UNCERTAINTY QUANTIFICATION IN LINE EDGE ROUGHNESS ESTIMATION USING CONFORMAL PREDICTION

A Thesis

by

INIMFON IDONGESIT AKPABIO

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

MASTER OF SCIENCE

Chair of Committee,	Serap A. Savari
Committee Members,	Kristen Maitland
	Stavros Kalafatis
	Dileep-Mantisseri Kalathil
	Philip Yasskin
Head of Department,	Miroslav. M. Begovic

May 2022

Major Subject: Computer Engineering

ABSTRACT

With its increasing pervasiveness across multiple industries, machine learning is expected to occupy greater significance in the semiconductor manufacturing industry. To foster trust and facilitate the adoption of machine-learning models, it is necessary to employ prediction intervals which summarize the performance and consistency of their predictions. Conformal prediction is a recent, and mathematically proven technique to approach prediction intervals for classification and regression problems. Unlike traditional approaches, conformal prediction does not require distributional assumptions such as Gaussianity. Furthermore, it can be combined with other techniques to yield a variety of interval predictions algorithms. We aim to illustrate the applications and performance of some of these methods on line edge roughness (LER) estimation. Experimental studies have shown that LER degrades the performance of semiconductor devices. While scanning electron microscope (SEM) is the method of choice for measuring LER, it is fraught with added uncertainty such as instrumental noise, edge effects, and Poisson noise. This work focuses on developing prediction intervals for LER estimates derived from EDGENet, which is a deep convolutional neural network trained on a large dataset of simulated SEM images. EDGENet was originally developed by our research group and directly outputs predictions of true edge positions from a corrupted SEM image.

DEDICATION

To my parents, brothers, aunt, grandmother, and late grandfather.

ACKNOWLEDGMENTS

I want to specially thank my advisor Dr. Serap Savari for her guidance, encouragement and massive contributions to this research and our other joint publications. She has also spent countless hours proof reading this document as well as our other research publications. The importance of her mentorship over the past 2 years cannot be overstated.

I also want to appreciate my committee members for their time and effort spent on proof reading this document and contributing to its overall quality. Special thanks to Dr. Narendra Chaudhary for laying the foundations of this research and sharing some of his experiences.

I am grateful to the Texas A&M University High Performance Research Computing Facility for enabling me to conduct part of the research and Mr. Prasad L. Hugar for assisting with some of the experiments.

Finally, I want to thank my extended family and friends for their support and encouragement throughout this process.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by my thesis committee consisting of Professor Serap Savari [advisor] and Professor Kristen Maitland of the Department of Biomedical Engineering and Professor Dileep-Mantisseri Kalathil of the Department of Electrical and Computer Engineering and Professor Stavros Kalafatis of the Department of Electrical and Computer Engineering and Professor Philip Yasskin of the Department of Mathematics.

The simulation data analyzed in Chapter II and III was provided by Dr. Narendra Chaudhary. The results presented in this work have also been published in joint publications with Dr. Serap Savari.

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

Funding Sources

My graduate studies were supported in part by scholarships provided by the Texas A&M Department of Electrical and Computer Engineering.

NOMENCLATURE

LER	Line Edge Roughness
SEM	Scanning Electron Microscope
AI	Artificial Intelligence
DL	Deep Learning
ML	Machine Learning
CP	Conformal Prediction
QR	Quantile Regression
CQR	Conformalized Quantile Regression
CNN	Convolutional Neural Network
FCNN	Fully-Connected Neural Network
MSE	Mean Squared Error
MAE	Mean Absolute Error
ReLU	Rectified Linear Unit
NLP	Natural Language Processing
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
BLSTM	Bidirectional Long Short Term Memory

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Artificial intelligence (AI) carries the potential to generate significant financial value for the semiconductor industry at multiple stages of operations: design, research, and sales [5]. Research from Mckinsey and Co. show that AI and Machine Learning (ML) currently generates around 5-8 billion dollars every year for semiconductor companies. Incorporating AI applications in the production process affords companies the ability to minimize losses and hence optimize their operations [6]. The reliance on AI technologies on semiconductor companies is expected to increase as more opportunities for application [7] become available [8]. Despite the success of AI in the semiconductor industry, there still remain multiple challenges that inhibit its full utilization and realization of smarter manufacturing [9]. In a meeting with multiple leaders of the manufacturing sector and the National Institute of Standards and Technology on implementations of AI in manufacturing, uncertainty and an absence of transparency were identified as one of the major obstacles to the adoption of AI [10]. Uncertainty quantification is important in aiding the process of decision-making and also fostering trust in automated systems [11]. Uncertainty quantification in machine learning depends on the nature of the task. In regression problems where the output is typically a continuous quantity, it suffices to provide an interval which is believed to contain the actual value with some probability. Similarly, we can apply this to classification problems by using probabilities to capture confidence [11]. There are a plethora of techniques for creating predictions intervals. In assessing these techniques, they must maintain the following two criteria: yield acceptable coverage and produce the shortest interval lengths.

One of such methods is conformal prediction (CP) [12]. Conformal prediction is an innovative, distribution free technique for generating prediction intervals based on minimal assumptions [4]. In fact, the main assumption is "exchangeability" and can be interpreted as how similar past samples are to future samples. Since conformal prediction does not rely on assumptions like Gaussianity, it can be applied to a variety of problems. In this study, we will consider the "inductive" flavor of conformal prediction [13] whose procedure involves first fitting a regression function on

a training set and subsequently defining a non-conformity score on a separate calibration set that defines the dissimilarity of a given sample. The defined non-conformity score can then be used to create prediction intervals for unseen test samples. Furthermore, multiple algorithmic approaches can be applied to improve the quality of conformal prediction. For instance, Candès [14] applies a quantile regression approach to realize adaptable intervals known as conformal quantile regression (CQR).

We will demonstrate the potential of conformal prediction when applied to line edge roughness (LER) estimation. An essential step in the fabrication of chips is lithography and involves projecting Extreme Ultraviolet (EUV) light unto the resist layer of a wafer to create precise patterns/features that dictate the structure and functionality of the chip. As a result of advances in lithographic technology, these features are typically in the order of nanometers and hence, require high-level precision. With such a high demand for precision, the semiconductor industry has developed multiple metrics to measure and study the quality of such features. Line edge roughness is defined as a measure of the deviation of a rough line edge on a semiconductor chip [19]. Such deviations typically occur on a scale that is relatively smaller than the resolution of the imaging equipment [19]. At scales below 10nm, LER can appear as the same size as features on a chip which can effectively begin to affect the performance and yield of transistors [16]. The most common method of measuring LER is the scanning electron microscope (SEM). However, it is plagued with additional error sources in the form of instrumental noise, edge effects, etc. We will turn our attention to EDGENet [3], [2] which is a Convolutional Neural Network (CNN) that accepts noisy 1024x64 SEM images as input and outputs line edge positions and an LER value prediction for both the left and right edges. EDGENet was trained using a simulated dataset of SEM images corrupted by random Poisson noise for which the ground truth LER is known. By studying the predictions of EDGENet and the simulated dataset, we can develop prediction intervals for the LER of each image. In the following sections we will provide a thorough theoretical background for conformal prediction and also introduce new algorithmic variants of CP. In the results section we will evaluate the performances of each algorithm on the test sub portion of our dataset.

1.1 Estimating Line Edge Roughness From SEM Images

Due to the heavy demand for accurate and repeatable measurement of nanostructure dimensions on semiconductor samples, Scanning Electron Microscope (SEM) images are widely employed in the field of metrology. This technique involves a specialized electron microscope that emits a focused beam of electrons. By scanning the focused beam across a material sample in a raster scan fashion, we can obtain an informative image of the sample at the sub-micron level. These beams come in contact with the topology of such samples and produce signals that give information about the geometry of any nanostructures on its surface. The amount of energy in the electron beam determines the quality of the resulting SEM image. High dose electron beams generally tend to produce images with lesser noise but require longer acquisition times and can ultimately damage any nanostructures on the surface of the sample. On the other hand, low-dose SEM images are particularly desirable due to their relatively short acquisition times and lesser resist shrinkage. Nevertheless, low dose SEM imaging introduces additional difficulties in determining edge geometries from the resulting images such as Poisson noise, Gaussian blur, and edge effects [17], [18]. Multiple approaches have been applied to edge estimation such as physical model-based regression [19] and filtering techniques [20]. These approaches often demand thorough selection and adjustment of multiple parameter to avoid altering edge geometries in images and sometimes are hampered by modelling assumptions.

Edge roughness measurements of nanostructures such as lines are necessary in establishing quality in semiconductor manufacturing. Nanostructure edges in SEM images are typically bright since the escape probability of secondary electrons increases at edges. Additionally, this brightness spans some nanometers (See Figure 1.1 for an example of a simulated noisy SEM image of a line); hence, the actual location of edges is not apparent. The ability to accurately estimate edge geometries and their corresponding roughness has significant impact on the yield of semiconductor manufacturing/fabrication processes. Therefore, it is necessary to investigate sophisticated algorithmic approaches to do this accurately.

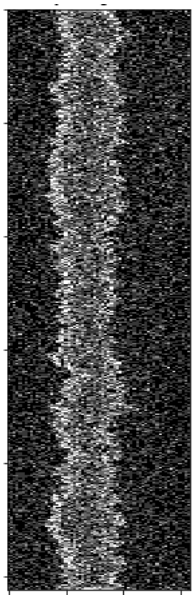

Figure 1.1: A simulated noisy SEM image of dimension 64x1024. The image has one line with two edges. Reprinted with permission from [1].

Advances in machine learning in recent years have birthed complex models such as deep convolutional neural networks [21] which perform exceptionally in popular image classification tasks [22]. Deep neural networks have already been shown to achieve state of the art results in image denoising problems [23]. Dr. Chaudhary developed EDGENet [3], a 17 layer CNN which outputs estimated edge positions given an input noisy SEM image of height 1024 pixels and width 64 pixels. Given the output edge positions, we can easily obtain an LER estimate by computing the standard deviation of the edge positions. We build on this work by developing multiple CP-based approaches to construct prediction intervals for the LER estimates from EDGENet. We first introduce ten preliminary CP-based techniques and ensemble schemes. Subsequently, we will introduce a new approach that improves performance by taking advantage of better modelling.

