Non-destructive Evaluation Sensor Data Processing and Fusion for Automated Inspection of Civil Infrastructure

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

by

Spencer P. Gibb

Dr. Hung M. La - Thesis Advisor
May 2018
We recommend that the thesis
prepared under our supervision by

SPENCER P GIBB

Entitled

Non-destructive Evaluation Sensor Data Processing and Fusion for Automated Inspection of Civil Infrastructure

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Hung M. La, Ph.D., Advisor

Sushil Louis, Ph.D., Committee Member

Gokhan Pekcan, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2018
Abstract

Data processing is an integral part of civil infrastructure inspection and enables the process to be efficient and easily interpreted. Civil infrastructures include bridges, parking garages, airport runways, and other structures that must remain in proper condition to endure daily use and ensure human safety. This thesis focuses on processing and fusion techniques for non-destructive evaluation sensors used in civil infrastructure inspection. A key aspect of the infrastructure inspection process is determining the condition of the reinforced concrete of the structure. An accurate method for automatic rebar detection in reinforced concrete structures is presented. A method for optimizing a classifier to detect cracks in pictures of concrete surfaces is also presented. In addition, a method to fuse the results from the rebar detection and crack detection methods is presented. The fusion method reduces the total inspection time by only performing the inspection using the necessary sensors. Experimental results are provided to validate the proposed methods.
Dedication

Dedicated to Jessica Turngren.
Acknowledgments

I would like to thank my advisor, Dr. Hung La, who gave me direction and insight into all aspects of my research and provided me with the wisdom I needed to achieve my goals and research.

To my committee members, Dr. Sushil Louis and Dr. Gokhan Pekcan, for the time they have taken to review this thesis and for advice on my research and publications.

Finally, I would like to thank my family, Karl, Sarah, and Reece, for their support both emotionally and financially, which helped me to keep working and going to school.

This work is supported by National Science Foundation under grants: NSF-IIP-I-Corps #1639092, and the National Aeronautics and Space Administration (NASA) under Grant No. NNX10AN23H issued through the Nevada Space Grant, and the Nevada Advanced Autonomous Systems Innovation Center (NAASIC) at the University of Nevada, Reno. This work is also supported by the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center (http://inspire-utc.mst.edu) at Missouri University of Science and Technology. The views, opinions, findings and conclusions reflected in this publication are solely those of the authors and do not represent the official policy or position of the USDOT/OST-R, or any State or other entity.
# Table of Contents

1 Introduction ................................................. 1  
  1.1 Motivation ........................................... 1 
  1.2 Background on Robotic Inspection Systems and Data Processing .... 3 
  1.3 Content ............................................ 4 

2 Ground-penetrating Radar and Data Processing ............ 6  
  2.1 Ground-penetrating Radar: A Non-destructive Evaluation Sensor . 6 
  2.2 Difficulties with GPR Data Processing and Interpretation .......... 7 
  2.3 A Statistical Pattern Recognition Approach to GPR Data Processing 7  
    2.3.1 Adaptive Histogram Equalization for Contrast Stretching .... 7 
  2.4 Histogram of Oriented Gradients as Hyperbola Features .......... 9 
  2.5 Training ........................................... 10  
    2.5.1 Naive Bayes Classification .......................... 11 
    2.5.2 Final Localization of Rebar .......................... 13 
  2.6 Summary ........................................... 16 

3 Visual Data Processing .................................. 17  
  3.1 Visual Data Collected for Civil Infrastructure Inspection .......... 17 
  3.2 Background on Methods for Crack Detection in Images of Concrete Surface .......................... 18
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Training Data Set</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>The parameters of the ((\mu, \lambda)) algorithm used to evolve the structure of CNNs.</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Automated rebar detection results.</td>
<td>40</td>
</tr>
<tr>
<td>5.2</td>
<td>The results from both networks after classifying 10 test images split into 256 pixel by 256 pixel sub-images. The total number of sub-images classified was 4880.</td>
<td>41</td>
</tr>
<tr>
<td>5.3</td>
<td>ER Deployment Time Savings</td>
<td>44</td>
</tr>
</tbody>
</table>
List of Figures

1.1 An image of concrete that has no spalling (top), and an image of spalling concrete (bottom) where corroded subsurface rebar lead to cracking in the concrete and weakness in the bridge. 

2.1 (a)-(c) Positive samples used in the training process that contain clear hyperbolas indicating the presence of rebar; (d)-(f) Negative samples used in the training process that do not contain a hyperbola. Each of these 6 samples is 50 by 15 pixels.

3.1 An example of max-pooling. The input (left) is split into four parts and each part is searched for the maximum local value. The output (right) contains the maximum values found in the search.

3.2 The network structure from [1] used as the baseline for this method.

3.3 Images from the “crack” training data (top), and images from the “no-crack” training data (bottom).

3.4 A block diagram showing the overall approach, from the training data to the output image showing where cracks were detected.

3.5 The performance of the GA across 10 generations with a population size of 30 individuals. This algorithm was run 10 times and the data in this graph represents average values across those runs.
5.1 Rebar localization results (red squares): (a) GPR image from East Helena Bridge, Helena, MT; (b) GPR image from Kendall Pond Road Bridge, Derry, NH; (c) GPR image from Ramp D, Lewiston, ME; (d) GPR image from Pleasant Valley Bridge, Reno, NV.

5.2 A visual example of the performance of the network generated by the genetic algorithm. The original image (top), the performance of the network in [1] (middle), and the performance of the network generated with the method in this work (bottom). False positives have been bounded by yellow rectangles and false negatives have been bounded by red rectangles.

5.3 The GPR contour plot (top) and crack map (middle) of one part of the collected data. The ER deployment map generated by the NDE sensor fusion process with $\omega = 5$ (bottom).

5.4 The GPR contour plot (top) and GPR crack map (middle) of another part of the collected data. A correlation between crack density and low points on the contour map is evident here. The ER map generated by the NDE sensor fusion process with $\omega = 5$ (bottom).
Chapter 1

Introduction

1.1 Motivation

The three infrastructures in the United States that are the most critical to the economic well being of the country are aviation, ports, and roads \[2\]. Consequently, maintaining civil infrastructure is a necessary part of ensuring healthy economic and social growth in modern day society \[3\]. It is estimated that deficiencies in civil infrastructures will cost the United States $3.9 trillion US by the year 2025 \[2\], and that the number of concrete highway bridges in the United States with deteriorating surfaces is over 180,000 \[4\]. Since all civil infrastructure is susceptible to eventual deterioration, and total deficiency if left unmaintained, it is necessary to prevent this deterioration when possible. Unfortunately, by the time that civil infrastructure is visibly damage as in Figure 1.1 it can be too late to fix or too costly to repair the damaged infrastructure because of how long the damage has been present. This further illustrates the need for preventative maintenance of civil infrastructure.

Preventing these expenses and deterioration is possible through building new
Figure 1.1: An image of concrete that has no spalling (top), and an image of spalling concrete (bottom) where corroded subsurface rebar lead to cracking in the concrete and weakness in the bridge.
civil infrastructure to eliminate the cost of repairing the previous infrastructure, and through maintaining current civil infrastructure. It is less costly to properly maintain current civil infrastructure than it is to build new infrastructure, but there is still a need for low-cost inspection and maintenance programs. For this reason, the Federal Highway Administration (FHWA) initiated the Long-Term Bridge Performance (LTBP) program to utilize non-destructive evaluation (NDE) technologies for bridge deck condition assessment [5]. Many civil infrastructure maintenance programs have since been developed [6–8]. In addition to manual civil infrastructure inspection, it has become increasingly desired to develop and deploy robotic systems for inspection [9–11]. Such systems allow for increased accuracy throughout the inspection process, as well as lower the required number of human inspectors who have to be present on-site during the inspection. Overall, robotic inspection systems have the potential to have many positive effects on the inspection process and are increasingly being researched.

