ANOMALY DETECTION ON WIND TURBINE BLADES USING AERIAL IMAGING, IMAGE PROCESSING, AND DEEP LEARNING

by

Bridger Kohl Altice

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Todd K. Moon, Ph.D. Major Professor	Mohammad Shekaramiz, Ph.D. Committee Member
Jacob H. Gunther, Ph.D. Committee Member	Jonathan Phillips, Ph.D. Committee Member

D. Richard Cutler, Ph.D.
Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2024

ABSTRACT

Anomaly Detection on Wind Turbine Blades Using Aerial Imaging, Image Processing, and Deep Learning

by

Bridger Kohl Altice, Master of Science
Utah State University, 2024

Major Professor: Todd K. Moon, Ph.D.
Department: Electrical and Computer Engineering

As global temperatures and C02 emissions rise, alternative energy sources to fossil fuels are being utilized, such as wind energy produced by wind turbines. Current maintenance techniques for these turbines are expensive, unsafe, and susceptible to human error. A viable solution to these maintenance problems is to use aerial drone imagery with ML (machine learning) techniques to detect faults in the blades. While neural networks and other machine learning architectures have been shown to effectively detect cracks in blades on static images of wind turbine blades, effective results have yet to be shown for blurry images of moving blades. Having turbines in the operation mode during inspection will lead to additional revenue and power generation at the cost of having blurry blade images. Using image processing techniques in concurrence with machine learning architectures may provide the edge needed to successfully detect damage features in blurry blade images. In this study, sharp wind turbine blade images are classified as healthy or faulty using custom CNN (convolutional neural network), ResNet-50, and Transfer Learning Xception machine learning architectures. Then a velocity-based Richardson-Lucy algorithm and a velocity-based Wiener deconvolution algorithm are used to deblur blurry rotating blade images. Finally, these deblurred blade images are classified using the three ML architectures previously implemented.

(96 pages)

PUBLIC ABSTRACT

Anomaly Detection on Wind Turbine Blades Using Aerial Imaging, Image Processing, and Deep Learning

Bridger Kohl Altice

In reaction to rising global temperatures and carbon dioxide emissions, many countries are looking to use energy sources other than fossil fuels. One such source of energy is wind energy, which can be harvested by wind turbines. By rotating at high speeds, the blades of these large turbines are able to convert wind energy to kinetic energy, which is then converted to electricity usable by the power grid. Traditional methods for inspecting these turbines for damages are expensive, unsafe, and susceptible to human error. These turbines are so tall and so large that inspectors run the risk of falling from large heights or being injured by falling turbine debris. It is also difficult to spot every damaged area on a blade so large. A solution to this problem is to have a drone fly up and take pictures of the blades. Afterwards, these pictures can be processed by a machine learning architecture, which is a specific type of AI (artificial intelligence). The AI will then tell the inspectors if damages exist on the blades. Wind turbines are normally turned off during inspection for the safety of the inspectors. If the turbines are left on during the inspection process to save money, the blades will likely be moving, so images of the blades may be blurry. This will make it more difficult for the AI to detect damages. This is why deblurring the images before further processing could be a great way to still have accurate results from the AI with rotating blades. In this study, the health of wind turbine blades is determined using specific machine architectures along with image deblurring techniques and a custom-made image set of wind turbine blades taken with a drone.

ACKNOWLEDGMENTS

I would like to thank my advisors Dr. Moon and Dr. Shekaramiz for pushing me every week to make progress on this research. Thank you Dr. Shekaramiz for allowing me to be part of this special project and for making collaboration on this project possible. Thanks Dr. Moon for your patience with me and guidance these past two years working in research and in the classroom. Thank you Dr. Gunther for your advice on image processing, and Dr. Phillips for teaching me how to be a better engineer through my coursework.

I am grateful for all the UVU students and peers who made this research possible. Thank you Edwin Nazario Dejesus and Mason Davis for answering all my questions and introducing me to machine learning architectures, Joshua Zander and Angel Rodriguez for their contributions gathering the rotating turbine blade images and data used in this research, and Vicky Odhiambo, Joshua Zander, and Jeremiah Engel for their contributions to creating the CAI-SWTB dataset used to train my machine learning models in this research.

Much gratitude to the UVU Machine Learning and Drone Lab for funding this research through the Office of the Commissioner of Utah System of Higher Education (USHE)-Deep Technology Initiative Grant 20210016UT.

Thanks to my cross-country and track and field coaches Artie Gulden and Sarah Nardoni for believing in both my running and academics and for providing me with the scholarship I needed to make this thesis possible. I am grateful for my teammates as well for listening to me ramble on about my research at practice.

Thanks to my awesome family for supporting me through this academic journey. I am excited to see where these experiences take me!

Lastly, thank you to my awesome girlfriend Maddie for feeding me and staying up with me as I put in many late nights working on this research. My brain and heart would not have been properly fed without you.

Bridger Altice

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

AI	Artificial Intelligence
CAI-SWTB Dataset	Combined Aerial and Indoor Small Wind Turbine Blade dataset
CHPC	Center for High Performance Computing
CNN	Convolutional Neural Network
DL	Deep Learning
ML	Machine Learning
MSE	Mean Squared Error
OTF	Optical Transfer Function
PSF	Point Spread Function
ResNet	Residual Network
RGB	Red Green Blue
RL	Richardson-Lucy
TL	Transfer Learning
UAV	Unmanned Aerial Vehicle
USHE	Utah System of Higher Education
USU	Utah State University
UVU	Utah Valley University
WT	Wind Turbine
Xception	Extreme Inception
YOLO	You Only Look Once

CHAPTER 1

Introduction

In response to the rapid and unsustainable consumption of fossil fuels, alternative renewable energy sources are being considered worldwide [3]. Some of the world's leading sources of renewable energy include wind, solar, hydro, geothermal, and bioenergy. Wind turbines are currently one of the fastest growing sources of renewable energy [4]. In 2022, wind energy made up nearly 7.33 percent of global electricity generation [5] and still created more power than all other non-hydro renewable energy sources, generating more than 2100 TWh [6]. This is why safe, inexpensive maintenance and repair of wind turbines is essential for meeting global clean energy needs. Wind turbines in the United States have an average height of about 280 feet [7] and weigh roughly 35 tons. Due to these large metrics, wind turbine inspectors risk falling from large heights or getting hit by heavy debris from damaged turbine blades. Hundreds of injuries or deaths have occurred during wind turbine inspections in the last decade, and even more occur which are not reported [8]. Additionally, wind turbine blade replacement can cost up to $200,000 on average [9].

Because current inspection methods require turbine blade motion to be switched off, there are large losses in power generation during inspection time. Inspections can take hours depending on the complexity of blade damages. As an example, for a single 1 MW turbine, a minimum of 3-6 MWh of electricity is lost per inspection. Based on the average price of 17.3¢/kWh in the United States as of January 2024, 1 MWh is worth around $173 [10]. This means that an inspection for a single 1 MW turbine would result in a minimum of $519-$1038 lost.

To mitigate the negative aspects of current inspection methods, alternative safer methods using aerial imagery from autonomous drones have been explored [11]. The current overall approach being developed in this thesis in collaboration with Utah Valley University's Machine Learning and Drone Lab is described as follows [12]. First, a drone is used to take images of wind turbine blades. These images are then run through ML algorithms to classify each image as either healthy or faulty (having damages). Damages can include holes, cracks, edge-erosion, and warping. Using this drone-ML method makes wind turbine inspection both safer and more efficient. As before with traditional inspection methods, a limitation of this method is that, while it is safer and more efficient than currently used manual techniques, the wind turbines must be switched off and the blades stopped in order to take sharp enough images for the ML algorithms to correctly classify each blade. Since the ML architectures are trained on sharp image blade data rather than blurry image data, it is likely that damages will only be able to be detected if the test images input to the architectures are sharp as well. In order to save power and money, a method for attaining sharp wind turbine blade images from a rotating blade is essential.

In this thesis research, a custom CNN ML architecture, transfer-learning ResNet-50 ML architecture, and Transfer-Learning Xception ML architecture are used to determine if images of wind turbine blades contain healthy or faulty blades. The UVU Machine Learning and Drone Lab's *Combined Aerial and Indoor Small Wind Turbine Blade* (CAI-SWTB) dataset of healthy and damaged wind turbine blades taken with a drone is used to train and test these machine learning models. Both static and blurry images are tested by these models. Then, a velocity-based RL (Richardson-Lucy) algorithm and a velocity-based Wiener deconvolution algorithm are used to deblur both artificial and real images of rotating wind turbine blades. Finally, the original three ML algorithms trained on sharp images are used to classify the deblurred blade images as healthy or faulty after being run through the velocity-based deblurring algorithms. Test results classifying sharp/static images, blurry images, and deblurred images are then compared. With good enough results, the algorithms in this study will be implemented by UVU's drone flight path-planning team to assess wind turbine damages in real time.

The rest of this thesis is organized as follows: Chapter 2 explores prior research using machine learning architectures and deblurring techniques. Chapter 3 showcases the results of using the ML/deep learning algorithms used in this study. The methods and results for deblurring techniques are defined in chapter 4. Chapter 5 details the outcome of classifying blurry and deblurred images with the ML models discussed in Chapter 3. This is followed by the conclusion.

CHAPTER 2

Background and Related Works

2.1 Machine Learning Architectures

2.1.1 Alternative Drone-Imagery Blade Damage Detection Methods

A small variety of different approaches to using drone imagery and deep learning have been explored for assessing damages on wind turbine blades. UVU has used path-planning algorithms to fly a drone autonomously which can explore and locate wind turbines using camera imagery [11]. Haar-like features and clustering, residual and wavelet-based neural networks, support vector machines with fuzzy logic, YOLO, thermal imagery, spectrograms, auto-encoders, and many more have all been applied in other works to detect faults in aerial imagery of wind turbine blades or classify the imagery as healthy or damaged [13–21]. Chinese researches alternatively employed UAVs capture images of wind turbine blades and then used a cascading classifier algorithm to identify and locate wind turbine blade cracks [22, 23].

2.1.2 Convolutional Neural Network (CNN)

CNNs (convolutional Neaural Networks) are a type of neural network that use kernels of differing sizes and convolution to output the next connection layers [12]. Using this method reduces the total number of computations when compared to a traditional fully connected neural network. Having less computations is important for large image classification and for training on data sets with many images. These kernels are comprised of weights and biases that can be trained through back-propagation to produce regional feature-extracting capabilities. A typical architecture of a CNN is illustrated in Figure 2.1. Here, convolutional kernels, fully connected layers, and pooling layers are shown. Pooling layers are used to provide down sampling while maintaining important features. A more in depth explanation of CNN architectures and how they are constructed can be found in [2, 24].

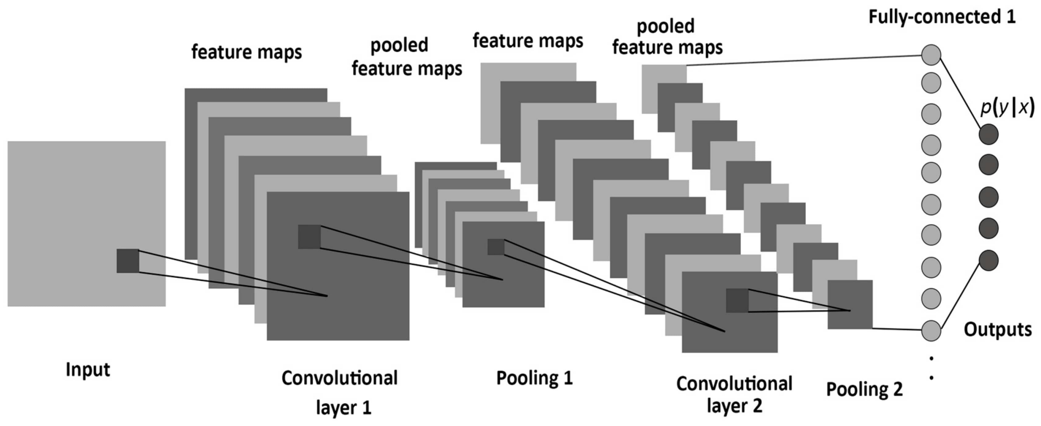


Fig. 2.1: The structure of a CNN, consisting of convolutional, pooling, and fully-connected layers [2].

In one study, researchers used image enhancement and augmentation to improve the quality and quantity of wind turbine blade images, and then used an attention mechanism-based CNN for surface damage detection of wind turbine blades [25]. Another study used a multi-variable spectrum imaging and CNN-based method to detect aerodynamic imbalance and mass imbalance in a wind turbine blade.

2.1.3 ResNet-50

ResNet, which stands for Residual Network, is a version of a CNN which utilizes skip and residual connections [26]. ResNet-50 is the popular 50-layer architecture in the ResNet family. Skip connections were applied to the nueral network to minimize the backpropagation loss in deep networks by allowing new paths for backpropagation to reach earlier network layers, thus eliminating the vanishing gradient problem. The vanishing gradient problem is when the gradient, which is used to update the model after each round of training, is diminished to not allow for effective updates of earlier layers. Using skip connections allows for more convolutional layers to be processed with less computational time per layer. Residual connections, a type of skip connection, are incorporated as well to allow for the residual function of the input layer to be included across layers.

ResNet-50 pretrained with ImageNet weights and custom fully-connected head layers have been used in other works to provide high test precision and recall on identifying small blade chips and cracks [27].

2.1.4 Xception

Xception, also known as Extreme Inception, was developed by Google in 2017 [28]. Xception takes properties of its predecessor, the Inception model, and magnifies them significantly, utilizing features such as depthwise separable convolutional layers and skip connections. Depthwise separable convolution reduces processing time and accounts for most input features by first performing convolution on each of the three color channels in an RGB image separately and then concatenating each output [28]. Within each convolutional layer, kernels of varying sizes are used to perform convolution, each followed by a 1×1 pointwise convolution. Then, each result is combined by going through filter concatenation. Using a variety of kernel sizes allows for more features to be extracted from each input. Overall Xception contains a total of 74 separable convolutional, activation, pooling, and fully connected layers.

Transfer learning Xception models have been used to classify various types of images, including weather images, images of garbage, and images of different environments in nature [29–31].

2.1.5 Transfer Learning

Transfer learning (TL) is the process of using pretrained neuron weights from one model and applying them to another model to be trained for a different purpose [32]. These pretrained weights are used so that a model has a reasonable starting point for training, rather than starting from scratch with default neuron weights [12].

One transfer learning approach is to freeze the pretrained feature extraction layers and then only train the remaining top dense layers on the desired dataset. The idea is that the frozen layers will already be trained for general image classification. Thus, only the top layers need to be fine-tuned for the specific purposes of the transfer learning model. Training only the top layers significantly reduces training time.

An alternative approach is to use the pretrained layers as a base and to further train all layers of the model, including the pretrained layers, with the new desired dataset. Since in this approach every layer is updated, it takes more time than the first approach to reach high test and validation accuracies, but it is still faster than training the model completely from scratch.

As previously stated, transfer learning Xception models have been used to classify various types of images, including weather images, images of garbage, and images of different environments in nature [29–31]. Other transfer learning model have been used to detect icing on wind turbine blades [33–35].

2.2 Deblurring Methods

2.2.1 Alternative Deblurring Methods

Since the center and tip of a rotating wind turbine blade are moving at different speeds, different amounts of blurring occur at each point [36]. One work addresses this nonlinear-blurring effect by cropping grayscale turbine blade images into sections, estimating the point spread function (PSF) using the Cepstrum technique and Radon transform in each section, and applying linear deblurring using a WF (Wiener filter) [37] individually on each section. The deblurred sections of the blade are then spliced back together to create the image of the entire blade. Wiener deconvolution is used in other research to deblur RGB images of people [38].

Another work addresses possible image blur created by the motion of drone rather than the rotation of the wind turbine blade [39]. This type of blur is avoided using an algorithm that automatically adjusts the movements and speeds of the drone depending on wind resistance in order to achieve sharp wind turbine blade images. An approach using a completely different method uses a DeblurGanv2 and inception-ResNet-v2 combination deep-learning model trained on real and synthetic pairs of static and blurry blade images to deblur wind turbine blade images [40]. The synthetic pairs are created by taking real blade images, cropping out the blade, applying a blur, then splicing the blurred blade back into the original background. Another deep learning approach was used in a separate study, where gyroscope measurements and deep neural networks were used to deblur blurry images caused by camera movement [41]. According to the study, their method calculated homography matrices from the gyroscope data and warped the input image to imitate the camera motion. Stacking the warped images aligned blur artifacts caused by camera movement channel-wise, and the network was trained to use image features related to the direction and magnitude of the blur. It is seems to be common these days to use some form of a neural network to perform deblurring rather than just using simpler image processing methods.

2.2.2 Richardson-Lucy Algorithm

The Richardson-Lucy algorithm is an iterative method used for deblurring images [36, 42–44]. There are two common types of Richardson-Lucy algorithms used for image deblurring: blind Richardson-Lucy deconvolution and classic Richardson-Lucy deconvolution with a prior known PSF (point spread function). The blind RL-deconvolution estimates the PSF with each iteration and applies it to the blurred image. Because the PSF is just an estimate, this method is ultimately less effective at deblurring than using the actual PSF to deblur an image. Classic RL-deconvolution with a known PSF is what is used in this work, and the PSF is calculated by taking into account the velocity of wind turbine blades, camera shutter time, position of the camera relative to the turbine blade, and other variables. The iterative equation for the RL-algorithm is

$$\hat{u}^{(t+1)} = \hat{u}^{(t)}\left(\frac{d}{\hat{u}^{(t)} \otimes P}\right)P^{*}, \tag{2.1}$$

where P represents the 2D PSF used to blur the image, P^{*} is the 2D PSF flipped horizontally and vertically, d represents the original blurry image (4.1), $\hat{u}^{(t)}$ represents the current estimate of the deblurred image after t iterations, $\hat{u}^{(t+1)}$ represents the next estimate of a deblurred image after $(t+1)$ iterations, and $\otimes$ represents 2D convolution. Other operations are element-wise operations, so to keep the results of 2D convolutions the same dimensions as the original blurry photo, padding is performed before the convolution. For the very first step of the algorithm, $t=0$ and $\hat{u}^{(0)}=d$.

2.2.3 Wiener Deconvolution

The purpose of the traditional 1D Wiener filter is to minimize the MSE (mean squared error) between the signal and the estimated/calculated signal found from the algorithm. For image deblurring, Wiener deconvolution using a Wiener restoration filter is used, which applies similar techniques from the Wiener filter to images or other 2D arrays of data [45,46]. The formulas for calculating the deconvolved 2D images using Wiener deconvolution, which take advantage of frequency domain operations rather than convolution in the time domain, are shown in (2.2), (2.3), (2.4), and (2.5). These are based on equations from Matlab's *deconvwnr* function, and the variable names and nomenclature are kept the same as in Matlab's documentation for consistency.

$$G(k,l)=\frac{H^*(k,l)}{|H(k,l)|^2+S_u(k,l)/S_x(x,l)} \tag{2.2}$$

In (2.2), k and l represent the x and y dimensions of the 2D array. $H(k,l)$ represents the OTF (optical transfer function), which is the Fourier Transform of the PSF (P), $H^*(k,l)$ is the complex conjugate of $H(k,l)$, and $|H(k,l)|^2$ is the absolute value of $H(k,l)$ squared. S_x is the signal power spectrum, and S_u is the noise power spectrum. Equation 2.2 can be modified to

$$G(k,l)=\frac{H^*(k,l)S_x(k,l)}{|H(k,l)|^2S_x(k,l)+S_u(k,l)} \tag{2.3}$$

in order to minimize possible problems associated with divisions, such as problems of diving by 0. Because the signal power spectrum S_x can't be known from an image of a blurry blade, in this work we make $S_u=1$, and $S_u=NSR$, where NSR is the noise to signal ratio. This is the hyperparamter which is adjusted for optimal deblurring. These changes result in

$$G(k,l) = \frac{H^*(k,l)}{|H(k,l)|^2 + NSR}. \tag{2.4}$$

The final step of the algorithm is

$$J = \mathcal{F}^{-1}[G(k,l) \times \mathcal{F}[I](k,l)]. \tag{2.5}$$

Here, J is the resulting 2D deblurred image, which is the same as $\hat{u}$ in the Richardson-Lucy algorithm. $\mathcal{F}$ and $\mathcal{F}^{-1}$ represent the Fourier Transform and the inverse Fourier Transform. I represents the blurry image input to the image, and this is the same as d in the Richardson-Lucy algorithm. In the equation, $G(k,l)$ and $\mathcal{F}[I](k,l)$ are being multiplied. Again, these variable names were chosen to be consistent with Matlab's documentation. Overall, Wiener deconvolution essentially divides out the PSF in the frequency domain, which is basically the same as deconvolution in the time domain, while also adjusting for the NSR.

The WF deblurring method used in this paper is similar to the method used in [38], except a python implementation of the *deconvwnr* function was created and used instead. Just as with the Richardson-Lucy algorithm, in this research the PSF is calculated by taking into account the velocity of wind turbine blades, camera shutter time, position of the camera relative to the turbine blade, and other variables.

