Dead End Body Component Inspections With Convolutional Neural Networks Using UAS Imagery

Ian Edward Nordeng

DEAD END BODY COMPONENT INSPECTIONS WITH CONVOLUTIONAL NEURAL NETWORKS USING UAS IMAGERY

by

Ian Edward Nordeng
Bachelor of Science, University of Wisconsin Madison, 2011

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

August
2018

This thesis, submitted by Ian Nordeng in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

[illegible]
Jeremiah Neubert

[illegible]
William Semke

[illegible]
Ronald Marsh

Douglas R. Olsen
Doug Olsen

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies of the University of North Dakota and is hereby approved.

[illegible]
Grant McGimpsey,
Dean of the School of Graduate Studies

July 27, 2018
Date

PERMISSION

Title	Dead End Body Component Inspections with Convolutional Neural Networks Using UAS Imagery
Department	Mechanical Engineering
Degree	Master of Science

Ian E. Nordeng
May, 1, 2018

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor Professor Jeremiah Neubert of the School of Engineering and Mines Mechanical Engineering Department at the University of North Dakota. He provided invaluable support and guidance throughout my research while giving me the freedom to pursue my research in my own direction. Without his guidance and support I would never have had the opportunity to pursue a career in machine learning or computer vision.

I would like to extend my gratitude to Doug Olsen, the initial principle investigator for this research, as well as the Research ND Grant program. Doug Olsen’s support and direction alongside funding from Research ND was instrumental in this work.

I would also like to thank the remaining members of my committee: Professor William Semke and Professor Ronald Marsh. Without their support and suggestions, this thesis would not be what it is today.

Alongside the above committee members, I would like to extend my gratitude to the Department of Aviation under the School of Aerospace. Their knowledge and connections with UAS technologies created an important partnership.

In addition, I would like to thank our private industry partners Field of View and SkyScopes. The images Field of View supplied during their work in the sensor trade study was heavily used in the training images of this work and without them, this research would have suffered. The inspection images of live power line components supplied by SkyScopes was also incredibly helpful in both the training data, as well as testing the resulting CNNs that were developed.

Finally, I would like to thank all my friends and family for their unfailing support and encouragement throughout the years. Without you I wouldn’t be where I am today. Thank you.

ABSTRACT

This work presents a novel system utilizing previously developed convolutional neural network (CNN) architectures to aid in automating maintenance inspections of the dead-end body component (DEBC) from high-tension power lines. To maximize resolution of inspection images gathered via unmanned aerial systems (UAS), two different CNNs were developed. One to detect and crop the DEBC from an image. The second to classify the likelihood the component in question contains a crack. The DEBC detection CNN utilized a Python implementation of Faster R-CNN fine-tuned for three classes via 270 inspection photos collected during UAS inspection, alongside 111 images from provided simulated imagery. The data was augmented to develop 2,707 training images. The detection was tested with 111 UAS inspections images. The resulting CNN was capable of 97.8% accuracy in detecting and cropping DEBC welds. To train the classification CNN if the DEBC weld region cropped from the DEBC detection CNN was cracked, 1,149 manually cropped images from both the simulated images, as well images collected of components previously replaced both inside and outside a warehouse, were augmented to provide a training set of 4,632 images. The crack detection network was developed using the VGG16 model implemented with the Caffe framework. Training and testing of the crack detection CNNs performance was accomplished using a random 5-fold cross validation strategy resulting in an overall 98.8% accuracy. Testing the combined object detection and crack classification networks on the same 5-fold cross validation test images resulted in an average accuracy of 73.79%.

CHAPTER I

INTRODUCTION

Infrastructure in the United States is a multi-trillion-dollar industry, of which the electrical grid makes up a significant portion [1,2,3]. These costs are often estimated based on installation costs [2]. Installation costs of new transmission lines have been estimated to range from \$150,000 to \$2 million per mile [2,3]. Maintenance of these transmission lines, including inspections, is estimated as three percent of the installation cost [2]. To reduce costs and extend the life of existing transmission lines, some electric generation and transmission cooperatives have begun to implement unmanned aircraft systems (UAS) to lower cost and provide higher resolution inspection imagery [4]. Using these high-resolution images gathered from UAS, this work provides an initial study on the effectiveness of convolutional neural networks (CNNs) and their potential in automated inspections of high tension power line dead-end body components (DEBCs). This work aims to reduce inspection costs of DEBCs by developing software utilizing deep learning CNNs capable of automatically detecting the DEBC from UAS imagery of high tension power lines, and then classifying if the component in question contains a defect in the form of a crack. While the individual CNN architectures used in this work have been developed previously, the approach to implementing them for use in power-line inspections is new.

The DEBC is a full tension device used to attach the conductor to the power line structure [5]. The component becomes energized in a live power-line to transmit power through jumper connections. This component consists of an outer aluminum sleeve with a four-bolt pad welded to one end, a steel forging with a steel eye, and an aluminum insert. The outer aluminum sleeve grips the aluminum strands of the power line, while the inner aluminum inserts grip the inner aluminum matrix core wires separately. The eye of the steel forging is connected to the insulator string on the dead-end tower or substation allowing for physical connection to the tower while insulating the tower from becoming energized. Energized jumper connectors attach to the outer sleeve pad and are used to connect pairs of powerline conductors. To aid inspection and maintenance of the DEBC, the weld portion of the component was detected with a bounding box annotation allowing for segmentation and analysis of possible partial failure due to cracks in the weld [5]. Figure 1 provides a representation of the DEBC with the jumper terminal attached [54].

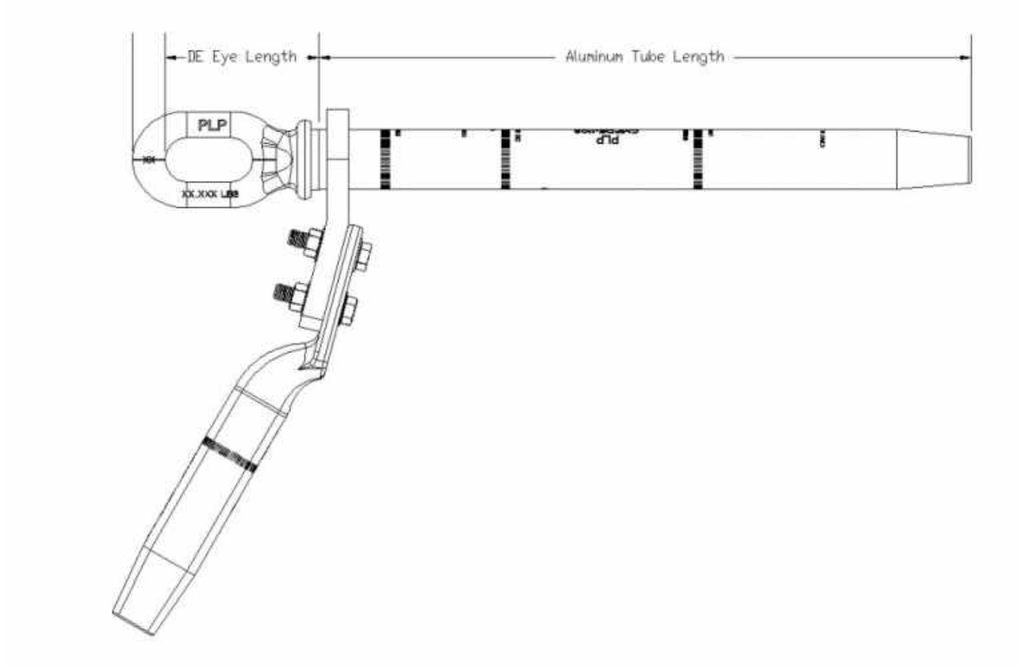


Figure 1: DEBC assembly with jumper attached.

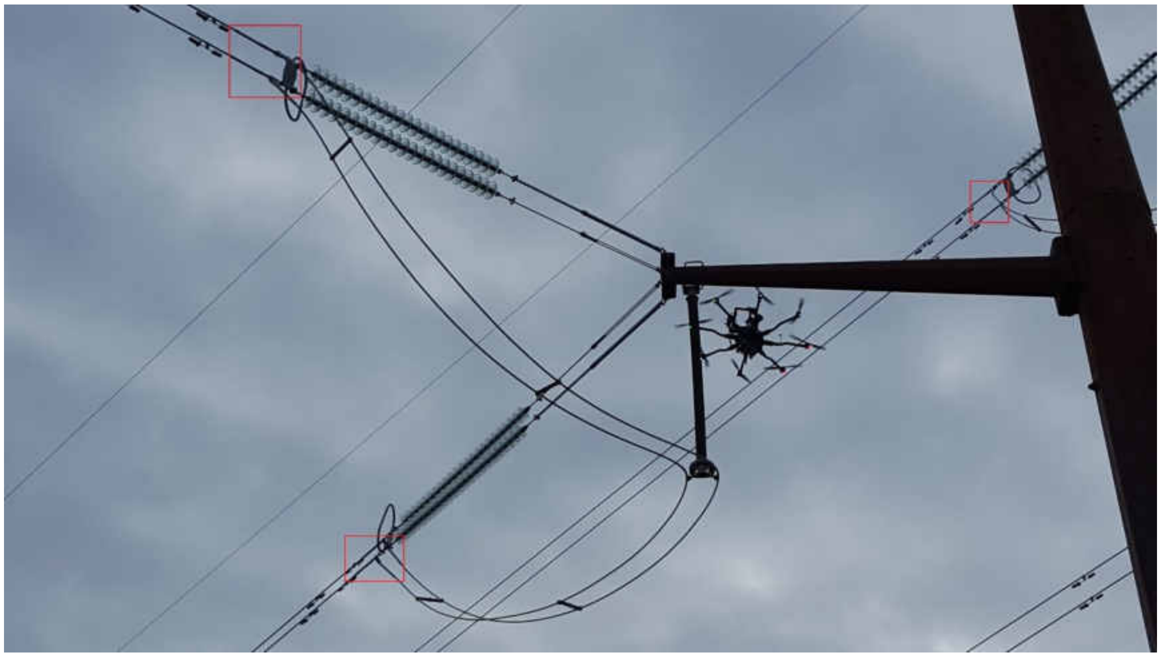


Figure 2: Image of UAS collecting DEBC inspection images. Regions in red contain DEBCs.

Before the implementation of UAS in inspections, maintenance inspections were performed with people either in bucket trucks, or manned vehicles such as helicopters [6, 7]. As can be expected, regularly sending either of these larger crafts to all high-tension powerline structures can be more expensive, more dangerous, and provide lower quality images than could be obtained with a small UAS. It has been estimated that the total cost of power line inspections using UAS is approximately half that of a manned helicopter inspection [55]. Small UAS can fly closer to the components to be inspected and may be outfitted with high quality camera sensors as can be seen in Figure 2 above. Even in the event of a small UAV crash there would be little to no damage to the infrastructure or operator [8, 9]. A disadvantage to the use of UASs is the large amount of data they can generate, and the time needed to analyze the data. For example, a UAS mission that collected 15,000 images took a team of two to four analysts over 400 hours, or 18 days to process [51]. Therefore, methods to combat the big data problem UAS pose [10] must be developed.

This work provides a method of utilizing CNNs to automatically detect the DEBC weld, then evaluating whether the component contains a partial failure due to cracks. This work attempted to achieve an accuracy of greater than 90% in the detection of the DEBC, while maintaining a greater than 80% accuracy in identifying cracked DEBC welds. Combining the two systems should provide an overall accuracy of greater than 72%.

Background

Previous work in automatic inspections of transmission lines has taken many forms. This paper broadly separates these fault detection methods as current fault location methods and visual fault detection. Current fault location methods in power transmission lines involve impacted distribution systems such as broken power lines or other failures in which the distribution of electricity is measurably impacted. Visual fault detection focuses on methods using image-based detections, either human or computer algorithm based.

Current Fault Detection. The types of faults detected in these systems include series, and more generally shunt faults [35, 38]. A series fault is defined as a fault where the conductor is disconnected at one location as would occur when the power system network contains broken lines [35]. Shunt faults occur when a connection between the power line core and sheath occurs, often due to old or damaged insulation. There have been many methods developed to automatically detect these faults, however in [35] they have been subdivided into conventional techniques or those employing artificial intelligence methods. Conventional techniques include travelling wave and impedance-based methods, while artificial intelligence methods include artificial neural networks, support vector machines, fuzzy logic, genetic algorithms, or a matching approach. Each of these techniques will be expanded on below.

The travelling wave method is based on the transmission and reflection of traveling waves found between the fault location and the line terminal [35]. This method requires high speed data acquisition devices to capture the transient waveform allowing for fault location. This method has been used more widely in transmission. The advantage of this method is that it is independent of the network configuration and installed devices [35]. A disadvantage to this method is the need to capture the transient waveform which requires expensive high-speed data acquisition devices consisting of sensors, fault transient detectors, and Global Positioning Systems (GPS) [39].

Impedance based methods use the impedance value from a measurement node to calculate the fault location using voltage and current data [35]. One advantage of the impedance method is that it is simpler and less expensive than the travelling wave method [35]. However, many impedance-based inspections suffer from increased uncertainty due to calculation errors [40].

Methods using various forms of artificial neural networks have been used for locating faults in distribution systems by relating patterns in the voltage phase and angle to network faults [35]. Many different neural network architectures have been proposed for the task of fault detection in distribution systems including but not limited to modular artificial neural networks [41], feed forward artificial neural network [42], and CNNs [43]. The advantage of artificial neural networks is that they are simple to implement; however, they are highly dependent on quality and quantity of the training data. Artificial neural networks are also very time consuming to train as the process is slow to converge. Furthermore, network parameters must be identified on a trial and error basis, and the algorithm must be re-trained whenever the system undergoes any change.

Support vector machines (SVM) have been used in the task of detecting transmission line faults by developing a classification between two classes as class 1 and class 0. The SVM model treats training data as points in space marked by their voltage and phase angle values. The SVM attempts to separate the two different classes by a gap that is maximized to be as wide as possible between the classes. The advantages of SVM use for distribution line fault detection includes the speed for detection and lower requirement for heuristics.

Fuzzy logic has been used for fault detection with the basic idea that the likelihood of a fault existing is based on mathematical models of vagueness using degrees of truth, and probability as a mathematical model of ignorance. Both degrees of truth and probability are represented by a number between zero, completely impossible, and one, entirely possible. The fuzzy logic-based classification takes a measurement of the fundamental current signals to calculate the characteristic features as input for the fuzzy logic system [35], and then classifies based on both the angular differences among sequence components of the fundamental fault current, as well as the relative magnitudes of the fundamental phase current [44]. Fuzzy logic systems have been shown to provide accurate classification results when there is a lack of sufficient statistical information. The disadvantage to fuzzy logic systems includes determining the global minimum using the fuzzy membership functions and that feature definition and extraction must be enhanced for the algorithm to classify properly.

The genetic algorithm method locates faults by treating the faulty section as an optimization problem by mimicking natural selection [35]. This algorithm works by evolving initially random parameters through selecting random individuals from the population, evaluating the individual's fitness through a fitness function, storing the fittest parameters, and randomly mutating the parameters iteratively until classification is acceptable. Advantages of the genetic algorithm for this task are the potential for increased simulation speed and the ability to reduce the dimension of possible solutions. The disadvantage to using the genetic algorithm for fault location for distribution systems is that results are not consistent due to the randomized process the algorithm relies on and may produce inaccurate results [35].

The matching approach makes a comparison between measured and simulated data through use of large databases for fault location identification [35]. Typically, the voltage sag or current data is recorded to identify the location of a fault. An Advantage of the matching method was its economical nature as it considers only measurement node voltage sags data. The disadvantage of the matching approach was the dependency on the simulation data stored in the database to match the data with actual fault data [35].

Visual Based Inspections. Electric power companies typically perform visual inspections of transmission lines for maintenance and inspections. Previously these visual inspections were often performed using helicopters equipped with various camera sensors [47]. Companies have been moving to UAS due to lower costs, some estimating half the cost of a manned helicopter [55], as well as reduced potential dangers to both crew members and infrastructure [8 ,47].

Many methods have been developed that automatically detect potential faults to reduce costs and time needed to review camera sensor images. Our focus is to detect minor faults at the component level to prevent larger faults in the distribution systems later. These detections are performed through automating visual inspections of components and detecting component partial failures before larger faults can occur. Multiple algorithms have been previously developed to detect such faults and will be expanded on below.

In [34], power line insulators are visually recognized, and faults visually detected using a variety of computer vision techniques. The insulators are detected using Difference of Gaussian (DoG) keypoints and grouped using a k-Nearest Neighbor (kNN) classifier, and then using a RANSAC method to fit the classified keypoints to a bounding box for the insulator. Faults were found using a local outlier factor (LOF) which provides a dissimilarity score through use of the distance of a descriptor to the kNN as an estimate of the local descriptor density. While less computationally expensive, kNNs do not generalize well and are not robust to noisy data due to not performing any learning. This lack of generalization is also a disadvantage for future use as expanding the number of classes to classify would be difficult. The LOF method was also rejected for fault classification as the cracks found in the welds were highly variable including minor cracks that are similar to non-cracked components.

The algorithm used in [36] focused on detecting power lines with a cluttered background for use with UAVs. This method developed a pulse connected neural network (PCNN) filter to remove the background clutter allowing for a Hough line transform to detect straight lines, and finally using a K-means clustering approach to discriminate power lines from other linear objects. A Drawback of this method include the apparent requirement for the UAV to fly directly above the power line. This method is also susceptible to detecting false positives from other features that appear as straight lines parallel to the power line.

Photogrammetric methods alongside low cost UAS were used in [45] to provide 3D mapping of power lines to enable power line maintenance regarding line sag. This method uses several aerial images tagged with spatial coordinates provided from an inertial measurement unit (IMU) and global navigation satellite systems (GNSS). The data is processed with bundle adjustment to optimize for accurate pose estimation. A filter was then applied to enhance the power line while reducing background noise. A cubic grid of points in 3D object space was then generated around targeted power lines and each grid point was projected to multiple aerial images. If the number of images was larger than a predetermined threshold, the grid point was classified as a power line point. With all power line points created, the 3D represented power line was generated by interpolating the point cloud using the parabola equation. This method was found to reconstruct the power line to allow power line sag measurements to within a few centimeters.

The method proposed in this paper includes and expands on the algorithm provided in [37]. This algorithm uses the Faster R-CNN algorithm with the VGG16 network architecture as its backbone to identify and locate any potential DEBCs found in an image through use of a bounding box annotation. The algorithm in [37], while achieving high accuracy in detection of the DEBC weld (97.8%), does not perform actual inspection tasks. To add a method to inspect the detected component, the proposed algorithm segments the bounding box annotation as a separate image and uses a CNN to classify the image regarding potential partial failures due to cracks in the DEBC weld. The use of a separate CNN to classify the cracks instead of adding a new class to the Faster R-CNN algorithm containing cracked DEBC welds was to increase the resolution of the classification images which is addressed in Chapter III.

While not directly related to power transmission lines, the method in [48] was developed to use images to detect potential cracks in nuclear power plant components. This method used images from videos taken in a raster scan pattern of the nuclear power plant components, along with the GoogLeNet architecture CNN [49], to fine-tune a CNN to classify the image patches generated as either containing a crack or not. This method achieved a true positive rate of 0.93 and false positive rate of 0.06 in detection of nuclear power plant component cracks. Notably, this method was able to detect many subtle cracks such as those within welds, scratches, and grind marks. While the cracks found in nuclear power plant components do not match and thus cannot accurately detect potential cracks found in the toe-weld cracks this paper focuses on, the method in [48] does provide an example of the successful use of CNNs for crack detection.

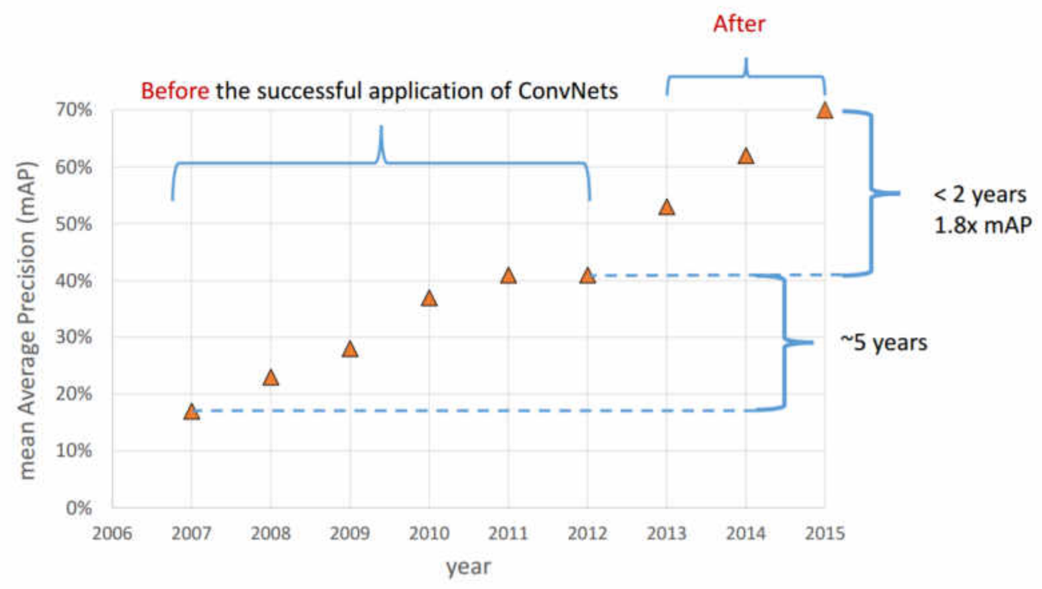


Figure 3: Top performance via mean average precision (mAP) in PASCAL VOC object detection, a yearly general object recognition challenge. As can be seen progress had slowed and leveled off before convolutional neural networks were utilized making year after year progress. Recovered from [46].

CNNs were selected in this study as they been shown as the state of the art method in general object classification as shown by several object classification challenges. Figure 3 provides an example of one object classification challenge, the

PASCAL Visual Object Classes challenge [13], over several years showing the potential CNNs have in recognizing general object classes in images. Due to the success CNNs have shown in such challenges, this work focuses on using them for detecting and classifying the DEBC weld for possible partial failure due to cracks.

CHAPTER II

CONVOLUTIONAL NEURAL NETWORKS

The convolutional neural network architecture using the Faster R-CNN [18] algorithm, along with the VGG16 architecture [11], was chosen for the purposes of aiding inspections in this work. These CNN architectures were used as they were among the state of the art in object detection available, as evidenced by the Pascal VOC 2007 challenge. Faster R-CNN was an object detection network which adds a region proposal network, as explained later, that determines the location of specified objects within an image with a bounding box annotation and classifies the object with another CNN as the backbone of the entire architecture. In this work, the VGG16 network, named so after the Visual Geometry Groups 16-layer CNN who developed it, was used as the backbone for Faster R-CNN. The same VGG16 network was also utilized in training and classifying potential cracks that may exist on the DEBC weld. The pretrained deep VGG16 model was implemented due to its high precision and public availability, as shown in [11].

Convolutional neural networks (CNNs) were used extensively in this work, and in this chapter, we will describe in detail what they are and how they work. Due to the complexity of CNNs, we will describe first what the building block of CNNs, the artificial neuron, is then describe a much simpler artificial neural network (ANN), before expanding to a full CNN.