1.2 Background on Robotic Inspection Systems and Data Processing

In the past, research efforts have focused on robotic systems for bridge deck inspection, which are typically autonomous and provide rapid data collection capabilities [9, 12, 13]. These robots are equipped with a variety of sensors for inspection, such as ground-penetrating radar (GPR), electrical resistivity (ER) sensors, impact echo sensors, acoustic arrays, global positioning system, inertial measurement units, wheel encoders, cameras for taking images of concrete surfaces, stereo cameras for vision-based navigation, and many other sensors [14]. Some research efforts have focused on
NDE data processing algorithms for the variety of sensors that are equipped on such robotic systems [15–17]. However, it is important for NDE data processing algorithms to meet the strict quasi-real time requirements of civil infrastructure systems. If data collection is performed on-site and NDE data processing is performed later and offline, then an autonomous robotic inspection system is less desirable to the organizations responsible for monitoring structural health [15,18]. As a result, NDE data processing is an integral part of the inspection process for robotic systems to offer true value and these algorithms are capable of performing better than ever as a result of the increasing processing power offered by portable computing solutions. Such solutions need to be able to offer high detection accuracy and be able to formulate the results in a human-readable way for on-site personnel to interpret. The focus of this thesis is providing an overview of our methods that have been developed for NDE data processing on autonomous robotic inspection systems [19,20].

1.3 Content

The remainder of this thesis is organized as follows. Chapter 2 presents the GPR sensor and a brief introduction to its applications. A statistical pattern recognition approach to detecting rebar in GPR scan images is then provided. Chapter 3 introduces the problem of detecting cracks in concrete surfaces using images from a camera on the robotic system. This method utilizes a new technique for neural network structure optimization that produces high accuracy results when detecting cracks. In Chapter 4 a fusion framework is presented that fuses the results from the GPR detection algorithm and crack detection to save time in the inspection process by only deploying time intensive sensors when necessary. Chapter 5 provides results from chapters 2, 3, and 4. Finally, a conclusion and ideas for future work are provided.
in Chapter 6.
Chapter 2

Ground-penetrating Radar and Data Processing

2.1 Ground-penetrating Radar: A Non-destructive Evaluation Sensor

Ground-penetrating radar (GPR) is a type of NDE sensor that has been increasingly used in applications for structural surveys since the 1980s [21]. Uses for GPR include bridge integrity inspection [21, 22], metro tunnel inspection [23], beam and pillar structural inspection, as well as many others. GPR provides a visual representation of metallic objects underneath concrete surfaces, which are indicated by their hyperbolic signatures in the scan image. Prior to recent research in automated object detection using GPR, object location was done manually or using commercial software [24]. Manually locating objects in a scan can be extremely time consuming considering scans can contain hundreds of rebar depending on the size of the
bridge being inspected and commercial software costs thousands of dollars in some cases; when combined with the cost of GPR itself, this can easily prevent users from successfully using the technology.

2.2 Difficulties with GPR Data Processing and Interpretation

Two of the primary concerns with modern methods are detection accuracy and the ability to perform detection in real-time. While research is still continuing in this field, recent methods utilize support vector machines, gradient descent, and various computationally intense methods for detecting rebar [12,13,25,27]. The accuracy of some of these methods has been high, but they are typically tested on ideal cases where scan data is not representative of aged structures, or simulated data, and they often have too high of a run time to be performed on-site [28,30].

2.3 A Statistical Pattern Recognition Approach to GPR Data Processing

2.3.1 Adaptive Histogram Equalization for Contrast Stretching

Due to the typically low-contrast nature of B-scan images obtained via GPR scans, it is necessary to stretch the contrast of these images prior to feature extraction, training, and classification [31]. Contrast stretching helps strengthen the accuracy
of the image classifier and makes it easier for humans to verify the work of the classifier, since it is often difficult or impossible to locate objects in low contrast images. Because standard histogram equalization leads to artifacts in the images, contrast stretching is accomplished in this thesis through Contrast Limited Adaptive Histogram Equalization (CLAHE) [32]. Peaks in a histogram are typically attributed to uniform regions in an image, and through the equalization process the intensity values within the image that correspond to the peaks are spread out across a wider range of intensity values. Although spreading peaks in this way is the primary use for histogram equalization, as it applies contrast stretching and compression, issues can arise when uniform regions of an image contain noise. The noise in uniform regions is then spread across the image as equalization is applied, which can lead to artifacts as the noise is overamplified throughout the image. The CLAHE algorithm presents a solution to this problem that allows adaptive histogram equalization to be performed without propagating noise across the image.

Given an image $I$, which is $M$ by $N$ pixels, histogram equalization is applied across nonoverlapping regions of the image sized $M/8$ by $N/8$ pixels, in the same manner as adaptive histogram equalization. However, the difference between the CLAHE method and standard adaptive histogram equalization is that a clip limit is set, which prevents noise being overamplified in homogeneous regions of the image and causing artifacts. The clip limit is used to determine how much of the histogram’s peak to remove prior to calculating the cumulative distribution function that is used in that region. The part of the histogram peak that is clipped can be redistributed across the bins of the histogram so that it is still present and does not incorrectly skew the cumulative distribution function of the image. When histogram equalization is applied to a GPR image with low contrast it typically leads to areas of the image being too bright or too dark. On the other hand, when CLAHE is applied, the bright
areas and dark areas are less prevalent, which makes it easier to classify regions of the image because the image has more defined features.

2.4 Histogram of Oriented Gradients as Hyperbola Features

Histogram of oriented gradients (HOG) was originally described by Dalal and Triggs in 2005 as a feature descriptor used for object detection [33]. This method uses gradient orientation across an image that is split into uniform cells. HOG features are invariant to geometric transformations and illumination and can be quickly computed. As a result, HOG features have been widely used for object detection in computer vision problems since 2005 [34,35]. The steps used to compute HOG features in this thesis are described in the following paragraphs.

Given a grayscale image $I$, the image can be globally normalized to make the feature selection process more invariant to illumination. Then first order gradients of the normalized image must be computed as

$$\nabla I_N = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$ (2.1)

where $g_x$ and $g_y$ are the normalized image gradients in the $x$ and $y$ direction respectively and $I_N$ is the normalized image. The magnitude of the normalized image is given by

$$magnitude(I_N) = \sqrt{g_x^2 + g_y^2}$$ (2.2)
and the orientation of the normalized image is given by

$$\theta = \text{atan}2(g_y, g_x) \times (180/\pi) + 90.$$ (2.3)

The image is then segmented into cells. In this thesis, it was empirically determined that 5 by 5 pixel cells yielded the best results throughout the training and classification processes. The training and testing image size used in this thesis is 50 by 15 pixels, which means that each training and testing image is made up of 30 cells. For each of the cells, a histogram of gradient orientations is accumulated. Finally, to increase the invariance of this method to illumination, block normalization is applied. Blocks consist of several cells; 3 cell by 3 cell blocks were used in this thesis. The cells are normalized within respective blocks yielding final image descriptors. The features vectors used in the training and classification methods for this thesis contain 240 elements, as each image is 10 by 3 cells and 8 bins were used for the accumulation of histograms of oriented gradients.

