

**Learning-based Image Scale
Estimation for Quantitative Visual
Inspection of Civil Structures**

by

Ju An Park

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Applied Science

in

Civil Engineering

Waterloo, Ontario, Canada, 2021

© Ju An Park 2021

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Chapters 1, 4, 5, 6, and 7 of this thesis were written based on the paper:

Ju An Park, Chul Min Yeum, and Trevor D. Hrynyk. Learning-based image scale estimation using surface textures for quantitative visual inspection of regions-of-interest. *Computer-Aided Civil and Infrastructure Engineering*, 36(2):227–241, 2021.

I carried out all tasks related to the paper, such as data collection, programming, experiment validation, and paper drafting and writing, under the supervision of my supervisor, Dr. Yeum. Dr. Hrynyk has provided the idea for texture-based scale estimation and assisted with the review of the paper manuscript. The [letter of copyright permission](#) for this paper follows the bibliography. Note that some of these chapters have been modified and/or expanded in comparison to the paper.

Abstract

The number of assets of civil infrastructure (e.g., bridges or roads) have been increasing to meet the demands of growing populations around the world. However, they degrade over time due to environmental factors and must be maintained and monitored to ensure the safety of its users. The increasing number of infrastructure assets which deteriorate over time is fast outpacing the rate at which they are inspected and rehabilitated. Currently, the main mode of structure condition assessment is visual inspection, where human inspectors manually identify, classify, track, and measure, as needed, deterioration over time to make assessments of a structure's overall condition. However, the current process is highly time consuming, expensive, and subject to the inspector's judgement and expertise, which could lead to inconsistent assessments of a given structure when surveyed by several different inspectors over a period of time. As a result, there is a clear need for the current inspection process to be improved in terms of efficiency and consistency.

Developments in computer vision algorithms, vision sensors, sensing platforms, and high-performance computing have shown promise in improving the current inspection processes to enable consistent and rapid structural assessments. Recent work often involves rapid collection and/or analysis of imagery captured from personnel or mobile data collection platforms (e.g., smart phones, unmanned aerial or ground vehicles) to detect and classify visual features (e.g., structural components or deterioration). These works often involve the use of advanced image processing or computer vision algorithms such as convolutional neural networks to detect and/or classify regions of interest. However, a major shortfall of vision-based inspection is the inability to deduce physical measurements (e.g.,

mm or cm) from the collected images. The lack of an image scale (e.g., pixel/mm) on 2D images does not permit quantitative inspection.

To address this challenge, a learning-based scale estimation technique is proposed. The underlying assumption is that the surface texture of structures, captured in images, contains enough information to estimate scale for each corresponding image (e.g., pixel/mm). This permits the training of a regression model to establish the relationship between surface textures in images and their scales. A convolutional neural network was trained to extract scale-related features from textures captured in images. The trained model is used to estimate scales for all images captured from surfaces of a structure with similar textures in subsequent inspections. The capability of the proposed technique was demonstrated using data collected from surface textures of three different structures. An average scale estimation error, from images of each structure, is less than 15%, which is acceptable in typical visual inspection settings. The source code and data are available from a data repository (GitHub)¹.

¹Code available on [GitHub](https://github.com/cviss-lab/LISE) (<https://github.com/cviss-lab/LISE>).

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Yeum. Thank you for your guidance, patience, and support throughout my studies. Starting from your supervision of my undergraduate capstone project to the end of my MASc studies, your input and feedback in all aspects of our work have been invaluable and have allowed me to contribute so much more than I would have otherwise during my time as a master's student. Thank you for a time well spent, for all the discussions, and for the free food!

I would also like to express my gratitude to Dr. Hrynyk and Dr. Thomson (who was my undergraduate capstone course professor!) for giving their invaluable time to read through the manuscript to provide comments and feedback to improve the thesis quality.

To my lab mates, in order of admittance to the CVISS lab, Max, Zaid, and Rishabh, thanks for being a cool bunch! Max, research has been so much more enjoyable because of your company during and before COVID-19, thanks!

I am grateful for the friends I have met through my graduate studies, Steven for setting up that dodge ball team where I got to meet many more friends. Despite legendarily losing almost all games, it was fun (go Team Haas!). And Cristobal, for being a good neighbour lab mate!

In an era of COVID-19, wrapping up the thesis from home, my father and mother have been a constant source of support and encouragement during this time. Thank you two for being my life mentors and walking with me in life to get this far. I always appreciate your love and support.

I would also like to thank Nowwar, who has also provided me with encouragement and support throughout during my research and preparation of the thesis. You have helped me see the world in a broader perspective. I am sincerely grateful to have you in my life.

Table of Contents

List of Figures	xi
List of Tables	xvi
1 Introduction	1
2 Literature Review	8
2.1 Vision-based inspection	9
2.1.1 Detection	9
2.1.2 Localization	12
2.1.3 Quantification	14
2.2 Research Gap	15
3 Convolutional Neural Networks	17
3.1 Overview	17

3.2	Convolutional Layer	19
3.3	Depthwise Separable Convolutional Layer	22
3.4	Pooling Layer	23
3.5	Dense Layer (Fully Connected Layers)	25
3.6	Dropout Layer	27
3.7	Batch Normalization Layer	28
3.8	Bottleneck Residual Block	29
4	Proposed Approach	31
4.1	Overview	31
4.2	Image regression using CNNs	36
4.3	Automated ground-truth training data generation	41
5	Experimental Design	44
5.1	Data collection	46
5.2	Model training	51
6	Results and Discussion	55
7	Conclusions and Future Work	65
7.1	Conclusions	65
7.2	Future Work	67

References	70
Letter of copyright permission	79
Appendix	86

List of Figures

1.1	Sample images of structural deterioration of concrete components: (a) Medium cracking on abutment wall, (b) severe delamination, spalling, and corrosion of exposed reinforcement on ballast wall, and (c) severe delamination and spalling of pier column with exposed reinforcement. Retrieved from [38] . . .	2
1.2	Image scale estimation using a (a) pen, (b) crack gauge, and (c) measuring tape. These images contain scenes of damaged components with objects included in the scene and have been collected in real environments by engineers during their field inspection [59]	5
1.3	Images captured from the same concrete wall but different locations. The image in (b) is captured at a closer distance than the one in (a)	6

3.1	An illustration of a conventional CNN model architecture. The CNN model can be described in two parts: the base model and the top layer. The features of an image relevant to the label being predicted are extracted through a series of convolutional and pooling layers in the base layer. The features then interpreted in the top layer through one or more sets of hidden dense layers to output a single prediction. The flattened layer simply is an unravelling of the previous layer’s output vector. Note that only a part of the weights has been shown from the flattened to the hidden dense layer to maintain simplicity	19
3.2	A sample convolution operation. The input matrix of size $5 \times 5 \times 1$ is convolved with the $3 \times 3 \times 1$ kernel with stride of 1. For this example, the Sobel filter in the horizontal direction of size $3 \times 3 \times 1$ is used to identify horizontal edge features. Each sub-box (also known as a window) is convolved with the filter to produce the coloured sub-box values in the convolved matrix. The ReLu activation function is then applied to the convolved matrix to yield the output vector of the convolutional layer	20
3.3	Sample pooling operations. In (a), an input matrix of size 4×4 is max pooled using a kernel size of 2×2 to produce a 2×2 matrix. In (b), 2D global average pooling is performed on an input matrix of size $4 \times 4 \times 4$ to produce a $1 \times 1 \times 4$ matrix	24

3.4	A sample dense layer example, using a 1D input matrix of size 2, a 3-neuron hidden dense layer with ReLu activation function and a bias of 0, and a 1-neuron output dense layer with a sigmoid activation function. The neuron consists of two values, where the left value is the weighted sum between the inputs and their corresponding weights, and the right value is the left value after the activation has been applied. The right value of the neuron is then output to the next layer for all the neurons in the dense layer	27
4.1	Overview of the proposed scale estimation technique	32
4.2	A 5 x 5 Aruco fiducial marker used for this study.	42
5.1	Overview of data collection and training data set generation methodology .	46
5.2	Overviews of test structures: (a) pedestrian bridge (PED), (b) building wall (BW), and (c) asphalt pavement (ASH)	47
5.3	Sample images collected from (a) PED, (b) BW, and (c) ASH. Note that the original images with a marker on the right and left are captured from close and far distance, respectively. The square patch next to the image is a magnified area corresponding to a box on the original image	48
5.4	Sample images collected from PED using (a) different camera and (b) different focal lengths. Like Figure 5.3, two images on the left and right are captured from different distances and a magnified area corresponding to a box on each image is on the right	50

5.5	Sample texture patches from an original image: Patches are randomly extracted (shown as red squares) from non-marker areas	51
6.1	Loss curves on a log-linear plot of training in (a) and testing in (b) for three different patch sizes: 100×100 , 350×350 , and 850×850 . The PED data set is used for this experiment	57
6.2	Actual-to-Predicted scale scatter (AtP) plots obtained from the PED training data set: (a) scales for all patches and (b) aggregated scales for each image using a median function. The red line indicates a correct prediction line and inner dashed and outer coloured bands indicate 10% and 20% error margins, respectively	58
6.3	AtP plots obtained from the PED testing data set: (a) scales for all patches and (b) aggregated scales for each image using a median function	59
6.4	AtP plots for (a) BW and (b) ASH testing data sets. The scales are aggregated using a median function	60
6.5	AtP plots for new PED testing data sets collected (a) using a different camera and (b) under different focal lengths. The scale estimation model in Figure 6.2 is applied to these two data sets	62

6.6 Box and whisker plots for (a) ASH, (b) BW, and (c) PED, ZOOM, and DIFF testing data sets. Error statistics of non-aggregated and aggregated scales are shown side-by-side for each data set. The error mean and median are represented as the X mark and solid line through the box, respectively. The box represents the inter-quartile range, where its upper and lower edge 75th and 25th percentile. The top and bottom of the vertical line represent the largest and lowest data point, excluding outliers. Error statistics shown for the aggregated scales are aggregated using the median function 64

List of Tables

5.1	Overview of data sets collected at different structures	49
5.2	Description of the augmented MobileNetV2 used for this study. Each line describes a single layer/block repeated n times, which have the same number of output channels, c . Variables $k_{m \times n}$, t , c , n , and s are kernel height and width, expansion factor, number of output channels, number of times the layer/block is repeated, and stride, respectively. Note that s describes the stride for the first layer/block in a series of repeated layers/blocks, and the stride after the first layer/block is 1 (e.g., for Conv 2D with $s = 2$ and $n = 2$, $s = 2$ for the first Conv 2D, and $s = 1$ for the 2nd Conv 2D). $k_{m \times n}$ for bottleneck residual blocks describes the kernel size for its depthwise convolution layer. The expansion factor is applied to the input channel length.	53
6.1	Overall scale prediction results for all three structures: aggregation using either a mean or median function	60

Chapter 1

Introduction

The deterioration of existing civil infrastructure, such as buildings, bridges, and nuclear power plants, is outpacing the ability to replace them. Lean capital budgets for construction and maintenance severely limit the amount of proactive rehabilitation work that can be undertaken. As a result, the inventory of aging and degrading infrastructure is growing and there is a pressing need for new technologies to assess the conditions of infrastructure reliably and economically.

Changes in external appearance of structures have historically been reliable indicators of structural degradation. Today, visual inspection remains a major component in maintenance and management of structures. For example, in accordance with the Ontario Structure Inspection Manual (OSIM) provided by the Ministry of Transportation of Ontario (MTO), transportation structures should be inspected by engineers, within the inspector's arm's length, at least every 2 years [38]. During these inspections, engineers

are required to classify and quantify types, condition, and area of material deterioration such as ones shown in Figure 1.1 (e.g., visual cracks, corrosion, and spalling) [38]. Existing visual inspection processes are largely manual operations and rely heavily on engineering judgement, which is time-intensive, and may lead to inconsistent assessments.



(a)



(b)



(c)

Figure 1.1: Sample images of structural deterioration of concrete components: (a) Medium cracking on abutment wall, (b) severe delamination, spalling, and corrosion of exposed reinforcement on ballast wall, and (c) severe delamination and spalling of pier column with exposed reinforcement. Retrieved from [38]

Recent advances in computer vision technologies, new vision sensors, sensing platforms, and high-performance computing, have transformed the way in which structures are inspected [1]. As sensors become smaller, lower in cost, and more powerful, a larger volume of high-quality visual data can be captured from structure inspections with high spatial

and temporal granularity. Powerful computer vision methods and machine learning algorithms enable automatic extraction of visual features and semantic information to detect visual changes of structures, which may be an indicator of structural damage. Modern computing offering unprecedented speed and performance, and state-of-the-art processing of visual data, has enabled opportunities to improve inspection practices. Harnessing new technologies has allowed the development of innovative, reliable, and automated inspection techniques that leverage *in situ* visual data collection and facilitates the application of reliable vision-based inspection.

Despite the tremendous efforts that have been made to advance vision-based inspection techniques, there is a significant limitation about the use of visual data — the image scale is unknown. Detecting regions of interest (ROIs) on images, using existing techniques, provides their size and locations relative to other features in the scene, but have no physical meaning. Their physical size and location can be obtained only when the scales of the corresponding images are known, which can be defined as a unit of pixels per unit length (e.g., pixel/mm or pixel/in.). Without knowing the scale, quantitative evaluation of visual features in an image may be impossible.

The simplest way to resolve scale is to capture images by a reference object of known dimension (e.g., pen, crack gauge, or tape measure) [59], as shown in Figure 1.2. Hence, if the inspection area and reference object are present on the same plane, scale may be resolved using a simple pixel-to-length relationship. However, it is impractical to place reference objects on a structure’s surface each time an image is captured. Additionally, the pixel to length correlation process requires that the structure surface be accessible to the inspector, who must place or attach the reference object. A more advanced method to determine scale

is to employ photogrammetry techniques using multiple images [18]; a stereoscopic camera configuration enables measuring physical depths using simple trigonometry. However, since a physical baseline length (intra-axial distance) should be known in advance to calibrate the images, it is not practical to implement this setup in the field, unless pre-built stereo cameras are used. Recently, tape measure applications have been introduced in modern smartphones, which use multiple images captured from planar scenes to enable structure from motion (SfM) or simultaneous localization and mapping (SLAM) algorithms [55]. SfM algorithms require calibration length, which is commonly obtained with the help of additional sensors, such as accelerometers, inertial sensors, or laser sensors. These applications are easy-to-use but its accuracy highly depends on the initial calibration process that is performed on start-up. In summary, existing vision-based methods provide reasonable measurement accuracy but require supplemental processes, in addition to simply collecting images. Although inspectors can obtain measurements using these methods, it is tedious and scales poorly with respect to the increasing number of infrastructure; the additional time and effort required to take a measurement cannot be ignored, especially in field inspections.

It is proposed that it is possible to remove the need to include a reference physical dimension (e.g., known object size, baseline length, or a measurement using extra sensor data) in the image scene by using surface textures. For example, Figure 1.3 shows surfaces of a concrete wall captured from different distances and locations but having similar textures. From examination of Figure 1.3a, it is difficult to get a sense of distance between the camera and the wall, as well as the approximate size of aggregates. However, if Figure 1.3b is examined after examining Figure 1.3a, it is easy to deduce that this image (Figure 1.3b)

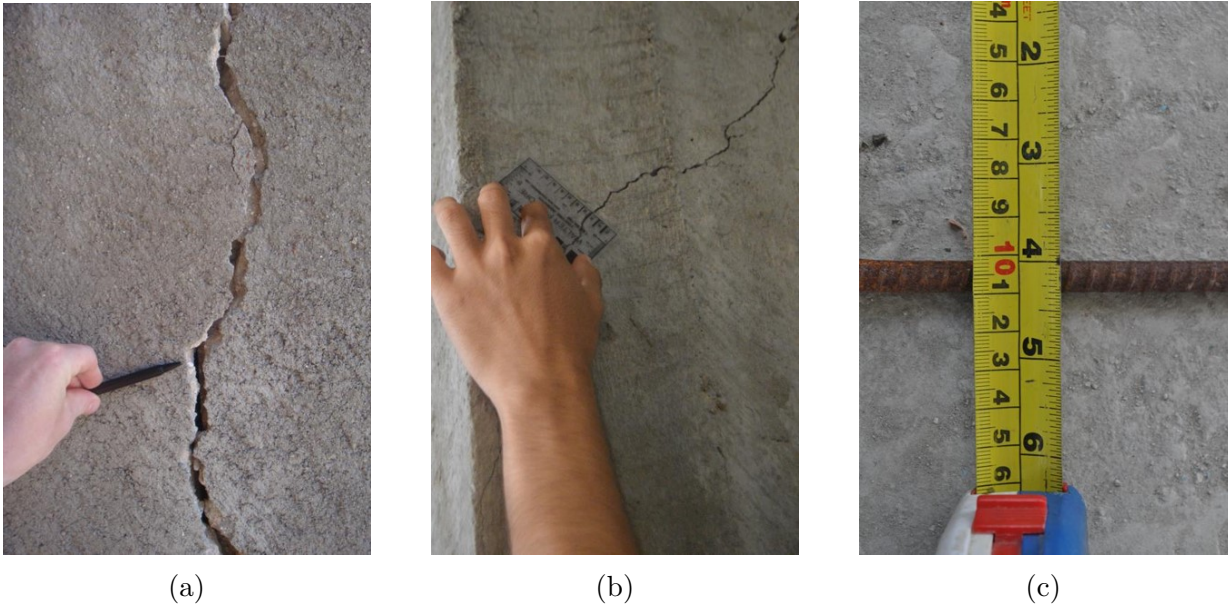


Figure 1.2: Image scale estimation using a (a) pen, (b) crack gauge, and (c) measuring tape. These images contain scenes of damaged components with objects included in the scene and have been collected in real environments by engineers during their field inspection [59]

was captured from a closer distance, although they were captured at different locations. This shows that the surface texture (pattern) made by distributions of different sizes of grains or particles forming the “texture” of a structure can reveal the scale information. Specifically, the textured pattern on the image plane could determine the standoff distances of the images and the size of visual contents. The previous assertion implies the possibility of building a model that can relate the surface texture in an image to the corresponding scale.

