linear support vector machines [58] and logistic regression which extend the basic linear model in different ways.

For deep neural network architectures, i.e., neural networks with several layers, the optimization problem in (3.8) and its extensions to networks with more layers are not convex, and the determination of globally optimum solutions is computationally intractable. Stochastic gradient descent [59] by means of the backpropagation algorithm [57] is the standard approach to obtaining good solutions for deep neural networks. Recent theoretical and empirical research suggests that the solutions obtained for deep neural networks are nearly always of similar quality independent of initial conditions [48, 60, 61].

The weight matrix dimensions in a neural network may vary by layer. In addition to fully-connected layers, neural network architectures can implement convolutional [62] layers, batch-normalization [63] layers and dropout [64] layers; we will describe these variations in the subsections below. In recent years neural networks have attained human-level performance on important problems [47, 49, 50] through increases in dataset sizes and extensive research on architectures [48]. One key development has been the replacement of some fully-connected layers with other types of layers such as convolutional [62] layers. Another insight is that deeper networks are often more effective at learning complex physical tasks [47, 49] because they offer the expression of more nonlinearities and hierarchical structure [48, 51], and there are neural networks with more than 100 layers [49]. A third major advance has come with the use of ReLU nonlinearities and residual networks [49]. Deep convolutional neural networks are now used in self-driving cars, image classification, and natural language processing, and they offer superhuman performance on board games [50].

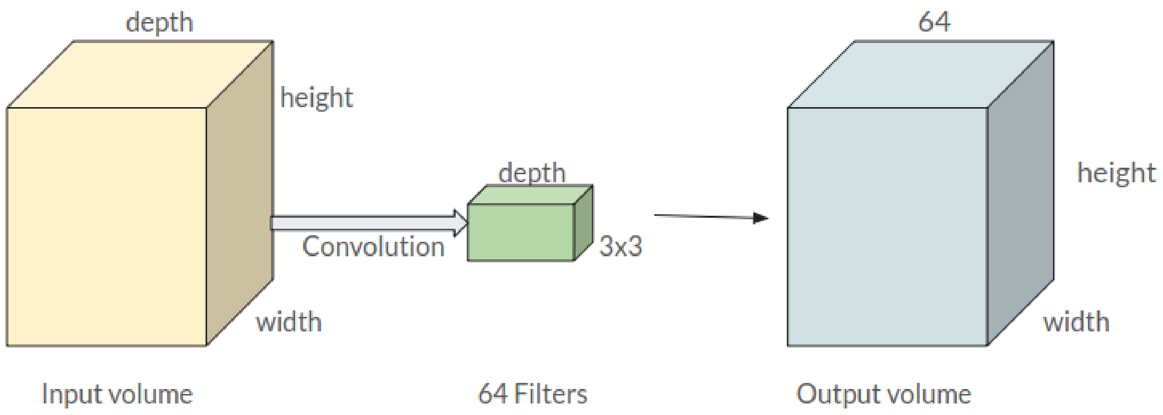


Figure 3.3: Convolutional layer with 64 filters. Reprinted with permission from [4].

3.1.1 Convolutional Layers

Recall that in a fully-connected neural network architecture each layer receives an input vector, multiplies it with a weight matrix and passes it as input to the next layer. The widths of these layers and the size of the input vector determine the dimensions of the weight matrices, and for image vectors the size of the problem is unwieldy [57]. Convolutional layers constrain the number of weights in an effective way by first arranging the input vector of a layer into a three-dimensional volume with dimensions width, height, and depth and by next performing convolution operations in each layer with multiple three-dimensional filters. Figure 3.3 illustrates a convolutional layer with 64 filters. The weights associated with these filters are the parameters of a layer. For example, color images use three values or channels per pixel to provide color information. An input color image of dimension $100 \times 100 \times 3$ could be convolved with 64 filters of dimension $3 \times 3 \times 3$ in a convolutional layer. That layer would have $64 \times 3 \times 3 \times 3$ trainable weights and 64 additional trainable bias terms, and the output volume of that layer would have dimensions $100 \times 100 \times 64$.

3.1.2 Dropout and Overfitting

Overfitting is a modeling error where a complicated function closely fits a small dataset. For example, fitting a cubic function to a linear dataset with a small number of points would result in overfitting. Regularization techniques like L1 and L2 regularization [65] reduce the possibility of overfitting a neural network architecture by constraining the norms of weight matrices. Dropout layers [64] which randomly drop network units and their connections during the training process force neural networks to learn "generic" features of training data and have become an important regularization technique to prevent the overfitting of deep neural networks.

3.1.3 Batch Normalization

In neural networks the stochastic gradient descent procedure performs gradient steps over small portions of the training data called *batches*. Normalizing the data to convert it into a zero mean and unit variance dataset facilitates the training of deep neural networks because it enables the training process to be more robust to the initialization of weights and the choice of the step size/learning rate used in stochastic gradient descent; normalization also provides some regularization [63]. A batch normalization layer [63] of a neural network normalizes the inputs to the next layer.

3.1.4 SEMNet

We devised our deep neural network named SEMNet to estimate denoised rough line images from noisy images with arbitrary levels of Poisson noise. SEMNet consists of 17 convolutional layers, 16 batch normalization layers and 16 dropout layers for regularization. SEMNet differs from earlier neural networks which were devised for the denoising of natural images in important ways; i.e., it was trained on relatively large SEM images as opposed to relatively small natural images and it incorporates dropout layers. Each convolutional layer of SEMNet except for the last layer has 64 filters of dimension $(3 \times 3 \times input\ depth)$ with bias terms and is followed by a batch normalization layer and a dropout layer with dropout probability of 0.2. The last convolutional layer has only one filter to output a grayscale image. Figure 3.4 illustrates the output volume or tensor obtained after each convolutional layer. SEMNet inputs a grayscale image with dimensions 64×1024 (width $\times$ height) and outputs another grayscale image with dimensions 64×1024. SEMNet has 559,233 total parameters out of which 557,185 are trainable parameters; the remaining parameters are associated with the batch normalization layers. We use the mean squared error

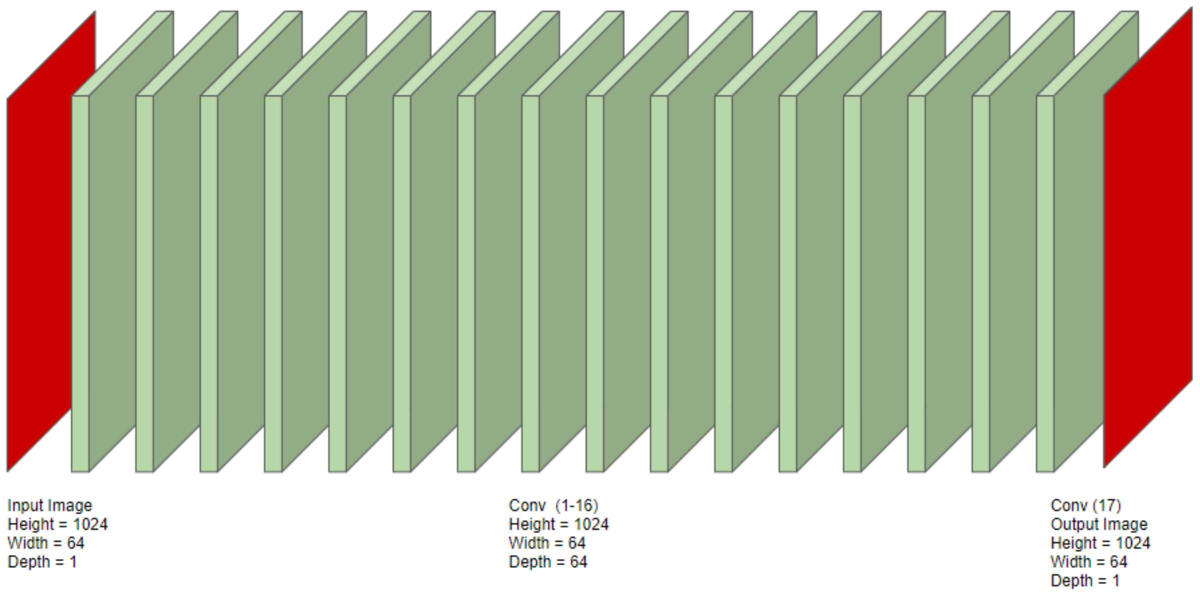


Figure 3.4: The 17 convolutional layers of SEMNet, which inputs a noisy SEM image of dimension 64×1024 and outputs a denoised image of dimension 64×1024. Reprinted with permission from [4].

(MSE) loss criterion for regression with SEMNet. Let x^i be the input noisy image matrix of dimension width $\times$ height, let $f(\cdot)$ be the neural network function predicting the output denoised image of the same dimensions, and let y^i be the target original image matrix of those dimensions. The Frobenius norm of an $m \times n$ matrix A with entries $a_{i,j}$ is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2}. \tag{3.9}$$

The mean squared error (MSE) loss for a collection of N image matrix pairs (x^i, y^i) is defined by

$$\text{MSE} = \frac{1}{N \times \text{width} \times \text{height}} \sum_{i=1}^{N} \|y^i - f(x^i)\|_F^2. \tag{3.10}$$

The proposed SEMNet architecture is one approach to denoising which has not been optimized. It is possible that the architecture could be made more efficient with techniques like downsampling or by changing the number of layers and/or the number of filters.

3.1.5 EDGENet

Our EDGENet neural network predicts two line edge positions from a 64×1024 noisy SEM image with pixel size 0.5×2 nm and containing one rough line as shown in Figure 3.1. The output of EDGENet is a 2-dimension matrix of size 2×1024 which contains the left and right edge positions of the line. The edge positions are reported with pixel-level precision and not with subpixel-level precision. EDGENet has seventeen convolutional layer with filter dimension of 3×3 and depth dimension equal to the depth of the input volume or tensor. The first four convolutional layers each have 64 filters, the next four convolutional layers each have 128 filters, the following four convolutional layers each have 256 filters and the subsequent four convolutional layers each have 512 filters. The last convolutional layer only has one filter. Figure 3.5 illustrates the output volume or tensor obtained after each convolutional layer. We use the mean absolute error (MAE) loss criterion for regression with EDGENet. Let x^i be the input noisy image matrix of dimension width$\times$height, let $f(\cdot)$ be the neural network function predicting the output array of edge positions, and let y^i be the target original array of edge positions with dimensions 2 $\times$ height. The L1-norm of an $m \times n$ matrix A with entries $a_{i,j}$ is defined by

$$\|A\|_1 = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|. \tag{3.11}$$

The mean absolute error (MAE) loss for a set of N training data pairs (x^i, y^i) is defined by

$$\text{MAE} = \frac{1}{N \times 2 \times \text{height}} \sum_{i=1}^{N} \|y^i - f(x^i)\|_1. \tag{3.12}$$

3.1.6 LineNet1 and LineNet2

We devised LineNet1 and LineNet2 to estimate denoised rough line images and the corresponding edge images from noisy images with arbitrary levels of Poisson noise. The paper [66] pointed out that neural networks can share information while performing multiple image process-

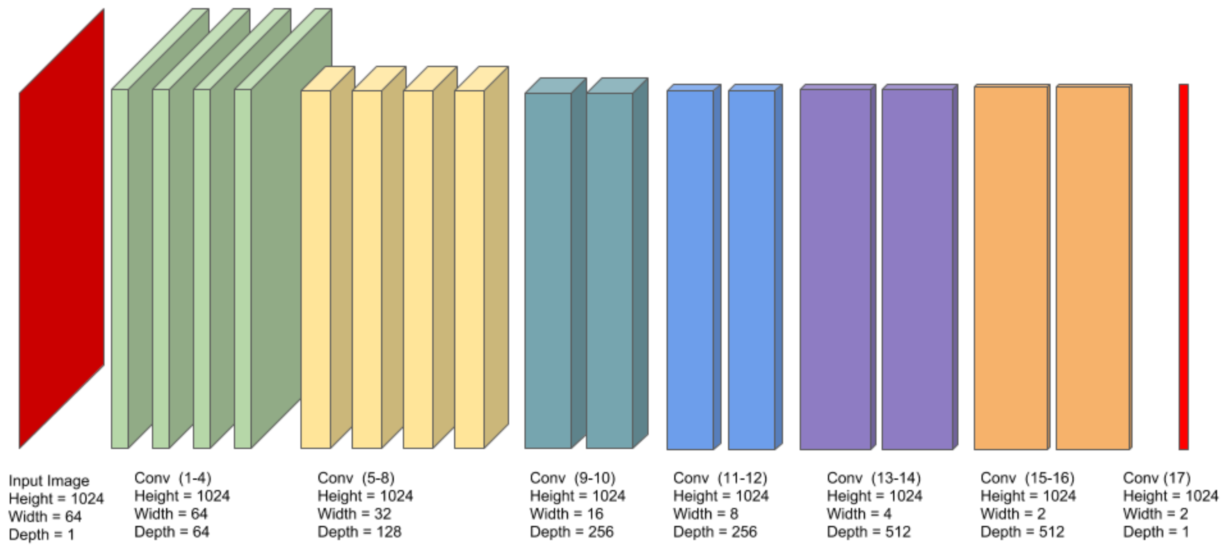


Figure 3.5: The 17 convolutional layers of EDGENet, which inputs a noisy SEM image of dimension 64×1024 and outputs a vector of dimension 2×1024. Reprinted with permission from [4].

ing tasks. LineNet1 and LineNet2 can save both time and memory by performing the tasks previously achieved by the two neural networks SEMNet [67, 4] and EDGENet [68, 4]. LineNet1 and LineNet2 [5] use neural network architectures similar to the SEMNet [67] architecture; i.e., they all have 17 convolutional [62], 16 batch-normalization [63] and 16 dropout [64] layers. Each convolutional layer of LineNet1 and LineNet2 except for the last layer has 64 filters of dimension $(3 \times 3 \times input\ depth)$ with bias terms and is followed by a batch normalization layer and a dropout layer with dropout probability of 0.2. The last convolutional layers of LineNet1 and LineNet2 each have two filters to output two grayscale images.

LineNet1 inputs a single-line SEM image of dimension 64×1024 pixels and outputs an array of dimension $64 \times 1024 \times 2$. The output array contains a denoised image and a grayscale edge image. Similarly, LineNet2 inputs a multiple-line SEM image of dimension 256×256 pixels and outputs an array of dimension $256 \times 256 \times 2$. We can estimate a larger denoised image and the corresponding edge image of larger dimension by first splitting the larger image into smaller images of dimension 256×256 pixels. We elected to study images with a pixel size of 0.5×2 nm, but the approach applies generally. We use MSE as the loss function for LineNet1 and LineNet2 since we found that the predicted edge image can have multiple missing edge positions if we instead use mean absolute error (MAE). LineNet1 and LineNet2 each have 559,810 parameters compared to the 10,972,993 parameters of EDGENet [68].

3.1.7 Related Denoising Work

Deep learning has been employed for image denoising in various ways over the years. Jain *et al.* first applied deep CNNs for image denoising [69]. Many successful denoisers have used their basic approach of minimizing a regression loss between network prediction images and ground truth images. For natural images corrupted by Gaussian noise, the state-of-the-art in image denoising is attained by deep neural networks [7, 52]. Zhang *et al.* used batch-normalization [63] and residual learning [49] in reference [7] to predict the noise at every pixel and compute a signal image from it. Additive white Gaussian noise (AWGN) is independent of the underlying signal; thus, it can be subtracted from a noisy image by assuming the image degradation model. If **x** is a ground truth image, **n** is AWGN noise and **y** is a corrupted noisy image, then by the image degradation model

$$\mathbf{y} = \mathbf{x} + \mathbf{n}. \tag{3.13}$$

Zhang *et al.* suggests in reference [7] that with residual learning, DnCNN implicitly removes the latent clean image in the hidden layers. It alters the inputs of each layer to be Gaussian-like distributed and less related to image content. There have been other approaches to denoising that have used an encoder-decoder architecture [70], symmetric skip connections [71], and generative adversarial networks (GANs) [72]. Mao *et al.* used an encoder-decoder architecture in reference [70] with symmetric skip connections for the denoising task. MemNet [71] by Tai *et al.* used recurrent persistent memory units in their architecture. Chen *et al.* has also presented an image restoration approach [72] based on generative adversarial networks (GANs). Tal *et al.* used residual learning by extracting negative noise components [73] from every layer of a 20 layer CNN. They performed Poisson denoising of natural images and found class-aware denoising [74] to be useful. We introduced our deep CNN network SEMNet [67, 4] in 2018 for the Poisson denoising of SEM images with the ultimate goal to improve line edge roughness measurements. We took several steps specific to the problem of SEM metrology.

Our first deep CNN network SEMNet [67, 4] differs from DnCNN in three important ways. Firstly, SEMNet uses 16 dropout layers with dropout probability of 0.2 after each pair of convolutional and batch-normalization layers. Secondly, SEMNet directly predicts a clean original image. It does not use residual learning to predict a noise image. Lastly, SEMNet was trained on large SEM images of dimension 64×1024 as opposed to relatively small patch sizes in DnCNN. We used dropout to increase regularization and to make the network more robust to slight deviations in signal and noise characteristics. We considered that it was needed because SEM image acquisition processes can be affected by other sources of errors like vibrations, temperature changes, and instrument errors. Additionally, we did not use residual learning to learn a noise image because Poisson noise is signal-dependent [75]. Poisson noise cannot be subtracted out by assuming the noise degradation model. Due to the high dependence of noise on the scene [75], we believe it is better to learn the noisy image to original image relationship. Furthermore, we trained the network on full-sized SEM images instead of cutting images into smaller patches. Cutting random rough line SEM images into many small patches can create artifacts around the edges. Due to the edge effect in SEM images, pixel intensities around the line edges are also affected by the neighboring nanostructure geometry. Since we aimed to measure line edge roughness, we believed full-size image training protects the integrity of edge measurements. Training on full-sized images can also help Poisson denoising by increasing scene awareness.

Subsequent to our SEMNet work, Weigert *et al.* introduced the CARE software [76] framework for image restoration in fluorescence microscopy data. They acquired pairs of low- and high-exposure-images to create their training dataset and employed the U-Net [77] architecture for their experiments. Lehtinen *et al.* introduced the NOISE2NOISE (N2N) [78] approach where independent pairs of noisy images can be used to train the network without any need for clean target images. Acquiring multiple low-dose and high-dose images for the exact target is not feasible for SEM metrology. SEM image acquisition at high dosages can increase resist shrinkage, and our goal is to resolve this problem and acquire images at low dosages. It is challenging to acquire the same image signal through multiple image acquisitions due to alignment and skew changes at the nanometer scale. Recently, Krull *et al.* introduced NOISE2VOID (N2V) [79], a training scheme which does not require noisy image pairs or clean target pairs. However, as mentioned in their paper [79], there are some limitations of this approach. Specifically, N2V makes two statistical assumptions: 1) the signal is not pixel-wise independent and 2) the noise is conditionally pixel-wise independent given the signal. The performance of N2V decreases with violations of these assumptions [79]. We do not know if these assumptions hold in SEM images. Ulyanov *et al.* show that the structure of deep image prior CNNs resonates [80] with the structure of natural image data. They input a random noise image to a CNN and train it to reconstruct a single noisy image as the output. They show that the network resists "bad" solutions and converges towards natural-looking images. This deep image prior network can do image denoising without requiring additional training data

[80]. However, we do not know if SEM images of random rough lines resonate with the structure of CNNs and if the prior will preserve the roughness of lines.

Our second deep CNN network EDGENet is not a denoising network. It predicts two line edge positions from a 64×1024 noisy SEM image containing one rough line. Its output is a 2-dimension matrix of size 2×1024, which contains the left and right edge positions of the line. Our third set of deep CNN networks are LineNet1 and LineNet2. We devised LineNet1 and LineNet2 to estimate denoised rough line images and the corresponding edge images from noisy images with arbitrary levels of Poisson noise. Schwartz *et al.* pointed out in [66] that neural networks can share information while performing multiple image processing tasks. The edge estimation problem from SEM images is unique to SEM metrology due to edge effects. We do not know of other deep CNN approaches designed for the edge estimation problem from SEM images.

3.2 Simulation and Training Datasets

Large datasets are needed to train deep convolutional neural networks. We developed our datasets of SEM images by simulation. We generated two separate datasets to train our neural networks. The first dataset contained single-line SEM images for the training of the SEMNet, EDGENet, and LineNet1 networks. The second dataset contained multiple-line SEM images for the training of the LineNet2 network.

3.2.1 Rough Edge Simulation

We first use the Thorsos method [53, 81] with normally distributed random variables to simulate rough line edges. Thorsos [53] described an approach to generate simulations of a surface height function which approximately satisfy a specified roughness spectrum and have Gaussian distributed heights and slopes. Mack [81] pointed out that the technique is widely used and is known to other communities as the Monte Carlo spectral method, and he analyzed the statistical bias associated with the method for rough lines of finite length and positive correlation length. The Palasantzas spectral model [82] is characterized by three parameters: σ represents the line edge roughness (LER), i.e., the standard deviation of edge positions, α denotes the roughness (or

Hurst) exponent and ξ is the correlation length. For $x > 0$, the gamma function $\Gamma(x)$ is defined by $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$. Each simulated rough line edge has the following underlying Palasantzas spectral model [82]:

$$PSD(f) = \frac{\sqrt{\pi}\Gamma(\alpha + 0.5)}{\Gamma(\alpha)} . \frac{2\sigma^2\xi}{(1 + (2\pi f\xi)^2)^{\alpha+0.5}}. \tag{3.14}$$

Each simulated edge has length 2.048 microns and corresponds to 1024 pixels. To consider a diverse collection of models we generated edges for eight values of LER (σ = 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8 nm), nine values of Hurst/roughness exponent (α = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9) and 35 values of correlation length (ξ = 6, 7, ..., 40 nm).

