PIXEL-LEVEL PAVEMENT CRACK DETECTION USING DEEP CONVOLUTIONAL NEURAL NETWORK WITH RESIDUAL BLOCKS

A Thesis presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by

YU HOU
Dr. Zhihai He, Thesis Advisor

MAY 2019

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled:

PIXEL-LEVEL PAVEMENT CRACK DETECTION
USING DEEP CONVOLUTIONAL NEURAL NETWORK WITH RESIDUAL BLOCKS

presented by Yu Hou, a candidate for the degree of Master of Science and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Zhihai He

Dr. Justin Legarsky

Dr. Yuyi Lin

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Dr. Zhihai He for his guidance and supervision. I would also like to thank my committee members, Dr. Justin Legarsky and Dr. Yuyi Lin, for their valuable comments on this research.

I would like to thank my senior lab mates who helped me on this research project, especially Jianhe Yuan. Without their patience, this research could not have been successfully finished.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Nomenclature

$BPNN$	Back propagation neural network
CFD	CrackForest dataset
CPU	Central processing unit
FFA	Free-Form Anisotropy
GPU	Graphics processing unit
LBP	Local binary pattern
$MFCD$	Muti-scale image fusion
MPS	Minimum path selection
$MSTS$	Minimum spanning tree
$ReLU$	Rectified linear unit
RGB	Red Green Blue

ABSTRACT

Road condition monitoring, such as surface defects and pavement cracks detection, is an important task in road management. Automated road surface defect detection is also a challenging problem in computer vision and machine learning research due to the large variety of pavement crack structures, variable lighting conditions, interfering objects on the road surface such as trashes, fallen tree leaves and branches. In this work, we develop a deep learning-based method for automated road surface defect and pavement crack detection. We design a deep convolutional neural network based on using residual blocks to predict the heatmaps which indicate the location and intensity of defects and cracks. To reduce false detection rates, we couple this heatmap prediction network with a binary classification network which is able to determine if the input image patch is normal or has defects. We test our method on the CFD benchmark dataset. Experiment results show that the proposed network is very effective for pavement crack detection and has more advanced performance than other methods.

Chapter 1

Introduction

Monitoring road conditions is important to prevent traffic accidents caused by road surface defects. Cracking is the most common type of pavement distress. Rapid and accurate detection of pavement cracks is helpful for evaluating the integrity of pavement, improving the level of road maintenance, and has important significance for ensuring traffic safety.

Existing pavement crack detection relies on humans to manually extract image features to detect and identify pavement cracks. This method is inaccurate and errors are easy to occur because different people provide different judgments. Therefore, it is difficult to judge the defective road objectively and scientifically. Also human visual inspection consumes a lot of manpower and time. In order to achieve the goal of efficient, rapid and accurate detection of pavement cracks, human visual inspection can be replaced by automatic defect detection methods.

Crack detection is also challenging with image data acquisition due to poor image quality, such as poor lighting conditions, complex surface types, and background texture. With the rapid development of research in crack detection, many open source datasets are accessible, papers are proposed and have significant performance on road defect detection. These papers can be grouped into several categories, such as intensity threshold methods, filter based methods, minimal path selection methods, machine learning methods, and deep learning method. The results depend on the quality of provided datasets as well as the accuracy of human-labeling.

Because the error of human-labeling, also known as manual marking, is unavoidable, using a bounding box to mark the crack seems to be a preferable method. Researchers from Japan provided a pavement crack dataset with hundreds of images collected by smart phones; that identify the coordinates of the bounding box and identify eight different types of cracks for each image. But its localization ability is relatively weak, and it cannot accurately point out the location and may miss some thin cracks. Details are preserved at pixel-level detection, which is a small scale detection. The image may be low-resolution, which has noisy background and topology complexity. The key challenge is how to minimize the effect of those low-resolution images.

To deal with the above challenges, deep learning methods seem to be more effective. A deep network has more feature extraction layers, so it can learn more information. Deep learning methods is able to reduce the influence of the image itself. However, too many network layers can lead to the increase of error.

Deep Residual Learning for Image Recognition by Kaiming He [8] has made tremendous improvements in many computer vision applications besides image classification tasks, such as object detection and recognition. The main contribution of his research is introducing the residual block, which can successfully avoid the problem caused by adding network layers. In this thesis, we propose a method using residual blocks and my experiment shows significant improvement on both accuracy and F1 score.

This thesis is organized as follows: Chapter 3 describes the problem we need to solve in this paper, the details will be discussed in Chapter 4, including data preparation, the architecture of the network and training details. Chapter 5 gives the experience process and results, as well as a comparison with other existing methods. Conclusions and perspectives are discussed in Chapter 6.

Chapter 2

Literature Review

Over past few years, pavement crack detection has become a popular research field. Many papers have worked on this and put forward usable solutions. In this part, we review their works and highlight their contributions on this research field.

2.1 Intensity Threshold Methods

Hu and Zhao [9] improve the local binary pattern method introduced by Pietikinen and Harwood [15], for pavement crack detection. Simple calculation, gray scale, and rotation invariant are three key characteristics of local binary pattern (LBP). It is proved theoretically that LBP is an efficient texture classification method. LBP operator is defined as:

$$LBP_{P,R} = \sum_{p=1}^{P-1} s(g_p - gc * 2^p), s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0, \end{cases} \tag{2.1}$$

where gray value g_0, $g_p, (p = 0, ..., P-1)$ corresponds to the center pixel and the ith neighbor pixels. Based on the above definition, the LBP operator is able to describe neighboring pixels by comparing them with the central pixel. If the value of the adjacent pixels are larger or equal to the value of the center pixel, set the value of this pixel to one, otherwise set it to zero. First, all local patterns were regrouped into five sub-classes to gather meaningful local structures. Second, roughly estimate the range of variance and select the appropriate threshold value accordingly. Then, in order to implement this algorithm quickly, put local pattern and new sub-classes into a table for faster reference.

Tang and Gu [19] propose an advanced image processing technology used to detect cracks in road damage analysis. Based on their analysis, they discover the defect part in the image usually has a small gray scale value, so when the pixel value is smaller than the threshold, it is considered as a cracked pixel and marked as one; if not, it is considered as a non-cracked pixel and marked as zero. The selection of threshold value determines the effect of detection; a histogram-based method is used to get the threshold. First, a histogram of the original image is generated, and then use the Gaussian filter for smoothing. After that, a final histogram can be used to find a good threshold value.

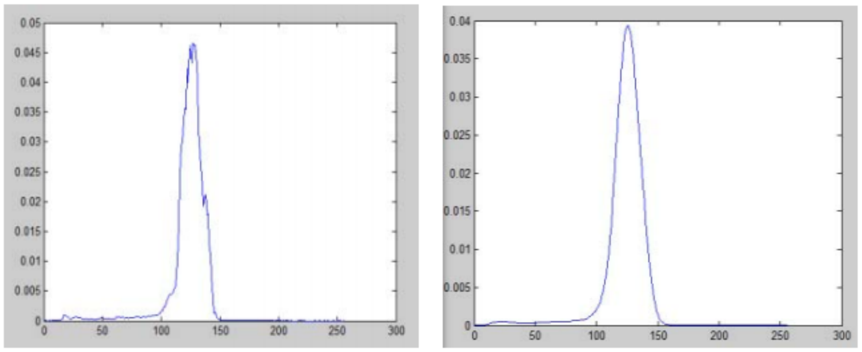


Figure 2.1: Original histogram and Gaussian smoothing histogram.

These works implement intensity histogram and dynamic threshold to segment the cracks from the backgrounds. These methods are able to obtain acceptable results on some pavement images; however, it has a high requirement for image quality. For example, the cracks in the image are very clear and they are obviously distinguished from the background. These methods are susceptible to disruption by noise, so the detection result may not be that good, especially when the image is taken under poor lighting conditions and the background has a complex texture. Moreover, these methods sometimes cannot successfully detect complete cracks and selecting a appropriate threshold value is challenging.

2.2 Filter Based Methods

Filter based methods are also widely adopted. Based on the high similarity of crack and edge in morphology, researchers use edge detection methods such as Sobel and Canny on pavement crack detection. Canny Edge Detection is a well-known edge detection algorithm. It was developed by John F. Canny [2]. It is mainly divided into four steps. In order to reduce the influence of noise, first, use a 5×5 Gaussian filter to get a smoothed image. Next, Sobel kernels are used to filter the image horizontally and vertically, then the derivatives of the horizontal direction (G_X) and the vertical direction (G_Y) are obtained. We can get the edge gradient and its direction of each pixel using the formula below:

$$EdgeGradient(G) = \sqrt{G_x^2 + G_y^2}, \tag{2.2}$$

$$Angel(\theta) = tan^{-1}(\frac{G_y}{G_x}). \tag{2.3}$$

Then at every pixel, check if it is at local maximum by calculating its gradient. Finally, for the edges detected, determine which ones are correct edges and which ones are wrong. So there are two thresholds needed: a minimum threshold and a maximum threshold. Any edges with the gradient larger than the minimum threshold should be the correct edge, while smaller than the maximum threshold should be the wrong edge. The gradient located between these two thresholds use their connectivity to classify edges or non-edges. If they are connected to an edge pixel, they are treated as part of the edge. In this way, we can finally get the edge of the image roughly.

Santhi *et al.* [17] use Canny operator with Butter-worth filter and achieve an acceptable result. First convert the three dimensions RGB image into gray scale image. Now a filter is needed to filter out the noise in the image. Butter-worth filter dynamically chooses the threshold of the filter according to the image. The threshold varies with the intensity of the image. The main factor determining the threshold is the intensity change between the foreground and background of images. Once the image is filtered, a pure sample of cracks can be obtained. Canny edge detection algorithm can be used for these samples.

Ouyang and Wang [14] propose a beamlet-based method for pavement crack detection. Beamlets are line segments with different locations, orientations and scales, and the beamlet transform is a set of line integrals along all beamlets collected. $f(x_1, x_2)$ is a continuous function on two-dimensional space, where x_1 and x_2 are two coordinates. The beamlet transform T_f of function f is defined as:

$$T_f(b) = \int_b f(x(l))\, dl, b \in B_E], \tag{2.4}$$

where B_E is the collection of all beamlets.

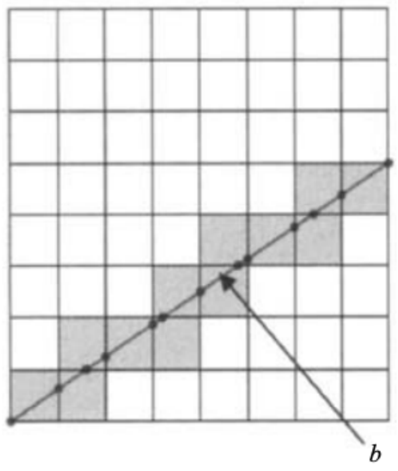


Figure 2.2: Beamlet transform.

In the discrete domain, the beamlet transform for all the points along the beamlets b is defined as,

$$f(x_1, x_2) = \sum_{i_1, i_2} f_{i_1,i_2} \phi_{i_1,i_2}(x_1, x_2), \tag{2.5}$$

where f_{i_1,i_2} means the pixel value of i_1, i_2,and ϕ_{i_1,i_2} means the weight function of each pixel. If $p(x_1, x_2)$ represents $[i_1/n, (i_1 + 1)/n] * [i_2/n, (i_2 + 1)/n]$, choose function ϕ_{i_1,i_2}

to fulfill the equation:

$$n^2 \int_{p(x_1,x_2)} \phi_{i_1,i_2}(x_1,x_2)\, dx_1 dx_2 = \delta_{i_1,i_2}, \tag{2.6}$$

δ_{i_1,i_2} is Kroneker symbol. Then f_{i_1,i_2} is a function that obeys:

$$f_{i_1,i_2} = Ave[p(x_1,x_2)]. \tag{2.7}$$

Canny operator uses the two-dimensional Gaussian function as the filter kernel. The discontinuity of the edge is mainly due to the threshold of Canny edge detector with high local gradient and amplitude caused by image noise. Comparing the results of Canny operator and beamlet transform, beamlet transform method achieves a higher signal-noise ratio and provides a more complete edge feature than Canny operator. The main problem of the filter based methods is that when the crack has a complex texture background, they cannot recognize the complete crack.

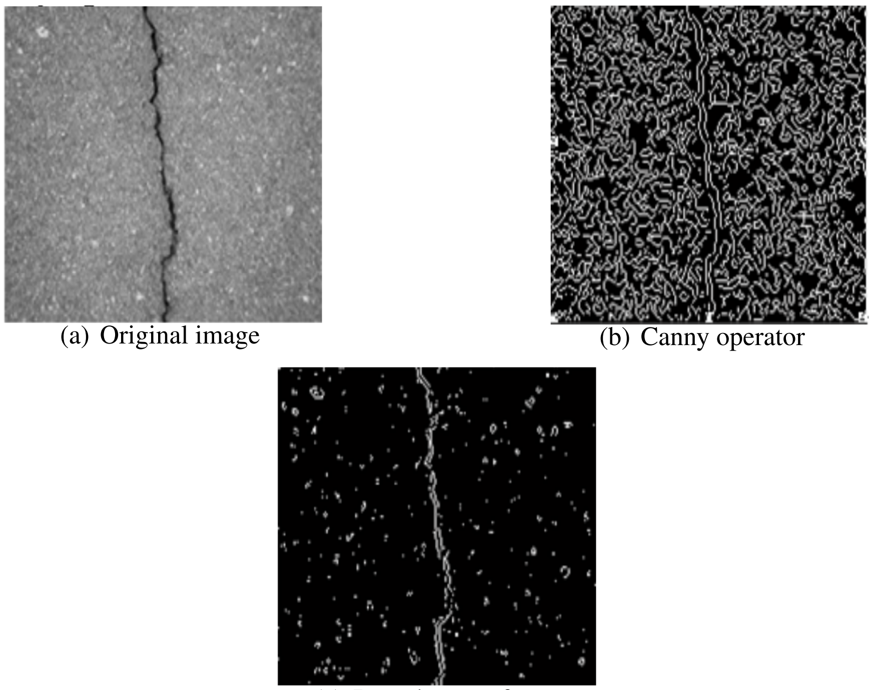

(a) Original image (b) Canny operator (c) Beamlet transform

Figure 2.3: Edge detection of pavement crack image.

The rapid development of signal processing techniques could be used for pavement crack detection. The Gabor filter, named by Dennis Gabor, is a linear filter for texture analysis. It can be used to analyze whether there is a specific frequency content in a particular point and in the surrounding area. This method is very suitable for detecting texture features of pavement surfaces. The Gabor filter consists of a real part and an imaginary part, these two parts can form a complex number or be used separately.

Complex number:

$$g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = exp(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2})exp(i(2\pi\frac{x'}{\lambda}+\psi)). \tag{2.8}$$

Real part:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \cos(2\pi \frac{x'}{\lambda} + \psi). \tag{2.9}$$

Imaginary part:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \sin(2\pi \frac{x'}{\lambda} + \psi). \tag{2.10}$$

Where

$$x' = x \cos\theta + y \sin\theta, \tag{2.11}$$

and

$$y' = -x \sin\theta + y \cos\theta. \tag{2.12}$$

In the above formulas, λ is the wavelength of sinusoidal factor, θ is the normal direction of Gabor function of parallel stripes, ψ is the phase offset, σ is the standard deviation, and γ is the spatial aspect ratio.