CHAPTER 3
ML Image Classification

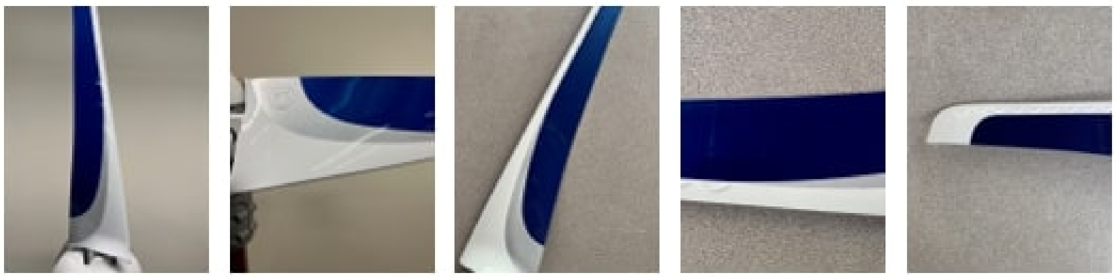
3.1 Image Collection

3.1.1 CAI-SWTB Dataset

To successfully train machine learning models to accurately classify wind turbine blade images, thousands of images are needed. Students at UVU's Machine Learning and Drone Lab have provided all of the images used for the first part of this study in the CAI-SWTB dataset, as this research is in collaboration with them. Due to the unavailability of sufficient public images of damaged commercial-grade large-scale turbine blades, a Primus Air Max wind turbine was deployed to create the small wind turbine dataset used in this research. A total of 6 blades were used to create a set of healthy and a set of faulty blades for the purpose of dataset generation. Students in UVU's mechanical engineering department simulated blade cracks, holes, and edge-erosion on the blades of the turbine by carving and drilling holes into them. How these damages appeared was determined by the students' structural analysis tests and simulations for the blades. This analysis was done to ensure that the damages on the blades in these images look as similar as possible to damages in images of in-use commercial wind turbine blades. The anomalies were applied to both the front and back of the blade surface. Following the creation of these simulated faults, images were collected to create the dataset.

Simulating a traditional snapshot of damage taken during human inspection, images of the newly created faulty and healthy Primus AirMax wind turbine blades were taken using a smartphone within an indoor environment. 4000 RGB images in total were collected within this environment containing a 50% faulty and 50% healthy distribution. Figure 3.1 illustrates a sample of the images collected in this environment. To introduce more realistic environmental features into our dataset, the small wind turbine was also placed outside on a balcony and images were taken utilizing DJI Mini 3 Pro and Mini SE drones, illustrated in Figure 3.2. This allowed a more diverse set of features to be present in the dataset including environmental variables such as clouds and sunlight. Samples from the dataset of these outdoor images are shown in Figure 3.3. A total of 2000 RGB aerial images were captured with the fixed distribution of faults being 50%.

By merging both sets, the CAI-SWTB dataset comprises 4000 indoor and 2000 outdoor images, yielding a cumulative total of 6000 images, each with the size of 300×300×3. These images were then split into faulty or healthy classes to enable binary classification, with 50% of the images in each class. The images for each class were then separated into splits of 70% for training, 20% for testing, and 10% for validation. The total number of images for each class is detailed in Table 3.1. The created *Combined Indoor and Outdoor Small Wind Turbine* Dataset (CAI-SWTB) can be accessed at the following link: `https://www.kaggle.com/datasets/mohammadshekaramiz/small-wind-turbine-blade-dataset-cai-swtb`.


(a) Sample images of healthy blades in an indoor environment.

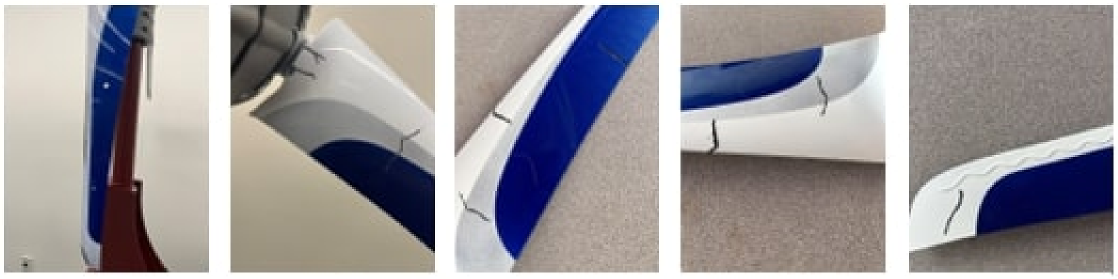

(b) Sample images of faulty (damaged) blades in an indoor environment.

Fig. 3.1: Sample blade images of the created dataset CAI-SWTB taken in an indoor environment.

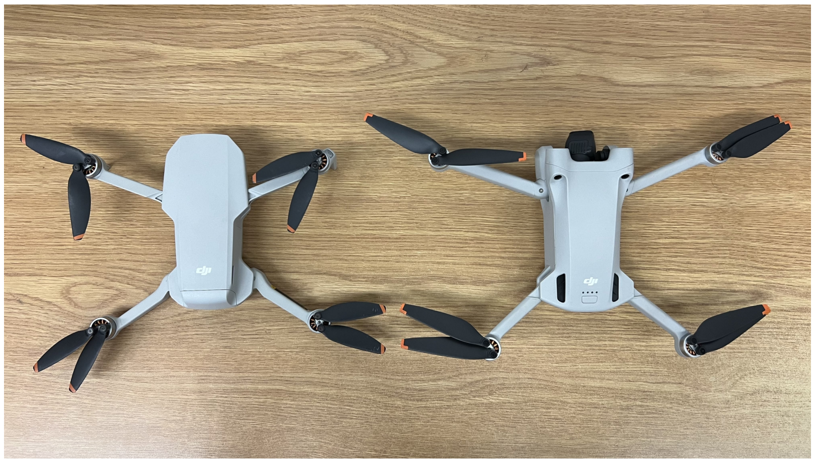
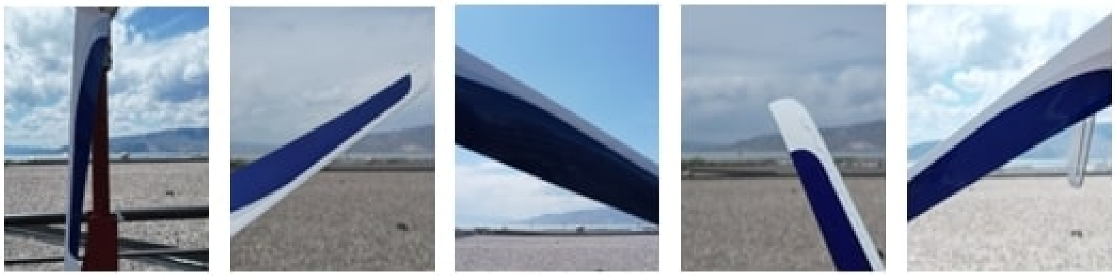
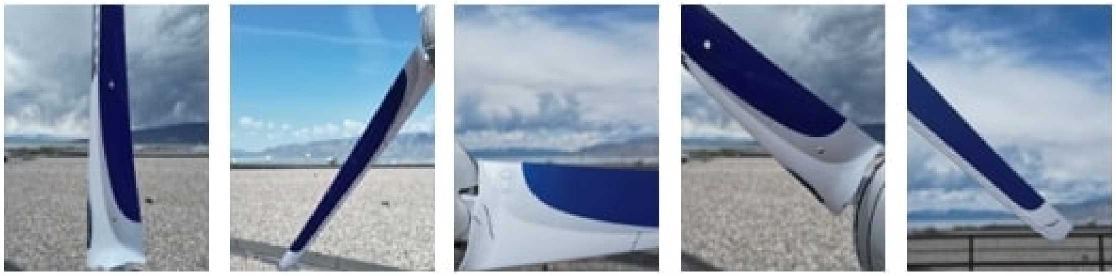


Fig. 3.2: Drones used for aerial imaging. From left to right: DJI Mini SE and DJI Mini 3 Pro.



(a) Sample images of healthy blades in an outdoor environment.



(b) Sample images of faulty (damaged) blades in an outdoor environment.

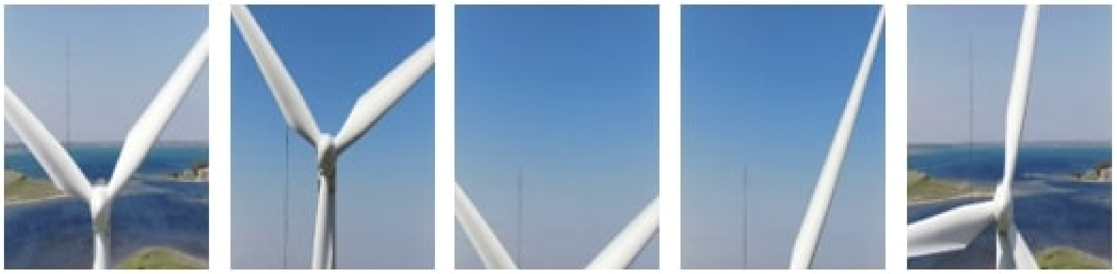
Fig. 3.3: Sample blade images of the created dataset CAI-SWTB taken in an outdoor environment.

Table 3.1: Small Wind Turbine Dataset Split (CAI-SWTB).

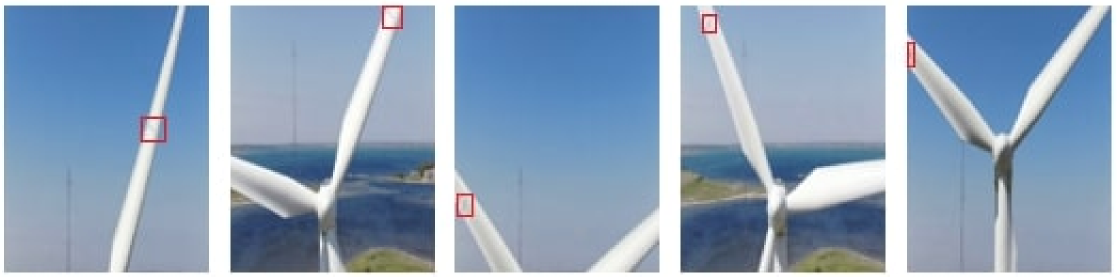
Image Label	***Training Set***	***Validation Set***	***Test Set***	***Total Number of Images***
Indoor Faulty	1400	200	400	2000
Indoor Healthy	1400	200	400	2000
Outdoor Faulty	700	100	200	1000
Outdoor Healthy	700	100	200	1000
Total	4200	600	1200	6000

3.1.2 Large-Scale Wind Turbine Dataset

To compare model performance further and gauge performance on large-scale turbines, the data set of Drone-based Optical and Thermal Videos of Rotor Blades Taken in Normal Wind Turbine Operation, was utilized [1]. This dataset contains images of faulty and healthy turbine blades in a variety of formats including Thermal, RGB, and low-light taken with DJI drones. To maintain a fair comparison between our dataset and this dataset, the daytime RGB images were selected from this dataset to be used for analysis. To preserve model input size and overall parameters, the images here were also down-scaled from their original size of 853×480 to 300×300 allowing the data to match the size of the CAI-SWTB dataset. Additionally, due to the imbalance of representation between faulty and healthy turbine blades in this dataset, a random selection of an equal split of faulty and healthy was established. This resulted in a total of 516 images collected with 258 of each class, faulty and healthy. These images were then randomly split into training, validation, and test sets with the same spit as CAI-SWTB, 70% training, 20% testing, and 10% validation. These split totals are shown in Table 3.2. Samples from this dataset are shown in Figure 3.4 with the healthy blades shown in Figure 3.4a and the faulty shown in Figure 3.4b. Here, positions of faults are highlighted with a box for ease of locating in the figure. The faults included in this dataset consist of cracks at various positions in the blade lengths as captured through drone aerial imagery.



(a) Sample images of healthy blades on the large wind turbine.



(b) Sample images of faulty blades on the large wind turbine. Boxes indicate where the fault lies in the image.

Fig. 3.4: Sample blade images of the large-scale wind turbine dataset [1].

Table 3.2: Drone-based Optical and Thermal Videos of Rotor Blades Taken in Normal Wind Turbine Operation Dataset Split [1].

Image Label	***Training Set***	***Validation Set***	***Test Set***	***Total Number of Images***
Faulty	181	25	52	258
Healthy	181	25	52	258
Total	362	50	104	516

3.2 ML Simulation Development

This section outlines the process of how each ML architecture was created and tested in this research. Python is the programming language used for this portion and the rest of this project, as it already contains many deep learning libraries such as keras, tensorflow, and pytorch. The path-planning and other machine learning teams on the project already use python as well, so it is easiest to integrate my code with theirs when we are all coding in the same language.

3.2.1 Hardware Specifications

Two computers were used to run the ML simulations. The first computer consists of GPU hardware obtained from Google Colaboratory. These computers contain an Intel Xeon CPU and an NVIDIA Tesla T4 GPU. The second computer accessed Center for High-Performance Computing (CHPC) resources from the University of Utah which utilizes varying GPUs.

3.2.2 Custom Convolutional Neural Network

The first model that was developed was a custom CNN. This architecture replicates the properties of VGG-19. It contains six convolutional layers, three max pool layers, a dropout layer, varying dense layers, activation functions, and optimization functions. The size of the input image was 300×300. Each convolutional layer used a relu activation function and a 3×3 kernel size. The first two 2D-convolutional layers used a filter size of 16, the next two used 32, and the last two used 64. This filter is chosen to be smaller in the first convolutional layers so the model can detect simpler/larger features. Once those are detected, the filter size is increased so the model can detect the smaller or more complex features in the images. Each maxpool layer is of size 2×2. The learning rate for this architecture is set to 0.001 to reduce the number of parameters being tuned. The model is illustrated in Figure 3.5 and depicts how the shape and size of the data change as it progresses through each layer. Code for this architecture is listed in Section 7.0.1.

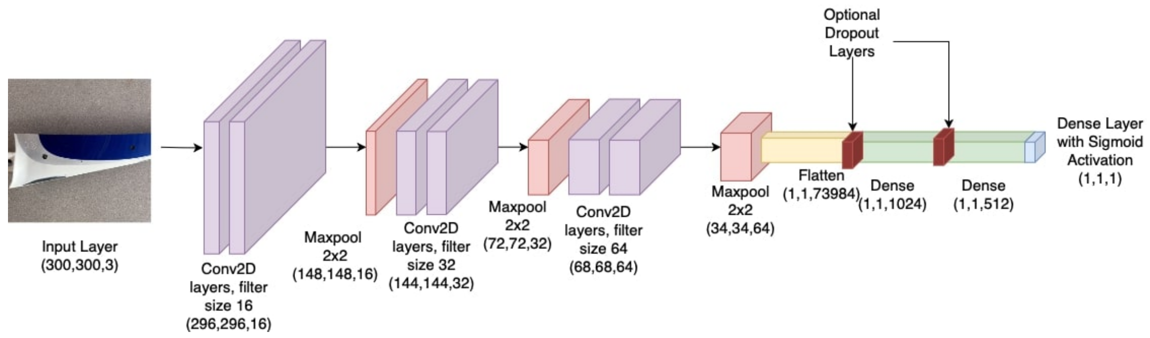

Optional Dropout Layers
Input Layer (300,300,3)
Conv2D layers, filter size 16 (296,296,16)
Maxpool 2x2 (148,148,16)
Conv2D layers, filter size 32 (144,144,32)
Maxpool 2x2 (72,72,32)
Conv2D layers, filter size 64 (68,68,64)
Maxpool 2x2 (34,34,64)
Flatten (1,1,73984)
Dense (1,1,1024)
Dense (1,1,512)
Dense Layer with Sigmoid Activation (1,1,1)

Fig. 3.5: Custom CNN model.

3.2.3 Transfer Learning ResNet-50

The second architecture developed was transfer-learning ResNet-50. For this architecture, the pretrained Keras ImageNet weights for ResNet-50 were applied. Initially, experiments were performed using the pretrained-layer-freezing approach where only the top layers were trained on the new wind turbine blade image set, but most of these models could only classify images with accuracies of up to 50%. Using the second approach, the pretrained neurons were set to 'trainable' mode so that the entire model, not just the top layers, could learn how to identify wind turbine blade damages specifically. This second approach achieved better accuracy and quicker convergence than the frozen-only approach, so the unfrozen approach became the standard for future simulations. The Keras 'top' neurons were not included, but custom layers at the end of the model were added instead. These layers were the same end layers that were used in the CNN model, and are shown in Figure 3.5. They consisted of a flatten layer, followed by a 1024 size dense layer, a 512 dense layer, and a 1 size dense layer. Code for this architecture is listed in Section 7.0.2.

3.2.4 Transfer Learning Xception

The last architecture developed was transfer-learning Xception. Similar to the TL-ResNet50 model above, the pretrained Keras ImageNet weights for Xception were applied, with the pretrained neurons set to 'trainable'. The final flatten and dense layers used in the CNN and ResNet50 models described previously were used in place of the Keras 'top' layers for the TL-Xception model as well, and are shown in Figure 3.5. These layers consisted of a flatten layer, followed by a 1024 size dense layer, a 512 dense layer, and a 1 size dense layer. Code for this architecture is listed in Section 7.0.3.

3.2.5 Hyper-Parameter Tuning

Hyper-parameter tuning is the process of changing model parameters in order to locate an optimal combination for performance. This includes the optimization algorithm, the number of epochs, the dropout layer rates, and the optimizer learning rates. In this research, hyper-parameter tuning was implemented using custom functions during the building of each architecture to insert the parameters requested for the current model. The parameters for the simulations included epochs, dropout rates, activation functions, optimization algorithms, batch size, pooling layers, and learning rate. In each simulation, the loss function was consistent with binary cross-entropy as well as image size and dataset split as noted in Tables 3.1 and 3.2. Table 3.3 below shows the hyper-parameters that were tested with each architecture to find the best combination.

Table 3.3: Hyper-Parameters Tested for Each Architecture.

Epochs	50	100	-	-	-
Batch Size	32	64	256	-	-
Optimizer	Adam	Nadam	RMSprop	-	-
Activation Function	sigmoid	softmax	-	-	-
Learning Rate	0.001	-	-	-	-
Dropout Rate	0.0	0.2	0.4	0.6	0.8
Pooling	Max	Avg	-	-	-

3.2.6 Pre-processing and Image Augmentation

To assist with model training and convergence, image augmentation was deployed to increase the number of unique images seen by a model. This applies different filters on the dataset such as tilting, zooming, and shearing. Before the images were augmented, they were first normalized through dividing the pixel values by 255. Image augmentation was implemented using Keras's pre-processing Image Data Generator. Random variations in zoom, shear, rotation, and flipping were applied to find the best possible augmentation parameters, and the values were chosen conservatively to maintain the fault features of the images. While the number of images in the training set remains fixed, the number of unique image variations seen by the model increases. The augmentation parameters used in this research are shown in Tables 3.4 and 3.9.

3.3 ML Sharp Image Classification Results

3.3.1 Image Classification using CAI-SWTB Dataset

The augmentation parameters used in tests for the CAI-SWTB dataset are shown in Tables 3.4.

Table 3.4: Image Augmentation Parameters for CAI-SWTB Dataset Tests.

Augmentation Parameters	***Setting***
Shear Range	0.2
Zoom Range	0.2
Rotation Range	40
Horizontal Flip	True
Fill Mode	Nearest

Through exhaustive searching using the hyper-parameters in Table 3.3, optimal hyper-parameter combinations were found and are shown in Table 3.5. These combinations of hyper-parameters are what gave the highest test accuracy results for each ML architecture trained on and tested using the CAI-SWTB dataset.

Table 3.5: Best Model Parameters. AF and LR Denote the Activation Function and Learning Rate, Respectively.

Architecture	Epochs	Batch Size	Optimizer	AF	LR	Dropout	Pooling
(TL)-Xception	**50**	**32**	**RMSprop**	**Sigmoid**	**0.001**	**0.6**	**Avg**
(TL)-ResNet-50	50	32	RMSprop	Sigmoid	0.001	0.6	Avg
Custom CNN	50	64	Adam	Sigmoid	0.001	0.0	Max

Following the training on the CAI-SWTB dataset making use of the best found hyper-parameters, each model was evaluated on the unseen test set. The performance of each model is described in Table 3.6. Comparing with other students working on this project at UVU, it was apparent that the transfer learning technique provided consistent improvements over the base variants for each model. In this table, it can be seen that the TL-Xception model achieved the highest accuracy of 99.92%. This was followed by TL-ResNet50 with an accuracy of 98.67%. The custom CNN created in this research obtained an accuracy of 96.67% which is notable considering it is 12 layers deep compared to the larger models of Xception with 71 layers and ResNet-50 with 50 layers.

Table 3.6: Compared Model Performance on Combined Indoor and Outdoor Test Data.

Architecture	Test Accuracy	Test Precision	Test Recall	F1-Score
(TL)-Xception	**0.9992**	**0.9983**	**1.0000**	**0.9983**
(TL)-ResNet-50	0.9867	0.9950	0.9787	0.9868
Custom CNN	0.9667	0.9600	0.9730	0.9664

To further visualize the performance of each model during training, their accuracy and loss values are plotted against the epochs and are included in Figs. 3.6,3.7, and 3.8. The transfer learning variants achieved highest accuracy at earlier epochs (see Fig. 3.6) compared to the base architectures created by the other UVU students working on this project. This is due to the leveraged weights from the ImageNet dataset training providing a better starting point for model convergence on this dataset. Subsequently, each transfer learning model also achieved minimized loss in less epochs. The Custom CNN model required more epochs to converge (see Figure 3.8) than the transfer learning models. Consequently, the loss value also follows this pattern.

