Computational Analysis of Neutron Scattering Data

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Benjamin Walter Martin

August 2015

Acknowledgements

I would like to thank Dr. Michael Berry for advising me during the writing of my dissertation. His willingness to work with me and allow me to suggest my own independent research direction is greatly appreciated. Thank you also to the other members of my PhD committee: Dr. Jens Gregor, Dr. Chad Steed, and Dr. Claudia Rawn for their feedback and input on my dissertation research.

I would also like to extend the greatest of thanks to Dr. Raju Vatsavai and Dr. Chris Symons for their support during the time that I have been in graduate school. Both Raju and Chris were my mentors during my time as an intern at Oak Ridge National Laboratory, and their technical instruction and encouragement was instrumental in my completion of my PhD. Thank you very much Raju and Chris for your invaluable guidance during my internship.

In addition, I would like to thank Dr. Ross Whitfield and Dr. Rick Archibald of Oak Ridge National Laboratory for their assistance in developing means to generate the data used in this work. Their explanations were essential in helping me understand the background material relating to this work as well as how to set up and run simulations for neutron scattering experiments.

Finally, I would like to thank Dr. David Keffer of the University of Tennessee for taking the time to discuss the work presented in this dissertation and giving me feedback on my approach. Dr. Keffer's input was helpful in understanding what questions a materials scientist may have concerning my work.

Abstract

This work explores potential methods for use in the detection and classification of defects within crystal structures via analysis of diffuse scattering data generated by single crystal neutron scattering experiments. The proposed defect detection methodology uses machine learning and image processing techniques to perform image texture analysis on neutron diffraction patterns generated by neutron scattering simulations. Once the methodology is presented, it is tested via a series of defect detection problems of increasing difficulty which utilize neutron scattering data simulated by a number of simulation techniques. As the problem difficulty is increased, the defect detection methodology is refined in order to adapt to challenges presented by the more difficult detection problems. The refinement process includes the development of a data-driven scaling method that aids in the texture analysis process by enhancing diffuse scattering textures in the diffraction patterns. The evaluation process for the defect detection methodology includes analysis and comparison of the computational complexities of the machine learning and image processing techniques. As part of this complexity analysis, a detailed study of the ORB keypoint extraction algorithm is also conducted and the computational complexity of the ORB algorithm is derived.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Defect detection in crystalline materials is an area of importance across a number of disciplines. The presence of defects within a crystal can affect a number of material properties including material strength (Sun et al., 2009), thermal conductivity (Zhan et al., 2014), properties relevant to the development of pharmaceuticals (Welberry and Goossens, 2014). This work seeks to use image processing and machine learning methods to detect crystal defects by analyzing reciprocal space imagery generated by single crystal neutron scattering experiments. The goal of the methodology is to perform automatic classification of defects within crystal structures and flag samples for which the methodology is uncertain of the presence of a defect.

This dissertation is organized as follows: First, the necessary background material on crystal structures and defects will be discussed. Next, a simple proof-of-concept will be presented that proposes a candidate methodology and uses a very simple defect detection problem to evaluate the effectiveness of the methodology. The methodology will then be examined in more detail in the context of a more difficult problem that utilizes a dataset generated by an open-source simulation package. A conclusion and discussion of future work will then follow.

1.1 Crystal Structures

A crystal is a material containing atoms that are arranged in a periodically ordered structure (Borchardt-Ott, 2012). The basic unit of a crystal is a structure of atoms called a "unit cell" (Evans, 1964). This unit cell is replicated and stacked in a repeating pattern to form a crystal lattice. An example of building a crystal lattice from a unit cell is given in Figure 1.1.

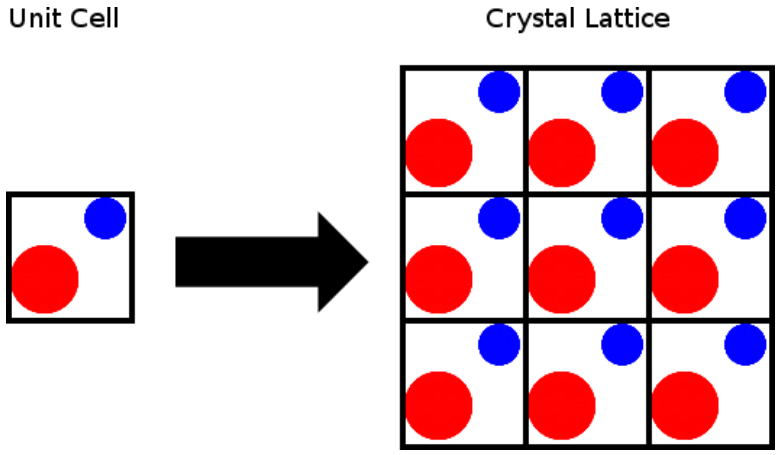


Figure 1.1: Building a crystal lattice using unit cells of atoms.

The repeating pattern present within a crystal lattice is referred to as "long-range order" (Evans, 1964). A defect occurs when some aspect of the crystal is changed such that the long-range order is disrupted (Chiang et al., 1996). Detection of defects within a crystal lattice will be the major focus of this work.

1.2 Neutron Scattering

Neutron diffraction is a means of analyzing crystal structures. A diagram outlining the neutron scattering process shown in Figure 1.2. Analysis of a crystal involves directing a beam of neutrons into a material sample and allowing the neutrons to be scattered by the atoms within the crystal structure (Schober, 2008). Neutron detectors are then used to detect the diffraction patterns generated by the scattered neutrons. These diffraction

patterns create a “reciprocal space” image that describes the structure of the material. An in-depth discussion of the definition of the reciprocal space, simulation of reciprocal space images for a neutron scattering experiment, and the role of the reciprocal space imagery in the detection of crystal defects is available in Sections 1.3 and 1.4.

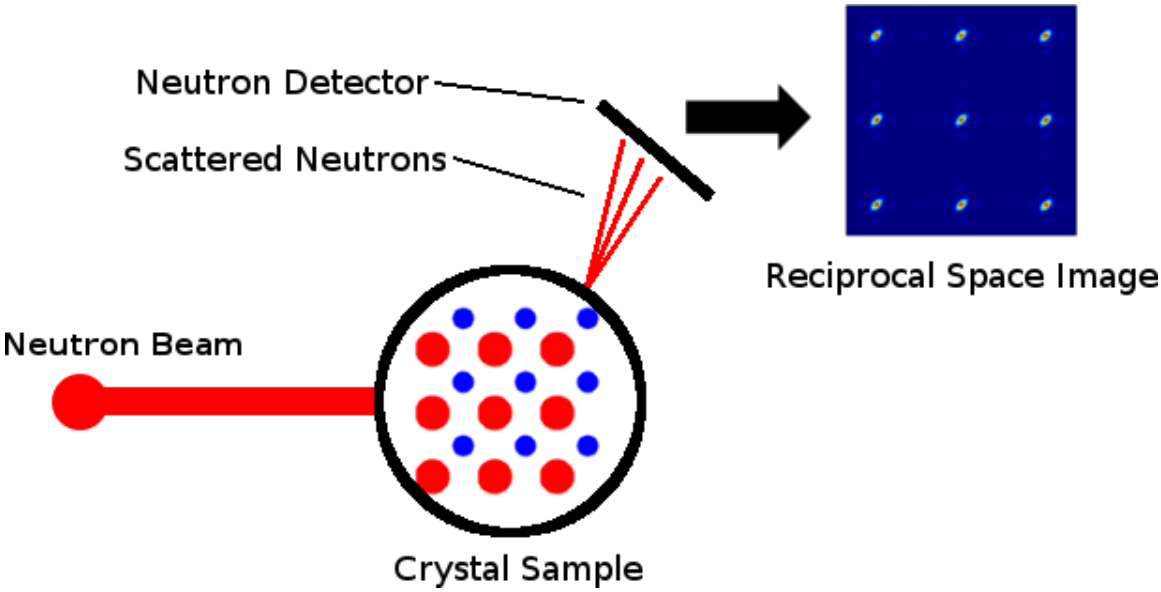


Figure 1.2: An overview of the neutron scattering process.

1.3 Reciprocal Space

A reciprocal space image is an intensity map of the scattered neutrons detected during a scattering experiment (Butler and Welberry, 1992). Figure 1.3 is an example of a reciprocal space image generated by a neutron scattering experiment. As illustrated in Figure 1.4, making changes to a crystal structure can cause changes in the reciprocal space image, and thus the reciprocal space image can be used to identify defects within a crystal. By examining the nature of the differences between the reciprocal space images for a known pure crystal and for a new unknown crystal, it is possible to observe signs of defects within the reciprocal space image for the new crystal structure.

The textures within reciprocal space images can be divided into two classes: high-intensity Bragg peaks, and low-intensity diffuse scattering. The Bragg peaks within

the image describe the average structure of the crystal structure (Egami and Billinge, 2012), and the diffuse scattering patterns describe deviation from the average crystal structure (Nield and Keen, 2001). Therefore, since a defect is a deviation from the average crystal structure, analysis of the diffuse scattering patterns will be the focus of this work.

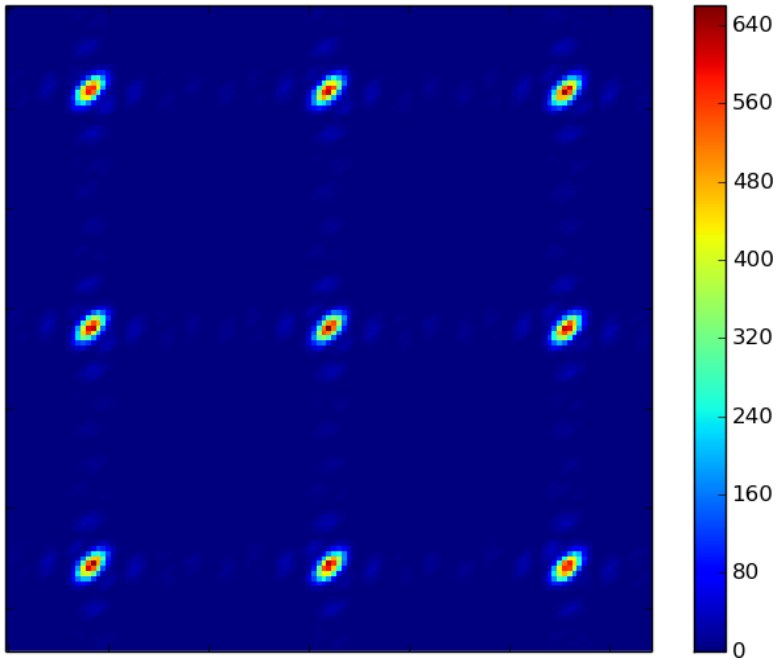


Figure 1.3: Sample reciprocal space image from a simulated neutron scattering experiment.

1.3.1 Mathematical Definition of Reciprocal Space

As shown by (Butler and Welberry, 1992), it is possible to mathematically generate a simulated reciprocal space image using a 2-dimensional discrete Fourier transform (DFT). The general form for the reciprocal space is given in Equation 1.1. In this type of simulation, the neutron beam is modeled as a plane wave that generates spherical plane waves that are scattered from the atoms within the crystal lattice (Pynn, 2008). In the equation, $A(\mathbf{k})$ is the complex scattering amplitude at vector location $\mathbf{k}$ in the image, N is the number of cells in the lattice, F_m is the structure factor for cell m, and $\mathbf{R}_m$ is the position vector for the mth cell in the lattice.

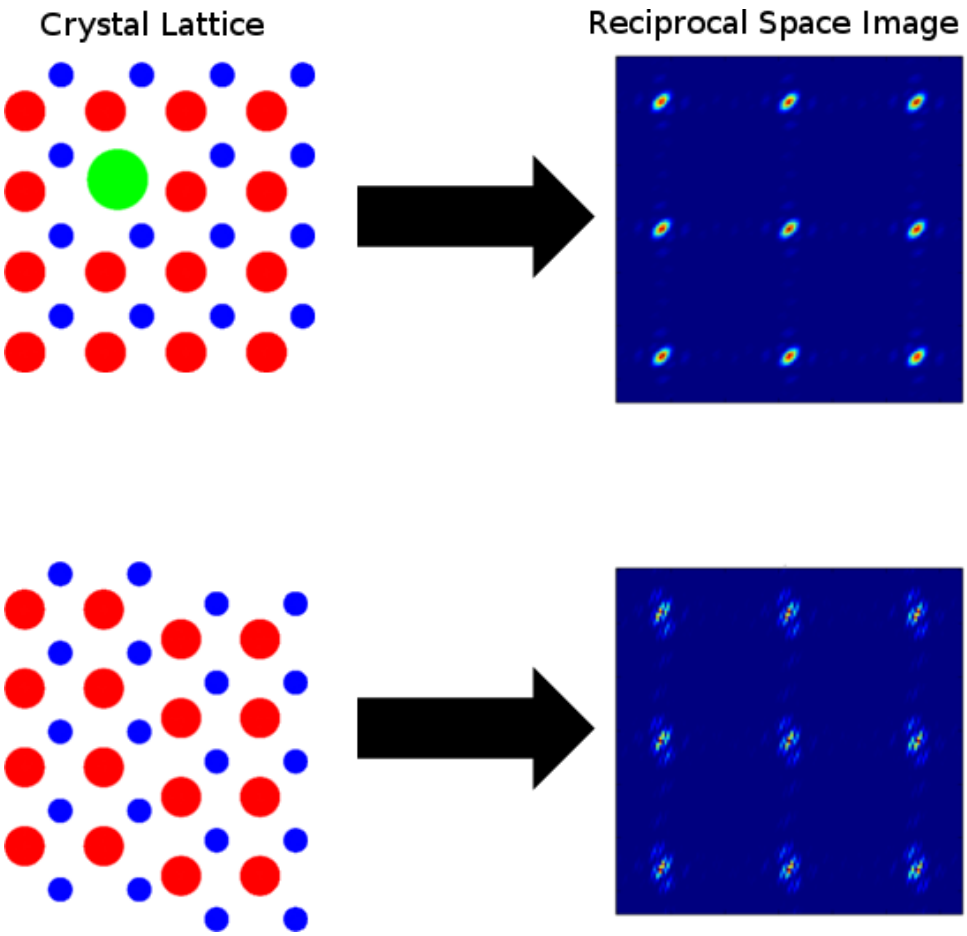


Figure 1.4: Different crystal structures can produce different reciprocal space images.

$$A\left(\mathbf{k}\right) = \sum_{m=1}^{N} F_m exp\left(i\mathbf{k} \cdot \mathbf{R}_m\right) \tag{1.1}$$

The structure factor for a cell can be calculated using formula in Equation 1.2, where N_m is the number of atoms in cell m, f_n is the atomic scattering factor for atom n (available via a lookup table), and $\mathbf{r}_n$ is the position vector for atom m within the cell.

$$F_m = \sum_{n=1}^{N_m} f_n exp\left(i\mathbf{k} \cdot \mathbf{r}_n\right) \tag{1.2}$$

Once the complex scattering amplitude $A(\mathbf{k})$ has been calculated, then the reciprocal space image can be calculated by multiplying $A(\mathbf{k})$ by its complex conjugate to get the magnitude of the intensity at pixel location $\mathbf{k}$ as shown in Equation 1.3.

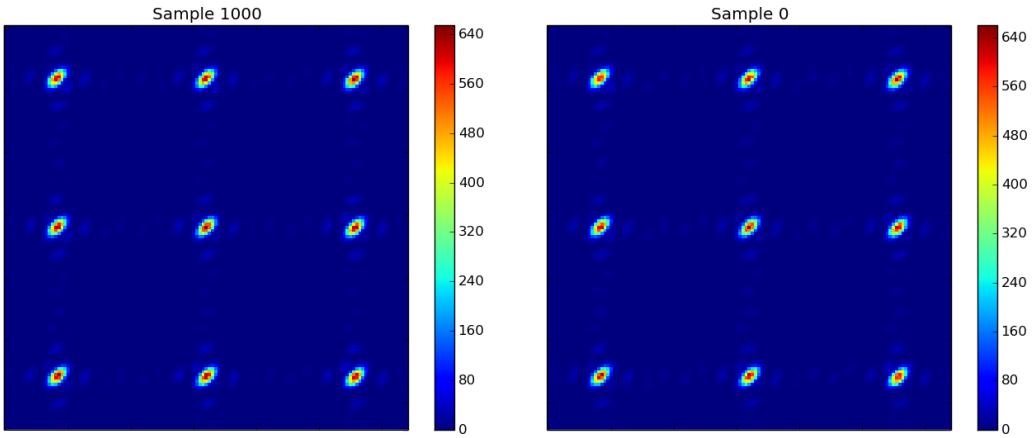
$$I(\mathbf{k}) = A(\mathbf{k}) A^*(\mathbf{k}) \tag{1.3}$$

The fact that the intensity of the reciprocal space image is the squared magnitude of the complex scattering intensity presents a very significant problem. By taking the magnitude, the phase information for the scattering is completely lost and thus makes it impossible to determine the crystal structure directly via the inverse Fourier transform (Pynn, 2008). While the phase information could be preserved in simulation by simply working with the complex data, in many cases the phase data cannot be detected in a real neutron scattering experiment (Egami and Billinge, 2012). This phenomena is known as the "phase problem", and addressing the phase problem is an open area of research (Welberry and Goossens, 2014). The methodology presented in this work seeks to overcome this limitation by focusing on analysis of the textures present in the magnitude data for a reciprocal space image.