2. PREDICTIVE INFERENCE

2.1 Conformal Prediction

Supervised learning problems typically involve the realization of a parametric function $f(\cdot)$ that maps an input variable x^i to its corresponding output y^i over multiple input/output pairs z^i = (x^i, y^i) in a dataset. This technique is commonly referred to as training or fitting a function to a dataset. Note that both x^i and y^i can be multidimensional variables. Supervised learning problems can come in two different flavors: regression or classification problems. In regression problems, y^i is typically a continuous variable while in classification problems, y^i is a discrete variable. The fitted parametric function is heavily dependent on factors such as the nature of the learning algorithm, cost function, hyperparameters and optimization schemes used in the training process. With conformal prediction, we aim to construct a well calibrated prediction interval C(x^i) which provides finite sample coverage guarantees on the output variable given a prespecified miscoverage rate. In other words, given a miscoverage rate $\alpha \in (0, 1)$ and some random test input variable, X^i, our marginal prediction interval C(X^i) must be such that:

$$P[Y^i \in C(X^i)] \geq 1 - \alpha. \tag{2.1}$$

The miscoverage rate is simply the proportion of samples for which we do not expect to achieve coverage. There are 2 key points to note here:

- The sample pairs, z^i = (x^i, y^i) must be exchangeable [24]. The exchangeability criterion is popularly used in literature and basically means that samples should be similar to each other.

- The prediction interval offers average coverage, meaning we only achieve the desired coverage on average over multiple pairs, z^i. Hence, given a single random input sample, X^i, our constructed interval could potentially undercover or overcover.

The absence of multiple overbearing assumptions such as Gaussianity offers this approach a feel

of simplicity and ubiquity. So long as exchangeability is satisfied, we are guaranteed to achieve the required coverage on average.

Conformal prediction can come in 2 flavors: inductive or split and transductive [24]. Our focus here is on inductive. The procedure is as follows:

1. Split the entire dataset into 3 categories: the training set Z_T, calibration set Z_C, and a test set Z_τ.

2. Train a base regression function $f(\cdot)$ on the training set.

3. Define a nonconformity score $\eta(z)$ for the problem and calculate nonconformity scores for the entire calibration set. Nonconformity scores are used as a measure of dissimilarity of a given random sample and help to enforce the need for exchangeability. The most popular nonconformity score is the absolute difference of the actual value and the predicted value.

$$\eta(z^i) = |y^i - f(x^i)| \tag{2.2}$$

4. Sort all the nonconformity scores calculated over the calibration set in decreasing order as $r_1, r_2, \ldots r_k$.

5. Given a pre-specified miscoverage rate α, we calculate m = $\lfloor \alpha(k+1) \rfloor$ as the (1-α)th percentile of the descending nonconformity scores.

6. Finally, the resulting prediction interval for a test sample x^j is given as:

$$[f(x^j) - r_m,\ f(x^j) + r_m] \tag{2.3}$$

While this method is both effective and simple, it produces constant sized intervals. In most applications it is desirable to have prediction intervals whose lengths vary to reflect the difficulty of the problem for each input sample. For instance, our application involves SEM images with different levels of noise. Hence, we would prefer a model that varies its interval length to capture

the difficulty of LER estimation in the presence of more noise. To address this issue we will now introduce a variant of CP known as normalized conformal prediction.

2.1.1 Normalized Conformal Prediction

Normalized CP tackles the problem of adaptable interval lengths by introducing a normalized nonconformity score $\eta_N(z^i)$. The central idea behind this normalized nonconformity score is to incorporate the difficulty of inputs into the original nonconformity scores that are used in CP. In practice this is achieved by modelling a separate normalization function $\phi(\theta; x^i)$ (θ represents the parameters of this function) on the training set. Different implementations exist for the normalized nonconformity function. Linusson [12] models $\phi()$ by training a function $\gamma(x)$ on all data pairs $(x, \eta(z^i))$. By selecting an additional sensitivity parameter β, the normalized nonconformity score is given by:

$$\eta_N(z^i) = \frac{|y^i - f(x^i)|}{\gamma(x^i) + \beta} \tag{2.4}$$

Once the normalized nonconformity score has been defined, we follow the same procedure in CP and calculate the nonconformity scores over the calibration set which are then sorted in descending order as $r_{N1}, r_{N2}, \ldots r_{Nk}$. Given a pre-specified miscoverage rate α, we calculate m = $\lfloor \alpha(k+1) \rfloor$ as the (1-α)th percentile of the descending normalized nonconformity scores. Finally, the resulting normalized conformal prediction interval for a test sample x^j is given as:

$$[f(x^j) - r_{Nm} * (\gamma(x^j) + \beta),\ f(x^j) + r_{Nm} * (\gamma(x^j) + \beta)]. \tag{2.5}$$

In our experiments we opt for a similar normalized nonconformity score to that used in [25]. We model our normalization function $\phi(\theta; x^i)$ as exp(-μ^i) where μ^i is also a function that is trained on all data pairs (x, $-\ln(\eta(z))$). Hence, our normalized nonconformity score is given by:

$$\eta_N(z^i) = \frac{|y^i - f(x^i)|}{\phi(\theta; x^i)} \tag{2.6}$$

Modelling the normalization function when high dimensional inputs are involved can be extremely difficult. Due to the large size of our input image (1024 pixels x 64 pixels), we chose to adopt a 2-part approach to modelling the normalization function which is detailed as follows:

1. First we train an autoencoder neural network [26] to reduce the dimensionality of a given input image x^i from a 2-dimensional 1024x46 vector to a 1-dimensional 64 length vector $p(x^i)$.

2. Secondly, we train a second network model, $q(x)$ that takes the compressed 64 length vector $p(x^i)$ as input and predicts -ln $|y^i - f(x^i)|$ (i.e. $\mu(x^i) = q(p(x^i))$ and $\phi(\theta; x^i) = \exp(-\mu(x^i))$).

Hence, given a future test image x^j, the resulting normalized conformal prediction interval is given as:

$$[f(x^j) - r_{Nm}\phi(\theta; x^j),\ f(x^j) + r_{Nm}\phi(\theta; x^j)]. \tag{2.7}$$

One quick observation here is that if our models are perfect then all nonconformity scores will turn out to be one and as a result, y^j will become an endpoint of our prediction interval. Hence, our prediction interval should give some indication of the difficulty of the problem. Another observation is that our intervals are inherently centered at f(x^i). While this does not pose any conspicuous disadvantages to predictive inference, it limits the flexibility of our prediction intervals. An alternative approach to this problem would be to directly predict the lower and upper limits of the prediction interval while maintaining marginal coverage guarantees. For this we will introduce a new approach known as quantile regression.

2.2 Quantile Regression

Quantile Regression (QR) [27] is a technique for predicting conditional quantiles and nonparametric probabilistic forecasting that is commonly used in statistical modelling. To provide more understanding, consider the basic regression problem under the mean absolute error criterion;

this is analogous to trying to predict the 50th quantile of the response variable. Hence, quantile regression is a special type of basic regression. In theory, quantile regression allows us to predicts all quantiles from the 0th to the 100th quantile to a respectable degree of accuracy depending on the algorithm and implementation.

This technique can effectively be adapted to constructing prediction intervals by fitting 2 quantile regressors; one for the lower bound and another for the upper bound on a proper training set. To provide more context, the conditional cumulative distribution function of a response random variable Y with respect to a given predictor random variable X is defined by $P[Y \leq y|X = x]$. By simply inverting this function we arrive at the definition for a conditional quantile function. For a given quantile level $\epsilon \in [0, 1]$:

$$q_\epsilon(x) = \inf\{y : P[Y \leq y|X = x] \geq \epsilon\}. \tag{2.8}$$

Hence, given a miscoverage rate $\alpha \in (0, 1)$, our prediction interval is given by:

$$P[q_{0.5\alpha}(x) \leq Y \leq q_{1-0.5\alpha}(x)|X = x] \geq 1 - \alpha. \tag{2.9}$$

There are multiple algorithmic implementations for quantile regressors such as random forests and neural networks. In our implementation we attempt to modify the deep convolutional network EDGENet to perform quantile regression. We will describe how we do this in the next section. The common loss function used in quantile regression is the pinball loss function. For a given quantile level $\epsilon \in [0, 1]$, the pinball loss function is defined as follows:

$$\rho_\epsilon(y, f(x^i)) = \begin{cases} \epsilon(y - f(x^i)), & y - f(x^i) \geq 0 \\ (1 - \epsilon)(f(x^i) - y), & y - f(x^i) < 0 \end{cases}$$

Observe that for the 50th quantile level, this resembles the mean absolute error loss function. Hence, given a quantile regression model $q_\epsilon(\cdot)$, N images where the i^{th} noisy SEM image is x^i,

and y^i is the ground truth value of the LER corresponding to that image, our training objective is to minimize:

$$\frac{1}{N}\sum_{i=1}^{N}\rho_\epsilon(y^i, f(x^i)). \tag{2.10}$$

As noted earlier, our algorithm of choice for implementing quantile regression is deep convolutional networks which are trained using some kind of gradient descent procedure. Unfortunately, the non-differentiability of our loss function $\rho_\epsilon(y, \hat{y})$ at $y = \hat{y}$ could adversely affect the training process [28]. In view of this, we consider the smoothed pinball loss function [29]:

$$s_{\epsilon,a}(y, f(x^i)) = \epsilon(y - f(x^i)) + a\ln(1 + \exp(\frac{f(x^i) - y}{a})), \tag{2.11}$$

where a is the smoothing parameter. By applying the smoothed pinball loss, we are able to realize an approximation $\hat{q}_\epsilon(\cdot)$ of $q_\epsilon(\cdot)$. To this end we train the weights/parameters of select layers in EDGENet to perform quantile regression. The rationale behind this is that since EDGENet is already trained to accurately estimate edge positions and hence LER values for images, we can leverage this information to also compute quantiles for LER values. As we will see in the next section, this is helpful because it allows us to build upon existing knowledge from a pre-trained model rather than training from scratch which can be computationally intensive. This approach is commonly known as transfer learning and will be discussed in the next section.

Despite its wide spread popularity, quantile regression does not guarantee coverage. This is noticeable in our QR experiments as we will see later in the results section. We can fix this by applying a conformalization procedure similar to CP as we will see in the next subsection.