Zalama *et al.* [20] propose a method using Gabor filters to detect and classify both longitude and transverse crack in the pavement image. In Figure 2.4, the size of the image is 384×128 pixels, while in Figure 2.5, the size of the image is 128×384 pixels, which contains both defective and non-defective images. These images are segmented into three 128×128 pixel images. It is difficult to find these cracks with human eyes since the dark areas of cracks usually present similar to those of normal areas. Although these images have been pre-processed, it is hard to detect and classify the crack from pavement images. However, these dark areas actually changes the texture of pavements, so we can detect them on human eyes. Gabor filter is used to realize the texture information.

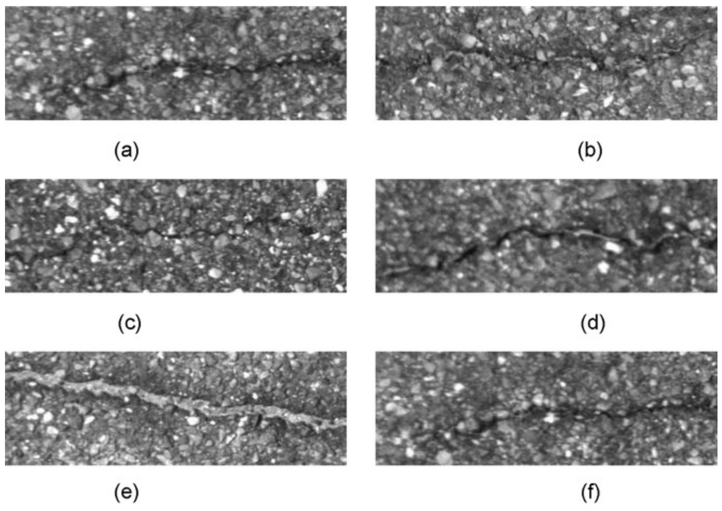


Figure 2.4: Pre-processed images contain transverse cracks.

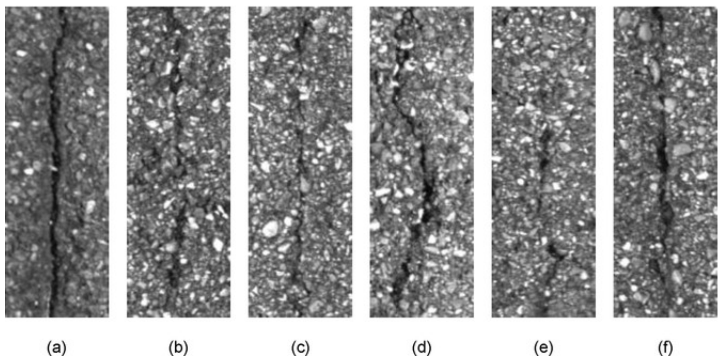


Figure 2.5: Pre-processed images contain longitudinal cracks.

The classifier based on Gabor filter for image feature extraction has achieved outstanding performance.

Salman *et al.* [16] report an algorithm can detect several different types of cracks, such as longitude cracks, transverse cracks, box cracks or crocodile cracks, even though their directions are different. The following figures show an image that includes various types of cracks. In Figure 2.7, the image on the right is the original image containing many red bounding boxes, which represent the detected crack segments.

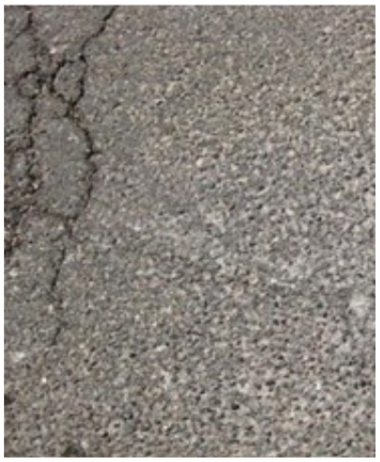


Figure 2.6: Mixed cracks.

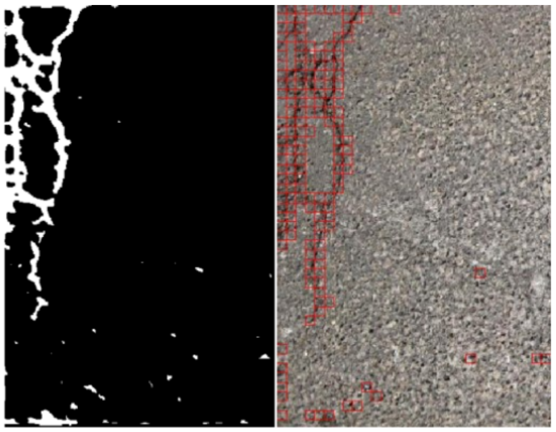
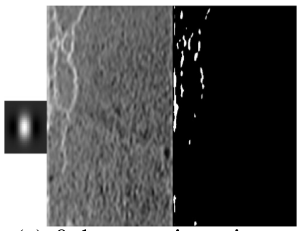
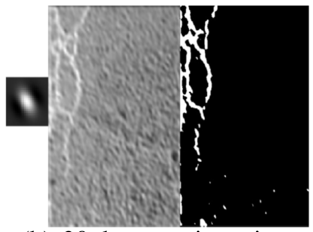
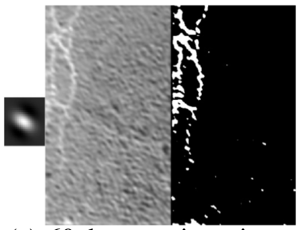
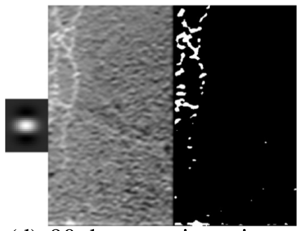


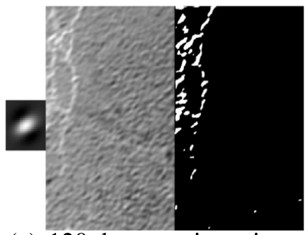
Figure 2.7: Gabor filter overall response.


(a) 0 degree orientation


(b) 30 degree orientation


(c) 60 degree orientation


(d) 90 degree orientation


(e) 120 degree orientation

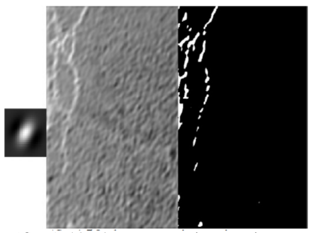

(f) 150 degree orientation

Figure 2.8: First convolution at various orientations.

As can be seen from the above figures, the filter responds to cracks in its corresponding direction. In Figure 2.8(a) the filter has responses in only the vertical direction. In Figure 2.8(b) and Figure 2.8(c) , the filter only has responses on the left side of the crack, while the responses of other sides are not obvious. In Figure 2.8(d) the filter only has responses on part of horizontal direction.

Instead of using a single filter, Chaudhuri *et al.* [4] explore a linear patterns detection by using matched filters. Zhang *et al.* [21] adopt the matched filtering algorithm for detecting pavement cracks. Unlike the traditional edge detection method, the matched filtering algorithm detects cracks by matching several pre-designed filters with the crack characteristics according to shape, orientation or intensity.

The algorithm approximates the intensity profile as a Gaussian curve.

$$f(x,y) = A[1 - kexp(-\frac{-d^2}{2\sigma^2})], \tag{2.13}$$

where $f(x,y)$ is the gray-scale intensity at coordinates (x,y); A is the gray-scale intensity of the background; k is the reflection of object, such as pavement cracks; d is the distance between (x,y) and line segments passing through the center of cracks; and σ is the spread of intensity.

A good filter must have the same shape with the intensity distribution. Use the following equation to determine it:

$$h_{opt}(d) = exp(-\frac{-d^2}{2\sigma^2}). \tag{2.14}$$

For pavement crack images, parameters in the first equation, such as A, k and σ, represent the local maximum background intensity, relative reflections and width of cracks respectively.

Compared with Sobel and Canny edge detectors, matched filter algorithms present a better performance.

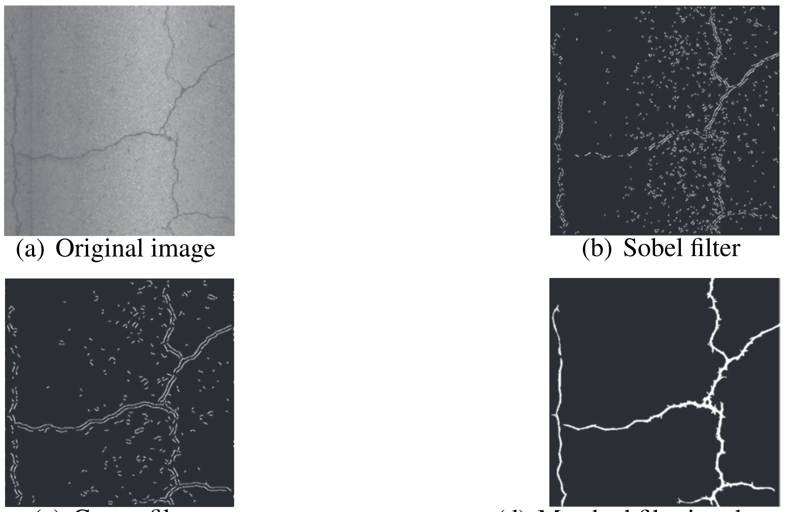

(a) Original image (b) Sobel filter (c) Canny filter (d) Matched filtering detection

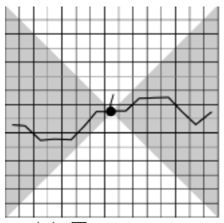
Figure 2.9: Experiment results.

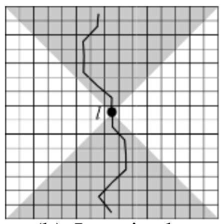
However these filters are impossible to distinguish very fine cracks, so these methods can not correctly detect pixels at the edge of the crack.

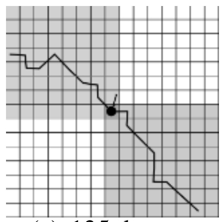
2.3 Minimal Path Selection Methods

Minimal path selection methods have attracted so many attention in past few years. In pavement crack detection field, this method is widely used in continuous crack detection. It judges the crack by searching for continuous pixels. Several methods based on minimal path selection are proposed.

Nguyen *et al.* [12] propose a method for pavement crack detection, called Free-Form Anisotropy. Graph theory, such as Dijkstra algorithm, provide a solution to find the minimal path efficiently.


(a) Transverse


(b) Longitude


(c) 135 degree

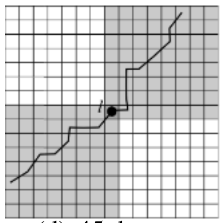

(d) 45 degree

Figure 2.10: Minimal paths of pixel l with four different angles.

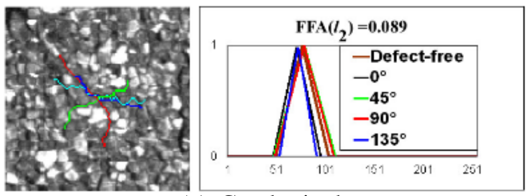
A minimal path means that the path has the smallest sum of pixel intensities. FFA of each pixel l can be calculated by:

$$FFA(l) = \frac{max_j[h(\pi_j, \pi_{bgd})] - min_j[h(\pi_j, \pi_{bgd})]}{max_j[h(\pi_j, \pi_{bgd})]}, \tag{2.15}$$

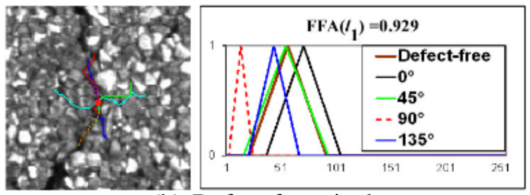
where

$$h(\pi_1, \pi_2) = sup[min(\pi_1, \pi_2)], \tag{2.16}$$

and j is the global orientation, $h(\pi_j, \pi_{bgd})$ computed the four minimal paths in Figure2.10 with pixel l.



(a) Crack pixel



(b) Defect-free pixel

Figure 2.11: Computation of FFA for 2 pixels width d = 30.

Figure 2.11 shows that FFA is able to identify cracks in various directions.

Amhaz *et al.* [1] develop an enhanced minimum path search algorithm which chooses endpoints on local scale and minimum paths on global scale. There are no restrictions on the direction or length of the path. First, select endpoints among the local minimal as the pixel which has a pixel value lower than the threshold.

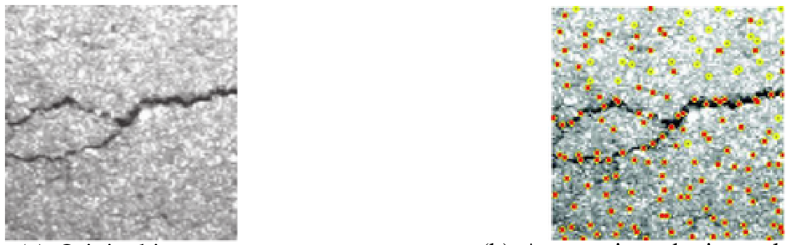

(a) Original image (b) Automatic endpoints selection

Figure 2.12: Endpoints selection example.

Next, use Dijkstra algorithm to calculate the minimum path between the selected endpoints. Only a small part of all the calculated paths belong to cracks. To select these paths, threshold method is a simple and fast solution. The whole process is shown in Figure 2.13.

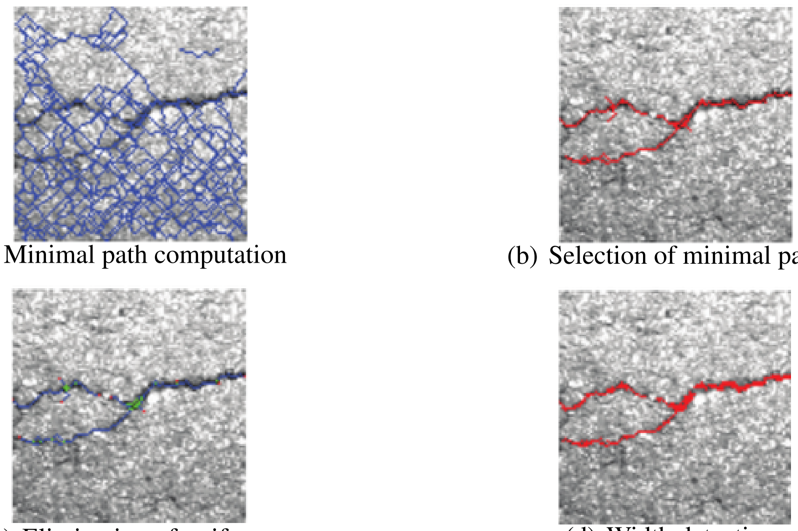

(a) Minimal path computation (b) Selection of minimal paths (c) Elimination of artifacts (d) Width detection

Figure 2.13: Steps of the MPS method.

FFA method has good performance on detecting continuous cracks; however, this method is greatly limited by direction and length.

MPS performs better than other methods in this case. In addition, in order to restore the continuity of pavement crack detection, the minimum spanning tree algorithm gradually shows its advantages.

Zou *et al.* [22] design a CrackTree framework, which solves the problem of shadow and discontinuity in pavement crack images. A geodesic shadow removal algorithm is adopted to eliminate the noise in the image and enhance the cracks at the same time. Next, a probability map is constructed using tensor voting technique. Finally, the minimum spanning tree (MSTS) algorithm is applied to detect continuous cracks and remove useless edges to obtain the whole crack shape. Comparison results indicate that the CrackTree method has better performance in existing minimal path algorithms. This work showed how to highlight and clear images with cracks. The result is a noiseless image, and emphasis was placed on the defective parts.

