8-29-2022 11:00 AM

# Developing an Efficient Real-Time Terrestrial Infrastructure Inspection System Using Autonomous Drones and Deep Learning

Marlin Manka, *The University of Western Ontario*

Supervisor: Haque, Anwar, *The University of Western Ontario*

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

# Abstract

Unmanned aerial vehicles (UAV), commonly referred to as drones (Dynamic Remotely Operated Navigation Equipment), show promise for deploying regular, automated structural inspections remotely. Deep learning has shown great potential for robustly detecting structural faults from collected images, through convolutional neural networks (CNN). However, running computationally demanding tasks (such as deep learning algorithms) on-board drones is difficult due to on-board memory and processing constraints. Moreover, the potential for fully automating drone navigation for structural data collection while optimizing deep learning models deployed to computationally constrained on-board processing units has yet to be realized for infrastructure inspection.

Thus, an efficient, fully autonomous drone infrastructure inspection system is introduced. Using inertial sensors, mounted time-of-flight (ToF) and optical sensors to calculate distance readings for obstacle avoidance, a drone can autonomously track around structures. The drone can localize and extract faults in real-time on low-power processing units, through pixel-wise segmentation of faults from structural images collected by an on-board digital camera. Furthermore, proposed modifications to a CNN-based U-Net architecture show notable improvements to the baseline U-Net, in terms of pixel-wise segmentation accuracy and efficiency on computationally constrained on-board devices.

After fault segmentation, the fault points corresponding to the predicted fault pixels are passed into a custom fault tracking algorithm; based on a robust line estimation technique, modifications are proposed using a quadtree data structure and a smart sampling approach. Using this approach, the drone is capable of following along faults robustly and efficiently during inspection to better gauge the extent of the spread of the faults.



## Keywords

UAV, Drone Infrastructure Inspection, Structural Health Monitoring, Robust Line Estimation, Deep Learning

# Summary for Lay Audience

Timely and high-quality structural inspections are necessary. However, manual inspection practices are still widely adopted, which have proven to be costly, time-consuming, and risky to inspectors who must manually assess these structures close-up. Technological advances in recent years have opened the possibility of automating parts of the inspection process: the data collection process and the analysis of collected data. Aerial vehicles called drones can be controlled by an offboard pilot and are beginning to be used to perform close-up structural inspections as opposed to humans. Instead of human senses and hand-held apparatuses respectively collecting qualitative and quantitative measurements, cameras and other sensors can be mounted on the drone to automatically collect this information during fly-by. However, processing this information is difficult on drones, due to their limited processing capabilities. Sensors also enable the possibility for fully autonomous navigation without the need for a human pilot. Yet, most current applications of drones for structural inspection require drones to be manually piloted.

Thus, proposed is a fully autonomous inspection system that uses a drone that can navigate on its own without the need for a manual pilot. This drone, mounted with a camera, can collect and process images during structural inspection in an efficient manner, to extract possible structural defects and faults (such as cracks) in live time, while also tracking along these faults during the inspection.

# Acknowledgments

I would like to thank my supervisor, Dr. Anwar Haque, for his guidance and inspiration for new ideas. I would also like to thank Dr. Haque for providing a lab space for drone testing and development, and for his continual involvement in the progress of our work.

Next, I would like to thank Kirk, Muhammad, Jerry and Bhavya, graduate students whom I had the pleasure of working alongside in the lab. I learned a lot from them and found great value in sharing ideas together. I would also like to thank Gopi, a former graduate student under Dr. Haque's supervision, who helped set me on the right track in terms of drone development. Also, special mention to Dr. Ayan Sadhu, who provided me with insights into his lab work regarding structural health monitoring, which inspired my deep learning approach.

Finally, I would like to thank my family and friends who have supported me throughout the entire journey of my graduate studies.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

## 1 Introduction

The maintenance of terrestrial structures, including civil and mechanical structures, requires timely and periodic inspection. The risk of failure is heightened due to the steep increase of aging infrastructure in recent years. According to the National Highway System (NHS) of Canada, between 2006 and 2010, the number of bridges 50 years or older increased by over 50%, compared to just 10% for bridges less than 10 years old [1]. In the United States of America, the 2021 Infrastructure Report Card released by the American Society of Civil Engineers (ASCE) found that 42% of all bridges are 50 years or older, 7.5% of which are structurally deficient [2]. Although structurally deficient bridges are not necessarily likely to imminently collapse, they require more frequent assessment to mitigate potential failure [3]. Due to the significant number of older bridges, the emphasis has shifted to maintaining existing bridges.

However, the rate and quality of inspection have been questioned. In a recent Audit by the Office of the Auditor General of Ontario, it was found that the Ministry of Transportation is unaware of whether structural maintenance is being done in a timely manner by separate regions [4]. Bridges and structures deteriorate at their own rate, and some are at greater risk of failure than others. Prioritizing frequent monitoring of bridges that are in *fair* or *poor* condition is paramount to early rehabilitation, to prevent further costs in repairs down the line and mitigate the potential risk of collapse. The quality of inspection is also an area of great concern. The Ministry of Transportation found numerous instances of missing, incomplete, or inaccurate inspection files due to incorrect recorded measurements and limitations in the inspection itself [4].

It is evident that the need for higher quality and more frequent inspection of bridges and terrestrial structures is greater than ever. However, current manual inspection methods consume significant man-hours and require expensive equipment and personnel to coordinate and perform inspections. In the United States of America, the average inspection cost per bridge is between $4,500 and $10,000 [6]. Furthermore, acquiring and

investing in the equipment for manual inspection, such as ladders, under-bridge trucks [5], man-lifts and scaffolding is costly [7]. Most inspection procedures last several days, depending on the size of the infrastructure. As a result, the time cost through man-hours spent manually conducting inspections is significant, limiting the feasibility of more frequent inspections.

With the recent advances in technology, the potential of improving upon traditional inspection processes through automation is beginning to be realized and tested. Unmanned aerial vehicles (UAV) – also referred to as drones – have been realized as a viable solution to automate infrastructure inspection, providing the possibility for more frequent, continuous, and high-quality inspections at a lower cost. Drones equipped with sensors and cameras can enable autonomous navigation, while remotely collecting structural data to be post-processed for structural health assessment. However, most of the current work focuses on optimizing parts of the inspection process through automation, while manual supervision and intervention is still required at some stage.

## 1.1 Challenges of Drone Infrastructure Inspection

Typically, drones are controlled by an off-board human operator for infrastructure inspection. As a result, challenges arise due to accessibility and cost limitations. Another approach is to autonomously control a drone via radio using ground control stations, such as computers or smartphones, which send position waypoints for the drone to fly to. Ground control stations can also process data collected by the drone during structural inspection. However, in remote inspections where the drone must fly beyond the range of the ground control station and where wireless connectivity is limited, an on-board companion computer mounted to the drone is a necessary alternative, which presents its own challenges. Both manually operated and automated approaches present challenges to inspection.

### 1.1.1 Manual Operational Constraints

Due to the complexity of some structures, some elements may be inaccessible for close observation to inspectors. For instance, tall structures present a challenge for close

inspection in high and poorly supported areas, such as windmills and cell towers. This presents a safety risk to inspectors – even with the aid of equipment, the risk of injury and death is heightened. Drones can help alleviate such risks, as they can be deployed and manually operated from a safer distance, using a transmitter that sends radio signals to the drone. The operator would need to be able to see the drone and obstacles around it, either directly via their line-of-sight or through a camera mounted onto the drone, capable of streaming a live feed to the operator. However, cameras can provide misleading depth perception for real-time avoidance. Thus, drones typically rely on other multi-directional sensors that measure precise distances to nearby objects, although it would be difficult for an operator to interpret such distance readings in real-time. Also, when inspecting more complex structures in busy environments, the radio signals from the transmitter can get obstructed and the operator's line-of-sight can get occluded, posing an elevated risk of unintentional drone collision. Moreover, drones need to be manually controlled by an operator in relative proximity to the inspection site, resulting in notable travel and manual operational costs that may limit more frequent, regular inspection.

### 1.1.2 On-board Computational Constraints

The companion computer communicates in close-range with the flight controller, which responds to commands and controls the speed of the motors accordingly using built-in sensors. Autonomous navigation makes use of sensors to enable the drone to 'sense' its environment and estimate self-position for localization. For real-time localization, algorithms must be not only accurate but also efficient; these algorithms must be inexpensive to enable real-time processing on low-power companion computers. Furthermore, memory and power-constrained companion computers limit the feasibility of performing intensive data processing tasks on drones – even more so for real-time applications.

## 1.2 Thesis Contribution

Given the limitations that arise due to manual operational costs and on-board power and memory constraints, the main contribution of this thesis is in realizing an efficient, fully

autonomous inspection system using drones. Namely, this thesis aims to address the following areas:

- Regular deployment of a drone equipped with a camera and sensors that can autonomously navigate and track around a structure without the need for manual control, while capturing and sending an image stream to a companion computer for further processing.

- Pixel-wise extraction and localization of structural faults using a semantic segmentation deep learning method, with the U-Net architecture at the core.

- Proposal of further modifications to the U-Net architecture to enable accurate, real-time fault localization on low-power processing units on-board the drone. An investigation is also conducted to compare the modified architectures with the state-of-the-art baseline U-Net architecture, in terms of performance and efficiency.

- Real-time fault tracking to better gauge the spread of faults along structural walls during inspection. Namely, a modified random sampling consensus approach is used to estimate the fitting line robustly and efficiently for a set of extracted fault points corresponding to the fault pixels predicted by the deep learning method.

## 1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 provides background information regarding the types of structural faults, image processing methods for identifying faults, and the viability of drones for inspection. Chapter 3 provides a literature review of the deep learning approaches for structural fault assessment and the application of such approaches for drone inspection. Chapter 4 provides a high-level overview of the proposed inspection system architecture and platforms developed upon. Chapter 5 provides a more thorough description of the proposed structural and fault tracking algorithms implemented to address the goal of accurate and efficient autonomous drone navigation. Chapter 6 proposes a

modified deep learning semantic segmentation approach for real-time fault extraction on low-power edge devices that can be equipped to drones. Finally, in chapter 7, a conclusion summarizing the findings of the proposed methods along with future enhancements is provided.

# Chapter 2

# 2 Background

This chapter will provide background information about terrestrial infrastructure faults, as well as image-based approaches for identifying faults. Then, the application of drones is discussed.

## 2.1 Defining Structural Faults

There are different types of faults that highlight structural problems. Cracks are a common fault found on the material surface of infrastructures. Cracks are of particular interest since they are important to determining the severity of structural damage, based on characteristics such as crack width, depth and change in direction. Such information is important to determining how quickly the fault should be repaired before further damage occurs. Cracks can be classified into two broad groups: *active* cracks, which are characterized as long and multidirectional, with noticeable displacement and misalignment in depth, width, and direction over an area; and *dormant* cracks, which show no such change in direction, and are typically characterized as hairlike or irregular [8]. Although both types of cracks may become enlarged over time, active cracks are particularly concerning as they may be caused by structural overloading, flaws in the design of the structure, or detrimental external conditions [9]. Being able to detect and distinguish active cracks is vital to initiating the timely repair of a structure and preventing failure.

Given that most structural elements use concrete or reinforced concrete, faults that occur on concrete surfaces that are of notable concern to structural integrity are listed below [8] [10]:

- *Hairline cracks*: Thin but deep cracks, which can result in more serious cracking over time. It is caused by improper settlement of the concrete while curing.

- *Spalling*: Concrete surface depressions in which the parts of the surface have cracked and delaminated. It is caused by pressure underneath the surface of the concrete, typically by poorly constructed joints or corrosion in the rebar in the reinforced concrete. Spalling can result in the corroded metal to become exposed, which is prone to further corrosion through exposure to air and water, undermining the integrity of the structural element.

- *Scaling*: Like spalling, but not as expansive or deep. Delamination occurs as air and water pockets rise to the concrete surface, forming blisters which break open.

- *D-Cracking*: Cracks that form parallel to or stem from longitudinal and transverse joints, due to periodic freezing and thawing. These cracks are deeper than surface cracks and expand outward towards the center of the concrete element over time.

- *Offset Cracking*: Cracks where the concrete is at different levels on either side of the crack. This is due to uneven surfaces below the concrete element.

- *Diagonal Corner Cracking*: Cracks that form from a corner joint of the concrete element. These cracks are the result of curling or warping at the corners of the concrete; since these corners have empty space below them, weight overload from above structural elements can cause these corners to crack downwards into the space.

Thus, it is important to not only be able to detect if a fault exists in an inspected area, but to also localize the region of the fault and determine how it is expanding or changing direction, to better gauge its severity according to the different fault types.

## 2.2 Identifying Faults in Images

Structural health monitoring (SHM) is a strategy for continuously evaluating and monitoring structural health. It is widely adopted as it can dynamically respond to adverse structural changes [11]. SHM relies on a periodic stream of measurements, which can be

provided through contact sensors such as inertial measurement units (IMU), fiber optic sensors, light detection and ranging (LiDAR) sensors, and ultrasonic wave sensors. However, in recent times, non-contact sensors such as digital cameras have gained popularity: they are easy to deploy, cost-effective, and inherently work with image-based processing techniques with minimal preprocessing. In a camera, each pixel is a sensor, so it can collect a large amount of structural data, represented as an RGB or grayscale image. Image-based processing within the field of computer vision has shown promising results for automated fault identification from images [12].

### 2.2.1 Edge Detection

Many of the traditional image processing techniques extract features using filter-based methods. A filter is an operation performed on an image to modify it from its original state. Commonly, a filter is applied to output a new image highlighting a target feature. The filter is applied to a neighbourhood of pixels surrounding each pixel in the input image [13]. Therefore, the output of each pixel depends on its neighbourhood and the values encoded in the filter. In edge detection, filters are used to preprocess images by removing noise and are the basis for detecting pixels corresponding to edges. Thus, filters are particularly useful for detecting edges and boundaries that correspond to cracks. Several filter-based methods that have been tested for crack detection include the fast Haar transform, fast Fourier transform, Morphological operator, Canny filter and Sobel edge detector. One major downside of these filter-based methods is that they use local features to determine cracks, which are susceptible to differing illumination conditions, distortion, local element material and occlusion from other outdoor elements due to lacking knowledge of the global context [14].

Moreover, some methods use intensity-thresholding techniques as a post-processing tool to further distinguish high intensity pixels from low intensity pixels that are often associated to cracks. Otsu thresholding is a popular thresholding method that aims to separate pixels into a foreground and background class, wherein the spread of the pixel intensities mapped to a specific class, also known as the variance, is minimized, while the variance between the two classes is maximized [15]. However, because cracks typically

make up a small percentage of an image and thus only a fraction of the low intensity points, Otsu thresholding can be unreliable in extracting pixels associated to cracks. Adaptive thresholding is another method which considers only local neighbourhoods of pixels when thresholding. However, this method is also vulnerable to similar pitfalls as filter-based methods. In general, using an intensity threshold is not always reliable, as pixels associated to noise, stains, and low-reflectance materials can also be classified as low-intensity [16].

Although parameters and features can be fine-tuned to improve detection performance on specific datasets, it would be difficult to generalize and scale these traditional image-based approaches to real-world situations.

### 2.2.2 Deep Learning

Machine learning is an area of artificial intelligence (AI) that has been heavily explored and tested in SHM research. These approaches rely on large datasets and require powerful computers for training. The purpose of training is to minimize the error between predictions and ground truth, by adjusting the parameters, each defining the weighted value of a feature over a feature space of the dataset. This error can be defined by a loss function or objective function, which takes the parameters as input, with the goal of finding the optimal parameter values [12].

Machine learning algorithms rely heavily on features that are acquired from image processing methods described in section 2.1. Therefore, features must be carefully selected to obtain meaningful results from machine learning-based algorithms, especially with the goal of identifying structural faults. Plus, machine learning-based methods have been shown to be less than suitable for full-scale infrastructures, where fault patterns are too complex to be captured and defined by a manually-extracted set of features [12].

Deep learning, inspired by the adaptability of the human brain, is a more powerful concept enabling machine learning to take upon human-like tasks more accurately. Deep learning is powerful as it is capable of automatically and optimally extracting features as part of the learning process. The more data provided, the more accurate these algorithms are [12]. The basis for deep learning methods is *neural networks*, in which input data is passed through

a network of computational layers that operate over the data to get a final classification result. Each of these layers is connected via *neurons*, also referred to as *nodes*. Nodes are responsible for combining data from previous layers via weighted connections, where each weight corresponds to the value of the feature that the network learns. The weighted sum of the inputs is then evaluated by a node's *activation function*, to determine the extent to which the weighted sum will impact the learning process in later layers. For each data sample, a forward pass is completed through the network, and after each pass, these weights are adjusted to minimize or optimize the output of a defined loss function. Deep neural networks (DNN) are essentially neural networks with many computational layers between an *input* and *output* layers, as shown in Figure 1 below. These computational layers, also known as *hidden* layers, is where the learning occurs, hence the notion of *deep* learning. With multiple hidden layers, these deep networks can learn from many layers of abstraction as opposed to shallower networks [17].

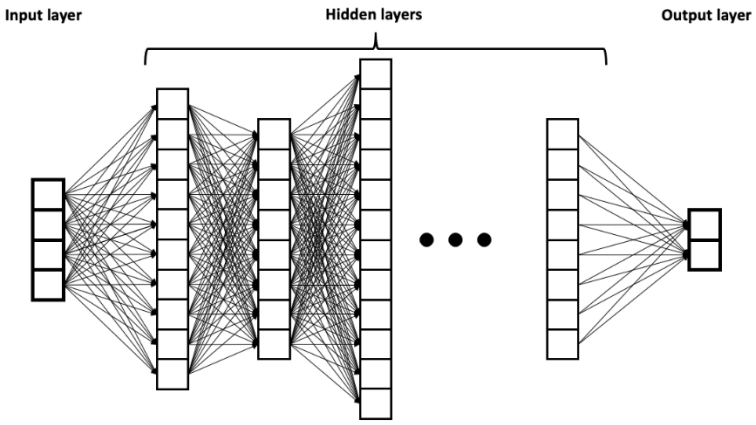


**Figure 1: Deep Neural Network [88]**

#### 2.2.2.1 Convolutional Neural Networks

Visual data in the form of images and videos can also be passed into DNNs called convolutional neural networks (CNN). The motivation behind CNNs for use in structural fault identification is that they are more robust to external factors such as lighting and fault irregularities, compared to traditional image processing and traditional machine learning approaches. It has also been shown that the performance of deep learning-based methods

is typically better than traditional methods for detecting faults. However, they require many training images that account for variations due to external factors representative of the real-world [14]. Furthermore, CNNs can also be computationally intensive.

Different levels of abstraction of the input image can be learned at different layers of a CNN. The initial layers typically extract lower-level information, such as edges and colours, whereas deeper layers later in the network extract higher level features such as shapes and objects, that provide more contextual information [17]. A CNN consists of several different types of layers: an *input* layer, *convolutional* layers, *subsampling* layers, *fully connected* layers, and an *output* layer. The input layer is passed a batch of images, in which each image has a defined width, height and channel size. For example, an input may consist of $A$ images with height $M$ and width $N$, and each image is a colour image defined by three channels of size $C$: a red, blue, and green channel. Such an input can be defined as a *tensor* of shape $(A \times M \times N \times C)$.

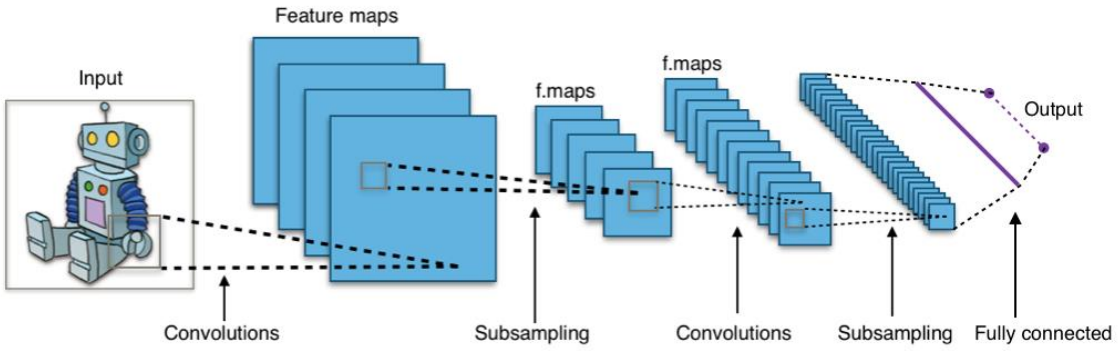


**Figure 2: Convolutional Neural Network with Fully Connected Layers [85]**

*Convolutional Layers*: A convolutional layer applies a filter, otherwise referred to as a kernel, over local regions of the input images, performing element-element multiplication to produce a filter response as an extracted *feature map* from the images. A convolutional layer is parameterized by the kernel size, the number of channels, the stride factor, and the padding [17]. The kernel is a window of size $f \times f \times C$, where $f$ is the height and width equal to or less than the width and height of the input image size. The kernel slides across the input image, passing over a certain number of pixels at a time, defined as the stride $s$, and outputs a new pixel value in the same manner as a traditional filter. The padding $p$

adjusts the size of the output feature map near the borders, which may be useful in situations where the height and width of the feature map must match that of the input image. Given that $k$ kernels are applied with a stride $s$ to the input image of resolution $M \times N$, the convolutional output size $H_{out} \times W_{out} \times C$ can be expressed as follows [18]:

$$\left(\left\lfloor\frac{M-f}{s}\right\rfloor+1\right) \times \left(\left\lfloor\frac{N-f}{s}\right\rfloor+1\right) \times k$$

The values of $k$ and $s$ are specific to the current convolutional layer and are not necessarily constant throughout the network. In the case that the output spatial dimensions should match those of the input spatial dimensions, padding $p$ is applied, which modifies the above expression as follows:

$$\left(\left\lfloor\frac{M+2p-f}{s}\right\rfloor+1\right) \times \left(\left\lfloor\frac{N+2p-f}{s}\right\rfloor+1\right) \times k$$

*Activation Function*: After each convolutional layer typically follows an activation function. The activation function is applied to the sum of the values of each pixel in the kernel, where each pixel is a weight, multiplied with each pixel in the input image within the receptive field of the kernel. In other words, if there are $k$ kernels to apply, with each kernel $i$ having a weight matrix $W^i$, a bias $b^i$, and $x_s$ denoting the receptive field captured by the kernel, applying an activation function $a$ will produce a convolution of $x_s$ as follows [18]:

$$Z_{i,s} = a[\text{sum}(W^i x_s) + b^i]$$

Commonly used activation functions in CNNs introduce *non-linearity* – this is important for updating the weights after each forward pass. The process of updating the weights is called *backpropagation*. It works by taking the derivative of the loss or objective function with respect to each of the weights, using the chain rule. These partial derivates are also referred to as the *gradients*. Finding such gradients also involves taking the derivative of the activation functions when passing back through the network. Activation functions take the input value as its parameter. Hence, when deriving *linear* activation functions with respect to the input, the result will be the coefficient of the input, which is a constant. Thus,

the weights would only be updated by a constant factor and prevent any real improvement to the output of loss or objective function. One of the most used non-linear activation functions is the rectified linear unit (ReLU), a piecewise linear activation function. It is simple, fast, and results in a more predictable gradient during backpropagation, compared to other non-linear activation functions.

*Subsampling Layers*: CNNs also typically have subsampling layers, also referred to as pooling layers. Their purpose is to down-sample, or reduce the dimensionality of the data, either by averaging or finding the maximum value in each region of the feature map from previous layers and passing the resulting value into the next layer. *Average pooling* refers to taking the average of a region, whereas *max pooling* takes the maximum value of a region.

*Regularizer*: Optionally, CNNs can also have dropout layers, which act as a regularizer to prevent overfitting on training data in large networks with many weights. This layer randomly sets some of the terms in the weighted sum in the output of convolutional layers to 0 with a pre-determined probability, such that these weighted terms do not contribute to the forward pass and backpropagation process. By doing this, the reliance between weighted features is diminished, allowing the network to learn features more robustly on randomly selected weighted terms [24].

*Fully Connected Layers:* These layers may also be used in CNNs after several rounds of convolution and subsampling. These layers follow the structure of a neural network, with nodes in each layer connected to every node in another layer. There are used when the expected output of the network is a classification result. However, there are CNNs have been designed that output an annotated image instead of a class, with the intent of fine-grain localization of where in an image an object is identified. In such networks, the fully connected layers are replaced with convolutional layers that up-sample feature maps to an approximate representation of the original input image. These networks are called fully convolutional networks (FCNs). Applications include object detection and finer segmentation of areas of interest in images. *Semantic segmentation* is a technique used to label pixels that are associated to separate classes, and is commonly used for multi-class

segmentation, although it can also be applied to binary class problems. In Figure 3 below, an FCN is used to produce a pixel-wise output label map matching the input image resolution, with the predicted classes annotated.

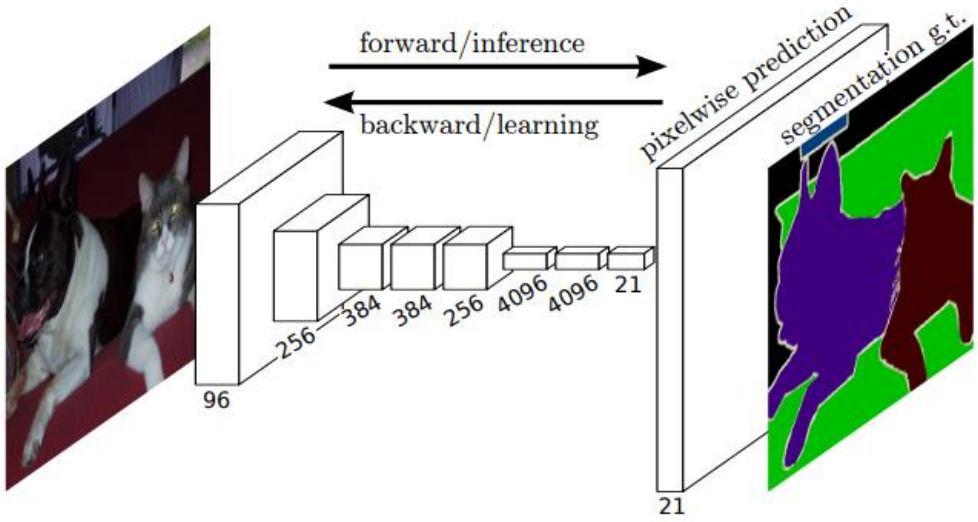


**Figure 3: FCN performing semantic segmentation. The number of channels (feature space size) produced by each convolutional layer is indicated [42]**

*Output Layer*: In the final output layer, a final classification value for each class or a segmentation map in the case of semantic segmentation is outputted. Typically, the raw output values are passed through a *SoftMax* function, which normalizes the real output values to a set of real values between 0 and 1, such that all values sum to 1. These normalized values can be interpreted as probabilities for each individual class, and then outputs the class with the highest probability as the final classification result. SoftMax can also be used for multi-class classification. The SoftMax function is a generalization of the sigmoid function used in binary classification. The SoftMax function is expressed as follows:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

#### 2.2.2.2 Network Architectures

There are several key CNN architectures that have been developed, which have proven instrumental in advancing their performance.

*AlexNet*: Developed by researchers from the University of Toronto [24], the AlexNet architecture was keyed as one of breakthrough CNN-based models, trained on a large image dataset called *ImageNet* [25]. The ImageNet dataset consists of over 15 million labelled high-resolution images, classified into around 22,000 categories [24]. AlexNet trained on this dataset showed a significant reduction in the error rate compared to the state-of-the art methods at the time and other models trained on ImageNet. The architecture consists of an initial layer with $11 \times 11$ convolutional filters, followed by max pooling and other convolutional layers with filters of varying size, resulting in around 60 million weights, or parameters. To prevent overfitting, the authors perform transformations on the images – referred to as augmentations – and use dropouts in the first two fully connected layers with a probability set to 0.5.