In this study, a learning-based scale estimation technique is developed that enables quantitative visual inspection. The estimated scales, defined as the pixels per physical dimension (e.g., pixel/mm or pixel/in.), facilitate the acquisition of physical measurements

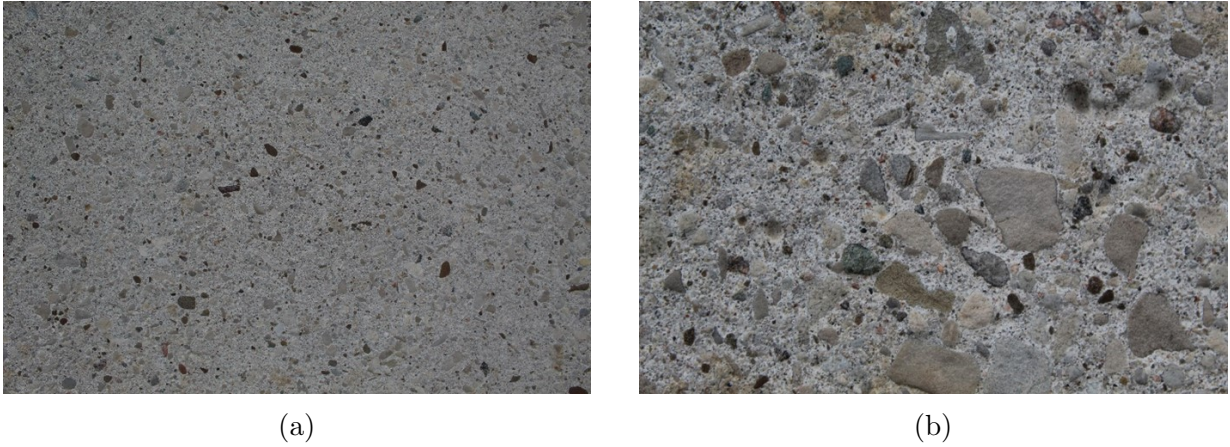


Figure 1.3: Images captured from the same concrete wall but different locations. The image in (b) is captured at a closer distance than the one in (a)

from images. A convolutional neural network (CNN) for regression is trained to correlate the structure’s surface textures captured on images to their corresponding scales. An existing CNN architecture (MobileNetV2) developed for image recognition [48] is modified to estimate a scale (scalar value) from an input image. Custom data augmentation methods and a loss layer are incorporated to improve accuracy. In addition, a marker-based automated training data generation method is developed to streamline the model training process. As an illustration of the robustness of the proposed technique, experimental studies are conducted using the images collected from three different settings: pedestrian bridge, building wall, and asphalt pavement. Afterwards, additional images were collected under different operational conditions to demonstrate the feasibility of the method in real-world inspection environments, where different cameras and optical zoom may be used.

The key innovation of this research is enabling quantitative visual inspection through use of CNNs augmented for regression, which estimates image scale directly from a struc-

ture’s texture. This technique can be easily integrated into other existing vision-based inspection techniques. Despite recent progress in feature extraction (e.g., damage detection, component detection) from visual data, the actual implementations of these technologies remain limited. This is because, without quantitative techniques, they often fail to deliver an end-to-end solution and still rely heavily on inspector judgement for decision making. Solely localizing and identifying ROIs from visual data may not provide significant benefits, especially when quantitative evaluations (e.g., size or length) are required, as outlined in typical inspection manuals [38, 23, 2], or to use modern damage-based structural assessment techniques that rely on damage measurements as model input [61]. Thus, the technique proposed in this study can add an important functionality to existing vision-based techniques, making them more practical and useful by facilitating the quantitative evaluation of any ROIs captured on images.

Chapter 2

Literature Review

Vision-based inspection techniques can be categorized into one of three types: detection, localization, and quantification. Detection techniques focus on identification of ROIs and is usually the most well explored subtopic of the three categories. Localization techniques usually involve mapping extracted features from a detection technique or sensed information from a sensor to a meaningful context where the data is organized in an easy-to-understand manner. Lastly, quantification techniques are used to extract numerical damage properties of civil structures and/or its components from one or more high-dimensional and complex data sets. For a comprehensive overview of vision-based damage detection, localization, and quantification techniques, the following surveys can be consulted [25, 51, 57].

2.1 Vision-based inspection

In recent years, numerous vision-based inspection techniques have been developed for application on civil structures. Most researchers and engineers, working in inspection industries, have focused on advancing data collection and feature extraction processes to enable detection, localization, and quantification of ROIs with increased robustness, efficiency, and/or accuracy. Exploration of literature for vision-based inspection is sub-divided into three of the previously mentioned categories.

2.1.1 Detection

Current inspection practices are highly manual and dependent on the experience of the inspector. This leads to long inspection times, and potential for inconsistent assessments of a structure. Researchers have proposed novel feature extraction methods to automatically detect and assess ROIs from visual data for specific visual inspection tasks outlined in the manual. [38] (e.g., damage detection and classification, structural components, geometric changes) These approaches show promise of improvements to the status quo by providing a methodology for rapid, accurate, and consistent inspection for one or more visual inspection tasks.

One of the most explored topics is crack damage detection. In vision-based inspection, non-deep learning computer vision techniques have been deployed to detect cracks. For example, [30] leverages crack breathing behaviour, a cyclic increase and decrease in crack width under repetitive fatigue loading, to implement an image-based crack segmentation

approach. This approach proposes taking two images of the same scene at different times, and performing a series of computer vision techniques, such as image registration, noise reduction, and binary thresholding to identify cracks. The underlying assumption is that images taken at different times would contain scenes of the same cracks at different magnitude of cyclical loads, which would result in slightly different crack widths between the two images. As a result, performing image registration and noise reduction would reveal changes in crack widths. Thus, this technique can distinguish real cracks from false cracks through crack breathing behaviour.

More recently, many deep learning crack-detection solutions have been proposed. [9] developed a real-time crack segmentation technique by training a custom encoder-decoder structure (called SDDNet) inspired by Segnet [4] on a labelled data set. SDDNet provides pixel-wise binary crack classification. On the other hand, [26] leverages a wall-climbing unmanned aerial system to collect images of building walls at a fixed distance and re-trains the SSDLite-MobileNetV2 [48] CNN network on a crack data set. After training, the network is deployed for real-time bounding box crack detection.

However, it is often difficult to distinguish real cracks from false cracks using only images. To ensure robust detection in the presence of false cracks, [3] combined two visual data types: colour and infrared images, which are collected using a RGB sensor and a laser driver/infrared sensor setup. The patch-based binary crack classifier identifies potential crack regions, while the infrared data helps filter the approach for false cracks, as false cracks have a heating profile that is visibly different from real cracks. Another method of robust crack detection is proposed by [62], which uses a 3D depth data set instead of an image data set to train a CNN segmentation model. By simply switching the visual data

from images to depth, the trained CNN is more robust against false cracks.

Deep learning models have also been applied to identify structural components and damages. For example, [40] implements a configuration of two CNNs for scene-aware bridge component segmentation of images. Given an image containing the scene of a bridge, the technique automatically classifies relevant portions of the image as either column, slab, beam, other structural component, or non-structural. To improve the performance of the component segmentation model, the authors use a CNN-based scene classifier, which predicts the output scenery of an image (e.g., building, greenery, person, pavement, bridges, etc.). By providing the scenery information predicted by the scene classifier to the segmentation model, the technique ensures that the segmentation model makes scene-aware predictions. [56] augments and trains a faster R-CNN [45] model for damage detection of reinforced concrete, which predicts bounding boxes of cracks, spalling, rebar exposure, and buckling for images of reinforced concrete columns. Lastly, to combat the lack of large, labelled image data sets required for training deep learning models in the domain of civil engineering, [14] proposed Structural ImageNet, a large, multipurpose, labelled data set designed to be able to train binary/multi-class models for component type identification, spalling condition assessment, damage level evaluation, and damage type determination. Lastly, [34] developed an automatic image retrieval algorithm to extract multiple views of a residential building of interest from Google maps panoramic images. Given an input image, the algorithm uses the GPS tag in the image metadata to extract an initial panoramic image from Google maps. Then several rectilinear images are generated from the panoramic image, and a residential building detector is used to identify images with a residential building. Next, feature matching is performed between the detected images and

the input image to determine the correct pair(s). Once found, multiple images of the building of interest containing scenes of the building at various viewpoints are automatically extracted from Google maps.

2.1.2 Localization

Access is arguably the main challenge associated with collecting visual data for large-scale civil infrastructure. Thus, remote and automated data collection systems have been proposed to increase the efficiency of inspections. Vision sensors (e.g., RGB and infrared cameras) integrated with mobile and static sensing platforms (e.g., drones, surveillance cameras, and crawling robots) are used to access and capture inspection regions with necessary levels of detail. After data collection, feature extraction of structure-relevant information is performed. However, feature detection is often not enough to enable end-to-end visual inspections, which require more comprehensive analysis, such as identifying the precise location of a defect on a structure. Thus, the detection algorithms are often embedded into a larger framework to use them to perform feature extraction, the results of which is then spatially and/or temporally mapped to provide information in an organized and coherent manner.

In recently proposed techniques, unmanned vehicles are often used to collect one or more sets of sensory data, after which photogrammetry methods are applied to project the collected information onto a map. [21] developed a framework for rapidly generating condition-aware building models. The authors use unmanned aerial vehicles (UAVs) to collect a video of the building overview, which is then processed to generate a 3D mesh

of the building using multi-view stereo techniques and to generate segmentation maps of building relevant classes (e.g., debris, building, cracks, spalling, and exposed rebar) for each video frame using a deep learning segmentation model. After processing, the model predictions are projected onto the 3D-mesh to create the condition-aware 3D building models.

[8] proposes a framework to extract ROIs of a building façade easily and rapidly from multiple viewpoints. To achieve this, images of the building façade collected using an UAV. Then the SfM algorithm [55] is used to generate an orthophoto of the building façade. Then, the orthophoto is used to manually or autonomously specify a ROI, after which images containing the ROI can autonomously be extracted for the image set for inspector assessment. Similarly, [58] also uses video footage from an UAV and SfM to generate a 3D point cloud model of a full-scale highway sign truss with welded connections. Then, image patches showing welded connections are automatically extracted and filtered using a binary CNN classifier to remove occluded welds from the extracted image set.

Like UAVs, unmanned ground vehicles (UGVs) have also been proposed to enable end-to-end structure inspection. The advantage of UGVs in comparison to UAVs is the increased loading capacity, allowing addition of multiple sensors onto one platform. [6] developed a ground robotic bridge inspection platform capable of producing to-scale, high-quality, dense, and colourized 3D point clouds for the underside of concrete bridges. To achieve this, the ground robot has one thermal sensor, one RGB vision sensor, and two lidar sensors on board. The multiple sensors are carefully calibrated prior to use, and SLAM [13] is applied to the collected visual data set to generate the point clouds. Lidar sensors are used to validate the accuracy of the SLAM results, while infrared data is used

to detect delamination on the structure.

2.1.3 Quantification

Quantification of ROIs and structure properties are critical in enabling end-to-end inspection frameworks. One typical reason for requiring ROIs to be resolved to a physical scale is because inspection manuals rely on physical measurements to identify the severity of damage on a structure [38, 23, 2]. However, literature on ROI quantification remains limited, due to the inherent lack of scale information in visual data. Meanwhile, two works, one from section 2.1.1 and another from section 2.1.2 previously introduced also deploy quantification measures. [6] developed a UGV data collection platform capable of generating 3D point clouds using a SfM from visual data. This 3D point cloud is scaled to match physical dimensions using a control point method and a subset approach, which requires measuring a feature on the structure while during the field inspection (e.g., column width). [26] developed a wall-climbing unmanned aerial system that collects visual data along a wall at a fixed distance. A combination of bounding box CNN model and image thresholding is used to segmentate cracks. Since the unmanned aerial system captures images at a fixed distance, the authors simply pre-compute the image scale, which is used to resolve crack measurement to a physical scale. However, if the distance between the vision sensor and the wall changes, the approach requires a different method to resolve crack measurements to a physical scale.

Furthermore, other features, such as bridge displacement measurements and bridge natural frequencies can also be quantified using vision-based approaches. For example,

[22] proposed a vision-based modal survey of civil infrastructure using UAVs. ArUco markers [15] are placed on many sections of the bridge, and an UAV is used to record video footage of sections of the bridge for a duration of time. Typically, such systems typically require recording of video of the entire scene of the bridge span for modal survey, but [22] adopts a divide and conquer strategy, taking several videos of sections of the bridge instead of one video containing the entire bridge. For each video footage, the displacement is measured through the tracking of ArUco markers, which is used to identify the local mode shapes. Then, the results of all the local mode shapes are scaled and combined using a decentralized modal analysis method to create a single mode shape profile for the bridge. Lastly, [33] proposed a dual-camera system to enable long-term displacement measurements of full-scale bridges while mitigating drift that occurs to the ego-motion of the sensor. Ego-motion of the sensor is caused by the self-weight of the camera system and the thermal expansion and contraction under cyclic temperature changes. To compensate for the ego-motion, a second camera is attached to the first camera to track a sub-target, the results of which are used to remove the camera’s ego-motion from the measurements.

2.2 Research Gap

To date, significant progress has been made in vision-based inspection techniques to improve the pace, consistency, reliability, and accuracy of inspection. However, the proposed techniques are not without limitations. Often, limited inspection budgets encourage the use of cost-efficient sensors such as mobile phone cameras over specialized equipment. This is because specialized equipment costs extra capital and requires additional training for op-

eration of the equipment. For example, UAVs, usually require two people for inspection, where one person controls the vehicle, and the other examines the real-time data feed to conduct data collection and preliminary inspection. Thus, it may be more favourable to improve existing inspection practices instead of transitioning to new inspection frameworks. For example, detection algorithms using colour images can be implemented without any change to current inspection practices. However, such vision-based techniques also have a major limitation. Visual data collected during inspection the lack physical scale information, and thus cannot resolve detected features to a physical scale. This work seeks to address this gap. The image scale estimation method proposed in this study enables quantification solely through the use of images. This is achieved by extracting the scale information embedded in the texture of the structure using a learning-based approach. As a result, the approach enables quantification of images containing scenes of ROIs, solely through the information contained in the image itself. Thus, no other specialized equipment other than the camera itself is required for the implementation of this approach.

Chapter 3

Convolutional Neural Networks

3.1 Overview

CNNs have recently mobilized researchers in many communities to pursue computer vision applications because they enable the learning of complex features (both deep and high level) for image recognition by leveraging large-scale databases [31]. CNNs often use one or more convolutional layers linked with weights and pooling layers that work to extract translation, scale, and rotation-tolerant features. CNNs are most successful when training classifiers using a large set of images as a source, along with a large set of parameters. Graphical processing units (GPUs) can also be exploited to implement CNNs to enable a scalable training process. Furthermore, CNN architectures have seen significant improvement, and their accuracy has been greatly increased [50, 20, 10, 48].

It is well known that the major advantage of CNNs for image recognition applications

is that the network is trained to automatically extract and interpret features to predict a class or value corresponding to input images. These features are essentially condensed representations of simple or complex visual patterns, such as corners, blobs, or colour that can be used to indicate what class or value the input image is associated with. In this study, such capabilities are utilized to build the scale estimation model. Since patterns of surface texture can contain image scale information, illustrated in Section 1, CNNs can be trained to extract and interpret texture patterns to predict the corresponding scale for a given image.

The CNN architecture mainly consists of two parts: (i) a base model and (ii) a top layer, as shown in a simple example in Figure 3.1. Common components of a base model are convolutional and pooling layers, which are key in extracting features that uniquely and compactly represent the input images. Then, these features are fed into the top layer to identify the correct label(s) corresponding to the input image. These value(s) can be binary, integer (class), multi-label, or real value(s). At least one fully connected layer with an activation function is placed at the beginning of the top layer to predict the output value(s) from features. The activation and loss function should be designed according to the output type [17, 41].

In the following subsections, common techniques and layers of a typical CNN base model and top layer, as well as those specific to the development of the scale estimation model are described. Detailed descriptions regarding CNNs and their components can be found by consulting the following literature [32].

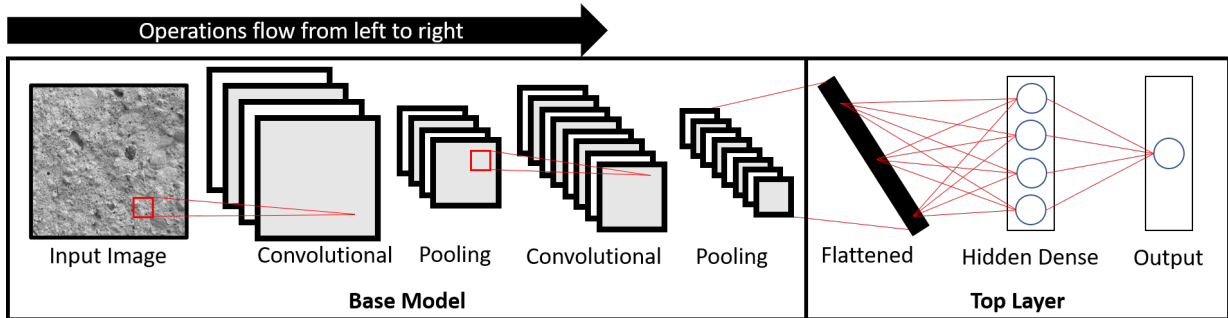


Figure 3.1: An illustration of a conventional CNN model architecture. The CNN model can be described in two parts: the base model and the top layer. The features of an image relevant to the label being predicted are extracted through a series of convolutional and pooling layers in the base layer. The features then interpreted in the top layer through one or more sets of hidden dense layers to output a single prediction. The flattened layer simply is an unravelling of the previous layer’s output vector. Note that only a part of the weights has been shown from the flattened to the hidden dense layer to maintain simplicity

3.2 Convolutional Layer

A convolutional layer (also called Conv 2D in this study) consists of k learn-able kernels (also known as filters) used to extract features to create a feature map, which can contain information pertaining to colours, simple morphologies, or complex morphologies (e.g., edge, corner, or shape of a nose). In implementation, kernel dimensions are three dimensional ($k_m \times k_n \times k_o$), with k_m , k_n , and k_o denoting the kernel’s height, width, and depth parameters. The convolution operation of a kernel on an input matrix (e.g., an image or a feature map) can be seen in Figure 3.2.

