

CLASSIFICATION AND PIXEL-LEVEL SEGMENTATION OF ASPHALT
PAVEMENT CRACKS USING CONVOLUTIONAL NEURAL NETWORKS

by

Tamim Adnan

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Construction & Facilities Engineering

Charlotte

2023

Approved by:

Dr. Don Chen (Chair)

Dr. Jake Smithwick

Dr. Yuting (Tina) Chen

Dr. Nicole Barclay

©
2023
Tamim Adnan
ALL RIGHTS RESERVED

ABSTRACT

TAMIM ADNAN. Classification and Pixel-Level Segmentation of Asphalt Pavement Cracks Using Convolutional Neural Networks. (Under the direction of DR. DON CHEN)

This study was conducted to answer two research questions. How do the chosen CNN models perform on publicly available asphalt pavement crack datasets for classification and segmentation? And how can these CNN models be fine-tuned to improve their performance? For classification, a ResNet50 model with transfer learning was explored, using eight different epochs and two optimizers, a total of sixteen combinations, and the optimal combination of these two hyperparameters was identified. On the other hand, the segmentation task was accomplished using a modified-UNet model on two different datasets, and the results were presented and discussed.

The dataset for classification was derived from a publicly available dataset, EdmCrack600. It was given the name AugCrack132. It has three principal types of asphalt pavement cracks: alligator, longitudinal, and transverse. The training dataset consists of 132 ground truth images, and the testing dataset has 56 raw crack images. The optimal classification accuracy occurred at epoch 500 with the 'adaptive moment estimation' or "Adam" optimizer algorithm, while the least accuracy occurred at epoch 40 with 'stochastic gradient descent' or "SGD" optimizer algorithm. The classification accuracy on the overall dataset varied from 8% to 58%, the F1 score was from 1.463% to 59%, the precision ranged from less than 1% to 68%, and the recall varied from 8.9% to 59%.

The modified-UNet model was trained and tested on two published pavement crack datasets to segment asphalt pavement cracks. The first dataset included 470 images and corresponding masks obtained from the EdmCrack600 dataset through a preprocessing, and it was named EdmCrack470. Furthermore, 206 images from the CRACKTREE260 dataset were used to evaluate the modified-UNet model, as a result, the dataset has been named CRACTREE206 here. At epoch 30, the model achieved an IoU of 67%, precision 96%, recall 65%, and F1 78% score for the EdmCrack470 dataset. Moreover, the values of the evaluation metrics for CRACKTREE206 are: IoU 60%, precision 95%, recall 58%, and F1 72%. In addition, the predicted masks were assessed based on three criteria.

The research highlights that transferred-ResNet50 has successfully classified the pavement crack types in some hyperparameter combinations. Hence, this model should be applied to a more organized classification dataset where ground truths of the cracks need to be more specific with severity levels. Also, this study recommended considering more hyperparameter combinations to evaluate the model performance for classification tasks.

Furthermore, the modified-UNet model could contribute to pavement crack segmentation. The addition of multi-scale layers in both encoder-decoder networks can be used to improve the performance. This inclusion will assist the model in differentiating the cracks from noises and redundant background information. Also, some additional data preprocessing and cleaning are the potential tasks to improve the accuracy of the predicted shapes.

From this study, it is clear that state-of-the-art CNN models can be used for evaluating both new and existing pavement crack datasets. Creating a new model for specific datasets is challenging because the biased parameters of the model would be less effective for another dataset. Also, it can raise complexities when the model framework is simple or complicated compared to the data size and structure. Therefore, fine-tuning the existing models can be more productive in pavement crack classification and segmentation tasks.

Keywords: Classification, Segmentation, Pavement Crack, Convolutional Neural Networks.

ACKNOWLEDGEMENTS

As an active author of this research, I'm grateful to my respected advisor, Dr. Don Chen, for his constant guidance, support, and encouragement throughout the process. I'm also thankful to the honorable committee members, Dr. Jake Smithwick, Dr. Yuting (Tina) Chen, and Dr. Nicole Barclay for their feedback and evaluation of this research with patience, time, and effort.

Now, I want to thank Dr. Wenwu Tang from the Geography and Earth Sciences Department, for allowing me into his class to learn high-performance computing for my research which has accelerated and influenced the progress of this thesis. In addition, I thank Dr. Tang's Ph.D. students, Zachery Slocum, Tianyang Chen and Yanfang Su for mentoring, and helping to understand the deep learning models.

The great academic advice and support from Sandra Krause, Assistant Dean for Graduate Academic Service, at UNC Charlotte was helpful to overcome my academic hurdles and challenges. I'm really grateful for such an academic environment from UNC Charlotte.

I'm extending my gratitude to my family, friends, Bangladesh Student Organization at UNC Charlotte (Ekush) for immense support during the entire journey of this research. My heartfelt gratitude goes to my parents, Afroza Khatun, and Abdul Aziz, including my younger brother Titumir and my uncle Md. Abdullahel Kafi for unwavering support and encouragement.

Finally, thank you to everyone who directly or indirectly inspired and supported me for this research.

DEDICATION

I'm dedicating this research to those who are the pillars of my success in every step of my life besides my parents, young brother and my uncle.

To my maternal grandparents Mojammel Hoque and Asma Hoque.

In the loving memory of my late paternal grandparents, Abdul Goni and Osiron Bibi, including my late paternal uncles, Abdul Gophur and Abdul Bador.

To my maternal uncles, Amir Abdullah. and Abduallahel Baqi.

To my five paternal aunts, Fatema Khatun, Nazma Khatun, Nasima Khatun, Surovi Khatun and Sumi Salma including my half-grandma, Rokeya Begum, and my elder cousin, Farida.

Finally, to my friends, relatives, cousins, teachers, neighbors, and all well-wishers who supported me to be myself.

TABLE OF CONTENTS

LIST OF TABLES	xi
LIST OF FIGURES	xii
CHAPTER 1: INTRODUCTION	
1.1 Motivation	1
1. 2 Problem statement	6
1.3 Research questions	6
1.4 Research objectives and scopes	6
1.5 Expected contributions	7
1.6 Research overview	7
CHAPTER 2: LITERATURE REVIEW	
2.1 Pavement management systems (PMS)	8
2.2 Different types of pavement cracks	9
2.2.1 Alligator crack	10
2.2.2 Longitudinal crack - (Wheel paths, outside of wheel path and lane joints)	11
2.2.3 Transverse crack	13
2.3 Convolutional neural network	14
2.3.1 CNN for classification	18
2.3.2 CNN for segmentation	19
2.4 Transfer learning	21
2.5. Hyperparameters tuning in convolutional neural networks	23
2.6 Some novel CNN architectures and frameworks for asphalt pavement crack classification and segmentation	24
2.7 Fine tuning existing CNN models from scratch or transfer learning for pavement crack classification and segmentation	26

2.8 Published pavement distress datasets and annotation methods	29
CHAPTER 3: METHODOLOGY	
3.1 Classification of crack types with transferred-ResNet50	32
3.1.1 Data preparation– extracting three types of cracks from EdmCrack600 dataset	33
3.1.2. Data preparation - augmentation of alligator cracks	35
3.1.3 Data preparation– augmentation in the code for increasing training data	36
3.1.4 Transfer learning in ResNet50	37
3.1.5 Experimental set up for training the transferred-ResNet50	38
3.1.6 Shell script writing for parallel computing	39
3.1.7 Evaluation method of transferred - ResNet50 model performance	39
3.2 Segmentation of the cracks with the modified-UNet	41
3.2.1 Data preparation of EdmCrack470 dataset for segmentation	43
3.2.2 Data preparation of CRACKTREE206 for segmentation	45
3.2.3 Experimental setup for training the modified-UNet model	45
3.2.4 Shell scripting for training the modified-UNet model	46
3.2.5 Evaluation method of the modified-UNet model	46
CHAPTER 4: RESULTS	
4. 1 Classification of pavement cracks using transferred-ResNet50	47
4. 2 Segmentation performance of the modified UNet Model	50
4.2.1 UNet model performance for segmenting EdmCrack600 Dataset	50

4.2. 2 UNet model performance for segmenting CRACKTREE dataset	57
CHAPTER 5: DISCUSSIONS	
5.1 Discussion on classification task with ResNet50	64
5.2 Discussion on segmentation task with modified-UNet	65
CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS	
6.1 Conclusions on classification task with ResNet50	67
6.2 Conclusions on segmentation task with UNet	67
6.3 Recommendations	68
REFERENCES	69
APPENDIXES	
Appendix A: Accuracy calculations in Table 5 on overall dataset in row c for in ResNet50 for classification	76
Appendix B: Alligator cracks from EdmCrack600 in AugCrack132 dataset	77
Appendix C: Corresponding masks of the alligator cracks in AugCrack132 dataset	78
Appendix D: Loss curves in different epochs and optimizer combinations in ResNet50	79
Appendix E: Shell script and run time email for transferred-ResNet 50	87
Appendix F: Python code for classification in transferred-ResNet50	88
Appendix G: Shell script and run time email for UNet for EdmCrack600 dataset	94
Appendix H: Python code for segmentation in modified-UNet for EdmCrack470 dataset	95

Appendix I: Python code for segmentation in modified-UNet for CRACKTREE206 dataset	101
Appendix J: Shell script and run time email for UNet for CRACKTREE260 dataset	107
Appendix K: Link of codes and datasets for classification, segmentation in GitHub and Google Drive	108

LIST OF TABLES

Table 1. State-of-the-art deep learning models for pavement crack classification and segmentation, source: [19]	4-5
Table 2. Evolution of CNN backbones collected from [49]	22
Table 3. Hyperparameters of CNNs	23-24
Table 4. Datasets on pavement distress for different tasks	30-31
Table 5. Performance of transferred-ResNet50 for classification of cracks classes	47-48
Table 6. Performance of the modified-UNet model on EdmCrack470 dataset.	51
Table 7. Performance of the modified-UNet model on CRACKTREE206 dataset.	58

LIST OF FIGURES

Figure 1. Crack detection using neural network, source: [18]	2
Figure 2. Pavement crack types, source: [4]	3
Figure 3. Pavement crack segmentation, source: [17]	4
Figure 4. The main process of road pavement management, source: [4]	8
Figure 5. (a) Alligator crack, (b) Longitudinal crack, and (c) Transverse crack	9
Figure 6. Alligator crack - low severity, source: [3]	10
Figure 7. Alligator crack - moderate severity, source: [3]	11
Figure 8. Alligator crack - high severity according, source: [3]	11
Figure 9. Longitudinal - low severe (On the wheel path), source: [3]	12
Figure 10. Longitudinal - low severe (Lane joints), source: [3]	12
Figure 11. Longitudinal - high severe (outside the wheel path), source: [3]	13
Figure 12. Longitudinal crack -high severity (lane joints), source: [3]	13
Figure 13. Transverse crack - low severity, source: [3]	14
Figure 14. Transverse crack - moderate severity, source: [3]	14
Figure 15. Three common layers in convolutional neural networks	15
Figure 16. Typical structure of convolutional neural network, source: [32]	15
Figure 17. Convolution a) Formula b) Operation	15-16
Figure 18. a) Stride b) Padding c) Max Pooling d) ReLU of convolutional neural network	16-17
Figure 19. Architecture of original ResNet50	19
Figure 20. Original UNet model architecture, source: reproduced by following [45]	20
Figure 21. Concept of transfer learning, source: reproduced by following [50]	22
Figure 22. Framework for transfer learning, source: reproduced by following [62]	26
Figure 23. Classification framework, source: [19]	28

Figure 24. Segmentation framework, source: [19]	29
Figure 25. Summary of the methodology	32
Figure 26. Classification task	33
Figure 27. Exceptional alligator crack with low severity, source dataset: [3]	34
Figure 28. Exceptional alligator crack with moderate severity, source dataset: [3]	34
Figure 29. Data structure of the AugCrack132 dataset for classification task	35
Figure 30. a) Vertical random flip b) Horizontal random flip to increase the alligator cracks	36
Figure 31. Augmented and original ground truths in a batch	37
Figure 32. Applying transfer learning to ResNet50	38
Figure 33. (a) Distribution of true positives, false positives, true negatives, false negatives, (b) Precision and (c) Recall. Source: reproduced by following [77]	40-41
Figure 34. Segmentation using the modified-UNet model for this study	42
Figure 35. Structure of the modified-UNet model for this research	43
Figure 36. Raw image and corresponding ground truth after center cropping, source: [3]	44
Figure 37. (a) Original image from [3], (b) The blanked image removed from the EdmCrack470 created in this study.	44-45
Figure 38. Intersection over Union formula, source: reproduced by following [76]	46
Figure 39. Loss curve at epoch 500 and optimizer “Adam”	49
Figure 40. Loss curve at epoch 1000 and optimizer “SGD.”	50
Figure 41. Loss curve for UNet Model on EdmCrack600 dataset	51
Figure 42. Predicted transverse crack from EdmCrack600 (a)	52
Figure 43. Predicted transverse crack from EdmCrack600 (b)	52
Figure 44. Predicted transverse crack from EdmCrack600 (c)	52
Figure 45. Scattered shape of the predicted cracks (a)	53

Figure 46. Scattered shape of the predicted cracks (b)	53
Figure 47. Scattered shape of the predicted cracks (c)	53
Figure 48. Predicted crack region has been scattered (a)	54
Figure 49. Predicted crack region has been scattered (b)	54
Figure 50. Predicting the backgrounds only rather than predicting crack regions (a).	54
Figure 51. Predicting the backgrounds only rather than predicting crack regions (b).	55
Figure 52. The shades as cracks and brighter zones as background (a)	55
Figure 53. The shades as cracks and brighter zones as background (b)	55
Figure 54. The shades as cracks and brighter zones as background (c)	56
Figure 55. The shades as cracks and brighter zones as background (d)	56
Figure 56. Less bright raw images with redundant background information affects the predicted mask.	56
Figure 57. Less bright raw images affect the predicted mask	57
Figure 58. Loss curve for UNet model on CRACKTREE206 dataset.	57
Figure 59. Comparison of predicted crack shapes (a)	58
Figure 60. Comparison of predicted crack shapes (b)	58
Figure 61. Comparison of predicted crack shapes (c)	59
Figure 62. Partially predicted crack shapes (a)	59
Figure 63. Partially predicted crack shapes (b)	59
Figure 64. Partially predicted crack shapes (c)	60
Figure 65. Almost predicted the shape but missed one little portion. (a)	60

Figure 66. Almost predicted the shape but missed one little portion. (b)	60
Figure 67. Shades are mixing with the ground truths (a)	61
Figure 68. Shades are mixing with the ground truths (b)	61
Figure 69. Affected by the shades of raw images	61
Figure 70. Affected by the shades of raw images.	62
Figure 71. Affected by the shades of raw images and bright zones are considered as background in the predicted mask	62
Figure 72. Less bright background caused for unrecognizable mask	62
Figure 73. Less bright background caused for unrecognizable mask	63
Figure 74. Less bright background caused for unrecognizable mask	63
Figure 75. Loss curve at epoch 10 and optimizer “Adam”	79
Figure 76. Loss curve at epoch 10 and optimizer “SGD”	79
Figure 77. Loss curve at epoch 20 and optimizer “Adam”	80
Figure 78. Loss curve at epoch 20 and optimizer “SGD”	80
Figure 79. Loss curve at epoch 30 and optimizer “Adam”	81
Figure 80. Loss curve at epoch 30 and optimizer “SGD”	81
Figure 81. Loss curve at epoch 40 at optimizer “Adam”	82
Figure 82. Loss curve at epoch 40 and optimizer “SGD”	82
Figure 83. Loss curve at epoch 50 and optimizer “Adam”	83
Figure 84. Loss curve at epoch 50 and optimizer “SGD”	83
Figure 85. Loss curve at epoch 100 and optimizer “Adam”	84
Figure 86. Loss curve at epoch 100 and optimizer “SGD”	84
Figure 87. Loss curve at epoch 500 and optimizer “Adam”	85
Figure 88. Loss curve at epoch 500 and optimizer “SGD”	85
Figure 89. Loss curve at epoch 1000 and optimizer “Adam”	86

Figure 90. Loss curve at epoch 1000 and optimizer “SGD”

86

CHAPTER 1: INTRODUCTION

1.1 Motivation

The transportation industry necessitates a significant percentage of the government budget for the development of modern transportation infrastructures. A large portion of the budget goes toward inspecting and maintaining the road infrastructures. Specifically, in the United States, road infrastructures have received a D grade indicating a poor pavement condition according to the 2021 ASCE (American Society of Civil Engineering) report card [1]. Therefore, many Departments of Transportation (DOTs) go through regular road condition assessments where the principal task is to identify and evaluate cracks, patching, rutting, and other distress [2]. Distress on the road infrastructures indicates the sign of damages. Assessing the severity of these damages is crucial to select appropriate treatments to ensure safer road networks and daily commutes [3]. Also, high-traffic cities regularly need quicker treatments to make up for the damages. These maintenance should be completed with optimal expenses to minimize the total lifecycle cost [4]. For these reasons, identifying and quantifying pavement distresses and treatments is a principal task in pavement maintenance which demands a lot of time, money, and inspection instruments with human effort. Thus, automatic pavement distress or crack assessment is ongoing research to replace such conventional distress inspection approaches in pavement management systems (PMS) [5], [6]. Gradually some efforts were made, for example, in designing of an image processing algorithm to predict the transverse or longitudinal cracks, which took binary images as input, and the algorithm could find the initial direction of cracks and follow the direction of breakpoints of the cracks for classification purposes [7]. With the progression of time, researchers have moved forward to applying machine-learning approaches such as Support Vector Machine (SVM), Random Forests (RF), or Decision Tree (DT) models to predict pavement crack classes [8]. Recently, neural networks in deep learning have grown significantly for classification, semantic segmentation, and detection or localization of cracks on pavement crack images.

Figure 1 presents crack detection by localizing the crack regions. Object detection models, such as YOLO, Faster RCNN, and mask-RCNN are suitable models for such detection tasks. These models have CNN architectures as backbones; for example, the YOLO model has three major components: backbone (CNN Architecture), neck, and head. Thus, it is understandable that the CNN model has implications for all of the three principal tasks of pavement crack assessment: crack detection, classification, and segmentation.

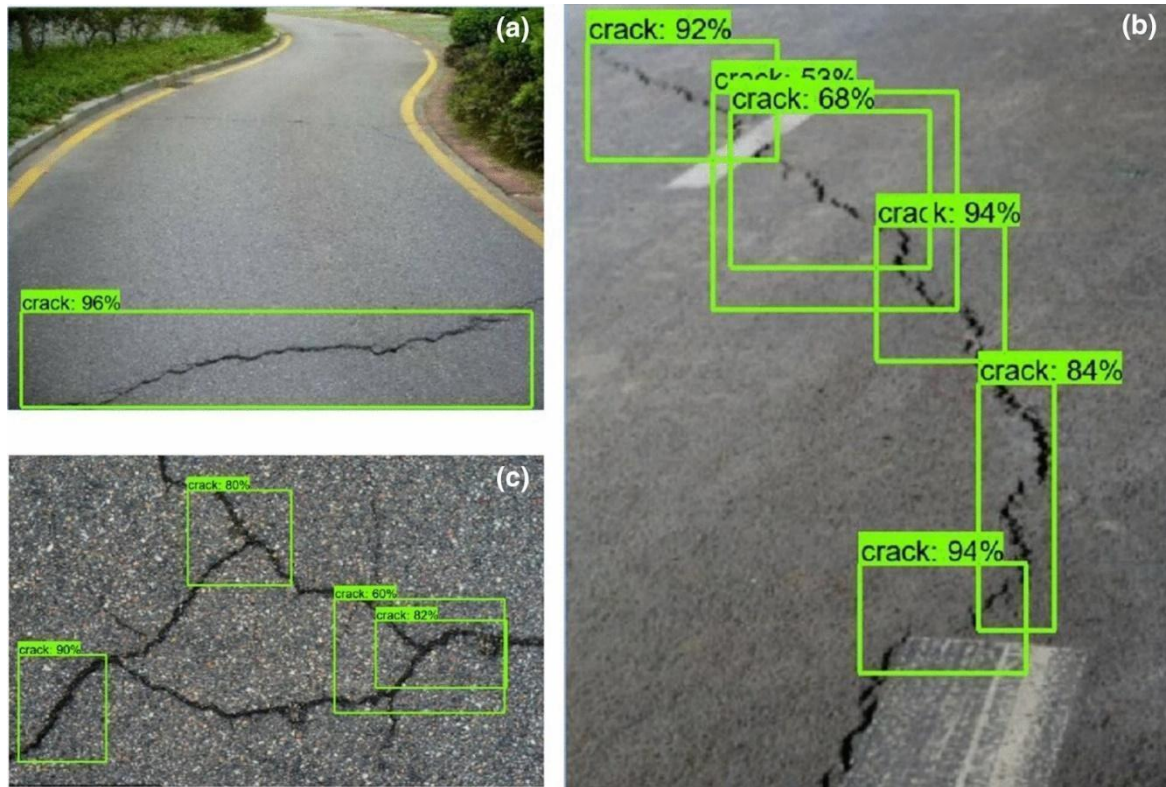


Figure 1. Crack detection using neural network, source: [18]

For classification and segmentation tasks, convolutional neural networks (CNN) is an increasingly popular model, a special type of Neural Network that makes mathematical convolution operations to extract features and patterns of characteristics. Some prominent CNNs are VGG 16 and VGG19 [9], GoogLeNet [10], AlexNet [11], and Inception [12] which were applied in various studies for different tasks including pavement crack assessments [13]. These models can classify different crack types as presented in Figure 2 and have been used by many researchers in multiple studies [14], [15].

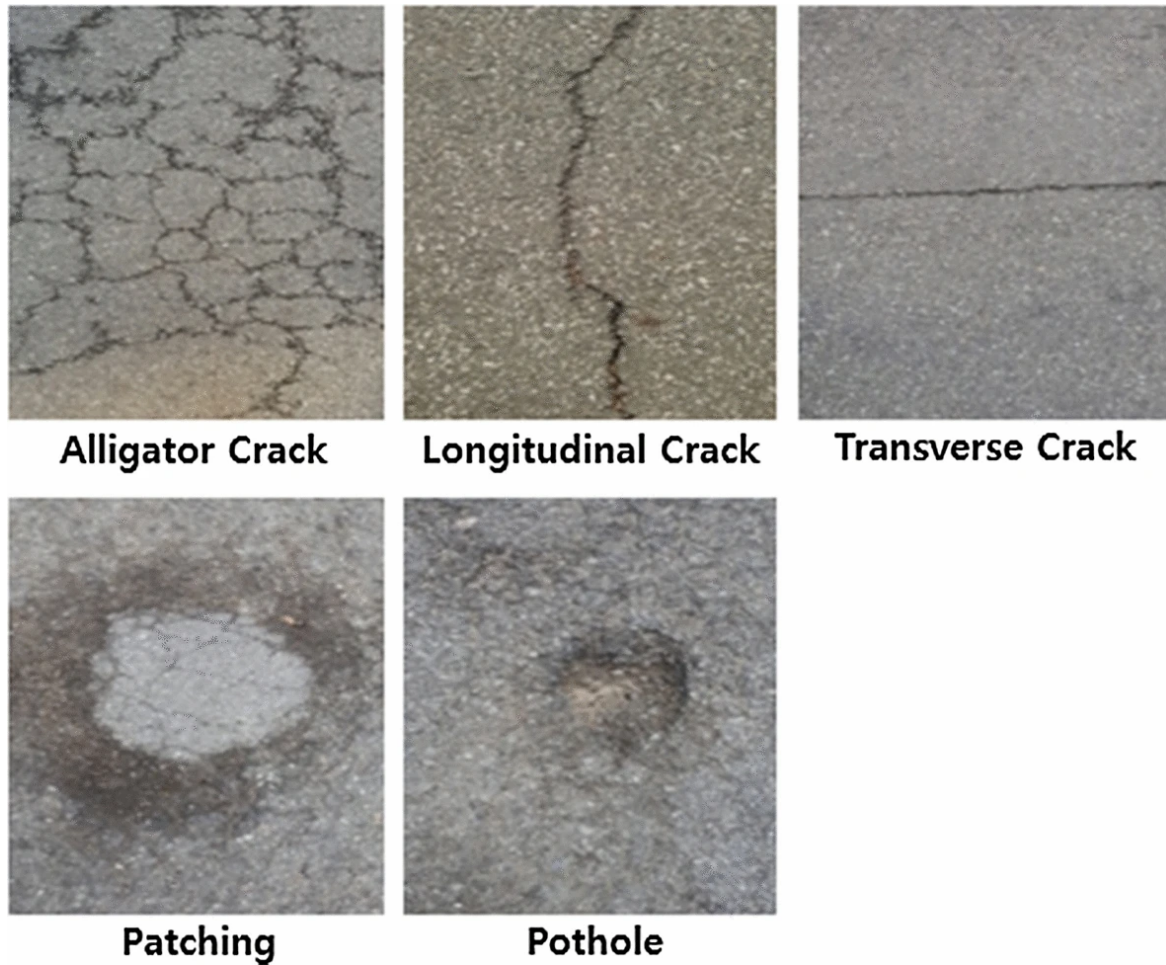


Figure 2. Pavement crack types, source: [4]

For segmentation tasks as shown in Figure 3, there are two types of CNN models commonly used: encoder-decoder architecture (UNet models) and fully convolutional networks (FCN). UNet models can extract features from input images in the encoder network by down-sampling or subsampling, then it up-samples or enlarges the elements just as the input size at the decoder network. On the other hand, FCN models store spatial information in the convolutional layers. Researchers have also proposed several novel CNN architectures for crack segmentation [3], [16].

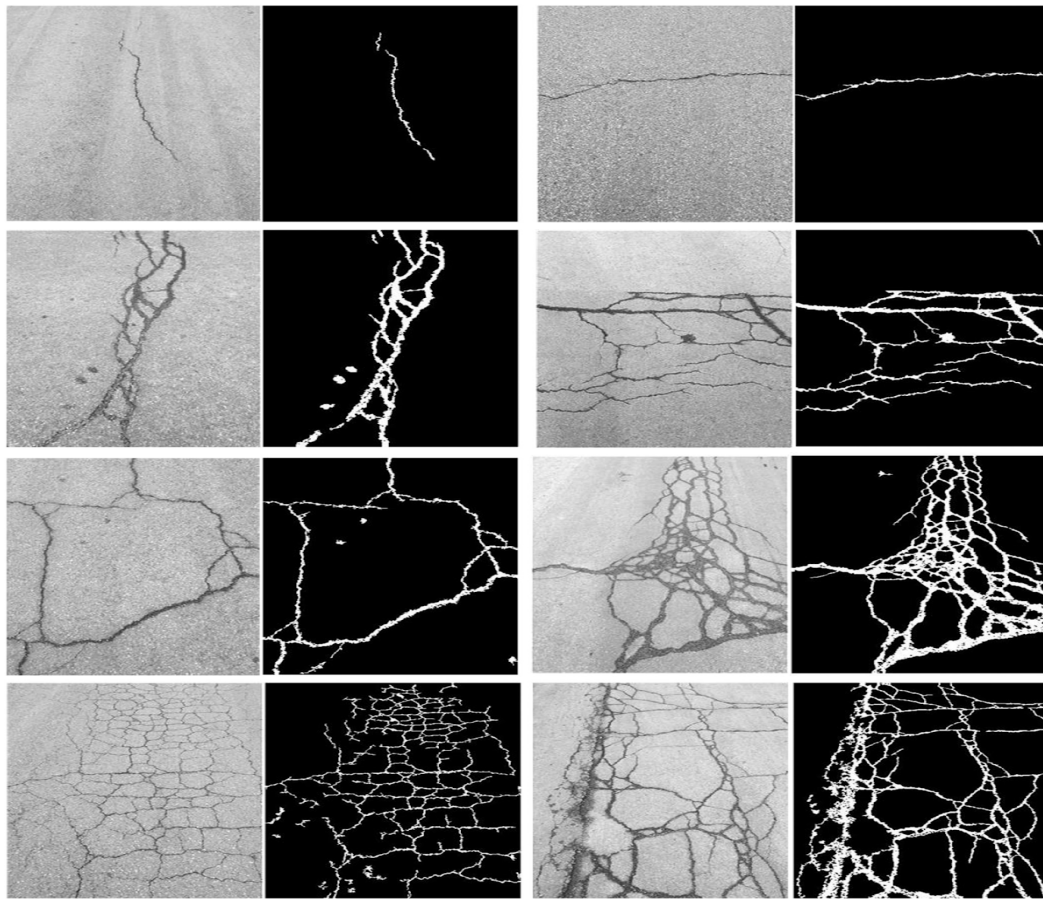


Figure 3. Pavement crack segmentation, source: [17]

Yang et al. [19] reviewed 40 pavement datasets and divided them into classification, segmentation and crack detection-oriented datasets, and listed the state-of-the-art models for pavement crack classification and segmentations which are mentioned in Table 1.

Table 1. State-of-the-art deep learning models for pavement crack classification and segmentation, source: [19]

Algorithms for Crack Classification	Algorithms for Crack Segmentations
AlexNet	Original Hierarchical Neural Network
VGG-16	SegNet
VGG-19	FCN-8s
GoogLeNet	UNet
ResNet-101	DeepLab v2
ResNeXt-101	DeepLab v3

ResNeSt-101	PSPNet V1
Swin Transformer-Base	ASPP-Net
Swin Transformer-MB	DeepCrack
ShuffleNet	CrackNet based DeepLab v3+
ShuffleNet V2	CrackNet based DenseNet
	CrackNet based Full Res-ResNet

Several studies have curated new pavement distress datasets besides proposing novel CNN architectures to assess pavement distress. These published datasets have also been useful during the training of existing state-of-the-art CNN models to classify the crack-types and segment them accordingly. Some researchers trained these latest and versatile CNN models from scratch, while many scholars applied a specific range of transfer learning and achieved a significant level of accuracy [20]. Transfer learning is effective for training deep neural networks with less data and time [21]. Hence, it is important to investigate these advanced model performances with different ranges of transfer learning for pavement crack assessment.

A study has shown a maneuver of exploring the performance of transferred CNN models in various conditions. In the Eisenbach et al. [22], the authors conceptualized a CNN model, then modified it with some additional layers. The two models were fine-tuned with some techniques of regularization method. Performance of the both models in those hyperparameter combinations were compared with other models on the same datasets. Additionally, some studies fine-tuned the existing CNN models with transfer learning for evaluating the cracks of various infrastructures (such as cracks in concrete infrastructures) [23], [24].

From these motivations, this study has investigated the performance of two widely accepted and progressive CNN models: one for classification and another for segmentation tasks. Transfer learning has been applied to the fully connected layer of ResNet50, an existing CNN model, which is responsible for classification tasks. For the segmentation task, this study used a modified-UNet model. This UNet model has been upgraded in the encoder-decoder section which can potentially overcome the cropping issue found in the originally developed UNet model in [45]. The modified-UNet model was used before in segmenting the cityscape dataset [75].

For classification, a new dataset has been created and named as AugCrack132 from a published segmentation-oriented dataset. The model for segmentation was trained and tested using two publicly available segmentation-oriented datasets. Segmentation is different from classification; thus, this study used transfer learning for the classification-oriented CNN model but trained the segmentation-oriented CNN from scratch.

From this research, it can be understood whether these progressive CNN models are sufficient for evaluating asphalt pavement cracks, on both currently published and the future datasets instead of developing new deep learning models or frameworks each time. Newly developed CNN architectures might be biased to specific datasets and raise model complexities when model structure is simple or complicated compared to the data size and structure. This is the reason this study is focusing on using the latest available CNN models.