2.2.1 Conformalized Quantile Regression

The lack of coverage guarantee in QR motivates the need for a means of adjusting the lower and upper bounds of its prediction intervals $\hat{q}_{0.5\alpha}$ and $\hat{q}_{1-0.5\alpha}$. Conformal quantile regression (CQR)

introduces a nonconformity score η_Q to enable us conformalized the quantile regression procedure.

$$\eta_Q(z^i) = \max\{\hat{q}_{0.5\alpha}(x^i) - y^i,\ y^i - \hat{q}_{1-0.5\alpha}(x^i)\}. \tag{2.12}$$

Similarly to CP, we compute this nonconformity measure on every sample in our calibration set and sort the resulting nonconformity scores in descending order as $r_{Q1}, r_{Q2}, \ldots r_{Qk}$. Given a prespecified miscoverage rate α, we calculate m = $\lfloor \alpha(k+1) \rfloor$ as the (1-α)th percentile of the descending nonconformity scores. Finally, the resulting CQR prediction interval for a test sample x^j is given as:

$$[\hat{q}_{0.5\alpha}(x^j) - r_{Qm},\ \hat{q}_{1-0.5\alpha}(x^j) + r_{Qm}]. \tag{2.13}$$

This interval offers valid marginal coverage. Multiple variations of CQR also exist. CQR-m [30] is a variation on CQR that utilizes an estimate of the regression median to supplement the both lower and upper quantiles. CQR-r [31] is similar to CQR-m but does not require an estimate of the regression median. CQR-r a modified version of the CQR nonconformity score:

$$\eta_{Q-r}(z^i) = \max\{\frac{\hat{q}_{0.5\alpha}(x^i) - y^i}{\hat{q}_{1-0.5\alpha}(x^i) - \hat{q}_{0.5\alpha}(x^i)},\ \frac{y^i - \hat{q}_{1-0.5\alpha}(x^i)}{\hat{q}_{1-0.5\alpha}(x^i) - \hat{q}_{0.5\alpha}(x^i)}\}. \tag{2.14}$$

Next we follow the process of computing nonconformity scores on all samples in the calibration set and sorting in descending order as $r_{Q-r1}, r_{Q-r2}, \ldots r_{Q-rk}$. Given a prespecified miscoverage rate α, we calculate m = $\lfloor \alpha(k+1) \rfloor$ as the (1-α)th percentile of the descending nonconformity scores. Finally, the resulting CQR prediction interval for a test sample x^j shown below will offer valid marginal coverage.

$$[(1 + r_{Q-rm})\hat{q}_{0.5\alpha}(x^j) - r_{Q-rm}\hat{q}_{1-0.5\alpha}(x^j),\ (1 + r_{Q-rm})\hat{q}_{1-0.5\alpha}(x^j) - r_{Q-rm}\hat{q}_{0.5\alpha}(x^j)]. \tag{2.15}$$

The QR/CQR based schemes pose the greatest amount of difficulty in training due to the computational complexity involved in learning quantiles from such high-dimensional input image data. Nevertheless, we believe that QR/CQR is most effective when dealing with defining/summary in-

put features that sufficiently capture domain knowledge. We will show in the results section that this is indeed true.

3. DEEP NEURAL NETWORKS AND DATASET

3.1 Simulated Dataset

Training deep neural networks like EDGENet requires large datasets of the order of thousands and even millions of data points. To keep up with the demand for a massive dataset, Dr. Chaudhary developed the entire SEM image dataset by simulation. In [2], EDGENet was trained using simulated SEM images of length 1024 pixels and width 64 pixels. Each image represents a magnified view (via electron microscope) of a single line nanostructure of size 0.5 nm x 2 nm. The process of creating these line images consists of first simulating rough edges before generating the final image.

3.1.1 Rough Edge Simulation

The first step in simulating our SEM images is simulating rough line edges, also known as linescans. We do this using the Thorsos method [32] with normal random variables. The Thorsos method is a technique for constructing a simulation of surface height which roughly satisfies a prespecified roughness spectrum and possesses heights and slopes with Gaussian distribution. The Thorsos method is also utilized in other applications and is perhaps more commonly known as the Monte Carlo spectral method. The generated linescans are also subject to a Palasantzas spectral model [33] with the following parameters: Roughness/Hurst exponent α, correlation length ξ, and line edge roughness (LER) σ (i.e. the standard deviation of edge positions).

$$PSD(f) = \frac{\sqrt{\pi}\Gamma(a+0.5)}{\Gamma(a)} * \frac{2\sigma^2\xi}{(1+(2\pi f\xi)^2)^{a+0.5}}, \tag{3.1}$$

where $\Gamma(\cdot)$ is the gamma function defined as follows:

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt. \tag{3.2}$$

Each edge is of length 1024 pixels. We created multiple edges using combinations of values of each of the 3 parameters. The ranges of parameter values are as follows:

- LER σ = [0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8] nm
- Hurst exponent α = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
- Correlation length ξ = [6, 7, . . . , 40] nm.

Hence, in total we have 2520 possible combinations of parameters each of which yields a different rough edge simulation.

3.1.2 Single-line SEM images

The next step involves feeding pairs of generated rough edges into the ARTIMAGEN software [34], [35] to generate a single-line SEM image. The pair of edges (See example in Figure 3.1) act as left and right edges which the ARTIMAGEN software uses to generate lines of width 10 nm and 15 nm. The lines are constructed using ARTIMAGEN's table of curves utility and the resultant images are of dimension 64 pixels by 1024 pixels. ARTIMAGEN is a convenient tool for this task because it is a publicly available and can generate many images in a short amount of time. The edge positions have to be discretized via rounding to match pixel coordinates since we cannot have fractional edge positions. Nevertheless, when dealing with EDGENet as we will see later, rounding is eliminated since it is not a differentiable operation. The position of a line in an image is not fixed and may vary from image to image. We also incorporate edge effects, random backgrounds and Gaussian blur into the images. The parameters used for Gaussian blur are default as recommended by the software: sigma = 0.5, astigmatism angle = 30 degrees, and astigmatism ratio = 1. ARTIMAGEN uses edge effects approximated by an exponential to simulate the brightness of edges relative to the background which has gray level in (0.1, 0.2) and density level 8.

These specifications allow us to generate 10080 SEM images in total as part of the original dataset. On top of this original dataset we apply 10 levels of Poisson noise with electron densities:

Figure 3.1: Examples of simulated pairs of edges generated using Thorsos method.

[2, 3, 4, 5, 10, 20, 30, 50, 100, 200] per pixel in ARTIMAGEN to form our noisy image dataset. The resulting noisy image dataset consists of 100800 SEM images of size 64x1024 pixels. EDGENet was trained on pairs (x^i, y^i) of noisy images x^i and edge positions y^i. The edge positions y^i is a 2x1024 array where each 1x1024 array corresponds to the edge positions/index of either the left or right edge.

3.2 Deep Convolutional Neural Networks

The advent of machine learning and deep learning technologies in recent years has introduced major advancements in image-based problems. In particular, supervised learning focuses on fitting a parametric function that maps an input x^i to an output y^i. Hence, if $y^i = f(x^i)$, supervised learning algorithms attempts to find some approximation $\hat{f}(x^i)$ to this function $f(x^i)$ using some loss function (i.e. the algorithm attempts to search for parameters that minimize this loss function).

Two common examples of loss functions are mean squared error and mean absolute error.

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

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y^i - f(x^i))^2 \tag{3.4}$$

For instance, a linear regression algorithm with MSE loss uses a linear approximation parametrized y, a weight matrix W and a bias vector b that minimizes the MSE loss over an entire dataset.

$$f(x^i) = Wx^i + b \tag{3.5}$$

$$\hat{f} = \min_{W,b} MSE(f, x, y) \tag{3.6}$$

For more complex problems, however, this mapping might be nonlinear in nature, hence, linear modelling does not suffice. Fully connected neural network layers tackle this deficiency by introducing a nonlinear activation function and maintaining a weight matrix that can be modified during training. The most common activation function used in practice is the Rectified Linear Unit (ReLU) which is defined as:

$$f(x) = \max(0, x). \tag{3.7}$$

Hence, the output a single layer fully connected neural network with input x^i and ReLU activation function is given as:

$$f_{FCNN}(x^i) = \max(0, Wx^i + b). \tag{3.8}$$

Aside from ReLU, other activation functions exist such as the tanh function and the logistic sigmoid function. In general, the output of a fully connected given a nonlinear activation function σ is given by:

$$f_{FCNN}(x^i) = \sigma(Wx^i + b). \tag{3.9}$$

By stacking subsequent layers on top of each other, neural networks can realize even more

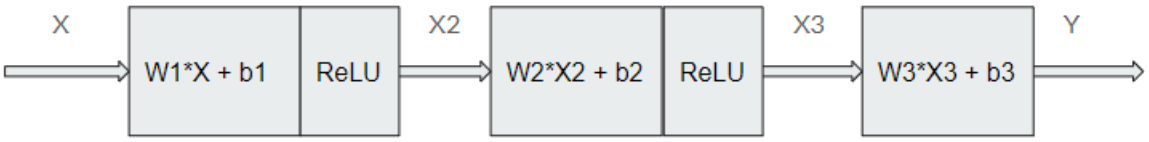

Figure 3.2: A simple neural network with 3 fully-connected layers. Reprinted with permission from [2].

complex nonlinear relationships. For instance, the output of a 3 layer fully connected network with ReLU nonlinearity as shown in Figure 3.2 is as follows:

$$f_{FCNN}(x^i) = W_3 \max(0, W_2 \max(0, W_1 x^i + b_1) + b_2) + b_3. \tag{3.10}$$

Notice that as we begin to stack layers, we realize a composition of nonlinear functions. As a result, this affords us greater representational power but also increases the complexity of the optimization landscape. Consequentially, this optimization problem is not necessarily convex and can be computationally demanding. Most deep neural networks employ some form of stochastic gradient descent via backpropagation [36] to find a global or locally optimal solution to such problems depending on whether the loss function is convex or not.

Deep neural networks as the name implies consist of multiple stacks of layers. These layers are not restricted solely to fully connected layers but also include convolutional layers, recurrent network layers, dropout layers, etc. Advances in computational and storage technologies as well as a global data boom have contributed massively to the growth of the field of deep learning. Deep networks in recent years have achieved human-like performances in many applications, particularly in image based applications and Natural Language Processing (NLP). Deep networks have been shown to be exceedingly successful in such tasks due to their high expressive power (via nonlinearities) and abstraction hierarchy [37].

Deep convolutional neural networks have been applied extensively to image based problems (e.g. facial recognition, object classification etc.) in recent years with much success. Convolutional networks can include different network layer types such as convolutional layers, pooling layers, batch normalization layers, dropout layers and even fully connected layers.

Convolutional layers work similar to a convolution operation. They possess multiple filters each represented by a 3 dimensional kernel that slides across an input image to produce an output image. The height and width of a kernel are usually the same and odd numbered. The most common choice of filter in practice is a 3x3 kernel and the number of filters used in each layer is typically a power of 2. The depth of the kernel is equal to the depth of the input image. For instance, given a convolutional layer with 64 filters, a 5x5 kernel, and accepts input RGB images (i.e. the image has a depth of 3), we would require 64 filters each with dimension 5x5x3 (assuming depth last notation). By sliding a kernel across an entire input image, we can extract local spatial information at different points throughout the image. The term local implies that the output at any point of the image depends not just on a single pixel value but also on its surrounding neighbor pixels as well. The kernel used determines what kinds of features are extracted. For instance, the kernel in Figure 3.3 is used to extract edges. By treating the filters as weights and allowing them to vary during training, we allow the network to automatically learn what kernels are most useful for a given image-based task. CNNs also exhibit spatial hierarchy which implies that lower layers in a network learn low-level features and higher layers learn more sophisticated features by intelligently combining the lower level features. For instance, in a facial recognition problem, lower layers learn to recognize simple features such as edges, circles, dark patches, hair; subsequent layers learn to recognize mouths, eyes, noses and the highest layers ultimately learn to recognize an entire human face [38]. Convolutional layers also utilize an activation functions to increase their expressive power.