The convolution operation can be explained via a rolling window approach. Assuming a stride of 1 and no padding (to be explained later), a window of the same size as the kernel starts in the top-left portion of the input matrix, and a dot product of the values inside

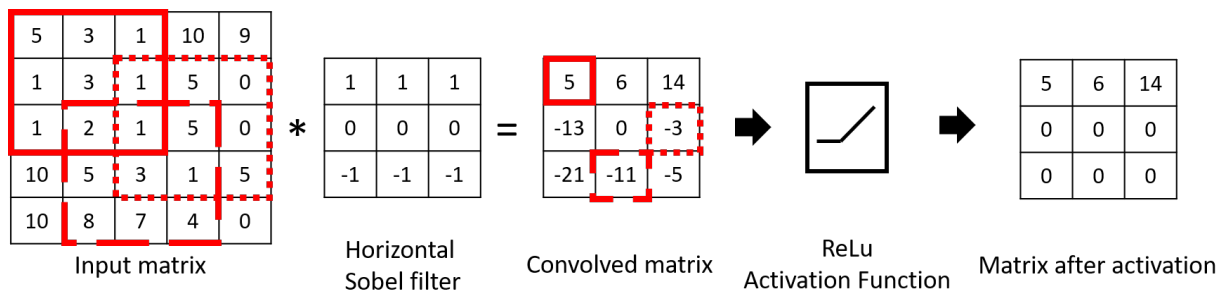


Figure 3.2: A sample convolution operation. The input matrix of size $5 \times 5 \times 1$ is convolved with the $3 \times 3 \times 1$ kernel with stride of 1. For this example, the Sobel filter in the horizontal direction of size $3 \times 3 \times 1$ is used to identify horizontal edge features. Each sub-box (also known as a window) is convolved with the filter to produce the coloured sub-box values in the convolved matrix. The ReLu activation function is then applied to the convolved matrix to yield the output vector of the convolutional layer

the window and the kernel is computed, yielding the value for the upper-left most element of the convolved matrix (a value of 5, in Figure 3.2). Then the window is shifted 1 element to the right, and the dot product is performed again with the kernel to produce the value for the element immediately right of the upper-left most element of the convolved matrix (a value of 6, in Figure 3.2). This is repeated until the window reaches the rightmost edge of the input matrix, after which it is shifted one row down and moved to the left most side of the input matrix. The process is repeated until the window reaches the bottom-right most part of the input matrix.

Important hyperparameters of the convolutional layer is the kernel size, number of kernels (which corresponds to the number of output channels), activation function, stride, and padding. The kernel size simply determines the height and width of the learn-able convolution kernel that is used to extract features from the input matrix. The number of kernels specifies how many individual learn-able kernels will be used, where each kernel

outputs a single channel. Note that since one kernel is used to create one channel of the feature map, the number of kernels is equivalent to the number of output channels (one channel is a single 2D slice of the feature map). The activation function is applied to each element in the feature map to introduce non-linearity, allowing CNNs to learn complex patterns. ReLu [39] has been the most used activation function in convolutional layers. However, Swish [44], a recently proposed activation function has been shown to outperform ReLu and as a result have seen increased usage. The stride parameter controls the number of elements by which the rolling window shifts in the horizontal and vertical direction. For example, given a horizontal stride of 2 and a vertical stride of 3, the rolling window shifts by 2 in the horizontal direction, and 3 in the vertical direction instead of 1 for each iteration, in comparison to the previously discussed example where a horizontal and vertical stride of 1 was used. The padding parameter can be used to control the output dimensions of the feature map. When an input matrix is processed through a convolutional layer, the width and height of the resulting matrix is smaller than the original matrix. In some cases, it is desirable to maintain the width and height of the original matrix after the convolution operation. To achieve this, the original matrix is padded with zeros to match the input height and width. Essentially, the output feature map's height and width can be calculated as,

$$d_o = \frac{d_i - d_k + 2 * d_p}{d_s} + 1$$

where d_o , d_i , d_k , d_p , d_s , is the output, input, kernel, padding, and stride length along an axis (height or width). It is important to note that convolutional layers typically perform

3D convolutions, where a single, learn-able 3D convolution kernel produces one 2D feature map. The depth parameter of the kernel is automatically adjusted to the number of channels of the input matrix. Lastly, there are other hyperparameters which are utilized more on a case-by-case basis, such as the dilation rate and bias.

3.3 Depthwise Separable Convolutional Layer

Efficiency has become a key topic of interest in the implementation of CNNs for problems with resource constraints. The computational requirements of a conventional convolutional layer grow increasingly inefficient as larger channel sizes are used. To improve the efficiency of convolutions, a variation of the convolutional layer called depthwise separable convolutional layer have been proposed, which have proven to perform just as well as conventional convolution layers while using a fraction of the number of parameters [48]. This has led to the rise of a new generation of state-of-the-art, efficient CNNs [10, 48, 53].

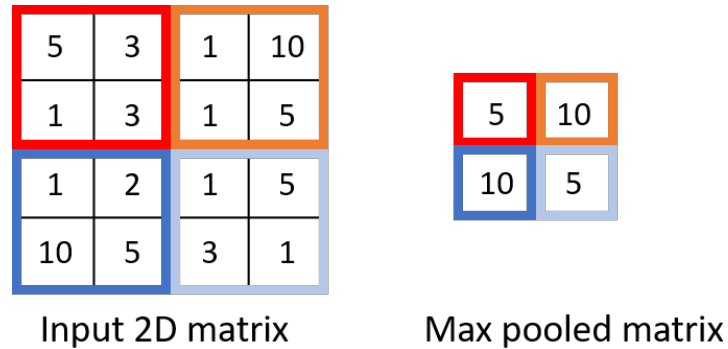
The efficacy of this convolutional layer variation can be illustrated using an example. Given an input matrix of size $5 \times 5 \times 3$ and a desired output of feature map with depth (interchangeable with the term, number of channels) of 64, the conventional convolution layer with kernel of size $3 \times 3 \times 3$ requires $3 \times 3 \times 3 \times 9 = 243$ multiplication operations to produce one channel of the $3 \times 3 \times 64$ feature map. To produce the entire feature map, $243 \times 64 = 15,552$ multiplication operations are required. In addition, each output channel requires an individual kernel, and the number of parameters required are $64 \times 3 \times 3 \times 3 = 1,728$. In comparison, the depthwise separable convolutional layer, instead uses an individual 2D convolution kernel of size 3×3 for each channel (which is a total

of three 2D kernels in this example) to produce an intermediate matrix of size $3 \times 3 \times 3$, which involves 243 multiplication operations. Then a total of 64, $1 \times 1 \times 3$ kernels are applied to the intermediate matrix to produce the $3 \times 3 \times 64$ feature map. In sum, a total of $3 \times 3 \times 3 \times 9 + 3 \times 3 \times 3 \times 64 = 1,971$ multiplication operations are performed and $3 \times 3 \times 3 + 3 \times 64 = 219$ parameters used. In this case, the depthwise separable convolutional layer achieves the same function as a regular convolutional layer while using only 12.7% of the number of parameters and operations. Because of this efficiency, depthwise separable convolutional layers have become quite common in recent CNN architectures [10, 48, 53].

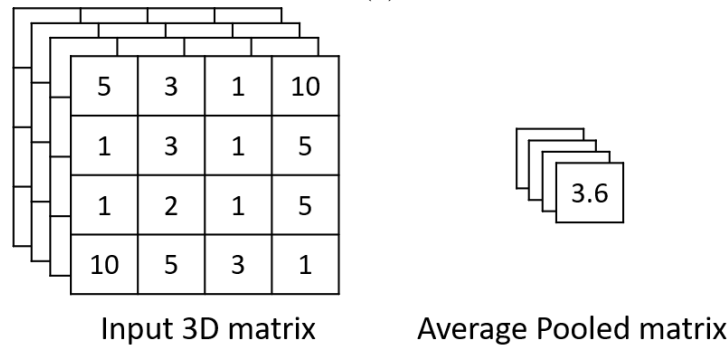
3.4 Pooling Layer

Pooling layers spatially aggregate the input matrix into a smaller, condensed matrix. There are two main benefits to using a pooling layer [49]. First, the model becomes more robust against local translations, meaning even if the image is shifted by a minor amount, the features output from the pooling layer mostly would not change. Second, pooling also acts a form of down sampling. The resulting feature map after a pooling operation is significantly smaller, reducing computational costs for following layers compared to if there was no pooling layer in place. Two types of pooling layers, namely average and max pooling are the most used in a CNN model. An example max pooling operation can be seen in Figure 3.3.

An example max pooling operation can be seen in Figure 3.3a. The pooling operation uses a window (also called a pool) of size $p_m \times p_n$. By default, horizontal and vertical stride lengths are equal to the window's horizontal and vertical lengths. The operation



(a)



(b)

Figure 3.3: Sample pooling operations. In (a), an input matrix of size 4×4 is max pooled using a kernel size of 2×2 to produce a 2×2 matrix. In (b), 2D global average pooling is performed on an input matrix of size $4 \times 4 \times 4$ to produce a $1 \times 1 \times 4$ matrix

starts at the upper-left corner of the input matrix which performs an aggregation (max or average) of the upper-left pool to produce the upper-left corner value of the output matrix. Then, the window shifts according to the set stride value and performs the aggregation on the new pool. This process is repeated in similar fashion with the rolling window of the convolutional layer until the pooling window reaches the bottom right most part of the matrix. Note that the pooling layer differs from convolutional layers in that the pooling is performed spatially in 2D instead of 3D. Thus, the number of channels is fixed and does

not change.

Main hyperparameters are pool size, stride, and padding. Pool size determines the height and width of the aggregation rolling window. Stride and padding parameters are essentially identical to the stride and padding described in section 3.2. However, the key difference is that, in implementation, it is common for the default stride values for pooling layers to be equal to the dimensions of the pool size, while for convolution layers, it is common to set the horizontal and vertical stride as 1, unless otherwise stated.

Lastly, there is a variation of the pooling operation known as global pooling, as shown in Figure 3.3b. Global pooling (assuming 2D global pooling) aggregates across the entire spatial axis (horizontal and vertical) to produce a single value. If the input matrix is a 3D matrix, containing a feature map with depth of C , then the global pooling operation aggregates the spatial components of each channel in the feature map to produce a 3D array of length $1 \times 1 \times C$. Global pooling layers have been commonly used to replace dense layers as they are less prone to overfitting [35]. However, they are also used to down sample the feature map at the end of the base model prior to feeding into dense layers.

3.5 Dense Layer (Fully Connected Layers)

A dense layer is the set of fully connected neurons with activation functions, where individual, learn-able weights are connected from each neuron to all the output values of the preceding layer [47]. Dense layers are typically placed in the top layer and is used to interpret the features extracted from the base model to generate a prediction. Depending

on the complexity of the problem, additional dense layers may be added in succession of one another to increase the learning capacity. Dense layers can be further categorized into two different types. If it is the last layer of the CNN, then it is the *output layer* responsible for generating predictions and has the number of neurons equal to the number of variables that are required to be predicted, with each neuron being responsible for the prediction of a single variable. Otherwise, the dense layer is an intermediary between two other layers and is referred to as a hidden layer, whose output values feed into the next layer. During training, the dense layer learns to interpret the output features of the previous layer through the automatic tuning of its weights. An example operation of a dense layer can be seen in Figure 3.4.

Specifically, a dense layer is composed of one or more neurons with an activation function. Each neuron is connected to all the values of the preceding layer via weights, shown as red lines in Figure 3.4. Each weight has a randomly initialized value which is automatically tuned during model training. The output value of each neuron is calculated as follows. First, a weighted sum between the values of the previous layer with a single neuron's weights is computed. If the bias is nonzero of the corresponding dense layer, it is added to the weighted sum of each neuron as a constant. Afterwards, the activation function is applied to this computed value to introduce non-linearity in the learning process. After the application of the activation function to the value, its result is either passed on to the following layer, or is output as a prediction. The main hyperparameters of the dense layer are d_n , d_{af} , and d_b , which are the number of neurons, activation function, and bias, respectively.

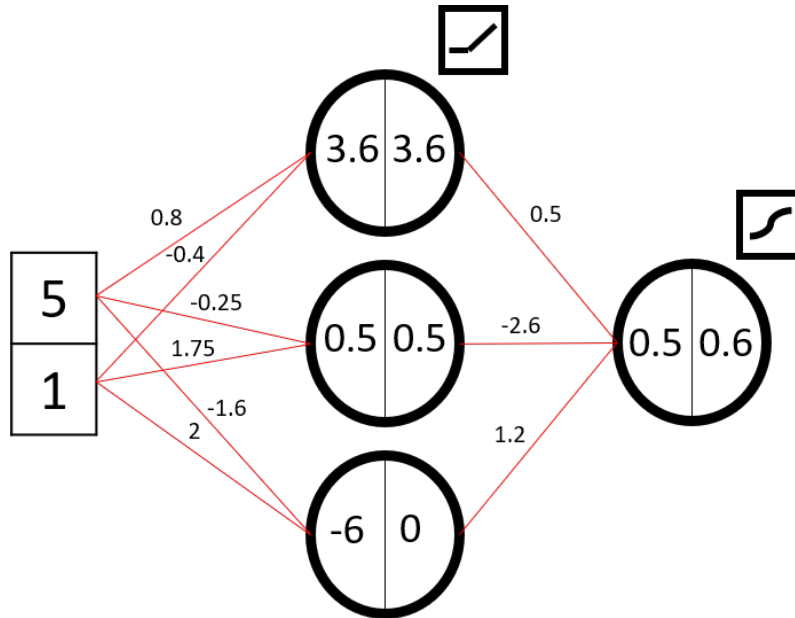


Figure 3.4: A sample dense layer example, using a 1D input matrix of size 2, a 3-neuron hidden dense layer with ReLu activation function and a bias of 0, and a 1-neuron output dense layer with a sigmoid activation function. The neuron consists of two values, where the left value is the weighted sum between the inputs and their corresponding weights, and the right value is the left value after the activation has been applied. The right value of the neuron is then output to the next layer for all the neurons in the dense layer

3.6 Dropout Layer

CNNs deploy a large number of tunable parameters to provide powerful learning capabilities, allowing it to accurately make predictions despite a highly variable and complex input data. However, a major drawback of using large number of weights is overfitting [60], a phenomenon where the model memorizes the training data set, leading to poor generalization. Dropout is a regularization technique that has been proposed to reduce model overfitting which occurs during training [52].

Dropout layers are placed in-between layers, usually after hidden dense, convolutional,

or pooling layers, and is active only during the training phase. The technique simply samples the input values from the previous layer by randomly setting a proportion of the input values to zero, while leaving the remaining proportion of input values unchanged. Afterwards, the augmented input values are passed onto the next layer. The proportion of input values to keep during training is empirically chosen. By randomly setting a proportion of the input values to zero during each iteration of training, dropout layers prevent neurons from significant depending on one or two other neurons (known as co-adaptation), but instead learn to depend on larger number of neurons in a more distributed manner. This change in behaviour leads to better generalization capabilities.

3.7 Batch Normalization Layer

Batch normalization is a technique applied typically after activations (e.g., ReLu, sigmoid, swish) to reduce the effect of internal covariate shift that occurs during model training [24]. Internal covariate shift is defined as the change in layer input distribution over the course of model training. This phenomenon gives rise to training difficulties, such as the saturation of layer input values, resulting in longer training times for model convergence and increased potential for training instability.

Internal covariate shift is reduced through the implementation of batch normalization, where the distribution of each mini batch is normalized to a more appropriate mean and standard deviation during training. Mean and standard deviation is also known as gamma and beta, which are learned parameters with default values of 0 and 1, respectively. As a result of applying this layer, total number of iterations required for model training are

significantly reduced (reported up to 14 times fewer training steps). In addition, model training instability is reduced, and in some cases, the layer acts as a regularizer, like that of the function of dropout layers, allowing for better model generalization.

Specifically, the layer operates differently between the two phases, training, and inference. During training, each mini-batch is standardized (to a mean of 0 and standard deviation of 1) using *mean and standard deviation of the mini-batch* and then rescaled to a distribution with mean and standard deviation of γ and β . During inference, however, the input image or mini batch of images is standardized using the *running mean and standard deviation* computed during model training, and then rescaled to a distribution with mean and standard deviation of γ and β . Overall, batch normalization is now commonly applied in most state-of-the-art CNNs, such as EfficientNet [53].

3.8 Bottleneck Residual Block

A bottleneck residual block is a series of three layers [48], designed with the main idea in mind that information residing in a high dimensional feature map can be embedded into a low dimensional one with minimal loss of information. When the block’s input dimension matches the output dimension, an additive residual connection [20, 48] is added (referred to as inverted residuals), connecting the previous block’s bottleneck layer with the current block’s bottleneck layer. Residual connections enhance model training by reducing the problem of vanishing gradient when implementing deep neural architectures. By using bottleneck residual blocks, significant computational and memory resources are saved, and allow models to reach state-of-the-art accuracies while using minimal number of parameters

and operations.

Specifically, the first layer is a $1 \times 1 \times C$ convolutional layer (where C is the number of channels of the input matrix) with a modified ReLu activation function where the maximum possible output value is set to 6 (known as ReLu6), and output channel size of $C \times t$, where t is a hyperparameter, called the expansion factor. The second layer is a depthwise separable convolutional layer with a 3×3 kernel, with stride s , and a ReLu6 activation function. Note that the depthwise separable convolutional layer is padded to maintain the height and width of the input matrix and has a stride of 1. Lastly, the third layer performs a $1 \times 1 \times C$ convolution layer with a linear activation function, with output channel size C' . Essentially, the first layer performs an expansion from a low-dimensional feature map to a high dimensional feature map to enable the application of nonlinear activation functions, such as ReLu6 while preserving information. The second layer spatially filters the high dimensional feature map, extracting meaningful features, while the third layer maps the filtered high dimensional feature map back to a low dimensional feature map. The third layer uses a linear activation function in comparison to the two preceding layers to better preserve information during the down sampling, is referred to as a linear bottleneck. Together, the three layers enable an efficient method of feature extraction.

The hyperparameters for designing a bottleneck residual block is t , C' , n , and s , which is the expansion factor (determining by how much to expand the low dimensional feature map), number of output channels at the end of the set of repeated blocks, number of times the block is repeated, and the stride for the first block in the set of repeated blocks (following repeating blocks have a stride of 1).

Chapter 4

Proposed Approach

4.1 Overview

The objective of the proposed scale estimation technique is to enable quantitative visual inspection using a single collected image, without relying on extra sensor data. The basic premise for this approach is that images of a unique surface texture contain scale information. A CNN-based regression model is trained to extract the scales from surface textures. If users take images including surface textures, which are similar to the one for training the model, the scales of the images can be estimated, and they can measure the size of the inspection region on the collected images. Moreover, if existing vision-based inspection algorithms that automatically detect areas of interest are integrated into this process, a fully automated procedure aimed toward damage detection, localization, and quantification can be developed and employed using only images.

Figure 4.1 presents an overview of the proposed technique. In general, once users collect an image of the target region for inspection (hereafter, TRI) in Step 1 (note that the TRI is concrete cracking damage shown in Figure 4.1), they can quantitatively evaluate the TRI in Step 5. Steps 2–5 are conducted automatically if the users complete training of the scale estimator and ROI detector in advance (or conducted in a semiautomatic manner if ROI(s) is manually detected in Step 2).