1.4 Defect Detection

As mentioned in Section 1.1, a crystal defect occurs when the long-range order of the crystal lattice is disrupted. When a defect is introduced into a crystal structure, the reciprocal space image is modified in a specific way that is determined by the defect type. Therefore, a particular type of defect will generate textural features within the reciprocal space image that can be looked at as a "fingerprint" that identifies the type of defect within the crystal. However, these changes in the reciprocal space image can be very subtle and may not be visible to the human eye. Thus simply identifying the defects via visual inspection may not be possible in some cases. As an illustration, Figure 1.5 shows a side-by-side comparison of two images: Figure 1.5a is a reciprocal space image for a pure

crystal structure, and Figure 1.5b is a reciprocal space image for the same crystal structure containing a substitution. While the images may look very similar, subtracting one of the images from the other as shown in Figure 1.6 reveals that there are very distinctive yet subtle differences between the images. Thus a computational methodology that can automatically detect these types of subtle defects would be greatly beneficial as it could potentially detect defects that are not visible to the human eye.



(a) Pure Crystal **(b)** Substitution

Figure 1.5: Sample reciprocal space images. Figure 1.5a shows a reciprocal space image for a pure crystal structure, and Figure 1.5b is a reciprocal space image for the same crystal structure containing a substitution defect.

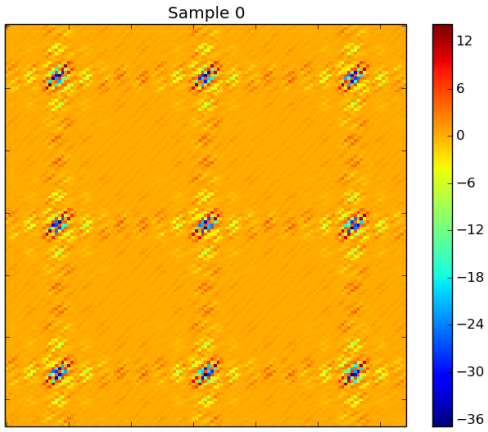


Figure 1.6: Difference of the two images in Figure 1.5. This image was generated by subtracting Figure 1.5a from Figure 1.5b.

1.5 Previous Work

The computational processing of neutron scattering data is an emerging field of research, and analysis methodologies for reciprocal space images are still being developed. A common method of defect detection is visual inspection by a human expert or a trial-and-error approach (Egami and Billinge, 2012). This is an arduous task that involves collecting reference reciprocal images for defective crystals from neutron scattering literature or simulation, visually observing the characteristics of the reference images, and then searching for similarities in the characteristics of the reference images to the new image for the crystal that is under analysis. If the new image shares enough visible similarities to any of the the reference images, then the new image is assumed to contain the same defects as the reference image. In many cases, this process is performed because a researcher has a suspicion as to the type of defect within the crystal and wants to verify that the defect in fact exists within the crystal. However, it is less likely that the researcher will identify a defect within an image if there is not prior suspicion that a defect is present.

Image texture analysis has shown to be a viable method of classifying images. Previous studies have shown keypoint-based texture analysis to be successful in areas such as scene classification (Ayers and Boutell, 2007) and change detection in satellite imagery (Martin and Vatsavai, 2013). All of these previous studies extract texture features and then use machine learning methods to analyze the textures in the images. However, in contrast to the crystal defect detection problem presented in this chapter, these previous studies focused on classifying images based on large differences between the image classes. This work seeks to evaluate the use of image texture analysis to detect subtle defects within the reciprocal space images and determine the best methods to use when applying image texture analysis to reciprocal space imagery analysis.

1.6 Summary

It has been established in this chapter that detection of defects within a crystalline material via analysis of reciprocal space imagery is an important problem that is currently a difficult problem to solve using current methods. Given the amount to effort required to manually analyze reciprocal space imagery in an effort to identify defects within the crystal structure, a computational method to automatically detect defects would be of great benefit to the scientific community. Relevant background material has also been presented in order to introduce the reader to crystal structure concepts necessary to understand the problems presented in the remainder of this work. The following chapters discuss a proposed methodology that can be trained to recognize defects by analyzing a reciprocal space image generated by a crystal and evaluate the effectiveness of this methodology using various simulated datasets.

Chapter 2

Proof of Concept: Defect Detection for Small Crystal Structures

2.1 Introduction

An initial test of the feasibility of automatically detecting crystal defects using reciprocal space images involved series of experiments performed on a small crystal structure. The goal was to train a classifier that could distinguish between different type of defects within a particular crystal. In order to maintain a controlled environment in which to evaluate candidate methodologies, a simple dataset for an extremely small simulated crystal structure was used in experimentation. This simple dataset was used with the intention of developing methodologies that could be used on larger structures. The work in this chapter seeks to specifically answer the following questions:

1. Is it possible to train a classifier which can automatically classify defects present within simulated reciprocal space imagery?
2. How can one extract descriptive features from the images?
3. Can only large defects be detected within the crystals, and can more subtle defects be detected as well?

4. Does the type of classifier used matter when performing defect detection?

2.2 Problem Background

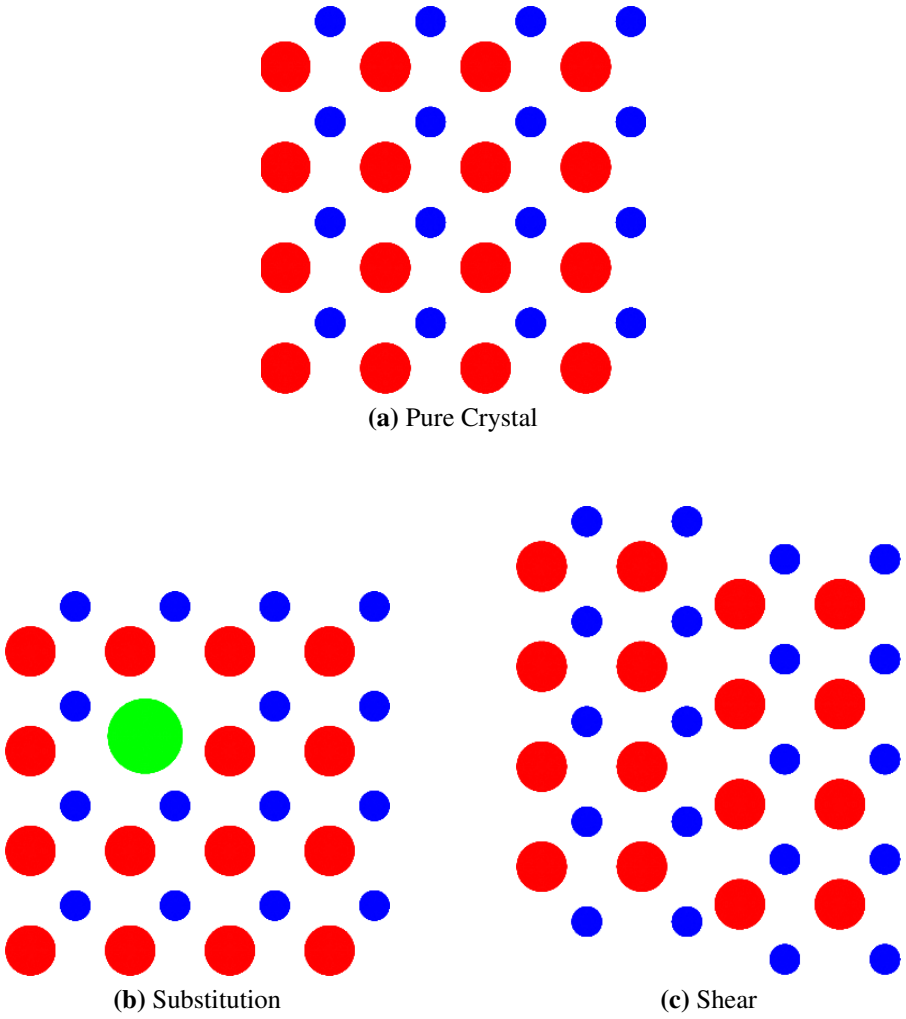
As stated in Chapter 1, crystalline structures consist of a lattice constructed of repeating patterns of "unit cells". A unit cell is a fundamental building block for a crystal and can contain a varying number of atoms. In a pure crystal, the cell structure remains uniform across the entire crystal, and defects occur when a cell within the crystal structure deviates from this uniform structure. There are many different types of simple defects that can occur in a small crystal structure such as the one used in this chapter. Below are the types of defects considered in this chapter:

- Substitutions — A cell in the crystal is replaced with another type of cell.
- Shear — Deformation along parallel planes intersecting the crystal structure.

Diagrams illustrating these defects are given in Figure 2.1. One thing to note is that the substitution defects analyzed in this work only modify a single unit cell at a time whereas the shear defects will affect unit cells along the entire shear plane. Therefore, given the formulation of the reciprocal space in Chapter 1, shear defects can potentially make larger changes to the reciprocal space image than the substitution defects. This observation on the effect of the defect size on the reciprocal space image was leveraged when constructing increasingly difficult experiments for use in evaluation the defect detection methodology.

2.3 Image Keypoint Extraction

In order to facilitate the processing of a reciprocal space image computationally, a numerical description of the image must be generated which captures the most relevant features from the image. These descriptors can describe the edges or corners within the image, the image's textures, and more. One class of image descriptors that are of



(a) Pure Crystal

(b) Substitution

(c) Shear

Figure 2.1: Defect diagrams for a simple crystal structure.

particular interest to this work are "keypoint descriptors". Keypoint extractors use a two-step approach to generate descriptors for an image. In the first step, areas of interest, or "keypoints", are identified within the image using a specific set of criteria predefined by the extractor. Once the keypoints are identified, a feature vector is then computed which mathematically describes the texture of the image in the area surrounding the keypoint. Keypoint feature extraction can be performed in a variety of ways, but in general a keypoint extraction algorithm will provide two pieces of data for each keypoint: the coordinates for the keypoint location within the image, and a descriptor for the image texture surrounding

the keypoint location. The specific methods used in the detection and computation steps are defined by the individual keypoint extraction algorithm. Figure 2.2 provides a visual illustration of the general process of extracting features from an image using a generic keypoint extraction algorithm.

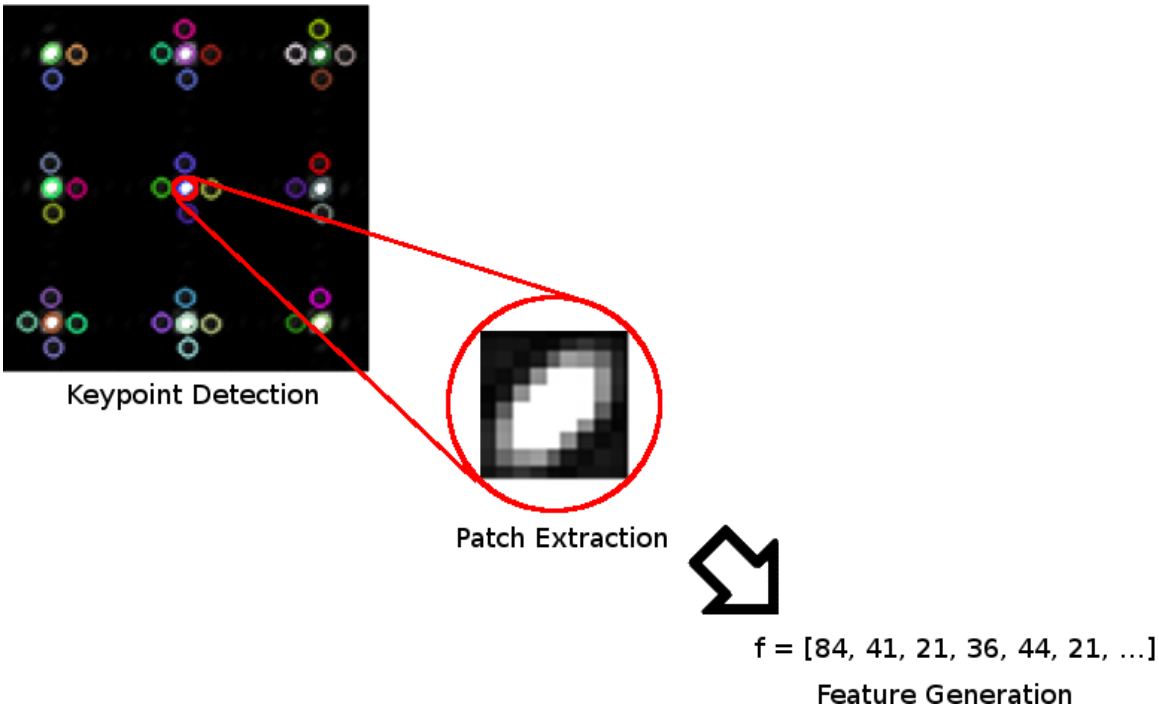
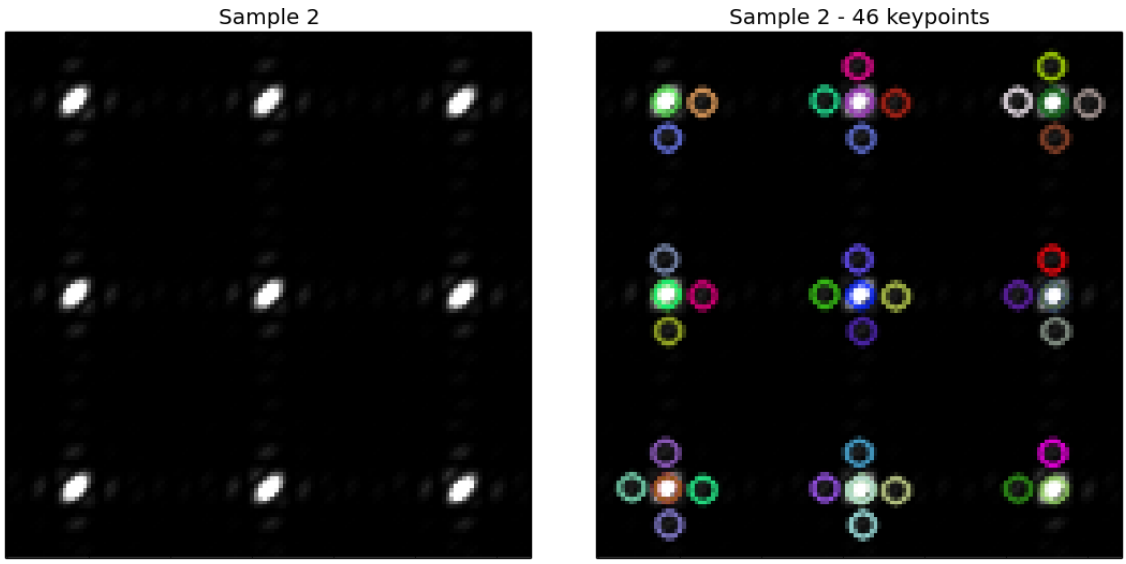


Figure 2.2: Illustration of the keypoint-based feature extraction process.

2.3.1 Scale Invariant Feature Transform (SIFT)

The Scale-Invariant Feature Transform (SIFT) is a common keypoint extraction methodology that was developed as a means to extract features that can be used to match objects within two different images. The features were designed to be scale and rotation invariant such that they are robust to changes in the image such as affine transformations, noise addition, and lighting changes (Lowe, 2004). For keypoint detection, SIFT uses a difference-of-Gaussian function to identify intensity extrema in the image which are designated as keypoints. Descriptors for the resulting keypoints are then generated by creating a 16x16 pixel patch centered on the keypoint location, partitioning the patch into a

4x4 grid, and calculating 8-bin gradient orientation histograms within the cells of the grid. This creates a 128-dimensional SIFT descriptor for each keypoint. These feature vectors can then be used as inputs to a classification algorithm. Figure 2.3 shows an example of a grayscale representation of a reciprocal space image and the locations of the keypoints detected for that image by SIFT.



(a) Input Image **(b)** Detected Keypoints

Figure 2.3: Example of image keypoint detection. The colored circles in Figure 2.3b are the locations of the keypoints detected for the input image. After the keypoint detection step, descriptors are calculated for the texture surrounding the keypoint locations. The SIFT keypoint detection algorithm described in Section 2.3.1 was used in this example.

2.4 Machine Learning Algorithms

This work utilizes supervised machine learning methods to train a model that can classify the keypoint descriptors extracted from the reciprocal space imagery. The general idea behind supervised learning is to train a learner to predict the class, or "label", for a given feature vector. The training process is accomplished by presenting the learner with a set of samples from a training dataset in which the samples have already been assigned labels.

The learner uses the samples within this training set to build a classification model that can then be used to predict the labels for new, unlabeled samples in a testing dataset.

2.4.1 Support Vector Machines

Support vector machines (SVMs) are a type of machine learner which have been widely adopted due to their accuracy, speed, and simplicity (Boser et al., 1992). A SVM attempts to find a hyperplane which linearly divides the classes and maximizes the separation margin between the classes. This is accomplished by training the SVM using data points for which the class is previously known and identifying "support vectors" which define the maximal margin boundaries. Figure 2.4 illustrates classification using a SVM for a small dataset.