*VGGNet*: The Visual Geometry Group (VGGNet) was developed by researchers from Oxford University [26]. It uses smaller $3 \times 3$ convolutional filters compared to those used in AlexNet, with a padding of 1 applied to maintain equal dimensionality in the input and output images. Layers that use these convolutional filters preserve image resolution over multiple convolutional layers, enabling deeper networks with reduced loss of image dimensionality. There are several versions of VGGNet, each with a different number of layers: VGGNet with 11 layers, 13 layers, 16 layers, and 19 layers. The network consists of between 133-144 million parameters, depending on the number of layers.

*GoogLeNet*: A deeper CNN-based network developed by researchers in collaboration with Google [27]. This network consists of 22 layers and roughly 5 million parameters, which is significantly fewer compared to AlexNet and VGGNet. The reduction in parameters is attributed to stacked sub-networks called inception modules [12]. A naïve inception module consists of multiple convolutional filters of different sizes and a max pooling layer

performed in parallel, with the outputs concatenated. However, larger convolutional filters, along with the concatenation of filters leading to many filters passed into subsequent layers, can result in a significant increase in the number of computations. Thus, a modified version of this module is also proposed, which performs a projection of the number of filters into a smaller feature dimensional space. This projection is achieved through $1 \times 1$ convolution, preserving the input image width and height while reducing the number of number of features in an efficient manner. Stacking these projection-based inception modules makes this network computationally efficient for deeper learning.

*ResNet*: More layers can be beneficial to the learning process to a certain extent. However, the more layers added, the closer the gradients of the loss function computed through backpropagation tend towards zero, impeding a deep network's ability to train effectively. This is referred to as the *vanishing gradient* problem, which has largely been addressed through the initial and intermediate batch normalization of the data [28]. Another issue with deep networks is known as the *degradation* problem: a phenomenon that causes the accuracy to get saturated with increased network depth. In [28], a residual network architecture called ResNet is proposed to address degradation. The authors found that the loss of accuracy can be attributed to diminishing returns in what each deeper layer learns. That is, deeper layers that learn very little, which are sequentially connected to previous layers, will obscure the outputs computed in earlier layers, as these deep layers tend towards learning the zero function. In a residual network, residual blocks (as shown in Figure 4) take the output from earlier layers and add them to the output of latter layers. This way, the network saves what was previously learned as an identity map (an unchanged output) and adds this identity to subsequent outputs in deeper layers. Consequently, residual networks tend towards learning the identity function rather than the zero function. It has been shown that the deeper the residual networks, the lower the error rate, while also being efficient in terms of the number of parameters as deeper networks can facilitate the feasibility of many small layers.

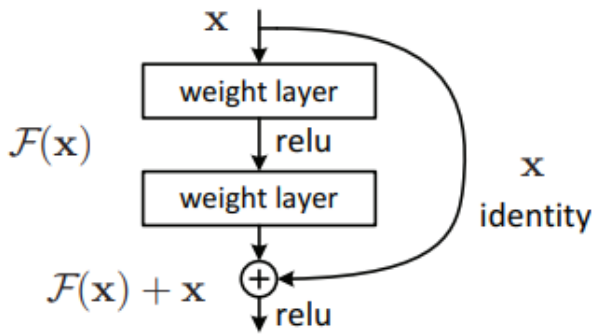


**Figure 4: Residual block in ResNet [28]**

## 2.3 Drones

Drones are key area of research and have been realized to have great potential for a wide variety of real-world applications, such as the delivery of goods, search and rescue missions, agriculture, and surveillance. Drones have also been verified to be safe, cost effective, and operable to the extent of being used for SHM purposes. Drones equipped with high-resolution cameras that capture high-quality images are proving a more reliable, cost-effective, and safe alternative that keeps inspectors distanced from potential hazards. Moreover, the images captured by drones provide comparable results to traditional inspection practices, especially for identifying faults such as spalling and cracks [3]. In a recent survey of state departments of transportation (DOT) within the United States of America, 56% of respondents stated that they were currently using or planning to use drones for bridge inspections, illustrating their growing use and applicability in the industry [19].

### 2.3.1 Classifying Drones

There are two main types of drones: rotary-wing and fixed-wing drones. Rotary-wing drones have multiple rotors, such as quadcopters. Fixed-wing drones have a single rigid wing akin to airplanes. As a result, the fixed-wing only requires energy to move the drone forward as it lifts rather than rotary-wing drones that spend energy to maintain their vertical height while moving forward. This makes fixed-wing drones more energy-efficient and allows them to have a longer battery life compared to rotary drones. With this longer battery

life comes increased flight time, which is ideal for tasks such as drone delivery over long distances. On the other hand, rotary-wing drone are preferred for tasks such as surveillance, search, and inspection, due to their high maneuverability and hovering capabilities [20].

**Figure 5: Left – rotary-wing drone [89]; right – fixed-wing drone [90]**

### 2.3.2 Drone Connectivity

Typically, drones are controlled by an off-board operator via radio transmission. A drone can be equipped with a radio receiver that responds to controls from an operator-controlled radio transmitter. Drones can also be controlled via radio by ground control stations, such as computers or smartphones, which send position waypoints for the drone to fly to. A common configuration is to have a radio connected to the ground control station, which sends data via wireless telemetry to the radio receiver on the drone. More specifically, the drone can communicate with the ground control station on specific radio frequency bands – most drones operate at frequencies of 2.4 GHz and 5.8 GHz. At lower frequencies, drones can travel further from ground control stations, travelling up to 6-7 km but at lower data rates compared to higher frequency bands. Ground control stations can also communicate with the drone via Wireless Fidelity (Wi-Fi) telemetry, which has higher data rates but shorter range than telemetry radios [21]. Existing ground control station software can also be used to create autonomous missions by setting pre-determined waypoints or regions of interest. Through the ground control station, missions can be uploaded to the drone's flight controller via telemetry. The flight controller is the brain of the drone, responding to commands and controlling the speed of the motors accordingly using built-in sensors.

However, in remote applications where the drone must fly beyond the range of the ground station, an on-board companion computer that is mounted to the drone is a necessary alternative. The companion computer communicates in close-range with the flight controller via a serial cable connected directly to the flight controller or Wi-Fi [22]. Commonly used companion computers on drones are based upon the open-source, programmable Arduino circuit boards, which are lightweight and portable microcontrollers.

To correctly respond to waypoint position commands, drones require a positioning system to position themselves accurately in the world. As a result, most drones used in outdoor settings require the Global Positioning System (GPS) and a GPS signal receiver for positioning. For the most part, GPS works adequately well in open outdoor spaces, but suffers from signal occlusion in congested areas, such as near large infrastructure, which can hamper the drone's ability to accurately position itself. GPS-based navigation, coupled with autonomous localization and positioning that uses sensors to enable the drone to 'sense' its environment and estimate self-position, is a solution to this problem. Drones send back position estimation data to the companion computer as feedback that these algorithms take in to continuously update the drone's position for autonomous flight. Besides providing drone position data, drones can also send task-specific data, such as sensor readings and images, that can be processed in real-time or offline. Processing information, particularly images, using deep learning with CNNs for the purposes of extracting semantic information, such as identifying faults in images for the purposes of SHM, can be highly intensive – more so performing such operations in real-time. Hence, on-board companion computers with limited processing capabilities present restrictions for these applications.

# Chapter 3

# 3 Literature Review

This chapter will provide an analysis of the CNN image-based structural fault assessment approaches. Ways of making these approaches more computationally efficient are discussed, as well as the extent to which image-based solutions are being used in drone-based infrastructure inspection.

## 3.1 Analysis of CNN-Based Fault Assessment Approaches

CNNs have become a heavily researched area of image-based processing since the early 2010s. Although CNNs were first introduced in the 1990s, limited training data and computational resources were available at the time. With larger datasets becoming widely available in online public domains, along with increased computing power through Graphics Processing Units (GPU) enabling parallel processing of data, training these deep learning algorithms to achieve accurate results within a reasonable time frame has become a more attainable task for various image processing tasks, including SHM.

### 3.1.1 Image Classification

An area of interest in image processing is distinguishing observations with similar features into individual classes, known as classification. Binary classification consists of predicting observations to be in one of two classes, rather than more than two classes in the more general multi-class classification case. A binary classifier determines which class an observation belongs to, based on a *probability* and *threshold* value: the probability value, with respect to the threshold, determines whether the observation belongs to one class or another. Thus, the resulting output to a binary classifier is a categorical value, typically denoted as ‘0’ or ‘1’.

This binary classification approach is important to single out a specific target amongst observations, which is especially useful in the case of identifying anomalies. As a result, binary classification techniques have been used extensively to identify faults in structural

images. CNNs are commonly used to facilitate binary classification of images, where each image is assigned a class label. Thus, distinguishing fault images is a matter of identifying the existence of faults in an image, rather than localizing them. Typically, in the context of classifying structural images, the typical structure of CNNs – as described in section 2.2.2.1 – is followed, with fully connected layers following the convolutional layers. However, the fully connected layers can be replaced by other final classifier layers. In [17], the authors perform a comparison between different classifiers for pavement crack detection in images. Using a base VGG-16 (VGGNet with 16 layers) pre-trained on ImageNet, they replace the fully connected layers with a single layer neural network classifier, a Support Vector Machine (SVM) and Random Forest (RF) classifier.

Due to the rarity in occurrence of faults in images, classes tend to be unbalanced, with most observations falling into the non-fault class. This can negatively impact the learner, as it would tend to classify observations as part of the non-fault, or *negative* class rather than classify them as part of the fault, or *positive* class. This can lead to an increase in *false negatives*, in which the classifier incorrectly outputs that an observation is a non-fault when in fact it is. Different approaches are proposed to mitigate this phenomenon. One approach is to weigh the positive class more heavily during the training process such that false negatives are penalized more heavily. In [31], a class-balancing weight is introduced to balance the contribution of the positives and negatives to the loss in detecting pavement cracks. Another approach is to intentionally resample positive-labelled samples, called *oversampling*. Also, particularly in the case of semantic segmentation, a crude yet effective approach is to take smaller crops of images that contain a greater fraction of the positive class than the entire image. In [30], an algorithm is implemented to extract random patches from training images of pavement surface cracks for pixel-level segmentation, such that each patch contains 60% of the target, or 'crack' class. The authors find that this ratio optimizes the precision while minimizing the false positive rate. Another more novel approach for semantic segmentation is used in [29], in which pixel-level crack segmentation is performed. Due to cracks being narrow and having a small area relative to the entire image, the authors find that the pixel annotation inaccuracies deter the performance of their CNN-based classifier. To handle this, they use pixel tolerances to allow positively labelled pixels by the classifier within a certain pixel range of a true label

to be considered a true positive. They find this significantly improved the performance of their classifier.

Another issue is that the images collected for training and validation are usually captured using high-resolution cameras. However, passing large, high-resolution images into a CNN is highly inefficient as the number of convolutional operations increases significantly. Some pre-trained networks also require that images of a relatively small, fixed size be passed in. Plus, large images need to be down-sampled significantly, and as a result, information describing relatively small, yet complex faults can get reduced or lost, given that faults cover a small proportion of entire images [32].

In [33], a GoogLeNet-based network is applied to images of concrete bridge surfaces to identify crack images. The authors use Inception modules to enhance the efficiency of their network. Moreover, 1,455 images with 4,160 × 3,120 pixel resolutions are collected. These images are cropped into smaller images of 256 × 256 pixels, which increases the dataset size to 60,000 images, providing more data for the model to train on. These cropped images are also downsized to 224 × 224 to match the required input size for the GoogLeNet architecture. In [32], images are divided into grids of different scales for road crack detection. The authors reason that due to the weights of the cracks being small relative to other larger-scale features in the images, crack information is limited. They argue it is necessary to divide the image such that the weight of the cracks in the individual patches becomes more significant. In this way, each grid is evaluated as a separate image to be classified. In [34], a sliding-window approach is used to scan across patches of crack images larger than 256 × 256 pixel resolutions. These patches are passed into a custom-trained CNN, classifying each patch of the original image separately.

### 3.1.2 Object Detection

The key difference between image classification and object detection in images is the ability for a classifier to localize the areas of faults from a *single* input image. CNNs can be repurposed as object detectors that output not only whether a fault exists in certain regions of an input image, but also the coordinates enclosing the regions *where* these faults occur [38] [39] [41] [54] [55] [57].

**Figure 6: Crack detection result shown with bounding boxes [57]**

A sliding window approach, as discussed in section 3.1.2, can be used to scan across small, sequential patches of an input image in a brute-force manner. However, this would be very computationally expensive for object detection, as many different locations and scales encapsulating possible objects of varying size and aspect ratio in the image would need to be considered and fed into a CNN. To address this, Region-Proposal Networks (RPN) use traditional image processing techniques to identify edges and shapes, to output a set of rectangular regions of interest where objects are likely to occur in an image. This set of proposed regions is much smaller than the number of regions considered by the brute-force method, making it more computationally feasible to feed through a CNN. In [35], RPNs are combined with CNNs to produce regions with CNN features, called R-CNN. A faster and more performant alternative is proposed in [36], coined as Fast R-CNN: instead of taking crops of proposed regions separately, Fast R-CNN feeds the entire image through convolutional layers to produce a feature map from which region proposals are extracted. By using a feature map, the network shares computations. Although this method is shown to be more computationally efficient than previous methods, the region proposal stage is still a bottleneck. Thus, Faster R-CNN is introduced in [37] to allow the network to predict

region proposals through a unified network. The authors also draw a comparison to methods that use pyramids of images and feature maps at different scales, for multi-scale feature extraction. They note that although multi-scale feature extraction may be superior in terms of accuracy, Faster R-CNN is considerably faster. In [38], a Faster R-CNN-based structural vision inspection method is proposed for quasi real-time detection of multiple damage types.

Nevertheless, multi-scale feature extraction is still a prevalent method for object detection, including fault detection. Modifications to multi-scale feature pyramid networks have shown promise for fault detection in real-time applications. In [39], a real-time crack detection algorithm for pavement crack detection is developed using a CNN with multi-scale feature layers. The initial convolutional layers are based on a truncated VGG-16 network which outputs feature maps. The feature maps are then passed into a multi-scale feature extraction block, where for each feature map at a different scale, the feature map of the next layer is computed and the predicted bounding boxes at the current scale are produced through $3 \times 3$ kernel convolutions. After this block, the predicted boxes from different feature maps are summed together in the output. Instead of fully connected layers, convolutional layers are used to allow for input images of varying sizes. Furthermore, convolutions reduce the amount of memory and computations required compared to fully connected layers; the fully connected layers consider all possible weighted connections between neurons in different layers, whereas convolutional layers only consider connections based on spatially local features. Using this methodology, the authors achieve a high accuracy while reaching a detection rate of 96.6 FPS on video frames of resolution $576 \times 1{,}024$.

The aforementioned methods apply a model to an image at multiple locations and scales. Another approach is to apply a single convolutional network to the entire input image and simultaneously predict bounding boxes and class probabilities for each box. One such approach is called You Only Look Once (YOLO) [40]. Unlike region-proposal based approaches, YOLO can garner greater contextual information by looking at the entire input image. Furthermore, it is fast, making it suitable for real-time detection. In [41], YOLOv3,

a newer version of the original YOLO, is modified to develop a lightweight aircraft crack detection system, YOLOv3-Lite. The system is comprised of a backbone network to extract crack features, a feature pyramid that combines crack features from different scales, and a YOLOv3 module to perform bounding box regression. In the backbone network, the authors use *depth-wise separable* convolutions, which is a form of factorized convolutions that reduces the number of multiplication operations and parameters of a standard convolution. Namely, depth-wise separable convolutions separate a standard convolution into two parts: a *depth-wise* convolution that applies a single filter to each channel of the input image, and a *point-wise* convolution that applies $1 \times 1$ convolutional filters to combine the channel outputs from the depth-wise convolution. The authors in [41] highlight a reduction in the number of computations by 8 to 9 times, compared to a standard convolution. A feature pyramid is then employed to capture crack feature maps at different scales, which are combined through concatenation of these feature maps using residual connections. Through concatenation, the fusion of lower-level features from large feature maps and higher-level semantic contextual features from small feature maps can effectively be achieved. Since smaller feature maps have a larger receptive field, larger cracks can be detected, whereas larger feature maps have a relatively smaller receptive field, making it possible to detect smaller, narrower cracks. The detection speed of YOLOv3-Lite is 50% faster than that of YOLOv3, while achieving an average precision close to that of YOLOv3.

### 3.1.3 Semantic Segmentation

Although object detection methods can localize the area of an object in an image, sometimes it is necessary to extract finer-level details about an object, such as its pose, shape, and spatial dimensions. Particularly with faults, it is advantageous to extract more detailed information about their width, height, and spread to better gauge their severity. Thus, pixel-wise segmentation of an image would be ideal to ascertain such details. In semantic segmentation, each pixel is assigned a class label. Fully convolutional networks (FCN) have been shown to be superior to other semantic segmentation approaches, in terms of performance and efficiency [42] [48] [49]. The output to an FCN that classifies each pixel as either a fault or background class is a segmented label map that annotates the classes of interest [29] [30] [31] [44] [48] [49] [53] [59].

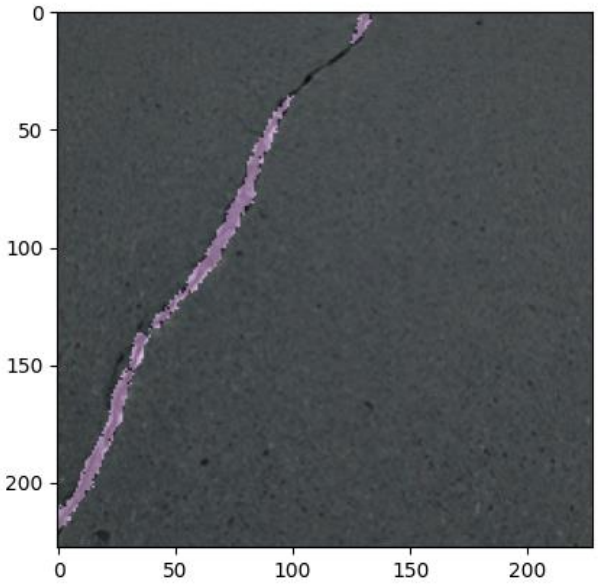


**Figure 7: Output of an FCN used for semantic segmentation of cracks**

To retrieve a label map, down-sampled feature maps are up-sampled to the matching spatial dimensions of the input image, with the number of output feature channels denoting the number of classes to be labelled. However, simply up-sampling from a down-sampled feature map that encodes high-level features results in a loss of finer details. As shown in [42], up-sampling by a stride factor of 32 from the final down-sampled feature map results in a very coarse output label map – this network is denoted FCN-32. To retrieve a finer, more detailed map, the feature maps from a shallower layer with lower-level details are fused with the deep, coarse up-sampled feature maps, similar to feature pyramids. The authors denote the copying of shallower layers as *skip connections.* In their implementation, a $1 \times 1$ convolution is applied to the feature map passed through each skip connection, before being fused with the corresponding up-sampled feature map through element-wise addition. The up-sampling layer increases the spatial dimensions of the feature map from the deeper layer by a factor of 2. Up-sampling is performed here through bilinear interpolation, which takes the distance-weighted average of the four nearest pixels to compute the resulting up-sampled pixel. The authors in [42] compare the fusion of the second-deepest layer with a final up-sampling layer of stride 16, denoted FCN-16, and the fusion of the third and second-deepest layers with a final up-sampling layer of stride 8,

denoted FCN-8. The results show progressively finer output maps, with FCN-8 achieving the best precision.

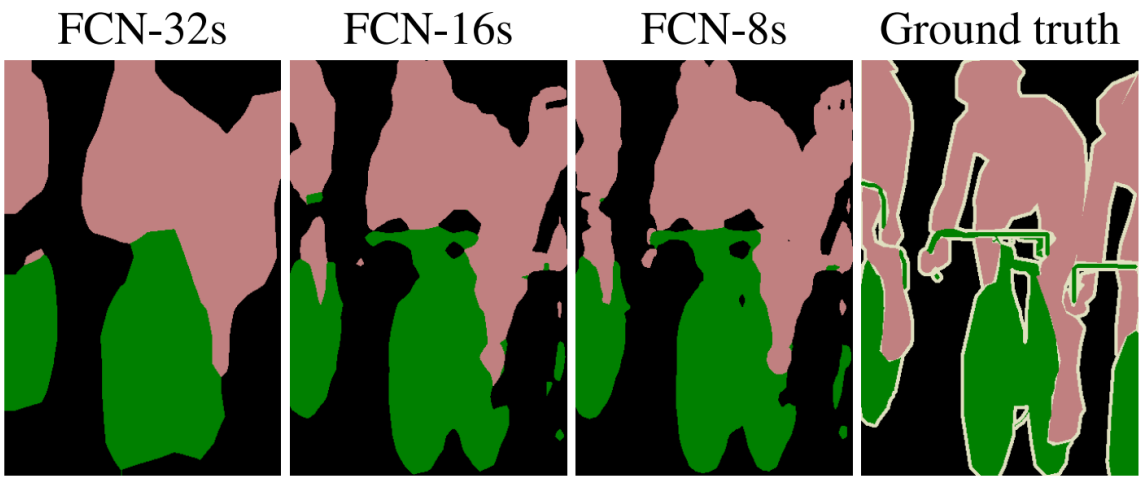


**Figure 8: Progressively finer output label maps [42]**

A well-known FCN-based network is SegNet [47]. This network consists of two main parts: an encoder that down-samples the feature maps at each step to gather greater context, and a decoder that reconstructs the segmented output image map through up-sampling layers. For each block in the encoder, there is a corresponding decoder block. The encoder consists of 13 convolutional layers from VGG-16. At the max-pooling step of each encoder block, the pooling indices are saved and passed to the corresponding decoder block via a skip connection, which is used to produce the sparse up-sampled feature maps. By passing the pooling indices instead of entire feature maps, the network memory is reduced. In [48], a SegNet-like network is proposed for segmentation and density evaluation in concrete surfaces. In [49], A pavement crack recognition system is developed using SegNet, in which the authors show its superior performance over FCN-8.

Expanding upon the idea of skip connections, [43] proposes an FCN-based end-to-end architecture for biomedical image segmentation, called U-Net. Similar to SegNet, the architecture of U-Net consists of an encoder as the contracting path, and a symmetric decoding expansive path that enables precise localization of low-level features. Unlike SegNet, in the contracting path, before down-sampling, a skip connection passes the entire feature map from the current block to the corresponding level in the expansive path, where

it is concatenated with an up-sampled feature map. The up-sampling layer consists of a bilinear interpolation followed by a 2 × 2 convolution that halves the number of feature channels. At each block of the contracting path and corresponding expansive path following the up-sampling and concatenation, two VGGNet-inspired unpadded 3 × 3 convolutions are applied. The U-Net architecture achieves very good performance for biomedical applications. U-Net has also been widely used for fault-based image segmentation, as it is able to perform segmentation precisely and efficiently.

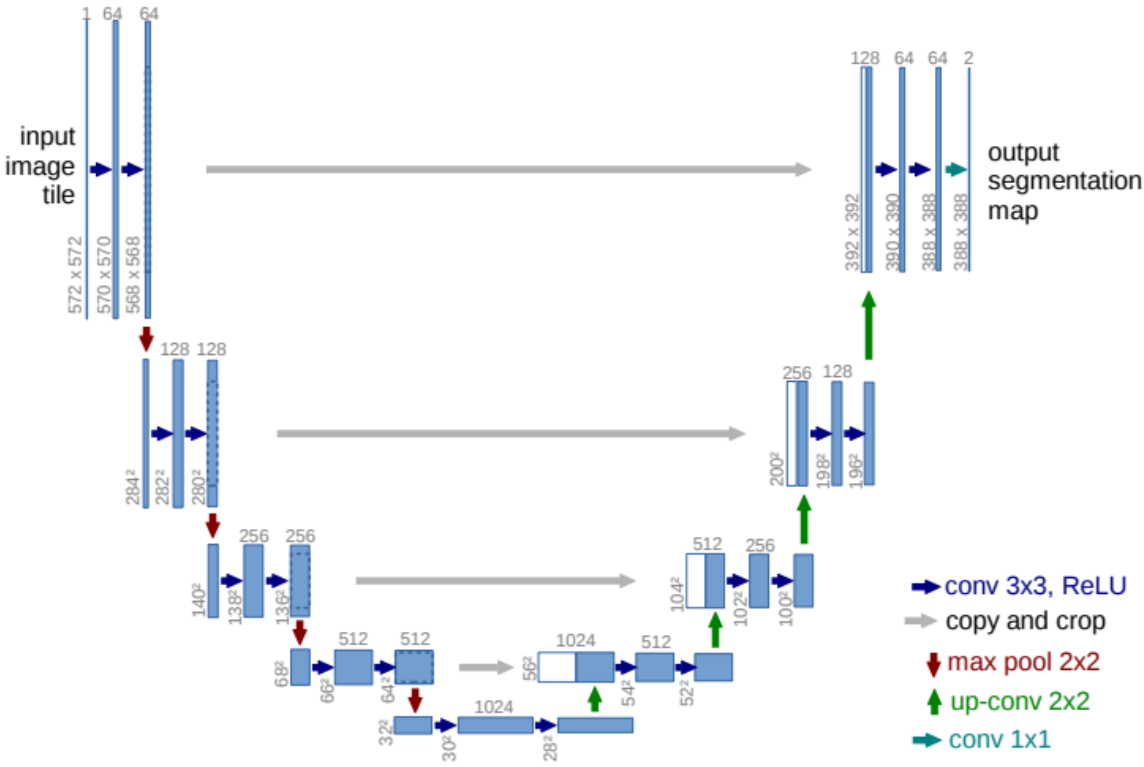


**Figure 9: U-Net Architecture [43]**

In [29], the authors propose modifications to improve the performance of U-Net for defect segmentation. One improvement is the addition of residual blocks at each block of the contracting and expanding paths, based on the residual connections introduced in ResNet. Another improvement is the inclusion of *dilated* convolutions that expand the kernel size by skipping pixels in the receptive field. By applying dilation at different rates, multiscale context can be extracted.

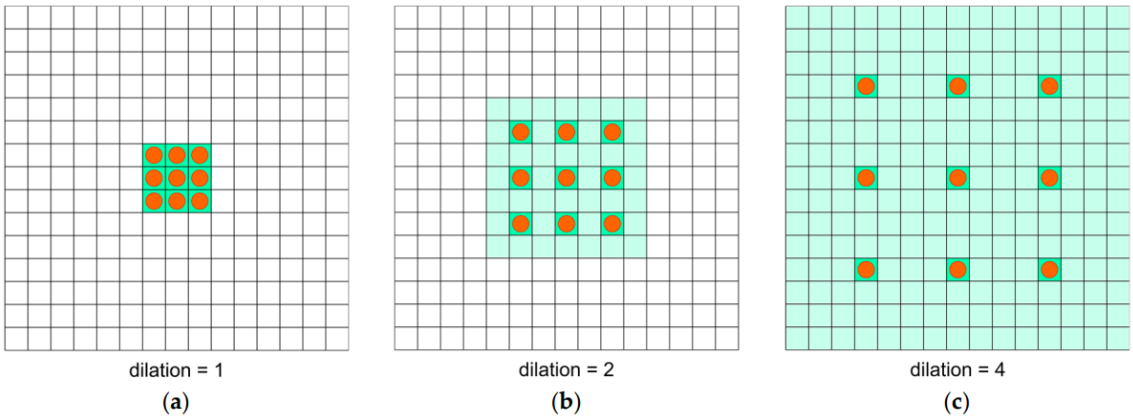


**Figure 10: 3 × 3 convolutions with different dilation rates: (a) – 1, (b) – 2, (c) – 4 [29]**

The authors in [29] propose applying multiple dilated convolutions in the bottleneck layer of U-net, which connects the output of the contracting path with the input of the expansive path. Furthermore, the authors note that instead of applying these dilated convolutions in parallel, a *waterfall* scheme that reuses the output of one dilation convolution as input to the next dilation convolution can outperform the parallel approach for segmentation tasks. A final addition is an *attention* block applied before the concatenation of the skipped connection and up-sampled feature map at each level. The attention block is used to amplify relevant information from the previous up-sampled layer while reducing the impact of background features. Testing combinations of these modified architectures on three different crack-based datasets, it was found that networks with residual blocks and the dilated convolutions outperformed the base U-net on all datasets, whereas the waterfall-based dilated approach and inclusion of attention block resulted in improved performance on one of the respective datasets.