### 2.5 Training

In order to train the Naive Bayes classifier used in this thesis, HOG features are extracted from a set of manually selected training images, comprised of two classes: images containing hyperbolas that would indicate the presence of rebar in a GPR scan, and images that do not contain hyperbolas. Each of the selected images is manually assigned a class label for the purpose of training the classifier. The classifier in the next section uses the information from the determined HOG feature vectors for training images as a basis for a priori knowledge about each class. Information on the number of training images and the class they belong to can be seen in Table
Figure 2.1: (a)-(c) Positive samples used in the training process that contain clear hyperbolas indicating the presence of rebar; (d)-(f) Negative samples used in the training process that do not contain a hyperbola. Each of these 6 samples is 50 by 15 pixels.

<table>
<thead>
<tr>
<th>Class</th>
<th>Class Name</th>
<th>Number of Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hyperbolas</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>Not Hyperbolas</td>
<td>300</td>
</tr>
</tbody>
</table>

I. In addition, examples of training images can be seen in Figure 2.1.

2.5.1 Naive Bayes Classification

A Naive Bayes classifier is used in this thesis that classifies new GPR images based on their computed HOG feature vectors, as well as the feature vectors from the training images. More advanced methods are not necessary since the classifier is not solely responsible for the localization of rebar in a GPR scan. This classifier was chosen for its simplicity and speed.

Given a vector of HOG features $x = (x_1, ..., x_n)$, where $n$ is the number of features in the vector, Bayes’ theorem states that the probability of a class given a sample is the product of the a priori probibility for that class and the probability of the sample
given the class. The Bayes model can be written as

\[
p(C_k|x) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{2.4}
\]

where \(Z\) is a constant scaling factor that depends on the contents of the feature vector and \(C_k\) is class \(k\). \(Z\) will change depending on the implementation of the Naive Bayes classifier.

The model from Equation (2.4) can be used as a classifier to assign class labels to test samples as follows

\[
\hat{y} = \arg \max_{k \in \{1, \ldots, K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{2.5}
\]

where \(\hat{y}\) is the assigned class label given a sample, which is chosen based on the maximum probability of a class given the sample. This method, the maximum a posteriori probability (MAP) estimate, classifies solely based on the posteriori probability determined by the Naive Bayes classifier for each class.

Prior to using this Naive Bayes classifier, a search area is determined in the test image. The search area is located based on a primary trait of all GPR scan images; they all contain a significant dark region in the image where the ground surface is. No searching should be done higher in the image than the ground surface. Once the ground surface is located through finding large occurrences of dark horizontal intensity values, the search window is limited again by performing edge detection on the area of the image below the ground plane. The end of the search area is given by taking the average \(y\) pixel location of each edge pixel and searching up to that average location. This works because the edge information shows where the general rebar location is along the \(y\) axis.
Searching and classification is performed on a sliding window starting from just below the ground plane to the average $y$ location of edges determined previously. The distance of this classification search can be changed in cases where bridge decks may contain rebar that is sporadically located along the $y$ axis of the image; however, significantly expanding the search window limits the real-time capabilities of any method including those outlined in this thesis. In addition, to limit the time it takes to apply the sliding window, odd numbered $y$ coordinates are skipped since they do not affect the accuracy of this method and only increase its run time. At each sliding window location within the search area, if the classifier assigns a hyperbola class label to the test image, that point is saved for further processing to more exactly localize hyperbolas within the image.

In the previous sections, an image is first contrast stretched using adaptive histogram equalization. Then the search area within the image is determined using edge detection and the location of the ground plane. Next, training is performed on previously selected images. Then classification is performed on a sliding window, using HOG features, across the search window within the image.

### 2.5.2 Final Localization of Rebar

Given a set of points in the original image, $P_{IN}$, indicating where a hyperbola was detected and classified by the Naive Bayes classifier, it is necessary to more accurately localize the hyperbola. Typical methods for hyperbola localization and fitting include use of the Hough transform or RANSAC \[26, 27\]. However, both of these methods are typically time consuming and therefore not ideal for real-time applications such as that of online automated rebar detection. The methods used in this thesis provide a real-time solution, we refer to as histogram localization, that is of linear time-complexity, since it just moves through a list of pixels in each step that only increases
Algorithm 1: Precise Hyperbola Localization

Input: \( P_{IN} = \{P_1, P_2, \ldots, P_n\} | P_n = (x_n, y_n), s = \) starting search location, \( e = \) ending search location

Output: \( P_{OUT} = \{P_1, P_2, \ldots, P_3\} \)

\[ x_{histogram}[\text{Image width}]; \text{for } x \leftarrow P_{IN}[0][0] \text{ to } P_{IN}[n][0] \text{ do} \]

\[ x_{histogram}[x] += 1 \]

\[ \text{end} \]

\[ \text{for } i \leftarrow 0 \text{ to } x_{histogram.length} \text{ do} \]

\[ \text{if } x_{histogram}[i] > 0 \text{ then} \]

\[ \text{maxima} \leftarrow \text{true} \]

\[ \text{for } j \leftarrow i-7 \text{ to } i+6 \text{ do} \]

\[ \text{if } j > -1 \text{ and } j < x_{histogram.length} \text{ then} \]

\[ \text{if } x_{histogram}[j] > x_{histogram}[i] \text{ then} \]

\[ \text{maxima} \leftarrow \text{false} \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{if } \text{maxima} == \text{true} \text{ then} \]

\[ \text{append maxima to Maxima list} \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{for } i \leftarrow 0 \text{ to } \text{Maxima list length} \text{ do} \]

\[ x \leftarrow -1, y \leftarrow -1 \]

\[ \text{for } j \leftarrow \text{search start} \text{ to } \text{search end} \text{ do} \]

\[ \text{if } \text{Image}[j, \text{Maxima list}[i]] > x \text{ then} \]

\[ x \leftarrow \text{Image}[j, \text{Maxima list}[i]] \]

\[ y \leftarrow j \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{append } x \text{ to } x_{coords}, \text{ append } y \text{ to } y_{coords} \]

\[ \text{end} \]

\[ \text{for } i \leftarrow 0 \text{ to } x_{coords} \text{ length} \text{ do} \]

\[ x \leftarrow x_{coords}[i], y \leftarrow y_{coords}[i] \]

\[ \text{intensity} \leftarrow \text{Image}[x, y] \]

\[ \text{final } x \leftarrow -1, \text{ final } y \leftarrow -1 \]

\[ \text{for } j \leftarrow y-3 \text{ to } y+2 \text{ do} \]

\[ \text{for } k \leftarrow x-3 \text{ to } x+2 \text{ do} \]

\[ \text{if } \text{Image}[j, k] > \text{intensity} \text{ then} \]

\[ \text{intensity} \leftarrow \text{Image}[j, k] \]

\[ \text{final } x \leftarrow k \]

\[ \text{final } y \leftarrow j \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{append (final } x, \text{ final } y) \text{ to } P_{OUT} \]

\[ \text{end} \]
in size as the size of the image and number of pixels increases. This method also allows for accurate rebar localization.

The points in the image are used to accumulate a histogram of their $x$ coordinates. From the accumulated histogram, local maxima are preserved while non-maxima are suppressed in order to yield a set of vertical lines indicating the locations of the most positive matches in a local area. In this thesis, a 13 pixel wide area was used to choose local maxima, which were allowed during this step since they will be eliminated as they converge to the same point in following steps.