1.2 Problem statement

This study aimed to conduct the pavement crack classification using ResNet50 by applying transfer learning and pavement crack segmentation by training a modified-UNet model from scratch.

The ResNet-50 model has been trained and evaluated for classification of different types of cracks using the AugCrack132 dataset. It was derived from a publicly available segmentation-oriented dataset EdmCrack600 [3]. Also, a modified-UNet model was trained and tested on two segmentation-oriented datasets. At the beginning, the model was trained and evaluated with the EdmCrack470 dataset, prepared from [3]. However, the EdmCrack470 dataset has noises, such as background information, brightness variations, and the model was mistakenly considering wrong things as cracks. This is why the model was trained and tested on the CRACKTREE206 dataset to observe how it performs on the other segmentation-oriented datasets.

Furthermore, exploration of these versatile models with published pavement crack datasets for classification and segmentation can present their mutual feasibility and interoperability to each other. As a result, assessment of the pavement cracks can be achieved with existing CNN models through optimal approaches rather than frequently developing novel CNN architectures.

1.3 Research questions

This study has used existing advanced CNN models for pavement crack classification and segmentation tasks, and to answer the following questions:

- How do the chosen CNN models perform when using publicly available asphalt pavement cracks datasets for classification and segmentation?
- How can these CNN models be fine-tuned to improve their performance?

1.4 Research objectives and scopes

This study aimed to conduct the pavement crack classification using ResNet50 by applying transfer learning and pavement crack segmentation by training a modified-UNet model from scratch. The main objectives of this study are:

- Creating a new dataset, the AugCrack132 dataset, was derived for multi-class classification. It was prepared from an existing segmentation-oriented pavement crack dataset which includes three principal types of asphalt

pavement cracks: transverse, longitudinal, and alligator collected in each corresponding folder.

- Applying transfer learning in a state-of-the-art CNN classifier model, ResNet50 to conduct multi- class classification of pavement cracks.
- Comparing the performance of transferred-ResNet50 on eight different epochs (10, 20, 30, 40, 50, 100, 500, 1000) and two optimizers (adaptive moment estimation or ‘Adam’, ‘stochastic gradient descent’ or ‘SGD’) in a total of 16 different experiments to optimize hyperparameters.
- Using a modified-UNet model for segmentation of asphalt pavement cracks from EdmCrack470 and CRACKTREE206 datasets which are two publicly available pavement crack datasets.

1.5 Expected contributions

As this research is fine-tuning two different CNN models for two different tasks, the contribution of this study for assessing pavement cracks are:

- A new classification-oriented dataset derived from a segmentation-oriented dataset.
- A new transferred model for crack classification: alligator cracks, longitudinal cracks, and transverse cracks.
- A new model for asphalt pavement crack segmentation.

1.6 Research overview

Chapter 2 is a literature review to understand the different types of cracks, convolutional neural networks, transfer learning, and hyperparameter optimization. The literature also reviewed the original ResNet50 and UNet model and the published pavement distress datasets. Afterward, the methodology is in Chapter 3, to illustrate the full research approach for classification and segmentation with transferred-Resnet50 and modified-UNet models, respectively. Chapter 4 presents the results. Then the following chapter 5 has discussed and analyzed based on obtained results. The next chapter 6 is to draw the conclusions and recommendations from this research. Then the references and appendix sections are provided.

CHAPTER 2: LITERATURE REVIEW

This study aimed to conduct the pavement crack classification using ResNet50 by applying transfer learning and pavement crack segmentation by training a modified-UNet model from scratch. The current progress of pavement management systems (PMS) for asphalt pavement crack assessment, definition of cracks and their severity levels are presented in this chapter. Another important section of this chapter is about convolutional neural networks (CNN). Then this chapter has explored the previous studies on asphalt pavement crack classification and segmentation which developed the novel neural network-based frameworks. The next section has also searched the studies that fine-tuned the existing and state-of-the-art neural networks for assessing the same tasks. The last section of this chapter represents some publicly available pavement crack datasets used in previous studies.

2.1 Pavement management systems (PMS)

Pavement management systems (PMS) are structured decision-making tools to maintain quality pavements and safe road networks in a cost-effective manner. PMS has steadily improved in decision-making because of continuous research with advanced technologies. It is significant for ensuring consistent decision-making across different organizational levels [25]. Ismail et al. [26] described the four components of PMS: network inventory, pavement condition evaluation, performance prediction models, and planning methods. These components are essential for developing a decision-making tool, particularly for cost-efficient rehabilitation and maintenance of pavements. Ragnoli et al. [27] defined the regular inspection, detection, and mitigation of pavement distress as the proactive approach of PMS. And also, the study [27] defined the replacement of the pavement surface as the conventional approach. The significance of the proactive approach to maintaining road pavements can be clearly understood from these two approaches. Ha et al. [4] has presented the essential steps of PMS for road damage maintenance in Figure 4.

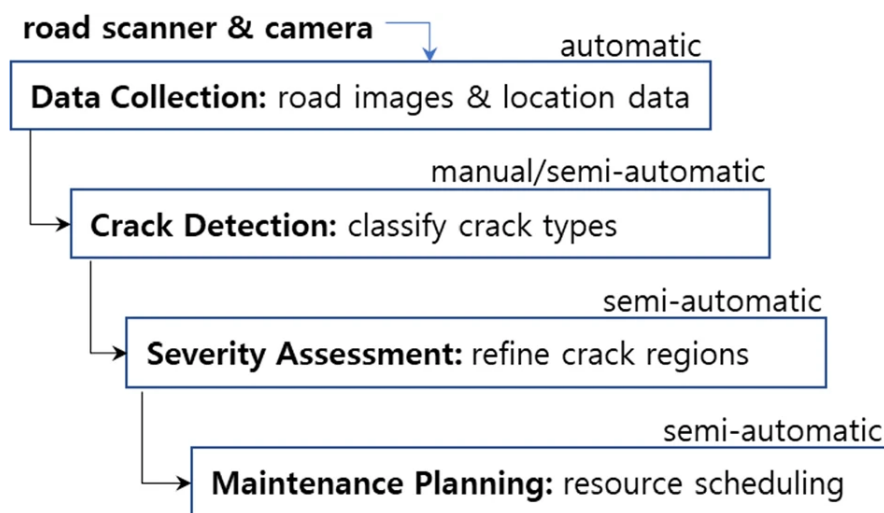


Figure 4. The main process of road pavement management, source: [4]

According to Figure 4, the automatic pavement data collection in PMS has been improved with advanced cameras and other visual devices, although automatic pavement crack detection and their severity assessments are still in the research phase. Addition of these two phases in PMS can advance the road maintenance systems by saving extra costs, time and efforts.

2.2 Different types of pavement cracks

Pavement distress includes cracks, patching, potholes, etc. Cracks are a common type of distress on asphalt pavement surfaces. In asphalt pavements, alligator, longitudinal and transverse are three major cracks [28]. The researcher of this study conducted a survey on the local roads near the UNC Charlotte campus to collect some pavement crack images. For example, Figure 5(a) is an alligator crack taken by the researcher's iPhone 13 pro max. In addition, longitudinal crack in Figure 5(b) and transverse crack in Figure 5(c) are captured with the same phone.

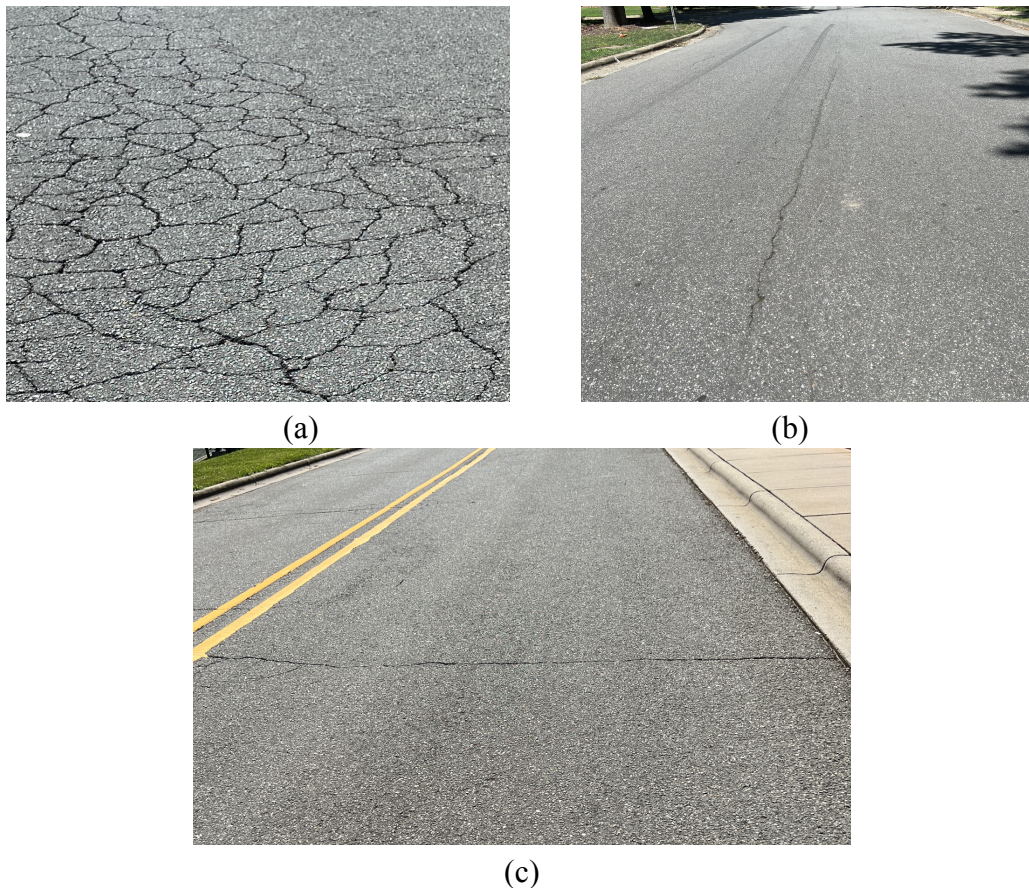


Figure 5. (a) Alligator crack, (b) Longitudinal crack, and (c) Transverse crack

(Source: the author of this research)

Severity of road cracks is determined by the dimension of the cracks [29]. This study has followed the NCDOT Digital Imagery Distress Evaluation Handbook [30] to find the severity of different cracks in EdmCrack600 dataset to prepare a new dataset, AugCrack132, for classification. Nevertheless, cracks of all severity conditions were grouped as their principal

crack type in the AugCrack132 dataset. For instance, alligator crack with any severity has been generalized as alligator crack there due to time constraints.

2.2.1 Alligator crack

Alligator crack is the most severe crack that occurred because of heavy loads on the road surfaces. Alligator crack has low, medium or high severities. Crack severities were not considered in this study due to time constraints. Therefore, alligator cracks included in the EdmCrack600 dataset [3] were extracted as “alligator crack” to form the AugCrack132 dataset for classification task.

Alligator crack - low severity: When the sealed or unsealed longitudinal crack stands on the wheel path then they should be considered as alligator cracks with low severity according to [30]. Also, for the crack areas with no or fewer interconnection with cracks of 1 / 8-inch width with no spalling, can be considered as alligator cracks with low severity [30]. Nevertheless, the single longitudinal cracks on the wheel have been considered as longitudinal cracks in this study as shown in Figure 9. The reason is that the deep learning models need additional training to understand the single longitudinal crack shapes as alligator cracks based on their location on the wheel path. Only if the longitudinal cracks on the wheel path in EdmCrack600 dataset are connected to other cracks, it is considered as an alligator crack for their low severity. For example, the crack in Figure 6 is a low severe alligator crack because it is on the wheel path with a few interconnected cracks. Moreover, Figure 27 at methodology chapter is another example of alligator crack of low severity where three longitudinal cracks were parallelly situated on wheel paths.



Figure 6. Alligator crack - low severity, source: [3]

Alligator crack - moderate severity: The moderate severe alligator cracks have some interconnections of cracks with spalling that form an alligator pattern. Widths of these cracks are 1/4 inch according to [30]. The crack image in Figure 7 obtained from the EdmCrack600 dataset has several connected cracks. Also, interconnections on each crack indicate a moderate severity of alligator crack.



Figure 7. Alligator crack - moderate severity, source: [3]

Alligator crack - high severity: According to [30], severe alligator cracks have severe or moderate spalled cracks that form an alligator pattern. Widths for such cracks are $\frac{3}{8}$ inch [30]. The alligator crack collected from the EdmCrack600 dataset in Figure 8 has high severity because it connects several cracks and has multiple interconnections.



Figure 8. Alligator crack - high severity according, source: [3]

2.2.2 Longitudinal crack - (Wheel paths, outside of wheel path and lane joints)

The single longitudinal cracks with any severity level on the wheel path has been included in longitudinal cracks in the AugCrack132 dataset in this study. The reason is to train the deep learning model on a consistent shape and orientation of longitudinal crack. In addition, the single longitudinal cracks on the outside of the wheel path and lane joints in any severity condition were included as the longitudinal crack in the AugCrack132 dataset. In the

EdmCrack600 dataset, most of the longitudinal cracks at both of these locations have low and high severe cracks.

Longitudinal - low severe: According to [30], crack widths cannot be measured because it is less than 1/4 inch, and it is a closed or unsealed crack. Figure 9 is longitudinal - low severe (on the wheel path) and Figure 10 is the low severe longitudinal cracks at lane joints. These cracks are in such a good condition that their widths cannot be estimated.

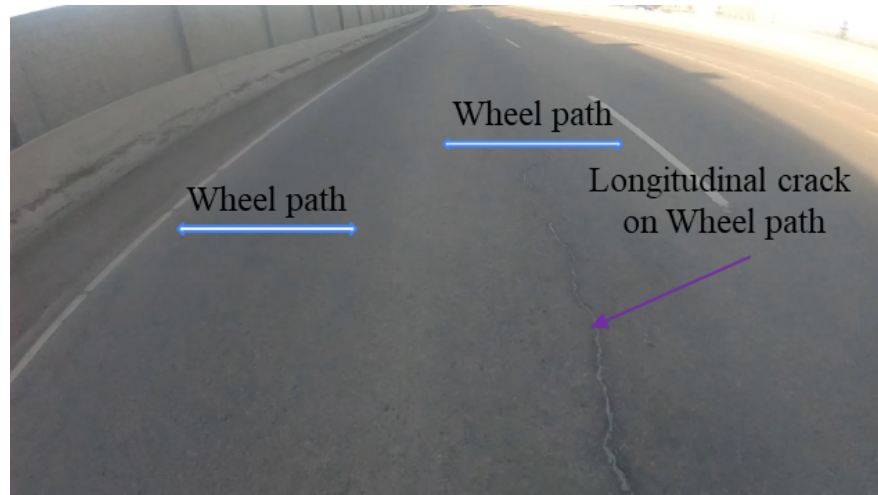


Figure 9. Longitudinal - low severe (On the wheel path), source: [3]

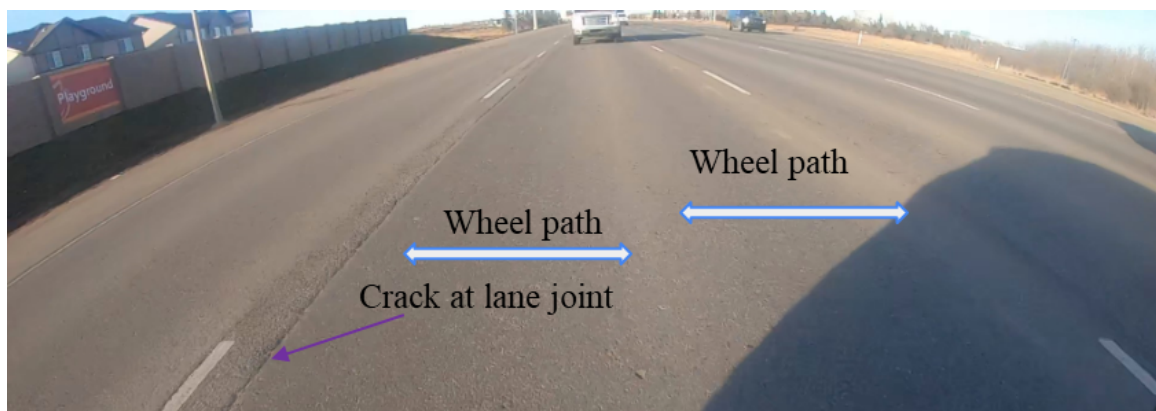


Figure 10. Longitudinal - low severe (Lane joints), source: [3]

Longitudinal - high severe: High severe longitudinal cracks have severe spalling or adjacent random cracking according to [30]. Longitudinal crack - outside the wheel path in Figure 11 and longitudinal crack at lane joints in Figure 12 have high severities. These were included in the AugCrack132 dataset as longitudinal cracks.

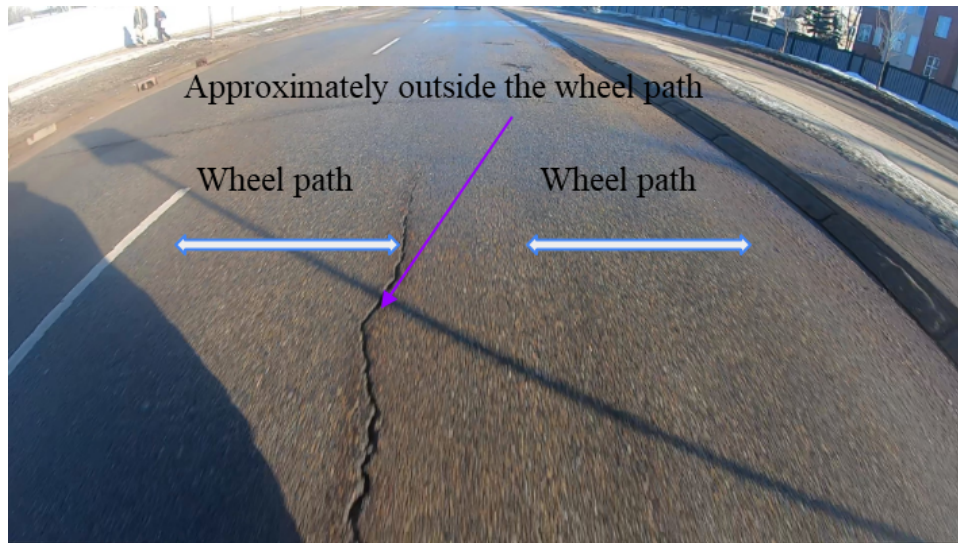


Figure 11. Longitudinal - high severe (outside the wheel path), source: [3]

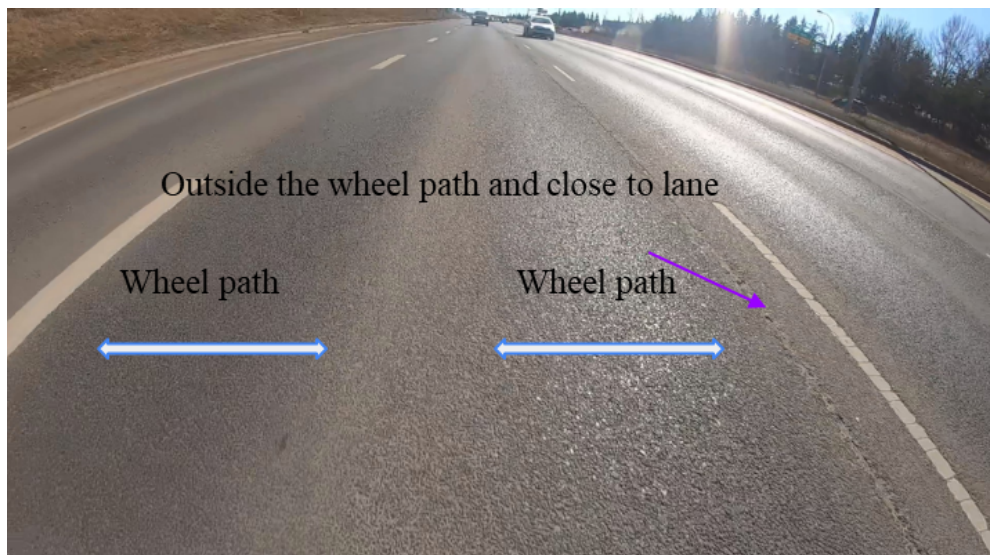


Figure 12. Longitudinal crack -high severity (lane joints), source: [3]

2.2.3 Transverse crack

This study has generalized the low and moderate severe transverse cracks found from the EdmCrack600 dataset as transverse cracks in the AugCrack132 dataset for classification.

Transverse crack - low severity: A closed or unsealed transverse crack with less width of $\frac{1}{4}$ inch is considered a low severe transverse crack [30]. For example, the transverse crack in Figure 13 is a transverse crack with low severity.

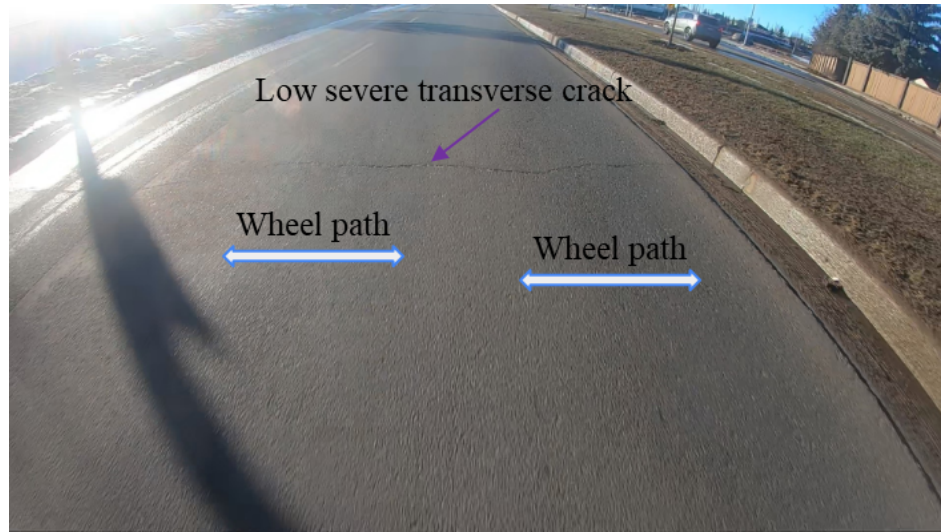


Figure 13. Transverse crack - low severity, source: [3]

Transverse crack - moderate severity: According to [30], if the crack width is between 1/4 inch to 1/2 inch, an unsealed transverse crack, having moderate severity [30]. For example, the transverse crack in Figure 14, a moderate transverse crack found in the source dataset, EdmCrack600. It was included in AugCrack132.

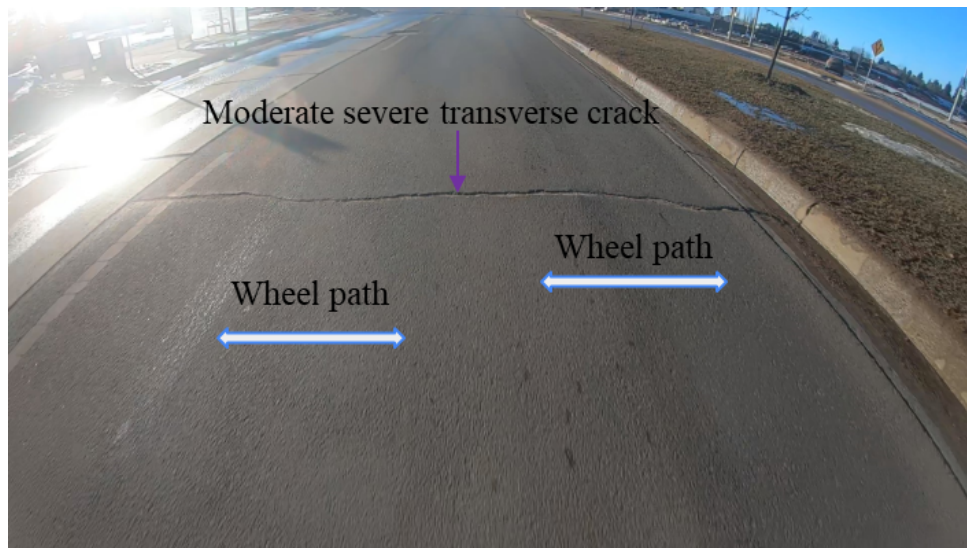


Figure 14. Transverse crack - moderate severity, source: [3]

2.3 Convolutional neural network

Neural networks (NN) in deep learning were inspired from the thinking process of a human brain, where neurons have connections and every layer stores and passes information to other layers. Convolutional neural network (CNN) is a growingly applied neural network which has three principal layers, namely convolution, pooling, and fully connected layers [31], as presented in Figure 15 and Figure 16. CNN models can classify, segment, and detect objects from visual data. Convolutional layers in a CNN can extract features with kernels. Moreover,

the max-pooling layers can down-sample, or subsample features by taking only maximum values from the feature maps. And the function of fully connected layers in the CNNs is to classify the characteristics from the features.

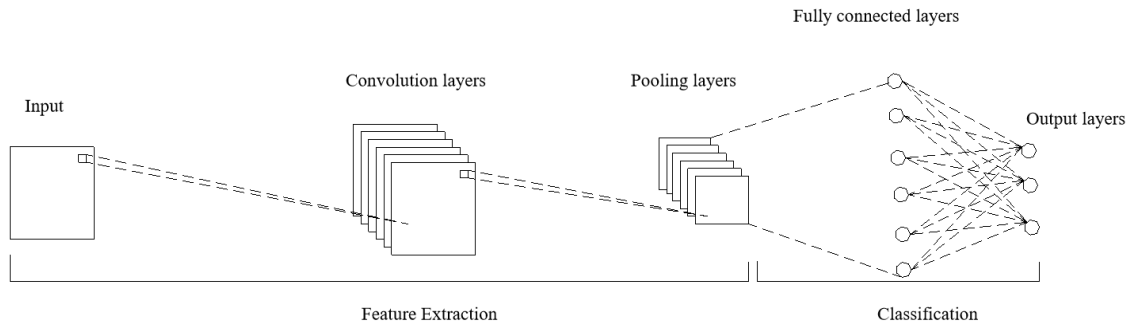


Figure 15. Three common layers in convolutional neural networks

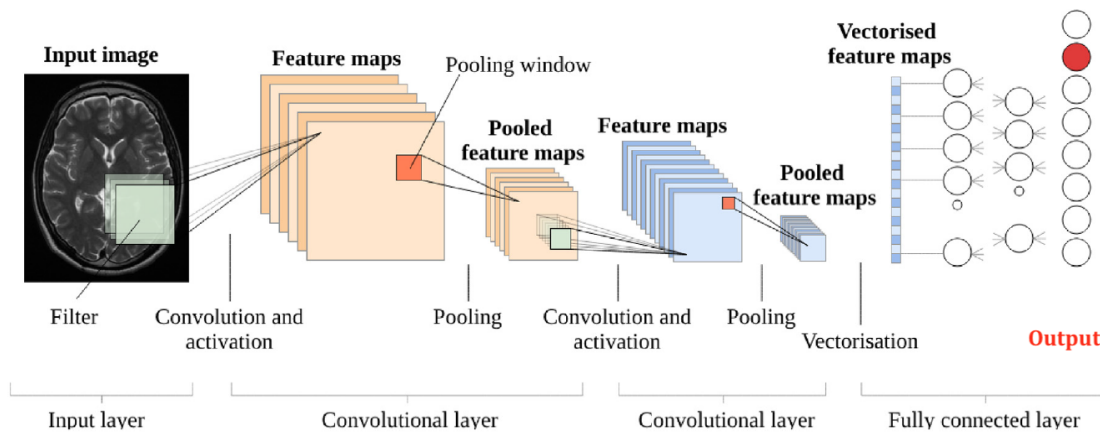
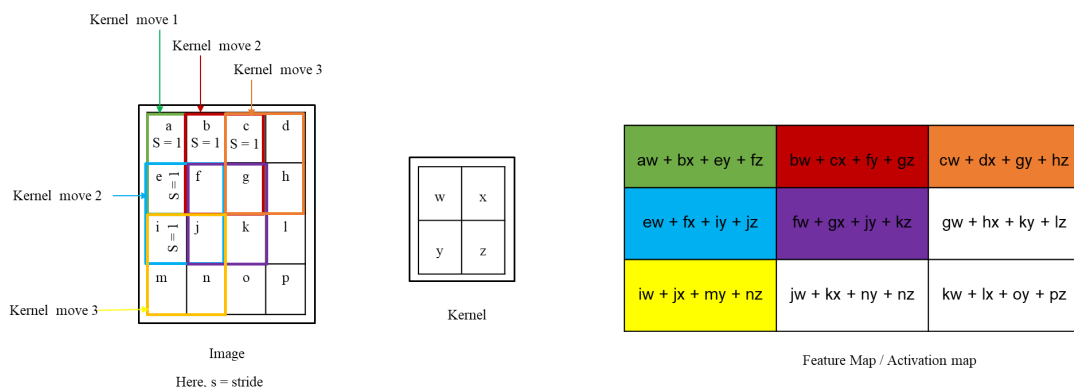
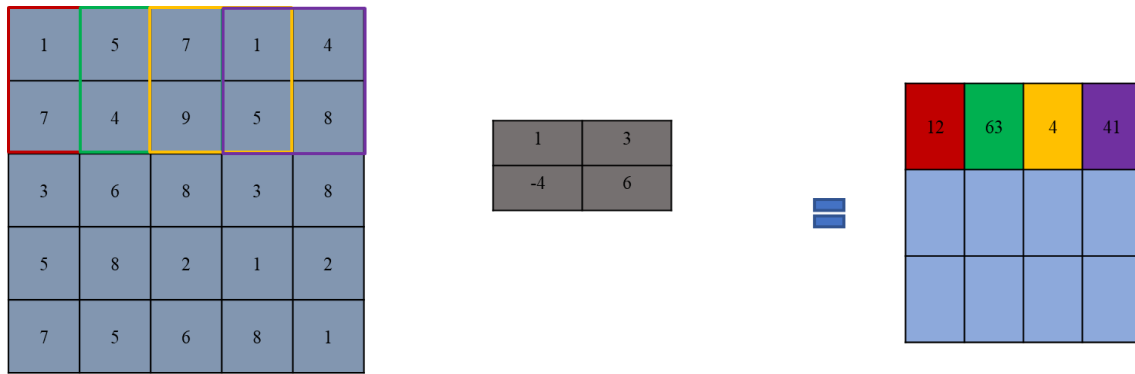


Figure 16. Typical structure of convolutional neural network, source: [32]

Convolution is a mathematical operation as shown in Figure 17 (a) and Figure 17 (b). It extracts features from input images by making dot vector multiplication with the filter or kernel (for a 2D filter, the filter is the same as the kernel). The output of the dot vector multiplication is a feature map as shown in Figure 17 (a).