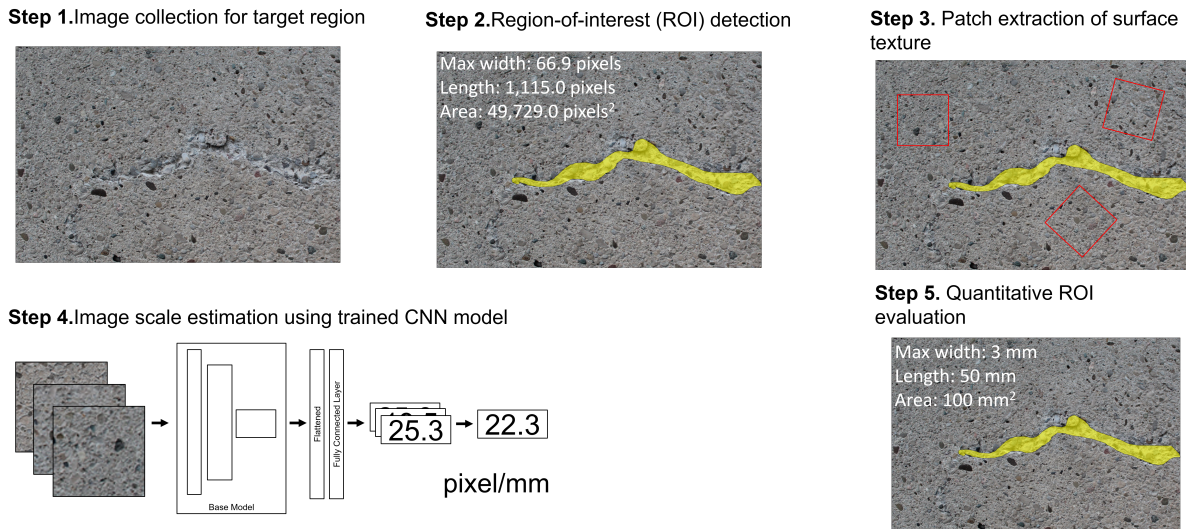


Figure 4.1: Overview of the proposed scale estimation technique

In Step 1, users collect the images of TRIs. The images should be captured reasonably parallel to the surface where TRIs are present and contain surface textures. These background textures are utilized for estimating the scale of the corresponding images. Image collection at Step 1 is the only manual work that users are required to conduct if vision-based ROI detectors are incorporated in the process via Step 2. Next, an ROI (concrete cracking in the case of Figure 4.1) detection algorithm is applied to the image in Step 2. The goal for ROI detection is to automatically quantify the ROI (e.g., crack length, size,

width) using the scale estimated in Step 5 as well as to facilitate the extraction of background textures that are not overlapped with ROIs. When users do not have automated ROI detectors, they can manually provide the location of ROIs on the image. For example, in Figure 4.1, a user could manually draw the boundary of the crack outline. In Step 3, patches for surface texture are randomly extracted from the background (non-ROI region) of each collected image. All patches are square and of the same fixed size. Large patches are extracted from the image background so that each patch is not susceptible to local variations on the texture. In Step 4, the scale of each image collected is estimated using the patches extracted from its background. The CNN architecture originally developed for general image classification purposes is modified and improved so that the network estimates an accurate numeric scale value from the input image. In actual implementation, Steps 2 to 4 are repeated multiple times so that a precise scale value may be derived through averaging. Finally, the actual size of TRI can be measured using the corresponding ROI on the image and its scale. The properties of an ROI represented by pixels can be converted to its physical dimension by simply dividing its image scale in Step 5. The proposed technique enables making a measurement on the entire region captured by the image.

Before illustrating the technical details of the proposed techniques, the assumptions made in the proposed technique are stated and explained.

- *Surface textures are prominent and unique across the structure being inspected.* This assumption becomes the basis that the scales of images can be estimated with one-time training at the beginning and used for multiple areas on the structure in the

future. For instance, in the case of concrete bridges, it is reasonable that surface textures would be similar because they are the product of the concrete mixtures that are largely confined to similar details for similar element types (e.g., concrete girders, pile caps), within specific geographic regions. If there is more than one unique texture presented in a single structure (e.g., several different textured concrete façades on a single structure), multiple estimators may be needed. In the case where there is no unique surface texture (e.g., a painted steel girder, or smoothed concrete surface) or new textures are introduced (e.g., due to partial retrofit or extensive concrete efflorescence damage), the proposed technique is not able to estimate the scales of the images collected.

- *Degree of self-similarity of the surface texture is low or negligible.* For highly self-similar textures, there is little to no visual difference between an image taken close to the surface and an image taken far away from a surface, at which it will be difficult for the estimator to determine scale. For civil structures fulfilling the first assumption, it is reasonable to assume that the texture's self-similarity is low or negligible. Consider the example of a concrete surface with visible aggregates. Such surface texture can have areas where the size distribution of aggregates minimally overlaps with other image scales. Images taken at similar scales (e.g., 30 mm/pixel, 35 mm/pixel, and 40 mm/pixel) can have minor visual differences for texture patches, which the estimator has difficulty picking up, but a scale of 200 mm/pixel has significant visual differences, which the estimator can learn. This is explained by the qualitative differences of images taken close-up versus far away, where the detail of the aggregate binder's texture is prominent in images taken closer to the image, as compared to it looking

less prominent as the distance from the camera increases. These visual cues allow the estimator to make relatively accurate predictions.

- *The surface where TRIs are located is reasonably flat, and the images are captured parallel to the surface.* The model is designed to estimate a single scale per image. By maintaining a single constant distance between an image plane and the surface, the entire region of the image has a single identical scale.
- *Images are captured at standoff distances ranging from 0.5 to 2.5 m.* This range is in-line with the recommended one (to two) arm’s length inspection distance noted in existing guidelines [38]. Empirically, the images captured within this range contain both TRI and background surface textures in most cases. If the TRIs are located far from the users, zoom can be used enlarge the TRIs to emulate the image being captured at a closer distance. In Section 5, the technique is experimentally demonstrated to work on optically zoomed images.
- *The images are sharp.* Out-of-focus or motion-blur causes a collection of blurry images. It is not recommended to use a high-ISO under low-light conditions because it may generate unwanted speckle noise on the images. These effects on images negatively influence the ability to estimate scale as a result of degrading and corrupting texture details. Basically, taking images under good lighting reduces the risk of motion blur and speckle noise, allowing fast shutter speed and low-ISO setup.

4.2 Image regression using CNNs

Since its inception, CNNs have seen significant progress in the development of CNN architecture and training algorithms and have been utilized in a broad range of applications, such as scene classification, object detection, and image-to-image translation [17]. As such, there are many choices of hyperparameters in network structure and training algorithms appropriate to each application. In implementation, instead of developing a CNN model from scratch, an appropriate CNN base model is selected from one of many state-of-the-art architectures in literature and is augmented according to the required specifications of a given project. Typical augmentations involve implementing a custom top layer or augmenting the pre-existing one, changing the number and types of layers, and/or changing the output layer’s activation function, and using a loss function designed for regression, in the case of this study.

There are many state-of-the-art CNN architectures for the base model, such as VGGNet [50], ResNet [20], Xception [10], and MobileNetV2 [48]. In this study, MobileNetV2 was implemented. MobileNetV2 makes use of a novel inverted residual structure with linear bottleneck layers in its architecture. This model achieves a competitive accuracy in the ImageNet competition while using 10 to 100 times lower number of parameters than previous well-performing CNN models, such as ResNet or VGGNet. This can dramatically reduce computation for training and validating the model. In this application, the complexity of the data set is expected to be lower than the existing benchmark data sets such as ImageNet because the proposed procedure requires texture patterns on images. Thus, in this case, MobileNetV2 is well-suited for the application domain.

The CNN model in this study learns to predict scale through an optimization process and requires training prior to use. First, a batch of patches, in random sequence, are selected from the training set. The model predicts a scale for each patch in the batch. After initial estimation, a loss function is used to assess model performance and measure the error in the model’s predictions. The error is used to calculate correction values for the weight parameters in the model. The correction values are then backpropagated into the model to update its weight parameters, resulting in a decrease in error. Then, a new set of patches are selected to form a batch to train the model. This process is repeated until all patches in the training set have been used to train the model and is referred to as a single epoch. The model is then trained for a predefined set of epochs, until a satisfactory accuracy is achieved, or if the model training accuracy converges. The batch size (e.g., number of patches per batch) is predetermined prior to the start of training.

Next, several data augmentation methods were designed for pre-processing input images to train a robust and unbiased scale estimation model. Data augmentation is a commonly used method that expands the training set to avoid overfitting problems by modifying the existing input data through label-preserved transformation [31]. It is important to develop an unbiased estimator. To do so, first, colour scale images are converted to grayscale. Regardless of image scales, the colours of the structure surface are easily affected by weather conditions, degradation, or lighting directions. This causes unwanted bias in the model. Thus, colour information is removed from the image, random brightness changes and channel mean shifts are added to mimic potential lighting variations (e.g., glare, cloudy, camera flash, or urban lighting), affecting the grayscale images. Second, fixed-size patches are extracted from random locations of the surface textures on the original images. The original

images, to be used for training, are captured from the target structure after a marker is attached to the surface. This marker automatically provides a scale of each image, which is explained in Section 4.3. Surface textures are rotational invariant, meaning that the scale information does not change due to rotation of the input texture patch. For preparing input texture patches, several rotated square patches are extracted from non-marker regions of the original images (see Figure 5.5). Third, further augmentation is done during the training phase to reasonably impose perturbation of the input training patches, which allows the model insensitive to the variations on actual testing data. Vertical and horizontal flips were added as they simply mirror the texture, which increases variation in the input patches. Note that rotational and horizontal shift augmentations are not included here because randomly cropping from raw images already simulates these augmentations.

Afterwards, MobileNetV2's existing top layer, is designed to support regression. The sigmoid activation function is replaced in existing MobileNetV2 with a ReLu activation function to produce a single positive scalar value to be estimated as image scales. Then, a loss function is selected to evaluate the predicted scales. The loss function is defined as the error between the predicted and actual scale for each patch. During training, the loss function is used to penalize the model's wrong predictions. The correction values are calculated from the model's errors and then backpropagated to update the model's parameters. Choosing the right loss function is crucial to building a robust model. For this application, a loss function must work for regression, and penalize errors uniformly, meaning both small and large values must be penalized proportionally to their value. Thus, a mean absolute percentage error (hereafter, MAPE) function is chosen for this application. There are several other loss functions that are used for regression, such as mean absolute

error, mean squared error, or mean squared logarithmic error. The key advantage of MAPE in comparison to other loss functions is that MAPE weighs loss equally across entire range of possible values. This is not the case for many loss functions that can be considered as biased, either penalizing smaller values more than larger values and vice versa. For example, if mean squared error is used, a predicted value of 11 and an actual value of 10 will produce a loss of 1. But for a predicted value of 55 and an actual value of 50, a loss value of 25 is produced. In both situations, the errors are equally 10% of its actual value but they are not equally weighted. Thus, for mean squared error, it weighs loss toward larger values, adding bias. For the proposed scale estimation model, it is desirable to be able to uniformly penalize loss across entire range of values. Thus, MAPE is an ideal loss function for this application. MAPE is a loss function that expresses error as a percentage metric. MAPE is calculated between the predicted and actual image scale for a given set of patches and is defined as,

$$MAPE = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{y_i} * 100\% \quad (4.1)$$

where y_i and \hat{y}_i is the actual and predicted image scale for a given crop, respectively, and i is for a given patch in a batch containing n patches.

Existing class imbalances within each data set are identified and mitigated via an oversampling approach. Class imbalance is used to describe any data set with an uneven distribution of its labels. Without any measures, significant bias can be added to the model due to class imbalances. For example, training on an imbalanced binary data set with 1 true label for every 100 false label results in the model learning more from false labels

than true labels. Such situations often yield models that predict the false label with high degree of accuracy (e.g., near 100%) while the true label is predicted with a low degree of accuracy (e.g., less than 20%). This is because the model is more tended to predict samples as negative predictions, even when a sample has a true label. The degree of bias introduced by data sets varies depending on the degree of class imbalance. While it is ideal to use a perfectly balanced data set to train a model. However, such data sets rarely exist. To mitigate class imbalances, several methods, such as weighted loss or oversampling [19, 27] have been proposed. For this study, number of patches extracted from scene-level texture images have been adjusted to create a data set with an approximately balanced distribution of scales. Number of patches extracted from each scene image has been calculated using a histogram of binned scales, extracting more patches from images belonging to a scale bin with low number of counts, while extracting less patches from images belonging to a scale bin with high number of counts. Essentially, by equalizing the distribution of counts for the training data set, the chance for the loss distribution to be imbalanced significantly decreases.

Lastly, for validating the trained scale estimation model, multiple rotated or non-rotated texture patches are randomly extracted from each test image (50 in this study). Afterwards, the model predicts the scales of all patches and the final single scale belonging to the test image is then aggregated to minimize the variations of the scale estimations due to local texture noise or bias. For this experiment, mean and median functions have been tested for aggregating scales estimated from patches and the results are compared in Section 6.

4.3 Automated ground-truth training data generation

CNNs have been successful in many applications by training robust features extracted from a massive collection of labelled data sets, such as COCO or ImageNet benchmark data set [11, 36]. In training CNNs, the accuracy is highly dependent on the quality and quantity of the images. In general, collecting and labelling a large number of images is far more time-consuming than building and validating models using CNNs. For the training of the proposed scale estimator, many images of surface texture with known scales are prepared in advance. The problem is that it would take a considerable amount of time to manually measure the scale of each collected image.

To remedy this issue, a simple and efficient way of preparing a ground-truth training database from the collected images was devised such that surface texture images and their corresponding scales can be automatically generated. A key idea is to use a fiducial marker with a known dimension and a detection algorithm. During data collection, the marker is included in the image scene by placing it on the structure’s surface prior to taking a picture. Afterwards, a marker detection algorithm is run to detect the marker’s outer boundaries in the scene. Since the dimension of the marker is known, the scale can be computed from the ratio of the pixel size of the detected boundary on the image to its physical size. Any fiducial marker can be used for this application permitting that a clear boundary of the marker can be obtained from the image. In this research, the ArUco marker and its detection algorithm implemented in an OpenCV library were used [16, 46].

ArUco markers are square planar fiducial markers (shown in Figure 4.2) with a black outer border and inner matrix of $N \times N$ squares, called bits (black has a value of 0, and

white has a value of 1) [15]. The size of the inner matrix (N) typical ranges from 4 to 7. The squares in the 1st, 3rd, and 5th columns are parity bits, and the squares in the remaining columns contain data bits. The parity bits are used to ensure that the data bits have been detected correctly, which is used to identify the specific identifier for the ArUco marker from a dictionary containing several markers. The markers are automatically detected using a sequence of computer vision techniques [46]: image resizing, adaptive image thresholding, contour extraction, contour filtering, marker corner approximation, and binarization. Note that at a given resolution, the algorithm looks for a marker of fixed pixel size - hence the detection algorithm is repeated for various image sizes, and the results compiled.

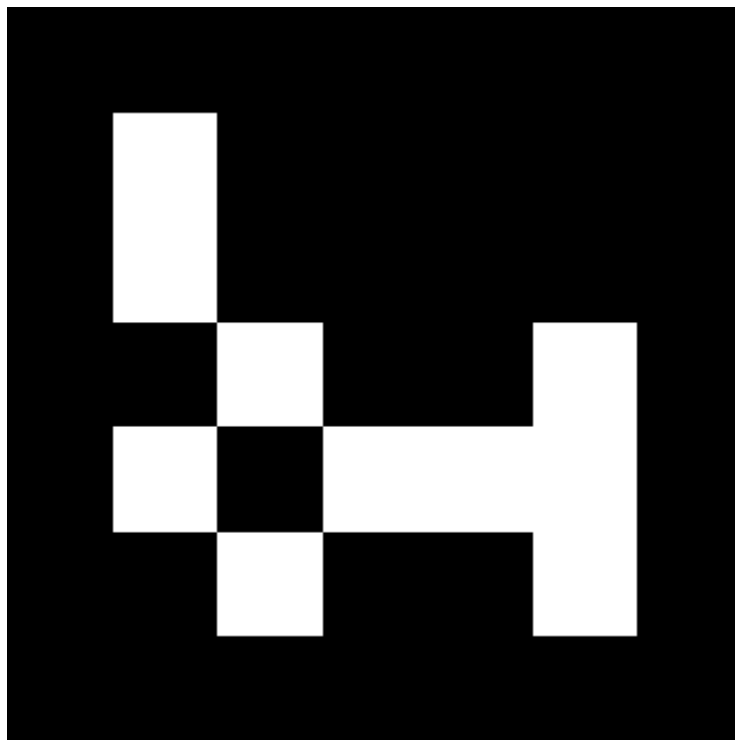


Figure 4.2: A 5 x 5 Aruco fiducial marker used for this study.

Images of scenes containing ArUco markers are shown in Figures 5.3 and 5.4 in Section 5. By utilizing the automated marker detection algorithm, the manual work required for building the ground-truth database is limited to collecting images of surface textures with the marker in the scene, as the labelling process is fully automated through automated marker detection and scale extraction.

Chapter 5

Experimental Design

The experiment design presented in this section was used to demonstrate the following aspects of the technique: (i) The relationship between images of surface textures and their scales can be established using CNNs; (ii) the technique works on various structures, each having different surface textures; and (iii) the performance of the technique can be used under different image collection conditions including the use of the camera different from the one used for training and the use of a zoom function to capture the remote region of more than one or two arm's length. All images captured for experimental validation include the marker to make a comparison between estimated scales of images with their actual scales computed from the marker detection proposed in Section 4.3. Of course, the images used in actual implementation are intended to be captured without markers, as seen in Figure 4.1. The marker presented on testing images in this experiment is to evaluate the accuracy of the scale estimation model developed.

To enable the scale estimation technique shown in Figure 4.1, a robust scale estimator must be trained. Training such models requires a labelled data set containing texture patches with known image scales. In this study, a rapid data labelling methodology is proposed, as shown in Figure 5.1. In step 1, multiple images of the structure texture are taken for each scene. Note that a fiducial marker is included in the image, which is used to automatically generate the training and testing data set. Using the marker’s known physical dimension (e.g., in mm or cm) as a reference, the pixel dimensions of the marker can be determined to calculate the image scale. The marker’s pixel dimensions are estimated using a ArUco marker detection algorithm [46] to calculate the image scale. Following marker detection, texture patches are randomly extracted from the scene (while avoiding the detected marker). Step 2 is repeated for each image in the collected data set, and once the scale calculation and patch extraction are complete for all images, the extracted patches with their corresponding scales form the texture-scale data set. In step 3, the texture-scale data set is split into training and testing data sets, where the former is used to train the CNN model, and the latter to validate the performance. Note that the split is performed on a scene-level (meaning that patches pertaining to one scene are not distributed in both training and testing data sets but are only in one or the other). This is to prevent data leakage, which is to ensure that the model has not seen any of the texture contained in the testing data set during the training phase. Details of each step are discussed in the following subsections.

