Deep Learning Implemented Structural Defect Detection on Digital Images

by

Wooram Choi

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

In partial fulfilment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

Civil Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Abstract

Periodical inspection is the dominant form of structural health monitoring (SHM). However, civil engineering societies in North America have expressed the common consent that the current inspection practice is not sufficient to ensure infrastructure safety. Moreover, the increasing number of aged infrastructures will require an advanced form of inspection systems.

The processes of vision-based methods for identifying damage using image processing algorithms (IPAs) are similar to human inspections because both use visual information. The outcomes of vision-based methods are much more intuitive than systems with traditional contact sensors. Accordingly, researchers have proposed a variety of different methods. For example, early research adopted IPAs directly into damage detection problems. The results from IPAs are intuitive but require manual decision-making processes. Further attempts have been made to establish automated decision-making systems using machine learning algorithms (MLAs). However, real-life applications are rare. The unavailability is mainly rooted in the fact that IPAs were developed and tested in controlled circumstances, while real-world situations often cannot be controlled. Mobile units with cameras have attracted great attention in the SHM discipline. This type of inspection can improve accessibility to infrastructures but still lacks automated damage detection. Even if IPAs and MLAs are integrated, the combined system (mobile units, IPAs, and MLAs) will likely be invalid in practice because this system inherits the limitations of IPAs. To overcome these challenges, IPAs should be replaced by advanced computer vision techniques.

In this thesis, deep learning (DL) is considered the key for surpassing the current state of vision-based approaches. Deep learning models are capable of learning features from raw data. Instead of manually developing IPAs, feeding raw data that were collected in uncontrolled environments and leading a machine to learn the features of the data may be a better approach. A deep learning model for classifying

images for damage detection into binary classes is introduced, and its performance is compared with IPAs. The results of the classification DL model demonstrate the possibility of replacing IPAs with DL models. A segmentation DL model is also introduced that demonstrates faster, more robust, more flexible, and more intuitive than competitive methods.

CO-AUTHORSHIP

This thesis has been prepared in accordance with the regulation of the integrated-article format stipulated by the Faculty of Graduate Studies at the University of Manitoba. Substantial parts of this thesis were submitted for publication to peer-reviewed technical journals as follows:

Choi, W., & Cha, Y.-J. (2020). SDDNet: Real-Time Crack Segmentation. *IEEE Transactions on Industrial Electronics, 67*(9), 8016–8025. DOI: 10.1109/TIE.2019.2945265, [Chapter 4]. I initiated this project by proposing the plan for researching this topic in my thesis proposal defense. I contributed to creating the dataset, designing and developing the deep learning model, conducting the comparative study, visualizing the results, writing the draft, and responding to the reviewers' comments.

Cha, Y.-J., **Choi, W**. & Büyüköztürk, O. (2017), Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks, *Computer-Aided Civil and Infrastructure Engineering*, *32*(5), 361-378, DOI: 10.1111/mice.12263, [Chapter 3]. Dr. Cha initiated the project by providing an idea of damage detection using deep learning. I contributed to building the dataset, designing and developing the deep learning model, integrating the sliding-window along with the deep learning model, writing the draft guided by Dr. Cha and Dr. Büyüköztürk, responding to the reviewers' comments guided by Dr. Cha and Dr. Büyüköztürk.

Acknowledgments

I express gratitude to all my committee members, Dr. Young-Jin Cha, Dr. Dimos Polyzois, Dr. Yang Wang, and Dr. David Lattanzi for guiding me in my program. I am also grateful to Ms. Julia Osso and Dr. Dagmar Svecova for consulting and helping me in a tough situation. I thank all my colleagues for being sincere friends.

I acknowledge the support from the Natural Sciences and Engineering Research Council of Canada (NSERC) via the Discovery grant (Common Personal Identifier: 1262624) and Engage grant (Application No.: 533690-18), as well as the Canada Foundation for Innovation via the John R. Evans Leaders Fund (Project 37394).

Table of Contents

List of Figures

List of Tables

Acronyms and notations

The following acronyms and notations that are employed in the chapters are listed here.

List	Description
1D, 2D, 3D	1-dimensional, 2-dimensional, 3-dimensional
BatchNorm	Batch normalization
CNN	Convolutional neural network
CV	Computer vision
DL	Deep learning
FC	Fully connected
FN	False negative
FP	False positive
GD	Gradient descent
IPA	Image processing algorithm
ML(A)	Machine learning (algorithm)
NN	Neural network
ReLU	Rectified linear unit
SHM	Structural health monitoring
TP	True positive
α	Learning rate
$b^{(l)}$	Bias at a layer of an ML model
$B^{(l)}$	Set of biases at the *l*-th layer of an ML model
$g(z)$	Operation that transforms z
ρ	Weight decay factor

$\mathcal{L}$	Loss function
$\mathcal{J}$	Cost function that aggregates the losses across all training instances
$\phi^{(l)}$	Weight at the *l*-th layer of an ML model
$\Phi^{(l)}$	Set of weights at the *l*-th layer of an ML model
$\psi^{(l)}$	Feature at the *l*-th layer of a ML model
$\Psi^{[i]}$	Set of features at a layer of an ML model that corresponds to the *i*-th training instance.
y, $\hat{y}$	Output of an ML model and the ground truth that corresponds to y

Chapter 1 Introduction

Summary

The concerns with the conditions of infrastructures have suggested their periodical and frequent inspection. Human-conducted on-site inspections are the most practical process to date, but research on current practices has reported several challenges. Numerous methods that apply image processing algorithms (IPAs) have been proposed to replenish the current inspection practice. However, IPAs are still rare in real-life applications. This chapter introduces representative IPAs that were previously proposed and analyzes their limitations to identify viable breakthroughs.

1.1 Motivations

Infrastructures including buildings, bridges, dams, etc. are becoming vulnerable to the failure of their functionalities as they deteriorate over time. The majority of infrastructures were built from the 1950s to the 1960s, and their designed longevities were approximately 50 years. Thus, these structures have either reached or exceeded their designed service life spans (American Society of Civil Engineers, 2017). This finding has encouraged governments to conduct routine visual inspections of existing facilities. For example, the Government of Canada plans to inspect bridges every 540 days or less (the Government of Canada, 2018), and the U.S. government regulates biannual inspections at a minimum (Federal Highway Administration, 2004).

However, the periodical inspections performed by field engineers have several limitations that have been highlighted for years: 1) the inspection results are not consistent due to the subjective perceptions, technical skills, and expertise of human inspectors (Phares et al., 2001); 2) human resources are limited, and older infrastructures should be inspected more frequently than newer infrastructures (Federal Highway Administration, 2004; the Government of Canada, 2018); and 3) specific parts of infrastructures are often inaccessible, and inspecting them is time-consuming (Wells and Lovelace, 2018), etc. Accordingly, methods for monitoring structural health have been proposed by researchers to overcome these challenges.

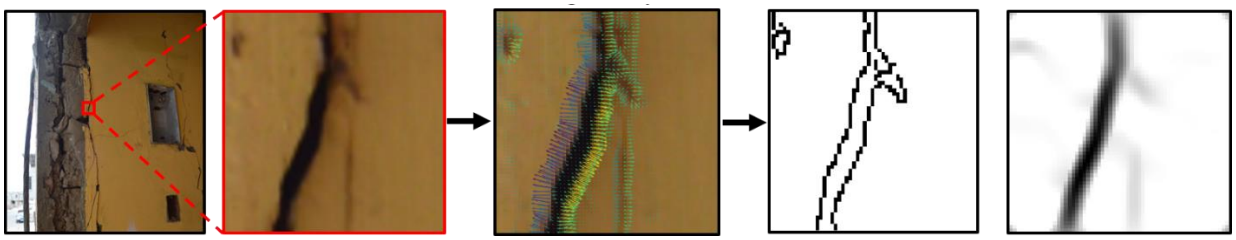
Early studies in structural health monitoring (SHM) mainly navigated the identification of dynamic behaviors of structures by analyzing the data collected from numerical models and physical vibrations measured from contact sensors of real structures (Rabinovich et al., 2007; Chatzi et al., 2011; Teidj et al., 2016). However, vibration signals collected by these sensors are vulnerable to uncertainties, such as environmental changes (Cornwell et al., 1999; Xia et al., 2012),

and the sensing range of a contact sensor is not enough to cover a massive structure (Kurata et al., 2012). These sensory systems also need to be inspected to check the functionality of the systems by visiting the site where the system is installed. If an alarm is raised, engineers eventually need to visit the site to verify the presence of damage.

Currently, the combination of computer vision (CV) and mobile units (e.g., drones and vehicles) is suggested as one of the best alternatives (Wells and Lovelace, 2018). The combined system can cover small and large structures with a single camera. The collected data provide intuitive information, such as the location, type and extent of the damage. However, an engineer still needs to manually analyze digital images, which is time-consuming even though the process is similar to human-conducted on-site inspection. Hence, developing methods for analyzing digital images and identifying damage may have a significant role in SHM.

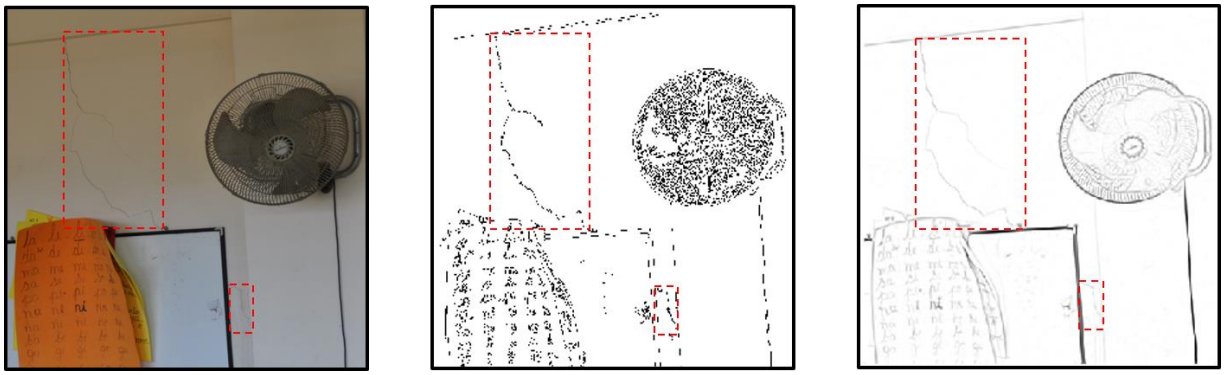
Subsequently, image processing algorithms (IPA) for extracting sensitive features of damage can be an alternative. Representative IPAs applied for damage detection include the fast Haar transform (Kaiser, 1998), fast Fourier transform (Cooley and Tukey, 1965), Sobel edge detector (Kanopoulos et al., 1988), Canny edge detector (Canny, 1986), etc. Abdel-Qader et al. (2003) investigated the effectiveness of these IPAs in detecting concrete cracks. Research articles related to IPAs were introduced (Sinha and Fieguth, 2006; Alaknanda et al., 2009; Yamaguchi and Hashimoto, 2009; Nishikawa et al., 2012). However, IPAs fundamentally take advantage of image gradients, and this dependency causes critical issues. For example, consider a system for detecting cracks in buildings, as shown in Figure 1.1(a), in which the edges represent the damage features. The image gradients shown in Figure 1.1(b) may enable the identification of reliable results (Figure 1.1(c)) in certain regions, as depicted by the red box in Figure 1.1(a). However, image gradients can also be observed on any object, as shown in Figure 1.2, where all the extracted features aside

from the red boxes are false detections. Therefore, these methods likely result in incorrect detections unless the image-obtaining conditions are fully controlled so that the obtained images contain only the damage features.



(a) Digital image (b) Image analysis (c) Feature extraction

Figure 1.1: Feature extraction example1



(a) Digital image (b) Edge detection (c) Ridge detection

Figure 1.2: Feature extraction example2

There have been attempts to combine IPAs with machine learning (MLAs). For example, Jahanshahi et al. (2011) and Cha et al. (2016) proposed methods that extract features using IPAs and trained a neural network (NN) and support vector machine (SVM) to discriminate extracted features from damage features. In the methods proposed by Yeum and Dyke (2015) and Ramana et al. (2018), the authors conducted object localization using a Cascade object detector before extracting features of damage. Despite the implementation of MLAs, its application to practical usage was not enough because the performances of IPAs were still susceptible to image-obtaining

conditions and lacked the ability to accurately extract features. This finding implies that any vision-based applications with IPAs are likely invalid for the detection of structural defects without prior knowledge of the location of damage.

Nevertheless, the vision-based approach is still an attractive method that can overcome the limitations of traditional IPA-based approaches. The works and corresponding results of this dissertation were initiated to address these limitations using deep learning (DL). DL might be one of the best compelling options in vision-based SHM. A particular branch of DL approaches that provides great properties in processing images is the convolutional neural network (CNN). CNNs are inspired by the behavior of the visual cortex (Ciresan et al., 2011) of mammals and are often referred to as DL because a CNN model usually consists of deep layers of operations. A pioneering research work (LeCun et al., 1998) showed its potential for discriminating images into several classes. Although the work aimed to classify small images of handwritten digits, the capability of automatic and adaptive feature extraction well presented the potential of CNN. Later, the development of parallel computing (Steinkrau et al., 2005) in graphics processing units (GPU) could accelerate DL research.

DL gained even greater attention when AlexNet (Krizhevsky et al., 2012) was introduced because the model showed the possibility of differentiating images into 1K classes. Subsequently, Simonyan and Zisserman (2014), He et al. (2016), and Huang et al. (2017) showed several ways to design a DL model; their findings have contributed to modern DL research. Comparing the performances between human-created IPAs and DL models, DL models may generally outperform IPAs. However, a DL model can only be built by training the model, and training requires a large volume of datasets. Several public datasets, such as Cityscape (Cordts et al., 2016), ImageNet (Deng et al., 2009), etc., for DL research are available, but those are not useful in developing DL

models for SHM. Therefore, researchers in SHM may need to build datasets for their particular purposes, which is one of the major works of developing a DL-implemented SHM.

1.2 Objectives

This dissertation aims to develop new methodologies that address the limitations of existing vision-based damage detection methods using deep learning models and suggest ways of detecting structural damage in uncontrolled conditions (e.g., outdoor environments). To prevent any hereditary limitations of previous vision-based methods, manual feature extraction[1] using IPAs are completely disregarded. Instead, CNNs fully replenish IPAs. Hence, an imperative task is to identify the architectures of CNNs that are capable of effectively extracting features from images. Creating datasets is another major objective, which should consist of images with complex features to train robust DL models.

1.3 Scope of work

In Chapter 2, fundamental operations, in DL are introduced. The explanations of fully connected layer, convolution, and pooling may provide ideas on how features can be extracted without IPAs. Furthermore, essential techniques (e.g., backpropagation, batch normalization, etc.) for building a DL model is also presented. Those operations and techniques are the foundations of the DL models introduced in the subsequent chapters. In Chapter 3, a DL model for classifying images into crack or non-crack classes is introduced. The deep learning model is composed of operations (e.g., convolution, pooling, etc.) that are described in Chapter 2. To minimize the

[1] The feature extraction algorithms manually created by humans are referred to as "manual feature extraction." This is for clarifying the meaning between feature extractions using an IPA and feature extractions using a DL model

susceptibility to luminance change caused by outdoor environments, the model is trained on images taken under extensively varying conditions. The usage of the model is limited to classifying each small image, and this property is not preferable in real practice. Thus, a sliding-window technique that extends the proposed model's capability to testing large images is integrated. In addition, comparative studies show that the proposed model may provide much more flexibility in real practice than previous vision-based approaches. In Chapter 4, a real-time DL model for detecting superficial damage of structures is presented. The deep learning model consists of standard convolution, densely connected separable convolution, and a modified atrous spatial pyramid pooling in addition to the operations explained in Chapter 2. In this work, the concept of detecting damage is defined as segmentation rather than classification because the results of segmentation provide better intuitive information and flexibility than those of image classification. However, a segmentation task requires classification of each pixel rather than each image, and the task is computationally much heavier than image classification. Therefore, a cost-efficient model that is dedicated to SHM is developed. The developed model does not require any IPAs, can accept any size of images, and provides robust results, although the images contain an extensive range of noisy features, complex background shapes, etc.

References

Abdel-Qader, I., Abudayyeh, O., & Kelly, M. E. (2003). Analysis of edge-detection techniques for crack identification in bridges. *Journal of Computing in Civil Engineering*, *17*(4), 255–263. DOI: 10.1061/(asce)0887-3801(2003)17:4(255)

American Society of Civil Engineers (2017). 2017 Infrastructure report card. Retrieved from https://www.infrastructurereportcard.org/wp-content/uploads/2019/02/Full-2017-Report-Card-FINAL.pdf

Alaknanda, Anand, R. S., & Kumar, P. (2009). Flaw detection in radiographic weldment images using morphological watershed segmentation technique. *NDT & E International*, *42*(1), 2–8. DOI: 10.1016/j.ndteint.2008.06.005

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-8*(6), 679–698. DOI: 10.1109/tpami.1986.4767851

Cha, Y.-J., You, K., & Choi, W. (2016). Vision-based detection of loosened bolts using the Hough transform and support vector machines. *Automation in Construction*, *71*, 181–188. DOI: 10.1016/j.autcon.2016.06.008

Chatzi, E. N., Hiriyur, B., Waisman, H., & Smyth, A. W. (2011). Experimental application and enhancement of the XFEM–GA algorithm for the detection of flaws in structures. *Computers & Structures*, *89*(7–8), 556–570. DOI: 10.1016/j.compstruc.2010.12.014

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 1237–1242. DOI: 10.5591/978-1-57735-516-8/IJCAI11-210

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, *19*(90), 297–301. DOI: 10.1090/s0025-5718-1965-0178586-1

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S. & Schiele, B. (2016). The Cityscapes Dataset for semantic urban scene understanding. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4700–4708. DOI: 10.1109/cvpr.2016.350

Cornwell, P., Farrar, C. R., Doebling, S. W., & Sohn, H. (1999). Environmental variability of modal properties. *Experimental Techniques*, *23*(6), 45–48. DOI: 10.1111/j.1747-1567.1999.tb01320.x

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. DOI: 10.1109/cvpr.2009.5206848

Federal Highway Administration. (2004). *National bridge inspections standards regulation (NBIS)*. Retrieved from https://www.govinfo.gov/content/pkg/FR-2004-12-14/pdf/04-27355.pdf

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. DOI: 10.1109/cvpr.2016.90

Huang, G., Liu, Z., Maaten, L. van der, & Weinberger, K. Q. (2017). Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261–2269. DOI: 10.1109/cvpr.2017.243

Jahanshahi, M. R., Masri, S. F., Padgett, C. W., & Sukhatme, G. S. (2011). An innovative methodology for detection and quantification of cracks through incorporation of depth perception. *Machine Vision and Applications*, *24*(2), 227–241. DOI: 10.1007/s00138-011-0394-0

Kaiser, G. (1998). The fast Haar transform. *IEEE Potentials*, *17*(2), 34–37. DOI: 10.1109/45.666645

Kanopoulos, N., Vasanthavada, N., & Baker, R. L. (1988). Design of an image edge detection filter using the Sobel operator. *IEEE Journal of Solid-State Circuits*, *23*(2), 358–367. DOI: 10.1109/4.996

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. DOI: 10.1145/3065386

Kurata, M., Kim, J., Lynch, J. P., Van der Linden, G. W., Sedarat, H., Thometz, E., … Sheng, L.-H. (2013). Internet-enabled wireless structural monitoring systems: Development and permanent deployment at the new Carquinez suspension bridge. *Journal of Structural Engineering*, *139*(10), 1688–1702. DOI: 10.1061/(asce)st.1943-541x.0000609

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. DOI: 10.1109/5.726791

Nishikawa, T., Yoshida, J., Sugiyama, T., & Fujino, Y. (2011). Concrete crack detection by multiple sequential image filtering. *Computer-Aided Civil and Infrastructure Engineering*, *27*(1), 29–47. DOI: 10.1111/j.1467-8667.2011.00716.x

Phares, B. M., Rolander, D. D., Graybeal, B. A., Washer, G. A. (2001). *Reliability of visual bridge inspection* (FHWA-RD-01-105). Retrieved from https://www.fhwa.dot.gov/publications/research/nde/pdfs/01020a.pdf

Rabinovich, D., Givoli, D., & Vigdergauz, S. (2007). XFEM-based crack detection scheme using a genetic algorithm. *International Journal for Numerical Methods in Engineering*, *71*(9), 1051–1080. DOI: 10.1002/nme.1975

Ramana, L., Choi, W., & Cha, Y.-J. (2018). Fully automated vision-based loosened bolt detection using the Viola–Jones algorithm. *Structural Health Monitoring*, *18*(2), 422–434. DOI: 10.1177/1475921718757459

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556*.

Sinha, S. K., & Fieguth, P. W. (2006). Automated detection of cracks in buried concrete pipe images. *Automation in Construction*, *15*(1), 58–72. DOI: 10.1016/j.autcon.2005.02.006

Steinkraus, D., Buck, I., & Simard, P. Y. (2005). Using GPUs for machine learning algorithms. *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. DOI: 10.1109/icdar.2005.251

Teidj, S., Khamlichi, A., & Driouach, A. (2016). Identification of beam cracks by solution of an inverse problem. *Procedia Technology*, *22*, 86–93. DOI: 10.1016/j.protcy.2016.01.014

The Government of Canada. (2018). *Guideline for bridge safety management*. Retrieved from https://www.tc.gc.ca/media/documents/railsafety/guideline-bridge-safety-management-english.pdf

Wells, J., & Lovelace, B. (2018). *Improving the quality of bridge inspections using unmanned aircraft systems (UAS)* (MN/RC 2018-26). Retrieved from https://www.dot.state.mn.us/research/reports/2018/201826.pdf

Xia, Y., Chen, B., Weng, S., Ni, Y.-Q., & Xu, Y.-L. (2012). Temperature effect on vibration properties of civil structures: a literature review and case studies. *Journal of Civil Structural Health Monitoring*, *2*(1), 29–46. DOI: 10.1007/s13349-011-0015-7

Yamaguchi, T., & Hashimoto, S. (2009). Fast crack detection method for large-size concrete surface images using percolation-based image processing. *Machine Vision and Applications*, *21*(5), 797–809. DOI: 10.1007/s00138-009-0189-8

Yeum, C. M., & Dyke, S. J. (2015). Vision-based automated crack detection for bridge inspection. *Computer-Aided Civil and Infrastructure Engineering*, *30*(10), 759–770. DOI: 10.1111/mice.12141

Chapter 2 Deep Learning Methodologies

Summary

Deep learning is undoubtedly the best alternative of the traditional IPA because DL models can be adaptively trained on data and automatically extract their features. This chapter describes how a DL model extracts features, is trained, and can be built. Training a DL model is challenging. Representative reasons and techniques to mitigate challenges (e.g., overfitting, vanishing/exploding gradients, etc.) are explained in this chapter. The contents of this chapter may provide fundamental backgrounds of the succeeding chapters.

2.1 Overview of deep learning

DL models are composed of deep and hierarchical layers of operations (Bengio et al., 2017). Each layer may extract features with or without learnable parameters. For example, as shown in Figure 2.1, the input image is fed into a DL model, and the model extracts features layer by layer, in which each input of a layer is the output of the previous layer. The presented example shows features of only four layers, but a greater number of layers usually accompany DL models. The remaining contents of this subchapter provide conceptual information of DL and its sub-operators. Further details are discussed in subsequent subchapters.

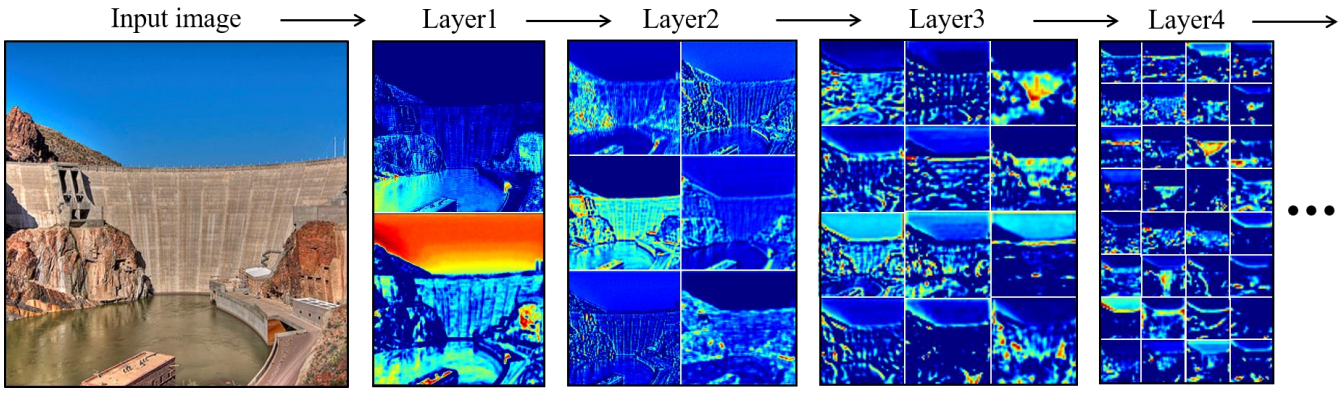


Figure 2.1: Hierarchical feature extraction of DL

One of the key operations with learnable parameters in DL is convolution, and a DL model with convolution is referred to as CNN. Other operations are often jointly employed in a DL model; these operations are explained in Chapter 2.3. Deep learning models have too many parameters (usually more than several million), and manually setting these parameters is impossible. Therefore, the parameters are initialized by random numbers, and an optimization algorithm is involved to automatically tune the parameters, in which the optimization process is referred to as training. In deep learning, gradient descent (GD) is considered one of the most effective optimization algorithms (LeCun et al., 2012). Details of the

algorithm are introduced in Chapter 2.4. The behavior of an optimization algorithm can be controlled by several parameters, which are independently defined as hyperparameters. Hyperparameters are set empirically via trials and errors. To train a CNN model, numerous instances are fed into the model; they are defined as the training set. In some applications, several instances in a training set are selected as validation data, which are applied to obtain hyperparameters by monitoring the training process. Once the hyperparameters are defined by trial and error, a model is trained using training and validation data, and the model is considered a trained model. A trained model is tested with a set of instances that have not been applied in training, and these instances are defined as test data. Testing is performed to ensure that the trained model also works on images that are not employed in training. A trained model is considered overfitted if the model only properly works on images that were applied in training. In practice, twenty percent of an entire dataset is considered the minimum proportion of test data.

2.2 Input process

In this thesis, the input data comprise a set of images, and the contents of this subchapter are provided based on the context. A digital image might be a color or grayscale image. In the case of a color image, there are different variations, but an image represented in the red, green, and blue (RGB) channels is the most common type of image. Accordingly, a digital image with RGB channels can be expressed as a three-dimensional (3D) tensor (i.e., height × width × channel) in a machine, in which a pixel in a color channel describes the brightness of the pixel, and each pixel has a number in the range of 0 to 255. This subchapter navigates fundamental input processing.

2.2.1 Data augmentation

Data augmentation is an essential process in training a model and known as an effective way to prevent overfitting, which is further discussed in Chapter 2.5. Augmentation can be performed by transforming the raw data. Both the augmented data and the raw data are employed in the training phase. Flipping, cropping, rotating, and shuffling RGB channels are the most common strategies. The combination of these strategies can also be applied, as shown in Figure 2.2. Assume that a model is trained to discriminate an image into bird or non-bird. The augmented images shown in the figure might be suitable examples of augmentation because all the augmented images can still be considered bird images. However, an augmented image must retain the same semantic meaning of the raw image. For example, if the colors of birds are important features (e.g., bird breed prediction), shuffling RGB channels should not be performed because the augmentation strategy changes the semantic meaning of data and other augmentation techniques can be potentially utilized unless there is no change in the semantic meaning.

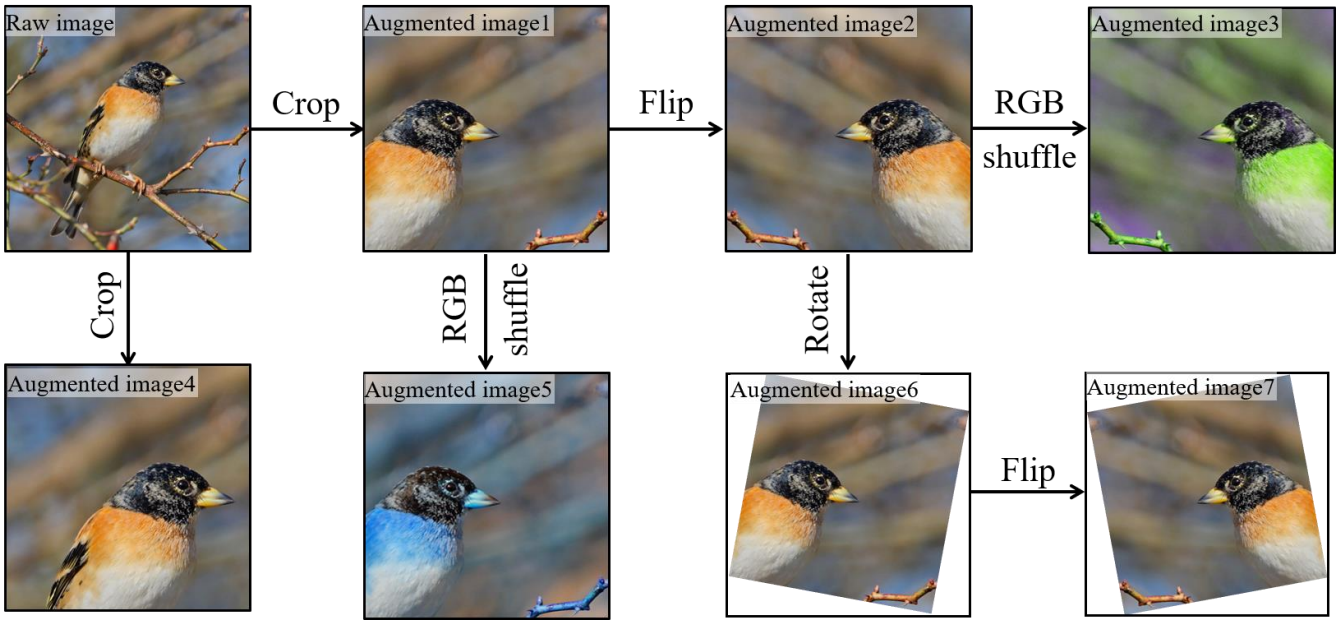


Figure 2.2: Data augmentation example

2.2.2 Image normalization

Normalizing an image is a common practice in training a DL model because it leads to stable and faster training (LeCun et al., 2012; Pal and Sudeep, 2016). There are several common strategies: 1) rescale pixel values from zero to positive one; 2) rescale pixel values from negative one to positive one; and 3) standardize the entire dataset. In the testing phase (refer to Chapter 2.1), images to be tested should be normalized by the same strategies that are employed in the training phase to obtain the expected results.

2.3 Operations in deep learning

This subchapter describes fundamental operations and layers that are extensively adopted in DL models. The quintessential form of a DL model is the standard NN, which is also often referred to as the feedforward NN and multilayer perceptron. One of the key operations in NN is a fully connected (FC) layer that is still extensively applied in DL models. A variant form of the FC layer is referred to as convolution, in which similar operations are performed in an FC layer but are attributed to sparse connectivity. Both the FC layer and convolution accompany trainable parameters. Pooling is another important operation, in which identical calculation is conducted without trainable parameters.

2.3.1 Fully connected layer

A fully connected layer has the property that the neurons of an input layer $\Psi^{(l-1)} = \left\{\psi_j^{(l-1)} \mid j \in \{1, 2, \ldots, n_{l-1}\}\right\}$ are connected to all neurons of the corresponding output $\Psi^{(l)} = \left\{\psi_i^{(l)} \mid i \in \{1, 2, \ldots, n_l\}\right\}$, as shown in Figure 2.3. The input and output of the *l*-th layer are fully connected by weights $\Phi^{(l)} = \left\{\phi_{i,j}^{(l)} \mid i \in \{1, 2, \ldots, n_l\}, j \in \{1, 2, \ldots, n_{l-1}\}\right\}$ and biases $B = \left\{b_{i,1}^{(l)} \mid i \in \{1, 2, \ldots, n_l\}\right\}$. The i-th neuron of the output is calculated by the operation g of the weighted sum (refer to Figure 2.3). The

weighted sum is considered feature extraction, and *g* can be any operation that may transform the features, introduce nonlinearity, etc. Note that weights and biases are referred to as parameters (i.e., learnable parameters; refer to Chapter 2.1); they are the main targets to be optimized in DL models. The optimization of weights and biases is discussed in Chapter 2.4. If an FC layer is positioned at the beginning of a model, $\Psi^{(0)}$ is a set of features (e.g., pixel, data point, etc.) of an input data (e.g., image, signal, etc.) to be fed into the model; otherwise, features are extracted from the previous layer (*l*-1). An FC layer is mostly positioned at the end of a DL model.