Another U-Net based approach for pavement crack segmentation is proposed in [44]. Similar to [29], residual blocks are used in the contracting path, using a pretrained ResNet-34 network. In the expansive path, *fire modules*, introduced by the SqueezeNet architecture [50], are applied after concatenation. A fire module consists of a projection that decreases the feature space, and two parallel paths of different convolutional filter sizes to capture missing features from the previous layer, from which the outputs of each path are concatenated. Fire modules are very similar to the inception modules used in GoogLeNet.

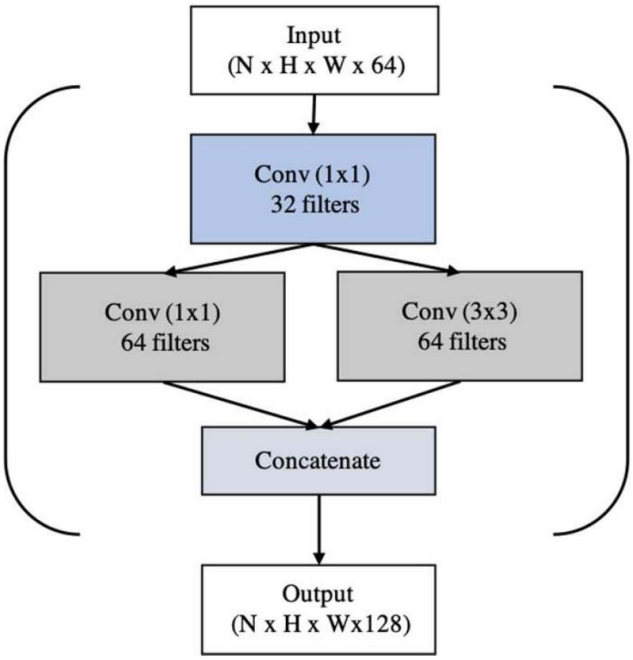


**Figure 11: A fire module used in Squeeze U-Net [45]**

Fire modules are shown to improve the performance of the base U-Net architecture. Moreover, fire modules reduce the number of parameters considerably compared to a standard convolution, making them useful for real-time applications where computational power is limited. In [45], the authors propose a network called Squeeze U-Net that uses fire modules for efficient image segmentation. They are used in [46] to enable real-time segmentation for autonomous driving.

## 3.2 Analysis of Drone-Based Inspection Solutions

In this section, a review of the studies that use image processing methods for drone structural fault inspection is conducted. Given the prevalence of deep learning for SHM, only studies published from 2017 onwards that use deep learning for image-based fault identification are considered. Although other variants of deep learning on multimedia exist, such as Deep Belief Networks (DBN), Auto Encoders (AE), and Recurrent Neural Networks (RNN), CNNs are the only deep learning approach to be investigated as they are the most prevalent in recent studies, particularly for images [12]. The studies are analyzed based on several key criteria:

- *Computational Constraints*: Studies that propose reductions to the model size for deployment to computationally and memory constrained on-board companion computers
- *Real-Time Capability*: Branching off computational constraints, whether the model is capable of real-time fault identification.
- *Fault Identification Method*: Indicates whether classification (C), fault detection (D), or semantic segmentation (S) of faults was used.
- *Automated Drone Tracking*: If tested on physical drones, it indicates whether the drone is automated to track around structures to some extent.
- *Obstacle Avoidance*: Indicates whether obstacle avoidance is employed.

**Table 1: Summary of 10 CNN Image-Based Drone Infrastructure Inspection Papers Published between 2017-2022**

| Reference | Computational Constraints Considered | Real-Time Capability | Fault Identification Method | Automated Drone Tracking | Obstacle Avoidance |
|---|---|---|---|---|---|
| **[17]** | x | x | C | ✓ | ✓ |
| **[51]** | ✓ | x | S | x | x |
| **[52]** | x | x | C | ✓ | ✓ |
| **[53]** | x | ✓ | S | x | x |
| **[54]** | ✓ | ✓ | D | x | x |
| **[55]** | x | x | D + S | x | x |
| **[56]** | x | x | C | x | x |
| **[57]** | x | x | C + D | x | x |
| **[58]** | ✓ | ✓ | C | x | x |
| **[59]** | x | x | S | x | ✓ |

According to Table 1, most drone-based solutions do not aim to reduce the complexity of their proposed models, with only 30% of the papers focusing on improvements for model efficiency on low-power drone-mountable devices. [54] and [58] leverage MobileNet, an efficient architecture designed specifically for mobile and embedded platforms [60]. The key behind the efficiency of MobileNet are depth-wise separable convolutions that replace standard convolutions. A variant of MobileNet designed for single-shot detection (SSD) of asphalt pavement distresses is used in [54] for real-time detection. MobileNet-V2, the next version of MobileNet, is used in [58] to enable on-board drone processing, while achieving high accuracy and real-time image processing at 7.4 FPS.

Also, most studies focus on the automated post-processing of images once collected by a drone but require the drone to be manually operated or make no explicit comments as to

how the drone tracks around civil structures during inspection. In [17], a Hexacopter drone is used to collect close-up image of structures, as it is highly stable and enables precise control. The drone is equipped with state-of-the-art sensors, enabling autonomous flight with minimal human intervention. However, the degree of autonomy is unclear. Furthermore, for the crack detection aspect, transfer learning is used to speed up training and reuse previously learned weights of a VGG-16 network pre-trained on ImageNet. The aim is to make training accurate and fast, rather than reducing the inference time for real-time applications. In [52], a simultaneous localization and mapping (SLAM) algorithm is used for autonomous navigation of a quadrotor drone in GPS-denied environments. Furthermore, a path revisit planning tool is integrated to revisit key points of the structure during inspection. The revisit planner takes in the output of a CNN that identifies cracks, to make informed decisions on potential crack points to revisit. However, there is a greater emphasis on providing real-time state estimation and obstacle avoidance for efficient autonomous drone navigation and less so on improving the efficiency of the crack detector. Moreover, the navigation system was only tested in indoor environments.

In general, there is minimal work that has been done to maximize autonomy for drone flight and tracking while optimizing CNN-based fault assessment models for resource-constrained, real-time applications. Many of the studies analyzed in chapter 3.1 look to reduce model complexity with the aim of achieving faster inference and a reduced memory footprint while approximating or achieving state-of-the-art performance. However, many of these studies validate and test their models with readily available datasets of pre-processed fault images captured in desirable conditions from ideal distances, which may fail to generalize to real-world images captured by drones. Most of the drone-based solutions focus on the post-processing of images once collected. Realizing a fully autonomous drone that can track around structures without any manual intervention is important to reducing manual operational barriers and costs. Another important aspect is the reduction of the computational overhead on power-constrained on-board devices. By reducing the power required to run these deep learning algorithms, the drone can preserve greater battery life to achieve longer flight times for continuous inspection and achieve real-time fault identification, opening the possibility of real-time decision-making during

inspection. The proposed system aims to address these aspects, as discussed in the later chapters.

# Chapter 4

# 4 Proposed Autonomous Drone Inspection System

In this chapter, the proposed fully autonomous drone-based structural inspection system is described. Namely, the components that drive the collection, communication, and processing of structural image data are explained at a high-level.

## 4.1 High-Level Architecture

The importance of a fully autonomous inspection system has been made apparent in the previous chapters. In light of the shortcomings of current drone inspection systems, the proposed system looks to address several key areas.

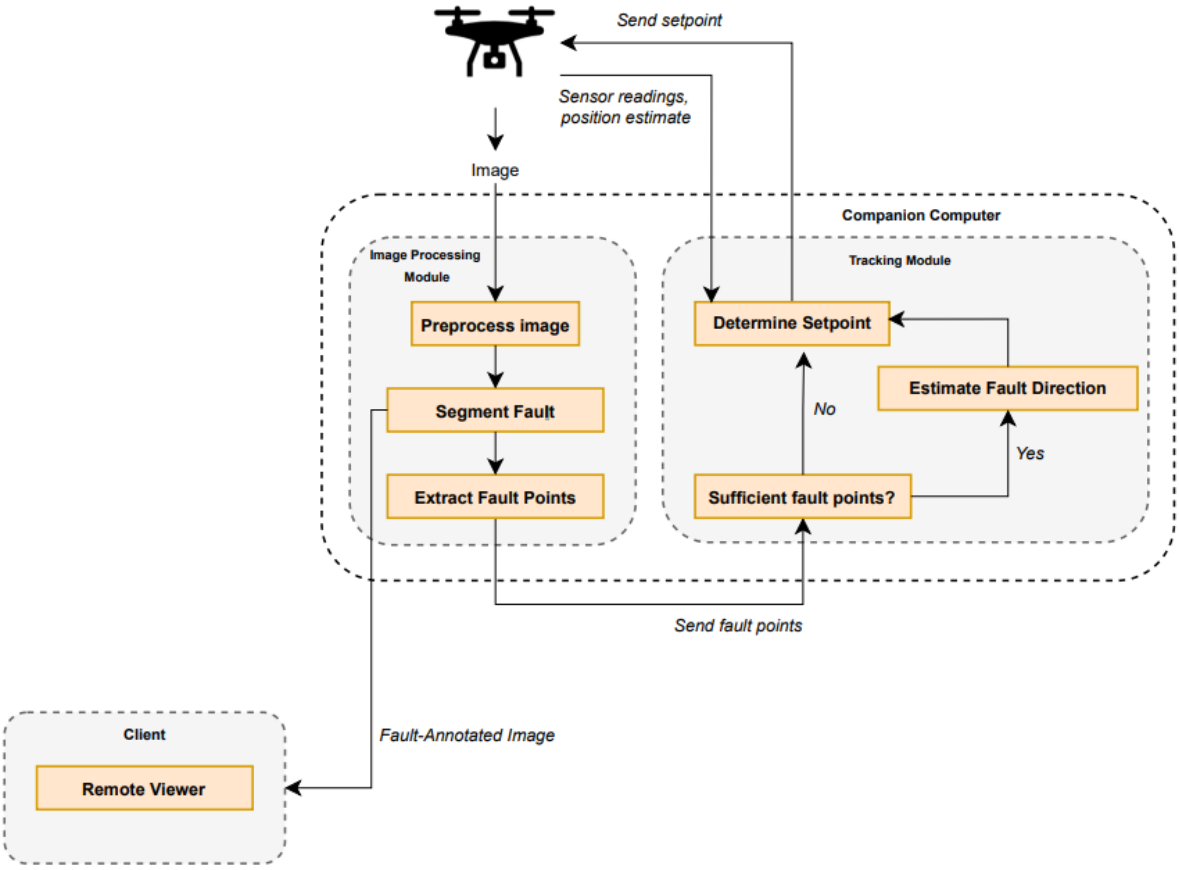


**Figure 12: High-Level Proposed System Architecture**

### 4.1.1 Autonomous Infrastructure Tracking

Fully automating drone navigation for infrastructure tracking is one key area the proposed system aims to address. To accomplish this, the drone is equipped with time-of-flight (ToF) ranging sensors, each one facing a certain direction (up, down, left, right, forward, back). A ToF sensor is a form of contact sensor that uses stimulated emission of electromagnetic radiation (laser) technology to perform distance calculations to objects, based on the time required to receive the reflected signal. Using these distance readings, the drone can determine its proximity to a given structure, enabling obstacle avoidance and exterior structural wall tracking. The downward-facing ToF sensor is particularly important to determining the drone's vertical height. Besides a downward-facing ToF sensor, a downward-facing camera can detect and measure the horizontal motion of surfaces as the drone navigates, enabling the drone to travel to desired setpoints. The ToF sensor distance readings, as well as the drone's position estimated by the flight controller, are sent to the tracking module running on the companion computer, from which it can determine setpoints to track along. Thus, the drone can be deployed from a station close to the structure of interest for regular autonomous inspection. Once the drone approaches the structure, several courses of action could happen:

- If the forward sensing ToF sensor on the drone detects that it is within a distance $D \leq$ the distance threshold $T$ of the structure, the drone will begin to track along it to the *right*.

- When the forward sensing ToF sensor detects that $D > T$, the drone will move *forward* into the open space.

- If the left-sensing ToF sensor on the drone detects that $D \leq T$ of the structure, the drone will *rotate counter-clockwise* by angle $\theta$ to face the wall, then track *right* along it.

- If the left and forward sensing ToF sensors detect that the drone is in a corner, and is within the threshold distance of the structure, the drone will *rotate clockwise* by angle $\theta$ to face the next wall, then track *right* along it.

- Once the drone detects a loop closure, such that it has completed the inspection around the entire structure, it will return to the base station.

In achieving this autonomous navigation, several assumptions are made:

- The GPS location of the structure is known, and the drone can travel to the GPS-specified location accordingly.

- During inspection, any obstacles near the structure, such as trees or telephone poles, can be avoided and disregarded for post-processing of the collected imagery.

- Rotation angle $\theta = 90^{o}$, as tests are conducted on straight-edged structural walls that orthogonally intersect each other.

- No loss of data during transmission to and from the drone.

### 4.1.2 Image Processing

As the drone navigates around the structure, it captures grayscale images in real-time. Mounted onto the drone is a front-facing camera capturing the images, which are sent to the processing unit on the companion computer. The companion computer is pre-loaded with a CNN-based model that can process and segment faults from images in real-time. Once processed and segmented, points corresponding to predicted fault pixels are extracted and mapped to the $x$, $y$ coordinate system, from which they are sent to the exterior wall tracking module on the companion computer. If a sufficient number of fault points are sent to the tracking module, a subroutine is invoked to track along the detected fault in real-time. This real-time dynamic tracking is important to determining the spread of a fault along certain areas of the structure without having to revisit such areas, saving drone flight

time and valuable battery life. Furthermore, a live stream of the fault-annotated images is sent via Wi-Fi to a client, for real-time visualization purposes.

## 4.2 System Design Choices

### 4.2.1 Drone Communication

Communication between the flight controller and companion computer is two-way. From the flight controller, estimated drone position and sensor readings are sent to the companion computer, and the companion computer uses this data to send back trajectory setpoints for the drone to travel along. In the proposed system, this communication can be done either UART, a serial hardware communication protocol, or via radio transmission. The images taken from the camera mounted to the drone are also sent to the companion computer, either via UART or Wi-Fi. On the hardware platform developed upon, the UART communication is limited in that only single characters can be sent at a time [61]. Thus, the proposed system uses radio transmission for communicating with the flight controller and Wi-Fi to stream images to the companion computer for processing. This image transmission approach via Wi-Fi is viable, given that the drone has its own access point (AP), to which the companion computer – with networking capabilities – in its proximity can connect to. In this setup, it is assumed that there is no loss in data as images are sent over Wi-Fi.

### 4.2.2 Inter-Module Communication

To facilitate communication between the modules running on the companion computer, a publisher-subscriber scheme is adopted. Each module has a node that either exchanges or receives messages over named buses called *topics*. Nodes that send data at a fixed rate to topics are called *publishers*, whereas nodes that subscribe to topics to receive messages sent by publishers are called *subscribers*. A node can publish data to multiple topics and can also have subscriptions to multiple topics, enabling one-to-many, many-to-many, and many-to-many communication [62]. The data message type can also be customized to better fit the needs of the application and reduce processing effort in packing and unpacking such messages. Moreover, data of any size can be sent at once, unlike socket connections

which fragment large data into smaller network packets. The publisher-subscriber scheme is important in the proposed system, as it enables continuous publishing of fault points from the image processing module to subscribing nodes that pass these fault points to the tracking module. Adopting this communication model allows the modules to run simultaneously as separate processes, rather than sequentially being invoked.

## 4.3 Development Platforms

*ROS*: The inspection system is built upon the Robotics Operating System (ROS), providing a set of tools and libraries for embedded development. ROS enables access and control of messages between nodes that operate a robotic system [78]. In the case of the proposed drone inspection system, ROS is used to facilitate the communication between different modules by providing a programmable interface for creating nodes – executable processes – that communicate with each other through publisher and subscriber topics over the ROS graph. ROS also provides predefined message types that are wrappers for ROS data types that can be sent to topics. In the proposed system, extracted points from the image processing module are passed to a publishing node, which publishes the points to a topic *t* that accepts a custom-built message type consisting of a list of points. Each point is defined as a Point type provided by the ROS *geometry_msgs* package, which contains wrappers for geometric primitive types such as points, vectors, and poses. On the other end, the tracking module then receives this list of points through a ROS subscriber node that subscribes to the custom Point list messages from *t*.

*Crazyflie*: The Crazyflie 2.1 is a miniature quadcopter developed by Bitcraze and is used to prototype the proposed system. The Crazyflie comes in a ready-to-built kit, including four 7 mm coreless DC-motors which can lift up to 42 g, including the weight of the Crazyflie itself without additional mounted hardware (27 g). The Crazyflie 2.X hardware platform is built on top of two microprocessors: an STM32F405 that handles all low-level and high-level flight control, including sensor reading, motor control, and telemetry, and an NRF51822 that handles radio communication and power management. The Crazyflie 2.1 is also equipped with a built-in inertial measurement unit (IMU) with 10 degrees-of-freedom. Furthermore, expansion decks can be mounted onto an expansion port to provide

enhanced capabilities in terms of sensing and positioning. The expansion decks communicate with the STM32F405 microprocessor over an expansion bus, which exposes communication buses and GPIO pins [79][80].

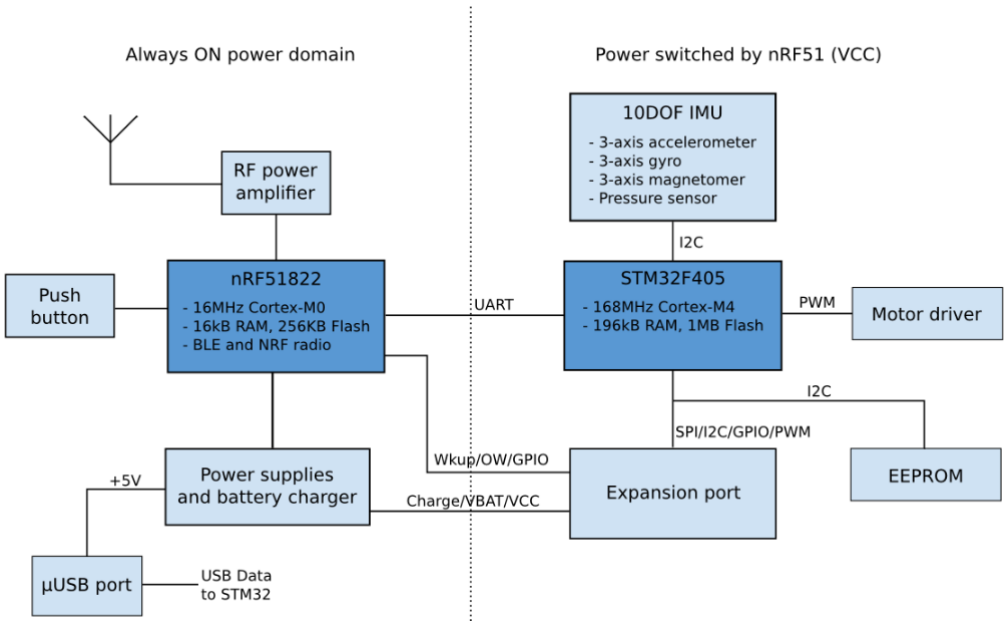


**Figure 13: The Crazyflie 2.X system architecture [80]**

For the proposed system's use-case, the expansion decks used include the following:

- *Flow deck v2*: Provides the Crazyflie with the ability to navigate by sensing both vertical motion through a VL53L1x ToF sensor that can measure vertical distances up to 4 m with mm precision, and horizontal motion through a PMW3901 optical flow sensor that measures horizontal movements in the *x*, *y* coordinate space, relative to the starting position [81].

- *Multi-ranger deck*: Gives the Crazyflie the ability to sense surrounding objects through VL53L1x ToF sensors pointing in five directions: front, back, left, right, and up. This deck enables obstacle avoidance for the proposed system [81].

- *AI-deck*: Capable of performing artificial intelligence-based workloads. It also consists of a Wi-Fi module and provides a Wi-Fi AP to stream images captured by an ultra-low power 320 × 320 Himax HM01B0 grayscale camera attached to the

AI-deck. However, given the constrained size and capabilities of the low power AI-deck processor and the limited UART communication between the AI-deck processor and the STM32F405 microprocessor on the Crazyflie [61], the image stream from the Himax HM01B0 camera is sent to an external device for processing. Also, due to the limited range of the access point on the AI-deck, this external device must be in close proximity, as stated in section 4.2.1. Thus, a larger, more powerful, yet portable edge device, that could viably be repurposed as a companion computer performing deep artificial intelligence operations on larger scale quadcopters is used in this study.

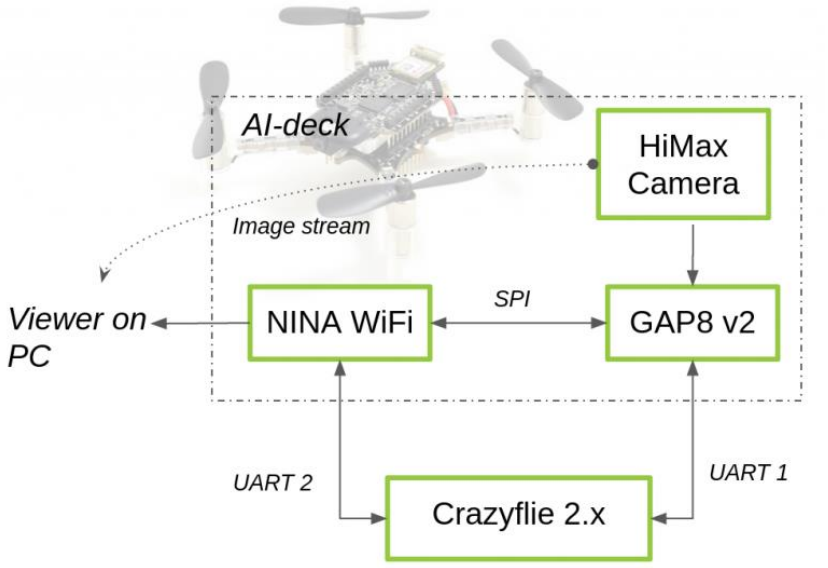


**Figure 14: AI-deck system architecture [61]**

From the software point-of-view, Bitcraze also provides a programmable interface to control the drone. Specifically, the Bitcraze client library, developed in the Python programming language, is leveraged to interface with drone commands for controlling the drone via radio transmission; the base commander [82] is used to send target setpoints for the *x*, *y*, *z* coordinates and drone rotation (yaw). The target setpoint is then sent as a packet to the Crazyflie via radio transmission using a packet protocol called Crazy Realtime Protocol (CRTP), designed to optimize packet prioritization enabling real-time control of the Crazyflie drone [83].

*Jetson Nano*: The Nvidia Jetson Nano Developer Kit is an embedded system-on-a-module, which is used as the companion computer in the prototype setup. It includes an integrated 128-core GPU, important for performing deep learning operations. Since the Jetson Nano is too large to be mounted onto a Crazyflie quadcopter, it is stationed near it, such that it is within range of the AP on the AI-deck.

# Chapter 5

# 5 Drone Tracking Method

In this chapter, the algorithmic approaches used for autonomous drone tracking, as well as the experimental results of the approach for structural fault tracking are described in detail.

## 5.1 Structural Tracking

While tracking around a structure, the drone can perform rotations, horizontal, and vertical movements. The drone is framed within the *global coordinate system*, where its position can be described by three degrees of freedom corresponding to the translational movements along the *x*, *y* and *z* axes; and its rotation can be described by one degree of freedom around the *z* axis. The drone also has its own its frame of reference in 3D space, based on its orientation, denoted as the *body coordinate system*. Affine transformations in 3D space enable the manipulation of 3D objects by altering their position and orientation. A 3D affine transformation can be expressed in matrix form as:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

That is, a position point $P = (x, y, z)^T$ can be transformed into position point $P' = (x', y', z')^T$ with matrix $M$ by applying matrix multiplication as $P' = MP$. $M$ can be expressed with a translation matrix as:

$$T = \begin{bmatrix} 1 & 0 & 0 & x' - x \\ 0 & 1 & 0 & y' - y \\ 0 & 0 & 1 & z' - z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$M$ can also be expressed with a rotation matrix. Rotation around the *z*-axis can be expressed as follows, with $\theta$ denoting the angle of rotation:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & -\cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This rotation matrix can be used to rotate *P* around the origin. However, being able to rotate around any arbitrary point in 3D space is necessary for transforming the drone's current position and orientation. To rotate *P* around a point *C* instead of the origin, *C* must first be translated to the origin of the global coordinate system before the rotation is applied. After performing the rotation, *C* is then translated back to where it was originally:

$$P' = T(C)\, R(\theta)\, T(-C)$$

The goal is to rotate *P* to *P*', to make the drone move in an opposing direction or update the axes of the body coordinate system when it yaws (rotation around the *z*-axis) by a certain angle. By letting *P* be the next setpoint without any transformation applied, with *C* being the current estimated drone position, the initial direction vector $\vec{r} = (P.x - C.x, P.y - C.y, P.z - C.z)^T$ is rotated to get a new direction vector $\vec{s} = (P'.x - C.x, P'.y - C.y, P'.z - C.z)^T$ that the drone travels along when instructed to change its direction of motion. The same approach is used to rotate the axes of the body coordinate system after yawing, such that subsequent direction commands would alter the drone's direction of motion according to its own coordinate system. Both scenarios are illustrated in Figure 15 below.

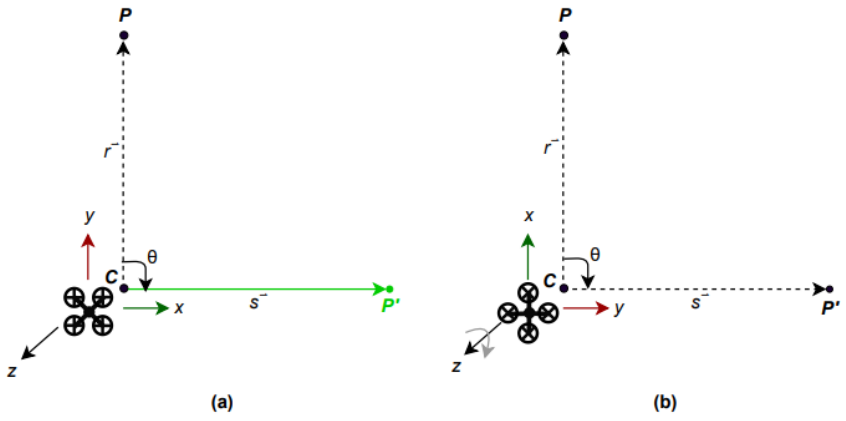


**Figure 15: (a) Changing the drone direction of motion; (b) rotating the drone**

## 5.2 Fault Tracking

When a fault is identified in real-time, the aim is to have drone react accordingly to follow along it. The direction of the fault spread can be determined given the extracted fault points. The direction can simply be defined by a linear model that best fits the fault points. Several methods exist for determining the best fitting linear model: *Least Squares* line fitting to minimize the sum of squared residuals between the predicted line and actual data points, and Principal Component Analysis (PCA), which determines the dimensions that contribute the most to the variance of the data. However, these approaches are sensitive to outliers. A more robust line fitting method that takes into the consideration the outliers is called Random Sample Consensus (RANSAC). The RANSAC algorithm takes in a dataset to find an optimal fitting result amongst the data points, by excluding outliers to find a linear model based upon the *inliers*. RANSAC uses a voting scheme to iteratively select the minimal set of random data points to fit the candidate model, forming a *consensus set*. This iterative process continues until a sufficiently high probability that all sampled points are inliers is satisfied. The number of iterations *k* required depends on several parameters: the determined outlier ratio *e* of the dataset, the probability *p* of sampling only inliers in all *k* iterations, and *n* minimum number of sampled data points required to estimate the model parameters. Hence, the inlier ratio is $1 - e$; this means the probability of sampling only inliers in a single iteration is $(1 - e)^n$, whereas the opposite probability of sampling at least one outlier is $1 - (1 - e)^n$. Furthermore, the probability of sampling at least one outlier in all *k* iterations is $(1 - (1 - e)^n)^k$, such that probability $1 - p$ of sampling at least one outlier in each of the iterations can be formalized as $1 - p = (1 - (1 - e)^n)^k$. Rearranging for *k* results in the following equation:

$$k = \frac{\log(1 - p)}{\log(1 - (1 - e)^n)^k}$$

After *k* iterations, the model with the *most* inlier points within a defined threshold distance *t* is chosen as the best fitting model.