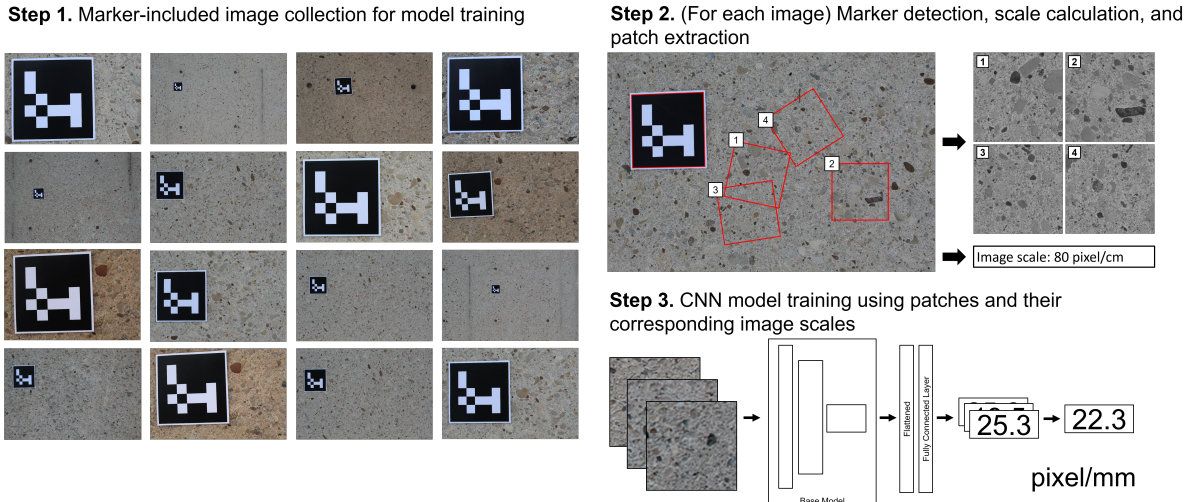


Figure 5.1: Overview of data collection and training data set generation methodology

5.1 Data collection

To validate the proposed technique, images are collected from three different structures having different surface textures. Three different structures (sites) were selected on the main campus of the University of Waterloo, Ontario, Canada: pedestrian bridge (PED), building wall (BW), and asphalt pavement (ASH). Figure 5.2 shows the overall view of each selected structure.

A set of images was collected from many different scenes at each structure. Here, the scenes indicate the surfaces in different locations on the same structure. Once the marker is attached to each scene, multiple images are captured by varying the distance between the camera and the scene. The distance is randomly chosen roughly from 0.5 to 2.5 m, which is the typical distance required for inspectors to perform a close-up inspection [38]. The entire area of the marker is fully included in each image. The images were collected



(a)

(b)



(c)

Figure 5.2: Overviews of test structures: (a) pedestrian bridge (PED), (b) building wall (BW), and (c) asphalt pavement (ASH)

near parallel to the scene during the daytime with good lighting conditions. The sample images collected from each structure are shown in Figure 5.3. In each row, two original images captured at close (left) and far (right) standoff distances are provided, and the square patch on the right of each original image is a magnified area corresponding to a box (in red) on the sample image. This box becomes a potential input for the scale estimation model.

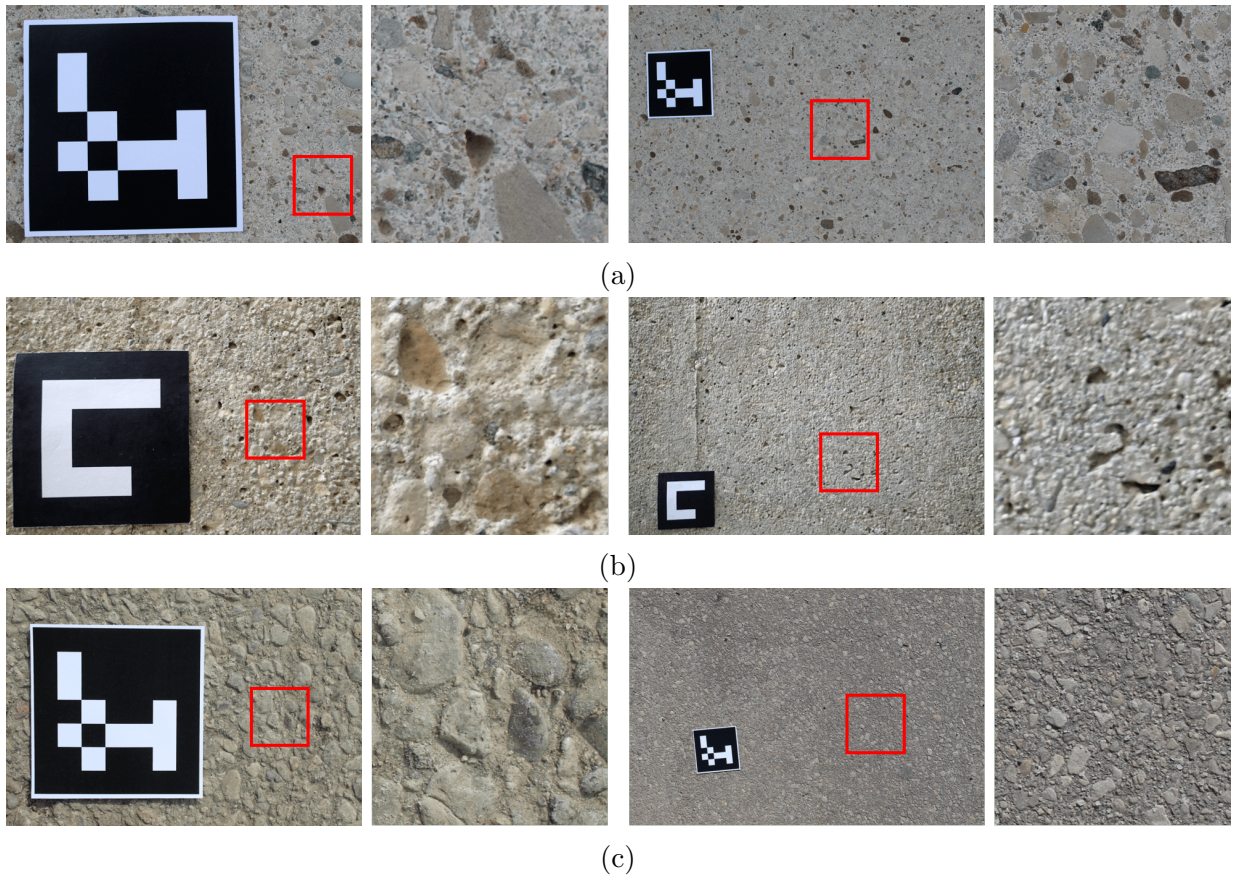


Figure 5.3: Sample images collected from (a) PED, (b) BW, and (c) ASH. Note that the original images with a marker on the right and left are captured from close and far distance, respectively. The square patch next to the image is a magnified area corresponding to a box on the original image

The collected images are split into training and testing sets by scenes. The training set is assumed to be the set of the images that inspectors (or users) collect initially with a marker from the structure being inspected to train the scale estimation network. The testing set is assumed to be the set of images that are used for visual inspection. Rather than randomly splitting all collected images, the images were randomly split by scenes, because the scenes from the testing set should not be included in the training set for

accurate performance evaluation. Likewise, inspectors in a real-world setting would likely capture images from the new scenes that may not be included in the training set. In Table 5.1, there are a number of images collected from each structure. Table 5.1 shows the total number of scenes and corresponding images collected from each structure. Around 20% of the scenes are assigned as the testing set in each data set. For training the model, a Canon 90D camera with a fixed focal length was used to collect all images for PED and ASH, and an Olympus TG-2 camera was used to collect the images for BW. Their resolutions are $5,184 \times 3,456$ and $3,968 \times 2,976$ pixels, respectively. The marker detection algorithm introduced in Section 4.3 could detect rotated markers on images as well; however, the images were collected in a way that the bottom side of the marker was near horizontal.

Structure	Total number of scenes (training/testing)	Total number of images (training/testing)
PED	22 (18/4)	191 (154/37)
BW	14 (12/2)	434 (352/82)
ASH	21 (17/4)	182 (149/33)

Table 5.1: Overview of data sets collected at different structures

In actual implementation, it is likely that inspectors may bring a different camera on site. Also, inspectors often capture inspection images in a remote distance using a zoom function. Thus, further experiment was conducted to validate the method under different operating conditions: different camera (DIFF) and focal length (ZOOM). First, 192 images from 22 different scenes are captured from PED using a different camera, Nikon Coolpix S9900 with image resolution $4,608 \times 3,456$. Second, 298 images from 23 different scenes are captured from PED by varying the focal length of Canon 90D using optical zoom. Sample images are shown in Figure 5.4. In Figure 5.4a, in comparison to the original PED data,

there are some colour differences when taken with a different camera, but no significant texture changes.

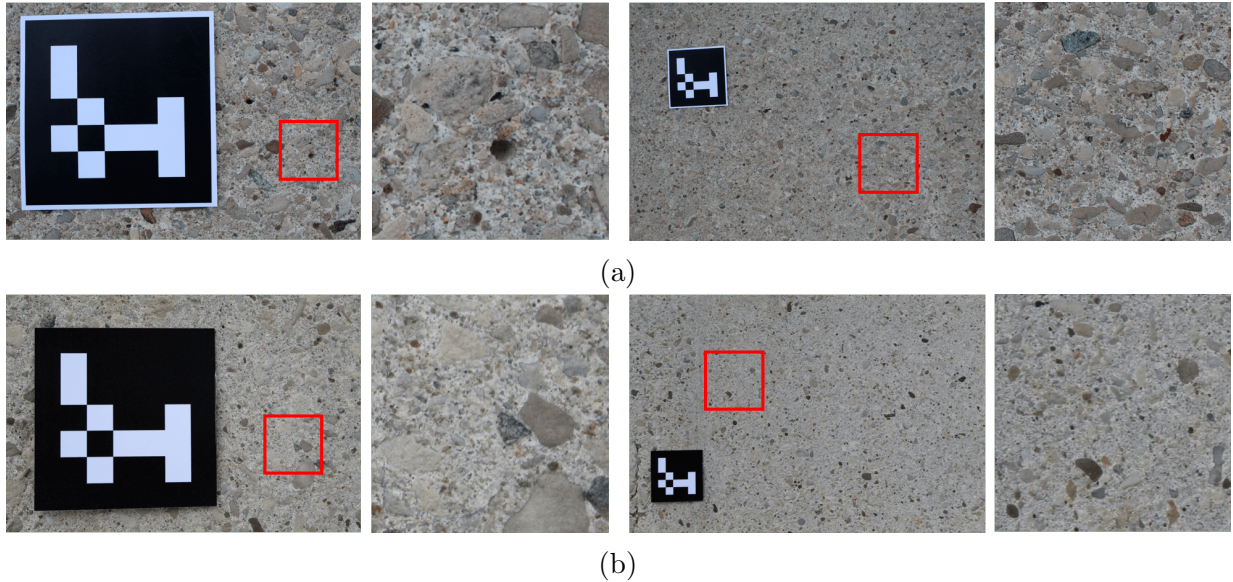


Figure 5.4: Sample images collected from PED using (a) different camera and (b) different focal lengths. Like Figure 5.3, two images on the left and right are captured from different distances and a magnified area corresponding to a box on each image is on the right

After collecting images from each structure, the texture database for training is generated. This can be done by leveraging the known dimensions of the marker on each image and the ArUco marker detection algorithm in OpenCV [16, 46]. Given an image with a marker, the marker corners are detected. Then, the pixel lengths of the marker sides are calculated from the detected corners. Ideally, pixel lengths for all four sides should be identical; but, for actual images, there is a discrepancy between the lengths of the sides due to unwanted perspective distortion, induced by unparallel images to the structure surface or inconsistent gaps between the marker and the surface. Thus, to account for this, the four side lengths are simply averaged to minimize errors on the scale computation.

Once the scale of each image is computed from the marker, texture patches for training are randomly extracted from non-marker areas on the images. Figure 5.5 shows the sample patches extracted from the original images. Rotated patches such as the one labelled as (1) in Figure 5.5 are transformed into the non-rotated square one to make the input form of the scale estimation model.

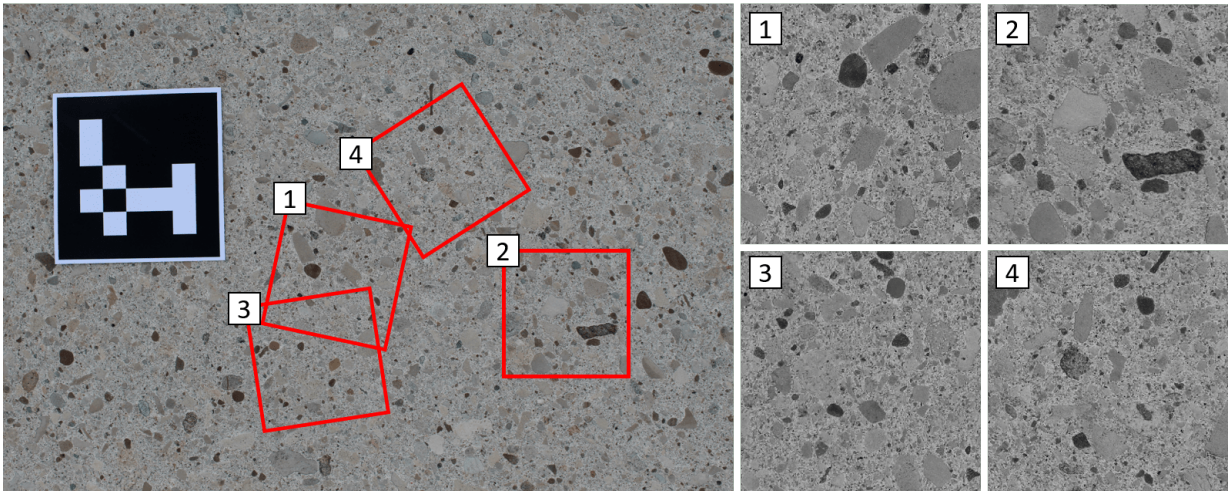


Figure 5.5: Sample texture patches from an original image: Patches are randomly extracted (shown as red squares) from non-marker areas

5.2 Model training

Four tests were conducted to evaluate the performance of the proposed model. To note, the training set was further split into two sets with ratio 80% to 20%, where 80% of the set was used to train the model, and 20% as the validation set. First, three different patch sizes were tested to demonstrate that sufficient texture information was acquired to permit accurate predictions. Prediction results were compared when square patch sizes

of 100×100 , 350×350 , and 850×850 from the PED data set are used as an input. Second, scale estimation models were developed and tested using three data sets (PED, BW, and ASH). This test demonstrated that the technique can learn various textures to associate with their scales. Third, the transferability of the technique to different cameras without further training was explored. The model trained on the PED data set collected using Canon DSLR 90 is implemented to a new PED data set collected using DIFF, and the accuracy of the scale estimation is evaluated. Lastly, the effect of changing the focal length on the model performance is examined. The model trained on the PED data set collected without using a zoom function is applied to a new PED data set of which images are captured with different focal lengths. Note that since it is demonstrated that the use of the larger patch outperforms the ones of smaller patches in the first test, the patch size of 850×850 was used from the second test onward.

In this experiment, a light-weight CNN model called MobileNetV2 [48] is utilized. The base model and top layer of the network architecture can be seen in Table 5.2. This network has an initial 2D convolution layer with an output of 32 channels followed by seven configurations of bottleneck residual blocks with different number of output channels, number of repeating layers, and stride. Then, a 2D convolution layer with a 1×1 kernel and an output of 1,280 channels is applied, followed by a global 2D average pooling layer. The top layer of MobileNetV2 was configured with a 1280-neuron dense layer, which is connected to a single-neuron dense layer output with the ReLu activation function. MAPE was used as the loss function. A stochastic gradient descent (SGD) optimizer was selected, and its learning rate, momentum, and stochastic decay are 1×10^{-4} , 0.9, and 0.01, respectively. Patches were randomly extracted from a non-marker region of each training image collected

and re-sized to 299×299 , which is an input size of MobileNetV2. Prior to feeding the texture patches into the model, data augmentations that included random intensity changes with $\pm 20\%$, and mean shifts with ± 50 , and horizontal and vertical flips were added. Then, the image pixel range was normalized to a range between -1 and $+1$ instead of using the mean and standard deviation of the data set. This is because the size of the collected data set was small and, as such, may introduce biases when using different cameras with filter settings that differed from the camera used for this experiment.

Model section	Input	Operation	$k_{m \times n}$	t	c	n	s
Base model	$299 \times 299 \times 1$	Conv 2D	3×3	-	32	1	2
	$150 \times 150 \times 32$	Bottleneck residual	3×3	1	16	1	1
	$150 \times 150 \times 16$	Bottleneck residual	3×3	6	24	2	2
	$75 \times 75 \times 24$	Bottleneck residual	3×3	6	32	3	2
	$38 \times 38 \times 32$	Bottleneck residual	3×3	6	64	4	2
	$19 \times 19 \times 64$	Bottleneck residual	3×3	6	96	3	1
	$19 \times 19 \times 96$	Bottleneck residual	3×3	6	160	3	2
	$10 \times 10 \times 160$	Bottleneck residual	3×3	6	320	1	1
	$10 \times 10 \times 320$	Conv 2D	1×1	-	1280	1	1
	$10 \times 10 \times 1280$	Global 2D average pooling	10×10	-	1280	1	-
Top layer	$1 \times 1 \times 1280$	Hidden Dense	-	-	1280	1	-
	$1 \times 1 \times 1280$	Output Dense	-	-	1	1	-

Table 5.2: Description of the augmented MobileNetV2 used for this study. Each line describes a single layer/block repeated n times, which have the same number of output channels, c . Variables $k_{m \times n}$, t , c , n , and s are kernel height and width, expansion factor, number of output channels, number of times the layer/block is repeated, and stride, respectively. Note that s describes the stride for the first layer/block in a series of repeated layers/blocks, and the stride after the first layer/block is 1 (e.g., for Conv 2D with $s = 2$ and $n = 2$, $s = 2$ for the first Conv 2D, and $s = 1$ for the 2nd Conv 2D). $k_{m \times n}$ for bottleneck residual blocks describes the kernel size for its depthwise convolution layer. The expansion factor is applied to the input channel length.