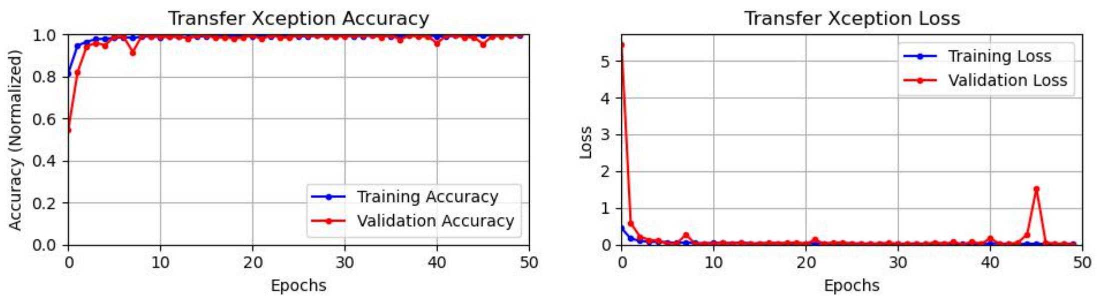


Fig. 3.6: Best run results from the proposed Transfer Learning-based Xception, i,e., (TL)-Xception.

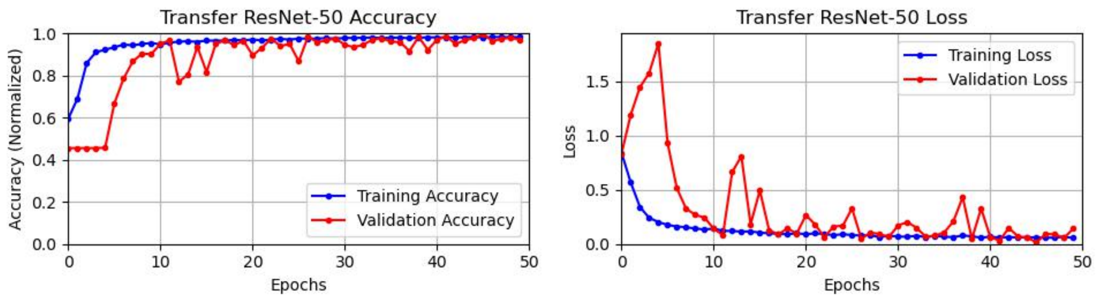


Fig. 3.7: Best run results from the proposed Transfer Learning-based ResNet-50, i.e., (TL)-ResNet-50.

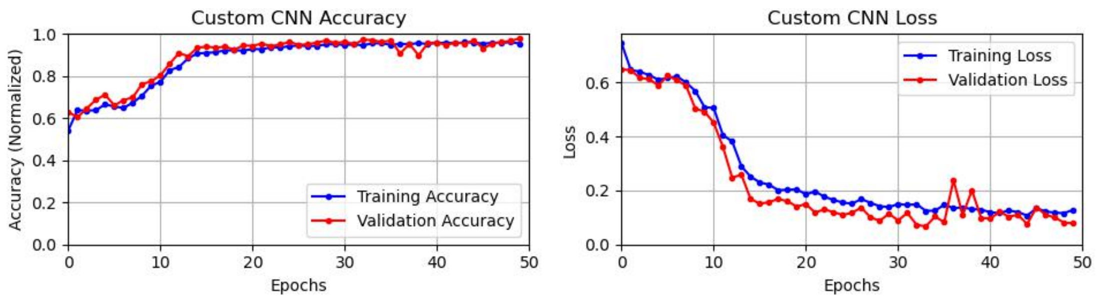


Fig. 3.8: Best run results from Custom CNN.

Model size and layer count is an important feature of CNN's as smaller models provide quicker inference and training time. However, lower layer counts can reduce the feature extraction capability of the model leading to lower accuracy. Consequently, the custom CNN with a lower layer count than the transfer learning models obtained the lowest test accuracy of 96.67% out of the three models. The trend in the data correlates the deeper architectures, by layer count, achieve a higher accuracy in the classification of wind turbine faults. The model with the highest accuracy in the research, TL-Xception, contains 71 layers and is the largest in the research.

Results on Indoor Images of CAI-SWTB dataset

For further analysis of the trained models, here we study how they performed on just the indoor portion of the dataset. It is important to note that these models were trained on the combination of indoor and outdoor images together prior to being evaluated on the test data from the Indoor subsection, totaling 800 images as seen in Table 3.1. The model performance on the indoor test images is shown in Table 3.7. The results show the TL-Xception model had the highest accuracy, achieving 100% accuracy on the indoor image test set. The proposed custom CNN also had notable performance on this set of data obtaining 98.62% accuracy, and tied the TL-ResNet-50 model which also obtained a test accuracy of 98.62%. Each model achieved above 90% accuracy on this subset of data showing promise for synthesizing faults.

Table 3.7: Compared Model Performance on Indoor Test Data.

Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-Xception	**1.000**	**1.000**	**1.000**	**1.000**
(TL)-ResNet-50	0.9862	0.9974	0.9750	0.9861
Custom CNN	0.9862	0.9974	0.9750	0.9861

Results on Outdoor Aerial Images of CAI-SWTB dataset

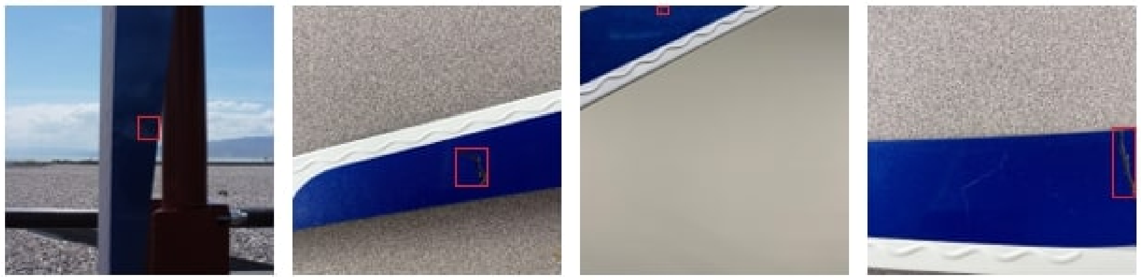
Following the evaluation of the indoor test data, the impact of the environmental features introduced in the outdoor aerial image portion of the dataset is investigated. The total test allocation of the aerial images included 200 healthy and 200 faulty. The performance on the aerial image subset of test data is illustrated in Table 3.8. Here, it can be seen that the highest accuracy is achieved by TL-Xception at 99.75%. This was followed by TL-ResNet-50, which obtained 99.75% accuracy on this set. The custom CNN's performance dropped on the outdoor data however, still obtained 95% accuracy. Overall, each model achieved above 90% accuracy showing a promising trend in the ability to determine faults in the domain of wind turbine blades.

Table 3.8: Compared Model Performance on Outdoor Test Data.

Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-Xception	**0.9975**	**0.9950**	**1.0000**	**0.9975**
(TL)-ResNet-50	0.9875	0.9899	0.9850	0.9875
Custom CNN	0.9500	0.9327	0.9700	0.9510

Common Miscategorization of Blade Images

In the investigation of commonly miscategorized images shown in Figure 3.9, it was found that the reflective surface of the blade was the primary cause of incorrect categorization. The reflections of the indoor lighting can be seen in Figure 3.9b and appear similar to the simulated cracks on the blade in shape, size, and placement. This led to an increase in false positives, when the model prediction is faulty and the true label is healthy, within the majority of model predictions. Figure 3.9a indicates common false negatives, predicted healthy when the true label is faulty, from the top performing architectures. Similar to the indoor reflections, the sun reflections of the outdoor images obscure the faults in a manner that led the architectures to perceive them as healthy. Another factor included dark cracks which blended in with the dark blue portion of the turbine blade. Here, the theorized cause for this is the difference in lighting conditions of the indoor *vs.* outdoor images. When the blade is not illuminated, the cracks become difficult to discern in the blue region of the blade. This is an important factor to the adoption of the proposed fault analysis method and can be solved for in the following ways. Most wind turbine blades are a solid white color rather than the multicolored one in our dataset, so images like the ones shown in Figure 3.9 would be less often mislabeled as *healthy* in commercial applications.


(a) Samples of false negative images. Boxes indicate fault locations on the images.

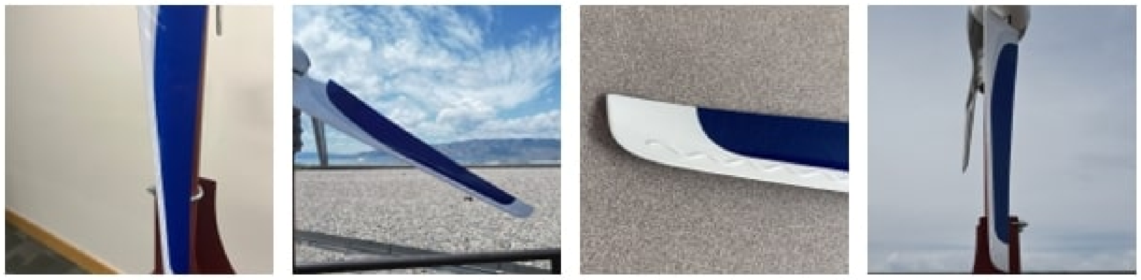

(b) Samples of false positive images.

Fig. 3.9: Sample of commonly misclassified images using the implemented deep learning architectures.

3.3.2 Image Classification using Large-Scale Wind Turbine Dataset

To allow for a fair comparison, each model performed hyper-parameter tuning on this new dataset consistent with the simulations conducted on the CAI-SWTB dataset using the search space defined in Table 3.3. The resulting tuned hyper-parameters are illustrated in Table 3.10. Since many of the cracks in these images appear near the image border, image augmentation parameters for models using this dataset had to be modified for the augmented images to ensure the existence of the blade cracks. For example, if an image contained a crack near the edge, and the image was stretched, zoomed, and/or rotated, the crack in the image would likely be moved out of the image window. However, horizontal flipping maintains the image contents while still providing more variations to help reduce over-fitting and result in better generalization. Thus, in order to maintain the intended features of the dataset, the augmentation parameters described in Table 3.9 were applied.

Table 3.9: Image Augmentation Parameters for Large-Scale Wind Turbine Dataset Tests.

Augmentation Parameters	***Setting***
Shear Range	0.0
Zoom Range	0.0
Rotation Range	0.0
Horizontal Flip	True
Fill Mode	-

Table 3.10: Best Model Parameters on Large-Scale Turbine Data. AF and LR Denote the Activation Function and Learning Rate, Respectively.

Architecture	**Epochs**	**Batch Size**	**Optimizer**	**AF**	**LR**	**Dropout**	**Pooling**
(TL)-Xception	50	32	RMSprop	Sigmoid	0.001	0.6	Avg
(TL)-ResNet50	50	16	Adam	Sigmoid	0.001	0.2	Avg
Custom CNN	100	16	Nadam	Sigmoid	0.001	0.2	Max

The evaluation of performance for the investigated models on this dataset is illustrated in Table 3.11. Here, the best accuracy obtained was 100% with transfer learning Xception. This was followed by the other transfer learning variant in this study, (TL)-ResNet-50, obtaining 93.27% accuracy. The non-transfer learning model, the Custom CNN, was only able to provide at its best a test accuracy 80.77%. When comparing the custom CNN model in comparison to other UVU students' base-models, the smaller dataset and limited features represented proved to be beneficial for the lower layer count. The limited number of images and inability to apply image augmentation caused their deeper models to over-fit quickly and fail to generalize the fault features. This over-fitting trend can also be seen in the loss *vs* epoch plots for the custom CNN (see Figure 3.12). Here, the validation loss becomes divergent while the training loss continues a downward trend indicating poor generalization on unseen data. To solve this issue, more images would be required to bolster the image count for the dataset while introducing variance to the representation presented to the model. However, with the inherent challenges of this limited dataset, the performance of the transfer learning architectures show promising results. Here, the leveraging of pre-trained model weights from the ImageNet dataset with additional fine-tuning allow for convergence and provide exceptional accuracy on this dataset. Although the TL-Xception and TL-ResNet50 models are deeper than the Custom CNN model, because of their pretraining, they were able to obtain higher test accuracies. When observing the TL-ResNet-50 validation accuracy (see Figure 3.11), once the model's training reached about 42 epochs, the validation accuracy began to climb dramatically. Because the validation accuracy continued to climb without plateauing, it is likely that training for more than 50 epochs would have yielded a higher test accuracy.

Table 3.11: Compared Model Performance on Large-Scale Turbine Data.

Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-Xception	**1.0000**	**1.0000**	**1.0000**	**1.0000**
(TL)-ResNet-50	0.9327	0.8814	1.0000	0.9369
Custom CNN	0.8077	0.8200	0.7885	0.8039

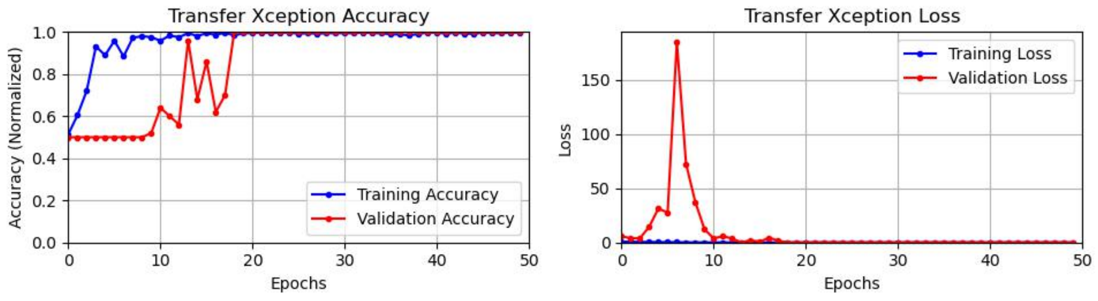


Fig. 3.10: Best run results from the proposed transfer Learning-based Xception, i,e., (TL)-Xception on large-scale turbine data.

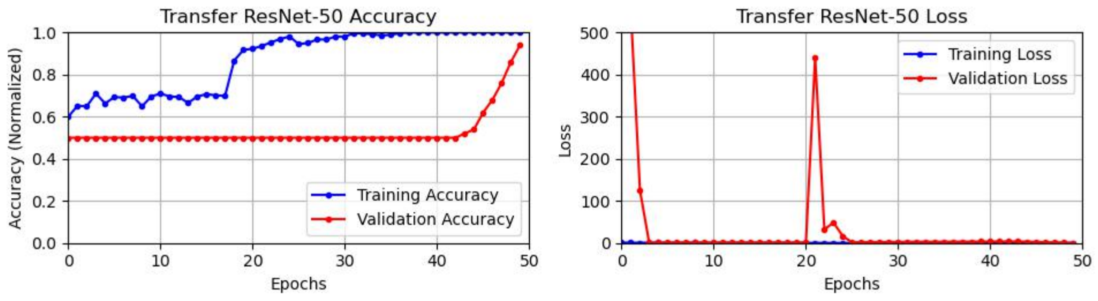


Fig. 3.11: Best run results from the proposed Transfer Learning-based ResNet-50, i.e., (TL)-ResNet-50 on large-scale turbine data.

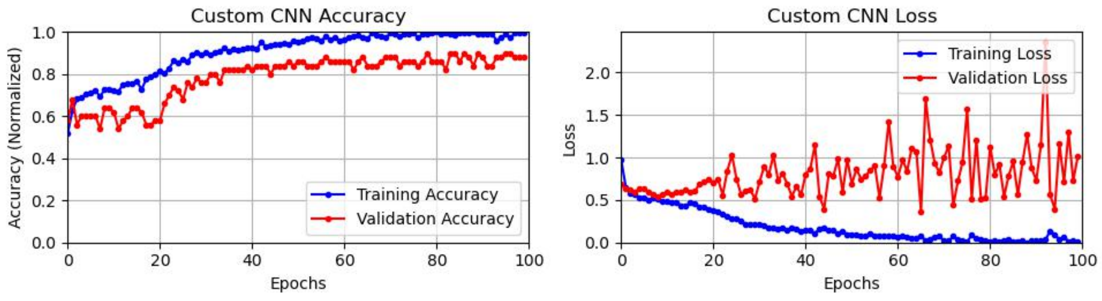


Fig. 3.12: Best run results from Custom CNN on large-scale turbine data.

CHAPTER 4
Deblurring/Image Processing

4.1 Rotating Wind Turbine Blade Images/Data Collection

To test the velocity-based Richardson-Lucy and Wiener deconvolution algorithms on real images with real motion blur, images of rotating wind turbine blades with motion blur were required. Data such as the distance from the camera to the wind turbine blade, the rpm of the blade, the number of centimeters per pixel along the blade in the image, the horizontal and vertical distances from the drone camera to the center of the turbine hub, and the camera shutter time were all required for accurate deblurring and PSF calculation. The images and data required for this portion of the project were provided by members of UVU's machine learning and drone lab.

Many different types of images and videos were taken from different angles and distances, but it was ultimately decided that the drone camera had to be very near to wind turbine, within one or two feet, for blade faults to appear large enough in the images. Because the wind turbine is so small, from a distance, small cracks are unnoticeable. This is true even with the turbine is not rotating. It was also decided that the best camera angle is when the camera is facing the front or back of the turbine blades head on. For reasons later discussed, blurry images of a turbine with the drone camera at an angle are what had to be used for this research. An example image taken of the small wind turbine from a distance is shown in Figure 4.1, and an example of an image taken of the small wind turbine up close is shown in Figure 4.2. A cropped section of the image in Figure 4.2 is what was used for image deblurring.

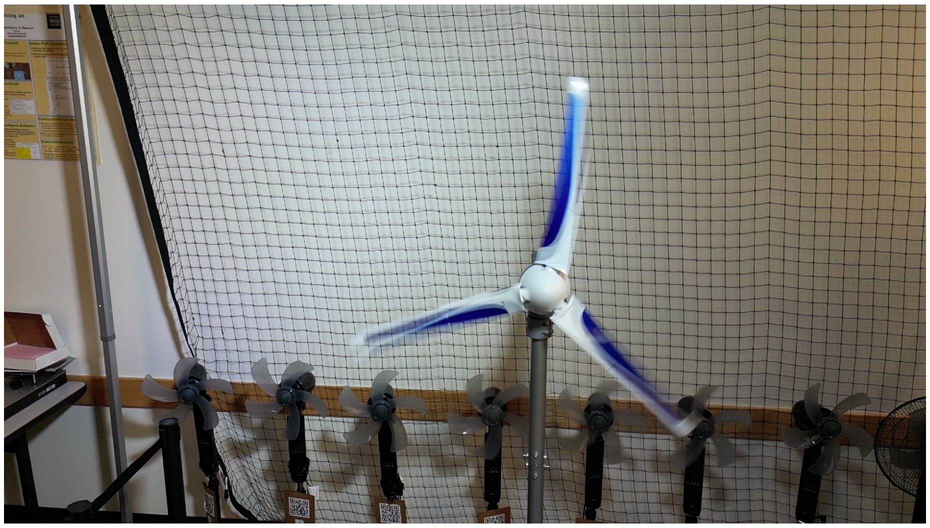


Fig. 4.1: Rotating turbine blade with camera distant from the turbine.

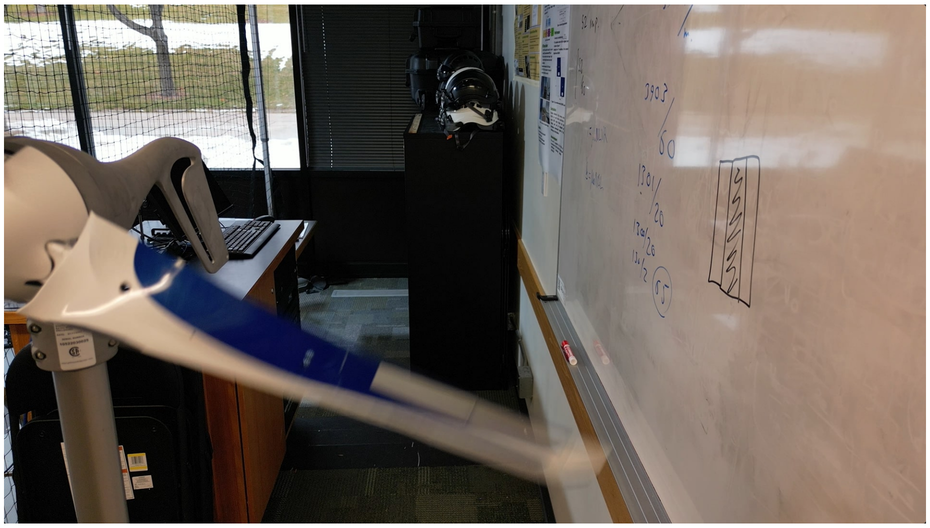


Fig. 4.2: Rotating turbine blade with camera near the turbine.

4.2 Richardson-Lucy Algorithm Development

The goal of this portion of the research is to deblur a close up image of a rotating wind turbine blade. This image would have real motion blur that may not be uniformly linear across the entire blade, but it would be close enough that linear deblurring methods could be applied. The background of these images would have no blur, only the blade. To develop an algorithm which could deblur a real image of an object in linear motion, the first step was to perform deblurring in a controlled, artificial environment. Once this was completed, deblurring could be applied to images of blades with real motion blur.