3.2.2 Single-line Dataset

To make 10080 lines, we generated eight edges for each of the 2520 possible combinations of parameters (σ, α, ξ). We next used the SEM simulator ARTIMAGEN [54, 55] to generate 10080 images of dimension 64×1024 pixels with pixel width 0.5 nm and pixel height 2 nm. We selected ARTIMAGEN because it is publicly available and because of its ability to generate a large number of SEM images from simulated rough lines relatively quickly; we selected our image dimensions to handle the computational demands associated with the training of a deep neural network. Each image contains a line of width 10 nm or 15 nm with two of the previously generated rough edges. The locations of the lines within the images vary. The simulation required a discretization of the edge positions because the library does not provide fractional edge positions. The discretization lattice size was equal to the pixel size. This is a limitation of the simulation as a real edge position can occur anywhere within a pixel. The lines were created using the table of curves feature of the ARTIMAGEN library. The features of the ARTIMAGEN simulator enabled us to construct images which incorporate random backgrounds, a fixed edge effect, fine structure and Gaussian blur. We used the parameter settings suggested by examples within the ARTIMAGEN library. For example, we added Gaussian blur to all images with a Gaussian point spread function with parameter values sigma = 0.5, astigmatism ratio = 1, and astigmatism angle = $30°$.

We use a background density of 8 with a gray level in the range $(0.1, 0.2)$. The edge effect in SEM images makes edges appear brighter than the rest of the structure. In the ARTIMAGEN library the edge effect profile is approximated by an exponential [54]. We used the parameter value determining the steepness of the transition to be 0.4 and the top edge value above base as 0.5 for the edge effect in all images. Similarly, fine structure was added with following parameter values (density = 70e-4, min_radius = 6, max_radius = 10, min_coefficient = 0.95, max_coefficient = 1.05).

Our original image set is the collection of these 10080 images. From each original image using the ARTIMAGEN library we generated ten noisy images corrupted by Poisson noise with electron density per pixel in the range $\{2,\ 3,\ 4,\ 5,\ 10,\ 20,\ 30,\ 50,\ 100,\ 200\}$. Thus, the noisy image dataset consists of 100800 images. From the noisy and original images we constructed a supervised learning dataset of pairs of images $(x^i,\ y^i)$ for the training of SEMNet, where the input x^i is a noisy image and the output y^i is the corresponding original image. For the training of EDGENet, the input x^i is a noisy image and the output y^i is an array of dimension 2×1024 with edge positions. For the training of LineNet1, the input x^i is a noisy image and the output y^i is an array of the original image with the edge image. Figure 3.6 shows one example of single-line dataset images with a noisy image, an original image and a corresponding edge image.

3.2.3 Multiple-line Dataset

The rough edges in multiple-line dataset images were also constructed using the Thorsos method with the same 2520 possible combinations of parameters for LER (σ), roughness exponent (α) and correlation length (ξ). This multiple-line dataset contains 50,400 simulated images with dimension of 256×1024 pixels. Each larger image contains three or four rough lines of width 15 nm or 10 nm separated by a space of twice the linewidth. The locations of the lines within the images vary. We trained LineNet2 on this multiple-line dataset of simulated images. LineNet2 is designed to construct denoised and estimated edge images of size 256×256 pixels. Hence the initial images of dimensions 256×1024 were each separated into four nonoverlapping images before being processed by LineNet2. Each set of four denoised and estimated edge images were

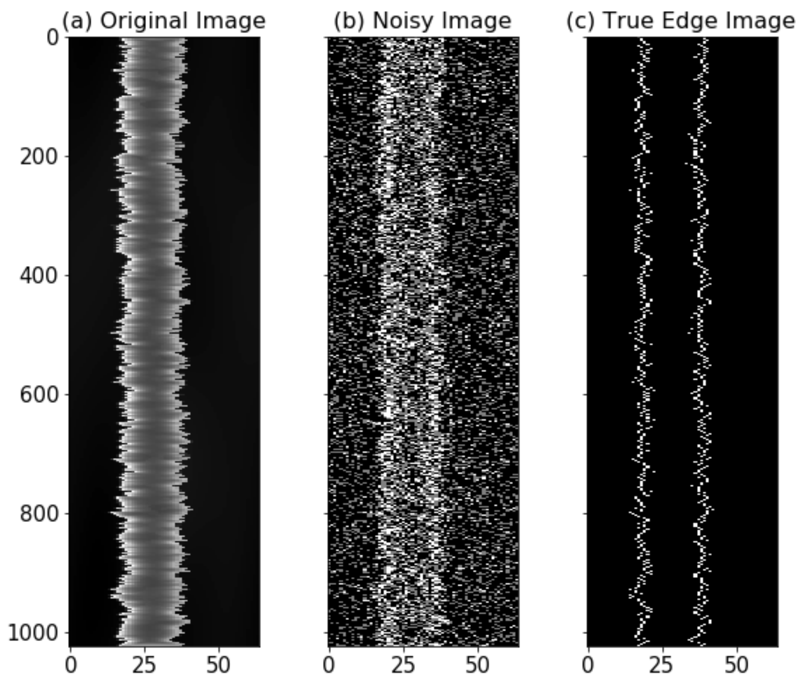


Figure 3.6: Single-line dataset images. (a) An original simulated SEM image of dimension 64×1024. (b) Noisy image after Poisson noise addition. (c) Edge image. The aspect ratio of the images has been scaled for a better view. Reprinted with permission from [5].

later recombined to form an image of dimension 256×1024. This approach can be generalized to handle arbitrary images. Figure 3.7 shows one example of a multiple-line dataset image with a noisy image, an original image, and a corresponding edge image.

3.3 Experiments and Results

The simulated dataset of 100800 noisy-original image pairs, edge images and edge arrays was divided into a training set, a validation set and a test set. The validation set consisted of the 2880 image pairs, edge images and edge arrays with correlation length $\xi = 20$ nm. The test set consisted of the 8640 image pairs, edge images and edge arrays with correlation length ξ in the set $\{10,\ 30,\ 40\}$ nm. The training set consisted of the remaining 89280 noisy-original image pairs with edge images and edge arrays. All SEM image inputs, edge images and edge arrays were normalized to have values in the range (0,1).

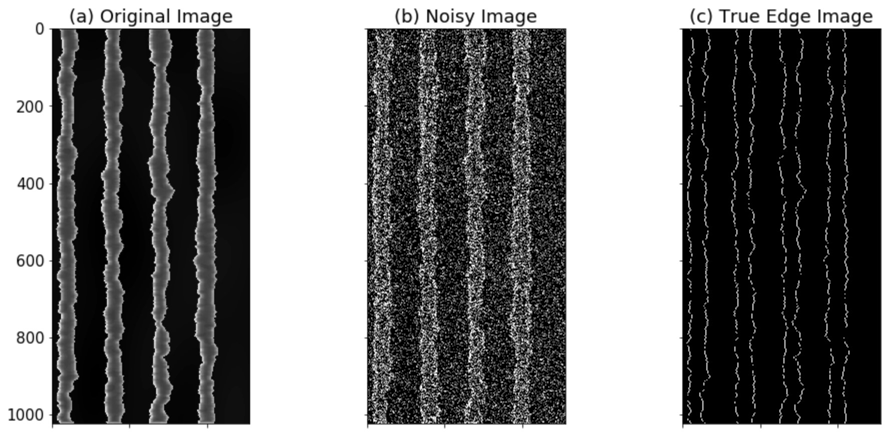


Figure 3.7: Multiple-line dataset images. (a) An original simulated SEM image of dimension 256×1024. (b) Noisy image after Poisson noise addition. (c) Edge image. The aspect ratio of the images has been scaled for a better view. Reprinted with permission from [5].

SEMNet, EDGENet and LineNet1 were trained with a Tesla K80 GPU and an Intel Xeon E5-2680 v4 2.40GHz node, and they were created and trained using the python programming language library Keras [83] with the Tensorflow [84] library backend. In the training of all three neural networks, we used the Adam [85] optimizer algorithm for stochastic gradient descent and selected a learning rate of 0.001 and batches with eight input-output pairs; for SEMNet each input-output pair is a noisy-original image pair while for EDGENet each input-output pair is a noisy SEM image together with the corresponding edge array. The LineNet1 input-output pair is a noisy SEM image together with the corresponding original image and edge image. The small batch size was chosen due to the memory constraints. An epoch of training consisted of stochastic gradient descent steps on the entire training set of 89280 image pairs and edge arrays.

The simulated multiple-line dataset for LineNet2 was also divided into a training set, a validation set and a test set. These LineNet2 dataset images had dimension 256×1024 pixels. The test set consisted of the 4320 noisy-original SEM images and edge images with correlation length ξ in the set $\{10,\ 30,\ 40\}$ nm. The validation set consisted of the 1440 noisy-original SEM images and edge images with correlation length $\xi = 20$ nm. The training set consisted of the remaining 44,640 noisy-original SEM images and edge images.

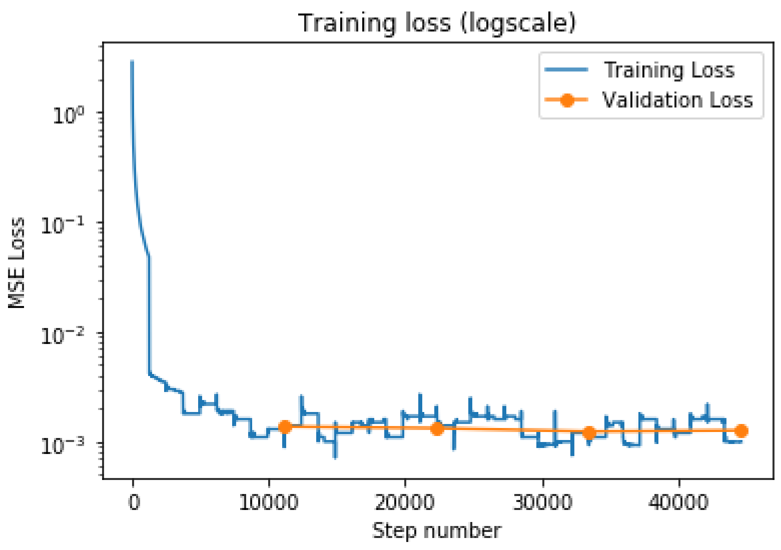


Figure 3.8: Training and validation loss for SEMNet. Reprinted with permission from [4].

3.3.1 SEMNet Results

We trained SEMNet for four epochs, and the training time was approximately 41 hours. We chose mean squared error (MSE) as the loss function for SEMNet. Figure 3.8 illustrates the evolution of mean squared error on the batches of the training set with respect to the number of stochastic gradient descent steps. Figure 3.8 also shows the loss on the validation set after each epoch of training. It is evident that the validation loss does not significantly differ from the training loss, and therefore overfitting is unlikely. After we trained SEMNet for four epochs we saved the final set of weights and biases in each layer and used these parameters to denoise the previously unseen images in the test set. The average denoising time per image of SEMNet was 1.96 seconds on a central processing unit. This runtime data does not include the time to load the model as this operation was performed only once.

Our first set of results compares SEMNet denoising to state-of-the-art image denoising algorithms. In particular, we consider the total variation (TV) algorithm (split Bregman optimization) [6], the convolutional neural network DnCNN [7] and a denoiser based on a Daubechies wavelet [8]. The DnCNN denoised images were generated through MATLAB's Neural Network toolbox and the other earlier algorithms we consider were available through the scikit-image python library. The top left image in Figure 3.9 shows a noisy test image not used in the training of SEMNet with a noise level of 2 electrons per pixel. The remaining images in Figure 3.9 consist of the corresponding original image and the images obtained by applying the four denoisers. Observe that the quality of the SEMNet denoiser is good even in the high noise regime.

We use a variety of performance metrics to assess the relative quality of the four denoisers on the test image dataset. The first is peak signal-to-noise ratio (PSNR), which is a common measure for the quality of image reconstruction. Let MAX_I denote the maximum pixel intensity of an image and let MSE as usual denote the mean squared error between a reconstructed image and the corresponding original image. Then PSNR (in dB) is defined by

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}\right), \tag{3.15}$$

and a larger value of PSNR indicates better denoising. For the other performance metrics we consider we need to combine the outputs of the denoising procedures with an edge detection algorithm, and we use the Canny algorithm for this purpose. In addition to examining LER and LWR we introduce a metric called the mean absolute pixel error (MABSE) which averages the absolute edge position estimation error measured in pixels over 2044 edge positions per line. We consider only 2044 positions per line instead of the total 2048 positions because the Canny edge detector does not detect edge positions at the image boundaries.

Table 3.1 quantifies the relative performances of the denoisers for three test images in terms of peak signal-to-noise ratio (PSNR) and demonstrates the superiority of SEMNet in different settings. Table 1 also includes the results from LER estimation for the central 1022 (out of 1024) pixels. The σ_C parameters for the Canny edge detection algorithm were chosen from the set {0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4} to optimize the LER estimation errors. SEMNet again attains the best performance and produces an output which is close to the true edge geometry. However, the Canny σ_C parameter must be optimized for every image because each image contains a different

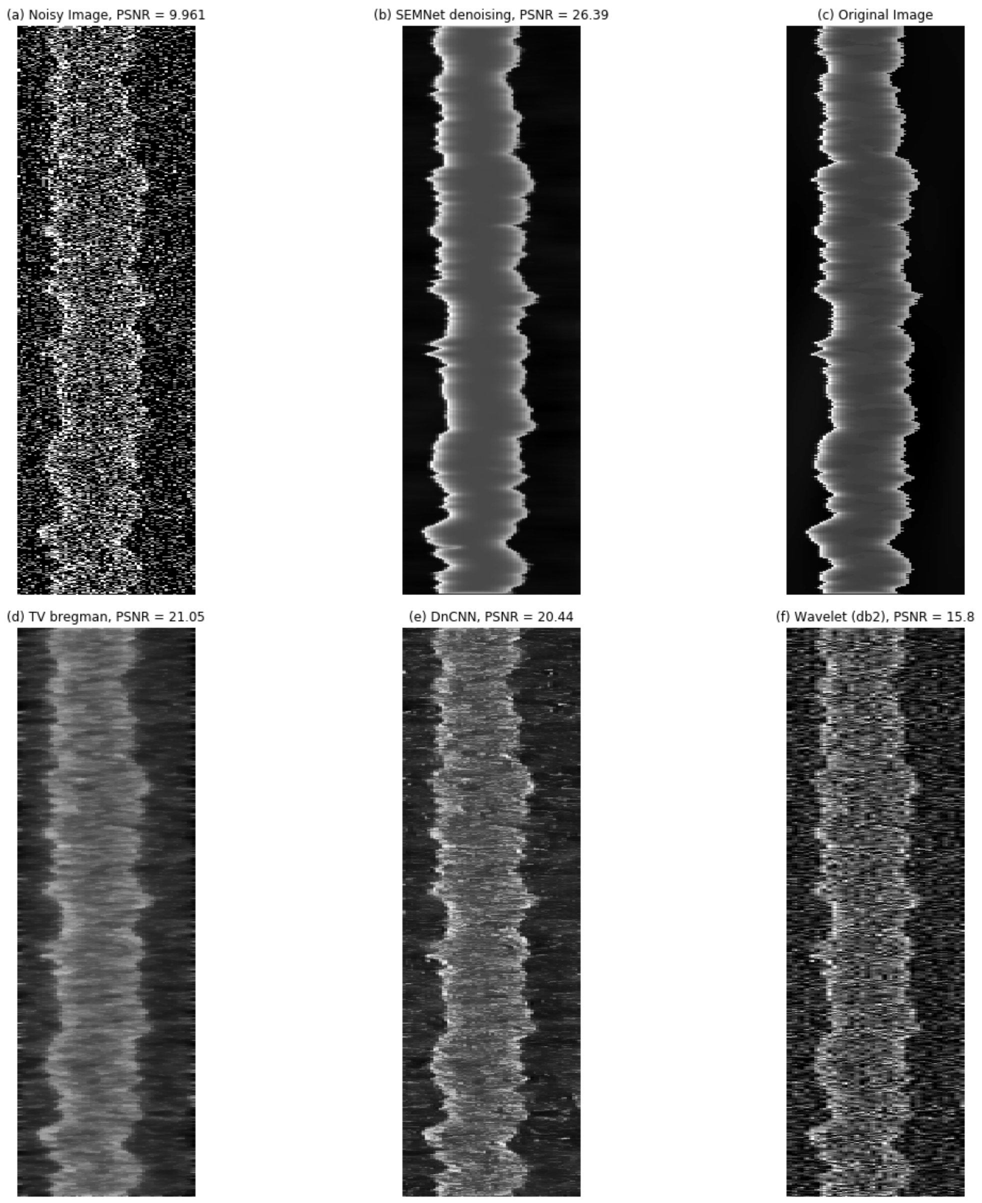

(a) Noisy Image, PSNR = 9.961
(b) SEMNet denoising, PSNR = 26.39
(c) Original Image
(d) TV bregman, PSNR = 21.05
(e) DnCNN, PSNR = 20.44
(f) Wavelet (db2), PSNR = 15.8

Figure 3.9: (a) Noisy image with a noise level of 2 electrons per pixel. (b) Estimated image from deep neural network SEMNet. (c) Corresponding original image without Poisson noise. (d) Total variation (Bregman method) [6] denoising. (e) DnCNN [7] denoising. (f) Wavelet (db2) [8] denoising. Reprinted with permission from [4].

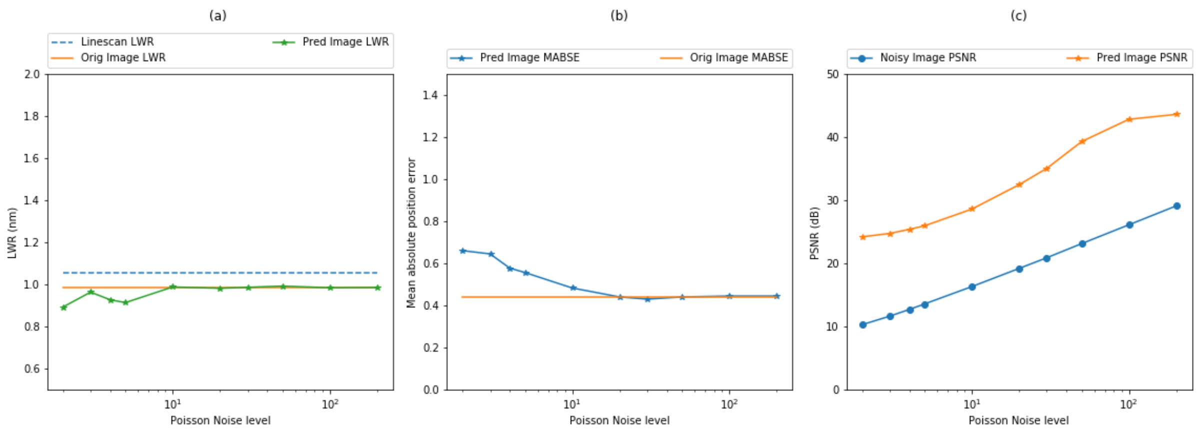


Figure 3.10: Results of SEMNet for image with parameters $\sigma = 0.8$ nm, $\xi = 10$ nm, $\alpha = 0.3$ and for Canny parameter $\sigma_C = 0.5$. The x-axis is on a logarithmic scale and specifies Poisson noise levels in electron density per pixel units. (a) Plots of estimated LWR from denoised images (Green), true LWR (Blue) and estimated LWR from original image with no noise (Orange). (b) Plots of mean absolute pixel error (MABSE) from noisy images (Blue) and from original image (Orange). (c) Plots of predicted image PSNR (Orange) and noisy image PSNR (Blue). Reprinted with permission from [4].

noise level and edge characteristics. We developed the second neural EDGENet to eliminate this weakness and directly detect line edges.