Artificial Neuron

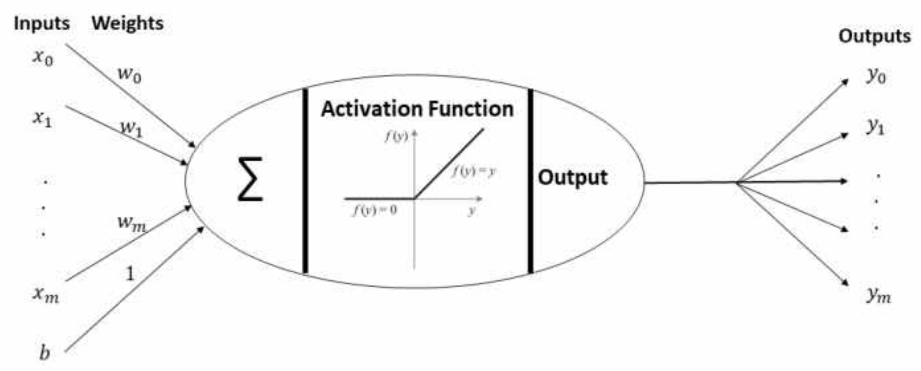


Figure 4: Visualization of an artificial neuron.

The artificial neuron is the building block all ANNs, including CNNs, are created from. An artificial neuron is provided inputs (either the data input to the neural network or the output from a previous neuron) that are adjusted by multiplying the weights of the neuron [50]. Neural networks often contain a bias as represented as *b* in Figure 4 which are summed alongside the weighted inputs. This bias is used to help shift the neurons output to the desired range. After summing the weighted inputs to the artificial neuron, they are provided as input to an activation function. The activation function is used to represent if the neuron in question activates or fires in regard to the input data, similar to biological neurons in mammalian brains. The resulting activations are then provided as the output of the neuron, which can be provided as input to more neurons, or presented as the output of the neural network. Figure 4 provides a visual representation of a general artificial neuron. Training a neural network to learn to perform a new task or adjust any of the ANN characteristics is performed in a process called backpropagation, where the weights and bias are adjusted to provide the desirable output/s given the relevant input. Backpropagation will be described in a later section [50].

Activation functions can take many different forms. However, due to the use of the rectified linear unit (ReLU) in this study, we will focus our efforts on describing the ReLU function. The output for the ReLU activation as y is represented as

$$y = \max\{0, x\}, \tag{1}$$

in which x represents the input [11, 12]. The output is therefore the nonlinear value of the maximal value as either zero, or the value of the input. The ReLU activation function provides several times faster training than previous neural network models utilizing other activation functions [11, 12, 16]. The ReLU activation also has the desirable property that normalization is not necessary to prevent saturation [11, 12, 15, 16].

Simple Artificial Neural Network

With the process of individual artificial neurons explained above, we will describe a simple artificial neural network (ANN) to help understand the more complex CNN described later. An ANN is a series of interconnected artificial neurons arranged in a specific format or architecture, often composed of several layers [12, 15, 16, 50].

There are three general types of ANN layers [12, 15, 16]. The first is the outside inputs provided to the ANN called the input layer. Next are the collection of artificial neurons which are hidden from the outside world and are likewise called the hidden layers. There may be anywhere from zero to many hidden layers depending on the ANN architecture. Last, there is a single output layer which provides the transference of the total computational output of the ANN.

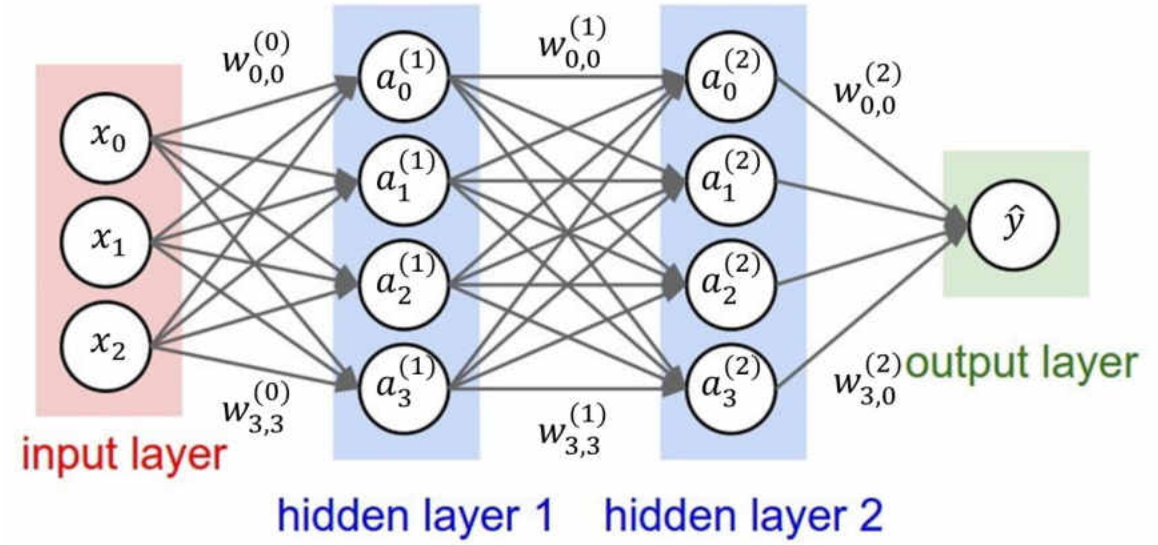


Figure 5: Simple fully connected ANN with one input layer, two hidden layers, and one output layer. The variables denote the input $\boldsymbol{x_i}$ at position $\boldsymbol{i}$, the weights as arrows labeled as $\boldsymbol{w_{i,j}^{(l)}}$ at layer $\boldsymbol{l}$ connecting the $\boldsymbol{j^{th}}$ position, the neuron activation as $\boldsymbol{a_i^{(l)}}$ and the final output as $\boldsymbol{\hat{y}}$ [52].

The simple ANN we describe here is representative of a two-hidden layer fully connected network. A fully connected network is defined as an ANN arranged such that all artificial neurons connect or provide output to all artificial neurons in the next layer [12, 15, 16, 50]. The architecture of this ANN is shown visually in Figure 5. These ANNs are used and manipulated with two different algorithms known as forward propagation and backpropagation. These two algorithms are explored in depth in the following sections.

Forward Propagation. The forward propagation algorithm is a process in which the ANN is provided input and provides a predicted output based on the input provided [12, 15, 16, 50]. With a trained ANN, forward propagation is used to predict desired outputs from the types of data the ANN was trained with. To train a new ANN or fine-tune an existing one to improve or predict new outputs, forward propagation is used alongside backpropagation as explained later.

Forward propagation is accomplished moving from the left to the right in Figure 5 [12, 15, 16, 50]. Starting with the input layer, the ANN first calculates the total input to the activation function for all artificial neurons in each layer as

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}, \tag{2}$$

where $A^{[l]}$ is a matrix containing all activations from all artificial neurons at layer l where the input layer is the zeroth layer. These inputs are computed from a series of matrix computations. The combined weights in each layer as $W^{[l]}$ is an $n^{[l]}$ by $n^{[l-1]}$ sized matrix where n is the number of hidden units or artificial neurons at the specified layer. The matrix $W^{[l]}$ is multiplied by all the activations of the previous layer stored in an $n^{[l-1]}$ by m matrix $A^{[l-1]}$ where m is the number of inputs from the entire dataset. This product can then have the bias vector $b^{[l]}$ of size $n^{[l]}$ added to it. The activation of the artificial neuron is then calculated through the process

$$A^{[l]} = g^{[l]}(Z^{[l]}), \tag{3}$$

where $g^{[l]}$ is the activation function defined for that layer. The input layer uses the input data itself as activations in the A^0 position. The output of the ANN is then the last activation performed by the final output layer [12, 15, 16, 50].

Several activation functions can be used for each layer such as the activation function ReLU provided in equation 1, however for binary classification, the output layer often uses a sigmoid activation function [11, 12, 15, 16]. The sigmoid activation function for a single artificial neuron is represented as

$$\hat{y} = \frac{1}{1 + e^x}, \tag{4}$$

where $\hat{y}$ is the output of the activation, and x the input. This activation function is often used for binary classification due to the fact it exists between zero and one and therefore provides a probability of the output being classified as either of the two binary classes. Once the output is calculated, it can either be provided to a user or other system or utilized in backpropagation to adjust the weights and biases throughout the ANN to either improve or train it for new purposes.

Backpropagation. Backpropagation is the process by which the weights and biases of the artificial neurons in the ANN are adjusted to learn new tasks, improve existing predictions, or fine-tune similarly known tasks [12, 15, 16, 50]. This process of backpropagation for an ANN follows a supervised learning process. Supervised learning consists of known, already classified training data. The ANN takes the input training examples and performs forward propagation. After forward propagation, a comparison of the output of forward propagation with the desired output as determined from the labeled training example is performed, and the weights and biases are adjusted to better match the desired output. A more detailed explanation of this process in the sections below beginning with the loss function.

Cost Function. After calculating the output for a forward pass with the known training data, this output and the known labeled training data is compared. A single training example is compared using a loss function, whereas the entire training set is compared with the cost function. We will begin by describing the loss function.

There are many ways to calculate the loss function, but for a simple binary classification the logistic regression loss function

$$L(\hat{y}, y) = -(y * \log \hat{y} + (1 - y) * \log(1 - \hat{y})), \tag{5}$$

is commonly used [11, 15]. In equation 5, L represents the loss function measuring how well the overall forward propagation output $\hat{y}$ matches the ground truth label y. In binary classification y is set as either one or zero.

The cost function is represented as,

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} L\left(\hat{y}^i, y^i\right), \tag{6}$$

where $J(W, b)$ is the cost J applied to the ANN parameters W and b, and m the number of training samples in the training set with i the individual training samples [11, 15, 53]. This cost function provides the average loss for the entire training set. To adjust or improve the ANN, the parameters W and b are adjusted to minimize J through a process called gradient descent.

Gradient Descent. Through an iterative process, the gradient descent algorithm adjusts the weights and biases of ANNs to converge to or close to a global optimum by minimizing the cost function [11, 12, 15, 16, 50, 53]. The parameters of each of the ANNs layers are adjusted starting from the last layer and working backwards to the first layer by calculating the derivatives of the loss function. These derivatives are computed as

$$\frac{\partial L}{\partial Z}^{[l]} = W^{[l+1]T} \cdot \frac{\partial L}{\partial Z}^{[l+1]} * \frac{\partial L}{\partial A}^{[l]} , \tag{7}$$

$$\frac{\partial L}{\partial W}^{[l]} = \frac{1}{m} \frac{\partial L}{\partial Z}^{[l]} \cdot \frac{\partial L}{\partial A}^{[l-1]T} , \tag{8}$$

and

$$\frac{\partial L}{\partial b}^{[l]} = \frac{1}{m} \sum_{i=1}^{n} \frac{\partial L}{\partial Z}^{[l][i]} , \tag{9}$$

where n is the number of artificial neurons in layer l. The ideal weights and biases will exist at the global minimal cost function J. To achieve these weights and biases, these parameters are adjusted using the previous gradients though the equations

$$W^l = W^l - \alpha \frac{\partial J(W, b)}{\partial W}, \tag{10}$$

and

$$b^l = b^l - \alpha \frac{\partial J(W,b)}{\partial b}, \tag{11}$$

where α represents the learning rate, a pre-defined hyperparameter. This process is performed iteratively until the cost function converges to its minimum, or the ANN performance matches the intended goals. This process uses the batch gradient descent algorithm which utilizes the entire training set for each iteration of backward propagation.

Convolutional Neural Networks

This work utilized a more complex ANN, the much deeper and more complex CNNs. These CNNs were chosen for the task of inspecting the DEBC as they have made a resurgence in general visual recognition tasks in recent years, overtaking other methods in image classification challenges [15, 23]. This was further exemplified by the previously mentioned Pascal VOC challenge, a yearly challenge from 2005-2012, with the goal of recognizing objects from several visual object classes through a supervised learning process [13]. The challenge has commonly been used as a comparison between different object detection networks [23]. CNNs have consistently outperformed other methods and have demonstrated increased precision of detection in the Pascal Visual Object Classes (VOC) challenge.

A CNN is a form of ANN which was developed for general object classification from images [11, 12, 14, 15, 16]. CNNs follow the same basic principles as described in the section above with ANNs but are much more complex adding many different techniques to improve performance. In the following sections, these differences will be described in detail starting from the adjustments to images as input, and as the different layer types as convolutional, max pooling, and final fully connected with SoftMax layers.

Input. With a focus on images, the input to a CNN differs in key areas. Due to the typical focus of CNNs working with images, the input to the network is provided as the raw pixel values of the image as three dimensions. These three dimensions are represented as the number of pixels in an images width, height, and depth where a depth of three provides the three red, green, and blue color channels. A CNN requires a specific size image, therefore each image provided to the CNN must be scaled to the specified size by

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{12}$$

where x' and y' are the desired width and height in pixels, x and y the provided width and height in pixels, and s_x and s_y the scaling factors for the width and height respectively. Once the image is resized to the new size, a method to interpolate the pixel intensity values is often used to estimate pixel values in unknown locations. Bicubic interpolation is among the most precise methods resulting in smooth gradations and was used in this work [31]. Bicubic interpolation computes a weighted average considering a four by four sized area or sixteen pixels. This is accomplished through use of a filter kernel computed as

$$h(x) = \begin{cases} 1 - (a+3)x^2 + (a+2)|x|^2 & if\ |x| < 1 \\ a(|x|-1)(|x|-2)^2 & if\ 1 \le |x| < 2 \\ 0 & otherwise, \end{cases} \tag{13}$$

where a represents the derivative at $x = 1$, often set to -0.5 producing a quadratic reproducing spline [31].

To aid in training performance, the input to the CNN is adjusted by subtracting the image mean. To subtract the image mean, the image mean must first be calculated by

$$x_\mu = \frac{\sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{d=0}^{D} x_{i,j,d}}{N}, \tag{14}$$

where i and j are the current image coordinates in the x and y position and d the depth as one of the three image color channels with $H, W,$ and D the maximal height, width, and depth of the image respectively. The term N in equation 13 represents the total number of pixels in the image in all three of the color channels. Once the image is scaled to the desired size and the image mean for the entire training data set is computed, the image mean is subtracted as

$$x = x - x_\mu, \tag{15}$$

from all images provided to the CNN.

Convolutional Layers. The convolutional layers use regions of locally connected neurons and are the workhorse of CNNs by acting as large banks of learnable convolutional filters. Convolutional filters are a type of neighborhood operator that may be applied to images in which the output pixel intensity value is determined by the local weighted sum from the pixel input values [31]. The weighed values and size of the local neighbors are defined by a kernel or mask which follows a sliding window approach through all elements of the original image. This convolutional process can be expressed as

$$g_{i,j} = \sum_{k,l} f(k,l)h(i-k, j-l)\,, \tag{16}$$

in which $g_{i,j}$ represents the new element of the image at location (i,j) of the image in which convolution is being performed, and h the convolutional kernel. Convolutional kernels are capable of a wide variety of effects with simple kernels capable of blurring, sharpening, and detecting edges in images. With CNNs using large numbers of learnable convolutional filters, they can learn to identify large numbers of features of an object in an image, and if enough of these features are found, classify said image as that class of objects. A singular pixel example of convolution on an image can be found in Figure 6.

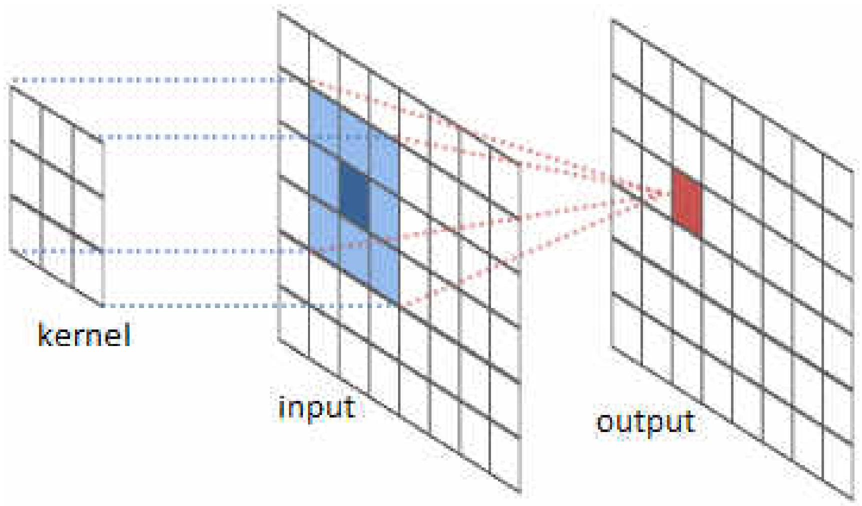


Figure 6: Visual interpretation of a convolutional filter enacted upon a single pixel of an image.

The first layer after the input image typically consists of the convolutional layer [11, 12, 14, 15, 16, 17, 53]. During forward propagation, these spatially connected regions are then convolved over the previous layer, mathematically represented as

$$y_{i^{l+1},\, j^{l+1},d} = \sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{d^l=0}^{D^l} f_{\,i,j,d^l} * x^{l}_{i^{l+1}+i,j^{l+1}+j,d^l}, \tag{17}$$

and then provided to the next layer, effectively creating learnable convolutional filters. In the above equation, f represents all the kernels in layer l where H, W, and D^l represent the total height, width, and depth of the previous layer respectively. This is implemented in practice by

$$vec(y) = vec(x^{l+1}) = vec(\phi(x^l)F), \tag{18}$$

in which vec represents the vectorization operator converting a higher dimensional tensor into a column-first order vector. Therefore, $vec(y)$ becomes a vector containing the output from the convolutional layer, $\phi(x^l)$ represents a matrix containing all inputs from the previous layer, and F a matrix containing all filter kernels as a fourth order tensor with HWD^l rows and D columns. Each convolutional layer typically contains many (potentially hundreds) of different filters, combines the results, and can be stacked with multiple convolutional layers following the current layer. The output from a convolutional filter provides a feature map that has the same height and width as the previous layer but increases the depth proportional to the number of learnable filters specified for that layer [11, 12, 14, 15, 16, 17, 53].

To adjust the weights in the convolutional layers to train the CNN, the loss function representing the cost the image matches the annotated training image calculated after a full forward pass, as described later, is minimized similarly to the ANN above, by adjusting weights throughout the CNN [11, 12, 14, 15, 16, 17, 53]. To adjust the weights of each layer in the CNN, two sets of gradients are computed and backpropagated through the CNN. These two gradients include the partial derivatives of a loss function z, as explained later, with respect to each layer's parameters, and the layers input. Adjusting the weights of the convolutional layers was performed computing the two derivatives as vectors where the gradient to update the convolution kernels was provided as $\frac{\partial z}{\partial vec(f)}$ and the gradient with respect to the input as $\frac{\partial z}{\partial vec(x^l)}$. The gradient to update the convolution kernel parameters is provided as

$$\frac{\partial z}{\partial vec(f)} = \phi(x^l)^T \frac{\partial z}{\partial vec(y)}), \tag{19}$$

and is used to update the parameters in the l-th layer. The gradient with respect to the input can be calculated by

$$\left[\frac{\partial z}{\partial vec(x^l)}\right]_{(i^l,j^l,d^l)} = \sum_{(p,q)\in m^{-1}(i^l,j^l,d^l)} \left[\frac{\partial z}{\partial vec(y)} F^T\right]_{(p,q)}, \tag{20}$$

where m represents the mapping of the index (p, q) in $\phi(x^l)$ in which $p = i^{l+1} + (H^l - H + 1) * j^{l+1}$, and $q = i + H * j + H * W * d^l$ [11, 12, 14, 15, 16, 17, 53]. With all the information necessary to both utilize and train the convolutional layers provided, the pooling layer can be described.

Pooling Layers. The purpose of the pooling layer is to reduce the dimensionality of the feature maps [11, 12, 14, 15, 16, 17, 53]. Decreasing the dimensionality reduces the amount of information and is generally performed several times throughout the CNN architecture to reduce the number of parameters and computation in the network, while also helping to control overfitting. Pooling layers operate independent on each depth slice of the input and resizes the feature maps, often using the max operation. In max pooling, the pooling operator maps a subregion, generally a 2x2 convolutional window with a stride of two, to the maximum value in that subregion. The stride controls the number of pixels skipped per subregion to prevent overlap of the kernel. Mathematically this is expressed as

$$y_{i^{l+1},j^{l+1},d} = \max_{0\le i<H,0\le j<W} x^{l}_{i^{l+1}*H+i,j^{l+1}*W+j,d}\,, \tag{21}$$

in which $0 \le i^{l+1} < H^{l+1}$, $0 \le j^{l+1} < W^{l+1}$, and $0 \le d < D^{l+1} = D^{l}$. Due to max pooling being a local operator, the computation is relatively simple for the forward pass [11, 12, 14, 15, 16, 17, 53].

Pooling layers do not require any parameters, and therefore performs no learning [11, 12, 14, 15, 16, 53]. During backpropagation, this results in the gradient with respect to the parameters, $\frac{\partial z}{\partial f} = null$. The gradient with respect to the input must still be calculated to adjust any layers performing any learning before the pooling layers. This is performed by

$$\frac{\partial z}{\partial vec(x^{l})} = S(x^{l})^{T}\frac{\partial z}{\partial vec(y)}, \tag{22}$$

where $S(x^l)$ is an indicator matrix in which a triplet of indices $(i^{l+1}, j^{l+1}, d^{l+1})$ specifies the row in S, and the column is specified by (i^l, j^l, d^l), and is defined as $S(x^l) \in \mathbb{R}^{(H^{l+1}, W^{l+1}, D^{l+1}) \, x \, (H^l, W^l, D^l)}$. This creates a very sparse matrix with exactly one nonzero entry per row and the location of the nonzero entries are recorded for use in the previous layer during backpropagation [11, 12, 14, 15, 16, 53].

Fully Connected with SoftMax Layers. Once the defined convolutional and pooling operations are completed, the fully connected layers are utilized, often with softmax, for generating the output for the defined classes the CNN predicts [11, 12, 14, 15, 16, 17, 53]. Fully connected layers work as described in the ANN above where the computation of any element for the output of x^{l+1} requires all elements from the input x^l. For simplicity, the fully connected layers can be calculated as a convolutional layer whose convolutional kernels are the same as the input where a convolutional kernel of size $H^l \, x \, W^l \, x \, D^l$ is used for the input layer size $x^l = H^l \, x \, W^l \, x \, D^l$. With D kernels, this creates a fourth order tensor with an output as $y \in \mathbb{R}^D$. With this information, the learning rules for the fully connected layers can utilize the same ones provided in the convolutional layers above. The final layer provides the probability that the image in question belongs to a set of defined classes and generally utilizes a softmax layer due to using the softmax activation function [11, 12, 14, 15, 16, 17, 53].