**Algorithm 1:** RANSAC

```
iterations ⇐ 0
bestFit ⇐ Null
inliers ⇐ []
while iterations < k do
    p1 ⇐ getRandomPoint(Points)
    p2 ⇐ getRandomPoint(Points)
    model ⇐ linearModel(p1, p2)
    candidateInliers ⇐ []
    for each p ∈ Points do
        if distance(model, p) ≤ t then
            Append p to candidateInliers
        end if
    end for
    if length of candidateInliers > inliers then
        bestFit ⇐ model
        inliers ⇐ canditateInliers
    end if
    iterations ⇐ iterations + 1
end while
return bestFit, inliers
```

RANSAC is used in the proposed solution to determine the slope of the line corresponding to the estimated direction of the fault in 2D space. In each iteration, two points are sampled as the minimum inlier set for estimating the model parameters. Although RANSAC is robust to outliers, it can be expensive, due to the number of points and the number of iterations, particularly when the outlier ratio is high. Hence, to reduce the number of points that need to be considered in each iteration, the 2D point space is spatially divided into voxels and hierarchically stored into a *quadtree*. In a quadtree, each voxel is represented conceptually as a node in the tree, where each node has a defined capacity of points and four children nodes. If the capacity of a voxel *V* is exceeded, the points are split amongst four sub-voxels, represented conceptually as children of the node for *V*.

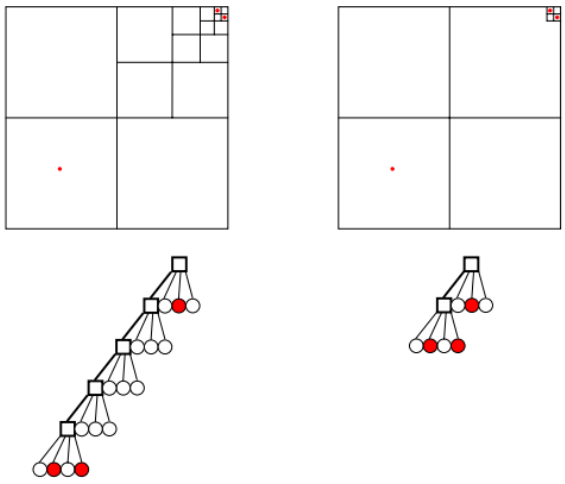

**Figure 16: Spatial representation (top) and conceptual representation (bottom) of quadtrees [84]**

Inspired by the approach in [65], all extracted fault points from each image frame are inserted into a quadtree. Once the capacity of the node is exceeded, the points in the parent node are passed into the corresponding children node, based on where their boundaries fall spatially in 2D space. As such, once all points are inserted, only the leaf nodes of the quadtree contain points. Then, a candidate line model is recursively generated using RANSAC and intersected with the quadtree's leaf nodes to determine which nodes intersect with the line – only points within these nodes are considered for determining inliers to the model. Namely, points within a threshold distance *t* of the generated candidate line intersecting the leaf nodes containing these points are considered inliers. As a result, only points within these intersection leaf nodes are considered, rather than the entire point set, resulting in a reduction of the time complexity for point-to-line comparisons to a logarithmic factor of the number of points. *Euclidean distance* is used to define the distance between a point *p* on the candidate line and a fault point *q* in *n*-dimensional space, where $n = 2$ in this case:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

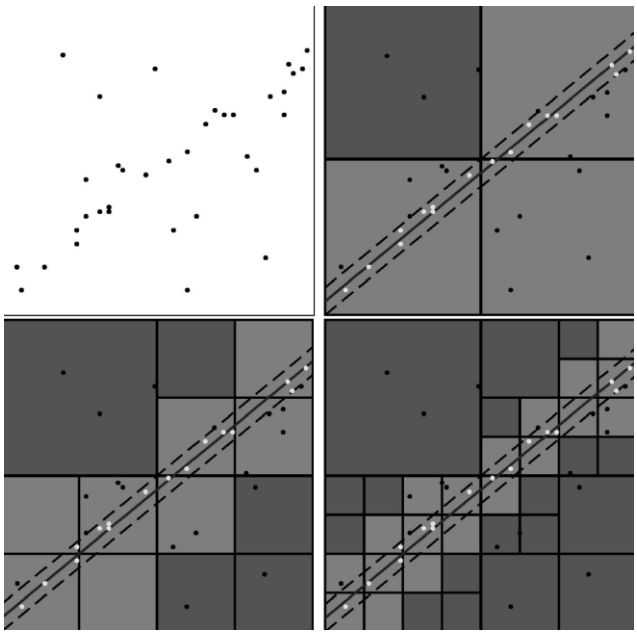

**Figure 17: Top left – Point set. Top right – a generated candidate line intersecting a quadtree (represented as voxels), with the dashed lines representing the inlier threshold. Bottom Left and Right – the complete quadtree, approximating the density of the points [65]**

Another improvement proposed for a more efficient RANSAC approach is a smart-sampling approach that takes advantage of the hierarchical nature of the quadtree to reduce the randomness of the minimum inlier sampling process. Namely, instead of sampling two random points from the 2D model space, an initial point is randomly sampled, and a subsequent point in relative proximity to that point is sampled. First, a randomly selected neighbouring leaf node to that from which the first point was sampled is checked for any points. If enough points exist, a random point from that neighbour is chosen as the subsequent point. Otherwise, the next randomly neighbouring leaf node is checked. If all the neighbouring leaf nodes – children to the current parent – have been visited and none of them contain enough points, the quadtree is recursed upwards, and the grandchild leaves that have not already been visited are checked. By sampling two points that are relatively close to each other, there may be a greater probability that these two points represent a coherent model, rather than two randomly sampled points, thus a greater probability that a more optimal solution can be reached within $k$ iterations. As shown in [86], carefully

selecting samples in close proximity to each other reduces the number of iterations required to detect a shape with a certain probability. However, points that are too close to each other, or in other words, within the *same* node may lead to spurious model estimations. Hence, points from the same leaf node are not selected, unless no other nodes contain enough points. To prevent selection of points in a sparsely populated node that are unlikely to model the dataset, only nodes with a cardinality greater than or equal to *half* the node capacity are chosen.

**Algorithm 2:** Modified RANSAC

```
while iterations < k do
    chosenLeaves ⇐ []
    leaf ⇐ quadtree.getRandomLeaf().setVisited()
    Append leaf to chosenLeaves
    while length of chosenLeaves < n do
        neighbours ⇐ leaf.getNeighbours()
        neighbour ⇐ neighbours.chooseRandomNeighbour()
        if neighbour.isVisited() is False then
            neighbour.setVisited()
            if neighbour.isLeaf() and ≥ capacity/2 then
                Append neighbour to chosenLeaves
            else
                neighbour ⇐ chooseRandomChild()
            end if
        else if all neighbours have been visited then
            leaf ⇐ leaf.getParent()
        end if
    end while
    p1 ⇐ getRandomPoint(chosenLeaves[1].getPoints())
    p2 ⇐ getRandomPoint(chosenLeaves[2].getPoints())
    model ⇐ linearModel(p1, p2)
    intersectedLeaves ⇐ quadtree.intersect(model)
    candidateInliers ⇐ []
    for each p ∈ intersectedLeaves do
        if distance(model, p) ≤ t then
            Append p to candidateInliers
        end if
    end for
    if length of candidateInliers > inliers then
        bestFit ⇐ model
        inliers ⇐ canditateInliers
    end if
    iterations ⇐ iterations + 1
end while
return bestFit, inliers
```

Although the modified RANSAC algorithm is tailored to find the general slope of the fault direction, it is still indeterminant whether the drone should travel up or down along this slope. Hence, the spread of the fault points mapped to the *x*, *y* coordinate system is determined, such that more fault points in the upper half of the view frame signals the drone to travel upwards at the predetermined slope; and conversely, more points in the lower half signals the drone to travel downwards at the predetermined slope when the drone is within a threshold distance $D \leq T$ of the structural wall. Given that the drone may travel in one direction initially, and then travel back along with opposite direction once the endpoint of a fault is reached, a loop closure detector is included to determine if the drone returns to the initial position where it first detected the fault; once reached, the drone will stop tracking along the fault and continue its default tracking around the structure.

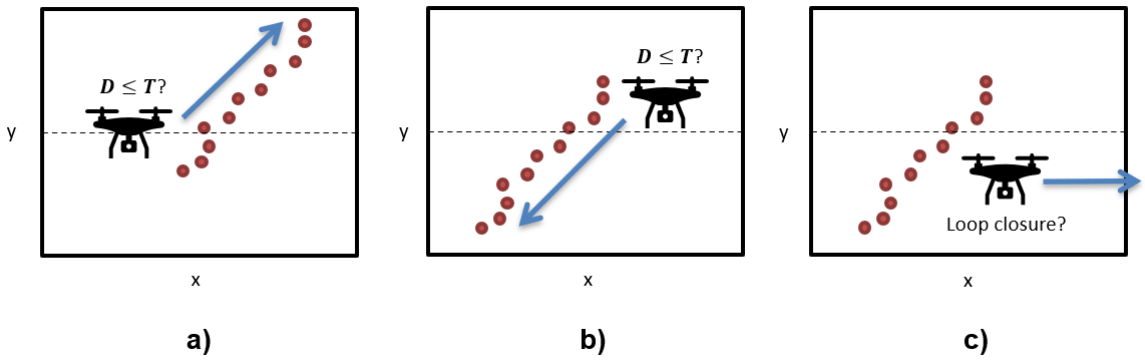


**Figure 18: Mapping extracted fault points to *x*, *y* coordinate view frame: (a) more points in the mapped to the upper half of the view frame; (b) more points in the mapped to the lower half of the view frame; (c) loop closure detected**

## 5.3 Evaluation Criteria

To evaluate the performance of modified RANSAC algorithm, the *mean squared error* (MSE) metric is used, which calculates the average of the squared differences between the predicted values and actual values. MSE is calculated as follows:

$$\frac{1}{n} \sum_{i=0}^{n} (Y_i - \hat{Y}_i)^2$$

Where $n$ is the number of data points, $Y_i$ are the actual values, and $\hat{Y}_i$ are the predicted values. MSE always results in a positive error value, and is a standard metric for evaluating the loss, particularly for determining how well a line fits a set of data points. However, since MSE is a squared loss function, it penalizes large errors more heavily. Since large errors are particularly undesirable for estimating the line of best fit produced by RANSAC on the inlier points, MSE is useful for this purpose.

Also evaluated is the *running time* to determine the efficiency of the modified RANSAC algorithm and its viability for real-time performance.

## 5.4 Experimental Results

*Test Setup and Parameters*: The original (baseline) and modified RANSAC algorithms are tested and compared on a system with 13 GB of RAM and an Intel Xeon CPU at 2.20 GHz. Several user-defined parameters values are chosen based on empirical testing. Namely, in performing RANSAC, an inlier threshold of 10 is set, and a probability of selecting only inliers in all iterations is set to 0.9999. The outlier ratio is determined programmatically, based on the spread of each data point in relation to the mean, determined by the *z*-score:

$$Z = \frac{(x - \mu)}{\sigma}$$

Where $x$ is the observed value, $\mu$ is the mean of all observations, and $\sigma$ is the standard deviation of all observations. The user-defined condition for an observed data point belonging to the inlier set is if -2 < *z*-score < 2; otherwise, the observation is considered an outlier. Also, for initializing the quadtree used in the modified RANSAC algorithm, each node in the quadtree is set with a capacity of $1/10$ the total number of points.

*Study One*: The line fitting performance of the baseline and proposed modified RANSAC algorithms are evaluated in this study. The MSE of the resulting line of best fit on the inlier set outputted by RANSAC is determined at varying distributions of the data defined by $\sigma$. At each $\sigma$ value, RANSAC is run 100 times to ensure reliability in the results and tested upon points that form a line, with random Gaussian noise introduced. As shown in Figure 19 below, the MSE for both the baseline and modified RANSAC increases as $\sigma$ increases,

as expected. However, as $\sigma$ approaches larger values, the MSE of the baseline begins to increase more quickly relative to the modified RANSAC.

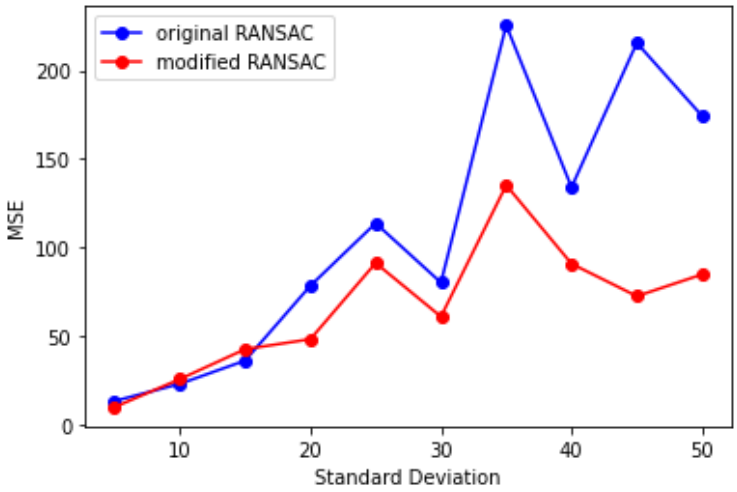


**Figure 19: The MSE between the resulting line of best fits and corresponding inliers at different standard deviations of the data**

*Study Two*: In this study, the predicted slopes of the outputted RANSAC models are compared to the actual slope. MSE is used to measure the sum of squared differences between the predicted slope value and a user-defined target slope value at different values of $\sigma$. As in study one, at each $\sigma$, RANSAC is run 100 times and tested upon points that form a line, with random Gaussian noise introduced. As shown in Figure 20 below, the MSE of the modified RANSAC is generally lower than that of the baseline, with the exceptions occurring at $\sigma = 10$, $\sigma = 15$, and $\sigma = 30$.

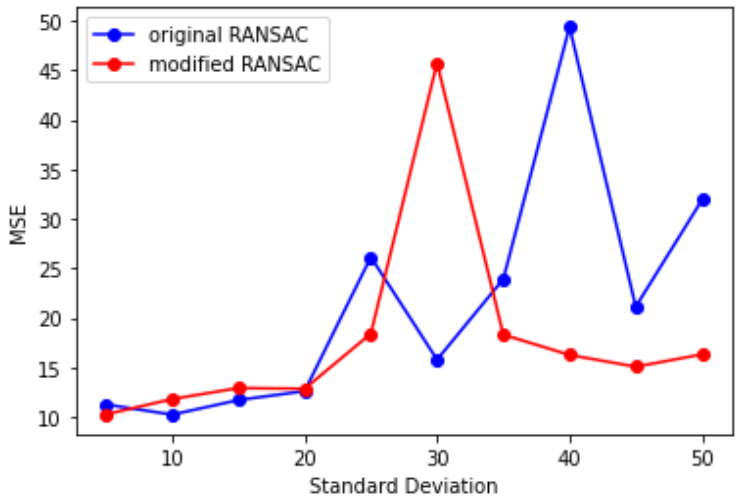


**Figure 20: The MSE between the predicted slopes and actual slopes at different standard deviations of the data**

*Study Three*: In this study, the running times of the baseline and modified RANSAC algorithms are compared. Investigated is the impact on the running time as a) the number of total points in the sample varies and b) the value of $\sigma$ impacting the sample distribution varies. Thus, RANSAC is run at varying samples sizes, where at each size, it is run 10 times to ensure enough reliability in the results. For each of the 10 runs, the average running time is used. In Figure 21 below, when $\sigma$ = 10, as the number of points in the sample increases, the running time of the modified RANSAC increases more quickly than that of the baseline. This is due to the overhead in rebuilding the quadtree for each RANSAC innovation when a new data sample is introduced (each frame). However, in Figures 22 and 23 below, when $\sigma$ = 20 and $\sigma$ = 40 respectively, the running time of the modified RANSAC grows less quickly relative to the baseline as the number of points increases. This is likely because the greater spread of the data imposed by a larger $\sigma$ increases the outlier ratio, which in turn increases the number of iterations in each invocation of the RANSAC algorithm; increased iterations means more points that need to compared to the candidate line, and since the modified version reduces the point-to-candidate line comparisons, this reduction overrides the overhead in reconstructing the quadtree when the sample size is large (>= 10,000). Although fault points extracted in a single image frame may be small in comparison to the total pixels in a frame, higher resolution frames where

faults only account for 5-10% of pixels can still easily produce 10,000 or more points, making the modified algorithm a viable approach for near real-time fault tracking, even when the number of fault points is large, reaching a maximum of 0.6 seconds processing time on a sample of 10,000 points with $\sigma$ = 40. However, further optimizations in initializing and constructing the quadtree may result in further decreased running times.

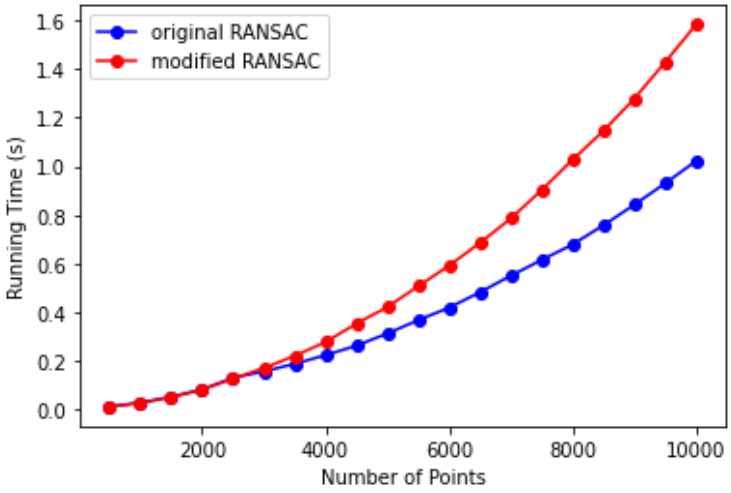


**Figure 21: The relationship between the number of points and the running time when $\sigma$ = 10**

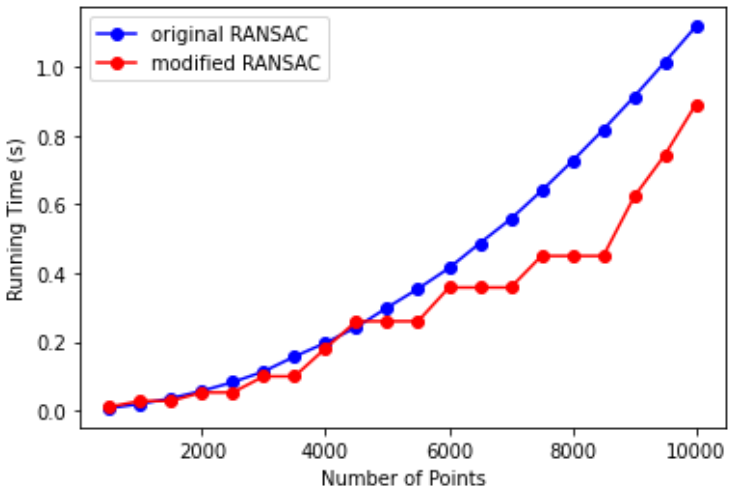


**Figure 22: The relationship between the number of points and the running time when $\sigma$ = 20**

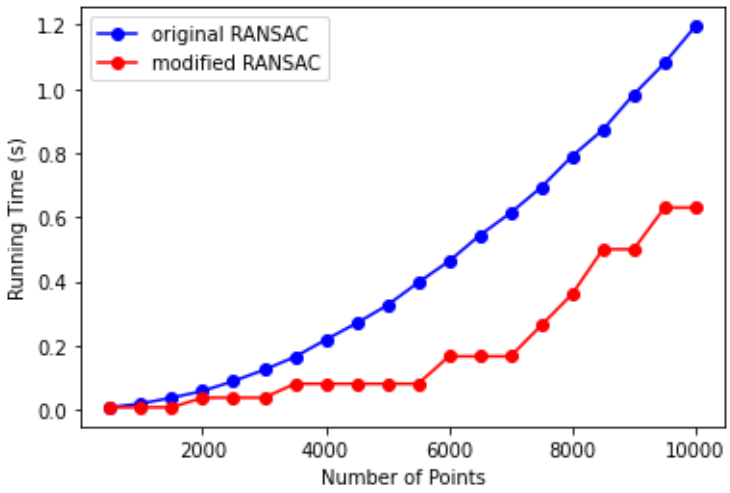


**Figure 23: The relationship between the number of points and the running time when $\sigma$ = 40**

*Study Four*: Also investigated is the visual performance of the modified RANSAC algorithm, on simulated data points with Gaussian noise and actual crack points with noise introduced. As shown in Figure 24 below, modified RANSAC is robust to outliers, as it can still pick out the points best representing a linear relationship in the data amongst noise.

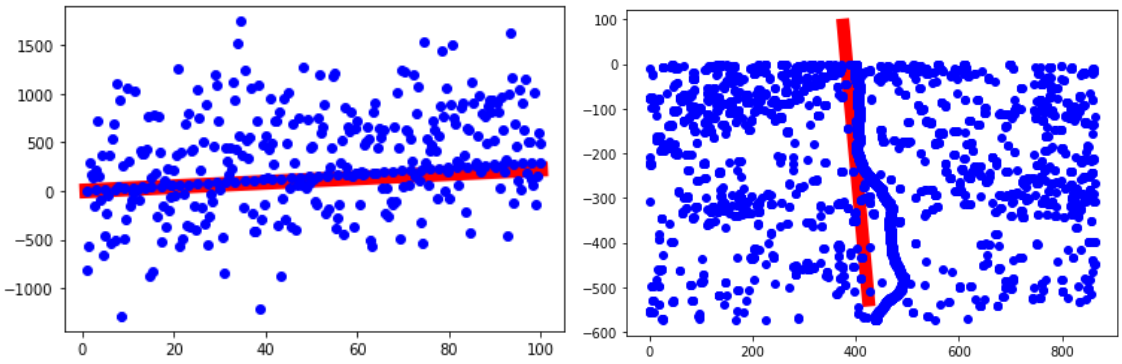

**Figure 24: Performance of the modified RANSAC algorithm on detection of a line with Gaussian noise (left) and on detection of points associated to a crack (right)**

# Chapter 6

# 6 Pixel-wise Fault Segmentation

In this chapter, a semantic segmentation model for pixel-wise image fault segmentation is proposed. Pixel-wise segmentation enables more precise tracking of faults along structural exteriors. The aim is to reduce the number of parameters and convolutional operations for real-time, low-power operation on edge devices, while achieving at or above a benchmark level of performance. U-Net is used as the base architecture for the proposed model, given that it is designed specifically for semantic segmentation tasks and is known for its ability to extract cracks precisely and efficiently at the pixel level. The proposed model is modified, in which the modified versions are compared in terms of several key metrics to determine which modifications most affect performance and inference speed on edge devices.

## 6.1 Analysis of Architectures Designed for Efficiency and Performance

The proposed model architecture uses some of the design choices employed by several key architectures that focus on efficiency through reduced network latency during inference as well as increased performance through spatial and feature attention. These networks are chosen due to their popularity and efficient performance.

### 6.1.1 Efficient Neural Network (ENet)

In [63], a novel CNN architecture named ENet (efficient neural network) is proposed for low latency operations in mobile applications. This architecture employs an encoder-decoder scheme similar to U-Net and SegNet. Namely, an input image is passed into an *initial* block, which performs a $3 \times 3$ convolution with stride 2 in parallel with max pooling, with the respective results concatenated. The rest of the network consists of *bottleneck* blocks (inspired by ResNet), which perform a single $3 \times 3$ convolution on projected lower-dimensionality feature maps on an extension branch in parallel with max pooling on the main branch when down-sampling (or $2 \times 2$ transpose convolutions with stride 2 when up-sampling), with the respective results summed.

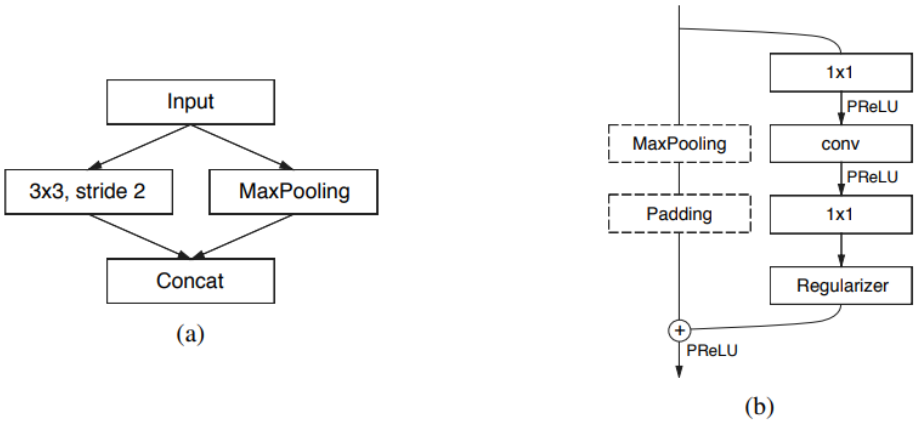


**Figure 25: (a) ENet initial block; (b) ENet down-sampling bottleneck block [63]**

The authors note several key design choices to improve the efficiency and performance of ENet, based on experimental results and intuition:

- *Reduced Down-sampling*: Although down-sampling is important to gathering greater context from reduced feature map resolutions, heavy down-sampling can lead to loss of spatial information, such as edge shape – this can be detrimental to detecting fault edges, particularly those belonging to cracks. Also, strong down-sampling requires equally strong up-sampling, which increases the model size and computations. Thus, the authors aim to limit down-sampling, and aim to garner greater context from down-sampled feature maps through dilated convolutions.

- *Early Down-sampling*: Processing large input feature maps early in the network is expensive. Visual information is highly spatial redundant and can be reduced into a compressed feature map for classification purposes. In ENet, the first two blocks focus on down-sampling with minimal feature maps produced.

- *Saving Max Pooling Indices*: As proposed in SegNet, the indices of the max pooling are saved and passed to the corresponding decoder block, reducing the memory requirements compared to copying the entire feature map.

- *Parallel Convolution and Pooling*: The authors in [63] note that pooling after a convolution is computationally expensive. By performing the pooling in parallel with the convolution and concatenating the resulting feature maps, a 10-fold speed-up of the inference time of the initial block was achieved.

- *Projection*: In each bottleneck block, projection reduces the dimensionality of the feature maps before a convolutional filter is applied, greatly reducing the number of the parameters and convolutional operations.

- *Factorizing Filters*: In addition to dilated convolutions, asymmetric convolutions decompose an $n \times n$ convolution into two smaller convolutions of $1 \times n$ and $n \times 1$ size. The authors use asymmetric convolutions with $n = 5$, noting that the cost of $5 \times 1$ and $1 \times 5$ together are similar to that of a single $3 \times 3$ convolution. This enables increasing the receptive field of the filter without significant additional computational cost.