The Canny edge detection scheme uses a filter, and a recent paper [86] advises against filtering. Therefore, to offer a second comparison among the denoisers we propose and consider an "intensity-based" edge detection scheme without a filter which applies the scientific python (SciPy) peak detector function find_peaks. A *peak* of a one-dimensional array of numbers refers to a local maximum, and in the case of a flat peak find_peaks returns the index of the middle sample of that peak. The function find_peaks enables the selection of subset of peaks which satisfy certain constraints. In our following definition of the intensity-based edge detection algorithm, the parameter $width$ represents the width of the line in pixels; in our case, $width = 20$ for a 10 nm line and $width = 30$ for a 15 nm line. The intensity-based edge detection algorithm takes as input the intensity profile of a linescan, detects a collection of peaks each with intensity above 20% of the maximum pixel intensity and with minimum peak distance of $width/4$, selects the two highest

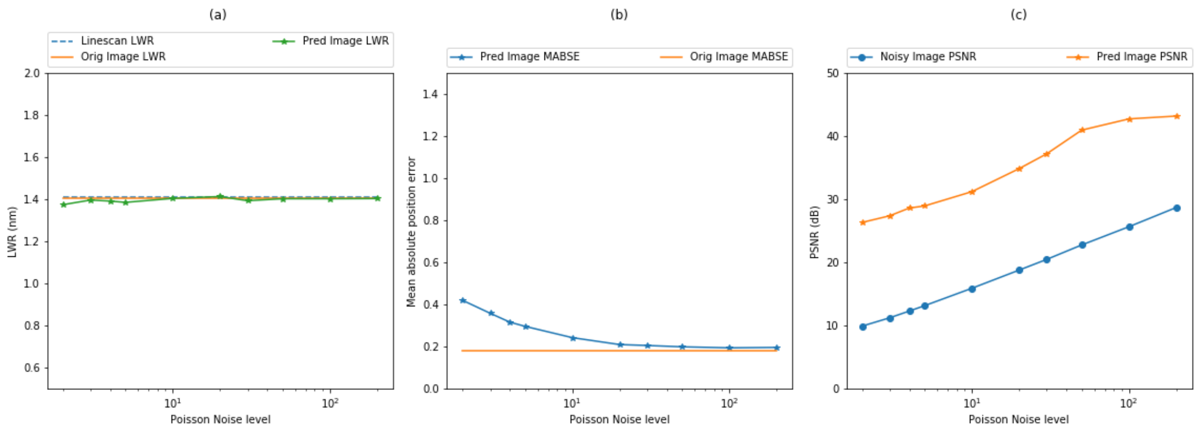


Figure 3.11: Results of SEMNet for image with parameters $\sigma = 1.2$ nm, $\xi = 40$ nm, $\alpha = 0.7$ and for Canny parameter $\sigma_C = 0.5$. The x-axis is on a logarithmic scale and specifies Poisson noise levels in electron density per pixel units. (a) Plots of estimated LWR from denoised images (Green), true LWR (Blue) and estimated LWR from original image with no noise (Orange). (b) Plots of mean absolute pixel error (MABSE) from noisy images (Blue) and from original image (Orange). (c) Plots of predicted image PSNR (Orange) and noisy image PSNR (Blue). Reprinted with permission from [4].

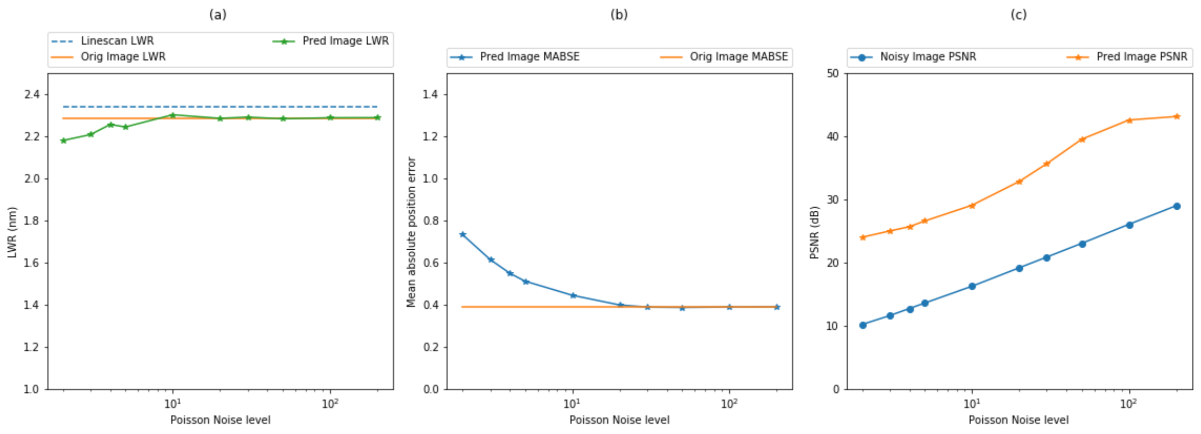


Figure 3.12: Results of SEMNet for image with parameters $\sigma = 1.6$ nm, $\xi = 30$ nm, $\alpha = 0.5$ and for Canny parameter $\sigma_C = 0.5$. The x-axis is on a logarithmic scale and specifies Poisson noise levels in electron density per pixel units. (a) Plots of estimated LWR from denoised images (Green), true LWR (Blue) and estimated LWR from original image with no noise (Orange). (b) Plots of mean absolute position error (MABSE) from noisy images (Blue) and from original image (Orange). (c) Plots of predicted image PSNR (Orange) and noisy image PSNR (Blue). Reprinted with permission from [4].

Table 3.1: Denoising and LER results. The LER data is for the central 1022 out of 1024 pixels for each image. Reprinted with permission from [4].

Original Image σ(nm), ξ(nm), α	Poisson noise level	PSNR noisy (dB)	Denoiser method	PSNR denoised (dB)	Canny σ_C	Left edge (nm) σ (true)	Left edge (nm) σ (obs.)	Right edge (nm) σ (true)	Right edge (nm) σ (obs.)
0.8, 10, 0.3	None	-	None	-	0.5	0.74	0.69	0.74	0.69
0.8, 10, 0.3	10	16.38	SEMNet	**28.66**	0.5	0.74	**0.68**	0.74	**0.69**
0.8, 10, 0.3	10	16.38	TV	22.39	1.5	0.74	0.59	0.74	0.59
0.8, 10, 0.3	10	16.38	DnCNN	25.24	1.5	0.74	0.61	0.74	0.68
0.8, 10, 0.3	10	16.38	Wavelet	20.91	2.5	0.74	0.58	0.74	0.64
1.2, 40, 0.7	None	-	None	-	0.5	1.07	1.06	1.21	1.20
1.2, 40, 0.7	2	9.96	SEMNet	**26.39**	0.5	1.07	1.03	1.21	**1.21**
1.2, 40, 0.7	2	9.96	TV	21.05	3	1.07	**1.10**	1.21	1.77
1.2, 40, 0.7	2	9.96	DnCNN	20.44	3	1.07	1.14	1.21	1.35
1.2, 40, 0.7	2	9.96	Wavelet	15.80	3	1.07	1.10	1.21	1.83
1.6, 30, 0.5	None	-	None	-	0.5	1.57	1.53	1.59	1.56
1.6, 30, 0.5	100	26.13	SEMNet	**42.64**	0.5	1.57	**1.53**	1.59	1.55
1.6, 30, 0.5	100	26.13	TV	22.56	1.0	1.57	1.46	1.59	1.50
1.6, 30, 0.5	100	26.13	DnCNN	30.64	0.5	1.57	1.52	1.59	**1.58**
1.6, 30, 0.5	100	26.13	Wavelet	28.67	1.0	1.57	1.51	1.59	1.55

peaks, and outputs the left peak position as the left edge and the right peak position as the right edge. The results in Table 3.2 show that SEMNet produces reliable LER results in images with high noise. Figure 3.13 depicts the linescan from a noisy image and the linescans obtained after denoising from different algorithms; the y-axis of Figure 3.13 represents a normalized intensity relative to the maximum pixel intensity.

Finally, Figures 3.10-12 offer additional results for the performance of SEMNet on three test images corrupted by varying amount of Poisson noise. Observe that in Figures 3.10(a) and 3.12(a) there is a difference between the true LWR used in the generation of rough lines with the Thorsos method and the estimated LWR from the original ARTIMAGEN image with no noise; this difference results from the corruption of edges with more power in the higher frequency range during the formation of the image and from the application of the Canny filter. Parts (a) and (b) of each figure show that the outcome of SEMNet offers similar LWR and MABSE performance to the

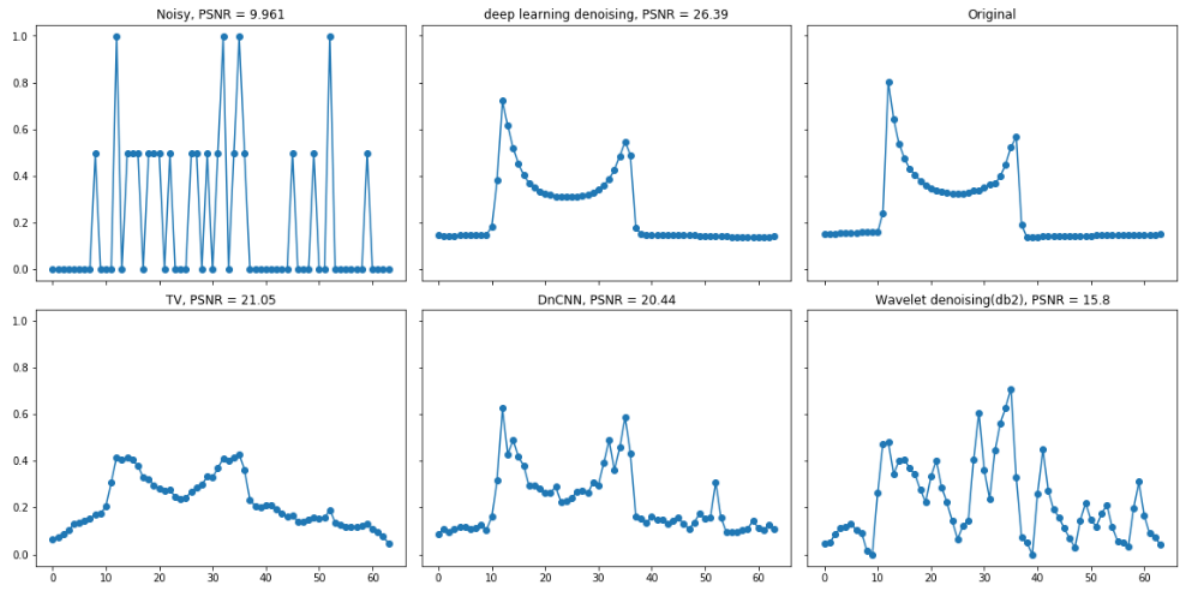


Figure 3.13: Linescan intensities; the y-axis represents a normalized intensity relative to the maximum pixel intensity. (a) Linescan intensity from a noisy image with a noise level of 2 electrons per pixel. (b) Estimated linescan intensity from SEMNet denoiser. (c) Linescan intensity from original image without Poisson noise. (d) Linescan intensity from total variation (Bregman method) denoising. (e) Linescan intensity from DnCNN denoising. (f) Linescan intensity from Wavelet (db2) denoising. Reprinted with permission from [4].

Table 3.2: LER results with the intensity-based algorithm. The LER data is for the central 1022 out of 1024 pixels for each image. Reprinted with permission from [4].

Original Image σ(nm), ξ(nm), α	Poisson noise level	PSNR noisy (dB)	Denoiser method	PSNR denoised (dB)	Left edge (nm) σ (true)	Left edge (nm) σ (obs.)	Right edge (nm) σ (true)	Right edge (nm) σ (obs.)
0.8, 10, 0.3	None	-	None	-	0.74	0.75	0.74	0.76
0.8, 10, 0.3	10	16.38	SEMNet	**28.66**	0.74	**0.74**	0.74	**0.74**
0.8, 10, 0.3	10	16.38	TV	22.39	0.74	0.70	0.74	1.37
0.8, 10, 0.3	10	16.38	DnCNN	25.24	0.74	0.74	0.74	1.21
0.8, 10, 0.3	10	16.38	Wavelet	20.91	0.74	1.10	0.74	1.74
1.2, 40, 0.7	None	-	None	-	1.07	1.06	1.21	1.21
1.2, 40, 0.7	2	9.96	SEMNet	**26.39**	1.07	**1.03**	1.21	**1.24**
1.2, 40, 0.7	2	9.96	TV	21.05	1.07	1.90	1.21	2.97
1.2, 40, 0.7	2	9.96	DnCNN	20.44	1.07	2.40	1.21	3.33
1.2, 40, 0.7	2	9.96	Wavelet	15.80	1.07	3.10	1.21	3.67
1.6, 30, 0.5	None	-	None	-	1.57	1.56	1.59	1.57
1.6, 30, 0.5	100	26.13	SEMNet	**42.64**	1.57	1.55	1.59	**1.56**
1.6, 30, 0.5	100	26.13	TV	22.56	1.57	1.72	1.59	2.11
1.6, 30, 0.5	100	26.13	DnCNN	30.64	1.57	1.55	1.59	1.56
1.6, 30, 0.5	100	26.13	Wavelet	28.67	1.57	**1.58**	1.59	1.66
0.8, 30, 0.6	None	-	None	-	0.76	0.77	0.68	0.70
0.8, 30, 0.6	5	13.22	SEMNet	**29.04**	0.76	**0.76**	0.68	**0.75**
0.8, 30, 0.6	5	13.22	TV	22.48	0.76	0.95	0.68	1.6
0.8, 30, 0.6	5	13.22	DnCNN	24.46	0.76	0.91	0.68	1.61
0.8, 30, 0.6	5	13.22	Wavelet	18.65	0.76	2.03	0.68	2.71

original image (with no Poisson noise) over a large range of noise levels. Part (c) of each of these figures shows that the output of SEMNet consistently has a significantly higher PSNR than the corresponding noisy input image over all noise levels.

3.3.2 EDGENet Results

EDGENet was trained for four epochs and the training time was approximately 45 hours and 30 minutes. We chose mean absolute error (MAE) as the loss function because it is better than MSE in penalizing all edge position errors as opposed to large edge position errors. Figure 3.14 plots the mean absolute error training loss as a function of the number of gradient descent steps in the Adam optimizer algorithm. The loss tends to decrease as the number of training steps increases. Figure 3.14 also shows the loss on the validation set after each epoch of training. The validation loss remains slightly lower than the training loss as the number of epochs increase, and therefore overfitting is unlikely. Further analysis of hyperparameters like the learning rate, the batch size and the number of epochs can potentially improve the training process. We saved the weights associated with the EDGENet model in a file of size 125 MB. This model file was later used to predict the line edge positions from the noisy SEM images in the test set. The average prediction time per image of EDGENet without including the time to load the model was approximately 2.4 seconds on a central processing unit.

We consider a few performance metrics to assess the quality of EDGENet on the test image dataset. Table 3.3 specifies three images and four Poisson noise levels and examines how these parameters affect the estimation of LER, LWR, and MABSE. Observe that even in the high noise regime the estimates of LER and LWR from the simulated test SEM images are very close to the true LER and LWR of the corresponding lines from the edges generated through the Thorsos method. The mean absolute pixel error is close to zero in the low noise regime.

Figures 3.15-3.17 show additional results for three of the test images considered in Table 3.3. The edge images have been constructed from the edge positions predicted by EDGENet. We estimate the power spectrum from $N = 1024$ point edge positions using multitaper [87, 88] spectrum estimation with six Slepian sequences, bandwidth W satisfying $NW = 4$, and adaptive weights.

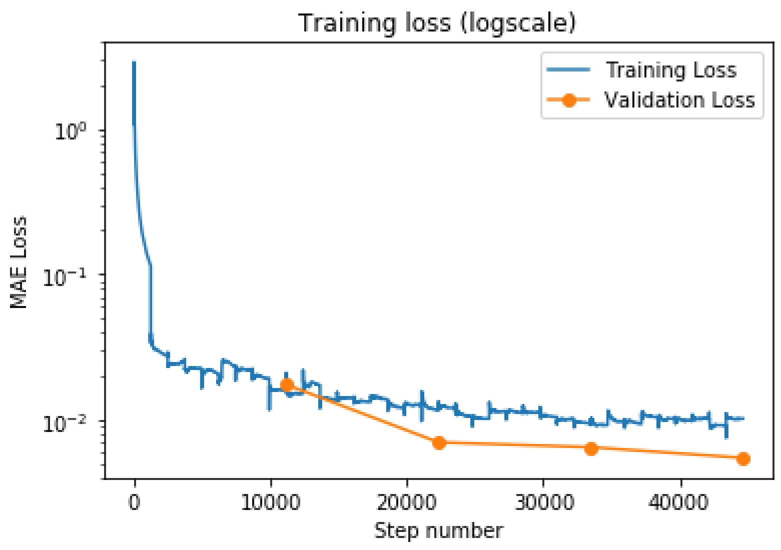


Figure 3.14: Training and validation loss for EDGENet. Reprinted with permission from [4].

The resulting power spectral density (PSD) estimates for the edges generated from the Thorsos method and the corresponding predicted edges from the simulated test SEM images have been plotted together with the underlying Palasantzas power spectral density. Observe that for all three test images the predicted edge's PSD estimate closely matches the original edge's PSD estimate. For the first two test images the deviations from the Palasantzas PSD arise because we consider a single predicted edge per plot as opposed to the average of the multitaper estimates from multiple edges. The large difference between the Palasantzas PSD and the original and predicted edge PSD estimates in the low frequency regions of Figures 3.17(e) and 3.17(f) arise because the edge roughness parameter (0.8 nm) is close to the pixel width (0.5 nm). We separately plot the left edge and the right edge spectra to assess the difference in the individual spectra of the true edge and the predicted edge. Observe that an estimate of the underlying Palasantzas PSD typically involves the average of multiple spectra per roughness condition.

Cizmar et al. [54] suggested that SEM image noise can be well approximated by a combination of Gaussian noise and Poisson noise. In Tables 3.4 and 3.5 we respectively present the results for SEMNet and EDGENet when some Gaussian noise is added to the existing Poisson noise test

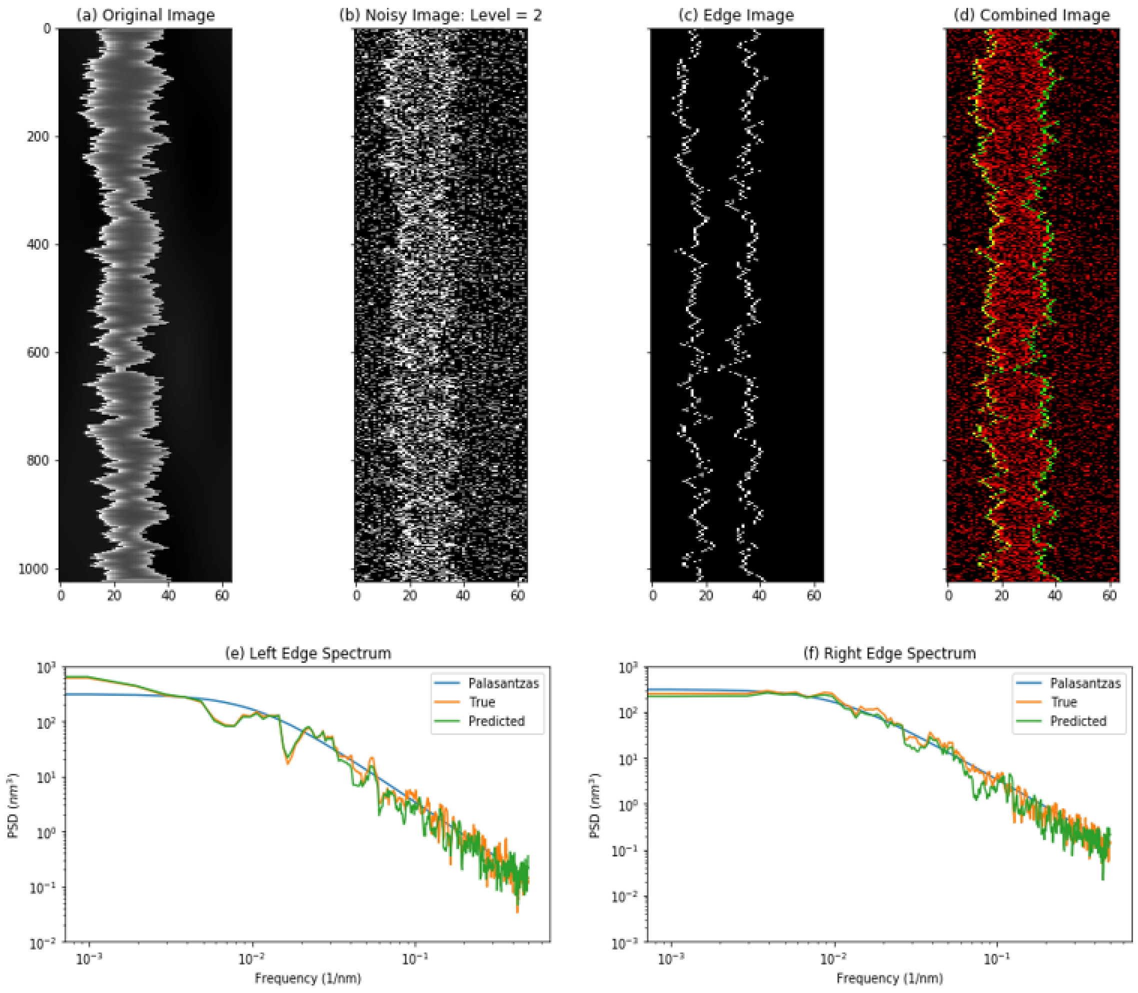


Figure 3.15: (a) Original image with $\sigma = 1.6$ nm, $\alpha = 0.5$, $\xi = 30$ nm. (b) Noisy image with a noise level of 2 electrons per pixels. (c) Predicted edge image. (d) Noisy image overlayed with edge image. (e) Left edge PSD; Palasantzas (Blue), true edge (Orange), predicted edge (Green). (f) Right edge PSD; Palasantzas (Blue), true edge (Orange), predicted edge (Green). Reprinted with permission from [4].