After histogram localization is performed, the pixels on each vertical line are compared against each other to find the location of the maximum intensity value. The pixel with the maximum intensity value is significant because the nature of GPR means that objects in the image with high intensity values are typically metallic, and rebar is the object that is being located. Once the highest intensity pixels along each vertical line are located, another search is performed for local maxima within a 5 pixel by 5 pixel region surrounding each of the high intensity pixels.

A final maxima search allows for any location that may be too low on the rebar or that may be off center to be correctly localized, which also accounts for multiple local maxima in the previous step since these maxima typically converge to the same point. For multiple remaining maxima, the maxima closest to the top of the image can be chosen. This should be performed on a small window to ensure that correct positives are not ignored. A summary of the previously described algorithm can be seen in Algorithm 1. Results showing the performance of the method detail in this section can be seen in Chapter 5.
2.6 Summary

In this chapter, a detailed description of a GPR rebar detection algorithm, which utilizes pattern recognition tools, image processing tools, and knowledge of GPR principles to accurately localize rebar within the image. A discussion of the included algorithms is provided as well as a description of the GPR sensor and its functionality and use cases. The work presented here was originally published in [36] and is utilized in our later publications [19,20]. Chapter 3 will review a method for crack detection in images of concrete surfaces.
Chapter 3

Visual Data Processing

3.1 Visual Data Collected for Civil Infrastructure Inspection

Image classification is used in many applications including face recognition, object detection, and scene recognition [37–39]. Typically, the image classification process involves modeling a set of training data so that it is possible to discern between two or more classes of objects or images using a classifier. Then, once the data is modeled, the classifier can be applied to a new set of data in an application setting. One example of a real-world image classification task is crack detection in images of concrete [12,15]. Locating cracks in images of concrete can be useful in applications such as civil infrastructure inspection, where it is desirable to be able to perform inspection of parking garages, bridges, and other structures in a fully autonomous manner [9,19,36]. Using the location of cracks found in images from an on-board camera, it is possible to offer a convenient and cost-effective map of the area.
3.2 Background on Methods for Crack Detection in Images of Concrete Surface

Crack detection in concrete and pavement has been attempted with many image processing, computer vision, and machine learning techniques. Initial attempts to detect cracks in images of concrete were performed using methods such as the Canny edge detector, Sobel filter, Laplace of Gaussian (LoG) \cite{16,17} for edge detection, and other simple methods. These methods do not perform well in cases where the images are noisy, have shadows, or have textured backgrounds. In an attempt to improve crack detection accuracy compared to these classic methods, newer modeling and learning techniques have been applied to crack detection in recent research. Among these techniques are random forest classifiers, neural networks, percolation-based edge detection, frequency domain filtering, and CNNs \cite{1,40,43}. While modern work tends to show high accuracy in crack classification, various models and techniques work better or worse depending on the test environment, camera system, lighting, the presence of other objects in the image, and the size and depth of cracks in the test images \cite{44}.

Some of our previous work has focused on implementing the CNN in \cite{1} on a robotic platform designed at the University of Nevada, Reno campus. This network is the best crack detection method, prior to the method discussed here, that we are currently aware of and is capable of being run in quasi-real time. However, because of the conditions discussed previously and the difficulty of the crack detection problem, this network is still not capable of detecting cracks reliably under difficult circumstances. For these reasons, a more robust solution to the crack detection problem is desired. While deep learning - and more specifically CNNs - provide the best results of any current learning method, constructing an optimal network is an
extremely time consuming task due to the size of the search space, which requires a brief introduction to CNNs in order to understand.

### 3.2.1 Convolutional Network Structure

There are several types of layers that make up the structure of CNNs including: convolution layers, pooling layers, and fully-connected layers. Convolution layers perform multiplication between the input image, or two-dimensional array, and a filter that is also a two-dimensional array. This multiplication happens at every location in the input array and the result is then summed. This process yields another two-dimensional array and can be described mathematically as follows:

\[
f(x, y) \ast g(x, y) = \sum_{i=-\frac{s}{2}}^{\frac{s}{2}} \sum_{j=-\frac{s}{2}}^{\frac{s}{2}} f[i, j] \cdot g[x - i, y - j],
\]

(3.1)

where \(f\) is the input image, or array, \(g\) is the filter, \(s\) is the size of the filter, which is a square filter, and \(x\) and \(y\) are locations in the input arrays. However, Equation (3.1) typically happens many times in each convolution layer but with different filters used for \(g\). Thus, there can be a set of filters used in each convolution layer called \(g_1, g_2, ..., g_f\), where \(f\) is the number of filters in each convolution layer. With a filter size of \(s\) and \(l\) convolution layers in the network, there are at least \(s^2 \cdot f \cdot l\) weights that need to be optimized during the training process, which can quickly create a problem that is not feasible depending on the application requirements.

The second major type of layer in a CNN is a pooling layer. There are several types of pooling layers, but the one used most frequently because of its efficacy is the max-pooling layer, which performs sub-sampling of its input through selecting the maximum value in each region and only passing that value on. An example of this process can be seen in Figure 3.1. Max-pooling is performed to reduce the complexity
Figure 3.1: An example of max-pooling. The input (left) is split into four parts and each part is searched for the maximum local value. The output (right) contains the maximum values found in the search.

of the output throughout the CNN, since it sub-samples and produces a lower dimensional output. However, information is not lost here as maximum values are capable of preserving the important information throughout the convolution process.

The last type of layer that is used often in CNNs is a fully-connected layer. This type of layer works in a similar way to a multilayer perceptron network, which is a classic type of neural network. In fully-connected layers, each input node is connected to each output node directly, and a weight is associated with each connection. These weights are optimized using the back-propagation process [45], which will not be discussed here for the sake of space; however, an output node in the fully-connected layer sums its value as follows:

\[
Output = \omega_1 V_1 + ... + \omega_N V_N,
\]

(3.2)

where there are \(N\) nodes connected to the output node in Equation (3.2), and each node has an associated weight, \(\omega_i\), and value, \(V_i\), where \(i = 1, ..., N\). Fully-connected layers are typically connected to the end of a CNN as a final way to introduce non-linearity into the network. However, large fully-connected layers need to be used with caution since they can quickly introduce many weights to be optimized and greatly increase the amount of training time required for the network. This is a result of the
$N^2$ weights that need to be optimized for each fully-connected layer.

### 3.2.2 Application of a Genetic Algorithm

CNNs require a large number of parameters to be optimized to function at peak performance, which can lead to long training times for networks and often require a deep understanding of the application and data in order to be used successfully. The list of parameters that must be considered in order to construct a CNN include: the number of layers in the network, the activation functions used in the network, the size of layer operations (for convolution and pooling/sub-sampling), the number of layers used in each convolution, the training parameters for the optimizer, and more. This results in an infinite number of possible network structures, and it is often too time consuming to get good results for image classification problems because of the complexity of constructing deep learning networks. To illustrate this point, the network structure used in [1] is shown in Figure 3.2. In Figure 3.2 there are three convolution layers, two pooling/sub-sampling layers, and two activation layers. In this case, the parameters that had to be determined in order to construct this network included convolution layer sizes, pooling layer sizes, activation function types, network depth, optimization function type, and learning rate of the optimization function. The purpose of this method is to use a GA to come up with a highly accurate CNN through optimizing the network parameters, which will help to eliminate some of the
difficulty in constructing an accurate network. Using the network in [1] as a baseline, it will be possible to validate the performance of newly constructed networks against the state of the art method.