4.2.1 Classic RL-Algorithm Code

Code for the Richardson-Lucy algorithm devised for this research is listed in Section 7.0.4 [36]. The algorithm listed was developed by following (2.1), which comes directly from the mainstream classic Richardson-Lucy algorithm method [42, 43]. This code is able to apply the RL-algorithm to both grayscale and RGB images. The main difference between deconvolving these two types of images is that with RGB images, extra care must be taken to perform the deconvolution on each of the color channels separately, and then they must be correctly combined back together to create the overall deblurred image. A square PSF kernel with odd dimensions and smaller dimensions than the blurry image is recommended for the best performance. This custom RL algorithm was tested against python's Skimage built-in RL algorithm function, and the same results were achieved.

4.2.2 Artificial Image

An artificial image was needed so that artificial blurring and then deblurring could be applied. A square, gray-scale image of a white turbine blade was made in Microsoft Paint for this purpose. The image was chosen to be gray-scale rather than RGB to simplify the blurring and deblurring for this first step. Two black holes were made on the turbine so it would be easy to show smearing of the dot after blurring, and the intactness of the dot after deblurring. This artificial image is shown in Figure 4.3.

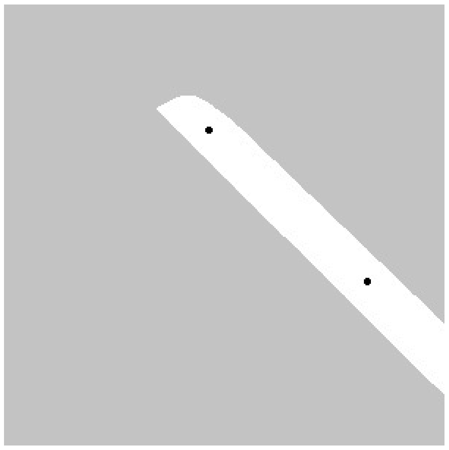


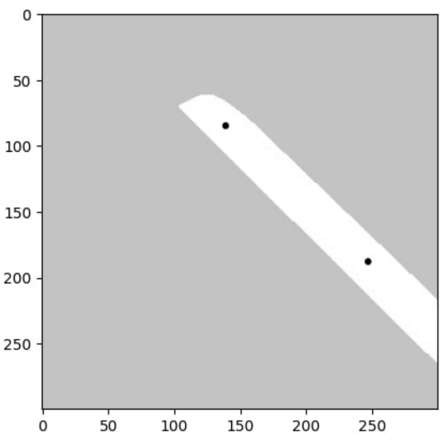
Fig. 4.3: Artificial wind turbine blade image.

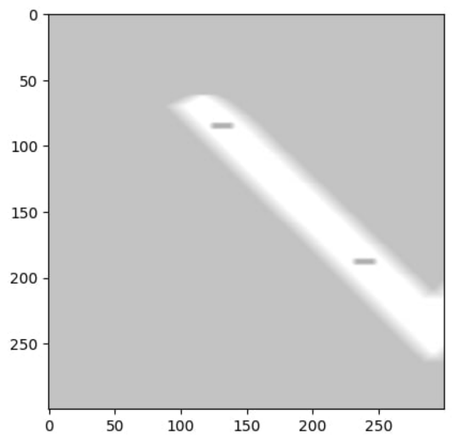
4.2.3 Artificial Blurring and RL Deblurring

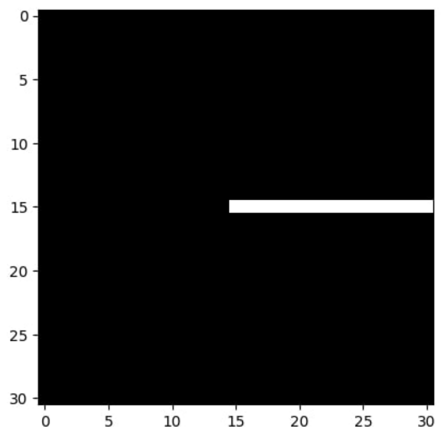
To first test the concept of applying the RL-algorithm to images of wind turbine blades, artificial linear blurs with known PSFs were applied to artificial grayscale images. To apply a blur to an image, the image must be convolved with the PSF as in (4.1), where u represents the original sharp, non-blurred image,

$$d = P \otimes u. \tag{4.1}$$

The original image u is padded appropriately so that d has the same dimensions as u. In this research, all images are 300x300 pixels to match the image sizing used to train and test the ML algorithms used by the UVU Machine Learning and Drone Lab [12]. The PSF dimensions and image dimensions are square to simplify the complexity of 2D convolutions and image resizing. Figures 4.4a, 4.5a, and 4.6a show the same original artificial image of a turbine blade with two black holes. Figures 4.4b, 4.5b, and 4.6b show the artificially blurred blade image after applying the respective PSF kernels illustrated in Figures 4.4c, 4.5c, and 4.6c. Figures 4.4d, 4.5d, and 4.6d demonstrate the outputs of the custom RL algorithm after 25 iterations on the blurred images in Figures 4.4b, 4.5b, and 4.6b.


(a) Original artificial blade image.


(b) Horizontally blurred artificial blade image.


(c) Horizontal PSF kernel.

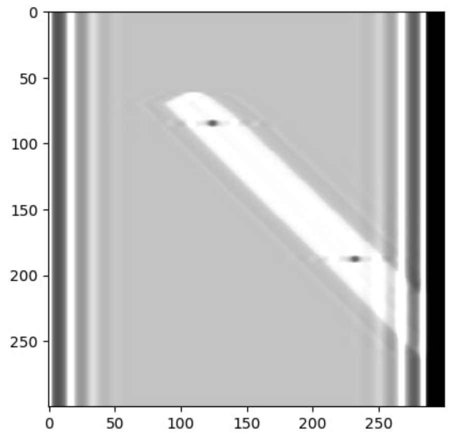
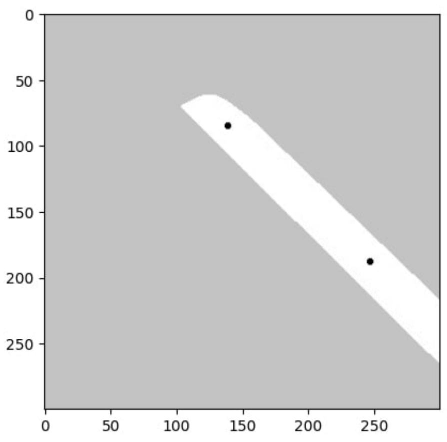
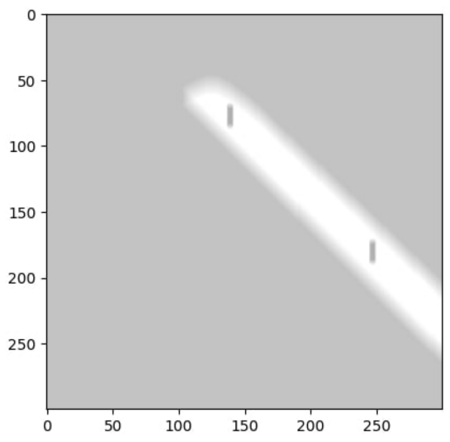

(d) RL deblurring: 25 iterations.

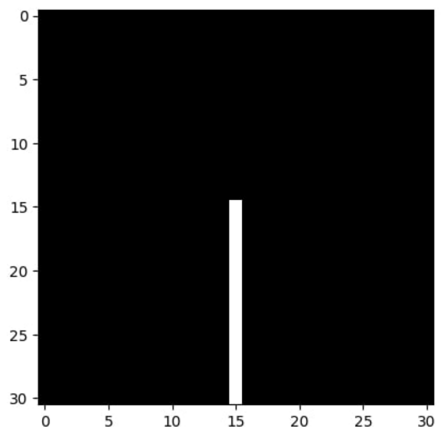
Fig. 4.4: Blurring and RL deblurring artificial grayscale image using horizontal PSF kernel.



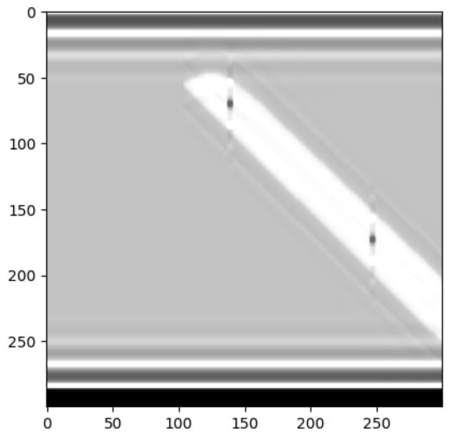
(a) Original artificial blade image.



(b) Vertically blurred artificial blade image.

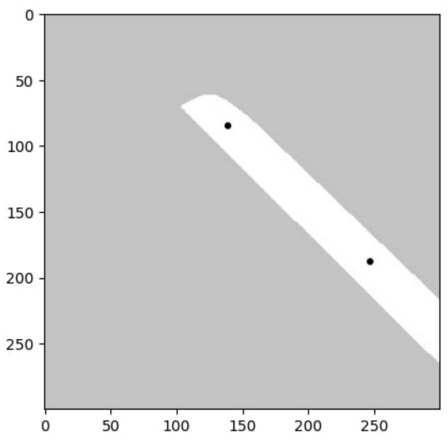


(c) Vertical PSF kernel.

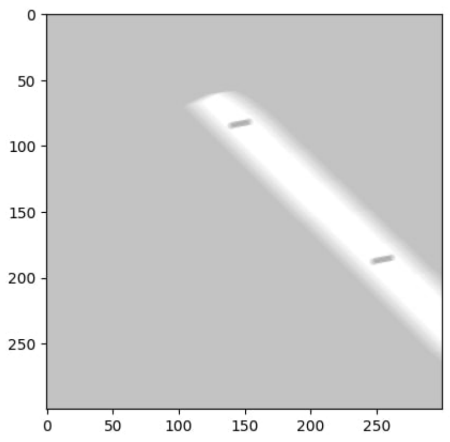


(d) RL deblurring: 25 iterations.

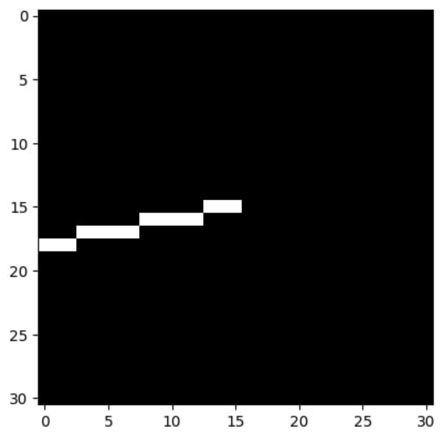
Fig. 4.5: Blurring and RL deblurring artificial grayscale image using vertical PSF kernel.



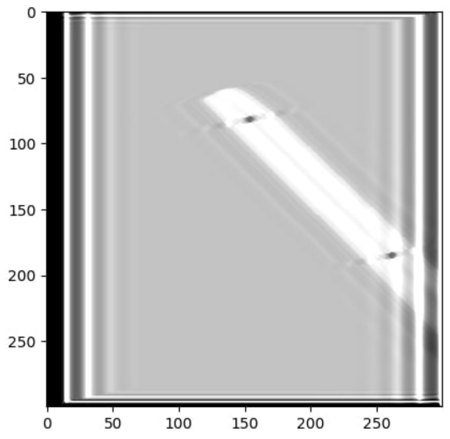
(a) Original artificial blade image.



(b) Diagonally blurred artificial blade image.



(c) Diagonal PSF kernel.



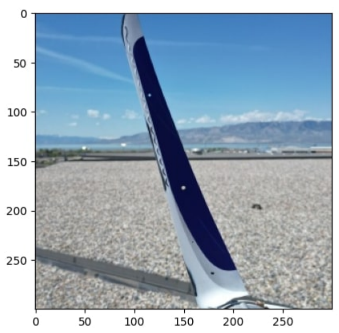
(d) RL deblurring: 25 iterations.

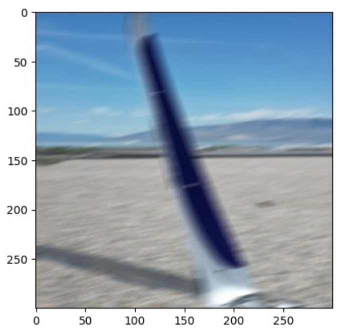
Fig. 4.6: Blurring and RL deblurring artificial grayscale image using diagonal PSF kernel.

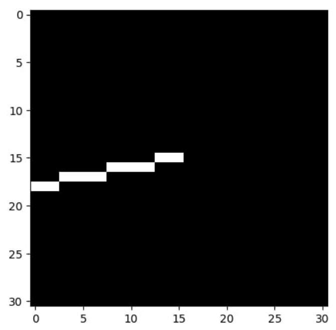
For deblurring the gray-scale images, 25 iterations of the RL-algorithm was chosen because it resulted in consistently sharp images with minimal distortions for this specific case. After deblurring, the artificial holes, which are shown as dots in the images, go from being smeared to being intact as dots again. As far as the PSF kernel is concerned, if the white line in the kernel is flipped horizontally and vertically, that new flipped line from the origin outward is the direction of the blur applied to the original image. Basically, the PSF kernel line is perpendicular to the direction of the blur. If the white kernel lines had run all the way across the kernel through the center of the kernel, then the images would be blurred in both directions, so this is why the line only stretches from the origin out to the edge in these tests. The number of pixels that the PSF kernel line spans is the number of pixels that the blurred image gets smeared by, and it is the number of pixels that the new image gets deblurred by. The size of the kernel does not have much effect on the resulting blurred and deblurred image other than the time it takes to complete the convolution, but the kernel size just needs to be large enough to fit desired PSF kernel line pixel length. The larger the length of the PSF kernel line, the larger the blur in the resulting blurred image. The opposite is also true, so the smaller the PSF kernel line the smaller the applied blur. The same kernel size of 31×31 is used in these tests for better comparison between the blurred images. Each PSF kernel was created using the line function from the cv2 python library.

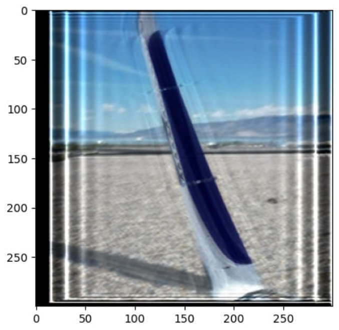
There can be seen in the deblurred images a significant ringing distortion along the edges of the image, and a smaller ringing along the blade and the holes in the blade. This ringing is a common side effect of using the RL-algorithm, but since the cracks or other damages are likely to be in the center of the wind turbine blade section image, the more significant ringing along the edges of the images should not impact the visibility of the desired cracked sections. The amount of horizontal blur in a blurry image corresponds to the amount of ringing on the top and bottom of the deblurred image, as shown in Figure 4.4d, and the amount of vertical blur corresponds to the amount of ringing on the left and right of the deblurred image, as shown in Figure 4.5d. A combination of ringing can be seen in the deblurred image in Figure 4.6d. The vertical and horizontal components of the PSF kernel line roughly correspond to how many pixels into the deblurred image is turned black along the edge. The black portions along the edge are in part due to the zero-padding of the blurry image that happens right before convolution in the algorithm. This zero-padding is done to prevent wraparound in the image from the left edge to the right, top edge to the bottom, or vice-versa. This means that after the image is resized, the zero-padded sections, which are black, bleed into the image instead.

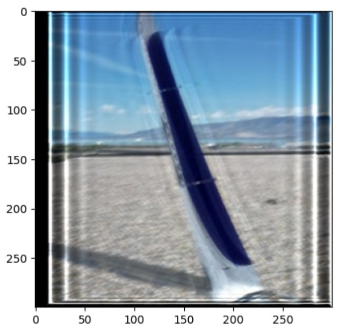
After the concept was well-defined for grayscale artificial images, artificial blurring and deblurring were applied to a real, RGB image containing holes from the *Combined Aerial* *and Indoor Small Wind Turbine Blade* (CAI-SWTB) data set. The results of blurring and deblurring this image are shown in Figure 4.7. Five different iteration amounts were applied to the deblurring to show how additional iterations affect the final deblurred image. As in the grayscale images, the holes in the blade become visible and recovered after deblurring the RGB image. Because this image has a lot more detail than the artificial grayscale images, the ringing around the blade is more apparent, but the quality of the deblurred blade image is still remarkably better than the blurred version. More iterations, as long as the PSF is correct, means a sharper image but more distortions around and inside the image.

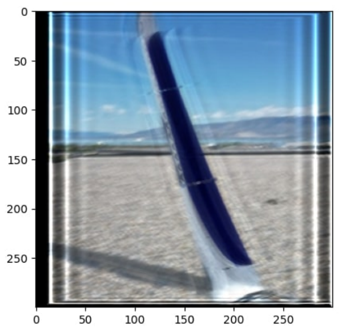

(a) Original real blade image.

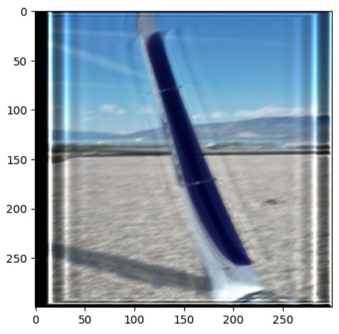

(b) Diagonally blurred real blade image.


(c) Diagonal PSF kernel.


(d) RL deblurring: 50 iterations.


(e) RL deblurring: 30 iterations.


(f) RL deblurring: 25 iterations.


(g) RL deblurring: 20 iterations.

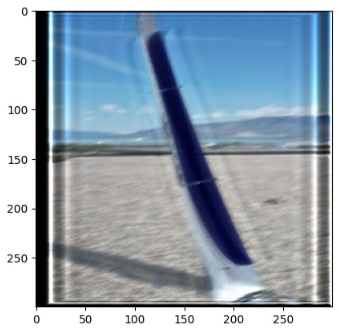

(h) RL deblurring: 15 iterations.

Fig. 4.7: Blurring and RL deblurring real RGB image using diagonal PSF kernel.

4.3 Wiener Deconvolution Algorithm Development

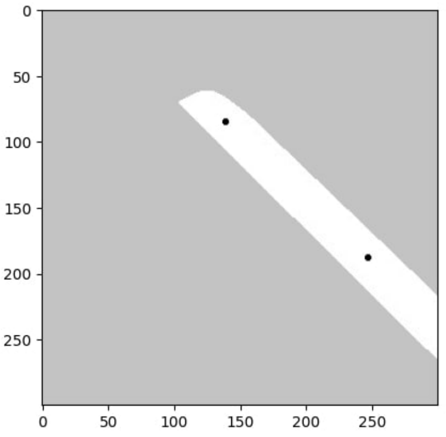
The Wiener deconvolution method was developed after the RL method could be used to successfully blur and deblur images, so similar steps were taken to develop a python implementation for Wiener deconvolution. First the Wiener deblurring was tested on artificial blade images, then on artificially blurred blade images, then finally on real rotating turbine blade images.

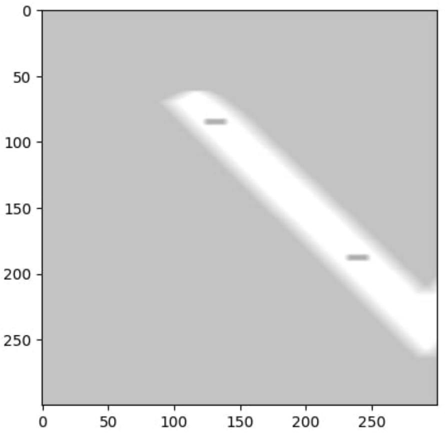
4.3.1 Wiener Deconvolution Code

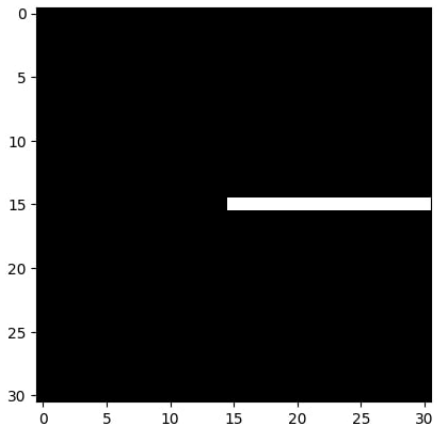
Code for the python implementation of Wiener Deconvolution devised for this research is listed in Section 7.0.5. Supporting functions found on github for this code are listed in Section 7.0.6. This implementation of Wiener deconvolution is based off of Matlab's *deconvwnr* function, (2.4), and (2.5) found in Matlab's *deconvwnr* documentation. To fully understand Wiener deconvolution, first images were blurred and deblurred in Matlab. Once this was a success, the python version was developed. The code in Section 7.0.5 is able to perform wiener deconvolution on both three-channel RGB images and single-channel grayscale images. Just as with the RL algorithm, a square PSF kernel with odd dimensions and smaller dimensions than the blurry image is recommended for the best performance.