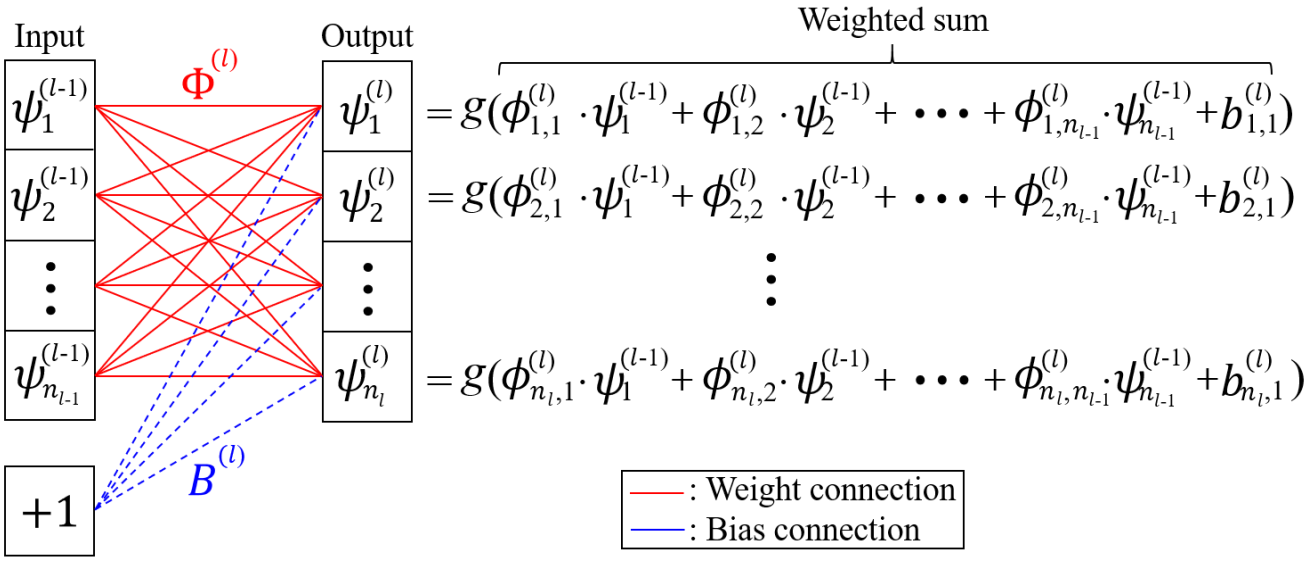


Figure 2.3: Example of fully connected network

2.3.2 Convolution

Convolution inherits properties that are similar to that in a fully connected layer (refer to Section 2.3.1), but the input and output are sparsely connected by shared weights, as shown in Figure 2.4. The n_ϕ number of weights at the *l*-th layer ($\phi_1^{(l)}$, $\phi_2^{(l)}$, …, $\phi_{n_\phi}^{(l)}$) performs the weighted sum to the input of the layer, in which the dimensions (i.e., width, height, or length) of the weights is usually smaller than that of the layer's input. The set of weights strides across the input dimension. A stride that is greater than one reduces

the dimension by approximately 1/stride. Each weighted sum might be further processed by an operation of g.

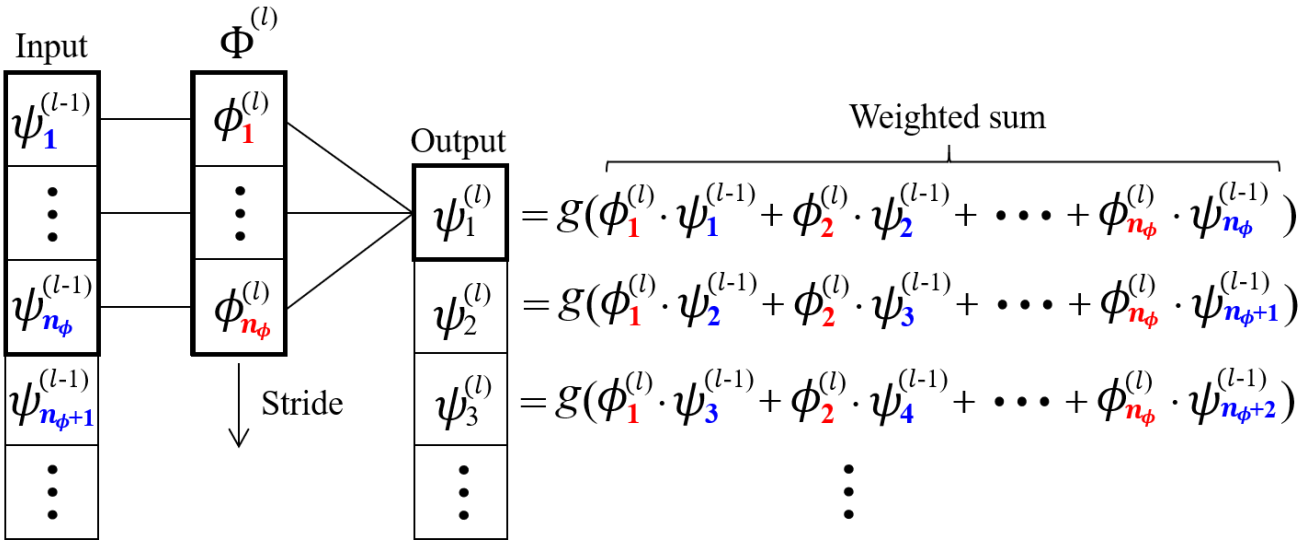
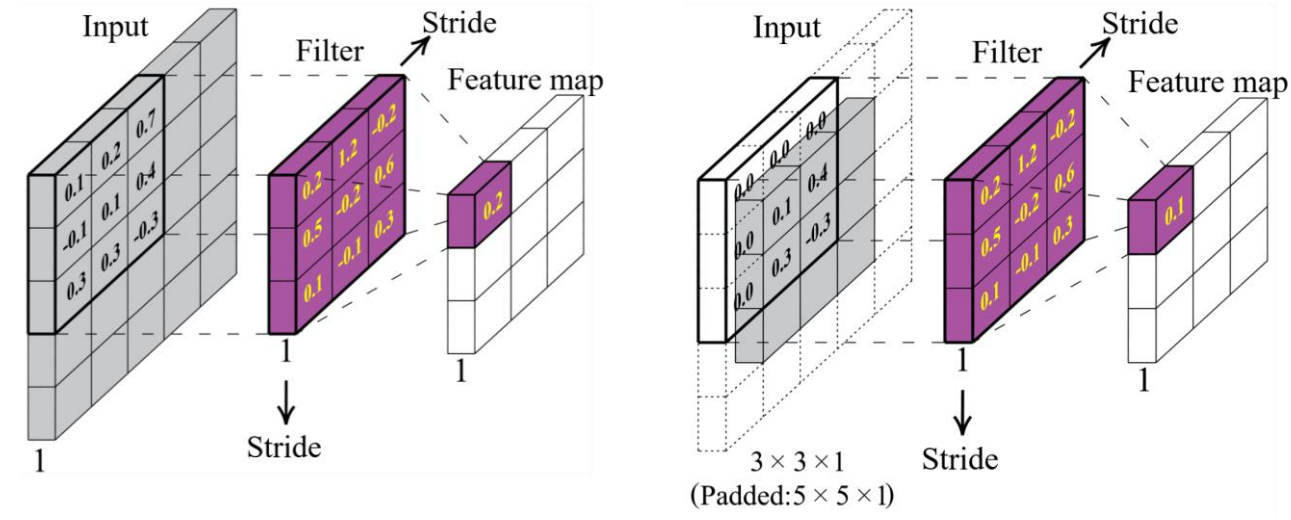


Figure 2.4: Convolution example with 1-D tensor

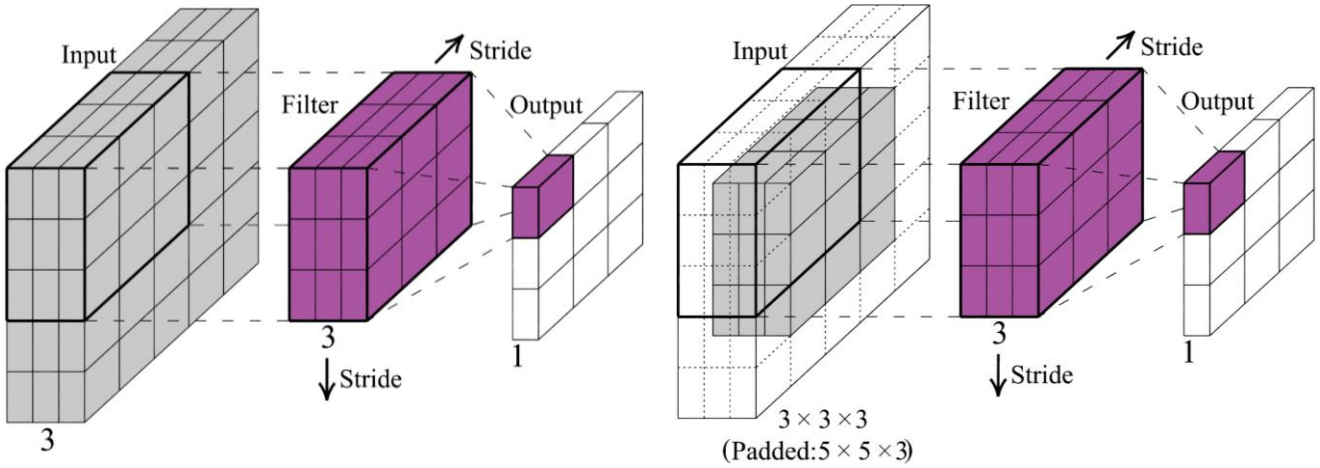
A set of weights is often referred to different terminologies, such as filter, receptive field, kernel, etc. To prevent confusion, a set of weights is referred to as 'filter' in this thesis. Note that the bias term is omitted in Figure 2.4 for simplicity, but each filter often accompanies a bias that is dependent on the configuration of a CNN. The striding filter allows application of the same practice even in multi-dimensional inputs. Hereafter, several figures that demystify the convolution operations are presented according to the following visualization rule: each output is presented with a certain color, in which the responsible filter to obtain the output is presented in the same color. As shown in Figure 2.5(a), an input of a 2-dimensional (2D) array with the spatial dimension of 5×5 can be convolved with a filter of the 2D array with the spatial dimension of 3×3. The filter conducts the weighted sum to the input by striding column-by-column and row-by-row. The output is accordingly a 2D array with the spatial dimension of 3×3. In this example, the spatial dimension is reduced from 5×5 to 3×3. To retain the spatial dimension of the output as that of the input, "padding" is commonly employed, as shown in Figure 2.5(b). The values

of the padded elements may differ depending on the application, but zeros (i.e., zero padding) are frequently applied. Hereafter, the set of extracted features by a filter is defined as a feature map for simplicity. A higher dimensional input can also be convolved with a filter, as shown in Figure 2.6(a), where the number of channels of input is 3, and that of the filter is accordingly equal to 3. Hereafter, "convolution" is denoted by ◉ in the figures throughout this thesis for simplicity.



(a) Without padding (b) With padding

Figure 2.5: Convolution example – single channel input



(a) Without padding (b) With padding

Figure 2.6: Convolution example – multi-channel input

In practice, it is required to produce more features; therefore, multiple filters are mostly applied to an input, as demonstrated in Figure 2.7. In the figure, the input is formed of an unspecified spatial dimension of *c* number of channels. Accordingly, the number of channels in each filter should be equal to *c*. *N* number of filters are convolved with the input, and the output feature map has *N* number of channels. The spatial dimension of the output feature map is dependent on the spatial dimension of the input and filters, stride values, and existence of paddings.

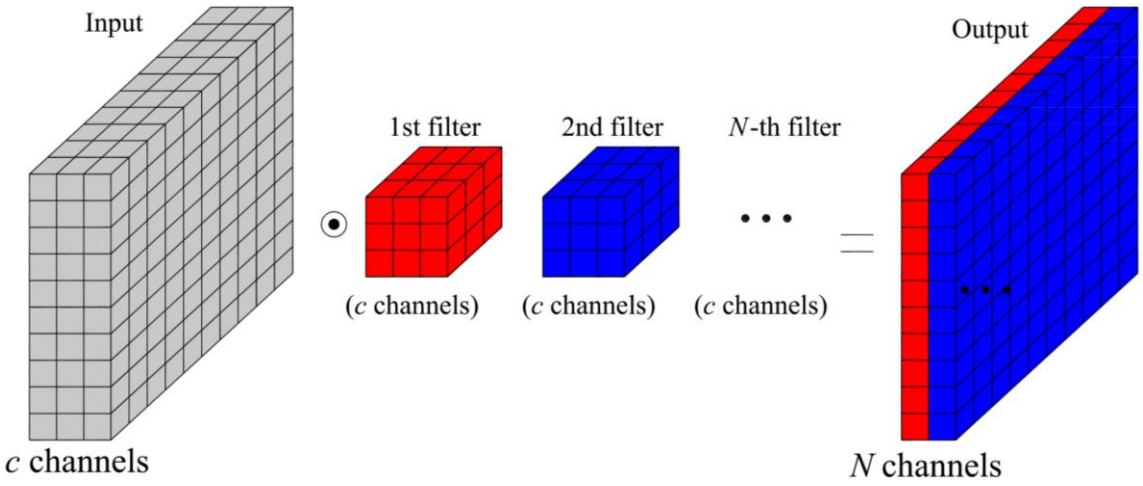


Figure 2.7: Convolution example – multi-channel input and multiple filters

To take into account the explanations from the beginning of Chapter 2.3.2, a feature ψ can be extracted by a convolution operation, which can be formally written by Eq. (2.1).

$$\psi^{(l)}_{h^{(l)},w^{(l)},\mathrm{c}^{(l)}} = \sum_{d_1^{(l)}} \sum_{d_2^{(l)}} \sum_{c^{(l-1)}} \phi^{(l)}_{d_1^{(l)},\, d_2^{(l)},\mathrm{c}^{(l-1)},\mathrm{c}^{(l)}} \times \psi^{(l-1)}_{\bar{h},\, \bar{w},\mathrm{c}^{(l-1)}}, \tag{2.1}$$

where

$\psi^{(l)}$ = a feature of the *l*-th layer.

$\phi^{(l)}$ = a weight of the *l*-th layer.

$h^{(l)}$ = height index of the *l*-th layer's feature map.

$w^{(l)}$ = width index of the *l*-th layer's feature map.

$c^{(l)}$ = channel index a feature map.

$d_1^{(l)}$ = height index of the *l*-th layer's filter.

d_2 = width index of the *l*-th layer's filter.

$\bar{h} = h^{(l)} + s_h - 1$

$\bar{w} = w^{(l)} + s_w - 1$

s_h = stride value of a filter in height direction.

s_w = stride value of a filter in width direction.

There are several variations in convolution, such as group convolution and depth-wise convolution. These variations inherit the same properties of the standard convolution (refer to Figure 2.7 and the corresponding explanations), except for dividing inputs. Regarding group convolution, an input is divided into several groups in the channel direction, and each group is convolved with the corresponding group of filters. For example, as shown in Figure 2.8, the input is divided into three groups. Accordingly, there are three groups of filters. The depth of channels in each input group is two, and accordingly, the depth of each filter's channels should be two. Each filter group consists of three filters, which produces three output feature maps for each input group.

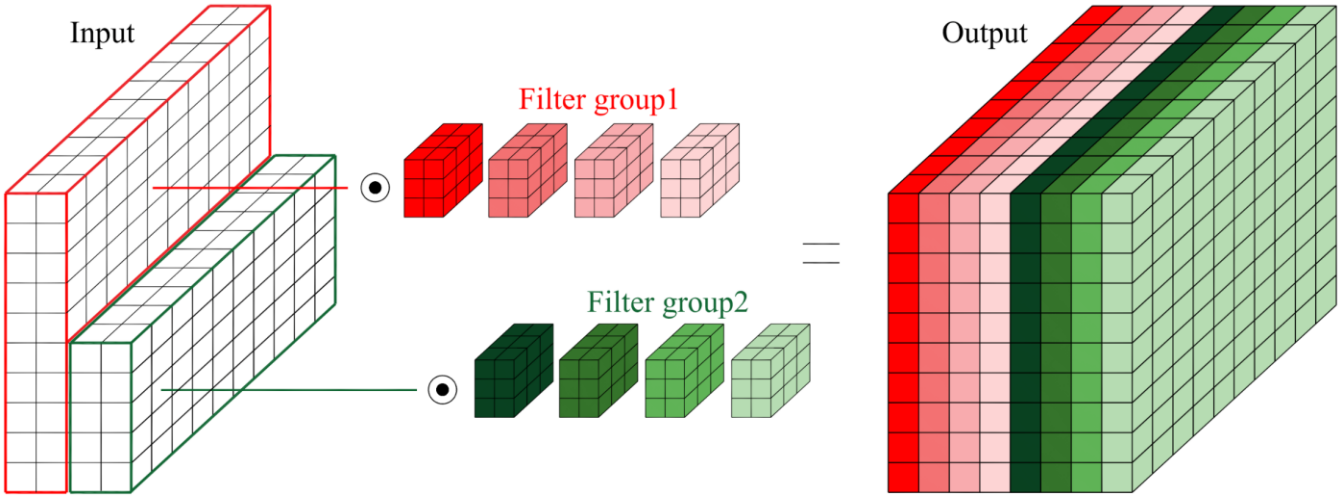


Figure 2.8: Group convolution

By setting the number of groups to the depth of input channels, the convolution becomes depth-wise convolution. For example, as shown in Figure 2.9, the input is divided into four groups. Therefore, there should be four groups of filters. The depth of each input group is one, and the depth of all filters should be one. In the presented example, the depth of the filters in each filter group consists of two filters, and the depth of the output is eight.

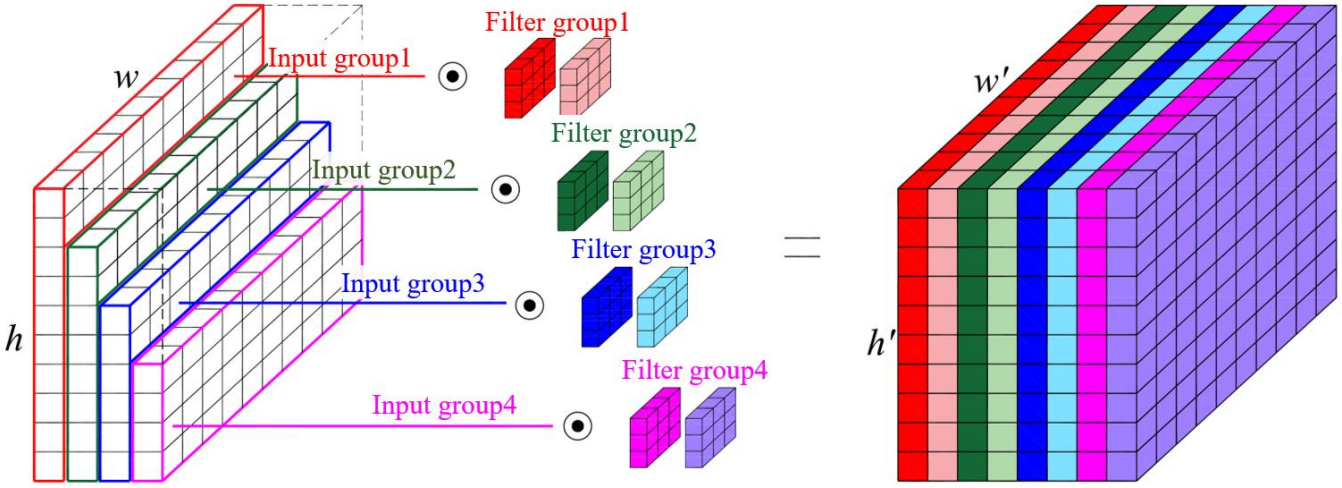


Figure 2.9: Depth-wise convolution

2.3.3 Pooling

The workflow of pooling is identical to convolution but without trainable parameters (i.e., weights and biases). Instead of extracting features by the weighted sum, the pooling operation collects features from an input. Max and average pooling operations are commonly employed in CNNs, and each of these operations are straightforward, as shown in Figure 2.10. Pooling operations perform operations that are identical to convolution (refer to Figure 2.4) with the pooling kernel $\mathcal{P} = \{P_i | \ i \in \{1, 2, \ldots, n_P\}\}$ instead of weights (i.e., trainable parameters): in the case of max pooling, P_i is one if the corresponding element in $\Psi^{(l-1)}$ is equal to max($\Psi^{(l-1)}$); otherwise, P_i is equal to 0. In the case of average pooling, P_i is equal to $1/n_p$.

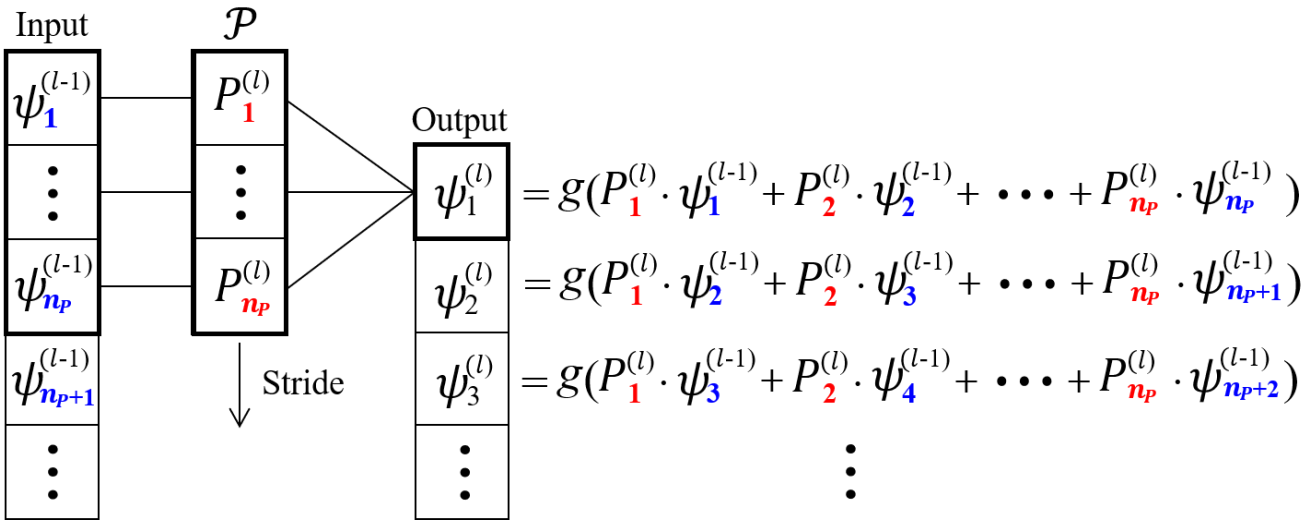
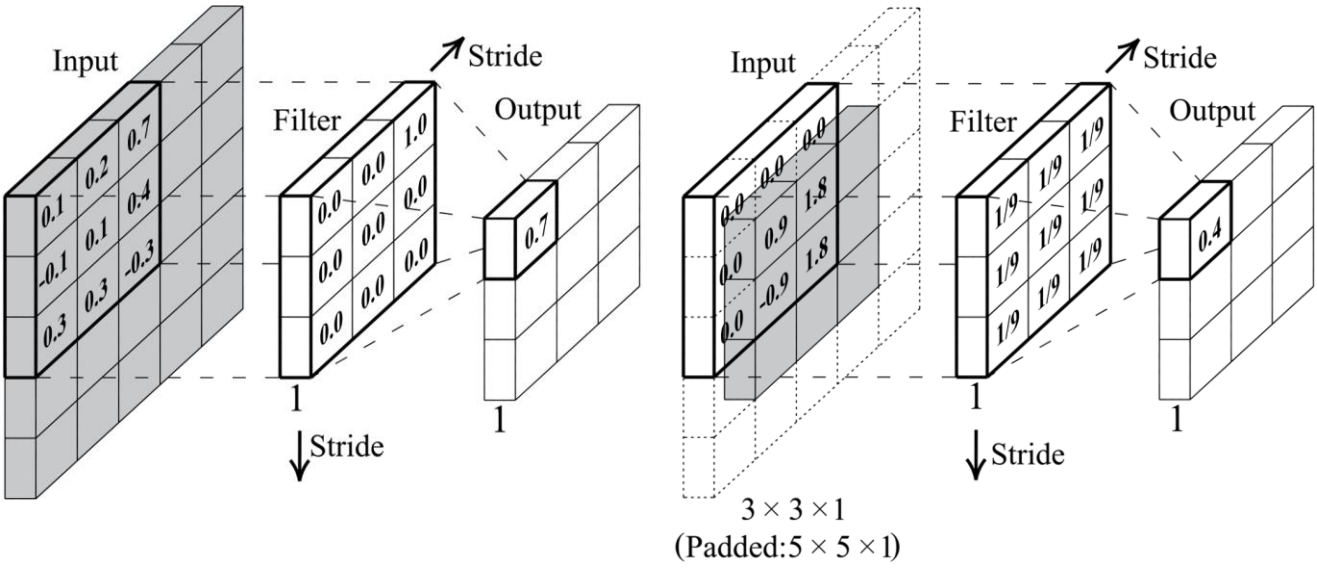


Figure 2.10: Pooling operation with 1D array

A pooling kernel also strides across the inputs, and the pooling operation can be employed in a higher-dimensional array. An example of max pooling on a 2D tensor is shown in Figure 2.11(a), where the max pooling kernel only collects the max value from the input. Padding can also be employed, as shown in Figure 2.11(b), where the type of kernel is the average pooling kernel, which takes the average

of the subarray in the input. If an input is a three-dimensional (3D) tensor, a pooling works as depth-wise convolution (refer to Chapter 2.3.2 and Figure 2.9).



(a) Max pooling without padding[2] (b) Average pooling with zero padding[3]

Figure 2.11: Pooling example with 2D array

2.4 Model optimization

In the previous section, the key operations of a CNN are discussed, where the calculation examples are shown in Figures 2.5 and Figure 2.11 with random numbers. In practice, these numbers are gradually tuned over numerous iterations until a model returns meaningful results. This process is referred to as optimizing a model, updating parameters (i.e., filters, weights, and biases), minimizing loss, training a model, etc. To update the filters of a model, the deviation between prediction and the corresponding ground truth should be measured, and a function that measures the deviation is known as a loss function,

[2] The max value within the subarray of the input is 0.7, and the corresponding pooling constant is set at 1.0, otherwise 0.0. The way of calculation is equivalent to that of Figure 2.10.
[3] The pooling size is 9 (=3×3), and all the pooling constants are set at 1/9. The way of calculation is equivalent to that of Figure 2.11.

cost function, objective function, etc. With information about the measured deviation, filters can be updated, in which an algorithm that defines how a filter is updated is referred to as an optimizer.

2.4.1 Cost and loss functions

The usage of the terminologies 'cost function' and 'loss function' in literature is intertwined. In this thesis, the terminologies are re-defined to prevent confusion as follows: the loss $\mathcal{L}$ is the function $\mathcal{F}$ that measures the distances between the prediction $y^{[i]}$ and the corresponding ground truth $\hat{y}^{[i]}$ for the $\underline{i}$-th instance of train data; and the cost function $\mathcal{J}$ aggregates all the losses calculated from the entire or several data points of a train data. Note that the bracketed superscripts utilized in this subchapter should not be confused with the parenthesized superscripts in Chapter 2.1. Let $y^{[i]} = \mathcal{H}\left(x^{[i]}; \Phi, B\right)$, where $\mathcal{H}$ is a model that performs operations (e.g., convolution, etc.; refer to Chapter 2.3) with the input *x*, set of weights Φ, and set of biases *B* of the model. The cost function $\mathcal{J}$ can be expressed by Eq. (2.3), which averages all the losses calculated from the *m* number of training instances. Note that a cost function may or may not average the corresponding loss, and it depends on configurations and preferences.

$$\mathcal{L}^{[i]} = \mathcal{F}\left(y^{[i]}, \hat{y}^{[i]}\right) \tag{2.2}$$

$$\mathcal{J} = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{[i]} \tag{2.3}$$

All the weights and biases of Φ and *B* are involved in the calculation of $\mathcal{L}$; therefore, $\mathcal{J}$ is differentiable with respect to each weight and bias. The GD facilitates this property to optimize a model, which is further discussed in Chapter 2.4.2. In this thesis, different loss functions are employed in the CNN models presented in Chapters 3 and 4, depending on the problems to be solved, such as classification and segmentation.

2.4.2 Backpropagation with gradient descent

The GD is one of the most popular algorithms for optimizing a DL model; it can be formally written by Eq. (2.4): the weight ϕ is updated ($\leftarrow$) by calculating the gradient (∇) of the cost function $\mathcal{J}$ with respect to ϕ, where the hyperparameter α is known as the learning rate, which defines how much $\nabla_\phi \mathcal{J}$ contributes to updating *w*, and $\nabla_\phi \mathcal{J}$ can be derived as (2.5). The GD is often referred to as the batch GD if *m* is equal to the total number of training instances. If *m* is smaller than the total number of training instances, the optimization is referred to as the mini-batch GD. In practice, the mini-batch GD is the most common choice due to the limitation of computational resources.

$$\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{J} \tag{2.4}$$

$$\nabla_\phi \mathcal{J} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\phi \mathcal{L}^{[i]} \tag{2.5}$$

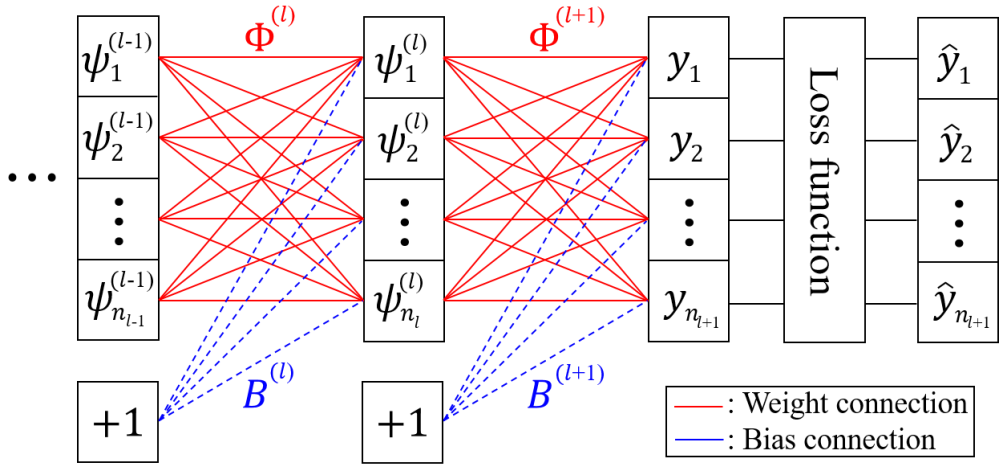


Figure 2.12: Example model with FC layers

A DL model is constructed by stacking layers. For example, assume that a model is configured with several FC layers (refer to Chapter 2.3.1), as shown in Figure 2.12. The gradients of the loss $\mathcal{L}$ with respect to each weight in the model can be calculated as follows:

$$\frac{\partial \mathcal{L}}{\partial \phi_{1,1}^{(l)}} = \frac{\partial \mathcal{L}}{\partial y_1} \times \frac{\partial y_1}{\partial \psi_1^{(l)}} \times \frac{\partial \psi_1^{(l)}}{\partial \phi_{1,1}^{(l)}} + \frac{\partial \mathcal{L}}{\partial y_2} \times \frac{\partial y_2}{\partial \psi_1^{(l)}} \times \frac{\partial \psi_1^{(l)}}{\partial \phi_{1,1}^{(l)}} + \cdots + \frac{\partial \mathcal{L}}{\partial y_{n_{l+1}}} \times \frac{\partial y_{n_{l+1}}}{\partial \psi_1^{(l)}} \times \frac{\partial \psi_1^{(l)}}{\partial \phi_{1,1}^{(l)}},$$

$$\frac{\partial \mathcal{L}}{\partial \phi_{i,1}^{(l)}} = \frac{\partial \mathcal{L}}{\partial y_1} \times \frac{\partial y_1}{\partial \psi_i^{(l)}} \times \frac{\partial \psi_i^{(l)}}{\partial \phi_{i,1}^{(l)}} + \frac{\partial \mathcal{L}}{\partial y_2} \times \frac{\partial y_2}{\partial \psi_i^{(l)}} \times \frac{\partial \psi_i^{(l)}}{\partial \phi_{i,1}^{(l)}} + \cdots + \frac{\partial \mathcal{L}}{\partial y_{n_{l+1}}} \times \frac{\partial y_{n_{l+1}}}{\partial \psi_i^{(l)}} \times \frac{\partial \psi_i^{(l)}}{\partial \phi_{i,1}^{(l)}},$$

$$\frac{\partial \mathcal{L}}{\partial \phi_{i,j}^{(l)}} = \frac{\partial \mathcal{L}}{\partial y_1} \times \frac{\partial y_1}{\partial \psi_i^{(l)}} \times \frac{\partial \psi_i^{(l)}}{\partial \phi_{i,j}^{(l)}} + \frac{\partial \mathcal{L}}{\partial y_2} \times \frac{\partial y_2}{\partial \psi_i^{(l)}} \times \frac{\partial \psi_n^{(l)}}{\partial \phi_{i,j}^{(l)}} + \cdots + \frac{\partial \mathcal{L}}{\partial y_{n_{l+1}}} \times \frac{\partial y_{n_{l+1}}}{\partial \psi_i^{(l)}} \times \frac{\partial \psi_i^{(l)}}{\partial \phi_{i,j}^{(l)}}.$$

The chain rule can also be applied to convolution layers. For example, assume that a model is configured with several convolution layers (refer to Chapter 2.3.2), as shown in Figure 2.13. The input of the *l*-th layer $\Psi^{(l-1)}$ is convolved with $\Phi^{(l)}$. The corresponding output $\Psi^{(l)}$ is the input of the (*l*+1)-th layer, which is convolved with $\Phi^{(l+1)}$. The output of (*l*+1)-th layer *y* is calculated and followed by a loss function.