### 6.1.2 Squeeze-and-Excitation Networks

Being able to highlight meaningful features and spatial regions across input channels while suppressing less relevant ones can allow CNNs to better focus on salient properties. In accordance with this notion, *Squeeze-and-Excitation* networks are introduced in [66], which consist of Squeeze-and-Excitation (SE) blocks that act as an attention mechanism to adaptively reweight feature map responses across the channel space. The SE block squeezes spatially to aggregate all feature maps across their spatial dimension and excites the squeezed tensor along the channel dimension to produce a set of per-channel weights, also known as a spatial squeeze and channel excitation block (cSE). Formally speaking, this involves taking a feature map tensor $U$ as input, where $U \in \mathbb{R}^{N \times C \times H \times W}$, with $N$ representing the batch size, $H$ and $W$ representing the feature map spatial resolution and $C$ representing the channel space. Performing a squeeze operation reduces $U$ to $\hat{U} \in \mathbb{R}^{N \times C \times 1 \times 1}$ through a global average pooling layer, before it is passed into an excitation module consisting of a multi-layer perceptron bottleneck, followed by a sigmoid activation function applied to the output tensor to rescale the activations to [0, 1]. The resulting tensor

$\hat{U}_{cSE}$ consisting of the recalibrated per-channel weights is then element-wise multiplied with the original $U$. By performing the squeeze operation, this ensures lower computational complexity compared to computing the per-channel weights over the full tensor and encapsulates information from the entire spatial receptive field in computing each reweighted feature map $c$ of $U$. Moreover, SE blocks are relatively simple in structure, and thus can be easily used in existing architectures, adding only a slight increase in model complexity and computational cost. In the case of FCNs, applying SE blocks as an attention mechanism at the skip connections can suppress irrelevant regions and poor feature representation [66].

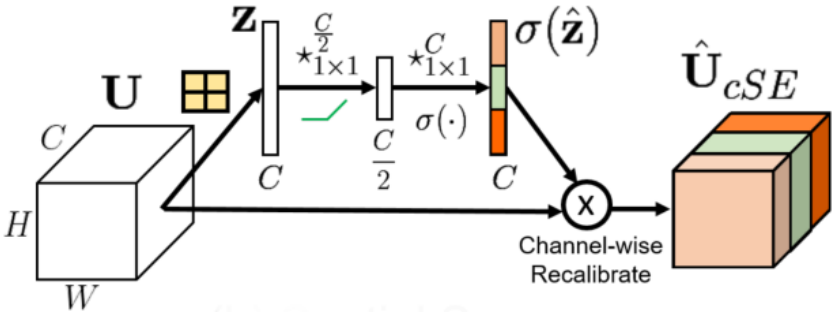


**Figure 26: An cSE block, the conventional SE block [67]**

Variations of the traditional cSE block have also been proposed. A channel squeeze and spatial excitation block (sSE) reduces the feature map tensor by applying a squeeze operation over the channel dimension and exciting over the spatial dimension, to highlight more relevant spatial locations and suppress irrelevant ones. This involves taking $U$ as input, where $U \in \mathbb{R}^{N \times C \times H \times W}$ and performing a squeeze operation to reduce $U$ to $U \in \mathbb{R}^{N \times 1 \times H \times W}$, by applying a $1 \times 1$ convolution to project the number of channels $C$ to 1. Each value of the projected tensor represents a linear combination of the representation of $C$ for a spatial location $(i, j)$. Next, a sigmoid activation function is applied to the reduced tensor. The resulting tensor $\hat{U}_{sSE}$ consisting of the recalibrated spatial weights is then element-wise multiplied with the original $U$, where each value in $\hat{U}_{sSE}$ corresponds to a weight of the importance of a spatial location [67].

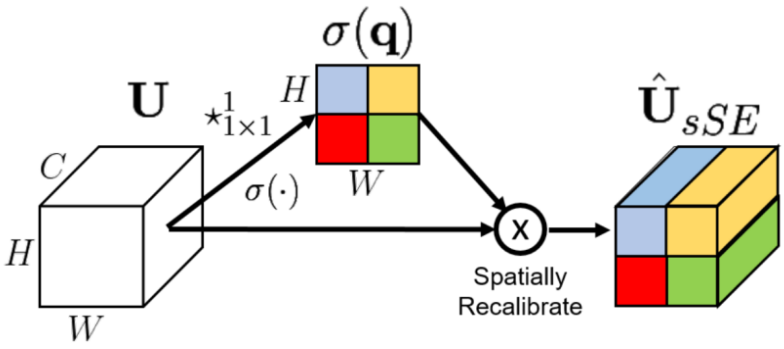


**Figure 27: An sSE block [67]**

Another variation of the cSE block involves a combination of cSE and sSE, which concurrently reweights both the spatial and channel-wise feature map responses through concurrent spatial and channel squeeze and channel excitation (scSE). Taking $U$ as input, where $U \in \mathbb{R}^{N \times C \times H \times W}$, $U$ is passed in parallel through to an cSE and sSE block, where the resulting tensors $\hat{U}_{cSE}$ and $\hat{U}_{sSE}$ are element-wise summed to produce $\hat{U}_{scSE}$. The scSE block more heavily reweights a location ($i$, $j$, $c$) in $U$ where there is a higher activation response denoting a location of high relevance [67].

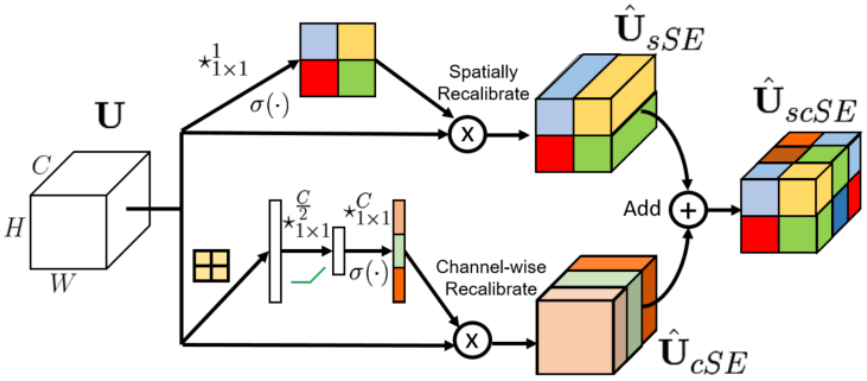


**Figure 28: An scSE block [67]**

To determine the complexity (in terms of the number of parameters) of an FCN consisting of scSE blocks within encoder-decoder blocks, consider an output feature map of $C$ channels. An cSE block introduces $C^2$ weights and an sSE block introduces $C$ weights. So, the model complexity with $n$ encoder-decoder blocks is:

$$\sum_{i=1}^{n}(C_i^2 + C_i)$$

Where $C_i$ is the number of output channels for the ith encoder/decoder block [67]. Based on experiments conducted in [67], the scSE block increases the number of parameters by 1.5%, which is a small increase to the overall network complexity.

### 6.1.3 MobileNets

In [60], *MobileNets* are introduced as a class of efficient models designed for embedded deep learning applications on mobile and edge devices. The main contributions in [60] are the use of depth-wise separable convolutions replacing each standard convolution layer in the network, and the introduction of hyperparameters that present a tradeoff between the model latency, size, and accuracy: a *width* multiplier $\alpha$ that reduces the number of channels in each layer and a *resolution* multiplier $\rho$ that reduces the input image resolution and every subsequent feature map resolution.

In [68], *MobileNetV2* is proposed as an improvement to the original MobileNets, which introduces *inverted residual blocks* with depth-wise separable convolutions and *linear bottlenecks*. A conventional bottleneck block first reduces the channel dimension of the input tensor of size $N \times C \times H \times W$ by a factor of $s$, through a $1 \times 1$ projection, resulting in a tensor of size $N \times \frac{C}{s} \times H \times W$. Next, a $3 \times 3$ convolution is applied to the reduced tensor, before it is projected back to the original size through another $1 \times 1$ convolution. In MobileNetV2, the $3 \times 3$ convolution is replaced with a depth-wise separable convolution in all bottleneck layers (excluding the initial layer). The bottleneck layers are inverted, such that the dimensionality of the input tensor is first *increased* by a factor $s$ through a $1 \times 1$ projection, resulting in a tensor of size $N \times sC \times H \times W$, before being decreased back to the original size $N \times C \times H \times W$. When down-sampling, a stride of 2 is used instead of pooling. Furthermore, a shortcut residual connection is conditionally added to perform element-wise addition between the input feature map and outputted bottleneck feature map – residual connections are omitted when down-sampling, as the $3 \times 3$ depth-

wise separable convolution performs a stride of 2. The inverted bottleneck layers and the shortcut residual connection make up the inverted residual block.

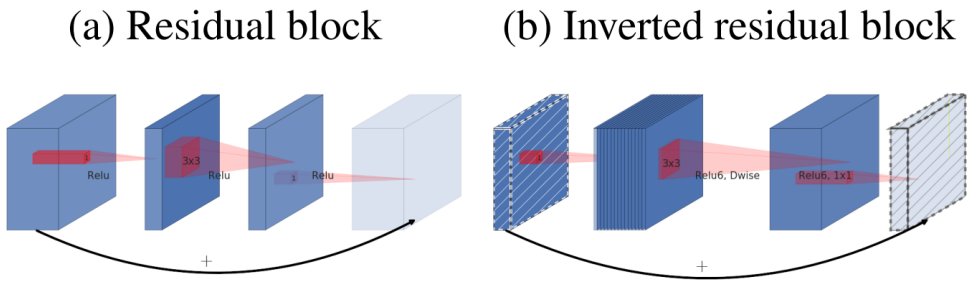


**Figure 29: The architecture of (a) a residual block and (b) an inverted residual block [68]**

The importance of using these shortcut residual connections stems from the vanishing gradient and degradation problems addressed in ResNet, and the fact that the bottlenecks contain the necessary information needed to be saved and passed to the next block. As shown in Figure 30 below, the shortcut between bottlenecks results in the highest accuracy and fewest operations.

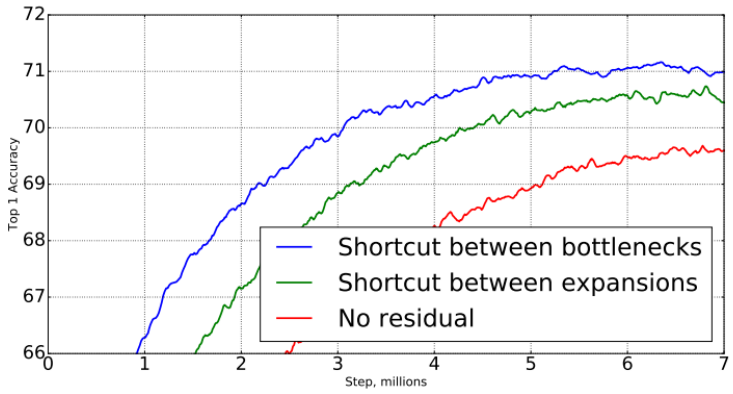


**Figure 30: The impact of the inclusion and location of shortcut residual connections on the accuracy and number of operations [68]**

The inverted residual block also uses considerably less memory than the conventional residual block. This is due to the inverted design, the shortcut residual connection between the bottlenecks, and the depth-wise separable convolutions; the total memory usage would

be dominated by the size of the bottleneck tensors, given that the expansive part consists of a memory-inexpensive depth-wise separable convolution. They are inexpensive because the depth-wise convolution part is performed on a per-channel basis of an inner tensor $L$, enabling $L$ to be represented as a channel-wise concatenation of $t$ intermediate tensors. If $L$ consists of $n$ channels, then each of the $t$ tensors is of channel size $n/t$. Given the constraint that only one intermediate block of size $n/t$ is always required in memory, and that a depth-wise convolution operates independently on single channels, this means $n = t$, such that only 1 channel is required to be kept in memory.

Linear bottlenecks are also introduced in MobileNetV2, wherein the non-linear activation function applied after the last convolution of the residual block is replaced with a linear activation function. Experimental evidence shows that non-linear activation functions, such as ReLU, can result in information loss as values less than 0 get discarded. Reducing the feature space from a higher to lower dimension, as does the final convolution of the inverted residual block, while applying a non-linear activation function, discards a significant amount of information. Hence, a linear transformation that preserves non-zero values is applied after the final convolution instead. As shown in Figure 31 below, not only does a linear bottleneck result in better accuracy, but also fewer operations.

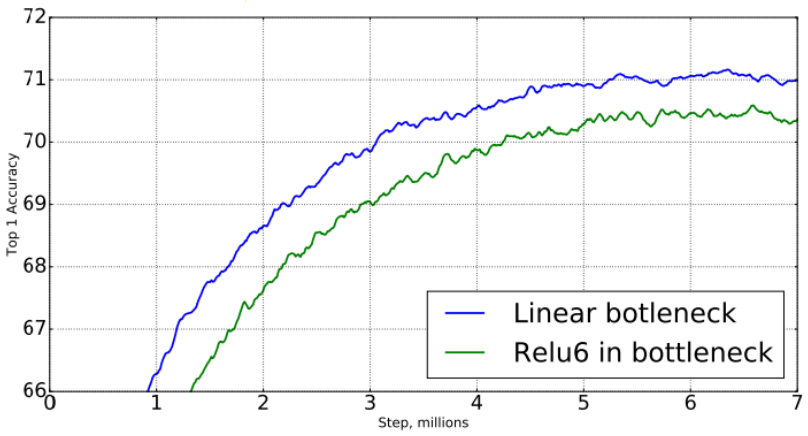


**Figure 31: The impact of non-linearity on the accuracy and number of operations [68]**

### 6.1.4 EfficientNets

A family of models, called *EfficientNets*, is introduced in [69]. The authors study the impact of uniformly scaling the network depth, width, and image resolution, and apply this scaling method to obtain the EfficientNet models. The *depth* refers to a coefficient of the number of layers at each stage of the network, whereas the *width* denotes a coefficient of the number of channels produced by each convolution. By carefully balancing these hyperparameters using a *compound scaling coefficient*, better performance is achieved. EfficientNets showed improved accuracy and efficiency compared to state-of-the-art CNNs. The EfficientNet architecture makes use of the inverted residuals introduced in MobileNetV2 – denoted as the mobile inverted bottleneck *MBConv* block – with the addition of an SE block as an optimization step performed after the depth-wise convolution and prior to the point-wise convolution.

In [70], *EfficientNetV2* is introduced as a new family of models that uses neural architecture search (NAS) to optimize training and parameter efficiency through non-uniform scaling. Furthermore, progressively resizing images and adaptively adjusting regularization during training resulted in improved training speeds and accuracy. Namely, an 11-fold increase in the training speed and up to 6.8x better parameter efficiency was reported on various datasets, including ImageNet. EfficientNetV2 also uses MBConv blocks. However, in earlier stages of the network, depth-wise convolutions are found to be slower and less effective than in later stages. Although depth-wise convolutions have fewer parameters and require less floating-point operations (FLOPs) than standard convolutions, the authors found they cannot fully use modern accelerators. Thus, the *Fused-MBConv* block is introduced as a replacement for MBConv blocks in earlier stages of the network, in which the initial $1 \times 1$ and depth-wise convolutions are replaced with a standard $3 \times 3$ convolution, as shown in Figure 32 below.

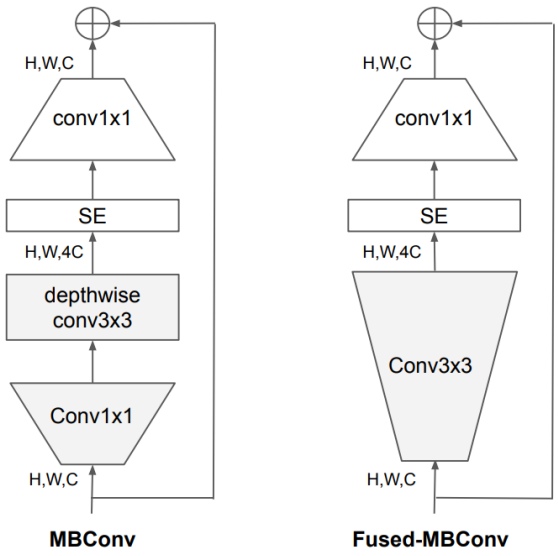


**Figure 32: The differences between an MBConv and Fused-MBConv block [70]**

## 6.2 Proposed Efficient U-Net Architecture

Using several of the network architectural designs described in section 6.1, together with U-Net and modifications to U-Net proposed in [29], a customized network architecture called *Efficient U-Net* is proposed. A description of the network architecture is provided, followed by the reasoning behind several key design choices. Then, modifications to the proposed architecture are discussed.

### 6.2.1 Network Architecture

The architecture follows the encoder-decoder scheme employed in U-Net but has some key differences.

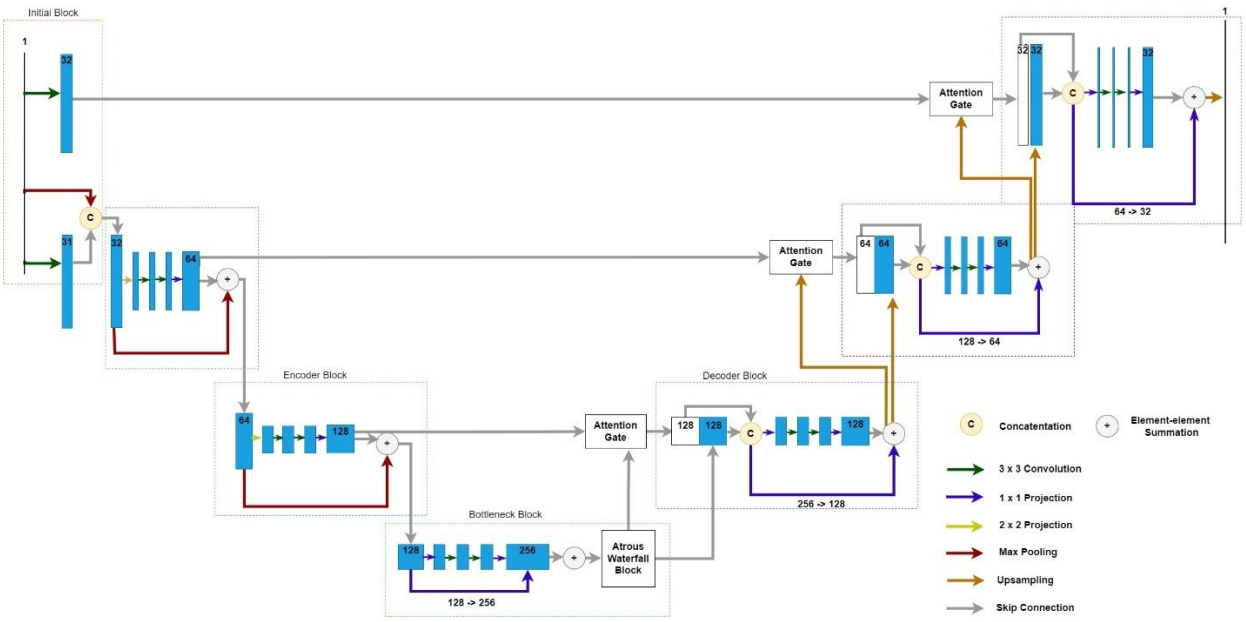


**Figure 33: Proposed Efficient U-Net Architecture. Blue blocks represent feature maps, with the width denoting feature space and height denoting resolution**

*Initial Block*: The initial encoder block follows closely to that of the initial block proposed in ENet (as shown in Figure 25a.). One addition is another extension branch that performs a convolutional operation to output a feature map of channel size matching that of the up-sampled feature map in the corresponding decoder of U-Net, which is necessary for concatenation.

*Encoder Block*: Following the initial encoder block, each subsequent down-sampling encoder block employs the ResNet-inspired parallel-branch scheme proposed in the down-sampling bottleneck blocks of ENet: a main branch performs the max pooling operation and an extension branch performs a $2 \times 2$ convolution with stride 2, as suggested in ENet, to project the input feature map into a dimensionality reduced feature space ¼th of the feature space size passed into the encoder block. Then, two $3 \times 3$ convolutions, as proposed in U-Net, are applied to the dimensionality-reduced feature map, before a $1 \times 1$ expansion is applied to the resulting feature map to increase the channel size to that of the desired output size. A batch normalization and activation function are applied between all convolutions. A regularizer is applied after the final expansion, with a dropout of probability set to 0.1. The resulting feature map is copied over a skip connection to be

concatenated with the corresponding decoder block (as in U-Net) before it is summed with the result of max pooling from the main branch. After the final summation, another activation function is applied.

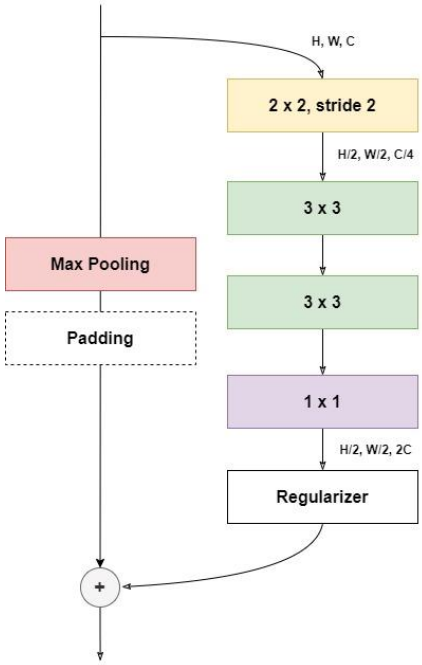


**Figure 34: Encoder block in Efficient U-Net**

*Bottleneck Block*: A modified U-Net bottleneck block is proposed. This block has two phases. The first phase consists of a parallel-branch scheme with a main branch that performs a 1 × 1 convolution to adjust the number of features, and an extension branch that performs a 1 × 1 projection, followed by two 3 × 3 convolutions and a subsequent 1 × 1 expansion. Besides this, other operations are the same as those proposed in the encoder block. The second phase is inspired by the atrous (dilated) waterfall scheme described in [29]. The outputted feature map of phase one is passed as input to phase two consisting of three main blocks: each block consists of a 1 × 1 projection reducing the feature dimensionality by a factor of 4, followed by a 3 × 3 dilated convolution with a rate *r*, and a 1 × 1 feature expansion. The output of each block is copied to a concatenation operation and passed to the next block, except for the last block. Blocks one, two and three apply dilated convolutions with $r = 1$, $r = 2$ and $r = 4$, respectively. As with the encoder blocks, a batch normalization and activation function are applied between each of the convolutions.

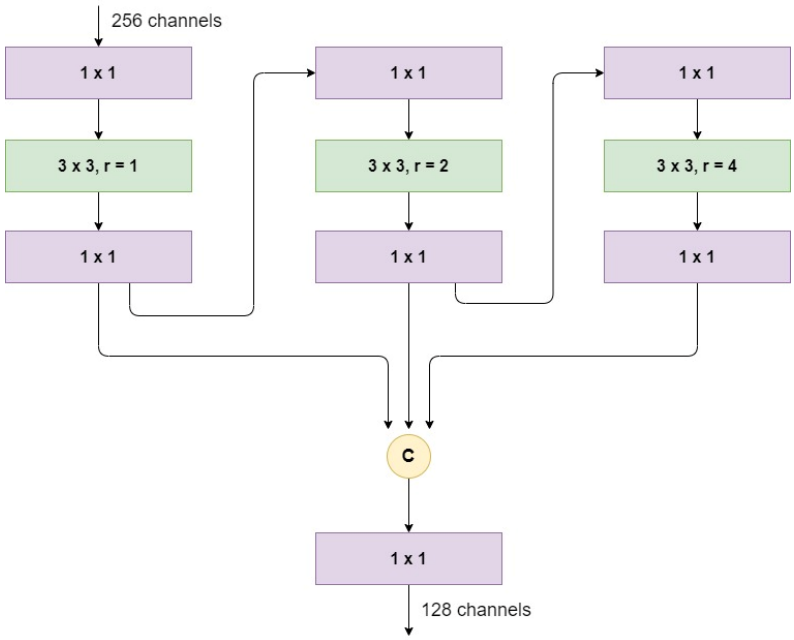


**Figure 35: Atrous Waterfall Block included in the bottleneck layer of Efficient U-Net. A feature map of 256 channels is taken in as input and the final 1 × 1 convolution reduces the output channel space to 128 channels**

*Decoder Block*: Consists of a bottleneck block symmetric to the encoder block, except that the main branch consists of a 1 × 1 convolution to adjust the number of features, and the 2 × 2 projection with stride is replaced with a 1 × 1 projection. Moreover, the feature map from the previous decoder block is up-sampled via bilinear interpolation and applied a 2 × 2 convolutional filter reducing the feature space. Based on [29], an attention gate is applied to the up-sampled feature map and the skip connection before they are concatenated together prior to being passed to the decoder. The cSE variant of the SE block is implemented in the attention gate, using an adaptive max pooling operation to squeeze the spatial dimensions.

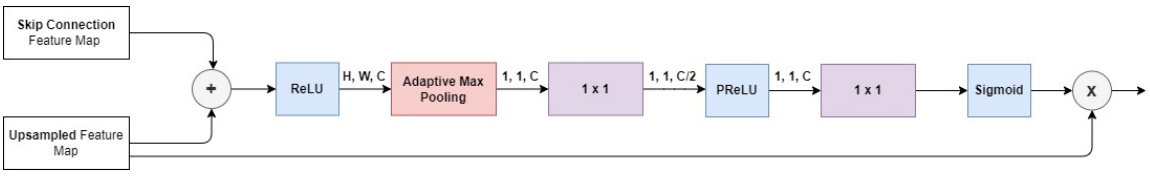


**Figure 36: Attention Gate in Efficient U-Net**

### 6.2.2 Design Choices

Here, design choices for the proposed Efficient U-Net architecture are explained, based on heuristics and results achieved in related works, with the goal of reducing the model size and number of computations while maintaining a high performance for fault segmentation.

*Network Width*: In the original U-Net, the initial encoder block outputs a feature map of channel size = 64, with subsequent encoder blocks increasing the channel size by a factor of 2. To reduce model size and parameter complexity, the initial block increases the channel size to only 32 and is increased by the same factor of 2 in subsequent encoder blocks.

*Early and Limited Down-sampling*: Following the intuition of early down-sampling, the proposed architecture uses an initial block performing only a single convolution. Furthermore, down-sampling is limited in the proposed model: down-sampling only occurs three times – once in the initial block and twice more afterwards – whereas the original U-Net performs down-sampling four times.

*Projection*: As shown in ENet, the number of parameters and convolutional operations can be decreased significantly through projection of the feature space to a lower dimension. Also noted in [63] is that in the down-sampling bottleneck block, simply doing a $1 \times 1$ projection with a stride of 2 discards 75% of the input feature map, which is not ideal, particularly when extracting faults that take up a small proportion of the input. Hence, the proposed encoder block increases the filter size to $2 \times 2$, to take the full input feature map into account.

*Smaller Feature Maps*: Unlike U-Net, which produces large feature maps, with the bottleneck block of U-Net outputting the largest feature map size of channel size = 1024, the bottleneck block in the proposed architecture outputs a feature of channel size = 256. As a result, the number of parameters and operations is greatly reduced.

*Atrous Waterfall Block*: As proposed for U-Net by the authors in [29], a waterfall scheme for atrous convolutions is employed in the bottleneck block of the proposed architecture. Additional to [29], each block of the proposed Efficient U-Net waterfall scheme performs

a projection before an atrous convolution, reducing the feature size by a factor of 4. Not only does this reduce the number of parameters and computations, but also increases the receptive field of the convolutional filter. Intuitively, the increase of the receptive field, coupled with the atrous convolutions at different rates, would capture a greater range of contextual information, which is important to distinguish faults from non-fault objects in the global scene. As suggested in [29], dilation rates of 1, 2 and 4 are used, given that faults are relatively small in scale compared to the rest of the scene.

*Feature Map Saving*: Following the U-Net architecture, the feature maps produced by each encoder block are saved and copied over skip connections instead of the max pooling indices. Particularly when segmenting small and narrow faults, saving just the indices from the dimensionality-reduced feature map produced from max pooling can result in a loss of lower-level fault information. Although saving the entire feature map requires more memory, it is a reasonable trade-off given that the initial block already considerably compresses the input image, amongst other memory-reducing design choices proposed – any further reduction in the features extracted at each level would effectively result in sparse up-sampled feature maps, insufficient for segmenting granular faults.

*Attention Gate*: When identifying the presence of structural faults in images, it is important to consider the fault from various layers of abstraction; it is not only important to extract the edges that define the lower-level features of a fault, but to also extract higher level features unique to faults within a global context that may be littered with noise and non-fault objects part of a greater scene. Thus, the importance of using attention gates to highlight faults from the background may be key to helping the model generalize to noisy real-world imagery. Moreover, attention gates applied on skip connections may help to suppress poor feature representation passed from earlier layers. The attention gate takes in the feature map passed through the skip connection and the up-sampled feature map and performs element-wise summation between both feature map tensors – if both tensor sizes do not match along the spatial dimensions, the up-sampled feature map tensor is resized accordingly. The element-wise summation will cause aligned weights to become larger, acting as an additional attention step before the summed tensor is passed through a ReLU activation and then through to an cSE block to highlight features of interest through

reweighting of the channel space. The resulting feature map is then element-wise multiplied with the original up-sampled feature map.