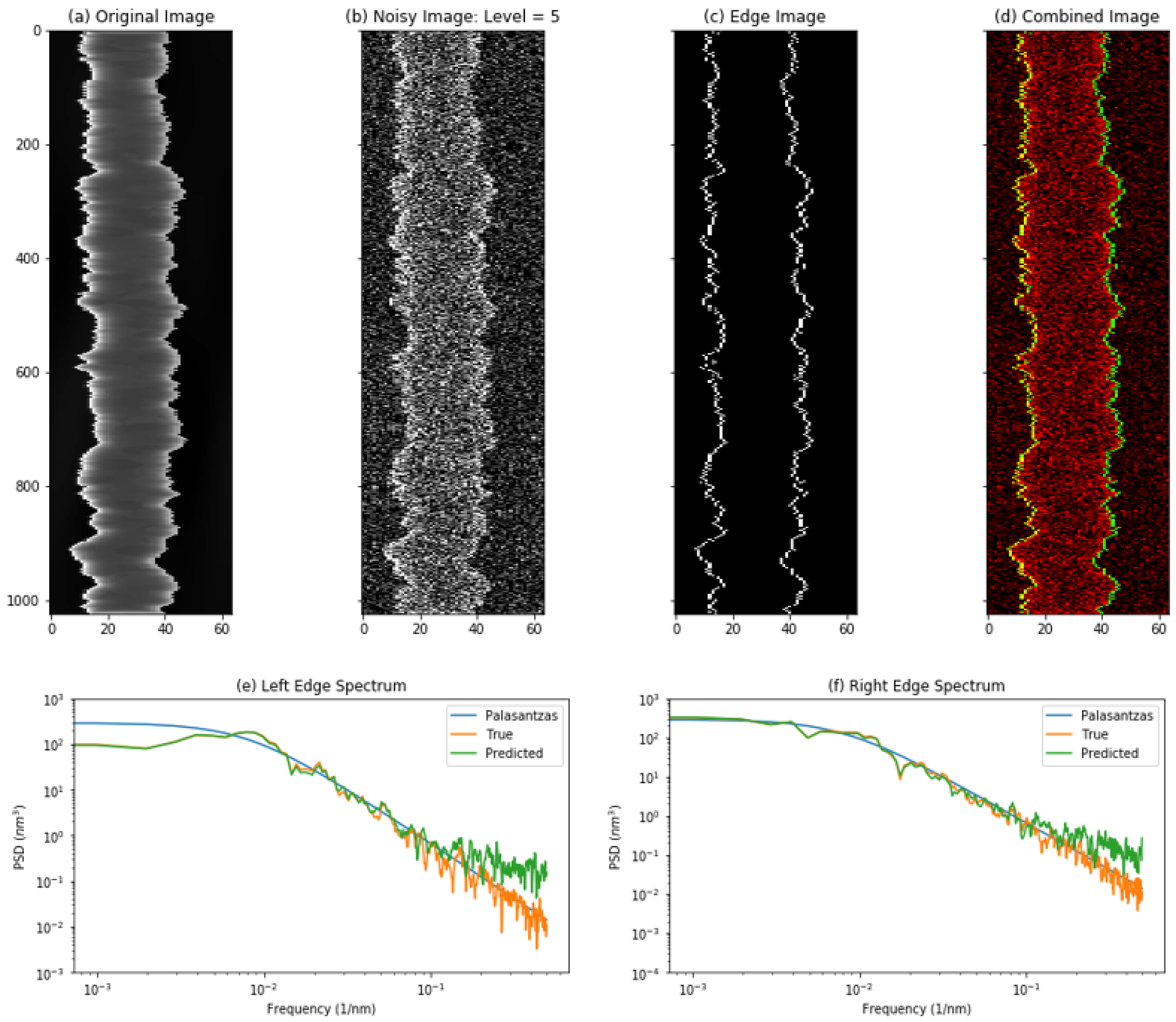


Figure 3.16: (a) Original image with $\sigma = 1.2$ nm, $\alpha = 0.7$, $\xi = 40$ nm. (b) Noisy image with a noise level of 5 electrons per pixels. (c) Predicted edge image. (d) Noisy image overlayed with edge image. (e) Left edge PSD; Palasantzas (Blue), true edge (Orange), predicted edge (Green). (f) Right edge PSD; Palasantzas (Blue), true edge (Orange), predicted edge (Green). Reprinted with permission from [4].

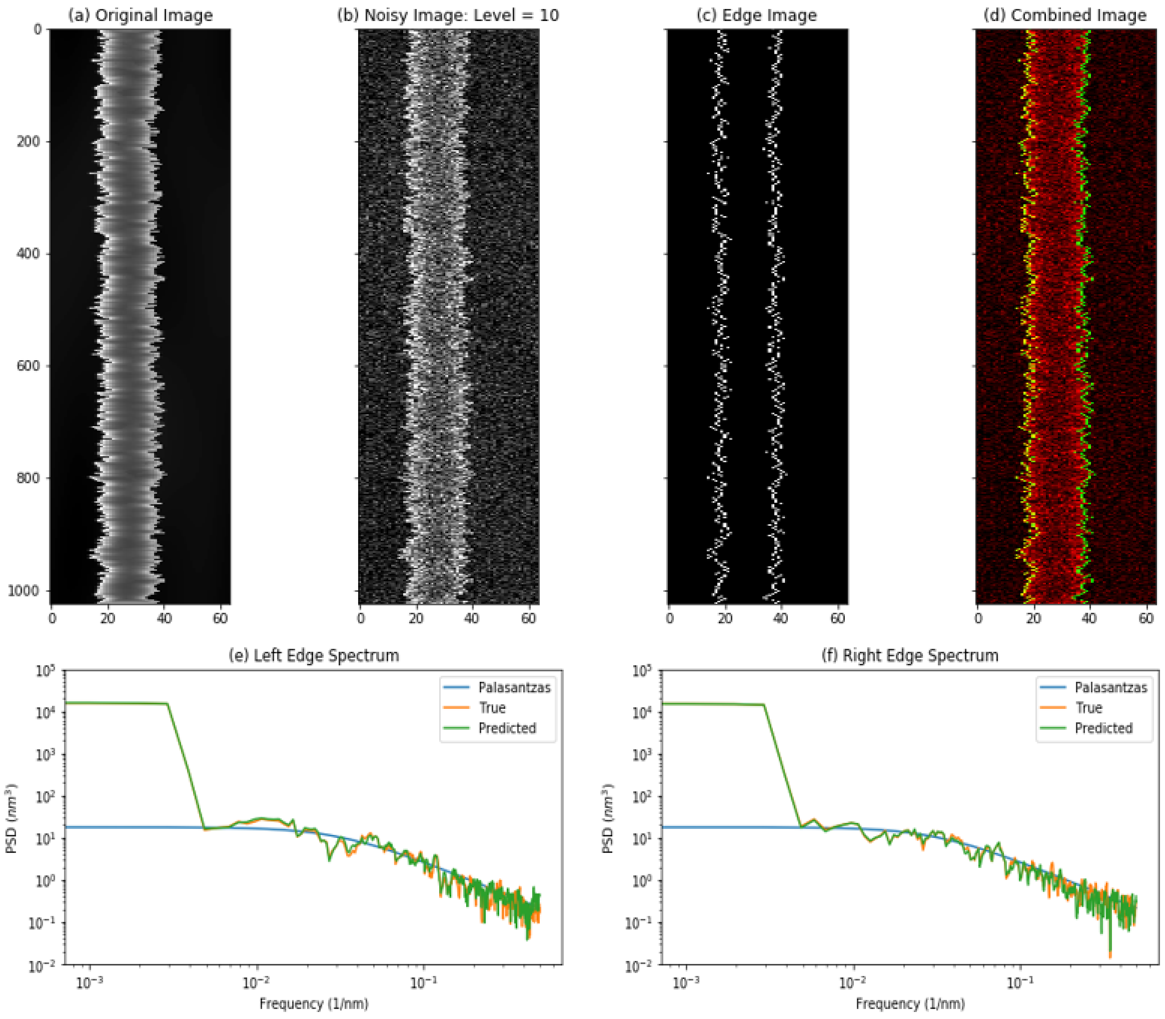


Figure 3.17: (a) Original image with $\sigma = 0.8$ nm, $\alpha = 0.3$, $\xi = 10$ nm. (b) Noisy image with a noise level of 10 electrons per pixels. (c) Predicted edge image. (d) Noisy image overlayed with edge image. (e) Left edge PSD; Palasantzas (Blue), true edge (Orange), predicted edge (Green). (f) Right edge PSD; Palasantzas (Blue), true edge (Orange), predicted edge (Green). Reprinted with permission from [4].

Table 3.3: Edge results for 1024 edge positions. Reprinted with permission from [4].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Mean absolute error (pixels)	Left edge (nm)		Right edge (nm)		LWR (nm)	
			σ (true)	σ (obs.)	σ (true)	σ (obs.)	(true)	(obs.)
0.8, 10, 0.3	2	0.61	0.75	0.71	0.74	0.66	1.06	0.95
0.8, 10, 0.3	5	0.40	0.75	0.73	0.74	0.67	1.06	0.97
0.8, 10, 0.3	10	0.22	0.75	0.75	0.74	0.72	1.06	1.04
0.8, 10, 0.3	100	0.01	0.75	0.75	0.74	0.74	1.06	1.06
1.2, 40, 0.7	2	0.44	1.06	1.06	1.21	1.23	1.41	1.42
1.2, 40, 0.7	5	0.27	1.06	1.06	1.21	1.22	1.41	1.42
1.2, 40, 0.7	10	0.15	1.06	1.08	1.21	1.21	1.41	1.42
1.2, 40, 0.7	100	0.01	1.06	1.06	1.21	1.21	1.41	1.42
1.6, 30, 0.5	2	0.69	1.56	1.53	1.59	1.46	2.34	2.23
1.6, 30, 0.5	5	0.38	1.56	1.57	1.59	1.52	2.34	2.28
1.6, 30, 0.5	10	0.23	1.56	1.57	1.59	1.59	2.34	2.33
1.6, 30, 0.5	100	0.01	1.56	1.56	1.59	1.58	2.34	2.33

images. We consider three Gaussian noise levels with $\sigma = 0.02, 0.05, 0.10$ assuming that the pixels of an image can have peak intensity of 1.

3.3.3 LineNet1 and LineNet2 Results [†]

The training processes for both LineNet1 and LineNet2 involved solving an optimization problem to minimize the mean squared error (MSE) between the predicted images and the original images. The results presented here use test dataset images which were not utilized during the training process. Figure 3.18 demonstrates the effectiveness of simultaneously denoising and estimating edge images in single-line images using LineNet1.

Figures 3.19 and 3.20 demonstrate the effectiveness of simultaneously denoising and estimating edge images in multiple-line images using LineNet2. The edge images predicted from LineNet1 and LineNet2 are grayscale images with pixel intensity between 0 and 1 instead of bi-

[†]Part of the data reported in this subsection is reprinted with permission from N. Chaudhary and S. A. Savari, "Simultaneous denoising and edge estimation from SEM images using deep convolutional neural networks," in *Proceedings of 30th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pp. 431–436, 2019. DOI: 10.1109/ASMC.2019.8791764. Copyright [2019] by IEEE.

Table 3.4: LER results for SEMNet with Poisson-Gaussian noise and the intensity based algorithm. The LER data is for the central 1022 out of 1024 pixels for each image. Gaussian σ is standard deviation for image with peak intensity of 1. Reprinted with permission from [4].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Gaussian noise σ	PSNR noisy (dB)	PSNR denoised (dB)	Left edge (nm) σ (true)	σ (obs.)	Right edge (nm) σ (true)	σ (obs.)
0.8, 10, 0.3	10	0.02	16.31	28.63	0.74	0.73	0.74	0.75
0.8, 10, 0.3	10	0.05	15.94	28.38	0.74	0.73	0.74	0.75
0.8, 10, 0.3	10	0.10	14.82	27.6	0.74	0.71	0.74	0.73
1.2, 40, 0.7	2	0.02	9.94	26.35	1.07	1.03	1.21	1.25
1.2, 40, 0.7	2	0.05	9.85	26.28	1.07	1.03	1.21	1.25
1.2, 40, 0.7	2	0.10	9.53	26.14	1.07	1.03	1.21	1.25
1.6, 30, 0.5	100	0.02	25.47	42.55	1.57	1.55	1.59	1.60
1.6, 30, 0.5	100	0.05	23.05	40.42	1.57	1.55	1.59	1.60
1.6, 30, 0.5	100	0.10	19.02	34.26	1.57	1.56	1.59	1.61
0.8, 30, 0.6	5	0.02	13.18	29.08	0.76	0.76	0.68	0.75
0.8, 30, 0.6	5	0.05	13.0	29.03	0.76	0.76	0.68	0.75
0.8, 30, 0.6	5	0.10	12.38	28.73	0.76	0.76	0.68	0.74

Table 3.5: Edge results for 1024 edge positions with Poisson-Gaussian noise. Gaussian σ is standard deviation for image with peak intensity of 1. Reprinted with permission from [4].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Gaussian noise σ	Mean absolute error (pixels)	Left edge (nm) σ (true)	σ (obs.)	Right edge (nm) σ (true)	σ (obs.)	LWR (nm) (true)	(obs.)
0.8, 10, 0.3	10	0.02	0.22	0.75	0.75	0.74	0.72	1.06	1.04
0.8, 10, 0.3	10	0.05	0.23	0.75	0.75	0.74	0.72	1.06	1.03
0.8, 10, 0.3	10	0.10	0.28	0.75	0.73	0.74	0.71	1.06	1.01
1.2, 40, 0.7	2	0.02	0.45	1.06	1.06	1.21	1.23	1.41	1.43
1.2, 40, 0.7	2	0.05	0.46	1.06	1.06	1.21	1.23	1.41	1.44
1.2, 40, 0.7	2	0.10	0.46	1.06	1.07	1.21	1.23	1.41	1.44
1.6, 30, 0.5	100	0.02	0.01	1.56	1.56	1.59	1.58	2.34	2.33
1.6, 30, 0.5	100	0.05	0.01	1.56	1.56	1.59	1.58	2.34	2.33
1.6, 30, 0.5	100	0.10	0.07	1.56	1.56	1.59	1.58	2.34	2.32
0.8, 30, 0.6	5	0.02	0.27	0.77	0.75	0.68	0.66	1.01	0.99
0.8, 30, 0.6	5	0.05	0.27	0.77	0.75	0.68	0.66	1.01	0.98
0.8, 30, 0.6	5	0.10	0.27	0.77	0.74	0.68	0.65	1.01	0.97

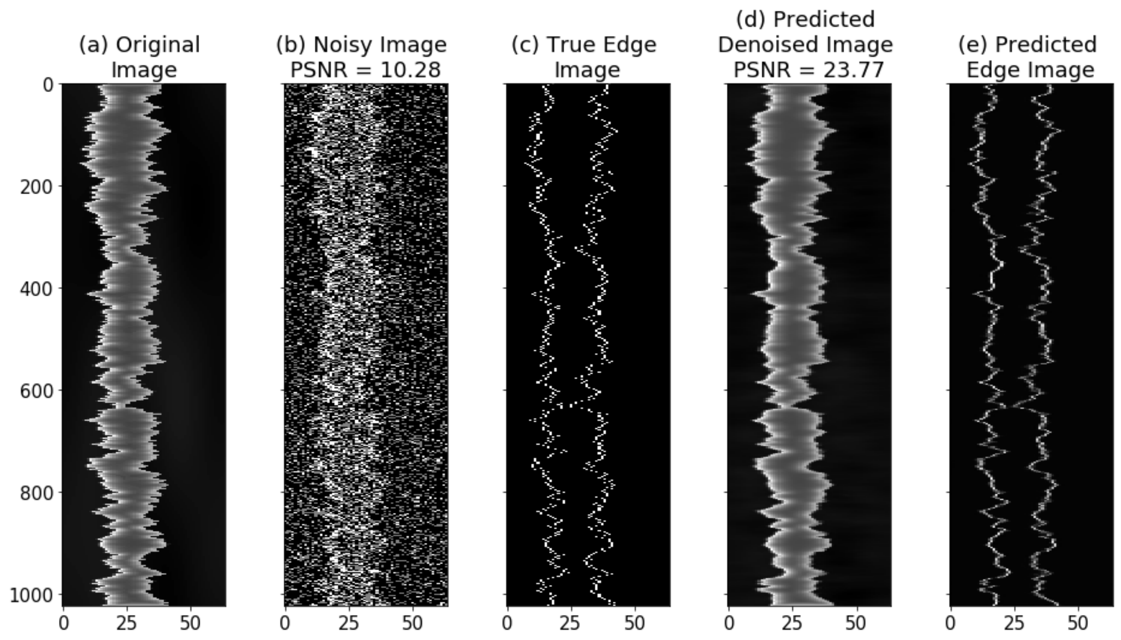


Figure 3.18: Results with LineNet1: (a) Original image with $\sigma = 1.6$ nm, $\alpha = 0.5$, $\xi = 30$ nm. (b) Noisy image with a noise level of 2 electrons per pixels. (c) Edge image. (d) Predicted denoised image. (e) Predicted edge image. Reprinted with permission from [5].

nary images. The grayscale values in a row of an edge image have approximate peaks. Figure 3.21 shows the pixel intensity values in a row of a grayscale edge image with four lines / eight edges. We apply the scientific python (SciPy) peak detector function find_peaks to estimate the exact edge positions in the predicted edge images.

Table 3.6 provides the peak signal-to-noise ratio (PSNR) results of LineNet1 and SEMNet [67]. Observe that the PSNR values are close even though LineNet1 is devoting some resources to edge detection. Table 3.7 provides the line roughness results of LineNet1 and EDGENet [68] on single-line images. EDGENet offers better accuracy than LineNet1 in the case of both high noise and high frequency at the price of significantly higher memory; recall that LineNet1 has only 559,810 parameters compared to the 10,972,993 parameters of EDGENet. LineNet1 and EDGENet have similar performance in other cases. Figure 3.22 compares the predicted edge spectra of LineNet1 and EDGENet for three different types of edges. Observe that the edge spectra do not significantly differ. Table 3.8 provides PSNR and line roughness results for multiple-line images of dimension

Table 3.6: Denoising results for single-line images with LineNet1 and SEMNet. Reprinted with permission from [5].

Original Image σ(nm), ξ(nm), α	Poisson noise level	PSNR (dB)		
		Noisy Image	Predicted LineNet1	Predicted SEMNet
0.8, 10, 0.3	2	10.35	23.89	24.27
0.8, 10, 0.3	5	13.64	25.58	26.02
0.8, 10, 0.3	10	16.38	28.26	28.66
0.8, 10, 0.3	100	26.18	41.87	42.89
1.2, 40, 0.7	2	9.96	26.39	26.39
1.2, 40, 0.7	5	13.20	29.07	29.0
1.2, 40, 0.7	10	15.96	31.22	31.26
1.2, 40, 0.7	100	25.71	42.07	42.79
1.6, 30, 0.5	2	10.28	23.77	24.09
1.6, 30, 0.5	5	13.67	26.33	26.65
1.6, 30, 0.5	10	16.33	28.89	29.14
1.6, 30, 0.5	100	26.13	41.47	42.64

256×1024 pixels. The line edge roughness data in Table 3.8 come from the average of multiple line edges in a single image. In summary, our results show that LineNet1's performance for image denoising and line edge roughness (LER) estimation are comparable to those of SEMNet [67] and EDGENet [68] on the same set of images. The average prediction time per image of LineNet1 and LineNet2 was 0.12 seconds on the K-80 GPU. In comparison, the prediction time per image of EDGENet was 0.15 seconds and that of SEMNet was 0.12 seconds on the K-80 GPU; recall, however that LineNet1 and LineNet2 are followed by a peak detection algorithm while EDGENet is a stand-alone line edge detection algorithm. Additionally, LineNet1 takes approximately 41 hours to train while SEMNet and EDGENet each take more than 40 hours to train.