Most convolution layers are followed by a pooling layer. Pooling layers are used to reduce the dimensionality of an input image/filter map. There are different types of pooling layers depending

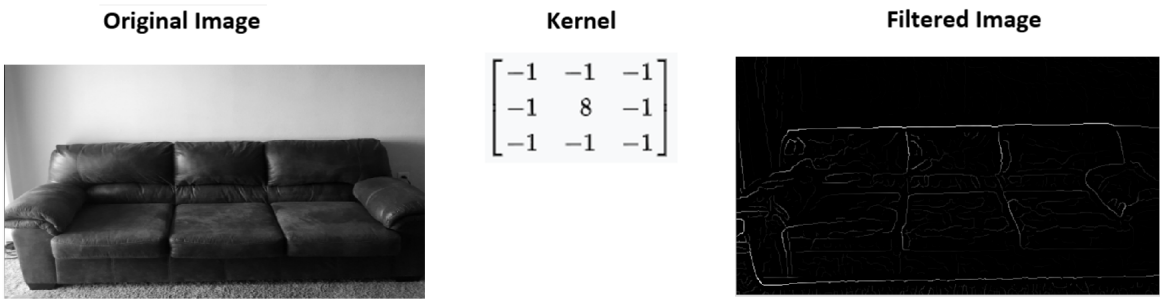

Figure 3.3: Effect of convolution kernel used for edge detection on an input image.

on the type of operation performed by the pooling layer such as max pool, average pool and global max/average pool. Max pool and average pool layers simply reduce a neighborhood of pixels to either its maximum or average pixel value respectively while global pools perform max/average operations over an entire filter map.

Batch normalization layers are used to normalize the output of a convolution or pooling layer. The "Batch" term refers to the nature of stochastic gradient descent which involves training over random batches of the entire dataset at a time. Batch normalization adjusts the mean of the output filter map to 0 and its variance to 1. This operation is necessary to improve training and provide some form of regularization [39].

Dropout layers are used mainly to curb overfitting in convolutional networks. Overfitting occurs when a network learns to perfectly fit a given training dataset but fail to accurately generalize towards unseen test data. Dropout layers randomly turn off a percentage of network units (set output to 0) to prevent interdependencies between units. This forces convolution filters to learn rich and robust features that are salient to the image problem.

3.2.1 EDGENet

EDGENet is a deep convolution neural network that takes in a noisy SEM line image of size 64x1024 pixels as input and outputs an estimate of the left and right edge positions in a 2x1024 array. EDGENet consists of 17 convolution layers with kernel dimensions 3x3x1. Aside from the

last layer, all convolution layers are followed by a batch normalization layer and a dropout layer with a 20% dropout rate. The first 4 layers each have 64 filters, layers 5 to 8 have 128 filters, layers 9 to 12 have 256 filters, and layers 13 to 16 have 512 filters. The final convolution layer has only one filter and outputs a filter map of dimensions 2x1024. Figure 3.4 shows the architecture of EDGENet and the outputs of each layer. EDGENet was trained using MAE loss and the training process was optimized using the Adam [40] optimizer at 0.001 learning rate. In order to support our CP and QR based algorithms, we added an extra layer to EDGENet that directly calculates LER (via a standard deviation operation) from the output edge positions.

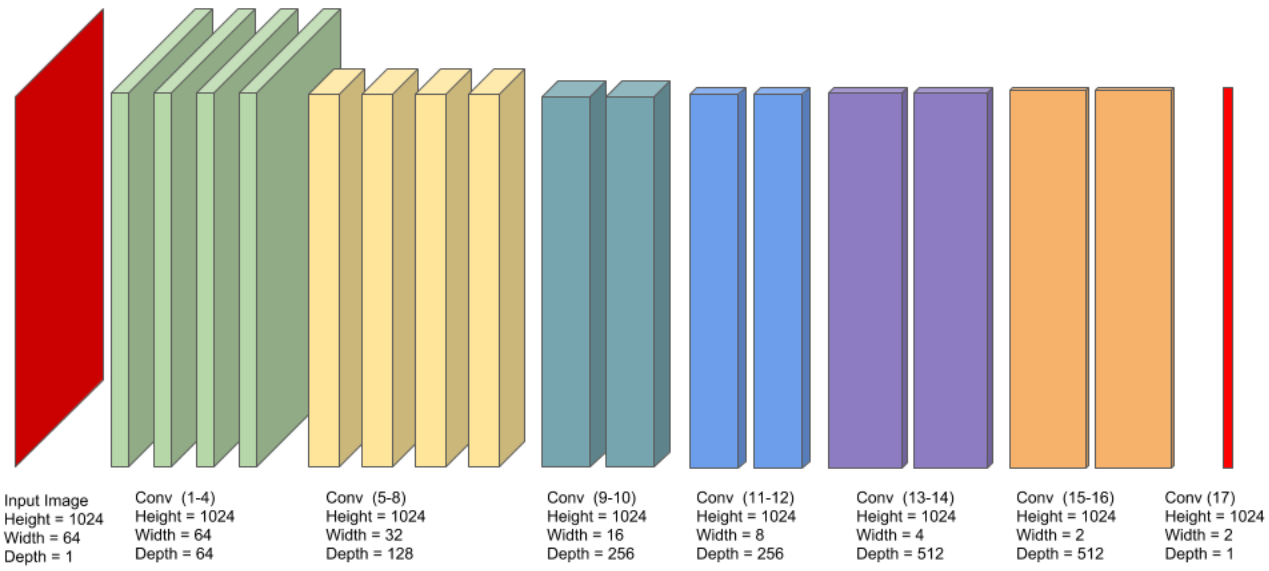

Figure 3.4: EDGENet architecture with 17 convolution layers. EDGENet takes a 64x1024 image as input and outputs 2x1024 array of edge positions. Reprinted with permission from [2].

3.2.2 Normalized Conformal Prediction Approach using Autoencoders

Recall that supervised learning problems attempt to construct a mapping between an input variable x^i and a target variable y^i. Unsupervised learning problems, on the other hand, do not involve a target variable and focus more on finding hidden patterns among input data. Autoencoders are a special type of unsupervised learning algorithm that are typically designed to reconstruct their input. Autoencoder architectures are symmetrical in design and consist of an encoder, bottleneck/hidden layer and decoder. They are usually applied to image inputs and are sometimes used for image denoising. Autoencoders learn robust latent representations that typically serve as compressed versions of the input image. Note that these latent representations do not necessarily have to be smaller than the original image but can also be larger. Recall that in our normalized CP experiments we first employ an auotencoder to compress the input image x^i from 64x1024 to 1x64. The compressed representation $p(x^i)$ attempts to capture the difficulty of the input image which is then fed to a second neural network $q(\cdot)$ that attempts to predict -$\ln|y^i - f(x^i)|$ where y^i and $f(x^i)$ are the actual and estimated LER values respectively. Our autoencoder has 12 convolutional layers each with kernel dimensions 3x3x1. Layers 1-4 make up the encoder, layers 5-7 make up the bottleneck and layers 8-11 make up the decoder portion of the network. The 1st and 11th layers have 64 filters, the 2nd and 10th layers have 128 filters, the 3rd and 9th layers have 256 filters, and the 4th and 8th layers each have 512 filters. The intermediate layers from 5 to 7 have just 1 filter and aid in compression. The final layer also contains a single filter. All convolutional layers utilize the ReLU activation function. The encoding and decoding layers are all followed by batch normalization and dropout layers with 20% dropout rate. Each decoding layer has a preceding upsampling layer. Upsampling layers act opposite to pooling layers and expand the width and height of filter maps by repeating pixel values.

Our second model $q(\cdot)$ is a fully connected neural network with 4 layers that accepts as input a 1x64 vector $p(x^i)$ and outputs an estimate of the error -$\ln|y^i - f(x^i)|$ for both edges. The first layer has 128 units and the second and third layers have 64 units with L2 regularization. The final layer has 2 units which output estimates of the error for both edges. Both network models were

trained with MSE loss and Adam optimizer at 0.001 learning rate.

3.2.3 Quantile Regression Approach using Transfer Learning

Our initial approach to quantile regression is underpinned by the pretrained EDGENet. We modified EDGENet by adding an extra layer that simply computes the standard deviation of both edge positions to directly compute LER estimates for both edges. This modification is necessary to allow us directly learn LER quantiles. This approach is meaningful because EDGENet is already trained to estimate edge positions. Hence, by slightly adjusting the target of our optimization problem, we can adapt the pre-learned features to train quantile regressors end-to-end. To this end, we employ a transfer learning type approach where we freeze the weights of lower layers of EDGENet so that salient prior information learned in lower layers are preserved and effectively utilized during training. To be specific, only the weights of the first 15 layers are frozen while the rest of the network is trained using pinball loss. We trained 2 separate networks for the lower and upper quantiles. This is necessary to avoid quantile crossing [28]. Quantile crossing occurs sometimes when trying to train 2 different quantile levels on the same network and the lower quantile estimates ends up being higher than the upper quantile estimate. Additionally, QR networks suffer heavily from undercoverage. Thus, in addition to separating quantiles, we also employ the smoothed pinball loss on our upper quantile networks to slightly alleviate these issues.

3.2.4 Quantile Regression Idealized Case

Despite its aforementioned shortcomings, QR remains a powerful tool for delineating predictor-response variable relationships. We argue that QR is significantly more effective when applied to a few summary input variables that sufficiently encapsulate domain knowledge. To demonstrate this, we constructed an ideal experiment to construct QR prediction intervals for the left edge at 10% miscoverage rate (90% coverage). As opposed to using a 64x1024 SEM image as input to our QR network, our ideal scheme uses 7 key parameters, of which 6 are involved in the generation of

a noisy SEM image. Hence, our input is a 1x7 vector consisting of the 3 Palasantzas parameters, line width, line position, noise density and EDGENet's estimate of the left edge LER. We designed lower and upper quantile regression neural networks to construct QR prediction intervals for LER. Both networks have 6 layers; the first 2 layers have 32 units, layer 3, 4 and 5 have 16, 8 and 4 units respectively. All layers use a ReLU nonlinear activation function and the entire network is trained with pinball loss for about 90 epochs and training batch-size of 16.

3.2.5 Experiments and Results

For each of the models discussed so far we will now examine their performance in 4 different sets of experiments. The experiments will summarize their coverage and interval width statistics for both the left and right edges each at 10% and 5% miscoverage rate (90% and 95% coverage). For the left edge at 90% coverage, we will also compare the results of the idealized QR experiment with other models to illustrate the benefit of combining QR with domain knowledge. Finally, we will also introduce five ensemble schemes which combine multiple models to improve performance.

It is necessary to study both left and right edges separately as a result of systematic differences that are introduced in the simulation procedure. ARTIMAGEN applies a Gaussian blur effect at a 30 degree angle to the horizontal. This results in astigmatic effects which occur when lines in an image begin to drift towards the right at 30 degrees to the horizontal. Due to the positioning of the right edge with respect to the right boundary of the image, these effects are markedly more random and pronounced than in the left edge. Furthermore, the rounding operation used on edge positions inevitably introduces more variations between both edges. Overall these differences significantly benefit our experiments by enhancing their diversity and richness.