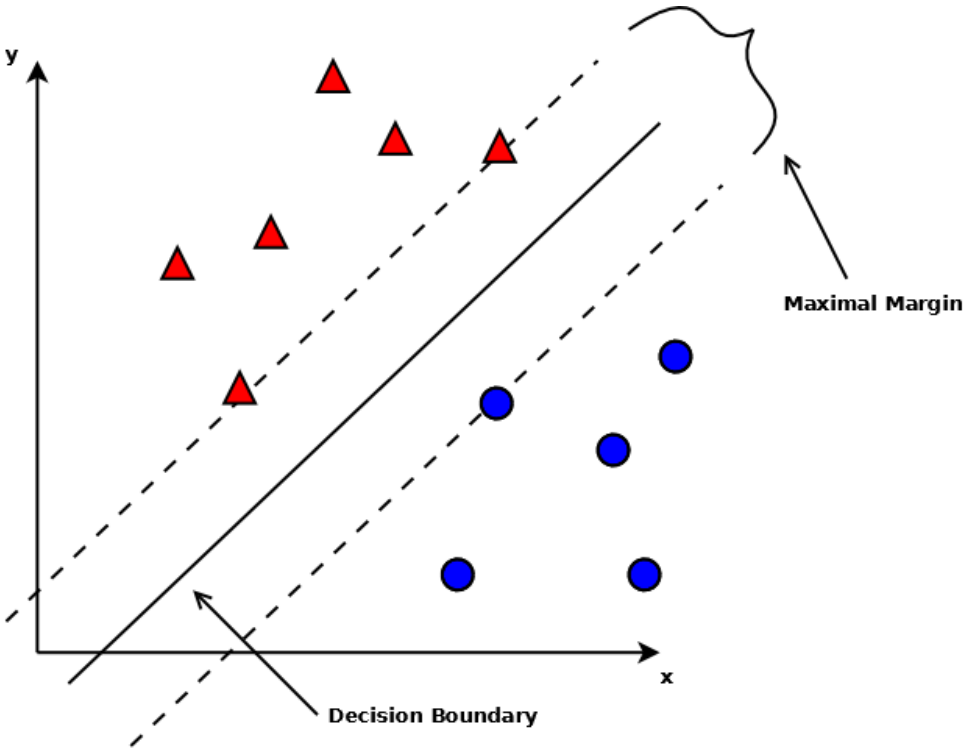


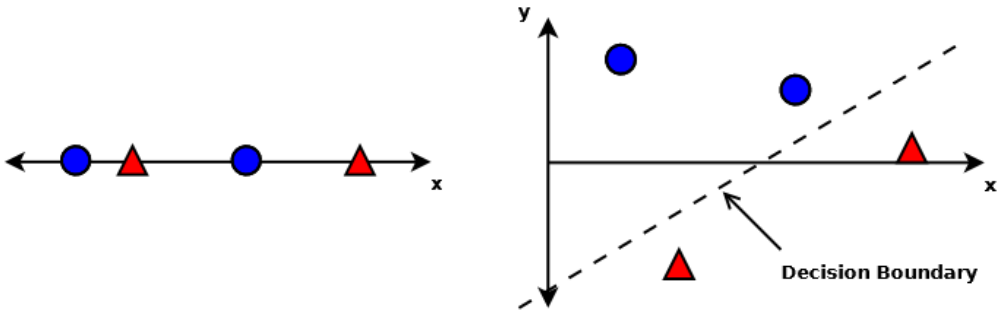
Figure 2.4: Classification via support vector machine. The points on the margin boundaries (dotted lines) are the support vectors for this dataset.

Typically a SVM only distinguishes between two classes, but SVMs can also be used for a multi-class problem by training a separate SVM model for each class. Each model then predicts whether or not an instance is a member of its assigned class. This ability to

do multi-class learning allows for a single SVM to be trained which can detect many types of crystal defects.

Although SVMs were designed to classify linearly separable data, a kernel function can be applied to the data during SVM training in order to allow for classification of data which is not linearly separable. A kernel function maps the data from one feature space to another with the goal of the data being linearly separable in the new feature space.

Figure 2.5 provides an example of using a kernel to transform data for linear separability. There are many types of kernel functions that are traditionally used with SVMs, but this chapter will evaluate both a linear kernel and a radial basis function (RBF) kernel.



(a) Input Space **(b)** Mapped Feature Space

Figure 2.5: Use of kernel to map data to higher dimensional space for linear separation by a SVM.

2.4.2 Ensemble Learning and Random Forests

Ensemble learning is a class of machine learning frameworks that use collections (ensembles) of learners to train a model. The basic idea behind ensemble learning is that a series of k learners with weak classification performance are trained individually and then combined to produce a strong classifier. This can lead to increased accuracy and robustness for the classifier because the output is a combination of "opinions" by the individual learners based on their analysis of a particular aspect of the problem space. It should be noted that the

machine learning algorithm utilized by the individual learners within an ensemble can often be selected irrespective of the ensemble framework.

There are a number of types of ensemble learning frameworks, but one specific framework of interest is "bootstrap aggregating" or "bagging" (Breiman, 1996). A bagging ensemble trains k base learners using different subsets of the training dataset. These subsets are created by randomly selecting N samples the training dataset with replacement such that N is much less than the number of total training samples. During classification, k labels are generated by the members of the ensemble and the final classification for the input sample is typically generated by taking a majority vote or average of the outputs by the ensemble members.

A random forest is a variant of the bagging framework that uses binary decision trees as the base learning method for its ensemble members (Breiman, 2001). Random forests implement the previously described bagging methodology, but also apply a "feature bagging" scheme during training. In feature bagging, each ensemble member learns to classify samples at each split of the decision tree using only a subset of the *features* within each feature vector. Thus an ensemble member within a random forest will focus on learning to correctly classify its subset of the training samples while considering only small portions of the features for a given sample. This allows the random forest to learn to filter noisy or irrelevant features while maintaining high accuracy and short training times (Breiman, 2001). Figure 2.6 illustrates the structure of a random forest.

2.5 Dataset Information

The dataset used in this chapter is a simple dataset containing reciprocal space imagery for a generic 8 cell by 8 cell planar crystal simulated using the methodology described in (Butler and Welberry, 1992). The dataset was not constructed with the intent of simulating a specific realistic crystal structure, but was instead designed to provide a simple problem which could be used to evaluate the feasibility of detecting crystal defects by analyzing reciprocal space imagery using machine learning methods. The crystal's unit

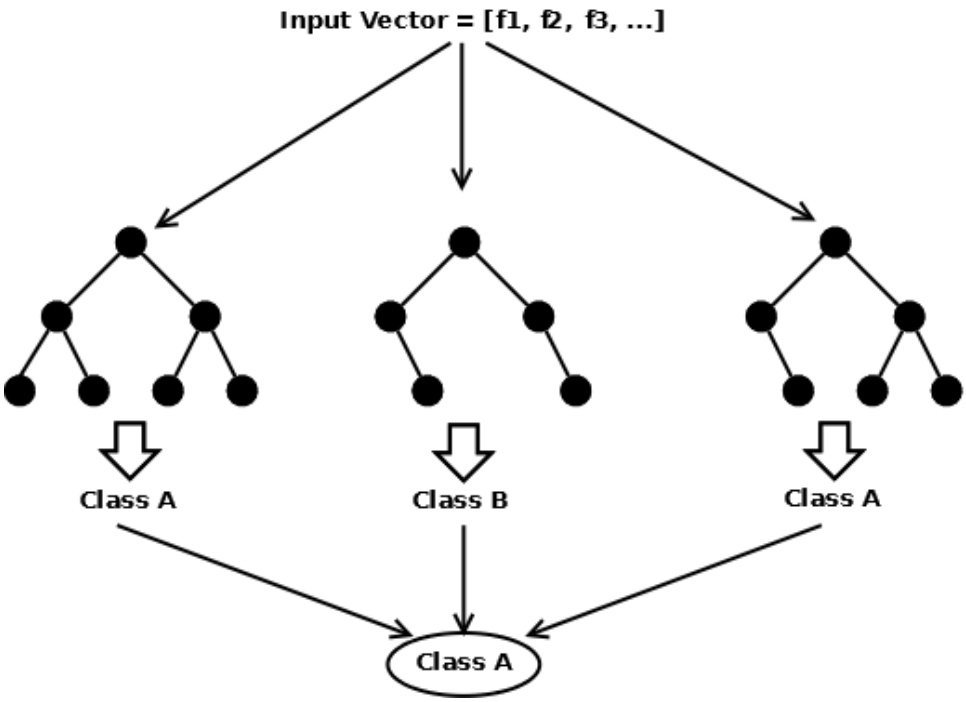


Figure 2.6: Classification via random forest containing 3 decision trees.

cells contained two atoms: an atom at location (0.35 Å, 0.35 Å) with scattering factor 0.25, and an atom at location (0.8 Å, 0.8 Å) with scattering factor 0.15. All atom coordinates were relative to one corner of the unit cell. The only simulation output provided was reciprocal space intensity maps which were in the form of a 129 pixel by 129 pixel image with floating point intensities on the range [0, 655]. The SIFT keypoint extractor requires that the intensities of all input images be on the interval [0, 255], and the intensities for the reciprocal space images were thus thresholded at 255 in order to accommodate for this constraint. For a given reciprocal space image, one of the three types of defects listed in Section 2.2 were present within the crystal structure. Furthermore, for the images for crystals containing substitution defects, two types of substitutions were present: small substitutions and large substitutions. A small substitution was defined as a substitution in which the new cell contained a single atom that had an atomic scattering factor on the range [0, 1]. Similarly, in a large substitution the new cell contained a single atom with an atomic scattering factor on the range (1, 2). The shear samples contained a shear of varying magnitude along a shear plane randomly placed within the crystal.

For the large substitution and shear classes, 1,000 reciprocal space images were available for analysis. The small substitution dataset had 200 images available, and thus, in order to balance the class representation when performing experiments, reduced sets of 200 large substitution and 200 shear images were used in the following experiments unless otherwise noted. All of the images had labels available which described the type of defect present within the image. In addition, the large substitution dataset included a set of parameters describing the location of the defect within each image. Figure 2.7 contains a representative sample of the reciprocal space images within the dataset. The figure shows that there was similarity between the images and that many of the images were hard to distinguish via inspection. As predicted in Section 2.2, one can observe that some of the shear samples have larger differences compared to the substitution samples due to the fact that a shear defect can affect a larger part of the cells in the lattice as compared to a substitution defects.

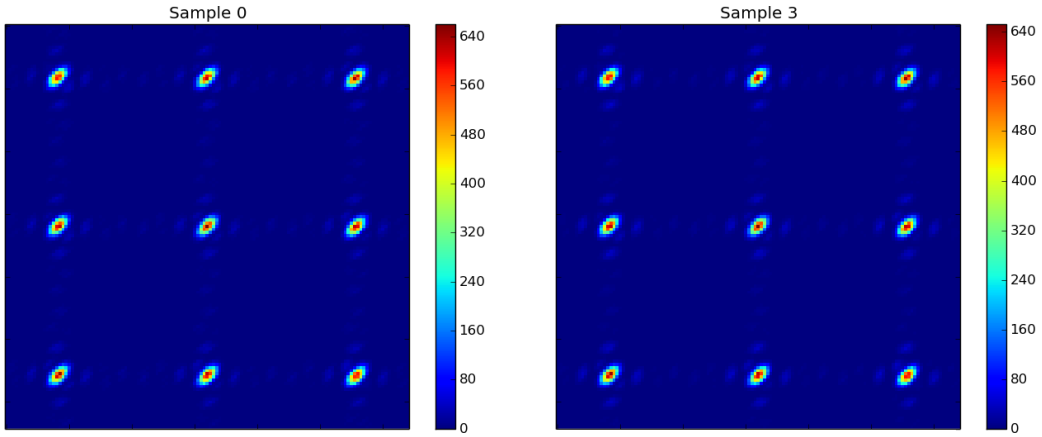
2.6 Defect Detection Methodology

Given the properties of the reciprocal space data, a methodology to detect and classify crystal defects within reciprocal space images was developed. The goal was to be able to classify crystal structure defects by learning from labeled reciprocal space images. With this in mind, the defect detection methodology was comprised of two-stages:

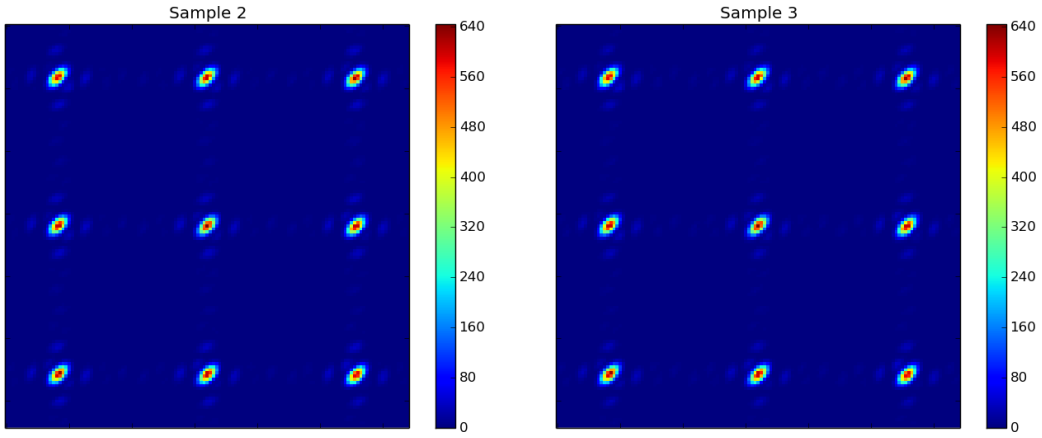
1. A feature extraction stage which analyzed the reciprocal space imagery and generated feature vectors which could be used in classification.

2. A machine learning stage which used the extracted feature vectors to train a model which was then used to detect defects in new reciprocal space images.

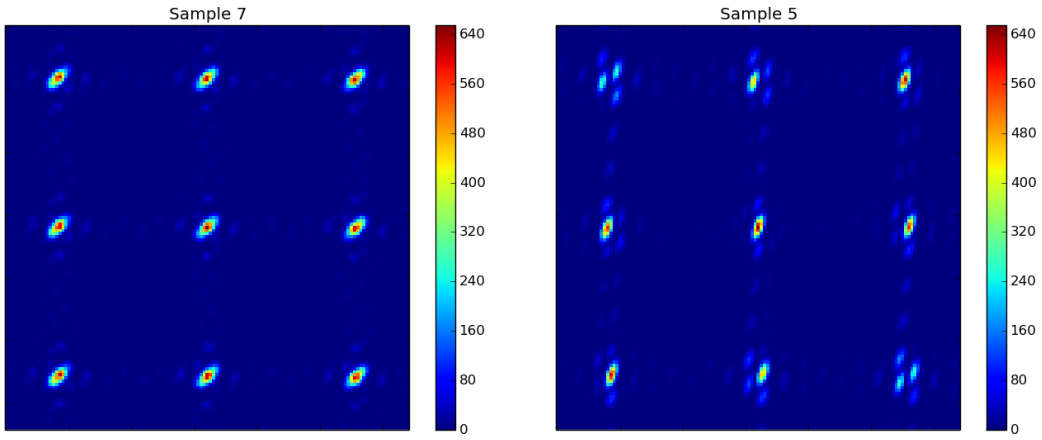
During the training phase of the algorithm, the keypoint extractor automatically detected the areas of interest within the reciprocal space image and then extracted a series of descriptors which described the image texture for each detected keypoint. Once the



(a) Large Substitution



(b) Small Substitution



(c) Shear

Figure 2.7: Representative reciprocal space images from the simulated small structure neutron scattering dataset.

descriptors were extracted, each descriptor was assigned a label that was the same as the label for the entire image. The machine learner was then trained to take a keypoint descriptor as an input and predict the type of defect present in the image that the keypoint descriptor was extracted from.

When classifying a new image, the keypoint descriptors for the entire image were extracted and individually labeled by the learning algorithm. Once all of the descriptors had a classification, the final label for the image was determined by a majority vote of the labels for the individual descriptors. The keypoint descriptors for an entire image were considered as independent features which describe the most unique aspects of an image. Therefore, class membership was assigned for an image based on the most frequent class assignment for its individual keypoint descriptors. This majority voting method for classification also allowed for filtering of features that were shared among all images or unique to a specific image because the predicted labels for many features are considered during class assignment. Figure 2.8 is a flowchart outlining the basic stages for this methodology.

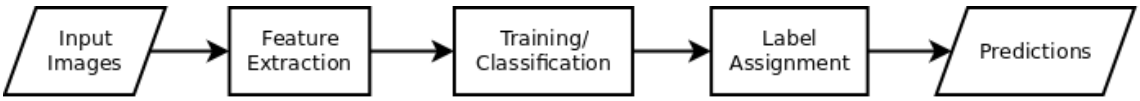


Figure 2.8: Flowchart for defect detection methodology.

2.7 Experiments

A series of experiments were performed which evaluated the effectiveness of the methodology described in Section 2.6 in classifying defects within the reciprocal space images contained the dataset described in Section 2.5. All of the experiments in this section used the Python bindings for the SIFT module provided by the OpenCV computer vision library (Bradski, 2000) to generate descriptors for the reciprocal space images. In addition, the "svm" and "ensemble" modules from the scikit-learn library (Pedregosa et al., 2011) were used for the SVM and random forest classification, respectively. The scikit-learn library uses libSVM (Chang and Lin, 2011) as its underlying SVM implementation. Two

SVMs were evaluated: one which used a linear kernel, and one which used a RBF kernel with a kernel coefficient of $\frac{1}{F}$, where F is the number of features. The random forest used in the experiments contained 10 decision trees.

2.7.1 Defect Type Classification

The series of experiments tasked the classifier with learning to predict the type of defect present in a given reciprocal space image. In order to perform an initial assessment of the overall performance of the methodology, an experiment was conducted which tasked the classifier with separating the samples into two classes: "substitution" and "shear". A total of 400 substitution samples (200 large substitutions and 200 small substitutions) and 200 shear samples were used in the experiments. During the training phase, a training dataset was generated by randomly selecting 10% of the images from each class and extracting keypoint descriptors from the images which were used to train either a SVM (linear or RBF kernel) or a random forest. Once the machine learning model was trained using the training dataset, predictions for the remainder of the images in the dataset were collected and tallied. In order to filter noise in the experimental results, 20 independent trials were performed for each experiment where a new training dataset was sampled and a new model was trained for each trial. The confusion matrices for each trial were summed across the 20 runs, and the aggregated confusion matrices are given in Tables 2.1, 2.2, and 2.3.