A MFCD algorithm is proposed by Li *et al.* [10], and this method also works on minimal path algorithm. This method proposes a more effective statistical parameter setting method, uses crack seed edge detection method mentioned above, and designs a window path generation algorithm. Overall, the calculation time is reduced and the efficiency of crack detection is improved.

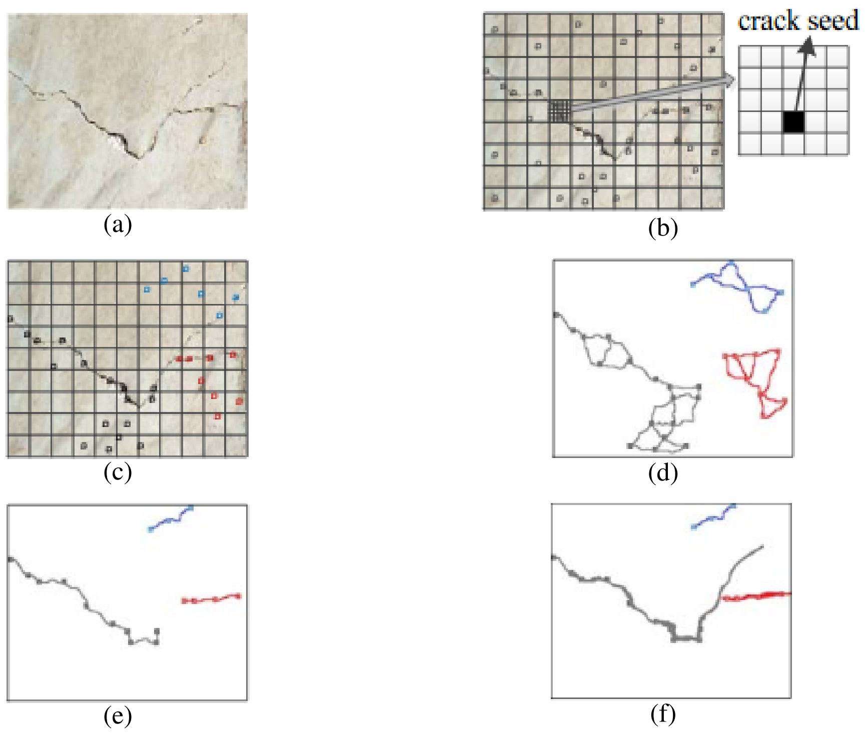


Figure 2.14: Steps of the MFCD method.

Figure 2.14(a) is an example of an original image. The MFCD method is described in Figure 2.14(b)-(f). First, the authors use Gaussian blurring to extract the crack seeds, then they cluster and filter those seeds, as shown in Figure 2.14 (b)(c). Then they find cracks by using a windowed minimal intensity path method, as shown in Figure 2.14(d). Finally, the authors detect the crack and propose a statistical model to calculate the crack fusion value at different scales, as shown in Figure 2.14(f). When the crack on the pavement surface is very blurred, or there are disturbing components such as oil spots in the pavement image, the performance will be reduced. Compared with the results of Canny and FFA, MFCD algorithm outperforms all counterparts.

In order to use minimum path or a minimum spanning tree algorithm on pavement crack detection, it is necessary to ensure that cracks are clear and continuous. However, it is uncertain in many practical cases.

2.4 Machine Learning Methods

Machine learning methods have become popular with the development of data and GPU. Methods focusing on feature extraction and pattern recognition have been widely used for pavement crack detection. In machine learning methods, images are are often divided into several sub-images for training. Furthermore, supervised training requires accurate labels.

Li *et al.* [11] use BP neural networks(BPNNs) on pavement crack detection. The design of BPNN structure is a complex and difficult process, a certain number of hidden layers and nodes need to be set. Hidden layers can be influenced by many factors, such as the choice of activation function, the structure of network, training algorithm, and training dataset. The structure of the BPNN is shown in Figure 2.15.

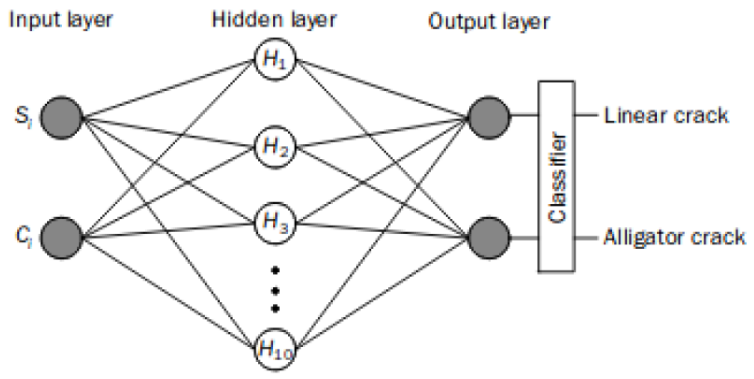


Figure 2.15: BPNN structure.

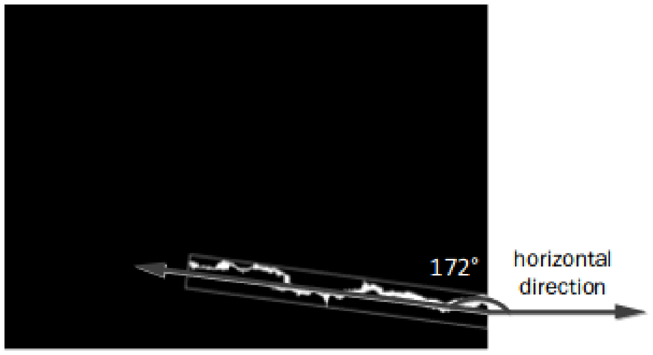


Figure 2.16: The direction angle.

In this paper, they also divide cracks into categories, and use a direction angle to determine whether the crack is transverse crack or longitude crack or alligator crack. The result shows that BP neural network can classify the linear crack image and the alligator crack image effectively. This paper also solves the crack classification and high computing cost problem, since the BP network provided in this paper only has ten nodes in the hidden layer, the input layer as well as the output layer both have two nodes.

Cord and Chambon [5] use a supervised learning algorithm Adaboost on pavement crack detection problem. AdaBoost, also known as Adaptive Boosting, is a very popular machine learning algorithm, which was first proposed by Freund and Schapire in 1997. This algorithm is used to generate arbitrary classifiers by merging lots of weak classifiers. It can be used with other machine learning algorithms together to improve the overall performance.

The training dataset is a subset of sub-images $x_i, 1 \leq i \leq m$, where m is the number of sub-images. $y_i \in 0, 1$ corresponding to the classification of x_i. Select a weak classifier with smallest error ϵ_t on D_t.

$$h_t : X \to 0, 1. \tag{2.17}$$

Initialize the weight:

$$D_t(i) = \frac{1}{m}, 1 \leq i \leq m. \tag{2.18}$$

Given D_t and h_t:

$$D_{t+1} = \frac{D_t(i)}{Z_t} c(x), \tag{2.19}$$

$$c(x) = \begin{cases} e^{-\alpha_t} & y_i = h_t(x_i) \\ e^{\alpha_t} & y_i \neq h_t(x_i), \end{cases} \tag{2.20}$$

$$D_{t+1} = \frac{D_t(i)}{Z_t} e_{-\alpha_t y_i h_t(x_i)}, \tag{2.21}$$

where $Z_t = normalization$ constant

$$\alpha_t = \frac{1}{2} ln \frac{1 - \epsilon_t}{\epsilon_t} > 0. \tag{2.22}$$

Therefore, after $t + 1$ iterators, The algorithm can select a classifier and extract the missing information better.

$$H_{final}(x) = sign(\sum \alpha_t h_t(x)). \tag{2.23}$$

AdaBoost is applied to extract the relevant texture feature that can describe the crack on pavement images.

In CrackIT by Oliveira and Correia [13], unsupervised learning is performed using mean and standard deviation to distinguish cracked and non-cracked blocks. This paper presents an image detection and characterization processing algorithm for pavement crack detection toolbox, in MATLAB environment. This toolbox includes numbers of useful tools for performance evaluation; provides solutions for two different crack detection algorithms, block-based crack detection and pixel-based crack detection. Different classification strategies can be used here. Then by using the training method of sample paradigm and the results of initial selection of blocks, the image dataset is divided into training and test set.

The proposed toolbox can achieve good crack detection and classification results, but as we discussed before, it is still not able to detect very fine cracks, especially because it is difficult to distinguish cracking from raveling distresses.

Shi *et al.* [18] propose a new CrackForest method, which based on random structured forests to characterize cracks. Texture features, standard deviations, and mean parameters contained important information to distinguish crack blocks from non-crack blocks. For a set of images, a corresponding set of binary images represent the manually marked crack edges in the sketch. The image patch of the marked crack at the central pixel will be used as the defect example. As shown in Figure 2.17.

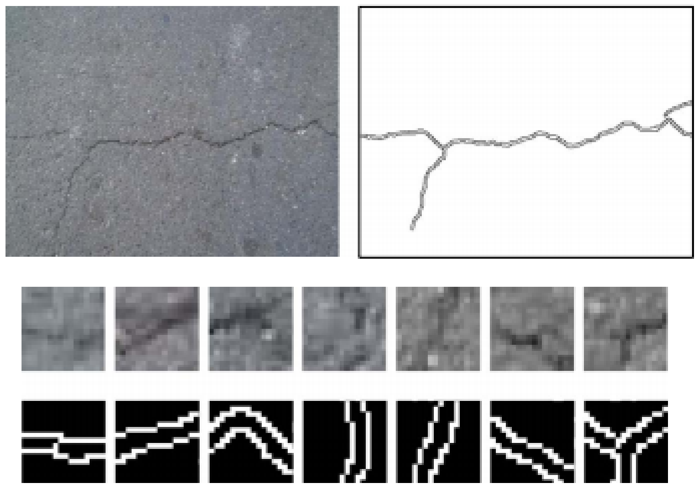


Figure 2.17: Extracted image patches and their hand-drawn contours.

Then for structured learning step, a state-of-art framework, random structured forests, is used to train a crack detect model. Decision tree $f_t(x)$ classifies samples $x < X$ by following the branches of the tree until they reach the leaf nodes. At the same time, node j in the tree is associated with a binary splitting function:

$$h(x, \theta_j) \in (0, 1), \tag{2.24}$$

with parameter θ_j. If $h(x, \theta_j)$ equals to 0, node j send x to left sub-tree, otherwise, node j send x to right sub-tree. As shown in Figure 2.18, each tree is trained independently. Given a node j and a training set $S_j \subset X \times Y$, we would like to find θ_j of $h(x, \theta_j)$ that maximizes information gain, or minimizes entropy.

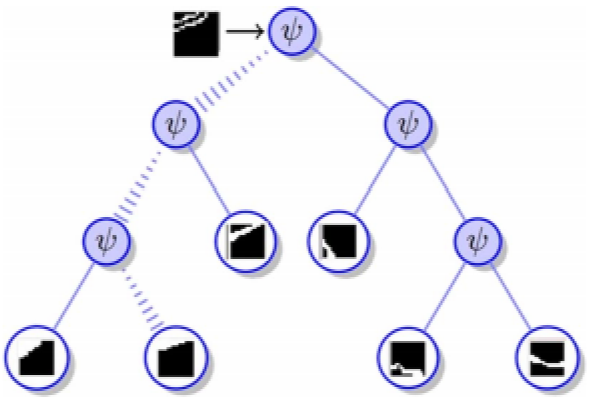


Figure 2.18: Path of an image patch.

Figure 2.19 shows an intuitive example.

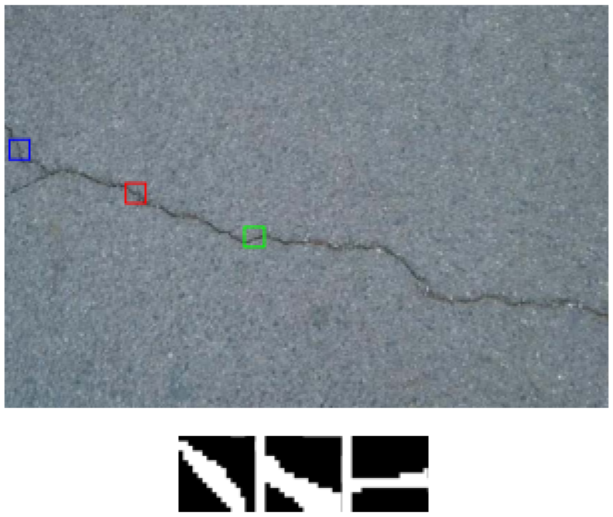


Figure 2.19: Assigning label to each image patch.

CrackForest has better performance than Canny, CrackIT, CrackTree, and FFA. However, those results are much more easily influenced by the extracted features. Also in complicated pavement conditions, finding all effective features is very difficult.

2.5 Deep Learning Methods

Deep learning methods have attracted significant attention due to their outstanding performance, better than traditional machine learning methods. Nowadays, deep learning methods are successfully applied on crack detection.

Cha *et al.* [3] use a sliding window to segment the image into blocks, and CNN is used to predict if a block contains cracks. The deep CNN contains four convolution layers, two pooling layers, a ReLU layer, and a softmax layer. To avoid the mis-classification when the cracks are on the edges of image, use a sliding window technique to scan the image space twice. This network is not influenced by the image conditions.

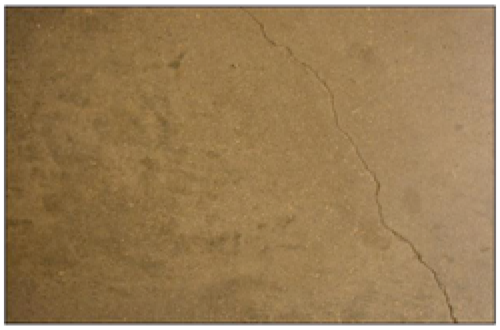


Figure 2.20: Original image.

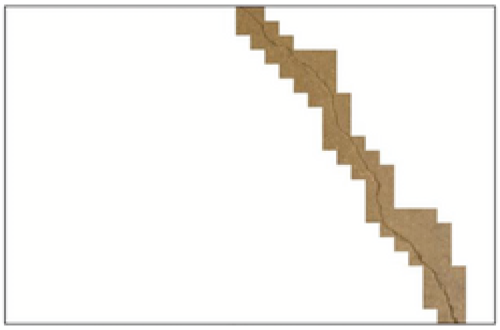


Figure 2.21: Detection result.

Although this method has shown acceptable results in the detection of pavement defects, the problem is it can only detect the patch level crack and doesn't solve the pixel level detection issue.