Recall that the entire dataset consists of 100800 noisy SEM images with corresponding LER values. The pixel values in images are normalized to [0,1]. We divide the entire dataset into training, calibration and test/validation sets. The training set comprises of the same 89,280 (input, output) pairs which were used to train EDGENet. The remaining 11520 images correspond to cor-

relation length values ξ in [10, 20, 30, 40] and make up the calibration and test sets. To maintain exchangeability we randomly split these 11520 images into 2 equal sets of 5760 such that all noisy versions of an original image are in the same set. (Recall that for every original image we generate 10 corresponding noisy images).

The models in this work were developed using Python's Keras library which is an extensive framework for machine learning and deep learning tasks. They were trained on compute systems running an Intel Xeon E5-2680 v4 processor with 2.4 GHz clock speed and Tesla K80 GPU. Additionally, we performed hyperparameter tuning on our network parameters such as batch size, optimizer, dropout rate, epochs, and learning rate. We found that a combination of the Adam optimizer with a learning rate of 0.001 for stochastic gradient works best. We also used a dropout rate of 20% and a batch size of 18 due to compute and memory restrictions. In general, almost all models required at most about 4 epochs for training after which we began to observe overfitting.

As we begin to analyze the results, we must first revisit the criteria for evaluating each model's performance. First, the miscoverage rate must be close to the prespecified value. Once this is satisfied, we prefer models that produce smaller interval widths on average. Tables 3.1 and 3.2 summarize the coverage and interval width statistics for both the left and right edges respectively at 10% miscoverage rate. Similary, Tables 3.3 and 3.4 summarize the coverage and interval width statistics for both the left and right edges respectively at 5% miscoverage rate. Observe that the coverage rates in Table 3.2 are strictly lower than their counterparts in Table 3.1. Furthermore, the interval widths in Table 3.2 are significantly higher than those in Table 3.1. This is likely due to systematic differences between the left and right edges. We believe that our models face increased difficulty in handling the increased randomness in the right edge. The examples in [14] illustrate the undercoverage of quantile regression networks. This phenomenon is also illustrated in all of our QR experiments. Nevertheless this issue is easily alleviated by conformalization. Our basic conformal prediction scheme yields decent coverage and small interval widths but the lack of adaptability of its interval widths offers no information about the difficulty of an input sample. Although, Normalized CP produces larger interval widths on average, notice that it yields smaller

minimum intervals than any other scheme. In particular, normalized CP yields smaller intervals than ordinary CP 60% of the time for the left edge and 45% of the time for the right edge. Similarly, it produces larger maximum intervals than any other scheme, hence, we conclude that normalized CP is the most adaptable scheme. Table 3.1 also shows the results of our ideal QR experiment. While we avoid making the unfair comparison with other models, it illustrates the potential of QR on fewer inputs that capture domain knowledge.

Our Normalized CP model takes a maximum time of about 20.7 ms to process an input sample while ordinary CP takes a maximum of about 5.6 ms. CQR and CQR-r consume a maximum processing time of about 499.3 ms and 504.1 ms respectively.

Table 3.5 shows the result of our five ensemble schemes. Recall that normalized CP produces the smallest and the largest interval widths. CQR and CQR-r avoid these extremes while maintaining decent coverage. Our ensemble schemes attempt to leverage the smallest intervals for both easy and difficult input samples from both models and thereby reduce the average interval width. Ensemble scheme 1 picks the shorter interval from normalized CP and CQR while Ensemble 2 picks the shorter interval from normalized CP and CQR-r. Ensemble 3 uses the the shortest interval among normalized CP, CQR and CQR-r. All ensembles are designed for a 10% miscoverage rate. As Table 3.5 illustrates, these first 3 ensemble schemes achieve shorter interval widths than even ordinary CP but suffer from slightly less coverage. Furthermore, consider that given any two models with miscoverage rates α_1 and α_2, the combined miscoverage rate of an their ensemble is at least $\max\{\alpha_1, \alpha_2\}$ and at most $\alpha_1 + \alpha_2$. Ensemble 4 picks the shorter interval prediction between normalized CP at miscoverage rate α_1 = 0.1 and CP at miscoverage rate α_2 = 0.05. Ensemble 5 picks the shorter interval prediction between normalized CP at miscoverage rate α_1 = 0.05 and CP at miscoverage rate α_2 = 0.1. From the results in Table 3.5, we observe that both Ensemble 4 and Ensemble 5 improve on the interval widths of prior schemes in Tables 1 and 2 as well as those of the first 3 ensemble schemes while still maintaining decent coverage.

Method	Coverage (%)	Average interval length (nm)	Interval length (nm)		Ratio of interval length to original LER	
			Minimum	Maximum	Minimum	Maximum
Conformal Prediction (CP)	90.22	0.135	0.135	0.135	0.065	0.487
Normalized CP	90.15	0.153	0.006	1.588	0.008	2.716
Quantile Regression	85.94	0.197	0.033	0.496	0.056	0.521
CQR	89.55	0.215	0.052	0.515	0.071	0.570
CQR-r	89.51	0.225	0.038	0.566	0.065	0.595
Ideal Experiment	89.77	0.103	0.032	0.239	0.028	0.501

Table 3.1: Coverage and interval length statistics for the LER of the left edge when the miscoverage rate α equals 0.1. Reprinted with permission from [4].

Method	Coverage (%)	Average interval length (nm)	Interval length (nm)		Ratio of interval length to original LER	
			Minimum	Maximum	Minimum	Maximum
Conformal Prediction (CP)	89.22	0.186	0.186	0.186	0.079	0.684
Normalized CP	88.96	0.241	0.017	1.109	0.012	2.141
Quantile Regression	83.26	0.224	0.047	0.573	0.055	0.731
CQR	88.61	0.251	0.074	0.600	0.079	0.823
CQR-r	88.26	0.265	0.056	0.677	0.065	0.864

Table 3.2: Coverage and interval length statistics for the LER of the right edge when the miscoverage rate α equals 0.1. Reprinted with permission from [4].

Method	Coverage (%)	Average interval length (nm)	Interval length (nm)		Ratio of interval length to original LER	
			Minimum	Maximum	Minimum	Maximum
Conformal Prediction (CP)	95.45	0.199	0.199	0.199	0.095	0.715
Normalized CP	95.33	0.214	0.009	2.220	0.010	3.798
Quantile Regression	88.75	0.292	0.091	0.776	0.110	1.122
CQR	95.40	0.333	0.132	0.818	0.139	1.264
CQR-r	95.50	0.350	0.108	0.932	0.131	1.346

Table 3.3: Coverage and interval length statistics for the LER of the left edge when the miscoverage rate α equals 0.05. Reprinted with permission from [4].

Method	Coverage (%)	Average interval length (nm)	Interval length (nm)		Ratio of interval length to original LER	
			Minimum	Maximum	Minimum	Maximum
Conformal Prediction (CP)	96.89	0.233	0.233	0.233	0.112	0.839
Normalized CP	94.41	0.313	0.022	1.441	0.016	2.781
Quantile Regression	92.60	0.358	0.103	0.912	0.159	1.122
CQR	95.45	0.385	0.131	0.940	0.182	1.216
CQR-r	95.26	0.340	0.115	1.017	0.177	1.251

Table 3.4: Coverage and interval length statistics for the LER of the right edge when the miscoverage rate α equals 0.05. Reprinted with permission from [4].

Method	Coverage (%)	Average interval length (nm)	Interval length (nm)	
			Minimum	Maximum
Ensemble 1	87.66	0.133	0.006	0.431
Ensemble 2	86.82	0.134	0.006	0.471
Ensemble 3	86.89	0.133	0.006	0.431
Ensemble 4	88.92	0.131	0.006	0.199
Ensemble 5	89.22	0.127	0.009	0.135

Table 3.5: Coverage and interval length statistics for 5 ensemble schemes for the LER of the left edge. For the first 3 ensemble strategies the miscoverage rate α of all models is approximately 0.1. The Ensemble 1 scheme applies the minimum length interval from Normalized Conformal Prediction and CQR. The Ensemble 2 scheme applies the minimum length interval from Normalized Conformal Prediction and CQR-r. The Ensemble 3 scheme applies the minimum length interval from normalized CP, CQR, and CQR-r. Ensemble 4 applies the minimum interval length from normalized CP at 0.1 miscoverage rate and CP at 0.05 miscoverage rate. Finally, Ensemble 5 applies the minimum interval length from normalized CP at 0.05 miscoverage rate and CP at 0.1 miscoverage rate. Reprinted with permission from [4].

4. IMPROVED APPROACH TO NORMALIZED CONFORMAL PREDICTION AND CONFORMAL QUANTILE REGRESSION

So far we have focused on constructing prediction intervals for LER directly from input noisy SEM images. In this section, we will present a different approach which uses an input "noise" image. We define a noise image as the absolute difference between a noisy image x^i and a denoised version of the noisy image x_D^i.

$$x_N^i = |x^i - x_D^i| \tag{4.1}$$

Figure 4.1 shows an example noise image. The denoised image $\hat{x^i}$ is obtained using SEMNet [2]. SEMNet is a 17 layer convolutional network that takes as input a 64x1024 noisy SEM image and outputs a denoised version of the same image (See Figure 4.2). The rationale behind the noise image is that we believe that there is some relationship between the amount of Poisson noise in an input image and the difficulty in constructing a valid LER prediction interval for that input image. Based on this new approach, we developed 3 new normalized CP models and 3 new QR/CQR based models.

4.1 New Normalized Conformal Prediction Models

In this section we discuss our new normalized CP models: NCPMaxPoolNet, NCPLSTMNet, and NCPBLSTMNet. These models accept a 64x1024 noise image as input and produce smaller intervals in general than previous models. All 3 models start with the same convolutional network layers which output multiple filter maps that are fed to a feature extraction layer that is unique to each model. Finally the extracted features are fed to fully connected layers that output an estimate of the error measure -$\ln |y^i - f(x^i)|$. This initial CNN consists of 3 convolution layers with 64, 128, and 256 filters each with a 3x3 kernel, 2x2 stride length, and ReLU nonlinearity. Each layer is followed by a batch normalization layer as well as a dropout layer with 2% dropout rate. The final layer outputs 256 64x128 filter maps.