*Activation Function*: In [63], the authors replace the ReLU activation function in the initial layers of ENet with Parametric ReLUs (PReLU) [64], which uses an additional parameter per feature map to learn the negative slope of non-linearities; it was found that replacement earlier in the network improved the results. The authors in [63] hypothesize the poor performance of ReLUs in the initial layers to be attributed to the limited depth of the ENet architecture, compared to deep networks such as ResNet. As a result, since the proposed Efficient U-Net architecture is also relatively shallow, the PReLU activation function is used in the initial, encoder, and bottleneck blocks. PReLU is also used in the cSE block to prevent further information loss from the squeeze operation followed by a $1 \times 1$ projection in lower dimensional feature space.

### 6.2.3 Proposed Modifications to Efficient U-Net

Modifications to the Efficient U-Net architecture are proposed and implemented to further analyze the impact of certain design choices on performance and model efficiency. Namely, besides benchmarking to the state-of-the-art U-Net, there are two key areas of interest that are investigated. The first area of interest is *how important attention gating is*, and whether a) reweighting the feature map responses across the spatial, channel, or combination of both dimensions results in better fault segmentation, and b) adding attention gating earlier in the network influences the fault segmentation performance. The second area of interest is investigating how the *parameter space*, *number of computations*, and *inference time* is impacted in relation to the fault segmentation performance by a) introducing mobile inverted residual blocks into the network, and b) removing the atrous waterfall block. The key areas of investigation are formalized below.

*Spatial Versus Channel Attention*: Focusing attention on spatial pixel regions where faults are more likely to occur intuitively makes sense, given the relatively small percentage of pixels that correspond to faults. Furthermore, highlighting spatial information from an up-sampled feature map concatenated with the corresponding down-sampled feature map helps retain spatial information from earlier in the network that may have been lost during

down-sampling. However, given the down-sampled feature map provides a more limited set of lower-level features whereas the up-sampled feature map has a richer set of features that better encapsulates properties pertaining to faults, concatenation may result in poorer feature representation. Hence, this motivates the importance of channel attention as well as spatial attention for fault segmentation. To assess the effectiveness of different attention mechanisms, several versions of Efficient U-Net are implemented, in which each version implements an attention gate with one of the following SE blocks: an cSE block, an sSE block, and an scSE block. Additionally, a version of the model is implemented without an attention gate to further assess the importance of attention gating.

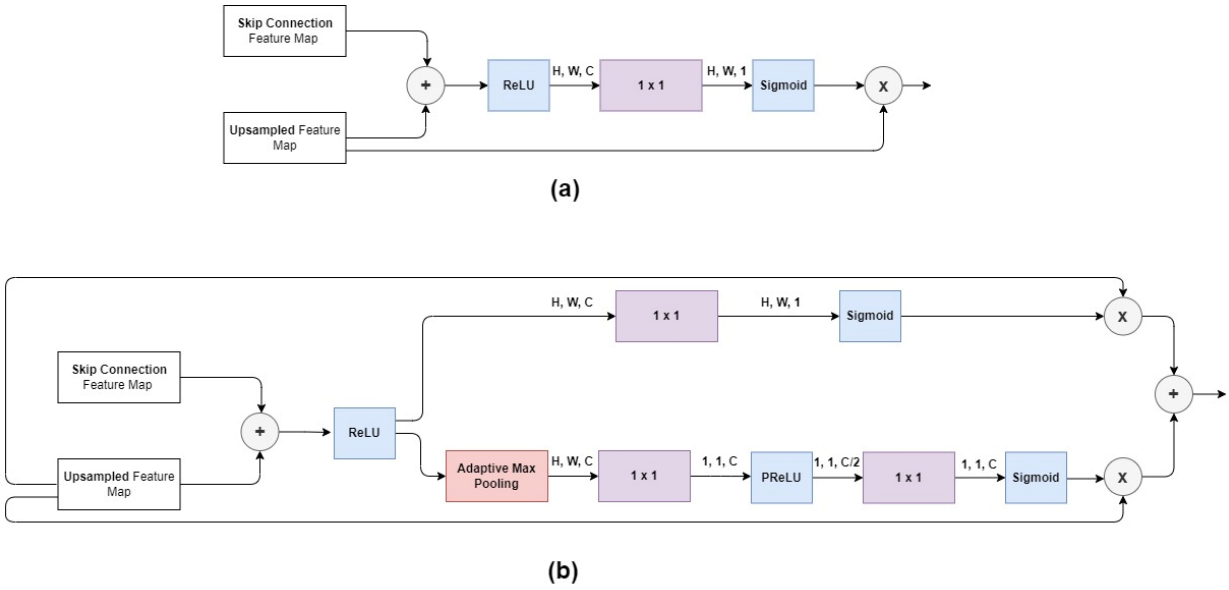


**Figure 37: Implemented SE block variants in the decoder block of the network: (a) sSE block; (b) scSE block**

*Attention Early Versus Later in the Network*: The role an SE block performs depends on where in the network they are used; earlier in the network, they strengthen shared low-level feature representations by equally exciting informative features, whereas later in the network, they become more specialized in more heavily reweighting relevant features of interest [66]. To analyze the impact of implementing attention earlier in the network on model performance, a modified version of Efficient U-Net is implemented, in which the convolutional branch of each of the encoders, decoders and bottleneck are replaced with

customized MBConv and Fused-MBConv blocks consisting of an SE block, as described in [69] and [70].

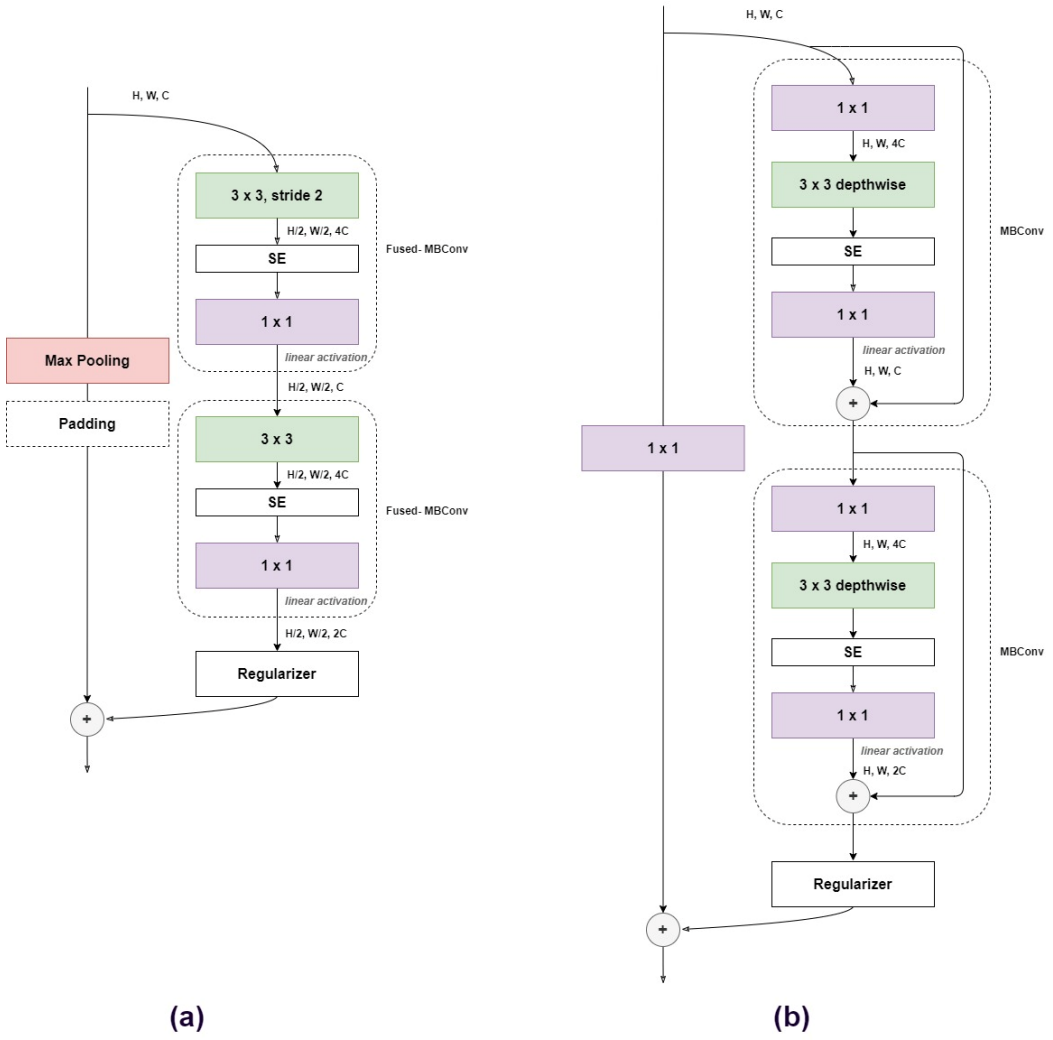


**Figure 38: Modifying the proposed Efficient U-Net with (a) two Fused-MBConv blocks in the encoder and (b) two MBConv blocks in the bottleneck**

As shown in Figure 38 above, based on the inverted residual block, two consecutive MBConv blocks are added at each of the encoders, decoders, and at the bottleneck to replace the double convolution in the projected lower dimensional feature space. Each MBConv block increases the feature space by a factor of 4, inverse to the projection factor performed by the base Efficient U-Net architecture. In the encoder, the first of the two MBConv blocks reduces the feature map resolution with a $2 \times 2$ convolution of stride 2.

The second MBConv block doubles the feature dimensionality by a factor of 2. In the bottleneck, since the input and output feature map resolution and the number of features remains constant, a residual connection is included in each of the MBConv blocks.

*MBConv on Model Performance and Complexity*: Following closely to [70], the MBConv blocks in the encoder are replaced with Fused-MBConv to optimize the use of accelerators. Also, depth-wise separable convolutions performed in the expansive part of the MBConv block may be more suitable than standard convolution performed in projected lower-dimensional feature space. Projection may compress information to the extent that finer-grain details of faults may get lost in the features extracted by convolutions on the dimensionality-reduced feature maps. On the other hand, depth-wise separable convolutions can reduce the number of computations without feature dimensionality reduction, which is important to preserving fault information. Thus, the projections and subsequent standard $3 \times 3$ convolutions in the extension branches and waterfall-based dilation phase are replaced with depth-wise separable convolutions, following the original MBConv block.

*Atrous Waterfall Block on Model Performance and Complexity*: Although the atrous waterfall block can capture a greater range of contextual information in the bottleneck stage of the network, this may come at the cost of a notable increase in the number of computations and parameters. Thus, the atrous waterfall block is omitted in some of the modified versions of the model and replaced with a $1 \times 1$ convolution.

## 6.3 Evaluation Criteria

To evaluate the performance and complexity of the proposed Efficient U-Net model in relation to the state-of-the-art U-Net model and the modified networks, several key metrics are considered for model training, validation, and complexity.

### 6.3.1 Training Metrics

To evaluate the loss during training, the *cross-entropy* loss function is commonly used to measure the likelihood of the output with respect to the true labels. Cross-entropy is calculated by taking the sum of the products of each true label and corresponding

probability of the prediction for that label in the output. The logarithmic function is applied to each probability to avoid the likelihood from going to zero due to multiplication with small probability values:

$$L_{CE} = -\frac{\sum_{i=1}^{N} x_i \log(\dot{x}_i)}{N}$$

Where $L_{CE}$ is the cross-entropy loss, $x_i$ is *i*th pixel value in the ground truth label matrix, $\dot{x}_i$ is the *i*th pixel value probability in the model prediction matrix, and *N* is the total number of pixels.

For the proposed binary pixel-wise classification, a variant of the cross-entropy loss function called *binary cross-entropy* loss is used. Binary cross-entropy considers the likelihood of an observation belonging to each of the classes. That is, the probability of the predicted class multiplied by the corresponding true class label, added to the probability of predicting the opposite class multiplied by the opposite class label. The binary cross-entropy loss $L_{BCE}$ is expressed as follows:

$$L_{BCE} = -\frac{\sum_{i=1}^{N} x_i \log(\dot{x}_i) + (1 - x_i) \log(1 - \dot{x}_i)}{N}$$

### 6.3.2 Validation Metrics

Although cross-entropy loss is a useful tool for evaluating the loss during the training process, it can be difficult to interpret. When validating the performance of a classifier, other metrics are typically used. *Accuracy*, also known as the error rate, is a standard evaluation metric, as it is intuitive and measures how often a classifier makes correct predictions, on average. Accuracy in classification problems is computed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where *TP* is the number of true positives in which the classifier correctly predicts the positive class, *TN* is the number of true negatives wherein the classifier correctly predicts the negative class, *FP* is the number of false positives in which the classifier incorrectly predicts a label to be part of the positive class, and *FN* is the number of false negatives.

However, due to the inherent nature of unbalanced classes in anomaly detection, particularly in pixel-wise image fault segmentation, accuracy can be rather misleading. Take, for example, a dataset in which 95% of the true labels are negative. In this case, a classifier that simply predicts every observation to be negative will still achieve 95% accuracy as the baseline accuracy. Thus, without applying any of the previously discussed preprocessing techniques on unbalanced classes, accuracy is not an ideal metric for evaluating such classes. Instead, it would be advantageous to consider other several key indicators that are more robust to class imbalances. Two important indicators include the number of true positives identified out of all positive predictions – referred to as the *precision* – and the proportion of all positives from the dataset correctly identified – referred to as the *recall*.

A more reliable validation metric than accuracy that is commonly used in pixel-wise segmentation for measuring the degree of overlap between the predicted label map and ground truth label map is called the *intersection over union* (IoU). This metric takes the intersection between the predicted label and ground truth, divided by the union of the predicted label and ground truth. The output is a value between 0 and 1, with an IoU closer to 1 corresponding to a greater overlap of the predicted label and ground truth. This can be expressed as follows:

$$IoU = \frac{TP}{FP + TP + FN}$$

The *Dice score* is another important validation metric that measures the degree of overlap between prediction and ground truth while taking recall and precision into account. Namely, the Dice score, also known as the *F1-score*, is equivalent to the harmonic mean of the precision and recall [29], and is formulated as follows:

$$D_{Score} = \frac{2TP}{2TP + FN + FP}$$

In other words, the Dice score takes the intersection multiplied by 2, divided by the intersection plus the union. This expression can also be rearranged with respect to precision and recall:

$$D_{Score} = \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

That is, the greater the precision and recall, the higher and better the dice score (between 0 and 1). Due to the nature of the harmonic mean, the Dice score will penalize very low precision or recall values, such that the more balanced the precision and recall, the higher the Dice score. Hence, the Dice score is a good metric for optimizing the precision and recall tradeoff. Moreover, the Dice score tends to give a measure of the average performance, whereas the IoU approximates the worst-case performance, which makes it advantageous to consider both metrics during evaluation.

### 6.3.3 Model Complexity Metrics

*Network latency*: To assess inference time or network latency when predicting upon a single image, t*ime* (measured in milliseconds) is used. Specifically, only the feed-forward of the neural network is measured, omitting the time required for GPU initialization and the transfer of data from the CPU to GPU. Network latency is also expressed in terms of FPS (frames per second), for benchmarking in relation to real-time inference.

*Computations*: The number of computations required in a single pass through a network is measured using multiply-accumulate operations (MAC). A MAC is an operation that includes one multiplication and one addition, each of which can be floating point operations. Roughly speaking, one MAC is equal to two floating point operations (FLOP). One possible advantage to using MACs over FLOPs is that neural networks compute mainly on multiply-accumulate operations, and thus improvements in the number of MACs would generally be more emphasized than those in the number of FLOPs.

*Parameters*: The number of parameters is also measured to further assess network complexity in terms of the number of weights and biases in each convolutional layer. The number of parameters for each convolution is $C \times w \times h \times C' + 1$, where $C$ is the input channel space size, $w$ is the width of the kernel, $h$ is the height of the kernel, and $C'$ is the output channel space size, with an addition of 1 to account for the bias term of each kernel.

## 6.4 Experimental Results

### 6.4.1 Data

Publicly available datasets of structural faults captured in real-world environments are used for training and validation of the proposed Efficient U-Net models as well as the baseline U-Net.

*Crack500*: One of the datasets training and validation is performed upon is a crack-based dataset called Crack500 [31][71]. The original dataset contains 500 road pavement color images of resolution 2,000 × 1,500, taken with cellular phones. Each image is accompanied with a pixel-wise annotated segmentation map.

*GAPs384*: Based on the German Asphalt Pavement Distress (GAPs) dataset presented in [72], the original GAPs384, a subset of GAP, consists of 353 training and 27 validation grayscale images of resolution 1920 × 1080, captured with a specialized imaging system with photogrammetrically calibrated monochrome cameras. Captured images include cracks, potholes, and inlaid patches. Moreover, each image is accompanied with a pixel-wise annotated segmentation map.

*CrackForest*: The CrackForest dataset [73][74] consists of 118 RGB color images of 480 × 320 resolution captured with an iPhone 5 camera. Captured images include cracks amongst noise such as oil stains, road markings, shoes, and shadows. As with the Crack500 and GAPs384 datasets, the images are accompanied with pixel-wise annotated segmentation maps. 89 samples are used for training, with 29 samples set aside for validation.

*Merged Dataset*: Samples from the Crack500, GAPs384, and CrackForest dataset, along with samples from the *Cracktree200* [75], *Aigle-RN & ESAR & LCMS* [76], *DeepCrack* [87] and masonry crack [77] datasets are combined into a merged dataset consisting of 9,793 training samples and 1,745 validation samples of 448 × 448 resolution. In particular, the Cracktree200 dataset introduces further noise, occlusion, shadows, and low contrast. Furthermore, non-crack samples are included in the merged dataset, capturing corners,

edges, and brick mortar that resemble cracks to allow the model training upon such samples to better distinguish between cracks and non-cracks. By combining diverse samples of various structures – pavement, concrete, and masonry – captured using different equipment within noisy and occluded environments, a model trained upon the merged dataset may better generalize to real-world settings.

*Data Preparation*: The image resolutions in the Crack500 and GAPs384 datasets are quite large, posing a constraint on the network scalability due to the limited amount of GPU memory (6 GB graphics-card memory). Thus, cropped images and accompanying annotated pixel-wise segmentation maps of resolution 640 × 360 from the original Crack500 dataset are used, resulting in 1,896 training images and 348 validation images. Similarly, a modified version of the GAPs384 dataset is used in this experiment, consisting of a cropped subset of 465 training and 44 validation images of 540 × 440 resolution. Images in the CrackForest dataset are scaled to 320 × 320 resolution before training. Each dataset is also augmented through random image rotation between -90° and 90°, and horizontal and vertical flipping with a probability $p = 0.5$. Due to some instances of low lighting, particularly in the GAPs384 dataset, a random brightness factor in the range of (-0.2, 0.2) is applied. It is also observed that there is a small fraction of labelled fault pixels in comparison to non-fault pixels. Hence, an oversampling approach is used to intentionally sample more of the pixels associated to the fault class during the data loading phase prior to training; class weights are used to determine the ratio of fault pixels to non-fault pixels, to more evenly sample pixels corresponding to observations from each class. Data is also shuffled prior to training on every dataset to ensure randomness in the sampling.

### 6.4.2 Evaluation

The models, training, and evaluation scripts are written in Python, using the *PyTorch* framework. Model training and validation, as well as inference speed testing, is conducted on an Nvidia GeForce GTX 1060 6 GB GPU. Inference speed testing is also performed on an Nvidia Jetson Nano Developer Kit, which provides a good indicator of performance on an edge device with constrained GPU processing and memory.

**Table 2: Comparison of processing and memory specifications**

| Machine | Memory (GB) | CUDA Cores | Floating-Point Performance (GFLOPs) |
|---|---|---|---|
| Jetson Nano | 4 | 128 | 472 |
| Nvidia GeForce GTX | 6 | 1280 | 4357 |

*Training Settings and Hyperparameters*: Each model is trained and validated on the Crack500, GAPs384, and CrackForest datasets. As illustrated in [29], a model trained solely on the CrackForest dataset is very sensitive to noise and lighting. Furthermore, the CrackForest and GAPs384 datasets are relatively small and limited in terms of the amount of encoded information; images in the CrackForest dataset are relatively small in resolution, whereas the grayscale images in the GAPs384 dataset only encode one channel. As a result, models trained on these datasets fail to generalize and struggle to distinguish faults in unseen images, as found in [29]. Thus, in this experiment, each of the models are first pretrained on the merged dataset before being trained on the individual datasets, wherein the weights learned during pretraining are saved and reused. Reusing the weights may result in the model reaching faster convergence during training and generalizing better on unseen data, since pretraining is performed on a wide variety of data in the merged dataset. Based on hyperparameters chosen in the literature, pretraining on the merged dataset is conducted for 15 epochs, with a learning rate of 0.001 at the start. A scheduled reduction in the learning rate by half every 5 epochs is performed, to prevent the model from overshooting the local minima of the loss. After pretraining, the models are trained on the individual datasets (Crack500, GAPs384 and CrackForest) for an additional 15 epochs, with a learning rate of 0.0005 at the start and a scheduled reduction by half every 5 epochs. The Adam optimizer is used to update the network weights during backpropagation. Moreover, a batch size of 4 is used. Only pixels with sigmoid values outputted from the model of 0.5 or greater – in the range of [0,1] – are considered as part of the fault class. On every epoch, the performance on the validation data is evaluated based on the Dice score and IoU, with the best values reported.

*Study One*: To compare the proposed Efficient U-Net variants based on performance, several sets of tests are conducted in this study. Two groups of models are constructed, with one group of models including the atrous waterfall block, and the other group omitting

the atrous waterfall block. Each group consists of a version with a separate implementation of the modified attention gate described in section 6.2.3. Within each group, the *impact of attention gating* is investigated on fault segmentation performance. Across both groups of models, the *effect of omitting an atrous waterfall block* on fault segmentation performance and efficiency is analyzed on each of the datasets. All other variables are controlled. The tested architectures are as follows:

- Baseline Efficient U-Net without attention gates nor the atrous waterfall block (EU-Net);
- Efficient U-Net with attention gates using an cSE block (EU-Net + cSE);
- Efficient U-Net with attention gates using an sSE block (EU-Net + cSE);
- Efficient U-Net with attention gates using an scSE block (EU-Net + scSE);
- Efficient U-Net with an atrous waterfall block (EU-Net + AWF);
- Efficient U-Net with an atrous waterfall block and attention gates implemented using an cSE block (EU-Net + AWF + cSE);
- Efficient U-Net with an atrous waterfall block and attention gates implemented using an sSE block (EU-Net + AWF + sSE);
- Efficient U-Net with an atrous waterfall block and attention gates implemented using an scSE block (EU-Net + AWF + scSE);

The performance of the Efficient U-Net model and its variants are compared according to the best Dice score and IoU achieved on the merged and individual validation datasets during training. Also reported is the Area Under the Precision-Recall Curve (AUPRC), which represents the recall and precision tradeoff at varying thresholds; the AUPRC is particularly useful in the case of imbalanced classes. The AUPRC, IoU, and Dice scores for each model are given in Tables 3-6, with the superior results bolded. Each table reports the results of each model on a specific dataset. The Dice Score and IoU are shown to correspond directly to each other, as models with the highest Dice Scores in each dataset,

except for the merged dataset, have the highest IoU. Similarly, the AUPRC tends in line with the Dice score and IoU across all the datasets – the higher the Dice score and IoU, the higher the AUPRC.

On 3 out of the 4 datasets, the addition of an attention gate (AG) results in improved Dice scores. The atrous waterfall (AWF) and non-AWF models that implement scSE attention gating have superior Dice scores compared to their respective non-AG and AG counterparts on the merged and Crack500 datasets (Tables 3 and 4). Similarly, attention gating results in improved performance on the GAPs384 dataset (Table 5), with sSE attention gating resulting in the highest Dice score amongst the AG and non-AG counterparts. Only on the CrackForest dataset (Table 6) does attention gating not result in the highest Dice scores. However, it was found that cSE attention gating results in notably lower Dice scores in the GAPs384 and CrackForest datasets, for both the AWF and non-AWF model versions.

Comparing each non-AWF model with its AWF counterpart, each of the AWF models in the Crack500 dataset, whereas all AWF models, except the scSE attention gating model on the merged dataset and the non-AG model on the CrackForest dataset, result in improved Dice scores on their non-AWF counterparts. However, in the GAPs384 dataset, only the non-AG AWF model results in a higher Dice score compared to its non-AWF counterpart – the AG AWF models only have slightly lower Dice scores compared to their non-AG AWF counterparts. The relatively poorer results on the CrackForest dataset may be due in part to the scaling down of the images prior to training, as described in section 6.4.1.

Overall, the best resulting models in each dataset included some combination of AWF and/or AG blocks, with scSE attention gating performing particularly well on larger merged and Crack500 datasets. On the merged dataset, the best Dice score achieved is 0.6287 (EU-Net + scSE), a notable increase from the baseline (no AWF and no AG) of 0.5635. On the Crack500 dataset, the best Dice score achieved is 0.7789 (EU-Net + AWF + scSE), an increase from 0.7215 achieved by the baseline. On the GAPs384 dataset, the best Dice score achieved is 0.4875 (EU-Net + sSE), an increase from 0.4281 achieved by

the baseline. On the CrackForest dataset, the best Dice score achieved is 0.6818 (EU-Net + AWF), an increase from 0.6191 achieved by the baseline.

**Table 3: Comparison of the best validation results of the proposed Efficient U-Net model and its variants on the merged dataset**

| Merged (Pretraining) | AUPRC | IoU | Dice |
|---|---|---|---|
| EU-Net (Baseline) | 0.3432 | 0.4094 | 0.5635 |
| EU-Net + cSE | 0.6031 | 0.4311 | 0.5896 |
| EU-Net + sSE | 0.6298 | 0.4235 | 0.5806 |
| EU-Net + scSE | **0.7077** | 0.4717 | **0.6287** |
| EU-Net + AWF | 0.6624 | 0.4515 | 0.6111 |
| EU-Net + AWF + cSE | 0.6235 | 0.4446 | 0.6022 |
| EU-Net + AWF + sSE | 0.6535 | 0.4400 | 0.5968 |
| EU-Net + AWF + scSE | 0.5945 | **0.4718** | 0.6286 |

**Table 4: Comparison of the best validation results of the proposed Efficient U-Net model and its variants on the Crack500 dataset**

| Crack500 | AUPRC | IoU | Dice |
|---|---|---|---|
| EU-Net (Baseline) | 0.6520 | 0.5847 | 0.7215 |
| EU-Net + cSE | 0.7531 | 0.6025 | 0.7419 |
| EU-Net + sSE | 0.6853 | 0.5946 | 0.7283 |
| EU-Net + scSE | 0.8076 | 0.6126 | 0.7490 |
| EU-Net + AWF | 0.7853 | 0.6206 | 0.7562 |
| EU-Net + AWF + cSE | 0.7660 | 0.6153 | 0.7527 |
| EU-Net + AWF + sSE | 0.6238 | 0.5950 | 0.7304 |
| EU-Net + AWF + scSE | **0.8545** | **0.6463** | **0.7789** |

**Table 5: Comparison of the best validation results of the proposed Efficient U-Net model and its variants on the GAPs384 dataset**

| GAPs384 | AUPRC | IoU | Dice |
|---|---|---|---|
| EU-Net (Baseline) | 0.5462 | 0.2772 | 0.4281 |
| EUNet + cSE | 0.4663 | 0.1804 | 0.3010 |
| EU-Net + sSE | 0.5600 | **0.3276** | **0.4875** |
| EU-Net + scSE | 0.5269 | 0.3027 | 0.4543 |
| EU-Net + AWF | 0.5531 | 0.3233 | 0.4844 |
| EU-Net + AWF + cSE | 0.4719 | 0.1799 | 0.2960 |
| EU-Net + AWF + sSE | 0.5127 | 0.3256 | 0.4868 |
| EU-Net + AWF + scSE | **0.5604** | 0.2998 | 0.4535 |

**Table 6: Comparison of the best validation results of the proposed Efficient U-Net model and its variants on the CrackForest dataset**

| CrackForest | AUPRC | IoU | Dice |
|---|---|---|---|
| EU-Net (Baseline) | 0.6367 | 0.4513 | 0.6191 |
| EU-Net + cSE | 0.3527 | 0.1518 | 0.2618 |
| EU-Net + sSE | 0.6392 | 0.4495 | 0.6167 |
| EU-Net + scSE | 0.6206 | 0.4095 | 0.5756 |
| EU-Net + AWF | **0.6437** | **0.5203** | **0.6818** |
| EU-Net + AWF + cSE | 0.3598 | 0.2017 | 0.3299 |
| EU-Net + AWF + sSE | 0.5973 | 0.4972 | 0.6609 |
| EU-Net + AWF + scSE | 0.5851 | 0.4009 | 0.5680 |

Also compared are the precision-recall curves of each set of models on each dataset, as shown in Figures 39 and 40. On the merged and Crack500 datasets, the AUPRCs of the scSE attention gated models are superior to its AG and non-AG counterparts. On the GAPs and CrackForest datasets, the AUPRCs of the non-AWF, sSE attention gated models are superior to its AG and non-AG counterparts, whereas the AUPRCs of the AWF, non-AG models are superior to its AG counterparts.