The VGG16 CNN architecture used in this work utilizes a softmax layer that performs a multinomial logistic regression objective to calculate the loss function representing the error in the classification [11, 12, 14, 15, 16, 17, 53]. Due to the need to perform multi-class classification, we define the output as a hypothesis that given a test input θ (in this instance an image), we estimate the probability the class label taking K different possible values or $P(y = k|x)$ for each value of $k = 1, \dots, K$. The softmax activation effectively provides the sigmoid activation described previously over many different classes as opposed to the binary classification scheme previously proposed. This output provides a normalized K-dimensional vector providing the K probabilities estimated. This hypothesis is described as

$$y = \begin{bmatrix} P(y = 1|\theta; x) \\ P(y = 2|\theta; x) \\ . \\ . \\ . \\ P(y = K|\theta; x) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(x^{(j)T}\theta)} \begin{bmatrix} \exp(x^{(1)T}\theta) \\ \exp(x^{(2)T}\theta) \\ . \\ . \\ . \\ \exp(x^{(K)T}\theta) \end{bmatrix}, \tag{23}$$

for a given test image. As mentioned previously, a cost function is utilized to update the CNN parameters throughout the architecture. This loss function takes the form of,

$$z = -\left[\sum_{i=1}^{n} \sum_{k=1}^{K} 1\{y^{(i)} = k\} log \frac{\exp\left(x^{(k)T}\theta\right)}{\exp(x^{(j)T}\theta)}\right], \tag{24}$$

where n represents the annotated samples and $1\{\dots\}$ is an indicator function so that $1\{\text{a true statement}\} = 1$ and $1\{\text{a false statement}\} = 0$. This is similar to the softmax regression loss function

$$P(y^{(i)} = k|\theta^{i}; x) = \frac{\exp\left(x^{(k)T}\theta^{i}\right)}{\sum_{j=1}^{K} \exp(x^{(j)T}\theta^{i})}, \tag{25}$$

which we sum over all K different potential values of the class label. An iterative optimization method must be used to solve for the parameters providing the minimal z for Equation 25 above due to the inability to solve it analytically. Taking the partial derivative, the gradient

$$\frac{\partial z}{\partial vec(x^{k})} = -\sum_{i=1}^{n} (\theta^{(i)}(1\{y^{(i)} = k\} - P(y^{(i)} = k|\theta^{i}; x)), \tag{26}$$

can be determined. The softmax layer contains no parameters, therefore $\frac{\partial z}{\partial vec(f)} = null$ [11, 12, 14, 15, 16, 17, 53].

The iterative optimization method utilized the mini-batch stochastic gradient descent algorithm [11, 12, 14, 15, 16]. A mini-batch is a variant of the batch gradient descent described previously in which the training set is divided into discrete smaller batches of samples selected stochastically. The SGD algorithm, represented as

$$vec(f^l) = vec(f^{l-1}) - \lambda \frac{\partial z}{\partial vec(f^l)}, \tag{27}$$

where λ was the learning rate, and the parameters are updated by the loss function from the set of examples from the mini-batch [11, 12, 14, 15, 16, 53].

Convolutional Neural Networks Architectures Used

To alert maintenance engineers of partial weld failures due to cracks of a DEBC, the ability to detect the weld portion of the DEBC was first required. It was assumed images in which the DEBC was to be detected were of inspection images collected via small UAS. Once the DEBC weld was detected, the region encompassing that weld was cropped, and the resulting segmented image was then evaluated on its condition as either cracked, or in good condition. To accomplish this, both the detected DEBC weld portion, and crack evaluation were performed using CNNs.

To first locate any potential DEBC welds in a given image using an object detection network, the Python reimplementation of Faster R-CNN was obtained from [18]. As previously mentioned, Faster R-CNN incorporates a small region proposal network that shares a common set of convolutional layers with a standard detection network and was built using the Caffe framework [24]. This region proposal network takes an image of any size as input, and outputs rectangular object proposals with a binary objectness score, or in other words, a measurement of belonging to a set of defined object classes *vs*. the general background. Training the layers specific to the region proposal network was accomplished through assigning a binary classification as positive for an object or negative for not an object to anchor boxes introduced by Faster R-CNN.

Anchors are the predicted bounding boxes given by Faster R-CNN to provide the region proposal for classification. These anchors are assigned as positive or containing an object of interest when they either have the highest Intersection-over-Union (IoU) overlap with a ground truth box, or IoU overlap greater than 0.7 with any ground truth box during training. Otherwise they are set as negative. Non-positive anchors do not contribute to training. The RPNs loss function for an image,

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*)\,, \tag{28}$$

is minimized to generate a trained RPN. In the loss function, *i* was the anchor index in a mini batch, p_i the predicted probability of the *i*'th anchor being an object. p_i^* represents the ground-truth label and was set to 1 for a positive anchor, and 0 if negative. t_i provides the vector representation of the four parameterized coordinates of the predicted bounding box, with t_i^* the ground-truth box associated with the positive anchor. The classification loss L_{cls} represents the log loss over the two object *vs*. non-object classes. The regression loss $L_{reg} = smooth_{L_1}(t_i - t_i^*)$ where $smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$. The L_{cls} and L_{reg} functions are normalized with N_{cls} as the number of anchors in a minibatch

(512), N_{reg} as the number of total anchors, and the balancing weight λ. Faster R-CNNs RPN generates approximately 2,400 different anchor positions. Each anchor position was tested at three different aspect ratios as 1:1, 2:1, and 1:2, and at three different scales as 128, 256, and 512. This leads to 21,600 anchor positions. These are reduced by removing cross-boundary anchors along with non-maximum suppression. Cross boundary anchors remove any anchors that extend beyond the image size. Non-maximum suppression removes any overlapping anchors with less IoU values than the highest one with the ground truth bounding box leaving N_{reg} as ~2,000. The classification of the region can then be performed once the regions for the object detected are determined. In this work, the VGG16 model was used to perform this classification.

The VGG16 model utilizes a total of 16 convolutional and max pooling layers, along with three fully connected layers, and ending with a soft-max layer [17]. This architecture was arranged by alternately stacking two convolutional layers followed by a max pooling layer twice, then increasing the stack of convolutional layers to three for the next three sections. The last layers comprise the three fully connected layers which feed into the final soft-max layer representing the class designations [17].

To classify if a DEBC detected from above contained a partial failure due to cracks, the Caffe [24] implementation of the VGG16 CNN architecture [17] was utilized. This VGG16 model matches the classifier portion of Faster R-CNN above for DEBC detection but lacks the region proposal network to detect the component. The method of using two different CNNs to locate the DEBC weld and to classify cracks in that region was to reduce the amount of down sampling involved with Faster RCNN's overall resizing due to Faster RCNN resizing the original image to 1000x600 or 600x1000 pixels, depending on the original image size [18]. With Faster R-CNNs RPN sharing computation with the object detection network (VGG16), the classification would occur on the detected region found in the resized image of 1000x600 pixels. To illustrate this difference in resolution, Figures 7 and 8 on the following pages provide sample images of two different DEBC welds cropped from the same region of both the original images of size 6000x1000, as well as the 1000x600 size images Faster R-CNN uses. As can be seen the welds from the smaller Faster R-CNN resized images are of lower quality and do not contain the same resolution as the original images making small cracks more difficult to detect. All image resizing used the binomial interpolation method as described above in Equation 13. Cropping the detected DEBC from the original image allows larger cropped sections and therefore potentially more information before resizing the image to the VGG16 required image size of 224x224 pixels.

The next section contains an overview of other CNN architectures that were considered other than that described above. These other CNN architectures were not used in the following work as the VGG16 architecture with the Faster R-CNN algorithm was found to outperform these other methods. This next section will provide comparisons in image classification challenges to provide comparisons on ability to recognize objects in images.

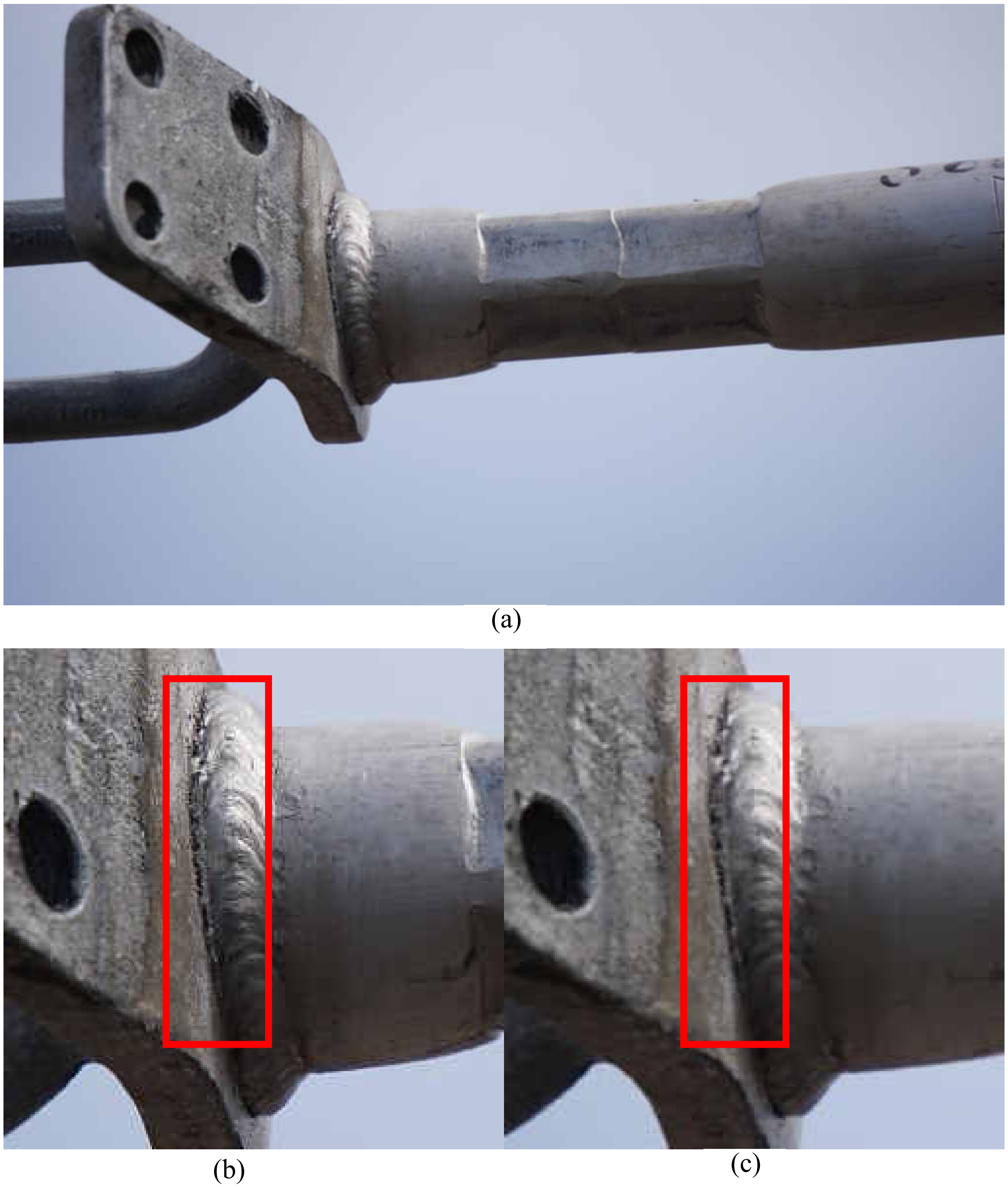
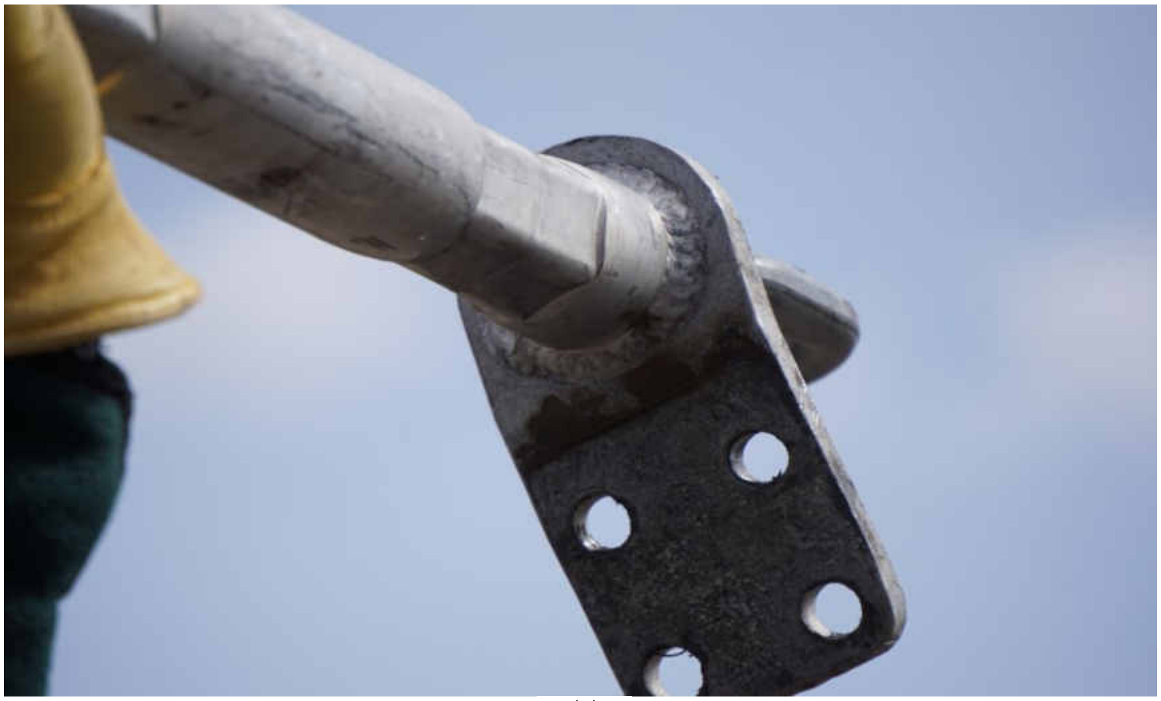
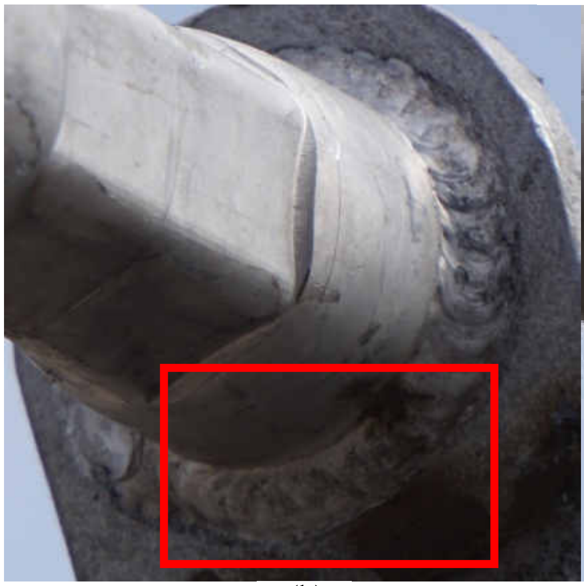


Figure 7: Images illustrating differences in resolution where red regions containing cracks of (a) Original image (b) DEBC weld region of original image (c) DEBC weld region of resized 1000x600 image Faster R-CNN uses.


(a)


(b)

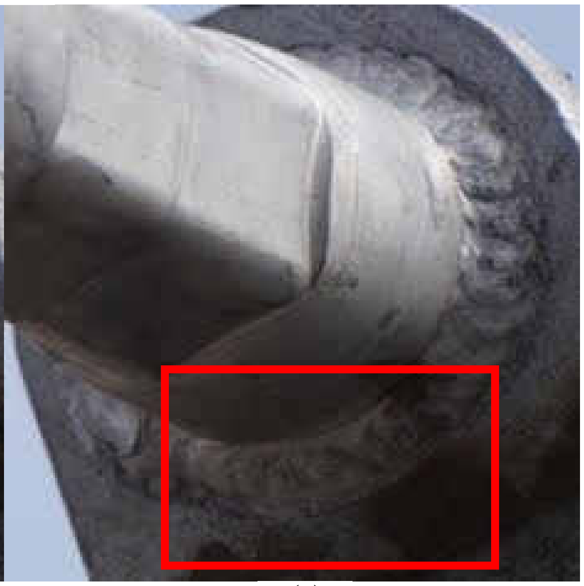

(c)

Figure 8: Images illustrating differences in resolution of (a) Original image (b) DEBC weld region of original image (c) DEBC weld region of resized 1000x600 image Faster R-CNN uses.

CNN Architectures Considered

To determine the most effective object detection CNN available, several different object detection CNN's were considered including R-CNN [19], SPPnet [20], YOLO [21], and Faster R-CNN [18]. Due to the post processing focus of the proposed algorithm, accuracy and precision were the main considerations for each network. A direct comparison of each network on the Pascal VOC 2007 [13] dataset as represented by the mean average precision (mAP) attained on the challenge by each network can be found in Table 1 below.

Table 1: mAP of each detection network considered.

CNN Considered	mAP in Pascal VOC 2007
R-CNN	58.5%
SPPnet	60.9%
YOLO	63.4%
Faster R-CNN	73.2%

Faster R-CNN was the latest improvement over R-CNN and introduced Region Proposal Networks (RPN) that share convolutional layers with the object detection network [18]. Region proposals are specific regions in the image determined as an object, previously provided by a separate algorithm per both R-CNN and Fast R-CNN [19, 22], and performing a convolutional network forward pass for each proposed region. Input images are resized to either 1000x600 or 600x1000 depending on whether the original image was taller or wider. The region proposals are created by adding two additional convolutional layers. The first layer encodes each convolutional feature map position into a feature vector. The second layer outputs an objectness score and regressed bounds for k region proposals, relative to various scales and aspect ratios, at each convolutional map position. These added layers create a fully-convolutional network that can be trained end-to-end for the task of generating detection proposals. Faster R-CNN also developed a training scheme that alternates between fine-tuning for the region proposal task, and fine-tuning for object detection with the proposals fixed. Faster R-CNN was chosen for our purposes as it achieved the highest mAP of all methods considered at 73.2% [18].

R-CNN was the first successful object detection algorithm utilizing a CNN, and increased mAP of the previous state of the art method by over 30% [19]. This was done by utilizing region proposals as provided by a separate algorithm. Although originally successful, there are many drawbacks to the region-based convolutional neural network. First, training was a multi-staged pipeline requiring finetuning the CNN on object proposals generated separately, then fitting a support vector machine (SVM) to the CNN features, and performing bounding box regression. Second, training was expensive in both hard drive space and time spent during training. The SVM and bounding box regression training requires storing features extracted from each object proposal to disc which may require storing hundreds of gigabytes of data. Third, detection was slow as features are extracted from each object proposal in each test image. Methods considered below improve both speed and precision over this implementation [19].

Spatial Pyramid Pooling (SPP-net) was proposed to speed up R-CNN by sharing computation [20]. SPP-net first computes a convolutional feature map for the entire input image, then classifies each object proposal using a feature vector from the shared feature map. Features are extracted through max pooling the portion of the feature map inside the proposal into a fixed output size. Training was still multi staged, as it must extract feature vectors, fine tune the network with log loss, train an SVM, and finally fit bounding box regressors. SPPnet cannot update the convolutional layers preceding the spatial pyramid pooling limiting accuracy of deep networks [20].

You only look once (YOLO), was a real-time object detection CNN [21]. This method applies a single neural network to the full image at test time to provide global context, divides the image into equally spaced regions, and predicts bounding boxes and probabilities for each region. This method provides a fast object detection that runs in real time, up to forty-five FPS, for the more computationally expensive and accurate model. As stated in [21], this network provides a mAP of approximately 10% less than Faster R-CNN.

CHAPTER III

METHODS

This chapter details all methods used to accomplish the goals of this work. In the goal to identify and classify partial failures of DEBC welds due to toe-weld cracking, two different previously developed CNN architectures, Faster R-CNN with a VGG16 backbone to locate and crop any DEBC welds from UAS inspections imagery, and another purely VGG16 CNN to classify the likelihood the cropped DEBC welds contain a partial failure due to toe-weld cracks, were trained/fine-tuned for the new task. This work involved slightly altering the two different CNNs to reduce the number of classes considered, collecting many images of the component in question and potential failures for training datasets, data augmentation of the images to provide invariance to different component conditions, and altering parameters including the amount of training iterations of the CNNs to improve performance. When tested the resulting CNNs provided a 97.8% accuracy to locate and crop the DEBC welds, 98.8% accuracy in a 5-fold cross validation strategy to classify cracked components, and an overall 73.8% accuracy with both CNNs combined on a difficult dataset. Each of the methods to train/fine-tune the new CNNs is expanded in detail in the following sections.

Data Collection

The original data to train and test the DEBC detection that was used to locate and crop DEBC welds from images was provided from two local businesses. The first provided by Field of View provided data from a study gathering simulated imagery [25], and the second provided by SkyScopes was of UAS inspections test flights. The provided simulated images collected data to determine multiple parameters for use onboard a UAS including the optimal camera sensor, viewing angle, and distance for a human to be able to identify potential maintenance concerns. Using that information, the UAS inspections test flights followed the recommendations provided and gathered live powerline inspections images of DEBC welds using UAS to test the effectiveness of UAS inspection flights for the purpose of identifying DEBC partial failures due to toe-weld cracks. This data provided images ideal for the purposes of this work.

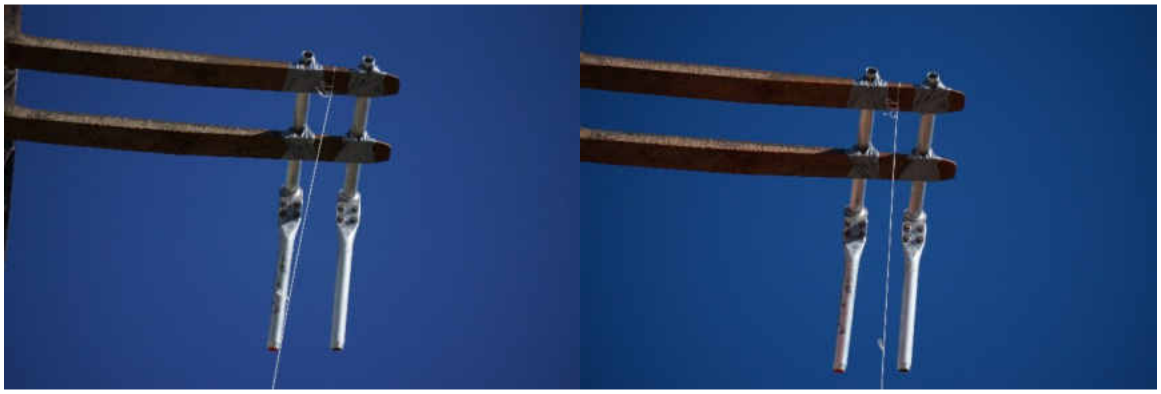


Figure 9: Images of DEBCs on bright sunny day 12m from visible Sony NEX-7 (left), and Sony a6000 multispectral (right) camera sensors.

In the provided simulated images, the cameras used to collect the data include the Sony NEX-7 [26], and a Sony a6000 [27] sensor converted to perform as a multispectral camera. Figure 9 provides a comparison of images provided by the two camera sensors. The converted Sony a6000 sensor provided 700-800nm wavelength light along with the standard visible light. Each image was of size 6000x4000 pixels. Two different DEBC’s, one which contained a cracked weld and one that did not, were attached to a forklift and lifted to the test height ranging from 4m to 12m from the camera height and tested under four different weather conditions. These weather conditions include a cloudless sunny day, a cloudless sunny day with the sun in view of the camera, a cloudy day, and a dark cloudy day. It was found that the differences in the Sony NEX-7 and converted multispectral Sony a6000 images were negligible for inspection purposes [25]. An example of each of the different weather conditions can be found in Figure 10. A total of 111 images from the simulated images, each of which contained two different DEBCs, were provided.