### 3.2.3 Related Work Using GAs to Optimize CNNs

There are three types of existing research related to using GAs to improve the results of CNNs: hyper parameter optimization, structure optimization, and loss function optimization. Hyper parameter optimization consists of using GAs to optimize parameters such as the learning rate of the optimization function at the end of a CNN, the number of layers used in convolution, and the type of optimization function used given a set of predefined functions [46–48]. Generally, hyper parameter optimization is simple because the search space is somewhat small and an exhaustive search may be possible given the range of each parameter and the training time for the CNN. Structure optimization, as performed in [49], consists of using a GA to optimize the size of the convolutional layers being used in the network. Unfortunately, this is performed on an extremely simple network, with small images, and on a problem that is already solved on the data set used for testing. While an exhaustive search may not be reasonable in the case of the network in [49], it would be reasonable to place further constraints on the problem to achieve a globally optimal solution without much effort. In addition, the data set used in [49] is a simple face recognition task for modern classification methods. Finally, loss function optimization is performed in [50], where an existing loss function is optimized using a GA. This type of optimization does not directly affect the structure or function of the CNN, but instead optimizes an added classification layer.
3.3 Evolving a Convolutional Neural Network Structure for Crack Detection

In this research, a GA has been applied to a variable depth CNN. This means that the GA evolves the network depth, the layer size, and the hyper parameters of the network. The size of layers in the CNN affects the level of details that are recognized by the CNN, while the hyper parameters control the number of representations of the image throughout the convolution process, which directly affects overall accuracy of the network. This combination guarantees that an exhaustive search is not feasible, which is a case where GAs have excelled previously.

Each time an evaluation of an individual takes place, a network must first be constructed based on the individual’s chromosome. The structure of the network depends largely on the number of convolution layers specified in the chromosome. A max-pooling layer is added to the network after every set of four convolution layers to reduce the complexity of the network. The network always ends with a fully-connected layer with two nodes, which uses a softmax activation function. The final fully-connected layer will output probabilities that the input image belongs to the “crack” class and the “no-crack” class, respectively. Everywhere else, the rectified linear unit (ReLU) activation function is used. Each pooling layer uses a 4 by 4 filter with a stride of 4. Stochastic gradient descent (SGD) is used as the optimization method, with a learning rate of 0.001. SGD is an optimization technique that is used in the back-propagation process to optimize the weights within the network. SGD is described mathematically as:

$$\theta = \theta - \alpha \cdot \nabla \theta \cdot J(\theta; x^{(i)}, y^{(i)}),$$

(3.3)
where $\theta$ is the parameter being optimized, $\alpha$ is the learning rate, $J$ is the objective function, $x^{(i)}$ is sample $i$, and $y^{(i)}$ is label $i$.

Once a model is constructed using a chromosome, the model is trained on a set of training data. The training data set has been labeled manually as either belonging to the “crack class” or to the “no-crack” class. Examples of images from both classes can be seen in Figure 3.3. The training data consists of 3000 images: 1500 crack images, and 1500 no-crack images. For comparison, the network in [1] utilizes 60,000 images in training. Each network is then given 30 epochs of training time to back-propagate and the epoch displaying the highest training accuracy is kept, so that any decrease in network performance or errors will not be accidentally incorporated into the evolution process. The network construction process is outlined in Algorithm 2.

3.3.1 Fitness Function

After a network is constructed based on an individual’s chromosome, it is necessary to evaluate the fitness of that individual for the selection and crossover processes.
Algorithm 2: Network Construction Based on Chromosome

**Input:** \( l \) = number of convolution layers  
\( s \) = convolution filter size  
\( f \) = convolution filter count  

**Output:** \( net \) = network containing the structure described by the chromosome

1. \( conv\_sections = \text{int}(l \div 4) \)
2. \( remainder = f \% l \)
3. \( net = \text{new Network}() \)
4. if \( conv\_sections > 0 \) then
   5. for \( i \leftarrow 0 \) to \( conv\_sections \) do
      6. for \( j \leftarrow 0 \) to 4 do
         7. \( \text{net.add_layer(type = “convolution”, size = s, filters = f)} \)
      8. end
      9. \( \text{net.add_layer(type = “pooling”)} \)
   10. end
   11. if \( remainder > 0 \) then
      12. for \( i \leftarrow 0 \) to \( remainder \) do
         13. \( \text{net.add_layer(type = “convolution”, size = s, filters = f)} \)
      14. end
      15. \( \text{net.add_layer(type = “pooling”)} \)
   16. end
   17. else
      18. for \( i \leftarrow 0 \) to \( remainder \) do
         19. \( \text{net.add_layer(type = “convolution”, size = s, filters = f)} \)
      20. end
      21. \( \text{net.add_layer(type = “pooling”)} \)
   22. end
23. \( \text{net.add_layer(type = “fully-connected”, size = 2)} \)
The fitness function used in this method is a simple accuracy metric. Each individual, and its corresponding network, is evaluated based on the accuracy that it has when classifying a set of test images. The test set contains 600 images: 300 images of cracks, and 300 images with no cracks. The fitness of a network is a decimal value between 0 and 1, and can be described mathematically as:

\[ f = \frac{c}{t}, \]  

(3.4)

where \( c \) is correct classifications, \( t \) is total classifications, and \( f \) is fitness.

Equation (3.4) was also applied to the base network found in [1], which returns a value of 0.8017, indicating that the base network correctly classified \( \approx 80\% \) of the test data correctly after being trained on the 3000 training images.

### 3.3.2 Genetic Algorithm

The method discussed here utilizes the \((\mu, \lambda)\) algorithm, which is detailed in Algorithm 3. Given a population of size \( \mu \), this evolutionary strategy utilizes either crossover, mutation, or a random choice to generate \( \lambda \) children. Of the total available \( \mu + \lambda \) individuals, \( \mu \) are selected to carry on to the next generation. Selection is performed using fitness proportional selection, meaning that individuals with a higher fitness are more likely to carry on to the next generation than those with lower fitness values. This algorithm was chosen because it has been shown that the canonical GA is not a good function optimizer [51]. In addition, this algorithm offered the ability to have a wider choice of individuals when generating a new generation, since it is entirely possible to keep the parent population if the offspring prove to perform poorly. The parameters for this algorithm can be found in Table 3.1. A brief review of the crossover, mutation, and random choice functionality has also been provided.
Algorithm 3: Evolutionary Strategy ($\mu$, $\lambda$)

**Input:** $p =$ population of individuals  
$\mu =$ individuals to select for next generation  
$\lambda =$ children to produce prior to selection  
$g =$ number of generations  
$cp =$ crossover probability  
$mp =$ mutation probability  

**Output:** $p =$ the input population, evolved in place

1 $p\_size =$ length($p$)
2 for $i \leftarrow 0$ to $p\_size$ do
3     evaluate($p[i]$)
4 end
5 offspring = new Population[$\lambda$]
6 for $i \leftarrow 0$ to $g$ do
7     for $j \leftarrow 0$ to $\lambda$ do
8         $r =$ rand([0, 1])
9         if $r < cp$ then
10            /* Generate an offspring using crossover. */
11            offspring[$j$] = crossover($p$)
12         end
13         else if $r < cp + mp$ then
14            /* Generate an offspring using mutation. */
15            offspring[$j$] = mutation($p$)
16         end
17         else
18            /* Choose an offspring from parents randomly. */
19            offspring[$j$] = choice($p$)
20         end
21     end
22 for $i \leftarrow 0$ to $\lambda$ do
23     evaluate(offspring[$i$])
24 end
25 /* Select $\mu$ individuals for the next generation. */
26 $p =$ selection($p$, offspring)
27 end
<table>
<thead>
<tr>
<th>Population size</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations</td>
<td>10</td>
</tr>
<tr>
<td>Selection</td>
<td>Fitness proportional</td>
</tr>
<tr>
<td>Crossover</td>
<td>One point</td>
</tr>
<tr>
<td>Probability of Crossover</td>
<td>0.67</td>
</tr>
<tr>
<td>Mutation</td>
<td>Bit flip</td>
</tr>
<tr>
<td>Probability of mutation</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 3.1: The parameters of the \((\mu, \lambda)\) algorithm used to evolve the structure of CNNs.