Table 2.1: Aggregated confusion matrix for 2-class SVM (linear kernel) experiment.

Actual Class \ Predicted Class	Substitution	Shear	Recall
Substitution	7200	0	100.0%
Shear	290	3310	91.94%
Precision	96.12%	100.0%	97.31%

Table 2.2: Aggregated confusion matrix for 2-class SVM (RBF kernel) experiment.

Predicted Class / Actual Class	Substitution	Shear	Recall
Substitution	7200	0	100.0%
Shear	441	3159	87.75%
Precision	94.23%	100.0%	95.92%

Table 2.3: Aggregated confusion matrix for 2-class random forest experiment.

Predicted Class / Actual Class	Substitution	Shear	Recall
Substitution	7200	0	100.0%
Shear	211	3389	94.14%
Precision	97.15%	100.0%	98.05%

Analysis of the tables shows that all three machine learning algorithms were able to successfully identify the defects present in the testing dataset with very high accuracy. However, Table 2.2 shows that the RBF kernel mistakenly placed more of the "shear" samples in the "substitution" class than the others and thus had poorer accuracy than the other two methods. The random forest had the highest accuracy of the three methods tested with the linear SVM having slightly lower accuracy than the random forest. As noted in Section 2.5, classification between some of the substitution and shear samples was not extremely difficult due to the visible differences between some of the reciprocal space images. However, the experiment was able to prove that keypoint descriptors were a suitable choice for extracting relevant features from the reciprocal space images.

In order to create a more difficult problem to further test the capability to the defect detection methodology, a second experiment was conducted which split the "substitution" class into two separate classes: "large substitutions" and "small substitutions" as defined

in Section 2.5. Each of the new classes generated by the split contained 200 samples each. All other aspects of this experiment were identical to the previous 2-class experiment. The differences between the two substitution classes were much more subtle than the differences between the substitution and shear classes of the previous experiment. Therefore, the new 3-class experiment explored whether the defect detection methodology was capable of classifying subtle defects in addition to larger ones. The results for these new experiments are given in Tables 2.4, 2.5, and 2.6.

Table 2.4: Aggregated confusion matrix for 3-class SVM (linear kernel) experiment.

Predicted Class / Actual Class	Large Substitution	Small Substitution	Shear	Recall
Large Substitution	2073	1527	0	57.58%
Small Substitution	1293	2307	0	64.08%
Shear	236	89	3275	90.97%
Precision	57.55%	58.81%	100.0%	70.87%

Table 2.5: Aggregated confusion matrix for 3-class SVM (RBF kernel) experiment.

Predicted Class / Actual Class	Large Substitution	Small Substitution	Shear	Recall
Large Substitution	1385	2215	0	38.47%
Small Substitution	623	2977	0	82.69%
Shear	219	122	3259	90.52%
Precision	62.19%	56.02%	100.0%	70.56%

The results in Tables 2.4, 2.5, and 2.6 show that splitting the substitutions into two subclasses caused a decrease in the classification accuracy for the all methods due to confusion between the large and small substitution classes. However, it can be observed that the classifiers were still able to distinguish between between the two types of

Table 2.6: Aggregated confusion matrix for 3-class random forest experiment.

Actual Class \ Predicted Class	Large Substitution	Small Substitution	Shear	Recall
Large Substitution	2588	1012	0	71.89%
Small Substitution	1395	2205	0	61.25%
Shear	137	35	3428	95.22%
Precision	62.81%	67.80%	100.0%	76.12%

substitutions for a majority of the substitution samples. Therefore, it can be determined that the system is robust enough to distinguish not only defect type, but also characteristics of the defect such as substitution size.

In the above 3-class experiments, there was no separation margin between the small and large substitution in the above 3-class tests. The largest scattering factor present in the small substitution dataset was 1.0, and the smallest scattering factor in the large substitution dataset was 1.001. In order to evaluate whether the size of the margin between the two separation classes had an effect on the accuracy of the classification, a third series of tests were performed which removed points from the large and small substitution classes such that a small substitution was limited to atoms with a scattering factor on the range $[0, 0.75]$. Once this new dataset was constructed, the 3-class experiment was repeated with the new dataset. Analysis of the classification accuracies using the larger margin for the substitution classes showed that widening the margin did not significantly affect the classification accuracies. Thus, more research into the nature of the differences between the large and small substitution images would be helpful in improving the accuracy of the classification system. This topic is discussed in more depth in Chapter 4.

It should be noted that for all experiments, there was a very small portion of the shear images for which no keypoints were detected due to the small size of the reciprocal space images. Therefore, the classifier was biased such that any image for which no keypoints were detected was automatically classified as “shear”.

Another thing to note is, in the unlikely event of a tie in the voting step, the algorithm defaulted to assigning a label of "substitution" in the 2-class experiment, and "large substitution" in the 3-class experiment. Improvements to this tie-breaking scheme are discussed in more depth in Chapter 3.

2.7.2 Substitution Location Prediction

In addition to detecting the type of defect within an image, experiments were performed to determine whether the methodology described in Section 2.6 could be trained detect specific properties of the defects present in a crystal. One parameter of interest was the location of the substituted cell within the large substitution dataset. The crystal lattice was 8 cells by 8 cells, so the substitution location can be described as an integer index for one of the 64 cells within the lattice.

In order to evaluate the methodology's ability to predict the substitution location, an experiment was designed which trained to predict the substitution location using 25% of the 1,000 reciprocal space images in an expanded version of the large substitution dataset. During the training process, the size of the substituted atom was not revealed to the classifier. Once training was complete, the remaining images were presented to the classifier and predictions for the substitution location were collected. As with the previous experiments, a SVM with a linear kernel, a SVM with a RBF kernel, and a random forest were evaluated, and the results were averaged over 20 runs. Table 2.7 summarizes the results of the experiments.

Table 2.7: Classification accuracies for substitution location experiments.

	SVM (Linear Kernel)	**SVM (RBF Kernel)**	**Random Forest**
Accuracy	94.80%	73.76%	95.67%

Evaluation of the results reveals that the location of a defect can be predicted by analyzing the reciprocal space image generated by a neutron scattering experiment. As was

the case in the experiments of Section 2.7.1, the random forest had the highest accuracy of all of the machine learning methods tested, with a linear SVM performing almost as well as the random forest, and the SVM with a RBF kernel once again performing poorly. Therefore, it can be determined that the defect detection methodology can be used to not only detect if a defect is present, but also extract information on certain properties of the defects themselves.

2.8 Machine Learning Algorithm Evaluation

In evaluating the the machine learning algorithms used in this chapter, two criteria were considered: classification accuracy and scalability. The relevance of classification accuracy is self-explanatory as the goal is for the classifier to produce reliable predictions. Scalability is important because the classifier should be able to quickly produce predictions as the number of training keypoints grows. Analysis with respect to the training keypoint volume is particularly important due to the fact that the number of training keypoints can be affected by the input image size and number of input images.

As shown in the above experiments, the classification accuracy for experiments using a random forest was consistently higher than the accuracy of a SVM with either a linear or RBF kernel. This result was particularly notable in the 3-class defect classification experiments. Therefore, the random forest was deemed the top performer with regard to classification accuracy.

Regarding scalability, it has been shown by (Bottou and Lin, 2007) that the computational complexity of training a SVM is between $O(N^2)$ and $O(N^3)$ for a dataset containing N training samples. The computational complexity of training a random forest with N training samples has also been shown to be $O(N * log(N))$ (Witten et al., 2011). Therefore, the training time of the random forest will scale better than the SVM as the number of training samples increases.

Given that the random forest performs well in both the area of accuracy and the area of scalability, it has been determined that the random forest is the better choice of classifier

for use in this methodology. With this in mind, random forests will be used exclusively for the rest of this work.

2.9 Summary

It has been shown that, for a simple crystal structure and defect set, it is possible to use keypoint descriptors and machine learning methods to classify single defects using reciprocal space imagery for simulated single crystal neutron scattering experiments. The SIFT keypoint extractor has been shown to be a viable method to detect and extract relevant feature descriptors from reciprocal space images, and it has been determined that the SIFT descriptors are sufficient to allow for distinguishing between images containing different defects. The experiments of this chapter also showed that the features generated by the SIFT extractor are rich enough to detect subtle differences between crystal defect types.

In addition, it has also been shown that a machine learning algorithm can trained to automatically classify the type of defect present in a crystal by analyzing the keypoint features, and that certain characteristics of the defects (such as substitution location) can be detected using the same machine learning methods. Tests revealed that random forests perform better than SVMs overall using both accuracy and scalability as an evaluation criteria.

Chapter 3

Defect Detection for Close-Packed Crystal Structures

3.1 Introduction

While the types of defects presented in Chapter 2 were helpful in performing a preliminary evaluation of the defect detection methodology presented in Section 2.6, the defects simulated in the data were merely simple examples and are not likely to be of interest in a realistic setting. In addition, the crystal structure used in testing did not simulate any sort of realistic crystal. However, the results of the experiments using this data did in fact prove that it is possible to identify defects using machine learning methods. Thus, given the success of the defect detection methodology in Chapter 2, the next logical step in development of the defect detection methodology is to evaluate its performance when detecting more complex defects on larger, more realistic crystal structures. The goal will be to detect two types of defects — stacking faults and short-range order defects — within a close-packed crystal structure. The questions to be answered by the work within this chapter are as follows:

1. Can the methodology of Chapter 2 be adapted for use with larger, complex crystal structures and defects generated using a sophisticated neutron scattering simulation package?

2. What sort of preprocessing (if any) should be done before training a classifier using the new simulated data?

3. How do alternatives to the SIFT keypoint extractor perform within this defect detection framework?

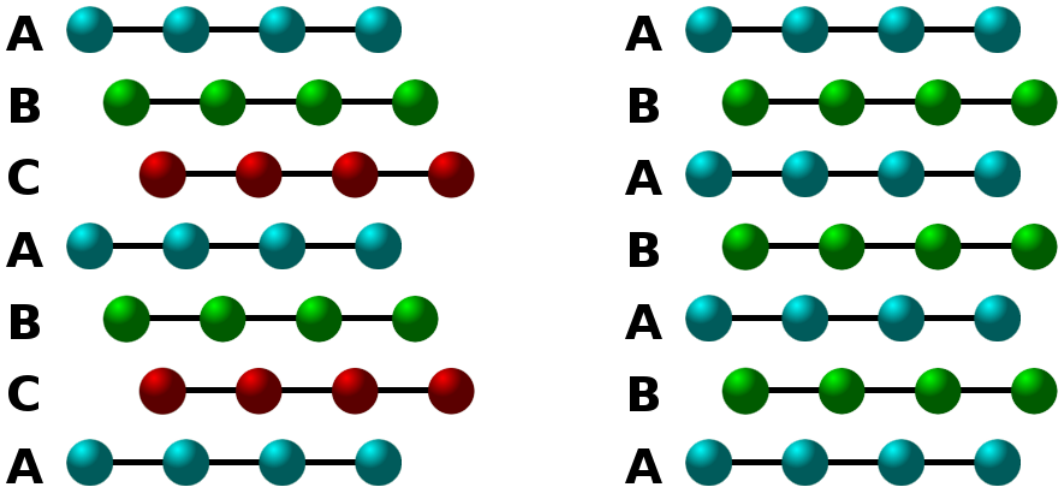
4. How can one evaluate the quality of the predictions made by the classifier?

3.2 Problem Background

3.2.1 Close-Packed Crystal Structures

The crystals analyzed in this chapter are close-packed crystal structures. In a close-packed crystal, it can be assumed that the crystal contains one type of atom, and this assumption will be made for the entirety of this chapter. Before discussing types of close-packed crystal structures, it is first necessary to discuss notation used when describing crystal structures. Close-packed crystals are created by stacking layers of atoms in various configurations. A common convention in literature is to denote crystal lattice layers using letters such as "A", "B", "C", etc. and describe stacked layer sequences as character strings such as "ABC" (Chiang et al., 1996). This layer notation will be used within the remainder of this work.

Close-packed crystal structures can occur in two basic types: cubic close-packed (CCP) and hexagonal close-packed (HCP) (Neder and Proffen, 2008). In a cubic close-packed (or face centered cubic) structure, the atoms are stacked in a 3-layer sequence such as ABCABC. In contrast, a hexagonal close-packed structure has a 2-layer ABABAB sequence. Figure 3.1 gives examples of close-packed structures.



(a) Cubic Close-Packed **(b)** Hexagonal Close-Packed

Figure 3.1: Diagrams of close-packed crystal structures. The stacking configurations in Figures 3.1a and 3.1b give examples of cubic close-packing and hexagonal close-packing, respectively.

3.2.2 Defects in Close-Packed Crystal Structures

Close-packed crystal structures can contain more complex defects than the small crystal structures evaluated in Chapter 2. The following are defects that are of particular interest to this work:

Stacking Faults

A crystal stacking fault occurs when a close-packed crystal structure changes from CCP to HCP or vice-versa (Neder and Proffen, 2008). Thus sequences such as ABCABABC and ABABCAB contain stacking faults since they do not strictly follow a cubic or hexagonal stacking structure. The number of layers involved in a stacking fault can vary, but it is typically low compared to the total number of layers within the crystal structure. An example of a stacking fault is illustrated in Figure 3.2.

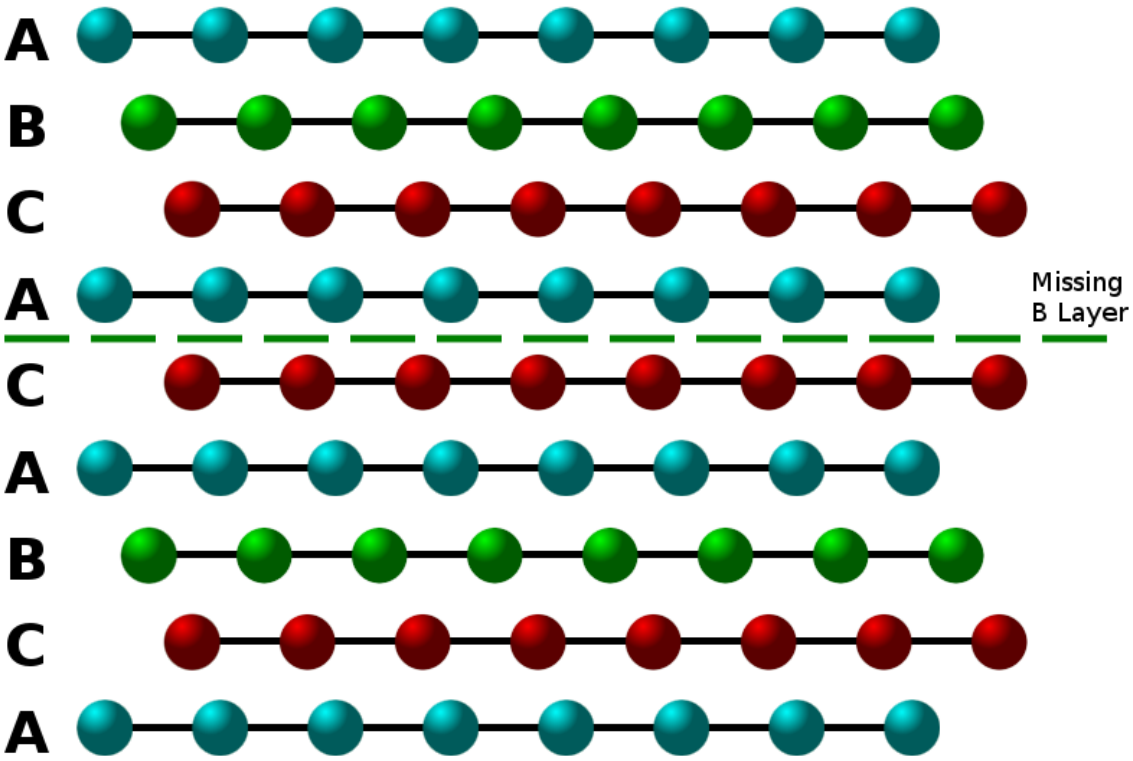
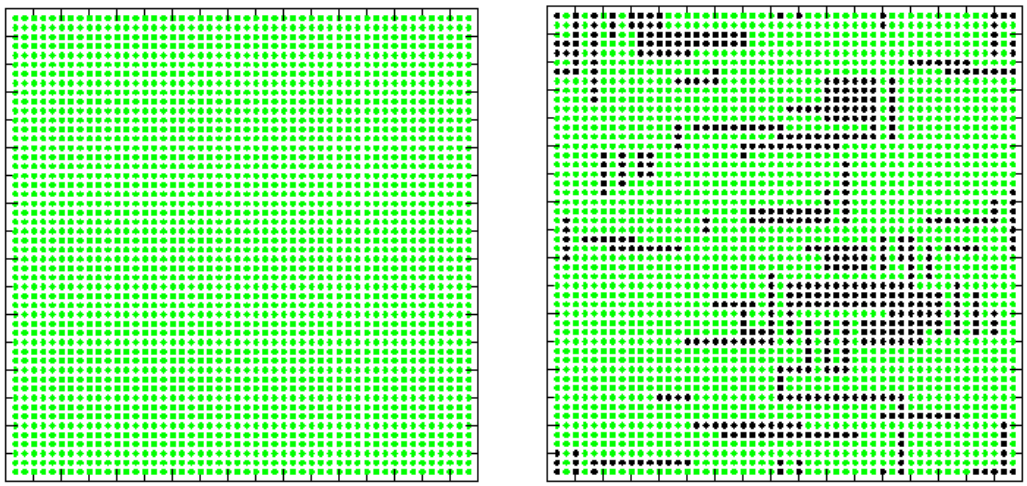


Figure 3.2: Example of HCP stacking fault within CCP structure.