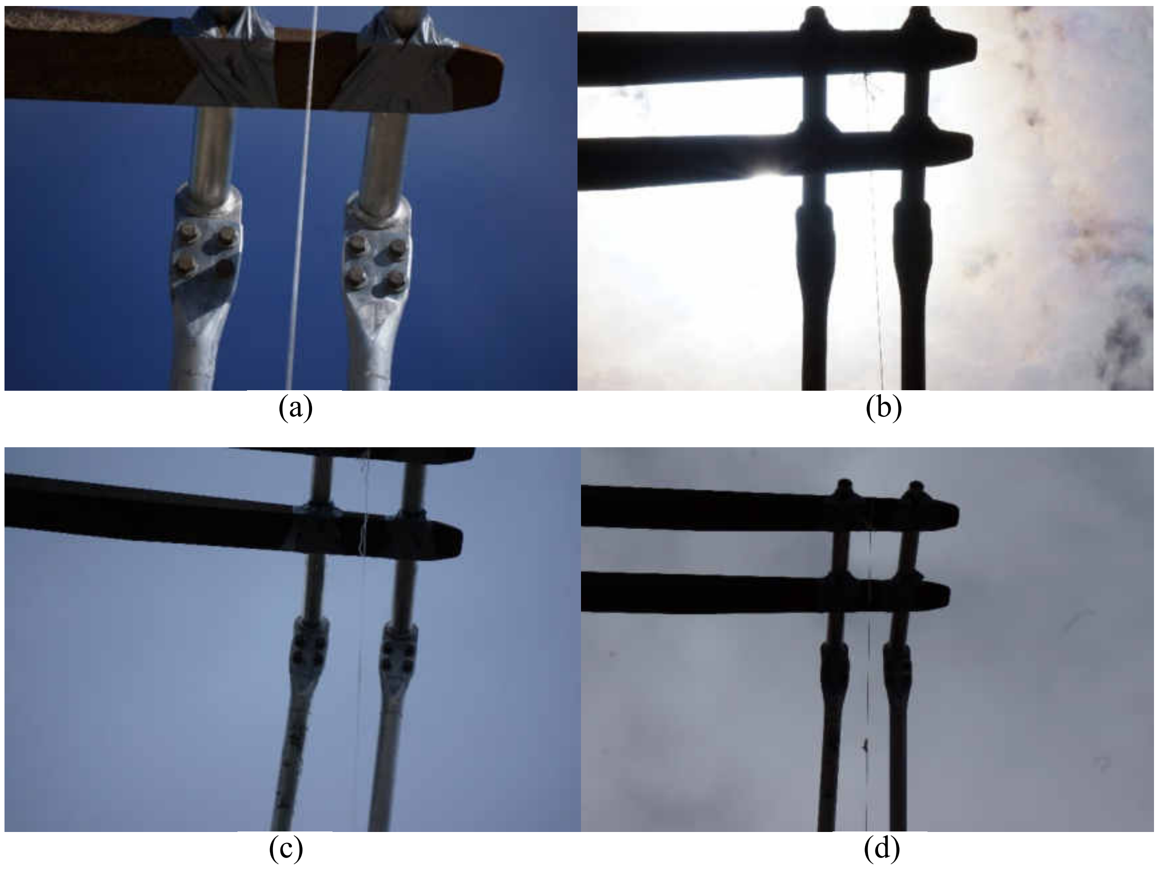


Figure 10: Images of each weather condition at varying distances/heights that simulated images were collected in including (a) sunny day (b) sunny day with sun in view (c) cloudy day (d) dark cloudy day.

From the UAS test flight, images were collected on live high voltage power lines which was accomplished using the FreeFly Alta 8 aircraft [28]. Thirty images were first collected from a test flight in which the UAS flew approximately 40m away. An additional 270 images were collected in which the UAS flew approximately 15-20m from the DEBCs imaged. The same Sony a6000 model camera [27] not converted to a multispectral sensor, as tested by Field of View with a 210mm lens, was used for data collection. For our purposes, only images with the DEBC within view were considered. From the UAS test flight, 30 training images were collected. A sample image of a FreeFly Alta 8 UAS that was utilized can be found in Figure 11 below.

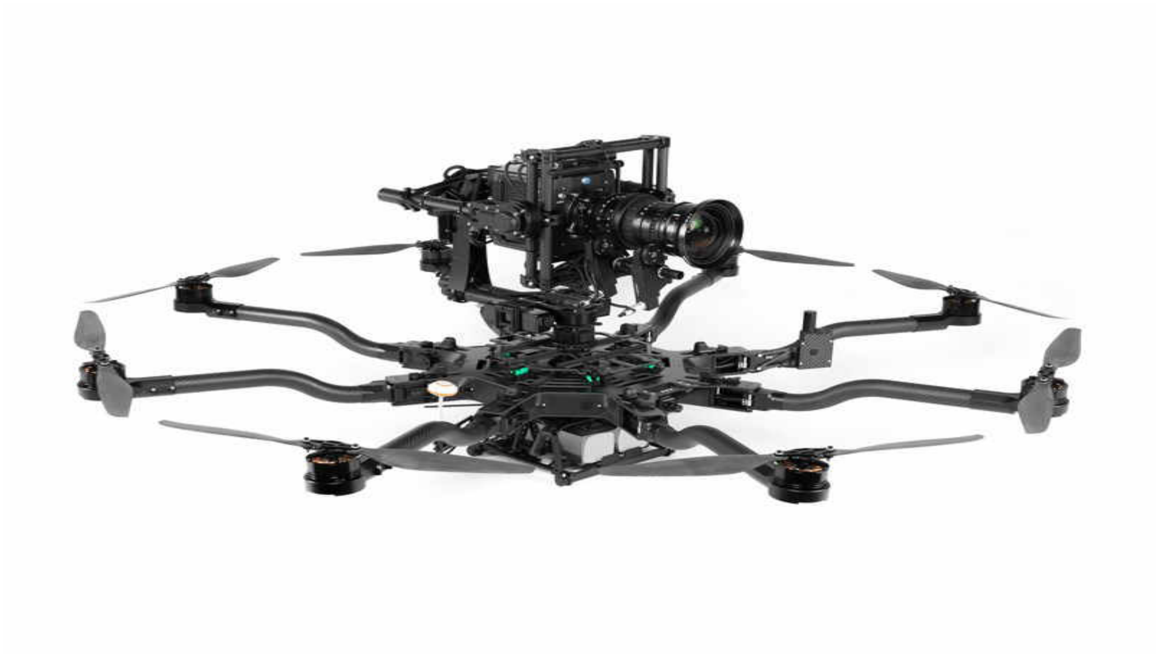


Figure 11: Image of a FreeFly Alta 8 UAS as used for inspection imaging of live DEBCs.

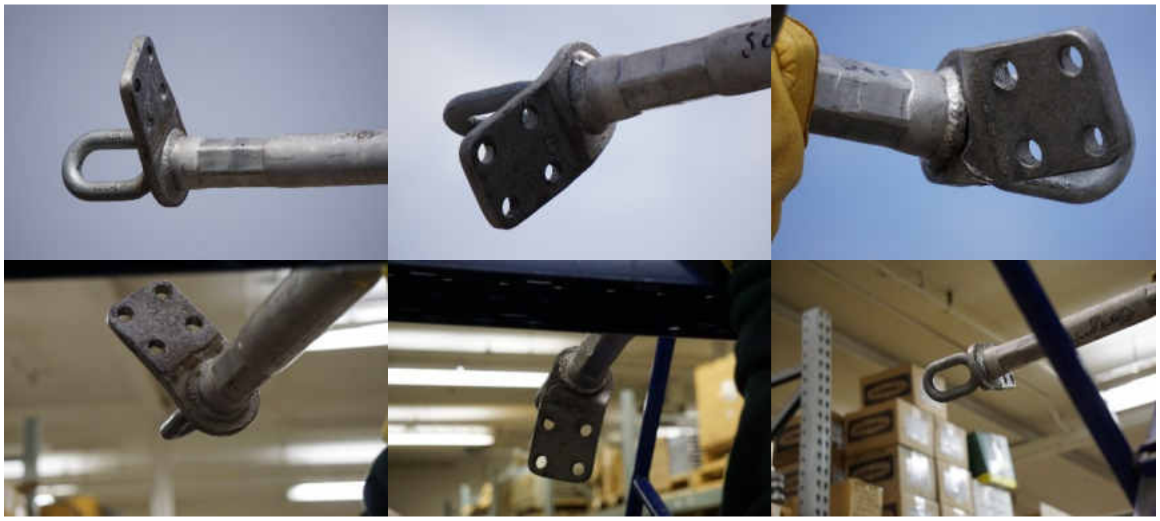


Figure 12: Example images from manually collected training data for crack classification. Top images collected outdoors, bottom images collected inside warehouse.

Due to a very small number of inspections imagery collected containing any crack features, a larger number of images containing DEBC welds were manually collected from previously replaced components, to ensure enough cracked images to train the crack classifier CNN were provided. An addition of 1,061 images were collected, both indoors and out, with varying distances and view angles. Examples of said images can be found in Figure 12. Due to the provided simulated images [25] containing direct comparisons between a cracked and non-cracked component, these images were also used. The provided simulated images also provided a direct comparison of cracked vs non-cracked DEBC welds under similar and variable lighting conditions aiding in lighting invariance. These images used the same non-modified Sony a6000 camera as above with a 210mm lens. Seven different welds were imaged in which five of the welds contained toe-weld cracks. Images were collected by manually holding the component above the camera at varying distance and view angles of the weld. While taking images, each welded component was also rotated at least 360º. This larger data set provided many of the possible angles and distances the component could be viewed from.

Given a total of 416 images, the training data set required a large amount of data augmentation to be performed to allow for a sufficiently large training set. As a comparison, the Pascal VOC [13] training dataset provides close to 5,000 images per category to be considered. The addition of another class of images considering insulators was added to help the network differentiate between insulators and DEBCs. Inclusion of the insulator class was due to background insulators causing a high number of false positive detections as discussed in Chapter IV. Sample images of the original training images provided can be found in Figure 13 below.

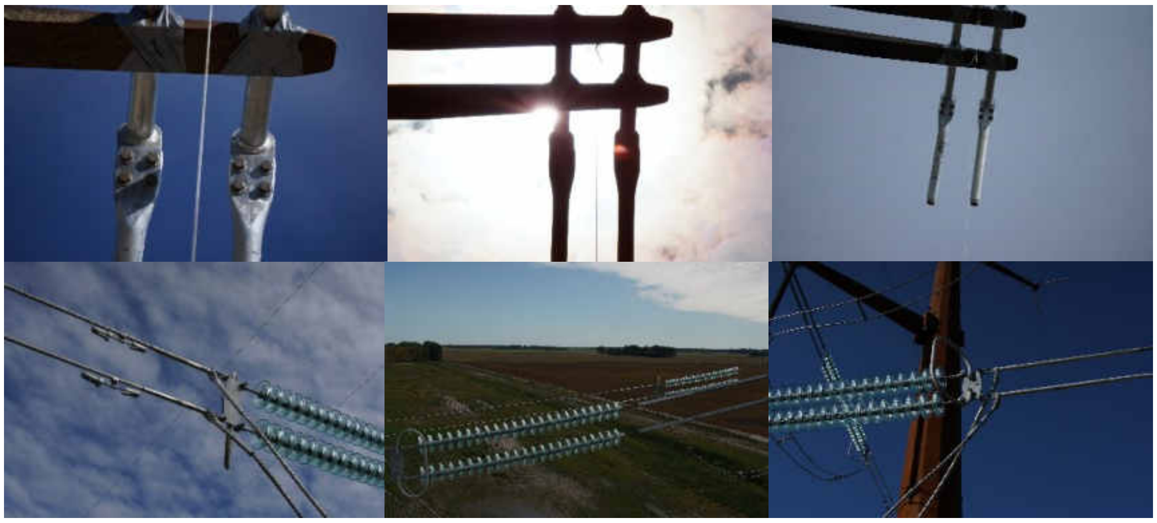


Figure 13: Example images from training data. Top images from simulated imagery, bottom images from test flight.

Data Augmentation

To generate enough data to properly train the two different CNNs to accurately detect the DEBC and then classify if the detection contains a crack, a series of simple image processing techniques were performed. All image processing methods were completed using OpenCV functions [29]. Due to the small number of images provided for the DEBC detection, most data augmentation was performed only for the DEBC detection training dataset.

The first data augmentation method was only performed on the data used in the DEBC detection and involved manually cropping the DEBC from each original image. Due to the Faster R-CNN algorithm automatically resizing all input images to 1000 pixels on the larger side, and 600 pixels on the smaller side, the cropped images allowed for increasing the robustness of the network to differences of scale. The cropped images were also used for multiple data augmentations as described below.

To account for various viewing angles in which the DEBC may be oriented, each cropped image from before was manipulated to create a series of rotations. These rotations were only performed on the DEBC detection training images. Rotations were performed using an SO(3) rotation matrix in degrees [30]. The rotation matrix as

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{28}$$

was applied to the image with R_z being a counterclockwise rotation about the z-axis.

Rotations were applied using inverse warping where each intended pixel location in the rotated image was computed, then the corresponding location in the original image was sampled [31]. The cropped images were rotated from their original position in 15º intervals to a total of 60º. The images were then cropped to remove the resulting black corners of the image from the rotations.

To account for possible out of focus or grainy images, two different morphological operations were performed on the images. Again, these operations were performed only on DEBC detection training datasets. These morphological operations included minor dilation and erosion [31]. This process was done by convolving a kernel (β) over the image I. β has a defined anchor point at the center of the kernel. For dilation, kernel β was convolved over the image and the maximal pixel value overlapped by β was computed and replaced by the image pixel in the anchor points position with that maximal value. This causes bright regions within an image to expand. Erosion was done similarly but instead uses the minimal pixel value for the anchor point causing bright regions. For our purposes, β was chosen to be of size [3x3] with only a single pass for slight erosion and dilation operations. Each of the above images were then flipped horizontally.

The last method performed on the DEBC detection data involved cropped 1000x1000 pixel sized patches in a raster scan pattern with 50% overlap from the original 6000x4000 images. This technique was performed to create translations of the DEBC, as well as allow for edge cases where the DEBC would be only partially visible due to being truncated at the edge of the cropped image. The 50% overlap was used to ensure parts of the image would always be visible. Lastly, the total images had to be manually sorted to remove images without the DEBC visible.

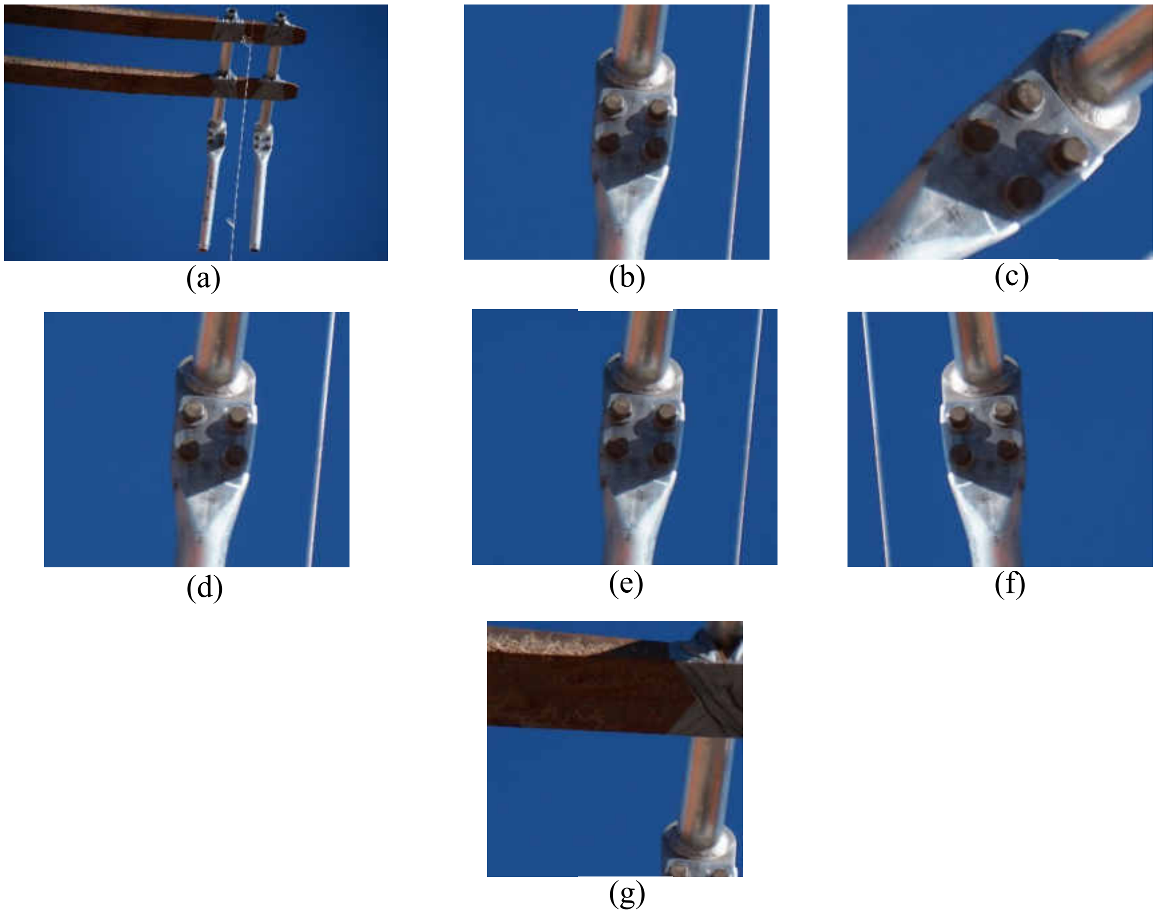


Figure 14: Example augmented data images (a) original image from simulated images (b) crop of left DEBC weld (c) rotation of component (d) dilation of component (e) erosion of component (f) horizontal flip of translation of component (g) translation through raster scan cropping

A total of 2,437 images were developed from these techniques to train the DEBC detection network from the original 111 simulated images provided. Sample images of the developed augmented images can be found in Figure 14 above. Along with the additional 270 UAS inspection images, this provided enough images to train the DEBC detection

CNN. With a severe lack of cracked components, many of these images were not used for the crack classification CNN as the number of non-cracked components would oversaturate the cracked ones.

Due to the larger number of true non-augmented images, the crack classification network performed less data augmentation than the DEBC detection images. To represent the cropped images from the DEBC detection, all components found in each image were manually cropped ensuring full visibility of the weld. These cropped images were developed to closely match the images the DEBC detection network would detect and segment as shown in the next section under Annotations. Due to the VGG16 CNN model utilizing only square 224x224 image sizes [17], the cropped images were created by determining if the height or width was larger, then setting the smaller side to the same value as the larger ensuring a square image. Each cropped image was then subjected to simple vertical and horizontal flips to increase the number of view angles. With all augmentations, a total of 4,618images were created and used for training this CNN. Figure 15 below provides a sample of the data augmentations performed for the crack classification CNN training dataset.

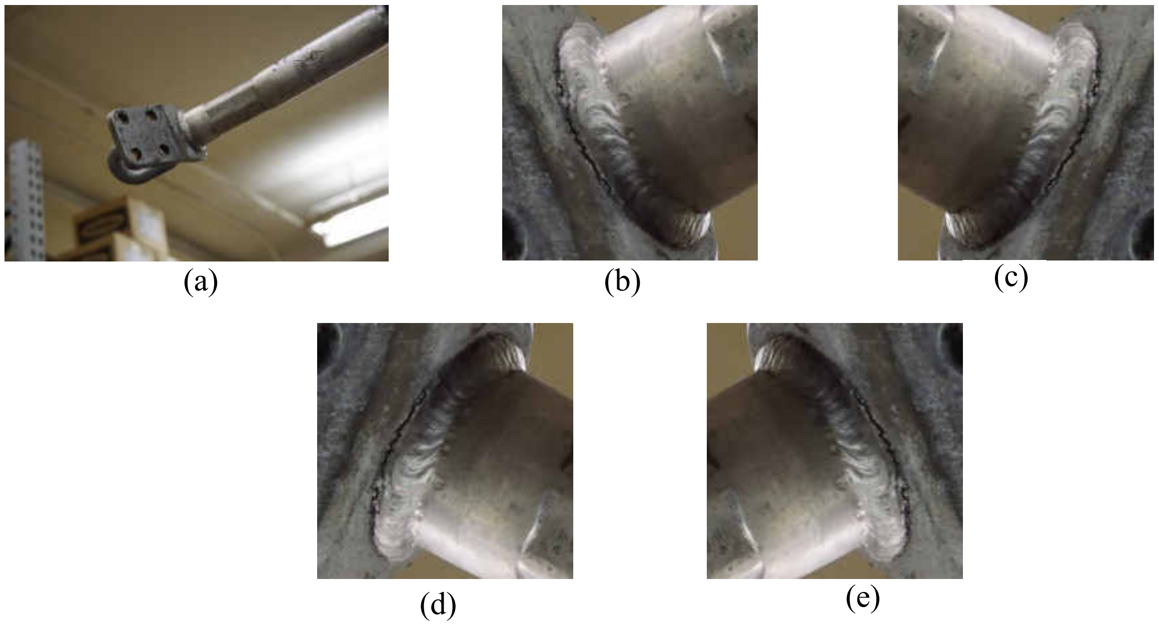


Figure 15: Example images from of data augmentation for crack classifying CNN (a) original image (b) cropped image of weld portion (c) horizontal flip of cropped image (d) vertical flip of cropped image (e) horizontal and vertical flip of cropped image

Annotation

Training both CNN models required all cases of the intended object categories to be annotated. The Faster R-CNN model required both the bounding box location, and object class for each component to be detected [18]. Bounding box locations were designated from the four coordinates as

$$\begin{aligned} t_x &= \frac{(x - x_a)}{w_a}, & t_y &= \frac{(y - y_a)}{h_a}, \\ t_w &= \log\left(\frac{w}{w_a}\right), & t_h &= \log\left(\frac{h}{h_a}\right), \\ t_x^* &= \frac{(x^* - x_a)}{w_a}, & t_y^* &= \frac{(y^* - y_a)}{h_a}, \\ t_w^* &= \log\left(\frac{w^*}{w_a}\right), & t_h^* &= \log\left(\frac{h^*}{h_a}\right) \end{aligned} \tag{29}$$

where the x, y, w, and h represent the bounding box's center (x, y) coordinate, and the width and height respectively. The x, x_a, and x^* represent the predicted bounding box, anchor box, and ground truth box respectively, similarly for the y, w, and h values as well. For the object classes, two classes were considered outside the background class, a catch all for all non-defined objects, as the DEBC weld, and finally background insulators. Each annotation for the DEBC weld was created to encompass the entire weld portion of the DEBC while limiting all other features. The second class, which considered the insulators often found in the background, was annotated, and added to the training set by encompassing the entire string of insulators as one object. With Faster R-CNN developed to train on the Pascal VOC dataset, the annotations performed matched the format using the LabelImg software [32]. The Pascal VOC format stores each bounding box annotations location, class, and image location on file in xml format [13]. All annotations were manually selected, and an annotation sample with highlighted bounding boxes is provided in Figure 16.