### 3.3.3 Crossover, Mutation, and Random Choice

For the \((\mu, \lambda)\) algorithm, one point crossover, bit flipping mutation, and random choice picking are all used to generate offspring. One point crossover was used in this method. In one point crossover, children are generated through picking a single point in the bit strings of the parent. Then the first child is formed from taking everything left of that point from the first parent and everything right of that point from the second parent. The second child is generated by doing the reverse. Mutation is performed by bit flipping. An offspring is generated through mutation in this case by selecting one parent from the parent population at random. Then, a random number is generated for each bit in the bit string. If the random number, \(r\), is below the probability that each bit has to be flipped, then the bit is flipped. Otherwise, the bit remains the same. Finally, if crossover or mutation are not performed, then an offspring must still be generated. In this case, the offspring is generated by randomly selecting a parent from the parent population. An overall diagram showing the overall methodology of this method is shown in Figure 3.4. The method and related information is available in [52]. More information on the crossover operation, mutation operation, and overall genetic algorithm can be found in [53].
Figure 3.4: A block diagram showing the overall approach, from the training data to the output image showing where cracks were detected.
3.3.4 Software and Implementation

Several free software packages were used in the implementation of this method. For the GA implementation, a Python package called DEAP was utilized \[54\]. Each network was constructed using the bit strings generated by DEAP and a package called Keras, which is a Python package for intuitive neural network creation \[55\]. Because of the time required to train each network, this project required several GPUs to be used. The evaluation of offspring was distributed across multiple machines using network communication over secure shell (SSH). This allowed for a static allocation of evaluation tasks between five machines running Nvidia 1080 GTX GPUs and greatly decreased the overall runtime of the method.

3.4 Experimental Results

The GA was run with a population size of 30 for 10 generations. The plot for this can be seen in Figure \[3.5\]. This graph shows the maximum, average, and minimum fitness of each generation across all 10 generations. Maximum fitness is the fitness of the best network in the population, the average fitness is the mean fitness of the population, and the minimum fitness is the fitness of the lowest performing network in the population. It can be seen from the graph that the performance of this method surpasses the performance of the network in \[1\] by the third generation. While the population converges by generation 7, there is still some diversity between the networks, in that they have varying numbers of convolution layers, varying numbers of pooling layers, different filter sizes used in the convolution layers, and varying numbers of filters used in each convolution layer. By the generation 9, the maximum fitness increases to 89.17%, which is 9% higher than the network in \[1\], which achieved a fitness of 80.17% after being trained on the same training data. Results from the method detailed here
Figure 3.5: The performance of the GA across 10 generations with a population size of 30 individuals. This algorithm was run 10 times and the data in this graph represents average values across those runs.
3.5 Summary

In this chapter, a method for optimizing CNN structures using a GA was detailed. This method was applied to the problem of crack detection in concrete surface images. The implementation details and associated algorithms are provided. Initial results are also discussed. This work will appear in two accepted publications in 2018 [52,56]. In Chapter 4, a sensor fusion framework will be discussed that utilizes results from both the GPR rebar detection and crack detection algorithms.
Chapter 4

Non-destructive Evaluation Sensor Fusion

4.1 Efficiency of Civil Infrastructure Inspection

In this work, an NDE sensor fusion framework is detailed, which is used to fuse the results of the rebar detection and crack detection processes. After participating in the National Science Foundation (NSF) Innovation Corps (I-Corps™) in 2015, we found that inspectors and infrastructure owners mainly wish to minimize any interruption to normal operation of the infrastructure during inspection. For this reason, we develop an inspection framework for our autonomous robotic system \cite{19,20} that fuses data collected from multiple sensors that are equipped on the system. Our inspection framework is capable of two inspection modes: fast inspection and in-depth inspection. In the fast inspection mode, the developed robotic system intelligently decides which areas need more in-depth inspection and saves time by not inspecting all areas in an in-depth manner. On the other hand, the in-depth inspection mode
allows all NDE sensors to be deployed across the entirety of the inspection area. Several NDE sensors are equipped on the robotic system used for testing, including GPR, two ER sensors, and a camera system. The GPR system allows us to use our work in [36] to locate subsurface rebar in the reinforced concrete, the ER sensors allow the corrosion of the concrete to be measured, and the camera system is used to build a crack map from the visual data that is collected. Through implementing our various NDE data processing algorithms on the robotic system’s on-board computers, we are able to perform civil infrastructure inspection in quasi-real time, where we can process data just after it is collected in small intervals.

4.2 An NDE Sensor Fusion Framework for Increased Inspection Efficiency

One of the primary contributions of this method is that it fuses sensor data to minimize inspection time. Of the sensors on the robotic platform, three are used for inspection: the camera, GPR and the ER sensor. Of these three sensors, ER takes the most time to deploy. The robot first needs to stop and spray water on the concrete (inspected spot) to create a conductive environment, and then deploy the ER sensors to collect data. This process means it may not be viable to deploy the ER sensor everywhere in inspection area. Through deploying the camera and GPR unit, it is possible to eliminate a majority of the inspection time that would result in using the ER sensor everywhere in the inspection area.

The crack map generated from the camera data, and the condition map generated from the GPR data can be fused into an ER deployment map so that the ER sensor is only deployed in areas where it is needed to ascertain the condition of the inspection
area, and not everywhere. Sensor fusion is a technique for improving the results of data from an individual sensor through combining it with data from another sensor. This technique is used for many types of sensors and data [57–60]. Although GPR data for condition mapping has not been fused with camera data for crack detection, research on fusing GPR data with other sensors does exist [61–63], and research on fusion of other NDE sensors also exists [64–67].

In our method, sensor fusion is accomplished through use of a confidence matrix. This matrix, $M_d$, can be seen in Equation (4.1).

$$M_d = \begin{bmatrix}
  d & 0 \\
  d & 0/1 \\
  0 & 0
\end{bmatrix} \quad (4.1)$$

Equation (4.1) shows the decision matrix where the first row is the case where the condition map is green in the given area, the second row is the case where the condition map is yellow in the given area, and the third row is the case where the decision map is red in the given area. The first column of the decision matrix represents the case where there are cracks present in the corresponding area of the crack map, and the second column represents the case where there are no cracks present in the corresponding area of the crack map.