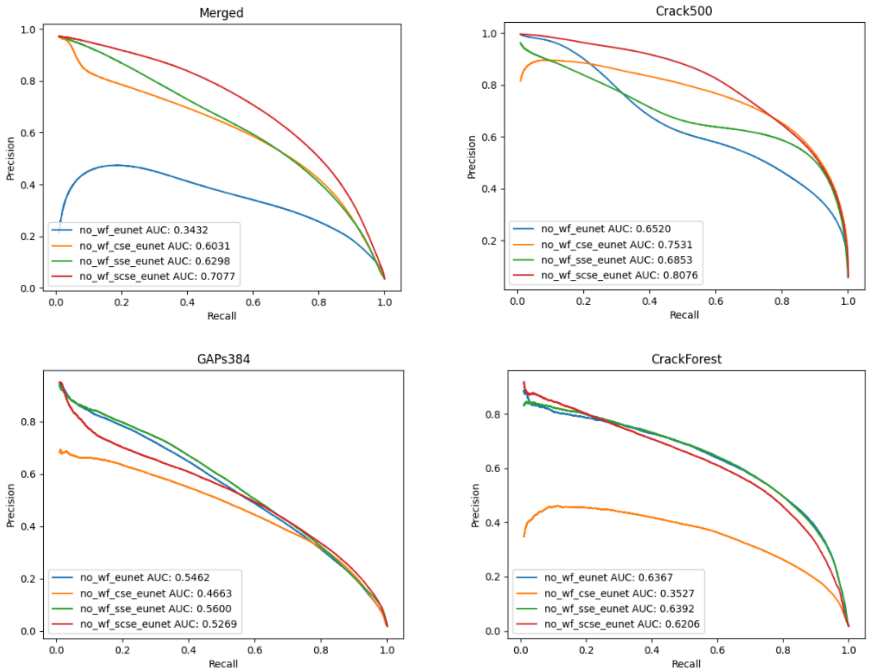


**Figure 39: The Precision-Recall Curves (PRC) and corresponding AUPRC scores for the Efficient U-Net models without the atrous waterfall block**

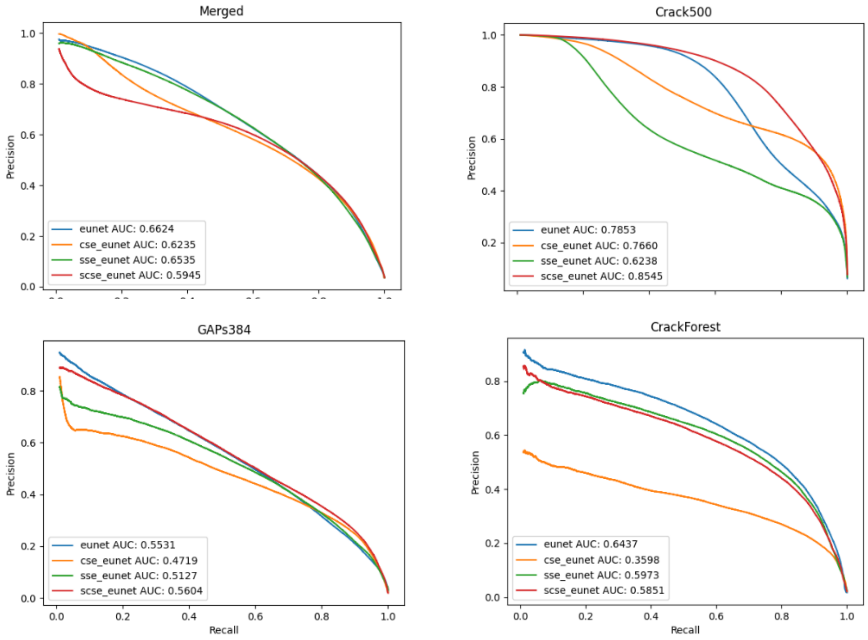


**Figure 40: The Precision-Recall Curves (PRC) and corresponding AUPRC scores for the Efficient U-Net models with the atrous waterfall block**

Although statistical performance can provide a good indication of performance, it is hard to interpret the segmentation quality from that alone. A visual investigation is also conducted to assess the properties of the segmented label maps produced by each model while drawing comparisons between the visual results and the statistical results obtained. As shown in Figures 41-44, there is a noticeable distinction between the baseline (No AG, No AWF) and the other models. For every dataset sample, the sSE attention gated models appear to outperform the baseline in segmenting narrow cracks (Figures 41-44), segmenting cracks captured in low brightness (Figure 43), and segmenting cracks impeded by shadows (Figure 41). Conversely, for every dataset, the cSE attention gates models appear to perform worse in extracting various cracks, resulting in thicker segmentations with some discontinuities (Figure 42) and completely missing very narrow cracks (Figures 41, 43 and 44) and cracks in low contrast (Figure 44). The cSE segmentations tend closer in line with statistical results, particularly for the GAPs384 (Figure 43) and CrackForest (Figure 44) dataset samples. However, the scSE model segmentation label map results are not much better than the cSE label maps, even in the merged (Figure 41) dataset in which

the scSE performed statistically well. However, similar could be said about the mismatch between the relatively high statistical cSE results compared to the baseline in the merged and Crack500 (Figure 42) datasets and the relatively poor segmentation results on their respective samples. This mismatch between statistical and visual results, particularly for the scSE models, may be due in part to a greater decrease in false positives than the baseline, than the decrease in true positives compared to the baseline, particularly for less frequent, fine cracks (Figures 41 and 43), which do not have many true positives to begin with. Thus, in these cases, the segmentation results may be visually poorer in comparison to the corresponding statistical results.

Comparing the segmentation label maps of the AWF and non-AWF models, it appears that the AWF + sSE models output finer extractions compared to the non-AWF + sSE models, which is particularly apparent in Figures 41 and 43. Across all the datasets, the AWF models tend to output finer segmentation label maps compared to their non-AWF counterparts, with little to no loss in detail, while also noticeably reducing the number of false positives on the merged dataset sample and better segmenting the extremely narrow crack in the top left of the CrackForest data sample and the bottom left of the Crack500 data sample. In general, the visual segmentation results tend in line with the statistical results for the AWF versus non-AWF models.

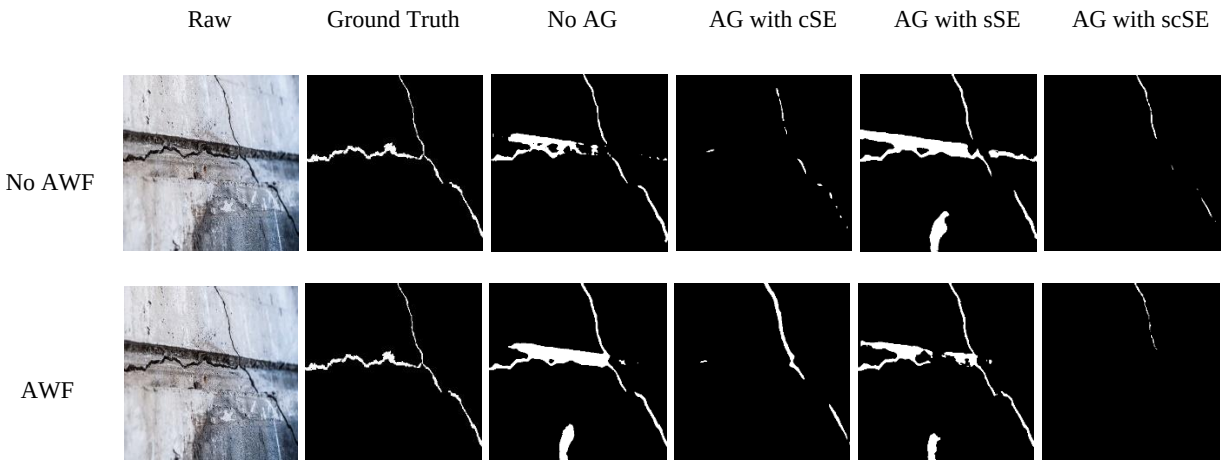


**Figure 41: The image, ground truth, and corresponding predicted label maps of each model version on the merged dataset**

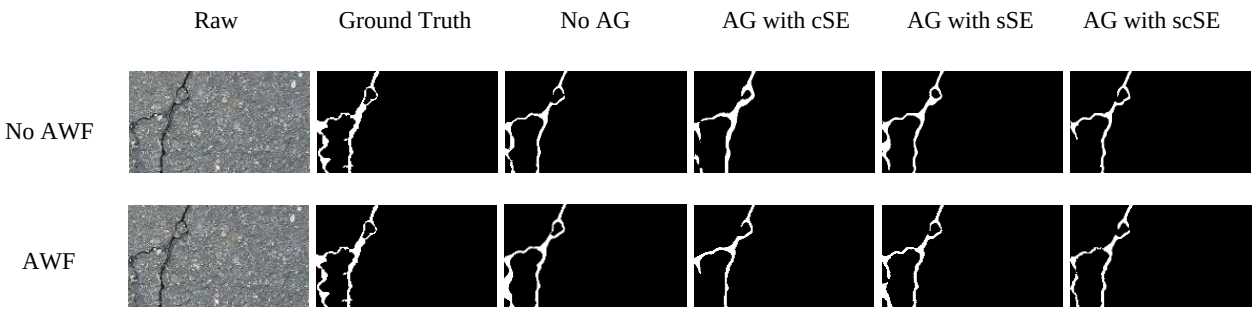


**Figure 42: The image, ground truth, and corresponding predicted label maps of each model version on the Crack500 dataset**

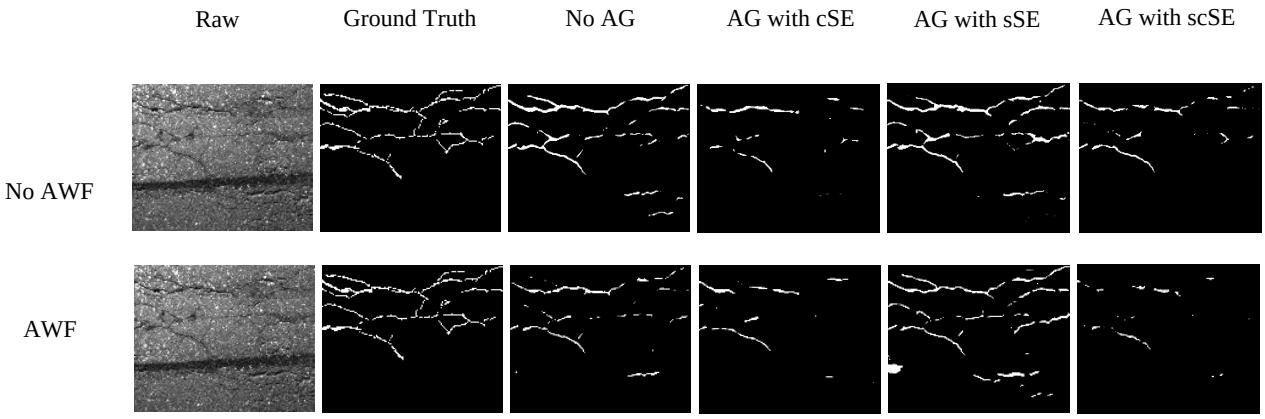


**Figure 43: The image, ground truth, and corresponding predicted label maps of each model version on the GAPs384 dataset**

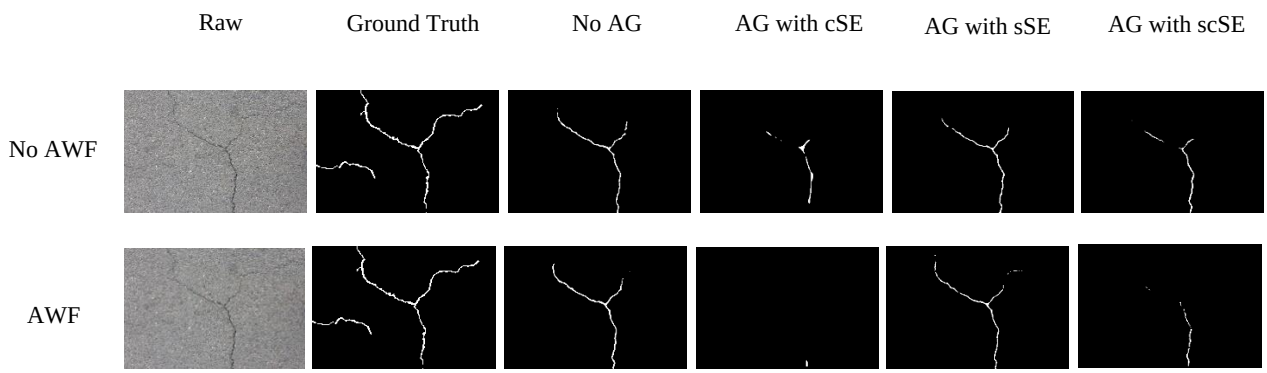


**Figure 44: The image, ground truth, and corresponding predicted label maps of each model version on the CrackForest dataset**

To investigate the parameter and computational efficiency, the number of parameters and GMACs required during inference are also analyzed and reported. Based on Table 7, as predicted, the models without the AWF block have a noticeable decrease in the number of parameters, by an average reduction of approximately 30%. Also, there is some reduction in the number of computations for non-AWF models, with an average reduction of approximately 17% in the GMACs reported compared to the AWF models. Furthermore, there is little additional computational overhead introduced by the AG models, compared to their non-AG counterparts: a 5% increase in the number of parameters in the scSE AG model compared to the non-AG model is the largest amongst the AG models, whereas less than a 1% increase in the GMACs is the largest amongst the AG models in relation to the non-AG models.

**Table 7: Comparison of the parameter and computational efficiency**

| **Model Version** | **#Params** | **GMACs** | | | |
|---|---|---|---|---|---|
| | | **320 × 320** | **448 × 448** | **540 × 440** | **640 × 360** |
| EU-Net (Baseline) | 425.32k | 1.46 | 2.88 | 3.40 | 3.31 |
| EU-Net + cSE | 446.83k | 1.47 | 2.89 | 3.41 | 3.31 |
| EU-Net + sSE | 425.54k | 1.47 | 2.89 | 3.41 | 3.31 |
| EU-Net + scSE | 447.05k | 1.47 | 2.89 | 3.41 | 3.32 |
| EU-Net + AWF | 603.88k | 1.77 | 3.47 | 4.09 | 3.98 |
| EU-Net + AWF + cSE | 625.39k | 1.77 | 3.47 | 4.09 | 3.99 |
| EU-Net + AWF + sSE | 604.11k | 1.77 | 3.47 | 4.09 | 3.99 |
| EU-Net + AWF + scSE | 625.61k | 1.77 | 3.48 | 4.10 | 3.99 |

The tradeoff between the model performance and computational and parameter efficiency is illustrated in Figures 45 and 46 below. The Dice score is evaluated in relation to the number of parameters (Figure 45) and the number of computations reported as GMACs on each resolution setting according to the image resolutions of each dataset (Figure 46). In Figure 45, apart from the scSE AG models, the addition of the AWF block results in notably higher Dice scores and number of parameters. In Figure 46, across the datasets, the addition of the AWF block results in increased Dice score and GMACs. However, there are some exceptions, including on the merged dataset where the non-AWF scSE AG model has nearly the same Dice score as the AWF + scSE model, but requires notably less computations. Similarly, the non-AWF cSE and sSE AG models produces higher Dice scores than their AWF counterparts on the GAPs384 dataset, while requiring less computations. These exceptions are illustrative of models that may provide good efficiency in terms of their fault segmentation capability per parameter and computation.

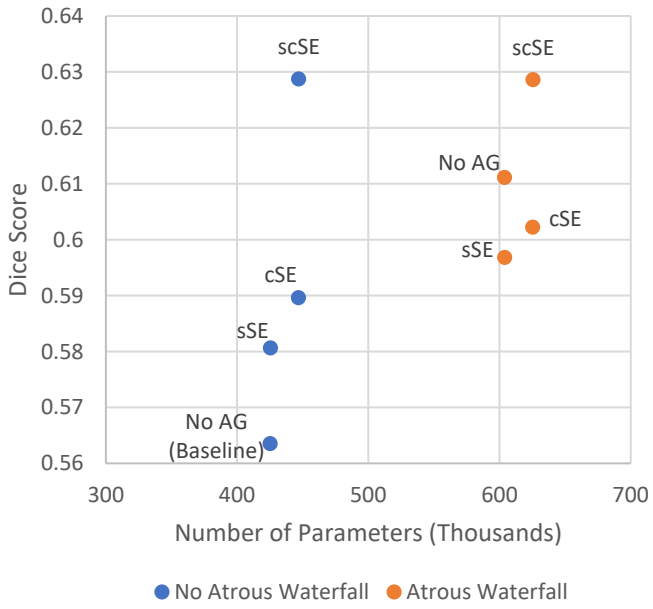


**Figure 45: The effect of adding the atrous waterfall block and using different attention gates on the Dice score (achieved on the merged dataset) and number of parameters**

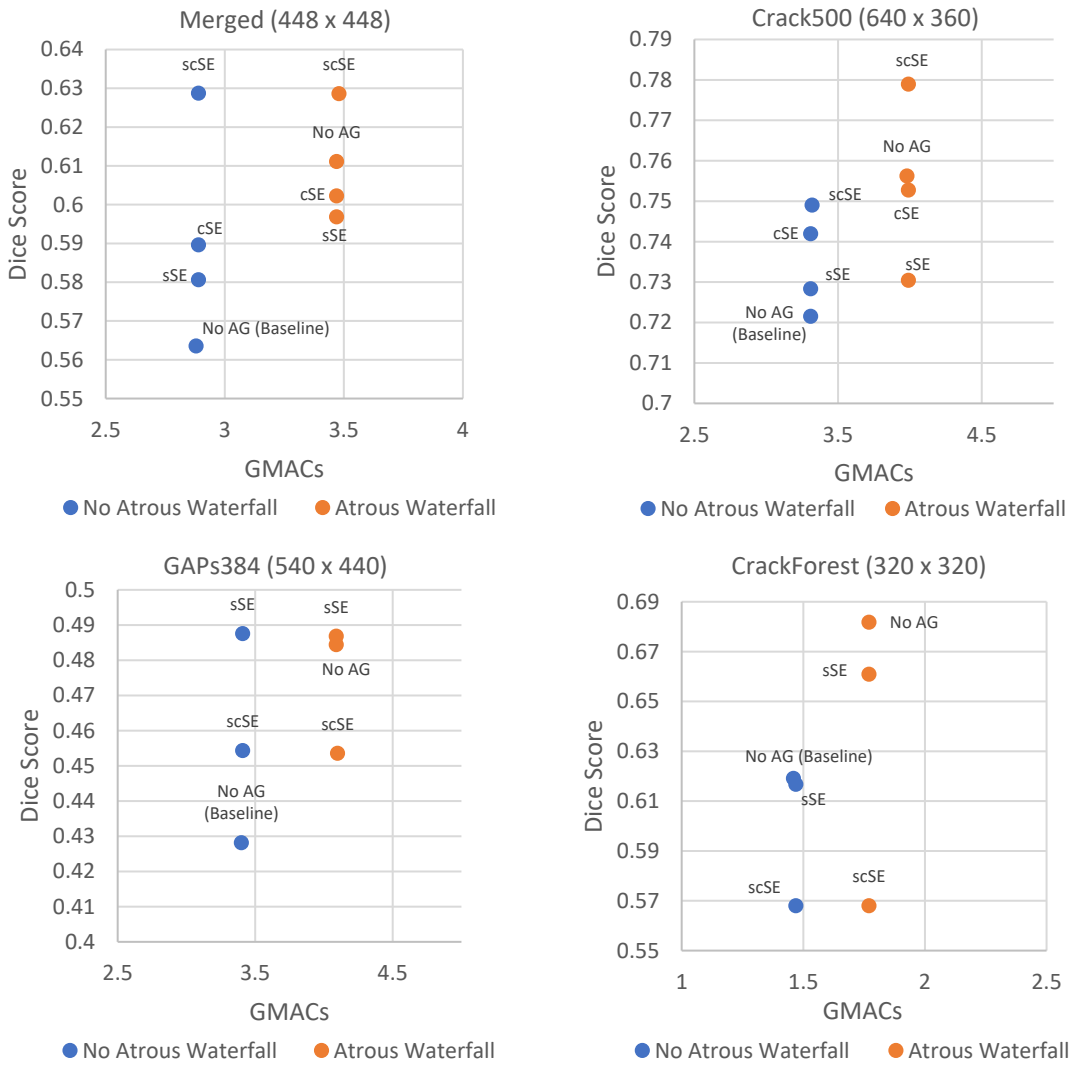


**Figure 46: The effect of adding the atrous waterfall block and using different attention gates on the dice score and number of computations (GMACs)**

*Study Two*: In this study, the best performing model on each dataset in study one, in terms of the Dice score, is modified with MBConv blocks as described in section 6.2.3. The performance of each MBConv model is then compared to its non-MBConv counterpart and the baseline state-of-the-art U-Net model in terms of segmentation performance, parameter and computational efficiency, model size, and the average inference time per frame. Note that since the AWF and Non-AWF scSE models have nearly identical Dice scores and IoU on the merged dataset in study one, both models are considered in this study on the merged dataset.

In Tables 8-11, the best AUPRC, IoU, and Dice scores achieved on the validation set of each dataset, as well as the average FPS achieved for each model is reported, with the superior results bolded. The FPS reported in Tables 8-11 are based on tests run on a Jetson Nano with images of 320 × 320 and 640 × 360 resolution, the former being sufficient for prototyping with the AI-deck camera mounted on the Crazyflie 2.1 quadcopter, and the latter suitable for larger-scale, real-world image fault segmentation applications. Each model is run for 300 repetitions to ensure reliable time measurement, with the average running time reported and converted to FPS. On all datasets except the CrackForest dataset (Table 11), a modified version of the Efficient U-Net model outperforms the baseline U-Net model in Dice score, IoU, and AUPRC, while the FPS of all evaluated modified Efficient U-Net models is significantly higher compared to U-Net. On the merged dataset (Table 8), the best Dice score of 0.6461 is achieved (MBConv-EU-Net + AWF + scSE), a significant increase from a Dice score of 0.5723 achieved on U-Net, while performing 1.6x faster than U-Net. On the Crack500 dataset (Table 9), the best Dice score of 0.7890 is achieved (MBConv-EU-Net + AWF + scSE), a significant increase from 0.7087 achieved on U-Net, while performing 1.6x faster than U-Net. On the GAPs384 dataset (Table 10), the best Dice of 0.4875 is achieved (EU-Net + sSE), an increase from 0.4524 achieved on U-Net, while respectively performing 3.9x and 4.1x faster on 320 × 320 and 640 × 360 resolution images compared to U-Net, which is the maximal speedup reported in Table 10. In Table 8, a 3.1x and 3.3x speedup on respective 320 × 320 and 640 × 360 resolution images are maximally achieved. In Table 9, a 2.6x and 2.9x speedup on respective 320 × 320 and 640 × 360 resolution images are maximally achieved. In Table 11, a 3.6x and 3.9x speedup on respective 320 × 320 and 640 × 360 resolution images are maximally achieved.

Next, the MBConv models and their non-MBConv counterparts are compared. On the merged and Crack500 datasets, the MBConv variant outperforms the non-MBConv model counterpart in Dice score and IoU, which may be due in part to the addition of attention gating earlier in the network. However, the FPS achieved on the MBConv models is lower than their non-MBConv counterparts; this slowdown may be attributed to the depth-wise convolutions not optimally utilizing modern accelerators as noted in [70], despite the replacement of the depth-wise convolutions with standard convolutions in earlier layers.

**Table 8: Comparison of the U-Net (baseline), Efficient U-Net, and MBConv Efficient U-Net models on the merged dataset**

| Merged (Pretraining) | AUPRC | IoU | Dice | FPS | |
|---|---|---|---|---|---|
| | | | | 320 × 320 | 640 × 360 |
| U-Net (Baseline) | 0.5854 | 0.4164 | 0.5723 | 4.7 | 2.2 |
| EU-Net + scSE | **0.7077** | 0.4717 | 0.6287 | **14.7** | **7.2** |
| MBConv-EU-Net + scSE | 0.6769 | 0.4466 | 0.6010 | 7.9 | 3.7 |
| EU-Net + AWF + scSE | 0.5945 | 0.4718 | 0.6286 | 12.1 | 6.3 |
| MBConv-EU-Net + AWF + scSE | 0.6855 | **0.4871** | **0.6461** | 7.6 | 3.6 |

**Table 9: Comparison of the U-Net (baseline), Efficient U-Net, and MBConv Efficient U-Net models on the Crack500 dataset**

| Crack500 | AUPRC | IoU | Dice | FPS | |
|---|---|---|---|---|---|
| | | | | 320 × 320 | 640 × 360 |
| U-Net (Baseline) | 0.5384 | 0.5706 | 0.7087 | 4.7 | 2.2 |
| EU-Net + AWF + scSE | 0.8545 | 0.6463 | 0.7789 | **12.1** | **6.3** |
| MBConv-EU-Net + AWF + scSE | **0.8848** | **0.6578** | **0.7890** | 7.6 | 3.6 |

**Table 10: Comparison of the U-Net (baseline), Efficient U-Net, and MBConv Efficient U-Net models on the GAPs384 dataset**

| GAPs384 | AUPRC | IoU | Dice | FPS | |
|---|---|---|---|---|---|
| | | | | 320 × 320 | 640 × 360 |
| U-Net (Baseline) | 0.5401 | 0.2998 | 0.4524 | 4.7 | 2.2 |
| EU-Net + sSE | **0.5600** | **0.3276** | **0.4875** | **18.2** | **9.1** |
| MBConv-EU-Net + sSE | 0.5306 | 0.3066 | 0.4634 | 9.7 | 4.5 |

**Table 11: Comparison of the U-Net (baseline), Efficient U-Net, and MBConv Efficient U-Net models on the CrackForest dataset**

| CrackForest | AUPRC | IoU | Dice | FPS | |
|---|---|---|---|---|---|
| | | | | 320 × 320 | 640 × 360 |
| U-Net (Baseline) | **0.6921** | **0.5427** | **0.7011** | 4.7 | 2.2 |
| EU-Net + AWF | 0.6437 | 0.5203 | 0.6818 | **16.9** | **8.5** |
| MBConv-EU-Net + AWF | 0.6596 | 0.4427 | 0.6085 | 8.1 | 3.9 |

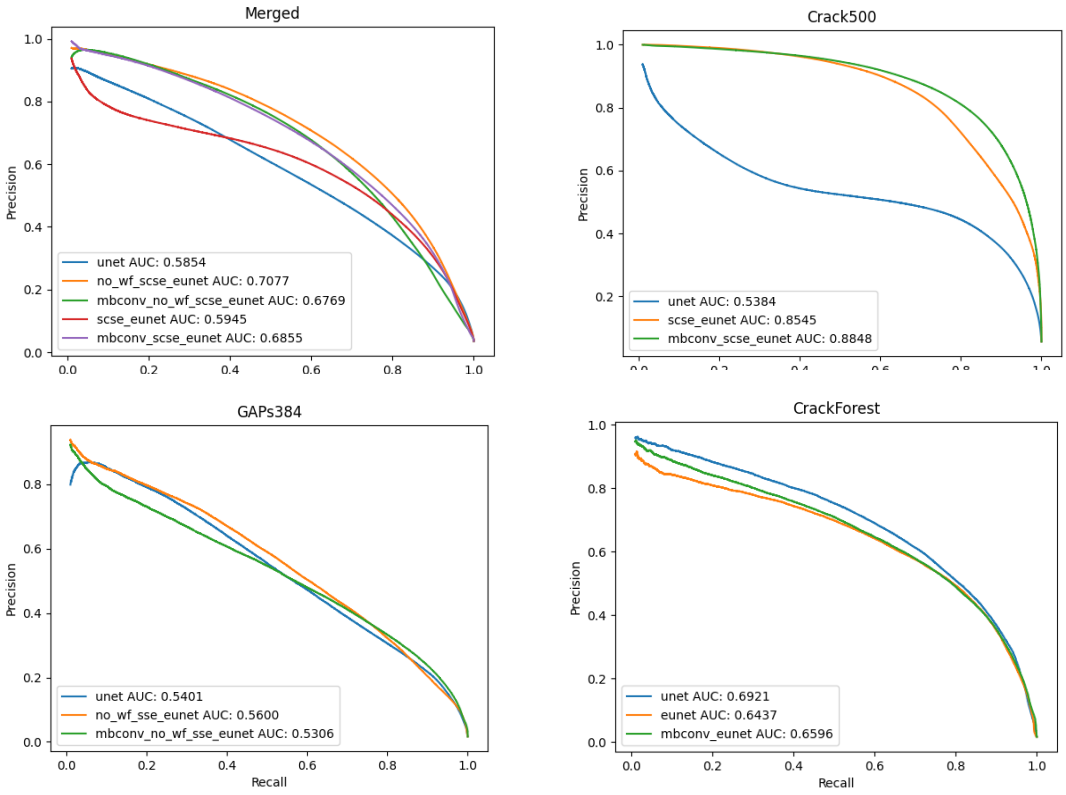


**Figure 47: The Precision-Recall Curves (PRC) and corresponding AUPRC scores for the U-Net (baseline), Efficient U-Net, and MBConv Efficient U-Net models**

In Table 12 below, the number of parameters, model size, and GMACs required during inference for the baseline U-Net, Efficient U-Net and MBConv counterparts are reported. The Efficient U-Net models have significantly less parameters and computations than U-Net, with an average decrease of 77% in the number of parameters. Moreover, the Efficient U-Net model sizes are significantly decreased compared to U-Net, with an average 76% size reduction. There is also a notable computational and parameter size reduction with the MBConv models in comparison to their non-MBConv counterparts, which makes sense given that the depth-wise separable convolutions in the MBConv blocks reduce the number of computations and parameters.