4.3.2 Artificial Blurring and Wiener Deblurring

To gain a full understanding of how Wiener Deconvolution worked, the same artificially created and artificially blurred images used to test the RL algorithm were used to test the Wiener Deconvolution algorithm. Wiener deblurring on artificailly blurred grayscale images is shown in Figures 4.8, 4.9, and 4.10. For each grayscale case, $NSR = 5$ was chosen because it seemed to effectively deblur the images while also limiting the amount of distortions caused by the deblurring. Wiener deblurring on an artificially blurred RGB image is shown in Figure 4.11. Five different NSR values were applied to the deblurring to show the deblurring effects of changing the NSR. As can be seen in Figure 4.11, a smaller NSR results in a sharper deblurred image but with more distortions. This is only true though when the PSF perfectly matches the expected PSF in the blurry image. As will be seen when applying Wiener deconvolution to real rotating turbine blades where the PSF is not perfect, having too small of an NSR value will result in distortions so bad that the image looks like a television with no signal. For this reason, an NSR of around five is likely to provide a good middle ground of effective deblurring with minimal distortions. The black deblurring distortions of the Wiener filter are actually quite similar to that of the Richardson-Lucy algorithm. This is due to the zero-padding of the blurry image before convolution.


(a) Original artificial blade image.


(b) Horizontally blurred artificial blade image.


(c) Horizontal PSF kernel.

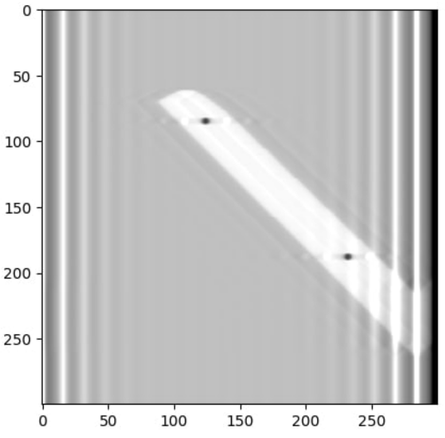
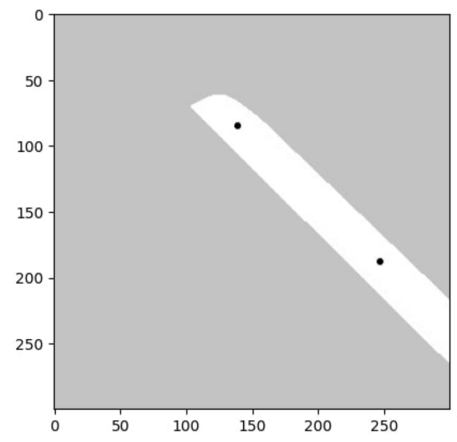
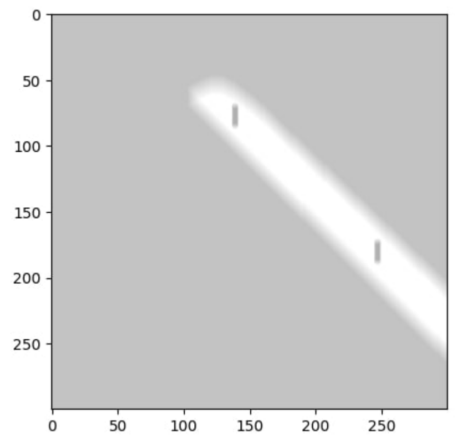
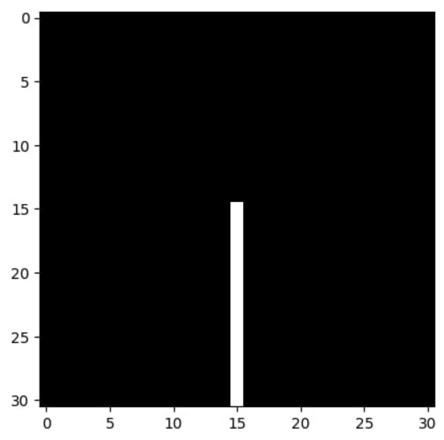

(d) Wiener deblurring: NSR = 5.

Fig. 4.8: Blurring and Wiener deblurring artificial grayscale image using horizontal PSF kernel.


(a) Original artificial blade image.


(b) Vertically blurred artificial blade image.


(c) Vertical PSF kernel.

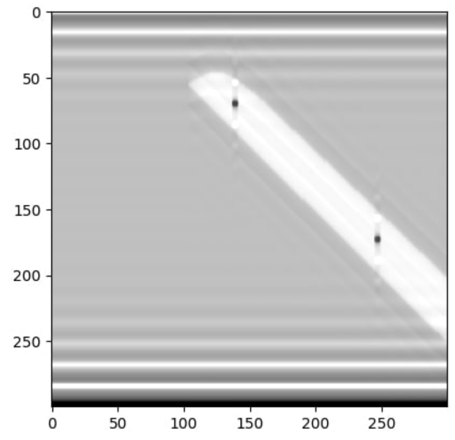
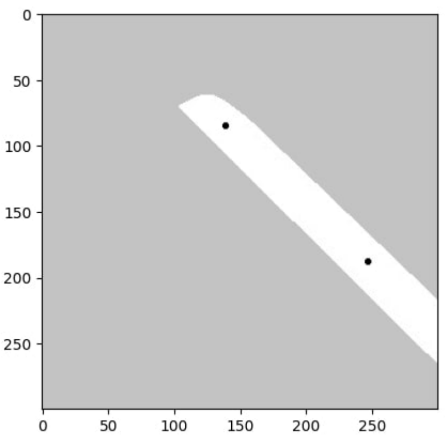
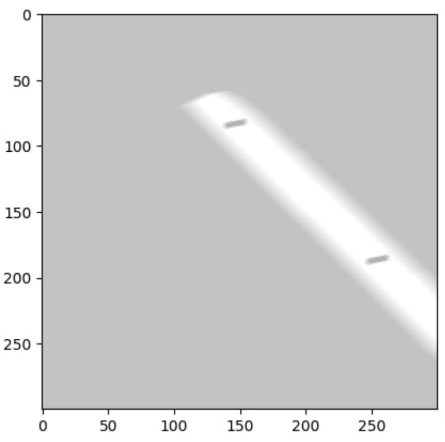
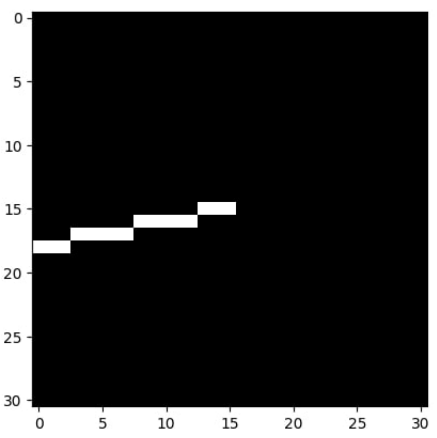

(d) Wiener deblurring: NSR = 5.

Fig. 4.9: Blurring and Wiener deblurring artificial grayscale image using vertical PSF kernel.


(a) Original artificial blade image.


(b) Diagonally blurred artificial blade image.


(c) Diagonal PSF kernel.

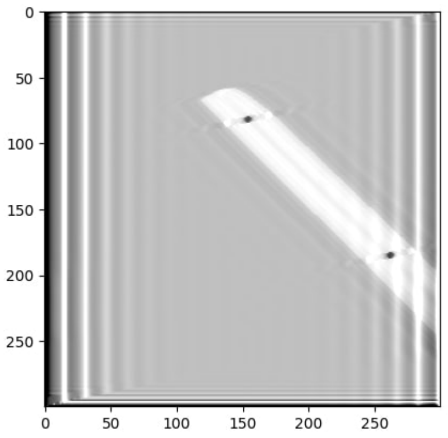
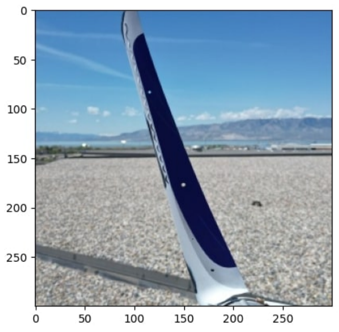
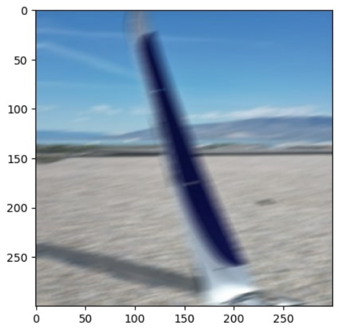
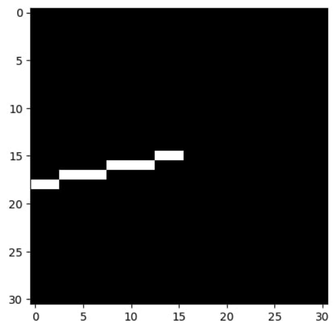
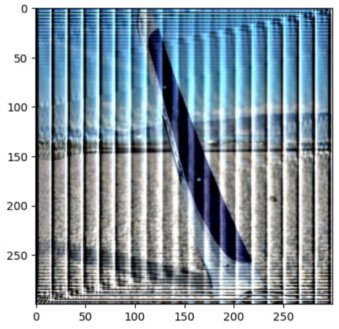

(d) Wiener deblurring: NSR = 5.

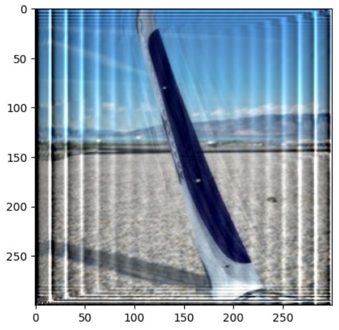
Fig. 4.10: Blurring and Wiener deblurring artificial grayscale image using diagonal PSF kernel.

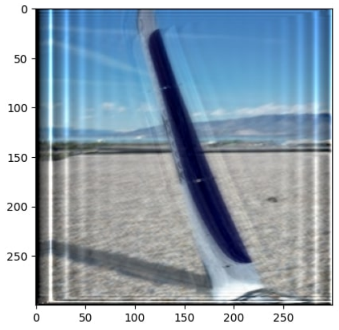

(a) Original real blade image.

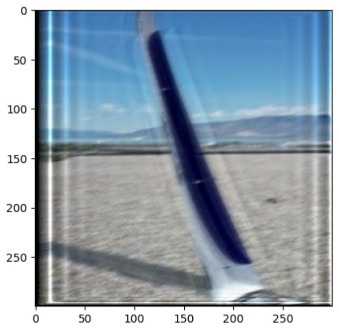

(b) Diagonally blurred real blade image.


(c) Diagonal PSF kernel.


(d) Wiener deblurring: NSR = 0.1.


(e) Wiener deblurring: NSR = 1.


(f) Wiener deblurring: NSR = 5.


(g) Wiener deblurring: NSR = 10.

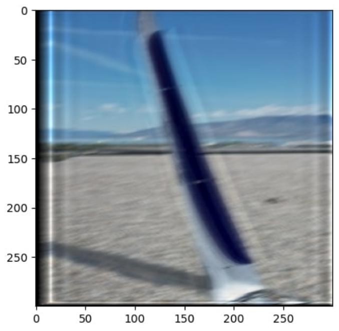
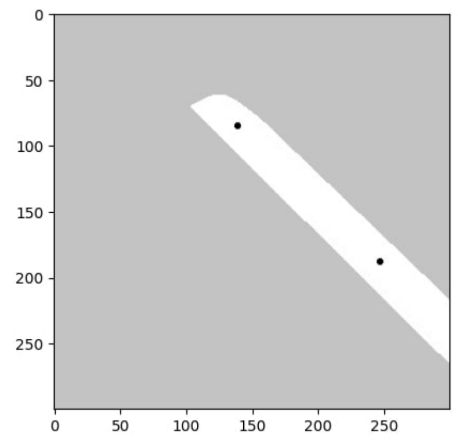
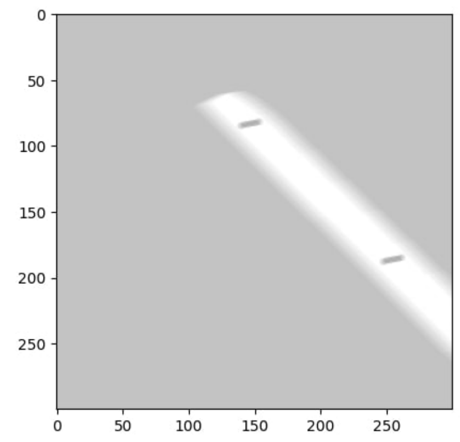

(h) Wiener deblurring: NSR = 25.

Fig. 4.11: Blurring and Wiener deblurring real RGB image using diagonal PSF kernel.

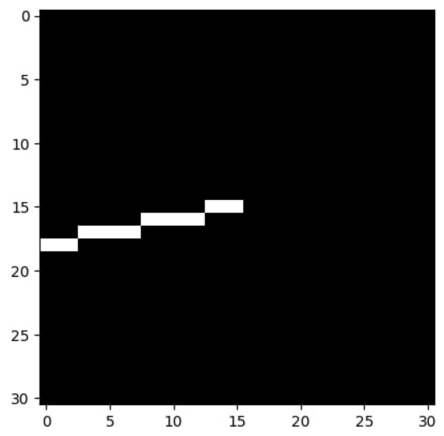
One thing to note is that before the blurry images are sent through the Wiener deblurring algorithm, they must be zero-padded to avoid wraparound in the image. Then at the end of the algorithm, they must be resized appropriately to the original image size. Like the RL-algorithm, after the image is resized, the zero-padded sections, which are black, bleed into the image. Zero-padding was done in the RL algorithm before convolution, but since in the Wiener algorithm convolution is done via multiplication in the frequency domain, zero-padding must be done before converting the image to the frequency domain and multiplication. Although zero-padding does cause wraparound, it greatly lessens the distortions that occur around the edges of the deblurred image. An example of a deblurred image with wraparound that was not zero-padded beforehand is shown in Figure 4.12d.



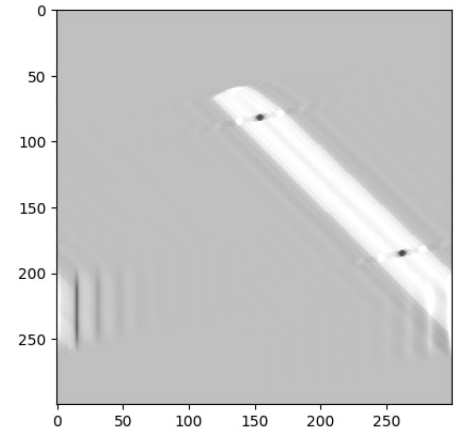
(a) Original artificial blade image.



(b) Diagonally blurred artificial blade image.



(c) Diagonal PSF kernel.



(d) Wiener deblurring no zero-padding: NSR = 5.

Fig. 4.12: Blurring and Wiener deblurring artificial grayscale image with no zero-padding, causing wraparound.

4.4 Velocity-Based PSF Estimation

Now that the basic idea of how the RL and Wiener Deconvolution algorithms work with artificial blurring and deblurring has been established, an approach where the PSF kernel needs to be estimated must be discussed. The direction and length of the PSF kernel line can be determined based off of the rotational velocity of the turbine blade, camera shutter time, and the radius from the center of the wind turbine hub to the damaged blade section in the observed image. This radius can be calculated from drone data which tracks the position of the drone camera relative to the wind turbine hub.

As stated previously, the direction of the PSF kernel line is perpendicular to the line between the relative position of the blade image and the center of the blade's hub. Because in the images for these tests the y-axis goes from the top down and not down up, this has to be accounted for in the calculated the perpendicular line. The size of the PSF kernel line, which affects the amount of pixels to blur or deblur, depends on both the camera shutter time and how the rotational speed of the overall turbine converts to the linear speed for the specific section of the blade in the image. The longer the shutter time, the longer light can enter into the camera lens, so the larger the smear that will appear on a moving wind turbine blade. Also, the faster the rotation of the turbine blades and the farther along the blade the observed section is, the higher the linear velocity of that section is and thus the more smearing that occurs in the blurry image. These relationships are described as follows:

$$\theta_{blade} = \tan^{-1}(\frac{y_{pos}}{x_{pos}}) \tag{4.2}$$

$$r = \sqrt{(x_{pos})^2 + (y_{pos})^2} \tag{4.3}$$

$$w = \frac{RPM(2\pi)}{60} \tag{4.4}$$

$$v = w(r) \tag{4.5}$$

$$bl = \frac{v(T)(ppc)}{2} \tag{4.6}$$

$$x_{bl} = (bl)\cos(\theta_{blade}) \tag{4.7}$$

$$y_{bl} = (bl)\sin(\theta_{blade}) \tag{4.8}$$

$$\begin{cases} CW & x_{line} = -\text{int}(y_{bl}),\ y_{line} = -\text{int}(x_{bl}) \\ CCW & x_{line} = \text{int}(y_{bl}),\ y_{line} = \text{int}(x_{bl}) \end{cases} \tag{4.9}$$

In these equations, θ_{blade} represents the angle from the wind turbine hub center to the blade, x_{pos} and y_{pos} are the horizontal and vertical distances in centimeters from the center to the observed blade section, r is the radius in centimeters from the hub center to the observed blade section, w is the blade's rotational velocity in radians/second of the turbine, RPM is the revolutions per minute of the turbine blade, v is the linear velocity in centimeters/second at the observed blade section, bl is the PSF kernel line blur length in pixels, T is the camera shutter time in seconds, and ppc is the pixels per centimeter ratio calculated from known distances in the image. The distances x_{bl} and y_bl are the x and y distances in pixels of the PSF kernel blur length, and x_{line} and y_{line} are the horizontal and vertical signed distances from the center of the PSF kernel outward, which are passed into the line function from the cv2 python library to create the calculated PSF kernel. The signs of these values are dependent on if the blade is rotating clockwise or counterclockwise, and they also take into account the top-down nature of the y-axis in the tested images.

The value for ppc (pixels per centimeter) could alternatively be calculated by inverting the GSD (ground sampling distance), which has units of centimeters/pixel. The calculation for GSD is

$$GSD = \frac{(H)(Sw)}{(f)(IMw)}. \tag{4.10}$$

In this equation, H is the distance from the camera to desired object (the front of the wind turbine hub/blade) in centimeters, Sw is the sensor width in centimeters, f is the focal length in centimeters, and IMw is the width of the original image in pixels. For this test deblurring an image with the RL-algorithm, the method of knowing distances along the blade within the image and counting pixels to calculate ppc was used, but in future tests ppc will be calculated using (4.11).

$$ppc = \frac{1}{GSD} \tag{4.11}$$

4.5 Rotating Wind Turbine Image Deblurring Results

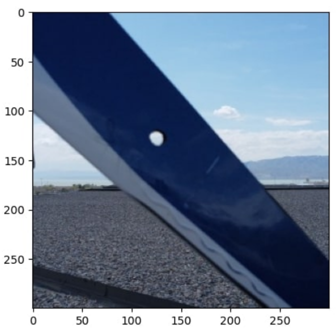
Using the relationships in (4.2)-(4.9) and the measurements in Table 4.1, the values in Table 4.2 were calculated and used to estimate the PSF kernel shown in Figures 4.13c and 4.14c. In Table 4.2, BL_{line} is the final resulting length of the white PSF kernel line used for deblurring after rounding to the nearest pixel. Because for this example the cropped image section shown in Figures 4.13b and 4.14b was not directly at the center of the larger image captured from the drone camera, and the drone camera was pointed at a slight downwards angle, vertical and horizontal distances provided by the drone had to be adjusted for optimal deblurring. Results from RL deblurring are shown in Figures 4.13d-4.13h with different iteration values, and results from Wiener deblurring are shown in Figures 4.14d-4.14h with different NSR values.

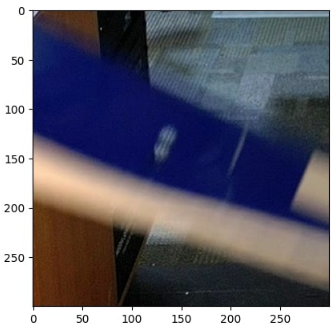
Table 4.1: Real Rotating Wind Turbine Blade Measurements.

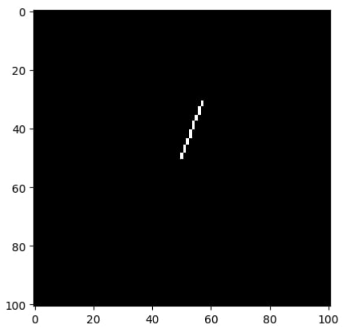
Metric	**Measurement**
Blade Width	5.25 cm
Blade Length	57.5 cm
H (Distance from Front of Hub)	49 cm
T	1/60 s
RPM	32.4 rpm
θ (Blade Angle)	-0.393 rad
x_{pos}	27.7 cm
y_{pos}	-11.5 cm
r	30.0 cm
Rotational Direction	CW

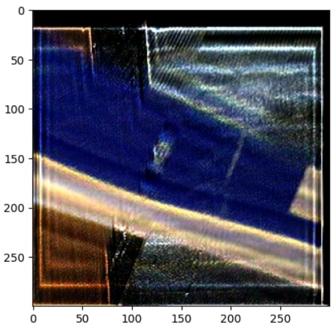
Table 4.2: Real Rotating Wind Turbine Blade Calculated Values.