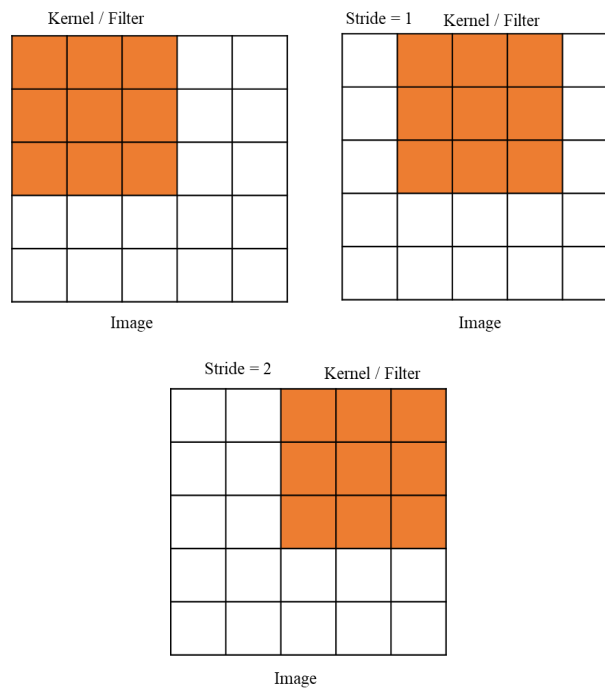


(a) Convolution operation, reproduced and modified by following [33]



b) Convolution operation reproduced and modified by following [34]
Figure 17. Convolution a) Formula b) Operation

Figure 18 presents (a) stride, (b) padding, (c) max pooling, and (d) ReLU. Stride is an important element of CNN as presented in Figure 18 (a). When the stride is 1, the $n \times n$ filter moves on the whole image after 1 pixel. Padding is important to ensure that the kernel strides over all the basic elements of the input matrix. It expands the input matrix by adding fake pixels to the borders of the matrix. If zero padding is 1, the original image pixel will have one pixel thick around the original image with 0 values [36]. Max pool as depicted in Figure 18 (c), extracts the maximum values from the feature maps of the input image to down-sample, or subsample the feature maps. Finally, the ReLU operation converts the negative values to zero for activating the layers, as shown in Figure 18 (d).



(a) Stride, reproduced and modified by following [35]

Image

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

Image with padding

(b) Padding, reproduced and modified by following [36]

one depth of an image

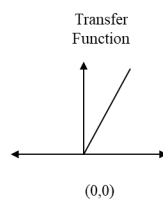
3	7	3	4
6	5	8	6
3	4	2	6
1	5	0	3

Filters 2 x 2 and stride 2

7	8
5	6

(c) Max pooling operation, reproduced and modified by following [37]

17	24	-30	38
19	-200	27	105
25	-10	29	-19
100	78	17	27



17	24	0	38
19	0	27	105
25	0	29	0
100	78	17	27

(d) ReLU activation function, reproduced and modified by following [37]

Figure 18. a) Stride b) Padding c) Max Pooling d) ReLU of convolutional neural network

Feedforward and Backpropagation are two important processes to train a neural network by tuning the parameters (bias and weights) of a CNN model on specific datasets. The feed-forward back propagation network mainly works to learn and map the relationship between the input and output by adjusting the weight values to minimize the error [38]. During the feedforward process, neural networks generally initialize random values for the weights and pass information from input to hidden nodes to output using activation function. Then in the backpropagation, gradient of loss or error is calculated from the predicted and true labels. According to this gradient of loss rate, the model adjusts the bias and weights (parameters) to minimize the loss [39]–[41]. There are some optimizers, such as ‘Adam’ and ‘SGD’, that can be used in the backpropagation process to adjust model parameters by converging the gradient of loss to a level and updating the weights accordingly [42]– [44].

2.3.1 CNN for classification

Figure 19 shows the architecture of the ResNet50 CNN model. It has five convolutional layers, an average pooling layer, one fully connected layer and a SoftMax. From the second to the fifth convolutional layers the ResNet50 has 3, 4, 6 and 3 residual blocks respectively. Residual blocks can connect the input and output of the convolution layers with skip connection or residual connections. The classification layer of the ResNet50 has 2048 parameters for 1000 classes.

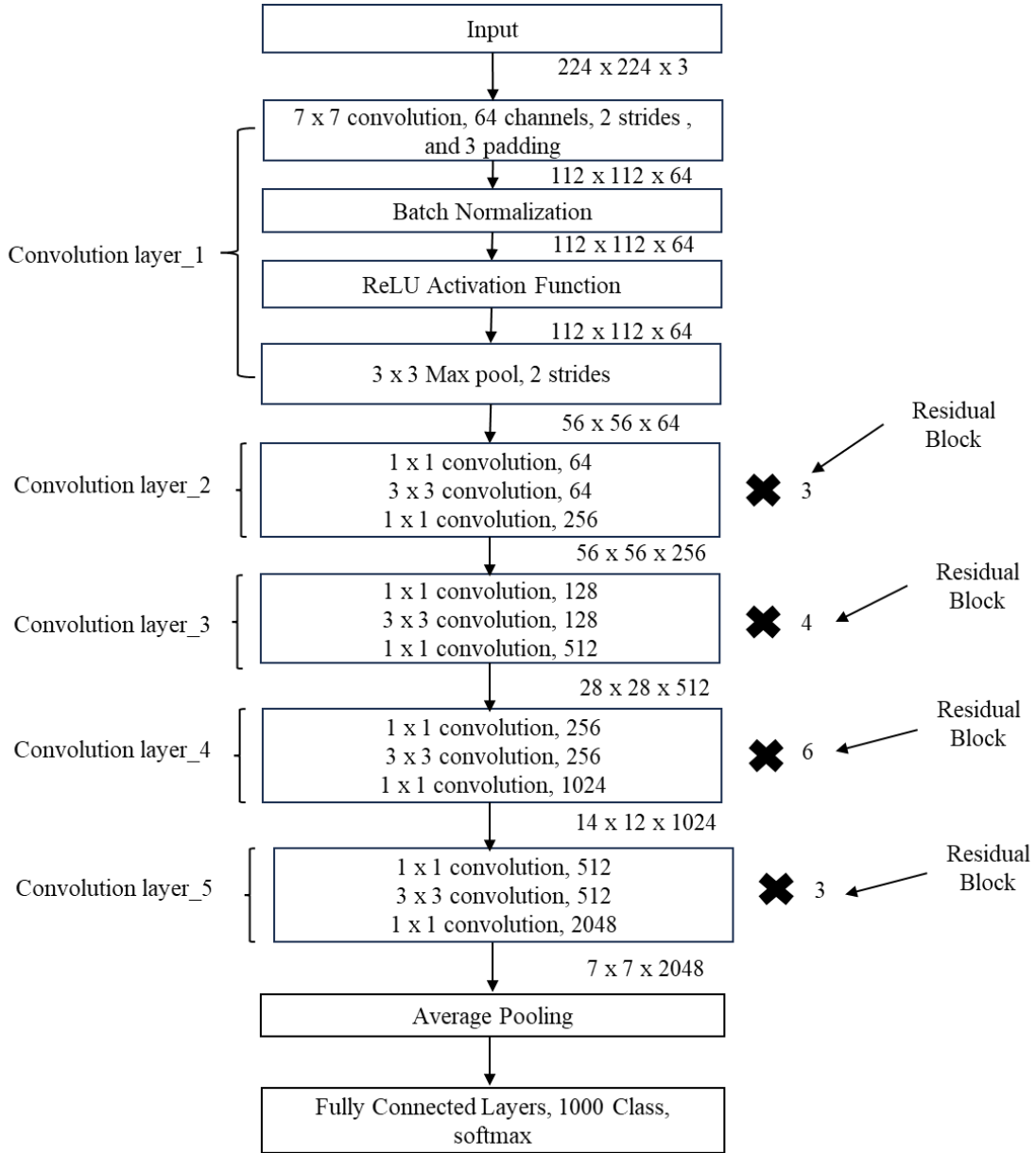


Figure 19. Architecture of original ResNet50

2.3.2 CNN for segmentation

Two CNN architectures are commonly used for image segmentation: a fully convolutional network (FCN) and encoder-decoder structure. The backbone of a fully convolutional network is similar to the common convolutional neural network model where fully connected layers are replaced with convolutional layers to create spatial maps for each class. Secondly, the encoder-decoder method that extracts information from input by sub-sampling or down-sampling in the encoder network, then up-samples the extracted feature in the decoder network to present the output. An example of a UNet model is shown in Figure 20. In a UNet model [45], skip connections are important to concatenate the feature maps in encoders and decoders networks. Skip connections mainly localize the targets.

From [45], it is clear that the UNet framework consists of an encoder-decoder network. The encoder proceeds with one convolution block, then a 2 x 2 max pool (kernel = 2, stride = 2) operation. The convolution block consists of firstly 3x3 convolution, then ReLU, and next is batch normalization, and it is followed by the second 3x3 convolution with ReLU and batch normalization. Here, stride is 1, and padding is “same” which is equal to 0 according to conv2d Pytorch documentation [46]. Through the encoder part, the model extracts feature from the input image and makes the sample shorter, called down-sampled or sub-sampled. Then, the model expands the features just as the input shape in the decoder part, called up-sampled.

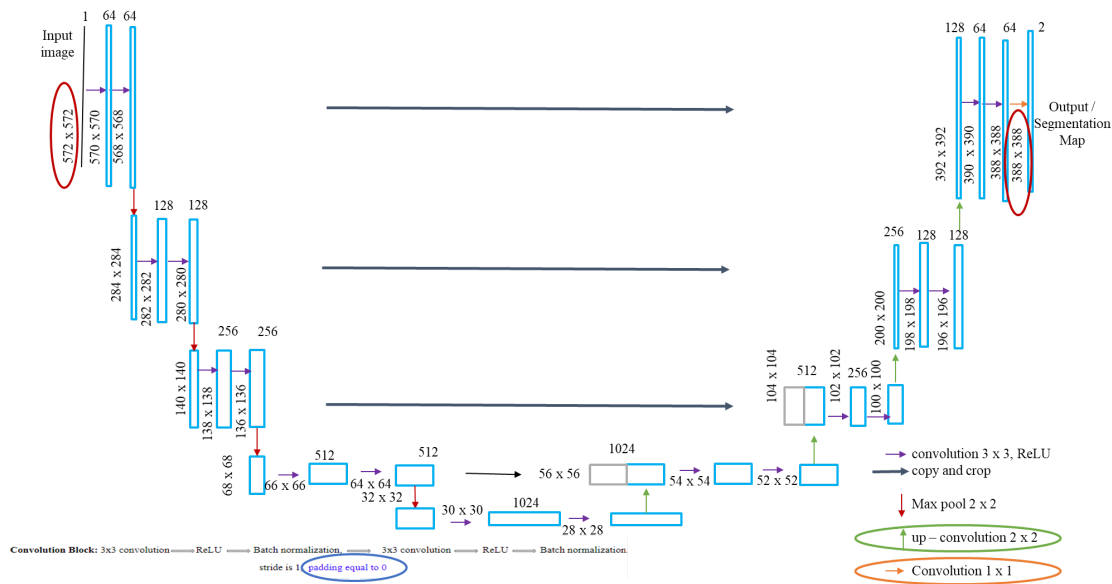


Figure 20. Original UNet model architecture, source: reproduced by following [45]

The size of the output after convolution operation is followed by equations 1 and 2 as shown in Figure 20.

$$W_{output} = \frac{W_{input} - k + 2p}{s} + 1 \dots \dots \dots \text{equation 1}$$

$$H_{output} = \frac{H_{input} - k + 2p}{s} + 1 \dots \dots \dots \text{equation 2}$$

Where

k is kernel,

F is a filter. It helps to extract specific information from images through convolution operation.

s is stride or steps with which the kernels move over the images for convolution operation.

p is padding that creates fake borders around the input image to avoid unprecedented cropping.

W and H are the height and width

k and F are equal when it is a 2d convolution operation. The size of output after the max pooling operation is followed by the equations given in 3 and 4 as presented in Figure 17 (c).

$$W_{output} = \frac{W_{input}-k}{s} + 1 \dots \text{equation 3}$$

$$H_{output} = \frac{H_{input}-k}{s} + 1 \dots \text{equation 4}$$

Figure 20 illustrates a sample encoder-decoder structure of initially proposed UNet model [45]. The input size of the initially proposed UNet model is 572 x 572. This size becomes 570 x 570 and 568 x 568 after the first and second convolution operations, respectively. The output size after convolution is calculated by following equations 1 and 2. After the two consecutive convolution operations, a max pool operation reduced the size from 568 to 284 by following equations 3 and 4.

$$W_{output} = (W_{input} - 1) * s - 2 * p + (k - 1) * d + \text{output padding} + 1 \dots \text{equation 5}$$

$$H_{output} = (H_{input} - 1) * s - 2 * p + (k - 1) * d + \text{output padding} + 1 \dots \text{equation 6}$$

Here, d is *dilation* and its default value is 1.

In the decoder section (Figure 20), each 2D transpose convolution is followed by a convolution block (same convolution block used in the encoder). The output size is calculated by following equation 5 and 6. The final layer takes a 1x1 convolution, which means 1 x 1 kernel operation. The 2D transpose convolution takes a 2 x 2 kernel, where strides are 2, and padding is 'same' or 0 for upsampling. Then the convolution block is applied. The output of this initially developed UNet model crops some portions of the output image because of the 1x1 convolution in the final layer in the decoder network.

2.4 Transfer learning

Transfer learning is a deep learning approach to transfer knowledge from one learning system to others for various tasks as shown in Figure 21. For classification tasks, freezing the input and hidden layers and then training only the final layers are popular approaches to transfer a pre-trained model in different tasks [47].

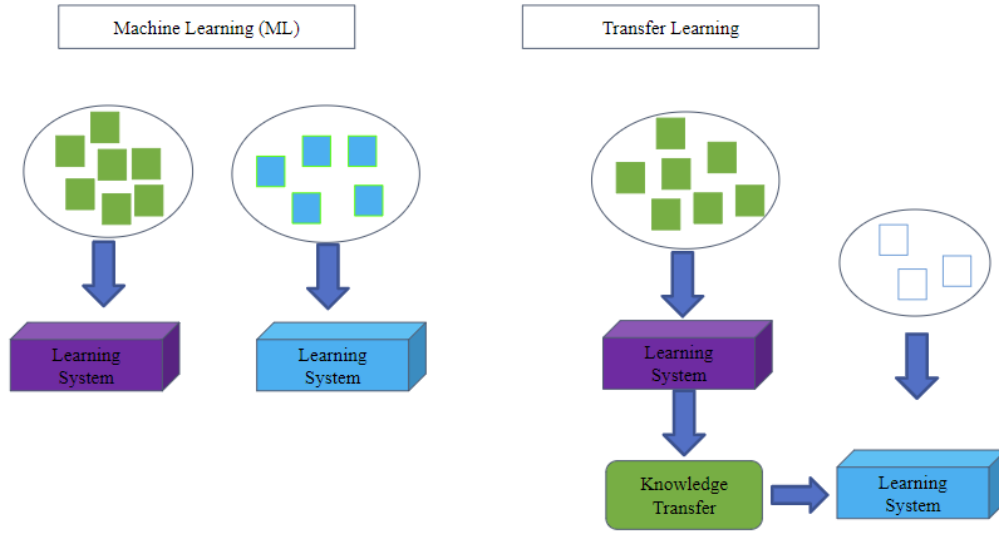


Figure 21. Concept of transfer learning, source: reproduced by following [50]

Table 2 showcases the evolution of CNN architectures from 1989 to 2020. During this time span, ConvNet to EfficientNet CNN backbones have been proposed to adapt with various types of structured and unstructured datasets. This simultaneous rise of different data structures and CNN models have promoted transfer learning. Transfer learning can be applied for multi-class classification in these CNN models for several scenarios: when the dataset is smaller and similar to the previous task, when the dataset is large and similar to the previous dataset, when the dataset is small and different, and when the dataset is large and different than the previous task [48].

Table 2. Evolution of CNN backbones collected from [49]

CNN	Releasing Year
Convnet	1989
LeNet	1998
AlexNet	2012
VGG	2013
GoogleNet	2014
Inception V1, V2, and V3	2014
ResNet	2015
DenseNet	2016
ResnetXt	2017
Channel Boosted CNN	2018
EfficientNet	2019/2020

2.5. Hyperparameters tuning in convolutional neural networks

Hyperparameter optimization plays a crucial role in improving the performance of deep learning models [51]. Different machine learning models need extensive hyperparameter tuning to complete specific tasks [82]. Some hyperparameter optimization methods are GridSearch, random search, gradient-based models, bayesian optimization - Gaussian process (BO-GP), sequential model-based algorithm configuration (SMAC), bayesian optimization - tree-structured parzen estimator (BO-TPE), hyperband, bayesian optimization hyperband (BOHB), genetic algorithm (GA), and particle swarm optimization (PSO) [52], [81]. There are some common hyperparameters for fine-tuning the CNN models available such as optimizer, learning rate, epochs, batch size, mini-batch size, and loss function, as shown in Table 3.

Table 3. Hyperparameters of CNNs

Hyperparameters	Function or Definition	Examples
Activation Function	A nonlinear transformation function, which decides whether a neuron should be active or not.	ReLU
Regularization	Reduces the overfitting issue of CNN models	Dropout (0.5 ~ 0.75)
Optimizer	Controls attributes such as learning rate or weights to reduce errors. “Adam”, “SGD”, “RMSProp” are commonly used optimizers to reduce loss of the training data. Optimizers can assure better training procedures as well.	“Adam”, “SGD”, “RMSProp”
Learning Rate	Ranges from 0 to 1, and is important for training the neurons	0.1, 0.01, 0.001 etc.
Loss Function	Makes a comparison between the goal and predicted results.	Binary Cross Entropy, Cross Entropy
Epochs	One epoch trains the model using the whole dataset in one cycle. A forward and backward pass makes one cycle. The model gets trained in each cycle on the whole dataset using the batch size.	The value varies.
Batch Size	Using this batch size, the model iterates on the whole dataset in one epoch and updates the model parameters at the end of each epoch.	Common batch sizes are 16, 32, 64, 128 or 256.

Mini Batch Size	When the mini-batch is used, the dataset is divided into smaller batches or mini-batches. Then the model keeps updating itself multiple times within one epoch following the average error over a mini-batch.	Divides the dataset into equal smaller batches which are called mini batches.
-----------------	---	---

2.6 Some novel CNN architectures and frameworks for asphalt pavement crack classification and segmentation

This section has explored some novel CNN architectures and frameworks which were developed for assessing pavement cracks particularly by classification and segmentation. While such architectures and frameworks are also proposed for concrete crack evaluations [53], [54], this study has explored CNN-based architectures and frameworks for asphalt pavement cracks because of the predominant use of asphalt pavement in road infrastructures.

Mei et al. [3] proposed a deep learning algorithm, ConnCrack, for segmentation tasks. It has two major parts: generators and discriminators. The generator part included a 121 densely connected neural network (DenseNet121) that consists of a standalone convolutional layer, a max pooling layer, four dense blocks, and three transition blocks. The generator creates target connectivity maps which is the distribution of a targeted pixel with its neighbor pixels. The discriminator part has 5-layer fully convolutional networks (FCN). Purpose of this combined ground truths and connectivity map was to create accurate output labels. The algorithm was trained and tested on publicly available datasets and a proposed dataset Called EdmCrack600. Performance metrics of ConnCrack model on EdmCrack600 dataset were: a precision of 80.88%, a recall of 76.64%, and a F1 score of 76.98%.

In addition, SegNet [55] is a proposed CNN architecture that has 13 convolutional layers encoded with batch normalization, ReLU activation function, and five max-pooling layers. The purpose of SegNet was to reduce the computational intensity of training on complicated image segmentation. This SegNet model has motivated some researchers to use it in crack segmentation for different infrastructures [56].

An example is a novel deep learning architecture, DeepCrack [57], for pavement crack segmentation. The encoder-decoder structure of DeepCrack fused convolution layers in both encoder and decoder parts at the same scale to calculate training losses. The main task of DeepCrack was to conduct pixel-level semantic segmentation of the cracks. The model was trained on the proposed CRACKTREE260 dataset but tested on four different datasets. The average F1 score on four test datasets was 87%, however, the trained DeepCrack model on the CRACKTREE260 dataset failed to detect the cracks that have a bright background.

Li et al. [16] proposed a framework to classify cracks and non-crack images from a dataset of 20,000 images with cracks and another 20,000 images without cracks. For this classification task, the proposed model was named as low-rank group convolution hybrid deep network (ILGCHDN). The classified crack images were segmented or binarized in another deep learning architecture that combined SegNet and dense condition random field (DCRF), which

is, in short, Seg-DCRF. For classification, image blocks were used, and if cracks were in the center of the image blocks, then they were marked as the positive sample and vice versa. Additionally, for segmentation, a crack dataset was prepared based on pixel-wise labeling with the LabelMe tool. For segmentation, 4,800 images were used for training, 1,600 for validation, and 1,600 for testing.

Eslami et al. [2] have proposed a novel deep learning model for an attention-based multi-scale convolution neural network called A+MCNN. This model was used for multi-class classification of both asphalt and concrete cracks. The dataset has four classes of distress: crack, crack seal, patch, and pothole. In addition, it has five non-distress classes, namely joint, marker, manhole cover, curbing, and shoulder; and two pavement classes, namely asphalt and concrete. The F1, precision and recall of A + MCNN framework were calculated, nonetheless, the classification performance was mainly evaluated using the F1 score. The performance of the A + MCNN was compared with both single and multi-scale CNNs. A + MCNN showed 24.2% better F1 score than single scale CNN. Also, it showed 4.9% improvements compared to the multi-scale CNNs.

Moreover, a novel crack segmentation architecture CrackNet [59] consists of five layers: one convolutional layer, one 1x1 convolutional layer, and two fully connected layers. CrackNet is unique in that it does not have max-pooling layers to reduce the size of the output feature map from previous layers. It was trained on 1,800 3D asphalt pavement images to find cracks in different circumstances, and it was tested on 200 images. The model achieved a precision of 90.13%, a recall of 87.63% and an F1 score of 88.86%.

The two important versions of the CrackNet model are CrackNet II [60] and CrackNet-V [61]. CrackNet II has more hidden layers and few parameters in the architecture. It was trained on 2,500 crack images and tested on 200 images, and its precision, recall, and F1 score were 90.20, 89.06, and 89.62%, respectively [61]. On the other hand, 3,083 asphalt pavement images were used for training, validation and testing the performance of a CrackNet-V model, in which 2,568 images were used for training, 15 images for validation, and 500 were for testing. The results showed that CrackNet-V achieved 87.12% for the F1 score, 90.12% for recall, and 84.31% for precision [61].

Another study has developed and evaluated the German Asphalt Pavement Distress (GAPs) dataset, using a CNN architecture called RCDNet. It is a small CNN model inspired by the LeNet model. There are four blocks in RCDNet, namely convolutional layers, max pooling, and fully connected layers. Firstly, it was built with caffe, a deep learning framework, and later Eisenbach et al. [22] trained it using keras based on theano, another deep learning library. However, RCDNet is not an established CNN architecture, therefore, Eisenbach et al. [22] conceptualized an architecture of ASINVOSnet which has eight convolutional layers, three max-pooling layers, and three fully connected layers. The input size for the ASINVOSnet 64 x 64. This ASINVOSnet was trained on GAPs dataset.

Eisenbach et al. [22] again modified the ASINVOSnet in its convolution structures and named the model mod-ASINVOSnet. The performance of these models were compared on several latest regularization techniques such as batch normalization with dropout, only batch normalization, weight decay and maximum normalization methods. The performance of these

two models with the regularization techniques were compared with the CRACK IT and RCDNet models. The modified-ASINOSnet produced the highest F1 score of 89.73% on validation dataset but showed worst performance for completely different test dataset.

2.7 Fine tuning existing CNN models from scratch or transfer learning for pavement crack classification and segmentation

Some researchers have used existing state-of-the-art CNN models by either applying transfer learning or training the models from scratch instead of proposing novel architectures. Figure 22 is a transfer learning framework for pavement crack assessment conducted in [62]. In this framework (Figure 22), the base network was frozen, and the top layers were trained and tested with trial and error on the dataset.

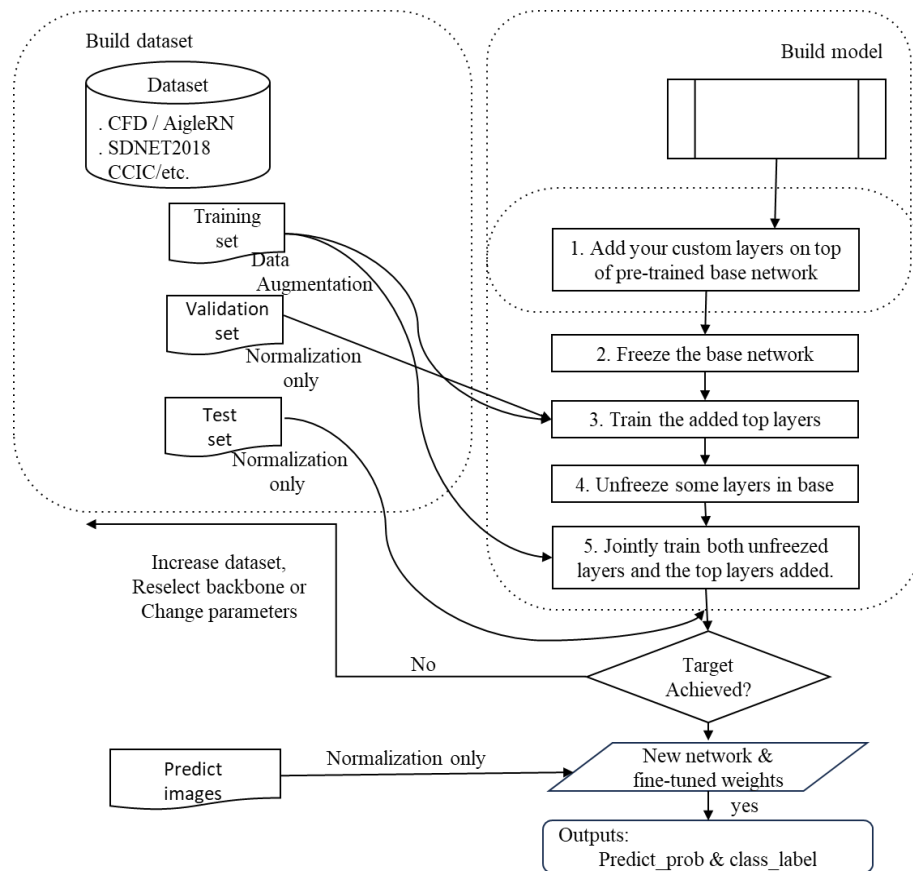


Figure 22. Framework for transfer learning, source: reproduced by following [62]

VGG 16 [9] is primarily trained on ImageNet dataset where there were 1,000 classes available to represent pencils, vehicles, animals, and to name a few. However, using a transfer learning approach, Gopalakrishnan et al. [14] fine-tuned the VGG16 on a dataset which consisted of crack and non-crack images. The goal was to train the model to predict crack and non-crack. VGG16 has 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. Gopalakrishnan et al. [14] only used the final classifier layer of the pre-trained VGG 16 model. A total of 1,056 images were used for training. These are HMA-Surfaced and PCC-surfaced pavement images collected from the FHWA and LTPP database. The

transferred model using the Adam optimizer achieved a 90% accuracy, and the value was consistently 90% for F1, precision, and recall as well.

The original ASINVOSnet used in [22] has eight convolutional layers, three max-pooling layers, and three fully connected layers which is similar to a VGG model with 4.0 million weights. Stricker et al. [15] updated the ASINVOS network with ten convolutional and four max-pooling layers. The updated ASINVOSnet is similar to a VGG model, with 5.9 million weights. The model was trained on the extended GAPs dataset. The input patch size in this transferred ASINVOSnet was updated to 160 x 160 from 64 x 64. Besides this model, ResNet10, ResNet 18, ResNet34, ResNet50 and CrackIT algorithms were evaluated on extended versions of GAPs dataset. The ResNet18 and ResNet34 achieved the best F1 score of 90.62% and 90.41% respectively amongst all of the models.

On the other hand, Hsieh et al. [63] have compared the performance of eight deep learning models for crack segmentation tasks. In the first three models, namely VGG16, VGG19, and ResNet50, the fully connected layers were replaced with Fully Convolutional Network (FCN). The other models were UNet, DeepCrack, CrackNet II, Image Translation Model pix2pix-UNet and pix2pix-ResNet. The conclusions were that skip connections in the UNet model improved crack segmentation by localizing the cracks, and the addition of deeper backbones in the VGG16, VGG19, and ResNet50 showed better performance for crack segmentations.

A study by [4] developed a deep learning system with existing models, namely, SqueezeNet, UNet, and Mobilenet-SSD models, for crack classification, segmentation, and detection. For classification, crack types were classified into alligator crack (AC), longitudinal crack (LC), transverse crack (TC), pothole, and patching. For severity assessment, two UNet models were applied. One was to segment the linear cracks and another was to segment the area cracks. Then the object detection model, mobilenet-SSD, was deployed to locate the crack regions. The system's classification and severity assessment accuracy were 91.2% in total.

Another study [58] proposed a deep learning framework called pavement distress segmentation network (PDSNET). In this study, two existing CNNs, the ResNet34 and UNet model, were studied. This PDSNET was trained on 4,000 pavement distress images for segmentation purposes. The mean intersection over union (mIoU) was 83.7% [58].

Jana et al. [64] compared Googlenet, Alexnet, and ResNet with transfer learning. These models were trained with 136 images that included 50 cracks, 50 non-cracks, and 36 potholes for classification. The dataset was divided into 60% for training, and 40% for testing. The conclusion of the study was that GoogleNet performed better than the other two models because of the higher number of layers and adaptation capabilities.

Ranjbar et al. [65] used pretrained AlexNet, GoogleNet, SqueezeNet, ResNet-18, ResNet 50, ResNet 101, DenseNet-201, and Inception V3 to detect and classify cracks. In their dataset, transverse and longitudinal cracks were marked as linear cracks. Also, the block and fatigue shaped cracks (alligator crack) were marked as surface cracks. Therefore, The dataset had three classes: linear cracking, surface cracking, and non-cracking. Then performance metrics such as accuracy, sensitivity, specificity, precision, and F1-score were calculated to evaluate

these model performances. The general value of accuracy metrics varied from 0.972 to 0.991, the general values of the sensitivity varied from 0.957 to 0.986, similarly values of specificity ranged from 0.962 to 1, the precision score was for non-crack class were higher than other two in the most models, and it's optimal score on linear and surface cracking were found with SqueezeNet at 0.97 to 0.991. Likewise, the F1 score was also higher from 0.989 to 1 for non-cracking. For the other two the SqueezeNet model showed an F1 score of 0.979. The research also developed wavelet transformation for segmentation of the cracks. The recommendation of this research was to use pre-trained DCNN models for pavement crack classification because of their highest level of accuracy.

Hong et al. [66] modified the UNet model by adding a convolutional block attention model to extract more features of cracks. Then the model was trained from scratch and the performance of crack segmentation was better than conventional UNet models. The number of trained images was 1,157 taken with unmanned aerial vehicles (UAVs) and the mean IoU was 77.47%.

Yang et al. [19] has developed two frameworks for pavement crack classification and segmentation (Figure 23). The framework for classification was based on the swine-transfer approach, the dataset included 10,000 images, and the validation dataset had 4,500 images. Finally, 500 images were used for validation and testing. The training images were reduced to 120 x 100 to minimize the computational intensities.