A workstation having an Intel Core i9-7940 CPU and a GPU, NVIDIA GTX1080Ti

with 11 GB video memory was used for training and testing the model. A batch size of 32 was used to train the model for 250 epochs. From this setup, a single epoch containing 4,440 patches requires 45 seconds, resulting in a total of approximately 3 hours and 8 minutes to train a single model for 250 epochs.

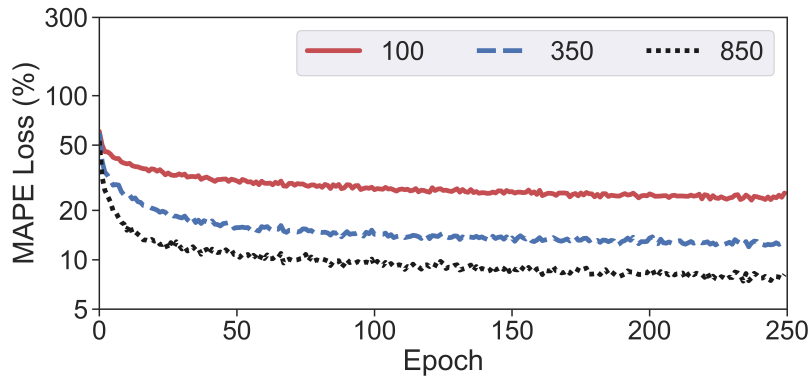
Chapter 6

Results and Discussion

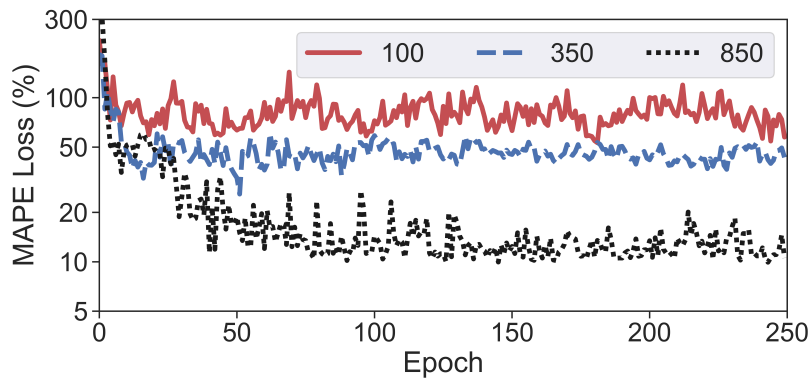
In this section, comparisons of the scale estimation performance depending on the patch sizes are shown. The three different sizes of the patches extracted from the PED data set, 100×100 , 350×350 , and 850×850 pixels, have been used for training and testing. Figure 6.1 shows the model's loss plot in log scale for three different patch sizes—where training curves are plotted in Figure 6.1a and testing curves in Figure 6.1b. From the figure, it is obvious that the loss curve of larger patches converges at a lower MAPE value than those of the smaller ones. This result demonstrates that since larger patches contain more texture information on the physical surface, they are less prone to overfit on local variations in texture or lighting. All three configurations converge roughly after 25 epochs of training, though there is a very slow decrease in loss over the duration of the entire training. The result shows that users should use a larger sized patch to improve accuracy. However, users should also consider the marker size when selecting an appropriate patch size, as the patch size is fixed when extracting patches for all collected images. If the patch size is too

large, every randomly selected square patch would either contain a portion of the marker (making it an invalid patch) or have parts of the patch simply out of bounds. In these cases, no patches would be extracted. Thus, it is important to choose a large enough patch size, but not too large such that there are boundary issues. Furthermore, marker sizes in the images vary depending on the distance of the camera from the surface. For example, when the camera is close to the surface, the size of the marker on the image also increases, leading to a smaller extractable texture region. Thus, a recommendation for determining the patch size is that given the camera resolution used for training, users should determine a reasonably large patch size that can be extracted from the image captured at the closest distance without overlapping with the marker (which was around 850×850 pixels in this study).

Using the patch size of 850×850 pixels, the scale prediction performance was evaluated. To directly observe the prediction performance, an Actual-to-Predicted scale scatter (AtP) plot was drawn to compare true and predicted scale estimations. The AtP plot for the PED training and testing data set is shown in Figures 6.2 and 6.3, respectively. The red dotted line is a true prediction line, which means that the model predicts the scale correctly as the estimated values are close to this line. The inner dashed and the outer lightly coloured bands are, in order, 10% and 20% error margins. Figure 6.2a shows the scale predictions of all patches (black dots) in the PED training data set. In the plot, sets of the black dots are aligned vertically because they are obtained from different patches of the same image, which has the same actual scale. The predicted scales are dispersed around their actual values due to estimation errors. Thus, the values are aggregated using a median function, shown in Figure 6.2b. Both median and mean functions were used for aggregation, but the



(a)



(b)

Figure 6.1: Loss curves on a log-linear plot of training in (a) and testing in (b) for three different patch sizes: 100×100 , 350×350 , and 850×850 . The PED data set is used for this experiment

median function is more robust to outliers, which may have been introduced from large local texture variations or blurring. Similar to Figure 6.2 for training, the AtP plots for the PED testing are in Figure 6.3. Note the scenes in the testing data sets were not used for training. The final aggregate prediction in Figure 6.3b is quite close to the actual scale values, and most of the predictions are within the 20% error margin. This means that any

region on the images can be measured with a 20% margin of error, using the trained scale estimation model.

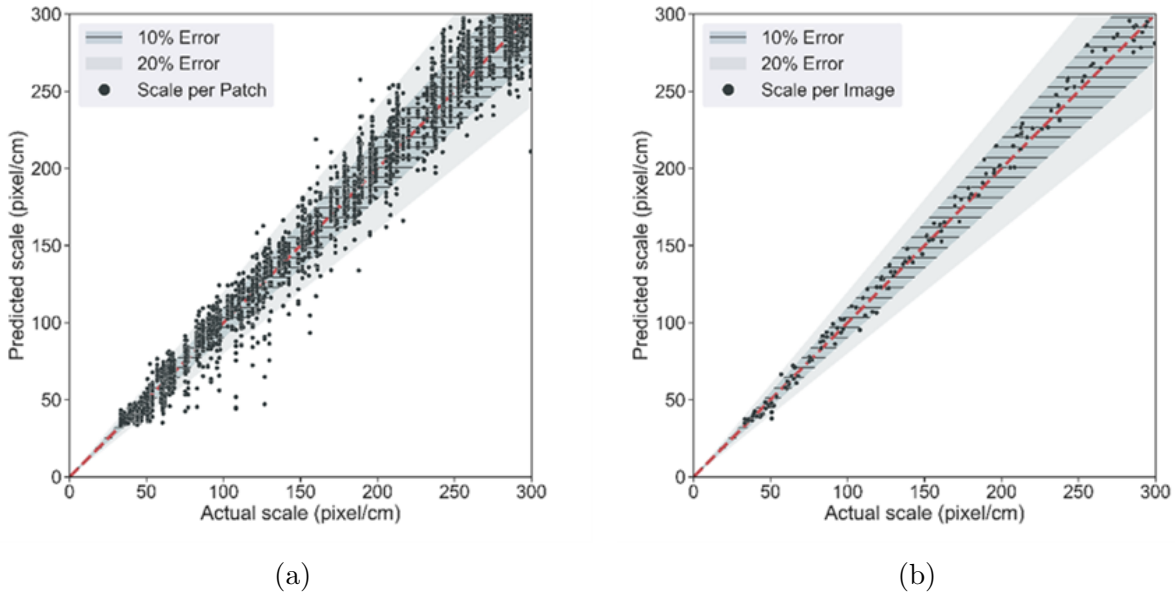


Figure 6.2: Actual-to-Predicted scale scatter (AtP) plots obtained from the PED training data set: (a) scales for all patches and (b) aggregated scales for each image using a median function. The red line indicates a correct prediction line and inner dashed and outer coloured bands indicate 10% and 20% error margins, respectively

The technique was also validated using BW and ASH data sets. The scale estimation model for each data set was trained using the same architecture and configuration as employed in the PED model. The aggregated AtP plots for each testing data set are shown in Figure 6.4. Note that since images for BW were captured with a lower resolution camera than those obtained for the other two data sets at the same distance range, the range of the scale in Figure 6.4a is smaller (up to 200 pixel/cm). In Figure 6.4b, the results from ASH testing data set have more dispersion in larger actual scales. These variations

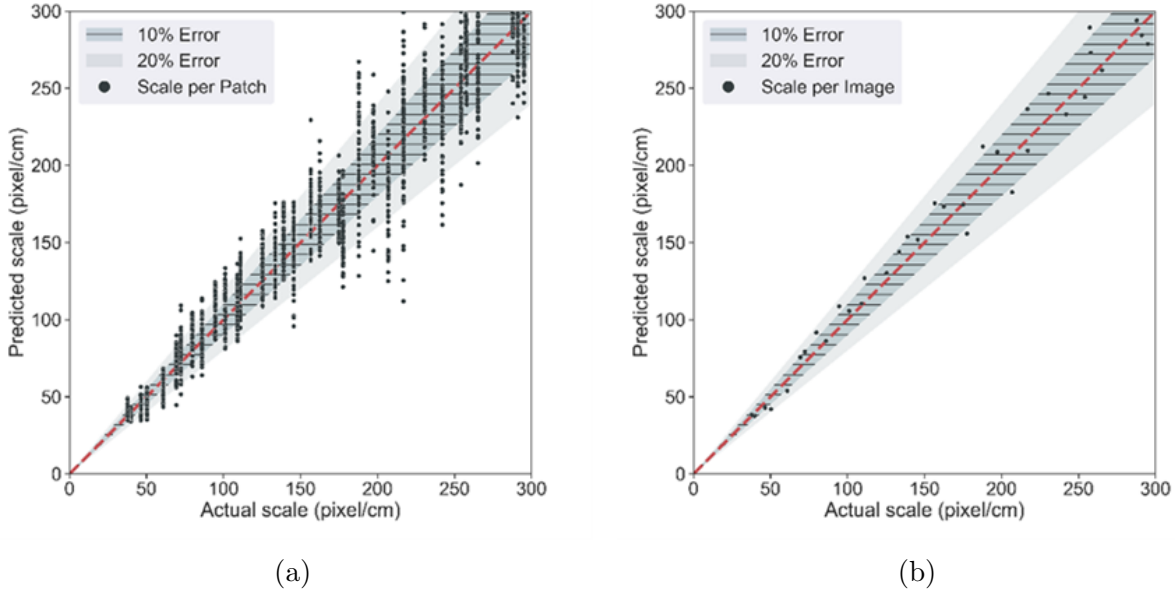


Figure 6.3: AtP plots obtained from the PED testing data set: (a) scales for all patches and (b) aggregated scales for each image using a median function

are caused by partial light variations by shading or debris on the ground. As such, images taken with shorter standoff distances are more prone to wrongly predict the scale if such variations are present in the scenes. As the scenes are captured at a closer distance, regions for extracting patches are smaller, being more susceptible to such local variations.

The scale prediction results for all three structures are summarized in Table 6.1. The table shows the average and standard deviation error of the three structure data sets used in this study. As previously shown, two aggregate functions used to aggregate multiple scale estimations for a given image into a single scale are mean and median. For both aggregations, the error is represented as mean values with their standard deviation. The preference of one function over the other is not immediately obvious; however overall, the

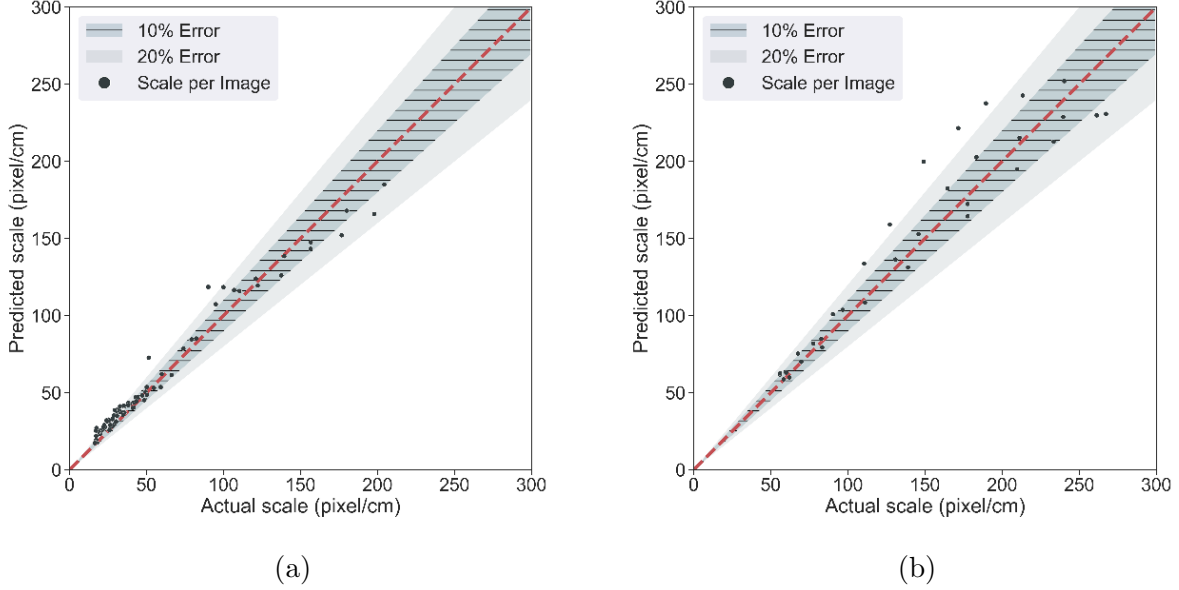


Figure 6.4: AtP plots for (a) BW and (b) ASH testing data sets. The scales are aggregated using a median function

performance of the median-based aggregation was shown to slightly outperform the mean-based aggregation. The scale for all three cases can be accurately estimated within a 20% error most of the time.

Aggregation	PED	BW	ASH
Mean	$6.7\% \pm 4.0\%$	$15.8\% \pm 13.6\%$	$10.5\% \pm 8.4\%$
Median	$7.3\% \pm 4.5\%$	$14.1\% \pm 11.9\%$	$9.9\% \pm 8.3\%$

Table 6.1: Overall scale prediction results for all three structures: aggregation using either a mean or median function

In addition, the transferability of the scale estimation model was explored by examining the impacts of (i) training and testing images being captured with different cameras and (ii) the zoom function being used to capture the scene in distance. First, for actual imple-

mentation, it is likely that users will collect images using different cameras from those that were used for collecting images for model training. Figure 6.5a shows the AtP plot aggregated using the median function. The scale estimation model for PED shown in Figure 6.2 is applied to the images collected from a different camera, used for this experiment. The mean error and its standard deviation are $11.8\% \pm 9.7\%$. Compared with the accuracy in Figure 6.3b and Table 6.1, prediction results are still reasonably good but less accurate than the one from the same camera images. This is because the model may have learned the camera's features in addition to features associated with the textures, such as lens distortion and colour variation. Although various data augmentations such as colour shift or intensity variations were imposed, the model has overfitted specific image features because it was trained using images from a single camera. The model could have been using images captured from different cameras to relieve the overfitting issues, but it requires extra effort to collect images.

Second, the effect of the zoom function was investigated. Figure 6.5b shows the aggregated AtP plot. Like the previous test, the scale estimation model developed using PED in Figure 6.2 is applied to the zoomed images. The median and its median absolute deviation is $16.3\% \pm 12.5\%$, which is comparable to the results presented in Table 6.1 and in the order of the 20% error bounds. Like the results obtained for the different camera, the prediction accuracy is reasonable, however, could not be deemed highly accurate. Nevertheless, the error levels obtained would still permit the application of existing crack classification procedures. For example, the International Atomic Energy Agency [23] guidebook, which employs some of the more stringent criteria, categorizes the damage severity in concrete structures using a series of grades: Grade I is used for members/structures with maximum

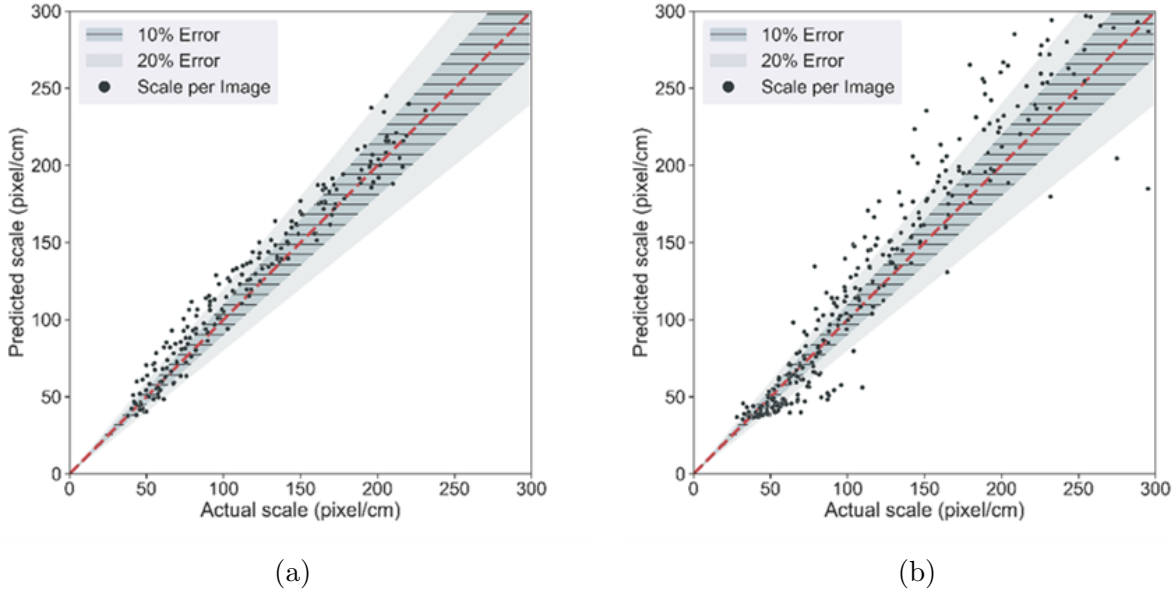


Figure 6.5: AtP plots for new PED testing data sets collected (a) using a different camera and (b) under different focal lengths. The scale estimation model in Figure 6.2 is applied to these two data sets

crack widths that are less than or equal to 0.20 mm, Grade II is assigned to structures with maximum crack widths that fall between 0.20 and 1.0 mm, and Grade III is assigned to structures with maximum crack widths that are greater than 1.0 mm. Similar ranges of pre-established crack width limits are established for transportation structures in North America [38, 2]. Considering the bounds on these types of pre-established crack classification categories, 15% average measurement error is not likely to have a critical impact on damage classification, with the exception of crack width estimates at the boundaries of the grade ranges.