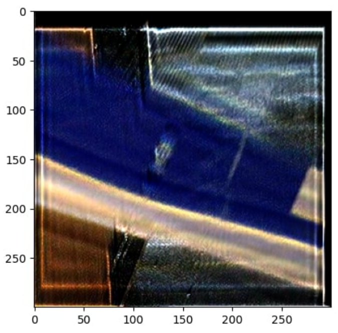
Metric	**Calculated Value**
ppc	24.35 pixels/cm
bl	20.66 pixels
x_{bl}	19.09 pixels
y_{bl}	-7.92 pixels
x_{line}	-19 pixels
y_{line}	7 pixels
bl_{line}	20 pixels

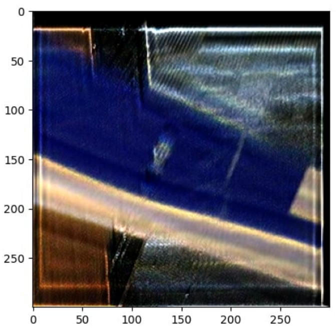

(a) Non-blurred real blade image with different orientation and background.

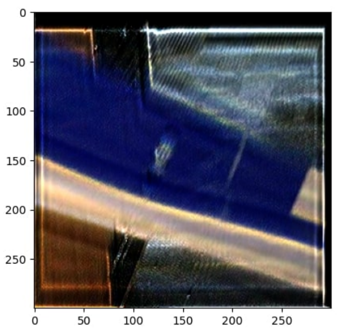

(b) Real blurred rotating blade image.


(c) Velocity-based PSF kernel.


(d) RL deblurring: 50 iterations.


(e) RL deblurring: 30 iterations.


(f) RL deblurring: 25 iterations.


(g) RL deblurring: 20 iterations.

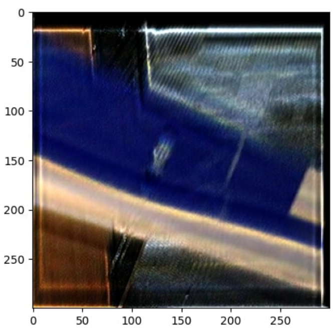
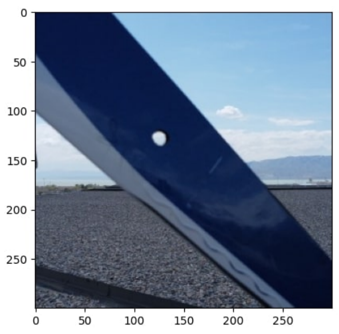
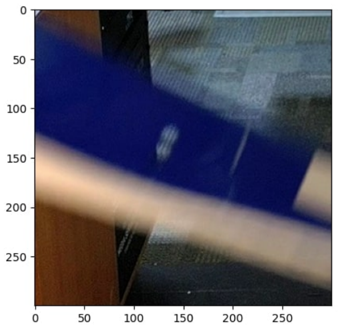
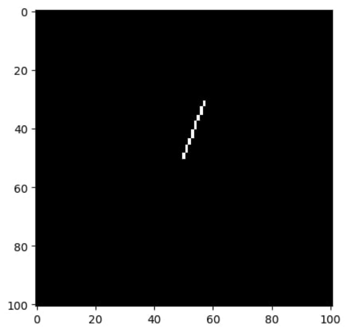
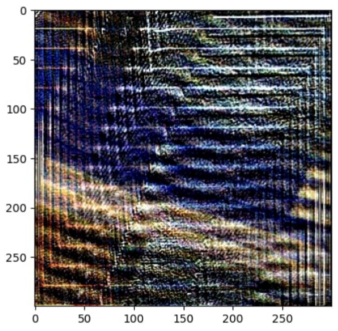

(h) RL deblurring: 15 iterations.

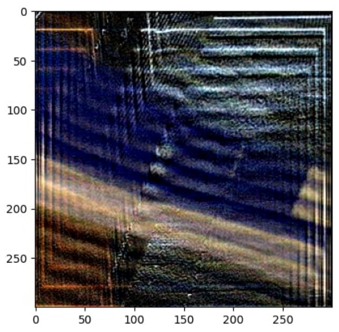
Fig. 4.13: RL deblurring real rotating RGB blade image with calculated velocity-based PSF kernel.

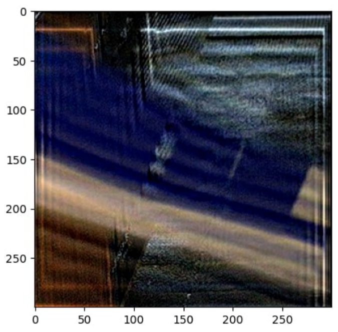

(a) Non-blurred real blade image with different orientation and background.

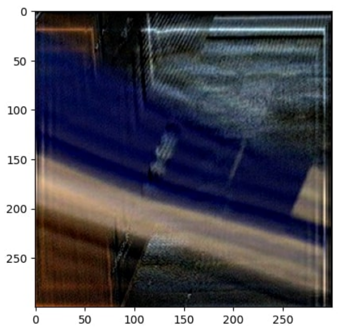

(b) Real blurred rotating blade image.


(c) Velocity-based PSF kernel.


(d) Wiener deblurring: NSR = 0.1.


(e) Wiener deblurring: NSR = 1.


(f) Wiener deblurring: NSR = 5.


(g) Wiener deblurring: NSR = 10.

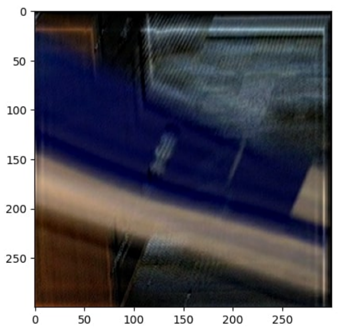

(h) Wiener deblurring: NSR = 25.

Fig. 4.14: Wiener deblurring real rotating RGB blade image with calculated velocity-based PSF kernel.

For the RL deblurring, the reconstructed hole and crack damages in the blade are able to be seen in the deblurred images in Figures 4.13d-4.13h, but the edge-erosion damage has been somewhat erased. This hole, crack, and edge-erosion can clearly be seen in Figures 4.13a and 4.14a. It seems that running the RL algorithm for 25 iterations provides a good balance of sufficient image deblurring with minimal distortions. This image is of the same blade used in these tests but was taken outside at different angles and distances from the camera. While there are some distortions and ringing in the resulting RL deblurred images, most of the blade damages have been clearly reconstructed. Therefore, the PSF estimator generated a kernel that was near the actual PSF kernel. Because there is a slightly different linear velocity along each part of the blade and any additional noise is not being accounted for, no PSF kernel will be perfect for deblurring the entire blurry blade image. A PSF kernel size of 101×101 was used here in order to fit the entire PSF kernel line.

For the Wiener deblurring, the resulting deblurred images in Figures 4.14d-4.14h did not reconstruct the hole and crack quite as well as the RL algorithm, but the results are similar. The edge-erosion was erased in these deblurred images as well. It almost looks like the PSF needed to be adjusted ever so slightly to correctly reconstruct the hole and crack as the RL algorithm did. This is interesting, because for the artificial blurring and deblurring, the RL and Wiener apprpoaches yielded very similar results, but for real image deblurring it seems the two methods start to differ. It is important to remember that this is only a single case, and other images with different PSFs and different amoutns of blur may cause different results. It seems that running the Wiener algorithm with $NSR \geq 5$ provides a good balance of sufficient image deblurring with minimal distortions.

CHAPTER 5
ML Image Classification with Blurry and Deblurred Images

5.1 Re-training ML Models

With the ML classification and deblurring algorithms working on their own, it is time to combine them. The final stage of this research is an attempt to show that deblurring images improves the classification accuracy over blurry images when run through the previously discussed ML models. The idea is that if the ML models can accurately classify deblurred images, then it must mean the deblurring did a good job at reconstructing the wind turbine blade.

Weights from the top performing CNN, ResNet50, and Xception models from chapter 3 were not saved, so simulations for these models had to be rerun in order to save their weights. The purpose of saving the weights from a model is so that these models can perform classification on different test sets without having to be retrained every time. The same hyperparameters which gave the highest test accuracies on the CAI-SWTB dataset shown in Table 3.5 were used for these new simulations. The same augmentation parameters were also used here that were used in previous tests for the CAI-SWTB dataset shown in Table 3.4. Because training and validation images are randomly ordered with each new training of a model, different training accuracies and losses can be obtained while training a model multiple times with the same hyperparameters. CNN, ResNet50, and Xception, were retrained three times, and the highest test accuracy models on the CAI-SWTB dataset out of the three for each architecture had their weights saved. The resulting models have similar results to those in chapter 3. These retrained model performance results are shown in Table 5.1. Their accuracy and loss values during model training are shown in Figures 5.1, 5.2, and 5.3.

Table 5.1: Compared Retrained Model Performance on Combined Indoor and Outdoor Test Data.

Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-Xception	**0.9992**	**1.0000**	**0.9983**	**0.9992**
(TL)-ResNet-50	0.9842	0.9949	0.9733	0.9840
Custom CNN	0.9642	0.9400	0.9917	0.9651

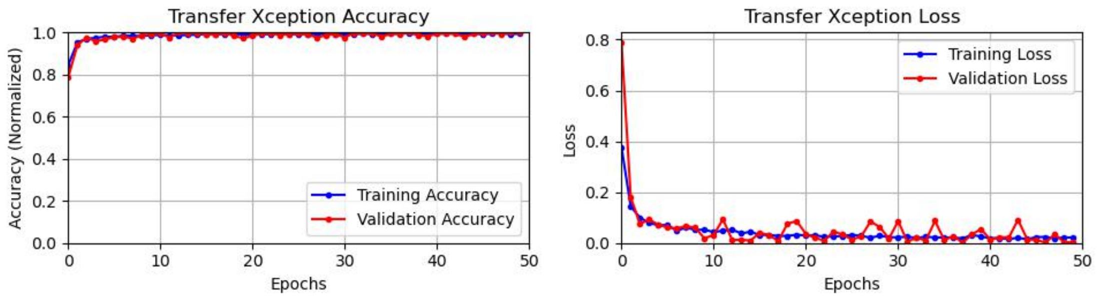


Fig. 5.1: Training results from retrained Transfer Learning-based Xception, i,e., (TL)-Xception.

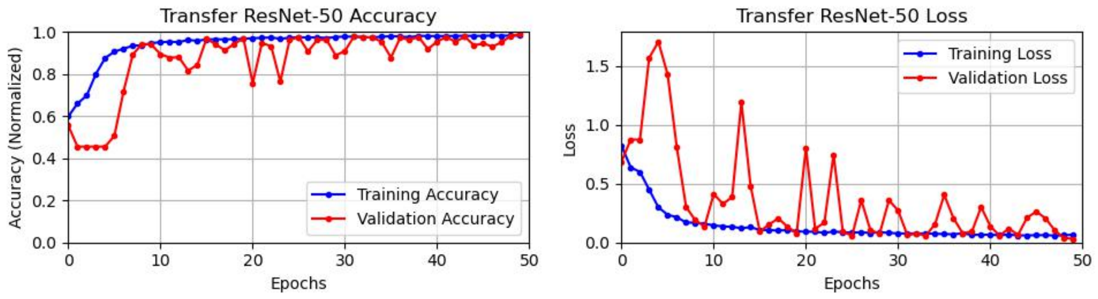


Fig. 5.2: Training results from retrained Transfer Learning-based ResNet-50, i.e., (TL)-ResNet-50.

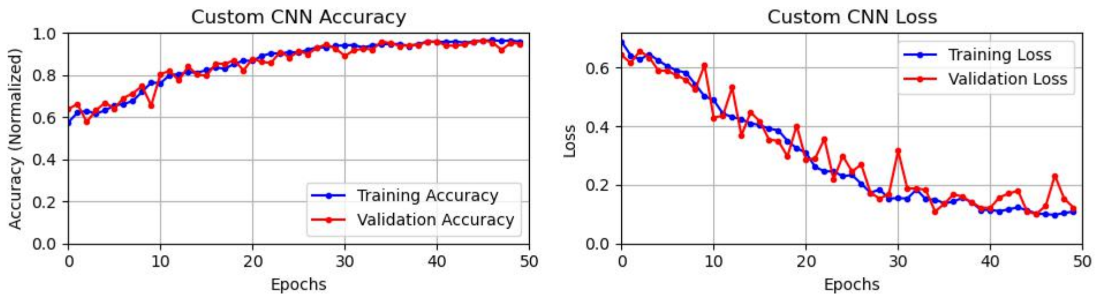


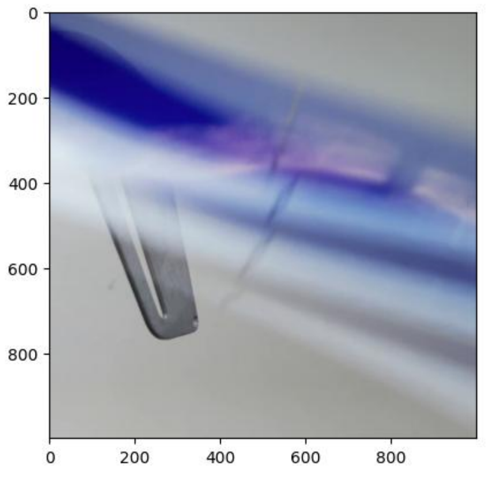
Fig. 5.3: Training results from retrained Custom CNN.

5.2 Real Blurry and Velocity-Based Deblurred Images

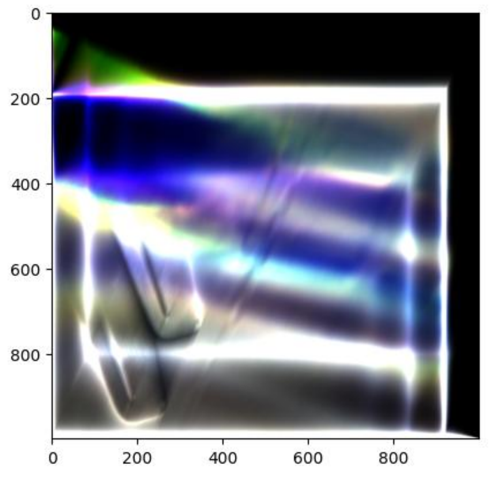
5.2.1 Real Rotating Blade Image Collection

Now that the weights from three high performing ML models for the CAI-SWTB dataset can be loaded for classification on multiple test sets, it is time to try using these models for classifying blurry and deblurred images. For this to happen, a sizable group of blurry-healthy and blurry-faulty blade images needed to be gathered. The only videos and images that were able to be used originally were videos where the drone camera was taking pictures at an angle, as in the real image in chapter 4. These videos were undesirable because this angle made it difficult to use the drone-location data to calculate the velocity-based PSF, so videos with the camera and drone facing the blades straight on was preferred. New videos of a rotating wind turbine were taken by students in UVU's Drone and Machine Learning Lab with the camera facing the blades straight on. These videos were accompanied with drone positioning data. Each video had the drone at a different location relative to the turbine hub, some showed the front of the blades and others the back, and a variety of blades each with different damages were used. The background was also mostly white. This was all in an attempt to make the real rotating blade test data as robust as possible while hopefully looking similar enough to the images in the CAI-SWTB dataset that the ML models were trained on. Frames from these videos were to be taken, and then blade sections were to be cropped out to use in the test set. The drone location was to be adjusted accordingly depending on where the image was cropped. Unfortunately, the turbine blades in these videos are spinning so fast that deblurring using linear deblurring techniques such as the RL and Wiener algorithms became impossible. For reference, the RPM used in the video previously used was 32.4 revolutions per minute, but the RPM in these videos was roughly 108 revolutions per minute, and both videos were shot using a camera with the same shutter time. The cropped images contained so much blur that the PSF line blur length had to be much larger than what had been used in previous deblurring tests. This larger blur length caused such significant distortions in the deblurred images that it ruined any hope for correct classification. The blades were also moving fast enough that the local linear blur became a rotational blur, so no amount of linear deblurring could have deblurred it anyway.

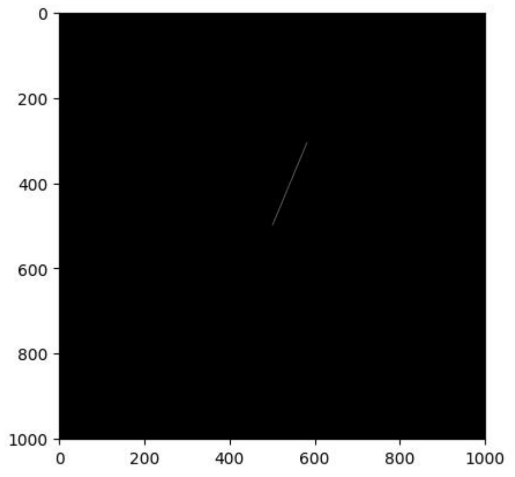
Examples of deblurring these faster RPM turbine blades are shown in Figure 5.4. One item to point out about these images is that they are size 1000x1000 pixels, whereas all the other images previously used in this research which are lower resolution of size 300x300 pixels. This increase in image resolution also proportionally scales the PSF blur line length, but it is all relative, so even if the images in this high-velocity example were lower resolution, similar amounts of distortion would have occurred. The PSF blur length in previous examples ranged from 15 to 21 pixels, but scaling the high speed blur length of 210 pixels shown in Figure 5.4c proportionally by 300/1000 gives a relative blur length of 63 pixels, which is three or four times the length of previously tested blur lengths for 300x300 pixel images. This makes sense, because the RPM in the high-velocity video is slightly more than three times that of the previously used video. The thickness of the kernel line does affect the deblurring, but it was kept at a thickness of 1 for all tests in this research.



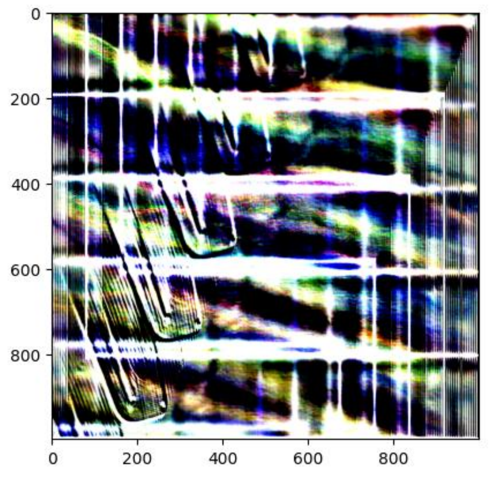
(a) Original high-velocity blurry turbine blade image.



(b) RL deblurring: 25 iterations.



(c) Velocity-based PSF kernel.

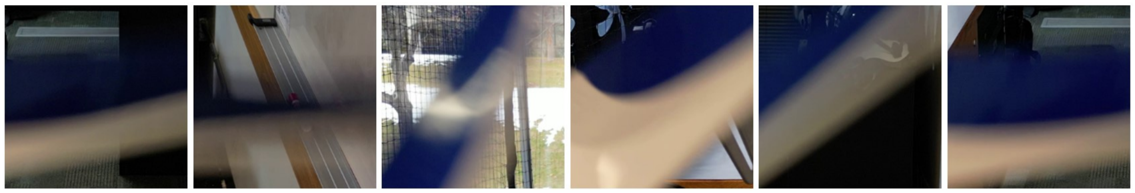


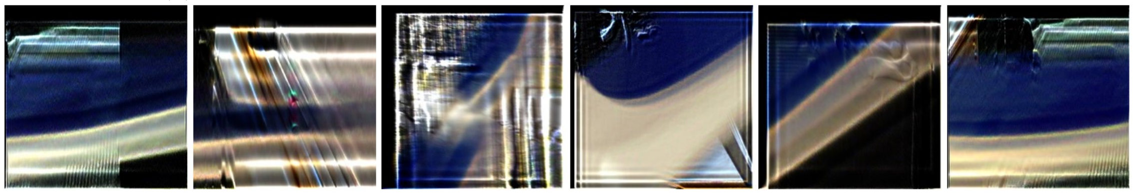
(d) Wiener deblurring: NSR = 5.

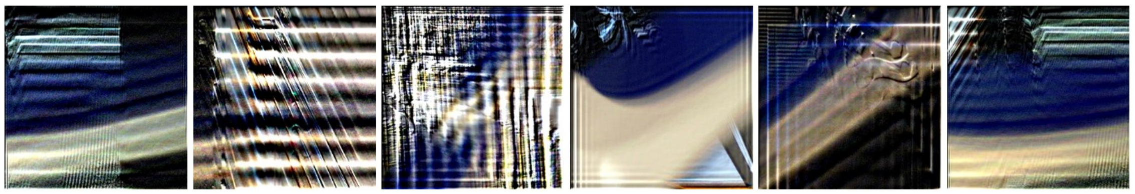
Fig. 5.4: Deblurring real high-velocity rotating RGB blade image with calculated velocity-based PSF kernel.

This limit on what images can and cannot be deblurred is an important discovery, because it gives a scope for what situations this system is practical. If the shutter time had been faster, then a turbine spinning this fast could have been deblurred because the blur length would have been smaller. This situation shows that there is a relationship between the RPM of the blade and the shutter time in regards to how well an image can be deblurred with these algorithms.

Because these faster RPM videos were unusable, two older videos with the camera at an angle had to be used for gathering real rotating turbine blade images. One video contained a wind turbine with three healthy blades, and the other with three faulty blades. Frames from these videos were gathered, and cropped images along different parts of each blade at different blade angles relative to the hub at were taken. Because the camera was at an angle, the positioning of the drone relative to the turbine hub, which was used for the PSF calculation, had to be estimated. This may have caused some inaccuracies when creating the PSFs, but they were likely very close to what they should have been. Where (which pixels) in each frame image that the cropped image was taken from was also taken into account when calculating the PSFs. A total of 50 healthy and 50 faulty blurry images were gathered for the real blurry test set. More images would have been gathered, but just the time it took to crop each image to the correct size, type the image's location, save the image under a specific name, and then calculate a PSF based off of this data was very time-consuming. Examples of these healthy blurry images and their deblurred counterparts are shown in Figure 5.5, and faulty examples are shown in Figure 5.6.