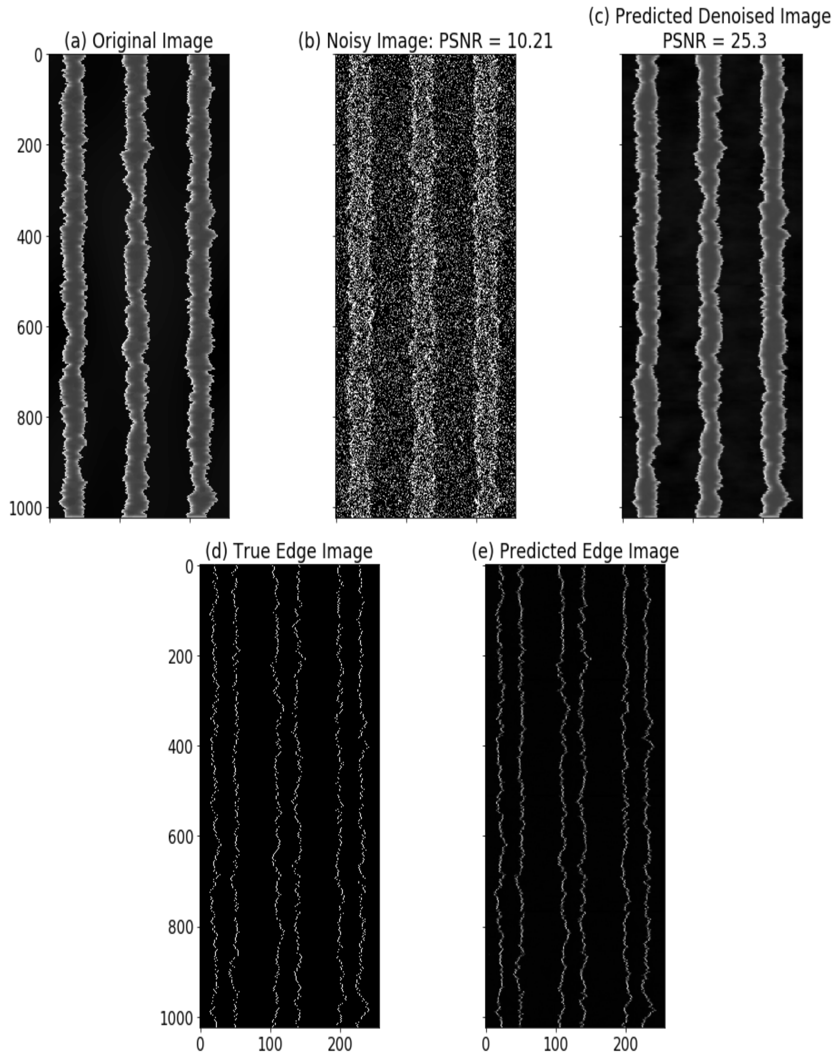


Figure 3.19: Results with LineNet2: (a) Original image with $\sigma = 1.6$ nm, $\alpha = 0.5$, $\xi = 30$ nm, linewidth = 15 nm. (b) Noisy image with a noise level of 2 electrons per pixels. (c) Predicted denoised image. (d) Edge image. (e) Predicted edge image. Reprinted with permission from [5].

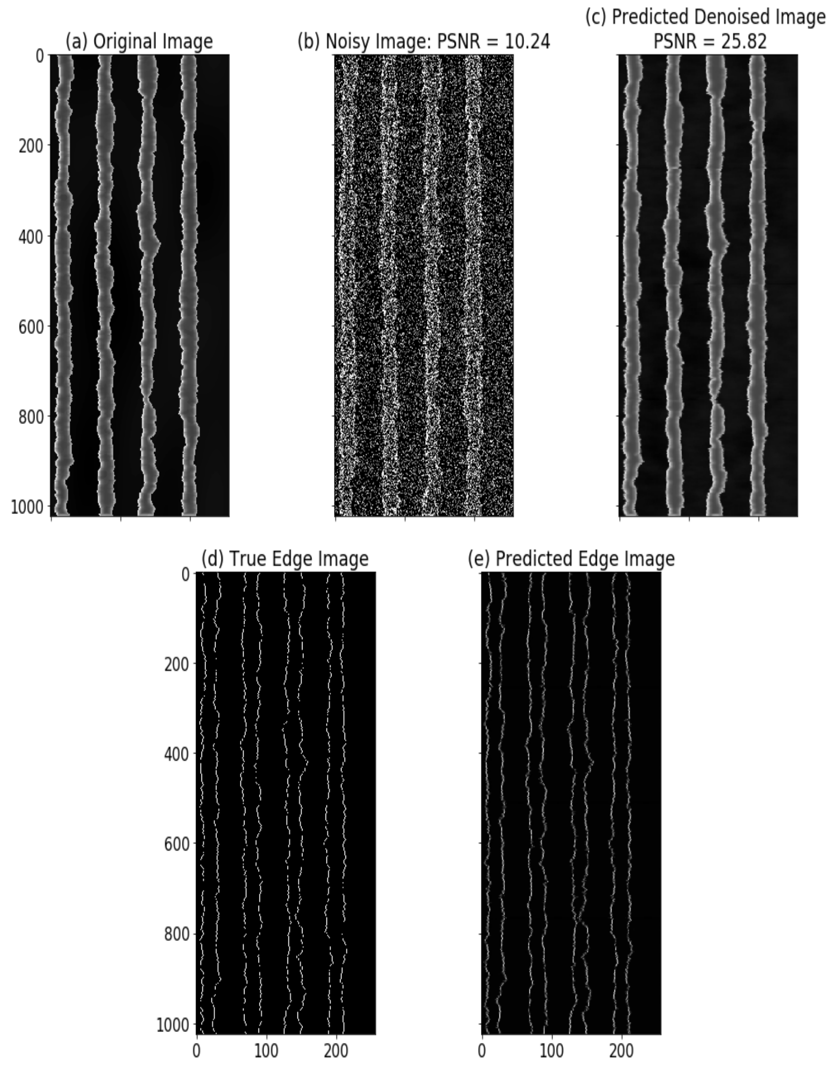


Figure 3.20: Results with LineNet2: (a) Original image with $\sigma = 1.2$ nm, $\alpha = 0.8$, $\xi = 40$ nm, linewidth = 10 nm. (b) Noisy image with a noise level of 2 electrons per pixels. (c) Predicted denoised image. (d) Edge image. (e) Predicted edge image. Reprinted with permission from [5].

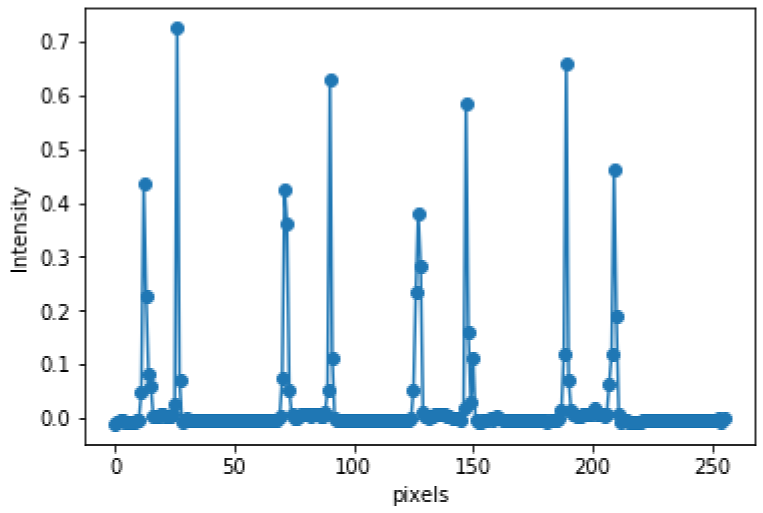


Figure 3.21: The pixel intensity values in one row of a predicted grayscale edge image. The original image has four lines / eight edges. Reprinted with permission from [5].

Table 3.7: Roughness results for single-line images with LineNet1 and EDGENet. Reprinted with permission from [5].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Left edge (nm) σ	Left edge (nm) σ (observed) LineNet1 | EDGENet	Right edge (nm) σ	Right edge (nm) σ (observed) LineNet1 | EDGENet	LWR (nm) σ	LWR (nm) σ (observed) LineNet1 | EDGENet
0.8, 10, 0.3	2	0.75	0.64 | 0.71	0.74	0.66 | 0.66	1.06	0.89 | 0.95
0.8, 10, 0.3	5	0.75	0.71 | 0.73	0.74	0.65 | 0.67	1.06	0.96 | 0.97
0.8, 10, 0.3	10	0.75	0.73 | 0.75	0.74	0.72 | 0.72	1.06	1.04 | 1.04
0.8, 10, 0.3	100	0.75	0.75 | 0.75	0.74	0.74 | 0.74	1.06	1.06 | 1.06
1.2, 40, 0.7	2	1.06	1.03 | 1.06	1.21	1.26 | 1.23	1.41	1.42 | 1.42
1.2, 40, 0.7	5	1.06	1.06 | 1.06	1.21	1.20 | 1.22	1.41	1.40 | 1.42
1.2, 40, 0.7	10	1.06	1.06 | 1.08	1.21	1.20 | 1.21	1.41	1.41 | 1.42
1.2, 40, 0.7	100	1.06	1.06 | 1.06	1.21	1.21 | 1.21	1.41	1.41 | 1.42
1.6, 30, 0.5	2	1.56	1.50 | 1.53	1.59	1.49 | 1.46	2.34	2.27 | 2.23
1.6, 30, 0.5	5	1.56	1.55 | 1.57	1.59	1.54 | 1.52	2.34	2.30 | 2.28
1.6, 30, 0.5	10	1.56	1.54 | 1.57	1.59	1.61 | 1.59	2.34	2.34 | 2.33
1.6, 30, 0.5	100	1.56	1.56 | 1.56	1.59	1.60 | 1.58	2.34	2.34 | 2.33

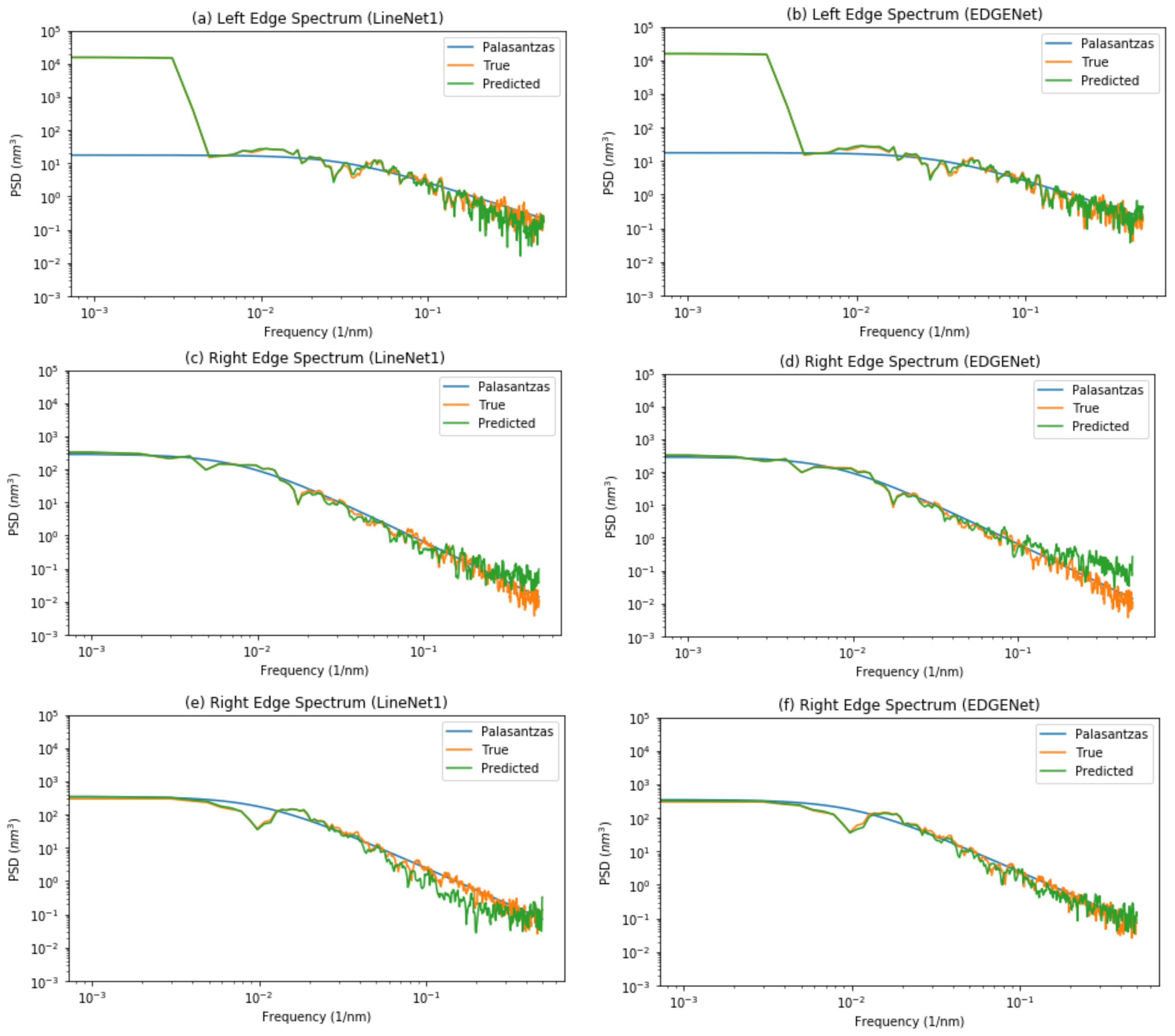


Figure 3.22: (a), (b) LineNet1 and EDGENet predicted edge spectra for $\sigma = 0.8$ nm, $\alpha = 0.3$, $\xi = 10$ nm, Poisson noise = 10 electrons per pixel. (c), (d) LineNet1 and EDGENet predicted edge spectra for $\sigma = 1.2$ nm, $\alpha = 0.7$, $\xi = 40$ nm, Poisson noise = 5 electrons per pixel. (e), (f) LineNet1 and EDGENet predicted edge spectra for $\sigma = 1.6$ nm, $\alpha = 0.6$, $\xi = 30$ nm, Poisson noise = 2 electrons per pixel. Reprinted with permission from [5].

3.3.4 Hyperparameter Search and Optimization

In this section, we conduct a hyperparameter search for deep CNNs to assess if the networks are fully optimized. Some memory and computational constraints at the university supercomputing facility restricted our hyperparameter search. For instance, the Tesla K80 GPU has approximately

Table 3.8: Results for multiple-line images with LineNet2. Reprinted with permission from [5].

Original Image σ(nm), ξ(nm), α, linewidth(nm)	Poisson noise level	PSNR (dB) Noisy Image	PSNR (dB) Predicted Image	Mean LER (nm) σ (true)	Mean LER (nm) σ (obs.)
0.8, 10, 0.3, 10	2	10.18	24.23	0.76	0.75
0.8, 10, 0.3, 10	5	13.48	26.52	0.76	0.75
0.8, 10, 0.3, 10	10	16.22	28.90	0.76	0.76
0.8, 10, 0.3, 10	100	26.03	42.33	0.76	0.76
1.2, 40, 0.8, 10	2	10.24	25.82	1.16	1.18
1.2, 40, 0.8, 10	5	13.54	28.71	1.16	1.17
1.2, 40, 0.8, 10	10	16.33	31.22	1.16	1.17
1.2, 40, 0.8, 10	100	26.11	42.50	1.16	1.16
1.6, 30, 0.5, 15	2	10.21	25.30	1.52	1.48
1.6, 30, 0.5, 15	5	13.58	27.74	1.52	1.52
1.6, 30, 0.5, 15	10	16.35	30.22	1.52	1.52
1.6, 30, 0.5, 15	100	26.17	42.60	1.52	1.52

12 Gigabytes of available GPU memory. Due to this, we needed to limit ourselves to a maximum batch size of eight with full-sized images. Additionally, one epoch of training on our dataset took nearly 10-11 hours for SEMNet, EDGENet, LineNet1, and DnCNN. All training experiments used the Adam [85] optimizer algorithm.

For our SEMNet network, we explore learning rates from the set {1e-3, 1e-5, 1e-7}, and batch sizes from the set {4, 8}. The batch size of two and one can be very small when images can have ten different Poisson noise levels. Figure 3.23 shows the mean squared error (MSE) training loss and the validation loss with different hyperparameters. We observe from the plots that the validation loss obtained with different hyperparameters stabilizes around approximately the same value. It gives us confidence in the optimality of the solution. The plots have a common y-axis, and the loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps. In our experiments, a learning rate of 1e-3 and a batch size of eight achieve the lowest validation loss for SEMNet.

With our EDGENet network, we explore learning rates from the set {1e-3, 1e-5, 1e-7} and use a batch size of eight. Figure 3.24 shows the mean absolute error (MAE) training loss and

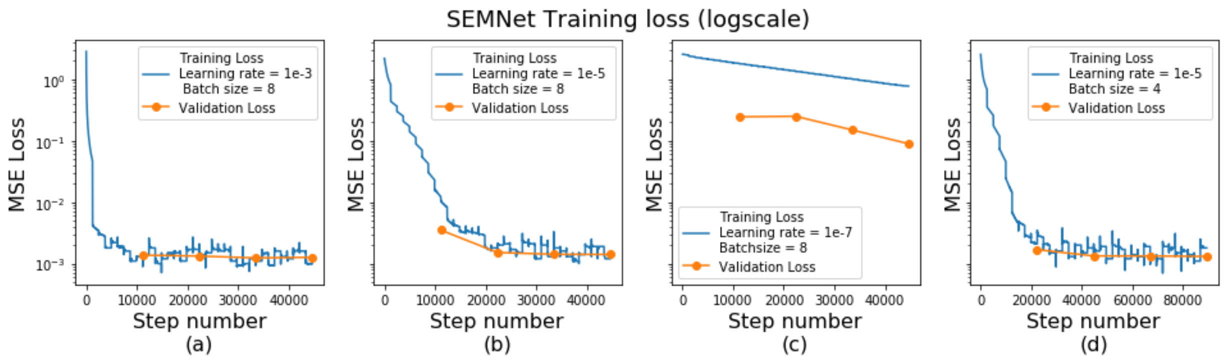


Figure 3.23: SEMNet hyperparameter search. Mean squared error (MSE) training and validation loss. The plots have a common y-axis, and loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps. (a) Learning rate = 1e-3, batch size = 8, (b) Learning rate = 1e-5, batch size = 8, (c) Learning rate = 1e-7, batch size = 8, (d) Learning rate = 1e-5, batch size = 4.

the validation loss with different hyperparameters. Whenever necessary, we increase the number of epochs to reach the lowest validation loss. In our experiments, a learning rate of 1e-3 and a batch size of eight achieve the lowest validation loss for EDGENet. Models trained with other hyperparameters have higher validation losses. The plots have a common y-axis, and loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps.

We select learning rates from the set {1e-3, 1e-5, 1e-7} and batch sizes from the set {4, 8} for the training of the LineNet1 network. Figure 3.25 shows the mean squared error (MSE) training loss and the validation loss with different hyperparameters. We again observe that the validation loss obtained with different hyperparameters stabilizes around approximately the same value, giving us confidence in the optimality of the solution. In our experiments, a learning rate of 1e-3 and a batch size of eight achieve the lowest validation loss for LineNet1.

In the previous experiments of Subsection 3.3.1, we compared our SEMNet denoiser with a pre-trained DnCNN model. The pre-trained DnCNN model was optimized to tackle several general image denoising tasks on natural images [7], such as blind Gaussian denoising, single image super-resolution, and JPEG deblocking. To have a fair comparison with SEMNet, we retrain and optimize the DnCNN network with our training dataset. In reference [7], Zhang *et al.* trained

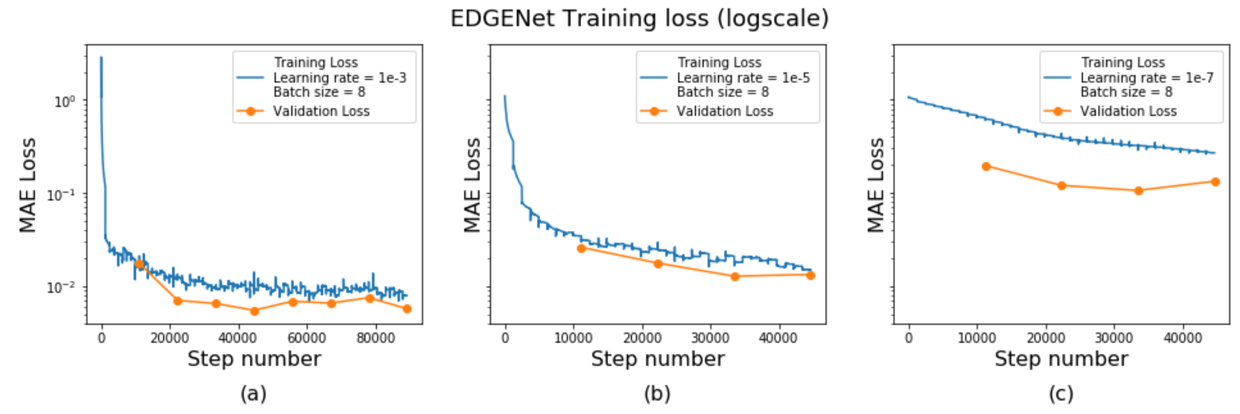


Figure 3.24: EDGENet hyperparameter search. Mean absolute error (MAE) training and validation loss. The plots have a common y-axis, and loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps. (a) Learning rate = 1e-3, batch size = 8, (b) Learning rate = 1e-5, batch size = 8, (c) Learning rate = 1e-7, batch size = 8.