One potential reason for the magnitudes of error obtained in these cases is that images

with zoom suffer from lens (barrel) distortion, causing scenes away from the centre of image to bend. This causes more variations and bias on the estimated scales. Finally, it should be noted that, in the case of arm's length inspections, the impact of focal length is arguably of lesser significance than camera type because at short standoff distances, adequate ROI and feature data can typically be captured with the application of the zoom function.

The error statistics of the three different structures, as well as the two tests of model transferability on different data collection scenarios are summarized as a box and whisker plot in Figure 6.6. The positive effect of aggregation (using median function in this figure) is prevalent through all 5 data sets, seeing an average mean error reduction ranging from 4% to 6%, and a consistent decrease in variance. Using aggregation results in more consistent scale predictions, which is more robust against local variations and minimally self-similar surface textures. Lastly, errors are comparably larger for ZOOM and DIFF data sets than they are for the PED data set. This is likely due to the model training overfitting to the unique visual characteristics of the specific camera/collection techniques used. The difference in focal lengths (for ZOOM and DIFF) and camera configurations (for DIFF) leads to subtle differences in the overall image quality such as tone and contrast that negatively influence model performance. However, despite this, it is evident that the correlation between texture and scale is decently preserved despite using differing collection techniques or cameras.

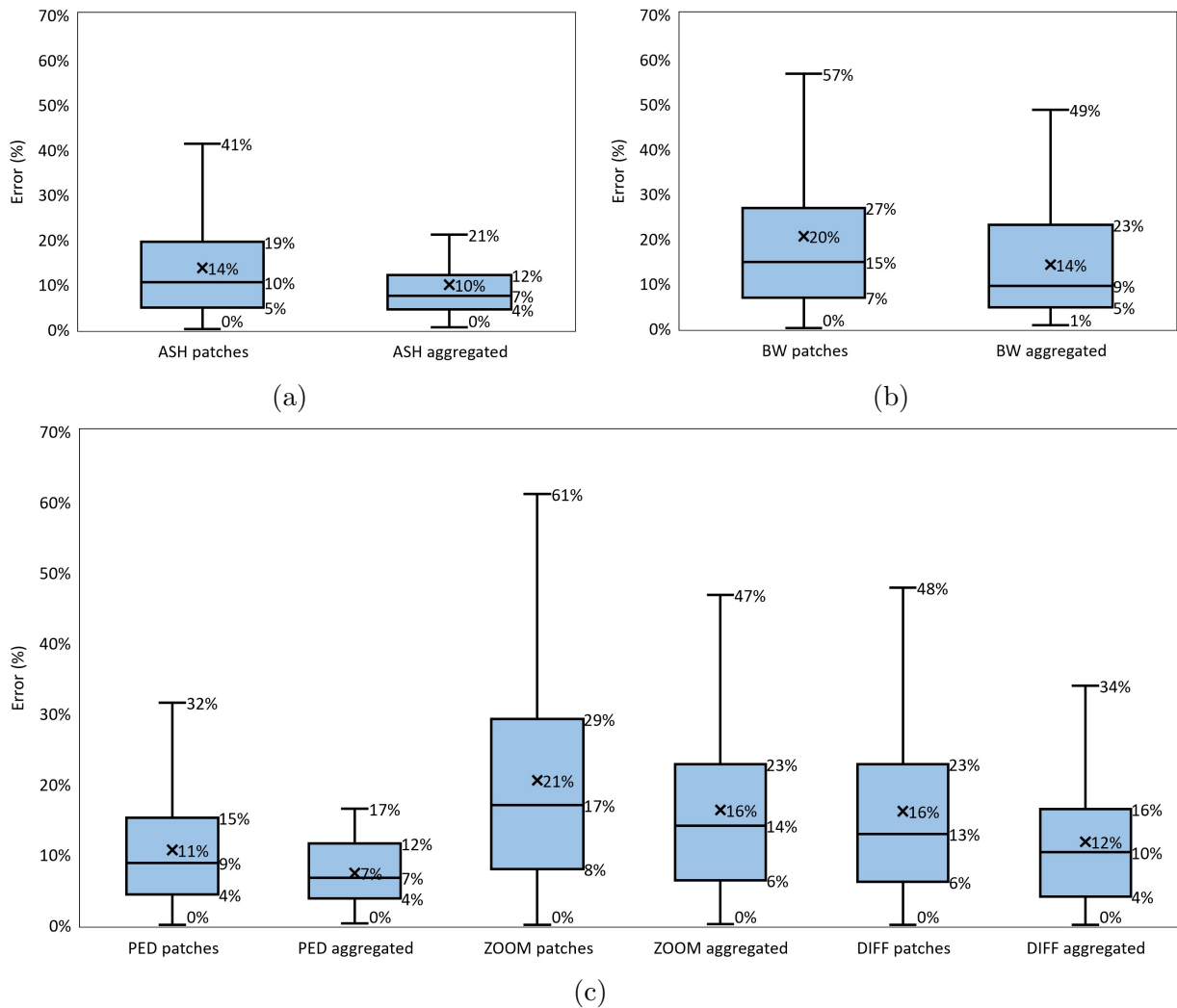


Figure 6.6: Box and whisker plots for (a) ASH, (b) BW, and (c) PED, ZOOM, and DIFF testing data sets. Error statistics of non-aggregated and aggregated scales are shown side-by-side for each data set. The error mean and median are represented as the X mark and solid line through the box, respectively. The box represents the inter-quartile range, where its upper and lower edge 75th and 25th percentile. The top and bottom of the vertical line represent the largest and lowest data point, excluding outliers. Error statistics shown for the aggregated scales are aggregated using the median function

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this study, a learning-based image scale estimation technique was developed. The technique leverages visually prevalent texture information contained in inspection images and translates that to an image scale. The image scale was used to convert pixel-based measurements of ROIs (e.g., spalling or cracks) into physical measurements (e.g., mm or cm), which can be used to enable quantitative evaluations outlined in typical inspection manuals.

To enable a robust and easy-to-implement surface texture to scale translation methodology, several techniques were utilized.

- A CNN-based scale estimation model was trained to predict the scale given a texture patch extracted from the image. To improve the model performance against local variations and minor degrees of self-similarity, scales are predicted for several patches

extracted from a single image and averaged using either a mean or median aggregation function to derive a final image scale. Furthermore, the training data was randomly augmented using random intensity changes, mean shifts, and horizontal and vertical flips and the image pixel range normalized to a range between -1 and $+1$ to combat small data set size and be more tolerant of introduced biases when using different cameras or different data collection methods.

- The MAPE loss function was selected after empirical analysis for its ability to equally distribute the loss across the possible scale range. This is important, as other loss functions, such as MAE or MSE would penalize error corresponding to larger scale values significantly more than error corresponding to smaller scale values, leading to a model with bias towards larger values. Furthermore, imbalances in scale distribution were eliminated through an automatic analysis of the histogram of scale counts combined with an oversampling approach. The scale distribution was equalized by extracting more patches from images corresponding to a scale bin with low number of counts, while less patches were extracted from images corresponding to a scale bin with high number of counts.
- To efficiently generate the ground-truth texture data set for training, an automatic data labelling and training data generation algorithm was implemented. The algorithm involves attaching a marker of known dimension on the structure's surface and collecting images with the marker included in the scene. Then, the algorithm automatically detects the marker, and calculates the scale for the image. Then, patches of the structure texture are extracted from the images without including any portion

of the marker, which form the training data set. The only manual part of this process is the data collection phase. Among various fiducial markers, ArUco markers and their corresponding detection algorithm was used.

To demonstrate the capability of the proposed scale estimation model, images collected from three different structures were used for training and testing the model. In addition, the proposed model was explored for the influence of two different operating conditions, which is using camera zoom and using a different camera. The technique was shown to successfully estimate image scale solely by inferring from the surface texture, with less than 20% mean error for all testing cases.

7.2 Future Work

This study has examined the implementation and validation of patch-based image scale estimation of surface textures using regression CNNs. Despite its advantages of being able to directly infer image scales using nothing but textures contained in the collected image, one main practical limitation is that images are assumed to be collected relatively parallel to the surface. This is because when collecting images relatively parallel to the surface, only one scale needs to be estimated. However, as is typical of data collection during visual inspection, images can also be taken at slight angles to the target scene, resulting in planar surfaces with projective distortion. For such images, no single scale value exists, and is difficult to convert pixel measurements into physical measurements using the technique proposed in this study. Furthermore, there remains room for improvement in

model accuracy.

There are several ideas which are being considered to improve the technique. First, there are improvements to the model training process and model design that have yet been undertaken. In the current study, the model only observes a single patch to output a single scale. However, the model can be augmented to accept multiple patches from a single image to output a scale prediction which would allow the model to predict the image scale while having significantly more information than the current model design, increasing robustness against local variations in texture in the CNN. Furthermore, the training data set can be artificially increased using the magnification augmentation. Note that, to implement the zoom augmentation the image scale needs to change based on the zoom factor applied on the patch. Second, inspection images often exhibit low to moderate degree of perspective distortion, meaning that the images are taken usually within 25 degrees perpendicular to the surface being captured. The same technique has yet to be applied to such images and their error statistics have yet to be explored. If the error introduced by the perspective distortion is low, and the model error statistics shown in this work can be improved through suggestions in the first improvement suggestion, the model could be applied to images of low perspective distortion with minimal changes to the technique. Lastly, a self-supervised CNN technique can be explored, which is inspired by the work of DeTone et al. [12]. A CNN model can learn to estimate the distance between two user selected points in an image, whether it exhibits significant projective distortion. The model architecture would take as input one patch, and the two difference vectors between the patch's centroid to points 1 and 2 to predict physical distance between the two points. The main motive behind this idea is that projective distortion for approximately flat surfaces is linear and

can be inferred by a CNN model using a patch of the distorted texture. Then, the positions of two points relative to the extracted patch can be inputted into the model get a distance prediction. The relative position of the points to the patch is given to relate the visual information to the locations of the points, as otherwise, the model has no way to correlate the visual information with the selected points.

References

- [1] Hojjat Adeli. Four decades of computing in civil engineering. In *CIGOS 2019, Innovation for Sustainable Infrastructure*, volume 54, pages 3–11. Springer Singapore, 2020.
- [2] American Association of State Highway and Transportation Officials (AASHTO). *Manual for bridge element inspection: 2015 Interim*. AASHTO, 2014.
- [3] Yun-Kyu An, KeunYoung Jang, Byunghyun Kim, and Soojin Cho. Deep learning-based concrete crack detection using hybrid images. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2018*, page 36. SPIE, 2018.
- [4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.
- [5] Yuequan Bao, Zhiyi Tang, Hui Li, and Yufeng Zhang. Computer vision and deep learning-based data anomaly detection method for structural health monitoring. *Structural Health Monitoring*, 18(2):401–421, 2019.

- [6] Nicholas Charron, Evan McLaughlin, Stephen Phillips, Kevin Goorts, Sriram Narasimhan, and Steven L. Waslander. Automated bridge inspection using mobile ground robotics. *Journal of Structural Engineering*, 145(11):04019137, 2019.
- [7] Fu-Chen Chen, Mohammad R. Jahanshahi, Rih-Teng Wu, and Chris Joffe. A texture-based video processing methodology using bayesian data fusion for autonomous crack detection on metallic surfaces: A texture-based video processing. *Computer-Aided Civil and Infrastructure Engineering*, 32(4):271–287, 2017.
- [8] Jongseong Choi, Chul Yeum, Shirley Dyke, and Mohammad Jahanshahi. Computer-aided approach for rapid post-event visual evaluation of a building façade. *Sensors*, 18(9):3017, 2018.
- [9] Wooram Choi and Young-Jin Cha. SDDNet: Real-time crack segmentation. *IEEE Transactions on Industrial Electronics*, 67(9):8016–8025, 2020.
- [10] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807. IEEE, 2017.
- [11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.
- [12] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 224–236, 2018.

- [13] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.
- [14] Yuqing Gao and Khalid M. Mosalam. Deep transfer learning for image-based structural damage recognition: Deep transfer learning for image-based structural damage recognition. *Computer-Aided Civil and Infrastructure Engineering*, 33(9):748–768, 2018.
- [15] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.
- [16] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Jose Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481–491, 2016.
- [17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, 2016.
- [18] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.
- [19] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

- [21] Vedhus Hoskere, Yasutaka Narazaki, Tu A. Hoang, and Billie F. Jr. Spencer. Towards automated postearthquake inspections with deep learning-based condition-aware models. In *7th World Conference on Structural Control and Monitoring, 7WCSCM*, 2018.
- [22] Vedhus Hoskere, Jong-Woong Park, Hyungchul Yoon, and Billie F. Jr. Spencer. Vision-based modal survey of civil infrastructure using unmanned aerial vehicles. *Journal of Structural Engineering*, 145(7):04019062, 2019.
- [23] International Atomic Energy Agency (IAEA). *Guidebook on Non-destructive Testing of Concrete Structures*. Number 17 in Training Course Series. 2002.
- [24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML 15*, page 448–456, 2015.
- [25] Mohammad R. Jahanshahi, Jonathan S. Kelly, Sami F. Masri, and Gaurav S. Sukhatme. A survey and evaluation of promising approaches for automatic image-based defect detection of bridge structures. *Structure and Infrastructure Engineering*, 5(6):455–486, 2009.
- [26] Shang Jiang and Jian Zhang. Real-time crack assessment using deep neural networks with wall-climbing unmanned aerial system. *Computer-Aided Civil and Infrastructure Engineering*, 35(6):549–564, 2020.
- [27] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

- [28] Shahid Kabir, Patrice Rivard, and Gérard Ballivy. Neural-network-based damage classification of bridge infrastructure using texture analysis. *Canadian Journal of Civil Engineering*, 35(3):258–267, 2008.
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.
- [30] Xiangxiong Kong and Jian Li. Non-contact fatigue crack detection in civil infrastructure through image overlapping and crack breathing sensing. *Automation in Construction*, 99:125–139, 2019.
- [31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [33] Junhwa Lee, Kyoung-Chan Lee, Seunghoo Jeong, Young-Joo Lee, and Sung-Han Sim. Long-term displacement measurement of full-scale bridges using camera ego-motion compensation. *Mechanical Systems and Signal Processing*, 140:106651, 2020.
- [34] Ali Lenjani, Chul Min Yeum, Shirley Dyke, and Ilias Bilonis. Automated building image extraction from 360° panoramas for postdisaster evaluation. *Computer-Aided Civil and Infrastructure Engineering*, 35(3):241–257, 2020.
- [35] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

- [36] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755, 2014.
- [37] Li Liu, Jie Chen, Paul Fieguth, Guoying Zhao, Rama Chellappa, and Matti Pietikäinen. From BoW to CNN: Two decades of texture representation for texture classification. *International Journal of Computer Vision*, 127(1):74–109, 2019.
- [38] Ministry of Transportation (MTO). *Ontario structure inspection manual: OSIM*. Ontario Ministry of Transportation, Bridge Office, 2000.
- [39] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.
- [40] Yasutaka Narazaki, Vedhus Hoskere, Tu A. Hoang, Yozo Fujino, Akito Sakurai, and Billie F. Spencer. Vision-based automated bridge component recognition with high-level scene consistency. *Computer-Aided Civil and Infrastructure Engineering*, 35(5):465–482, 2020.
- [41] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv:1811.03378 [cs]*, 2018.
- [42] Mohammad Hossein Rafiei and Hojjat Adeli. A novel machine learning-based algorithm to detect damage in high-rise building structures. *The Structural Design of Tall and Special Buildings*, 26(18):e1400, 2017.

- [43] Mohammad Hossein Rafiei, Waleed H. Khushefati, Ramazan Demirboga, and Hojjat Adeli. Supervised deep restricted boltzmann machine for estimation of concrete. *ACI Materials Journal*, 114(2), 2017.
- [44] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
- [45] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 91–99, 2015.
- [46] Francisco J. Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38–47, 2018.
- [47] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [48] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520. IEEE, 2018.
- [49] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101, 2010.

- [50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
- [51] Billie F. Spencer, Vedhus Hoskere, and Yasutaka Narazaki. Advances in computer vision-based civil infrastructure inspection and monitoring. *Engineering*, 5(2):199–222, 2019.
- [52] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [53] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, 2019.
- [54] Zheng Tong, Jie Gao, Aimin Sha, Liqun Hu, and Shuai Li. Convolutional neural network for asphalt pavement surface texture analysis: Convolutional neural network. *Computer-Aided Civil and Infrastructure Engineering*, 33(12):1056–1072, 2018.
- [55] Philip H. S. Torr and Andrew Zisserman. Feature based methods for structure and motion estimation. In *Vision Algorithms: Theory and Practice*, pages 278–294, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [56] Yang Xu, Shiyin Wei, Yuequan Bao, and Hui Li. Automatic seismic damage identification of reinforced concrete columns from images by a region-based deep convolutional neural network. *Structural Control and Health Monitoring*, 26(3):e2313, 2019.

- [57] X. W. Ye, T. Jin, and C. B. Yun. A review on deep learning-based structural health monitoring of civil infrastructures. *Smart Structures and Systems*, 24(5):567–585, 2019.
- [58] Chul Min Yeum, Jongseong Choi, and Shirley J Dyke. Automated region-of-interest localization and classification for vision-based visual assessment of civil infrastructure. *Structural Health Monitoring*, 18(3):675–689, 2019.
- [59] Chul Min Yeum, Shirley J Dyke, Bedrich Benes, Thomas Hacker, Julio Ramirez, Alana Lund, and Santiago Pujol. Postevent reconnaissance image documentation using automated classification. *Journal of Performance of Constructed Facilities*, 33(1):04018103, 2019.
- [60] Xue Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022, 2019.
- [61] Jarrod Zaborac, Apostolos Athanasiou, Salvatore Salamone, Oguzhan Bayrak, and Trevor D Hrynyk. Crack-based shear strength assessment of reinforced concrete members using a fixed-crack continuum modeling approach. *Journal of Structural Engineering*, 146(4):04020024, 2020.
- [62] Allen Zhang, Kelvin CP Wang, Yue Fei, Yang Liu, Siyu Tao, Cheng Chen, Joshua Q Li, and Baoxian Li. Deep learning-based fully automated pavement crack detection on 3d asphalt surfaces with an improved cracknet. *Journal of Computing in Civil Engineering*, 32(5):04018041, 2018.