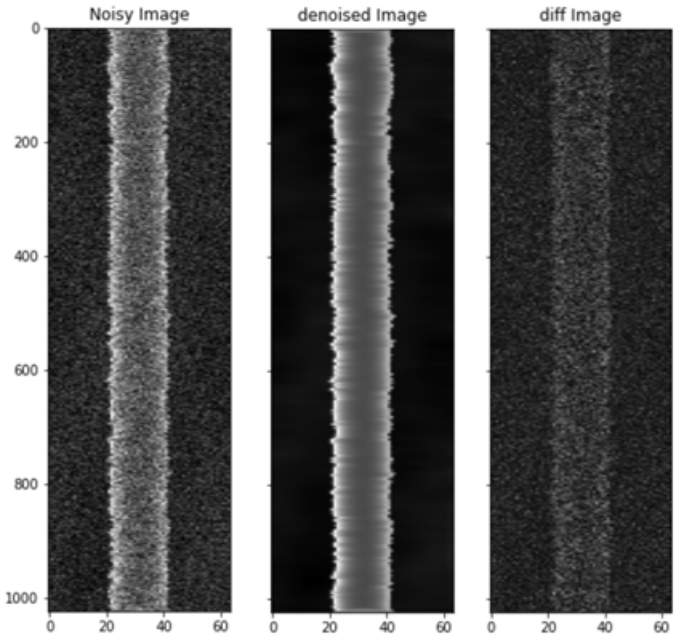

Figure 4.1: Noise image labelled as diff image and its corresponding noisy and denoised images. The noise/diff image is simply the absolute difference of the noisy and denoised images. The denoised image is obtained from SEMNet.

Following the CNN, the next set of layers serve as a dimensionality reduction for the output filter maps. By so doing, they extract salient features necessary for estimation. NCPMaxPoolNet employs a basic global maxpool layer as a feature extraction mechanism. Recall that a global maxpool layer simply outputs the maximum value in each filter map. Thus the 64x128x256 output from the CNN is reduced to a 1x1x256 vector.

NCPLSTMNet and NCPBLSTMNet employ an additional convolution layer with 64 units and a maxpool layer after the intitial CNN layers to reduce the CNNs' output filter map dimension from 64x128x256 to 8x16x64. Subsequently, the reduced output is reshaped to 64x128 (i.e. 64 128-length vectors) and fed as input to a Long Short Term Memory (LSTM) [41] which etracts useful information. LSTMs are an special type of Recurrent Neural Networks (RNN). RNNs are a type of deep networks that are specially designed to deal with sequential data (i.e. $x^i = x_1^i, x_2^i, \ldots, x_T^i$). RNNs have been successfully applied to sequential problems like speech recognition due to their ability to store and persist temporal information between nearby data elements using a hidden state.

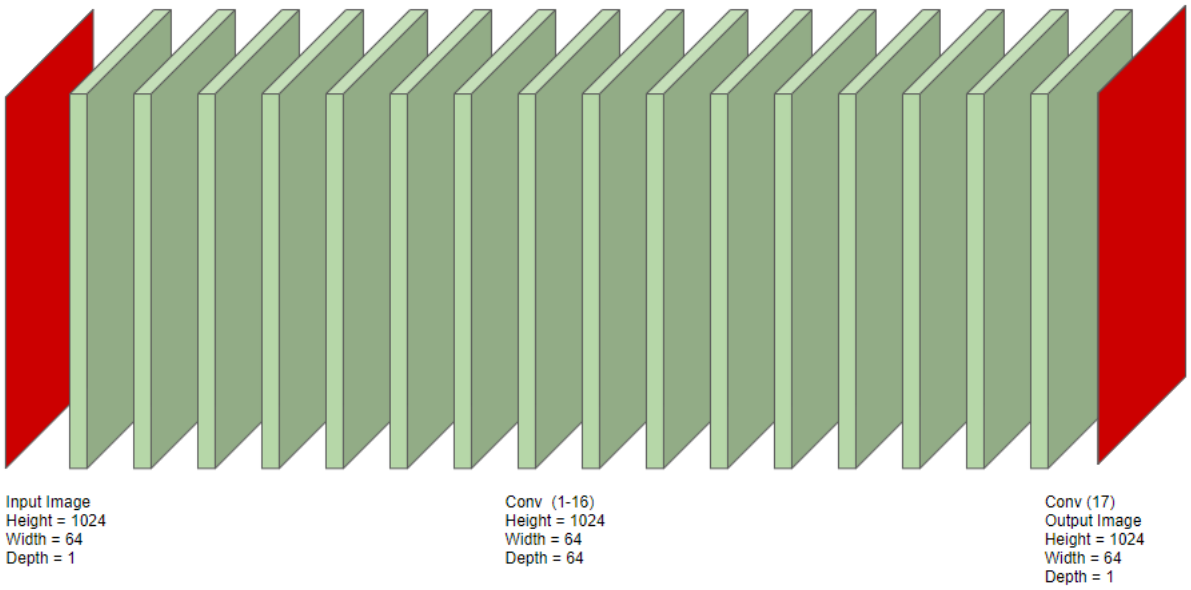

Figure 4.2: SEMNET architecture with 17 convolution layers. SEMNET takes a 64x1024 noisy SEM image as input and outputs a 64x1024 denoised SEM image. Reprinted with permission from [2].

Figure 4.3 shows the architecture of an unfolded RNN layer. Observe that the hidden state transfers temporal information from one element/time-step to the next. Furthermore, the hidden state serves as output at each time step making RNNs ideal for NLP problems such as predicting the following word at different points in a sentence. While RNNs are effective at storing short-term memory they lack the ability to persist long-term information over the length of a sequence. This arises as a result of the vanishing gradient problem. Vanishing gradient is a phenomenon that occurs during training where the continuous derivatives used for weight updates become arbitrarily small and thus stifle the update of network parameters. LSTMs solve this problem utilizing 3 information gates to preserve salient information across an entire sequence in a structure known as a cell memory C_t. Figure 4.4 illustrates that information from the hidden state is either added to or removed from the cell memory via the 3 gates: forget gate f_t, input/update gate i_t, and output gate o_t.

$$
\begin{aligned}
\overline{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
C_t &= f_t * C_{t-1} + i_t * \overline{C}_t \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
\tag{4.2}
$$

$W_k \in R^{N\text{x}d}$ and $b_k \in R^N$ are weight and bias parameters for gate k where N represents the number of LSTM units. $\alpha(\cdot)$ represents the sigmoid activation function (i.e. $(1 + e^{-x})^{-1}$) and ranges strictly from 0 to 1. $C_t \in R^N$ is the cell memory and is propagated across each LSTM cell unit. $\bar{C}_t \in (-1, 1)^N$ is the cell memory activation vector and consists of information from both the hidden state of the previous position in a sequence and the current input element of that sequence. This information is passed through a tanh activation function and is added to the current cell memory C_t via the input gate i_t. The input gate acts as a multiplier and controls how much of this information is added to the C_t. Essentially, all 3 gates are linear combinations of the previous hidden state and current input element that is passed through a sigmoid activation function which constrains its output value to [0,1]. Hence, each gate essentially acts as a multiplier. The forget gate controls how much of the previous cell memory is retained in the current cell memory and the output gate extracts key information from the current cell memory to construct the current hidden state. Overall, this complex architecture allows LSTMs to efficiently propagate salient information throughout a sequence in the forward direction. In NCPLSTMNet we use an LSTM layer with 64 cell units to efficiently extract key information and reduce the prior 64x128 output to a 64 length vector. Our results demonstrate the strength of this approach.

LSTMs can effectively propagate information in one direction but in some applications such as

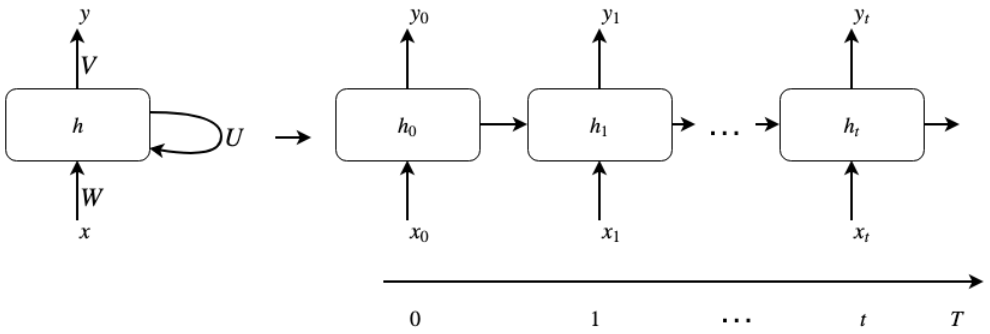

Figure 4.3: Architecture of unfolded Recurrent Neural Network for sequential data.

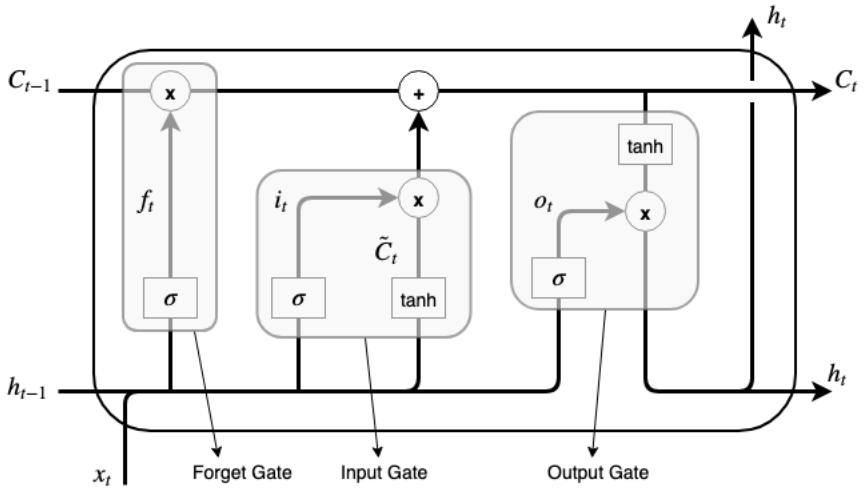

Figure 4.4: A Long Short Term Memory cell showing input, forget and output gates as well as hidden state and cell memory.

handwriting recognition and language translation it is necessary to propagate information in both the forward and backward directions. Bidirectional Long Short Term Memory (BLSTM) networks include an extra LSTM layer that works in parallel but processes an input sequence in the reverse direction. While BLSTMs essentially double the number of parameters in the LSTM layer, they have outperformed LSTMs in numerous applications [42]. NCPBLSTMNet replaces the LSTM layer in NCPLSTMNet with a BLSTM layer and achieves even better performance.