(a) Sample images of healthy blurry blades in an indoor environment.


(b) RL deblurring: 25 iterations.


(c) Wiener deblurring: NSR=5.

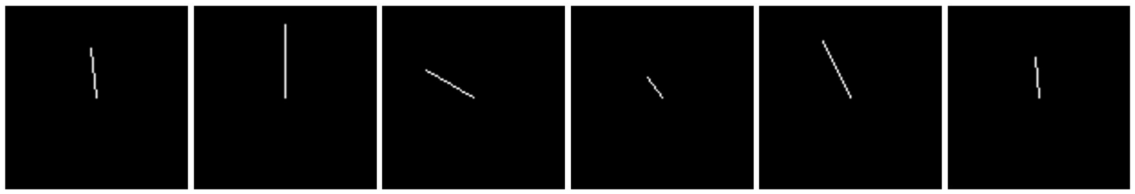
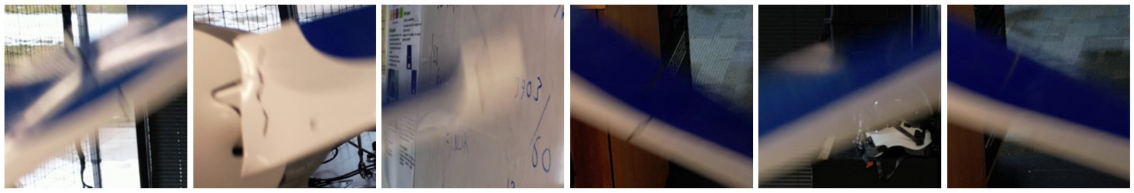
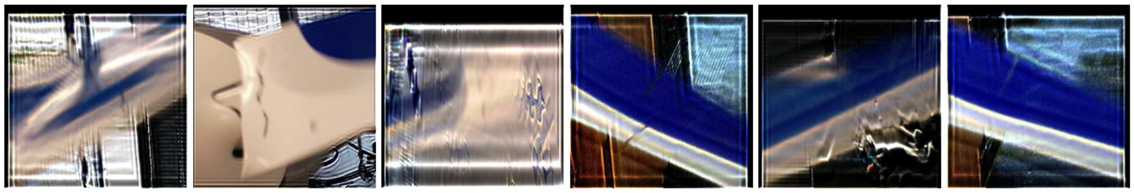

(d) Velocity-based PSF kernels.

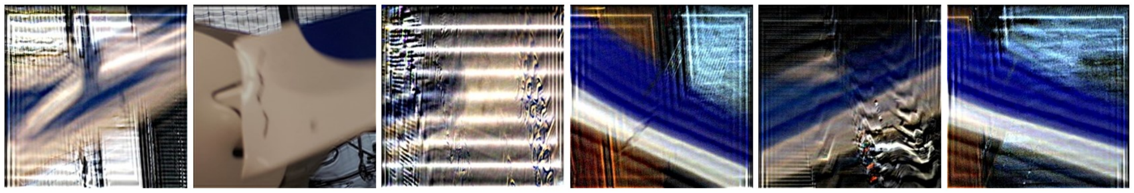
Fig. 5.5: Sample healthy blurry and deblurred blade images of the created real blurry dataset and their corresponding calculated velocity-based PSF kernels.



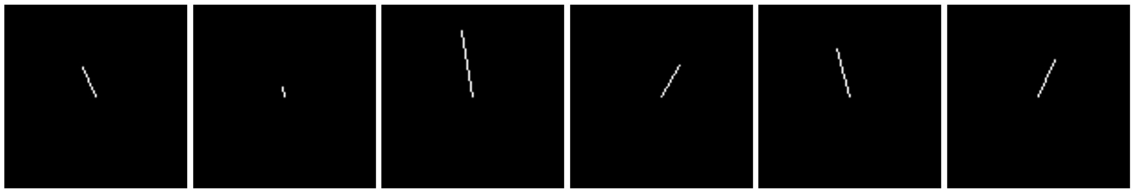
(a) Sample images of faulty blurry blades in an indoor environment.



(b) RL deblurring: 25 iterations.



(c) Wiener deblurring: NSR=5.



(d) Velocity-based PSF kernels.

Fig. 5.6: Sample faulty blurry and deblurred blade images of the created real blurry dataset and their corresponding calculated velocity-based PSF kernels.

The blade images that show blade sections closer to the hub, as in the third image from the right in Figure 5.5 and the second image from the left in Figure 5.6, are more clearly deblurred than the images showing the tip of the blade, as in the third image from the left in Figure 5.6. This is because the linear and rotational velocity that occur at the tip of the blade is faster than that of sections closer to the hub, so a larger PSF blur line had to be used, causing greater distortions to the image. The distorted backgrounds after deblurring, as in the second image from the right in Figure 5.6 with the distorted space helmet, can get in the way of the deblurred blade reconstruction. This is one reason that these videos/images may needed to have been taken in a different setting with a different background better matching that of either the indoor or outdoor CAI-SWTB dataset image backgrounds. With these problems in mind, it is clear from looking at the images that some images seem to be clearly deblurred, and others seem to be in worse shape than their blurry counterparts. With this in mind, the real blurry dataset was deblurred five times with different iteration amounts for the RL algorithm and five times with different NSR values for the Wiener algorithm. This was in an effort to be robust when trying to find a set of hyperameters that would deblur most of the images in the dataset well enough for accurate classification when tested and classified with the ML models. These specific hyperparameters were chosen based off of what seemed to work through trial and error. Some were also chosen to show the extremes of what kind of deblurring and classification happen when the parameters are extremely high or low. The iterations and NSR values tested are shown in Table 5.2.

Table 5.2: Parameters used for RL and Wiener Deblurring.

Number of Iterations (RL Deblurring)	15	20	25	30	50
NSR (Wiener Deblurring)	0.1	1.0	5.0	10.0	25.0

5.2.2 ML Classification on Real Blurry and Velocity-Based Deblurred Images

For comparison purposes, the real blurry turbine blades were classified using the three specified ML models. The results of this classification is shown in Table 5.3. From looking at the table, it is apparent that the models trained on sharp images did not do well classifying real rotating blurry blade images. This is likely do to both blur and possibly these real images having different backgrounds than the backgrounds in the images the models were trained on.

Table 5.3: Compared Retrained Model Performance on Real Blurry Rotating Turbine Blade Test Data.

Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-ResNet-50	**0.5800**	**0.2600**	**0.7222**	**0.3824**
Custom CNN	0.5600	0.4400	0.5790	0.5000
(TL)-Xception	0.4500	0.1400	0.3684	0.2029

Results from deblurring the real blurry images using RL-deblurring and Wiener deblurring are shown in Tables 5.4 and 5.5 respectively. Overall, RL-deblurring seemed to improve the test accuracy over the blurry images. The highest test-accuracy of 0.65 was achieved using transfer learning ResNet-50 and 25 iterations with the RL-algorithm. This was an improvement over the highest blurry image dataset test accuracy of 0.58 also using transfer learning ResNet-50. While for accurate classification test accuracies of 90% or higher are desired, this still shows that RL-deblurring images can improve healthy vs. faulty blade image classification. The results of Wiener deblurring were not as great though, with a highest test accuracy of 0.56 using transfer learning Xception and $NSR = 0.1$ with the Wiener algorithm. From looking at the real blurry images that were deblurred using an NSR of 0.1, most of the images were so distorted and deep-fried that one could not even tell there was supposed to be a turbine blade in the image. This is why it is surprising that using an NSR of 0.1 provided the highest test accuracy for Wiener deblurring. Given that all of the Wiener deblurring test accuracies were so low and similar to the blurry classification results though, it is safe to say that classifying real Wiener deblurred images using a sharp image classifier was not successful in this case.

Table 5.4: Compared Retrained Model Performance on Real RL-Deblurred Rotating Turbine Blade Test Data.

Architecture	Iterations	Test Accuracy	Test Precision	Test Recall	F1-Score
(TL)-ResNet-50	**25**	**0.6500**	**0.5000**	**0.7143**	**0.5882**
(TL)-ResNet-50	30	0.6200	0.4800	0.6667	0.5581
(TL)-ResNet-50	15	0.6200	0.3600	0.7500	0.4865
(TL)-ResNet-50	50	0.6100	0.5800	0.6170	0.5979
(TL)-ResNet-50	20	0.6100	0.4200	0.6774	0.5185
Custom CNN	15	0.6000	0.8200	0.5694	0.6721
Custom CNN	25	0.5800	0.9200	0.5476	0.6866
Custom CNN	30	0.5600	0.9400	0.5341	0.6812
Custom CNN	20	0.5600	0.8600	0.5375	0.6615
(TL)-Xception	15	0.5500	0.1000	1.0000	0.1818
(TL)-Xception	20	0.5400	0.0800	1.0000	0.1482
Custom CNN	50	0.5300	0.9600	0.5161	0.6713
(TL)-Xception	25	0.5200	0.0400	1.0000	0.0769
(TL)-Xception	30	0.5100	0.0200	1.0000	0.0392
(TL)-Xception	50	0.5100	0.0200	1.0000	0.0392

Table 5.5: Compared Retrained Model Performance on Real Wiener-Deblurred Rotating Turbine Blade Test Data.

Architecture	NSR	Test Accuracy	Test Precision	Test Recall	F1-Score
(TL)-Xception	**0.1**	**0.5600**	**0.2600**	**0.6500**	**0.3714**
(TL)-Xception	25	0.5500	0.1000	1.0000	0.1818
(TL)-Xception	5	0.5400	0.0800	1.0000	0.1482
Custom CNN	0.1	0.5100	1.0000	0.5051	0.6711
Custom CNN	1	0.5100	1.0000	0.5051	0.6711
(TL)-ResNet-50	5	0.5100	0.6600	0.5077	0.5739
(TL)-Xception	1	0.5100	0.1200	0.5455	0.1967
(TL)-Xception	10	0.5100	0.0200	1.0000	0.0392
(TL)-ResNet-50	0.1	0.5000	0.9200	0.5000	0.6479
Custom CNN	5	0.5000	0.9200	0.5000	0.6479
(TL)-ResNet-50	10	0.4900	0.5200	0.4906	0.5049
Custom CNN	10	0.4800	0.8400	0.4884	0.6177
(TL)-ResNet-50	1	0.4700	0.7400	0.4805	0.5827
Custom CNN	25	0.4300	0.6600	0.4521	0.5366
(TL)-ResNet-50	25	0.4200	0.2800	0.3889	0.3256

5.3 Artificially Blurred and Deblurred CAI-SWTB Test Set Images

5.3.1 Artificial Blurring/Deblurring

Test accuracy results from deblurring real rotating turbine images were not as high as what was desired, so a baseline needed to be created for comparison. If images from the CAI-SWTB test set were to be artificially blurred, deblurred, and then classified, the absolute best possible accuracies could be achieved and compared to the real blurry results. This is because the sharp CAI-SWTB test images have already been shown to be classified with accuracies of nearly 100%, and the deblurring would be with the perfect PSF across the entire image, providing the most optimal deblurring possible. The only artifacts that would change the classification of the deblurred images from the original sharp images are the distortions from the deblurring.

The blurring is not be exactly like the blurring in the real blurry images, because the background is also blurry and not just the blade, but the images are similar enough for test purposes. The 50 healthy and 50 blurry PSFs that were used to deblur the real rotating blade images were used to blur and deblur the sharp CAI-SWTB test set images. How it worked was the first 50 healthy images used PSFs 1 through 50 in order, and then the next 50 healthy images used the 50 PSF images in order again. This repeated until all the healthy images were assigned one of the 50 PSFs. This method was also used for the faulty images. This PSF selection was done to provide the artificially blurred images with a variety of different blurs and to provide similar blurs in this data to the blurs of the real rotating turbine dataset. This also was done to save time, because trying to calculate specific PSFs depending on the angles of the blades for 600 healthy and 600 faulty images would have been too time-consuming. Because of this, the blur in these artificially blurred images is for the most part not in the expected direction a blur would be based off of the blade angles relative to the hub, but for test purposes it is good enough. Examples of artificially blurred and deblurred images are similar to the examples shown in Figure 4.11 from chapter 4. The same blurring parameters were used as what is shown in Table 5.2.

5.3.2 ML Classification on Artificially Blurred and Deblurred Images

Results from classifying artificially blurred images are shown in Figure 5.6. The highest test accuracy of 0.7083 was achieved using the custom CNN. One thing to note is that the lowest blurry test accuracy of 0.6433, with no deblurring involved, still beat every single real blurry and deblurred classification except the highest RL-deblurred classification of 0.65. This alone shows that the blurry images used in the real blurry dataset were not similar enough to the test images in the CAI-SWTB for accurate classification using these models.

Table 5.6: Compared Retrained Model Performance on Artificially Blurred CAI-SWTB Test Data.

Architecture	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
Custom CNN	**0.7083**	**0.4233**	**0.9845**	**0.59207**
(TL)-Xception	0.6650	0.3300	1.0000	0.4962
(TL)-ResNet-50	0.6433	0.2867	1.0000	0.4456

Results from deblurring the artificially blurred images using RL-deblurring and Wiener deblurring are shown in Tables 5.7 and 5.8 respectively. Unsurprisingly, Rl-deblurring the artificially deblurred images overall improved the classification test accuracies, with a highest test accuracy of 0.8025 using the transfer learning Xception model and 50 iterations. This was a roughly 10% increase from the blurry test accuracy results. Improvements were seen using the transfer learning Xception and ResNet-50 models, but not the custom CNN model, which actually had worse classification test accuracies after deblurring.

Classification after Wiener-deblurring had mixed results. The highest test accuracy out of both RL and Wiener-deblurring was 0.8125 using transfer learning Xception and $NSR = 5$. Again, transfer learning Xception and ResNet50 classification on the deblurred images seemed to mostly improve from the artificially blurred classification, and the custom CNN did not improve. While Wiener-deblurring provided the highest test accuracy, most of the Wiener test accuracies were lower than the lowest RL-deblurred test accuracy of 0.6533. This means that for certain NSR values, deblurring actually makes classifying the images worse using ML models trained on sharp images.

Table 5.7: Compared Retrained Model Performance on RL-Deblurred from Artificially Blurred CAI-SWTB Test Data.

Architecture	**Iterations**	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-Xception	**50**	**0.8025**	**0.6300**	**0.9618**	**0.7613**
(TL)-Xception	30	0.7967	0.6017	0.9863	0.7474
(TL)-Xception	25	0.7892	0.5883	0.9833	0.7362
(TL)-Xception	20	0.7792	0.5683	0.9827	0.7202
(TL)-Xception	15	0.7600	0.5250	0.9906	0.6863
(TL)-ResNet-50	30	0.7367	0.5217	0.9152	0.6645
(TL)-ResNet-50	25	0.7342	0.4917	0.9547	0.6491
(TL)-ResNet-50	50	0.7250	0.5800	0.8169	0.6784
(TL)-ResNet-50	20	0.7208	0.4617	0.9585	0.6232
(TL)-ResNet-50	15	0.7067	0.4350	0.9526	0.5973
Custom CNN	50	0.6992	0.9817	0.6273	0.7654
Custom CNN	30	0.6867	0.9867	0.6167	0.7590
Custom CNN	25	0.6750	0.9817	0.6085	0.7513
Custom CNN	20	0.6692	0.9700	0.6056	0.7457
Custom CNN	15	0.6533	0.9450	0.5968	0.7316

Table 5.8: Compared Retrained Model Performance on Wiener-Deblurred from Artificially Blurred CAI-SWTB Test Data.

Architecture	**NSR**	**Test Accuracy**	**Test Precision**	**Test Recall**	**F1-Score**
(TL)-Xception	**5**	**0.8125**	**0.6383**	**0.9795**	**0.7730**
(TL)-Xception	10	0.8042	0.6167	0.9867	0.7590
(TL)-Xception	25	0.7567	0.5150	0.9968	0.6791
(TL)-Xception	1	0.7450	0.5900	0.8551	0.6982
(TL)-ResNet-50	5	0.7217	0.6150	0.7818	0.6884
(TL)-ResNet-50	10	0.7008	0.5383	0.7975	0.6428
(TL)-ResNet-50	25	0.6575	0.4350	0.7838	0.5595
(TL)-ResNet-50	1	0.6533	0.7167	0.6361	0.6740
Custom CNN	25	0.6425	0.6833	0.6317	0.6565
(TL)-Xception	0.1	0.6208	0.5783	0.6321	0.6040
Custom CNN	10	0.6158	0.8233	0.5819	0.6819
Custom CNN	5	0.5950	0.9000	0.5590	0.6900
(TL)-ResNet-50	0.1	0.5608	0.8500	0.5385	0.6593
Custom CNN	1	0.5367	0.9517	0.5200	0.6726
Custom CNN	0.1	0.5158	0.9767	0.5082	0.6686

5.4 Analysis

After seeing the test results from the artificially blurred and deblurred classification, which is a best-case deblurring scenario, it is apparent that classifying blurry and deblurred images using a ML model trained on sharp images may not be the best approach. It was previously hypothesized that if the deblurring was effective enough, the deblurred images would look similar to sharp blade images, thus giving high classification test accuracies. In the best case deblurring classification scenario, this was shown to be false. If the blackened edges which are roughly the size of the PSF blur line length were cropped out of the deblurred images, this could have possibly provided an even better best case scenario, for these cropped images would look more similar to the original sharp images. This would require further testing to see if classification accuracies were improved. For classifying deblurred images in the future, it is recommended to train the ML models on deblurred images rather than sharp images. This could not be done in these experiments because there was not enough blurry image data to do so. Despite the drawbacks shown in this research, the reason it is better to still deblur images and train models on those rather than training models on blurry images and just classifying the blurry images is because of the next step in the overall project. After image classification, the faulty images are going to be run through a YOLO model to place bounding boxes around the faults. If the faulty images that are passed into YOLO are blurry, faults may not even appear in the blurry images, or they may appear on incorrect parts of the blade, causing impossible or inaccurate bounding box placement. If images can be deblurred and faults reconstructed, then the bounding boxes can be placed in the correct spots. Since YOLO is primarily used for videos rather than images, in the future it will still be worth a try to see if a YOLO model trained on sharp images with bounding boxes around faults can detect these faults in videos of rotating turbine blades.

Another alternative to using the proposed deblurring methods is to simply get a high speed global shutter camera, or a very high speed rolling shutter camera with a very small shutter time. A global shutter camera captures all of the pixels in the camera array simultaneously, but a rolling shutter captures pixels line by line, which takes time. If a picture of an object in motion is being taken with a rolling shutter, mismatches between the moving object and the line by line pixel data will cause artifacts in the image that cannot be fixed by deblurring. Global shutter images do not have these artifacts, so essentially all of the blur can be deblurred if done correctly. The rolling shutter artifacts are part of the lower quality of some of the real blurry and deblurred images, which in turn affected their classification test accuracies. Higher quality cameras are much more expensive than what are currently being used on this project, but they will dramatically reduce or erase the blur in rotating wind turbine blade images. If higher quality cameras such as those discussed were used though, deblurring could be performed on images of blades with much higher velocities because the blur length would be smaller.

CHAPTER 6
Conclusion and Future Work

In this study, binary fault classification on small wind turbine blades was conducted using a newly created dataset of 6000 RGB images. This created dataset included simulated faults of cracks, holes, and edge erosion in varying environments. A custom convolutional neural network was investigated along with two existing transfer learning architectures: transfer learning ResNet-50 and transfer learning Xception. Through an exhaustive search of hyper-parameters, optimal sets were found for each architecture on both the created CAI-SWTB dataset and the large-scale wind turbine dataset, providing the best accuracy. Here, it was shown that the proposed transfer learning Xception architecture outperformed the other architectures by achieving 99.92% accuracy for the CAI-SWTB dataset and 100% on the large-scale wind turbine dataset.

Next a velocity-based Richardson-Lucy algorithm and a velocity-based Wiener deconvolution algorithm were developed and applied to artificial and real wind turbine blade images. First artificial blurring and deblurring techniques were used to show how the RL and Wiener algorithms work in a controlled setting with grayscale and RGB images and to demonstrate the edge-ripple effects the algorithms have on the deblurred images. Then the RL and Wiener algorithms were applied to an image of a section of a small rotating turbine blade with real blur. The direction and size of the PSF kernel line were determined based off of the rotational velocity of the turbine blade, camera shutter time, and the relative position of the blade image to the center of the blade. The result of the deblurring was a successful recovery of damages shown on the blade.

Finally, healthy and faulty blade image classification was performed on real blurry and deblurred images using the discussed RL and Wiener deblurring methods. The highest test accuracy for real blurry images was 0.5800 using TL-ResNet-50, for RL-deblurred images was 0.6500 using TL-ResNet50 and 25 iterations, and for Wiener deblurring it was 0.5600 using TL-Xception and $NSR = 0.1$. For comparison, artificial blurring and deblurring was done on the CAI-SWTB dataset, and then these images were classified using the same ML models used on the real blurry images. The highest test accuracy for the artificially blurred images was 0.7083 using the custom CNN and 50 iterations, for RL-deblurred images was 0.8025 using TL-ResNet50 and 50 iterations, and for Wiener deblurring it was 0.8125 using TL-Xception and $NSR = 5$. More work still needs to be done here, but one way to improve these results in the future is to train the ML models used for classification on deblurred turbine blade images rather than sharp images. Another way to possibly improve the results is to crop out the blackened edges in the deblurred images caused by the PSF blur length.