Letter of copyright permission

**JOHN WILEY AND SONS LICENSE
TERMS AND CONDITIONS**

Dec 22, 2020



This Agreement between University of Waterloo -- JU AN PARK ("You") and John Wiley and Sons ("John Wiley and Sons") consists of your license details and the terms and conditions provided by John Wiley and Sons and Copyright Clearance Center.

License Number 4974311325017

License date Dec 22, 2020

Licensed Content Publisher John Wiley and Sons

Licensed Content Publication Computer-Aided Civil and Infrastructure Engineering

Licensed Content Title Learning-based image scale estimation using surface textures for quantitative visual inspection of regions-of-interest

Licensed Content Author Ju An Park, Chul Min Yeum, Trevor D. Hrynyk

Licensed Content Date Aug 19, 2020

Licensed Content Volume 0

Licensed Content Issue 0

Licensed Content Pages 15

12/22/2020

RightsLink Printable License

Type of use	Dissertation/Thesis
Requestor type	Author of this Wiley article
Format	Print and electronic
Portion	Full article
Will you be translating?	No
Title	Learning-based image scale estimation using surface textures for quantitative visual inspection of regions-of-interest
Institution name	University of Waterloo
Expected presentation date	Feb 2021
Requestor Location	University of Waterloo 8 DAWSON CRESCENT MILTON, ON L9T5H8 Canada Attn: University of Waterloo
Publisher Tax ID	EU826007151
Total	0.00 CAD
Terms and Conditions	

TERMS AND CONDITIONS

This copyrighted material is owned by or exclusively licensed to John Wiley & Sons, Inc. or one of its group companies (each a "Wiley Company") or handled on behalf of a society with which a Wiley Company has exclusive publishing rights in relation to a particular work (collectively "WILEY"). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the billing and payment terms and conditions established by the Copyright Clearance Center Inc., ("CCC's Billing and Payment terms and conditions"), at the time that

<https://s100.copyright.com/AppDispatchServlet>

2/6

you opened your RightsLink account (these are available at any time at <http://myaccount.copyright.com>).

Terms and Conditions

- The materials you have requested permission to reproduce or reuse (the "Wiley Materials") are protected by copyright.
- You are hereby granted a personal, non-exclusive, non-sub licensable (on a stand-alone basis), non-transferable, worldwide, limited license to reproduce the Wiley Materials for the purpose specified in the licensing process. This license, **and any CONTENT (PDF or image file) purchased as part of your order**, is for a one-time use only and limited to any maximum distribution number specified in the license. The first instance of republication or reuse granted by this license must be completed within two years of the date of the grant of this license (although copies prepared before the end date may be distributed thereafter). The Wiley Materials shall not be used in any other manner or for any other purpose, beyond what is granted in the license. Permission is granted subject to an appropriate acknowledgement given to the author, title of the material/book/journal and the publisher. You shall also duplicate the copyright notice that appears in the Wiley publication in your use of the Wiley Material. Permission is also granted on the understanding that nowhere in the text is a previously published source acknowledged for all or part of this Wiley Material. Any third party content is expressly excluded from this permission.
- With respect to the Wiley Materials, all rights are reserved. Except as expressly granted by the terms of the license, no part of the Wiley Materials may be copied, modified, adapted (except for minor reformatting required by the new Publication), translated, reproduced, transferred or distributed, in any form or by any means, and no derivative works may be made based on the Wiley Materials without the prior permission of the respective copyright owner. **For STM Signatory Publishers clearing permission under the terms of the [STM Permissions Guidelines](#) only, the terms of the license are extended to include subsequent editions and for editions in other languages, provided such editions are for the work as a whole in situ and does not involve the separate exploitation of the permitted figures or extracts**, You may not alter, remove or suppress in any manner any copyright, trademark or other notices displayed by the Wiley Materials. You may not license, rent, sell, loan, lease, pledge, offer as security, transfer or assign the Wiley Materials on a stand-alone basis, or any of the rights granted to you hereunder to any other person.
- The Wiley Materials and all of the intellectual property rights therein shall at all times remain the exclusive property of John Wiley & Sons Inc, the Wiley Companies, or their respective licensors, and your interest therein is only that of having possession of and the right to reproduce the Wiley Materials pursuant to Section 2 herein during the continuance of this Agreement. You agree that you own no right, title or interest in or to the Wiley Materials or any of the intellectual property rights therein. You shall have no rights hereunder other than the license as provided for above in Section 2. No right, license or interest to any trademark, trade name, service mark or other branding ("Marks") of WILEY or its licensors is granted hereunder, and you agree that you shall not assert any such right, license or interest with respect thereto
- NEITHER WILEY NOR ITS LICENSORS MAKES ANY WARRANTY OR REPRESENTATION OF ANY KIND TO YOU OR ANY THIRD PARTY, EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO THE MATERIALS

OR THE ACCURACY OF ANY INFORMATION CONTAINED IN THE MATERIALS, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, ACCURACY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, USABILITY, INTEGRATION OR NON-INFRINGEMENT AND ALL SUCH WARRANTIES ARE HEREBY EXCLUDED BY WILEY AND ITS LICENSORS AND WAIVED BY YOU.

- WILEY shall have the right to terminate this Agreement immediately upon breach of this Agreement by you.
- You shall indemnify, defend and hold harmless WILEY, its Licensors and their respective directors, officers, agents and employees, from and against any actual or threatened claims, demands, causes of action or proceedings arising from any breach of this Agreement by you.
- IN NO EVENT SHALL WILEY OR ITS LICENSORS BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR ENTITY FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, INDIRECT, EXEMPLARY OR PUNITIVE DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, PROVISIONING, VIEWING OR USE OF THE MATERIALS REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.
- Should any provision of this Agreement be held by a court of competent jurisdiction to be illegal, invalid, or unenforceable, that provision shall be deemed amended to achieve as nearly as possible the same economic effect as the original provision, and the legality, validity and enforceability of the remaining provisions of this Agreement shall not be affected or impaired thereby.
- The failure of either party to enforce any term or condition of this Agreement shall not constitute a waiver of either party's right to enforce each and every term and condition of this Agreement. No breach under this agreement shall be deemed waived or excused by either party unless such waiver or consent is in writing signed by the party granting such waiver or consent. The waiver by or consent of a party to a breach of any provision of this Agreement shall not operate or be construed as a waiver of or consent to any other or subsequent breach by such other party.
- This Agreement may not be assigned (including by operation of law or otherwise) by you without WILEY's prior written consent.
- Any fee required for this permission shall be non-refundable after thirty (30) days from receipt by the CCC.
- These terms and conditions together with CCC's Billing and Payment terms and conditions (which are incorporated herein) form the entire agreement between you and WILEY concerning this licensing transaction and (in the absence of fraud) supersedes

all prior agreements and representations of the parties, oral or written. This Agreement may not be amended except in writing signed by both parties. This Agreement shall be binding upon and inure to the benefit of the parties' successors, legal representatives, and authorized assigns.

- In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall prevail.
- WILEY expressly reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.
- This Agreement will be void if the Type of Use, Format, Circulation, or Requestor Type was misrepresented during the licensing process.
- This Agreement shall be governed by and construed in accordance with the laws of the State of New York, USA, without regards to such state's conflict of law rules. Any legal action, suit or proceeding arising out of or relating to these Terms and Conditions or the breach thereof shall be instituted in a court of competent jurisdiction in New York County in the State of New York in the United States of America and each party hereby consents and submits to the personal jurisdiction of such court, waives any objection to venue in such court and consents to service of process by registered or certified mail, return receipt requested, at the last known address of such party.

WILEY OPEN ACCESS TERMS AND CONDITIONS

Wiley Publishes Open Access Articles in fully Open Access Journals and in Subscription journals offering Online Open. Although most of the fully Open Access journals publish open access articles under the terms of the Creative Commons Attribution (CC BY) License only, the subscription journals and a few of the Open Access Journals offer a choice of Creative Commons Licenses. The license type is clearly identified on the article.

The Creative Commons Attribution License

The [Creative Commons Attribution License \(CC-BY\)](#) allows users to copy, distribute and transmit an article, adapt the article and make commercial use of the article. The CC-BY license permits commercial and non-

Creative Commons Attribution Non-Commercial License

The [Creative Commons Attribution Non-Commercial \(CC-BY-NC\) License](#) permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.(see below)

Creative Commons Attribution-Non-Commercial-NoDerivs License

The [Creative Commons Attribution Non-Commercial-NoDerivs License \(CC-BY-NC-ND\)](#) permits use, distribution and reproduction in any medium, provided the original work is properly cited, is not used for commercial purposes and no modifications or adaptations are made. (see below)

Use by commercial "for-profit" organizations

Use of Wiley Open Access articles for commercial, promotional, or marketing purposes requires further explicit permission from Wiley and will be subject to a fee.

Further details can be found on Wiley Online Library
<http://olabout.wiley.com/WileyCDA/Section/id-410895.html>

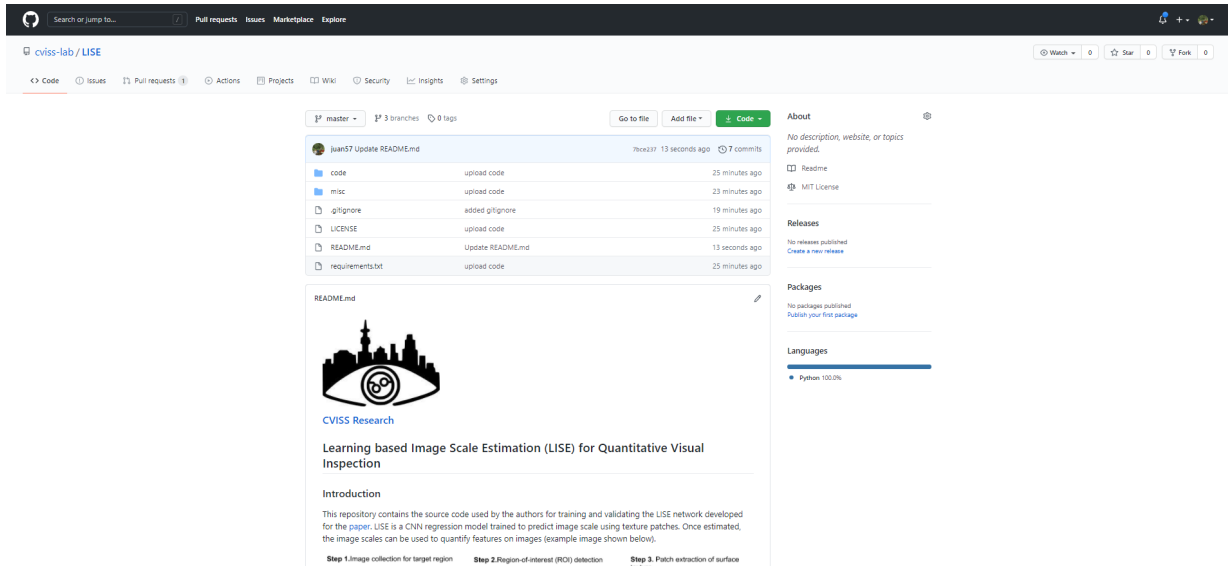
Other Terms and Conditions:

v1.10 Last updated September 2015

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.

Appendix

This appendix shows the main page and readme of the project published in the GitHub repository (<https://github.com/cviss-lab/LISE>). All source code and data have been shared through this repository. The page will be maintained by Computer Vision for Smart Structure Lab (cviss.net)



The screenshot displays the GitHub repository interface for 'cviss-lab / LISE'. The repository is on the 'master' branch with 3 branches and 0 tags. A commit by 'juan57' is shown, updating the README.md file 13 seconds ago. The file list includes 'code', 'misc', '.gitignore', 'LICENSE.md', 'README.md', and 'requirements.txt'. The README.md content is visible, featuring a logo with a city skyline and an eye, the text 'CVISS Research', and the title 'Learning based Image Scale Estimation (LISE) for Quantitative Visual Inspection'. An introduction section follows, explaining the repository's purpose and the LISE model. Three steps are listed: 'Step 1. Image collection for target region', 'Step 2. Region-of-interest (ROI) detection', and 'Step 3. Patch extraction of surface'. The right sidebar shows 'About' (no description), 'Releases' (no releases), 'Packages' (no packages), and 'Languages' (Python 100.0%).

The README file of the GitHub repository is shown. The README contains a summary of the proposed technique, as well as instructions on how to use the source code.



[CVISS Research](#)

Learning based Image Scale Estimation (LISE) for Quantitative Visual Inspection

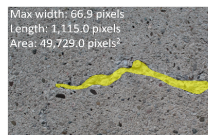
Introduction

This repository contains the source code used by the authors for training and validating the LISE network developed for the [paper](#). LISE is a CNN regression model trained to predict image scale using texture patches. Once estimated, the image scales can be used to quantify features on images (example image shown below).

Step 1.Image collection for target region



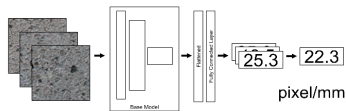
Step 2.Region-of-interest (ROI) detection



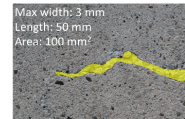
Step 3. Patch extraction of surface texture



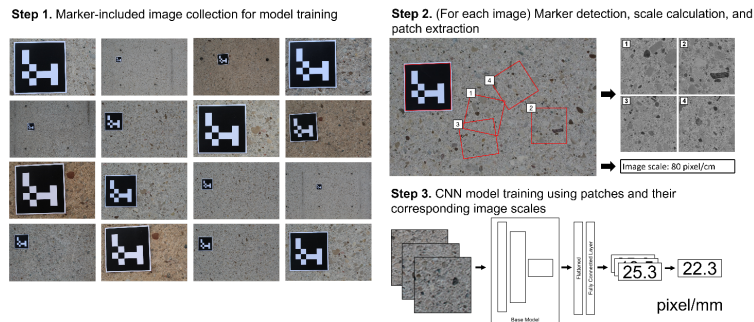
Step 4.Image scale estimation using trained CNN model



Step 5. Quantitative ROI evaluation



This repository strictly deals with the generation of the patch-scale image dataset, and the training of the CNN model. Note that for the pretrained models, all models use greyscale patch size of 850 X 850 pixels as input. The training framework is shown in the following image. The data generation algorithm uses ArUco markers as a method to calculate image scale.



Dependencies

LISE was built using the following dependencies ([requirements.txt](#)).

Python version: **3.7.7**

```
Keras==2.4.3
opencv_contrib_python==4.5.1.48
pandas==1.0.3
tqdm==4.46.0
numpy==1.18.4
matplotlib==3.1.3
Shapely==1.7.1
tensorflow-gpu==2.3.0
scikit_learn==0.24.1
```

NOTE for training with a cpu, use tensorflow instead of tensorflow-gpu

Sample usage example

Creating the training dataset from collected images

Step 1: unzip the sample dataset into the "datasets" folder

Download [sample_PED.zip](#) from the [data repository](#) and unzip to the **dataset** folder .

Step 2: Detect markers and generate the patch-scale dataset

In [create_montage_markers_by_scene.py](#), add this command to the main section of the code (at the bottom) like so:

```
if __name__ == '__main__':
    create_n_by_n_markers(n_crops=1, m_images=50, raw_folder='../datasets/PED/',
        out_folder='../datasets/PED/2_detected_imgs', marker_len=9.4)
```

n_crops: number of patches to include in one image (should be N^2)

m_images: number of images to extract

raw_folder: path to the collected dataset

out_folder: output folder of detected results

marker_len: in units of cm, the physical dimensions of the marker.

NOTE: in the case the algorithm cannot find the marker, the problem will visualize the image and ask the user to manually select the four points. In this case, simply click the 4 corners of the marker and click "c" to continue. To reset the corner selections, click "r".

Step 3: Split the dataset into training and testing

There should be **crop_dataset.csv** and **img_dataset.csv** in **datasets/PED/2_detected_imgs**.

crop_dataset.csv: each row contains the a patch image file path and its scale (used to train the model)

img_dataset.csv: each row contains a image file path and its scale (used to rapidly generate the crop dataset without rerunning the detection algorithm. See function

create_n_by_n_markers_from_df in **create_montage_markers_by_scene.py**)

Change the **pth** variable in **split_data.py** to **"../datasets/PED/2_detected_imgs"** and run it.

Note: the **validation_split** ratio can also be changed to control the ratio of images between training and testing datasets.

```
python split_data.py
```

This will split the **img_dataset.csv** and **crop_dataset.csv** into training and testing portions.

```
test_1_data_crop_dataset.csv
train_1_data_crop_dataset.csv
```

The two csv files are used to train the scale estimation model.

Training a image scale estimation model

Using the crop dataset generated in the previous section, we can train a patch-based scale estimator:

In **"model_training.py"**,

three variables can be adjusted to train/validate models:

- **train** (list of tuples): each tuple contains, in order the training configurations specific to the model being trained.
- **train_config** (dict): contains the training configurations applicable to all models
- **test** (list of tuples): each tuple contains configurations specific to a model test

For this example, ensure that the **train** and **train_config** variable looks as follows:

```
train = [
    # Sample
    ('../output/PED_sample', # Output folder
     '../datasets/PED/2_detected_imgs/train_1_data_crop_dataset.csv', # Path
    to the training crop csv
     'mape', # Loss function to use
     0.001, # Learning rate
     'reg', # does nothing
     '../datasets/PED/2_detected_imgs/test_1_data_crop_dataset.csv"), # Path
    to the test crop csv
]
```

```

train_config = {
    "epochs": 250, # number of epoches
    "output_pth": '',
    "pth_to_labels": "",
    'img_norm': '-1_to_+1', # image normalization
    'norm_labels': False, # Normalize labels?
    'greyscale': True, # Color or greyscale images?
    "lf_setting": 'mape',
    'learning_rate': 0.001,
    "image_augmentations": { # image augmentations
        "channel_shift_range": 50.0,
        "brightness_range": [0.8, 1.2],
        "horizontal_flip": True,
        "vertical_flip": True,
    }
}

```

All model training results will be output in the **output** folder, which contains:

- best_model.h5: best performing model
- model.h5: most recent model
- hist.csv: Loss history
- training_config.json: training configuration used
- results: folder containing image results of patches
- train.csv and test.csv: the csvs used for training and validation - test.csv also contains model.h5 predictions
- loss.jpg: loss curves.

BibTeX Citation

```

@article{park2020LISE,
  author = {Park, Ju An and Yeum, Chul Min and Hrynyk, Trevor D.},
  title = {Learning-based image scale estimation using surface textures for
quantitative visual inspection of regions-of-interest},
  journal = {Computer-Aided Civil and Infrastructure Engineering},
  volume = {36},
  number = {2},
  pages = {227-241},
  doi = {https://doi.org/10.1111/mice.12613},
  url = {https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12613},
  year = {2021}
}

```