**Table 12: Comparison of the model size, number of parameters and computational efficiency**

| Model Version | #Params | Model Size (MB) | GMACs | | | |
|---|---|---|---|---|---|---|
| | | | 320 × 320 | 448 × 448 | 540 × 440 | 640 × 360 |
| U-Net (Baseline) | 1927.01k | 22.1 | 16.35 | 32.07 | 37.93 | 36.81 |
| EU-Net + scSE | 447.05k | 5.24 | 1.47 | 2.89 | 3.41 | 3.32 |
| MBConv-EU-Net + scSE | 372.38k | 4.51 | 0.90 | 1.77 | 2.10 | 2.03 |
| EU-Net + AWF + scSE | 625.61k | 7.34 | 1.77 | 3.48 | 4.10 | 3.99 |
| MBConv-EU-Net + AWF + scSE | 377.52k | 4.62 | 0.91 | 1.79 | 2.12 | 2.05 |
| EU-Net + sSE | 425.54k | 4.99 | 1.47 | 2.89 | 3.41 | 3.31 |
| MBConv-EU-Net + sSE | 369.69k | 4.46 | 0.89 | 1.74 | 2.07 | 2.00 |
| EU-Net + AWF | 603.88k | 7.08 | 1.77 | 3.47 | 4.09 | 3.98 |
| MBConv-EU-Net + AWF | 376.12k | 4.59 | 0.91 | 1.79 | 2.12 | 2.05 |

# Chapter 7

# 7 Discussion and Conclusions

The need for automated infrastructure inspection calls for the application of drones, which have the capability to autonomously navigate structures and process data. In remote environments, due to limited connectivity, a companion computer mounted onto a drone is the most viable solution. However, given the processing and memory constraints of companion computers, achieving real-time processing for fault segmentation and tracking is a difficult task.

In this thesis, a prototype of a fully autonomous drone infrastructure inspection system is proposed and developed for real-time fault segmentation, designed for computationally constrained environments. To develop this prototype, a Crazyflie 2.1 quadcopter is equipped with a flow deck for navigation, a multi-ranger deck for obstacle avoidance, and an AI-deck with an attached $320 \times 320$ grayscale camera for capturing and sending a live image stream via Wi-Fi to the Jetson Nano, a viable companion computer. Next, robust line estimation algorithms are investigated, wherein a modified version of the RANSAC algorithm is implemented, using a quadtree and a smart sampling approach. The modified RANSAC is evaluated in comparison to the baseline RANSAC algorithm. The modified RANSAC implementation achieves lower MSE on the outputted inlier set, particularly for larger $\sigma$, and generally achieves a lower MSE between the predicted slope of the fitted line and the actual slope. Also, as the outlier ratio and sample size increases, the greater the reduction is in the running time achieved by the modified RANSAC over the baseline RANSAC algorithm, with processing times well below the 1 second mark without GPU acceleration. Thus, by using the modified RANSAC algorithm for robust line estimation, the direction line of the faults can be efficiently determined for near real-time drone fault tracking.

Also proposed and developed in this thesis is the Efficient U-Net CNN architecture, based upon U-Net, ENet, Squeeze-and-Excitation Networks, and the addition of attention gating and an atrous waterfall block inspired by [29]. Further investigated is the importance of attention gating, with variations of the cSE, sSE and scSE blocks implemented in different

models. Experimental results show that scSE attention gating works well on the larger trained datasets, and that in general, AG models outperform the non-AG models in Dice score and IoU. Specifically, on visual investigation of the segmented label maps, sSE attention gating appears superior, capable of extracting fine, narrow cracks. Also investigated is the importance of the atrous waterfall block, in which it was found that models including such block generally outperformed the models omitting it, in terms of statistical and visual performance. Furthermore, the best performing models from the above study are evaluated and compared against versions of the models wherein the standard bottleneck blocks are replaced with MBConv blocks. In the larger merged and crack500 datasets, the MBConv model outperforms the non-MBConv model in Dice score and IoU but achieves a lower FPS. All custom models except those tested on the CrackForest dataset outperform U-Net, with the MBConv models in the merged and Crack500 datasets achieving an increase of 0.0738 and 0.0803 in the Dice score, respectively. Also, all models run faster at inference than U-Net, with 18.2 FPS being the highest achieved on $320 \times 320$ images processed on a Jetson Nano, compared to 4.7 FPS achieved by the baseline U-Net. Thus, the proposed Efficient U-Net model variants can achieve real-time or near real-time speeds for fault segmentation on a computationally constrained device, while outperforming the baseline U-Net model in both inference speed and segmentation capability. This enables the drone to adequately extract the faults in real-time during the inspection, which holds promise for larger-scale drone inspection applications.

## 7.1 Limitations and Future Work

For fault segmentation, although the MBConv models achieve a high performance, they perform more slowly than their non-MBConv counterparts, despite requiring less parameters and computations. This is likely due to the depth-wise convolutions, and as such, further running time tests should be performed in the future in which all MBConv blocks are replaced with the Fused-MBConv blocks. This way, all depth-wise convolutions are replaced with standard convolutions. Furthermore, although the MBConv models performed better on some datasets, it is not evident enough from testing how much of this performance improvement is due to the addition of attention gating earlier in the networks. Thus, further network modifications are needed to better determine the impact of early

attention gating. Also, not tested is the impact of introducing pixel tolerances on segmentation performance. As noted in related works, pixel tolerances can help handle annotation inconsistencies across different datasets and can result in considerable improvement in statistical results, as shown in [29].

Although training is performed on various datasets, it may be advantageous to augment the datasets with images of structural defects captured by the camera mounted onto the drone. Furthermore, the publicly available datasets used in this thesis are relatively limited in the number of images with background noise and occlusion, which further motivates the use-case of augmentation through the addition of drone-captured images during inspection. Also, prior to each round of training, the data batches are randomly selected during evaluation; in future studies, using a constant seed to reduce variability in the sampled data may be worth consideration.

In terms of the modified RANSAC algorithm, although promising results are obtained on a CPU, modifying the implementation of the algorithm such that it can be executed on the GPU for further acceleration is another possible area of future investigation.

In structural health monitoring (SHM), being able to not only detect the presence of structural faults but to also determine *where* these faults occur on structures is of great value for localization in three-dimensional (3D) space. Moreover, being able to visualize the spread of the faults and see the bigger picture would allow engineers to make better decisions for treating these faults and determining their severity. However, images captured via digital cameras are limited in terms of how much information they can capture, given that they only provide a two-dimensional (2D) representation of the physical environment. Moreover, images taken at a relative proximity to the inspected structure can only capture a fraction of entire structure; unless taken from afar, it would be very difficult to garner any topological sense of the structure for mapping and 3D reconstruction purposes from individual images alone. To address this, it may be advantageous to produce a 3D fault map, which maps fault points from a 2D camera coordinate system to a 3D global coordinate system. This can be achieved by approximating structural points based on sensor distance readings, estimated drone position, and camera pose. Furthermore,

quantifying the mapped faults in terms of depth, width and spread would help to better assess the severity of the structural damage.

The next step for this system as a whole is to scale it up to a larger drone onto which the Jetson Nano may be mounted, along with multidirectional ToF sensors for obstacle avoidance and a higher resolution camera for structural image capturing. Such a drone should be capable of flying outdoors in varying conditions and be deployed remotely without need for manual intervention on-site. Furthermore, the drone should be able to locate the structure to travel to via GPS coordinates passed to the system. In the current system, the drone is only capable of flying to an $x$, $y$ coordinate location relative to its starting position. Having the drone autonomously navigate around large structures for extended periods of time can easily drain its battery life, thus calling for the need to synchronize multiple drones for continuous inspection.

# Bibliography


[1] Transport Canada, “Road transportation,” *Transport Canada,* 13-Jul-2020. [Online]. Available: https://tc.canada.ca/en/corporate-services/policies/road-transportation.

[2] “Bridges,” *ASCE's 2021 Infrastructure Report Card* |, 21-Jan-2022. [Online]. Available: https://infrastructurereportcard.org/cat-item/bridges/.

[3] Y. Xu and Y. Turkan, “Brim and UAS for bridge inspections and management,” *Engineering, Construction and Architectural Management*, vol. 27, no. 3, pp. 785–807, 2019.

[4] “Value for money audit: Inspection and ... - auditor.on.ca.” [Online]. Available: https://www.auditor.on.ca/en/content/annualreports/arreports/en21/AR_InspectBridges_en21.pdf.

[5] J. Valença, I. Puente, E. Júlio, H. González-Jorge, and P. Arias-Sánchez, “Assessment of cracks on concrete bridges using image processing supported by Laser Scanning survey,” *Construction and Building Materials*, vol. 146, pp. 668–678, 2017.

[6] “Design of a bridge inspection system (bis) to reduce time ...” [Online]. Available: https://catsr.vse.gmu.edu/SYST490/490_2014_BI/BIS_FinalReport.pdf.

[7] A. M. Abdallah, “A Study on Bridge Inspections: Identifying Barriers to New Practices and Providing Strategies for Change.” Order No. 28547957, Colorado State University, Ann Arbor, 2021.

[8] “Technical documents,” *GSA*, 30-Nov-2017. [Online]. Available: https://www.gsa.gov/node/1100?Form_Load=88348.

[9] “Active crack,” *Active Crack - an overview | ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/engineering/active-crack#:~:text=Dormant%20cracks%20may%20be%20induced,cracks%20and%20indicate%20severe%20problems.

[10] /author/concrete-Construction-Staff, “D-cracking of pavements,” *Concrete Construction*, 10-Feb-2011. [Online]. Available: https://www.concreteconstruction.net/how-to/d-cracking-of-pavements_o.

[11] “Structural Health Monitoring,” *Structural Health Monitoring - an overview | ScienceDirect Topics*.[Online]. Available: https://www.sciencedirect.com/topics/engineering/structural-health-monitoring#:~:text=Structural%20health%20monitoring%20(SHM)%20system,reliability%20and%20life%20cycle%20management.

[12] S. Sony, K. Dunphy, A. Sadhu, and M. Capretz, “A systematic review of convolutional neural network-based structural condition assessment techniques,” *Engineering Structures*, vol. 226, p. 111347, 2021.

[13] B. Desai, U.Kushwaha, S.Jha, “Image Filtering -Techniques, Algorithm and Applications,” *GIS Science Journal*, vol. 7, p. 970, 2020.

[14] Q. Mei and M. Gül, “Multi-level feature fusion in densely connected deep-learning architecture and depth-first search for crack segmentation on images collected with smartphones,” *Structural Health Monitoring*, vol. 19, no. 6, pp. 1726–1744, 2020.

[15] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[16] M. Zhong, L. Sui, Z. Wang, and D. Hu, “Pavement crack detection from Mobile Laser Scanning Point Clouds using a time grid,” *Sensors*, vol. 20, no. 15, p. 4198, 2020.

[17] K. Gopalakrishnan, H. Gholami, and A. Agrawal, “Crack damage detection in unmanned aerial vehicle images of civil infrastructure using pre-trained deep learning model,” *INTERNATIONAL JOURNAL FOR TRAFFIC AND TRANSPORT ENGINEERING*, vol. 8, no. 1, pp. 1–14, 2018.

[18] Q. Zhang, K. Barri, S. K. Babanajad, and A. H. Alavi, “Real-time detection of cracks on concrete bridge decks using deep learning in the frequency domain,” *Engineering*, vol. 7, no. 12, pp. 1786–1796, 2021.

[19] E. Jeong, J. Seo, and J. Wacker, “Literature review and technical survey on bridge inspection using unmanned aerial vehicles,” *Journal of Performance of Constructed Facilities*, vol. 34, no. 6, p. 04020113, 2020.

[20] G. Gugan and A. Haque, “Towards the development of a robust path planner for autonomous drones,” *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020.

[21] “# telemetry radios/modems,” *Telemetry Radios/Modems | PX4 User Guide*. [Online]. Available: https://docs.px4.io/master/en/telemetry/.

[22] “# basic concepts,” *Basic Concepts | PX4 User Guide*. [Online]. Available: https://docs.px4.io/master/en/getting_started/px4_basic_concepts.html.

[23] K. Máthé and L. Buşoniu, “Vision and control for uavs: A survey of general methods and of inexpensive platforms for infrastructure inspection,” *Sensors*, vol. 15, no. 7, pp. 14887–14916, 2015.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional Neural Networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[26] K. Simonyan, and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv*: 1409.1556, 2014.

[27] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] R. Augustauskas and A. Lipnickas, "Improved pixel-level pavement-defect segmentation using a Deep Autoencoder," *Sensors*, vol. 20, no. 9, p. 2557, 2020.

[30] M. David Jenkins, T. A. Carr, M. I. Iglesias, T. Buggy, and G. Morison, "A deep convolutional neural network for semantic pixel-wise segmentation of road and pavement surface cracks," *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018.

[31] F. Yang, L. Zhang, S. Yu, D. Prokhorov, X. Mei, and H. Ling, "Feature pyramid and hierarchical boosting network for pavement crack detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1525–1535, 2020.

[32] L. Xiao, W. Li, J. Huyan, Z. Sun, and S. Tighe, "Crack grid detection and calculation based on Convolutional Neural Network," *Canadian Journal of Civil Engineering*, vol. 48, no. 9, pp. 1192–1205, 2021.

[33] X. Zhao, S. Li, H. Su, L. Zhou, and K. J. Loh, "Image-based comprehensive maintenance and inspection method for bridges using Deep Learning," Volume 2: Mechanics and Behavior of Active Materials; Structural Health Monitoring; *Bioinspired Smart Materials and Systems; Energy Harvesting; Emerging Technologies*, 2018.

[34] Y.-J. Cha, W. Choi, and O. Büyüköztürk, "Deep learning-based crack damage detection using convolutional neural networks," *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, no. 5, pp. 361–378, 2017.

[35] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[36] R. Girshick, "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[37] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[38] Y.-J. Cha, W. Choi, G. Suh, S. Mahmoudkhani, and O. Büyüköztürk, "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 9, pp. 731–747, 2017.

[39] D. Ma, H. Fang, N. Wang, B. Xue, J. Dong, and F. Wang, "A real-time crack detection algorithm for pavement based on CNN with multiple feature layers," *Road Materials and Pavement Design*, pp. 1–17, 2021.

[40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[41] Y. Li, Z. Han, H. Xu, L. Liu, X. Li, and K. Zhang, "Yolov3-Lite: A lightweight crack detection network for aircraft structure based on depth-wise separable convolutions," *Applied Sciences*, vol. 9, no. 18, p. 3781, 2019.

[42] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation*," Lecture Notes in Computer Science*, pp. 234–241, 2015.

[44] S. L. Lau, E. K. Chong, X. Yang, and X. Wang, "Automated pavement crack segmentation using u-net-based convolutional neural network," *IEEE Access*, vol. 8, pp. 114892–114899, 2020.

[45] N. Beheshti and L. Johnsson, "Squeeze U-net: A memory and energy efficient image segmentation network," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[46] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, and B. Nessler, "Speeding up semantic segmentation for autonomous driving," *Conference on Neural Information Processing Systems (NIPS 2016)*, 2016.

[47] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[48] C. V. Dung and L. D. Anh, "Autonomous concrete crack detection using deep fully convolutional neural network," *Automation in Construction*, vol. 99, pp. 52–58, 2019.

[49] T. Chen, Z. Cai, X. Zhao, C. Chen, X. Liang, T. Zou, and P. Wang, "Pavement crack detection and recognition using the architecture of segnet," *Journal of Industrial Information Integration*, vol. 18, p. 100144, 2020.

[50] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size," *arXiv preprint arXiv*:1602.07360, 2016.

[51] V. Hoskere, Y. Narazaki, T. Hoang, and B. Spencer Jr, "Towards automated post-earthquake inspections with deep learning-based condition-aware models," *arXiv preprint arXiv*:1809.09195, 2018.

[52] F. Kucuksubasi and A. G. Sorguc, "Transfer learning-based crack detection by autonomous uavs," *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC)*, 2018.

[53] H. S. Munawar, F. Ullah, A. Heravi, M. J. Thaheem, and A. Maqsoom, "Inspecting buildings using drones and Computer Vision: A machine learning approach to detect cracks and damages," *Drones*, vol. 6, no. 1, p. 5, 2021.

[54] W. Wu, M. A. Qurishee, J. Owino, I. Fomunung, M. Onyango, and B. Atolagbe, "Coupling deep learning and UAV for Infrastructure Condition Assessment Automation," *2018 IEEE International Smart Cities Conference (ISC2)*, 2018.

[55] J.-H. Lee, S.-S. Yoon, H.-J. Jung, and I.-H. Kim, "Diagnosis of crack damage on structures based on image processing techniques and R-CNN using Unmanned Aerial Vehicle (UAV)," *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2018*, 2018.

[56] L. Ali, N. K. Valappil, D. N. Kareem, M. J. John, and H. Al Jassmi, "Pavement crack detection and localization using convolutional neural networks (cnns)," *2019 International Conference on Digitization (ICD)*, 2019.

[57] A. Reddy, V. Indragandhi, L. Ravi, and V. Subramaniyaswamy, "Detection of cracks and damage in wind turbine blades using artificial intelligence-based Image Analytics," *Measurement*, vol. 147, p. 106823, 2019.

[58] S. Mohan, O. Shoghli, A. Burde, and H. Tabkhi, "Low-power drone-mountable real-time Artificial Intelligence Framework for road asset classification," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2675, no. 1, pp. 39–48, 2020.

[59] Y. Z. Ayele, M. Aliyari, D. Griffiths, and E. L. Droguett, "Automatic Crack Segmentation for UAV-assisted bridge inspection," *Energies*, vol. 13, no. 23, p. 6250, 2020.

[60] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv*:1704.04861, 2017.

[61] K. McGuire, "Do ai decks dream of tutorials?," *Bitcraze*. [Online]. Available: https://www.bitcraze.io/2021/09/do-ai-decks-dream-of-tutorials/.

[62] "Understanding Ros 2 topicsℑ," *Understanding ROS 2 topics - ROS 2 Documentation: Foxy documentation*. [Online]. Available: https://docs.ros.org/en/foxy/Tutorials/Topics/Understanding-ROS2-Topics.html.

[63] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv*:1606.02147, 2016.

[64] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE international conference on computer vision*, pp. 1026-1034. 2015.

[65] J. Elseberg, D. Borrmann, and A. Nüchter, "One Billion points in the cloud – an octree for efficient processing of 3D laser scans," ISPRS Journal of Photogrammetry and Remote Sensing, vol. 76, pp. 76–88, 2013.

[66] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[67] A. G. Roy, N. Navab, and C. Wachinger, “Concurrent spatial and channel ‘squeeze & excitation’ in fully convolutional networks,” *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pp. 421–429, 2018.

[68] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[69] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *International conference on machine learning*, pp. 6105-6114, PMLR, 2019.

[70] M. Tan and Q. Le, “Efficientnetv2: Smaller models and faster training,” *International Conference on Machine Learning*, pp. 10096-10106, PMLR, 2021.

[71] L. Zhang, F. Yang, Y. Daniel Zhang, and Y. J. Zhu, “Road crack detection using deep convolutional neural network,” *2016 IEEE International Conference on Image Processing (ICIP)*, 2016.

[72] M. Eisenbach, R. Stricker, D. Seichter, K. Amende, K. Debes, M. Sesselmann, D. Ebersbach, U. Stoeckert, and H.-M. Gross, “How to get pavement distress detection ready for deep learning? A systematic approach,” *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[73] Y. Shi, L. Cui, Z. Qi, F. Meng, and Z. Chen, “Automatic road crack detection using random structured forests,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 12, pp. 3434–3445, 2016.

[74] L. Cui, Z. Qi, Z. Chen, F. Meng, and Y. Shi, “Pavement distress detection using random decision forests,” *Data Science*, pp. 95–102, 2015.

[75] Q. Zou, Y. Cao, Q. Li, Q. Mao, and S. Wang, “Cracktree: Automatic crack detection from Pavement Images,” *Pattern Recognition Letters*, vol. 33, no. 3, pp. 227–238, 2012.

[76] R. Amhaz, S. Chambon, J. Idier, and V. Baltazart, “Automatic crack detection on two-dimensional pavement images: An algorithm based on minimal path selection,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 10, pp. 2718–2729, 2016.

[77] D. Dais, İ. E. Bal, E. Smyrou, and V. Sarhosis, “Automatic Crack Classification and segmentation on masonry surfaces using convolutional neural networks and transfer learning,” *Automation in Construction*, vol. 125, p. 103606, 2021.

[78] “Wiki,” *ros.org*. [Online]. Available: http://wiki.ros.org/Documentation.

[79] “Crazyflie Platform Overview,” *Bitcraze*. [Online]. Available: https://www.bitcraze.io/documentation/system/platform/.

[80] “Hardware architecture of the crazyflie 2.X,” *Bitcraze*. [Online]. Available: https://www.bitcraze.io/documentation/system/platform/cf2-architecture/.

[81] “Diva portal.” [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1440184/FULLTEXT01.pdf.

[82] Bitcraze, “Crazyflie-lib-python/commander.py at master · bitcraze/Crazyflie-lib-Python,” *GitHub*. [Online]. Available: https://github.com/bitcraze/crazyflie-lib-python/blob/master/cflib/crazyflie/commander.py.

[83] “CRTP - communication with the crazyflie,” *Bitcraze*. [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/crtp/.

[84] D. Eppstein, M. T. Goodrich, and J. Z. Sun, “The skip quadtree,” *Proceedings of the twenty-first annual symposium on Computational geometry - SCG '05*, 2005.

[85] C. Kone, “Introducing convolutional neural networks in Deep Learning,” *Medium*, 06-Nov-2019. [Online]. Available: https://towardsdatascience.com/introducing-convolutional-neural-networks-in-deep-learning-400f9c3ad5e9.

[86] R. Schnabel, R. Wahl, and R. Klein, “Efficient RANSAC for point-cloud shape detection,” *Computer Graphics Forum*, vol. 26, no. 2, pp. 214–226, 2007.

[87] Y. Liu, J. Yao, X. Lu, R. Xie, and L. Li, “DeepCrack: A deep hierarchical feature learning architecture for crack segmentation,” Neurocomputing, vol. 338, pp. 139–153, 2019.

[88] “File:example of a deep neural network.png - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:Example_of_a_deep_neural_network.png.

[89] “File:Quadcopter Camera drone in flight.jpg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:Quadcopter_camera_drone_in_flight.jpg.

[90] “File:aeronautics orbiter UAV.jpg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:Aeronautics_Orbiter_UAV.jpg.

# Appendices

## Appendix A: Experimental Results

**Table 13: Running times and FPS on a Jetson Nano (Fastest times bolded)**

| **Jetson Nano** | **Time per image (ms)** | | **FPS** | |
|---|---|---|---|---|
| | **320 × 320** | **640 × 360** | **320 × 320** | **640 × 360** |
| U-Net (Baseline) | 213 | 465 | 4.7 | 2.2 |
| EU-Net + scSE | 68 | 139 | 14.7 | 7.2 |
| MBConv-EU-Net + scSE | 126 | 269 | 7.9 | 3.7 |
| EU-Net + AWF + scSE | 83 | 158 | 12.1 | 6.3 |
| MBConv-EU-Net + AWF + scSE | 132 | 274 | 7.6 | 3.6 |
| EU-Net + sSE | **55** | **110** | **18.2** | **9.1** |
| MBConv-EU-Net + sSE | 103 | 223 | 9.7 | 4.5 |
| EU-Net + AWF | 59 | 118 | 16.9 | 8.5 |
| MBConv-EU-Net + AWF | 124 | 259 | 8.1 | 3.9 |

**Table 14: Running times and FPS on a GeForce GTX 1060 (Fastest times bolded)**

| **GeForce GTX 1060** | **Time per image (ms)** | | **FPS** | |
|---|---|---|---|---|
| | **640 × 360** | **1280 × 720** | **640 × 360** | **1280 × 720** |
| U-Net (Baseline) | 27 | 104 | 37.0 | 9.6 |
| EU-Net + scSE | 40 | 96 | 25.0 | 10.4 |
| MBConv-EU-Net + scSE | 75 | 245 | 13.3 | 4.1 |
| EU-Net + AWF + scSE | 44 | 103 | 22.7 | 9.7 |
| MBConv-EU-Net + AWF + scSE | 84 | 247 | 11.9 | 4.0 |
| EU-Net + sSE | **25** | **44** | **40.0** | **22.7** |
| MBConv-EU-Net + sSE | 63 | 189 | 15.9 | 5.3 |
| EU-Net + AWF | 28 | 47 | 35.7 | 21.3 |
| MBConv-EU-Net + AWF | 67 | 202 | 14.9 | 5.0 |

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Marlin Manka |
| **Post-secondary Education and Degrees:** | M.Sc. Candidate, Computer Science<br>The University of Western Ontario<br>2020-2022<br><br>B.Sc., Computer Science<br>The University of Western Ontario<br>2015-2020 |
| **Honours and Awards:** | Dean's Honor List<br>The University of Western Ontario<br>2016-2020 |
| **Related Work Experience** | Teaching Assistant<br>The University of Western Ontario<br>2020-2022 |