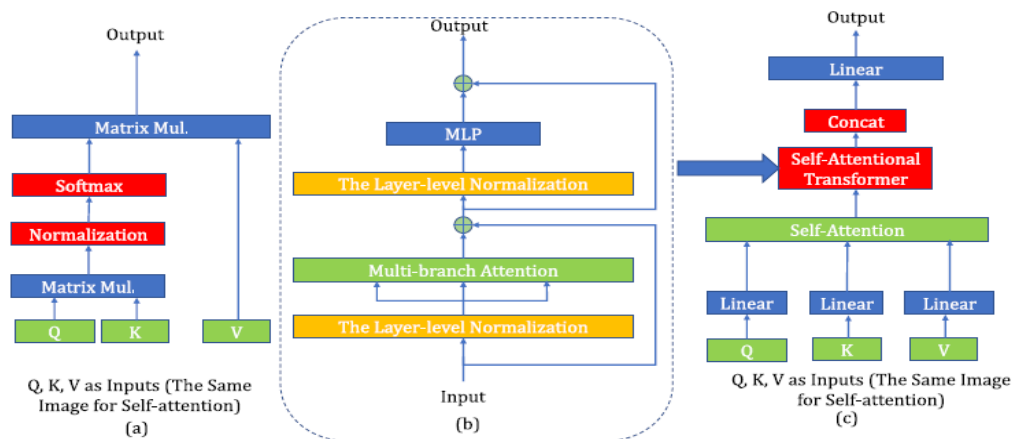


Figure 23. Classification framework, source: [19]

For Segmentation in [19], a multi-stage-pyramid framework was proposed with full resolution residual networks (FRRNs), as shown in Figure 24. A total of 11,000 images in the dataset were used for segmentation and their resolution was 600 x 480. The dataset was splitted into 6,000 for training, 3,000 for validation, and 1,650 for testing. The result indicated that the proposed framework can integrate existing state-of-the-art CNN models for segmentation, for example, integration with DeepLab V3+, DenseNet and Full Resolution ResNet.

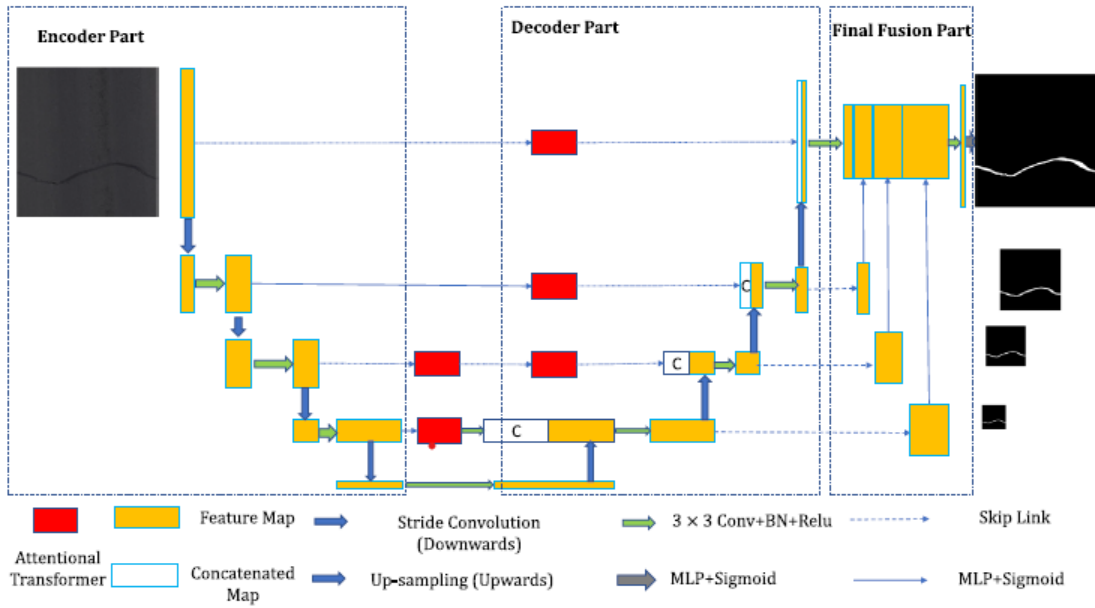


Figure 24. Segmentation framework, source: [19]

Another research [67] applied the ResNet model with transfer learning for classifying the damages of structural infrastructures due to earthquakes. The segmentation was conducted using a UNet model.

Mandal et al [68] trained three deep learning models, YOLO, CenterNet, and EfficientDet models, using 21,041 pavement images included in a big dataset released by IEEE global road detection challenges [69]. This dataset has a collection of road images taken from Japan, the Czech Republic, and India. Amongst the three models, the YOLO model achieved the maximum F1 score of 58%, and 57%, respectively, for prediction and classification of distresses.

2.8 Published pavement distress datasets and annotation methods

Recently, some popular datasets on pavement cracks have been published. For example, CRACK500 [70], EdmCrack600 [3], GAPs [22] and also IEEE on global road detection challenge [69]. A summary of these asphalt pavement crack datasets is included in Table 4.

Table 4. Datasets on pavement distress for different tasks

Asphalt pavement distress datasets	Models	Task	Labeling methods	Accuracy
Asphalt pavement (20,000 Crack and 20,000 non-Crack) [16]	Low-rank group convolution hybrid deep network (ILGCHDN)	Classification	256 x 256 image blocks	Precision 0.9726, Recall 0.9802. and F1 0.9764.
8000 Cracks [16]	Seg-DCRF	Segmentation	Pixel-level Tool: LabelMe	The mIoU of 0.86, 85 and 0.83.
CRACKTREE260 [57]	DeepCrack	Segmentation	Pixel-level	Average 87% F1 on four test datasets.
EdmCrack600 [3]	ConnCrack	Segmentation	Pixel-level	Precision, 80.88%, Recall 76.64%, and F1 score 76.98%.
GAPs version 1 [22]	RCDNet, ASINVOSnet, modified-ASINVOSnet	Classification	Pixel-level	F1 score of 89.73% in modified-ASINOSnet for validation dataset, but showed worse in completely different test dataset.
GAPs version 2 [15]	Original ASINVOSnet, modified-ASINVOSnet ,CRACKIT ResNets 18, 34, and 50	Classification	Pixel-level	ResNet18 and ResNet34 achieved 90.62% (F1), and 90.41% (F1).

Big data released by IEEE [69] with road images from Japan, the Czech Republic, and India.	YOLO, CenterNet, and EfficientDet	Classification and detection	Pixel-level	F1 score of 0.67 on test1 and 0.66 on test2 for YOLO.
4000 pavement distress images for segmentation purposes. [58]	PDSNET	Segmentation	Pixel-level	mIoU 83.7%
CRACK500 [70]	A proposed deep convolutional neural network (ConvNets)	Crack Detection	Pixel-level	F1 score: 0.8965

CHAPTER 3: METHODOLOGY

This chapter describes the procedure for applying the ResNet50 for classifying cracks and the modified-UNet model for segmenting cracks. The AugCrack132 dataset was developed by extracting three principal crack types, alligator, longitudinal and transverse cracks, from the EdmCrack600 dataset to train the transferred-ResNet50 classifier using a transfer learning approach. On the other hand, the segmentation task used the EdmCrack470 and CRACKTREE206 datasets. Figure 25 shows the flowchart of the classification and segmentation procedures. The structure of this chapter is: section 3.1 presents steps involved in the classification task. The model preparation for transfer learning, the transferred-ResNet50 model setup with hyperparameter tuning, shell script writing, and performance evaluation approach are also included in this section. Then section 3.2 describes the complete procedure for the segmentation task, including the model set up approach, hyperparameters, shell scripting for running the model using the UNC Charlotte high-performance computing cluster (HPCC), and the performance evaluation approach based on IoU, precision, recall and F1 score are also included in this section.

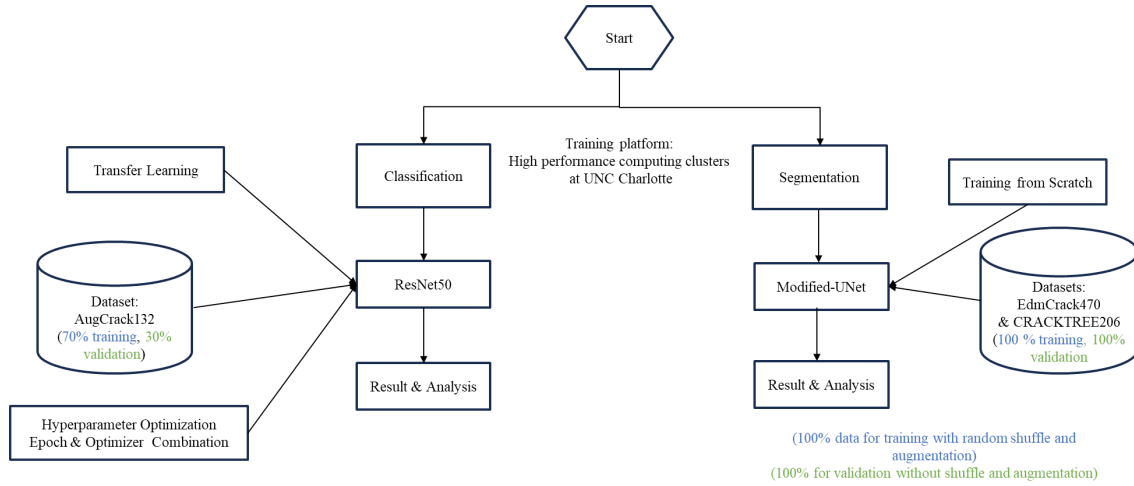


Figure 25. Summary of the methodology

3.1 Classification of crack types with transferred-ResNet50

This study applied transfer learning to ResNet50 for classification tasks. Figure 26 delineates the full procedure of classification task. It includes data preparation, transfer learning, hyperparameter optimization and evaluation steps.

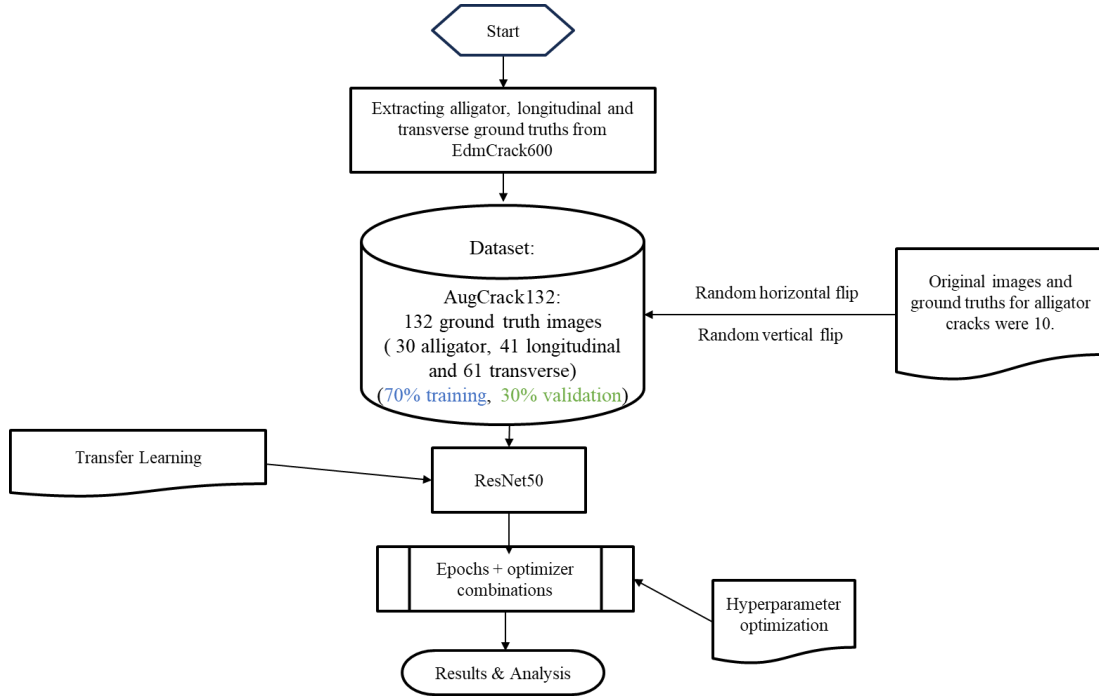


Figure 26. Classification task

3.1.1 Data preparation– extracting three types of cracks from EdmCrack600 dataset

This research extracted alligator, longitudinal and transverse cracks from the EdmCrack600 dataset to create a classification-oriented dataset called AugCrack132. There were two reasons to use the EdmCrack600 as the source dataset for creating the new dataset. The first one was related to its data collection method. The pavement crack images of this dataset were captured with a GoPro camera placing it at the rear side of a vehicle. This approach can be an economic data collection method for PMS. Furthermore, the dataset exhibits an uneven distribution of crack images, mirroring a realistic unbalanced condition for classification.

There were only ten alligator cracks found from the EdmCrack600 dataset. The crack presented in Figure 27 taken from the EdmCrack600 dataset has one longitudinal crack on the wheel path with two parallel cracks. It indicates the alligator crack with moderate severity. This is the reason it has been included as an alligator crack in the newly created classification dataset. Moreover, the crack in Figure 28 taken from the same source has very high severities which is similar to an alligator crack.



Figure 27. Exceptional alligator crack with low severity, source dataset: [3]



Figure 28. Exceptional alligator crack with moderate severity, source dataset: [3]

Most of the cracks found in the source dataset were transverse cracks (more than 300), and the new dataset became unbalanced. As a result, the selected classification model showed biases toward the transverse cracks because of its large quantities. The number of transverse cracks were reduced to mitigate the bias of the model. This study involved the reduction of ground truth of transverse cracks and an evaluation of the model on the updated dataset. A similar trial and error experiment were done for finalizing the number of ground truths of longitudinal cracks.

An augmentation approach using horizontal and vertical random flip was used to increase the ground truths of alligator cracks from 10 to 30. Finally, the new classification-oriented dataset includes 132 ground truths of the cracks of three types: 61 transverse, 41 longitudinal, and 30 alligator cracks. As the model has been trained on a smaller training dataset, this study prepared a testing dataset to evaluate the model performance, particularly, to observe whether the model can differentiate all crack types simultaneously.” The testing dataset includes 5

alligator cracks, 21 longitudinal cracks, and 30 transverse cracks. This new classification dataset has been given the name AugCrack132.

The transferred-ResNet50 model needs a data structure for classification tasks, as shown in Figure 29. The main folder included the subfolders of the crack types for the training dataset. Also, the similar structure was followed for the testing dataset.

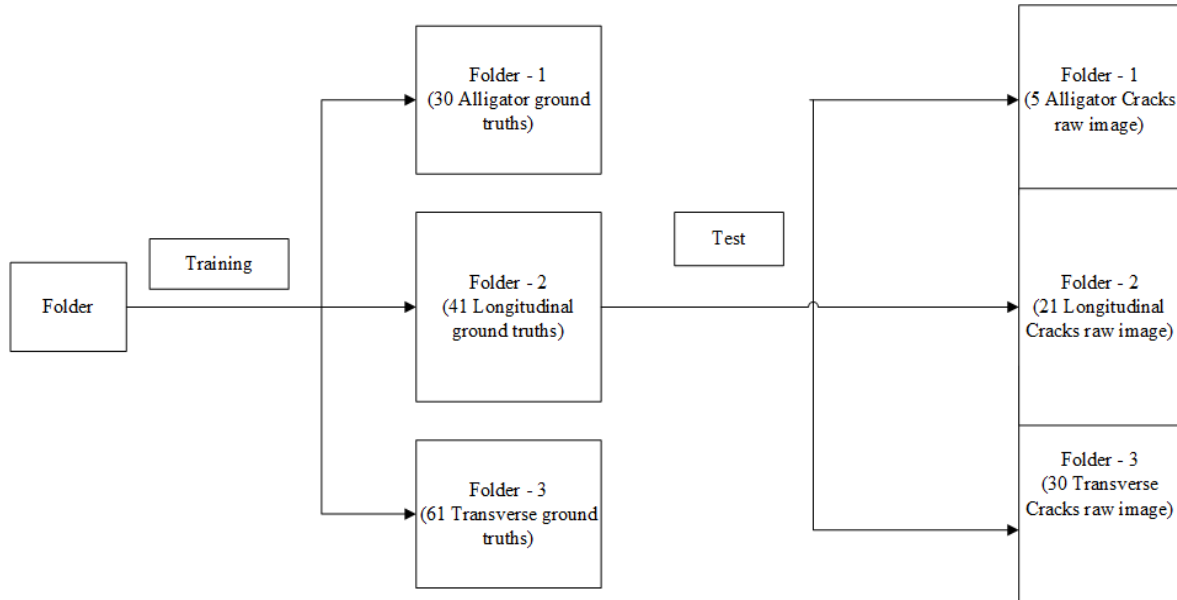
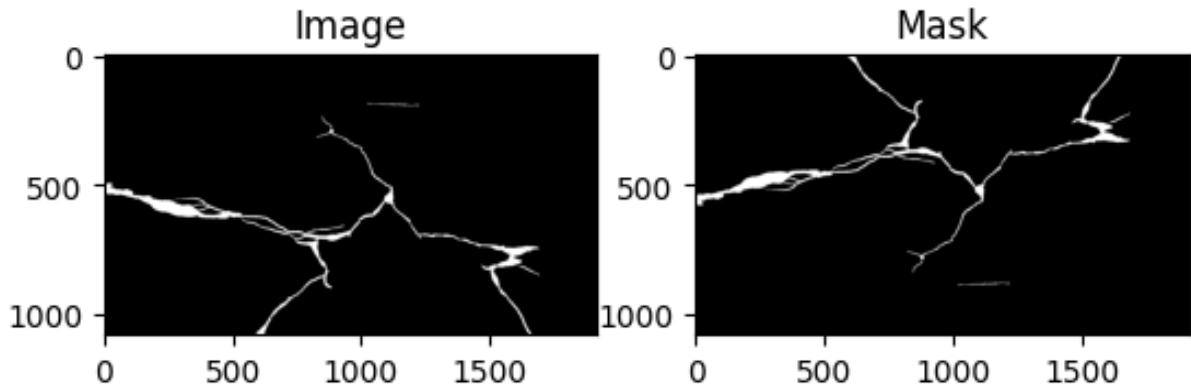


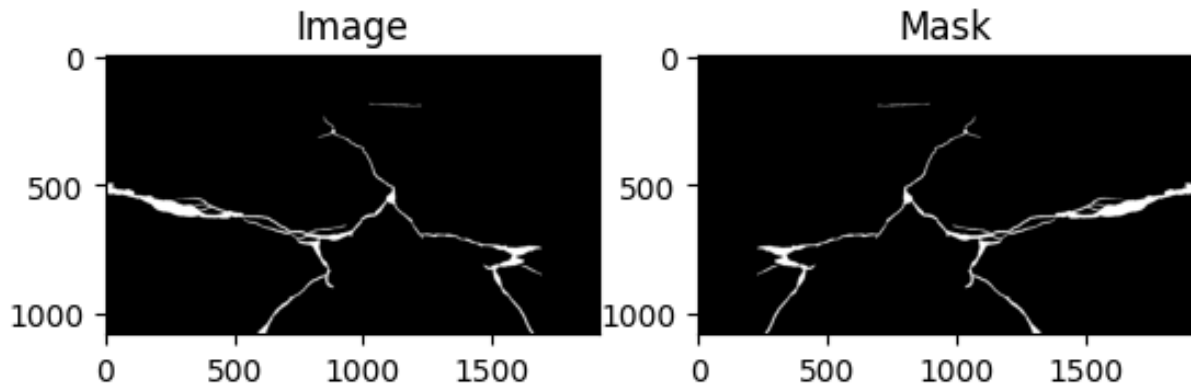
Figure 29. Data structure of the AugCrack132 dataset for classification task

3.1.2. Data preparation - augmentation of alligator cracks

Data augmentation was completed in two steps. At the first step, the random horizontal and vertical flipping were applied to increase the ground truths of alligator cracks from 10 to 30. Figure 30 (a) and Figure 30 (b) are the examples of the vertical and horizontal random flip. The second augmentation operation was conducted with a pipeline in the python programming code of the transferred-ResNet50 to increase the training data as presented in 3.1.3 and Figure 31.



a) Vertical random flip on alligator crack, source: [3]



(b) Horizontal random flip on alligator crack, source: [3]

Figure 30. a) Vertical random flip b) Horizontal random flip to increase the alligator cracks

3.1.3 Data preparation– augmentation in the code for increasing training data.

This study added an augmentation pipeline to the existing python programming code of transferred-ResNet50 to train the model with more data. As a result, the model can be trained on more than 132 ground truths of the cracks. Figure 31 shows a batch size with a combination of augmented and original data. The functions used in the augmentation pipeline are described in the following:

- Resizing to 224 by 224: The images have been resized from 1920 x 1080 to 224 x 224 as the transferred-ResNet50 model requires this input size.
- Five crops (224): It crops the image from four sides and the center to crop one crack from the five different sides, as a result, from one image, there were five different augmented images with 224 x 224 dimensions created during the training time.
- Lambda cropping: This operation is crucial to ensure the five crops are operating properly.

- Random horizontal flip probability 0.5: The reason for using this operation is to flip the cracks horizontally and train the model with various orientations of the cracks.
- Random vertical flip probability 0.5: It creates the vertical orientations of the cracks.
- Random rotation to 0.5 to 5 degrees: Transverse and longitudinal cracks are rotation sensitive. Therefore, it was rotated from 0.5 to 5 degrees to keep their orientations by definition.
- To tensor: It transfers the image pixels to tensors to operate the images properly with the Pytorch packages.
- Normalization: It converts the image pixels from $[0, 255]$ to Pytorch float tensor within the range of $[0,1]$ for mean and standard deviation, respectively.

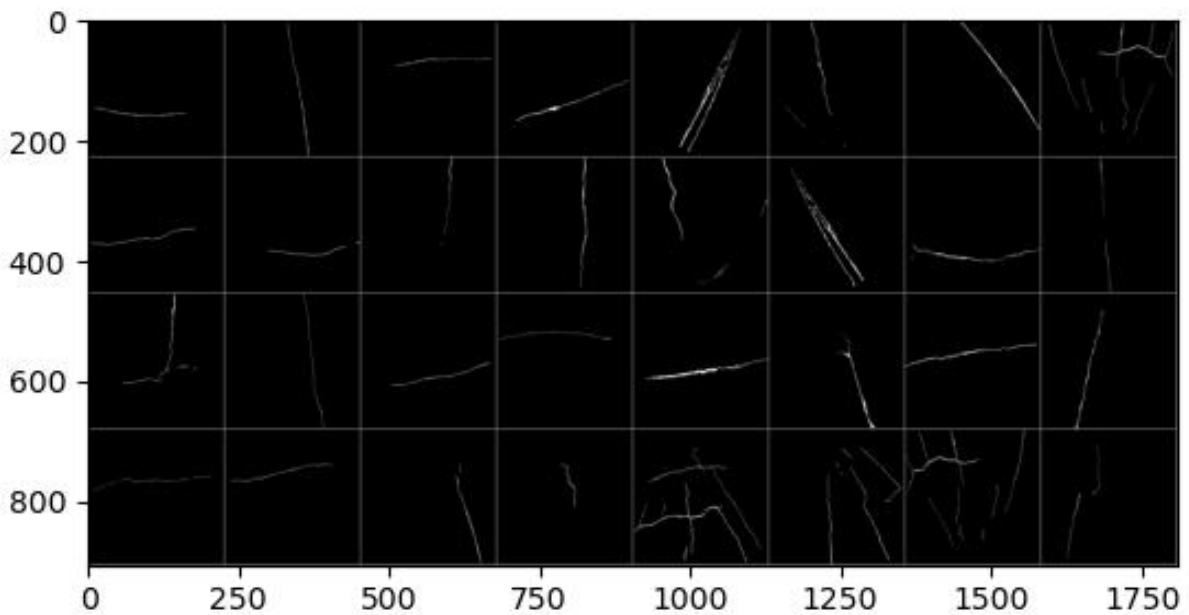


Figure 31. Augmented and original ground truths in a batch

3.1.4 Transfer learning in ResNet50

This study applied transfer learning to the output layer of the ResNet50 model as shown in Figure 32. In the classification layer, a ReLU activation function has been used after the first fully connected layer to activate another classification layer to store the crack information. Model parameters were reduced to 512 from 2048 for the new layer to be compatible with the smaller dataset. Then a softmax function was added to create probability maps for each crack type. The transfer learning was applied in the ResNet50 model by following [71]–[74].

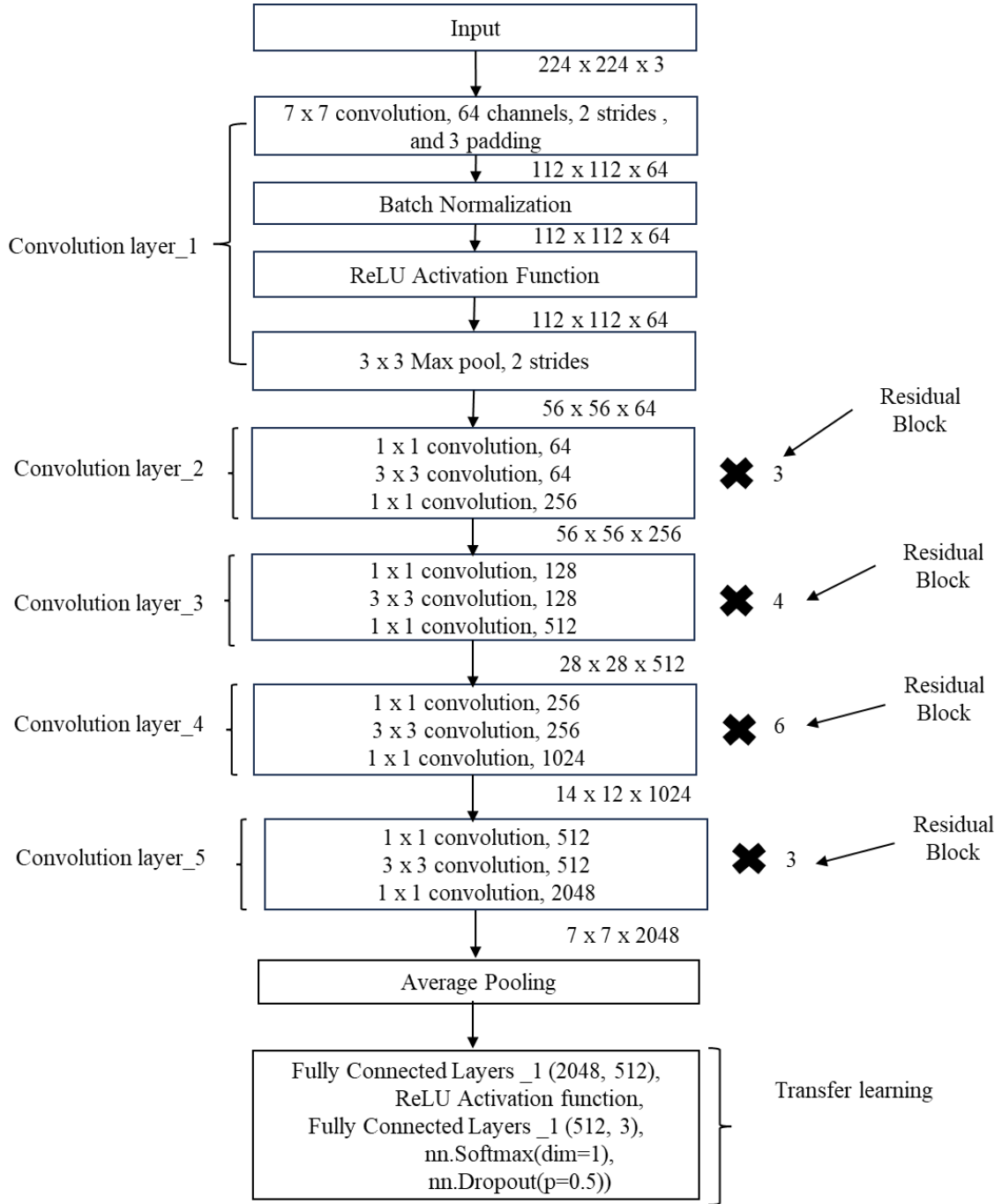


Figure 32. Applying transfer learning to ResNet50

3.1.5 Experimental set up for training the transferred-ResNet50

The learning rate, batch size and loss function for the model was set to 0.001, 32, cross-entropy loss. Two optimizers, adaptive movement estimation (Adam) and stochastic gradient descent (SGD) were used with eight different epochs (10, 20, 30, 40, 50, 100, 500, and 1000) to observe the model performance in a total of sixteen combinations of optimizer

and epoch. During the training phase, the dataset was split into two subdatasets: 70% (92 ground truths) for training and 30% (40 ground truths) for validation data.

3.1.6 Shell script writing for parallel computing

A shell script, a programming language in the linux operating system, was created for running the transferred-ResNet 50 model under the sixteen setup combinations using parallel computing in HPCC. The script was created in a way that it could generate output files for each epoch and optimizer combinations, such as, "output_epoch_optimizers_1000_Adam.txt". The complete shell script is provided in appendix E.