Short-Range Order (SRO)

A solid material can be described as crystalline if it exhibits the same periodic structure throughout the entire crystal. This periodicity is referred to as long-range order (Carter and Norton, 2013). However, small areas of disorder can occur within a crystal structure and create short-range order (SRO) defects. In a SRO defect, the crystal structure is disturbed via either displacement or occupation. This behavior can include tendencies for similar atoms or vacancies to cluster together which are defined by correlation factors for the x-axis and the y-axis of the planar crystal (Neder and Proffen, 2008). A positive correlation factor leads to more clustering of atoms/vacancies whereas a negative correlation factor causes the atoms/vacancies to intersperse throughout the crystal. Figure 3.3 gives examples of long-range and short-range order within a simple cubic crystal structure.



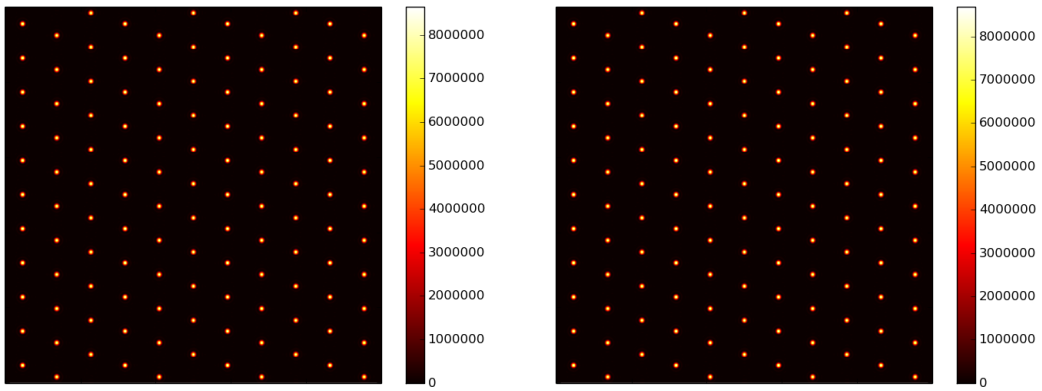
(a) Long-Range Order **(b)** Short-Range Order

Figure 3.3: Examples of long-range order (left) and short-range order (right). Each green dot represents a cell containing a single atom, and each black dot represents a vacancy.

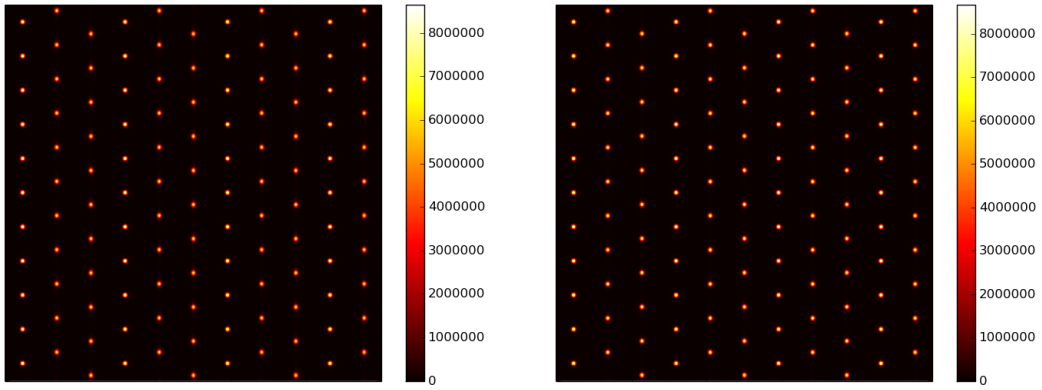
3.3 Dataset Information

The dataset used in this chapter was simulated using the DISCUS simulation package developed at Los Alamos National Laboratory (Proffen and Neder, 1997). The structure simulated was a 100 cell by 100 cell planar lattice with each cell containing a single silicon atom. Various defects and stacking configurations were introduced into the structure and the simulated reciprocal space image was generated for the modified crystals. In order to thoroughly test the capability of the defect detection methodology, a dataset was designed such that it contained crystals with no defects, stacking faults, or SRO for both HCP and CCP crystal structures. These crystal properties created a total of six possible combinations of the close-packed structure types and defects with each of the six combinations containing a total of 100 samples. The output from each DISCUS simulation was the reciprocal space image for the crystal, which was stored as a monochrome intensity map. Figures 3.4 and 3.5 provide examples of the defect classes to be analyzed in this chapter for both CCP and HCP structures, respectively.

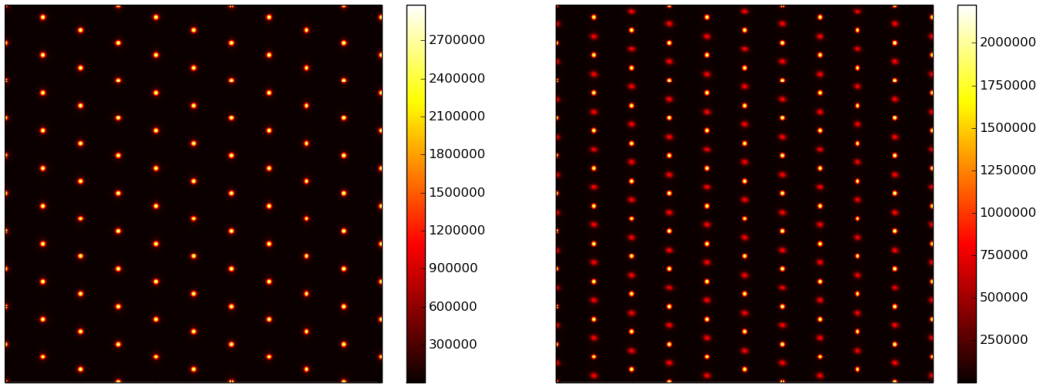
Visual inspection of the images reveals significant challenges raised by this defect detection problem as compared to the defect detection problem presented in Chapter 2.



(a) No Defect

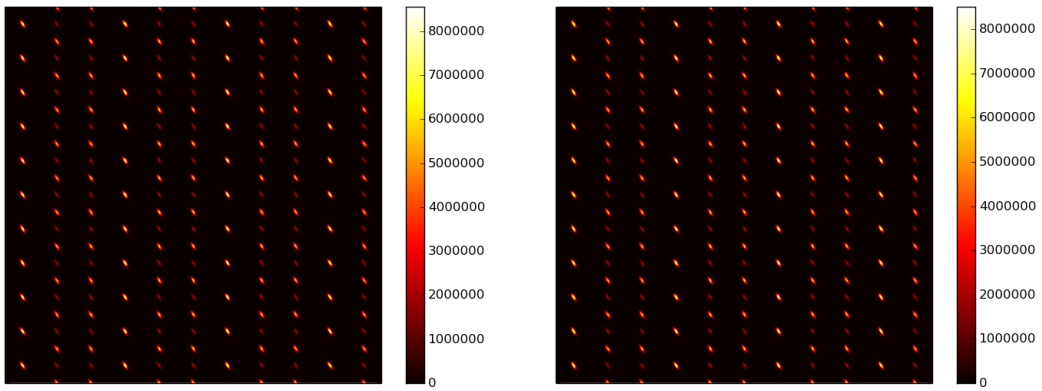


(b) Stacking Fault

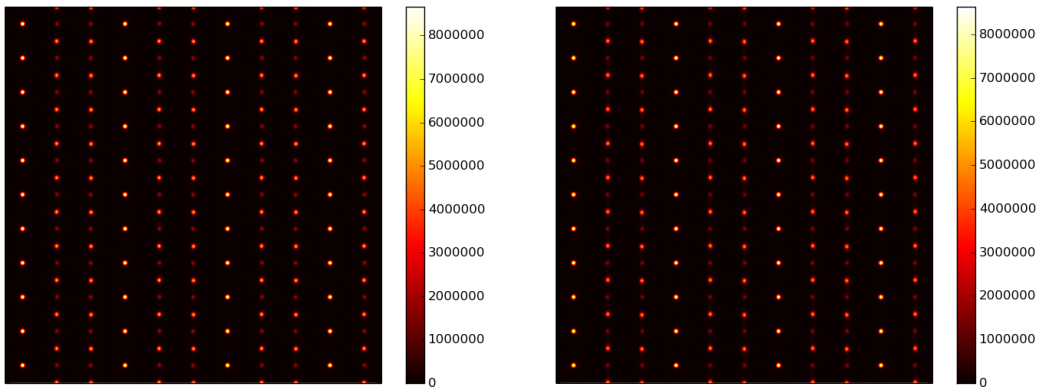


(c) Short-Range Order

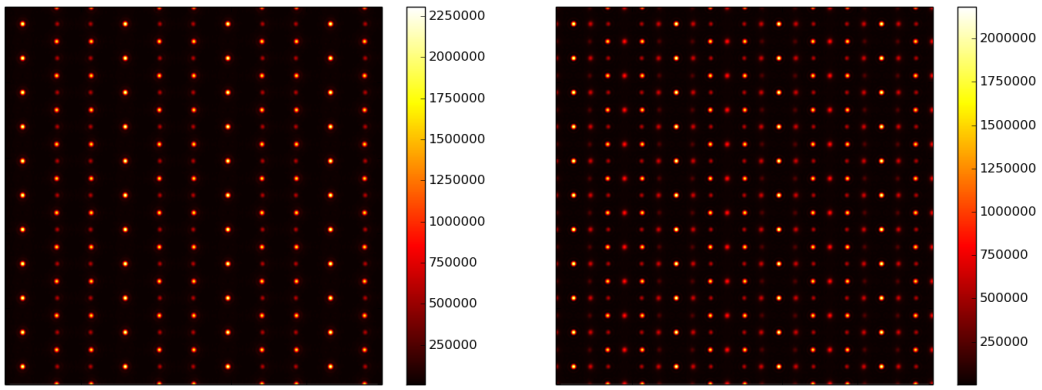
Figure 3.4: Representative reciprocal space images from the classes contained within the cubic close-packed crystal structure dataset.



(a) No Defect



(b) Stacking Fault



(c) Short-Range Order

Figure 3.5: Representative reciprocal space images from the classes contained within the hexagonal close-packed crystal structure dataset.

The most obvious challenge is that the patterns within the reciprocal space imagery can change depending on whether the structure is CCP or HCP. The changes between the "No Defect" and "Stacking Fault" image classes appear to be subtle at first glance, but some of the SRO images can be observed to contain visible changes in the image. However, what makes the SRO dataset troublesome from a classification point of view is that some of the visible changes can cause a CCP SRO image to look more like a HCP image containing a stacking fault or no defect at all. Therefore, this classification problem is much more complex than the problem presented in Chapter 2 and would be more difficult to solve via visual inspection alone. It should be noted that the intensity scales of the images can change from structure to structure and from defect type to defect type. However, this detail is of no consequence in the context of this work as keypoint extraction algorithms typically require that the intensities of the input images be rescaled to [0, 255].

3.4 Defect Detection Methodology

Given the description of the close-packed crystal structure defects presented in Section 3.2.2 and the description of the dataset provided in Section 3.3, the goal was to determine how to apply the defect detection methodology developed in Chapter 2 to this new, more complex dataset. However, a few modifications to the methodology were necessary before it could be applied to the close-packed crystal structure data.

First of all, precursory analysis of the dataset identified the need to rescale the intensity of the reciprocal space images for the close-packed crystal structures before descriptors could be extracted. The pixel intensity range for the images simulated using DISCUS was much larger than that of the images in Chapter 2. This large intensity range would most likely cause a keypoint extractor to focus in the intense Bragg peaks and ignore the textures in the diffuse data. Therefore, some sort of scaling methodology would be necessary in order to reduce the intensity range and focus on just the diffuse scattering data.

A second necessary modification was in the area of training the machine learning algorithm. The size of the reciprocal space images for the close-packed crystal structures

was 501 pixels by 501 pixels as compared to the smaller 129 pixel by 129 pixel images evaluated in Chapter 2. The larger image sizes could lead to a larger number of keypoints being generated as compared to the small structure tests in Chapter 2 thus greatly increasing the required training times and slowing down the defect detection system overall. Therefore, it was necessary to develop a scheme to reduce effect of the number of keypoints generated for these larger images.

In addition to the implementation of the above changes, alternatives to the SIFT keypoint extraction algorithm discussed in Chapter 2 were evaluated. The goal in testing these alternatives to SIFT was to determine if the accuracy or speed of the defect detection methodology could be improved by changing the keypoint extraction algorithm. This topic is discussed in more depth in later sections. In light of the proposed adjustments to the algorithm, the flowchart presented in Chapter 2 has been revised and is given in Figure 3.6.

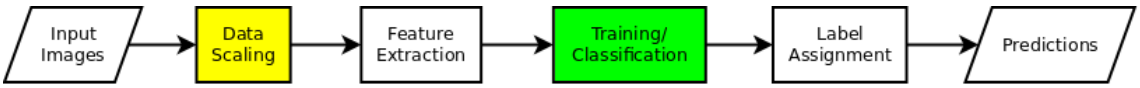
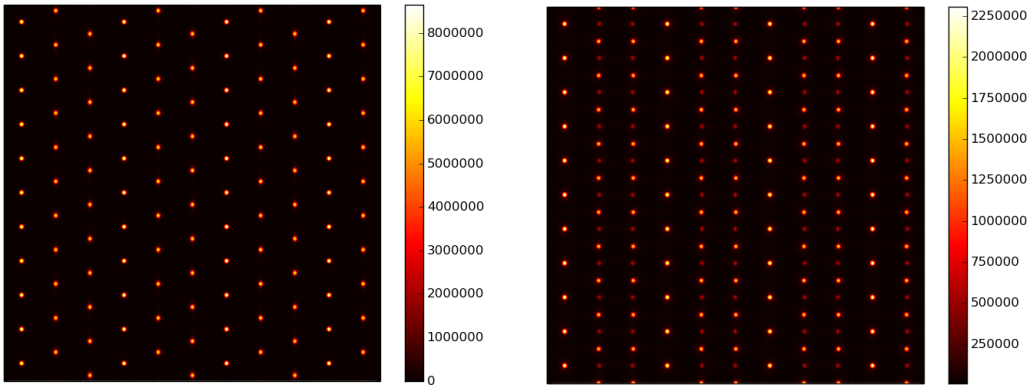


Figure 3.6: Flowchart for the revised defect detection methodology. The step highlighted in yellow was a necessary addition to the methodology presented in Chapter 2, and the step highlighted in green required was modified in order to address the larger volume of keypoints detected in the large structure data.

3.5 Data Preprocessing

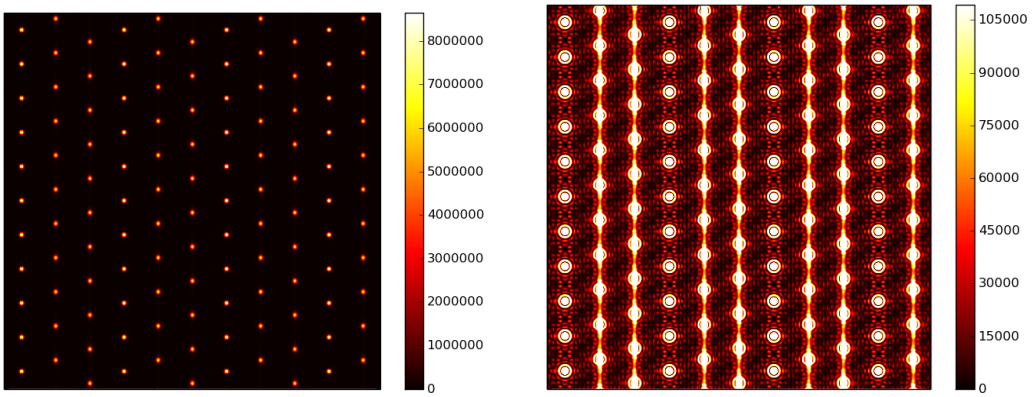
As mentioned in Section 3.4, one particular question to be answered was how to handle the large scale of the reciprocal imagery for the close-packed crystal structures. The goal for the feature extraction step was to capture the diffuse scattering textures in order to detect any potential defects within the crystals. However, precursory experimentation with the SIFT keypoint extractor further revealed that the large intensity of the Bragg peaks within the images made it difficult for the SIFT extractor to detect keypoints within the diffuse scattering regions of the images. Such problems could also potentially arise with other keypoint extractors. Therefore, it was necessary to develop a methodology to preprocess

the data in order to reduce the range of the intensities before any defect detection could be performed. Figure 3.7 gives a few examples of unscaled reciprocal space images that were used in the preprocessing experiments, and Figure 3.8 gives an example of the type of scaling desired as the result of this process. In the figure, intensity of the diffuse textures is increased so they are more visible to the keypoint extractor.



(a) **(b)**

Figure 3.7: Unscaled reciprocal space images.



(a) Before Scaling **(b)** After Scaling

Figure 3.8: Illustration of the effect of scaling on a sample reciprocal space image.

Further analysis of the images revealed that the pixel intensity range was on the order of $[0, 10^6]$. However, the distribution of the pixel intensities across the range was most definitely not uniform. Figure 3.9 is a log plot for a 10-bin histogram of the pixel intensities. It can be observed from the histogram that there are very few high-intensity

pixels, and a majority of the pixels fell in the low-intensity bins. With this in mind, the problem to be solved was determining the best way to rescale these high-intensity pixels without introducing texture distortions into the image. In order to accomplish this, it was determined that an intensity threshold should be calculated for the image and all pixels with intensities above the threshold should be rescaled in some manner. Such a solution would require that a threshold intensity be calculated in a data-driven manner such that it could be applied to an image without the need to be re-calibrated for each image.