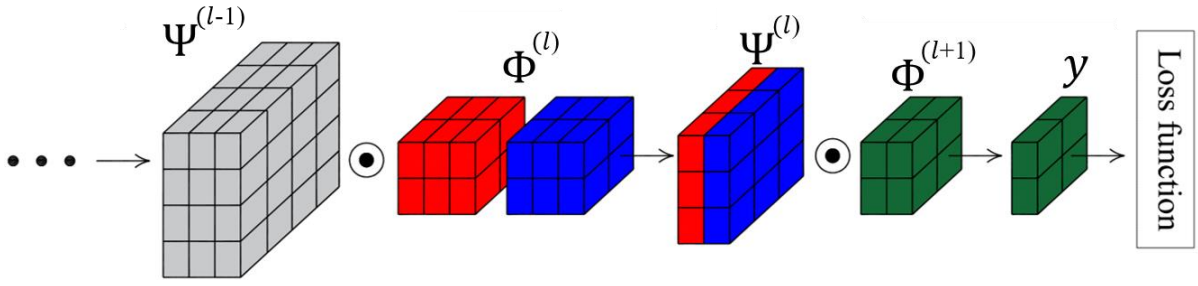


Figure 2.13: Example model with convolution layers

In the given example, the gradient with respect to $\phi_{1,1,1,1}^{(l)}$, as an example, can be derived as

$$\frac{\partial \mathcal{L}}{\partial \phi_{1,1,1,1}^{(l)}} = \frac{\partial \mathcal{L}}{\partial y_{1,1,1}} \times \frac{\partial y_{1,1,1}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial \mathcal{L}}{\partial y_{1,2,1}} \times \frac{\partial y_{1,2,1}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial \mathcal{L}}{\partial y_{2,1,1}} \times \frac{\partial y_{2,1,1}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial \mathcal{L}}{\partial y_{2,2,1}} \times \frac{\partial y_{2,2,1}}{\partial \phi_{1,1,1,1}^{(l)}},$$

where

$$\begin{aligned}\frac{\partial y_{1,1,1}}{\partial \phi_{1,1,1,1}^{(l)}} = {} & \frac{\partial y_{1,1,1}}{\partial \psi_{1,1,1}^{(l)}} \times \frac{\partial \psi_{1,1,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial y_{1,1,1}}{\partial \psi_{2,1,1}^{(l)}} \times \frac{\partial \psi_{2,1,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial y_{1,1,1}}{\partial \psi_{3,1,1}^{(l)}} \times \frac{\partial \psi_{3,1,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial y_{1,1,1}}{\partial \psi_{1,2,1}^{(l)}} \times \frac{\partial \psi_{1,2,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} \\ & + \frac{\partial y_{1,1,1}}{\partial \psi_{2,2,1}^{(l)}} \times \frac{\partial \psi_{2,2,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial y_{1,1,1}}{\partial \psi_{3,2,1}^{(l)}} \times \frac{\partial \psi_{3,2,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial y_{1,1,1}}{\partial \psi_{1,3,1}^{(l)}} \times \frac{\partial \psi_{1,3,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} + \frac{\partial y_{1,1,1}}{\partial \psi_{2,3,1}^{(l)}} \times \frac{\partial \psi_{2,3,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}} \\ & + \frac{\partial y_{1,1,1}}{\partial \psi_{3,3,1}^{(l)}} \times \frac{\partial \psi_{3,3,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}},\end{aligned}$$

similarly

$$\frac{\partial y_{1,2,1}}{\partial \phi_{1,1,1,1}^{(l)}} = \sum_{i=1}^{3} \sum_{j=1}^{3} \frac{\partial y_{1,2,1}}{\partial \psi_{i,j,1}^{(l)}} \times \frac{\partial \psi_{i,j,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}},$$

$$\frac{\partial y_{2,1,1}}{\partial \phi_{1,1,1,1}^{(l)}} = \sum_{i=1}^{3} \sum_{j=1}^{3} \frac{\partial y_{2,1,1}}{\partial \psi_{i,j,1}^{(l)}} \times \frac{\partial \psi_{i,j,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}},$$

$$\frac{\partial y_{2,2,1}}{\partial \phi_{1,1,1,1}^{(l)}} = \sum_{i=1}^{3} \sum_{j=1}^{3} \frac{\partial y_{2,2,1}}{\partial \psi_{i,j,1}^{(l)}} \times \frac{\partial \psi_{i,j,1}^{(l)}}{\partial \phi_{1,1,1,1}^{(l)}},$$

Note that each neuron or feature denoted by ψ may be accompanied by a nonlinear function. For the backpropagation of a pooling layer, the derivative of a pooling operation is the same as that of convolution because pooling performs an operation that is identical to convolution (refer to Chapter 2.3.3).

Optimization algorithms have been evolved from the GD, and some of these algorithms are discussed in Chapters 3 and 4.

2.5 Overfitting and regularization

Assume that a model is optimized with a training set over a number of iterations. The model at each iteration may have different abilities to successfully accomplish a given task but generally improves over several iterations. Accordingly, the evaluation index (e.g., accuracy) for the training set is expected to increase. However, a trained model often shows much worse performance for unseen data (e.g., validation and test data), as shown in Figure 2.14, and the model is considered overfitted to the training set because it tends to memorize the features of the training set rather than generalize them. There are several countermeasures that can mitigate the challenge. One of the prominent countermeasures is data augmentation, which is discussed in a previous subchapter (refer to Chapter 2.2.1) as a part of input processing. Others are detailed in the succeeding subchapters.

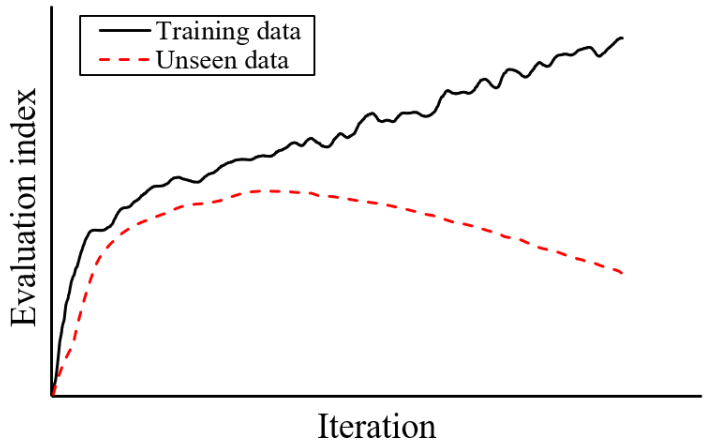


Figure 2.14: Evaluation index on training and unseen data

2.5.1 Weight decay

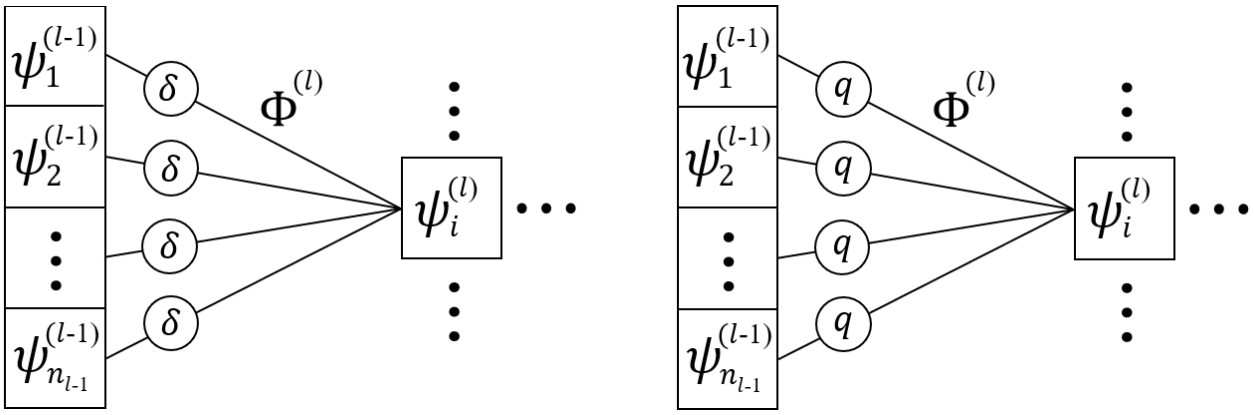
Weight decay is a technique that adjusts weights to be regularized by penalizing weights (Krogh and Hertz, 1992). This technique can be established by adding a vector norm to a cost function. In training a DL model, adding the L2-norm of weights has become a common practice, and the cost function with the L2-norm can be formally written by Eq. (2.6): the first term on the right side is equal to Eq. (2.3); the second term of the right side is the L2-norm of the model weights; ρ is the weight decay factor that defines how much weights are penalized. The derivative of the cost function (Eq. (2.6)) can be written in Eq. (2.7), and the weight w is penalized by Eq. (2.4).

$$\mathcal{J} = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}^{[i]} + \frac{\rho}{2m}\sum_{i=1}^{m}\|\Phi\|^2 \tag{2.6}$$

$$\nabla_{\phi}\mathcal{J} = \frac{1}{m}\sum_{i=1}^{m}\nabla_{\phi}\mathcal{L}^{[i]} + \rho\phi \tag{2.7}$$

2.5.2 Dropout

Dropout (Srivastava et al., 2014) is a regularization technique that disconnects the connections between input neurons and output neurons with a probability when training a model. An example of an FC unit with dropout connections is depicted in Figure 2.15(a). In the figure, the random variable $\delta \in \{0, 1\}$ is a dropout gate, which follows the Bernoulli distribution with the probability p (known as the dropout rate). The mathematical form of the feature extraction (i.e., weighted sum; refer to Chapter 2.3.1) between $\Psi^{(l-1)} \in \left\{\psi_1^{(l-1)}, \psi_1^{(l-1)}, \ldots, \psi_{n_{l-1}}^{(l-1)}\right\}$ and the i-th extracted feature at the next l-th layer $\psi_i^{(l)}$ can be expressed by Eq. (2.8) (Srivastava et al., 2014). If p is set to 0.5, half of the connections are disconnected by the equation. The dropout rate is considered a hyperparameter that is to be empirically set.



(a) Dropout in training (b) Dropout in testing

Figure 2.15: Dropout

$$\psi_i^{(l)} = \sum_{j=1}^{n_{l-1}} \delta_j^{(l)} \cdot \phi_{n_l,j}^{(l)} \cdot \psi_j^{(l-1)} \tag{2.8}$$

In backpropagation, the gradients at the example layer with respect to each weight can be calculated as follows:

$$\frac{\partial \mathcal{L}}{\partial \phi_{n_l,j}^{(l)}} = \cdots \times \delta_j \cdot \frac{\partial \psi_{n_l}^{(l)}}{\partial \phi_{n_l,j}^{(l)}} \times \cdots.$$

Once a model is optimized, the input of the layer with the dropout gate is s by $q(1-p)$, as shown in Figure 2.15(a), and Eq. (2.8) in testing the model can be re-written as Eq. (2.9) (Srivastava et al., 2014).

$$\psi_{n_l}^{(l)} = \sum_{j=1}^{n_{l-1}} q_j^{(l)} \cdot \phi_{n_l,j}^{(l)} \cdot \psi_j^{(l-1)} \tag{2.9}$$

Dropout is intended to prevent co-adaptation[4], and an FC layer (refer to Chapter 2.3.1) tends to address the issue (Gumbira and Kożuszek, 2018) more than a convolution layer. A convolution filter consists of a set of shared weights (refer to Chapter 2.3.2), while an FC layer has dense connections between input neurons and output neurons.

2.6 Activation functions

The series of convolution and pooling conduct linear operations, and constructing a model by combinations of linear operations is simply repeating linear operations. However, the problems to be solved by DL are highly nonlinear. Therefore, certain operations that introduce nonlinearity should be included in a DL model; these operations are referred to as activation functions. The simplest activation function is a binary step function, as shown in Figure 2.16, where any input greater than zero is either activated (=1) or not activated (=0). However, this type of function is mostly unusable for training a DL because no useful gradients exist, and a model cannot be optimized without gradients (refer to Chapter 2.4.2).

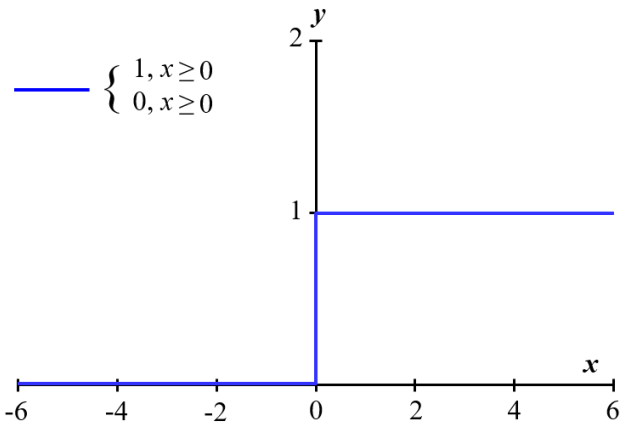


Figure 2.16: Binary step function

[4] a phenomenon that a model tends to build strong connections between certain neurons, and other neurons do not or less contribute to updating weights.

Hyperbolic tangent ($\tanh(x)$) and sigmoid ($(1+e^{-x})^{-1}$) functions can be the alternatives of a step function, as depicted in Figure 2.17. The corresponding gradients are plotted in Figure 2.17. As the figure shows, both nonlinear functions are continuously differentiable, and the functions have been extensively applied as activation functions in NN.

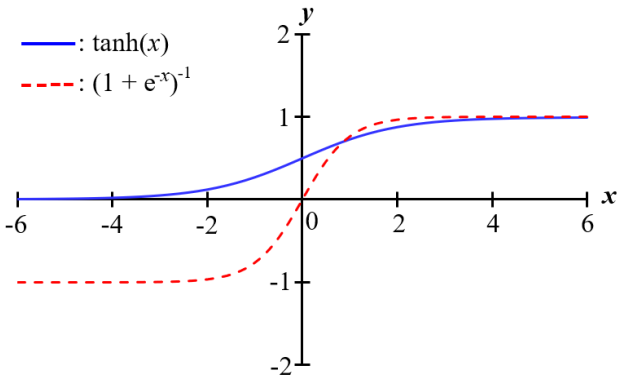


Figure 2.17: Tanh and sigmoid

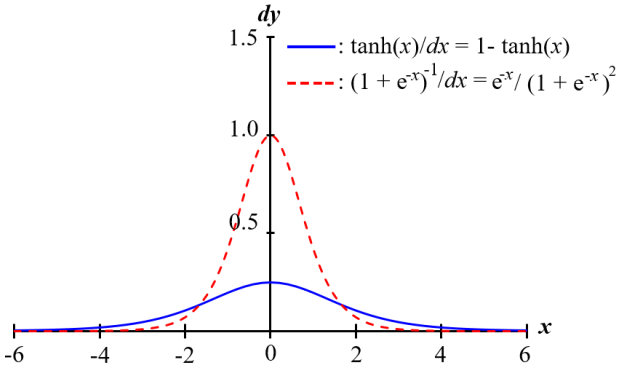


Figure 2.18: Gradient of tanh and sigmoid

However, if the variance of an input data is not within a certain range, then the gradient values are too small, and a model cannot be updated. For example, as shown in Figure 2.18, the gradients of the tanh and sigmoid functions at $x > +2$ and $x < -2$ become smaller and are almost zero at $x = \pm 6$. This result means that $\nabla_{\phi}\mathcal{J}$ of Eq. (2.4) is almost zero, and a ϕ is updated very slowly if an input has a high variance. In addition, small gradients cause a more serious issue in a model with a deep architecture because small gradients are multiplied by the chain rule. Consequently, the gradient at a deep layer will be almost zero, and the entire model will not be optimized. Nair and Hinton (2010) highlighted the vanishing gradient issue and proposed a rectified linear unit (ReLU), which is shown in Figure 2.19; the corresponding gradients are plotted in Figure 2.20. As shown in the figures, the ReLU has nonlinearity and is continuously differentiable over all x, and the gradients are not vanishing unless $x < 0$. Due to these excellent properties, the ReLU has become one of the standard activations in DL. Several variations of ReLU, such as exponential linear units (ELUs) (Clevert et al., 2015), scaled-exponential linear units

(SELUs) (Klambauer et al., 2017), etc., have been proposed. There have been no distinctively better activations between the variations in the ReLU. Ramachandran et al. (2017) proposed Swish activation and claimed that it outperformed all the variations in the ReLU. However, Eger et al., (2019) showed contrast results that indicate that the superiority of these activation functions is dependent on trials and tasks. The activations that are prevalent in modern DL models are summarized in Table 2.1.

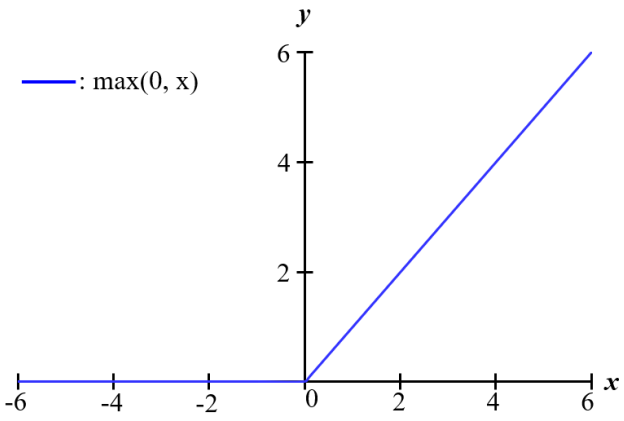


Figure 2.19: ReLU

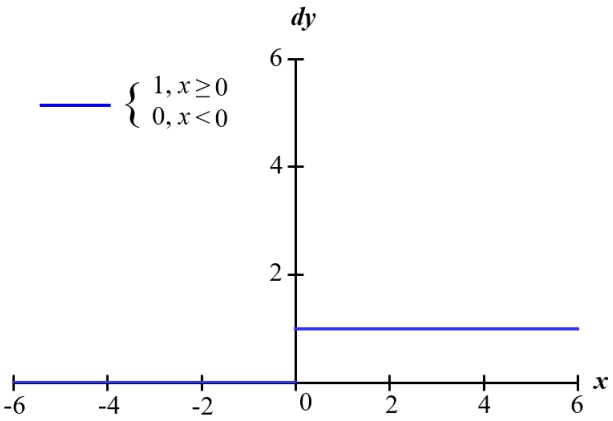


Figure 2.20: Derivative of ReLU

Table 2.1: Activation functions

Activation	Formula	Derivative	Remark
Tanh	$\tanh(x)$	$1-\tanh^2(x)$	–
Sigmoid	$\sigma(x)=\frac{1}{1+e^{-x}}$	$\sigma(x)\cdot\left(1-\sigma(x)\right)$	–
ReLU (Nair and Hinton, 2010)	$\begin{cases} x & \text{for } x>0 \\ 0 & \text{for } x\leq 0 \end{cases}$	$\begin{cases} 1 & \text{for } x>0 \\ 0 & \text{for } x\leq 0 \end{cases}$	–
Leaky ReLU (Maas et al., 2013)	$\begin{cases} x & \text{for } x\geq 0 \\ \bar{\alpha}\cdot x & \text{for } x<0 \end{cases}$	$\begin{cases} 1 & \text{for } x\geq 0 \\ \bar{\alpha} & \text{for } x<0 \end{cases}$	$\bar{\alpha}$ = constant $(0<\bar{\alpha}<1)$
Parametric ReLU (He et al., 2015)			$\bar{\alpha}$ = trainable parameter
ELU (Clevert et al., 2015)	$\begin{cases} x & \text{for } x\geq 0 \\ \bar{\alpha}\cdot(e^x-1) & \text{for } x<0 \end{cases}$	$\begin{cases} x & \text{for } x\geq 0 \\ \bar{\alpha}\cdot(e^x-1) & \text{for } x<0 \end{cases}$	$\bar{\alpha}$ = constant (>0) or trainable parameter
SELU (Klambauer et al., 2017)	$\bar{\lambda}\cdot\begin{cases} x & \text{for } x\geq 0 \\ \bar{\alpha}\cdot(e^x-1) & \text{for } x<0 \end{cases}$	$\bar{\lambda}\cdot\begin{cases} 1 & \text{for } x\geq 0 \\ \bar{\alpha}\cdot(e^x) & \text{for } x<0 \end{cases}$	$\bar{\lambda}$ = constant (=1.0507) $\bar{\alpha}$ = constant (=1.6733)
Swish (Ramachandran et al., 2017)	$x\cdot\sigma(\bar{\beta}\cdot x)$	$\bar{\beta}\cdot\sigma(x)+\sigma(\bar{\beta}\cdot x)\cdot\left(1-\bar{\beta}\cdot\sigma(x)\right)$	$\bar{\beta}$ = constant $\left(\geq 1\right)$ or trainable parameter

2.7 Challenges of training a deep learning model

Training a DL model is challenging. Two main reasons are exploding gradients and vanishing gradients, and a simple example may provide intuitions about both reasons. For example, assume that a DL model is configured, as shown in Figure 2.21, where each layer has a single feature and single weight. In this example, biases are omitted for simplicity. Note that the extracted features denoted by ψ may accompany an activation function (refer to Chapter 2.6).

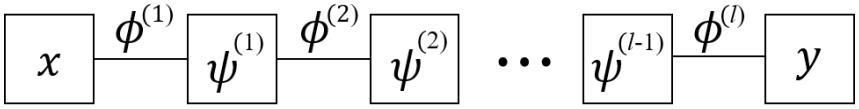


Figure 2.21: Simple deep learning model

Applying the backpropagation to update $\phi^{(1)}$ of the model, as an example, yields the following derivative:

$$\frac{\partial \mathcal{L}}{\partial \phi^{(1)}} = \frac{\partial \mathcal{L}}{\partial y} \times \frac{\partial y}{\partial \psi^{(l-1)}} \times \cdots \times \frac{\partial \psi^{(2)}}{\partial \psi^{(1)}} \times \frac{\partial \psi^{(1)}}{\partial \phi^{(1)}},$$

where if saturating activation functions (e.g., sigmoid) are employed in the model, each derivative (e.g., $\frac{\partial \psi^{(1)}}{\partial \phi^{(1)}}$, $\frac{\partial \psi^{(2)}}{\partial \psi^{(1)}}$, …) returns small gradient values (e.g., 0.2 or less for sigmoid). Accordingly, $\frac{\partial \mathcal{L}}{\partial \phi^{(1)}}$ may return a very small gradient value or is often vanished. Even if a non-saturating activation function, such as the ReLU, is applied, the gradients will remain vanished if the input of a layer has negative values. Regardless of the type of activation function, gradients can be exploded, which can be observed by examining one of the previous derivatives. For example,

$$\frac{\partial \psi^{(2)}}{\partial \psi^{(1)}} = \frac{\partial g(\psi^{(1)} \times \phi^{(1)})}{\partial (\psi^{(1)} \times \phi^{(1)})} \times \frac{\partial \left(\psi^{(1)} \times \phi^{(1)}\right)}{\partial \psi^{(1)}} = \frac{\partial g(\psi^{(1)} \times \phi^{(1)})}{\partial (\psi^{(1)} \times \phi^{(1)})} \times \phi^{(1)},$$

where g is an activation function. If $\phi^{(1)}$ is relatively and substantially greater than $\frac{\partial g(\psi^{(1)} \times \phi^{(1)})}{\partial (\psi^{(1)} \times \phi^{(1)})}$, $\frac{\partial \psi^{(2)}}{\partial \psi^{(1)}}$ can be greater than one, and multiplying gradient values greater than one may return a large value of $\frac{\partial \mathcal{L}}{\partial \phi^{(1)}}$ and can be exploded.

2.8 Batch normalization

To overcome the challenges of training a DL model, Ioffe and Szegedy (2015) proposed batch normalization (BatchNorm), which is extensively applied in modern DL models. In Chapter 2.2.2, the standardization of raw data is explained, and BatchNorm is designed to conduct similar operations for the input data of any layer. The mean (μ) and variance (Var) of a set of features (Ψ) over a mini-batch at a layer are calculated by Eq. (2.10) and Eq. (2.11), respectively (Ioffe and Szegedy, 2015). In the context of NN (refer to Chapter 2.3), m is the mini-batch size (i.e., the number of instances in the mini-batch). In the context of CNN (refer to Chapter 2.3), m is equal to the mini-batch size × height of Ψ × width of Ψ. Using the mean and variance, Ψ is standardized (i.e., zero mean and unit variance), and $\widehat{\Psi}$ can be calculated by Eq. (2.12) (Ioffe and Szegedy, 2015), in which ε is a small constant (e.g., 10^{-8}, etc.) to prevent zero division. Finally, $\widehat{\Psi}$ is further scaled and shifted by the trainable parameters γ and β, respectively, in Eq. (2.13) (Ioffe and Szegedy, 2015). Note that the role of β functions as the bias of a model, and the biases of a FC layer and a convolution layer can be accordingly omitted if a BatchNorm is followed.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} \Psi^{[i]}, \tag{2.10}$$

$$Var = \frac{1}{m}\sum_{i=1}^{m} \left(\Psi^{[i]} - \mu\right)^2, \tag{2.11}$$

$$\widehat{\Psi}^{[i]} = \frac{\Psi^{[i]} - \mu}{\sqrt{Var + \varepsilon}}, \tag{2.12}$$

$$\widetilde{\Psi}^{[i]} = \gamma\widehat{\Psi}^{[i]} + \beta. \tag{2.13}$$

In backpropagation, by the chain rule, the gradient with respect to γ and β can be calculated by Eq. (2.14) and Eq. (2.15) (Ioffe and Szegedy, 2015), respectively.

$$\frac{\partial \mathcal{J}}{\partial \beta} = \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} \times \frac{\partial \widetilde{\Psi}^{[i]}}{\partial \beta} = \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} \times \frac{\partial \left(\gamma \widehat{\Psi}^{[i]} + \beta\right)}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}}, \tag{2.14}$$

$$\frac{\partial \mathcal{J}}{\partial \gamma} = \frac{\partial \mathcal{J}}{\partial \gamma \widetilde{\Psi}^{[i]}} \times \frac{\partial \gamma \widetilde{\Psi}^{[i]}}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} \times \widehat{\Psi}^{[i]}. \tag{2.15}$$

The gradients of $\mathcal{J}$ with respect to $\Psi^{[i]}$ should also be derived to calculate the gradients with respect to a weight of the preceding layer, and the corresponding gradients can be derived as Eq. (2.16) (Ioffe and Szegedy, 2015).

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial \Psi^{[i]}} &= \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{\partial \widehat{\Psi}^{[i]}}{\partial \Psi^{[i]}} + \frac{\partial \mathcal{J}}{\partial \mu} \times \frac{\partial \mu}{\partial \Psi^{[i]}} + \frac{\partial \mathcal{J}}{\partial Var} \times \frac{\partial Var}{\partial \Psi^{[i]}} \\ &= \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{1}{\sqrt{Var + \beta}} + \frac{\partial \mathcal{J}}{\partial \mu} \times \frac{1}{m} + \frac{\partial \mathcal{J}}{\partial Var} \times \frac{2}{m}\left(\Psi^{[i]} - \mu\right) \end{aligned} \tag{2.16}$$

where

$$\frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} = \frac{\partial \mathcal{J}}{\partial \gamma \widetilde{\Psi}^{[i]}} \times \frac{\partial \gamma \widetilde{\Psi}^{[i]}}{\partial \widetilde{\Psi}^{[i]}} = \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} \times \frac{\partial \widetilde{\Psi}^{[i]}}{\partial \gamma \widetilde{\Psi}^{[i]}} \times \gamma = \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} \times \frac{\partial \left(\gamma \widehat{\Psi}^{[i]} + \beta\right)}{\partial \gamma \widetilde{\Psi}^{[i]}} \times \gamma = \frac{\partial \mathcal{J}}{\partial \widetilde{\Psi}^{[i]}} \times \gamma,$$

$$\frac{\partial \mathcal{J}}{\partial Var} = \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{\partial \widehat{\Psi}^{[i]}}{\partial Var} = \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{\partial \left(\frac{\Psi^{[i]} - \mu}{\sqrt{Var + \varepsilon}} \right)}{\partial Var} = \sum_{i=1}^{m} \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \left(\Psi^{[i]} - \mu \right) \times \frac{\partial (Var + \varepsilon)^{-\frac{1}{2}}}{\partial Var}$$

$$= \sum_{i=1}^{m} \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \left(\Psi^{[i]} - \mu \right) \times -\frac{1}{2} \times (Var + \varepsilon)^{-\frac{3}{2}},$$

$$\frac{\partial \mathcal{J}}{\partial \mu} = \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{\partial \Psi^{[i]}}{\partial \mu} + \frac{\partial \mathcal{J}}{\partial Var} \times \frac{\partial Var}{\partial \mu} = \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{\partial \left(\frac{\Psi^{[i]} - \mu}{\sqrt{Var + \varepsilon}} \right)}{\partial \mu} + \frac{\partial \mathcal{J}}{\partial Var} \times \frac{\partial \left(\frac{1}{m} \sum_{i=1}^{m} \left(\Psi^{[i]} - \mu \right) \right)^2}{\partial \mu}$$

$$= \sum_{i=1}^{m} \frac{\partial \mathcal{J}}{\partial \widehat{\Psi}^{[i]}} \times \frac{-1}{\sqrt{Var + \beta}} + \frac{\partial \mathcal{J}}{\partial Var} \times -\frac{2}{m} \sum_{i=1}^{m} \left(\Psi^{[i]} - \mu \right).$$

In the testing phase, images are individually (i.e., mini-batch size of one) tested by the trained model, and $\widehat{\Psi}$ (Eq. (2.11) cannot be correctly calculated because the mean and variance are only calculated for a single instance. To overcome it, moving mean (μ') and moving variance (Var') are also calculated in training by Eqs. (2.17) and (2.18), respectively. λ is known as momentum constant, and 0.99 is a common choice for Eqs. (2.17) and (2.18). μ' and Var' are used instead of μ and Var in the testing phase.

$$\mu' = \lambda \times \mu' + (1 - \lambda) \times \mu, \tag{2.17}$$

$$Var' = \lambda \times Var' + (1 - \lambda) \times Var. \tag{2.18}$$

The true attributes of the BatchNorm are not clearly investigated. The creators of BatchNorm (Ioffe and Szegedy, 2015) claimed that the BatchNorm is effective because it mitigates internal-covariate shift[5], but Santurkar et al. (2018) claimed that BatchNorm is irrelevant to internal-covariate shift.

[5] When learnable parameters of a layer change during training, the input of the subsequent layers also change.

2.9 Constructing a deep learning model

A deep learning model can be constructed by stacking the explained operations in Chapters 2.2 - 2.8. For example, LeNet-5 (LeCun et al., 1998) is one of the earliest CNN models that consist of two standard convolution layers (refer to Chapter 2.3.2), two pooling layers (refer to Chapter 2.3.3), and a fully connected layer (refer to Chapter 2.3.1), as shown in Figure 2.22. Each of the standard convolutions accompanies 5×5 filters followed by the hyperbolic tangent as an activation (refer to Chapter 2.6) to introduce nonlinearity into the CNN model. Each of the pooling layers accompanies the stride (refer to Chapter 2.3.2) of two for reducing the spatial dimension of feature maps. The model is purposed to classify images of hand-written digits from zero to nine; therefore, the output of the model has ten elements.

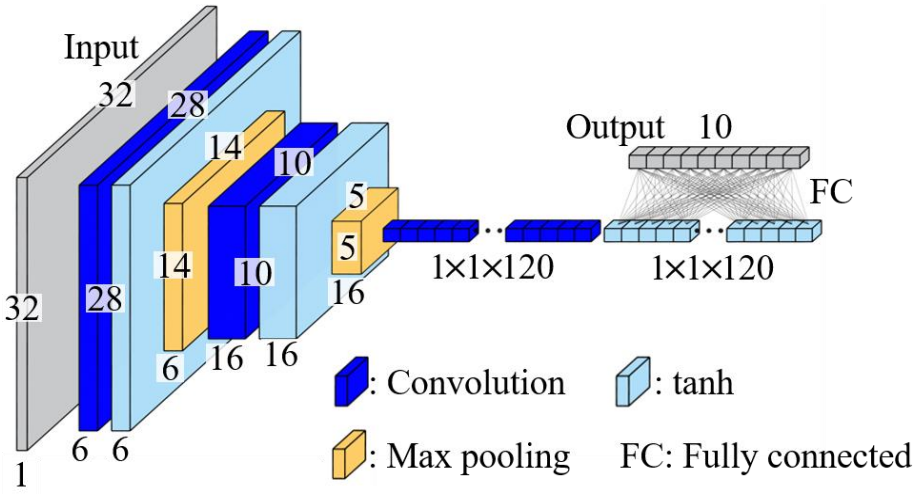


Figure 2.22: LeNet-5 architecture

There are no strict rules in building a DL model, and researchers should experimentally configure DL models. Chapter 3 introduces a unique DL model for classifying small regions of images into two classes. The model involved a fully connected layer (refer to Chapter 2.3.1), standard convolution (refer to Chapter 2.3.2), max pooling (refer to Chapter 2.3.3), gradient descent (refer to Chapter 2.4.2), weight decay (refer to Chapter 2.5.1), dropout (refer to Chapter 2.5.2), ReLU (refer to Chapter 2.6), and the Batch

normalization (refer to Chapter 2.8). In addition to the enumerated techniques, the momentum optimization and the cross-entropy loss were used to develop the DL model, and the details are given in Chapter 3. In Chapter 4, another unique DL model for segmenting damage features from images is introduced. The model involves standard convolution (refer to Chapter 2.3.2), depth-wise convolution (refer to Chapter 2.3.2), gradient descent (refer to Chapter 2.4.2), weight decay (refer to Chapter 2.5.1), dropout (refer to Chapter 2.5.2), ReLU (refer to Chapter 2.6), and the Batch normalization (refer to Chapter 2.8). Additionally, the adaptive momentum algorithm, separable convolution, and mean intersection-over-union loss were used, and those are detailed in Chapter 4.