3.1.7 Evaluation method of transferred - ResNet50 model performance

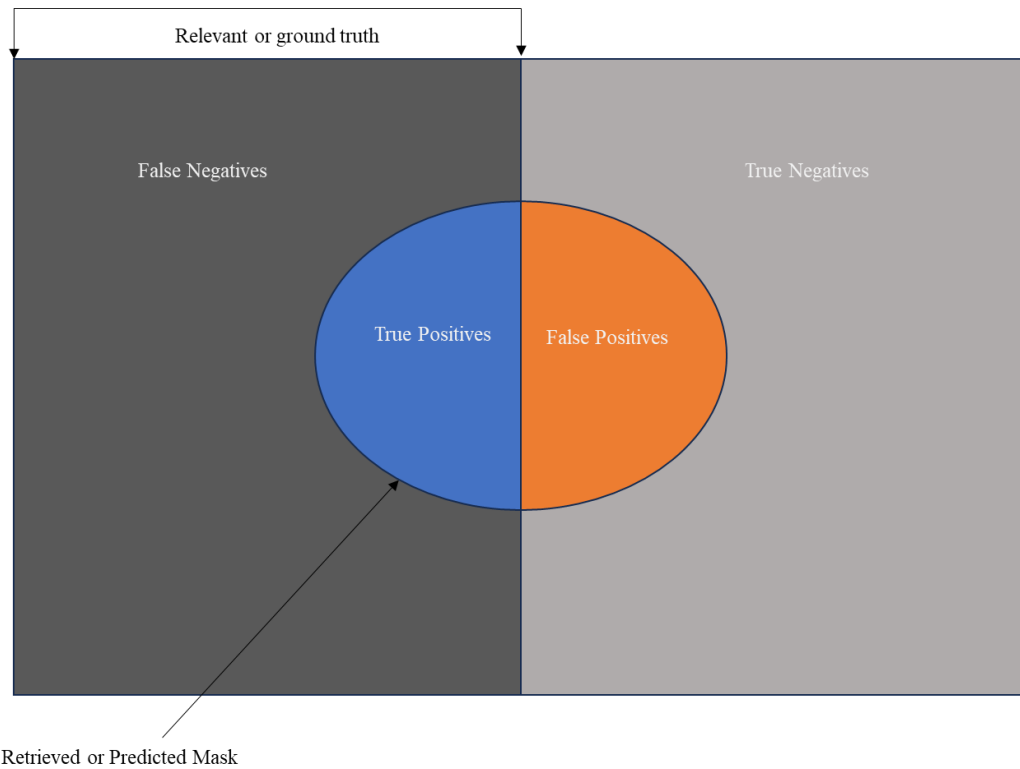
The classification model utilized the test loader function to evaluate test images using the specified batch size. For classification tasks, standard evaluation metrics are accuracy, precision, recall, and F1 score. Equations 7 and 8 are fundamental for calculating accuracy in classification tasks. These two equations were obtained from a Pytorch documentation [83] to show how the accuracy can be calculated.

$$\text{Prediction on overall dataset} = \frac{\text{Correctly Prediction of all classes on the test batch}}{\text{Test batch size}} \dots \text{equation 7}$$

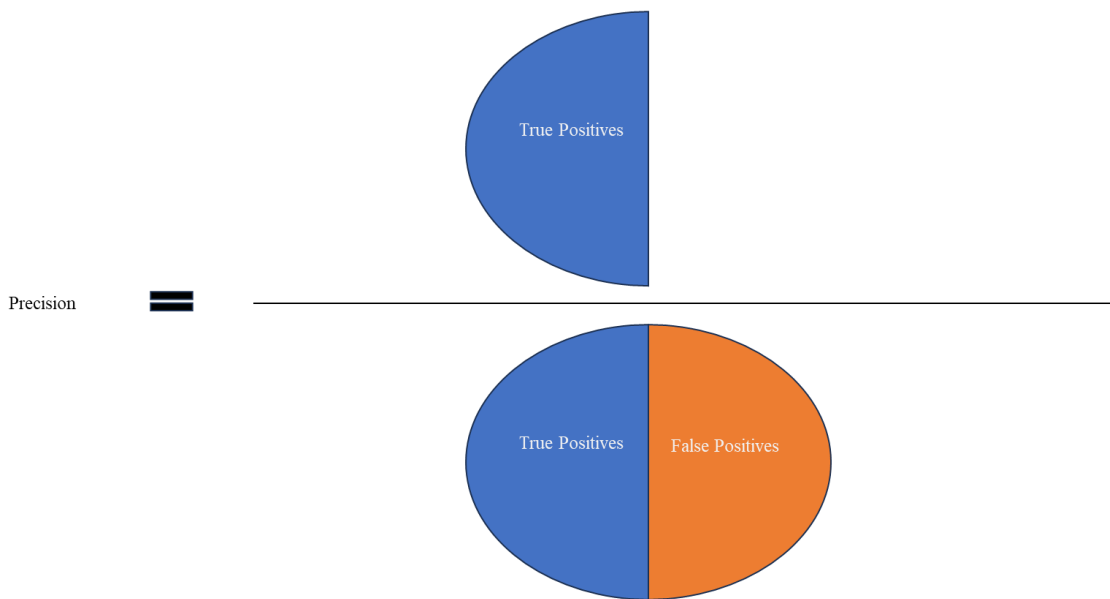
$$\text{For class - wise prediction} = \frac{\text{Correct prediction of the class}}{\text{Total number predicted of the class in the batch}} \dots \text{equation 8.}$$

It is challenging to find the exact number of any crack types in the batch size of the test loader. The main reason is that the test data loader randomly takes images from the three crack types to the batch. For example, x, y, and z are the number of crack images taken in the test batch size where x_1 , y_1 , and z_1 are correctly predicted cracks from each type. Appendix A provides the accuracy calculation.

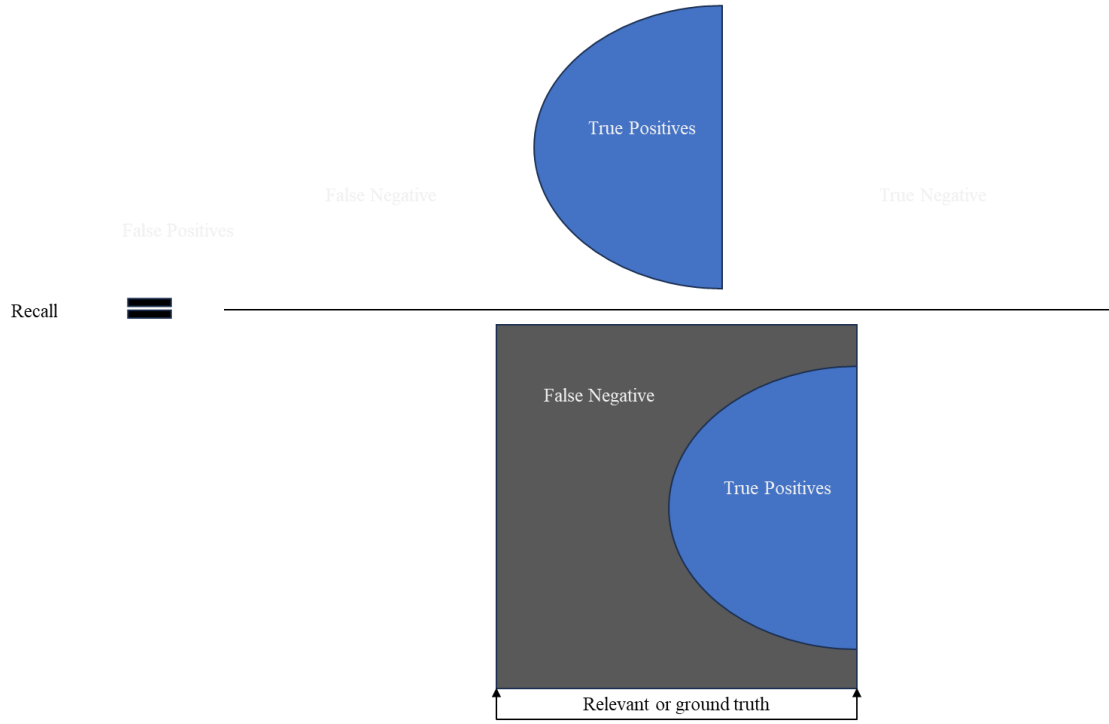
Figure 33 shows the distribution of true positives, true negatives, false positives, and false negatives. Here, the precision indicates the ratio of correctly predicted true positives to the all predicted positive values by the model. The recall means the rate of accurately predicted true positives by the model amongst the all positives in the ground truths or labeled data. Finally, the F1 score is to balance the recall and precision values as the harmonic mean. From the F1 score, average precision and average recall can be understood.



(a) Distribution of true positives, false positives, true negatives, false negatives [77]



(b) Precision



(c) Recall

Figure 33. (a) Distribution of true positives, false positives, true negatives, false negatives, (b) Precision and (c) Recall. Source: reproduced by following [77]

Ground truth is the sum of true positives and false negatives. Predicted masks are the summation of true positives and false positives. Intersection includes true positives from the predicted masks and ground truths. Union imparts true positives, false negatives, false positives, and true negatives.

3.2 Segmentation of the cracks with the modified-UNet

For the segmentation task, this study used a modified-UNet model which was used in segmenting the images of a previous study [75]. Figure 34 represents the full process of applying the modified-UNet model in this study.

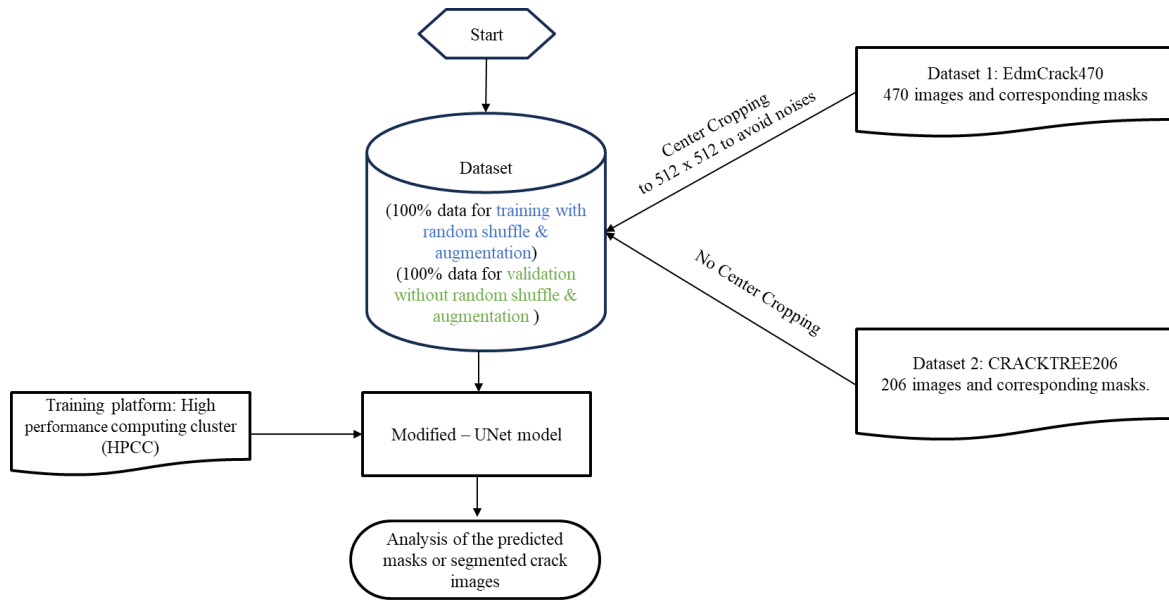


Figure 34. Segmentation using the modified-UNet model for this study

This modified-model is special because of its decoder network. It has 3 x 3 convolution (3 x 3 kernel), 2 strides, 1 padding and output padding in 2D transpose convolution. Also, the convolution block used in both encoder-decoder sections used 1 padding while it was zero in the original model. The different decoder network of the modified-UNet model has encouraged this research to explore it for pavement crack segmentation. Figure 35 shows the architectures of the modified-UNet model.

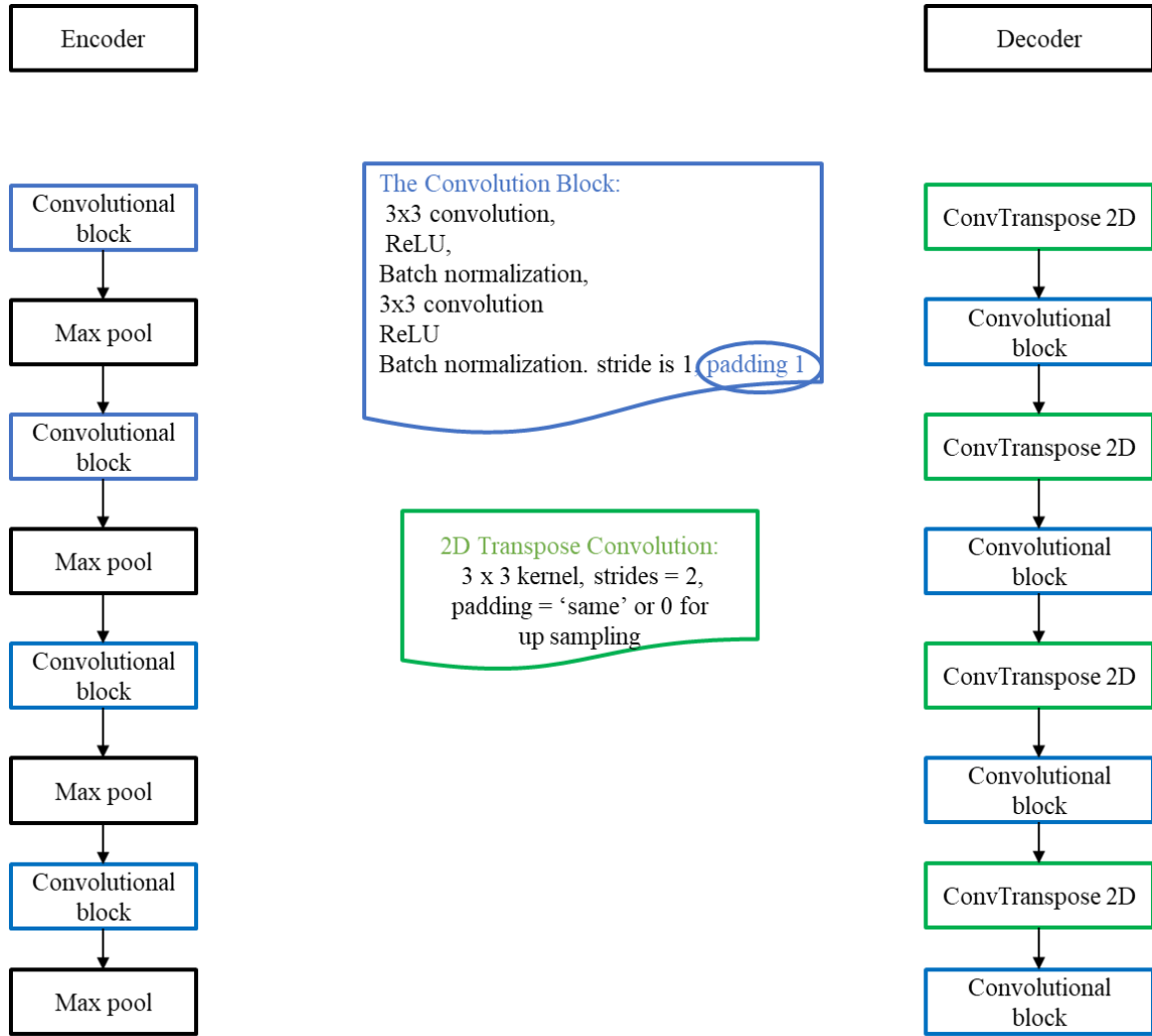


Figure 35. Structure of the modified-UNet model for this research.

3.2.1 Data preparation of EdmCrack470 dataset for segmentation

The raw images in EdmCrack600 have additional information such as grass, noises, tree shades, etc. All of the images were cropped to 512 x 512 from center regions to remove the redundant noises and store sufficient relevant information. The ground truths of the images were also cropped to 512 x 512 from the center regions to align the crack positions with the original images. Figure 36 shows the raw image and ground truths after cropping.

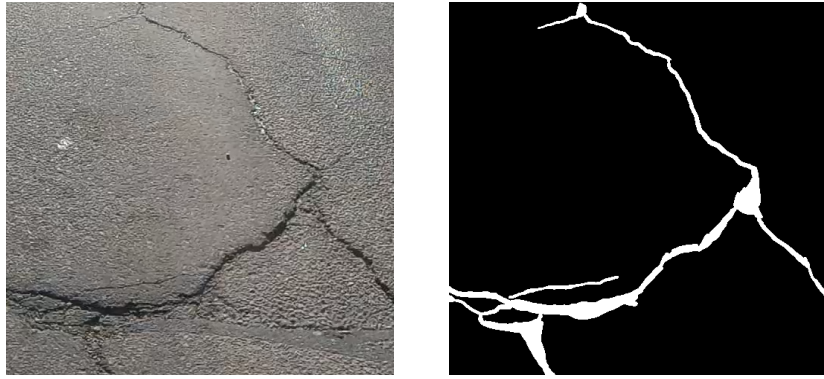
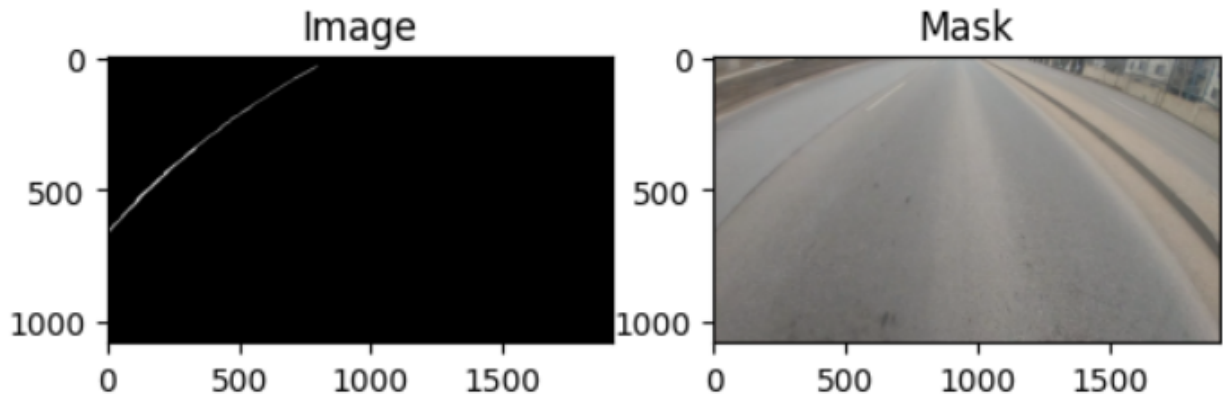
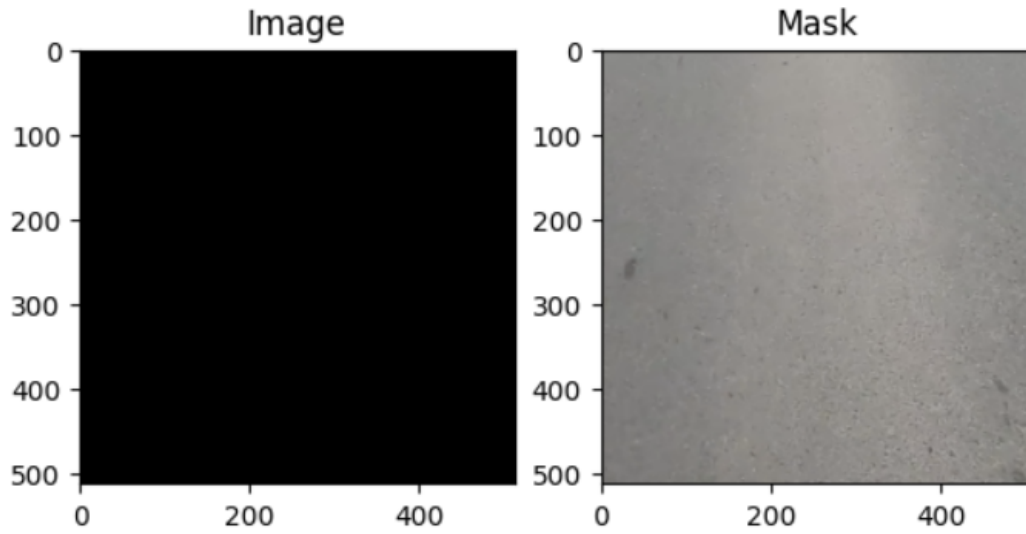


Figure 36. Raw image and corresponding ground truth after center cropping, source: [3]

Some cracks disappeared after cropping from the center regions because of their positions in different areas, for example, Figure 37 (b) shows no cracks as it is the cropped version of Figure 37 (a). Such blank images and corresponding masks were excluded from the newly created EdmCrack470 dataset for this study.



(a). Ground truth and raw cracks images (1080 x 1920) before center cropping, source: [3]



(b). Ground truth and raw crack after cropping (512 x 512) from the center

Figure 37. (a) Original image from [3], (b) The blanked image removed from the EdmCrack470 created in this study.

3.2.2 Data preparation of CRACKTREE206 for segmentation

In CRACKTREE260, 206 images have the same dimension and the other 54 are in different sizes with varied color and textures. Therefore, this study used the same-sized 206 images and corresponding masks for training and testing the modified-UNet model. The original images of this dataset have minimal noises although there are some variations of brightness and shades.

3.2.3 Experimental setup for training the modified-UNet model

A python code was developed for preprocessing the EdmCrack600 and CRACKTREE260 datasets. The required input size for the model was 256 x 256. The augmentation pipeline in the code included resize, random blurring, tensor conversion and normalization functions for training dataset.

However, the testing dataset was preprocessed with resize, tensor conversion and normalization functions. Consequently, the training dataset became different from the testing dataset despite both the training and testing datasets having the same images and corresponding masks. This is why it was also important to observe the model performance on the training dataset for proper evaluation.

The learning rate was set at 0.001 the batch size for the training part was 32; for testing, it was changed to 128 to obtain more predicted or segmented crack images. This study used the same experimental set up as the one described in [75] to be time efficient.

3.2.4 Shell scripting for training the modified-UNet model

The segmentation task is computationally intensive. The shell scripting and completion times for UNet for both datasets are provided in appendix I.

3.2.5 Evaluation method of the modified-UNet model

For segmentation tasks, the necessary evaluation metrics are intersection over union (IoU), precision, recall, and F1 score. The values of evaluation metrics were calculated on the entire test batch size rather than evaluating individual images. Testing the individual images would require additional programming configuration, trials, and time. The values of the performance metrics are sensitive to the contrast adjustment of the predicted masks to compare their alignment with the ground truth. Figure 38 shows the similarities of the ground truths and the predicted masks using true positives, false negatives, and false positives to describe the intersection over union.

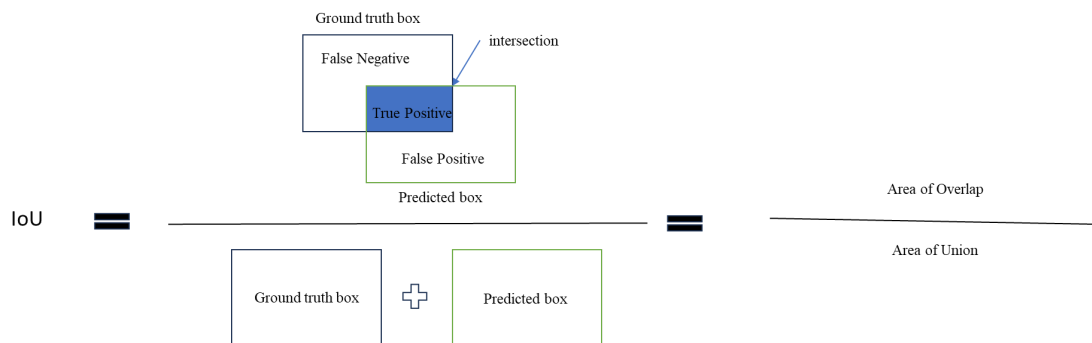


Figure 38. Intersection over Union formula, source: reproduced by following [76]

CHAPTER 4: RESULTS

This section presents the results obtained from the classification and segmentation task. Outputs related to these tasks are described in section 4.1 and 4.2, respectively. The optimal outputs are documented. Other outputs are stored in the GitHub and google drive for further references.

4. 1 Classification of pavement cracks using transferred-ResNet50

Table 5 shows the performance of the transferred-ResNet50 model in each combination of epochs and optimizers. The total training time was 4 hours 21 minutes and 49 seconds in HPCC. From this comparison table, the optimal output combination can be determined.

Table 5. Performance of transferred-ResNet50 for classification of cracks classes

Epoch	Optimizer	On overall dataset	On alligator cracks	On longitudinal cracks	On transverse cracks	F1, Precision and Recall
10	Adam	53.0%	0.0%	9.5%	93.3%	F1 = 43.0% Precision = 48.0% Recall = 54.0%
	SGD	53.0%	0.0%	4.8%	96.7%	F1 = 40.0% Precision = 48.0% Recall = 54.0%
20	Adam	46.0%	0.0%	57.1%	46.7%	F1 = 45.0% Precision = 45.0% Recall = 46.0%
	SGD	53.0%	0.0%	0.0%	100%	F1 = 37.0% Precision = 29.0% Recall = 54.0%
30	Adam	53.0%	0.0%	0.0%	100.0%	F1 = 37.0% Precision = 29.0% Recall = 54.0%
	SGD	53.0%	0.0%	0.0%	100.0%	F1 = 39.0% Precision = 30.0% Recall = 54.0%

40	Adam	21.0%	80.0%	14.3%	16.7%	F1 = 25.0% Precision = 62.0% Recall = 21.0%
	SGD	8.0%	100.0%	0.0%	0.0%	F1 = 1.463% Precision < 1.0% Recall = 8.9%
50	Adam	53.0%	0.0%	57.1%	60%	F1 = 51.0% Precision = 50.0% Recall = 54.0%
	SGD	53.0%	0.0%	0.0%	100%	F1 = 37.0% Precision = 29.0% Recall = 54.0%
100	Adam	57.0%	0.0%	14.3%	96.7%	F1 = 47.0% Precision = 48.0% Recall = 57.0%
	SGD	53.0%	0.0%	0.0%	100%	F1 = 37.0% Precision = 29.0% Recall = 54.0%
500	Adam	58.0%	40.0%	81.0%	46.7%	F1 = 59.0% Precision = 68.0% Recall = 59.0%
	SGD	53.0%	0.0%	0.0%	100%	F1 = 37.0% Precision = 29.0% Recall = 54.0%
1000	Adam	33.0%	40.0%	81.0%	0.0%	F1 = 23.0% Precision = 17.0% Recall = 34.0%
	SGD	58%	0.0%	23.8%	93.3%	F1 = 52.0% Precision = 51.0% Recall = 59.0%

The optimal correct prediction on the total dataset was 58.0% for both ‘Adam’ and ‘SGD’ optimizers at epoch 500 and 1000, respectively. The model was able to classify three types of cracks simultaneously at epochs 40, and 500 using the “Adam” optimizer, while it failed to

classify alligator cracks correctly in any epoch with “SGD”. The possible reason can be cited from [80] is that ‘Adam’ can converge the model parameters faster than ‘SGD’ because of its low directional sharpness, a performance indicator of optimizer algorithms.

On the other hand, the model could accurately classify the longitudinal cracks at the maximum of 81.0% using "Adam" at epochs 500 and 1000. With the ‘Adam’ optimizer the transferred-ResNet50 had also correctly predicted the longitudinal cracks at 57.1% at epochs 20 and 50. The accurate prediction of this crack type by the model with ‘SGD’ was a maximum of 23.8% at epoch 1000. Finally, the model could find the transverse crack types with 100.0% accuracy for several combinations while it happened only once for 'Adam' at epoch 30 and several times for 'SGD' at epochs 20, 30, 50, 100, and 500.

According to [80], the ‘Adam’ optimizer algorithm can converge the model parameters faster than the "SGD" for classification tasks which has been evident here. The unbalanced AugCrack132 dataset had only 5 alligator cracks for testing. Also the number of longitudinal cracks were less than the transverse cracks. In spite of unbalanced conditions in the dataset, the ‘Adam’ optimizer could converge the model parameters during the backpropagation process. Hence, the model could identify all three types of cracks concurrently with some accuracies in two combinations (epoch 40 and 500 with ‘Adam’ optimizers). In contrast, the ‘SGD’ optimizer led the model to outperform on transverse crack only because of its larger quantity over the two types.

Figure 39 is the loss curve with the ‘Adam’ optimizer at epoch 500. This time the model could correctly classify three types of cracks at the same time on the unbalanced dataset. The training loss reduced to less than 60.0% and accordingly validation loss showed the almost similar trend and ended at the close margin of 60.0%. It indicates a slightly under-fitting, although the difference is very small and can be considered as a balanced training.

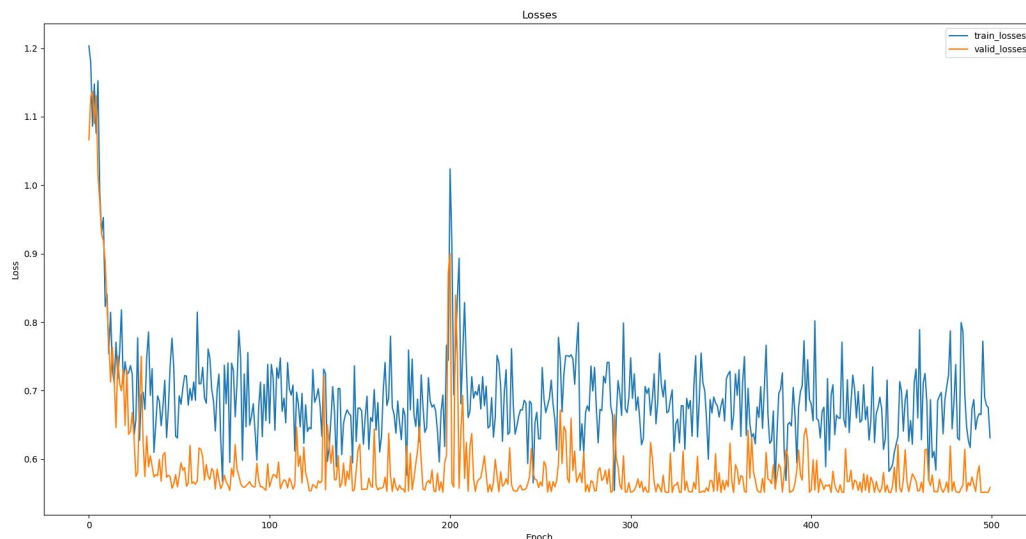


Figure 39. Loss curve at epoch 500 and optimizer “Adam”

On the other hand, the loss curve in Figure 40 shows ‘SGD’ at epoch 1000. It shows that the validation and loss curve reduced gradually and merged at the 70.0% loss. This is an

underfitting sign because the loss curves needed to reduce the loss curve to a reasonable percentage.

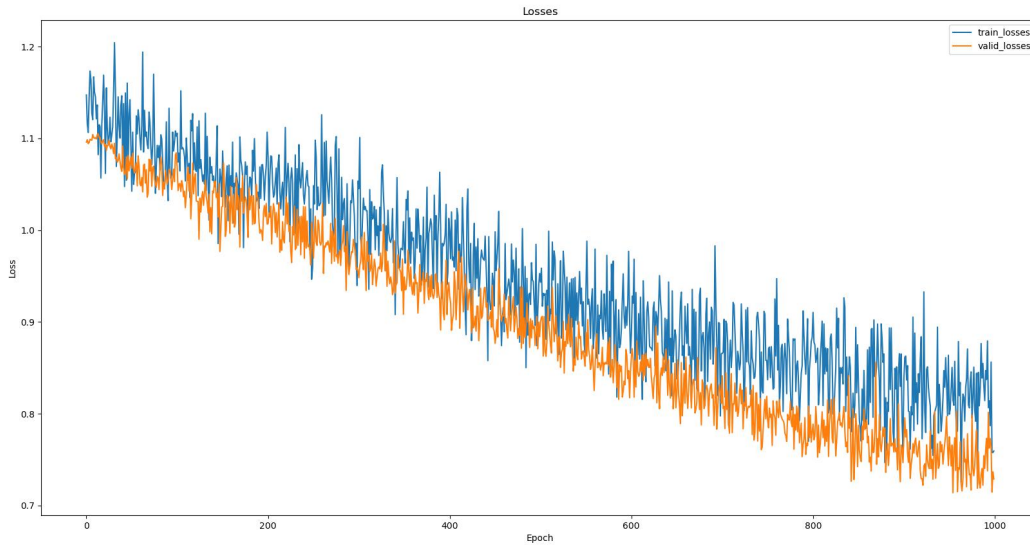


Figure 40. Loss curve at epoch 1000 and optimizer “SGD.”

4. 2 Segmentation performance of the modified UNet Model

The UNet model was trained and tested on the EdmCrack470 and CRACKTREE206 datasets. For both datasets, because of data scarcities. The model took 3 hours, 7 minutes, and 15 seconds on the 470 images and corresponding masks of the EdmCrack470 dataset on HPCC to complete the training and testing process. On the contrary, the procedure took 1 hour, 22 minutes, and 29 seconds for 206 images and corresponding masks of the CRACKTREE206 dataset. The performance of the UNet model on the both datasets is described in subsections 4.2.1 and 4.2.2, respectively.

4.2.1 UNet model performance for segmenting EdmCrack600 Dataset

The evaluation of UNet model performance is described in three subsections. The 4.2.1.1 is the analysis of the loss curve, the 4.2.1.2 presents evaluation metrics and 4.2.1.3 is for evaluating the predicted masks by the model on three criteria.

4.2.1.1 Analysis of the loss curve of the UNet model on EdmCrack470 dataset

The loss curve in Figure 41 shows that both train and validation losses reduced to less than 10.0%. The two curves show a continuous descending trend, which indicates that the model can continue to learn and further improvements in its performance is possible. However, the training process was stopped due to time constraints.