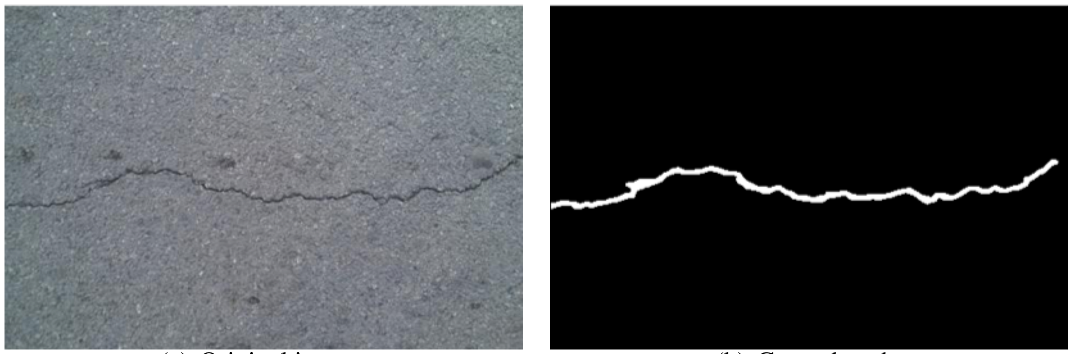
Fan *et al.* [7] propose a structured prediction based pavement crack detection method with CNN. The network can learn the crack of small patch and then find the whole crack in pixel-level. The network contains four convolution layers, two pooling layers, and three fully connection layers. After training, the model can predict the crack structure of each patch from the input image, then all results are combined to gather a probability output. Finally, a decision threshold is set to obtain the final binary result.

This method, compared with other traditional defect detection methods, has shown superior performance in all aspects and does not need pre-processing. However, it has so many parameters that need to be set manually, and these parameters will have a great effect on the overall performance.

Chapter 3

Problem Statement

Pixel-level crack detection results are represented by binary images, black pixels are positive while white pixels are negative. As shown in following figures.



(a) Original image (b) Ground truth

Figure 3.1: Crack detection.

Figure 4.7 shows the result pixels corresponding to the original pixels. Due to the difference of color, intensity, and other features between positive and negative pixels, a deep convolutional neural network can learn these features by giving the label. After training, the model is able to distinguish non-crack and crack pixels from an original image automatically.

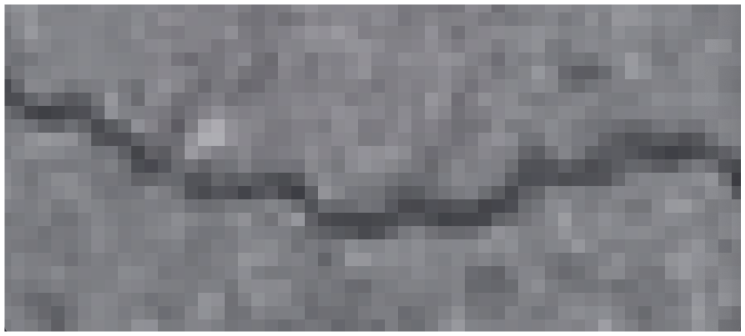


Figure 3.2: Original pixels.

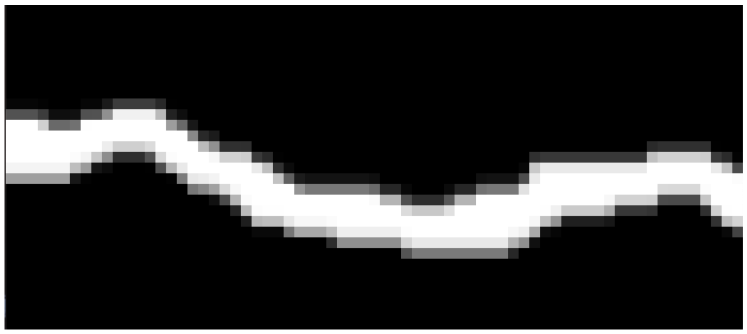


Figure 3.3: Labeled pixels.

My goal is to detect and mark every pixel of cracks from input pavement image by using a deep convolutional neural network. To determine the performance of the method, the testing image will be processed to binary image and compare each pixel with the ground truth.

Chapter 4

Proposed Methods

In this chapter, we will introduce a method for pixel-level pavement crack detection using deep CNN and residual blocks. The entire process is divided into three operations. Data preparation, which includes data relabeling as well as image processing; Network description, which introduces the design of the whole network and parameter tuning; Model training and testing, which provides a crack detection model to identify the cracks on a open source image dataset. As mentioned in the literature review, CNN can extract features based on the provided label, and the results are superior in pixel level.

4.1 Methodology

In order to achieve accurate, effective pavement crack detection, in this thesis, we model the pavement crack detection problem to image feature extraction problem. Feature extraction refers to extracting more advanced features from the original pixels, which can capture the differences between different categories. We label the pixels in the original image into non-crack and crack pixels, and we use the deep convolutional neural network to train a classification model using the features of an image and its corresponding labels.

We design the network and use it to train a model, when the depth of the network increases, the accuracy of the network should increase at the same time. However, one of the problems of network depth increase is that these additional layers need parameter updating, using the gradients propagate backward and forward. So the gradients of the advanced layers will be very small when the network depth is increased. This means that the learning of these layers is basically stagnant, which causes gradient disappearance. Additionally, when the network is deeper, the parameter space is larger and the optimization problem becomes more difficult. When Kaiming He and his team [8] propose a Residual Network, he put forward a residual block algorithm, which makes training a deeper network possible without reducing the accuracy. We adopt this residual block algorithm to train my model in this thesis. Details will be discussed in the following sections.

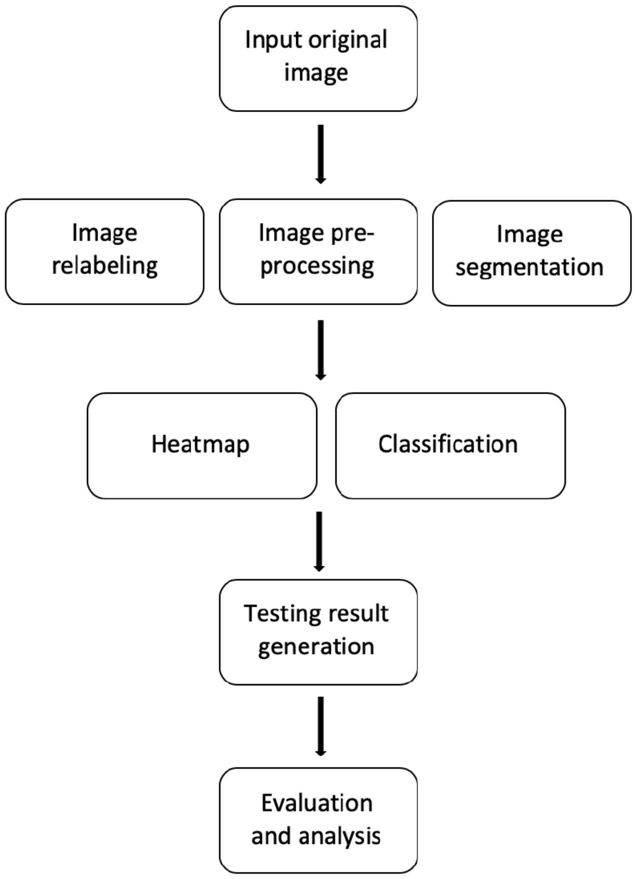


Figure 4.1: Flowchart of the proposed method.

4.2 Construction of Deep Learning Framework MXNet

In this thesis, we choose MXNet deep learning framework to build the training network. MXNet is a fully functional, flexible, extensible framework for deep learning, which supports the most advanced technologies in the deep learning model, including convolutional neural network (CNN). Functions in MXNet framework are very clear and easy to understand, and the framework implements the commonly used optimization algorithm to train the model. Also, MXNet supports CPU and GPU model training methods, so the training speed is faster than other deep learning frameworks. It provides outer interface for Python and C language; integrates a variety of algorithms to facilitate user modification and debugging at the same time. The system architecture of MXNet is shown in the following Figure 4.2. From top to bottom, it indicates the embedding of various main languages, the support of interfaces (matrix operations, symbolic expressions, distributed communication), the unified system implementation of the two programming modes, and the support of each hardware. In generaral, MXNet is very suitable for our research topic, so we choose MXNet as the framework of model training and testing in this thesis.

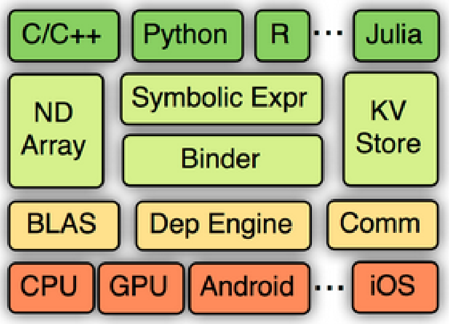


Figure 4.2: System architecture of MXNet.

MXnet supports Linux, Windows, and Mac operating systems. The main operating system used in this thesis is Ubuntu 16.04. Next thing is to install CUDA and CUDNN. They can be downloaded directly from the official website and installed according to the instructions. Note that when installing CUDA, we need to choose GPU hardware support and corresponding NVIDIA driver, so that we can train the model on GPU later. Then, install Mxnet by using pip module. First install the specified version of Python language and the latest version of PIP module, then use pip to install MXNet, select the specified version and the corresponding CUDA version.

4.3 Data Preparation

We use a CFD dataset [18][6] containing 118 RGB images with a resolution of 320×480 pixels. Images in this dataset are taken manually from Beijing, China. Hand labeled ground truth is also provided and is used to measure the results. There are a lot of time the results of training are influenced by labels. We use a unified label while other method are also trained using same label, so we can compare the result with ground truth directly, without considering the error caused by self-labeling. Images in this dataset contain noisy pixels for example, shadows, oil spots, and water stains, very close to the reality of our lives. The label for each image is stored in a Matlab file. As can be seen from the sample figures below, the edge of the crack pixels are marked as one while other pixels are marked as zero in the .mat file. Before training, we relabel the cracks in the image according to the label information, transfer them to binary images, then we can use these relabeled binary images as our ground truth.

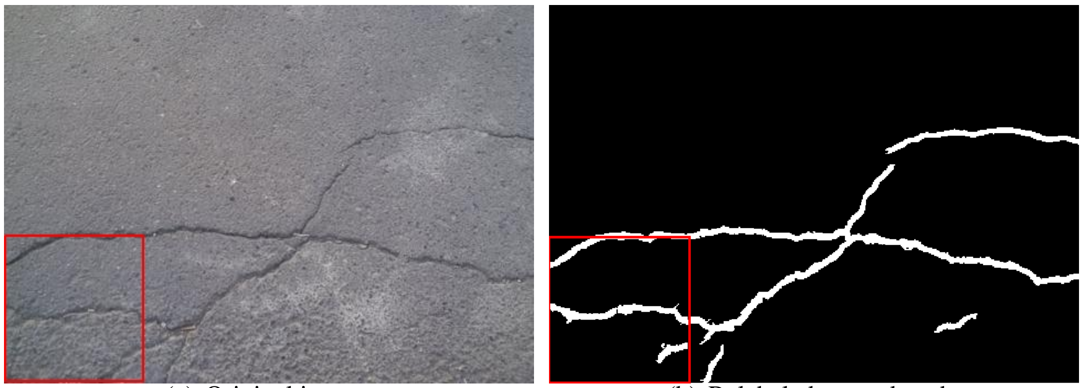
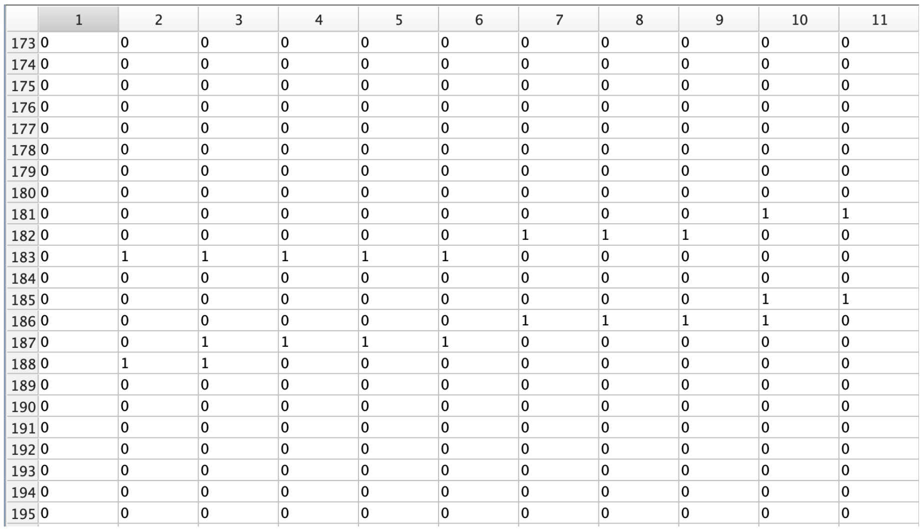

(a) Original image (b) Relabeled ground truth

Figure 4.3: Sample 1.



	1	2	3	4	5	6	7	8	9	10	11
173	0	0	0	0	0	0	0	0	0	0	0
174	0	0	0	0	0	0	0	0	0	0	0
175	0	0	0	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0	0	0	0
177	0	0	0	0	0	0	0	0	0	0	0
178	0	0	0	0	0	0	0	0	0	0	0
179	0	0	0	0	0	0	0	0	0	0	0
180	0	0	0	0	0	0	0	0	0	0	0
181	0	0	0	0	0	0	0	0	0	1	1
182	0	0	0	0	0	0	1	1	1	0	0
183	0	1	1	1	1	1	0	0	0	0	0
184	0	0	0	0	0	0	0	0	0	0	0
185	0	0	0	0	0	0	0	0	0	1	1
186	0	0	0	0	0	0	1	1	1	1	0
187	0	0	1	1	1	1	0	0	0	0	0
188	0	1	1	0	0	0	0	0	0	0	0
189	0	0	0	0	0	0	0	0	0	0	0
190	0	0	0	0	0	0	0	0	0	0	0
191	0	0	0	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	0	0	0	0
193	0	0	0	0	0	0	0	0	0	0	0
194	0	0	0	0	0	0	0	0	0	0	0
195	0	0	0	0	0	0	0	0	0	0	0

Figure 4.4: Location of sample 1 in .mat file.