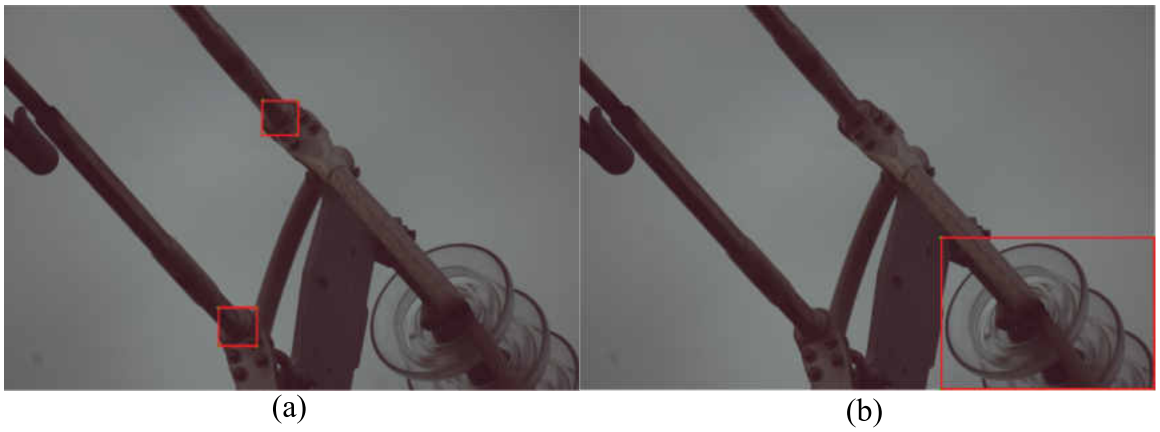


Figure 16: Example images of bounding box annotations for one image (a) DEBC welds (b) insulator

To annotate the crack classification CNN, all images of the cropped DEBC welds developed as described in the data augmentation section above were annotated in a binary classification method. Each image was reviewed and identified manually as either containing a cracked component or considered a weld in good condition. All images designated as in good condition were labeled as “0”, while all welds designated as cracked were labeled as “1”. These annotations were provided in a text file containing the image name followed by the numeric designations on one line for each image used. The VGG16 network [17] was then adjusted to perform the binary classification on just two image classes.

Testing Data

Images to test the network performance were necessary to evaluate the two CNNs once trained. Test images were kept separate from training data as doing otherwise would artificially increase metrics for the CNN as the CNN in question would already have been trained on that image specifically [11, 13, 14, 15]. The two different CNNs developed were tested using two different methods, due to the availability of image data.

To test the DEBC weld detection CNN, newer inspection images were provided by the company who performed inspection test flights. This test data included 111 images which include 115 total DEBCs taken from a closer range and viewing angle in which the DEBC was easier to view. Examples of these images can be found in Figure 17. These test images were kept separate from the training data.

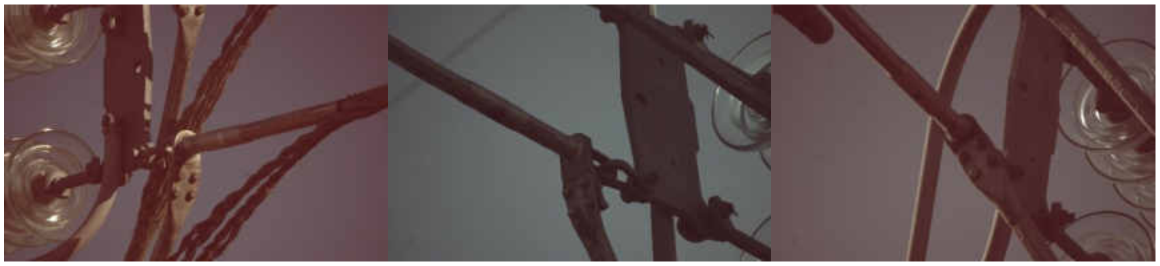


Figure 17: Example images of DEBC weld detection test images

The crack classification CNN was both trained and validated by performing a random 5-fold cross validation strategy [33]. After randomizing the training set of images, the entire set of images was separated into equally sized portions called folds. Five different networks were trained by training with four of the folds and using the remaining for validation and testing. The fold used for validation and testing was alternated so that each fold was used as validation once and only once. This produces five different trained networks and the resulting accuracy of each was then averaged to determine the approximate overall precision the total network would have had it been trained with the entire training set. Each crack classification network therefore contained approximately 3,694 training images, and 924 validation images.

Training

All training for each of the CNNs developed were trained utilizing a Titan X GPU. As per [18] the combined region proposal network with the complex VGG16 model requires approximately 11GB of GPU memory. The Titan X GPU was ideal as it provides 12GB of GPU memory.

To train the DEBC weld detection CNN, training was performed using the alternating optimization method per [5]. This method performed a 4-step training to learn shared features between the region proposal network, with the VGG16 model as a separate detection network for classification. First, the RPN was trained as above. This was accomplished by minimizing the objective function from equation 24 following multi-task loss. Second, the detection network, utilizing the VGG16 architecture, was trained separately without sharing layers. Third, the detection network was used to initialize the region proposal network training but fixed the shared layers, fine-tuning only the region proposal network layers. The final stage shared the convolutional layers, keeping them fixed, and fine-tuned the fully connected layers, which formed a unified network.

The DEBC weld detection networks evaluated were fine-tuned from an ImageNet pre-trained VGG16 model over differing numbers of iterations, and numbers of images following the methods in [14]. The learning rate was set as 0.0001 for 60k minibatches and 0.00001 for the next 20k minibatches, momentum as 0.9, and weight decay as 0.0005 as provided by the VGG16 model. Multiple networks were developed while varying the number of training iterations from 40,000 to 150,000, often alternating the higher number of iterations in the first and third stage, and the lower number of iterations in the second and fourth. These alternating numbers of iterations were done as the default iterations were set to alternate from 80,000 to 40,000. The most visually accurate networks based on the number of iterations run were chosen for a full evaluation of the network. Results of these more accurate networks can be found in Chapter IV. Time training the network was heavily dependent on the number of iterations but took approximately three to seven days of nonstop training.

The crack classification network also utilized the VGG16 network architecture. Output from the soft-max layer was adjusted to perform a binary classification as either a cracked component, or component in good condition. The CNN was then trained using a 5-fold cross validation strategy. Randomizing the training set of images, one fifth of the training set was removed to be used for validation and testing and repeated five times for each fold. This produces five different trained networks and the resulting accuracy of each was then averaged to determine the approximate overall precision of the total network. Each network was trained using a batch size of 32, learning rate of 0.001, momentum of 0.9, and weight decay of 0.0005. The networks were all trained to a maximum of 30,000 iterations.

Algorithmic Process

Once both networks were trained and adequately tested, they were combined as a singular system. To classify possible DEBC partial failures due to toe weld cracks, the algorithm developed used the two different CNNs developed above to detect and crop the DEBC weld portion of UAS inspections imagery and classify the likelihood the cropped image contains a crack or not. A flowchart of the entire algorithm can be found in Figure 18 and will be explained in further depth below.

Once the algorithm begins, the user is prompted to provide input for the location of an input and output folder for the inspections images. The input folder is expected to contain all inspection images to be classified, whereas all outputs from the algorithm will be later stored in the output folder. The algorithm then loops through images existing in the input folder, and if there exists an image that has not been considered, the algorithm makes a temporary resized image to the Faster RCNNs required size of 1000x600 pixels of the original image to detect DEBCs. The DEBC detection CNN is then loaded and using the temporary resized image, any detected DEBC weld regions is found in the image from the DEBC detection CNN output. That same region is then found in the original sized image and a square region is cropped from the original sized image using the largest side as the height and width of the new image, and then the new image is stored in the output folder. A new temporary image is created from the newly cropped image and resized to the VGG16 crack classification CNN size of 224x224. Using the new 224x224 pixel sized cropped image, the crack classification CNN is loaded and the

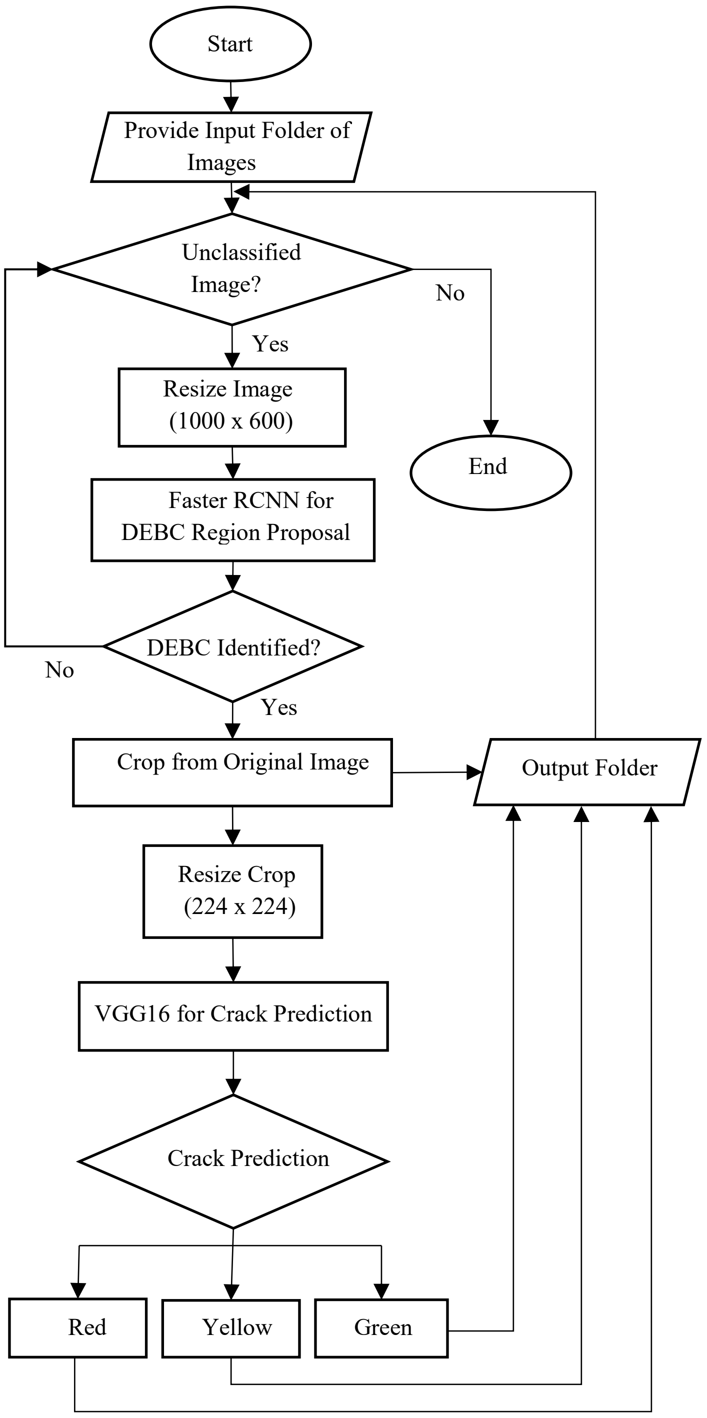


Figure 18: Flowchart diagram for entire algorithm with both CNNs used.

probability of the weld containing a crack is provided and grouped as either red, yellow, or green. The red classification is considered as most likely cracked, yellow is considered as questionable, and green as most likely not cracked. This classification is stored in a text file for each group with the name of the new image, and its classification. The algorithm then loops to the next image in the input folder to continue classifying all images in the input folder if more exist.

CHAPTER IV

RESULTS

The resulting fully trained CNNs, developed as explored in Chapter III, were tested with their respective testing data. First, the results of the DEBC weld detection CNN will be described, as tested on the 111 test images provided through UAS inspection test flights. Next the results of testing the crack classification CNN as tested with the 5-fold cross validation will be detailed. Lastly, the two networks combined was examined and the effectiveness of both CNNs working in conjunction was determined.

The 111 test images developed in Chapter III's Testing data section were used to evaluate several differently trained networks while varying both the number of iterations the network was trained with, and the number of training images to ensure a robust DEBC weld detection CNN. ROC and Precision-Recall curves were developed for each trained network to determine network accuracy. Data for the curves was generated by setting the threshold for detection to the value of 0.1 and storing all detections and confidence intervals.

Both the ROC and Precision-Recall curves for the DEBC weld detection CNNs were developed by recording the true positives, true negatives, false positives, and false negatives for the 111 test images while varying the threshold from 0.1 to 1 in 0.05 intervals. True positives were classified as matches if the detection appeared visually correct allowing for the user to easily view and evaluate the vast majority of the DEBC weld and its condition within the detected bounding box. True negatives were only considered from the list of false positives. As the threshold increased and the false positives were no longer detected, they became true negatives. False positives were defined as any detections that were not of a DEBC weld or did not contain the entire weld visible from the image. False negatives were tallied for any DEBC weld not detected in the dataset. Both ROC and Precision Recall curves were developed by calculating the recall/true positive rate (TPR), the false positive rate (FPR), and precision as $recall/TPR = \frac{TP}{TP+FN}$, $FPR = \frac{FP}{TN+FP}$, and $Precision = \frac{TP}{TP+FP}$ respectively. The ROC curve for all three networks considered was plotted using the TPR, and FPR, whereas the Precision Recall curve was plotted using the precision and recall at each threshold value. After finding a high rate of false positives due to detecting insulators as DEBC's, an additional 270 images from UAS inspections images were included in the network "with insulators" as a separate class. The network was retrained with the more successful number of iterations, 100,000, 80,000, 100,000 and 80,000.

As per [15], a curve only dominates in the ROC space if it also dominates in the precision recall space. The DEBC weld detection CNN trained with 100,000, 80,000, 100,000, and 80,000 iterations for the four training stages, includes insulators as a separate class, and was also trained with the additional 270 images dominates in both ROC and Precision-Recall space, especially at higher threshold values. Using the CNN corresponding to this curve. The ideal threshold was found by use of the F-measure.

Figure 19 provides both the ROC and Precision-Recall Curves.

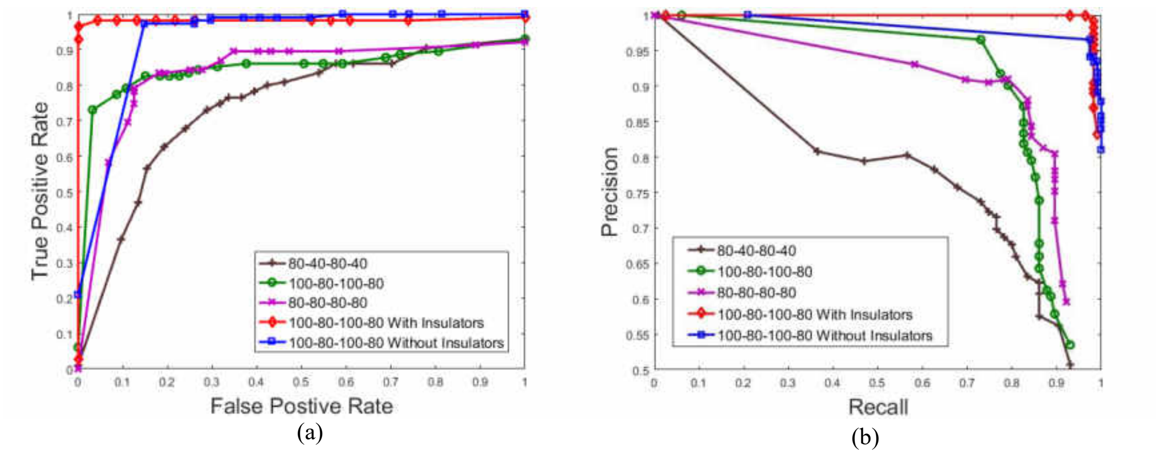


Figure 19: (a) ROC curve of five different networks, numbers list the iterations of each stage of training for that network (in thousands). The curve with insulators includes additional images and a separate class considering insulators, the curve without insulators includes the additional images, but not the separate insulator class. (b) Precision Recall curve for the same five networks in (a)

The general formula for the F-measure as,

$$F_\beta = (1 + \beta^2)\frac{Precision * Recall}{(\beta^2 * Precision) + Recall}, \tag{30}$$

represents a measure for the effectiveness of retrieving the intended information with β times as much importance to recall than to precision. A β value of two was chosen to determine the threshold value to use as false negatives were deemed as worse than potential false positives. The F_2 measure provided the ideal threshold as 0.85, the highest F_2 measure, and was found as $F_2 = 0.6168$. The confusion matrix for this CNN, when evaluated with the 111 test inspection images with a threshold of 0.85, can be found in

Table 5: Confusion matrix for network 100, 80, 100, 80 (in thousands) with additional images and insulator class, threshold 0.85.

	Predicted Negative	Predicted Positive	Total
Actual Negative	TN: 22	FP: 1	23
Actual Positive	FN: 2	TP: 113	115
Total	24	114	138

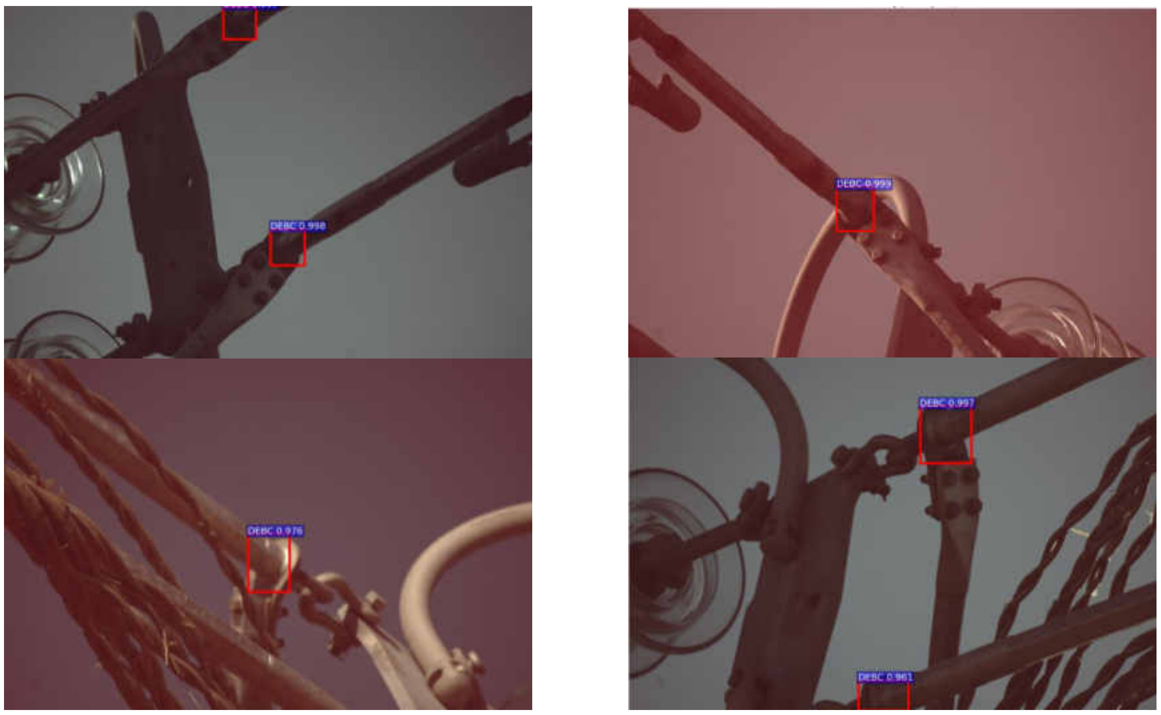
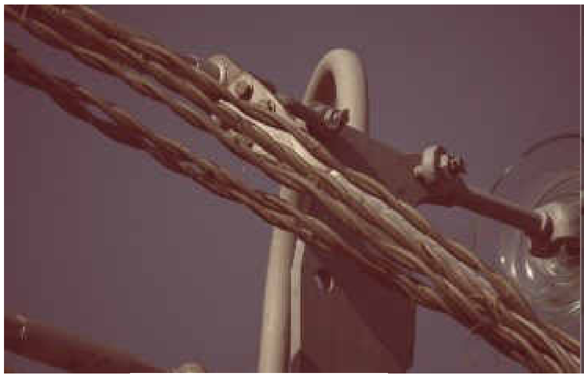


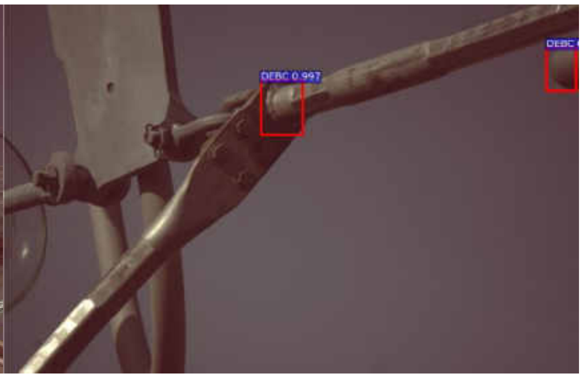
Figure 20: Example results of successful DEBC detections.

Table 5. From this data and threshold, the accuracy, $acc. = \frac{TP+TN}{Total}$, was also found.

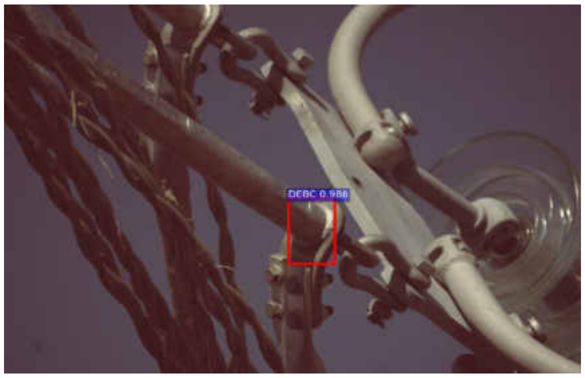
Accuracy was determined as 97.8% while maintaining a precision of 99.1%. As can be seen, the DEBC weld detection CNN performed remarkably well with only two false negatives, and one false positive from the possible 115 DEBC welds found in the test data set. Figure 20 above provides a few examples of how the network accurately located the DEBC weld in a variety of postions and poses, including DEBC welds truncated by the edge of the image. Figure 21 provides the failed detections.



(a)



(b)

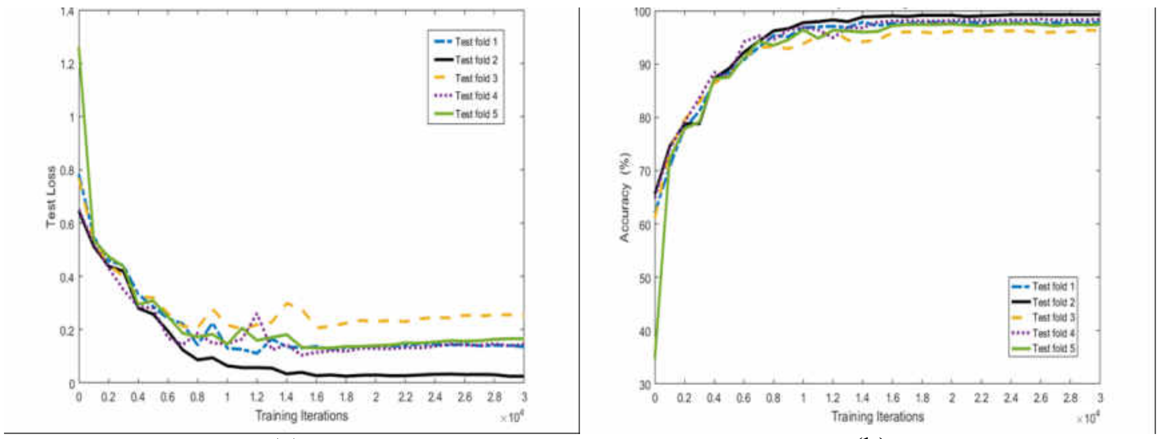


(c)

Figure 21: Failures in detection. (a) Occluded DEBC weld not detected. (b) False positive detection on far right due to truncated damper (c) Top truncated DEBC weld not detected

Though there were very few DEBC weld detection failures in this test dataset, some limitations of the CNN were found. False negatives were attributed to poor views of the welds. As can be seen in both false negatives, the weld was either heavily occluded as shown in Figure 21 (a), or partially truncated in (c). The DEBC weld detection CNN appeared to occasionally fail without full view of the weld. The false positive occurred due to a general rounded shape due the edge of a truncated damper on the powerline.