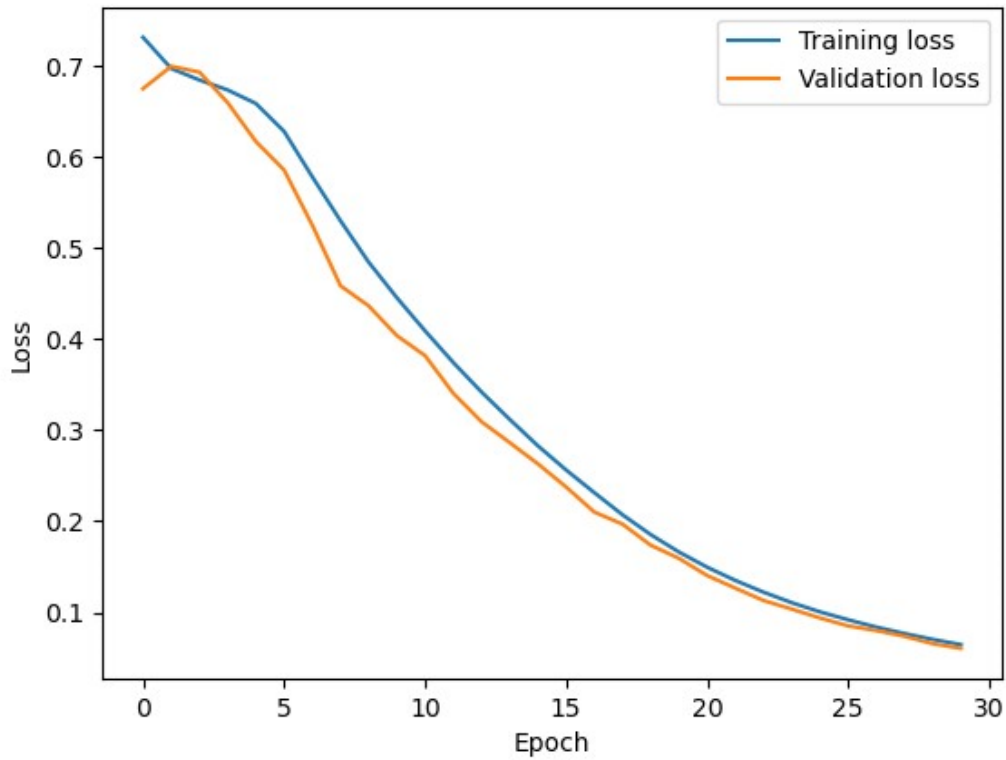


Figure 41. Loss curve for UNet Model on EdmCrack600 dataset

4.2.1.2 Evaluation metric elements of the UNet model on EdmCrack470 dataset

Table 6 represents the evaluation metrics of the UNet model for EdmCrack470 dataset. The IoU is 67.0%, which means that the ground truth and predicted masks overlap with 67.0% pixels. Also, the model achieved the precision score of 96.0%, which means it can predict 96% true positive pixels correctly from the all positives of the predicted masks. The recall is 65.0%, which indicates that the model can identify and predict 65.0% of the true positive pixels accurately from the ground truth. Finally, the F1 score is 78.0%, which means that the average of precision and recall is 78.0%.

Table 6. Performance of the modified-UNet model on EdmCrack470 dataset.

IoU	Precision	Recall	F1
67.0%	96.0%	65.0%	78.0%

4.2.1.3 Evaluation of the predicted masks on EdmCrack470 dataset

This section describes the evaluation of the predicted masks based on three criteria: comparing the extracted crack shapes to ground truths, identifying model bias toward predicting background information, and observing the impact of noise (shades, brightness, spots) on crack shape prediction from original images.

In Figures 42, 43 and 44, the model was able to segment almost the full shape of the cracks from the raw images, and the predicted masks were similar to ground truths. However, the predicted masks produced some spots from the noises of the original images. Such spots are more visible in Figures 45, 46 and 47, where the cracks are scattered but clearly their shapes are understandable.

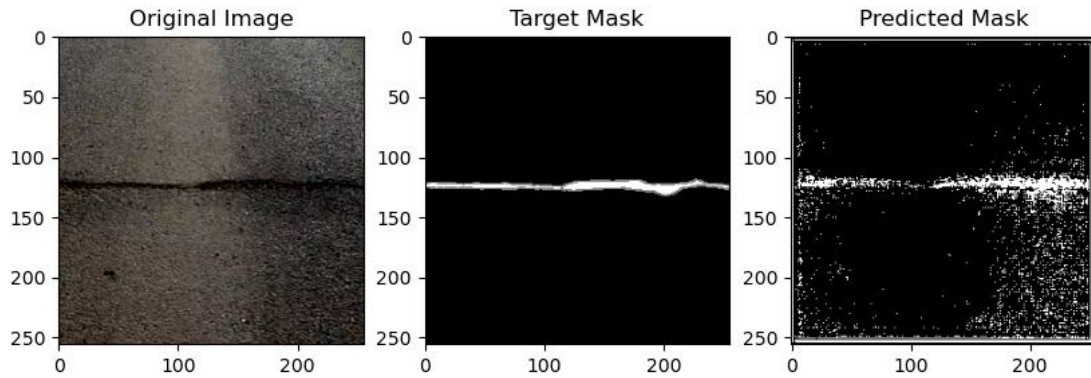


Figure 42. Predicted transverse crack from EdmCrack600 (a)

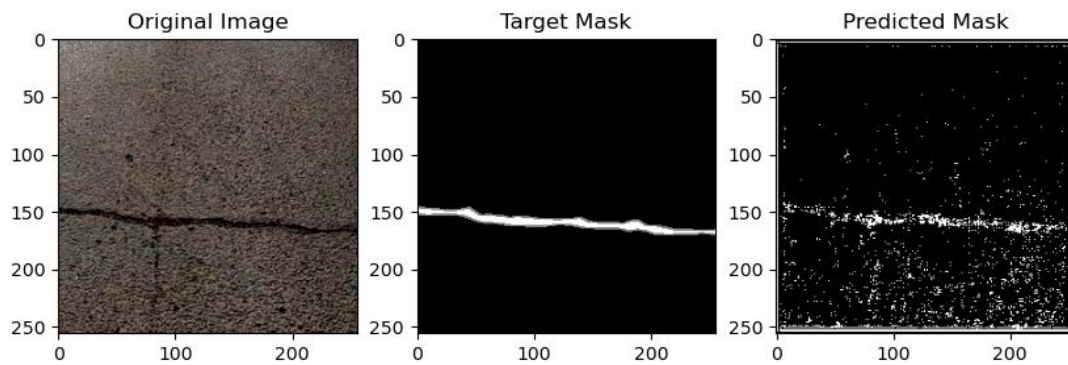


Figure 43. Predicted transverse crack from EdmCrack600 (b)

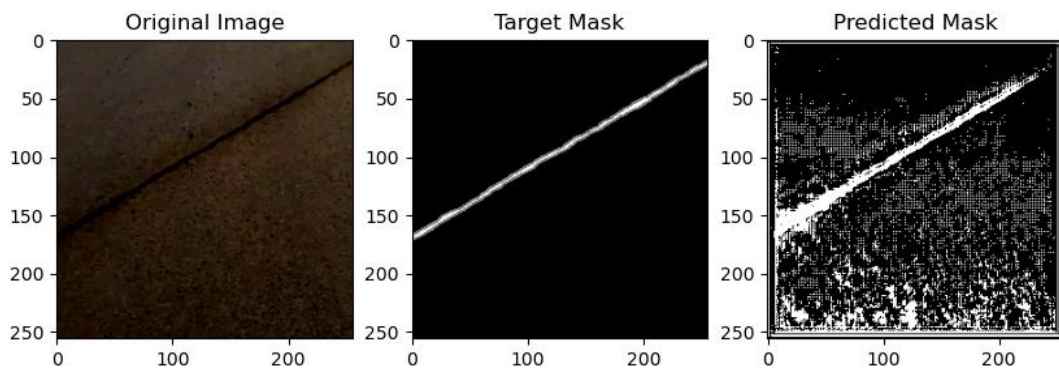


Figure 44. Predicted transverse crack from EdmCrack600 (c)

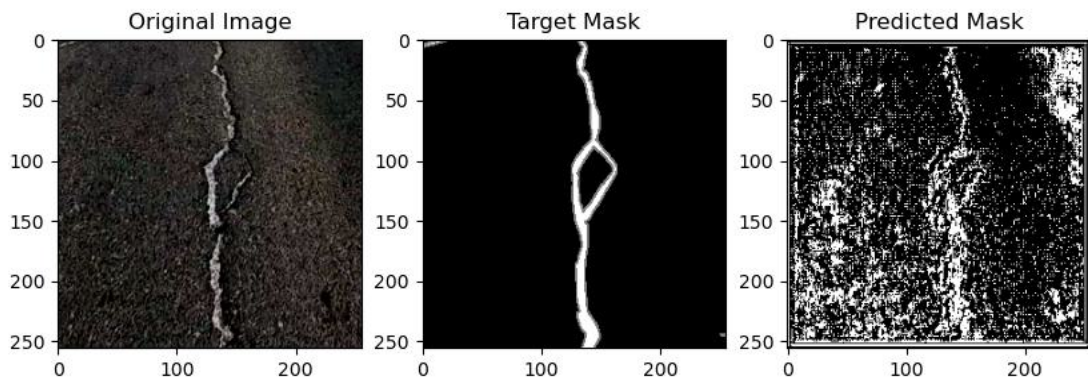


Figure 45. Scattered shape of the predicted cracks (a)

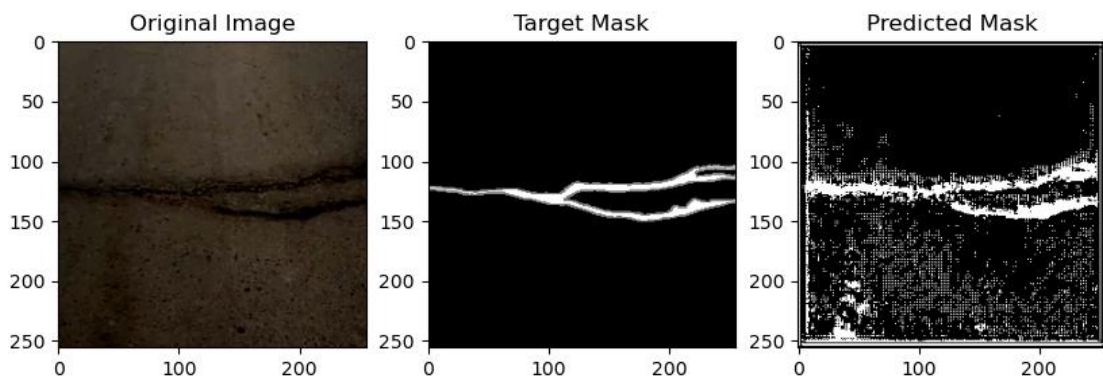


Figure 46. Scattered shape of the predicted cracks (b)

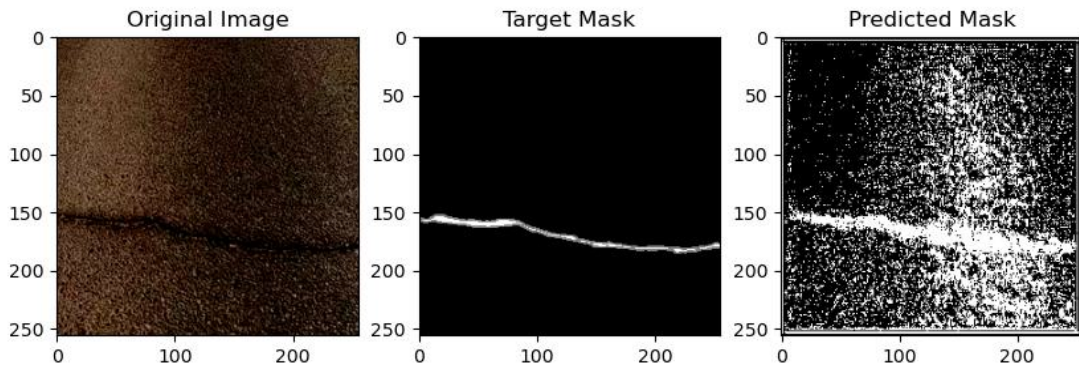


Figure 47. Scattered shape of the predicted cracks (c)

Figures 48 and 49 delineate the segmented crack shapes with distortions and visible spots. These predicted masks are similar to the ground truths but less effective for severity assessment of cracks.

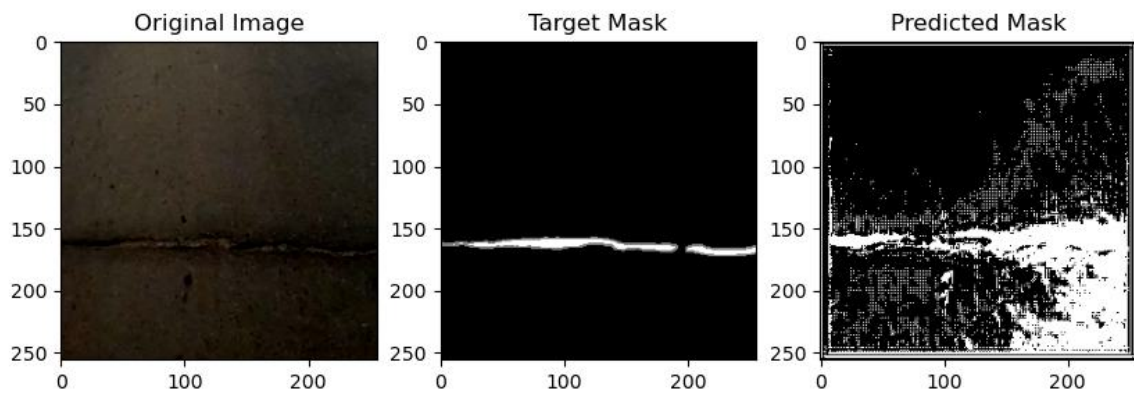


Figure 48. Predicted crack region has been scattered (a)

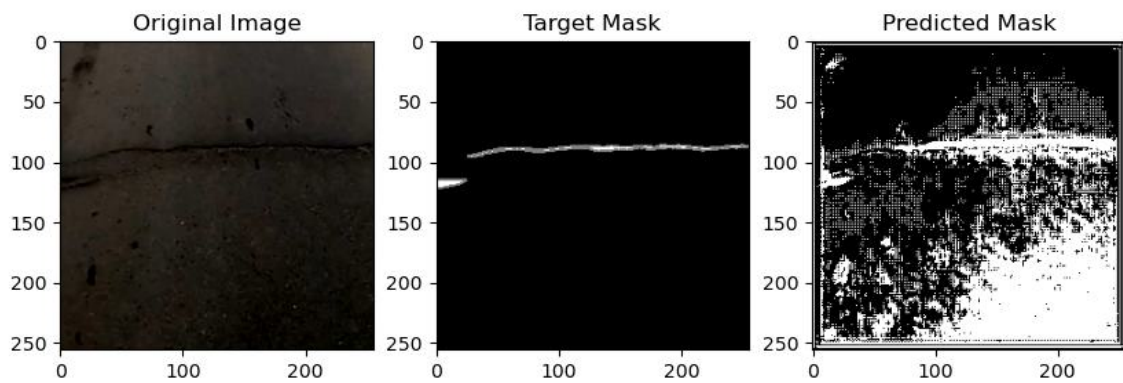


Figure 49. Predicted crack region has been scattered (b)

The UNet model missed the crack regions and only predicted the backgrounds in Figures 50 and 51.

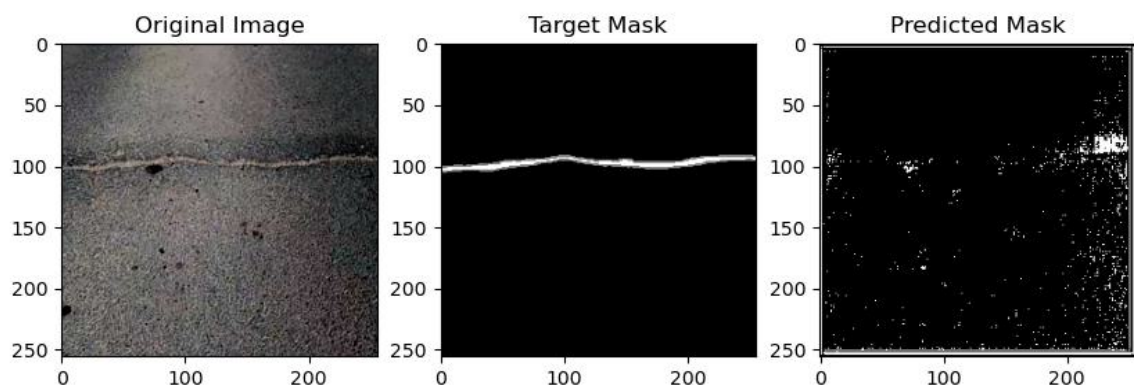


Figure 50. Predicting the backgrounds only rather than predicting crack regions (a)

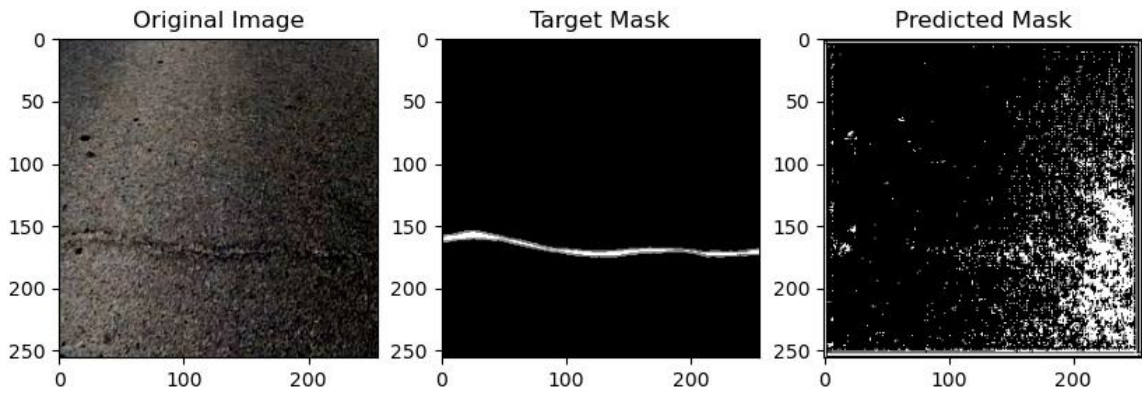


Figure 51. Predicting the backgrounds only rather than predicting crack regions (b).

Variations of brightness, shades, and spots on the raw images had affected the segmented images. Figures 52 and 53 depict that the brighter and shaded regions of the original images were assumed as cracks and backgrounds respectively in the segmented mask.

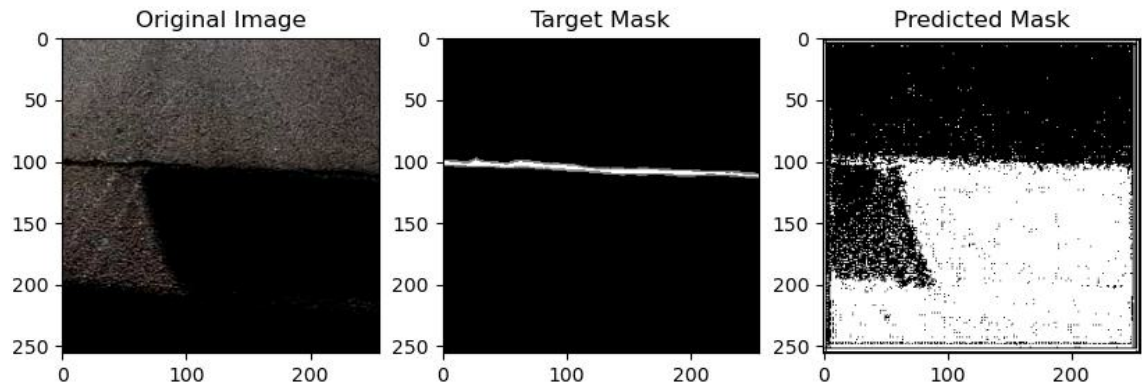


Figure 52. The shades as cracks and brighter zones as background (a)

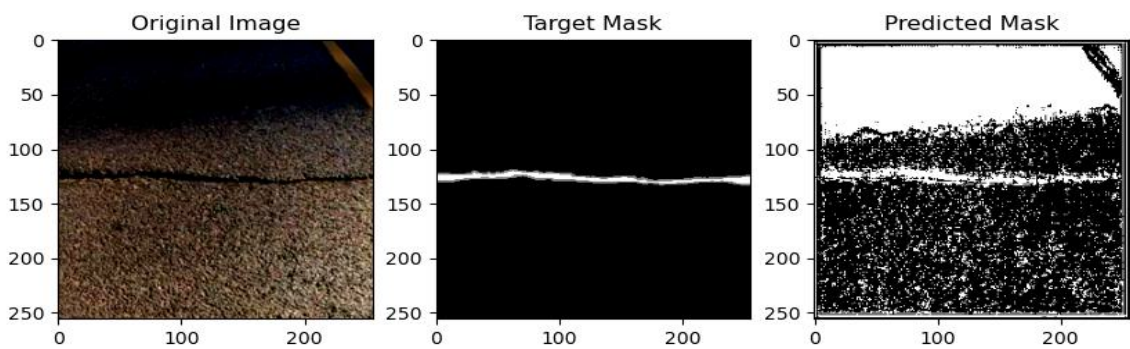


Figure 53. The shades as cracks and brighter zones as background (b)

Figures 54 and 55 are also impacted by the shaded and brighter regions, but the model predominantly segmented the background information and missed the original crack regions.

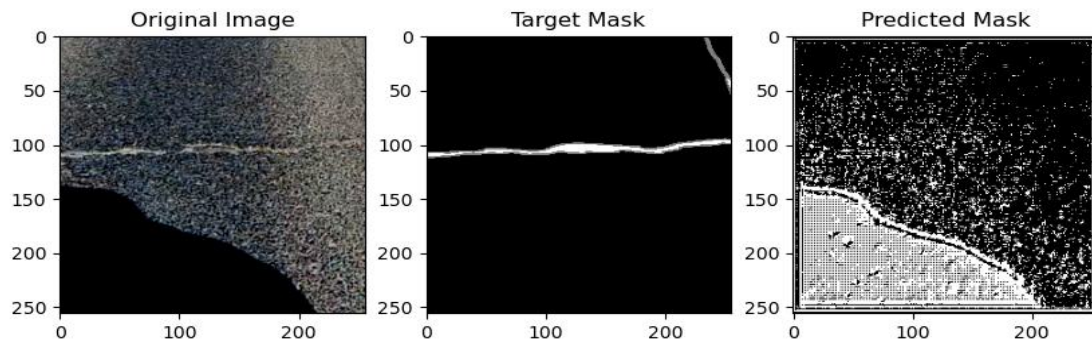


Figure 54. The shades as cracks and brighter zones as background (c)

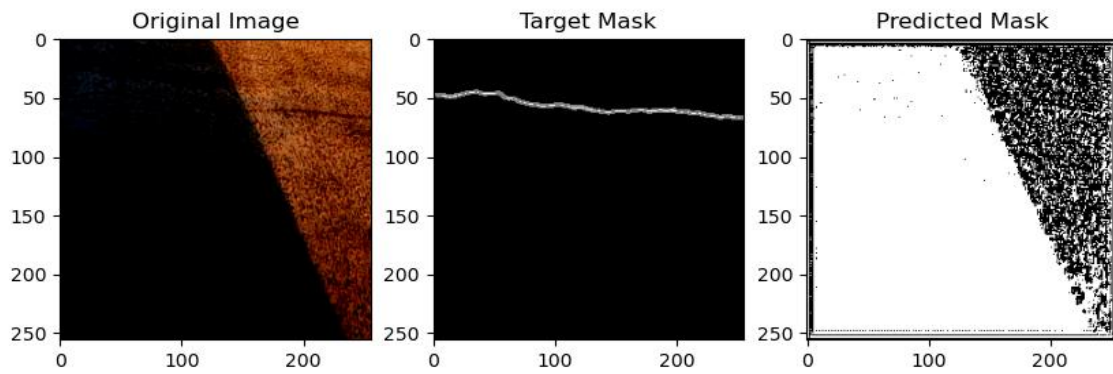


Figure 55. The shades as cracks and brighter zones as background (d)

Figures 56 and 57 have shown some confusing predicted masks because of the less bright backgrounds. In addition, Figure 57 shows the effect of the manhole on the predicted mask.

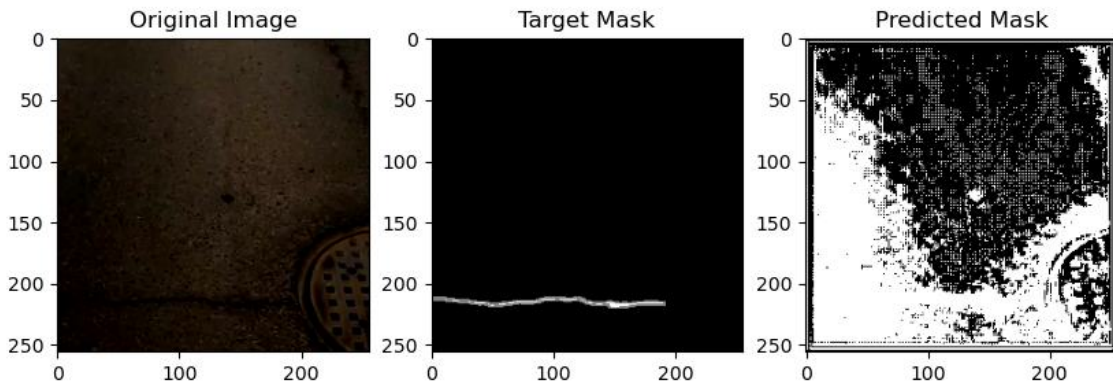


Figure 56. Less bright raw images with redundant background information affects the predicted mask

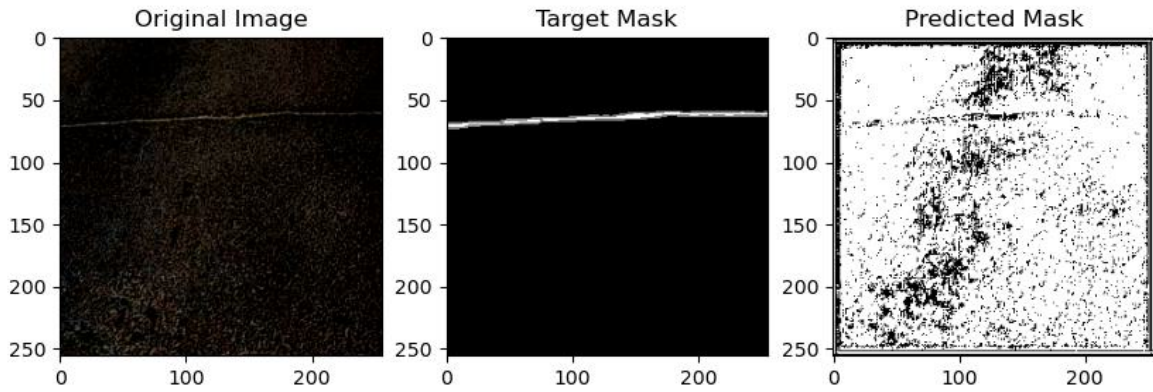


Figure 57. Less bright raw images affect the predicted mask

4.2. 2 UNet model performance for segmenting CRACKTREE dataset

The UNet model was again trained and tested using the CRACKTREE206 dataset. Section 4.2.2.1 shows the analysis of the loss curve, section 4.2.2.2 represents the values of evaluation metrics, and section 4.2.2.3 evaluates the predicted masks.

4.2.2.1 Analysis of the loss curve of the UNet model on CRACKTREE206 dataset

The loss curve in Figure 58 has shown that training and validation loss reduced significantly to nearly 30.0%. Due to time constraints, the training process was stopped and it was assumed as the minimal impact of underfitting issue because the both loss curves would merge minimizing the losses in the closer epoch numbers.

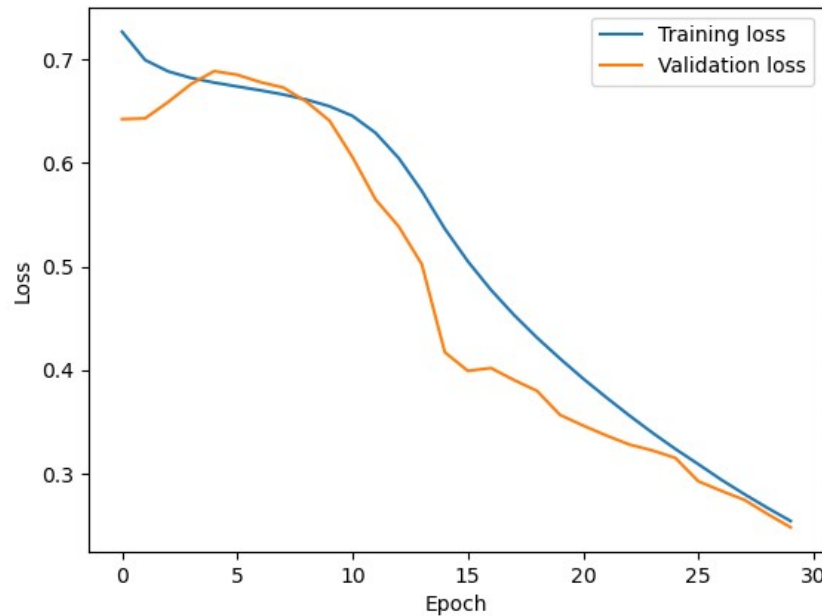


Figure 58. Loss curve for UNet model on CRACKTREE206 dataset.

4.2.2.2 Evaluation metric elements of the UNet model on CRACKTREE206 dataset

Table 7 shows the evaluation metrics of the UNet model on the CRACKTREE206 dataset. The IoU score is 60.0%, which means that there are 60.0% pixels overlaps from the predicted masks to ground truths. The precision is 95.0%, indicating that 95.0% of predicted true positives by the model are accurate from all predicted true positive pixels. The recall is 58.0%, which means that the model could recapture 58.0% of the true positives accurately from the ground truths. Finally, the F1 score is 72.0%, which is the average of precision and recall.

Table 7. Performance of the modified-UNet model on CRACKTREE206 dataset.

IoU	Precision	Recall	F1
60.0%	95.0%	58.0%	72.0%

4.2.2.3 Evaluation of predicted masks by the UNet model on CRACKTREE260 dataset

Figures 59, 60, and 61 show that the model has predicted the crack shapes reasonably well, while these shapes have some distortions and additional spots.