References

Ian Goodfellow, Yoshua Bengio, & Aaron Courville (2016). *Deep learning*. MIT Press.

Clevert, D.-A., Unterthiner, T. & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus), *arXiv preprint arXiv:1511.07289*.

Eger, S., Youssef, P. & Gurevych, I. (2019). Is it time to swish? comparing deep learning activation functions across NLP tasks, *arXiv preprint arXiv:1901.02671*.

Gumbira, A., & Kozuszek, R. (2018). Does fragile co-adaptation occur in small datasets? *2018 Baltic URSI Symposium (URSI)*. DOI: 10.23919/ursi.2018.8406745

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/iccv.2015.123

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv preprint arXiv:1502.03167*.

Klambauer, G., Unterthiner, T., Mayr, A. & Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, 971-980.

Krogh, A. & Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, 950-957.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. *Lecture Notes in Computer Science*, 9–48. DOI: 10.1007/978-3-642-35289-8_3

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. DOI: 10.1109/5.726791

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning (ICML) Workshop on Deep Learning for Audio, Speech, and Language Processing*.

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807-814.

Pal, K. K., & Sudeep, K. S. (2016). Preprocessing for image classification by convolutional neural networks. *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 1778-1781. DOI: 10.1109/rteict.2016.7808140

Ramachandran, P., Zoph, B. & Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Santurkar, S., Tsipras, D., Ilyas, A. & Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, 2483-2493.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), Dropout: a simple way to prevent neural networks from overfitting, *The journal of machine learning research*, *15*(1), 1929-1958.

Chapter 3 Deep Learning Application: Crack Detection

Summary

This chapter introduces a method for detecting concrete cracks from digital images using a combination of a deep learning (DL) model and a sliding-window scheme. The deep learning model was developed for classifying small images (256 × 256 pixels) into the crack or non-crack category. While the DL model showed satisfactory results for the testing process, it was incapable of localizing crack features on images. The sliding-window scheme was combined with the DL model to make up for this deficiency. A comparative study between the proposed model and IPAs (e.g., Canny edge detector and Sobel edge detector) was conducted, and the corresponding results showed reliable performances on images with a wide array of changes in luminance conditions that could not be addressed by the IPAs.

Chapter 3 has been reproduced with modifications to figures and formatting from Cha et al. (2017). This chapter has been reprinted with permission from the copyright holder, John Wiley & Sons, Inc.

3.1 Introduction

Vision-based damage detection using IPAs are unlikely applicable in uncontrolled environments because IPAs are incapable of selectively extracting features of objects to be detected (refer to Chapter 1.1). An adaptive feature extractor is preferable in such cases. A CNN can automatically extract features (refer to Chapter 2). Using this property of CNNs, a method for detecting concrete cracks was proposed by Cha et al. (2017). The overall flow of the method is shown in Figure 3.1. The scope of this work includes configuring a DL model (refer to Chapter 3.2), building a dataset (refer to Chapter 3.3), training the DL model (refer to Chapter 3.6), and localizing cracks on large images (refer to Chapter 3.7) with the sliding-window method.

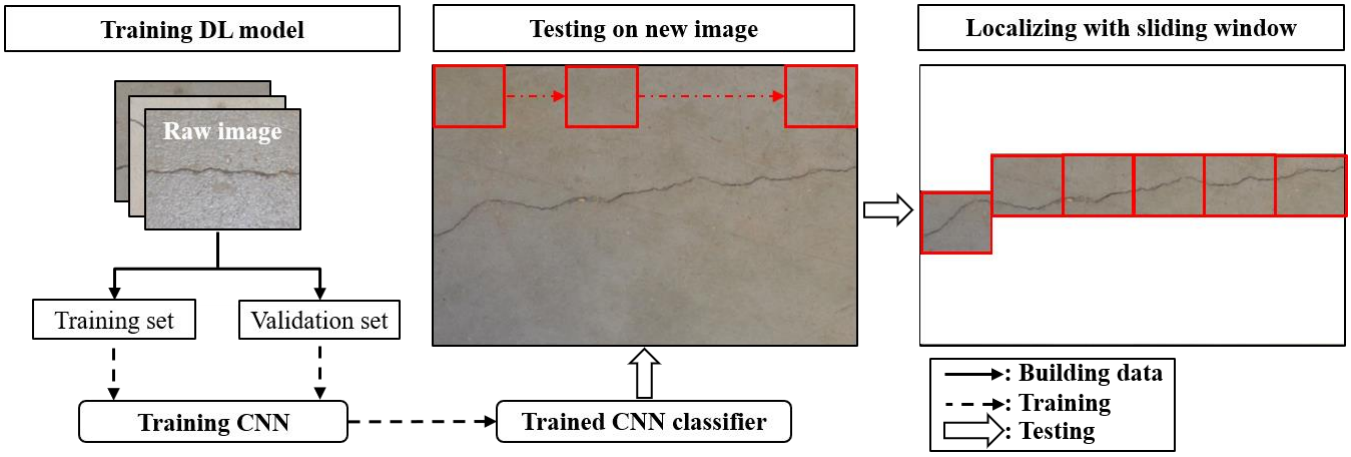


Figure 3.1: Schematic diagram of the proposed method [reproduced from Cha et al. (2017)]

3.2 Architecture configuration

A CNN model for classifying images into the crack and non-crack categories was configured, and its architecture is illustrated in Figure 3.2. The proposed model was designed to accept digital images of 256 × 256 pixels. Convolution (refer to Chapter 2.3.2), BatchNorm (refer to Chapter 2.8), and max pooling (refer to Chapter 2.3.3) operations were sequentially arranged, and a stride of 2 was applied to each

convolution and max pooling. Consequently, the feature map size was reduced to 1 × 1, and the layer was designed to have 96 features. Thereafter, the features were activated by the ReLU (refer to Chapter 2.6) followed by an FC layer. The connections in the FC layer were randomly disconnected with the dropout (refer to Chapter 2.5.2) rate of 0.5 during training. The last layer of the CNN model had two feature elements, because the proposed model was designed for binary classification. At the beginning of the training, all the weight values were initialized to obtain a normal distribution with zero mean and a standard deviation of 0.01 (refer to Chapter 2.4). Cross-entropy with the Softmax function was used as the loss function (refer to Chapter 2.4.1), as detailed in Chapter 3.4. Convolution and pooling layers are often accompanied with paddings (refer to Chapter 2.3.2) in modern CNN architectures, but such paddings were not applied in the proposed model, because this implementation does not require the maintaining of spatial dimensions. The detailed properties of each layer are summarized in Table 3.1.

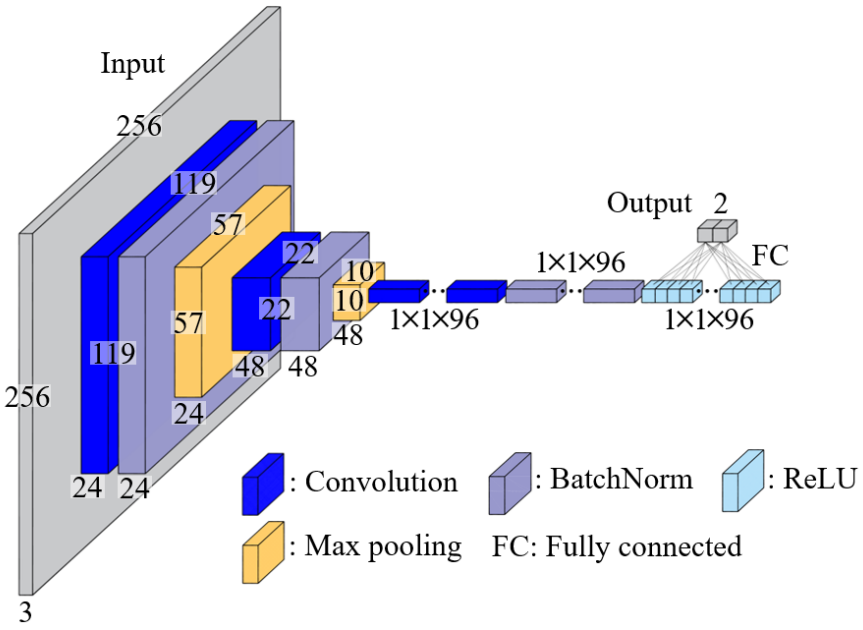


Figure 3.2: Overall architecture [reproduced from Cha et al. (2017)]

Table 3.1: Model architecture summary [Cha et al. (2017)]

Operation	Kernel size	Filter number	Stride	Feature map size
Input	-	-	-	256×256×3
Convolution	20×20×3	24	2	119×119×24
BatchNorm	-	-	-	119×119×24
Max pooling	7×7×1	-	2	57×57×24
Convolution	15×15×24	48	2	22×22×48
BatchNorm	-	-	-	22×22×48
Max pooling	4×4×1	-	2	10×10×48
Convolution	10×10×48	96	2	1×1×96
BatchNorm	-	-	-	1×1×96
ReLU	-	-	-	1×1×96
FC	1×1×96	2	-	1×1×2

3.3 Dataset generation

A dataset was manually built to train and test the CNN model (refer to Chapter 3.2). Images were taken around the University of Manitoba with a hand-held DSLR camera (Nikon D5200). The distances to concrete surfaces from the camera were approximately 1.0 to 1.5 m, but a few images were intentionally taken within 0.1 m for testing. The luminance intensity of each image was widely varied. Three hundred nine high-resolution images (e.g., 6000×4000, 4865 × 3072, etc.) were used for training and validation, and 54 high-resolution images were used for testing the trained model. The proposed model was configured to have an FC network, and the trained model could only classify images of certain sizes. To overcome this limitation, a simple sliding-window technique was employed to enable the proposed model

to test large image sizes. Accordingly, the training images were cropped to the size of the sliding window. The cropping size was fixed at 256 × 256 pixels for the following reasons: a number of cropped images had features that could not be distinguished as either cracks and elongated features (e.g., scratches), and images cropped to a greater extent reduced the informativeness of the testing result. Several representative images are shown in Figure 3.3: these images were taken under different environments, and the cropped images had sharp and blurry features, and regions with high luminance. Each of the cropped images were annotated as either crack or non-crack.

(a) Sharp (b) Blurry (c) High luminance

Figure 3.3: Representative training images [Cha et al., (2017)]

However, a number of cropped images had cracks on the edges, as shown in Figure 3.4, and were not included in either the crack or the non-crack category for the following reasons: the spatial dimensions of these images were reduced layer by layer, and such features had a much lower chance of being captured by the filters in a higher layer compared to those in the middle of the image spaces. Moreover, it was not possible to identify whether those features were a part of an actual crack, which would have likely led to poor annotations. Last, it would not be possible to check if the trained model classified such images correctly. The above-mentioned difficulties could also be tackled by the simple sliding-window technique, which is detailed in Chapter 3.7.

Figure 3.4: Disregarded images [Cha et al., (2017)]

The images presented in this subchapter are shown in their native RGB color maps. However, when the images were fed into the model, the channel-wise mean values were subtracted (refer to Chapter 2.2.2). This preprocessing was intended to shift the distribution of the input images to zero, which helps in DL model optimization (LeCun et al., 2012).

3.4 Cross-entropy loss with the Softmax

A loss function should be employed to optimize a CNN model. Accordingly, the cross-entropy loss ($\mathcal{L}_{ce}$) in Eq. (3.1) was chosen for the proposed model. $\hat{y}$ denotes the one-hot-encoded true labels, p is the Softmax value calculated using Eq. (3.2), which is considered as a discrete probability distribution because the function exponentially normalizes *y* for *N* number of classes (2 in this implementation); therefore $p \in (0, 1)$. *k* is introduced to clarify that the denominator is independent from the numerator of the function.

$$\mathcal{L}_{ce} = -\sum \hat{y} \log(p) \tag{3.1}$$

$$p = \frac{e^{y_j}}{\sum_{k=1}^{N} e^{y_k}} \qquad \forall_j \in \{1, ..., N\} \tag{3.2}$$

3.5 Gradient descent with momentum algorithm

A batch GD was designed to update weights by calculating the gradient of a cost function with respect to each weight from an entire dataset (refer to Chapter 2.4.2). However, a batch GD is not applicable in DL models in most cases, because a DL model contains too many weights for calculation, and a large amount of memory is required to perform gradient calculations for a whole dataset. Alternatively, a mini-batch GD was chosen to optimize the CNN model. In addition to the optimizer, the momentum algorithm was considered, as it is known to accelerate the convergence of training. Figure 3.5 demonstrates how the weights (ϕ_1 and ϕ_2) of a model are updated, wherein the center of each plot is the global minimum of $\mathcal{J}$. As shown in the figure, the GD with momentum updates the weights more efficiently, the intuition being that in this case, the GD considers the previous step of the weight update (v_1) to perform the current step of the weight update (v_2), and the previous step is scaled by a constant (λ), as shown in Figure 3.5 (right).

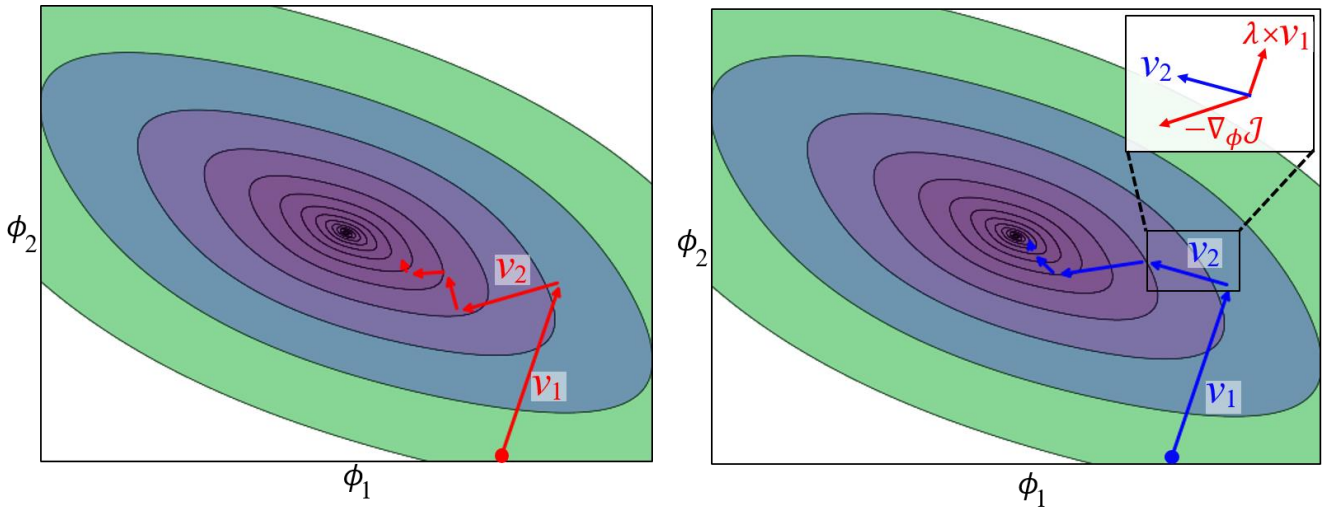


Figure 3.5: GD without momentum (left) vs GD with momentum (right)

Formally, the gradient updating procedure can be expressed by Eqs. (3.3) and (3.4), where α is the learning rate (refer to Chapter 2.4.2), $\nabla_\phi \mathcal{J}$ is the derivative of a cost function with respect to a weight (refer to Chapter 2.4.2), and λ is the momentum constant defining to what extent the previous update

(v_{t-1}) is important to update weight ϕ at current step t. The hyperparameters used in the training are detailed in Chapter 3.6.1.

$$v_t = \lambda v_{t-1} - \nabla_\phi \mathcal{J} \tag{3.3}$$

$$\phi_t \leftarrow \phi_t + \alpha v_t \tag{3.4}$$

3.6 Model training

The proposed model was coded using the MatConvNet (Vedaldi and Lenc, 2015) and deployed to a workstation (CPU: Intel Xeon E5-2650 v3 @2.3 GHz, RAM: 64 GB, and GPU: Nvidia Geforce Titan X × 2ea).

3.6.1 Hyperparameters

A number of trials was required to find the set of hyperparameters because no guidelines or precedents existed for configuration in such a case. Using a small and decreasing learning rate is a common practice (Wilson and Martinez, 2001), and the learning rate α was scheduled to be logarithmically decreased, as seen in Eq. (3.5). Note that "epoch" is a practical term used in model training. It indicates how many times an entire dataset was fed into a model. For example, suppose 100 instances are set as the mini-batch out of 1000 instances. Then, ten iterations are required to update the weights of the model within an epoch. In Eq. (3.5), lr_1 and lr_2 define the maximum and minimum learning rates within a total number of epochs, respectively. In this implementation, the total number of epochs was set at 60, while the maximum and minimum learning rates were set at 10^{-2} and 10^{-6}, as shown in Figure 3.6. The momentum λ and weight decay ρ (refer to Chapter 2.5.1) were set at 0.9 and 10^{-4}, respectively. The mini-batch size was set at 100 over all the training iterations.

$$\alpha_{\text{epoch}} = 10^{\alpha_{\text{epoch-1}} - \frac{lr_1 - lr_2}{\text{total epoch}}} \tag{3.5}$$

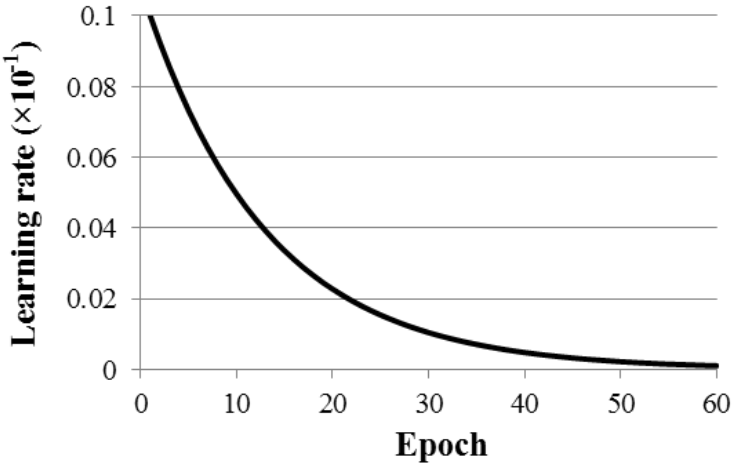


Figure 3.6: Scheduled learning rate [Cha et al., (2017)]

3.6.2 Training results

In accordance with the explained details (refer to Chapters 3.4 to 3.6.1), the proposed model was trained for 60 epochs. The total training duration was approximately 90 minutes on the GPU (refer to Chapter 3.6), but it may require several hours to train the model on a CPU. The ratio of concrete crack to non-crack images was 1:1, while that of the training set to the validation set was 4:1. Accordingly, the training and validation accuracies were calculated from 32K and 8K images, respectively. Figure 3.7 depicts the summary of the training results over the epochs. The best training and validation accuracies were obtained in the 51st and 49th epochs, respectively. The validation accuracy fluctuated but remained close to the training accuracy. As Figure 3.7 shows, the model almost converged after 50 epochs, and the training process was stopped in the 60th epoch.

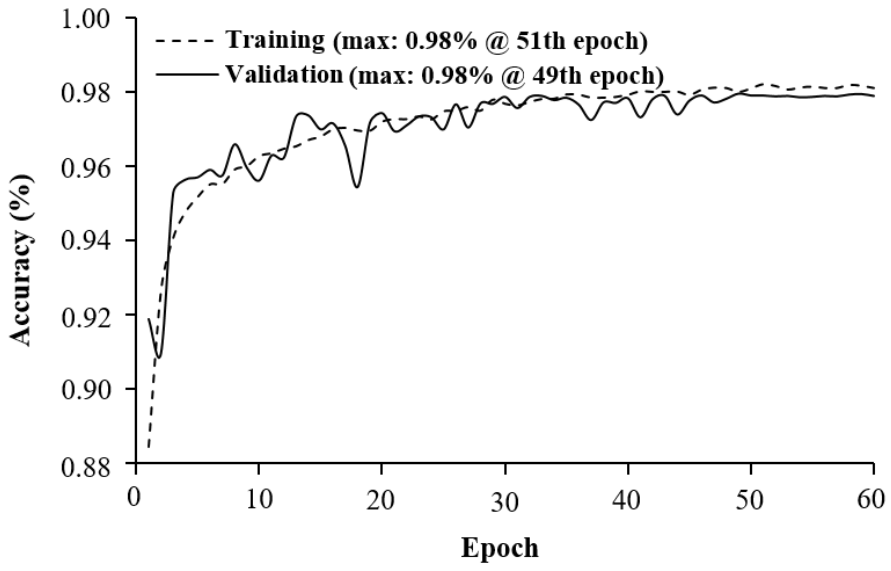


Figure 3.7: Accuracies over epochs [Cha et al., (2017)]

A simple experiment was conducted to estimate the desirable number of training instances. The results may offer heuristic information about building a model with an objective similar to that of this implementation in the future. In this experiment, seven different models were additionally trained on the datasets with 2, 4, 8, 10, 16, 20, and 28K images. Each model was trained under the same context as the model trained on the dataset with 40K images. The architectures of each model were also identical (refer to Chapter 3.2). The results of the experiment are summarized in Figure 3.8. According to the findings of this parametric study, at least 10K images are required to obtain a reasonable CNN classifier with a validation accuracy of 0.97 in the concrete crack detection problem.

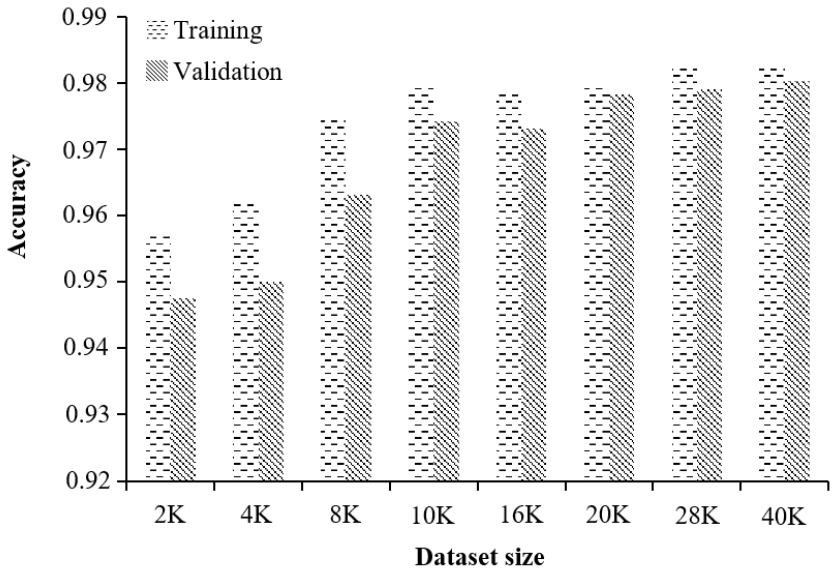


Figure 3.8: Results of the experiment to estimate the desirable number of training instances [reproduced from Cha et al. (2017)]

3.7 Model testing

The model trained on the dataset with 40K images was considered as the final model. However, the proposed model could only accept images of size 256 × 256 pixels (refer to Chapter 3.2), and thus, additional preprocessing was required. The preprocessing resembled the sliding-window technique, and the schematic testing plan with the post-processing is illustrated in Figure 3.9.

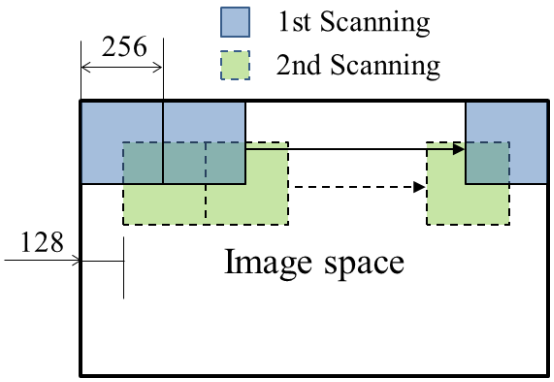


Figure 3.9: Testing with post processing [Cha et al., (2017)]

3.7.1 Testing the trained and validated CNN

The performance of the final model was tested on 54 large images. Note that these images were not used to train the model. The complete evaluation metrics are provided in Table 3.2. The overall accuracy was calculated as 0.97, which was similar to the accuracy obtained from the validation set (refer to Chapter 3.6.2). Several representative results are shown in Figures 3.10 to 3.14. The scales on each axis of the images in the figure may provide a guideline for the dimensions of the tested image. The transparent patches in the right panel of each image show the regions classified as "crack patch" (positive), while the blanks indicate the regions classified as "non-crack" (negative). False predictions (i.e., false positives (FPs) and false negatives (FNs) were also observed and are highlighted as the magenta and blue boxes. Some of the FN and FP patches are highlighted as green and red boxes and enlarged to present the characteristics of the false predictions. The proposed model with the post-processing could test an image of 5888 × 3584 pixels in approximately 4.5 seconds.

Table 3.2: Summary of test results [Cha et al. (2017)]

No.	Pos.	Neg.	TP(iii)	TN(iv)	FP(v)	FN(vi)	Accuracy	Precision	Recall	F1	Remark
1	126	482	103	473	9	23	0.95	0.92	0.82	0.87	Figure 3.10
2	162	446	143	438	8	19	0.96	0.95	0.88	0.91	Figure 3.11
3	55	553	54	538	15	1	0.97	0.78	0.98	0.87	Figure 3.12
4	37	571	35	566	5	2	0.99	0.88	0.95	0.91	Figure 3.13
5	58	550	41	550	0	17	0.97	1.00	0.71	0.83	Figure 3.14
6	45	269	42	266	3	3	0.98	0.93	0.93	0.93	-
7	23	291	23	289	2	0	0.99	0.92	1.00	0.96	-
8	35	279	35	275	4	0	0.99	0.90	1.00	0.95	-
9	31	283	25	283	0	6	0.98	1.00	0.81	0.89	-

10	31	283	29	281	2	2	0.99	0.94	0.94	0.94	-
11	32	282	32	279	3	0	0.99	0.91	1.00	0.96	-
12	30	284	30	277	7	0	0.98	0.81	1.00	0.90	-
13	30	284	30	283	1	0	1.00	0.97	1.00	0.98	-
14	31	283	31	281	2	0	0.99	0.94	1.00	0.97	-
15	31	283	30	253	30	1	0.90	0.50	0.97	0.66	-
16	38	276	32	271	5	6	0.96	0.86	0.84	0.85	-
17	28	286	28	285	1	0	1.00	0.97	1.00	0.98	-
18	34	392	34	389	3	0	0.99	0.92	1.00	0.96	-
19	30	396	30	391	5	0	0.99	0.86	1.00	0.92	-
20	23	403	23	400	3	0	0.99	0.88	1.00	0.94	-
21	36	390	34	376	14	2	0.96	0.71	0.94	0.81	-
22	39	387	38	366	21	1	0.95	0.64	0.97	0.78	-
23	27	399	26	396	3	1	0.99	0.90	0.96	0.93	-
24	27	399	25	391	8	2	0.98	0.76	0.93	0.83	-
25	22	404	22	386	18	0	0.96	0.55	1.00	0.71	-
26	34	392	34	373	19	0	0.96	0.64	1.00	0.78	-
27	33	393	30	377	16	3	0.96	0.65	0.91	0.76	-
28	31	395	31	381	14	0	0.97	0.69	1.00	0.82	-
29	33	393	33	379	14	0	0.97	0.70	1.00	0.83	-
30	30	396	30	395	1	0	1.00	0.97	1.00	0.98	-
31	46	380	45	379	1	2	1.00	0.98	0.96	0.97	-
32	31	316	31	295	21	0	0.94	0.60	1.00	0.75	-
33	49	298	43	298	0	6	0.98	1.00	0.88	0.93	-
34	53	294	49	292	2	4	0.98	0.96	0.92	0.94	-

35	30	317	27	314	3	3	0.98	0.90	0.90	0.90	-
36	26	321	24	310	11	2	0.96	0.69	0.92	0.79	-
37	43	304	36	301	3	7	0.97	0.92	0.84	0.88	-
38	56	291	55	277	14	1	0.96	0.80	0.98	0.88	-
39	48	299	44	290	9	4	0.96	0.83	0.92	0.87	-
40	43	304	42	280	24	1	0.93	0.64	0.98	0.77	-
41	52	295	52	281	14	0	0.96	0.79	1.00	0.88	-
42	57	290	57	266	24	0	0.93	0.70	1.00	0.83	-
43	50	297	50	253	44	0	0.87	0.53	1.00	0.69	-
44	41	306	41	288	18	0	0.95	0.69	1.00	0.82	-
45	69	278	68	262	16	1	0.95	0.81	0.99	0.89	-
46	57	290	57	262	28	0	0.92	0.67	1.00	0.80	-
47	73	274	63	269	5	10	0.96	0.93	0.86	0.89	-
48	24	323	24	322	1	0	1.00	0.96	1.00	0.98	-
49	21	326	19	324	2	2	0.99	0.90	0.90	0.90	-
50	28	319	26	319	0	2	0.99	1.00	0.93	0.96	-
51	55	292	52	284	8	3	0.97	0.87	0.95	0.90	-
52	27	320	23	307	13	4	0.95	0.64	0.85	0.73	-
53	33	314	33	310	4	0	0.99	0.89	1.00	0.94	-
54	31	316	31	295	21	0	0.94	0.60	1.00	0.75	-
∑	2265	18488	2125	17966	522	141	0.97	0.80	0.94	0.87	-

Pos.: Crack patches; Neg.: non-crack patches; TP: True positives; TN: True negatives; FN: False negatives; Accuracy:{(iii)+(iv)}/{(i)+(ii)}; Precision: (iii)/{(iii)+(v)}; Recall: (iii)/{(iii)+(vi)}; F1: 2 × (precision × recall)/(precision + recall)

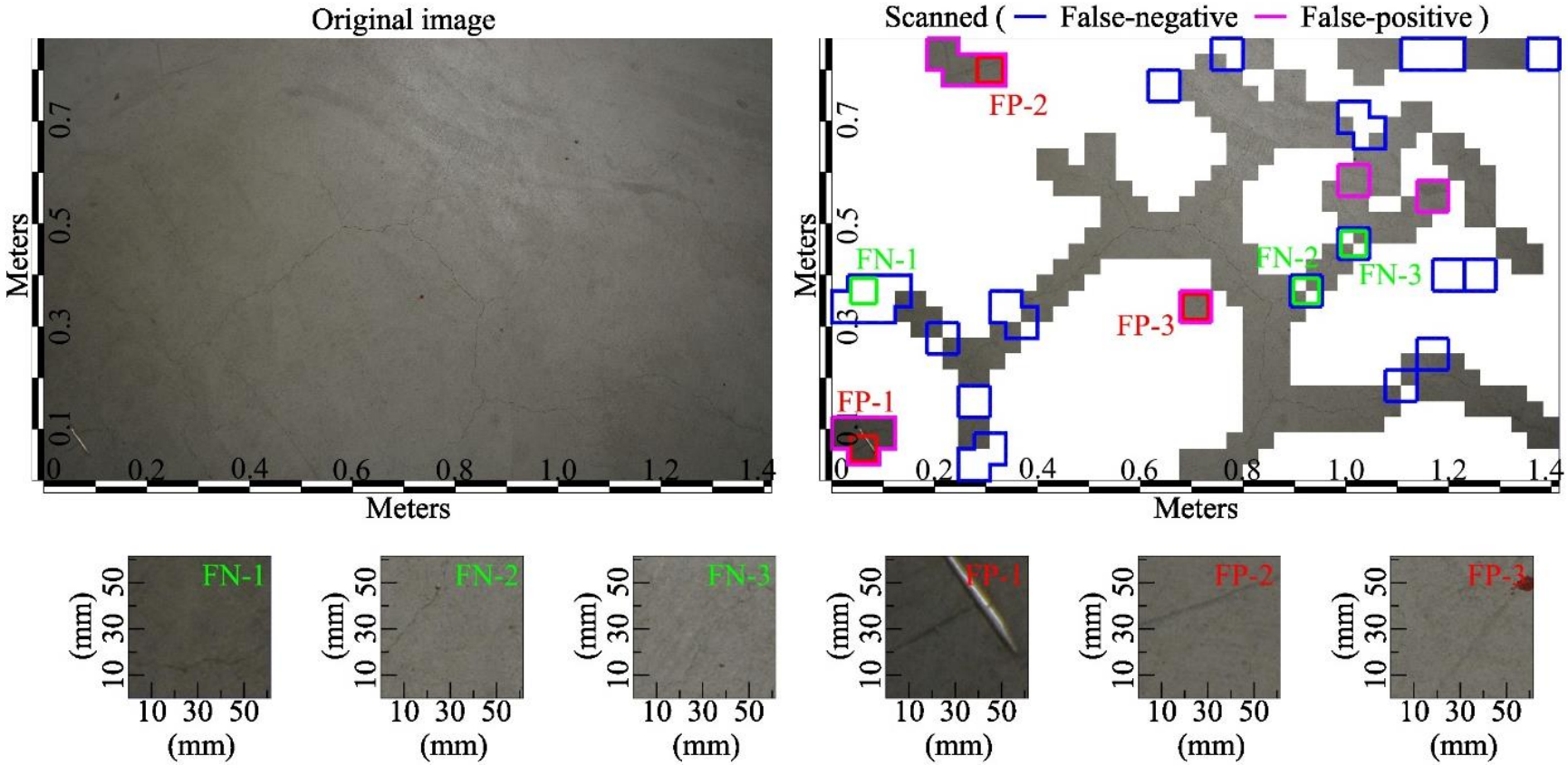


Figure 3.10: Representative testing result (1) – thin cracks [Cha et al. (2017)]