DEBC weld detection CNNs trained without insulators as a separate class also had several false positives due to insulators. This shows false positives may appear due to other rounded shapes the CNN was not trained to identify.



(a) (b)

Figure 22: Metrics determined for every 1000 iterations during training each of the 5-folds (a) Loss calculated (b) Accuracy

Results for the crack classifcation CNN were found through a 5-fold cross validation method. First each of the five folds were trained over 30,000 iterations to determine the optimal amount of training. This optimal amount should coincide with a minimum loss calculated on the test data. The data used for testing during each of the five folds had both the accuracy and loss determined after every 1000 iterations of training. Figure 22 provides a graph of both the loss and accuracy of the test data over the entire 30,000 training iterations. From the graph found in Figure 22 (a), it can be seen that the average minimal test loss was achieved at 16,000 iterations, and was calculated as 0.12372. The accuracy at 16,000 iterations was found to be 97.55%, close to the maximum accuracy found.

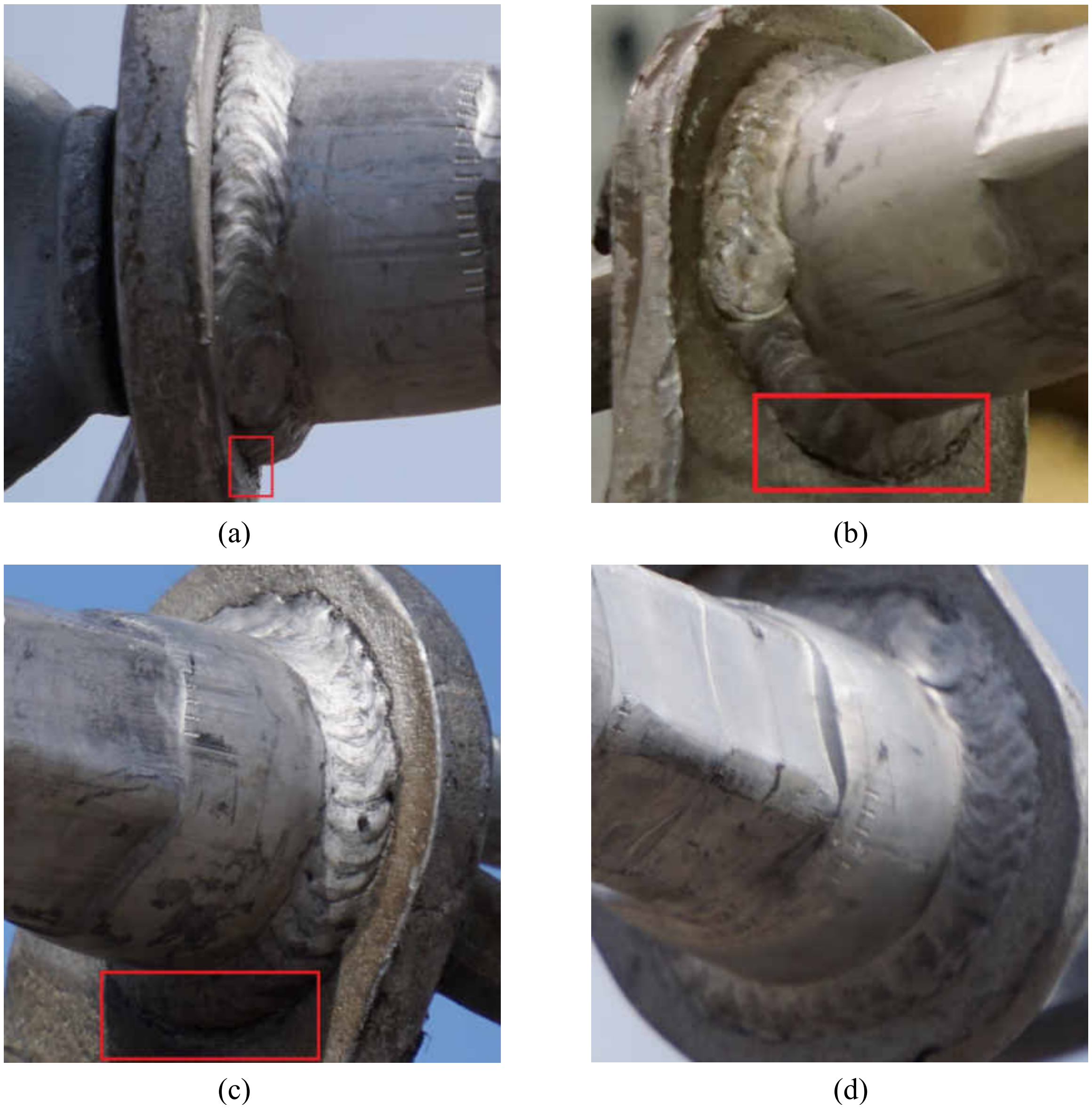
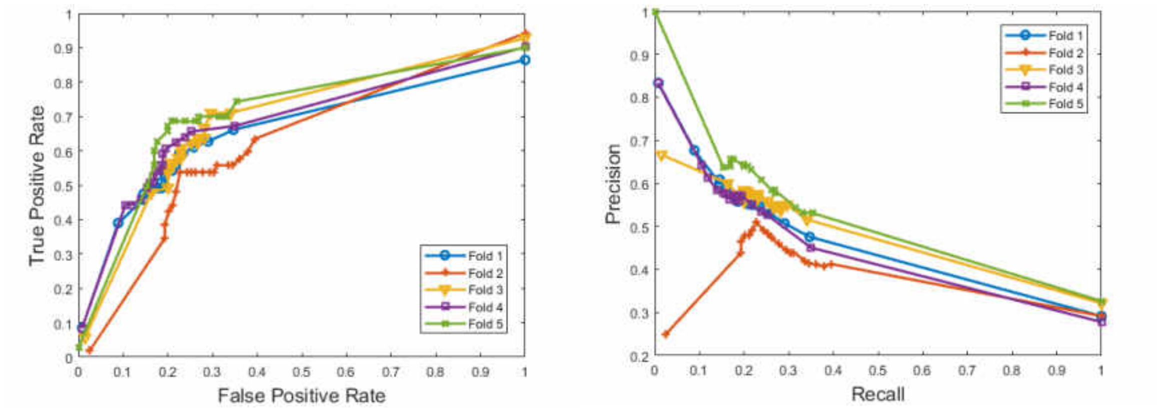


Figure 23: Failures in crack classification with confidence level object is cracked (a) edge case classified as cracked (1.0 confidence) whereas human identified otherwise (b) difficult to see crack classified as not cracked (0.0097) (c) difficult to see crack classified as not cracked (0.04055) (d) blurred photo misclassified as cracked (0.85661)

A few sample failures of the crack classification CNN can be found in Figure 23. The main sources of failures were found in edge cases where the crack was hardly visible, blurred images, and minor cracks that are hard to see at the reduced resolution.

With both the DEBC weld detection and crack classification CNNs developed, the two networks were combined and tested. To accomplish this, the bounding box detection from the DEBC weld detection CNN was cropped from the original image, then the resulting cropped images were provided to the crack classification CNN. Both the cropped image, and crack classifiction were saved. The data the combined system was tested on included the original, unaltered, non simulated images contained in the test set for each of the five folds the crack classification was tested with. The use of these images was due to them being a difficult dataset to test with, providing many different view angles, and the only remaining images that the combined dataset had not been trained with. Results for the combined network were calculated similarly to the five-fold cross validation performed on the crack classification CNN due to the configuration of the test data and trained crack classificaiton networks.

The results for the combined system were determined and analyzed through use of ROC and precision recall curves for each of the five folds of test data. The curves were calculated by comparing the annotated data as a cracked DEBC weld image or not, and the result of the combined CNN networks. The resulting output as the cropped images of DEBC weld detections, and text file of crack classificaitons for each detection were manually determined to account for false positive weld detections. Due to using only the original non augmented images, and not including any images from the provided simulated images, an average of 185 test images per fold were tested. The resulting ROC and Precision Recall curves can be found in Figure 24.



(a) (b)

Figure 24: Curves of the combined DEBC weld detection and five different crack classification CNNs as tested on the crack classification test data (a) ROC curve of the five different combined networks (b) Precision-Recall curve for the same five networks in (a)

Both ROC and Precision-Recall curves were developed as before. The DEBC weld detection CNN threshold was set to 0.85 as found previously, while varying the crack classification threshold to determine the ideal value. The thresholds were set for three levels of severity as Red, Yellow, and Green. Images classified as Red were considered likely cracked and maintanence of the component due to cracks should be considered. Images falling in the Yellow classification were considered questionable, and a human operator should view the component to determine component status. Images classified Green were likely in good condition and not in need of maintanence. The threshold for the Red classification was found starting at an upper bound of 1.0 and using the largest average $F_{0.5}$-measure, which weights the precision higher, as 0.5127 corresponding to a threshold of 0.95 for the lower bound. At the threshold of 0.95, the average accuracy and precision was found as 72.21% and 59.89%. The F_2-measure was used to find the lower bound of the Yellow classification with the highest F_2 score as $F_2 = 0.6198$ corresponding to the threshold of 0.1. The average accuracy and precision at this threshold was 65.66% and 47.70%. Green was set for all values lower than the lower bound of Yellow.

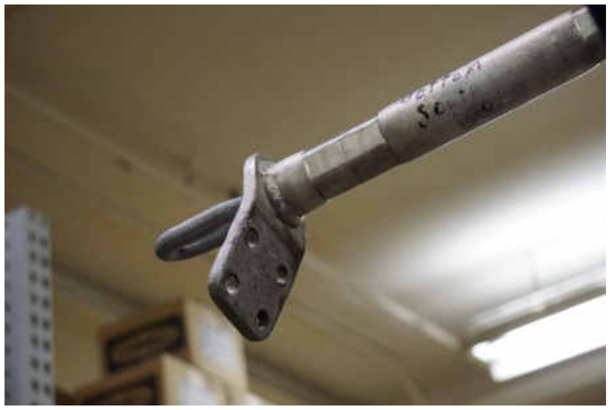
Table 6: Confusion matrix for combined DEBC weld detection and crack classification labeled as Red only.

	Predicted Negative	Predicted Positive	Total
Actual Negative	TN: 113.6	FP: 18.2	131.8
Actual Positive	FN: 31.8	TP: 27.2	59
Total	145.4	45.4	190.8

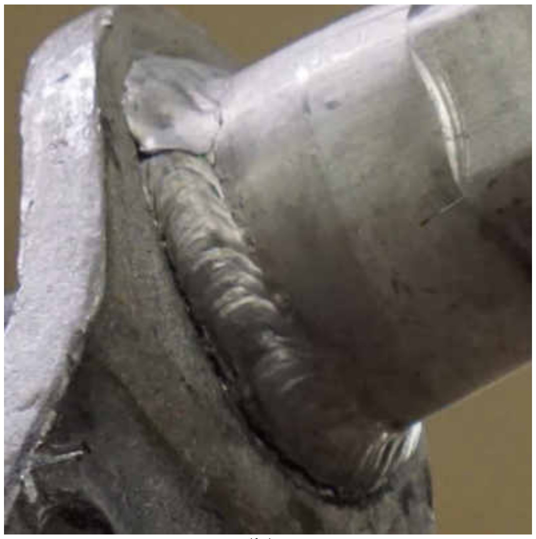
Table 7: Confusion matrix for combined DEBC weld detection and crack classification labeled as Red and Yellow.

	Predicted Negative	Predicted Positive	Total
Actual Negative	TN: 90.2	FP: 40.4	130.6
Actual Positive	FN: 18.6	TP: 41.6	60.2
Total	108.8	82	190.8

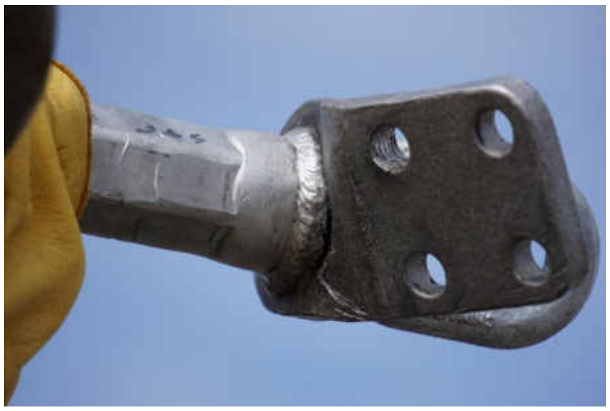
With the above threshold values, the resulting confusion matrix averages for the five networks tested was found and shown in Table 6 and Table 7. Table 6 provides the threshold for the Red classification, while Table 7 provides the images classified as either Red or Yellow. The Red classification provided propotionaly less false positives, while the combined Red and Yellow detected a higher portion of the cracks, but also contained more false positives. This data set provided a larger sample of negative images, which the combined CNNs were able to accurately determine many of the non-cracked welds. Many of the false positives cracks were detected from false DEBC weld detection false positives. Sample images of accurately detected and classified cracked welds can be found in Figure 25.



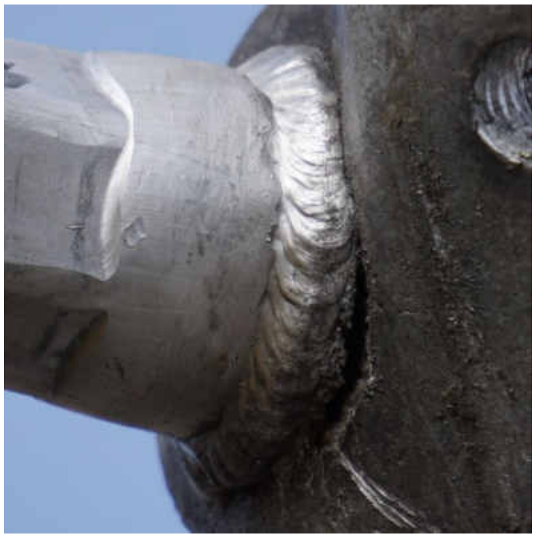
(a)



(b)



(c)



(d)

Figure 25: Success in DEBC weld detection and crack classification (a) Original image (b) Cropped DEBC weld detection of (a) successfully detected and classified as cracked at 0.99132 confidence (c) Original image (d) Cropped DEBC weld detection of (a) successfully detected and classified as cracked at 0.99999 confidence

As can be seen from Figure 24, Table 6, and Table 7 previously, the combined networks performed worse than seperately. This was largely due to both false negatives and false positives due to the DEBC weld detection CNN. On this new test data, the

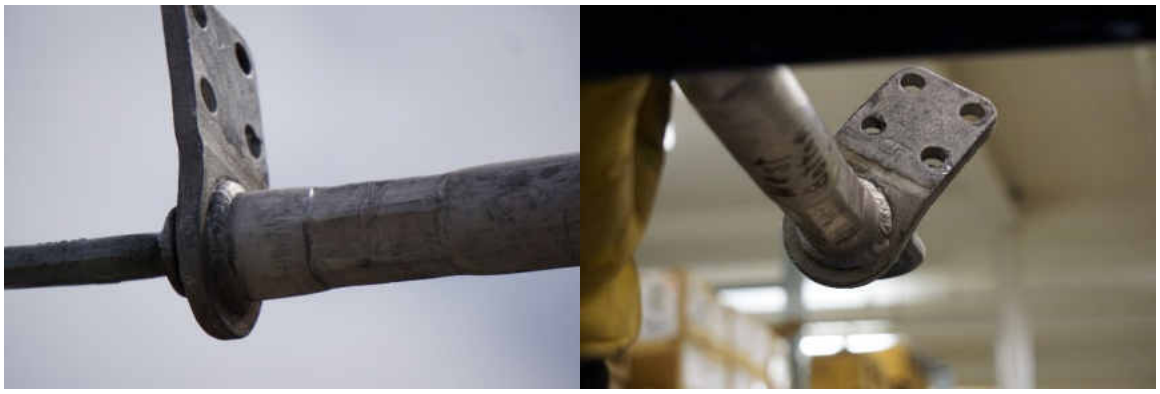


Figure 26: Example images in which the DEBC weld detector failed to detect the weld.

DEBC weld detection CNN struggled to identify a number of DEBC welds, while providing several false positives. This was attributed to the newer data coming from a different distribution of data from what the DEBC detection CNN was trained with. False negatives typically consisted of view angles to the weld in which the DEBC weld detection was not trained for. The main source of these errors occurred when the DEBCs were viewed upside down as shown in Figure 26. During practical inspections, the weld would not be viewed from these angles. The false positive DEBC weld detections were mostly found as artifacts due to the data collection process. Due to the images being collected by someone manually holding the DEBC above the camera, many images contained artifacts such as gloves, sleeves, and other typically round objects the DEBC weld detector mistook as DEBC welds. These artifacts seen in the data collection would typically not be found in standard inspection images. As both of these cases would not occur in the typical inspection process, the actual practical application may achieve higher results. Examples of the DEBC weld detector providing false positives due to such artifacts can be found in Figure 27.

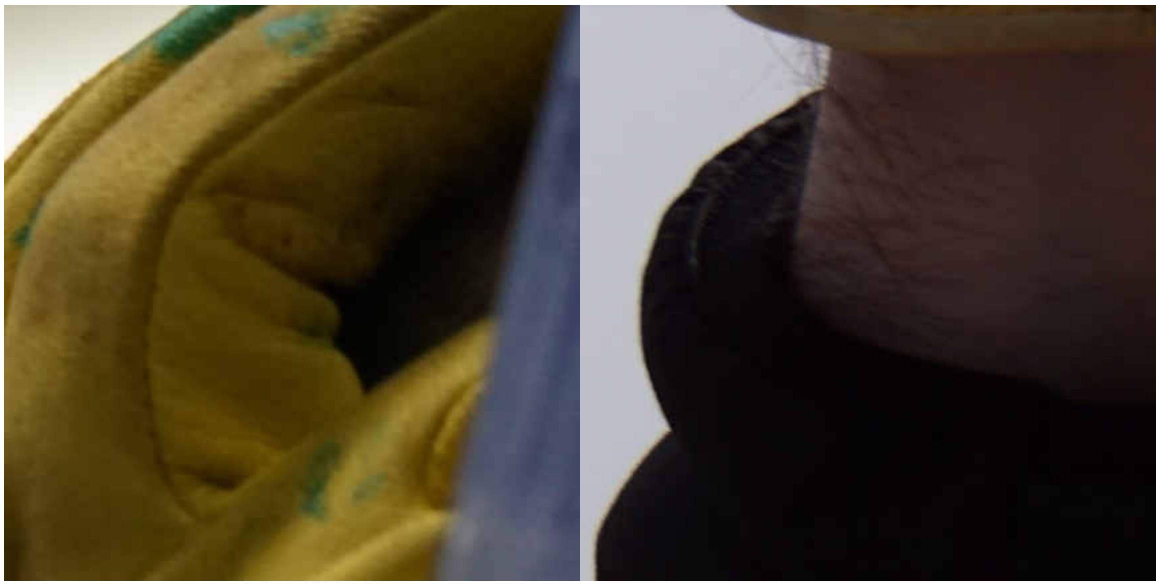
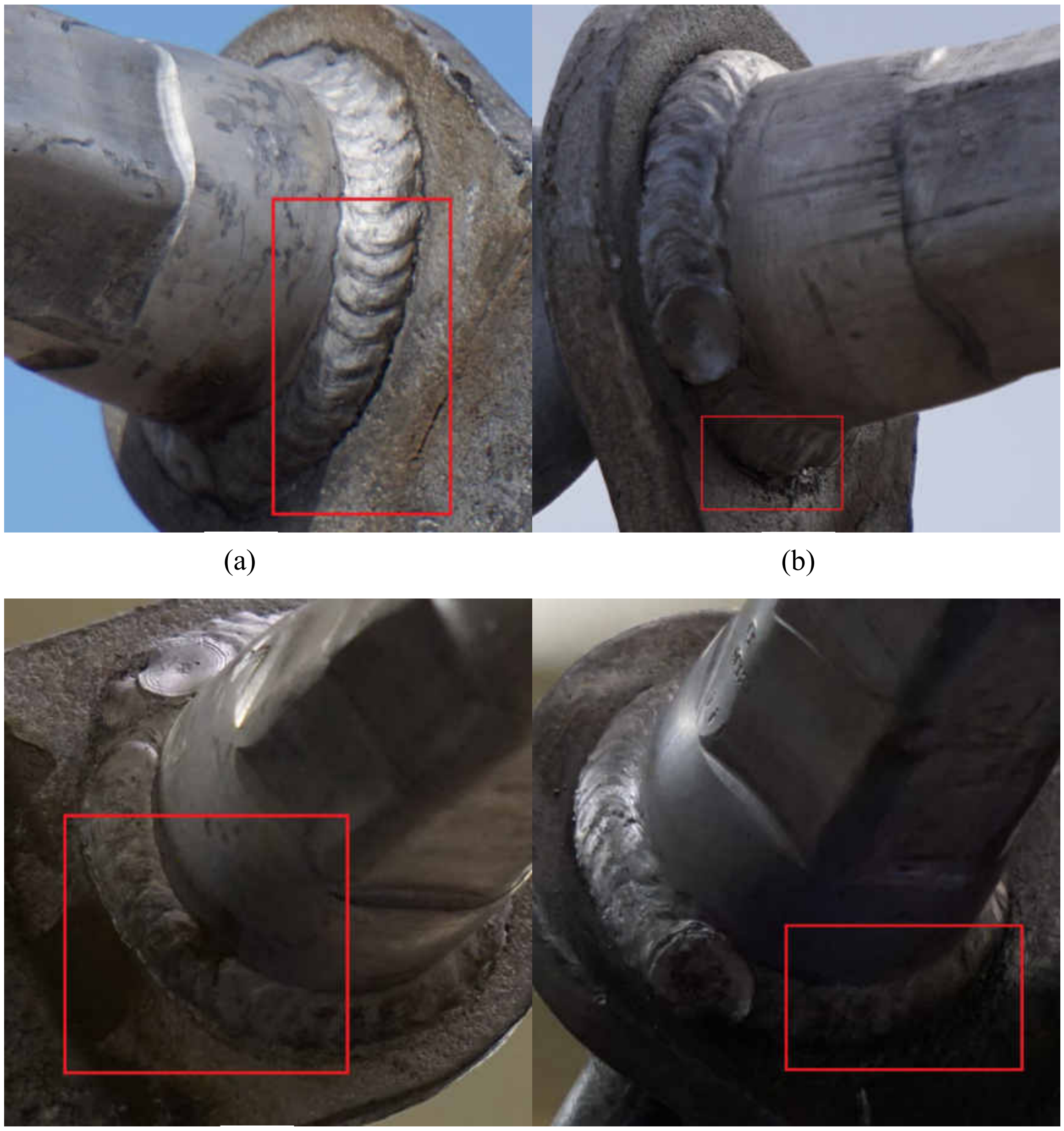


Figure 27: Example images of false positive DEBC weld detections cause by a glove to the left and the sleeve of a jacket to the right.