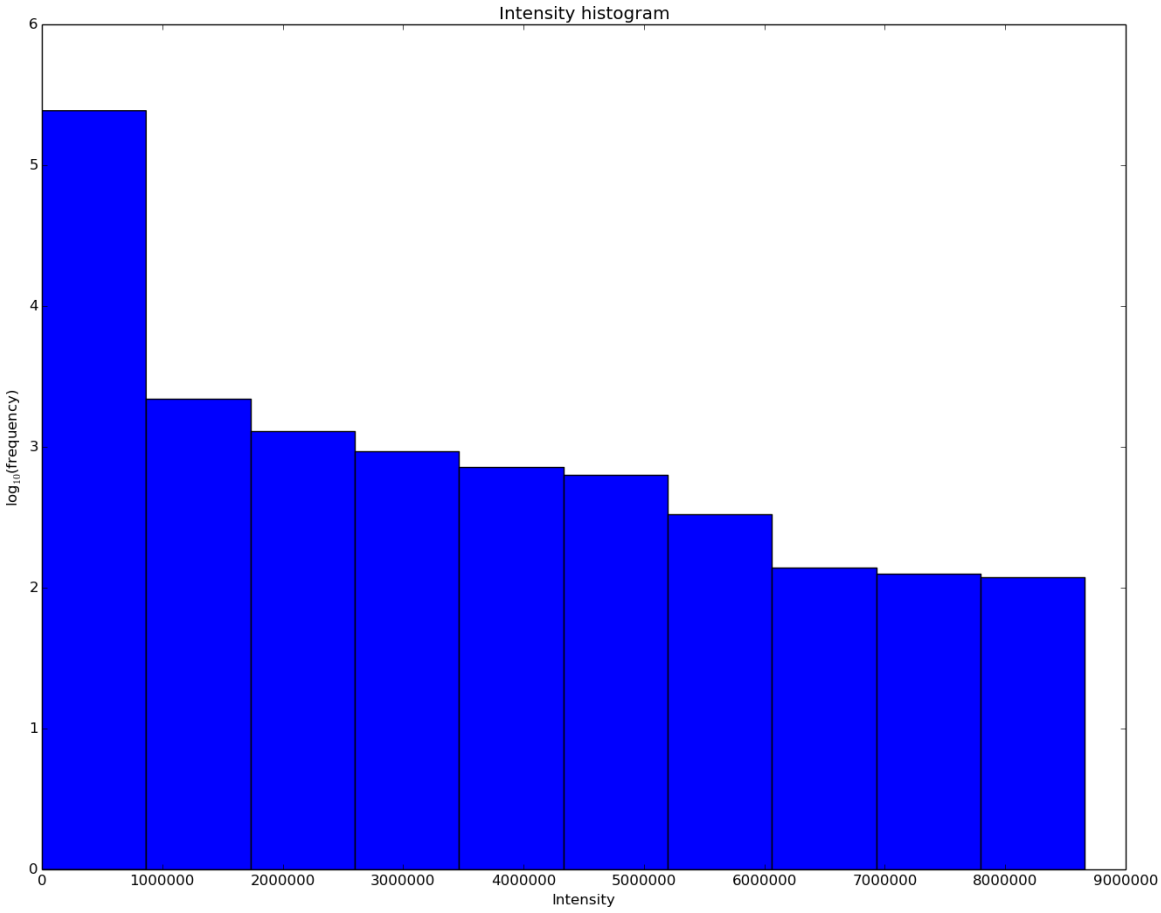


Figure 3.9: A 10-bin histogram (logarithm scale y-axis) for pixel intensities within a representative reciprocal space image.

In order to assist with the development of this data-driven scaling methodology, an interactive data visualization tool was created to facilitate the quick evaluation of a series of scaling methodologies for a reciprocal space image. A screenshot of this tool is provided

in Figure 3.10. The tool allows a user to load a reciprocal space image and use sliders to set the upper and lower intensity thresholds for a reciprocal space image and see the effects of the threshold on the image. The thresholds can be modified and the resulting scaled images displayed in real time. In addition, features are included that allow a user to calculate keypoints for an image and evaluate the effect of the threshold selection on the keypoint detector. Figure 3.11 shows the keypoint plotting mode for the tool. Other features are also available such as the ability to modify the color scale for the image and view the statistics for the pixel intensities. A menu containing a few scaling presets allows for quick analysis of multiple images. New presets can also be added by modifying the tool's code. The available threshold mode options "clamp" and "slice" control how pixels above the selected upper threshold are to be handled. Selecting the "clamp" option causes the program to set all pixels above the selected upper threshold to be equal to the threshold. Mathematically, this operation is equivalent to $p_{new} = min(p, t)$ for pixel intensity p and threshold intensity t. The "slice" option instead zeros the intensities of pixels with intensities above the threshold. Other options are also available via checkboxes in the left-hand panel that apply a log scale to the image intensities, display the slider values as percentages, and also apply a log scale to the slider movement for fine-tuned adjustment.

Given the features provided by the tool, a typical use case would be to load a reciprocal space image into the viewer window and use the sliders to interactively create and evaluate the keypoint detection for different threshold values and keypoint extractors. As the user evaluates thresholds for the image, the colormap selection menu could be used to apply various color scales in order to better visualize the textures in the data. The other features provided such as the threshold mode, checkbox options, scaling presets, and the data statistics box are available to make the overall experience more convenient for the user.

Using this tool, a few types of thresholds were evaluated. The initial threshold candidate tested set the threshold at a fixed percentage of the maximum intensity. This method was effective in some cases, but did not perform well if the scale of the intensity range for the image changed significantly. As shown in Figure 3.12a, setting the threshold as a fixed percentage of the maximum intensity properly scaled some images and accentuated

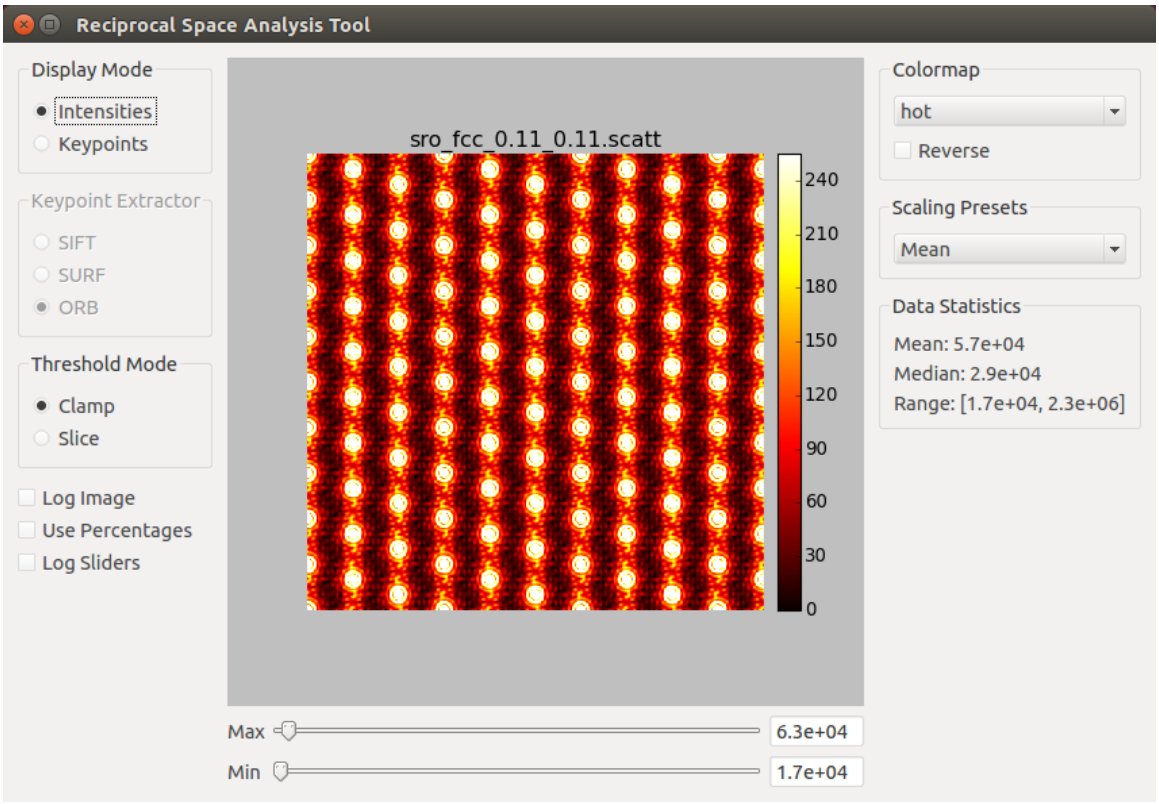


Figure 3.10: Screenshot of the reciprocal space image analysis tool displaying the intensity map for the reciprocal space image.

the diffuse scattering textures. However, images such as the one in Figure 3.12b set the threshold too low and caused the diffuse scattering textures to be "washed out".

In order to address this problem, the decision was made to set the threshold based on the median or mean intensity within an image. After extensive evaluation of the keypoint detection for the candidate thresholds, it was determined that a threshold of the mean pixel intensity for the image was sufficient to reduce the intensity of the Bragg peaks while not significantly affecting the textures within the diffuse scattering data. Evaluation of the keypoint detection for the images also revealed that the "slice" threshold option discussed above led to the detection of additional keypoints which did not constructively contribute to the defect detection process. This issue was due to the fact that the zeroing of values outside of the threshold boundaries introduced new irrelevant textures into the image instead of highlighting the textures within the threshold boundaries. Therefore, the "clamp" option

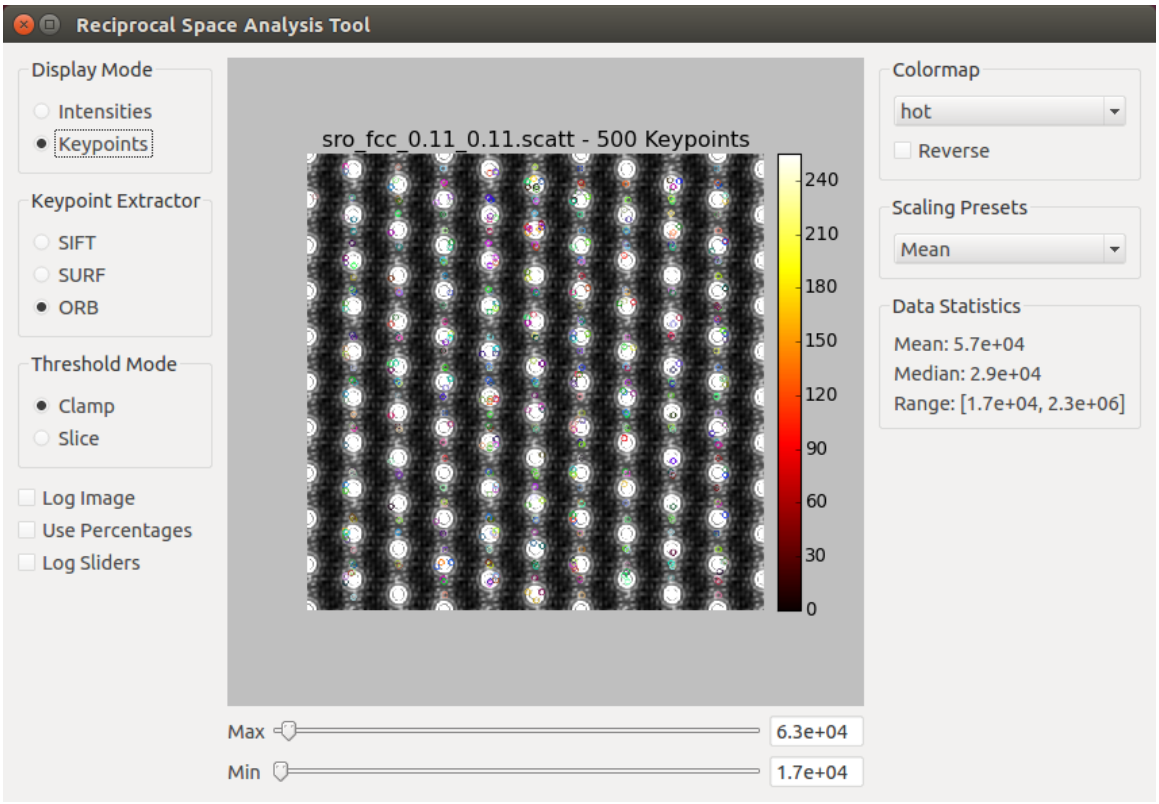
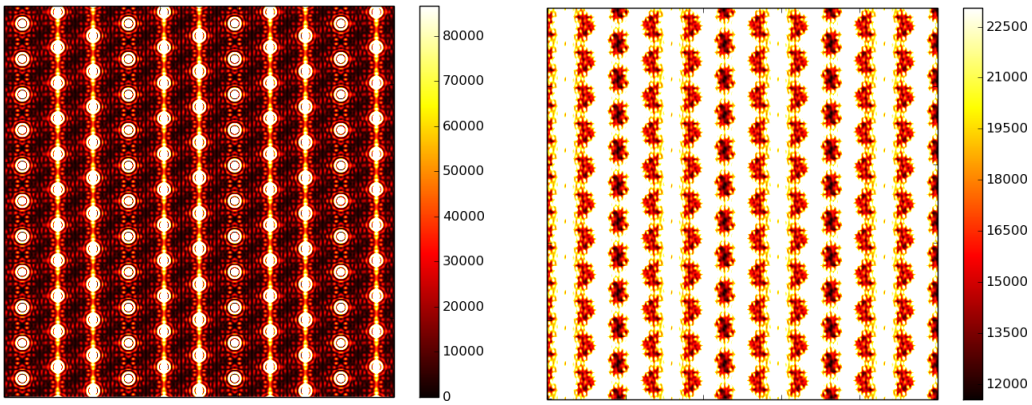


Figure 3.11: Screenshot of the keypoint plotting feature for the reciprocal space image analysis tool. The colored circles within the grayscale intensity image indicate the center of a keypoint identified by the keypoint detector.

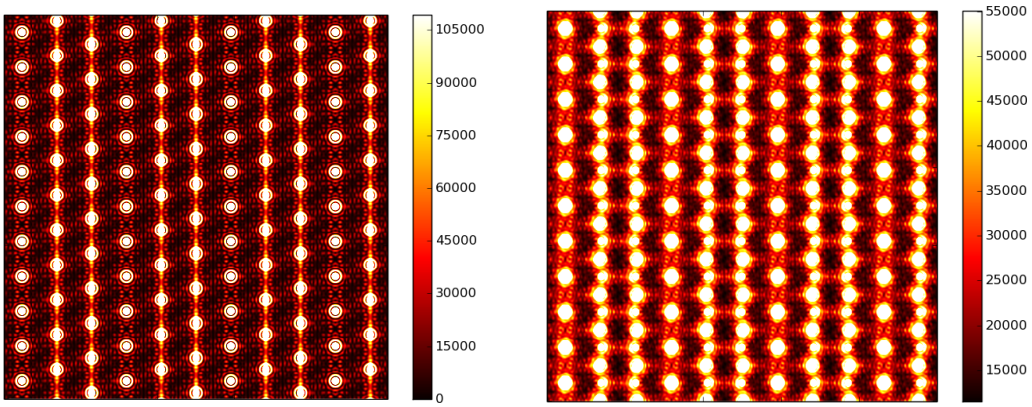
was selected for use in all experiments. Figure 3.13 shows the images given in Figure 3.7 scaled using this methodology. After scaling the images, it becomes apparent that there were subtle details within the diffuse data that were not visible in the images before the scaling was performed.

The code for the tool was written entirely in Python (van Rossum and Drake, 2011). The PyQt4 library (Harwani, 2011) was used to create the graphical user interface for the tool, Python bindings for the OpenCV library (Bradski, 2000) were used for keypoint extraction, and the matplotlib library (Hunter, 2007) was used to plot the reciprocal space images.



(a) **(b)**

Figure 3.12: Illustration of the shortcomings of applying a fixed percentage threshold to the images from Figure 3.7. Using a threshold that is a fixed percentage of the maximum produces sharp diffuse scattering textures in Figure 3.12a, but does not perform as well for the image in Figure 3.12b. These images were created by defining the threshold T to be 1% of the maximum intensity in the original image and scaling all pixel intensities I such that $I_{new} = min\,(I, T)$.



(a) **(b)**

Figure 3.13: Images from Figure 3.7 scaled using a threshold of the mean intensity. These images were created by defining the threshold M to be mean intensity for the original image and scaling all pixel intensities I such that $I_{new} = min\,(I, M)$.

3.6 Image Keypoint Extraction

As was the case in Chapter 2, keypoint extraction was used to extract features from the large structure reciprocal space images. In addition to again evaluating the SIFT keypoint extractor discussed in Section 2.3.1, other keypoint extraction methods were evaluated and

compared to SIFT. This section gives a brief overview of the key features of each keypoint extraction algorithm, but readers seeking more specific details on these methods should refer to the references for these algorithms provided in the following sections.

3.6.1 SURF: Speeded Up Robust Features

SURF is an alternative to SIFT that focuses on quicker execution speed and better feature quality (Bay et al., 2008). The SURF algorithm first uses an integer approximation method to speed up the keypoint detection methodology developed by SIFT. A descriptor is then calculated for a 20 pixel by 20 pixel patch centered at each keypoint. This patch is subdivided into a 4 by 4 grid, and sums of the horizontal and vertical Haar wavelet responses and their absolute values are calculated to generate a 4-dimensional vector for each grid cell. These 4-dimensional vectors for all 16 grid cells are then concatenated to create a 64-dimensional descriptor for the keypoint.