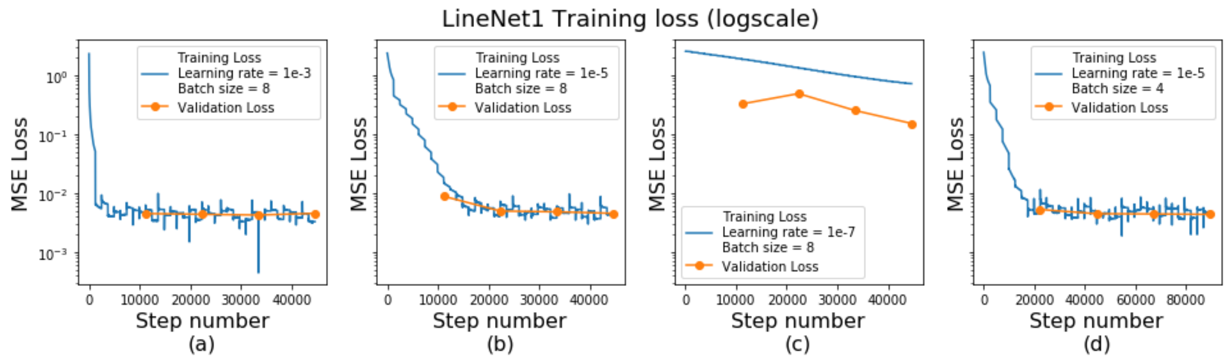


Figure 3.25: LineNet1 hyperparameter search. Mean squared error (MSE) training and validation loss. The plots have a common y-axis, and loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps. (a) Learning rate = 1e-3, batch size = 8, (b) Learning rate = 1e-5, batch size = 8, (c) Learning rate = 1e-7, batch size = 8, (d) Learning rate = 1e-5, batch size = 4.

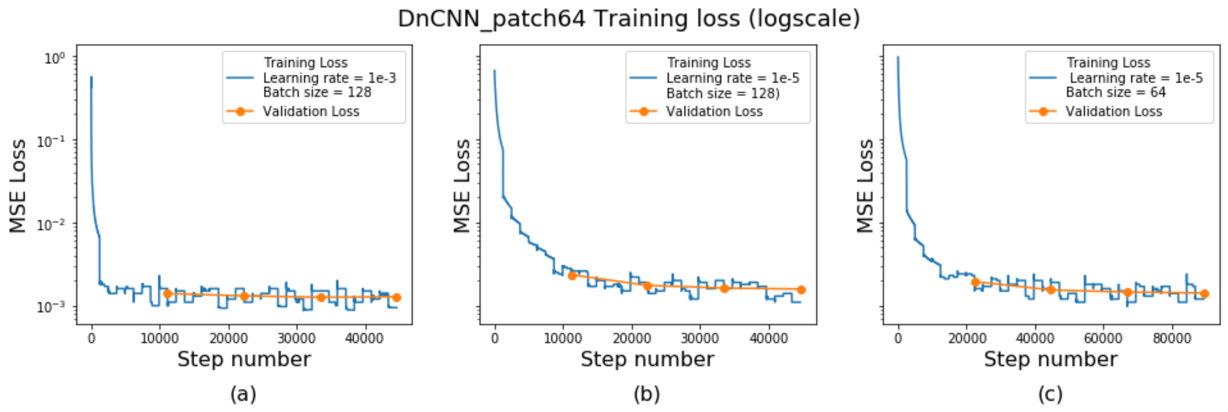


Figure 3.26: DnCNN with patch size of 64×64. Mean squared error (MSE) training and validation loss. The plots have a common y-axis, and loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps. (a) Learning rate = 1e-3, batch size = 128, (b) Learning rate = 1e-5, batch size = 128, (c) Learning rate = 1e-5, batch size = 64.

the DnCNN network with small image patches. Thus, in our new experiments, we train one version of DnCNN with a patch size of 64×64. We call this model DnCNN_patch64. Figure 3.26 shows the mean squared error (MSE) training loss and the validation loss with different hyperparameters. We select learning rates from the set {1e-3, 1e-5} and batch sizes from the set {64, 128} for the hyperparameter search. In our experiments, a learning rate of 1e-3 and a batch size of 128 achieve the lowest validation for DnCNN_patch64. The validation loss doesn't seem to improve with other hyperparameters.

To have an even more fair comparison, we also train a version of DnCNN on full-sized SEM images of dimension 64×1024. We call this model DnCNN_fullimage. Figure 3.27 shows the mean squared error (MSE) training loss and validation loss with different hyperparameters. We explore learning rates from the set {1e-3, 1e-5} and batch sizes from the set {4, 8} for the hyperparameter search. In our experiments, a learning rate of 1e-3 and a batch size of eight achieve the lowest validation loss for DnCNN_fullimage. The validation loss doesn't seem to improve with other hyperparameters.

To compare the optimized DnCNNs with SEMNet, we repeat the denoising experiments of Table 3.2 and Table 3.4. Table 3.9 shows these denoiser comparisons. We observe that for images

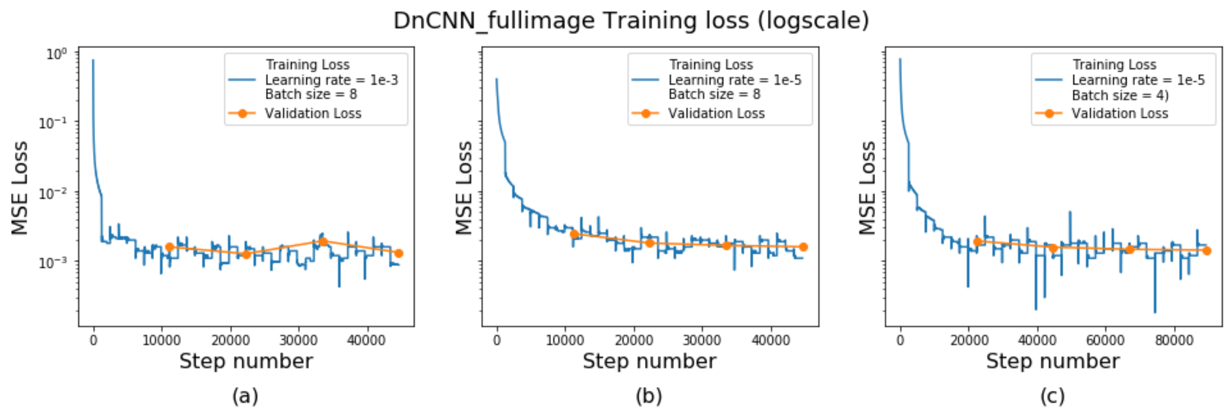


Figure 3.27: DnCNN with full sized image. Mean squared error (MSE) training and validation loss. The plots have a common y-axis, and loss is on a logarithmic scale. The x-axes represent the number of gradient descent steps. (a) Learning rate = 1e-3, batch size = 8, (b) Learning rate = 1e-5, batch size = 8, (c) Learning rate = 1e-5, batch size = 4.

with high dose or low amounts of Poisson noise, DnCNN_patch64 and DnCNN_fullimage provide slightly higher PSNR compared to SEMNet. However, for images with low dose or high amounts of Poisson noise, SEMNet gives slightly higher PSNR values. Figure 3.28 shows that the SEMNet generated image visually has somewhat better edge effect reconstruction, although the difference between images is minimal. The significant difference between SEMNet and DnCNN performance is observed when we change the noise characteristics by adding a small amount of Gaussian noise to the Poisson noise. The training data did not contain images with Poisson-Gaussian noise. SEMNet provides better PSNR results with Poisson-Gaussian noise. Figure 3.29 shows that DnCNN denoisers are unable to remove the Gaussian noise entirely. Figure 3.30 shows pixel intensity values in one row of denoised images in the presence of Poisson-Gaussian noise. Again, we observe that DnCNN denoisers seem unable to remove the Gaussian noise entirely. We also note that DnCNN_patch64 performs slightly worse compared to DnCNN_fullimage when noise characteristics change. SEMNet appears to be more robust to changes in noise characteristics. This effect could be due to the generalization provided by the dropout layers. Another reason could be that learning the noisy image to original signal mapping is more beneficial, compared to learning the noisy image to noise mapping with residual learning.

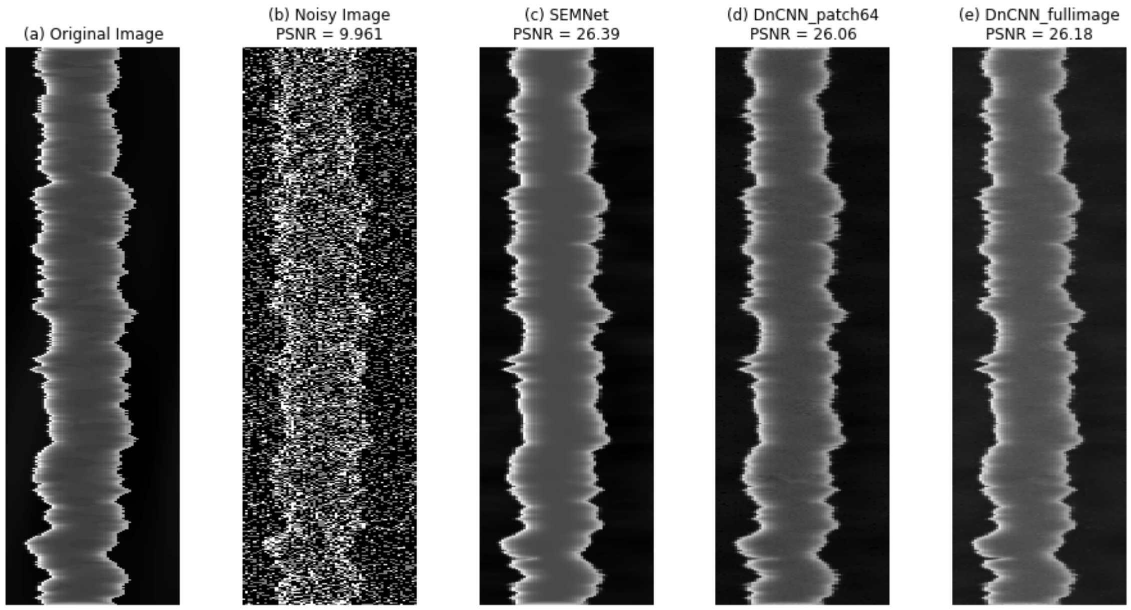


Figure 3.28: Comparison of SEMNet with DnCNN denoisers in the presence of only Poisson noise. (a) Original image without any noise, (b) Noisy image with a Poisson noise level of 2 electrons per pixel, (c) SEMNet denoising, (d) DnCNN denoising with a patch size of 64×64, (e) DnCNN with full image size.

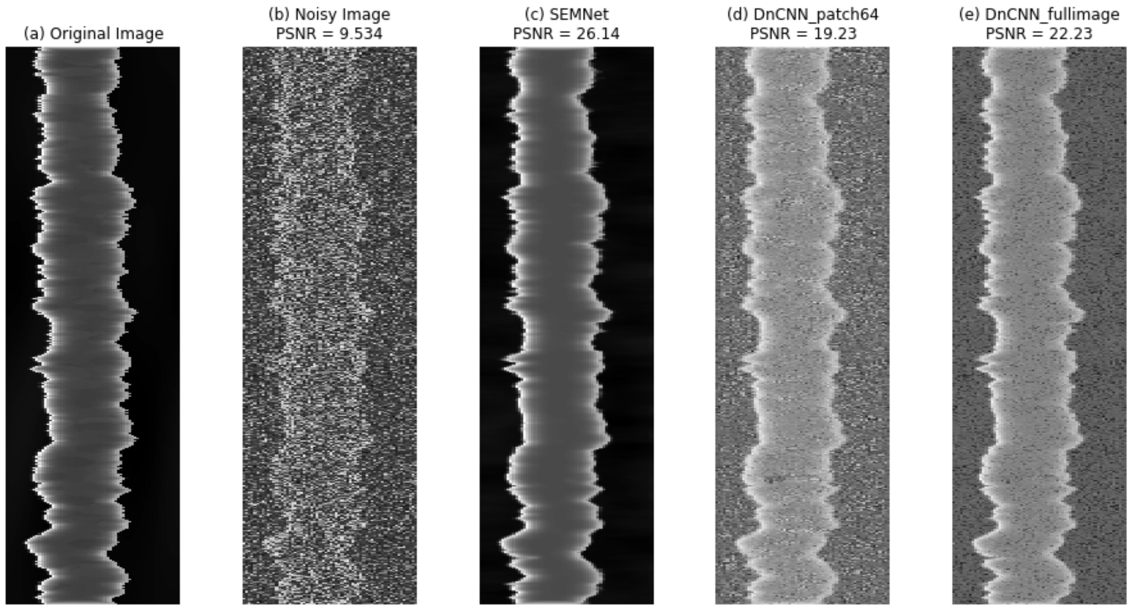


Figure 3.29: Comparison of SEMNet with DnCNN denoisers in the presence of Poisson-Gaussian noise. (a) Original image without any noise, (b) Noisy image with Poisson-Gaussian noise (Poisson = 2 electrons per pixel, Gaussian noise $\sigma = 0.1$ assuming peak intensity of 1), (c) SEMNet denoising, (d) DnCNN denoising with patch size of 64×64, (e) DnCNN with full image size.

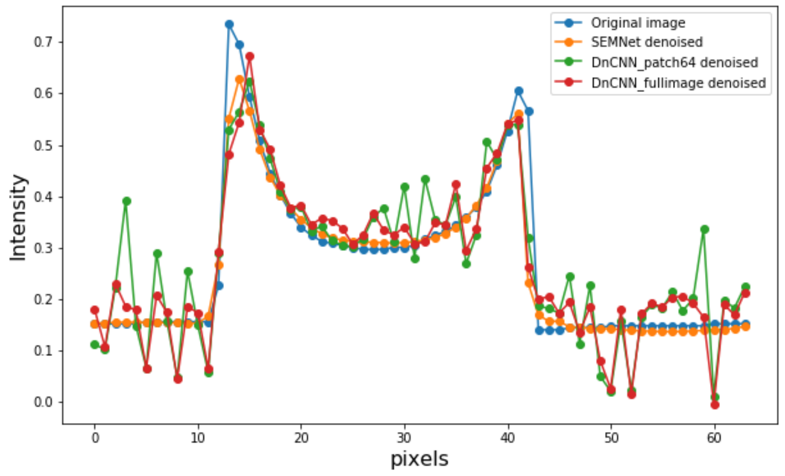


Figure 3.30: The pixel intensity values in one row of denoised images in the presence of Poisson-Gaussian noise. Noisy image with Poisson-Gaussian noise (Poisson = 2 electrons per pixel, Gaussian noise $\sigma = 0.1$ assuming peak intensity of 1. The DnCNN denoisers do not seem to be able to remove Gaussian noise completely.

Table 3.9: Denoising results for SEMNet, DnCNN_patch64, DnCNN_fullimage. Some images contain only Poisson noise, and some include Poisson-Gaussian noise. Gaussian σ is standard deviation for an image with peak intensity of 1.

Original Image σ(nm), ξ(nm), α	Poisson noise level	Gaussian noise σ	Noisy Image PSNR (dB)	SEMNet denoiser PSNR (dB)	DnCNN patch64 PSNR (dB)	DnCNN fullimage PSNR (dB)
0.8, 10, 0.3	10	None	16.38	28.66	**29.00**	28.71
0.8, 10, 0.3	10	0.02	16.31	28.63	**28.90**	28.56
0.8, 10, 0.3	10	0.05	15.94	**28.38**	28.20	27.89
0.8, 10, 0.3	10	0.10	14.82	**27.6**	25.35	25.88
1.2, 40, 0.7	2	None	9.96	**26.39**	26.06	26.18
1.2, 40, 0.7	2	0.02	9.94	**26.35**	25.66	25.79
1.2, 40, 0.7	2	0.05	9.85	**26.28**	24.09	24.55
1.2, 40, 0.7	2	0.10	9.53	**26.14**	19.23	22.23
1.6, 30, 0.5	100	None	26.13	42.64	**43.58**	41.14
1.6, 30, 0.5	100	0.02	25.47	42.55	**43.20**	41.18
1.6, 30, 0.5	100	0.05	23.05	40.42	**40.73**	39.44
1.6, 30, 0.5	100	0.10	19.02	**34.26**	32.06	31.75
0.8, 30, 0.6	5	None	13.22	**29.04**	28.86	28.95
0.8, 30, 0.6	5	0.02	13.18	**29.08**	28.53	28.68
0.8, 30, 0.6	5	0.05	13.0	**29.03**	27.54	27.78
0.8, 30, 0.6	5	0.10	12.38	**28.73**	24.04	25.48

3.4 Visualization Techniques and Improvements to LineNet1 [‡]

In this section, we consider multiple visualization tools to improve our understanding of LineNet1; one of these techniques is new to the visualization of denoising CNNs [3]. We use the resulting insights from these visualizations to motivate a study of two variations of LineNet1 with fewer neural network layers.

While CNNs have impressive performance for many tasks, there is a widespread desire to better understand them [89]. It is challenging to mathematically understand CNNs, so the machine learning community has introduced certain visualization techniques to help investigate their operation. A recent paper [90] claims that CNNs respond more to image textures than to object shapes in certain image recognition problems; this feature could enhance the usefulness of CNNs for roughness metrology applications. We consider three techniques in this section and believe that the application of one of them is new to the visualization of denoising CNNs although it has been applied previously in machine learning for style transfer [91] and for the representation of kernel functions [92]. The first technique examines the output images generated at each convolutional layer. The second technique considers the images which maximize the mean activation of filters in the convolutional layers. In the third technique, which appears to be new to the visualization of denoising CNNs, we propose a nonlinear dimensionality reduction technique which takes as input a SEM image and outputs a vector which represents a coarse summary of how a CNN perceives the texture of the image based on the features it has learned during the training process. Our visualization experiments suggest that there is some redundancy in the LineNet1 architecture. A smaller architecture with comparable performance would be advantageous as it would have shorter training and run times. Therefore, we consider a variant of LineNet1 with six convolutional layers and another with eleven convolutional layers. The experimental results indicate that the larger variant can still jointly denoise and estimate edge images in the presence of high Poisson noise while the smaller variant cannot. We also investigate the robustness of LineNet1 and its variants to the noise

[‡]Part of the data reported in this section is reprinted with permission from N. Chaudhary and S. A. Savari, "Towards a visualization of deep neural networks for rough line images," in *Proceedings of SPIE*, vol. 11177, 111770S, 2019. DOI: https://doi.org/10.1117/12.2535667. Copyright [2019] by SPIE.

model by considering test images with additional Gaussian noise.

3.4.1 Layer Outputs and Filters

The first convolutional layer of LineNet1 has 64 filters of dimension 3×3. We can visualize these filters by creating a grayscale representation of them. Figure 3.31 shows the 64 filters from the first convolutional layer of LineNet1. We can observe from Figure 3.31 that some filters resemble low pass filters while others resemble edge detectors. The output of each convolutional layer is a tensor of dimension $width \times height \times 64$ for LineNet1. This output can be visualized as 64 images of dimension $width \times height$. We visualize the output of LineNet1 at Layers 1, 4, 8, 12 and 16 with images of dimension 64×1024. Figure 3.32 illustrates a noisy input image to LineNet1 with a Poisson noise level of 5 electrons per pixel. Figure 3.33(a) shows eight images output from the first convolutional layer corresponding to filter numbers 48 to 55. Figures 3.33(b) and 3.33(c) respectively show samples of images output from the fourth and eighth convolutional layers. These images do not reveal the purpose of those layers. However, Figure 3.33(d) shows sample images from the twelfth layer, and here we can observe that the network is trying to separate edge information from the rest of the image. Figure 3.33(e) shows sample images from the sixteenth layer, and both the denoised image and the edge image have been constructed. We also observe possible redundancy as multiple output images across different layers seem similar.

3.4.2 Filter Activation and Activation Maximization

The activation of a filter in an internal layer for an input image refers to the mean pixel value of the output image corresponding to that filter. In the visualization of classification CNNs, researchers have been interested in finding input images which approximately maximize the activation of a filter [93, 94], and we likewise consider this approach for LineNet1. To iteratively generate such an image for a particular filter, initialize the algorithm with an input image containing random pixel values and compute its activation and the gradient with respect to the input image pixels. Next use gradient ascent without regularization to update the input image. Repeat this process for 20 steps to produce an (approximately) activation maximization input image. Figure 3.34(a)

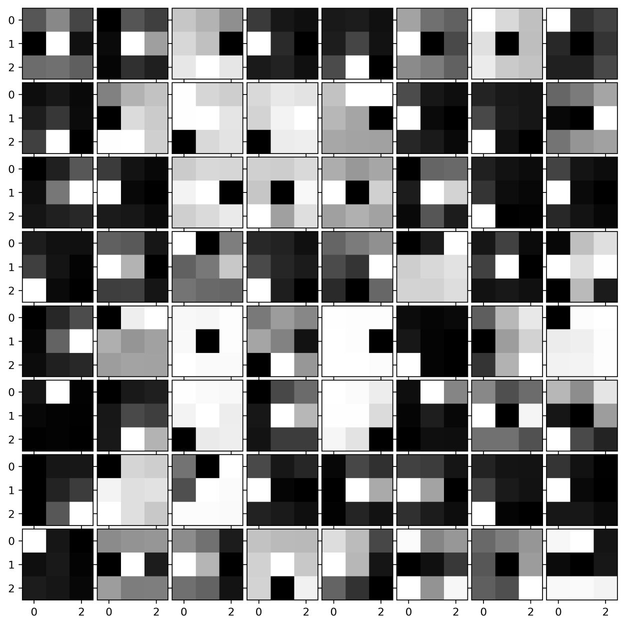


Figure 3.31: A grayscale representation of the 64 3×3 filters from the first convolutional layer of LineNet1. Reprinted with permission from [3].