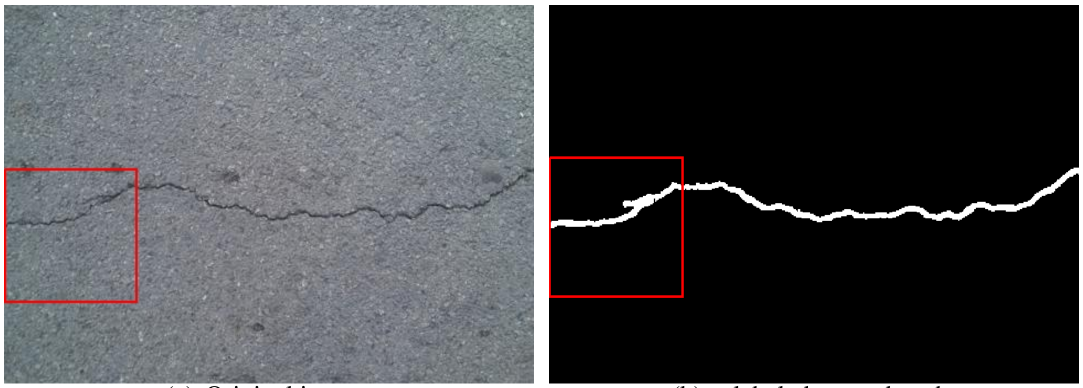
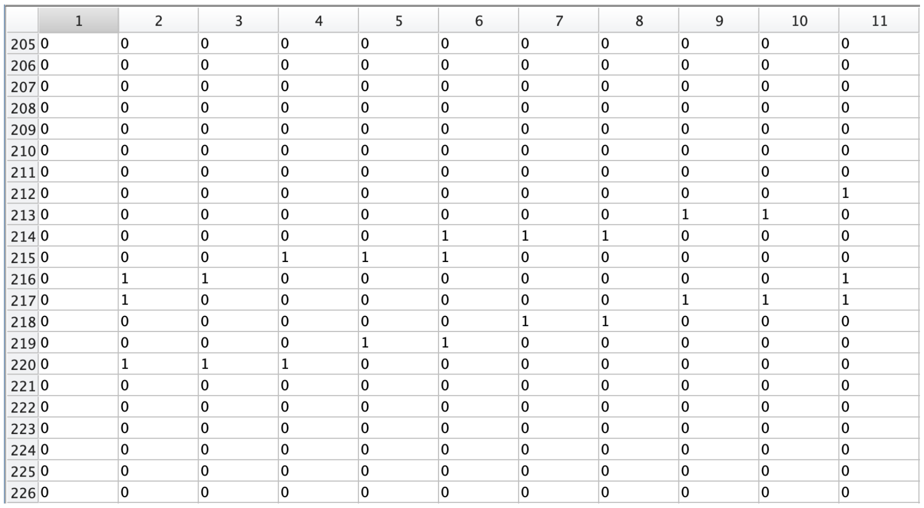
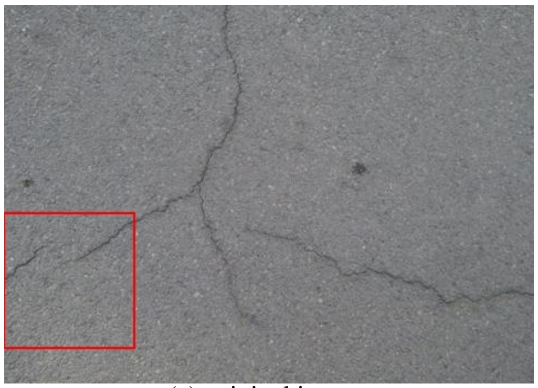

(a) Original image (b) relabeled ground truth

Figure 4.5: Sample 2.



	1	2	3	4	5	6	7	8	9	10	11
205	0	0	0	0	0	0	0	0	0	0	0
206	0	0	0	0	0	0	0	0	0	0	0
207	0	0	0	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0	0	0	0
209	0	0	0	0	0	0	0	0	0	0	0
210	0	0	0	0	0	0	0	0	0	0	0
211	0	0	0	0	0	0	0	0	0	0	0
212	0	0	0	0	0	0	0	0	0	0	1
213	0	0	0	0	0	0	0	0	1	1	0
214	0	0	0	0	0	1	1	1	0	0	0
215	0	0	0	1	1	1	0	0	0	0	0
216	0	1	1	0	0	0	0	0	0	0	1
217	0	1	0	0	0	0	0	0	1	1	1
218	0	0	0	0	0	0	1	1	0	0	0
219	0	0	0	0	1	1	0	0	0	0	0
220	0	1	1	1	0	0	0	0	0	0	0
221	0	0	0	0	0	0	0	0	0	0	0
222	0	0	0	0	0	0	0	0	0	0	0
223	0	0	0	0	0	0	0	0	0	0	0
224	0	0	0	0	0	0	0	0	0	0	0
225	0	0	0	0	0	0	0	0	0	0	0
226	0	0	0	0	0	0	0	0	0	0	0

Figure 4.6: location of sample 2 in .mat file.


(a) original image

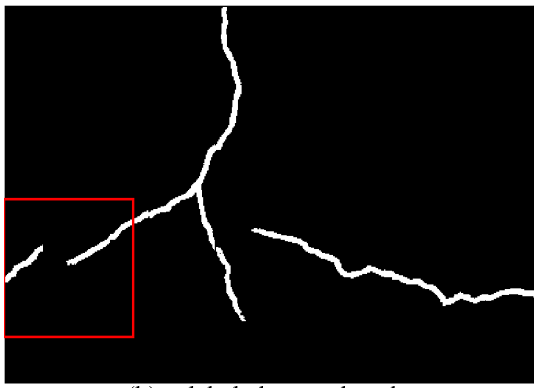
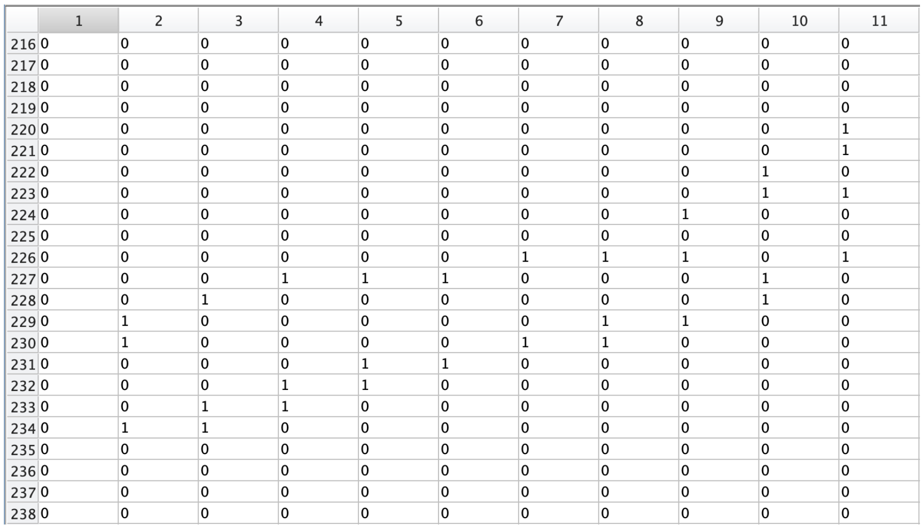

(b) relabeled ground truth

Figure 4.7: Sample 3.



	1	2	3	4	5	6	7	8	9	10	11
216	0	0	0	0	0	0	0	0	0	0	0
217	0	0	0	0	0	0	0	0	0	0	0
218	0	0	0	0	0	0	0	0	0	0	0
219	0	0	0	0	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0	0	0	1
221	0	0	0	0	0	0	0	0	0	0	1
222	0	0	0	0	0	0	0	0	0	1	0
223	0	0	0	0	0	0	0	0	0	1	1
224	0	0	0	0	0	0	0	0	1	0	0
225	0	0	0	0	0	0	0	0	0	0	0
226	0	0	0	0	0	0	1	1	1	0	1
227	0	0	0	1	1	1	0	0	0	1	0
228	0	0	1	0	0	0	0	0	0	1	0
229	0	1	0	0	0	0	0	1	1	0	0
230	0	1	0	0	0	0	1	1	0	0	0
231	0	0	0	0	1	1	0	0	0	0	0
232	0	0	0	1	1	0	0	0	0	0	0
233	0	0	1	1	0	0	0	0	0	0	0
234	0	1	1	0	0	0	0	0	0	0	0
235	0	0	0	0	0	0	0	0	0	0	0
236	0	0	0	0	0	0	0	0	0	0	0
237	0	0	0	0	0	0	0	0	0	0	0
238	0	0	0	0	0	0	0	0	0	0	0

Figure 4.8: Location of sample 3 in .mat file.

In order to train a better learning model, hundreds of images are not enough. In addition, a small dataset can easily lead to over-fitting of the model. A simple and effective way to solve this problem is image segmentation. We use a 256×256 sliding window with overlap 32 pixels in horizontal direction and 64 pixels in vertical direction go over the whole image. In this way, each image can be cut into 16 sub-images.

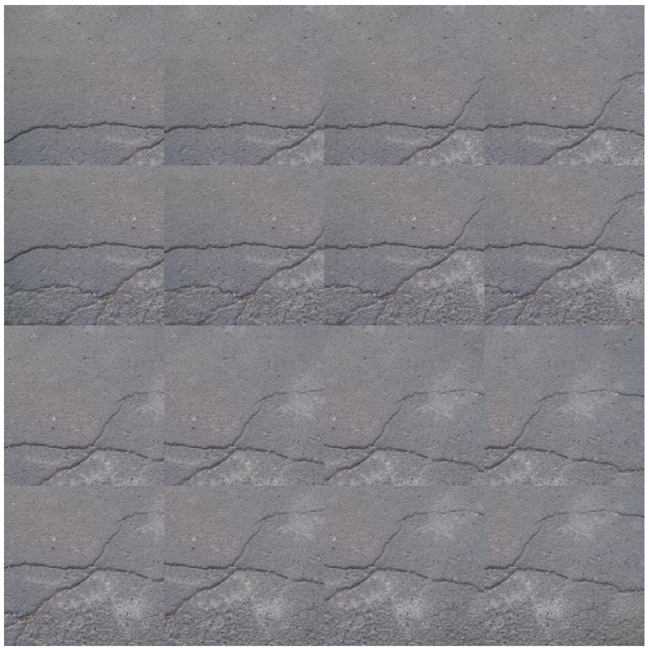


Figure 4.9: Original image segmentation.

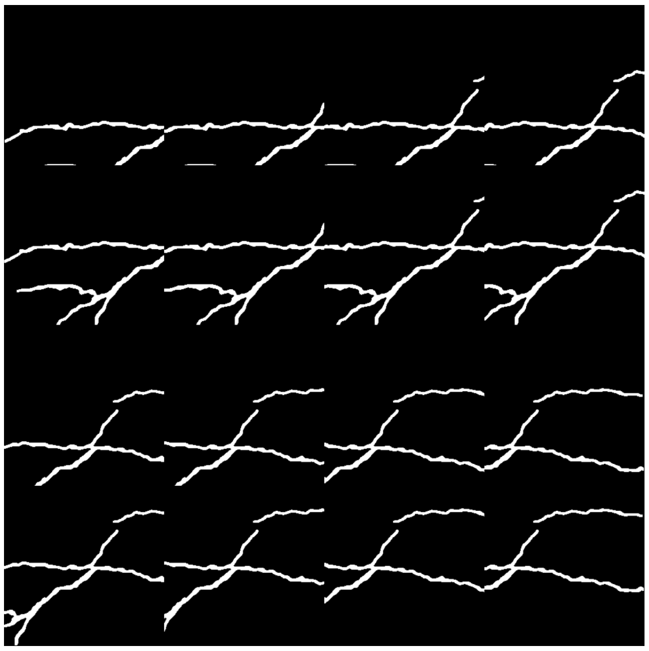


Figure 4.10: Ground truth segmentation.

As mentioned earlier, we use the MXNet framework for network building and model training. When the MXNet framework is used for image-related projects, it reads image files by combining .lst with images. First, we generate a .lst file, which includes the corresponding list of image paths and labels. That is to say, we can control the changes of training set and test set by maintaining this list. The advantages are flexible and space-saving. The disadvantages are if the image format does not meet the requirements, it is easy to make mistakes, and if the image folder corresponding to some image paths in the list is deleted, all of these may report errors. Image data has two categories, so there are two sub-folders for image files belonging to each category. Then we can simply use a im2rec.py tool provided by MXNet to generate the .lst file, and use this .lst file to create .rec file. So far, the data preparation has been completed.

4.4 Model Training

The training process is composed of two processes, those being forward propagation and backward propagation. After every forward and backward propagation, the parameters are updated once, then the process is repeated with new parameters.

A *Forward Propagation*

Forward propagation is the process of calculating parameters of each layer from the input, getting the output, and calculating the loss according to the difference between calculated output and the real value. The process can be expressed by the equation below:

$$a^2 = f(z^2) = f(a^1 * W^2 + b^2), \tag{4.1}$$

where f is the activation function, a and z is the output of each layer, W is the weight, b is bias.

B *Loss Function*

When training networks, the loss function always needs to be determined. There are many choices of loss function, the most commonly used one is the mean-square error function (MSE). The loss function is:

$$Loss = \frac{1}{2}\sum_{k=1}^{n}(z_k - y_k)^2, \tag{4.2}$$

where n is the number of the output, and usually it is the number of classes.

C *Backward Propagation*

Backward propagation is the process of calculating the loss function and using a stochastic gradient descent algorithm to update the parameters. We use δ to indicate the error. To calculate the error,

$$\delta^L = f^{'}(z^L) \bullet (y - a^L), \tag{4.3}$$

$$\delta^l = (W^{l+1})^T \bullet f'(z^l), \tag{4.4}$$

L is the last layer of the network, l is the index of other layers of the network. To update the weight:

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial Loss}{\partial z} = \delta, \tag{4.5}$$

$$\Delta W^l = -\eta\frac{\partial Loss}{\partial W^l}, \tag{4.6}$$

$$\frac{\partial Loss}{\partial W^l} = x^{l-1}(\delta^l)^T, \tag{4.7}$$

where η is the learning rate.

D *Weight Initialization*

When training the network, the weights should be assigned randomly at the beginning. Obviously, randomly picked parameters cannot lead to good results. In the training process, we usually start with a bad model and repeatedly update the weights, yielding in the end a model with high accuracy. In parameter initialization, we usually initialize the weight to a small random number, but cannot initialize to zero. Because the zero weight will cause the parameters of back propagation to not be updated effectively, and this would directly cause the training of the network to fail. There are several methods for weight initialization, such as random generation of small random numbers, standard initialization, and Xavier initialization.

Because of the limitation of data quantity and the increase of network layers and training parameters, CNN is often prone to over-fitting. Earlier we discussed data augmentation, a method for improving the quality of training samples, and overcoming over-fitting. In addition to this method, we can also apply weight decay, like the L1 and L2 regularization methods, or the dropout method, for preventing over-fitting.

4.5 Network Description

4.5.1 Convolutional Neural Network

The simplest structure of a neural network consists of an input layer, hidden layers, and an output layer. Each layer of network has multiple neurons. The upper layer of neurons maps to the next layer of neurons through an activation function. Each neuron has a corresponding weight, and the output is our classification category. The network is composed of a large number of neurons connected to each other. Each neuron receives the input of the linear combination of the front neurons. At first, it is simply linear-weighted. Then, a non-linear activation function is added. After the non-linear transformation, the output is sent to the next neuron. The connection between each two neurons represents a weight. Different weights and activation functions will lead to different outputs of the neural network. When a problem is complex, requires a lot of data and labels, as well as a lot of parameters, a traditional neural network is prone to over-fitting. Moreover, with the exponential increase of parameters, the operational speed will become very slow. For example, we have an image with size (256, 256, 1) that represents an image with one channel (where one represents a gray scale image). So, our network needs $256 \times 256 = 65536$ neurons. If hidden layers use 15 neurons, and there are two outputs, then to calculate the number of parameters we simply need $65536 \times 15 \times 2 + 15 + 2$. The number is too large, as well as the computation.

Convolutional neural network provides a better method for extracting feature parameters. A typical CNN contains convolution layers, pooling layers, and fully connected layers.

A *Convolution layer*

The convolution layer is used to extract the features. Each convolution kernel is a different feature extractor. (Notice that the depth of the input image should be the same as that of the convolution kernel.) Our input image is the RGB image. The depth of the input image is three, and the depth of the convolution kernel should be three respectively. $x_{i,j}$ represents the element on ith row and jth column of the input, $w_{m,n}$ represents the weight on mth row and nth column, w_b is the bias, $a_{i,j}$ represents the element on ith row and jth column of the output, f is the activation function, we use ReLU function here.

The formula for calculating convolution is

$$a_{i,j} = f(\sum_m \sum_n w_{m,n} x_{i+m,j+n} + w_b). \tag{4.8}$$

The calculation process and result of convolution are shown in the following figures. Here we set the bias equal to zero and stride is one.