The explanation for the matrix in Equation (4.1) is that GPR serves as the primary source of information about the area that is being inspected. When the GPR data indicates the inspection area is in good condition (green on the condition map), then crack information is not necessary, as it does not provide any additional certainty that the inspection area is in good or poor condition. Element (1, 2) in the decision matrix represents this case. In addition, if the GPR shows that the inspection area is in poor condition, crack information is not required since the GPR provides high
certainty that the inspection area needs repair and in what precise location. Elements (3, 1) and (3, 2) in the decision matrix represent these cases. The final cases are where the GPR data does not show conclusively what the condition of the inspection area is, which requires fusion with crack data to determine if the ER sensor should be deployed to make a final decision on the condition of the area in question. Elements (1, 1) and (2, 1) in the decision matrix represent these cases. In the case of element (2, 2), since there are no cracks present in that area, it is not possible to use the crack information in conjunction with the GPR data, so the deploying the ER can be left up to the user. In the case of element (2, 1) in the decision matrix, it is necessary to analyze the presence of cracks in the given area with respect to the surrounding area, as in Equations (4.2), (4.3), (4.4), (4.5). This decision matrix is applied across the inspection area by splitting the area into equally sized cells and making a decision on a cell by cell basis as to whether the ER sensor should be deployed.

\[ d = \text{round}(\omega \ast \frac{\sigma_C - \mu_{IA}}{m_{IA} - \mu_{IA}}) \]  \hspace{1cm} (4.2)

In Equation (4.2), \( \omega \) represents an adjustable weight that is a value greater than 1, which can be increased to place more importance on the cracks in the crack map.

\[ \sigma_C = \sum_{i,j=1}^{N} I_C(i,j) \]  \hspace{1cm} (4.3)

In Equation (4.3), \( I_C(i,j) \) is the intensity value at pixel \((i,j)\) in the crack map, where \(i\) and \(j\) both go from 1 to \(N\). These values are summed across the entire cell, which gives the total amount of cracks in that cell, \(C\).

\[ \mu_{IA} = \frac{1}{M} \sum_{1}^{M} \sum_{i,j=1}^{N} I_C(i,j) \]  \hspace{1cm} (4.4)

In Equation (4.4), \( \mu_{IA} \) represents the average intensity value of each cell, \(C\),
across the entirety of the inspection area, $IA$, where $M$ is the number of cells in the inspection area.

$$m_{IA} = \max(\sigma_1, \sigma_2, ..., \sigma_C)$$  \hspace{1cm} (4.5)

In Equation (4.5), $m_{IA}$ is the maximum amount of cracks present in a single cell across the entire inspection area. Using Equation (4.2) allows each cell to be represented in terms of its crack density (the amount of cracks present in the respective cell) with respect to the rest of the cells in the inspection area. The value of $d$ is rounded to yield a binary decision on whether the ER sensor should be deployed in that cell. The results of this fusion method can be seen in the visual examples provided in Chapter 5.

4.3 Summary

This chapter details a sensor fusion method for taking the results from the two previously discussed algorithms and fusing their data to save time during the inspection process. This method utilizes the fact that certain sensors on a robotic system take significantly more time to deploy, and only need to be deployed in areas where there is a disagreement between the other sensors on the condition of a certain section of the inspection area. A review of the need for such frameworks is also provided. This work was originally published in [20] and will be published in an extended journal version of the same paper in [56]. Chapter 5 provides a set of test results showcasing the performance of the previously discussed methods.
Chapter 5

Field Testing and Validation

5.1 GPR Rebar Detection Validation

The method for GPR rebar detection discussed in this work has been validated on scans from three bridges that were collected prior to the research done for this thesis and provided by Geophysical Survey Systems, Inc., as well as scans from a bridge that were collected during the course of this research. All processing was done on a 5-year-old system running an i5 2500k processor. Rebar detection results can be seen in Table 5.1. Included in the results is the average run time for images of each bridge, accuracy, precision, and the number of rebar that were present in the GPR images of each bridge, as well as the location of each bridge. Examples of rebar localization (red squares) can be seen in Figure 5.1.

The results show that this method performs well on the data from the four bridges used in this thesis. The accuracy of the method decreases slightly in cases where adaptive histogram equalization did not affect how clear the hyperbolas are. This is the case with the photos of the Kendall Pond Road Bridge in New Hampshire, which had
Figure 5.1: Rebar localization results (red squares): (a) GPR image from East Helena Bridge, Helena, MT; (b) GPR image from Kendall Pond Road Bridge, Derry, NH; (c) GPR image from Ramp D, Lewiston, ME; (d) GPR image from Pleasant Valley Bridge, Reno, NV.
<table>
<thead>
<tr>
<th>Bridge name</th>
<th>Location</th>
<th>Number of GPR images</th>
<th>Total rebar in images</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Run time per image</th>
</tr>
</thead>
<tbody>
<tr>
<td>East Helena Bridge</td>
<td>Helena, MT</td>
<td>14</td>
<td>1055</td>
<td>99.15%</td>
<td>98.22%</td>
<td>32.40s</td>
</tr>
<tr>
<td>Kendall Pond Road Bridge</td>
<td>Derry, NH</td>
<td>12</td>
<td>2284</td>
<td>91.46%</td>
<td>97.79%</td>
<td>32.91s</td>
</tr>
<tr>
<td>Ramp D</td>
<td>Lewiston, ME</td>
<td>14</td>
<td>3699</td>
<td>92.89%</td>
<td>93.78%</td>
<td>55.46s</td>
</tr>
<tr>
<td>Pleasant Valley Bridge</td>
<td>Reno, NV</td>
<td>20</td>
<td>13206</td>
<td>96.67%</td>
<td>99.59%</td>
<td>118.32s</td>
</tr>
</tbody>
</table>

Table 5.1: Automated rebar detection results.

feint hyperbola. In addition, the classifier performs better in cases where hyperbola are clearly visually separable, as opposed to in cases where they are cluttered and appear to overlap. In cases of overlap, some rebar will not be detected. Overall, the precision of this method remains high, indicating that rebar are not found in areas where none exist.

Finally, with respect to run time of the method, it performs well in all cases. Increased run time is proportional to the length of the bridge. A longer bridge usually contains more rebar and therefore requires more processing. However, all of the run times listed in this thesis are short enough that they can be run as part of a fully autonomous robotic system, in real-time [16][17].
<table>
<thead>
<tr>
<th>Network from [1]</th>
<th>Correct Classifications</th>
<th>Incorrect Classifications</th>
<th>Accuracy</th>
<th>False Positives</th>
<th>False Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>4335</td>
<td>545</td>
<td>0.888</td>
<td>481</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Best Network from GA</td>
<td>4586</td>
<td>294</td>
<td>0.94</td>
<td>280</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 5.2: The results from both networks after classifying 10 test images split into 256 pixel by 256 pixel sub-images. The total number of sub-images classified was 4880.

5.2 Crack Detection Using an Evolved CNN Structure

To provide a visual example of the performance difference between the method detailed in this work and the network in [1], the best network from the GA was used on an image containing cracks. The network in [1] was tested on the same image. The networks classified sub-images within the input image, since the training process was performed using 256 by 256 pixel images. The input image was split into 256 by 256 pixel sub-images and each sub-image was provided to the networks. If a network classified the input sub-image as a “crack” image, then the output image contains that sub-image, but if the network classified the input sub-image as a “no-crack” image, then the output image has had that section removed. This means there were no cracks detected where the output images are black. The original image and output images can be seen in Figure 5.2. The original image was chosen because it showcases several aspects that make crack classification difficult, including: dark spots in the background that look similar to oil and can be easily mistaken for cracks, yellow material that the networks have not been exposed to, and difficult lighting conditions.

It can be seen in Figure 5.2 that the best network from the GA performs bet-
ter than the network from [1] in two major ways: lack of false positives and crack continuity. The middle image in Figure 5.2 contains several positives in the bottom right corner of the image where cracks were detected, but there are none actually present in the original image. On the other hand, there are no such false positives in the bottom right corner of the bottom image in Figure 5.2. Cracks are typically not straight, which can be seen in Figure 5.2. Because of this, certain sections of cracks may be straight and then suddenly diverge from their path. The best network from the GA does an adequate job of detecting different regions of the same crack that may not appear continuous, but are in fact, whereas the network from [1] classifies these regions as “no-crack” regions and loses the crack continuity.