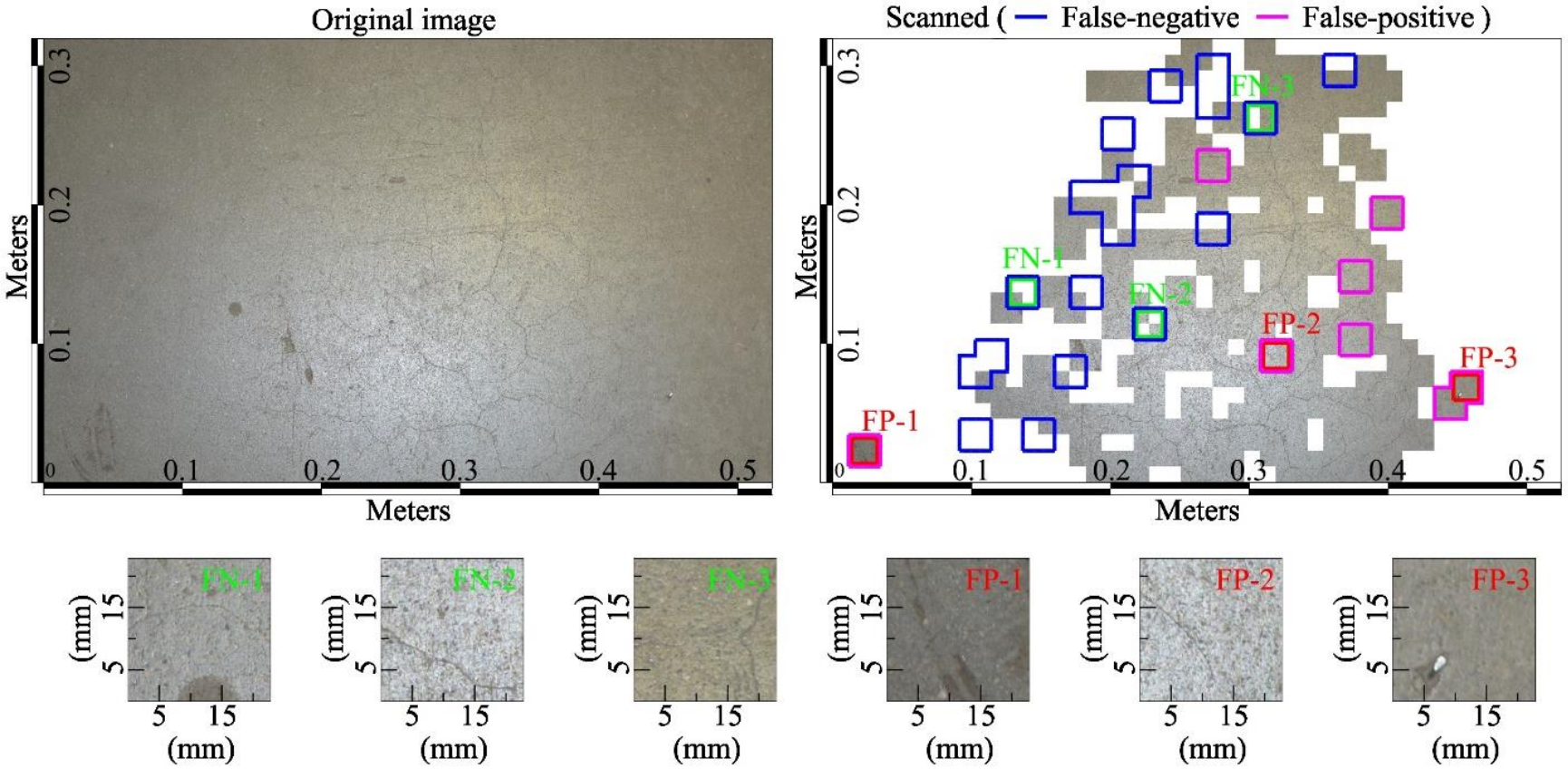


Figure 3.11: Representative testing result (2) – thin cracks with high luminance spots [Cha et al. (2017)]

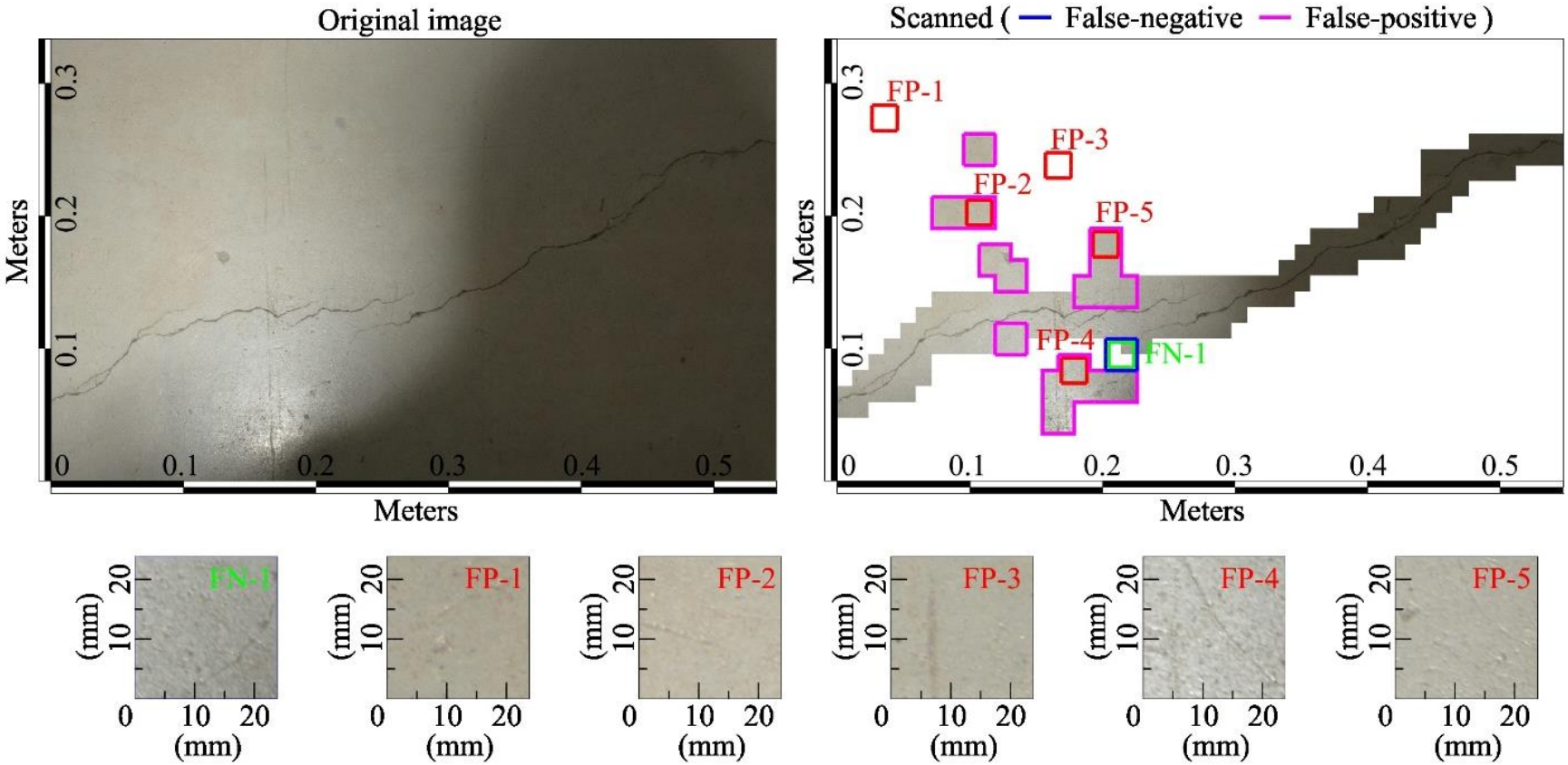


Figure 3.12: Representative testing result (3) – with shadows [Cha et al. (2017)]

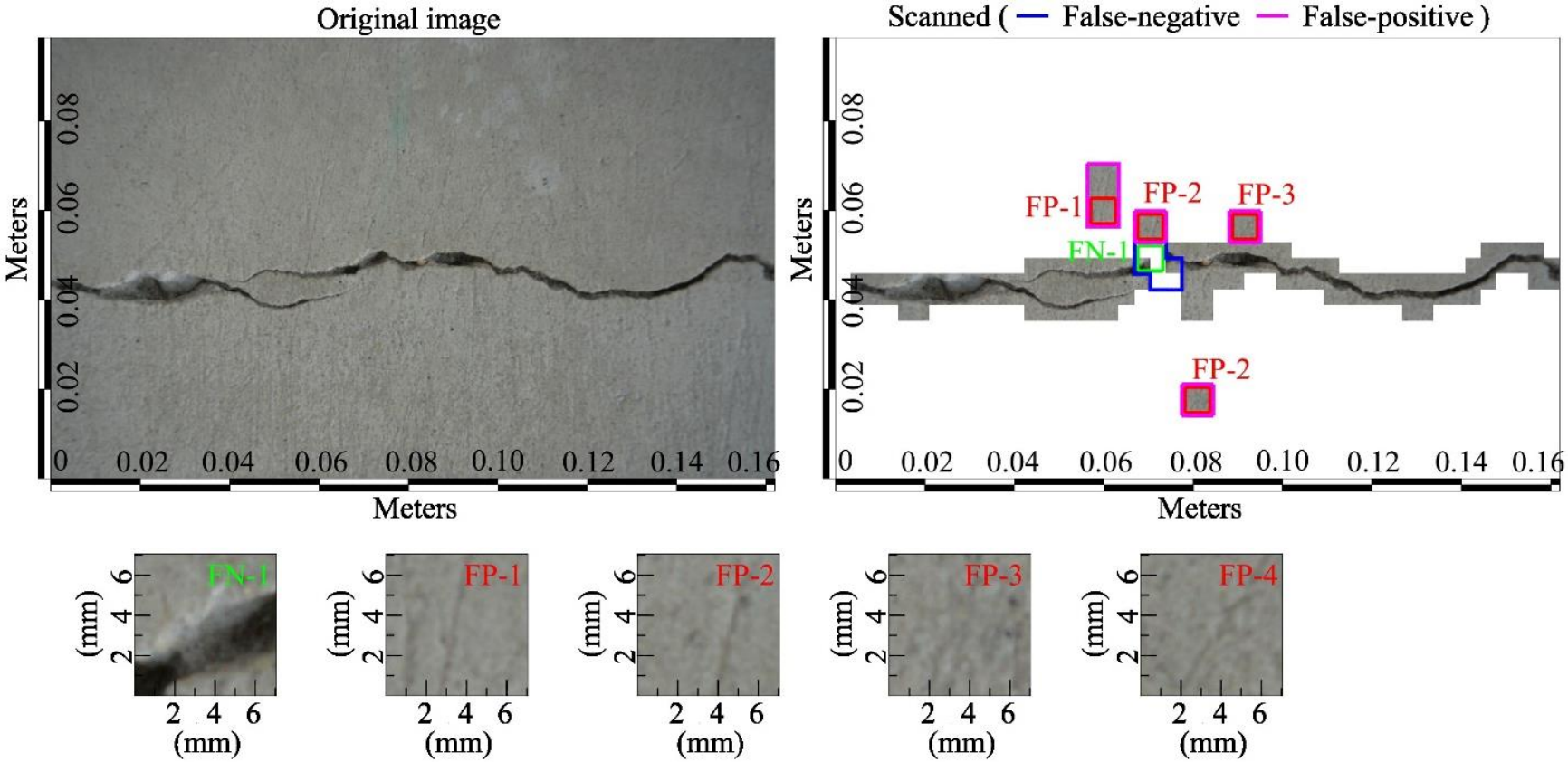


Figure 3.13: Representative testing result (4) – closeups [Cha et al. (2017)]

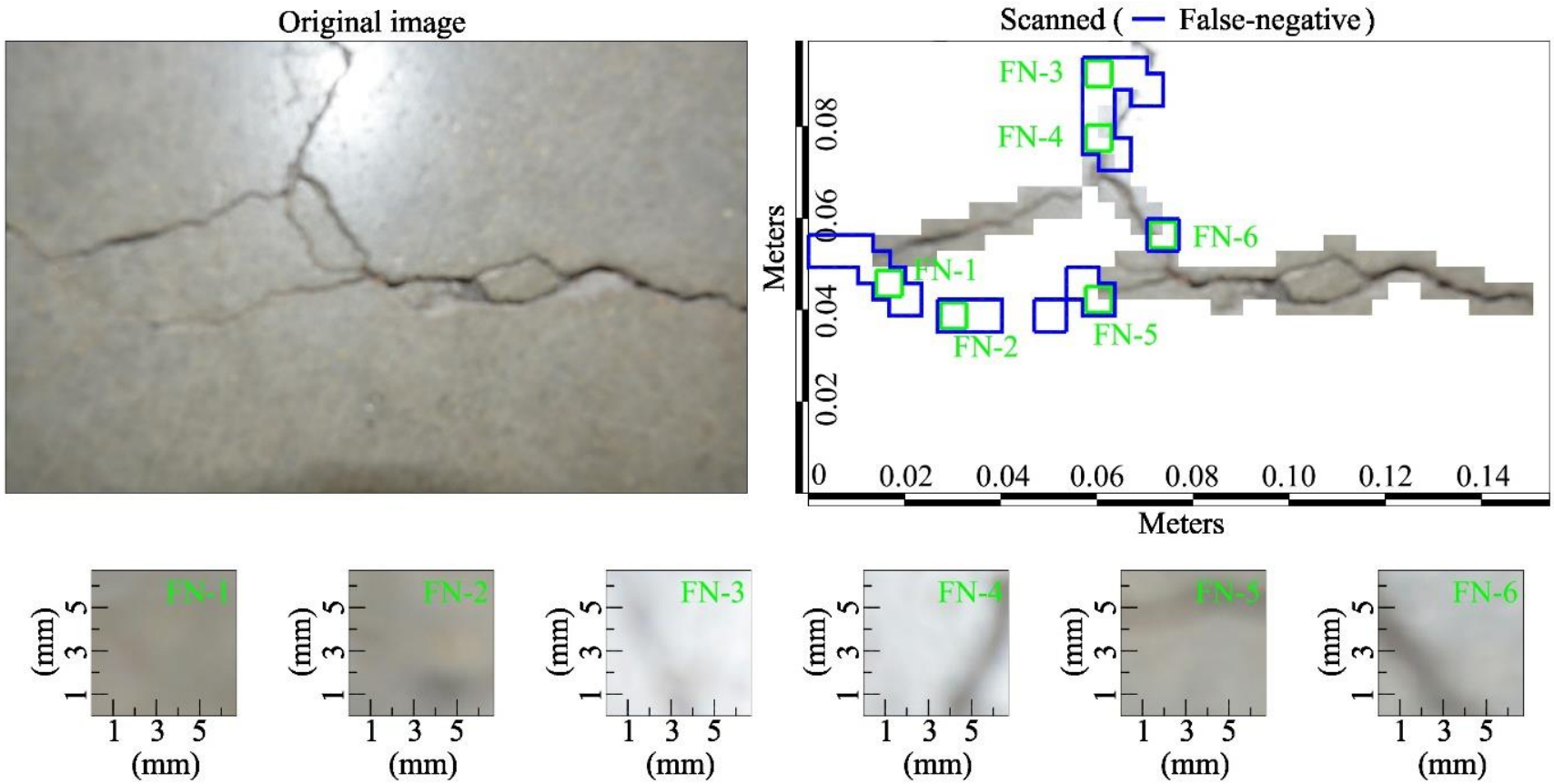


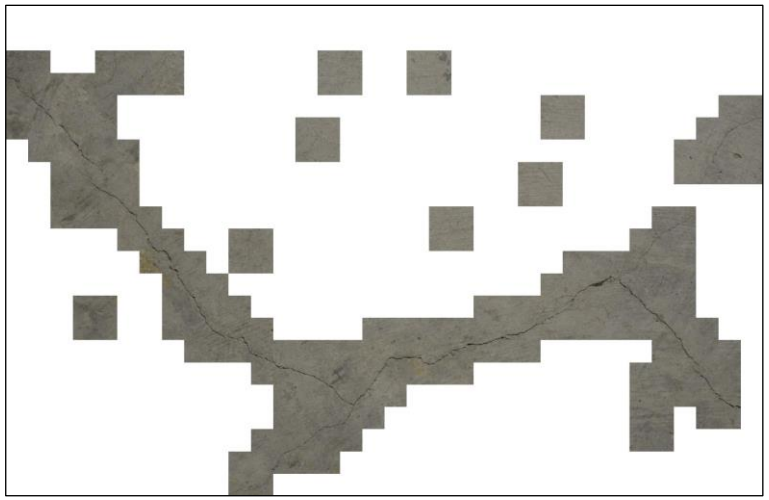
Figure 3.14: Representative testing result (5) – closeups, blurring, and with luminance spots [Cha et al. (2017)]

Figure 3.10 shows the test results of an image with very thin cracks and uniform luminance. The thickest crack width was 4 pixels (approximately 1.5 mm). The FN predictions were mostly observed on the periphery of the image center because of image distortions in the thin crack regions. To study the luminance condition sensitivity, an image with a strong luminance spot and another with a shadow were tested, as shown in Figures 3.11 and 3.12. Figure 3.11 shows that FNs were mainly detected on the edges of the luminance spot. Figure 3.12 shows only one FN prediction, but a number of FPs were observed because the features of the scratches resembled cracks. Considering all the results discussed thus far, the proposed method can potentially be used under various luminance conditions. An image was intentionally taken approximately 70 mm away from a concrete surface and tested to study whether the proposed method is susceptible to changes in distance (refer to Figure 3.13), and the accuracy was calculated to be 0.99. Figure 3.14 shows an image taken 60 mm away from a concrete surface and the corresponding results. The image was blurry due to the short distance from the concrete surface. In addition, the image contained a high luminance spot. Nevertheless, the proposed method returned an accuracy of 0.97. Considering all the results, the proposed method is valid regardless of distance changes.

3.7.2 Comparative studies

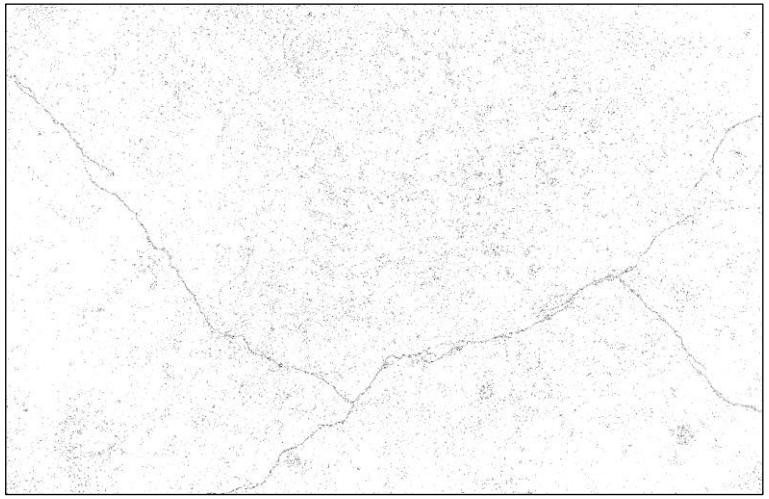
A comparative study was conducted to assess to what extent the proposed method outperformed traditional methods. Canny and Sobel edge detectors are the most popular conventional methods for identifying cracks from images. The results of these methods were compared with those of the proposed model. In the case of an image of a concrete surface with a relatively clean texture and uniform luminance, as shown in Figure 3.15, the results of the proposed method (Figure 3.15(b)) and the Sobel edge detector (Figure 3.15(d)) are comparable.

(a) Original image



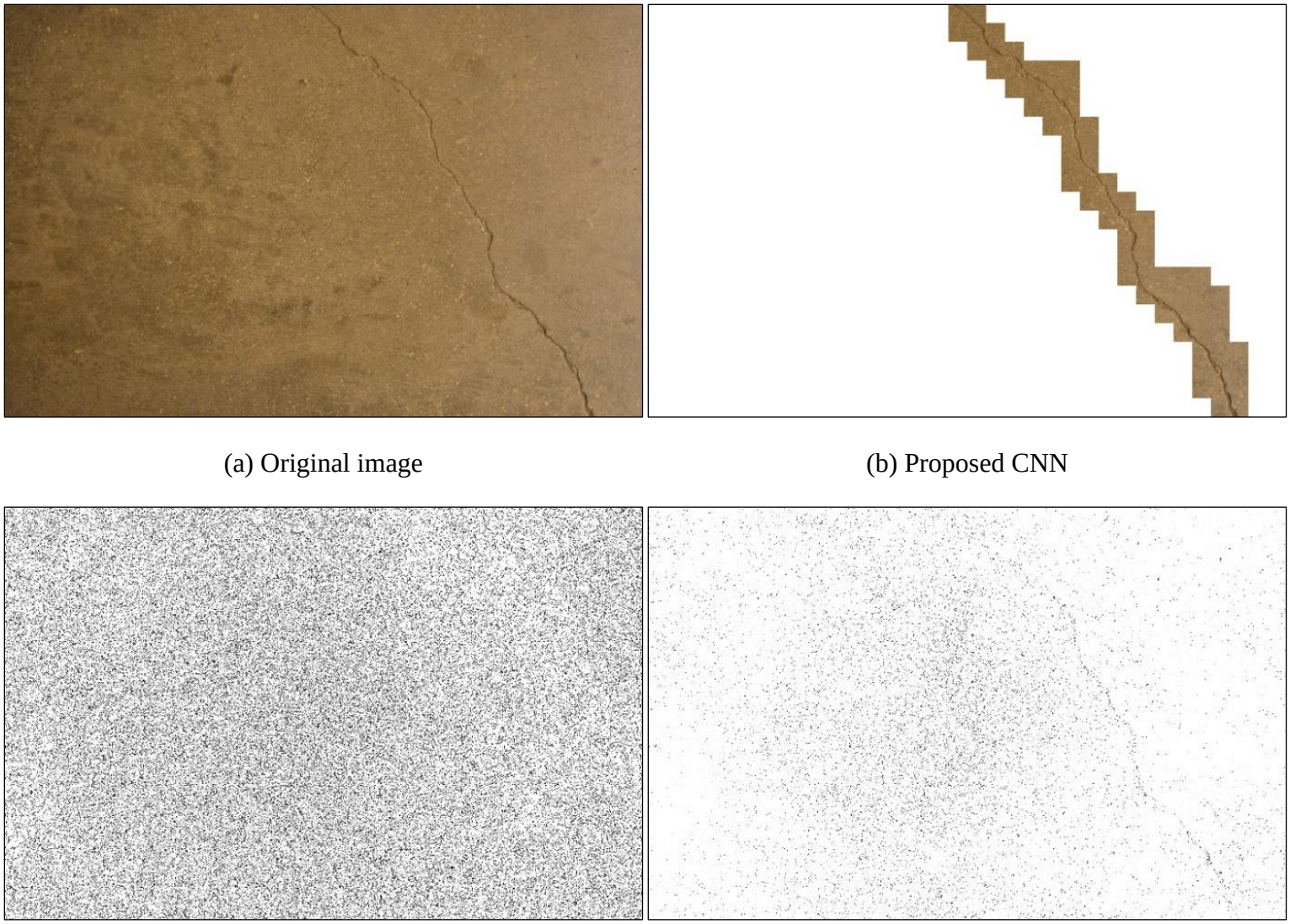
(b) Proposed method

(c) Canny edge detector applied on (a)



(d) Sobel edge detector applied on (a)

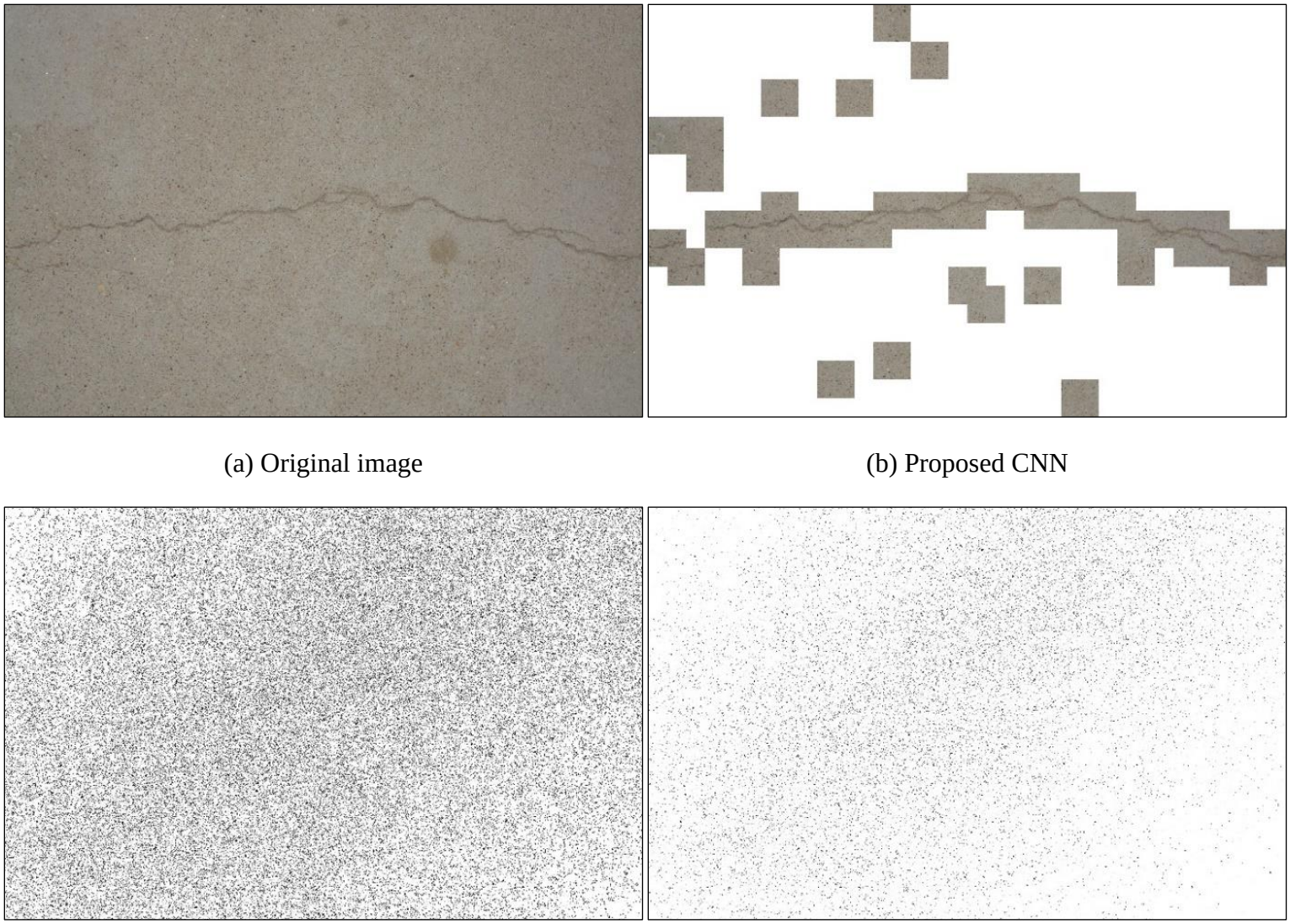
Figure 3.15: Image with uniform luminance condition



(a) Original image (b) Proposed CNN

(c) Canny edge detection (d) Sobel edge detection

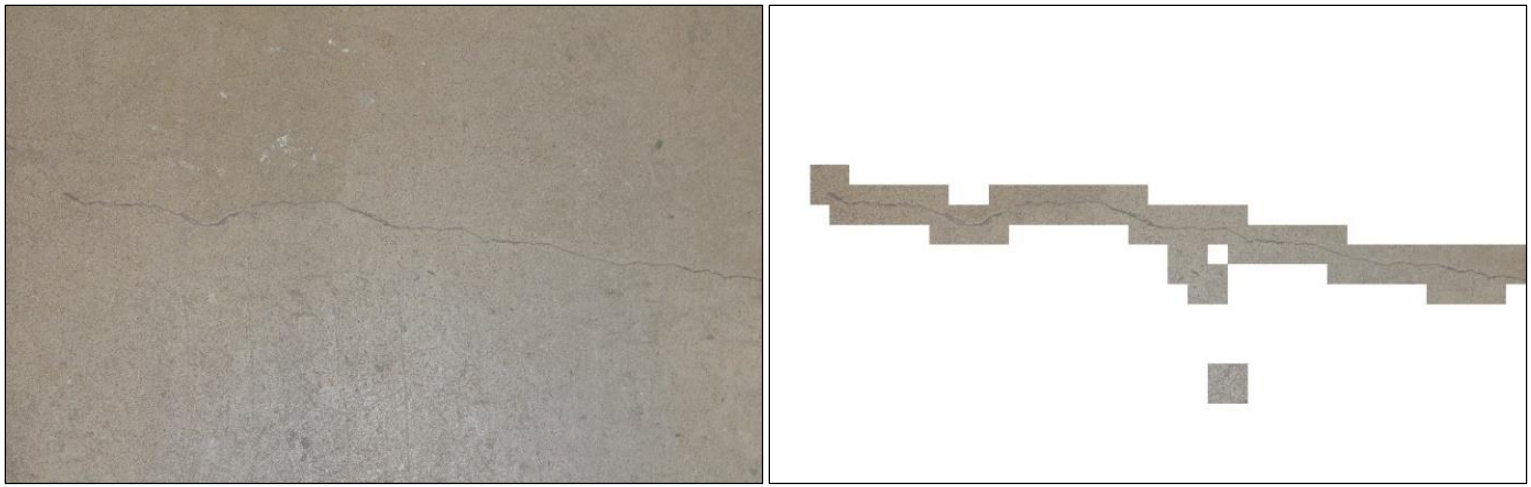
Figure 3.16: Image with noisy texture and uniform luminance (1) [Cha et al. (2017)]



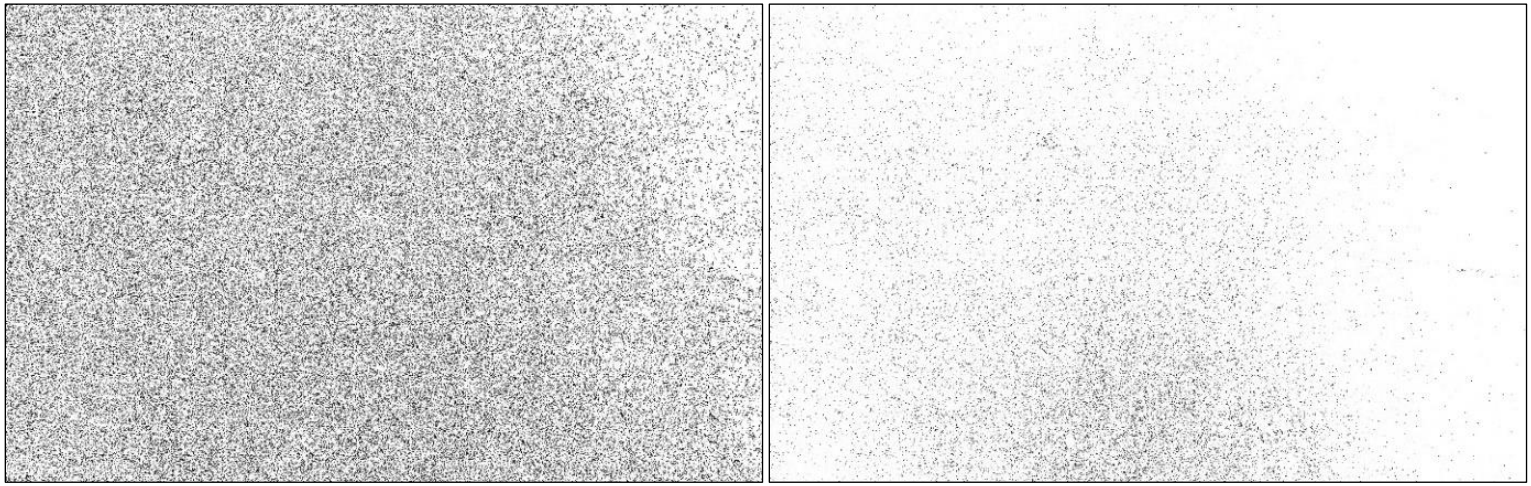
(a) Original image (b) Proposed CNN

(c) Canny edge detection (d) Sobel edge detection

Figure 3.17: Image with noisy texture and uniform luminance (2) [Cha et al. (2017)]