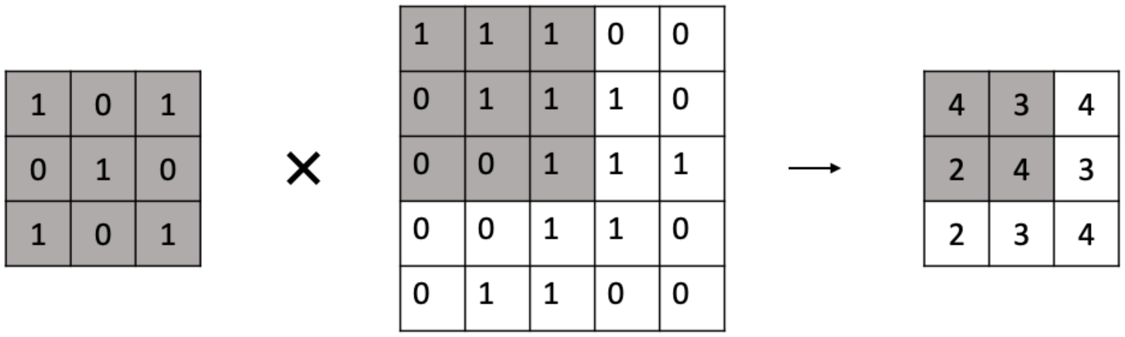


Figure 4.11: Calculation process and result of convolution.

Stride can be set to a number greater than one. For example, when the stride is two, the calculation process and result is as follows:

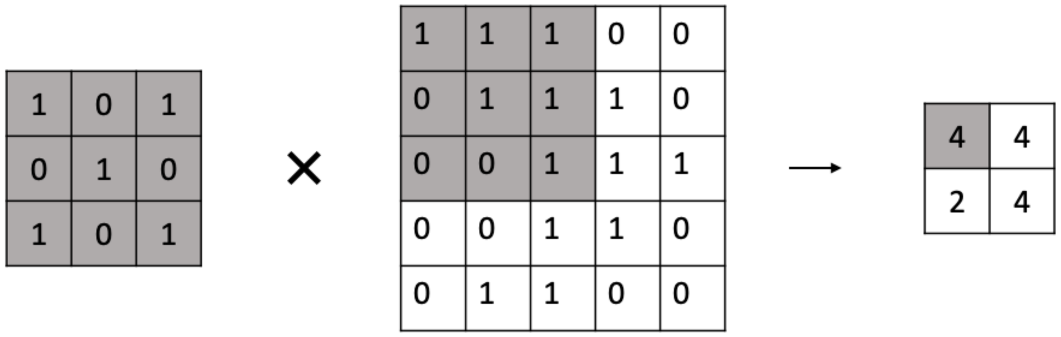


Figure 4.12: Calculation process and result of convolution, stride 2.

When the stride is two, the output size becomes 2×2. With the increase of stride, the output size of convolution decreases. The image size, kernel size, stride, and output size after convolution are related. In fact, they satisfy the following relationships:

$$W_{output} = \frac{(W - F + 2P)}{S} + 1, \tag{4.9}$$

where W is the image size, F is the kernel size, S is stride, and P is zero padding, which we will discuss later.

B *Pooling layer*

The main purpose of the pooling layer is down-sampling, which further reduces the number of parameters by removing the unimportant samples from the output image without affecting the image quality. Generally, there are two methods of pooling:

a Max pooling

Take the maximum value in the sliding window.

b Average pooling

Take the average of all values in the sliding window.

We use a max pooling with a size of 2×2 and stride of one. If we take the maximum value of each window, the size of the output will change from 3×3 to 2×2:

$$W_{output} = (W - F) + S. \tag{4.10}$$

From the above example, there will be the following changes:

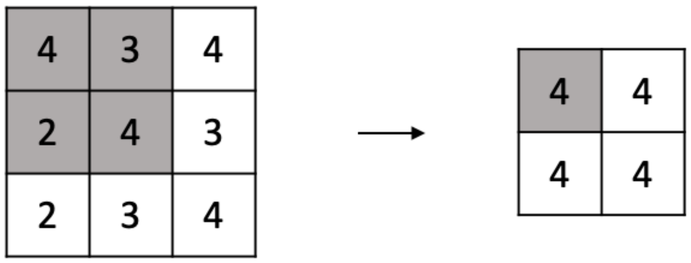


Figure 4.13: Calculation process and result of max pooling.

So far, the image size has reduced from 5×5 to 3×3 through the convolution layer and 2×2 through the pooling layer. When we add more layers, the image size will continually reduce. At this point, we will use a zero-padding method, which can help us to ensure that the size of the image remains as original after each convolution or pooling layer. For example, if we add zero padding to the above example and use a 3×3 convolution kernel, stride one, the size of the output image will be the same as that of the input image, as shown in the following figure:

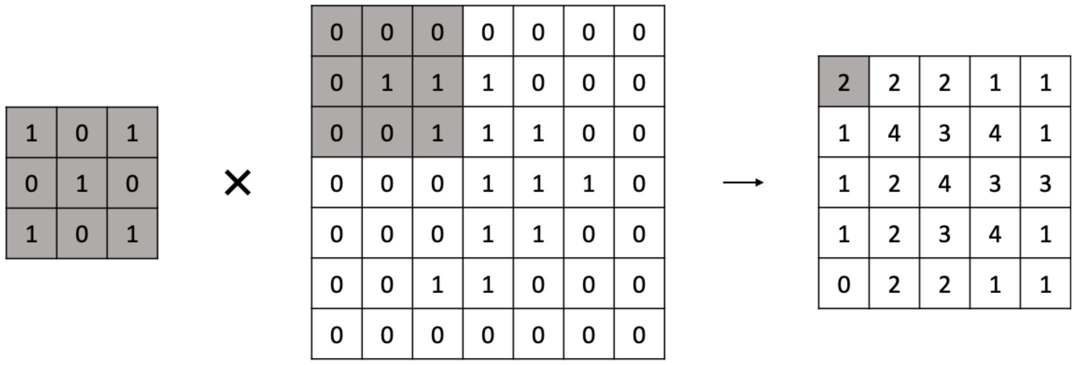


Figure 4.14: Calculation process and result of add zero padding.

We want to keep the image size after the convolution and pooling layer, so we usually choose the convolution kennel of 3×3 and zero padding of one, or the convolution kernel of 5×5 and zero padding of two, so the original size of the image can be retained after calculation.

C *Fully connected layer* The fully connection layer is the last layer of deep neural network, which transfers all the output results' values obtained through the previous layer into a $1 \times n$ array, and then calculates the classification probability by a softmax function. The softmax function is shown below:

$$P_j = \frac{e^{a_j}}{\sum_{k=1}^{T} e^{a_k}}, \tag{4.11}$$

where T is the number of classes.

D *Batch Normalization* Batch Normalization is the normalization of each batch of data. When we train a model, the parameters will be updated in each layer. The reason is the updating of the training parameters in the previous layer will lead to the change of the data distribution in the later layer. For example, the input of the second layer of the network is calculated by the parameters of the first layer and input. The parameters of the first layer keep changing during the whole training process, and this will change of the distribution of input in each later layer of the network. In convolutional neural network, BN can be used to speed up model training and improve the accuracy of the model.

E *Activation function*

Without the activation function, the output of each layer is a linear function of the previous input. That means no matter how many layers the neural network has, the output is a linear combination of inputs. The activation function introduces non-linear factors to neurons, so the neural network can approximate any non-linear function. Additionally, the neural network can be applied to non-linear models.

We usually use ReLU as the activation function of neurons. The ReLU function is actually a maximum functionthe output in neurons is:

$$f(x) = max(0, x). \tag{4.12}$$

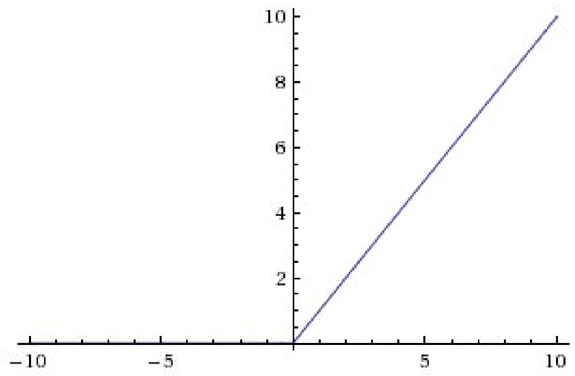


Figure 4.15: ReLU activation function.

Krizhevsky *et al.* find that deep convolution neural networks with ReLU obtain faster training speed than their equivalents with tanh because ReLU only needs a threshold to get the activation value, instead of calculating a lot of complex operations.

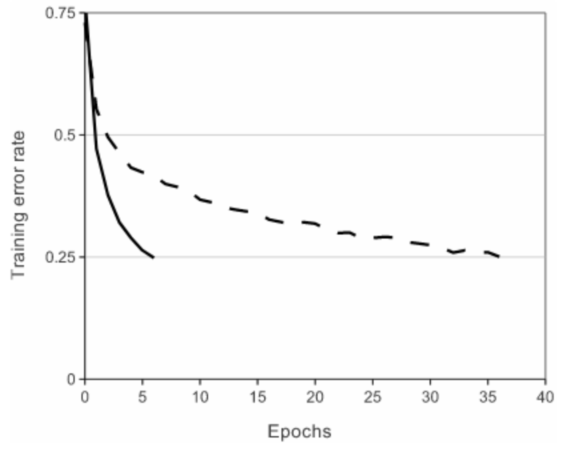


Figure 4.16: ReLU(solid line) is faster than tanh(dashed line).

4.5.2 Residual Block

From the previous section, we have discussed the basic structure of a convolutional neural network and how convolution layers extract features. It is easy to feel that the more layers a network has, the better the performance it will be; however, this is not true. Tests on multiple datasets show that simply adding more convolution layers does not make the trained model better.

One method for optimizing the loss function when we train a network, is the gradient descent method. This method updates weights until the gradient converges to zero. So we need to calculate the gradient using back propagation. With the increase of layers, the gradient tends to be zero. Once the gradient equals to zero, the gradient explosion problem appears, which is one reason why the error rate of the network increases with the increase of layers.

To solve the gradient explosion problem, Kaiming He and his team [8] propose the architecture of Deep Residual Learning by introducing a residual block.

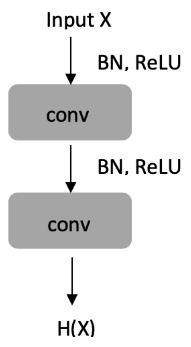


Figure 4.17: Plain layer.

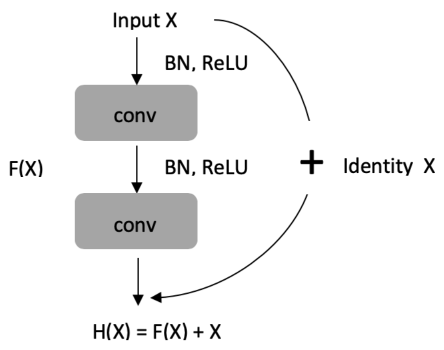


Figure 4.18: Residual block.

the input is x, and we want the fitting result (output) $H(x)$. In the use of residual block, the output is $F(x) + x$, where $F(x)$ is also the fitting result of x. As the gradient continues to decrease to zero during the training process, simply add plain layers, the output, $H(x)$, tends to be zero. If this occurs we probably cannot get any fitting results through the whole network in the end. If we use the residual block, while $F(x)$ approaches to zero, the output still has an identity x. By adding more layers we can get the same output as the input and this, at a minimum does not hurt the performance of the network. Our goal is higher than not just hurting the performance, for we want the network to extract more features. We can imagine that, if the additional layers are able to learn some more useful information, we will get a better model and a better testing result.

$$y_i = h(x_{i_l}) + F(x_i, W_i), \tag{4.13}$$

$$x_{i+1} = f(y_i), \tag{4.14}$$

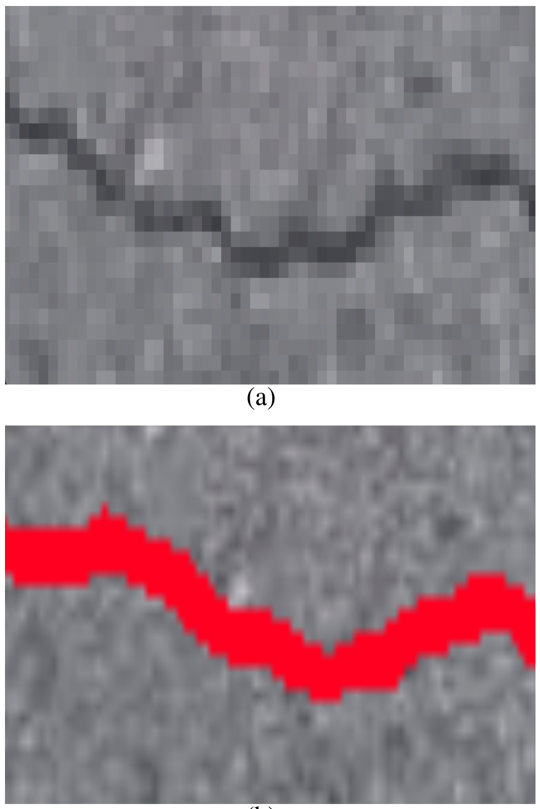
x_i represents the input of the ith residual unit and y_i represents the output of the ith residual unit. h is an identical transformation, F is a residual function, and f is a ReLU function. The performance of these two expressions is the same, but the experimental results of a residual network show the training error of 34-layer plain network is larger than the 18-layer network, but the training error of 34-layer residual network is better than the 18-layer residual network.

4.5.3 Heatmap

A heatmap is a representation of data useing color to represent values. A heatmap provides a convenient and intuitive way to visualize information in images. In the testing result, each pixel has a score. The score is used to determine whether a pixel is a crack pixel or not. A pixel with a higher score is likely to be a crack pixel and vice versa.

As shown in Figure 4.19, all pixels with a higher score are marked as red. From Figure 4.19(b), it can be seen that the region of the red pixels is basically the same as that of the crack in the original image Figure 4.19(a). By using a heatmap, the cracks in the pavement image can be directly distinguished.

We generate a heatmap to highlight the crack pixel. After that, we transfer the output into grey scale images, because the ground truth are all gray scale images, so that we can compare the output with the ground truth.


(a)

(b)

Figure 4.19: Heatmap example.

4.5.4 Proposed Network

The proposed network structure is shown below. The figures are generated by adapting the code from GitHub.

The main network structure and parameters of each layer refer to Figure 4.20 and Table 4.1. Input patches are a 256×256 RGB images. After each convolution layer, use the batch normalization and ReLU activation function to optimize parameters and improve training efficiency.