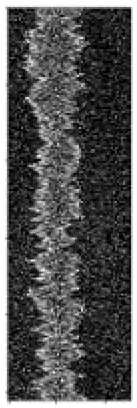
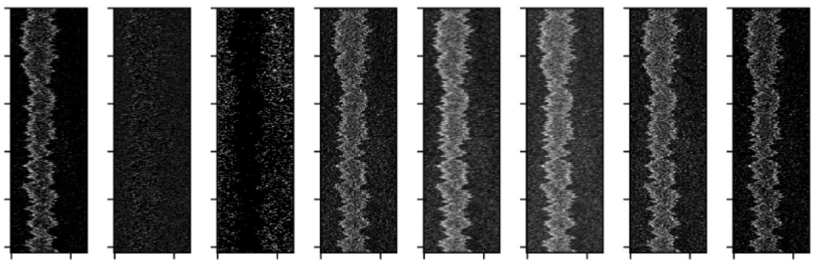
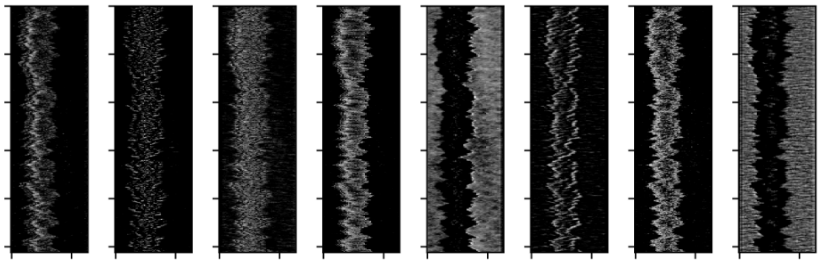
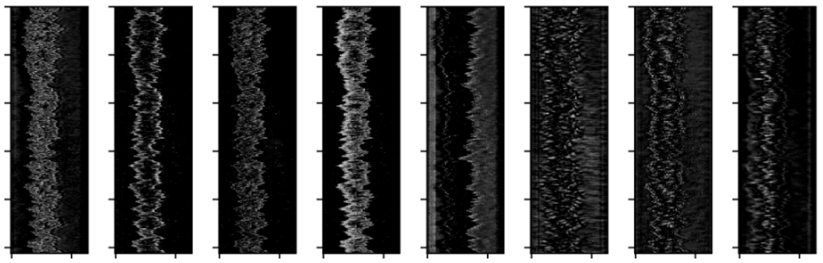
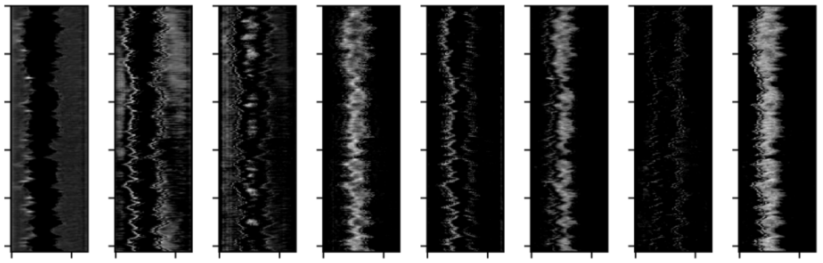


Figure 3.32: An input noisy SEM image with a Poisson noise level of 5 electrons per pixel. Reprinted with permission from [3].


(a) A sample of the outputs from the first convolutional layer of LineNet1.


(b) A sample of the outputs from the fourth convolutional layer of LineNet1.


(c) A sample of the outputs from the eighth convolutional layer of LineNet1.


(d) A sample of the outputs from the twelfth convolutional layer of LineNet1.

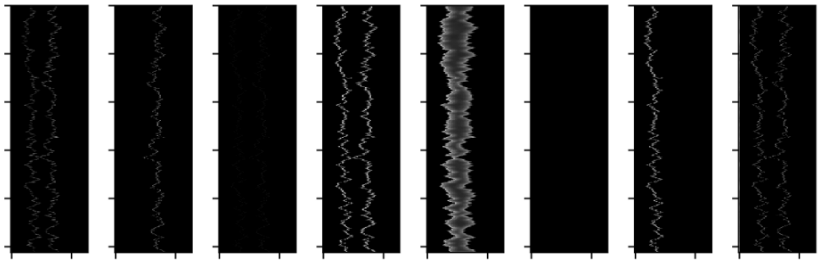
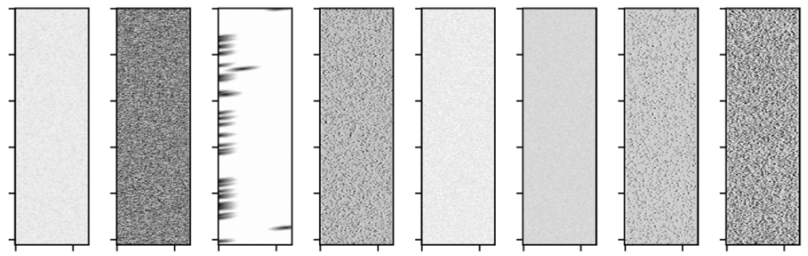

(e) A sample of the outputs from the sixteenth convolutional layer of LineNet1.

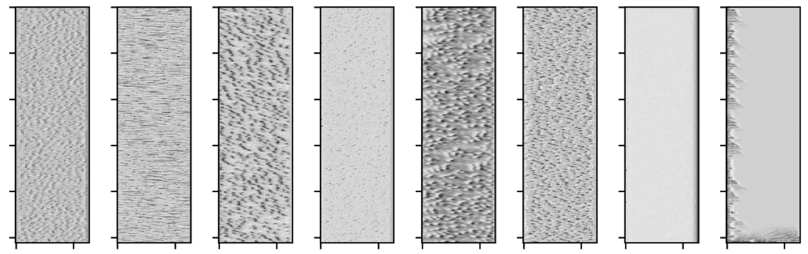
Figure 3.33: (a) The outputs from the first convolutional layer of LineNet1 corresponding to filter numbers 48-55. (b)-(e) A sample of the outputs from the fourth, eighth, twelfth and sixteenth convolutional layers, respectively, of LineNet1. Reprinted with permission from [3].

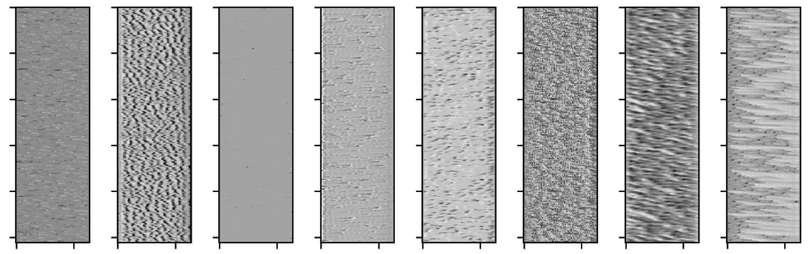
shows eight activation maximization images from the first convolutional layer corresponding to filter numbers 48 to 55. Observe that the activation maximization images contain a small range of textures. Similarly, Figure 3.34(b) shows images from the fourth layer, and they also show a small range of textures. Figure 3.34(c) shows the images of the eighth layer and Figure 3.34(d) shows the images of the twelfth layer. These layers display a wider and more complex range of textures. Figure 3.34(e) shows the images of the sixteenth layer. These images do not contain texture information, which is consistent with the earlier observation that by this point in the network LineNet1 has constructed the denoised image and the edge image. Figure 3.34 also suggests the possibility of redundancy within LineNet1 as there appear to be similar activation maximization images from filters in different layers.

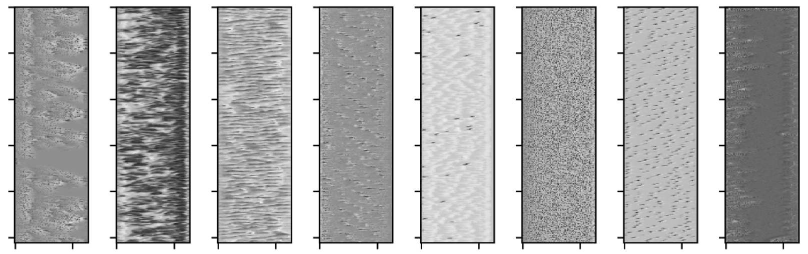
3.4.3 Gram Matrices and Texture

For our third visualization approach, we propose a nonlinear dimensionality reduction technique which takes a SEM image as input and outputs a vector which represents a coarse summary of how a CNN perceives the texture of the image based on the features it has learned during the training process. This technique is based on Gram matrices, which can be used to describe the style or texture of an image; this approach is inspired by work on texture synthesis [95] and image style transfer [91], but appears to be new in the visualization of denoising CNNs. Each layer of a CNN can be interpreted as a nonlinear filter bank, and given an input image the corresponding set of output images from a layer can be understood as feature maps. The Gram matrix associated with an input image and convolutional layer summarizes the correlations among the output images from that layer. Suppose that internal convolutional layer l of a CNN has N_l filters; for LineNet1 there are 16 internal convolutional layers and $N_l = 64$ for all l. Consider the vectorized version of each output image of internal convolutional layer l, which is a vector with M_l elements; for LineNet1 each output image has dimension 64×1024, so $M_l = 65536$ for all l. Store the N_l vectors for internal convolutional layer l in a matrix $F^l \in \mathbb{R}^{M_l \times N_l}$. The corresponding Gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$ for internal convolutional layer l describes the inner products of each pair of vectors associated with that layer. Our final summary of internal convolutional layer l is the Frobenius


(a) A sample of the activation maximization input images for the first convolutional layer of LineNet1.


(b) A sample of the activation maximization input images for the fourth convolutional layer of LineNet1.


(c) A sample of the activation maximization input images for the eighth convolutional layer of LineNet1.


(d) A sample of the activation maximization input images for the twelfth convolutional layer of LineNet1.

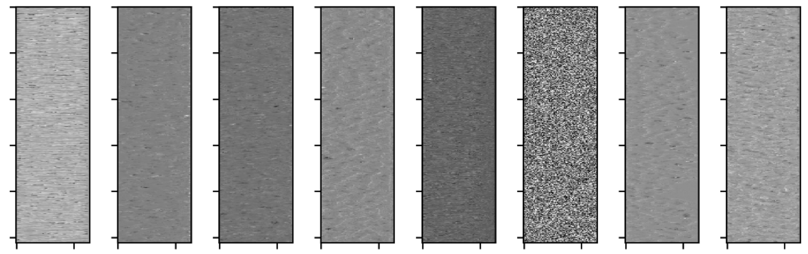

(e) A sample of the activation maximization input images for the sixteenth convolutional layer of LineNet1.

Figure 3.34: (a) The activation maximization input images for the first convolutional layer of LineNet1 corresponding to filter numbers 48-55. (b)-(e) A sample of the activation maximization input images for the fourth, eighth, twelfth and sixteenth convolutional layers, respectively, of LineNet1. Reprinted with permission from [3].

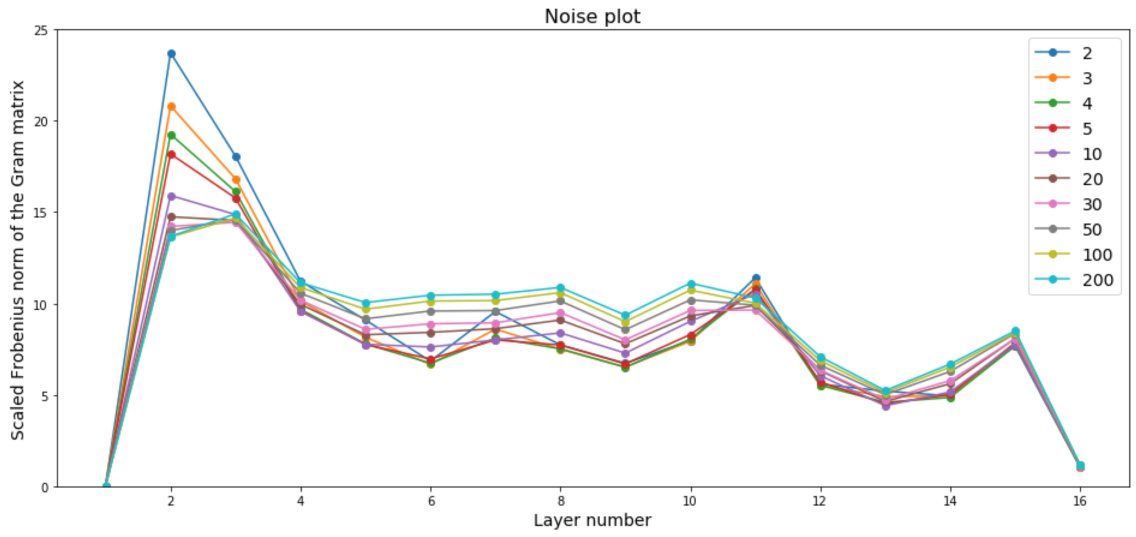


Figure 3.35: The scaled Frobenius norms of the Gram matrices corresponding to the 16 internal convolutional layers of LineNet1. The original rough line image has Palasantzas spectral model parameters $\sigma = 0.8$ nm, $\alpha = 0.3$ and $\xi = 10$ nm. The depicted noise levels are in units of electrons per pixel. Reprinted with permission from [3].

norm of G^l divided by $2N_lM_l$; this indicates how strongly an input image activates that layer. The dimension of the summary vector for a CNN is the number of internal convolutional layers of the CNN; for LineNet1 we produce a 16-dimensional vector for each image. Figure 3.35 considers an original rough line image with Palasantzas spectral model parameters $\sigma = 0.8$ nm, $\alpha = 0.3$ and $\xi = 10$ nm and plots the elements of the ten summary vectors associated with ten levels of Poisson noise. Other rough line images in our test dataset have similar behaviors in their summary vectors. Figure 3.35 shows that the initial four to five layers are strongly activated by the noise in the input image. Layers 6 to 10 provide a steady response to noise except when the Poisson noise level is 2 electrons per pixel. From Layer 11 onwards the activations appear to depend less strongly on the noise level. The initial layers of LineNet1 are devoting resources to the image denoising problem while the last few layers may be focusing more strongly upon the edge detection problem; note that the conventional wisdom for classification CNNs is that edge detection occurs in the first layers of those networks. The steady response to noise in some middle layers of Figure 3.35 also suggests that LineNet1 may have more layers than needed.

Recall that our previous visualization experiments also suggest the possibility of redundancy in LineNet1. To test this hypothesis, we consider a new variant of LineNet1 with six convolutional layers and another with eleven convolutional layers. As with the original version of LineNet1, each convolutional layer is followed by a batch normalization layer and a dropout layer with dropout probability set to 0.2.

3.4.4 Modified LineNet1 Experiments and Results

We train LineNet1_6layer and LineNet1_11layer in the same way we had earlier trained LineNet1; i.e., we use the same training set, the same number of training epochs, the same batch size and the same Adam [85] optimizer algorithm. Recall that LineNet1 outputs a grayscale denoised image and a grayscale edge image. We use the scientific python (SciPy) peak detector find_peaks on the grayscale edge image to estimate the exact edge positions. This algorithm fails to detect the edge positions in the output from some high noise input images, so in those cases we report the LER/LWR values as N/A. The training time for LineNet1_6layer, LineNet1_11layer and LineNet1 are approximately 14 hours, 27 hours and 41 hours, respectively, on a Tesla K80 graphical processing unit (GPU). The average prediction times per image of LineNet1_6layer, LineNet1_11layer and LineNet1 are approximately 0.038 second, 0.075 second and 0.14 second, respectively, on a Tesla K80 GPU.

Our first set of results show the robustness of the original seventeen-layer LineNet1 in the presence of a different type of noise. SEM image noise is often modeled by a combination of Gaussian noise and Poisson noise [54] while we trained our CNNs on images with only Poisson noise. Table 3.10 presents the results for the seventeen-layer LineNet1 when some Gaussian noise ($\sigma_G = 0.02, 0.05, 0.1$ assuming a maximum pixel intensity of 1) is added to the earlier Poisson noise test images and shows continued good performance. Table 3.11 shows the denoising performance of LineNet1_6layer, LineNet_11layer and LineNet1 in terms of peak signal-to-noise ratios (PSNR). Observe that LineNet1_11layer and LineNet1 have comparable PSNR performance, and both outperform the PSNR performance of LineNet1_6layer at the highest Poisson noise levels. Table 3.12 shows the LWR results for LineNet1_6layer, LineNet1_11layer and LineNet1. Observe

Table 3.10: LER results for LineNet1 for test images with Poisson-Gaussian noise in conjunction with the SciPy peak detector find_peaks. The parameter σ_G is the standard deviation of Gaussian noise for an image with a maximum pixel intensity of 1. Reprinted with permission from [3].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Gaussian noise σ_G	PSNR noisy (dB)	PSNR denoised (dB)	Left edge (nm)		Right edge (nm)	
					σ (true)	σ (obs.)	σ (true)	σ (obs.)
0.8, 10, 0.3	10	0.02	16.31	28.23	0.75	0.73	0.74	0.72
0.8, 10, 0.3	10	0.05	15.94	27.95	0.75	0.73	0.74	0.72
0.8, 10, 0.3	10	0.10	14.82	27.09	0.75	0.71	0.74	0.71
1.2, 40, 0.7	2	0.02	9.94	26.34	1.06	1.04	1.21	1.21
1.2, 40, 0.7	2	0.05	9.85	26.27	1.06	1.04	1.21	1.22
1.2, 40, 0.7	2	0.10	9.53	26.15	1.06	1.05	1.21	1.23
0.8, 30, 0.6	5	0.02	13.18	28.9	0.77	0.76	0.68	0.66
0.8, 30, 0.6	5	0.05	13.0	28.87	0.77	0.76	0.68	0.65
0.8, 30, 0.6	5	0.10	12.38	28.54	0.77	0.76	0.68	0.65

that as the number of layers in the network decreases the denoising capability also tends to diminish, and LineNet1_11layer seems to work well up to a Poisson noise level of 4 electrons per pixel while LineNet1_6layer appears to work well up to a Poisson noise level of 10 electrons per pixel.

The difference between the estimated edges from different networks becomes more apparent when we view the spectra of edges. Figure 3.36 shows three spectra for LineNet1_6layer, LineNet1_11layer and LineNet1. The SEM image for that figure has a Poisson noise level of 4 electrons per pixel with a Gaussian noise level of $\sigma_G = 0.05$, and the Palasantzas spectral model parameters for the rough edges were $\sigma = 1.0$ nm, $\alpha = 0.5$ and $\xi = 30$ nm. The spectra of LineNet1_11layer and the original LineNet1 are similar. To compare the predicted spectra with the true spectra, we measure the Euclidean distance between the two spectra assuming each spectra is a vector of 1024 elements. Figure 3.37 plots the Euclidean distance of the predicted edge spectra from the true edge spectra for multiple Poisson noise levels and a fixed Gaussian noise level of $\sigma_G = 0.05$. Observe that the Euclidean distance between the spectrum obtained through LineNet1_6layer with the true spectrum is much larger than the corresponding Euclidean distances between the spectra obtained from LineNet1_11layer and LineNet1 with the true spectrum.

Table 3.11: Denoising (PSNR) results for LineNet1, its six-layer variant and its eleven-layer variant for test images with Poisson-Gaussian noise. The parameter σ_G is the standard deviation of Gaussian noise for an image with a maximum pixel intensity of 1. Reprinted with permission from [3].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Gaussian noise σ_G	PSNR noisy (dB)	PSNR denoised (dB)		
				6layer	11layer	17layer
0.8, 10, 0.3	10	0.02	16.31	28.65	28.78	28.23
0.8, 10, 0.3	10	0.05	15.94	28.31	28.53	27.95
0.8, 10, 0.3	10	0.10	14.82	27.21	27.71	27.09
1.2, 40, 0.7	2	0.02	9.94	25.32	26.22	26.34
1.2, 40, 0.7	2	0.05	9.85	25	26.13	26.27
1.2, 40, 0.7	2	0.10	9.53	23.53	25.99	26.15
0.8, 30, 0.6	5	0.02	13.18	28.8	29.02	28.9
0.8, 30, 0.6	5	0.05	13.0	28.61	28.96	28.87
0.8, 30, 0.6	5	0.10	12.38	27.11	28.65	28.54

Table 3.12: LWR results for LineNet1, its six-layer variant and its eleven-layer variant in conjunction with the SciPy peak detector find_peaks for test images with Poisson-Gaussian noise. The parameter σ_G is the standard deviation of Gaussian noise for an image with a maximum pixel intensity of 1. Reprinted with permission from [3].