3.6.2 ORB: Oriented FAST and Rotated Brief Features

The ORB feature extractor was developed as an alternative to SIFT and SURF which boasts more robustness to image noise and real-time execution speeds (Rublee et al., 2011). It uses a combination of an oriented FAST (oFAST) corner detector and a rotation-aware BRIEF (rBRIEF) texture descriptor to detect keypoints and generate corresponding descriptors. (For more details on the implementation of oFAST and rBRIEF, interested readers can refer to the analysis of the ORB algorithm presented in Appendix A.) Once the keypoint locations have been identified by oFAST, ORB then selects the N "best" keypoints according to the Harris corner measure (Harris and Stephens, 1988a) and extracts rBRIEF texture features for a 31 pixel by 31 pixel patch centered at each of the N keypoint locations. The result is a 256-bit binary descriptor for each keypoint.

3.7 Machine Learning

As noted in Chapter 2, comparison of SVMs and random forests revealed that random forests performed better in the preliminary defect detection tests. Therefore, random forests were used exclusively for classification in this chapter.

However, even though random forests had good performance in the tests of Chapter 2, there were still obstacles to overcome when performing classification for the close-packed crystal structures. As noted in Section 3.4, the reciprocal space images for the close-packed crystal structures had much larger dimensions than the images in Chapter 2. As a result, more keypoints were generated for by the reciprocal space images for the close-packed crystal structures. Initial testing revealed that this increase in keypoints greatly increased the time required to perform classification for the new dataset as compared to the dataset evaluated in Chapter 2.

In order to address this larger volume of keypoints, the training process was modified such that the random forest was trained using a only a small, randomly-sampled subset of the detected keypoints for a given image. This modification greatly reduced the amount of time required to train the model without affecting the accuracy of the classifier in a significant way. During the testing phase all keypoints were assigned labels for a new testing image and a final classification was made using a majority vote all keypoints detected within the image as was the case in the experiments of Chapter 2.

3.8 Experiments

Using the modified defect detection methodology described in Section 3.4, a series of experiments was conducted to evaluate the accuracy of the methodology when detecting defects in the larger, more complex close-packed crystal structures, and to evaluate the quality of the descriptors generated by the keypoint extractors.

Each experiment tasked the random forest with learning to label each reciprocal space image as belonging to one of three disjoint classes: “no defect”, “stacking fault”, or “SRO”.

Note that there was no distinction made between whether the crystal was HCP or CCP in these classes. It was left up to the machine learning algorithm to learn to ignore the whether the crystal was HCP or CCP, and to focus solely on the presence of defects within the crystal. For each input image, the mean of the pixel intensity was calculated for the image and all pixels with an intensity higher than the mean intensity were set equal to the mean. After this upper bound was placed on the maximum intensity for the image, the intensities for the resulting image were then rescaled linearly to the range [0, 255] in preparation for feature extraction. In order to accelerate the classifier training process in the experiments, the machine learner only used 10% of the extracted keypoints from a given image as training inputs. All three of the keypoint extraction methodologies discussed in Section 3.6 were evaluated in a series of three experiments, and the results of each experiment were aggregated over 100 independent trials in order to filter any noisy experimental results. Results for the SIFT, SURF, and ORB experiments are available in Tables 3.1, 3.2, and 3.3, respectively.

Table 3.1: Aggregated confusion matrix for defect detection experiment using SIFT keypoint descriptors.

Actual Class \ Predicted Class	No Defect	Stacking Fault	SRO	Recall
No Defect	17314	669	52	96.00%
Stacking Fault	1015	16967	0	94.36%
SRO	122	107	17754	98.73%
Precision	93.84%	95.63%	99.71%	96.36%

Analysis of the tables shows that the revised defect detection methodology performed very well in classifying the presence of a defect (or lack thereof) within the crystal structures. In addition, all of the keypoint extraction methods had very high classification accuracy with SIFT yielding the highest accuracy, followed by ORB and SURF. Therefore, at this point one could potentially determine that SIFT is the "best" of the three

Table 3.2: Aggregated confusion matrix for defect detection experiment using SURF keypoint descriptors.

Actual Class \ Predicted Class	No Defect	Stacking Fault	SRO	Recall
No Defect	17791	208	0	98.84%
Stacking Fault	744	17286	0	95.87%
SRO	1459	1347	15165	84.39%
Precision	88.98%	91.75%	100.0%	93.04%

Table 3.3: Aggregated confusion matrix for defect detection experiment using ORB keypoint descriptors.

Actual Class \ Predicted Class	No Defect	Stacking Fault	SRO	Recall
No Defect	17656	328	43	97.94%
Stacking Fault	1070	16952	1	94.06%
SRO	766	1795	15389	85.73%
Precision	90.58%	88.87%	99.71%	92.59%

methodologies to use with this type of reciprocal space imagery due to its high accuracy. However, a discussion on the evaluation of the keypoint extraction algorithms will be deferred until Section 3.10.

3.9 Prediction Evaluation Criteria

As mentioned in Chapter 2, the tie-breaking for the voting scheme in experiments of Section 2.7 was arbitrary. There was opportunity for improvement in this area, and some mechanism needed to be developed to address the unlikely event of a tie occurring during

the keypoint voting step. This section discusses requirements for such a methodology for handling ties and proposes a potential solution.

As a first step in addressing how to handle a tie, it had to be determined exactly what should be the purpose of the tie-breaking mechanism. The primary goal of the defect detection methodology presented in this work is to reduce the amount of human evaluation that is necessary to detect defects within crystals. However, at this stage it is unrealistic to assume that a computational methodology could be developed which can detect any type of defect with perfect accuracy. Therefore, there should still be a mechanism in place by which a human expert can evaluate the quality of the predictions made by the classifier and flag predictions for analysis by a human expert if certain prediction confidence criteria are not met. This need is particularly relevant in the case of tie-breaking as a generalized machine learning-based methodology will not be acquainted with domain knowledge in crystallography which may be used by a human expert to classify a defect within a crystal.

Therefore, when considering a label that has been assigned to a sample, the real question to be asked is how much confidence should be placed in the label derived from the keypoint voting scheme. It is unreasonable to assume that a label assigned via tie (or near-tie) between two or more classes is as reliable as a label assigned via a unanimous vote. Instead, the classification provided via an "uncertain" classification should be used more as a heuristic which can guide a human expert in classification of "difficult" samples within a dataset. Thus instead of placing the burden of tie-breaking on the label assignment step, a means to describe the confidence of the vote was implemented such that every label assignment was accompanied by a confidence measure. In the experiments, this confidence measure was calculated as the percentage of keypoints which received the "winning" vote during the keypoint voting step. The advantage that such a confidence measure offered over a heuristic for assigning a label was that a confidence measure allows for the quality of label assignments to be measured. Therefore, for an experiment a researcher could potentially define a minimum confidence requirement for labels assigned to samples under analysis. If the confidence for an assigned label did not meet the minimum confidence threshold,

it could be flagged for for review by human experts. Table 3.4 summarizes the average confidence for the experiments presented in Section 3.8.

Table 3.4: Mean confidence for defect detection experiments.

Keypoint Extractor	Mean Confidence
SIFT	75.98%
SURF	81.61%
ORB	79.39%

Analysis of the average confidence measure for each experiment shows that the SURF experiment generated the highest confidence measures while the SIFT experiment yielded the lowest average confidence. It should be noted that this ordering is the exact opposite of the keypoint extraction algorithms when they were ranked by classification accuracy in Section 3.8. This observation is interesting because it shows that a high average confidence is not necessarily required to produce a high classification accuracy. However, a higher average confidence would definitely be preferred as it would reduce the number of samples that are flagged for human evaluation.

3.10 Keypoint Extractor Evaluation

Evaluation for keypoint extractors was conducted in a similar manner to the evaluation of the machine learning algorithms in Chapter 2. The keypoint extractors were evaluated based on two criteria: feature quality and extractor scalability.

The feature quality was evaluated with respect to the ability of the machine learning algorithm to use the features to distinguish between the different defect classes. In this regard, the feature quality for the three extractors was roughly the same with a slight advantage going to the SIFT extractor due to a slightly higher classification accuracy observed in the SIFT experiments.

Regarding extractor scalability, due to the fact that features are extracted on an image-by-image basis, the scalability of each feature extraction algorithm will be with respect to

the image dimensions. (Drews et al., 2011) has shown that the complexity of the SIFT and SURF algorithms as the image dimensions increase is $O(mn + k)$ for an image with dimensions m pixels by n pixels containing k keypoints. However, this complexity can be simplified to $O(mn)$ since mn is much less than k in the images used in this work. Similarly, analysis of the complexity of the ORB algorithm given in Appendix A reveals that the simplified complexity of the ORB algorithm is also $O(mn)$ for an image with dimensions m pixels by n pixels. Thus the scalability of the SIFT, SURF, and ORB methods are approximately the same with respect to the image dimensions.

However, this observation that the scalability of the keypoint extractors is the same with respect to the image size does not imply that their execution time is the same when used to detect keypoints for a series of images. Figure 3.14 shows runtime benchmark graphs for a single run of the classification methodology using each of the three keypoint extraction algorithms. The computer used in these experiments utilized a 2.66 GHz quad core processor with 10 GB of RAM. Analysis of the graph reveals some interesting observations regarding keypoint extraction.

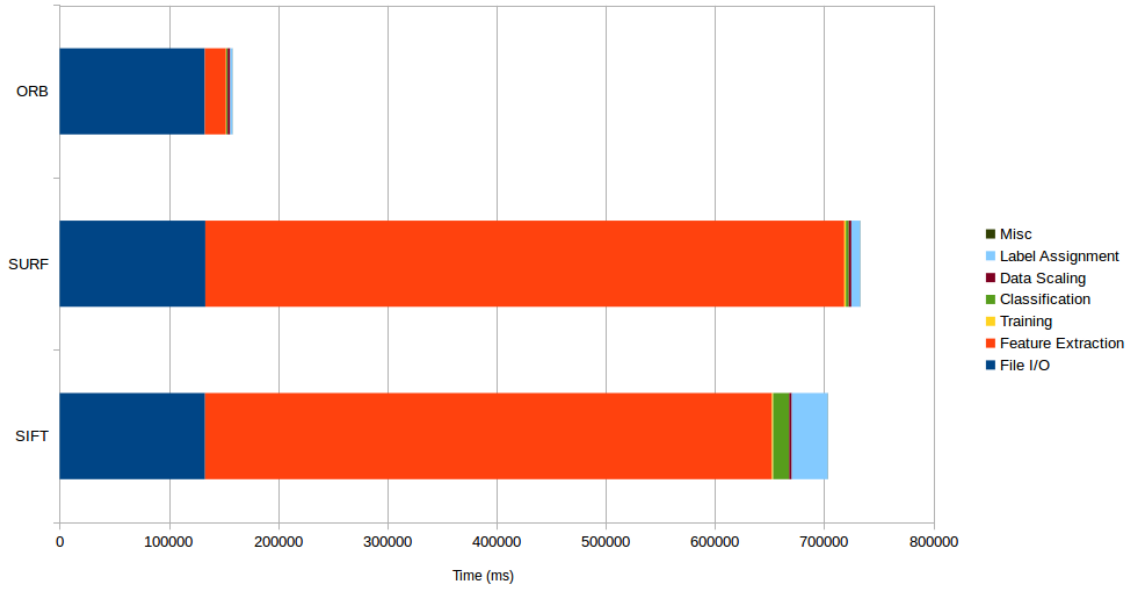


Figure 3.14: Runtime graphs for the experiments described in Section 3.8.

First of all, even though the runtime complexity of the keypoint extraction algorithms is the same as the image size increases, the amount of time required for each algorithm to extract keypoints for a set of images varies. The benchmark graph in Figure 3.14 shows that tests using ORB spend the smallest amount of time performing keypoint extraction with SIFT requiring approximately 27 times the extraction time required by ORB and SURF requiring the most time at 30 times the extraction time of ORB. With this in mind, the ORB algorithm has a distinct advantage over the other methods as it runs faster than the other methods while scaling at approximately the same rate as the image dimensions increase.

The second observation is that the choice of keypoint extraction algorithm affects the running time for the subsequent steps of the methodology (e.g., training, classification, and label assignment). This difference in running times is due to the fact that the dimensionalities of the descriptors generated by the keypoint extractors are different for all three algorithms: ORB has a 32-dimensional descriptor (256 binary descriptors stored as 32 8-bit integers), SURF has a 64-dimensional descriptor, and SIFT generates a 128-dimensional descriptor. A longer descriptor increases the complexity of the problem to be solved by the machine learning algorithm which leads to longer execution times for training the classifier and assigning labels to new images. It is reasonable to evaluate longer descriptors which could potentially contain more information and lead to higher classifier accuracies, but analysis of the results for the experiments shows that there is not a substantial benefit offered by using a longer descriptor. It should be noted that SIFT did in fact produce a slightly higher accuracy than ORB in the experiments, but this improvement in accuracy via SIFT was at the cost of requiring approximately 25 times more processing time than was required by ORB for feature extraction, training, classification, and label assignment. Therefore, when choosing a keypoint extractor it is necessary to consider the costs/benefits of each method and determine if a slight increase classification accuracy is worth the cost of a longer execution time.

3.11 Summary

The discussion and experiments of this chapter has shown that the crystal defect detection methodology developed in Chapter 2 for a simple single crystal neutron scattering dataset can be adapted for use with data simulated by a sophisticated neutron scattering simulator. In addition, it has been shown that the methodology is also capable of detecting and distinguishing between more complex types of defects within a larger crystal structure.

Analysis of the need for preprocessing the neutron scattering data was also performed, and a methodology to scale the data in preparation for processing by the defect detection methodology was developed. This methodology was designed such that it reduced the intensity of the Bragg peaks within the reciprocal space imagery while accentuating the diffuse scattering patterns which describe the defects within the crystal. A tool has been developed using Python which assisted with the development of this scaling methodology.

Alternatives to the SIFT keypoint extractor were evaluated, and the strengths and weaknesses of the extractors were evaluated and discussed. A comparison of the classification accuracies resulting from the use of the extractors was presented as well as a comparison of the scalability of the keypoint extraction algorithms themselves. The result of this analysis was the conclusion that the keypoint extractor should be selected based on whether the user is wanting to maximize accuracy or minimize the time required to complete the classification.

The question of tie-breaking in the keypoint voting scheme and prediction confidence were also discussed. An in-depth inquiry into the goals for the use of the defect classifier and the role of human experts in this process was also presented. The result of this analysis yielded the development of a measure by which a confidence value can be assigned to a prediction for the defect present within a sample. This confidence measure can be used human experts to identify samples for which the methodology is uncertain of the prediction and flag these samples for human evaluation.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

It has been shown in this work that crystal defects can be detected via computational analysis of reciprocal space imagery generated by simulated single crystal neutron scattering experiments. The proposed defect detection methodology used a keypoint-based feature extractor to generate texture features describing the most relevant portions of the images. It then used a supervised machine learning algorithm to analyze the feature vectors and automatically classify defects. The methodology trained the classifier using individual keypoint features extracted from the training images and then used a keypoint voting scheme to produce classifications for new testing images.

In addition to developing the defect detection methodology, the choice of machine learning algorithm used to perform classification within the methodology was explored and advantages of certain classifiers were noted. A number of keypoint-based image feature extraction methods were also evaluated and were shown to generate features from the reciprocal space images that were rich enough to describe the aspects of the images that indicate the presence of a defect within the crystal. The advantages of specific keypoint extraction algorithms were also assessed. As part of the deployment process for the keypoint feature extractors, a preprocessing methodology was developed for the reciprocal

space images that reduced the intensity of the Bragg peaks within the image and caused the diffuse textures to be more pronounced. A software tool was developed that assisted with the development of this preprocessing scheme.

In order to validate the defect detection methodology presented in this work, it was tested on a series of increasingly difficult problems using reciprocal space data simulated using multiple techniques. These problems started with a small crystal structure containing toy defects and then scaled up to a larger structure containing more complex defects. As the defect detection problems became more difficult, new insight was gained into how to improve the defect detection process and which computational methods performed the best overall. Areas of future work were also identified during the experimentation process and are presented in the next section.

4.2 Future Work

The following topics are areas of interest that could be potential topics of ongoing research in the area of computational crystal defect detection within single crystal neutron scattering experiments.

4.2.1 Real Data Analysis

The most relevant topic of future work is testing the defect detection methodology with data generated by real scattering experiments. All of the data in this work was simulated, and it was thus free from noise that could occur in a real scattering experiment. Therefore, it would be instructive to test the defect detection methodology on a dataset consisting of experimentally generated reciprocal space images in order to further evaluate the performance of the methodology and adapt it for use with experimental data if necessary.

4.2.2 Experimentation with Multiple Defects

The crystal structures analyzed in this work were limited to only one type of defect for the entire crystal structure. An interesting exercise would be to relax this limitation and allow for multiple types of defects to be present within a single crystal. Given that the reciprocal space images are magnitude maps of the neutron scattering patterns, it is unlikely that the patterns generated by a combination of multiple types of defects will translate to a linear combination of the individual scattering patterns for each defect. Therefore, much work will need to be done in order to determine the best way to handle multiple defects within a single crystal.