Of the DEBC welds that were accurately detected, the crack classification CNN also had some failures. Most failures were, as previously, difficult edge cases and minor difficult to see cracks. The classifier did appear to struggle to identify cracks more than previously tested, possibly due to being more translationly variant as the training images were centered about the weld, more so than the images provided through the DEBC weld detection CNN. Examples of failed crack classifications can be found in Figure 28.



(a) (b)

(c) (d)

Figure 28: Example images of failed crack classification on accurate DEBC weld detections (a) Slight crack difficult to see classified as crack 0.00003 confidence (b) Poor view of larger crack classified as crack at 0.00042 confidence (c) Minor difficult to see crack at 0.01128 confidence (d) Large crack difficult to see classified as crack 0.08601

CHAPTER V

DISCUSSION

In this work, deep convolutional neural networks were applied and evaluated based their capability of the detection of DEBC's from dead end high tension power lines and to classify the components condition as either cracked, or in good condition. Two different CNNs were trained and utilized, one to detect the DEBC weld from images and crop that region, the second was used to classify the resulting cropped image as either in good condition or if they contained a partial failure due to a toe-weld crack. The two CNNs developed were first evaluated separately. Evaluation of the DEBC weld detector was performed on 111 UAS test inspection images. Due to the very limited number of cracks found in live high-tension power lines, a few previously replaced DEBCs were manually imaged and a 5-fold cross validation strategy was performed to both train and evaluate the crack classification CNN. Once the two CNNs were combined, the same test set used in the 5-fold cross validation for the five developed crack classifiers was utilized as that was the only remaining data not used in training of either CNN.

The DEBC weld detection CNN was developed by using the Faster R-CNN algorithm and implemented with the VGG16 model. The CNN was trained using a total of 416 original images from provided simulated images and UAS test inspection images that were augmented to a total of 2,707. Training was accomplished by fine-tuning the CNNs following the four-stage alternating optimization method as per [46] using an ImageNet pretrained model. After training multiple CNNs for this task while varying different parameters, such as the amount of training iterations, learning rates, and number of object classes, and threshold values for detection, it was determined the ideal amount of training for this dataset and CNN model was at 100,000, 80,000, 100,000, and 80,000 for the four stages of training and included three different object classes as a DEBC weld, insulators, and a general background class. The ideal threshold for detection was found as 0.85. The CNNs were tested on 111 images of UAS test inspection images and achieved an accuracy of 97.8% and precision of 99.1%. This CNN was used to detect possible DEBC welds found in an image and crop the relevant portion out for further inspections by the second CNN to classify the components condition.

The cropped DEBC weld image was then saved for the user to review and provided to the crack classification CNN. The crack classifier utilized the same VGG16 model and was trained end-to-end. Due to a lack of cracked DEBC welds found during the inspections process, several images of a select few previously replaced DEBCs containing cracks were manually imaged. A total of 1,095 images were created by cropping the weld portion from each image, and slight data augmentations generated a total of 4,632 training images. Both training and testing were performed using a 5-fold cross validation strategy. Training was performed for 30,000 iterations while testing the validation data every 1,000 iterations. It was found that 16,000 iterations were the ideal amount of training for this dataset and network model. Averaging the results of the 5-fold cross validation resulted in an accuracy of 97.55%.

The two networks were then combined and tested. Due to using all the cracked

DEBC images during training, the five CNNs developed for the 5-fold cross validation were tested using the same orignal images. First, the DEBC weld detection CNN was utilized to crop all detected welds found in the crack classification data and the images were separated by the validation data for each of the five folds utilized for the crack classification network. For each of the corresponding validation data, the resulting cropped image was classified and saved to a text file. Thresholds for the liklyhood of a crack were develop as Red, Yellow, and Green where Red was most likely cracked, Green most likely not cracked, and Yellow questionably cracked. Once evaluated, the combined CNNs provided an average of 73.79% accuracy and 59.92% precision in detecting cracks via the Red classification. Combining both Red and Yellow classifications provided accuracies of 69.08% and precision of 50.73%. Failures were largely due to the difficult views the images provided in this dataset as many of the welds were in orientations that would not occur in practice, as well as minor difficult to see cracks.

This algorithm using the combined CNNs trained to detect and classify the DEBC of high tension powerlines provides among the first applications of CNNs to the task of maintenance inspections in power line infrastructure. The proposed algorithm provides a faster way for inspection engineers to quickly supply images to classify and quickly receive compenent failure classifications due to toe-weld cracking. The intent of this work was to reduce time and cost for inspections of high tension power lines by working towards automation of the DEBC weld.

Conclusions and Future Work

The goal of this work was to develop an algorithm capable of detecting the weld portion of a DEBC from UAS inspections imagery of high voltage power lines at an accuaracy greater than 90%, and then identifying if that detected component contained a partial failure due to toe-weld cracking at a greater than 80% accuracy. Combined the system was to have an overall accuracy of greater than 72%. The intended purpose of such an algorithm was to aid inspections engineers reviewing the large amount of data a UAS may produce during maintanence inspections by providing a step towards automating the expensive process of performing maintanence inspections.

The developed algorithm used two different CNNs to accomplish the goals listed above. The first CNN utilized the Faster R-CNN architecture with the VGG16 backbone and was trained to locate and identify the location of the DEBC weld in a given inspections image. A new image containing the DEBC weld was created from a cropped section of the detected component weld region found from the first CNN. The second CNN received the new image of the cropped DEBC weld portion and used the VGG16 architecture to classify the likelihood the component was cracked as three different levels listing priority as Red, Yellow, or Green. The Red classification was considered as most likely cracked and should be considered for maintanence, the Yellow classification was considered as questionable and should be further reviewed for possible maintanence, and a Green classification was considered as most likely in good condition.

The accuracy and precision of the algorithm was found for both CNNs separately, as well as combined. The accuracy and precision of the first CNN as the DEBC detection CNN in identifying the weld portion of DEBCs was determined as 97.8% and of 99.1% respectively when tested on 111 UAS test inspection flight images. The accuracy of the crack classification was determined as 97.55% found through a 5-fold cross validation strategy of the second CNN. Combining the two CNNs provided an accuracy of 73.79% and a precision of 59.92% when testing on the same difficult 5-fold cross validation images. These results match or exceed the original goals set forth to detect both the component and failure classifications.

Future improvements to the network are suggested. CNNs are currently an active and heavily researched topic with improvements to the architecture consistently provided. Utilizing newer models such as [21] could increase not only accuracy of the detections, but also processing speed. Per [16], removing difficult training images may increase the effectiveness of the algorithm. The training data from the simulated imagery study provided several images where the DEBC was difficult to see due to situations such as the sun in the direct background. The data generated for the crack classification also contained many poor view angles to the cracks, as well as some very small and difficult to see cracks. With the evidence shown in this work that inclusion of image classes that provide large number of false positives, adding additional classes and training images may help to reduce false positives further. This could also increase the utility of the proposed algorithm as it could search for several other components or maintenance concerns.

CHAPTER VI

APPENDICES

USER MANUAL

DEBC Crack Detection Software

University of North Dakota

August 2017

USER'S MANUAL

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

GENERAL INFORMATION

1.1 System Overview

This software is an aid to automatically detect welds of dead-end body component (DEBC) images and classify if they contain a partial failure due to cracking. The software provides the following:

- A software system compatible with the Windows 10 platform.
- Easy to use graphical user interface.
- Saves automatically cropped detections of the DEBC weld for your review and records.
- Records classification of potential partial failures of the cropped DEBC weld due to cracking.
- Saves three levels of severity of possible cracked partial failures in the DEBC weld as red, yellow, and green, each in a separate text file.
  - Red: Likely a cracked component and should be reviewed and considered for maintenance.
  - Yellow: Potentially cracked, requires human expertise to ensure if component is in good condition or not.
  - Green: Likely in good condition and not in need of maintenance.
- System name or title: DEBC Crack Detection
- System category:
  - *Application:* performs clearly defined functions for which there is a readily identifiable consideration and need
- Operational status:
  - Operational

1.2 Project References

References that were used in preparation of this document in order of importance to the end user.

https://chrome.google.com/webstore/detail/chrome-remote-desktop/gbchcmhmhahfdphkhkmpfmihenigjmpp?hl=en

1.3 Authorized Use Permission

1.4 Points of Contact

Below is a list of Points of Contacts relevant to this project:

Contact Name	Contact Type	Department	Telephone Number	Email	Oversight Function
Ian Nordeng	Student Developer	Mechanical Engineering	(920) 427-2795	Ian.nordeng@und.edu	Project Developer
Jeremiah Neubert	Associate Professor	Mechanical Engineering	(701) 777-2107	Jeremiah.neubert@engr.und.edu	Advisor

1.4.1 Information

The points of organizational contact (POCs) that may be needed by the document user for informational and troubleshooting purposes are currently not available.

1.4.2 Coordination

The list of organizations that require coordination between the project and its specific support function (e.g., installation coordination, security, etc.) are currently not available.

1.4.3 Help Desk

Help desk information including responsible personnel phone numbers for emergency assistance is currently not available.

1.5 Organization of the Manual

User’s Manual v0.01.

1.6 Acronyms and Abbreviations

Acronyms and abbreviations used in this document and the meaning of each.

App:	Application
MS:	Microsoft

DEBC: dead-end body component
CNN: Convolutional Neural Network
GUI: Graphical user interface

2.0 SYSTEM SUMMARY

SYSTEM SUMMARY

This software is intended for use with inspection images of a high-voltage powerline component, the DEBC, and is to be used to aid in the automatic classification of potential partial failures due to cracks in the weld. The system utilizes two different CNNs. The first CNN is used to detect possible DEBC welds in the image and crop them out for classification and saves them in a designated folder. The second CNN classifies if the cropped component detected previously contains a crack or partial failure. These classifications are designated by severity through either red, yellow, or green where red likely contains a crack, yellow is a possible crack, and green is likely in good condition. A simple graphical user interface is provided to allow ease of use to provide images to classify, designate where to save the output, and easily use the developed CNNs.

2.1 System Configuration

The GUI for the automated DEBC crack classification project is based upon Python 2.7. The user interface is built using the Tkinter libraries. These libraries were tied in with the two different CNNs developed to both detect DEBC welds, and to classify if the welds contain cracks or not.

2.2 Data Flows

Users interact with the system through a clickable interface. This interface is used for the user to provide the necessary information to the algorithm, while also providing feedback to the user as to the status of the algorithm.

3.0 GETTING STARTED

GETTING STARTED

This software provides a basic GUI consisting of a title screen, file choosers, and progress bars. The title screen provides the ability to choose where the images you wish to classify reside on your hard drive, as well as where you wish to store the output of the CNNs. Supported image formats include jpg and png files. Once these are provided, the two CNNs developed can be initiated, and the images are cropped as DEBC welds are detected, and then classified by severity of the DEBC condition due to cracking. Progress of the classification is provided through a progress bar to alert you how long the process may take. A help button is also provided to give basic instructions to use the software. An exit button is used to exit the software.

3.1 Requirements

Hardware:

- Video Card (Tested with Nvidia Titan X and Nvidia GeForce GTX 1080Ti)
    - Minimum 4GB of memory.
    - CUDA support
- Hard Drive: 4 GB

Software:

- Windows (Tested on Windows 10)
    - Microsoft Visual Studio 2015 with C++ programming language
    - CUDA 8.0
    - Python 2 with Microsoft C++ compiler package
    - OpenCV
    - Cmake
    - Git

3.2 Installation

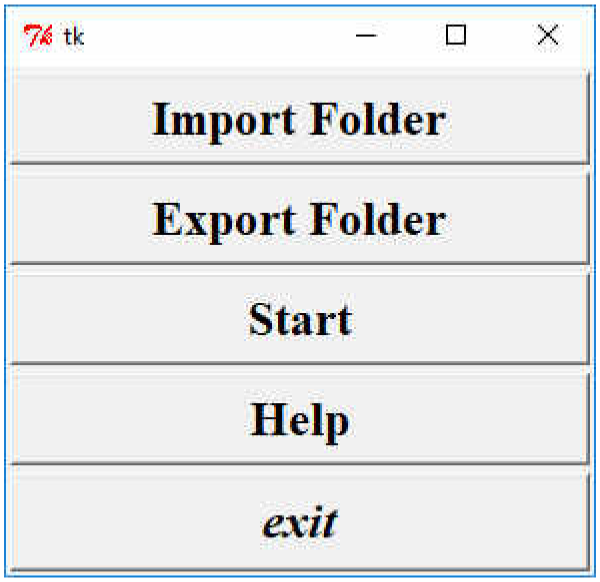
It is assumed you have properly installed the dependent software listed above and ensured the cv2.py located in the opencv\build\python\2.7\x64 is placed in the Python27\Lib\site-packages folder.

To install the DEBCCrackNet, you must run the build_win.cmd contained in the py-faster-rcnn/caffe-fast-rcnn/scripts folder in the command prompt. To accomplish this:

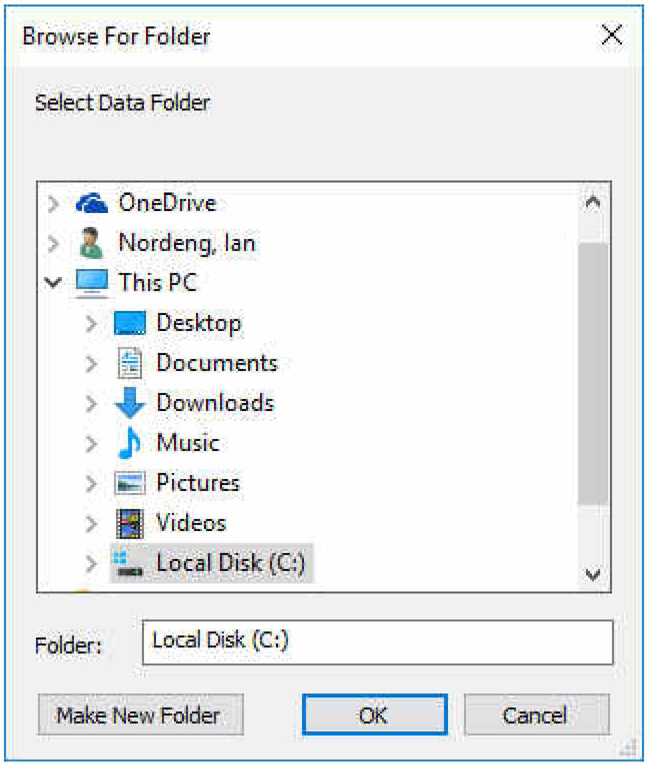
1. Start the windows command prompt and navigate to the folder py-faster-rcnn/caffe-fast-rcnn/scripts/ which contains the build_win.cmd script.
2. Type build_win.cmd and the installation process should commence. This will install the required CNN software, along with many other dependent software libraries packaged with this software.

3.3 System Menu

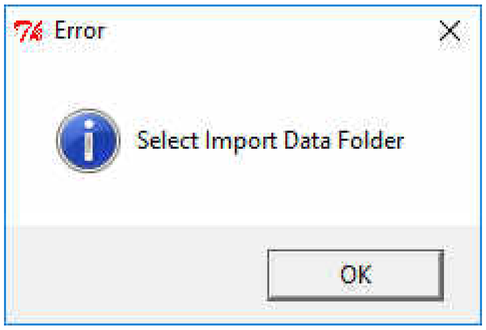
To run the software, you must run the "DEBC_crack_detection" file contained in the py-faster-rcnn folder. Once started the main menu is shown as:



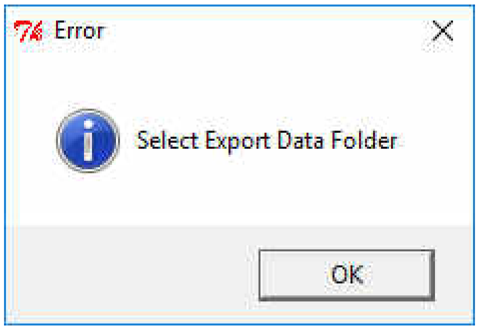
The top button of the main menu, represented as "Import Folder", is to provide the path where the images you intend to classify exist on the computer's hard drive. Clicking on the "Import Folder" button brings up a new window to select the desired folder in which the intended image to classify resides as shown as:



The "Export Folder" button performs a similar action but is instead used to select the folder you wish to store the cropped images of the DEBC weld as detected by the system, and the resulting classification text files of crack severity of said welds. Both the "Import Folder", and "Export Folder" options must be selected before continuing with the "Start" button. If either import or export folders are not selected, an error message will prompt you to select the desired missing information. This is shown as either:

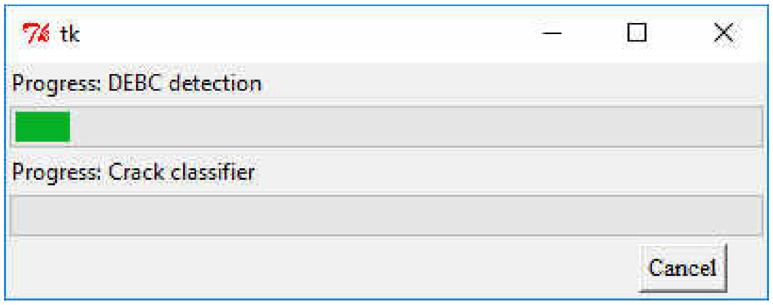

Error

Select Import Data Folder

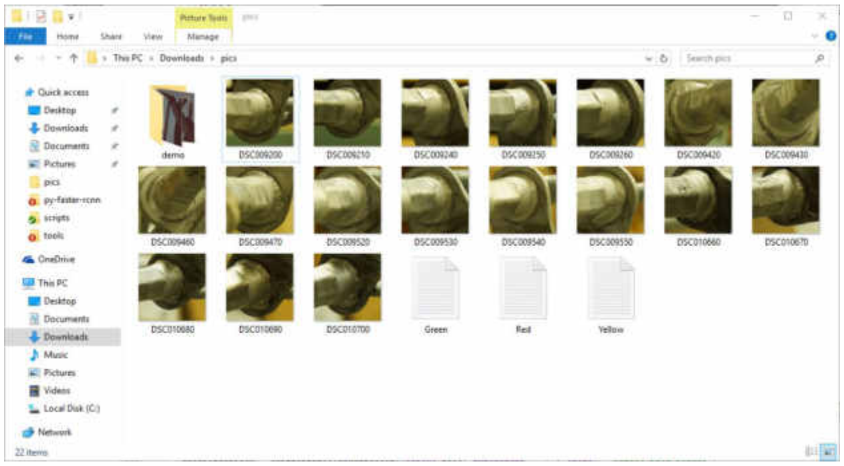

Error

Select Export Data Folder

Once both an “Import Folder” and “Export Folder” are selected, you may click on the “Start” button. The “Start” button starts by initializing the DEBC detection CNN. Once initialized a set of two progress bars will appear and are displayed in the window:

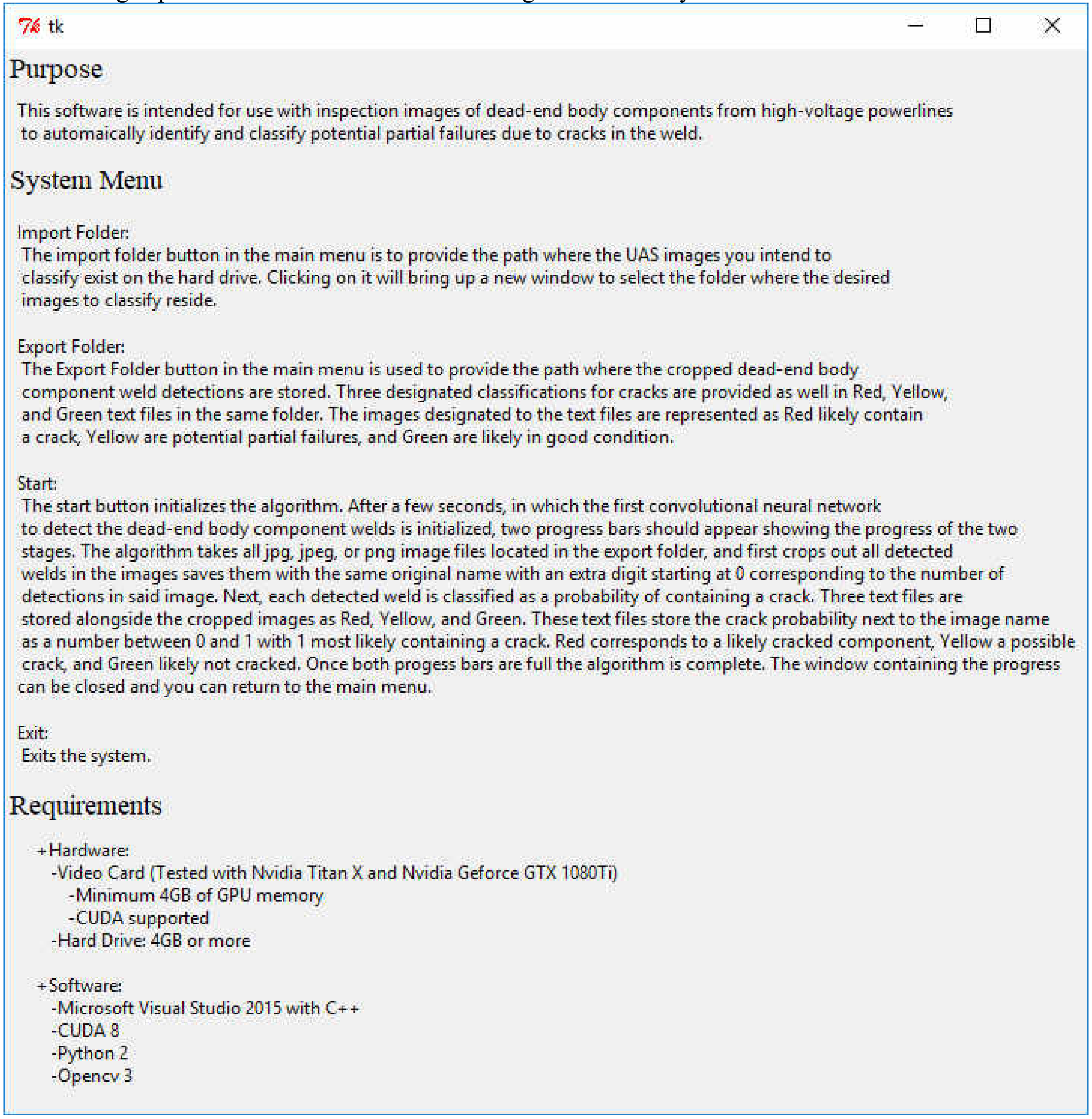


The top progress bar represents the amount of the total images contained in the input folder which have been considered by the DEBC detection network. The program creates a new image for any detected DEBC weld as a cropped segment of the original image containing only the weld. The cropped images are saved in the folder provided as the export folder and are named after the original image with an appended “_” followed by a digit. For example, with an original image titled DSC00015.jpg, a new cropped detection would be saved as DSC00015_0.jpg in the designated export folder. The digit represents the number of detected DEBC welds contained in the original image starting from 0. The process can be stopped at stopped at any time to return to the main menu by clicking the “Cancel” button. Supported image formats include jpg and png files. Once the DEBC detection is complete for all images in the input folder, the crack classification CNN is initialized and all images in the export folder are considered. This CNN reads all detected welds from the previous CNN, and outputs three different text files as Red, Yellow, and Green. A sample of the output of both the saved cropped detected images, as well as the text files from the detections is shown as:



Each text file stores the name of each image with the corresponding crack classification next to it as a number from zero to one with one being the highest probability of the cropped image containing a crack. The three designations of Red, Yellow, and Green represent the severity of a potential crack where Red represents a high likelihood the component contains a crack, Yellow may contain a crack, and Green likely in good condition. These color designations are split by threshold values in which Red thresholds are set by default as greater than or equal to 0.9, Yellow less than 0.9 and greater than or equal to 0.15, and Green anything less than 0.15. These thresholds are set from the config_thresh.txt file located in the py-faster-rcnn folder and can be altered as you see fit.