In addition to the visual example provided in Figure 5.2, a larger comparison of results is provided in Table 5.2. Ten images of concrete surface were split into 256 pixel by 256 pixel sub-images, which resulted in a total of 4880 images. These images were classified by each network as being either “crack” images or “no-crack” images. False positives are images that do not contain any cracks, but were classified as “crack” images by the classifier. False negatives are images that do contain cracks, but were classified as “no-crack” images by the classifier. In this test the best network from the GA performs approximately 5% better than the previous best network. It is also important to note that the best network from the GA has 46% less false positives than the previous best network and 76% less false negatives.

5.2.1 Insights

There are several valuable insights, both intuitive and not, that can be gained from the evolution of CNN structures by a GA. First, deeper networks are not always better. While adding more layers can introduce more non-linearity to the features being generated, if too many pooling layers are introduced, it becomes more difficult
for the features to be separable because they consist of less elements. For this reason, networks with 7 to 11 convolution layers performed better than networks with 16 convolution layers, which was the maximum allowed by the implementation. Second, more filters is almost always better than less filters because a higher number of filters produce a deeper representation of features in images. This is intuitive, and reinforced by the fact that all of the top networks produced by the GA used more than 20 filters in each convolution layer. Third, odd filter sizes tend to perform better than even filter sizes. This is a result of the convolution operation, which needs to have the filter centered at a pixel in the input image. All of the best performing networks used odd filter sizes. Finally, it is important to note that of the \( \approx 16,000 \) networks possibilities based on the implementation, the method described here only needed to construct 300 networks to find a more accurate structure.

### 5.3 NDE Fusion Field Test

The NDE sensor fusion method discussed here was deployed in two separate concrete areas, as can be seen in Figure 5.3 and Figure 5.4. A table showing the percentage of ER deployment time can be seen in Table 5.3. The table shows that the proposed NDE sensor fusion method saves a large percent of the inspection time that would otherwise be spent on deploying the ER sensor every foot within the inspection area. As \( \omega \) increases, the number of areas where the ER is deployed also increases, meaning time saved decreases. Despite the increase in \( \omega \), a large portion of time is saved throughout the inspection process when the proposed sensor fusion method is employed. If the sensor fusion method were not employed, with a cell size of one square foot, the ER sensor would need to be deployed at each cell in the inspection area. There are a total of 258 cells in the first inspection area, and 390 in the second inspection area.
Table 5.3: ER Deployment Time Savings

<table>
<thead>
<tr>
<th>Importance of cracks in sensor fusion ($\omega$)</th>
<th>Percent of ER Deployment Time Saved (Inspection Area 1 as in Figure 5.3-Bottom)</th>
<th>Percent of ER Deployment Time Saved (Inspection Area 2 as in Figure 5.4-Bottom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>98.837</td>
<td>98.931</td>
</tr>
<tr>
<td>1.25</td>
<td>97.674</td>
<td>98.504</td>
</tr>
<tr>
<td>1.5</td>
<td>96.511</td>
<td>98.290</td>
</tr>
<tr>
<td>1.75</td>
<td>96.511</td>
<td>98.076</td>
</tr>
<tr>
<td>2</td>
<td>94.961</td>
<td>98.076</td>
</tr>
<tr>
<td>2.5</td>
<td>94.186</td>
<td>97.008</td>
</tr>
<tr>
<td>3</td>
<td>93.798</td>
<td>95.512</td>
</tr>
<tr>
<td>3.5</td>
<td>93.41</td>
<td>95.299</td>
</tr>
<tr>
<td>4</td>
<td>92.635</td>
<td>95.085</td>
</tr>
<tr>
<td>5</td>
<td>92.248</td>
<td>94.444</td>
</tr>
<tr>
<td>10</td>
<td>89.922</td>
<td>92.094</td>
</tr>
<tr>
<td>50</td>
<td>85.271</td>
<td>88.247</td>
</tr>
</tbody>
</table>
Considering the time required to move the robot to the cell, spray water, move the robot again to align the ER sensor with the damp area, and finally deploy the ER sensor, several seconds are required for each cell to be inspected. An ER deployment time of several seconds can lead to an overall inspection time of more than an hour if the inspection area is large, which is not feasible depending on the cost of traffic control to keep the inspection area closed down for inspection.

5.4 Summary

In this chapter, results from the GPR rebar detection algorithm, crack detection algorithm, and sensor fusion framework are provided. The provided results are both in tabular and visual formats for easy interpretation. Comparisons with other methods are provided where applicable and possible. Chapter 6 provides a conclusion to this thesis and ideas for future work based on these contributions.
Figure 5.2: A visual example of the performance of the network generated by the genetic algorithm. The original image (top), the performance of the network in [1] (middle), and the performance of the network generated with the method in this work (bottom). False positives have been bounded by yellow rectangles and false negatives have been bounded by red rectangles.
Figure 5.3: The GPR contour plot (top) and crack map (middle) of one part of the collected data. The ER deployment map generated by the NDE sensor fusion process with $\omega = 5$ (bottom).
Figure 5.4: The GPR contour plot (top) and GPR crack map (middle) of another part of the collected data. A correlation between crack density and low points on the contour map is evident here. The ER map generated by the NDE sensor fusion process with $\omega = 5$ (bottom).
Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presented the reasoning behind the need for NDE data processing in civil infrastructure inspection, as well as two methods for NDE data processing for NDE sensors and one method for NDE sensor fusion.

The first NDE data processing method presented is one for GPR data, which works to localize sub-surface rebar in each of the GPR scans. This method utilizes classic pattern recognition and image processing algorithms to pick the location of each rebar. The method is tested on several GPR scans from reinforced concrete bridge decks both locally and nationally. The test results indicate that the GPR rebar detection algorithm is applicable to real world test scenarios and maintains high detection accuracy across varying conditions.

The second NDE data processing method is a CNN structure optimization technique that utilizes a GA to tune the parameters of the CNN structure, with the goal of finding an accurate CNN for classifying images of concrete surface containing cracks.
The chapter containing this method provides the reader with an introduction to the core aspects of what makes CNNs perform well at classification tasks. The method is outlined in algorithms that show both the function of the GA and how CNN structures are constructed based on the bit strings utilized by the GA. The results of this method are presented and show a significant performance increase when compared to the previous best method.

The NDE sensor fusion method presented here utilizes both GPR data and the output for the CNN used for crack detection, and uses them to reduce inspection time. This reduction is accomplished through only deploying more time-consuming sensors in areas where there is no consensus between the GPR and crack data. A weight variable is introduced to vary the amount of importance that is placed on the crack data collected during the inspection, which leads to a further reduction in inspection time. Two test scenarios are provided to validate this method.

6.2 Future Work

With the increase in computing power and the constant development of new classification techniques, there will always be ways to improve upon NDE data processing techniques like those presented in this thesis. One possible direction of future work is to extend the CNN optimization method for crack detection to additional types of sensor data. Such a process could be applied to GPR scan images, thermal camera data, other types of concrete defects aside from cracks, and any type of sensor data that can be represented by image data. Another possible direction of future work is to implement new ways of driving inspection decisions and processes through NDE sensor fusion. Applications for this include addition sensor fusion frameworks to make the inspection process smarter, and NDE sensor fusion driven path planning which
would allow a robotic system to decide where to inspect next based on the fusion of sensor data it has collected so far.
Bibliography