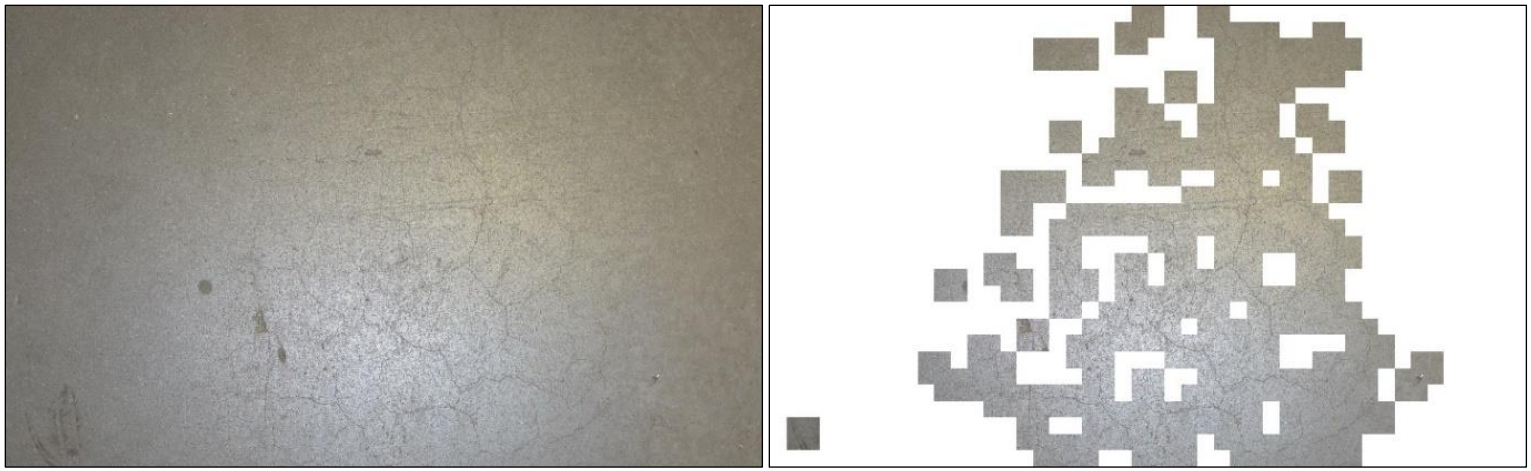


(a) Original image (b) Proposed CNN

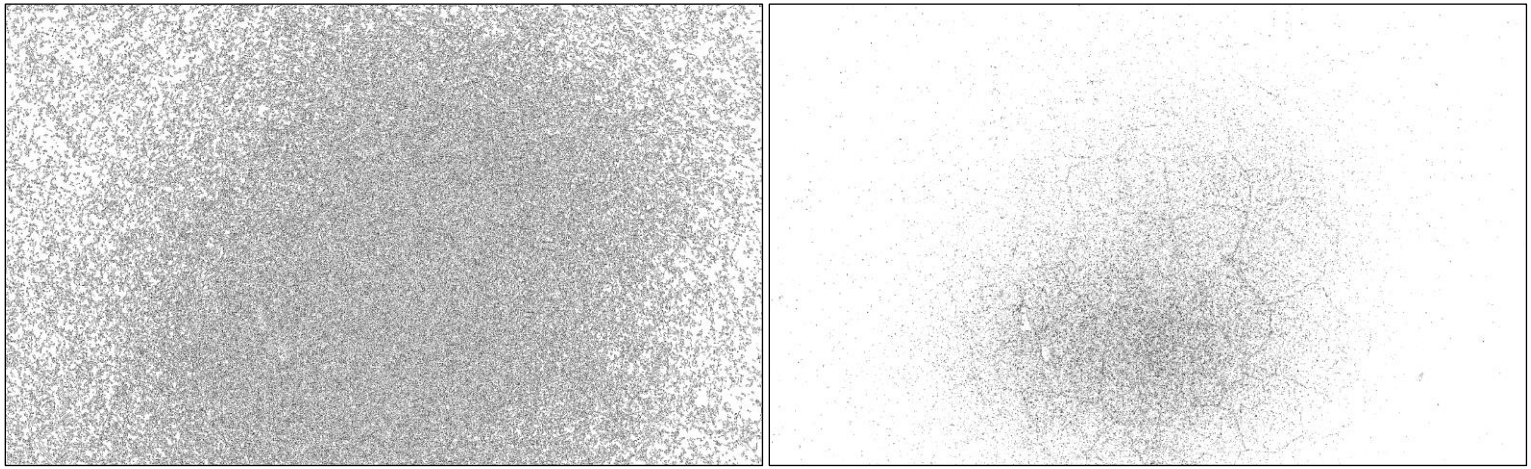


(c) Canny edge detection (d) Sobel edge detection

Figure 3.18: Image with thin crack and slightly varying luminance [Cha et al. (2017)]

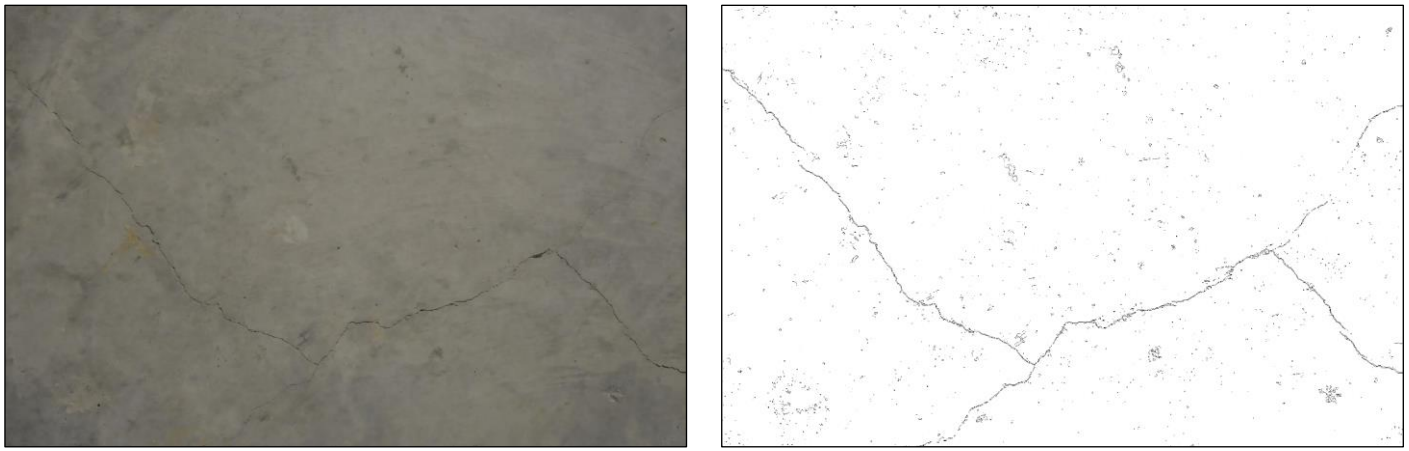


(a) Original image (b) Proposed CNN



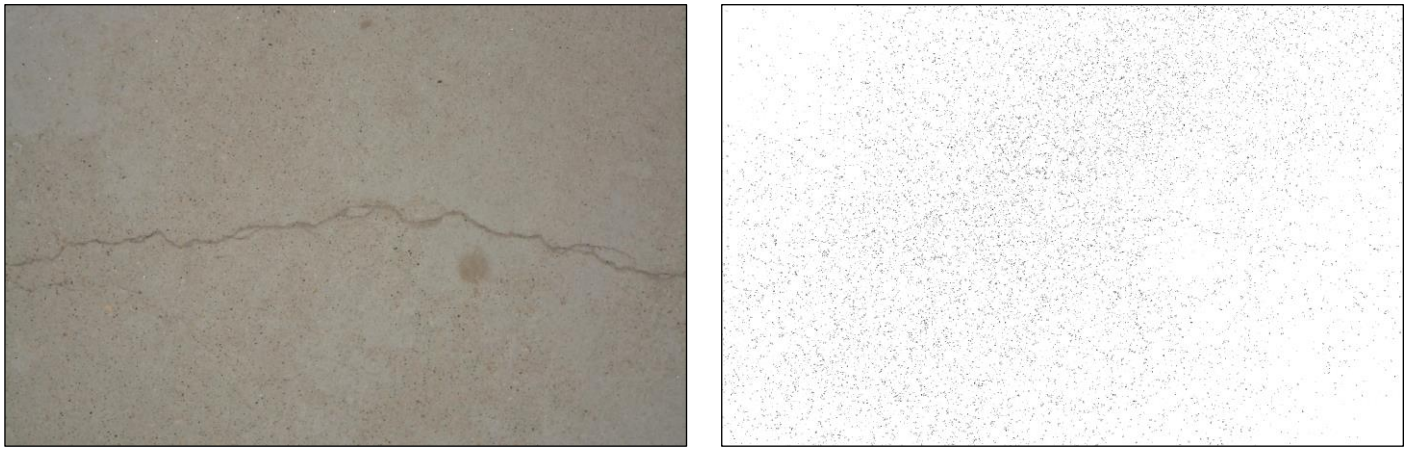
(c) Canny edge detection (d) Sobel edge detection

Figure 3.19: Image with thin cracks and strong luminance change [Cha et al. (2017)]



(a) Denoised image (b) Sobel Edge detector

Figure 3.20: Sobel edge detector with denoising method (1)



(a) Denoised image (b) Sobel Edge detector

Figure 3.21: Sobel edge detector with denoising method (2)

However, the traditional methods often return unreliable results if an image contains noise. For example, the images shown in Figures 3.16(a) and 3.17(a) were taken under uniform luminance conditions, but the traditional methods did not return meaningful information due to the concrete texture, as shown in Figures 3.16(c), 3.16(d), 3.17(c), and 3.17(d), whereas the proposed method effectively identified the cracks, as shown in Figures 3.16(b) and 3.17(b).

Further testing with the traditional methods on images with non-uniform luminance also did not return meaningful information, but the proposed method continued to provide consistent results. For example, Figure 3.18(a) shows an image with a thin crack and slightly varying luminance. Both the Canny and Sobel edge detectors did not identify the crack (Figures 3.18(c) and 3.18(d)). The same results were returned for an image with strong luminance (refer to Figure 3.19).

Denoising techniques may improve the performance of the traditional methods, and additional experimental results to this end are presented in Figures 3.20 and 3.21. A number of denoising techniques are available, but the edge-aware denoising[6] method proposed by Gastal and Oliveira (2012) was chosen to preserve the features of the cracks (i.e., edges) from the original image. One of the best working examples is shown in Figure 3.20, where Figure 3.20(a) is the denoised image of Figure 3.15(a). As shown in Figure 3.20(b), the traditional method returned more informative results compared to those in Figure 3.15(d). However, defining the denoising parameters for an image is a manual process, and the set of optimal parameters is dependent on the image conditions. For example, the parameters[7] used for obtaining Figure 3.20(a) were reused to denoise the image shown in Figure 3.17(a), and the denoised image is

[6] The code for edge-aware denoising is available at http://inf.ufrgs.br/~eslgastal/AdaptiveManifolds/ (accessed Dec. 2019). To obtain the denoised image shown in Figure 3.20(a), "sigma_s" and "sigma_r" of "adaptive_manifold_filter.m" were set at 5 and 0.1, respectively. Complete details of the denoising method are not provided in this thesis in the interest of conciseness.

presented in Figure 3.21(a). The denoised image was processed by the Sobel edge detector. The result appears in Figure 3.21(b), which does not show meaningful information.

3.8 Discussions and conclusions

In this work, deep learning was implemented to detect concrete cracks from images. The deep learning model used in this study was a classification model, and the last operation involved an FC layer that restricted the size of the input images to 256 × 256 pixels. An image dataset was manually created to train the DL model, wherein each image was taken under a wide range of conditions. Three hundred nine and 54 images were used in training and testing, respectively. The images in each subset were randomly cropped into images of 256 × 256 pixels due to size limitations. The training and testing accuracies were found to be approximately 0.98. To overcome the size limitations, a sliding-window technique was used for preprocessing, following which large image sizes could be tested. The trained model and the sliding-window technique were further tested on 54 large images, and the corresponding test results indicated consistent model performance. Traditional methods, such as the Canny and Sobel edge detectors with and without edge-aware denoising, were compared with the proposed method. Several test results from the traditional methods were comparable with the proposed model when the tested images contained clean textures and uniform luminance. Otherwise, the proposed method showed better results.

However, the proposed model was tested only on images with relatively monotonous backgrounds and flat surfaces, whereas in reality, damage can also be found against complex backgrounds and geometries. In addition, the configured architecture was too naïve. The layers before the first ReLU activation (which was the only activation in the model) conducted simple linear transformations. Thus, the model expended a considerable amount of computation resources for linear operations. High accuracies were returned most likely because the images had monotonous backgrounds and flat surfaces,

thus easing the model's task. Moreover, defining the size of the sliding window also posed issues. The next chapter discusses these aspects in more detail to draw closer to achieving the aim of developing a field-ready method.

References

Cha, Y.-J., Choi, W., & Büyüköztürk, O. (2017). Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, *32*(5), 361–378. DOI: 10.1111/mice.12263

Gastal, E. S. L., & Oliveira, M. M. (2012). Adaptive manifolds for real-time high-dimensional filtering. *ACM Transactions on Graphics*, *31*(4), 1–13. DOI: 10.1145/2185520.2185529

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. *Lecture Notes in Computer Science*, 9–48. DOI: 10.1007/978-3-642-35289-8_3

Vedaldi, A., & Lenc, K. (2015). MatConvNet. *Proceedings of the 23rd ACM International Conference on Multimedia - MM '15*. DOI: 10.1145/2733373.2807412

Wilson, D. R., & Martinez, T. R. (2001). The need for small learning rates on large problems. *International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, 1, 115-119. DOI: 10.1109/ijcnn.2001.939002

Chapter 4 Deep Learning Application: Crack Segmentation

Summary

The previous work presented in Chapter 3 demonstrated potential but is arguably applicable in practice. This chapter investigates the limitations of the previous method and presents an alternative method, which is to define the task as segmenting damage features by classifying each pixel rather than classifying each image. The damage detection scheme allowed to build a robust model that had no restrictions on image size, was free from the repetitive calculations of the previous method, produced more intuitive results, and processed images in real time. The comparative study revealed that the proposed model outperformed other recent works in every aspect.

Chapter 4 has been reproduced with modifications to figures and formatting from Choi and Cha (2020). The chapter has been reprinted with permission from the copyright holder, IEEE.

4.1 Introduction

The combination of a convolutional neural network (CNN) for classification with a sliding-window technique was presented in Chapter3 (Cha et al., 2017) and has been further extended by Chen and Jahanshahi (2018), Kang and Cha (2018), Ali and Cha (2019). However, there are several notable limitations. For example, as depicted in Figure 4.1, the input image is cropped, and each of the cropped regions is individually classified rather than processing the entire image at once. This requires repetitive computations and consequently results in longer processing time. In addition, the localized areas are strictly dependent on the size of a sliding-window rather than the size of objects, which leads to an additional challenge in defining the window size.

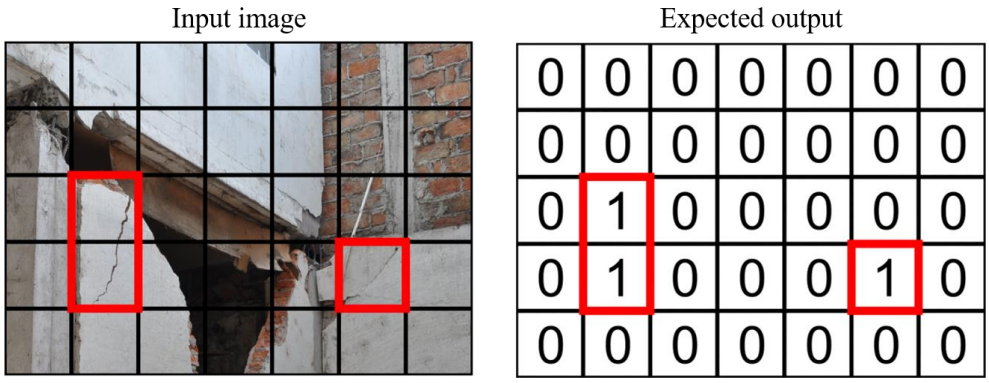


Figure 4.1: Object localization with sliding-window

This disadvantage shifted researchers' attention to flexible localization algorithms, such as Faster R-CNN (Ren et al., 2015) and YOLO (Redmon et al., 2016). The object localization scheme is to predict the coordinates of each object's centroid ($\bar{H}$ and $\bar{W}$) and its bounding box profiles ($\bar{h}$ and $\bar{w}$), as represented in Figure 4.2.

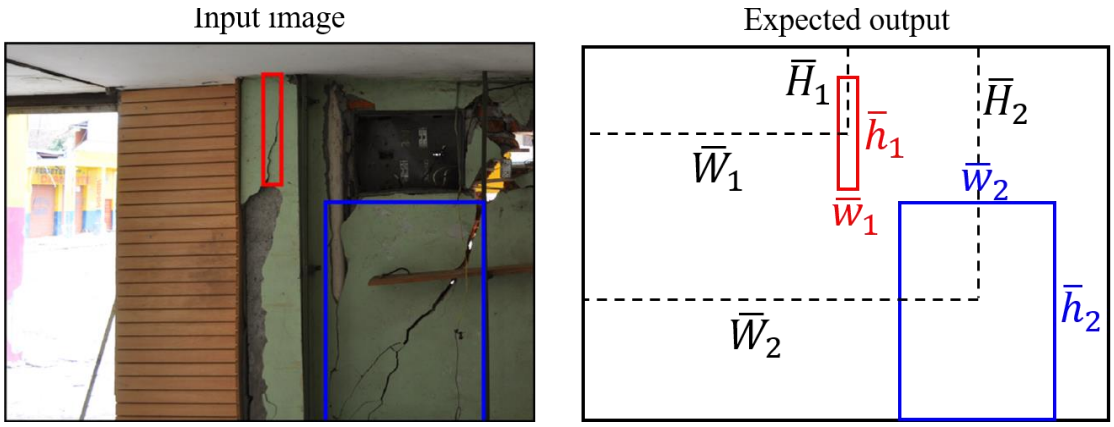


Figure 4.2: Object localization with bounding box

Cha et al. (2018), Li et al. (2018), Xue and Li (2018), and Beckman et al. (2019) adopted Faster-RCNN for localizing superficial damage. However, most structural damage, such as cracks and spalling, is not expected to have typical shapes (e.g., vehicles, buildings, etc.), and the corresponding outcomes are not sufficiently informative if objects have thin elongated features presenting in a diagonal direction. Hence, these approaches are less preferable in SHM. To produce more informative results, localizing damage features is defined as semantic segmentation in this study.

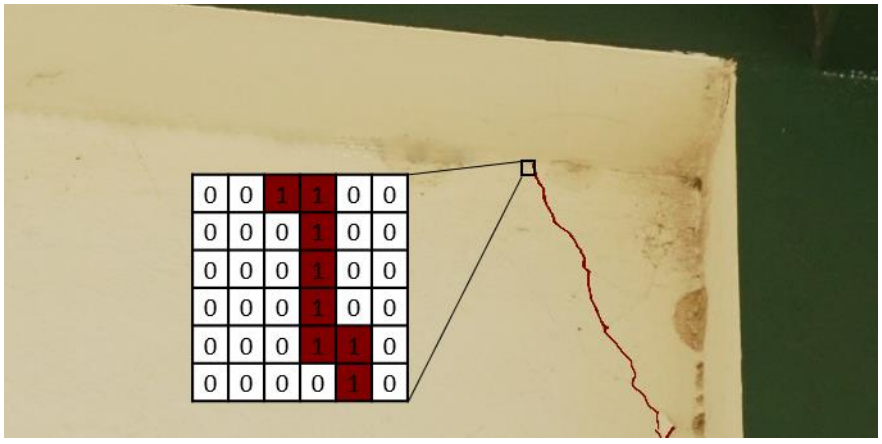


Figure 4.3: Semantic segmentation [Choi and Cha (2020)]

Sematic segmentation is one of the key tasks in DL, which has been actively studied in the field of medical image analysis (Ronneberger et al., 2015) and self-driving vehicles (Siam et al., 2017). Semantic segmentation is conceptually similar to image classification, as demonstrated in Figure 4.3; each pixel of an image is classified in such a way that the pixels of zeros and ones indicate background and crack pixels, respectively. Several articles (Yang et al., 2018; Zhang et al., 2018; Dung, 2019; Liu et al., 2019; Nayyeri et al., 2019) introduced damage segmentation methods. However, these studies aimed to segment damage features only from monotonous images, while real practice primarily involves images with complex backgrounds.

In this chapter, segmenting concrete cracks from images with complex features is primarily discussed. Cracks are the most usual instances of superficial damage and are hardest to obtain satisfying results from due to its thin features compared to other types of damage. The objectives of this study are listed as follows: building a real-time method for segmenting damage features; negating a wide range of background features and crack-like patterns.

4.2 Architecture configuration

In segmentation, a DL model with a FC layer (refer to Chapter 2.3.1) is less preferable because the model can only accept a fixed size of input. For example, the DL model presented in the previous chapter has an FC layer. The model is designed to accept an input of 256×256 pixels. If various sizes of images are fed into the model, the number of elements feeding into the FC layer is also various, while the FC layer can extract features from the input with 96 elements (refer to Figure 3.2 and Table 3.1). Removing all FC layers of a model is a simple solution that addresses the limitation, and a model without any FC layer is referred to as a fully convolutional network (Long et al., 2015). The proposed model in this chapter follows the convention of a fully convolutional network.

Monumental DL models have demonstrated significant capability in recognizing images. For example, Simonyan et al. (2014), He et al. (2016), and Huang et al. (2017) presented their DL models trained on ImageNet (Deng et al., 2009), which consists of 1K number of classes with 18 million images. Contrarily, the number of classes recognized in SHM research is much less. This suggests that developing a task-specific model might be beneficial in computation without performance deterioration. Therefore, a DL model was configured instead of adopting the monumental DL models (Simonyan et al. (2014), He et al. (2016), and Huang et al. (2017)). Hereafter, the model is referred to as the semantic damage detection network (SDDNet). The model was inspired by the DenseNet (Huang et al., 2017) and DeepLabV3+ (Chen et al., 2018).

4.2.1 Characteristics of target objects

To develop a task-specific model, the characteristics of target objects were investigated. Objects from generic images (e.g., humans, etc.) usually have well-distinguishable features, and peripheral information may less important to segment the objects. For example, assume that humans are the target objects to be segmented, in which small or partial images without any contextual information can still represent features of humans regardless of variations (i.e., jogging, riding a bicycle, etc.), as shown in Figure 4.4.

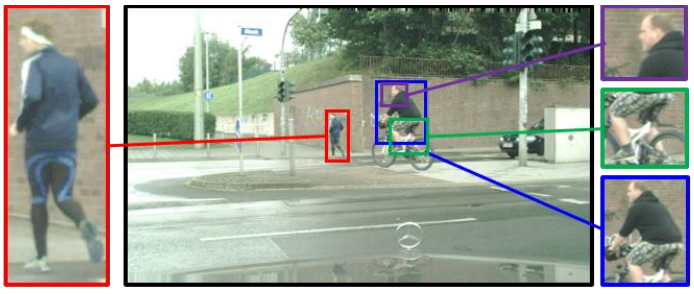


Figure 4.4: Generic image data

However, simple and irregular-shaped objects are difficult to distinguish even with human eyes without peripheral information. This is because the features residing in a small region do not represent the objects. For example, as depicted in Figure 4.5, each of the cropped regions has very similar features even though they belong to completely different classes. Therefore, leading a DL model to see a large field of view (FOV) is preferable.

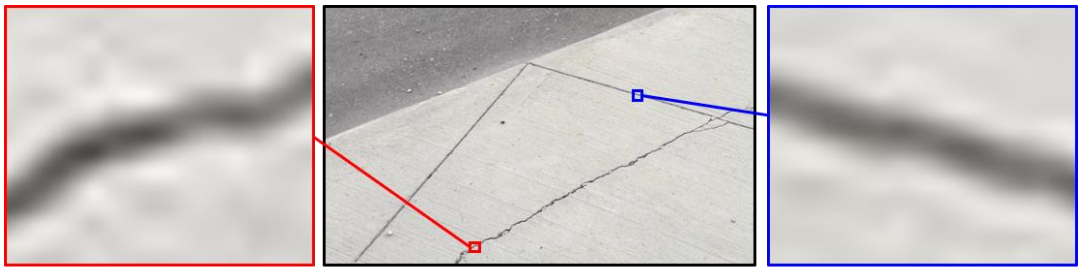


Figure 4.5: Real crack vs crack-like feature [Choi and Cha (2020)]

4.2.2 Overall architecture

The SDDNet model has encoder and decoder parts. Figure 4.6 displays the SDDNet schematically, in which the operations before the decoder are considered as the encoder. The roots of the model are accompanied by two standard convolutions (Conv. in Figure 4.6). The densely connected separable convolution (DenSep in Figure 4.6; see Chapter 4.2.3 for details) modules are followed several times. Each densely connected separable convolution module conducts separable convolutions, and a separable convolution involves point-wise and depth-wise convolutions (Chen et al., 2018). The details of these can be found in the next subchapter. In the figure, the first values of each parenthesis indicate the number of filters of the standard convolutions. The first values of each bracket are the number of separable convolutions. The second values (denoted by ‘s’) of either parentheses and brackets are the stride values.

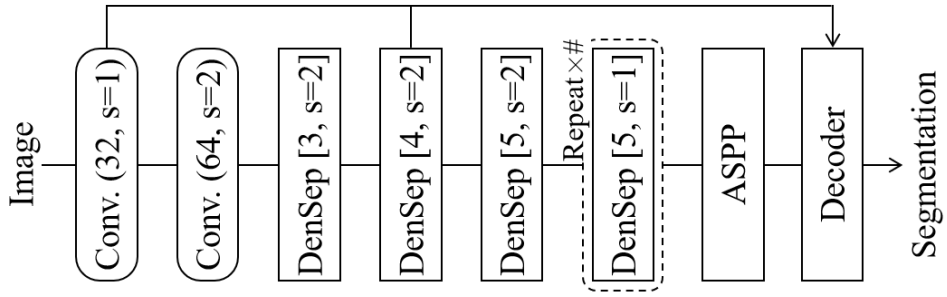


Figure 4.6: Schematic diagram of SDDNet architecture [Choi and Cha (2020)]

The purpose of atrous spatial pyramid pooling (ASPP in Figure 4.6) is extracting multi-scale features. It was slightly modified from the original ASPP (Chen et al., 2018), and the corresponding details are provided in Chapter 4.2.4. The spatial dimension of input feature maps at the ASPP module is about 16 times smaller than the images due to the four strides (the second Conv. and the first to third Densep modules). Accordingly, the decoder module was configured for restoring the spatial dimension using low-level features from the first standard convolution and the second DenSep module.

Several common settings are as follows: the filter sizes of the standard convolutions and depth-wise convolutions are set at 3×3; point-wise convolutions are equal to the standard convolution with a filter size of 1×1; all convolution operations except for the last are followed by BatchNorm (Ioffe and Szegedy, 2015); zero paddings are applied to the input feature maps of all the standard and depth-wise convolution; ReLU (refer to Chapter 2.6) is chosen as an activation function.

4.2.3 DenSep module

The DenSep module is the core of the proposed model. It conducts separable convolutions several times, each of which performs point-wise (PW) and depth-wise (DW) convolutions. The densely connected separable convolution module having the profile of '[4, s=2]' is illustrated in Figure 4.7 as an example.

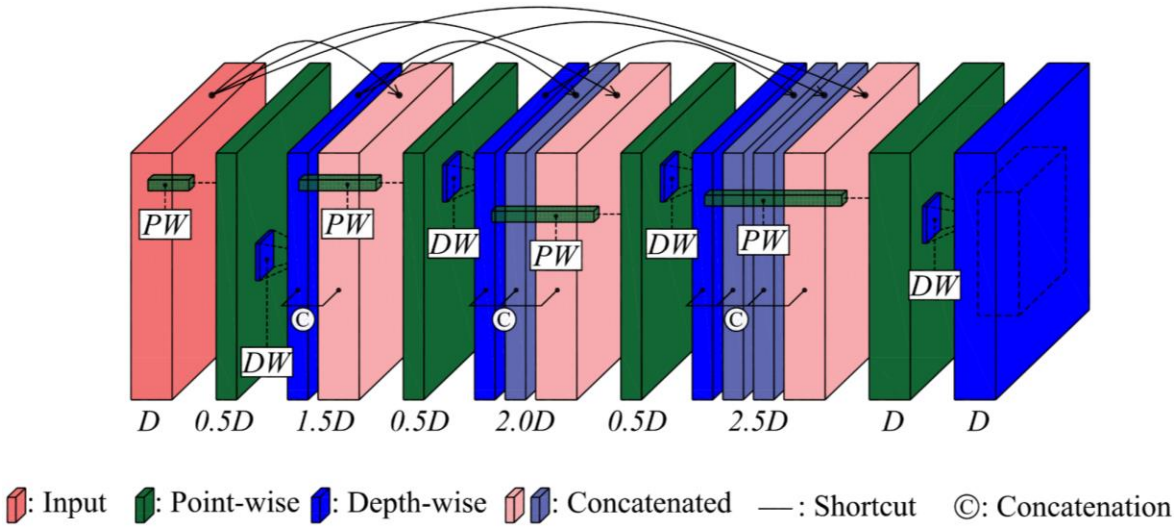


Figure 4.7: DenSep module [reproduced from Choi and Cha (2020)]

The primary advantage of a separable convolution is its efficiency because it involves a much smaller number of computations compared to the standard convolution. The intended usage was to apply filters in the order of DW and PW, but DenSep module is designed to apply PW before DW. The purpose of order switching is to squeeze a feature map with PW convolution to further reduce computational cost. The output feature maps of separable convolutions within a DenSep module are concatenated, which resembles the Dense Block of DenseNet (Huang et al., 2017). In the DenSep modules, the last DW convolutions only involve strides to reduce spatial dimensions. Rectified linear units are applied after all DW convolutions. An additional interpretation of the DenSep module is discussed in Appendix A.

Figure 4.8 depicts the differences of convolution operations (i.e., PW, DW, and the standard convolutions), in which the 1×1 convolution is identical to PW convolution. Convolutions fundamentally involve weighted sums (refer to Chapter 2.1), in which the multiplications are the major computations. Therefore, the number of multiplications can directly be considered as the computational costs, and the costs of the standard, PW, and DW convolutions are given by Eqs. (4.1) – (4.3). In the equations, D_{in} and D_{out} imply the input and output depth of the feature map, respectively. Separable convolution performs

PW and DW convolutions sequentially; therefore, computational costs of a separable convolution is equal to the sum of Eqs. (4.2) and (4.3).

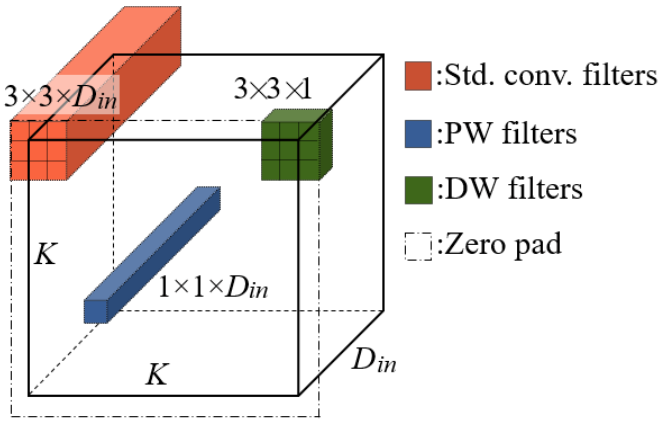


Figure 4.8: Comparison of convolution operations [Choi and Cha (2020)]

$$9 \times K^2 \times D_{in} \times D_{out} \tag{4.1}$$

$$K^2 \times D_{in} \times D_{out} \tag{4.2}$$

$$9 \times K^2 \times D_{out} \tag{4.3}$$

The number of multiplications associated with the convolution operations can be calculated based on the above equations as shown in Table 4.1. As the results demonstrate, there is an approximately 70% reduction in computational cost compared with the standard convolution module.