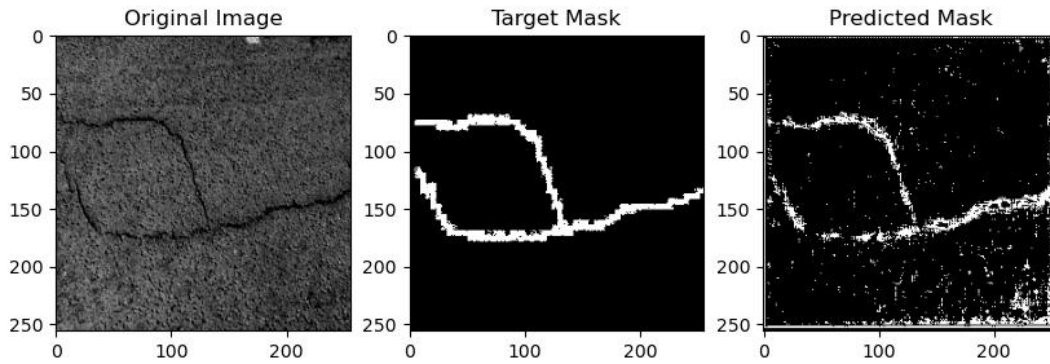


Figure 59. Comparison of predicted crack shapes (a)

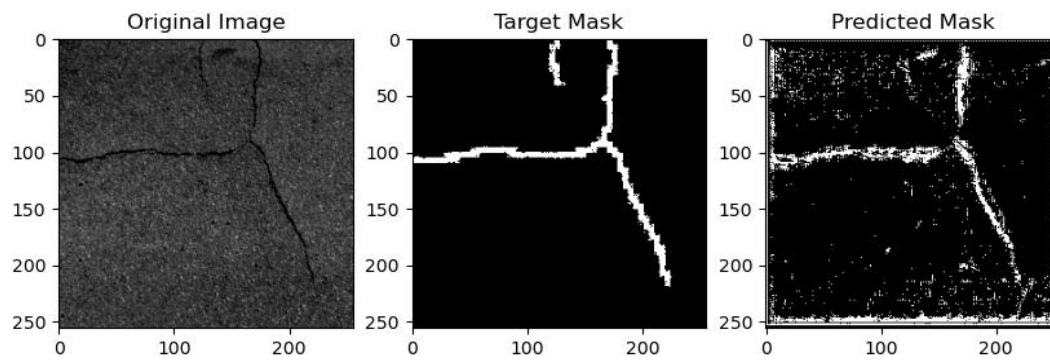


Figure 60. Comparison of predicted crack shapes (b)

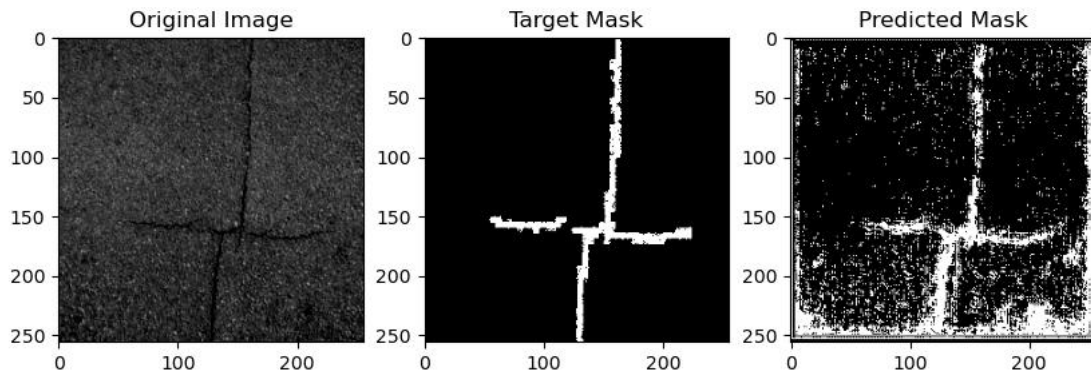


Figure 61. Comparison of predicted crack shapes (c)

Figures 62, 63, 64 and 65 show that the model has identified the complete shapes of the cracks, however, some portions of the crack regions are missing. Also, Figures 65 and 66 have a few redundant spots because of lights and shades. In Figure 66, the background of the crack in the raw image is less bright, nonetheless, the model differentiated the crack regions and backgrounds.

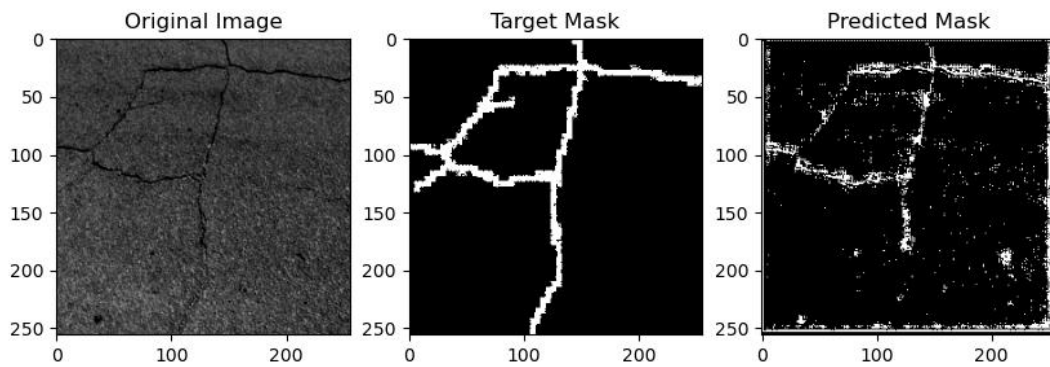


Figure 62. Partially predicted crack shapes (a)

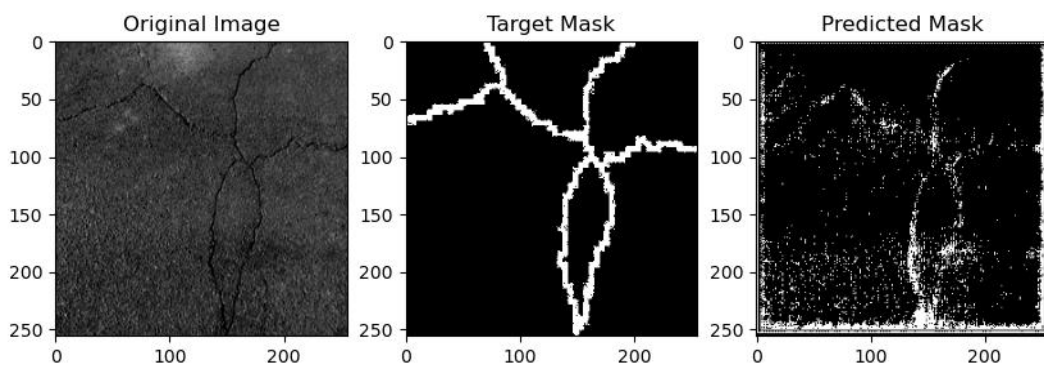


Figure 63. Partially predicted crack shapes (b)

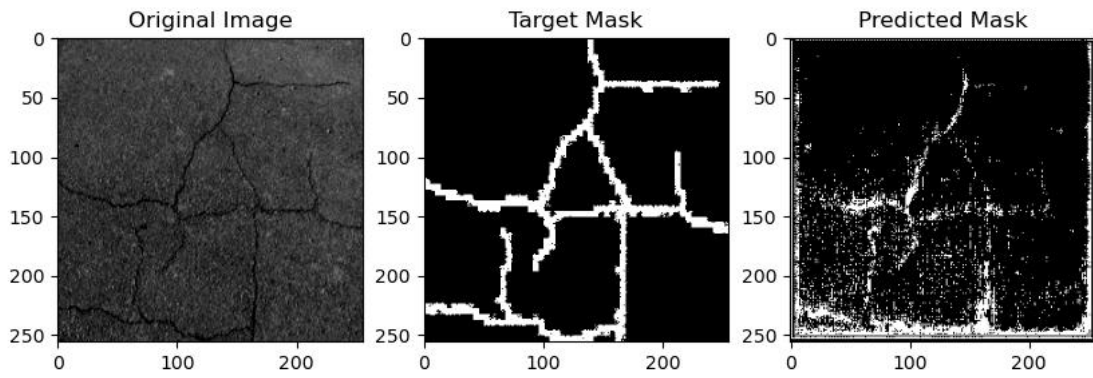


Figure 64. Partially predicted crack shapes (c)

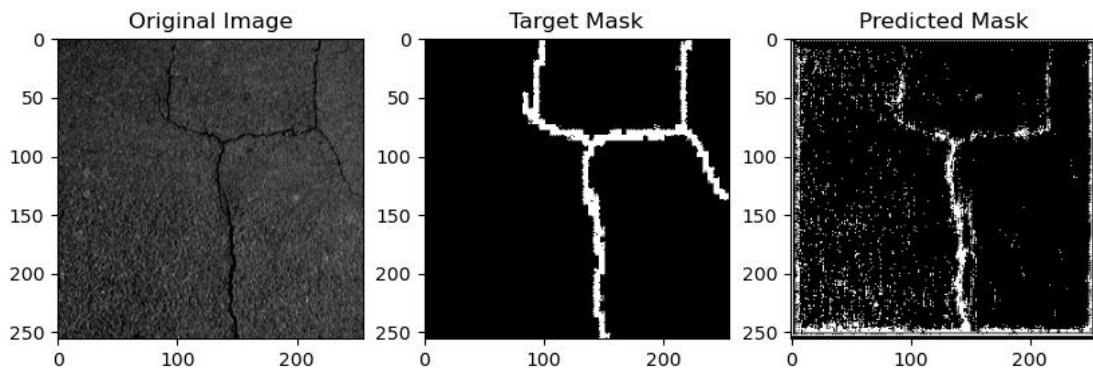


Figure 65. Almost predicted the shape but missed one little portion. (a)

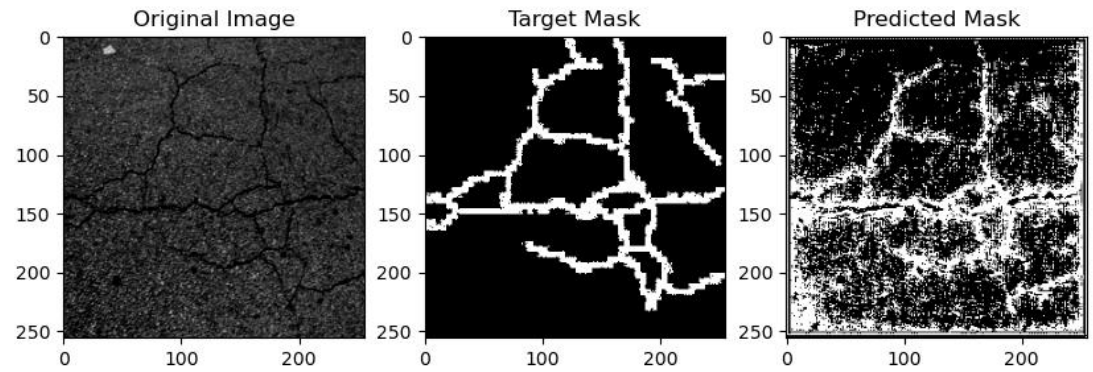


Figure 66. Almost predicted the shape but missed one little portion. (b)

Figure 67 represents predicted masks generated from the CRACKTREE260 dataset that are affected by the additional backgrounds of the original images, and the model predicted the shaded regions as the crack. However, the model fails to identify the crack regions in Figure 68.

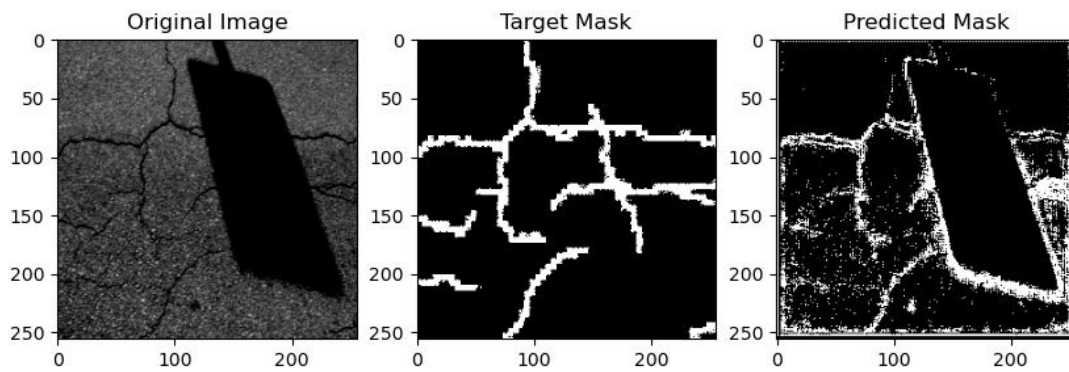


Figure 67. Shades are mixing with the ground truths (a)

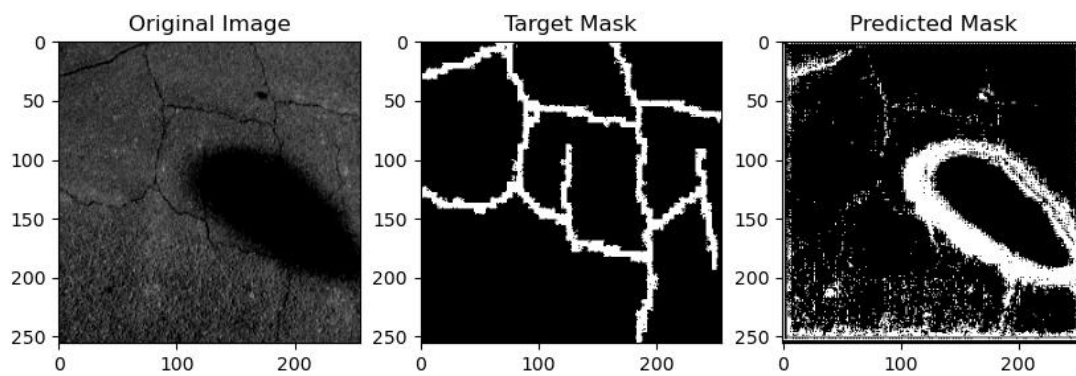


Figure 68. Shades are mixing with the ground truths (b)

In Figures 69, 70, and 71, crack regions are not predicted in the masks, instead, the model prediction is affected by the edges of the shades because it considers those regions as crack. The brightness can cause the model to segment wrong regions as crack.

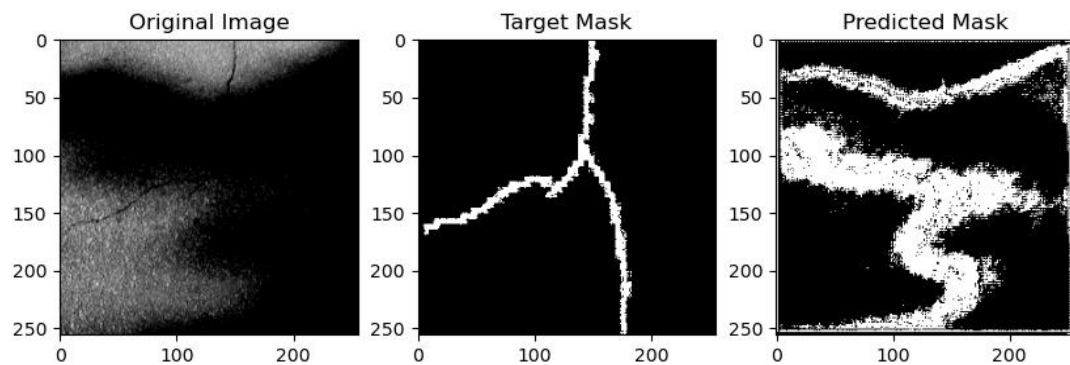


Figure 69. Affected by the shades of raw images

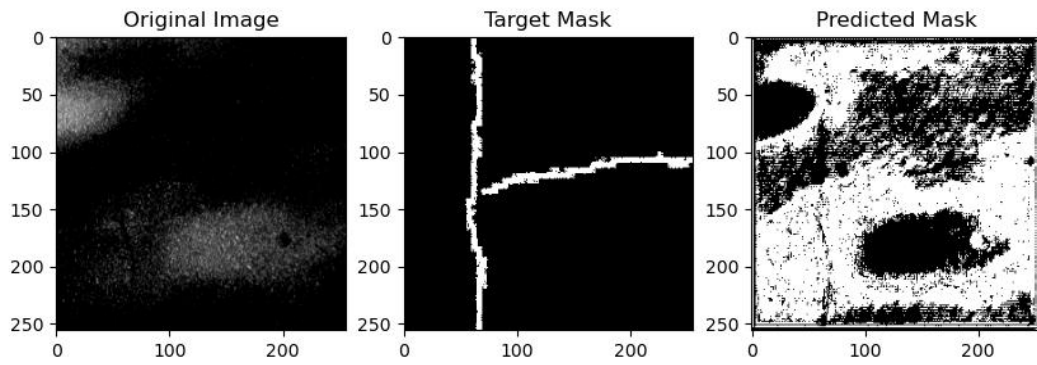


Figure 70. Affected by the shades of raw images.

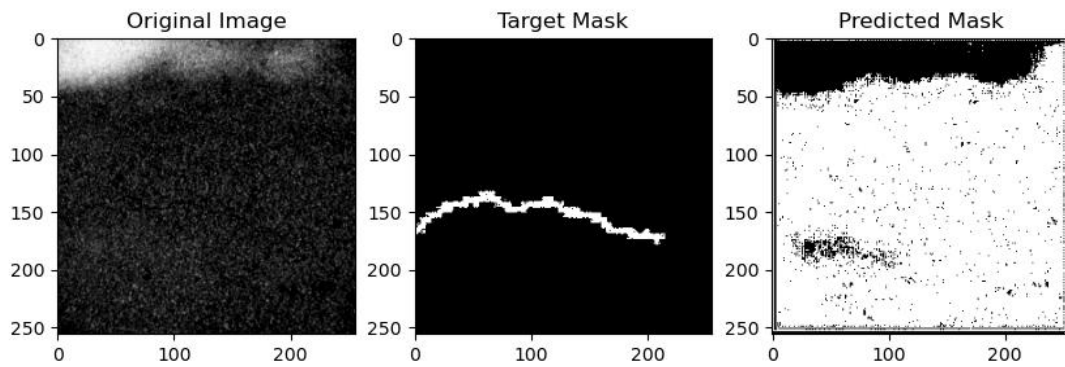


Figure 71. Affected by the shades of raw images and bright zones are considered as background in the predicted mask

For the less bright crack images, the model was not able to effectively differentiate the cracks regions and background in this dataset as presented in Figures 72, 73 and 74.

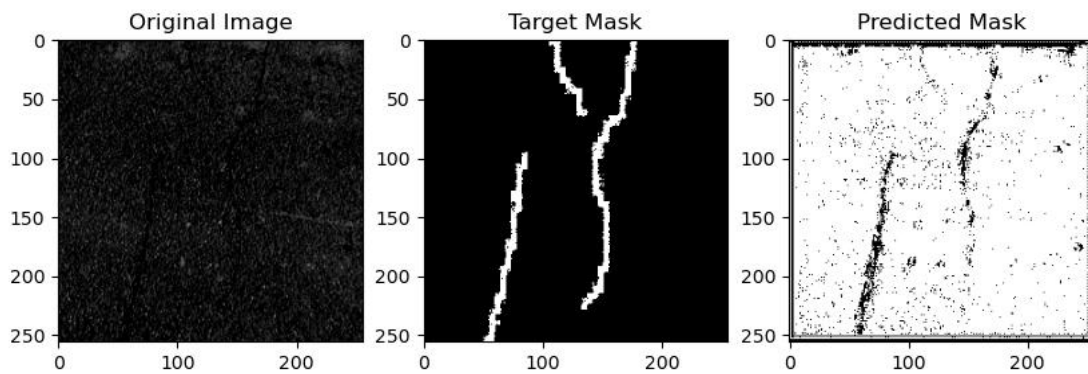


Figure 72. Less bright background caused for unrecognizable mask

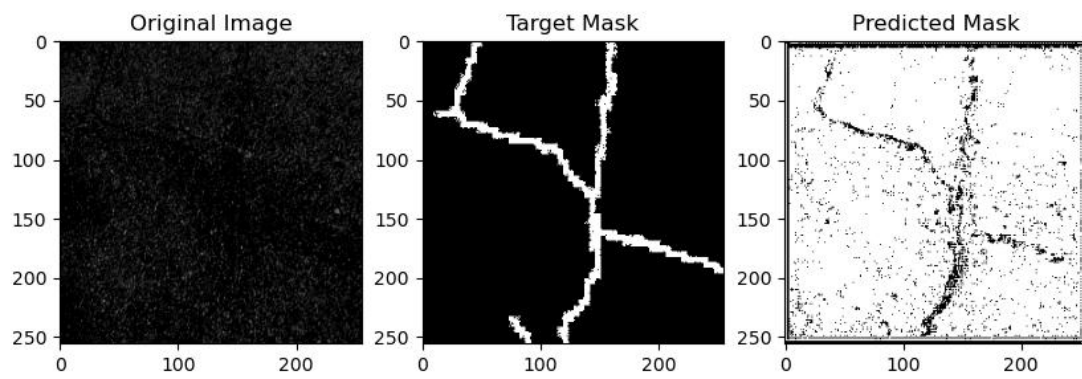


Figure 73. Less bright background caused for unrecognizable mask

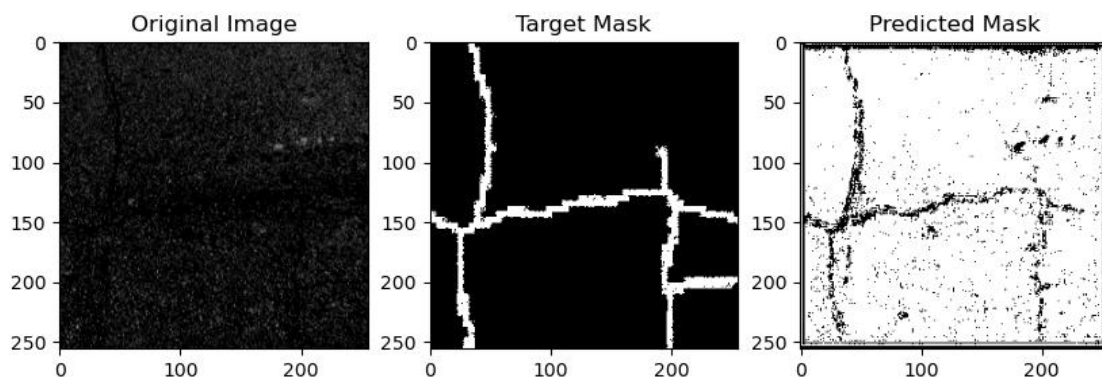


Figure 74. Less bright background caused for unrecognizable mask

CHAPTER 5: DISCUSSIONS

This study indicates that Reproducing the consistent satisfactory values of the evaluation metrics from deep learning models in every training and testing trial was challenging. Similar situations occurred in some other deep learning studies [78]. The training process of the deep learning models might account for such unstable values of the evaluation metrics, where the model randomly selects training images to the batch size. These shuffled batches differ in every experiment; therefore, the models are trained differently in every trial. This phenomenon is the reason for producing different values of the evaluation metrics by the model in each trial [79]. Moreover, the inconsistencies in the training process also affected the predicted masks, consequently, it is necessary to adjust the contrast of these masks to compare them with the ground truths. Furthermore, this study faced challenges obtaining adequate CPU support due to the heavy traffic at UNC Charlotte HPCC. Henceforth, this study only applied random seed values to mitigate these challenges in segmentation tasks. The reason is that classification tasks could reproduce approximately closer values of the evaluation metrics in different experiments in this study. Comparatively less computation intensity of classification task might lead to model to produce nearly consistent outputs in different experiments.

5.1 Discussion on classification task with ResNet50

The transferred ResNet-50 model showed its efficiency in classification tasks. Some facts are discussed in the following based on the performance of the model on the AugCrack132 dataset:

- For the classification task, the performance of the transferred-ResNet50 model at epoch 500 with the ‘Adam’ optimizer was a reasonable combination because the model could predict all types of cracks at more than 40%.
- The ‘Adam’ could optimize the model parameters to predict cracks from all crack types simultaneously given the fact that there are only 5 alligator cracks used for testing. It means that this optimizer algorithm could converge the model parameters (bias and weights) to three crack types successfully even though the dataset was unbalanced. On the other hand, the SGD optimizers showed bias to the transverse cracks because it outnumbered the other two types.
- Although the Adaptive moment estimation (Adam) optimizer performed better on the classification tasks than the stochastic gradient descent (SGD), for a large dataset, the model should be trained using both optimizers.
- The ground truths of the alligator cracks need to be more specific for classification. The alligator cracks’ shapes, and the corresponding ground truths need to be defined to train a classification-oriented CNN model. Although the model has predicted the alligator cracks in two combinations (at epoch of 40 and 500 with ‘Adam’ optimizer), the ground truth needs improvement.

- Computation power is crucial to obtain the proper outputs because it was challenging to deploy enough CPUs from the HPCC due to tight schedules of the servers.

5.2 Discussion on segmentation task with modified-UNet

This modified-UNet model has shown segmentation performance on the EdmCrack470 and CRACKTREE206 asphalt pavement crack datasets. There are some critical discussions and analysis have been made in the following based on the segmentation task conducted in this research.

- Raw images in the EdmCrack470 dataset have much more noise than the CRACKTREE206 dataset. This is the reason the predicted masks from EdmCrack470 dataset were more affected by the noises. The model mistakenly predicted some noises as crack and background.
- Raw images in the CRACKTREE206 dataset are affected by the brightness issues. In addition, there are some shaded regions that caused the model to predict wrong crack shapes.
- The EdmCrack470 dataset has more transverse cracks, therefore, predicted masks from this dataset are biased towards transverse shapes. On the other hand, the CRACKTREE206 has cracks with different shapes. Therefore, the model was able to predict different types of cracks using the CRACKTREE206 dataset.
- Addition of scale wise layers in encoder-decoder sections might have potential to improve the qualities of the predicted masks by differentiating the cracks and noises. Some CNN architectures such as DeepCrack, ConnCrack, and PDSNET are designed with multi-scaling features to store the crack information in different layers. In DeepCrack [57] multi-scale layers are combined including different convolution layers. The sophisticated connection of the convolution layers, maxpool and unpool operations in encoder and decoder networks needs an adequately large dataset to train the all hidden layers and pass the information in each scale. With a small dataset, augmentation pipelines and insufficient computational support can raise model complexity.
- The modified-UNet model structure is comparatively less sophisticated than the aforementioned CNNs, therefore, it can be effective for pavement crack segmentation with minimal complexities to different types of data structures. The addition of multi-scale layers in this CNN can outperform the advanced and complicated CNNs for asphalt pavement crack segmentation with a capacity of identifying cracks and anomalies.
- The addition of a multi-scale layer in this modified-UNet model can increase the model's potential to segment the less bright cracks. DeepCrack models suffer the same issues and use crack images with bright backgrounds to resolve the issue [57]. This is why images with proper brightness can significantly improve the performance of segmentation tasks.

- Contrast adjustment is important for appropriately predicting masks because it helps the model to differentiate noise and cracks.
- Reproducing the same result is a challenging task in deep learning models. Therefore, this study used random seed values in the programming code in the segmentation task.

CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions on classification task with ResNet50

The transferred-ResNet50 model classified the three crack types simultaneously from the newly created dataset. However, the source dataset of the AugCrack132 dataset, does not have enough alligator cracks, in addition, it was a segmentation-oriented dataset, which means that there were no crack types specified in the labeled ground truths. This study categorized the cracks from the origin dataset into three types and used the original ground truths. The adaptive moment estimation or “Adam” optimizing algorithm could optimize the model parameters for classifying three crack types at the same time.

In this study, the transfer learning in the classification layer of the transferred-ResNet50 model proved to be effective. At the same time, the same setup needs to be utilized in a dataset with more variation of cracks and more classification oriented ground truths.

The hyperparameter optimization has also made this research to explore the model performance in different combinations of epochs and optimizers. The number of such hyperparameter combinations can be increased to further improve the model performance.

6.2 Conclusions on segmentation task with UNet

The UNet Model showcased satisfactory training results on both datasets; however, the training processes were computationally and time intensive. It took more than 1 hour for 30 epochs to complete the training and testing process using the smaller dataset. Therefore, for a large dataset, the model parameters need adjustments with more computational support.

The addition of new layers to the model architecture can improve the model performance because the model is more capable to extract and differentiate the cracks and shades properly. For example, the DeepCrack model has fused the encoder and decoder networks at different scales. Also, in the ConnCrack model [3], the connectivity maps cooperated to predict true labels. The modified UNet model in this study might need such upgrades with multi-scale layers in future for differentiating the crack shapes and noises in segmentation tasks.

The quality of segmented images can be improved with a better cleaning process of the original images. This study conducted the center cropping on the original images of EdmCrack600 datasets to exclude the redundant background information. However, some additional data cleansing can increase the accuracy in the predictions. For example, Mei et al. [3] applied morphological operations to preprocess the EdmCrack600 dataset to train a novel CNN model, ConnCrack [3]. This study only cropped the images and corresponding masks from center regions to avoid the noises, and additional image processing operations could enhance the quality of the images and corresponding masks, resulting in better model performance. Lastly, equally distributed brightness on the original images might play a significant role in improving the accuracy of the predicted masks.

6.3 Recommendations

This study has applied the transferred-ResNet50 model for classifying and used an UNet model for segmentation asphalt pavement cracks. recommendations for further broadening the horizon of this study are:

- Increasing the classification categories to include severity levels of asphalt cracks should be further researched. The *NCDOT Digital Imagery Distress Evaluation Handbook* [30] defines the cracks with different levels of severity. Based on this manual, future studies can include datasets that have more crack types with severity conditions. As a result, a more advanced multi-class classification can be achieved using a model that is similar to CNN models with transfer learning that were developed in this study.
- For the segmentation task, data cleaning, preprocessing and quality should be emphasized in the future research.
- This study tuned the ResNet50 classifier with several epochs and optimizer combinations to investigate the model performance on pavement crack classification. For future studies, a similar tuning process should be considered for the modified UNet model for segmentation tasks.
- The trained models should be tested on completely unknown datasets to evaluate their biases. Due to the data scarcities, and time constraints related to preprocessing, it was challenging to prepare completely different testing datasets. This issue should be resolved in future studies.
- Besides the hyperparameter optimization, mathematical optimizations could improve the results by adjusting the model parameters with the datasets. For the crack classification and segmentation, the label data provides more pixelated information on the background compared to the label of the cracks. As a result, the models extract the crack information from limited true positive pixelated ground truths. In this regard, mathematical optimization can be a further research approach to fine-tune these two models for asphalt pavement crack classification and segmentation. The built-in optimization algorithms such as “Adam” or “SGD” update the gradients of the model parameters during the backpropagation. Developing a custom optimization algorithm with appropriate mathematical variables and parameters might outperform the built-in optimizers for pavement crack datasets.
- Adequate computational power can be significant to achieve accurate outcomes. This study encountered challenges in accessing sufficient CPUs on the UNCC HPCC due to the heavy usages from other users.