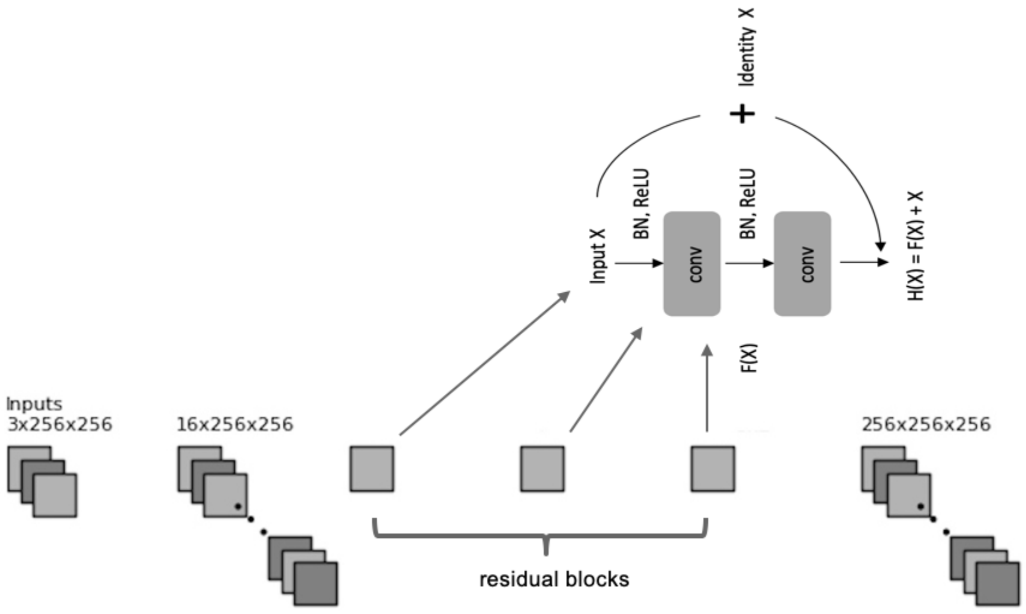


Figure 4.20: Main network structure.

	convolution layer	three residual blocks	convolution layer
filter	16	64	256
kernel	(5×5)	(5×5)	(3×3)
stride	1	1	1
padding	2	2	1

Table 4.1: Main network parameters.

For pavement crack detection, we not only need to detect how many defect pixels there are, but also need to know where these defect pixels are located. As shown in Figure 4.21 and Table 4.2, after the main network, add another convolution layer with one filter, kernel one, stride one, then flatten it as the output. Output size is $256 \times 256 = 65536$, each output indicates a pixel in the input image and a score to determine whether the pixel is a crack pixel or not.

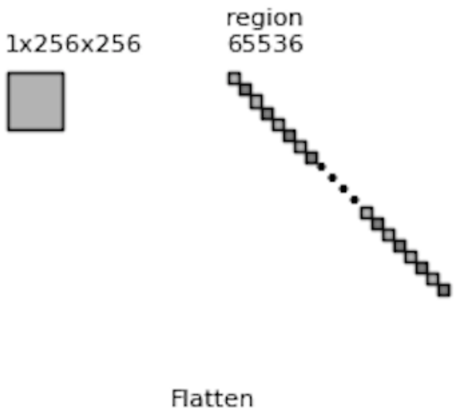


Figure 4.21: Network structure part 1.

	convolution layer
filter	1
kernel	(1×1)
stride	1
padding	0

Table 4.2: Network part 1 parameters.

For pavement crack classification, as shown in Figure 4.22, the classifier has three more convolution layers, two max pooling layers, two fully connection layers.

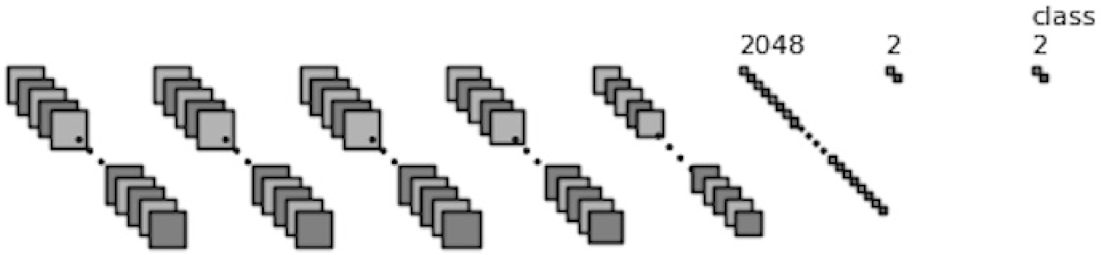
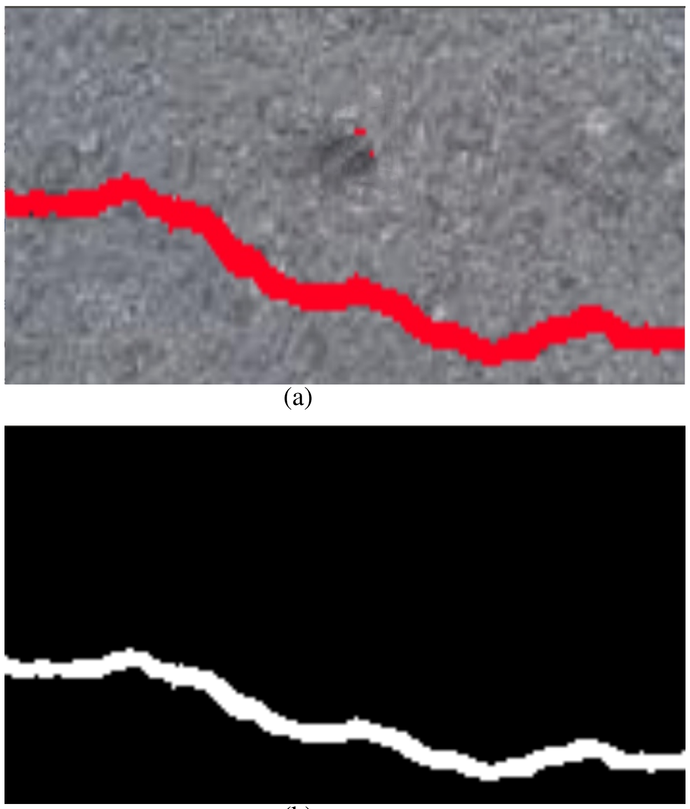


Figure 4.22: Network structure part 2.

If the testing result produces many false predictions, these will hurt the overall performance. In order to fix this issue, train a classifier, which contains two classes, where one indicates the non-crack patch and the other indicates the crack patch. If a predicted patch contains crack pixels, but the classifier predicts a non-crack patch, the classifier will remove this false alarm automatically.


(a)

(b)

Figure 4.23: False alarm removal.

Figure 4.23 illustrates this process. The false predictions in the black bounding box are removed.

Chapter 5

Experiment

We randomly select 13 images as the testing image in the dataset and the other 103 images as the training image. In the testing process, we write a Python script to read the testing image and the trained model, set a threshold according to the score of each pixel. The pixel whose score is higher than the threshold is considered as a crack pixel, otherwise the pixel is a non-crack pixel. Finally we evaluate the test result with ground truth and compare the performance with other existing methods.

5.1 Evaluation

Once we complete a machine learning model, we want to know the performance of the model prediction. Accuracy can tell the learning effect of the model, but only accuracy is not enough. For this reason, we usually calculate the precision, recall and F1-score to present how good the model is. F1-score can be regarded as a weighted average of model precision and recall. Its maximum value is one, minimum value is zero. The larger the value is, the better the model is. To calculate the F1-score, there are several definitions need to be clarified in Table 5.1.

	Positive Ground Truth	Negative Ground Truth
True Prediction	True Positive(TP)	True Negative(TN)
False Prediction	False Positive(FP)	False Negative(FN)

Table 5.1: Definitions.

A *Error*

Error is the ratio of all incorrect predictions. Error is used to measure the level of incorrect prediction.

$$Error = \frac{FalsePredictions}{AllPredictions} = \frac{FP + FN}{TP + TN + FP + FN}. \tag{5.1}$$

B *Accuracy*

Accuracy is the ratio of all correct predictions. Accuracy is used to measure how good the model is.

$$Accuracy = \frac{TruePredictions}{AllPredictions} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{5.2}$$

C *Precision*

Precision refers to the proportion of samples with a predicted value of true and a ground truth of positive in all samples with a predicted value of true.

$$Precision = \frac{TP}{TP + FP}. \tag{5.3}$$

D *Recall*

Recall, also known as recall rate, refers to the proportion of samples with a predicted value of true and a ground truth of positive in all samples with a ground truth of positive.

$$Recall = \frac{TP}{TP + FN}. \tag{5.4}$$

E *F1-score*

F1-Score, also known as Balanced Score, is defined as the weighted average value of precision and recall.

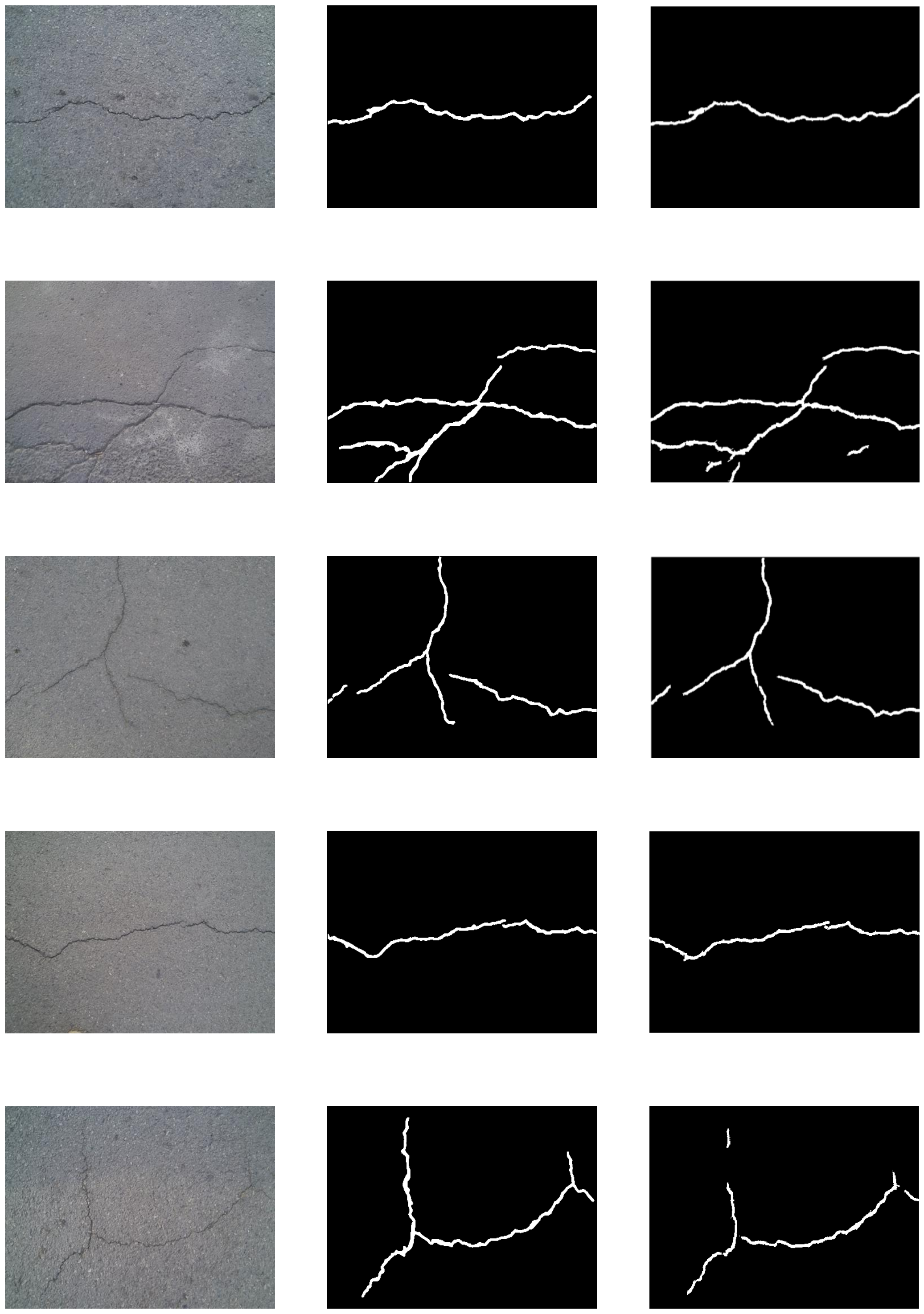
$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{5.5}$$

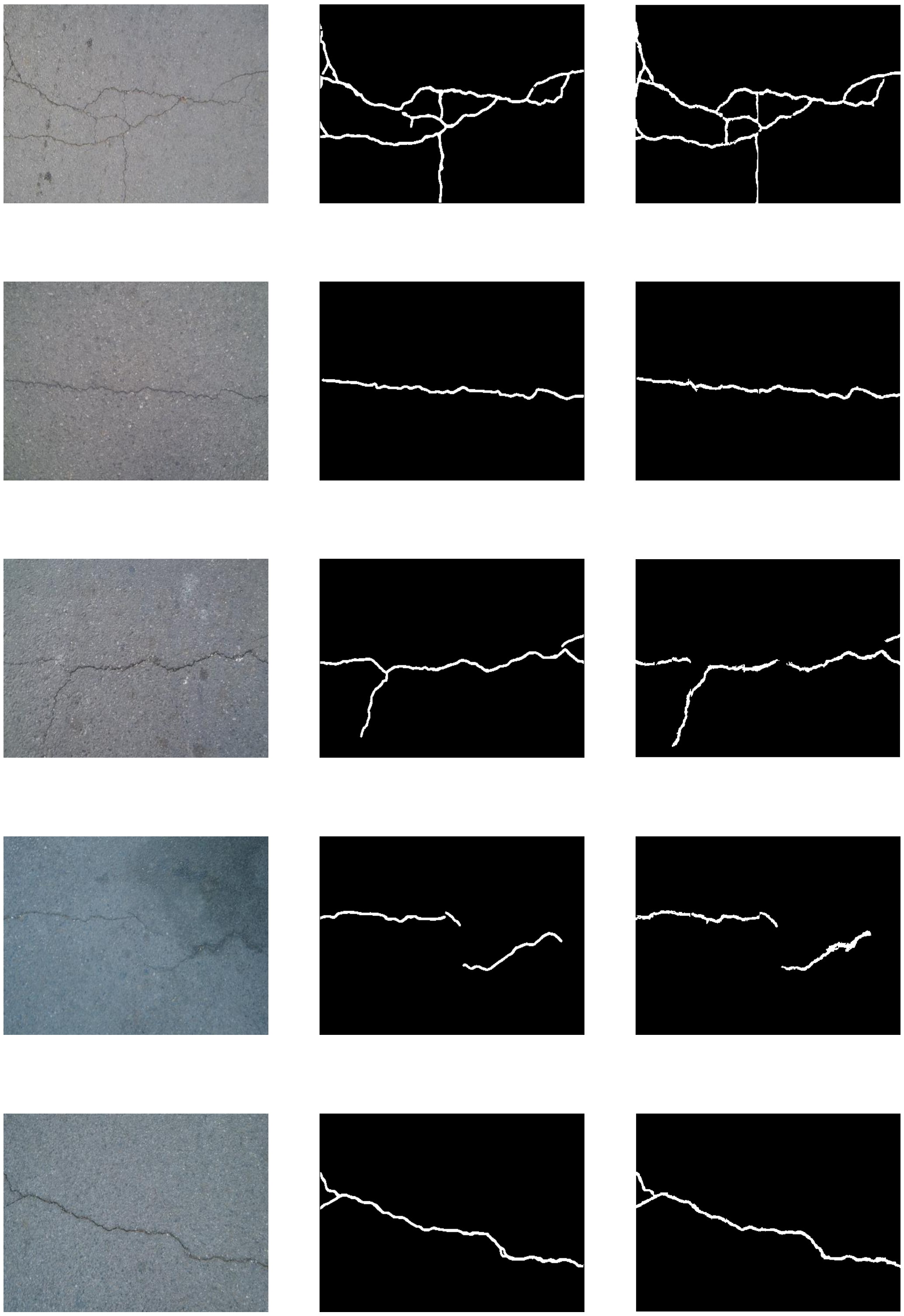
Since that the ground truth is labeled manually and relabeled by myself, in the original image there are transitional areas between crack and non-crack pixels. Under this consideration, we accept within two pixels and five pixels vicinity.