For image classification, more advanced machine learning architectures, such as YOLO (You Only Look Once), could be implemented to draw bounding boxes around the cracks, holes, edge-erosion, or other damages in the image. This would improve the wind turbine maintenance and inspection process by showing the exact position of the faults along the blade. Designing algorithms to calculate the size of blade faults could also be explored. As far as classifying healthy and faulty images of rotating wind turbine blades, alternative deblurring methods could be explored such as training deep learning architectures like DeblurGAN to perform the deblurring. Higher quality but more expensive cameras with global shutters or cameras with rolling shutters but very small shutter times could also be used instead to mitigate most or all blur in rotating turbine blade images.

CHAPTER 7
Code

7.0.1 Custom Convolutional Neural Network Model Structure

```
#Defining our layers and filters
model = Sequential([
    Conv2D(16, (3,3), activation = 'relu', input_shape = (300, 300,3)),
    Conv2D(16, (3,3), activation = 'relu'),
    MaxPool2D((2,2)),
    Conv2D(32, (3,3), activation = 'relu'),
    Conv2D(32, (3,3), activation = 'relu'),
    MaxPool2D((2,2)),
    Conv2D(64, (3,3), activation = 'relu'),
    Conv2D(64, (3,3), activation = 'relu'),
    MaxPool2D((2,2)),
  ])
if rate > 0.05:
  model.add(Dropout(rate))
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
if(activate == 'Leaky'):
  model.add(Dense(1))
  model.add(LeakyReLU())
else:
  model.add(Dense(1, activation = activate))
if(optimize == 'AdamW' or optimizer == 'Adafactor'):
  opt = optimizer_choosing(optimize)
  model.compile(optimizer =  opt, loss = 'binary_crossentropy', metrics = ['
    accuracy'])
  model.summary() # prints out the summary of the model
  history = model.fit(train_iterator, epochs = epoch, validation_data =
    val_iterator) # wasn't here in Edwin's code
  return [model, history]
else:
  model.compile(optimizer =  optimize, loss = 'binary_crossentropy', metrics
     = ['accuracy'])
  model.summary() # prints out the summary of the model
  history = model.fit(train_iterator, epochs = epoch, validation_data =
    val_iterator)
```

7.0.2 Transfer Learning ResNet-50 Model Structure

```
#Defining our layers and filters
model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
          input_shape=(300,300,3),
          pooling='avg',classes=2)
for layer in pretrained_model.layers:
    layer.trainable=True

model.add(pretrained_model)
if rate > 0.05:
  model.add(Dropout(rate))
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
if(activate == 'Leaky'):
  model.add(Dense(1))
  model.add(LeakyReLU())
else:
  model.add(Dense(1, activation = activate))
if(optimize == 'AdamW' or optimizer == 'Adafactor'):
  opt = optimizer_choosing(optimize)
  model.compile(optimizer =  opt, loss = 'binary_crossentropy', metrics = ['
    accuracy'])
  model.summary() # prints out the summary of the model
  history = model.fit(train_iterator, epochs = epoch, validation_data =
    val_iterator) # wasn't here in Edwin's code
  return [model, history]
else:
  model.compile(optimizer =  optimize, loss = 'binary_crossentropy', metrics
     = ['accuracy'])
  model.summary() # prints out the summary of the model
  history = model.fit(train_iterator, epochs = epoch, validation_data =
    val_iterator)
```

7.0.3 Transfer Learning Xception Model Structure

```
#Defining our layers and filters
model = Sequential()

pretrained_model= tf.keras.applications.Xception(include_top=False,
                  input_shape=(300,300,3),
                  pooling='avg',classes=2,
                  weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=True

model.add(pretrained_model)
if rate > 0.05:
  model.add(Dropout(rate))
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
if(activate == 'Leaky'):
  model.add(Dense(1))
  model.add(LeakyReLU())
else:
  model.add(Dense(1, activation = activate))
if(optimize == 'AdamW' or optimizer == 'Adafactor'):
  opt = optimizer_choosing(optimize)
  model.compile(optimizer =  opt, loss = 'binary_crossentropy', metrics = ['
    accuracy'])
  model.summary() # prints out the summary of the model
  history = model.fit(train_iterator, epochs = epoch, validation_data =
    val_iterator) # wasn't here in Edwin's code
  return [model, history]
else:
  model.compile(optimizer =  optimize, loss = 'binary_crossentropy', metrics
     = ['accuracy'])
  model.summary() # prints out the summary of the model
  history = model.fit(train_iterator, epochs = epoch, validation_data =
    val_iterator)
```

7.0.4 Richardson-Lucy Algorithm

```
# Richardson-Lucy deconvolution with known PSF
def richardson_lucy(blurry_image, PSF, color, num_iter=25):
    # blurry_image and PSF are ndarrays with decimal values ranging between
    0 and 1
    # PSF recomended to be square with odd dimensions less than size of
    blurry_image

    PSF_mirror = np.flipud(np.fliplr(PSF))
    # color==1 means RGB image deblurring
    if color == 1:
      d_r = blurry_image[:,:,0]
      d_g = blurry_image[:,:,1]
      d_b = blurry_image[:,:,2]
      uhat_r = blurry_image[:,:,0]
      uhat_g = blurry_image[:,:,1]
      uhat_b = blurry_image[:,:,2]
      for t in range(1,num_iter+1,1):
```

```
    # red
    uhatPSF_r = fftconvolve(uhat_r, PSF,
       mode='same').astype(np.float32)
    relative_blur_r = np.divide(d_r, uhatPSF_r)
    parentheses_r = fftconvolve(relative_blur_r,
       PSF_mirror, mode='same')
    uhat_r = (np.real(np.multiply(uhat_r,
       parentheses_r)))
    # green
    uhatPSF_g = fftconvolve(uhat_g, PSF,
       mode='same').astype(np.float32)
    relative_blur_g = np.divide(d_g, uhatPSF_g)
    parentheses_g = fftconvolve(relative_blur_g,
       PSF_mirror, mode='same')
    uhat_g = (np.real(np.multiply(uhat_g,parentheses_g)))
    # blue
    uhatPSF_b = fftconvolve(uhat_b, PSF,
       mode='same').astype(np.float32)
    relative_blur_b = np.divide(d_b, uhatPSF_b)
    parentheses_b = fftconvolve(relative_blur_b,
       PSF_mirror, mode='same')
    uhat_b = (np.real(np.multiply(uhat_b,parentheses_b)))

    uhat = np.dstack((((uhat_r)),
             ((uhat_g)),
             ((uhat_b))))
  # color==0 means grayscale image deblurring
  else:
   d = blurry_image
   uhat = d # out predicted image
   for t in range(1,num_iter+1,1):
    uhatPSF = fftconvolve(uhat, PSF,
       mode='same').astype(np.float32)
    relative_blur = np.divide(d, uhatPSF)
    parentheses = fftconvolve(relative_blur,
                PSF_mirror, mode='same')
        uhat = np.real(np.multiply(uhat,parentheses))
    return uhat
```

7.0.5 Wiener Deonvolution Algorithm

```
def wiener_1channel(img_blurry,PSF,NSR):
  # Get size of PSF
  m = PSF.shape[0]
  n = PSF.shape[1]

  tb_border = int((n-1)/2) # top and bottom border
  lr_border = int((m-1)/2) # left and right border

  # Pad image with 1px of 0's on each side
  input_img = cv2.copyMakeBorder(img_blurry, tb_border, tb_border, lr_border
    ,
                                    lr_border, cv2.BORDER_CONSTANT, value
    =0.0)

  eps = 2**(-52)
  i_blur = (np.copy(input_img))
  h = (np.copy(PSF))
  I = fft2(i_blur)
  H = psf2otf(h, i_blur.shape)
  H_star = np.conj(H)
  H_abs  = np.abs(H)

  denom = np.multiply(H_abs,H_abs)+NSR

  for i in range(denom.shape[0]):
    for j in range(denom.shape[1]):
      denom[i,j] = max(denom[i,j],eps)

  G = np.divide(H_star,denom)
  J = G*I
  j = np.real(ifft2(J))
  j_resized = j[tb_border:j.shape[0]-tb_border,lr_border:j.shape[1]-
    lr_border]
  return j_resized

def WF_deblurring(img_blurry,PSF,bl,NSR,color):
  # PSF = point spread function
  # bl = blur length
  # NSR = noise to signal ratio
  i_blur = (np.copy(img_blurry))
  h = (np.copy(PSF))
  if color==1: #RGB
    # red
    jr = wiener_1channel(i_blur[:,:,0],h,NSR)
    # green
    jg = wiener_1channel(i_blur[:,:,1],h,NSR)
    # blue
    jb = wiener_1channel(i_blur[:,:,2],h,NSR)

    # combine
    #img_deblurred = normalize(np.dstack((jr,jg,jb)),colour)
    img_deblurred = (np.dstack((jr,jg,jb)))*bl
  else: #grayscale
    img_deblurred = wiener_1channel(i_blur,h,NSR)


  return img_deblurred
```

7.0.6 Supporting Functions for Wiener Deconvolution (sourced from github)

```
def zero_pad(image, shape, position='corner'):
    """
    Extends image to a certain size with zeros

    Parameters
    ----------
    image: real 2d ‘numpy.ndarray‘
        Input image
    shape: tuple of int
        Desired output shape of the image
    position : str, optional
        The position of the input image in the output one:
            * ’corner’
                top-left corner (default)
            * ’center’
                centered

    Returns
    -------
    padded_img: real ‘numpy.ndarray‘
        The zero-padded image

    """
    shape = np.asarray(shape, dtype=int)
    imshape = np.asarray(image.shape, dtype=int)

    if np.all(imshape == shape):
        return image

    if np.any(shape <= 0):
        raise ValueError("ZERO_PAD: null or negative shape given")

    dshape = shape - imshape
    if np.any(dshape < 0):
        raise ValueError("ZERO_PAD: target size smaller than source one")

    pad_img = np.zeros(shape, dtype=image.dtype)

    idx, idy = np.indices(imshape)

    if position == 'center':
        if np.any(dshape % 2 != 0):
            raise ValueError("ZERO_PAD: source and target shapes "
                             "have different parity.")
        offx, offy = dshape // 2
    else:
        offx, offy = (0, 0)

    pad_img[idx + offx, idy + offy] = image

    return pad_img

def psf2otf(psf, shape):
    """
    Convert point-spread function to optical transfer function.

    Compute the Fast Fourier Transform (FFT) of the point-spread
    function (PSF) array and creates the optical transfer function (OTF)
    array that is not influenced by the PSF off-centering.
    By default, the OTF array is the same size as the PSF array.

    To ensure that the OTF is not altered due to PSF off-centering, PSF2OTF
    post-pads the PSF array (down or to the right) with zeros to match
    dimensions specified in OUTSIZE, then circularly shifts the values of
    the PSF array up (or to the left) until the central pixel reaches (1,1)
    position.

    Parameters
    ----------
    psf : `numpy.ndarray`
        PSF array
    shape : int
        Output shape of the OTF array

    Returns
    -------
    otf : ‘numpy.ndarray‘
        OTF array

    Notes
    -----
    Adapted from MATLAB psf2otf function

    """
    if np.all(psf == 0):
        return np.zeros_like(psf)

    inshape = psf.shape
    # Pad the PSF to outsize
    psf = zero_pad(psf, shape, position='corner')

    # Circularly shift OTF so that the 'center' of the PSF is
    # [0,0] element of the array
    for axis, axis_size in enumerate(inshape):
        psf = np.roll(psf, -int(axis_size / 2), axis=axis)

    # Compute the OTF
    otf = np.fft.fft2(psf)

    # Estimate the rough number of operations involved in the FFT
    # and discard the PSF imaginary part if within roundoff error
    # roundoff error  = machine epsilon = sys.float_info.epsilon
    # or np.finfo().eps
    n_ops = np.sum(psf.size * np.log2(psf.shape))
    otf = np.real_if_close(otf, tol=n_ops)

    return otf
```

REFERENCES

[1] X. Chen, "Drone-based optical and thermal videos of rotor blades taken in normal wind turbine operation," 2023. [Online]. Available: https://dx.doi.org/10.21227/yzs5-1067

[2] S. Albelwi and A. Mahmood, "A framework for designing the architectures of deep convolutional neural networks," *Entropy*, vol. 19, no. 6, p. 242, 2017.

[3] W. F. Pickard, "Smart grids versus the Achilles' heel of renewable energy: Can the needed storage infrastructure be constructed before the fossil fuel runs out?" *Proceedings of the IEEE*, vol. 102, no. 7, pp. 1094–1105, 2014.

[4] A. T. Belge, S. Mishra, and S. Alegavi, "A Review on Alternative Energy Sources," in *2022 5th International Conference on Advances in Science and Technology (ICAST)*, 2022, pp. 558–563.

[5] L. Fernández, "Share of electricity generation from wind energy sources worldwide from 2010 to 2022," 2023, accessed 18 August 2023. [Online]. Available: https://www.statista.com/statistics/1302053/global-wind-energy-share-electricity-mix/

[6] International Energy Agency, "Tracking wind electricity," accessed 18 August 2023. [Online]. Available: https://www.iea.org/energy-system/renewables/wind

[7] U.S. Energy Information Administration, "Wind turbine heights and capacities have increased over the past decade," 2017, accessed 18 August 2023. [Online]. Available: https://www.eia.gov/todayinenergy/detail.php?id=33912

[8] S. Asian, G. Ertek, C. Haksoz, S. Pakter, and S. Ulun, "Wind turbine accidents: A data mining study," *IEEE Systems Journal*, vol. 11, no. 3, pp. 1567–1578, 2017.

[9] L. Zou, H. Cheng, and Q. Sun, "Surface damage identification of wind turbine blade based on improved lightweight asymmetric convolutional neural network," *Applied Sciences*, vol. 13, no. 6330, p. 2, 2023, accessed 19 August 2023. [Online]. Available: https://doi.org/10.3390/app13106330

[10] U.S. Bureau Of Labor Statistics, "Average energy prices for the United States, regions, census divisions, and selected metropolitan areas," 2024, accessed 17 February 2024. [Online]. Available: https://www.bls.gov/regions/midwest/data/averageenergyprices_selectedareas_table.htm

[11] B. Pinney, B. Stockett, M. Shekaramiz, M. A. Masoum, A. Seibi, and A. Rodriguez, "Exploration and Object Detection via Low-Cost Autonomous Drone," in *2023 Intermountain Engineering, Technology and Computing (IETC), Orem, UT, USA*, 2023, pp. 49–54.

[12] B. Altice, E. Nazario, M. Davis, M. Shekaramiz, T. K. Moon, and M. A. S. Masoum, "Anomaly detection on small wind turbine blades using deep learning

algorithms," *Energies*, vol. 17, no. 5, 2024. [Online]. Available: https://www.mdpi.com/1996-1073/17/5/982

[13] C. Seibi, Z. Ward, M. A. S. Masoum, and M. Shekaramiz, "Locating and extracting wind turbine blade cracks using Haar-like features and clustering," in *2022 Intermountain Engineering, Technology and Computing (IETC)*, 2022, pp. 1–5.

[14] L. M. N'Diaye, A. Phillips, M. A. S. Masoum, and M. Shekaramiz, "Residual and wavelet based neural network for the fault detection of wind turbine blades," in *2022 Intermountain Engineering, Technology and Computing (IETC)*, 2022, pp. 1–5.

[15] C. Seegmiller, B. Chamberlain, J. Miller, M. A. S. Masoum, and M. Shekaramiz, "Wind turbine fault classification using support vector machines with fuzzy logic," in *2022 Intermountain Engineering, Technology and Computing (IETC)*, 2022, pp. 1–5.

[16] Y. Zhang, Y. Zhang, H. Li, L. Yan, X. Wu, and H. Wang, "Wind turbine blade cracking detection under imbalanced data using a novel roundtrip auto-encoder approach," *Applied Sciences*, vol. 13, no. 21, 2024.

[17] Y. Zhu and X. Liu, "A lightweight CNN for wind turbine blade defect detection based on spectrograms," *Machines*, vol. 11, no. 1, 2023.

[18] Y. Hang, J. Wang, Y. Han, B. Fan, and C. Zhang, "Research on an intelligent identification method for wind turbine blade damage based on CBAM-BiFPN-YOLOV8," *Processes*, vol. 12, no. 1, 2024.

[19] M. Memari, M. Shekaramiz, M. A. Masoum, and A. Seibi, "Data fusion and ensemble learning for advanced anomaly detection using multi-spectral RGB and thermal imaging of small wind turbine blades," *Energies*, vol. 17, no. 3, 2024.

[20] L. Hang, C. An, and Y. Yang, "Wind turbine surface defect detection method based on YOLOv5s-L," *NDT*, vol. 1, no. 1, pp. 46–57, 2023.

[21] Y. Yuan, G. Wang, and J. Fan, "WT-YOLOX: An efficient detection algorithm for wind turbine blade damage based on YOLOX," *Energies*, vol. 16, no. 9, 2023.

[22] L. Wang and Z. Zhang, "Automatic detection of wind turbine blade surface cracks based on UAV-taken images," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7293–7303, 2017.

[23] L. Wang, Z. Zhang, and X. Luo, "A two-stage data-driven approach for image-based wind turbine blade crack inspections," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 3, pp. 1271–1281, 2019.

[24] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[25] Q. Chen, Z.-H. Liu, and M.-Y. Lv, "Attention mechanism-based cnn for surface damage detection of wind turbine blades," in *2022 International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*, 2022, pp. 313–319.

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

[27] A. Iyer, L. Nguyen, and S. Khushu, “Learning to identify cracks on wind turbine blade surfaces using drone-based inspection images,” 2022. [Online]. Available: https://s3.us-east-1.amazonaws.com/climate-change-ai/papers/neurips2021/37/paper.pdf

[28] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA*, 2017, pp. 1800–1807.

[29] N. An, “Xception network for weather image recognition based on transfer learning,” in *2022 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE)*, 2022, pp. 330–333.

[30] Rismiyati, S. N. Endah, Khadijah, and I. N. Shiddiq, “Xception architecture transfer learning for garbage classification,” in *2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*, 2020, pp. 1–4.

[31] X. Wu, R. Liu, H. Yang, and Z. Chen, “An xception based convolutional neural network for scene image classification with transfer learning,” in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, 2020, pp. 262–267.

[32] L. Mihalkova and R. J. Mooney, “Transfer learning from minimal target data by mapping across relational domains.” in *In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), Pasadena, CA, USA*, vol. 9, 2009, pp. 1163–1168.

[33] C. Zhang, J. Bin, and Z. Liu, “Wind turbine ice assessment through inductive transfer learning,” in *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2018, pp. 1–6.

[34] H. Yun, C. Zhang, C. Hou, and Z. Liu, “An adaptive approach for ice detection in wind turbine with inductive transfer learning,” *IEEE Access*, vol. 7, pp. 122 205–122 213, 2019.

[35] R. Yue, G. Jiang, X. Jin, Q. He, and P. Xie, “Spatio-temporal feature alignment transfer learning for cross-turbine blade icing detection of wind turbines,” *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–17, 2024.

[36] B. Altice, T. K. Moon, and M. Shekaramiz, “Velocity-based wind turbine blade deblurring using Richardson-Lucy algorithm,” in *2024 Intermountain Engineering, Technology and Computing (IETC)*, 2024, pp. 215–220.

[37] Y. Du, H. Wu, and D. Garcia Cava, “A motion-blurred restoration method for surface damage detection of wind turbine blades,” *Measurement*, vol. 217, p. 113031, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S026322412300595X

[38] P. Biswas, A. Sarkar, and M. Mynuddin, "Deblurring images using a wiener filter," *International Journal of Computer Applications*, vol. 109, pp. 36–38, 01 2015.

[39] Y. Wang, Q. Lu, and B. Ren, "Wind turbine crack inspection using a quadrotor with image motion blur avoided," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 1069–1076, 2023.

[40] Y. Peng, Z. Tang, G. Zhao, G. Cao, and C. Wu, "Motion blur removal for uav-based wind turbine blade images using synthetic datasets," *Remote Sensing*, vol. 14, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/1/87

[41] J. Lee, S.-W. Ji, S.-J. Cho, J.-P. Hong, and S.-J. Ko, "Deep learning-based deblur using gyroscope data," in *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, 2020, pp. 1–4.

[42] W. H. Richardson, "Bayesian-based iterative method of image restoration," *Journal of the Optical Society of America (1917-1983)*, vol. 62, no. 1, p. 55, Jan. 1972.

[43] L. B. Lucy, "An iterative technique for the rectification of observed distributions," *Astronomical Journal*, vol. 79, p. 745, Jun. 1974.

[44] C. Prajitha and D. Pradeepa, "A performance comparison of existing de-blurring algorithms for digital images," in *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*, 2014, pp. 1–5.

[45] R. Gonzalez, R. Woods, and S. Eddins, *Digital Image Processing Using Matlab*. Prentice Hall, 2003.

[46] L. R. Welch, "Wiener hopf theory," accessed 26 May 2024. [Online]. Available: https://web.archive.org/web/20060920081221/http://csi.usc.edu/PDF/wienerhopf.pdf