A "Help" button is also provided to aid you in how to use the system as well as what is required. The help button brings up a new window with text describing much of the system and shown as:



Purpose

This software is intended for use with inspection images of dead-end body components from high-voltage powerlines
to automaically identify and classify potential partial failures due to cracks in the weld.

System Menu

Import Folder:
The import folder button in the main menu is to provide the path where the UAS images you intend to
classify exist on the hard drive. Clicking on it will bring up a new window to select the folder where the desired
images to classify reside.

Export Folder:
The Export Folder button in the main menu is used to provide the path where the cropped dead-end body
component weld detections are stored. Three designated classifications for cracks are provided as well in Red, Yellow,
and Green text files in the same folder. The images designated to the text files are represented as Red likely contain
a crack, Yellow are potential partial failures, and Green are likely in good condition.

Start:
The start button initializes the algorithm. After a few seconds, in which the first convolutional neural network
to detect the dead-end body component welds is initialized, two progress bars should appear showing the progress of the two
stages. The algorithm takes all jpg, jpeg, or png image files located in the export folder, and first crops out all detected
welds in the images saves them with the same original name with an extra digit starting at 0 corresponding to the number of
detections in said image. Next, each detected weld is classified as a probability of containing a crack. Three text files are
stored alongside the cropped images as Red, Yellow, and Green. These text files store the crack probability next to the image name
as a number between 0 and 1 with 1 most likely containing a crack. Red corresponds to a likely cracked component, Yellow a possible
crack, and Green likely not cracked. Once both progess bars are full the algorithm is complete. The window containing the progress
can be closed and you can return to the main menu.

Exit:
Exits the system.

Requirements

+Hardware:
  -Video Card (Tested with Nvidia Titan X and Nvidia Geforce GTX 1080Ti)
    -Minimum 4GB of GPU memory
    -CUDA supported
  -Hard Drive: 4GB or more

+Software:
  -Microsoft Visual Studio 2015 with C++
  -CUDA 8
  -Python 2
  -Opencv 3

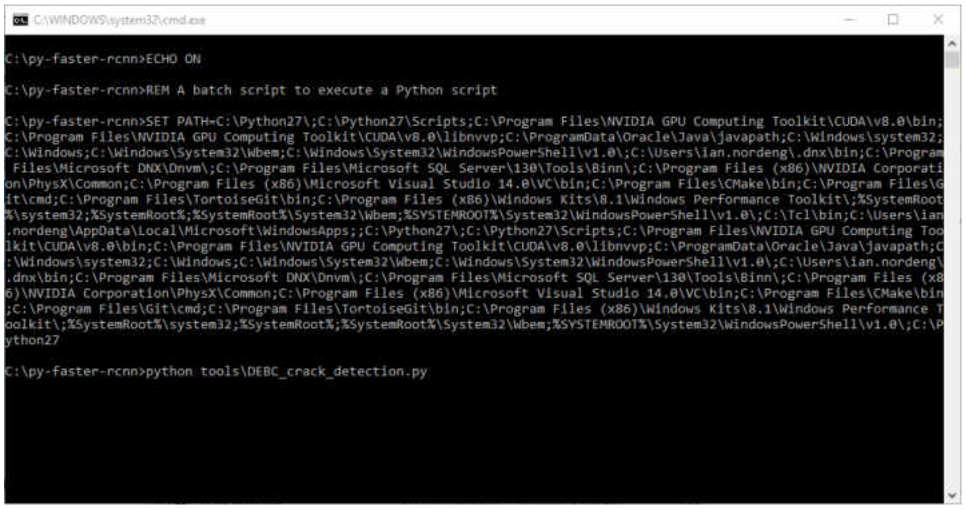
3.4 Exit System

To exit the system, click the exit button on the bottom of the main menu.

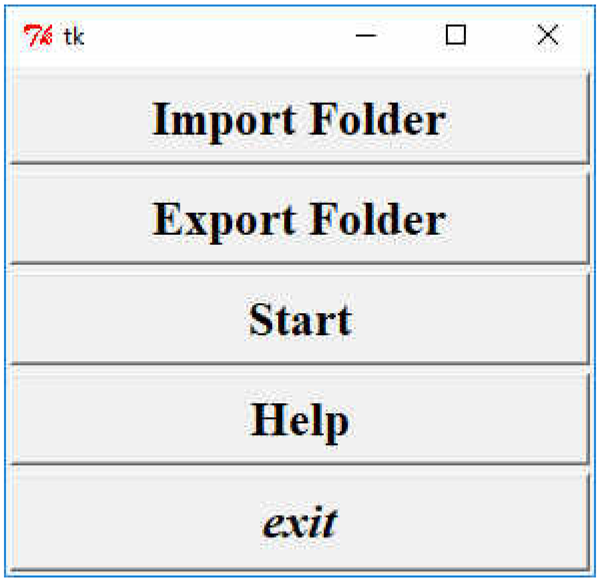
3.5 Demo Example

To fully introduce the intended process for classifying potential DEBC weld cracks, a demo has been created as a guide for the system. The following provides step-by-step instructions to show you how to use the software to analyze the demo images.

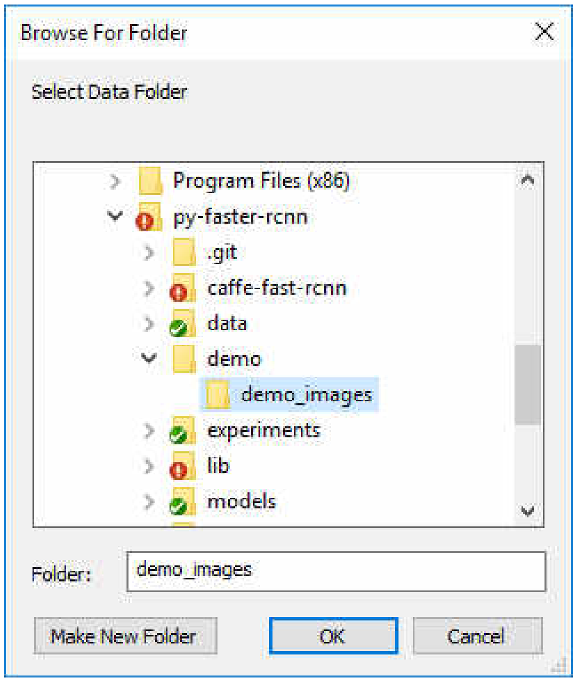
Step 1:
After completion of the installation process, navigate to the main folder titled py-faster-rcnn where it was installed. Inside this folder you will find a file titled “DEBC_crack_detection”. Double click this file. This will bring up a command prompt that will run in the background, as well as the main menu.



```
C:\WINDOWS\system32\cmd.exe

C:\py-faster-rcnn>ECHO ON

C:\py-faster-rcnn>REM A batch script to execute a Python script

C:\py-faster-rcnn>SET PATH=C:\Python27\;C:\Python27\Scripts;C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\bin;
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\libnvvp;C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;
C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\ian.nordeng\.dnx\bin;C:\Program
 Files\Microsoft DNX\Dnvm\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\NVIDIA Corporati
on\PhysX\Common;C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin;C:\Program Files\CMake\bin;C:\Program Files\G
it\cmd;C:\Program Files\TortoiseGit\bin;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\;%SystemRoot
%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Tcl\bin;C:\Users\ian
.nordeng\AppData\Local\Microsoft\WindowsApps;;C:\Python27\;C:\Python27\Scripts;C:\Program Files\NVIDIA GPU Computing Too
lkit\CUDA\v8.0\bin;C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\libnvvp;C:\ProgramData\Oracle\Java\javapath;C
:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\ian.nordeng\
.dnx\bin;C:\Program Files\Microsoft DNX\Dnvm\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x8
6)\NVIDIA Corporation\PhysX\Common;C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin;C:\Program Files\CMake\bin
;C:\Program Files\Git\cmd;C:\Program Files\TortoiseGit\bin;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance T
oolkit\;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\P
ython27

C:\py-faster-rcnn>python tools\DEBC_crack_detection.py
```

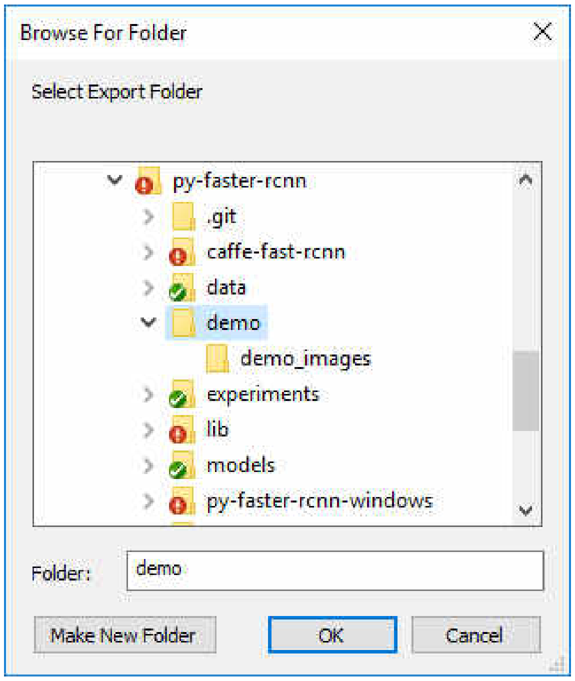


Step 2:
In the main menu click on the “Import Folder” button on the top of the menu. This will provide a file dialog choosing window. Navigate inside this window to the demo folder contained in the py-faster-rcnn folder and select the demo_images folder and click “OK”.



This folder contains three sample demo images which we will be classifying. The file dialog chooser will disappear after clicking “OK”.

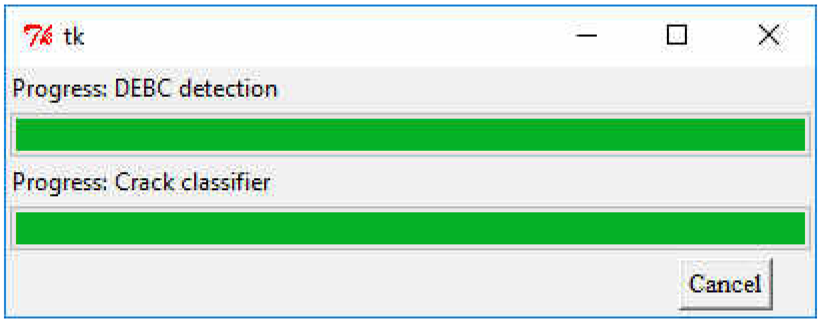
Step 3:
Click on the “Export Folder” button 2nd from the top of the main menu. This will open another file dialog



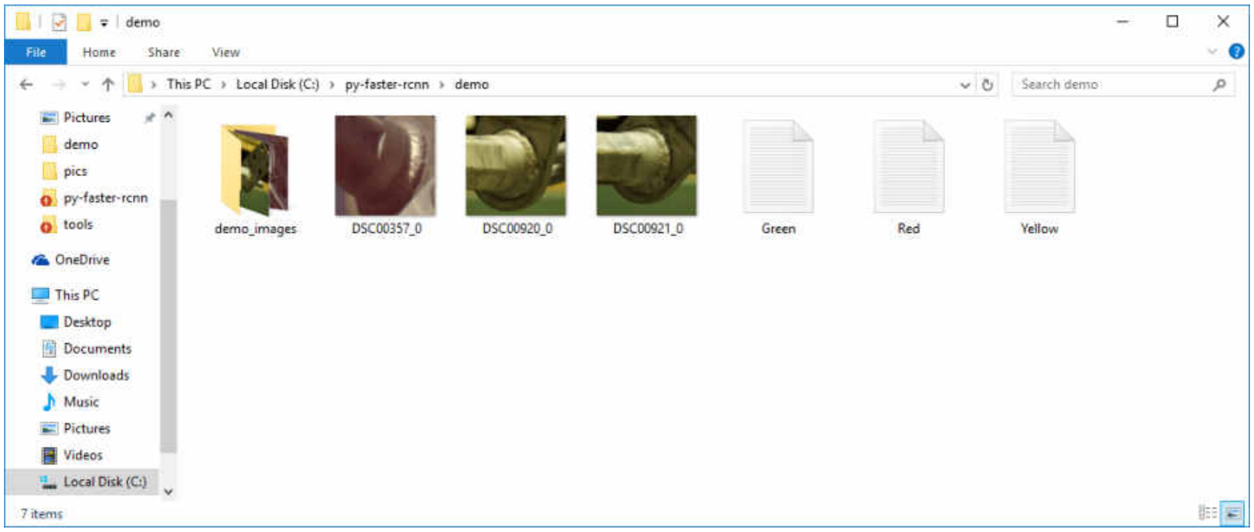
chooser. Inside the new window, navigate to the same demo folder contained in the py-faster-rcnn folder and select the demo folder, then click "OK".

This will select the demo folder as the output where the cropped DEBC detected images will be stored, as well as the classifications of potential cracks in three separate text files.

Step 4:
With both an import and export folder created, you may then click on the "Start" button, which is located third from the top in the main menu. You will see the background command prompt provide information on loading the CNN used, and a window containing two progress bars will appear.



The top progress bar will fill first representing the number of images considered for detection of the DEBC weld from the folder selected from Step 2 above or the import folder (demo_images in this case). Any detected DEBC weld will be cropped and saved to the export folder selected from Step 3 above, the demo folder in this instance. Once the top progress bar is filled the next stage of classifying the crack severity is performed. Next the crack classification will generate three text files in the demo folder, also selected in the "Export Folder" section, as Red, Yellow, and Green. All cropped images now in the demo folder will be considered, and the confidence level of a crack will be stored in one of the three text files based on the risk of a crack detected with Red being a likely crack, Yellow a potential crack, and Green likely in good condition. This step is completed when the second progress bar is full.



Step 5:
Classification of the demo images is now complete. To exit the progress bar window, you may click on either the "Cancel" button in the bottom right, or the "X" in the top right to close the progress bar window.

Step 6:

You may now exit the program by either Clicking on the “exit” button on the bottom of the main window, or the “X” in the top right of the main window. The main window will then exit, and you may press any key on the remaining command prompt to fully exit the system.

4.0 USING THE SYSTEM (ONLINE)

USING the SYSTEM (Online)

For remote use, the system is compatible with Google's remote desktop application. Google remote desktop is an extension to Google Chrome which allows you to control one computer from another remotely over the internet. With images being provided to the computer running this software obtaining images, you may use this remote desktop extension to provide a portable system that can be implemented in the field.

CHAPTER VII

REFERENCES

[1] McNichol, E.: It’s time for states to invest in infrastructure, 2016. Available: http://www.cbpp.org/research/state-budget-and-tax/its-time-for-states-to-invest-in-infrastructure.

[2] Monti, M., Rose, S., Mullins, K., Wilson, E.: Planning and CapX2020 Building trust to build regional transmission systems. For CapX2020, 2016.

[3] North Dakota Transmission Authority Annual Report. NDTA, 2015.

[4] Drones to Cut Inspection Costs in Half, Improve Distribution Reliability. In T&D World Jan. 23, 2015.

[5] PLP Compression Dead-end & Jumper Terminal for ACSR & ACSS Conductors. Preformed Line Products Company, April 2016.

[6] Ma, L., Chen, Y.: Aerial Surveillance System for Overhead Power Line Inspection. In CSOIS, 2004.

[7] Cawley, J., Gerald, H.: Occupational electrical injuries in the United States, 1992-1998, and recommendations for safety research. Journal of safety research, 34(3), pp. 241-248, 2003.

[8] Olsen, D., Dvorak, D., Lemler, K., Dunlevey, M., Vinger, C., Nordeng, I., Neubert, J.: Powerline inspections with upward looking sensor using UAS. In AUVSI XPONENTIAL, 2017.

[9] Katrasnik, J., Pernus, F. and Likar, B.: A survey of mobile robots for distribution power line inspection. IEEE Transactions on Power Delivery, 25(1), pp. 485-493, 2010.

[10] Sharma, A.: Drone Data Adds a New Horizon for Big Data Analytics. In InfoQ, 2014. Available: http://www.infoq.com/news/2014/09/drone-data-big-data-analytics.

[11] Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[12] Schmidhuber, J.: Deep Learning in Neural Networks: An Overview. In arXiv, 2014.

[13] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A.: The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*, 2010.

[14] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. Cambridge, MA: MIT Press, 2016. Print.

[15] Li, F., Johnson, J., Yeung, S.: Lecture Collection | Convolutional Neural Networks for Visual Recognition (Spring 2017). Stanford Lecture Series on Youtube, 2017. Available: https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv.

[16] Wu, J.: Introduction to Convolutional Neural Networks. LAMBDA Group, 2017.

[17] Simonyan, K. & Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR* 2015.

[18] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Code available: https://github.com/rbgirshick/py-faster-rcnn. In *NIPS,* 2015.

[19] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR* 2014.

[20] He, K., Zhang, X., Ren, S., Sun., J.: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *ECCV,* 2014.

[21] Redmon, J., Divvala, S., Girshick, R., Farhadi. A.: You Only Look Once: Unified, Real-Time Object Detection. In *arXiv,* 2016.

[22] Girshick, R.: Fast R-CNN. In *ICCV,* 2015.

[23] Girshick, R.: Fast R-CNN Object Detection with Caffe. From Caffe CVPR presentation, 2015. Available: http://tutorial.caffe.berkeleyvision.org/caffe-cvpr15-detection.pdf

[24] Jia, Y. et al.: Caffe: Convolutional Architecture for Fast Feature Embedding. Available http://caffe.berkeleyvision.org/installation.html. In *ACM,* 2014.

[25] Lemler, K., Heichel, J., Dvorak, D.: Powerline Sensor Trade Study Sensor Test Plan. 2016.

[26] Sony.: Interchangeable Lens Digital Camera α Handbook. 2011. Retrieved from https://docs.sony.com/release/nex-7_handbook.pdf.

[27] Sony.: Interchangeable Lens Digital Camera α6000 Instruction Manual. 2014. Retrieved from https://esupport.sony.com/US/p/model-home.pl?mdl=ILCE6000&LOC=3# /manualsTab.

[28] Freefly Systems. Alta 8 Aircraft Flight Manual. 2017. Retrieved from http://freeflysystems.com/app/uploads/2016/10/2017-01-16_ALTA8_Manual_RevB_01.pdf.

[29] OpenCV Dev Team, (2016) OpenCV Documentation Retrieved from http://opencv.org/documentation.html.

[30] Blanco, J.L.: A tutorial on SE(3) transformation parameterizations and on-manifold optimization. University of Malaga, Tech. Rep, 3. 2010.

[31] Szeliski, R.: Computer vision: algorithms and applications. Springer Science & Business Media. 2010.

[32] Tzutalin. LabelImg. Git code available at: https://github.com/tzutalin/labelImg. 2013.

[33] Arlot, S. and Celisse, A.: A survey of cross-validation procedures for model selection. Statistics surveys, 4, pp. 40-79. 2010.

[34] Oberweger, M., Wendel, A., Bischof, H.: Visual Recognition and Fault Detection for Power Line Insulators. Computer Vision Winter Workshop. 2014.

[35] Gururajapathy, S.S., Mokhlis, H., Illias, H.A., Fault location and detection techniques in power distribution systems with distributed generation: A review. Renewable and Sustainable Energy Reviews, 74, 949-958.

[36] Li, Z., Liu, Y., Hayward, R., Zhang, J., and Cai, J.: Knowledge-based power line detection for UAV surveillance and inspection systems. Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference (pp. 1-6). IEEE.

[37] Nordeng, I. E., Hasan, A., Olsen, D., and Neubert, J.: DEBC Detection with Deep Learning. Scandinavian Conference on Image Analysis (pp. 248-259). Springer, Cham, 2017.

[38] Gonen, T.: Electrical power transmission system engineering: analysis and design. CRC Press, 2011.

[39] Transgrid Network Capability Incentive Project Action Plan for 2014/15-2017/18 Period. 2014.

[40] Adly, C.M.F., Girgis, A., and Lubkeman, D.: A fault location technique for rural distribution feeders. IEEE Trans, 29. 1993.

[41] Koley, E., Verma, K., and Ghosh, S.: An improved fault detection classification and location scheme based on wavelet transform and artificial neural network for six phase transmission line using single end data only. SpringerPlus, 4(1), p. 551. 2015.

[42] Dehghani, F. and Nezami, H.: A new fault location technique on radial distribution systems using artificial neural network. Proceedings of the electricity distribution, 22nd international conference and exhibition, p. 1-4. 2013.

[43] Zhang, R., Li, C., and Jia, D.: A new multi-channels sequence recognition framework using deep convolutional neural network. Procedia Computer Science, 53, pp. 383-390. 2015.

[44] Das, B., and Redd, J.V.: Fuzzy-logic-based fault classification scheme for digital distance protection. IEEE Transactions on Power Delivery, 20(2), pp. 609-616. 2005.

[45] Oh, J., and Lee, C.: 3D power line extraction from multiple aerial images. Sensors 17, no. 10, pp. 2244. 2017.

[46] Girshick, R.: Fast R-CNN Object detection with Caffe. Proceeding of the 13th European Conference on Computer Vision. 2014.

[47] Pagnano, A., Höpf, M. and Teti, R.: A roadmap for automated power line inspection. Maintenance and repair. Procedia CIRP, 12, pp. 234-239. 2013.

[48] Schmugge, S.J., Rice, L., Nguyen, N.R., Lindberg, J., Grizzi, R., Joffe, C., and Shin, M.C.: Detection of cracks in nuclear power plant using spatial-temporal grouping of local patches. In Applications of Computer Vision (WACV), 2016 IEEE Winter Conference, pp. 1-7. 2016.

[49] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A.: Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

[50] Gershenson, C.: Artificial neural networks for beginners. In arXiv preprint cs/0308031. 2003.

[51] Ofli, F., Meier, P., Imran, M., Castillo, C., Tuia, D., Rey, N., Briant, J., Millet, P., Reinhard, F., Parkan, M., and Joost, S.: Combining human computing and machine learning to make sense of big (aerial) data for disaster response. In Big data, 4(1), pp. 47-59. 2016.

[52] Viter, J.: Evaluation of a Python algorithm for parallel convolution. 2017, retrieved from http://jeanvitor.com/convolution-parallel-algorithm-python/

[53] Wu, J.: CNN for Dummies, Lambda Group, 2016.

[54] Compression Dead-End. 2018, retrieved from http://preformed.com/energy/transmission /compression-dead-end-hardware/compression-dead-end-cmpde

[55] Montambault, S., Beaudry, J., Toussaint, K., and Pouliot, N.: On the application of VTOL UAVs to the inspection of power utility assets. In 1st International Conference on Applied Robotics for the Power Industry, Oct. 2010, pp. 1-7. 2010.