Table 4.1: Number of computations [Choi and Cha (2020)]

Sequence	DenSep		Standard convolution	
	Operation	Multiplications	Operation	Multiplications
1	PW	$0.5K^2D^2$	PW	$0.5K^2D^2$
	DW	$4.5K^2D$	Conv.	$2.25K^2D^2$
2	PW	$0.75K^2D^2$	PW	$0.75K^2D^2$
	DW	$4.5K^2D$	Conv.	$2.25K^2D^2$
3	PW	K^2D^2	PW	K^2D^2
	DW	$4.5K^2D$	Conv.	$2.25K^2D^2$
4	PW	$1.25K^2D^2$	PW	$1.25K^2D^2$
	DW	$4.5K^2D$	Conv.	$2.25K^2D^2$
5	PW	$3.0K^2D^2$	PW	$3.0K^2D^2$
	DW	$4.5K^2D$	Conv.	$9.0K^2D^2$
Total	-	$K^2D(6.5D+27)$	-	$K^2D(24.5D)$

4.2.4 ASPP module

A large filter size can indeed capture features due to its large FOV, which however increases the computational costs exponentially. For example, suppose there is a filter with the size of $m \times m$. Another filter with c times more FOV than that of the $m \times m$ filter has the size of $cm \times cm$, which has c^2 times greater number of weights. Chen et al. (2018) solved the dilemma by introducing the ASPP module; a modified version of ASPP is adopted in the proposed model.

The original ASPP is composed of the global average pooling (GAP), PW convolution, and three atrous separable convolutions (ASC) in the standard order (i.e., DW first and PW later). The dilated DW filter is the only difference from the standard separable convolution, as depicted in Figure 4.9.

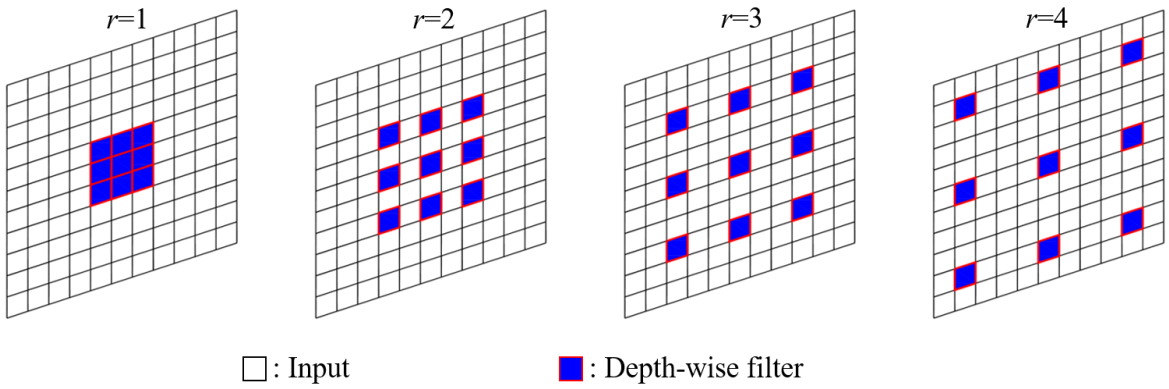


Figure 4.9: Dilated depth-wise convolution filter [Choi and Cha (2020)]

In this figure, the dilation rate (denoted as '*r*') indicates the number of skips to the neighboring elements of a filter. Each of the dilated DW filters can extract features while it sees a large FOV. For example, a dilated filter with a rate of 2 can extract features from the FOV of 5×5, but the filter still has 9 weights rather than 25 weights. Hence, the ASC with a dilation rate of greater than one can be considered as that filter captures via a larger FOV without increasing computational costs. In addition, multi-scale feature maps can be obtained by aggregating the feature maps of the ASCs with various rates.

The modified ASPP module is configured as follows: GAP is not adopted because it strongly regularizes a CNN model, whereas the proposed model has a significantly small number of weights that result in performance deterioration; PW convolution is kept as original; four ASC modules (three in the original ASPP) with different dilation rates from the original ASPP are included; each of the ASC modules performs the reversed order of the standard ASC, in which the dilation rates of DW filters are set at 1 to

4, respectively. Figure 4.10 demonstrates how the modified ASPP is configured, in which the output feature maps of PW and ASC are concatenated (denoted as ©), and the concatenated feature map is aggregated by another set of PW filters to generate multi-scale features. The multi-scale feature maps are activated by the ReLU and the dropout layer is followed to prevent overfitting; the rate was set at 0.5 in training.

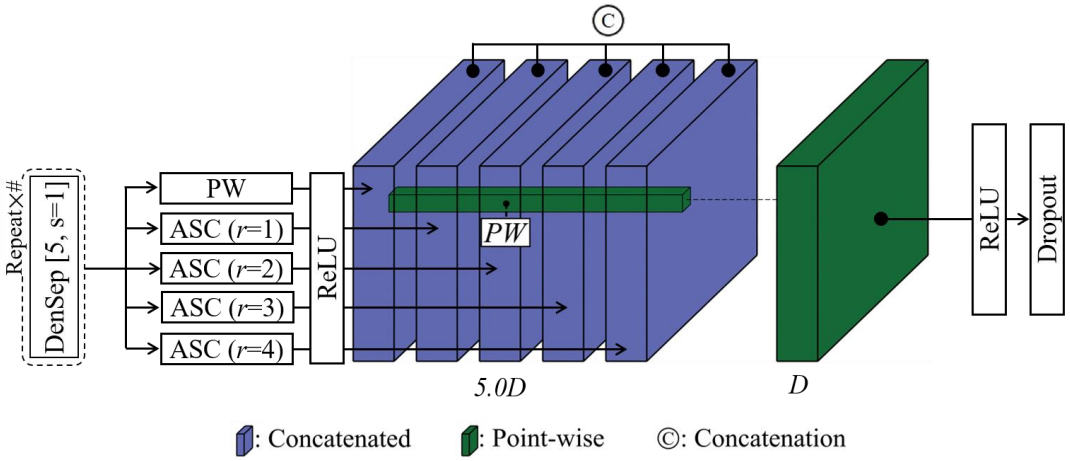


Figure 4.10: Modified ASPP module [reproduced from Choi and Cha (2020)]

4.2.5 Decoder module

One of the main objectives of this study is to obtain fine-grained segmentation. However, the spatial dimensions are gradually reduced by applying strides four times (refer to Chapters 4.2.2 and 4.2.3), and the model returns 16 times smaller size of feature maps. Accordingly, extracting segmentations from the small feature maps causes coarse segmentation results. Hence, the decoder module was designed to restore the spatial dimension of the small feature maps, as described in Figure 4.11.

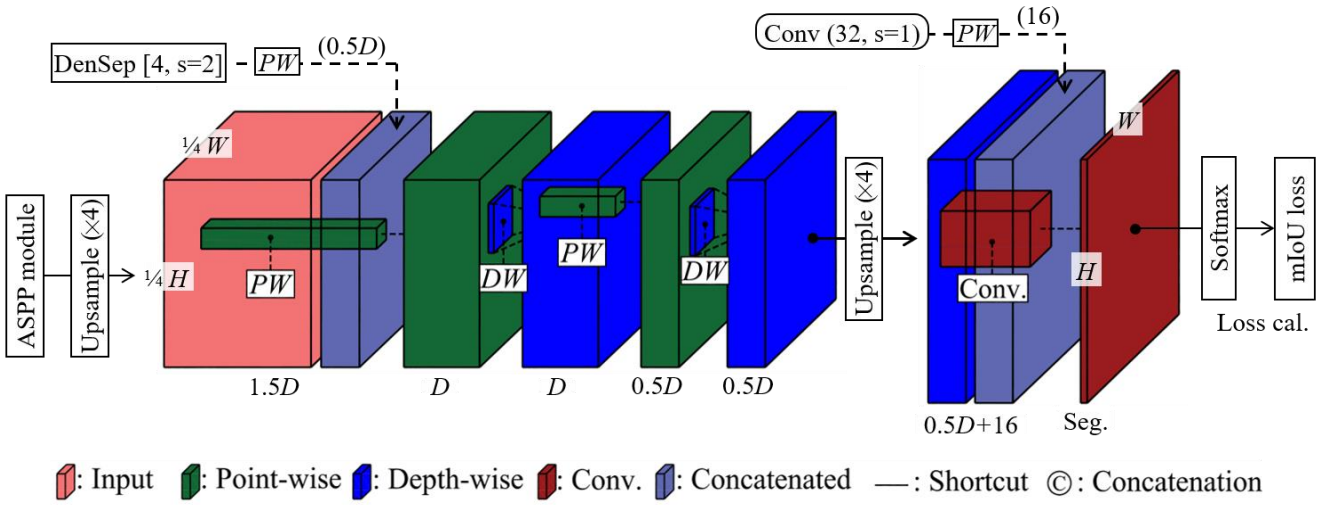


Figure 4.11: Details of the decoder module [reproduced from Choi and Cha (2020)]

The module was designed to restore the spatial dimension two times. In the first step, the feature maps extracted by the modified ASPP module (refer to Chapter 4.2.4) was up-scaled by the scale factor of 4. Then, the scaled feature maps were concatenated with the low-level feature maps extracted by the last PW convolution in the second DenSep module (refer to Figure 4.6), which has 1/4 spatial dimension of the input images. Thereafter, two separable convolutions were applied to the concatenated feature maps and up sampled by the scale factor of 4 and concatenated again with another low-level feature maps from the first standard convolution layer (refer to Figure 4.6), which has the same spatial dimension of the input images. Finally, the last standard convolution generated the segmentation (denoted by Seg. in Figure 4.11) results. In the training phase, the model was optimized by Adam (refer to Chapter 4.4.2), and the mean intersection-over-union (mIoU) loss (refer to Chapter 4.4.3) was used for calculating the loss of the model (denoted by Loss cal. in Figure 4.11).

4.2.6 Model customization

The SDDNet can be customized by setting different number of repetitions (refer to Figure 4.6) of the DenSep module and values of *D* (refer to Figure 4.7). Hereafter, customized models are referred to as SDD-*R*#*D*#, in which *R*# implies the number repetitions and *D*# indicate the *D* values. In this chapter, two models are presented: SDD-R6D64 was trained to achieve the objectives (refer to Chapter 4.1) and considered as the final model; SDD-R6D32 was trained for comparative studies, which was compared with DeepCrack (Liu et al., 2019). Note that the number of PW filters in the DenSep module with six repetitions was set at *D* instead of 0.5*D* (refer to Figure 4.7) for the customized models reported in this chapter.

4.3 Dataset generation

The SDDNet was trained in a supervised learning scheme, which requires an image-label-paired dataset. However, public datasets at present consist of mostly monotonous images, which is not suited to the specific objective (refer to Chapter 4.1), and so a dataset was manually created. The dataset consists of 200 digital images with various sizes: 55 images were taken by smartphones; 104 and 41 images were downloaded from Datacenterhub (Sim et al., 2017) and Google Images, respectively; 55 images were manually taken by smartphones. Due to the variance of image sources, images were taken under various conditions in terms of distance, luminance, FOV, image quality, and other factors. The size of images in widths and heights are between 513 and 1920 pixels (513×513 at minimum and 1920×1080 at maximum).

The segmentation labels (ground truths) were manually annotated by marking pixels, in which the pixels that belonged to cracks were labeled 1, and any other pixels (background) were labeled 0 (refer to Figure 4.3). Affinity Photo (https://affinity.serif.com/en-gb/photo), a commercial software application for

photo editing, was used to generate the ground truth. The crack features in the dataset can be categorized by their characteristics as follows:

- Thick crack: a crack with thickness of 5 or more pixels.
- Thin crack: a crack with a thickness of 1-4 pixels
- Blurry crack: either a thick (i.e., thick-blurry) or thin (i.e., thin-blurry) crack with blurry features, which are still highly recognizable.
- Faint crack: a crack that is not recognizable without careful observations.
- Crack-like feature: a feature which resembles a crack without peripheral information (refer to Chapter 4.2.1).

The total number of pixels belonging to the crack was about 260 times less than that of pixels belonging to the background, and any CNN models may face the challenge of being optimized. The dataset was split into two subsets: 160 images for training and 40 images for testing. Hereafter, the dataset is referred to as Crack200.

4.4 Training details

This subchapter describes the details of how the SDDNet models were trained. The SDD-R6D64 was pretrained (refer to Chapter 4.4.4) on the Cityscape Dataset before training on the Crack200 (refer to Chapter 4.4.5), in which the Cityscape Dataset was modified to have six classes. The specifications of the infrastructure used for training the models are as follows:

- CPU: Intel Core i7-6850K
- GPU: Nvidia Titan XP × 4ea
- Memory: 128 GB

The GPUs were used only in training the model, however; an old GPU, Nvidia Titan X, was instead used in testing to highlight real-time performance (refer to Chapter 4.4.7).

4.4.1 Training strategy

The input size in the beginning of training was set at 513×513 pixels. If an image had a larger size, the image was randomly cropped by the specified size. Considering the largest image size is about 1920×1080, the cropping size, which is approximately 1/4 compared to the largest image, was reasonable as the cropped region may still contain enough context for recognizing objects. Additionally, the specific number was beneficial at the layer where strides were applied. After the model converged, the cropping size was changed to 1009×1009 to lead the dilated DW weights in the modified ASPP module working well on large images in testing. The model can be tested on any image size without restrictions because the proposed model follows the conventions of a fully convolutional neural network (refer to Chapter 4.2). The explained strategies were commonly applied in both pretraining (refer to Chapter 4.4.4) on the modified Cityscape Dataset and finetuning on the Crack200 (refer to Chapter 4.4.5).

4.4.2 Adaptive momentum optimizer

One of critical hyperparameters (refer to Chapter 3.6.1) in training DL models is learning rate (refer to Chapter 2.4.1), but defining an optimal learning rate has been a cumbersome issue. The momentum algorithm (refer to Chapter 3.5) can make training a DL model less challenging, but the algorithm introduces an additional hyperparameter to be empirically set. Adaptively applying learning rates for each weight at each update might greatly mitigate the cumbersome issue, and an adaptive momentum (Adam) optimizer was proposed by Kingma and Ba (2014), which was a combined form of the momentum algorithm (refer to Chapter 3.5) and RMSProp (Tieleman and Hinton, 2012). The overall computing process of the Adam optimizer is as follows: 1) the moving averages η and r for $\nabla_{\phi}\mathcal{J}$ and $\left(\nabla_{\phi}\mathcal{J}\right)^2$ are

calculated by Eqs. (4.4) and (4.5), respectively. β_1 and β_2 are exponential decay rates that are set at 0.9 and 0.999, respectively by default; 2) η and r are biased towards zero during initial training steps, so that those are corrected by Eqs. (4.6) and (4.7); 3) a weight is updated by Eq (4.8), where α is learning rate, and ε is a small constant to preserve numerical stability.

$$\eta \leftarrow \beta_1 \eta + (1 - \beta_1) \nabla_\phi \mathcal{J}, \tag{4.4}$$

$$r \leftarrow \beta_2 r + (1 - \beta_2) \left(\nabla_\phi \mathcal{J}\right)^2, \tag{4.5}$$

$$\hat{\eta} \leftarrow \frac{\eta}{1 - (\beta_1)^t}, \tag{4.6}$$

$$\hat{r} \leftarrow \frac{r}{1 - (\beta_2)^t}, \tag{4.7}$$

$$\phi \leftarrow \phi - \alpha \frac{\hat{\eta}}{\sqrt{\hat{r}} + \varepsilon}, \tag{4.8}$$

4.4.3 Cost function and hyperparameters

The cross-entropy loss function (refer to Chapter 3.4) has been widely used in classification and segmentation; however, this is not a good choice when training a model on an imbalanced dataset, and the Crack200 is one of the most extreme cases of this. Accordingly, Rahman et al. (2016) proposed the intersection-over-union (IoU) loss which can address the issue, but it was not applicable to pretraining on the model of the modified Cityscape Dataset (refer to Chapter 4.4.4) because the dataset has 6 classes while the IoU loss function only works for binary classes. Hence, the IoU loss function is modified to the mIoU cost function, which is defined by Eq. (4.9) corresponding to a mini-batch size of *m* with a *C* number of classes. *I* (Eq. (4.10)) and *U* (Eq. (4.11)) are the intersection and union calculated by the probability maps of *y* from the Softmax function and the corresponding one-hot-encoded ground truths of $\hat{y}$, respectively. The mIoU score is one of the most frequently used metrics for reporting segmentation

performances, and the score can be calculated by disregarding the "1-" term and substituting *y* for one-hot-encoded prediction from Eq. (4.9).

$$\mathcal{J}_{mIoU} = 1 - \frac{1}{C}\frac{\sum_{i=1}^{m} I_i}{\sum_{i=1}^{m} U_i} \tag{4.9}$$

$$I_i(y, \hat{y}) = y_i \cdot \hat{y}_i \tag{4.10}$$

$$U_i(y, \hat{y}) = y_i + \hat{y}_i - I_i \tag{4.11}$$

The Adam optimizer was used for training (refer to Chapter 4.4.2). The learning rates were scheduled over training iterations (*iter*) based on the cyclical learning rate policy proposed by Smith (2017), and it can be calculated by Eq. (4.12) and Eq. (4.13). The hyperparameters used for training the SDD-R6D64 are summarized in Table 4.2.

$$lr_2 + (lr_1 - lr_2) \times \max(0, 1 - \Omega) \times \tau^{iter} \tag{4.12}$$

$$\Omega = \left| \frac{iter}{step} - 2 \times \left\lfloor \frac{1+iter}{2 \times step} \right\rfloor + 1 \right| \tag{4.13}$$

Table 4.2: Hyperparameters for training SDD-R6D64 [Choi and Cha (2020)]

	Pretraining on the Cityscape	Finetuning on the Crack200
lr_1 / lr_2 / step	0.003 / 0.00001 / 2000	0.001 / 0.00001 / 2000
Input size	513×513 (1009×1009[a])	513×513 (1009×1009[a])
Mini-batch size	32(8[a])	32(8[a])
τ	0.99996	0.99996
Weight decay	0.00004	0.00004

a: These are the hyperparameters used after the convergence was noticed while in training on small images (refer to Chapter 4.4.1).

4.4.4 Pretraining on the Cityscape Dataset

The SDD-R6D64 was pretrained on the modified Cityscape Dataset before training the model on the Crack200. The Cityscape Dataset (https://www.cityscapes-dataset.com/) is intended for evaluating a vision algorithm in recognizing urban scenes, and it provides 3475 images with the corresponding ground truths of the scene taken from various cities in the world. The dataset was chosen for pretraining to verify the trainability of the proposed model on a multi-class dataset. In addition, the wide range of features might be beneficial in negating non-crack features from images in the Crack200. However, the primary objective was to effectively segment a few classes, while the Cityscape Dataset has 31 classes. Hence, the Cityscape Dataset was modified to have 6 classes; similar objects were combined into the same classes; some of object labels residing a large space in images were renumbered by zeros (i.e., background). Accordingly, the modified Cityscape Dataset was relabeled, as summarized in Table 4.3.

Table 4.3: Modified Cityscape Dataset [Choi and Cha (2020)]

Label	1	2	3	4	5	6	0
Object	Road, sidewalk, parking	Building, wall, bridge	Pole, traffic, light, sign	Tree	Human	Vehicle	Others

The SDD-R6D64 was pretrained for about two days, and the training profile with the simple-moving-average (SMA) is provided in Figure 4.12. The input size was set at 513×513 for about 100K iterations but changed to 1009×1009 thereafter until 180K iterations. The model at each iteration has different weights, and there were no clear indications to identify which set of weights at a certain iteration is the best for fine-tuning on the Crack200. Therefore, it was assumed that any set of weights within a period of convergence has comparably same contributions to fine-tuning. As Figure 4.12 demonstrates, a convergence was noticed between 140K and 180K, where the period (40K iterations) was sufficient for

feeding the entire dataset into the finetuning model considering the mini-batch size of 8 with the input dimension of 1009×1009. One of the models within the period of convergence was chosen as a pretrained model, and all of the weights except the weights of the last PW filters in the decoder (refer to Figure 4.11) were directly adopted in fine-tuning.

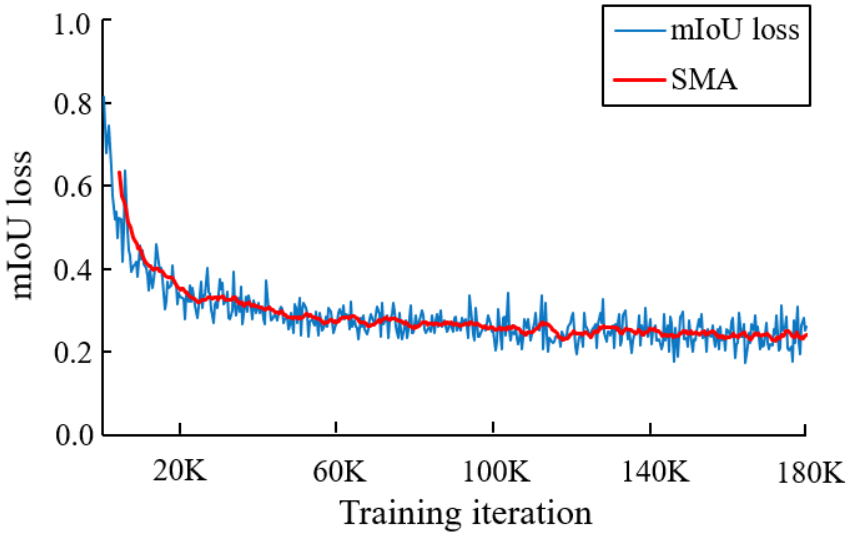


Figure 4.12: mIoU loss over training iteration [Choi and Cha (2020)]

4.4.5 Training on the Crack200

The pretrained model (refer to Chapter 4.4.4) was further finetuned (refer to Chapter 4.4.5) on the Crack200 (refer to Chapter 4.3) according to the training procedures (refer to Chapter 4.4.1). The finetuned model was tested on the Crack200's test set. Figure 4.13 displays the mIoU scores of both the finetuned SDD-R6D64 and the model trained from scratch over 40K iterations. The total duration of finetuning and training from scratch on the Crack200 was approximately six hours. As the figure demonstrates, the finetuned model slightly outperformed all training iterations compared to the model trained from scratch. The finetuned model was converged at the mIoU of approximately 0.8, as depicted in Figure 4.13; the input size was changed to 1009×1009 pixels and further finetuned for 10K more iterations. As a result, a slightly better mIoU score of 0.846 was obtained, as summarized in Table 4.4; the table also includes other

evaluations metrics that are calculated by Eqs. (4.14) – (4.16). In the context of crack segmentation, precision indicates the fraction of true crack pixels among the pixels that are predicted as crack pixels. Recall indicates the fraction of true crack pixels over all ground-truth pixels of cracks. F1 score is a harmonic average of precision and recall. mIoU indicates the similarity between predicted crack pixels to the ground-truth pixels of cracks over classes (crack and background). The model that achieved the highest mIoU score was considered to be the final model, and the corresponding results are presented with a comprehensive visualization in the next subchapter.

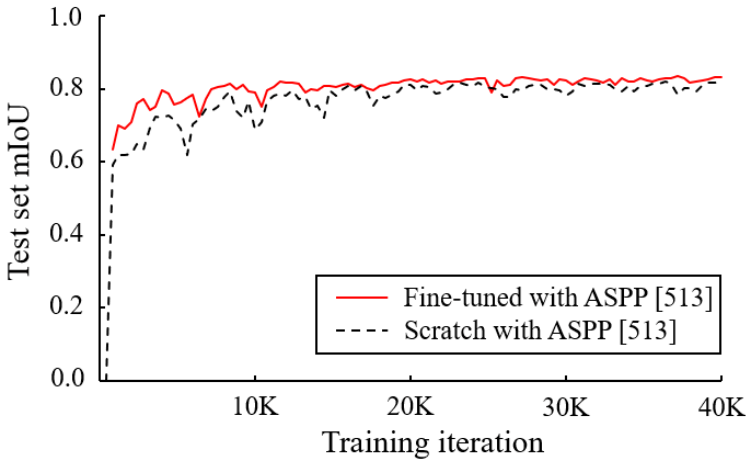


Figure 4.13: Comparative profiles of SDD-R6D64 using two different training approaches [Choi and Cha (2020)]

Table 4.4: Evaluation metrics of SDD-R6D64 with different training strategies [Choi and Cha (2020)]

Model variation	Input size	Precision	Recall	F1	mIoU
Fine-tuned with ASPP	1009×1009	0.805	0.834	0.819	0.846
	513×513	0.770	0.839	0.803	0.835
Fine-tuned without ASPP	1009×1009	0.752	0.855	0.800	0.833
	513×513	0.756	0.844	0.798	0.831
Scratch with ASPP	513×513	0.785	0.802	0.793	0.827

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}} \tag{4.14}$$

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} \tag{4.15}$$

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.16}$$

4.4.6 Segmentation results

This subchapter presents several test results, where the segmentation results are demonstrated in accordance with the following rules. Each panel of image results are denoted by "Raw," and "Seg." for raw images and the segmentation results superimposed on the raw images, respectively. True positive (TP), false negative (FN), and false positive (FP) pixels are represented by red, green, and blue pixels, respectively. The pixels of TP (red) and FN (green) are the ground truths. The pixels of TP and FP are the pixels predicted as cracks by the final model. The caption of each result image indicates the mIoU score, in which the bracketed number alongside of the caption is the size of image. The regions where crack features reside are considerably smaller than images, and the results are barely noticeable. Therefore, such regions are enlarged for better visualization and marked as region-of-interest (ROI), which may enable fully exposing the characteristics of both merits and demerits.

In accordance with the visualization rules, several representative results of the final model are presented in Figures 4.14 to 4.17. The proposed model effectively negated the features from irrelevant objects, although the tested images had complex crack-like features. Furthermore, the proposed model could detect either thick and thin cracks lying on a few pixels. Common failures were also analyzed: 1) FN predictions were mostly observed around TP pixels. However, such false predictions are acceptable due to the fact that the definitive ground truths of objects cannot be obtained, which is particularly true in

the case of fine cracks; 2) the proposed model was unable to segment faint cracks, such as those displayed in Figures 4.15-ROI-1, 4.16-ROI-1, 4.16-ROI-3, and 4.17-ROI-1; 3) the proposed model also could not segment the cracks on spalling regions, as depicted in Figure 4.16-ROI-2. Other test results can also be viewed online (https://github.com/choiw-public/SDDNet).

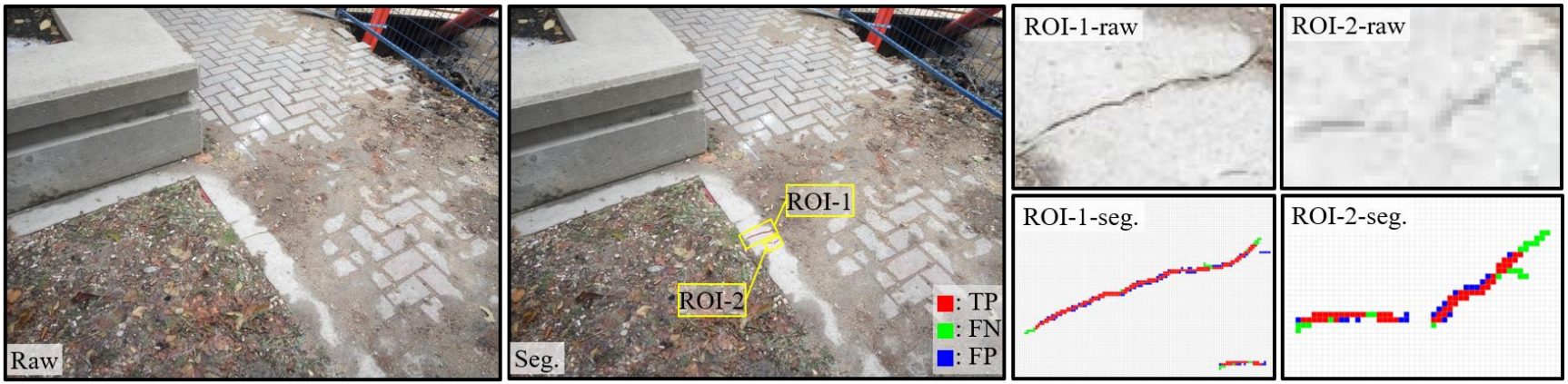


Figure 4.14: Representative test result (1) – mIoU= 0.830 [1920×1440] [Choi and Cha (2020)]

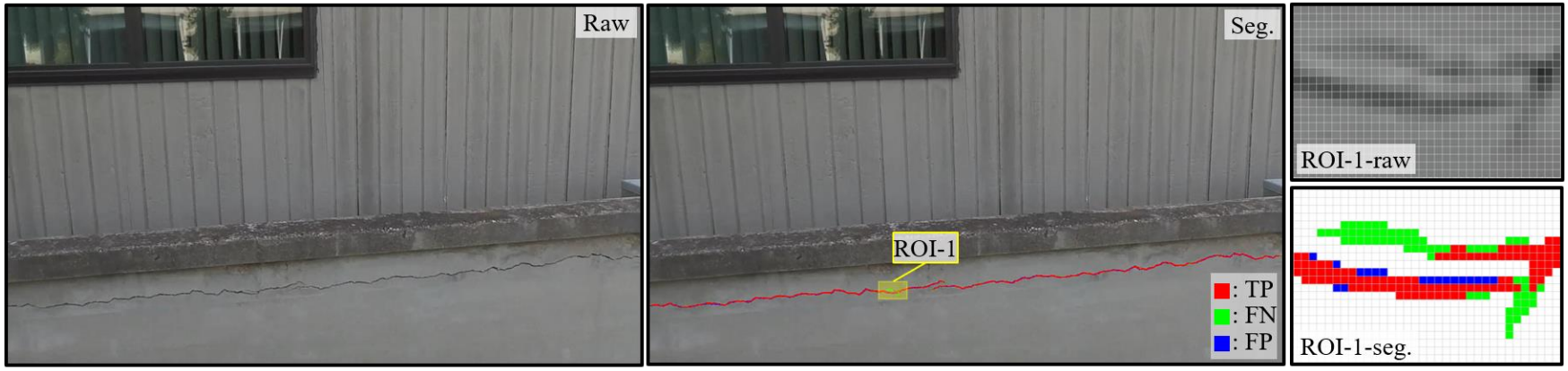


Figure 4.15: Representative test result (2) – mIoU = 0.909 [1280×720] [Choi and Cha (2020)]

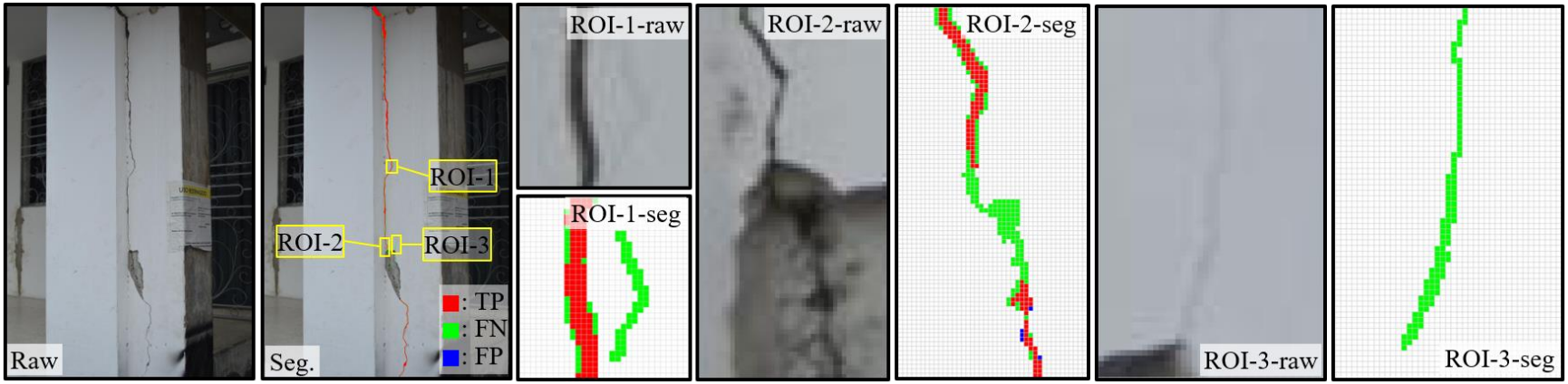


Figure 4.16: Representative test result (3) – mIoU= 0.898 [1276×1920] [Choi and Cha (2020)]

Figure 4.17: Representative test result (4) – mIoU= 0.828 [1920×1275]

4.4.7 Comparative studies and discussions

This subchapter provides the results from comparative studies to prove the following claims: 1) developing a model dedicated to a specific purpose is highly beneficial (refer to Chapter 4.2); 2) a model should not be trained on images with monotonous features to adapt the model to real practice (refer to Chapter 4.1).

To support the first claim, the SDDNet was compared with the DeepCrack (Liu et al., 2019), which is the latest model for crack segmentation at the time this study was conducted. The DeepCrack composers reported the evaluation metrics of six DeepCrack models, and these were also compared to other segmentation models reported earlier. The authors claimed that the DeepCrack models outperformed any of the other models, in which each of DeepCrack models had different merits. Among these models, only the best models were selected to be compared. To fully demonstrate the efficiency of the SDDNet, a customized model, SDD-R6D32, was additionally configured. The number of weights of the SDD-R6D32 was about 3 times smaller than that of the full model (i.e., the SDD-R6D64).