The last section of these networks is made up of fully-connected layers. Note that NCPLSTM-

Net and NCPBLSTMNet were developed after NCPMaxPoolNet and hence are slightly superior in terms of design. NCPMaxPoolNet has 2 branches of fully connected layers that cater to both left and right edges. Each branch has 4 fully-connected layers that are made of 64, 16, 4 and 1 units respectively and each employs the ReLU activation function. Thus a single NCPMaxPoolNet model outputs error estimates for both edges. However, due to systematic differences in both edges it is more effective to train separate models for each edge. To this end we designed NCPLSTMNet and NCPBLSTMNet to handle a single edge at a time. Similar to NCPMaxPoolNet, they have 4 fully-connected layers with 32, 8, 4 and 1 units respectively and each employs the LeakyReLU activation function except for the last layer which uses ReLU. The LeakyReLU function is defined as:

$$f(x, \alpha) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0, \end{cases}$$

where α is a preselected hyperparameter. LeakyReLU has been shown to improve training performance and curb overfitting [43]. All 3 networks were trained using MAE loss. All 3 normalized CP models achieve better performance than the previous schemes that involved first reducing the dimension of the input image with an autoencoder and then running a separate neural network for estimation. The reasons for this are:

- They learn dimensionality reduction and error estimation in an end-to-end style.
- Noise images potentially offer more information to estimate error (-$\ln |y^i - f(x^i)|$).

4.2 New Quantile Regression and Conformalized Quantile Regression Models

A major issue with the previous approach to QR was the complexity of EDGENet and the high-dimensionality of the input image. As we demonstrated with the ideal experiment, QR performs better with fewer summary inputs. Our new approach to QR is inspired by the success of the new Normalized CP models. Instead of accepting an image as input, we use combinations of EDGENet's prediction and the estimates of the error measure -$\ln |y^i - f(x^i)|$ from the new

normalized CP models. We developed 3 such models: QRMaxpoolNet, QRMaxpool-LSTMNet, and QRMaxpool-BLSTMNet. QRMaxpoolNet takes 2 inputs: the LER estimate from EDGENet and the error measure estimate from NCPMaxPoolNet. QRMaxpool-LSTMNet takes 3 inputs: the LER estimate from EDGENet, the error measure estimate from NCPMaxPoolNet and the error measure estimate from NCPLSTMNet. QRMaxpool-BLSTMNet takes 3 inputs: the LER measure estimate from EDGENet, the error measure estimate from NCPMaxPoolNet and the error measure estimate from NCPBLSTMNet. For each model, we train separate networks for each edge. The conformal counterparts of these models are called: CQRMaxpoolNet, CQRMaxpool-LSTMNet, and CQRMaxpool-BLSTMNet.

Each model has two symmetrical branches (one for each quantile level) each with two layers; a hidden layer and an output layer. The hidden layer is a simple fully-connected layer with as many units as the number of inputs. The output layer has one unit that estimates the respective quantile level. The model features a residual connection between the EDGENet LER input and the output layer (See Figure 4.5). Hence, the model essentially learns to adjust the LER estimate to construct lower and upper bounds for a prediction interval. Note that aside from EDGENet's LER prediction, the other inputs to our QR models are estimates of the error measure (-$\ln |y^i - f(x^i)|$) from our new normalized CP models.

4.3 Experiments and Results

In this section we compare the performance of our new schemes to that of the old schemes. To train the normalized CP models, we extract noise images and compute the error measure (-$\ln |y^i - f(x^i)|$) for each corresponding noisy image in our dataset using the procedure described in Section 4.1. This forms our input and output pairs. Similar to the previous set of experiments, we divide the entire dataset of 100800 (input, output) pairs into training, calibration and test/validation sets. The training set comprises of 89,280 (input, output) pairs and the remaining 11520 pairs correspond to images with correlation length values ξ in [10, 20, 30, 40] and make up the calibration and test sets. To maintain exchangeability we randomly split these 11520 pairs into 2 equal sets

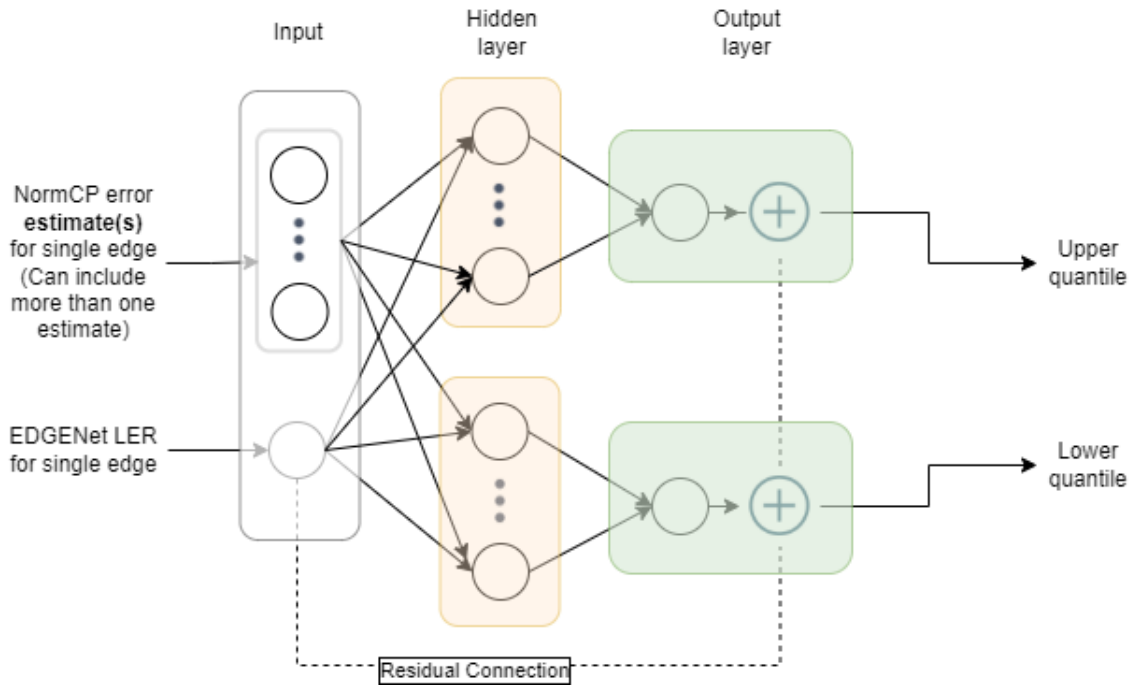

Figure 4.5: Architecture of Quantile regression networks.

of 5760 such that all pairs corresponding to an original image are in the same set. (Recall that for every original image we generate 10 corresponding noisy image which are then used to derive a noise image). Each QR model depends on its constituent normalized CP models for training (e.g. QRMaxpool-LSTMNet depends on NCPMaxPoolNet, NCPLSTMNet, and EDGENet for input). To train our QR models we simply run its constituent normalized CP models on each derived noise image in the entire dataset and use the output error measure estimates as input for training. Additionally, we incorporate the output LER from running EDGENet on a noisy image as part of the input. Hence, the input to each QR model consists of the EDGENet's LER prediction and the error measure estimates of its constituent normalized CP models. The training output is simply the actual corresponding LER value. Having defined our (input, output) pairs, we then partition the entire dataset as we did for our normalized CP models.

Each model was developed tested using Python's Keras lilbrary on compute systems running an Intel Xeon E5-2680 v4 processor with 2.4 GHz clock speed and Tesla K80 GPU. Network parameters such as batch size, optimizer, dropout rate, epochs, and learning rate were fine tuned

to optimize training performance. We use the Adam optimizer with 0.001 learning rate and limit the number of training epochs for all models except QRMaxpoolNet to a maximum of 3 to avoid overfitting. We used a low dropout rate of 2% in the newer models as well as a batch size of 18.

Tables 4.1 and 4.2 demonstrate the coverage and interval width results of our new models for both the left and right edge at a 10% miscoverage rate. Tables 4.3 and 4.4 demonstrate the coverage and interval width results of our new models for both the left and right edge at a 5% miscoverage rate. First, observe that almost all of the new schemes outperform the old schemes. This illustrates that the noise image carries special information that is salient to constructing prediction intervals. As expected, our QR schemes undercover significantly but produce surprisingly tight prediction intervals. This is desirable because the conformalized versions of these models also yield tight intervals while achieving decent coverage. Furthermore, the performance of our QR/CQR based schemes validates our claim that QR-based schemes work effectively with fewer summary parameters. Our QR schemes act similarly to ensemble schemes by expertly combining the performance of our normalized CP models to produce even tighter intervals. One anomaly in Table 4.3 is that the coverage of the conformalized version of QRMaxpool-LSTMNet is lower than that of QRMaxpool-LSTMNet itself. While this is rare, it arises due to the errors in the nonconformity calculation for CQR being predominantly negative, hence, the conformalized procedure effectively reduces the length of its intervals leading to less coverage. It is also worth noting that QRMaxpool-BLSTMNet and CQRMaxpool-BLSTMNet at 0.1 miscoverage rate achieve interval widths of 0.096 and 0.117 respectively on average which are close to the 0.103 average interval width of the ideal QR experiment in the previous results section. Hence, it is reasonable to assume that the ideal QR experiment is not necessarily a benchmark and can be surpassed with even better modelling techniques.

Notice in Tables 4.1 - 4.4 that our newer QR/CQR models struggle with coverage at both 90% and 95%. One likely explanation for this is that our new QR models are unavoidably limited by the quality of predictions of the new normalized CP models as well as EDGENet (since the outputs of these models are fed as input to the QR models). While there is no fixed acceptable range for

coverage, we prefer coverage to be within +/-1%. Beyond +/-2%, coverage performance becomes significantly less desirable.

Despite the improved performance, our normalized CP models are burdened with the overhead of computing a noise image for each input noisy image. On average, the process of extracting a noise image with SEMNet takes about 1430 ms (execution times are measured on CPU). Furthermore, our normalized CP models (NCPMaxPoolNet, NCPLSTMNet, NCPBLSTMNet) also require an additional 76 ms, 377 ms, and 412 ms respectively to process an input noise image on average. This overhead is compounded even further for our QR based models which not only require noise image extraction but also additional processing time from their constituent normalized CP models. Without this overhead, each QR neural network model consumes approximately 90 ms to process an input. With overhead, our most complex QR scheme (QRMaxpool-BLSTMNet) can consume as much as 1984 ms of execution time from processing an input noisy image to constructing a prediction interval. There are two key points to note here. The first point is that the reported execution time with overhead does not include the processing time of the base regression algorithm (EDGENet) since the QR schemes are simply applied on top of the base regression algorithm. The second is that we can considerably reduce our overall execution time by replacing SEMNet with LineNet1 [44]. LineNet1 takes an input noisy SEM line image and estimates both a denoised version of the noisy image as well as its edge positions. This eliminates the need for both SEMNet and EDGENet and makes our overall procedure more efficient. These execution times are all significantly higher than the original times reported in Section 3.2.5 and demonstrate the trade off between time complexity and performance.

Method	**Coverage (%)**	**Average interval length (nm)**
Conformal Prediction (CP) from [4]	90.22	0.135
Normalized CP from [4]	90.15	0.153
NCPMaxPoolNet	89.36	0.127
NCPLSTMNet	90.30	0.130
NCPBLSTMNet	90.43	0.124
QRMaxpoolNet	81.39	0.146
CQRMaxpoolNet	87.97	0.169
QRMaxpool-LSTMNet	71.39	0.094
CQRMaxpool-LSTMNet	88.18	0.134
QRMaxpool-BLSTMNet	79.60	0.096
CQRMaxpool-BLSTMNet	88.14	0.117

Table 4.1: Coverage and interval length statistics for the LER of the left edge when the miscoverage rate α equals 0.1.