5.2 Results and Analysis

The testing result on CFD is shown in Figure 5.2. The decision threshold is set to five, and the result is shown as binary image, white area indicates the crack. From left to right:

original image, ground truth, binary output.





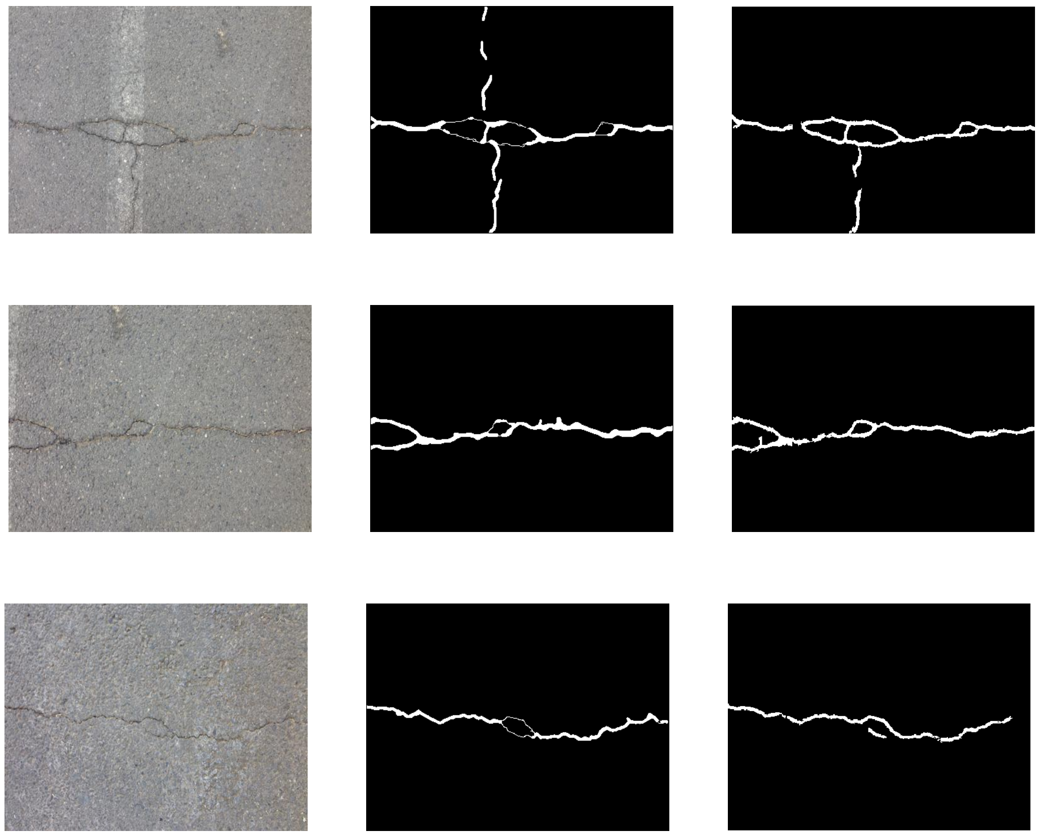


Figure 5.1: Crack detection result by proposed method.

From the binary images in Figure 5.2, it can be seen that this method can successfully detect the crack on the pavement surface, and avoid the influence of the image itself, such as poor illumination condition, noise and complex texture.

	Precision	Recall	F1-score
5-pixel	0.929	0.892	0.904
2-pixel	0.882	0.858	0.868

Table 5.2: Analysis of precision, recall and F1-score.

Table 5.2 shows with different vicinity, the different precision, recall and F-1 score. With two or five pixels vicinity, the evaluation result is pretty good.

5.3 Comparison with Existing Methods

We compare the proposed method with six other existing methods. They are Canny, CrackTree, CrackIT, FFA, CrackForest and MFCD. It can be seen clearly that the proposed method has the best performance among all the methods.

Method	Precision	Recall	F1-score
Canny	0.122	0.222	0.158
CrackTree	0.732	0.765	0.708
CrackIT	0.672	0.767	0.716
FFA	0.786	0.684	0.732
CrackForest	0.823	0.894	0.857
MFCD	0.899	**0.895**	0.880
Proposed Method with 5-pixels	**0.929**	0.892	**0.904**

Table 5.3: Precision, recall and F1-score of different methods.

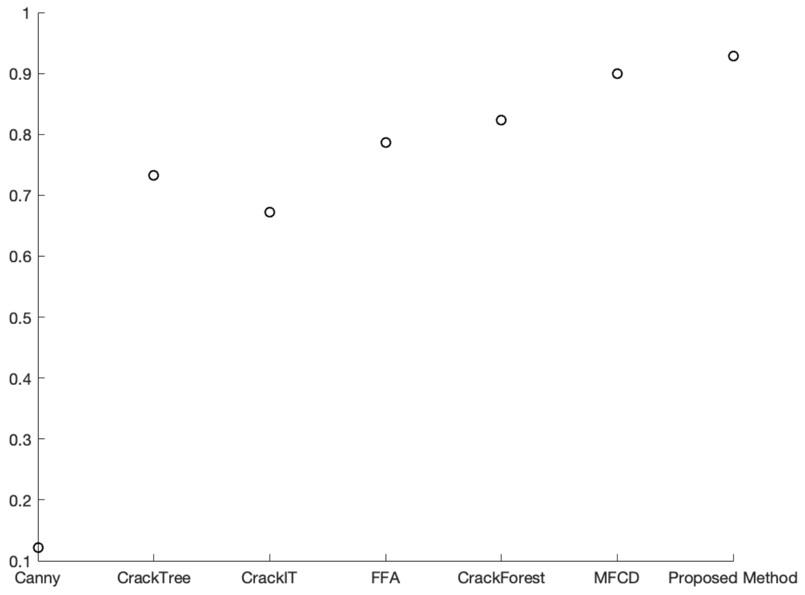


Figure 5.2: Comparison of precision.

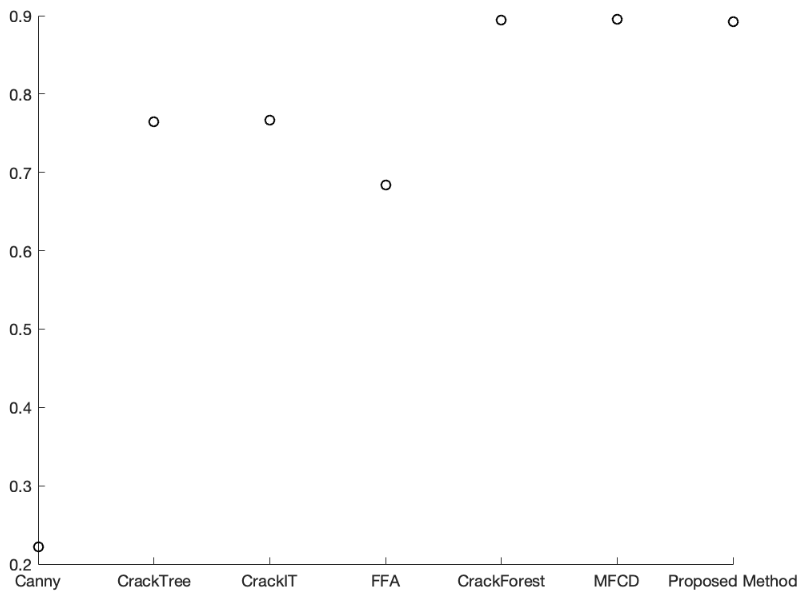


Figure 5.3: Comparison of recall.

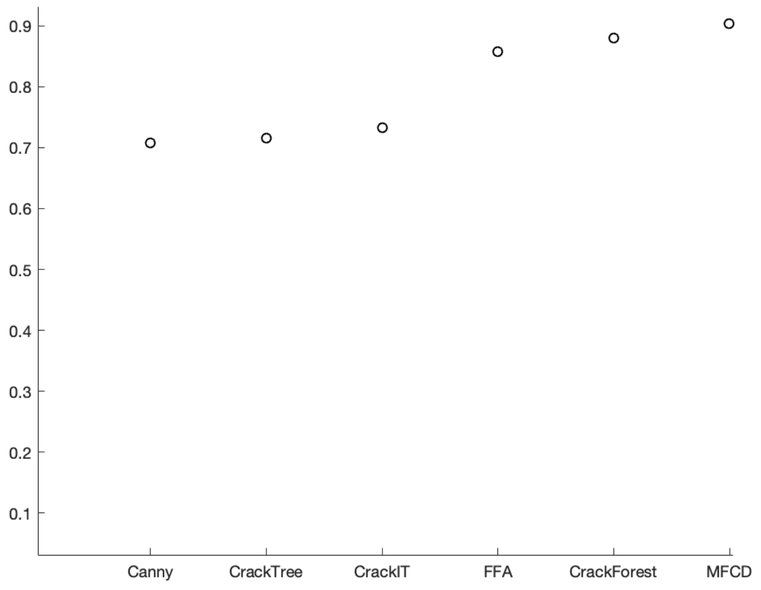


Figure 5.4: Comparison of F1-score.

Chapter 6

Conclusion and Future Works

In this thesis, we review some existing algorithms and networks on pavement crack detection research field. We propose a deep neural network, in order to gather better detection results, we adopt the residual block and use it in the network structure design. Using residual block allows me to add more layers in the training network, extract more feature from input, without reducing the accuracy. Before training, all training images and testing images need to be pre-processed and relabeled. We use a sliding window to segment a whole image into 16 sub-images, so that in the training process, the network can learn more useful images. In the Experiment chapter, we compare my proposed method with other popular and widely-used methods. The comparison results are presented in tables and graphs. It should be clear that the proposed method in this thesis has better results under the same pavement crack detection dataset.

We only focus on the Detection at this time, and the proposed network can describe the crack exactly, without considering the Classification problem. Although the network can be used to classify the type of cracks, it still needs more labels that contain class information of cracks such as longitude cracks, transverse cracks, and more other crack types. In addition, we only apply my method on CFD, since CFD is the most commonly used dataset for pavement crack detection right now. There are actually other datasets available. Because the pavement surface conditions are very diverse and complex, it is also necessary to verify whether the proposed method can be applied to various situations and whether it still has good results. There still are more works need to be done.

Bibliography

[1] Rabih Amhaz, Sylvie Chambon, Jrme Idier, and Vincent Baltazart. A new minimal path selection algorithm for automatic crack detection on pavement images. *2014 IEEE International Conference on Image Processing, ICIP 2014*, pages 788–792, 01 2015. doi: 10.1109/ICIP.2014.7025158.

[2] John Canny. A computational approach to edge detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986.

[3] Young-Jin Cha, Wooram Choi, and Oral Buyukozturk. Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 32:361–378, 03 2017. doi: 10.1111/mice.12263.

[4] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum. Detection of blood vessels in retinal images using two-dimensional matched filters. *IEEE Transactions on Medical Imaging*, 8(3):263–269, Sep. 1989. ISSN 0278-0062. doi: 10.1109/42.34715.

[5] Aurlien Cord and Sylvie Chambon. Automatic road defect detection by textural pattern recognition based on adaboost. *Computer-Aided Civil and Infrastruc-*

ture Engineering, 27(4):244–259, 2012. doi: 10.1111/j.1467-8667.2011.00736.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8667.2011.00736.x`.

[6] Limeng Cui, Zhiquan Qi, Zhensong Chen, Fan Meng, and Yong Shi. Pavement distress detection using random decision forests. In *International Conference on Data Science*, pages 95–102. Springer, 2015.

[7] Zhun Fan, Yuming Wu, Jiewei Lu, and Wenji Li. Automatic pavement crack detection based on structured prediction with the convolutional neural network. *CoRR*, abs/1802.02208, 2018.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[9] Yong Hu and Chunxia Zhao. A novel LBP based methods for pavement crack detection. *JPRR Vol 5, No 1 (2010); doi:10.13176/11.167*, 2010.

[10] H. Li, D. Song, Y. Liu, and B. Li. Automatic pavement crack detection by multi-scale image fusion. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12, 2018. ISSN 1524-9050. doi: 10.1109/TITS.2018.2856928.

[11] Li Li, Lijun Sun, Guobao Ning, and Shengguang Tan. Automatic pavement crack recognition based on bp neural network. *PROMET-Traffic&Transportation*, 26(1):11–22, 2014.

[12] T. S. Nguyen, S. Begot, F. Duculty, and M. Avila. Free-form anisotropy: A new method for crack detection on pavement surface images. In *2011 18th IEEE In-*

ternational Conference on Image Processing, pages 1069–1072, Sep. 2011. doi: 10.1109/ICIP.2011.6115610.

[13] H. Oliveira and P. L. Correia. Crackit an image processing toolbox for crack detection and characterization. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 798–802, Oct 2014. doi: 10.1109/ICIP.2014.7025160.

[14] Aiguo Ouyang and Yaping Wang. Edge detection in pavement crack image with beamlet transform. 09 2012. doi: 10.2991/emeit.2012.451.

[15] Ojala M. Pietikinen and D. Harwood. A comparative study of texture measures with classification based on feature distributions. *Pattern Recognition, vol. 29*, 1996.

[16] M. Salman, S. Mathavan, K. Kamal, and M. Rahman. Pavement crack detection using the gabor filter. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2039–2044, Oct 2013. doi: 10.1109/ITSC.2013.6728529.

[17] B Santhi, G Krishnamurthy, Swami Siddharth, and P.K. Ramakrishnan. Automatic detection of cracks in pavements using edge detection operator. *Journal of Theoretical and Applied Information Technology*, 36:199–205, 02 2012.

[18] Y. Shi, L. Cui, Z. Qi, F. Meng, and Z. Chen. Automatic road crack detection using random structured forests. *IEEE Transactions on Intelligent Transportation Systems*, 17(12):3434–3445, Dec 2016. ISSN 1524-9050. doi: 10.1109/TITS.2016.2552248.

[19] Jinshan Tang and Yanliang Gu. Automatic crack detection and segmentation using a hybrid algorithm for road distress analysis. *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3026–3030, 2013.

[20] Eduardo Zalama, Jaime Gmez-Garca-Bermejo, Roberto Medina, and Jos Llamas. Road crack detection using visual features extracted by gabor filters. *Computer-Aided Civil and Infrastructure Engineering*, 29(5):342–358, 2014. doi: 10.1111/mice.12042. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12042`.

[21] Allen Zhang, Qiang (Joshua) Li, Kelvin C. P. Wang, and Shi Qiu. Matched filtering algorithm for pavement cracking detection. *Transportation Research Record*, 2367 (1):30–42, 2013. doi: 10.3141/2367-04. URL `https://doi.org/10.3141/2367-04`.

[22] Qin Zou, Yu Cao, Qingquan Li, Qingzhou Mao, and Song Wang. Crack tree: Automatic crack detection from pavement images. *Pattern Recognition Letters*, 33:227–238, 02 2012. doi: 10.1016/j.patrec.2011.11.004.