Original Image σ(nm), ξ(nm), α	Poisson noise level	Gaussian noise σ_G	LWR (nm)			
			σ (true)	σ 6layer	σ 11layer	σ 17layer
0.8, 10, 0.3	2	0.05	1.06	N/A	0.99	0.91
0.8, 10, 0.3	4	0.05	1.06	1.21	1.01	0.98
0.8, 10, 0.3	10	0.05	1.06	1.04	1.05	1.04
0.8, 10, 0.3	100	0.05	1.06	1.06	1.06	1.06
1.2, 40, 0.7	2	0.05	1.41	N/A	1.65	1.41
1.2, 40, 0.7	4	0.05	1.41	2.22	1.45	1.41
1.2, 40, 0.7	10	0.05	1.41	1.42	1.42	1.41
1.2, 40, 0.7	100	0.05	1.41	1.41	1.41	1.41
1.0, 30, 0.5	2	0.05	1.28	N/A	1.26	1.23
1.0, 30, 0.5	4	0.05	1.28	1.38	1.27	1.25
1.0, 30, 0.5	10	0.05	1.28	1.26	1.27	1.25
1.0, 30, 0.5	100	0.05	1.28	1.28	1.28	1.28

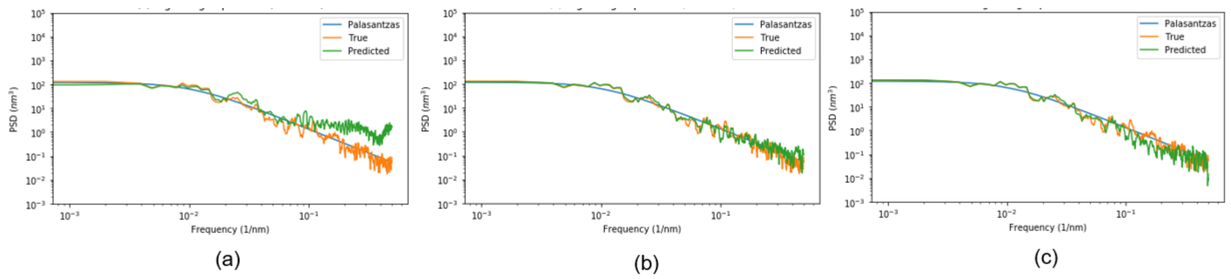


Figure 3.36: Predicted spectra and the Euclidean distances between each estimated spectrum and the corresponding true spectrum. (a)-(c) For our LineNet1_6layer network, LineNet1_11layer network and LineNet1 network the Euclidean distances between the predicted and true spectra are respectively 94.4 nm^3, 28.2 nm^3 and 19.5 nm^3. The SEM image for this figure has a Poisson noise level of 4 electrons per pixel with a Gaussian noise level of $\sigma_G = 0.05$, and the Palasantzas spectral model parameters for the rough edges were $\sigma = 1.0$ nm, $\alpha = 0.5$ and $\xi = 30$ nm. Reprinted with permission from [3].

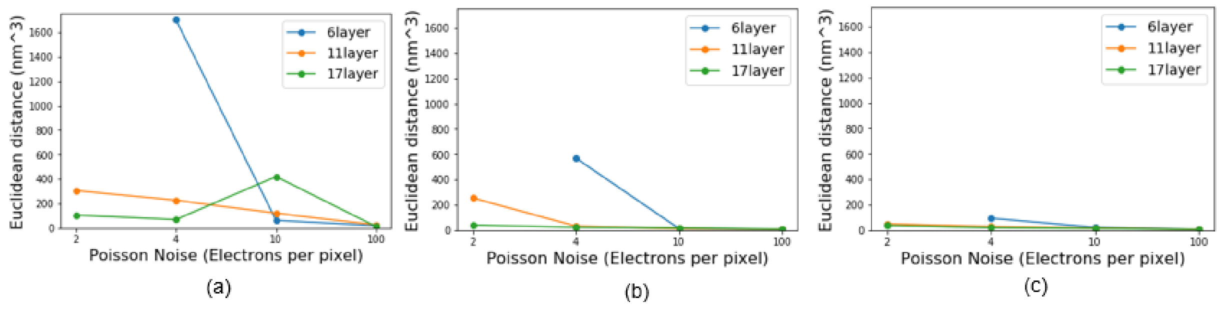


Figure 3.37: The Euclidean distances between the predicted edge spectra and the true edge spectra for LineNet1 and its two variants when the test images have a fixed Gaussian noise level of $\sigma_G = 0.05$. (a)-(c) The Palasantzas spectral model parameters for the rough edges of the test images are respectively $\sigma = 0.8$ nm, $\alpha = 0.3$, $\xi = 10$ nm for (a), $\sigma = 1.2$ nm, $\alpha = 0.7$, $\xi = 40$ nm for (b), and $\sigma = 1.0$ nm, $\alpha = 0.5$, $\xi = 30$ nm for (c). Reprinted with permission from [3].

4. SUMMARY AND CONCLUSIONS

The end of Moore's law will affect the industries that rely on the continuous growth of computation power. To prevent the end of Moore's law, engineers need to generate innovative solutions which utilize knowledge from multiple engineering fields. In this thesis, we examined two of the challenges facing integrated circuit fabrication and proposed solutions which employed knowledge from data compression, computer architecture, and the deep learning fields.

In the first part of this thesis, we discussed the existing multibeam systems and the challenges associated with them. We have presented a family of multibeam arrays and proposed scanning strategies for them. We have also discussed mask data characteristics and compression constraints specific to aperture array-based multibeam systems. To improve the communication and computational efficiency of multibeam mask writing systems, we proposed a datapath architecture that was inspired by multibeam direct-write tools and circuit testing. The proposed datapath architecture used parallel data compression and decompression. Our parallel compression algorithm will help to address two important problems in mask writing, namely, the data volume and the data preparation time. Our decompression architecture can be attached to the existing deflection plate architecture. We have also shown that video- and pixel-based representations are useful file formats for multibeam systems. We have demonstrated that data compression can be an important tool to address the "big data" problems of the mask industry. In the future, we would like to implement the parallel architecture in hardware.

In the second part of this thesis, we discussed the SEM metrology problem and its importance to the integrated circuit fabrication process. We have proposed multiple deep learning-based solutions for SEM metrology. We have simulated multiple SEM image datasets for the training of deep convolutional neural networks. We have shown that deep supervised learning is effective in the Poisson denoising of SEM images and in finding the edge positions in the noisy SEM images. We have also shown that deep learning is effective in the simultaneous denoising and edge estimation of SEM images. Our results show that time and memory resources can be saved by performing these tasks simultaneously using neural networks. Additionally, we have considered multiple visualization techniques that offered clues about redundancies in our deep convolutional neural networks. Visualization helps in understanding the workings of large convolutional neural networks. The vectors of scaled Frobenius norms of the Gram matrices for an original input image with different noise levels motivated a study of two smaller variants of our LineNet1 network. Our results indicate that it is possible to reduce the number of convolutional layers of LineNet1 from seventeen to eleven with a modest reduction in performance. The semiconductor industry has the opportunity to leverage advances in deep convolutional neural networks together with vast amounts of real and simulated data to learn complex physical processes. Deep learning can advance the field of semiconductor metrology. We hope our work will influence the industry to utilize deep learning-based solutions. In the future, we would like to work with a large class of realistic SEM data.

REFERENCES

[1] N. Chaudhary and S. A. Savari, "Parallel compression/decompression-based datapath architecture for multibeam mask writers," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 16, no. 4, 043503, 2017.

[2] N. Chaudhary, Y. Luo, and S. A. Savari, "Impact of parallelism on data volumes for a multibeam mask writer," *Journal of Vacuum Science & Technology B*, vol. 34, no. 6, 06KF01, 2016.

[3] N. Chaudhary and S. A. Savari, "Towards a visualization of deep neural networks for rough line images," in *Proceedings of SPIE*, vol. 11177, 111770S, 2019.

[4] N. Chaudhary, S. A. Savari, and S. S. Yeddulapalli, "Line roughness estimation and Poisson denoising in scanning electron microscope images using deep learning," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 18, no. 2, 024001, 2019.

[5] N. Chaudhary and S. A. Savari, "Simultaneous denoising and edge estimation from SEM images using deep convolutional neural networks," in *Proceedings of 30th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pp. 431–436, 2019.

[6] T. Goldstein and S. Osher, "The split Bregman method for L1-regularized problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 323–343, 2009.

[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] S. G. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, vol. 9, no. 9, pp. 1532–1546, 2000.

[9] C. Klein and E. Platzgummer, "MBMW-101: World's 1st high-throughput multi-beam mask writer," in *Proceedings of SPIE*, vol. 9985, 998505, 2016.

[10] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[11] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. S. B. Altaf, N. E. Jerger, and G. H. Loh, “Modular routing design for chiplet-based systems,” in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pp. 726–738, 2018.

[12] M. Feldman, *Nanolithography: The Art of Fabricating Nanoelectronic and Nanophotonic Devices and Systems*. Cambridge, UK: Woodhead Publishing, 2014.

[13] B.-J. Lin, “Immersion lithography and its impact on semiconductor manufacturing,” *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 3, no. 3, pp. 377–501, 2004.

[14] H. Matsumoto, H. Inoue, H. Yamashita, H. Morita, S. Hirose, M. Ogasawara, H. Yamada, and K. Hattori, “Multi-beam mask writer MBM-1000 and its application field,” in *Proceedings of SPIE*, vol. 9984, 998405, 2016.

[15] H. Matsumoto, H. Yamashita, T. Tamura, and K. Ohtoshi, “Multi-beam mask writer MBM-1000,” in *Proceedings of SPIE*, vol. 10454, 104540E, 2017.

[16] T. A. Brunner, X. Chen, A. Gabor, C. Higgins, L. Sun, and C. A. Mack, “Line-edge roughness performance targets for EUV Lithography,” in *Proceedings of SPIE*, vol. 10143, 101430E, 2017.

[17] P. De Bisschop and E. Hendrickx, “Stochastic effects in EUV lithography,” in *Proceedings of SPIE*, vol. 10583, 105831K, 2018.

[18] P. De Bisschop, “Stochastic effects in EUV lithography: random, local CD variability, and printing failures,” *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 16, no. 4, 041013, 2017.

[19] B. D. Bunday, M. Bishop, D. W. McCormack, J. S. Villarrubia, A. E. Vladár, R. Dixson, T. V. Vorburger, N. G. Orji, and J. A. Allgair, “Determination of optimal parameters for CD-SEM measurement of line-edge roughness,” in *Proceedings of SPIE*, vol. 5375, pp. 515–534, 2004.

[20] J. S. Villarrubia and B. Bunday, “Unbiased estimation of linewidth roughness,” in *Proceedings of SPIE*, vol. 5752, pp. 480–489, 2005.

[21] M. Chandramouli, F. Abboud, N. Wilcox, A. Sowers, and D. Cole, “Future mask writers requirements for the sub-10nm node era,” in *Proceedings of SPIE*, vol. 8522, 85221K, 2012.

[22] F. E. Abboud, M. Asturias, M. Chandramouli, and Y. Tezuka, “Mask data processing in the era of multibeam writers,” in *Proceedings of SPIE*, vol. 9235, 92350W, 2014.

[23] P. Petric, C. Bevis, A. Brodie, A. Carroll, A. Cheung, L. Grella, M. McCord, H. Percy, K. Standiford, and M. Zywno, “REBL nanowriter: Reflective electron beam lithography,” in *Proceedings of SPIE*, vol. 7271, 727107, 2009.

[24] A. Carroll, L. Grella, K. Murray, M. A. McCord, P. Petric, W. M. Tong, C. F. Bevis, S.-J. Lin, T.-H. Yu, T.-C. Huang, T. P. Wang, W.-C. Wang, and J. J. Shin, “The REBL DPG: recent innovations and remaining challenges,” in *Proceedings of SPIE*, vol. 9049, 904917, 2014.

[25] E. Slot, M. J. Wieland, G. de Boer, P. Kruit, G. F. ten Berge, A. M. C. Houkes, R. Jager, T. van de Peut, J. J. M. Peijster, S. W. H. K. Steenbrink, T. F. Teepen, A. H. V. van Veen, and B. J. Kampherbeek, “MAPPER: high throughput maskless lithography,” in *Proceedings of SPIE*, vol. 6921, 69211P, 2008.

[26] N. Chaudhary, Y. Luo, and S. Savari, “A parallel multibeam mask writing method and its impact on data volumes,” in *Proceedings of SPIE*, vol. 10032, 1003206, 2016.

[27] D. M. Tennant, “Limits of conventional lithography,” in *Nanotechnology* (G. Timp, ed.), pp. 161–205, New York, USA: Springer, 1999.

[28] V. Constantoudis, G. Patsis, A. Tserepi, and E. Gogolides, “Quantification of line-edge roughness of photoresists. II. Scaling and fractal analysis and the best roughness descriptors,” *Journal of Vacuum Science & Technology B*, vol. 21, no. 3, pp. 1019–1026, 2003.

[29] C. A. Mack and B. D. Bunday, “Using the analytical linescan model for SEM metrology,” in *Proceedings of SPIE*, vol. 10145, 101451R, 2017.

[30] T. Verduin, P. Kruit, and C. W. Hagen, “Determination of line edge roughness in low-dose top-down scanning electron microscopy images,” *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 13, no. 3, 033009, 2014.

[31] H. Matsumoto, “Blanking system for multi charged particle beams, and multi charged particle beam writing apparatus,” May 19 2016. US Patent 20,160,141,142.

[32] E. Platzgummer, C. Klein, and H. Loeschner, “Electron multibeam technology for mask and wafer writing at 0.1 nm address grid,” *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 12, no. 3, 031108, 2013.

[33] E. Platzgummer, C. Klein, and H. Loeschner, “eMET POC: realization of a proof-of-concept 50 keV electron multibeam mask exposure tool,” in *Proceedings of SPIE*, vol. 8166, 816622, 2011.

[34] E. Platzgummer, “Maskless lithography and nanopatterning with electron and ion multibeam projection,” in *Proceedings of SPIE*, vol. 7637, 763703, 2010.

[35] M. M. Mano and M. D. Ciletti, *Digital Design: With an Introduction to the Verilog HDL*. New Jersey: Pearson, 5th ed., 2012.

[36] H. Matsumoto, T. Iijima, M. Ogasawara, H. Inoue, and R. Yoshikawa, “Multi charged particle beam writing method and multi charged particle beam writing apparatus,” Apr. 6 2017. US Patent App. 15/384,021.

[37] L. Zhao, Y. Wei, and T. Ye, “Analysis of multi-e-beam lithography for cutting layers at 7-nm node,” *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 15, no. 4, 043501, 2016.

[38] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. Massachusetts, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[39] A. Chandra and K. Chakrabarty, “System-on-a-chip test-data compression and decompression architectures based on Golomb codes,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 355–368, 2001.

[40] V. Dai and A. Zakhor, “Lossless layout compression for maskless lithography systems,” in *Proceedings of SPIE*, vol. 3997, pp. 467–477, 2000.

[41] V. Dai, *Data Compression for Maskless Lithography Systems: Architecture, Algorithms and Implementation*. PhD thesis, University of California, Berkeley, 2008.

[42] J. Yang and S. A. Savari, “Lossless circuit layout image compression algorithm for maskless direct write lithography systems,” *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 10, no. 4, 043007, 2011.

[43] J. Yang, *Lossless Circuit Layout Image Compression Algorithms for Multiple Electron Beam Direct Write Lithography Systems*. PhD thesis, University of Michigan, Ann Arbor, 2012.

[44] N. Chaudhary, Y. Luo, S. A. Savari, and R. McCay, “Lossless layout image compression algorithms for electron-beam direct-write lithography,” *Journal of Vacuum Science & Technology B*, vol. 33, no. 6, 06FD01, 2015.

[45] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[46] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, New York, USA: Curran Associates, Inc., 2012.

[48] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[49] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[50] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[51] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?," *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223–1247, 2017.

[52] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4509–4522, 2017.

[53] E. I. Thorsos, "The validity of the Kirchhoff approximation for rough surface scattering using a Gaussian roughness spectrum," *The Journal of the Acoustical Society of America*, vol. 83, no. 1, pp. 78–92, 1988.

[54] P. Cizmar, A. E. Vladár, B. Ming, and M. T. Postek, "Simulated SEM images for resolution measurement," *Scanning*, vol. 30, no. 5, pp. 381–391, 2008.

[55] P. Cizmar, A. E. Vladár, and M. T. Postek, "Optimization of accurate SEM imaging by use of artificial images," in *Proceedings of SPIE*, vol. 7378, 737815, 2009.

[56] G. A. Seber and A. J. Lee, *Linear Regression Analysis*, vol. 329. New Jersey, USA: John Wiley & Sons, 2012.

[57] A. Karpathy, "Stanford University CS231n: Convolutional Neural Networks for Visual Recognition," 2018.

[58] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[59] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[60] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Proceedings of Machine Learning Research*, vol. 38, pp. 192–204, 2015.

[61] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2933–2941, New York, USA: Curran Associates, Inc., 2014.

[62] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks* (M. A. Arbib, ed.), pp. 255–258, Cambridge, MA, USA: MIT Press, 1995.

[63] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of Machine Learning Research*, vol. 37, pp. 448–456, 2015.

[64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[65] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *Proceedings of the Twenty-first International Conference on Machine Learning*, pp. 78–86, ACM, 2004.

[66] E. Schwartz, R. Giryes, and A. M. Bronstein, "DeepISP: Learning end-to-end image processing pipeline," *arXiv preprint arXiv:1801.06724*, 2018.

[67] N. Chaudhary, S. A. Savari, and S. S. Yeddulapalli, "Deep supervised learning to estimate true rough line images from SEM images," in *Proceedings of SPIE*, vol. 10775, 107750R, 2018.

[68] N. Chaudhary, S. A. Savari, and S. S. Yeddulapalli, "Automated rough line-edge estimation from SEM images using deep convolutional neural networks," in *Proceedings of SPIE*, vol. 10810, 108101L, 2018.

[69] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in *Advances in Neural Information Processing Systems 21* (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 769–776, New York, USA: Curran Associates, Inc., 2009.

[70] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 2802–2810, New York, USA: Curran Associates, Inc., 2016.

[71] Y. Tai, J. Yang, X. Liu, and C. Xu, "MemNet: A persistent memory network for image restoration," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4539–4547, 2017.

[72] J. Chen, J. Chen, H. Chao, and M. Yang, "Image blind denoising with generative adversarial network based noise modeling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3155–3164, 2018.

[73] T. Remez, O. Litany, R. Giryes, and A. M. Bronstein, "Deep convolutional denoising of low-light images," *arXiv preprint arXiv:1701.01687*, 2017.

[74] T. Remez, O. Litany, R. Giryes, and A. M. Bronstein, "Class-aware fully convolutional Gaussian and Poisson denoising," *IEEE Transactions on Image Processing*, vol. 27, no. 11, pp. 5707–5722, 2018.

[75] S. W. Hasinoff, "Photon, Poisson noise," in *Computer Vision: A Reference Guide* (K. Ikeuchi, ed.), pp. 608–610, Boston, USA: Springer, 2014.

[76] M. Weigert, U. Schmidt, T. Boothe, A. Müller, A. Dibrov, A. Jain, B. Wilhelm, D. Schmidt, C. Broaddus, S. Culley, M. Rocha-Martins, F. Segovia-Miranda, C. Norden, R. Henriques, M. Zerial, M. Solimena, J. Rink, P. Tomancak, L. Royer, F. Jug, and E. W. Myers, "Content-

aware image restoration: pushing the limits of fluorescence microscopy," *Nature Methods*, vol. 15, no. 12, pp. 1090–1097, 2018.

[77] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, Springer, 2015.

[78] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2Noise: Learning image restoration without clean data," in *Proceedings of the Thirty-fifth International Conference on Machine Learning*, vol. 80, pp. 2965–2974, 2018.

[79] A. Krull, T.-O. Buchholz, and F. Jug, "Noise2Void - learning denoising from single noisy images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2129–2137, 2019.

[80] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9446–9454, 2018.

[81] C. A. Mack, "Generating random rough edges, surfaces, and volumes," *Applied Optics*, vol. 52, no. 7, pp. 1472–1480, 2013.

[82] G. Palasantzas, "Roughness spectrum and surface width of self-affine fractal surfaces via the k-correlation model," *Physical Review B*, vol. 48, no. 19, pp. 14472–14478, 1993.

[83] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[84] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[85] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the Third International Conference on Learning Representations*, 2015.

[86] G. F. Lorusso, T. Sutani, V. Rutigliani, F. van Roey, A. Moussa, A.-L. Charley, C. Mack, P. Naulleau, C. Perera, V. Constantoudis, M. Ikota, T. Ishimoto, and S. Koshihara, "Need for LWR metrology standardization: the imec roughness protocol," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 17, no. 4, 041009, 2018.

[87] D. J. Thomson, "Spectrum estimation and harmonic analysis," *Proceedings of IEEE*, vol. 70, no. 9, pp. 1055–1096, 1982.

[88] Y. Luo and S. A. Savari, "Multitaper and multisegment spectral estimation of line-edge roughness," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 16, no. 3, 034001, 2017.

[89] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual analytics in deep learning: An interrogative survey for the next frontiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 1–20, 2019.

[90] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," *arXiv preprint arXiv:1811.12231*, 2018.

[91] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

[92] G. R. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite programming," *The Journal of Machine Learning Research*, vol. 5, no. 1, pp. 27–72, 2004.

[93] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," Tech. Rep. 1341, University of Montreal, 2009.

[94] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[95] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D.

Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 262–270, New York, USA: Curran Associates, Inc., 2015.