Method	**Coverage (%)**	**Average interval length (nm)**
Conformal Prediction (CP) from [4]	89.22	0.186
Normalized CP from [4]	88.96	0.241
NCPMaxPoolNet	89.58	0.242
NCPLSTMNet	89.22	0.186
NCPBLSTMNet	89.24	0.181
QRMaxpoolNet	84.83	0.156
CQRMaxpoolNet	88.58	0.168
QRMaxpool-LSTMNet	65.59	0.077
CQRMaxpool-LSTMNet	89.01	0.128
QRMaxpool-BLSTMNet	75.64	0.096
CQRMaxpool-BLSTMNet	89.35	0.133

Table 4.2: Coverage and interval length statistics for the LER of the right edge when the miscoverage rate α equals 0.1.

Method	**Coverage (%)**	**Average interval length (nm)**
Conformal Prediction (CP) from [4]	95.45	0.199
Normalized CP from [4]	95.33	0.214
NCPMaxPoolNet	94.57	0.160
NCPLSTMNet	95.43	0.190
NCPBLSTMNet	95.33	0.171
QRMaxpoolNet	91.89	0.142
CQRMaxpoolNet	92.45	0.144
QRMaxpool-LSTMNet	93.58	0.171
CQRMaxpool-LSTMNet	93.04	0.169
QRMaxpool-BLSTMNet	85.83	0.135
CQRMaxpool-BLSTMNet	93.28	0.165

Table 4.3: Coverage and interval length statistics for the LER of the left edge when the miscoverage rate α equals 0.05.

Method	**Coverage (%)**	**Average interval length (nm)**
Conformal Prediction (CP) from [4]	96.89	0.233
Normalized CP from [4]	94.41	0.313
NCPMaxPoolNet	94.43	0.312
NCPLSTMNet	94.38	0.233
NCPBLSTMNet	94.31	0.226
QRMaxpoolNet	88.13	0.121
CQRMaxpoolNet	93.49	0.143
QRMaxpool-LSTMNet	91.82	0.152
CQRMaxpool-LSTMNet	93.75	0.162
QRMaxpool-BLSTMNet	90.74	0.139
CQRMaxpool-BLSTMNet	93.54	0.152

Table 4.4: Coverage and interval length statistics for the LER of the right edge when the miscoverage rate α equals 0.05.

5. SUMMARY AND CONCLUSIONS

For the semiconductor industry to take full advantage of digital transformation, it is necessary to fully leverage deep learning technologies. Deep learning can greatly accelerate decision making and hence, improve device yield as well as profits in the manufacturing process. Major players in the industry like KLA-Tencor have already begun to leverage deep learning techniques to develop machines for defect/fault detection in the fabrication process at a commercial scale. In order to encourage industry wide adoption of these technologies it is necessary to incorporate uncertainty estimation and coverage guarantees into deep learning models. In this work we focused mainly on networks that process SEM images as input, however, these techniques can also be extended to many other kinds of deep learning type problems. Aside from prediction intervals, other techniques also exist for uncertainty quantification. A lot of research effort is currently being devoted to developing new methods to achieve this. Furthermore, our results in Chapter 4 demonstrate the importance of modelling. Advanced techniques are likely to produce better guarantees of coverage and foster trust in deep learning techniques.

5.1 Further Study

Attention based models are a hot topic of research in the deep learning community and have already produced outstanding results that are the state of the art in both natural language processing as well as computer vision problems. Our next steps would be to leverage attention in our noise image models. At the time of writing, we have successfully built and tested a custom implementation of squeeze and excitation networks in our experiments and realized an average interval width of 0.12 on the left edge while maintaining valid coverage. This is clearly a step in the right direction and leaves plenty of room for improvement.

REFERENCES

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

[2] N. Chaudhary, S. A. Savari and S. S. Yeddulapalli, "Line roughness estimation and Poisson denoising in scanning electron microscope images using deep learning," *J. Micro/Nanolith. MEMS MOEMS*, **18**(2), 024001, 2019.

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

[4] I. I. Akpabio, S. A. Savari, "Uncertainty quantification of machine learning models: on conformal prediction," *J. Micro/Nanopattern., Mater., Metrol.*, vol. 20, no. 4, pp. 041206-1–041206-14, Oct-Dec. 2021.

[5] S. Göke, K. Staight, and R. Vrijen, "Scaling AI in the sector that enables it: Lessons for semiconductor-device makers," Article, McKinsey & Company, April 2, 2021. https://www.mckinsey.com/industries/semiconductors/our-insights/scaling-ai-in-the-sector-that-enables-it-lessons-for-semiconductor-device-makers

[6] Nitesh Mirchandani, "How is AI Transforming the Semiconductor Industry: Top Use Cases and Benefits," em CK Birla Group, April 2, 2021. Accessed on December 28, 2021. [Online]. Available: https://www.birlasoft.com/articles/ai-semiconductor-industry-use-cases-and-benefits#:~:text=According%20to%20a%20McKinsey%20study,to%20the%20range%20of%2017%25.

[7] C. He, H. Hu, and P. Li, "Applications for Machine Learning in Semiconductor Manufacturing and Test," *In 2021 5th IEEE Electron Devices Technology & Manufacturing Conference, EDTM*, pp. 1–3, IEEE. April 2021.

[8] G. Batra, Z. Jacobson, S. Madhav, A. Queirolo, and N. Santhanam, "Artificial-intelligence hardware: New opportunities for semiconductor companies," Article, McKinsey & Company, December 2018. Available: https://www.mckinsey.com/~/media/McKinsey/Industries/Semiconductors/Our%20Insights/Artificial%20intelligence%20hardware%20New%20opportunities%20for%20semiconductor%20companies/Artificial-intelligence-hardware.pdf

[9] B. Bajic, I. Cosic, M. Lazarevic, N. Sremcev, and A. Rikalovic, "Machine learning techniques for smart manufacturing: Applications and challenges in industry 4.0," *Department of Industrial Engineering and Management Novi Sad, Serbia*, pp. 29, 2018.

[10] Symposium: Strategy for Resilient Manufacturing Ecosystems Through Artificial Intelligence. Report from the First Symposium Workshop: Aligning Artificial Intelligence and U.S. Advanced Manufacturing Competitiveness. December 2 and 4, 2020. Facilitated by UCLA. Supported by the National Science Foundation and the National Institute of Standards and Technology. March 2021.

[11] U. Bhatt, J. Antorán, Y. Zhang, Q. V. Liao, P. Sattigeri, R. Fogliato, G. Melançon, R. Krishnan, R. Stanley, O. Tickoo, and L. Nachman, "Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty," *In Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 401–413, July 2021.

[12] H. Linusson, "An introduction to conformal prediction," *The 6th Symposium on Conformal and Probabilistic Prediction with Applications (COPA 2017)*, Tutorial 1, 2017.

[13] G. Shafer and V. Vovk, "A tutorial on conformal prediction," *J. Mach. Learn. Res.* **9**, pp. 371–421, 2008.

[14] Y. Romano, E. Patterson, and E. J. Candès, "Conformalized quantile regression," *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp. 3543–3553, 2019.

[15] C. A. Mack, "Measuring Line Edge Roughness: Fluctuations in Uncertainty," *The Lithography Expert*. 2008.

[16] "Line Edge Roughness (LER)," *Semiconductor Engineering, Deep Insights for the Tech Industry. (n.d.)*. Accessed on January 3, 2022. [Online], Available: https://semiengineering.com/knowledge_centers/manufacturing/lithography/line-edge-roughness-ler/

[17] 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.

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

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

[20] 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.

[21] 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.

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

[23] 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.

[24] V. Vovk, A. Gammerman, G. Shafer, Algorithmic Learning in a Random World, 1st ed., Springer, New York, 2005.

[25] H. Papadopoulos, K. Proedrou, V. Vovk, and A. Gammerman, "Inductive confidence machines for regression," *In T. Elomaa, H. Mannila, and H. Toivonen, editors, Machine Learning: ECML 2002, ECML 2002, Lecture Notes in Computer Science*, vol. 2430, Springer, Berlin, Heidelberg, 2002.

[26] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and Helmholtz free energy," *NIPS'93: Proc. of the 6th Int. Conf. on Neural Inf. Proc. Sys.*, pp. 3–10, 1993.

[27] R. Koenker, G. Bassett Jr., "Regression quantiles," *Econometrica*, vol. 46, no. 1, pp. 33–50, 1978.

[28] K. Hatalis, A. J. Lamadrid, K. Scheinberg, and S. Kishore, "Smooth pinball neural network for probabilistic forecasting of wind power," arXiv.org preprint arXiv:1710.01720v1[stat.ML], 2017.

[29] S. Zheng, "Gradient descent algorithms for quantile regression with smooth approximation," *Int. J. Machine Learning and Cybernetics*, vol. 2, no. 3, pp. 191, 2011.

[30] D. Kivaranovic, K. D. Johnson, and H. Leeb, "Adaptive, distribution-free prediction intervals for deep networks," *Proc. of the Twenty Third Int. Conf. on Artificial Intelligence and Statistics, PMLR*, vol. 108, pp. 4346–4356, 2020.

[31] M. Sesia and E. J. Candès, "A comparison of some conformal quantile regression methods," *Stat*, vol. 9, no. 1, pp. e261, 2020.

[32] E. I. Thorsos, "The validity of the Kirchoff approximation for rough surface scattering using a Gaussian roughness spectrum," *J. Acoust. Soc. Am.*, vol. 83, no. 1, pp. 78–92, 1988.

[33] 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. 14 472–14 478, 1993.

[34] 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.

[35] 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, pp. 737815, 2009.

[36] J. Zhang, "Gradient descent based optimization algorithms for deep learning models training." arXiv preprint arXiv:1903.03614, 2019.

[37] R. Ilin, T. Watson and R. Kozma, "Abstraction hierarchy in deep learning neural networks," *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 768–774, 2017. doi: 10.1109/IJCNN.2017.7965929.

[38] A. Elmahmudi, H. Ugail, "Deep face recognition using imperfect facial data," *Future Generation Computer Systems*, vol. 99, pp. 213–225, 2019.

[39] 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.

[40] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," *Proc. 3rd Inter. Conf. on Learning Representations (ICLR 2015), May 9 Conference Poster Session Board 11*, 2015.

[41] S. Hochreiter, J. Schmidhuber, "Long short-term memory," *Neural Computation 9*, vol. 8, pp. 1735-–1780, 1997.

[42] A. Graves, J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5–6, pp. 602–610, 2005.

[43] V. Singla, S. Singla, D. Jacobs and S. Feizi. "Low curvature activations reduce overfitting in adversarial training," arXiv.org preprint arXiv:2102.07861, 2021.

[44] N. Chaudhary, 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.