4.2.3 Defect Texture Analysis

The methodology developed in this work analyzed textures within patches extracted from reciprocal space images and proved to work well for the datasets that were tested in the experiments. However, it is not clear which properties of the textures the classifier should deem important when designating a keypoint descriptor as containing a specific type of defect. Analysis of the textures within the patches identified by the keypoint detector could potentially answer a number of questions about texture "signatures" that are unique to a certain type of defect and could be used to more reliably identify a certain type of defect within a crystal. Such analysis could also be useful in solving problems such as the 3-class experiments presented in Section 2.7. In these experiments, there was an issue with the classifier becoming confused when distinguishing between small and large substitution defects. Analysis of the texture patches extracted by the keypoint detector could potentially assist in developing a means to reduce the amount of confusion between the classes in these experiments.

4.2.4 Sensitivity Quantification

The experiments in this work evaluated a number of types of defects of varying sizes. An interesting exercise would be to quantify exactly how sensitive the defect detection methodology is to the severity of the defect within the crystal. While this may seem like a straightforward problem, there may be many parameters that would need to be considered when constructing tests for evaluation of the sensitivity of the defect detection methodology. For example, in the case of stacking faults one would have to first determine if it is important to consider stacking faults of n consecutive layers as equivalent to n single-layer faults that are not consecutive. If these two cases cannot be considered as equivalent, the question to be answered is which sequence is more difficult for the classifier to detect. Similar questions would more than likely arise with other types of defects as well. Another area to be considered is the sensitivity with respect to the specific atom types contained within the crystal. It is currently unknown whether crystals containing a particular element would be more suitable for use with this methodology as compared to crystals containing other elements.

4.3 Summary of Contributions

The following is a summary of contributions made by this dissertation:

1. Evaluation of a data processing methodologies for use with simulated reciprocal space imagery for single crystal diffuse neutron scattering experiments.

2. Analysis of characteristics of reciprocal space imagery dataset.

3. Development of scaling methodology for use with simulated reciprocal space imagery.

4. Creation of a graphical user interface that can be used to assist with analysis of the intensities within reciprocal space images.

5. Formalization of a methodology that performs automatic defect detection within crystals by analyzing reciprocal space imagery. The methodology was evaluated using simulated single crystal diffuse neutron scattering experiments for the following defect classes:

   (a) Identification of simple defect types for small simulated crystal structures.

   (b) Prediction of substitution location for small simulated crystal structures.

   (c) Detection of defects within simulated close-packed crystal structures.

6. Comparison of keypoint extractor and machine learner performance in the context of the defect detection methodology presented in this work.

7. Runtime complexity analysis for the ORB feature extraction algorithm.

Bibliography

Ayers, B. and Boutell, M. (2007). Home interior classification using sift keypoint histograms. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–6. 8

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359. 44

Borchardt-Ott, W. (2012). *Crystallography: An Introduction*. Springer-Verlag Berlin Heidelberg. 2

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA. ACM. 15

Bottou, L. and Lin, C.-J. (2007). Support vector machine solvers. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large Scale Kernel Machines*, pages 1–28. MIT Press, Cambridge, MA. 27

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 21, 42

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140. 17

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32. 17

Butler, B. D. and Welberry, T. R. (1992). Calculation of diffuse scattering from simulated disordered crystals: a comparison with optical transforms. *Journal of Applied Crystallography*, 25(3):391–399. 3, 4, 17

Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C., and Fua, P. (2012). BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298. 65, 67

Carter, C. and Norton, G. (2013). *Ceramic Materials: Science and Engineering*. SpringerLink : Bücher. Springer. 32

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 21

Chiang, Y., Birnie, D., and Kingery, W. (1996). *Physical Ceramics: Principles for Ceramic Science and Engineering*. MIT series in materials science and engineering. Wiley. 2, 30

Drews, P., de Bem, R., and de Melo, A. (2011). Analyzing and exploring feature detectors in images. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 305–310. 50

Egami, T. and Billinge, S. (2012). *Underneath the Bragg Peaks: Structural Analysis of Complex Materials*. Pergamon Materials Series. Elsevier Science. 4, 6, 8

Evans, R. (1964). *An Introduction to Crystal Chemistry*. Cambridge University Press. 2

Harris, C. and Stephens, M. (1988a). A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 23.1–23.6. Alvety Vision Club. 44

Harris, C. and Stephens, M. (1988b). A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK. 66

Harwani, B. M. (2011). *Introduction to Python Programming and Developing GUI Applications with PyQT*. Course Technology Press, Boston, MA, United States, 1st edition. 42

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95. 42

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110. 13

Martin, B. W. and Vatsavai, R. R. (2013). Image change detection via ensemble learning. volume 8743, pages 874305–874305–7. 8

Neder, R. B. and Proffen, T. (2008). *Diffuse Scattering and Defect Structure Simulations: A Cook Book Using the Program DISCUS*. Oxford University Press, Inc., New York, NY, USA. 30, 31, 32

Nield, V. and Keen, D. (2001). *Diffuse Neutron Scattering from Crystalline Materials*. Oxford science publications. Clarendon Press. 4

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. 21

Proffen, T. and Neder, R. B. (1997). *DISCUS*: a program for diffuse scattering and defect-structure simulation. *Journal of Applied Crystallography*, 30(2):171–175. 33

Pynn, R. (2008). Neutron scattering – A non-destructive microscope for seeing inside matter. In Liang, L., Rinaldi, R., and Schober, H., editors, *Neutron Applications in Earth, Energy and Environmental Sciences*. Springer. 4, 6

Rosten, E. and Drummond, T. (2005). Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511. 65

Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443. 65

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision*, Barcelona. 44, 64

Schober, H. (2008). Neutron scattering instrumentation. In Liang, L., Rinaldi, R., and Schober, H., editors, *Neutron Applications in Earth, Energy and Environmental Sciences*. Springer. 2

Sun, P.-L., Zhao, Y., Cooley, J., Kassner, M., Horita, Z., Langdon, T., Lavernia, E., and Zhu, Y. (2009). Effect of stacking fault energy on strength and ductility of nanostructured alloys: An evaluation with minimum solution hardening. *Materials Science and Engineering: A*, 525(12):83 – 86. 1

van Rossum, G. and Drake, F. L. (2011). *The Python Language Reference Manual*. Network Theory Ltd. 42

Welberry, T. R. and Goossens, D. J. (2014). Diffuse scattering and partial disorder in complex structures. *IUCrJ*, 1(6):550–562. 1, 6

Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition. 27

Zhan, H., Zhang, Y., Bell, J. M., and Gu, Y. (2014). Thermal conductivity of Si nanowires with faulted stacking layers. *Journal of Physics D: Applied Physics*, 47(1):015303. 1

Appendix

Appendix A

Complexity Analysis of the ORB Keypoint Extraction Algorithm

A.1 Introduction

This appendix provides additional analysis of the ORB keypoint extraction algorithm as described in (Rublee et al., 2011). The overall goal of this analysis is to determine the computational complexity of the ORB algorithm in order to support the comparison to the SIFT and SURF algorithms given in Section 3.10. The structure of this appendix is as follows: A detailed description of the ORB algorithm is presented, the computational complexities of the different components of the algorithm are calculated and used to determine the overall computational complexity of the ORB algorithm, assumptions are discussed which can be used to simplify the ORB complexity calculation, and finally a conclusion is presented summarizing the findings of this appendix.

A.2 ORB Algorithm Summary

In general, a keypoint extraction algorithm comprises of two phases: a keypoint detection phase, and a keypoint extraction phase. The ORB algorithm utilizes modifications two

different algorithms to separately handle the detection and extraction phases. For keypoint detection, ORB uses oFAST which is an oriented variant of the FAST corner detection algorithm (Rosten and Drummond, 2005, 2006). Once the oFAST detection algorithm has produced a set of keypoints for the image, ORB uses a rotation-aware version of the BRIEF feature descriptor (Calonder et al., 2012) called rBRIEF to generate a descriptor that describes the texture of a 31 pixel by 31 pixel patch centered at the keypoint location. The following subsections describe details of the oFAST and rBRIEF algorithms relevant to the calculation of the computational complexity of the ORB algorithm. As mentioned before in Chapter 3, interested readers seeking more details on the ORB algorithm should refer to the references provided in this appendix.

A.2.1 oFAST: Oriented FAST

FAST is an algorithm that detects corners within a monochrome image by comparing the intensity of a pixel being tested for "cornerness" to surrounding pixels within the image (Rosten and Drummond, 2005, 2006). This is accomplished by measuring the intensities of a "ring" of pixels surrounding the pixel in question. If the intensities of a segment of N contiguous pixels that lie on the ring differ from the center pixel by more than a set difference threshold, then the region surrounding the pixel is considered to be a corner. The size of the ring that is used by the FAST algorithm can vary, but a ring containing 16 pixels and a segment of length $N = 9$ contiguous pixels is used in the ORB algorithm. Figure A.1 illustrates the detection of a corner in the FAST step. In the figure, the pixel being tested is marked by a white frame. Pixels exceeding the difference threshold are marked by blue frames, and pixels not exceeding the difference threshold are marked by red frames. There are 11 pixels that exceed the difference threshold, and therefore FAST will designate a pixel patch centered at the white-framed pixel as a corner.

One shortcoming of the FAST algorithm is that there is not a measure of the cornerness for the detected corners. Thus the authors of the ORB algorithm noted large responses by FAST along edges as well as corners. In order to address this issue, the ORB algorithm

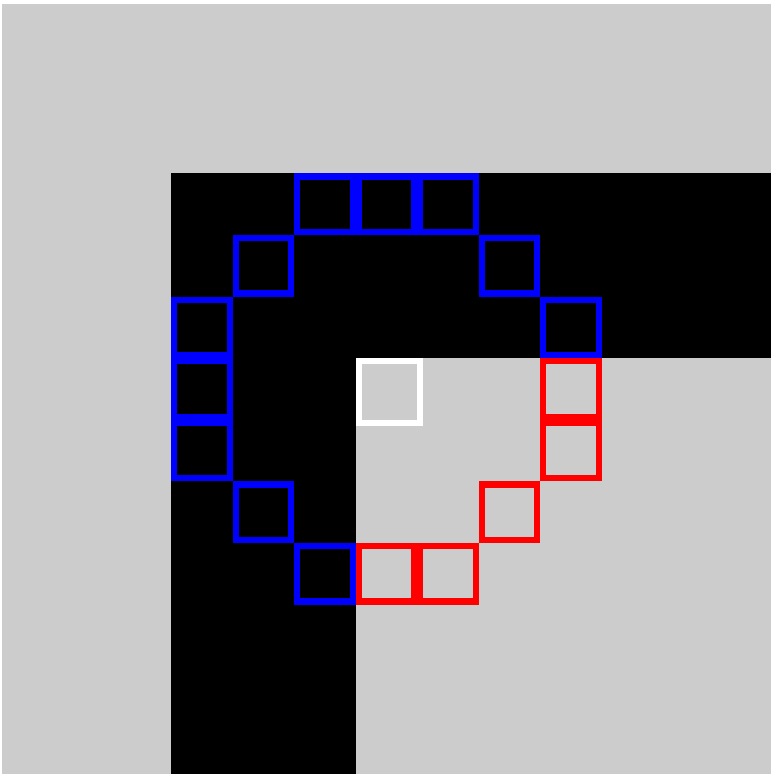


Figure A.1: Corner detection using FAST.

ranks all detected "corners" using a Harris corner measure (Harris and Stephens, 1988b). Once the detected corners have been ranked by cornerness using the Harris measure, the ORB algorithm selects the top k as the keypoints for the image. In addition, ORB generates scale-invariant features by using a scale pyramid containing S scales to generate stable keypoints for varying levels of blurring for the image.

In order to ensure rotation invariance of the keypoints detected in the image, the orientation of the keypoint needs to be considered during the feature generation step. The standard FAST algorithm does not provide a means to measure the corner orientation, and thus ORB algorithm calculates an intensity centroid for each keypoint that it uses to determine the orientation of the corner. This provides the "oriented" component of the "oriented FAST" algorithm.

A.2.2 rBRIEF: Rotation-Aware BRIEF

Once the keypoints and their orientations have been detected by the oFAST algorithm, descriptors need to be extracted that describe the texture of the image at the keypoint location. ORB uses a variant of the BRIEF feature extraction algorithm (Calonder et al., 2012) to generate features for the detected keypoints. The BRIEF algorithm generates pixel location pairs within a patch centered at the keypoint location. These pixel pairs are generated such that pixel coordinates (x, y) are on the distribution $(X, Y) \sim$ i.i.d $Gaussian(0, \frac{1}{25}S^2)$ where S is the size of the square pixel patch. In the case of rBRIEF, the patch size is typically chosen as $S = 31$, and a total of 256 pixel pairs are generated for each patch.

After the pixel pairs are generated, rBRIEF extracts pixel patches centered at each keypoint location detected by oFAST and smooths them using a 5 pixel by 5 pixel window. For each pixel patch, rBRIEF uses a rotation matrix to "steer" the positions of the pixel pairs in the direction of the patch orientation detected by oFAST. This steering step is a key feature of rBRIEF and improves on the standard BRIEF feature via the addition of rotation invariance.

Once the pixel pairs have been generated and steered, rBRIEF then defines the binary features to be comparisons of the intensities of the pixel pairs using the comparison rule given in Equation A.1. In the equation, τ is the binary feature value, $\mathbf{p}$ is the pixel patch, $\mathbf{p}(\mathbf{x})$ is the pixel intensity of pixel at location $\mathbf{x}$, and $\mathbf{p}(\mathbf{y})$ is the pixel intensity of pixel at location $\mathbf{y}$.

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) = \begin{cases} 1, & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0, & \text{if } \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases} \tag{A.1}$$

Finally, the binary feature vector is created by creating a vector of 256 bits containing the binary features created by the comparisons. The equation used to achieve this is given in Equation A.2.

$$f_n(\mathbf{p}) = \sum_{1 \leq i \leq n} 2^{i-1} \tau\left(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i\right) \tag{A.2}$$

A.3 ORB Complexity Analysis

Now that the components of the ORB algorithm have been described, the computational complexity of the ORB algorithm can be determined. In order to assist with the analysis of the components of the ORB feature extraction process, pseudocode for the oFAST and rBRIEF algorithms is provided in Algorithms 1 and 2, respectively.

Algorithm 1 oFAST Algorithm

```
for each scale S do
  for each pixel at location (x, y) do
    Compare intensity of pixel to 16 pixel ring surrounding it
    Designate pixel as corner if 12 contiguous pixels in ring are lighter/darker than it
  end for
  Rank cornerness of all d detected corners using Harris corner measure
  Designate k highest ranked corners as keypoints
  for each keypoint do
    Detect orientation of keypoint using intensity centroid
  end for
end for
```

Algorithm 2 rBRIEF Algorithm

```
Generate i.i.d. sample pairs using Gaussian distribution
for each keypoint detected by oFAST do
  Extract a 31x31 patch at keypoint center
  Smooth patch using 5 by 5 window
  "Steer" sample pairs toward orientation detected by oFAST
  Calculate feature vector from sample pair comparisons
end for
```

Inspection of the oFAST algorithm in Algorithm 1 reveals that the first inner for loop evaluates the cornerness of each pixel in the image. Therefore, the worst-case computational complexity of the loop is $O(mn)$ for an image with dimensions of m pixels by n pixels. This loop is followed by a ranking step that involves sorting the corners

the Harris corner measure. The ranking step involves sorting the corners and has the complexity $O(d * log(d))$ for d detected corners. Finally, the top k corners are selected as keypoints and the orientation is detected. Orientation detection is a constant-time operation and thus the last inner for loop has the complexity $O(k)$. Therefore, total complexity for the oFAST algorithm is $O(S(mn + d * log(d) + k))$ for an image with dimensions of m pixels by n pixels, d detected corners, a threshold of k keypoints, and S scales.

Analysis of the rBRIEF algorithm is a bit simpler. Algorithm 2 shows that there is a single for loop that executes a series of constant-time steps. Therefore, the complexity of rBRIEF is $O(k)$ for k detected keypoints.

Combining the analysis of the oFAST and rBRIEF algorithms shows that the computational complexity of the ORB algorithm is $O(S(mn + d * log(d) + k) + k)$ for an image with dimensions of m pixels by n pixels, containing d detected corners, and a threshold of k keypoints.

It should be noted that there are some simplifications that can be made to this computational complexity equation. More specifically, the number of keypoints k and the number of detected corners d is typically much smaller than the total number of pixels. This is particularly relevant in the tests of Chapter 3 as the images contain approximately 250,000 pixels. Therefore, the $d * log(d)$ and k terms in the computational complexity equation can be ignored. In addition, the number of scales S is typically a small constant and can be ignored as well. Therefore, the simplified computational complexity can be defined as $O(mn)$. This is the computational complexity that is used in the analysis of Chapter 3.

A.4 Conclusion

This appendix has presented a detailed description of the ORB feature extraction algorithm and has analyzed the computational complexity of the algorithms. As part of this analysis, the ORB algorithm was separated into its two algorithmic components, oFAST and rBRIEF, and these components were analyzed independently of each other. Once the two algorithm

components were analyzed, the computational complexity of the entire ORB algorithm was calculated and simplified using various assumptions regarding the size of the image under analysis. The resulting simplified computational complexity was found to be $O(mn)$.

Vita

Benjamin Walter Martin is a computer engineer specializing in machine learning and data processing. He received his Bachelor of Science and Master of Science degrees from the University of Tennessee, Knoxville in 2009 and 2012, respectively. In 2015, he received his PhD in computer engineering also from the University of Tennessee, Knoxville. During Benjamin's time in graduate school at the University of Tennessee, he was also an intern at Oak Ridge National Laboratory where he received additional training in machine learning and image processing techniques. Aside from machine learning, Benjamin's interests also include software control systems and robotics.