The core of the DeepCrack models were configured based on VGG-16 (Simonyan and Zisserman, 2014), which aimed to classify images into 1K classes, and there are about 14 million parameters in DeepCrack models. On the contrary, the compared SDDNet model had only 0.16 million parameters, which was 88 times smaller than any of the DeepCrack models. To accomplish comparisons under fair conditions, the following environments were set up: 1) DeepCrack's train and test sets were used in training and testing, respectively; 2) the SDD-R6D32 was trained from scratch; 3) the same GPU model was used for comparing testing times. The comparison results are summarized in Table 4.5. The DeepCrack models were expected to achieve better evaluation metrics due to a much greater number of weights compared to the SDD-R6D32. However, the SDDNet model produced better results in every aspect. In terms of processing time, the fastest DeepCrack model processed 9 images (544×384 pixels)

per second, while the SDD-R6D32 processed 76 images per second under the same testing environments, which was at least 8 times faster processing than DeepCrack models.

Table 4.5: Evaluation metrics of SDD-R6D32 and DeepCrack [Choi and Cha (2020)]

Model variation	Parameters	Precision	Recall	F1	mIoU	FPS
DeepCrack-CRF		0.868	0.846	0.857	0.836	2.500
DeepCrack-GF	14	0.852	0.866	0.859	0.859	8.475
DeepCrack-Aug		0.861	0.869	0.865	0.802	9.174
SDD-R6D32	**0.160**	**0.871**	**0.870**	**0.870**	**0.879**	**75.816**

An additional experiment was conducted to support the second claim. In this comparison, the trained DeepCrack models were not considered because the previous experiment (refer to Table 4.5) revealed that the SDD-R6D32 already outperformed all DeepCrack models. Accordingly, the SDD-R6D64 trained on the DeepCrack's dataset were compared with the SDD-R6D64 trained on the Crack200. The properties (e.g., number of weights, etc.) of the compared models in this experiment were identical except for the dataset. The SDD-R6D64 has about 0.5 million parameters, which was three times greater than the SDD-R6D32. However, it is still 26 times fewer than that of the DeepCrack models. In terms of processing time, the SDD-R6D64 could process 67 images per second, which was at least 7 times faster than all the DeepCrack models. Table 4.6 is the summary of the comparisons, in which the metrics denoted by 'a' demonstrate that the SDD-R6D64 trained on the DeepCrack Dataset returned better results than both all the DeepCrack models (refer to Table 4.5) and the SDD-R6D32 (refer to Table 4.5) trained on the DeepCrack's dataset. The detailed observations of Table 4.6 may provide additional support for the second claim.

Table 4.6: Evaluation metrics of SDD-R6D64 trained and tested on monotonous and complex datasets [Choi and Cha (2020)]

Training set	Tested on the DeepCrack's test set				Tested on the Crack200's test set			
	Precision	Recall	F1	mIoU	Precision	Recall	F1	mIoU
DeepCrack	0.874[a]	0.870[a]	0.872[a]	0.880[a]	0.213[b]	0.561[b]	0.309[b]	0.587[b]
Crack200	0.875[c]	0.870[c]	0.872[c]	0.881[c]	0.805[d]	0.834[d]	0.819[d]	0.846[d]

a: SDDNet model trained and tested on the DeepCrack's train and test sets
b: SDDNet model trained on DeepCrack's training set; tested on Crack200's test set
c: SDDNet model trained on Crack200's training set; tested on DeepCrack's test set
d: SDDNet model trained and tested on Crack200's train and test sets (refer to Chapter 4.3)

- **The models denoted by 'a' and 'b'**: The high evaluation metrics (denoted by 'a' in Table 4.6) cannot be an indication of sufficiency in real practice because a model trained on images of monotonous features likely returns poor results in images of complex features. This can be noticed by comparing models 'a' and 'b.' Specifically, the significant deterioration of precision scores from '0.874[a]' to '0.231[b]' is an indication of extremely high FP predictions. In other words, the model failed to discriminate crack-like features and complex backgrounds.
- **The models denoted by 'c' and 'd'**: The SDD-R6D64 models trained on images of complex features (i.e., Crack200) exhibited consistent performance in testing images of either monotonous or complex features. Focusing on the precision scores (0.875[c] and 0.805[d]), the model could effectively negate crack-like features and complex backgrounds.
- **The models denoted by 'a' and 'c'**: Although model 'a' was trained only on Crack200's training set, it still returned comparable results to model 'c', which was trained on the DeepCrack's dataset, when the models were tested on DeepCrack's test set. In addition, model 'a' also outperformed all the DeepCrack models without using any of the DeepCrack's dataset.

The testing results of the Crack200's test set using the SDD-R6D64 trained on the DeepCrack's dataset are compared with the final model in Figures 4.18 to 4.21. These results clearly demonstrate how inferior a model trained on images of monotonous background is and that training a model on images of monotonous features is not applicable to real practice.

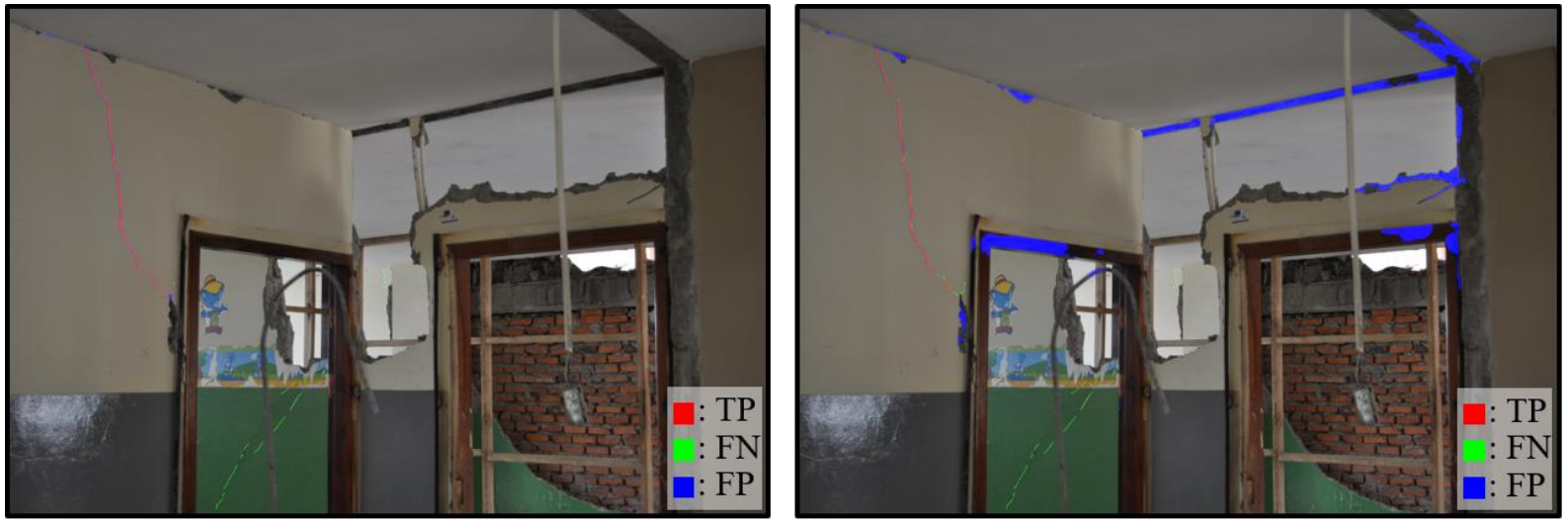


Figure 4.18: Contrast result 1 − models trained on complex background (left) and monotonous background (right)

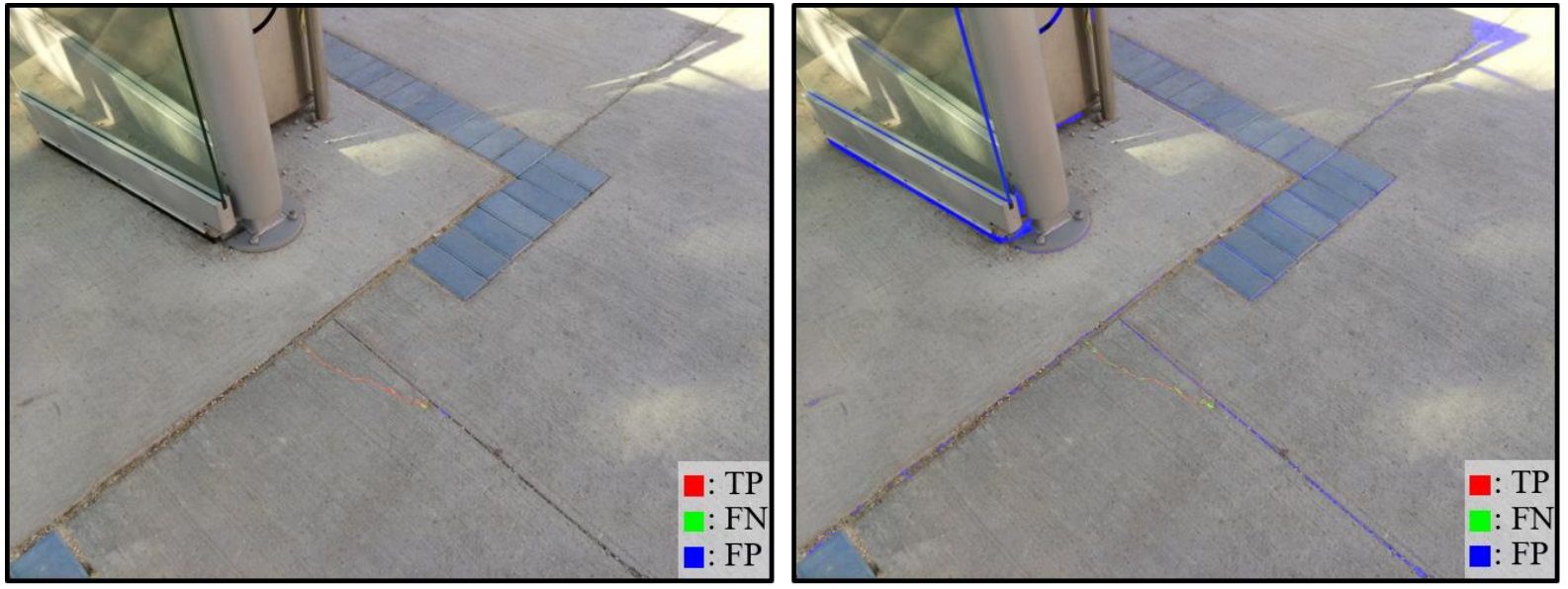


Figure 4.19: Contrast result 2 − models trained on complex background (left) and monotonous background (right)

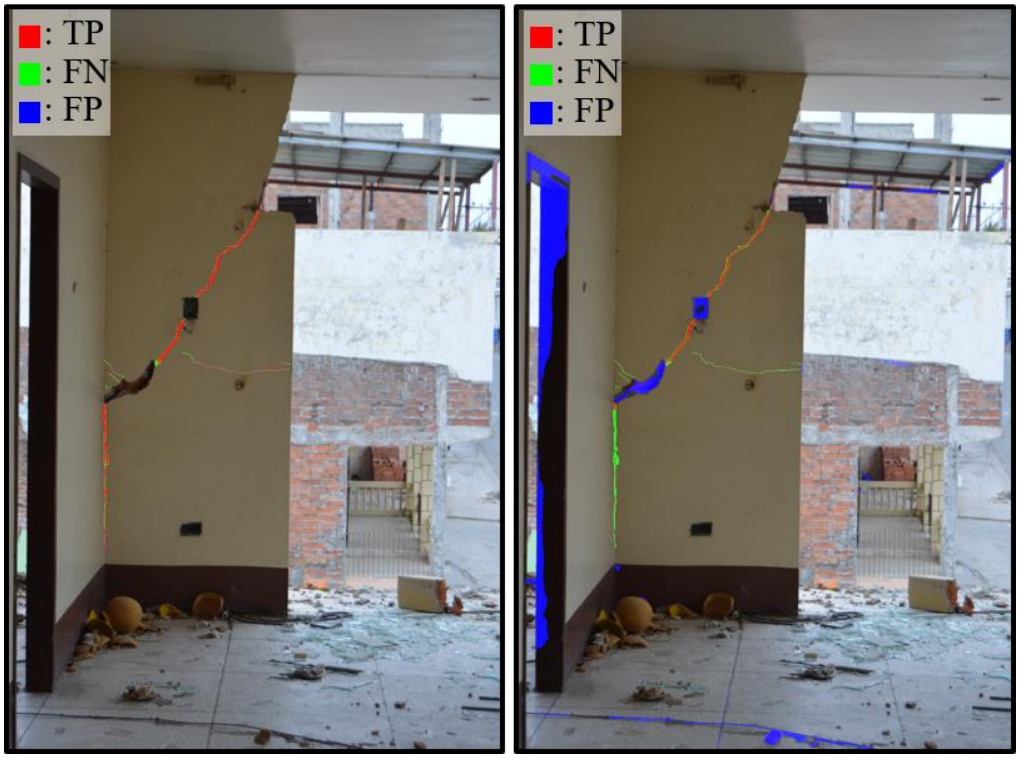


Figure 4.20: Contrast result 3 − models trained on complex background (left) and monotonous background (right)

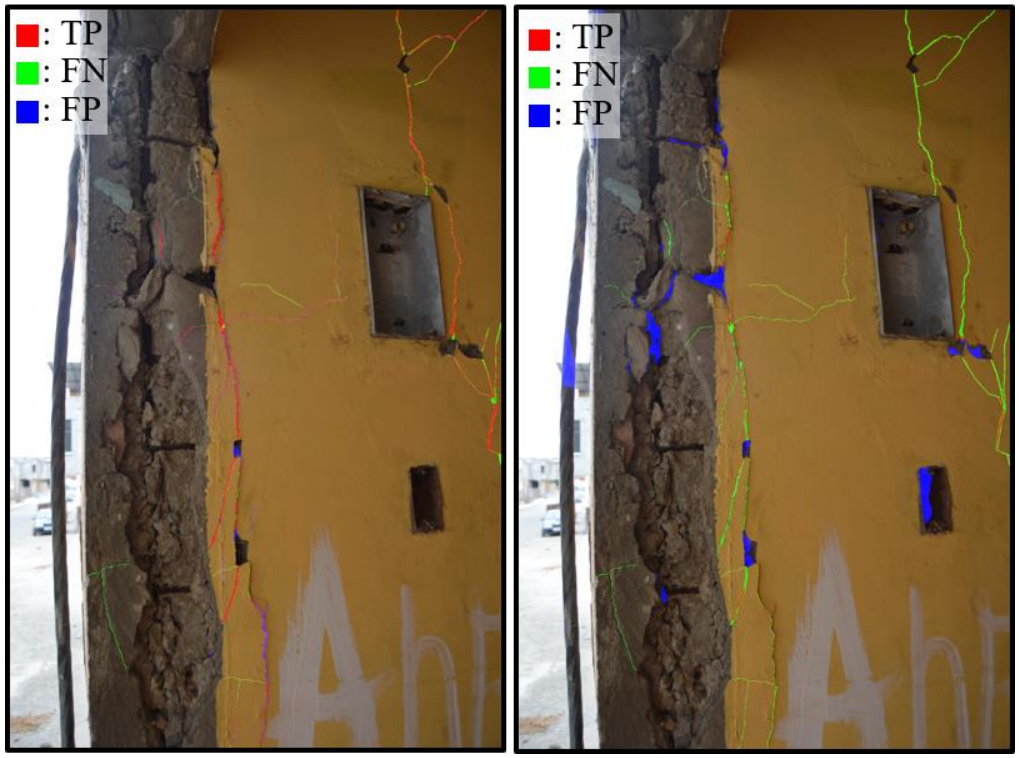


Figure 4.21: Contrast result 4 − models trained on complex background (left) and monotonous background (right)

To highlight the efficiency of the SDD-R6D64 on large images, three different sizes of images were tested, and the results are summarized in Table 4.7. As the results demonstrate, the model could process 12 large images (1920×1080 pixels) in a second. CrackNet-II (Zhang et al., 2018) is another recent model, and the authors reported that their model processed an image of 1025×512 pixels in 1260 milliseconds, while the SDD-R6D64 could process the same size of image in 27.50 milliseconds, which is 46 times faster than the compared model.

Table 4.7: Processing time of SDD-R6D64 [Choi and Cha (2020)]

Image size	Milliseconds/image (FPS)
1920×1080	80.40 (12)
1025×512	27.50 (36)
720×480	18.10 (55)

Another great feature of the SDDNet is that the models require an extremely small volume of storage (3 megabytes for the SDD-R6D64). This will naturally lead to more flexible hardware configurations, such as reducing physical dimensions, lowering technical requirements, and cost savings for deploying a SDDNet model to a mobile device.

Despite of the satisfying performance of the SDDNet on evaluation scores and processing time, the performance of the model was affected by image quality (i.e., faint cracks were not properly segmented; see Chapter 4.4.6). However, this remains as a common challenge for any type of computer vision algorithms rather than the model's specific limitation.

4.5 Conclusion and discussion

This study was conducted to prove the following claims: 1) building a model dedicated to a specific purpose is much beneficial; 2) training a model on monotonous images is not valid in real practice. To prove the claims, a DL model dedicated to structural health monitoring was proposed. The SDDNet was constructed by the combination of the standard convolutions, DenSep modules, a modified ASPP module, and a decoder module. The primary objective was effectively segmenting crack features from a wide range of images with complex features. There were no public datasets that were suited to the specific purpose when this research was under progress, and the Crack200 was manually created. The proposed model was pretrained on the modified Cityscape Dataset prior to finetuning the model on the Crack200. The finetuned model was tested on the Crack200's test set. Several representative results were posted to highlight and expose the advantages and disadvantages as fully as possible. Recent models were compared with the proposed model, and the comparative studies supported the claims with comprehensive results. The highlights of the proposed model can be summarized as follows:

- The proposed model effectively segmented crack features, while negating crack-like features and complex backgrounds.
- The proposed model could segment cracks unless crack features were too faint. Consequently, the model achieved the mIoU score of 0.846 and an F1 score of 0.819.
- The proposed model outperformed recent models in every aspect even though the model had 88 times fewer number of weights than the compared models.
- The proposed model could process images of 1025×512 in real time (36 FPS). This was about 46 times faster than the processing time of a recent model.

- The size of the model was about 3 megabytes, and this property might be greatly beneficial in building an SHM device.

Although this chapter highlighted the effectiveness of the proposed model in several ways, the current state of the model is too early to apply in real practice. However, the unavailability is not only applicable to the proposed model in this chapter, but also any other DL models for segmenting structural damage. This is because the datasets for developing the DL models were too small, and the datasets cannot be said to reflect the real problems. To properly adopt DL models to real practice, a vast amount of images with well annotated ground truths would essentially need to be built, such as 1.28 million images in the ImageNet. Unfortunately, this goal is not what a research team with a few annotators can attain and is the potential subject for further research.

References

Ali, R., & Cha, Y.-J. (2019). Subsurface damage detection of a steel bridge using deep learning and uncooled micro-bolometer. *Construction and Building Materials*, *226*, 376–387. DOI: 10.1016/j.conbuildmat.2019.07.293

Beckman, G. H., Polyzois, D., & Cha, Y.-J. (2019). Deep learning-based automatic volumetric damage quantification using depth camera. *Automation in Construction*, *99*, 114–124. DOI: 10.1016/j.autcon.2018.12.006

Cha, Y.-J., Choi, W., Suh, G., Mahmoudkhani, S., & Büyüköztürk, O. (2017). Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Computer-Aided Civil and Infrastructure Engineering*, *33*(9), 731–747. DOI: 10.1111/mice.12334

Chen, F.-C., & Jahanshahi, M. R. (2018). NB-CNN: Deep learning-Based crack detection using convolutional neural network and Naïve Bayes data fusion. *IEEE Transactions on Industrial Electronics*, *65*(5), 4392–4400. DOI: 10.1109/tie.2017.2764844

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. *Computer Vision – ECCV 2018*, 833–851. DOI: 10.1007/978-3-030-01234-2_49

Choi, W., & Cha, Y.-J. (2019). SDDNet: Real-time crack segmentation. *IEEE Transactions on Industrial Electronics*. DOI: 10.1109/tie.2019.2945265

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. DOI: 10.1109/cvpr.2009.5206848

Dung, C. V., & Anh, L. D. (2019). Autonomous concrete crack detection using deep fully convolutional neural network. *Automation in Construction*, *99*, 52–58. DOI: 10.1016/j.autcon.2018.11.028

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2016.90

Huang, G., Liu, Z., Maaten, L. van der, & Weinberger, K. Q. (2017). Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2017.243

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv preprint arXiv:1502.03167*.

Kang, D., & Cha, Y.-J. (2018). Autonomous UAVs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging. *Computer-Aided Civil and Infrastructure Engineering*, *33*(10), 885–902. DOI: 10.1111/mice.12375

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, R., Yuan, Y., Zhang, W., & Yuan, Y. (2018). Unified vision-based methodology for simultaneous concrete defect detection and geolocalization. *Computer-Aided Civil and Infrastructure Engineering*, *33*(7), 527–544. DOI: 10.1111/mice.12351

Liu, Y., Yao, J., Lu, X., Xie, R., & Li, L. (2019). DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing*, *338*, 139–153. DOI: 10.1016/j.neucom.2019.01.036

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: /10.1109/cvpr.2015.7298965

Nayyeri, F., Hou, L., Zhou, J., & Guan, H. (2018). Foreground–background separation technique for crack detection. *Computer-Aided Civil and Infrastructure Engineering*, *34*(6), 457–470. DOI: 10.1111/mice.12428

Rahman, M. A., & Wang, Y. (2016). Optimizing intersection-over-union in deep neural networks for image Segmentation. *Advances in Visual Computing*, 234–244. DOI: 10.1007/978-3-319-50835-1_22

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2016.91

Ren, S., He, K., Girshick, R. & Sun, J. (2015), Faster r-cnn: Towards real-time object detection with region proposal networks, in *Advances in Neural Information Processing Systems*, 91-99.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science*, 234–241. DOI: 10.1007/978-3-319-24574-4_28

Siam, M., Elkerdawy, S., Jagersand, M., & Yogamani, S. (2017). Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. DOI: 10.1109/itsc.2017.8317714

Sim, C., Villalobos, E., Smith, J. P., Rojas, P., Pujol, S., Puranam, A. Y. & Laughery, L. A. (2017). Performance of low-rise reinforced concrete buildings in the 2016 Ecuador Earthquake. *Purdue University Research Repository*. DOI:10.4231/R7ZC8111

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556*.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. DOI:10.1109/wacv.2017.58

Tieleman, T. & Hinton, G. (2012), Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, *COURSERA: Neural networks for machine learning*, *4*(2), 26-31.

Xue, Y., & Li, Y. (2018). A fast detection method via region-based fully convolutional neural networks for shield tunnel lining defects. *Computer-Aided Civil and Infrastructure Engineering*, *33*(8), 638–654. DOI:10.1111/mice.12367

Yang, X., Li, H., Yu, Y., Luo, X., Huang, T., & Yang, X. (2018). Automatic pixel-level crack detection and measurement using fully convolutional network. *Computer-Aided Civil and Infrastructure Engineering*, *33*(12), 1090–1109. DOI:10.1111/mice.12412

Zhang, A., Wang, K. C. P., Fei, Y., Liu, Y., Tao, S., Chen, C., … Li, B. (2018). 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. DOI:10.1061/(asce)cp.1943-5487.0000775

Chapter 5 Conclusions and Recommendations for Future Research

Summary

Deep learning implemented methods for detecting structural defects were investigated in this thesis. It was presented that the suggested methods were superior to previous works in many different aspects. The model's robust performance was achieved by configuring a domain-specific DL model and training the DL model on a manually created dataset with images of complex features. However, the amount of data used for this research was still small. Therefore, building a significantly larger dataset is recommended prior to continuing this type of research (i.e., damage identification). To realize automated vision-based SHM, multi-temporal 3D reconstruction is worth investigating.

5.1 Conclusions

The reviewed articles in Chapter 1 indicated that the number of infrastructures that have reached or exceeded their designed longevity is growing year by year. Hence, monitoring their conditions has been an on-going and will also be a long-standing issue. However, research articles questioned the effectiveness of the current practice (i.e., human conducted on-site inspection) mainly because of the limited number of human resources. Image processing algorithms (IPAs) were suggested by researchers to establish methods for automated vision-based defect detection. The methods with IPAs were capable of extracting features using image gradient, but their inadaptability to outdoor environments was a critical issue. The succeeding efforts into combining IPAs and traditional ML models were also not successful due to the inheritance of limitations in IPAs. Recently, mobile units (e.g., drones) with IPAs are suggested as modern inspection systems, but the form of inspection still inherits the limitations of IPAs. Therefore, developing adaptive and robust methods that can break the limitations of the current state was set as the primary goal, and DL was considered to be the best option in this thesis.

Chapter 2 provided detailed explanations and backgrounds of DL and its operations that were used in Chapter 3 and Chapter 4: how a digital image is processed; how a DL model can extract features automatically, what are the challenges in training a DL model, how those challenges can possibly be mitigated.

One of the main reasons for the unsuccessful implementations of IPAs was their inadaptability in luminance changes, which initiated the study presented in Chapter 3. A deep learning model was proposed to overcome the challenge. The DL model was trained for achieving robust results against the images of noisy features caused by a wide array of luminance changes. The trained model was combined with a sliding-window technique to enable the method to localize damage regions on large images. The method

was compared with several well-known IPAs, and the results showed that the proposed method could handle the images that were not successfully processed in the IPAs. Additional tests showed that the results of IPAs could be improved by denoising techniques, but the usage was quite limited because the parameters in denoising process should be defined manually for each image, which aspect had no benefits in real-world problems.

Adopting prebuilt models and retraining the models by switching datasets was a commonly applied strategy in this type of topic. However, this study argued the strategy, and a DL model named by SDDNet was independently developed in Chapter 4. In this work, detecting damage was defined as segmenting damage features. By configuring the DL model in the convention of fully convolutional neural network, the size restriction of input images was resolved. The configured architecture could extract multi-scale features by the DenSep and ASPP modules. Accordingly, the model could greatly reduce FP predictions that could not be accomplished in other competitive DL models. The study also demonstrated that developing a DL model dedicated to a task would be a superior approach as there was a significant amount of reduction in calculations while the model performed better.

5.1.1 Implications of the research

This research suggested modern vision-based approaches for detecting structural defects using DL. The major obstacle was the dependency on IPAs, but the DL models presented in Chapters 3 and 4 successfully eliminated the dependency while provided better results than IPAs. This thesis suggested that a DL model should be trained on images with complex features; otherwise, the DL model will fail in real practice unless the process of image acquisition is fully controlled. This may provide new guidance for succeeding works. Reusing a DL model, which was pretrained on generic images, was the dominant practice. However, this study suggested the characteristics of the target objects (i.e., structural defect) are different from generic objects (e.g., human, vehicle, etc.) suggested the necessity of developing a domain-

specific model. This finding may encourage researchers and engineers to independently develop methods dedicated to their disciplines rather than adopting renowned DL models.

5.1.2 Limitations

The classification DL model presented in Chapter 3 showed the potentials of a DL implemented method, but several limitations were noted. The proposed DL model was configured with a FC layer, which restricted the size of inputs. The testing schemes with the sliding-window or similar techniques inevitably led a DL model to focus only on a small region, while damage features can be properly distinguished only if the contexts within a large FOV of images are considered. If a large FOV is selected, the corresponding results would not be meaningful in terms of damage locations. In addition, the testing scheme required repetitive calculations.

The segmentation DL model presented in Chapter 4 surmounted the drawbacks of the DL model presented in Chapter 3. However, the proposed DL model was incapable of segmenting faint cracks because the features of faint cracks were too subtle to be extracted.

5.2 Recommendations for future research

Building a well-annotated large dataset can be a reasonable start to realize a practical vision-based damage detection system before any other actions are made. This is because a number of articles proved the effectiveness of DL implemented applications, but most DL models were trained and tested on only a small number of images (e.g., a few hundred) that most likely reflect only a tiny portion of real problems. However, building a dataset is too laborious (especially segmentation), and a DL model that automatically generates ground truths or minimizes the required labours in annotating images can significantly contribute to pushing this type of research to the next level.

In terms of the performance of DL models, the presented works in this thesis can be further improved by adopting several known DL techniques. The attention mechanism is a particularly promising technique that readily improves the performances of DL models. In addition, more efficient DL models can be developed via automated machine learning (referred to as AutoML), which can possibly replace the exhaustive human-conducted DL model search.

There is mutual a consensus that a camera-equipped mobile unit is a futuristic form of data acquisition to established vision-based infrastructure monitoring. However, mobile units often accompany data corruption (e.g., rolling shutter effect, blur, etc.), and a method for restoring the corrupted data is worth investigating. This is especially meaningful when detecting damage with fine features (e.g., faint cracks).

The researchers working on this type of topic (i.e., damage detection using DL) tend to focus on improving the detectability of structural damage. However, to realize automated vision-based SHM, research on combining all the results and comprehensively interpreting what they indicate (e.g., grades of infrastructures, risk prediction) is highly recommended. This type of work may require tracking the holistic damage presence of infrastructures. Therefore, research on building multi-temporal condition maps of infrastructures in 3D virtual space, which should reflect high-resolution textures to take into account small and subtle features, is also recommended.

Appendix A

This appendix describes how a DenSep module can contribute to extracting features from various FOVs. Consider a CNN without shortcut connections, as shown in Figure A.1, where the illustrated model can be thought of either a CNN with an input of 1-D tensor (see Figure 2.4) or a top-down view of a CNN with an input of 2-D tensor (see Figure 2.5). The red blocks of each layer are convolution operation, where three red blocks (FOV=3) on the left conduct the weighted sum (see Chapter 2.3.2) and return a single feature (red on the right) remapped onto the succeeding feature map. In other words, each feature of a succeeding feature map (darker blue) can be thought of as a representation of features within a FOV of the previous feature map (brighter blue). Without shortcut connections, as shown in Figure A.1, each convolution layer extracts features from a feature map of a fixed size of FOV. On the other hand, a DenSep module can extract features from various FOVs, as shown in Figure A.2. Each PW filter aggregates features (green) from previous feature maps of multiple FOVs, and the represented feature map and the feature maps of previous layers are concatenated. Therefore, features can be extracted from multiple FOVs. Note that shortcut connections for the layers after the second are not drawn for simplicity.

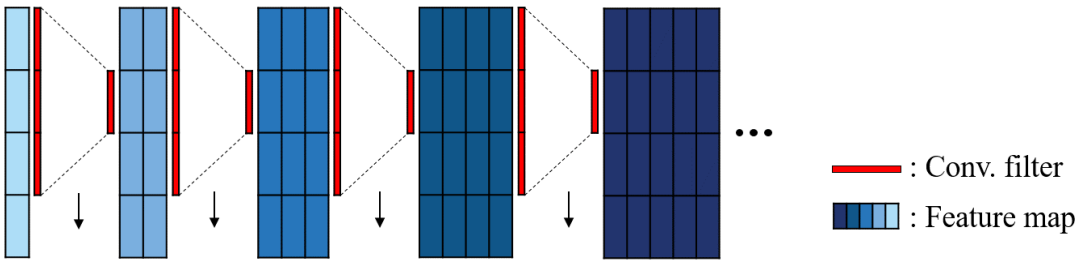


Figure A.1: CNN without shortcut

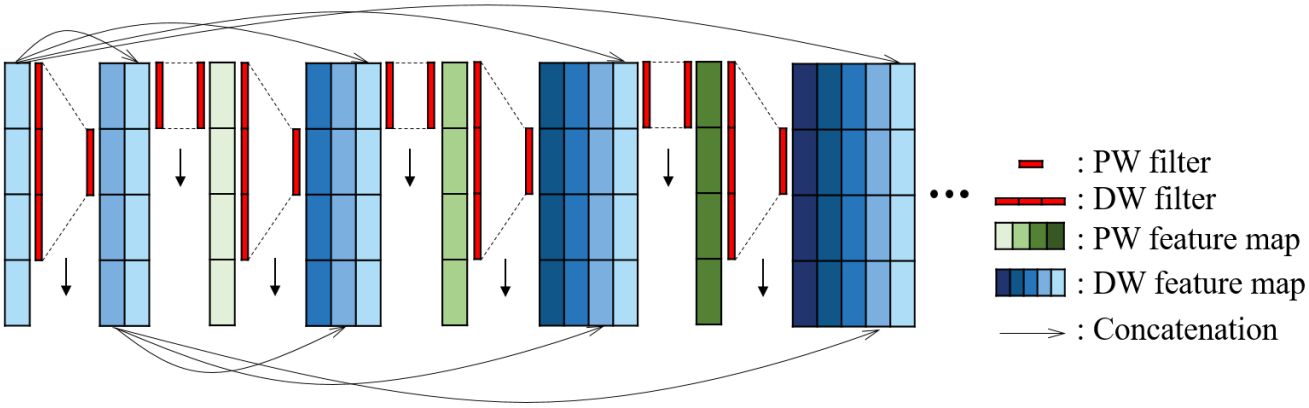


Figure A.2: CNN with shortcut