REFERENCES

- [1] “America’s Infrastructure Report Card 2021 | GPA: C-.” <https://infrastructurereportcard.org/> (accessed Jun. 14, 2023).
- [2] E. Eslami and H.-B. Yun, “Attention-based multi-scale convolutional neural network (A+ MCNN) for multi-class classification in road images,” *Sensors*, vol. 21, no. 15, p. 5137, 2021, [Online]. Available: <https://www.mdpi.com/1424-8220/21/15/5137/pdf>
- [3] Q. Mei and M. Gül, “A cost effective solution for pavement crack inspection using cameras and deep neural networks,” *Constr Build Mater*, vol. 256, p. 119397, 2020, [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0950061820314021>
- [4] J. Ha, D. Kim, and M. Kim, “Assessing severity of road cracks using deep learning-based segmentation and detection,” *J Supercomput*, vol. 78, no. 16, pp. 17721–17735, 2022, [Online]. Available: <https://link.springer.com/article/10.1007/s11227-022-04560-x>
- [5] H. D. Cheng and M. Miyojim, “Automatic pavement distress detection system,” *Inf Sci (N Y)*, vol. 108, no. 1, pp. 219–240, 1998, doi: 10.1016/S0020-0255(97)10062-7.
- [6] T. B. J. Coenen and A. Golroo, “A review on automated pavement distress detection methods,” *Cogent Eng*, vol. 4, no. 1, p. 1374822, 2017, [Online]. Available: <https://www.tandfonline.com/doi/pdf/10.1080/23311916.2017.1374822>
- [7] Y. Sun, E. Salari, and E. Chou, “Automated pavement distress detection using advanced image processing techniques,” in *2009 IEEE International Conference on Electro/Information Technology*, 2009, pp. 373–377. doi: 10.1109/EIT.2009.5189645.
- [8] N. Sholevar, A. Golroo, and S. R. Esfahani, “Machine learning techniques for pavement condition evaluation,” *Autom Constr*, vol. 136, p. 104190, 2022, doi: 10.1016/j.autcon.2022.104190.
- [9] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.” arXiv, 2015. doi: 10.48550/arXiv.1409.1556.
- [10] C. Szegedy *et al.*, “Going Deeper With Convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deepier_With_2015_CVPR_paper.html
- [11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
- [12] “Deep Learning: Understanding The Inception Module | by Richmond Alake | Towards Data Science.” <https://towardsdatascience.com/deep-learning-understand-the-inception-module-56146866e652> (accessed Jun. 14, 2023).
- [13] N. Kheradmandi and V. Mehranfar, “A critical review and comparative study on image segmentation-based techniques for pavement crack detection,” *Constr Build Mater*, vol. 321, p. 126162, 2022, doi: 10.1016/j.conbuildmat.2021.126162.
- [14] K. Gopalakrishnan, S. K. Khaitan, A. Choudhary, and A. Agrawal, “Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection,” *Constr Build Mater*, vol. 157, pp. 322–330,

- 2017, [Online]. Available:
<https://www.sciencedirect.com/science/article/am/pii/S0950061817319335>
- [15] R. Stricker, M. Eisenbach, M. Sesselmann, K. Debes, and H.-M. Gross, "Improving Visual Road Condition Assessment by Extensive Experiments on the Extended GAPs Dataset," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8. doi: 10.1109/IJCNN.2019.8852257.
 - [16] G. Li, Q. Liu, W. Ren, W. Qiao, B. Ma, and J. Wan, "Automatic recognition and analysis system of asphalt pavement cracks using interleaved low-rank group convolution hybrid deep network and SegNet fusing dense condition random field," *Measurement*, vol. 170, p. 108693, 2021, doi: 10.1016/j.measurement.2020.108693.
 - [17] S. Li, Y. Cao, and H. Cai, "Automatic pavement-crack detection and segmentation based on steerable matched filtering and an active contour model," *Journal of Computing in Civil Engineering*, vol. 31, no. 5, p. 04017045, 2017, [Online]. Available: [https://ascelibrary.org/doi/abs/10.1061/\(ASCE\)CP.1943-5487.0000695](https://ascelibrary.org/doi/abs/10.1061/(ASCE)CP.1943-5487.0000695)
 - [18] Z. Han, H. Chen, Y. Liu, Y. Li, Y. Du, and H. Zhang, "Vision-based crack detection of asphalt pavement using deep convolutional neural network," *Iranian Journal of Science and Technology, Transactions of Civil Engineering*, vol. 45, pp. 2047–2055, 2021, [Online]. Available:
<https://link.springer.com/article/10.1007/s40996-021-00668-x>
 - [19] G. Yang *et al.*, "Datasets and processing methods for boosting visual inspection of civil infrastructure: A comprehensive review and algorithm comparison for crack classification, segmentation, and detection," *Constr Build Mater*, vol. 356, p. 129226, Nov. 2022, doi: 10.1016/J.CONBUILDMAT.2022.129226.
 - [20] M. Aliyari, E. L. Droguett, and Y. Z. Ayele, "UAV-based bridge inspection via transfer learning," *Sustainability*, vol. 13, no. 20, p. 11359, 2021, [Online]. Available: <https://www.mdpi.com/2071-1050/13/20/11359/pdf>
 - [21] M. Żarski, B. Wójcik, and J. A. Mischczak, "KrakN: Transfer Learning framework for thin crack detection in infrastructure maintenance," *SoftwareX*, vol. 16, p. 100893, 2021, doi: 10.1016/j.softx.2021.100893.
 - [22] M. Eisenbach *et al.*, "How to get pavement distress detection ready for deep learning? A systematic approach," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2017-May, pp. 2039–2047, Jun. 2017, doi: 10.1109/IJCNN.2017.7966101.
 - [23] P. Hühthwohl, R. Lu, and I. Brilakis, "Multi-classifier for reinforced concrete bridge defects," *Autom Constr*, vol. 105, p. 102824, 2019, [Online]. Available: [https://www.repository.cam.ac.uk/bitstream/handle/1810/301324/paper%20\[1\]_accepted_version.pdf?sequence=1](https://www.repository.cam.ac.uk/bitstream/handle/1810/301324/paper%20[1]_accepted_version.pdf?sequence=1)
 - [24] Ç. F. Özgenel and A. G. Sorguç, "Performance comparison of pretrained convolutional neural networks on crack detection in buildings," IAARC Publications, 2018, pp. 1–8. [Online]. Available:
https://www.researchgate.net/profile/Caglar-Oezgenel/publication/326676263_Performance_Comparison_of_Pretrained_Convolutional_Neural_Networks_on_Crack_Detection_in_Buildings/links/5d6f91af4585151ee49b8bf5/Performance-Comparison-of-Pretrained-Convolutional-Neural-Networks-on-Crack-Detection-in-Buildings.pdf
 - [25] S. Hudson, ... W. H.-P. M., and undefined 1992, "Minimum requirements for standard pavement management systems," *books.google.comSW Hudson, WR Hudson, RF CarmichaelPavement Management Implementation, STP, 1992•books.google.com*,

- Accessed: Jul. 28, 2023. [Online]. Available:
<https://books.google.com/books?hl=en&lr=&id=laZiI07As8oC&oi=fnd&pg=PA19&dq=Hudson,+S.W.,+Hudson,+W.R.,+and+Carmichael,+R.F.,+1992.+%E2%80%9CMinimum+Requirements+for++++++Standard+Pavement+management+Systems%E2%80%9D.+In+Pavement+Management+Implementation,+eds++++++F.B.+Holt+%26+W.L.,+Gramling,+STP+1121,+American+Society+for+Testing+&ots=6K8FLdv6FA&sig=rT2vs89QaJx6BVSnr1XnfKyDg>
- [26] N. Ismail, A. Ismail, and R. Atiq, "An overview of expert systems in pavement management," *European Journal of Scientific Research*, vol. 30, no. 1, pp. 99–111, 2009, [Online]. Available:
https://www.researchgate.net/profile/Amiruddin-Ismail/publication/228623751_An_o_verview_of_expert_systems_in_pavement_management/links/00463521c32ca9cae000000/An-overview-of-expert-systems-in-pavement-management.pdf
 - [27] A. Ragnoli, M. R. De Blasiis, and A. Di Benedetto, "Pavement distress detection methods: A review," *Infrastructures (Basel)*, vol. 3, no. 4, p. 58, 2018, [Online]. Available: <https://www.mdpi.com/2412-3811/3/4/58/pdf>
 - [28] H. Lee and J. Kim, "Development of a crack type index," *Transp Res Rec*, vol. 1940, no. 1, pp. 99–109, 2005.
 - [29] P. G. Scholar, "Review and analysis of crack detection and classification techniques based on crack types," *Int. J. Appl. Eng. Res*, vol. 13, pp. 6056–6062, 2018, [Online]. Available: https://www.ripublication.com/ijaer18/ijaerv13n8_63.pdf
 - [30] N. Carolina, "NCDOT Digital Imagery Distress Evaluation Handbook," 2011.
 - [31] "Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network | upGrad blog." <https://www.upgrad.com/blog/basic-cnn-architecture/> (accessed Jun. 14, 2023).
 - [32] "Neural network types - Questions and Answers in MRI." <https://mriquestions.com/deep-network-types.html> (accessed Jun. 14, 2023).
 - [33] "Convolutional Neural Networks, Explained | by Mayank Mishra | Towards Data Science." <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939> (accessed Jun. 14, 2023).
 - [34] "Convolutional Neural Networks — A Beginner's Guide | by Krut Patel | Towards Data Science." <https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022> (accessed Jun. 14, 2023).
 - [35] "What is 'stride' in Convolutional Neural Network? | by Ting-Hao Chen | Machine Learning Notes | Medium." <https://medium.com/machine-learning-algorithms/what-is-stride-in-convolutional-neural-network-e3b4ae9baedb> (accessed Jun. 14, 2023).
 - [36] T.-H. Chen, "What is 'padding' in convolutional neural network." Medium, 2017.
 - [37] "Understanding of Convolutional Neural Network (CNN) — Deep Learning | by Prabhu | Medium." <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148> (accessed Jun. 14, 2023).
 - [38] "ResNet50. ResNet-50 is a convolutional neural... | by Aditi Rastogi | Dev Genius." <https://blog.devgenius.io/resnet50-6b42934db431> (accessed Jun. 14, 2023).

- [39] “Backward Propagation | Backward Propagation Working in Neural Network.” <https://www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/> (accessed Jun. 14, 2023).
- [40] “Backpropagation in a Neural Network: Explained | Built In.” <https://builtin.com/machine-learning/backpropagation-neural-network> (accessed Jun. 14, 2023).
- [41] “Backpropagation Definition | DeepAI.” <https://deepai.org/machine-learning-glossary-and-terms/backpropagation> (accessed Jul. 28, 2023).
- [42] Y. Xue, Y. Tong, and F. Neri, “An ensemble of differential evolution and Adam for training feed-forward neural networks,” *Inf Sci (N Y)*, vol. 608, pp. 453–471, 2022, doi: 10.1016/j.ins.2022.06.036.
- [43] “| notebook.community.” https://notebook.community/aimacode/aima-python/notebooks/chapter19/Optimizer%20and%20Backpropagation?fbclid=IwAR15XOj5KHCsUt7e3REbWYQGkzKbg7dVqAGGrmsECw8_y3fR7c3eB3fls9Y (accessed Jul. 28, 2023).
- [44] “Mathematical Understanding of Backpropagation | by Tanoy Debnath | Jul, 2023 | Medium.” <https://medium.com/@kuettanoydebnath/mathematical-understanding-of-backpropagation-7315fec826fc> (accessed Jul. 28, 2023).
- [45] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation.” arXiv, 2015. doi: 10.48550/arXiv.1505.04597.
- [46] “Conv2d — PyTorch 2.0 documentation.” <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html> (accessed Jun. 14, 2023).
- [47] “What Is Transfer Learning? A Guide for Deep Learning | Built In.” <https://builtin.com/data-science/transfer-learning> (accessed Jun. 14, 2023).
- [48] “How Transfer Learning works. Transfer Learning will be the next... | by HAFEEZ JIMOH | Towards Data Science.” <https://towardsdatascience.com/how-transfer-learning-works-a90bc4d93b5e> (accessed Jun. 14, 2023).
- [49] “Evolution of CNN architectures | Mastering PyTorch.” <https://subscription.packtpub.com/book/data/9781789614381/5/ch05lvl1sec17/evolution-of-cnn-architectures> (accessed Jul. 28, 2023).
- [50] “Face Recognition using Transfer Learning | by Chirag Goel | Medium.” <https://medium.com/@chiraggoelit/face-recognition-using-transfer-learning-9986728c443d> (accessed Jul. 28, 2023).
- [51] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019. [Online]. Available: <https://library.oapen.org/bitstream/handle/20.500.12657/23012/1007149.pdf>
- [52] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020, doi: 10.1016/J.NEUCOM.2020.07.061.
- [53] W. Song, G. Jia, H. Zhu, D. Jia, and L. Gao, “Automated Pavement Crack Damage Detection Using Deep Multiscale Convolutional Features,” *J Adv Transp*, vol. 2020, p. e6412562, 2020, doi: 10.1155/2020/6412562.

- [54] Y. Ren *et al.*, “Image-based concrete crack detection in tunnels using deep fully convolutional networks,” *Constr Build Mater*, vol. 234, p. 117367, 2020, [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0950061819328193>
- [55] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Trans Pattern Anal Mach Intell*, vol. 39, no. 12, pp. 2481–2495, 2017, [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7803544/>
- [56] X. Zhang, D. Rajan, and B. Story, “Concrete crack detection using context-aware deep semantic segmentation network,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 34, no. 11, pp. 951–971, Nov. 2019, doi: 10.1111/MICE.12477.
- [57] Zou, Q., Zhang, Z., Li, Q., Qi, X., Wang, Q., & Wang, S. (2019). DeepCrack: Learning hierarchical convolutional features for crack detection. *IEEE Transactions on Image Processing*, 28(3), 1498–1512. <https://doi.org/10.1109/TIP.2018.2878966>.
- [58] T. Wen *et al.*, “Automated pavement distress segmentation on asphalt surfaces using a deep learning network,” *International Journal of Pavement Engineering*, vol. 0, no. 0, pp. 1–14, 2022, doi: 10.1080/10298436.2022.2027414.
- [59] A. Zhang *et al.*, “Automated Pixel-Level Pavement Crack Detection on 3D Asphalt Surfaces Using a Deep-Learning Network,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 32, no. 10, pp. 805–819, 2017, doi: 10.1111/mice.12297.
- [60] A. Zhang *et al.*, “Deep Learning–Based Fully Automated Pavement Crack Detection on 3D Asphalt Surfaces with an Improved CrackNet,” *Journal of Computing in Civil Engineering*, vol. 32, no. 5, p. 04018041, Jul. 2018, doi: 10.1061/(ASCE)CP.1943-5487.0000775.
- [61] Y. Fei *et al.*, “Pixel-level cracking detection on 3D asphalt pavement images through deep-learning-based CrackNet-V,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 1, pp. 273–284, 2019, [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8620557/>
- [62] K.-L. Lu, “Evaluation and Comparison of Deep Learning Methods for Pavement Crack Identification with Visual Images,” *arXiv preprint arXiv:2112.10390*, 2021, [Online]. Available: <https://arxiv.org/pdf/2112.10390>
- [63] Y.-A. Hsieh and Y. J. Tsai, “Machine Learning for Crack Detection: Review and Model Performance Comparison,” *Journal of Computing in Civil Engineering*, vol. 34, no. 5, p. 04020038, 2020, doi: 10.1061/(ASCE)CP.1943-5487.0000918.
- [64] S. Jana, S. Thangam, A. Kishore, V. Sai Kumar, and S. Vandana, “Transfer learning based deep convolutional neural network model for pavement crack detection from images,” *International Journal of Nonlinear Analysis and Applications*, vol. 13, no. 1, pp. 1209–1223, 2022, [Online]. Available: http://journals.semnan.ac.ir/article_5672_b61e33f6480f24f30ac07df168ede3d2.pdf
- [65] S. Ranjbar, F. M. Nejad, and H. Zakeri, “An image-based system for pavement crack evaluation using transfer learning and wavelet transform,” *International Journal of Pavement Research and Technology*, vol. 14, pp. 437–449, 2021, [Online]. Available: <https://link.springer.com/article/10.1007/s42947-020-0098-9>
- [66] Z. Hong *et al.*, “Highway crack segmentation from unmanned aerial vehicle images using deep learning,” *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2021, [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9622267/>
- [67] Y. Bai, B. Zha, H. Sezen, and A. Yilmaz, “Deep cascaded neural networks for automatic detection of structural damage and cracks from images,” *ISPRS Annals of*

- the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, 2020, [Online]. Available: <https://par.nsf.gov/servlets/purl/10352805>
- [68] V. Mandal, A. R. Mussah, and Y. Adu-Gyamfi, "Deep Learning Frameworks for Pavement Distress Classification: A Comparative Analysis," *Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020*, pp. 5577–5583, Dec. 2020, doi: 10.1109/BIGDATA50022.2020.9378047.
 - [69] D. Arya *et al.*, "Global Road Damage Detection: State-of-the-art Solutions," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 5533–5539. doi: 10.1109/BigData50022.2020.9377790.
 - [70] F. Yang, L. Zhang, S. Yu, D. Prokhorov, X. Mei, and H. Ling, "Feature Pyramid and Hierarchical Boosting Network for Pavement Crack Detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1525–1535, 2020, doi: 10.1109/TITS.2019.2910595.
 - [71] "Transfer Learning for Computer Vision Tutorial — PyTorch Tutorials 2.0.1+cu117 documentation." https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html (accessed Jun. 15, 2023).
 - [72] "tutorials/beginner_source/transfer_learning_tutorial.py at main · pytorch/tutorials · GitHub." https://github.com/pytorch/tutorials/blob/main/beginner_source/transfer_learning_tutorial.py (accessed Jun. 14, 2023).
 - [73] "Transfer learning with ResNet-50 in PyTorch | Kaggle." <https://www.kaggle.com/code/pmigdal/transfer-learning-with-resnet-50-in-pytorch> (accessed Jun. 14, 2023).
 - [74] "CNN Model With PyTorch For Image Classification | by Pranjal Soni | TheCyPhy | Medium." <https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48> (accessed Jun. 14, 2023).
 - [75] "Image-Segmentation-with-UNet-PyTorch | Kaggle." <https://www.kaggle.com/code/gokulkarthik/image-segmentation-with-unet-pytorch/notebook> (accessed Jun. 14, 2023).
 - [76] "Intersection Over Union for Object Detection | Baeldung on Computer Science." <https://www.baeldung.com/cs/object-detection-intersection-vs-union> (accessed Jun. 15, 2023).
 - [77] "Precision and recall - Wikipedia." https://en.wikipedia.org/wiki/Precision_and_recall (accessed Jul. 28, 2023).
 - [78] "python - Machine learning model keeps on giving the same result even with different outputs - Stack Overflow." <https://stackoverflow.com/questions/64148327/machine-learning-model-keeps-on-giving-the-same-result-even-with-different-output> (accessed Jul. 30, 2023).
 - [79] A. L. Beam, A. K. Manrai, and M. Ghassemi, "Challenges to the Reproducibility of Machine Learning Models in Health Care," *JAMA*, vol. 323, no. 4, pp. 305–306, Jan. 2020, doi: 10.1001/JAMA.2019.20866.
 - [80] Y. Pan and Y. Li, "Toward Understanding Why Adam Converges Faster Than SGD for Transformers," May 2023, Accessed: Jul. 28, 2023. [Online]. Available: <https://arxiv.org/abs/2306.00204v1>
 - [81] "A Comprehensive Guide on Hyperparameter Tuning and its Techniques." <https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/> (accessed Aug. 15, 2023).

- [82] “A Comprehensive Guide on Hyperparameter Tuning and its Techniques.”
<https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/> (accessed Aug. 15, 2023).
- [83] “Models and pre-trained weights — Torchvision 0.15 documentation.”
<https://pytorch.org/vision/stable/models.html> (accessed Aug. 16, 2023).

APPENDIXES

Appendix A: Accuracy calculations in Table 5 on overall dataset in row c for in ResNet50 for classification

Alligator (5 images), longitudinal (21) and transverse (30) = 56 tested images. The 32 images will be chosen randomly in the batch size of every iteration. Therefore, it is not possible to say the exact number of x or y or z. but $x + y + z = 32$. Here x is alligator, y is longitudinal, and z is transverse cracks.

For class-wise prediction classification accuracy calculation 8 is followed here:

For example,

For 53% overall correction on the dataset means it's 17 images correct prediction of 32 predicted images. (32 is the batch size).

$x + y + z = 32$, but $x = \text{alligator}$, $y = \text{longitudinal}$ and $z = \text{transverse}$.

let's, 0% of x + 0% of y + 100% of z = 53% of 32

Therefore, $z = 16.96$ (total 17 transverse cracks were in the batch of 32 images and all of them were predicted correctly)

The same goes, for 21% of totally corrected prediction at epoch 40 and optimizer "Adam"

$x + y + z = 32$ (total sum is 32) as 32 images come randomly from 3 classes.

80% of x + 23.8% of y and 6.7% of z = 21% of 32

if $x = 4$, then $y = 9.648$, and $z = 18.352$. (as the exact number of x, y or z is not possible to say,

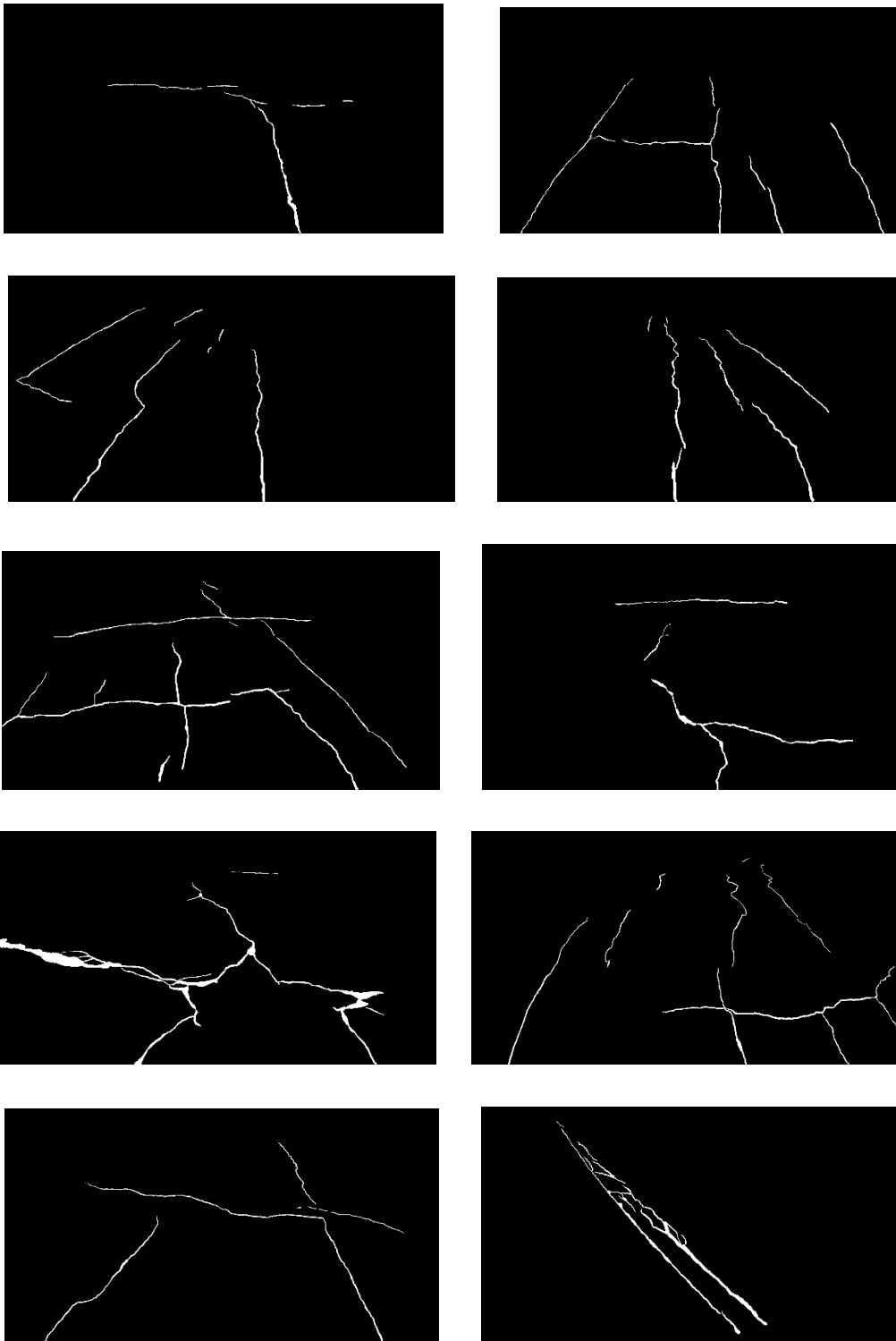
let's validate the equation with $x=4$) (x should be equal less than 5 in all cases)

Therefore, the values $x = 4$, $y = 9.648$, and $z = 18.352$ do indeed satisfy the equation $0.8x + 0.238y + 0.067z = 0.21 * 32$

Appendix B: Alligator cracks from EdmCrack600 in AugCrack132 dataset.



Appendix C: Corresponding masks of the alligator cracks in AugCrack132 dataset



Appendix D: Loss curves in different epochs and optimizer combinations in ResNet50

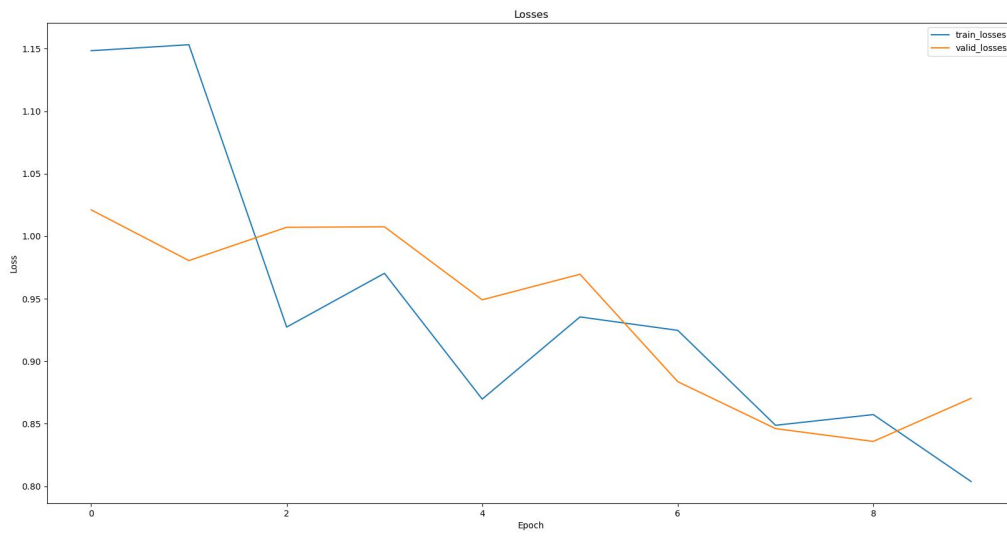


Figure 75. Loss curve at epoch 10 and optimizer “Adam”

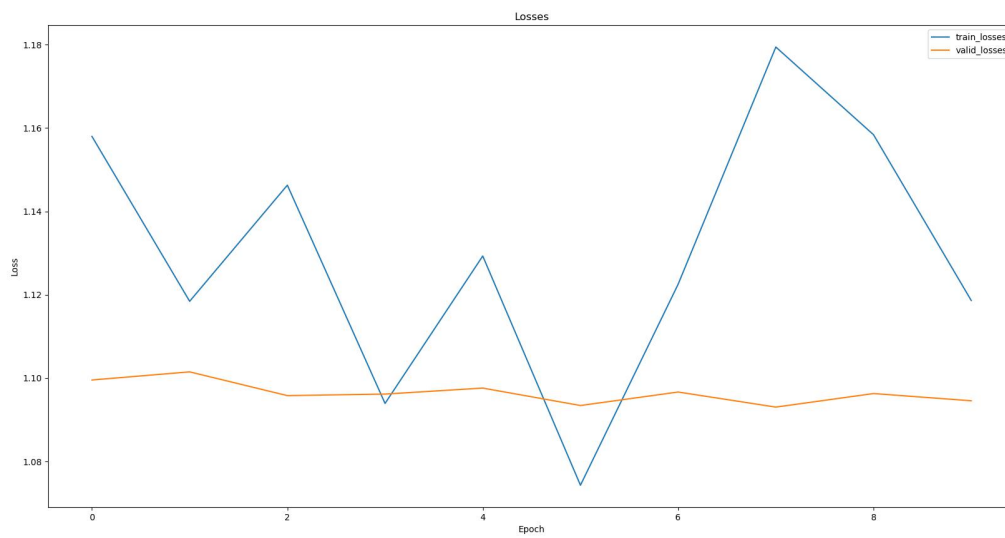


Figure 76. Loss curve at epoch 10 and optimizer “SGD”

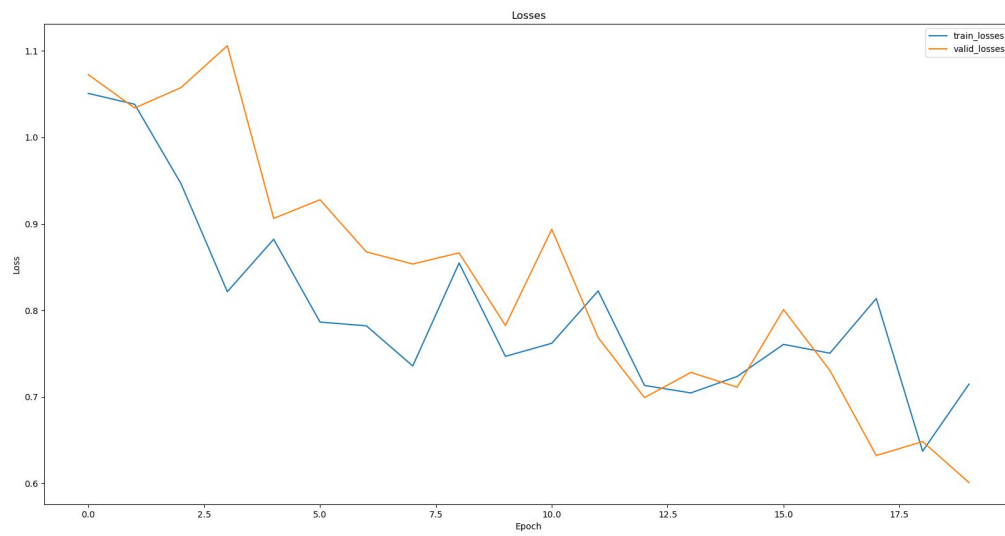


Figure 77. Loss curve at epoch 20 and optimizer “Adam”

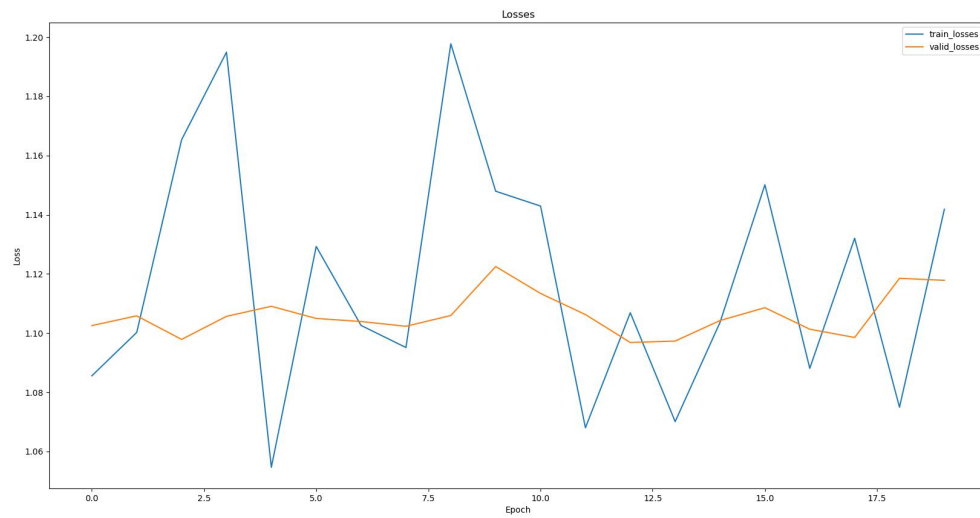


Figure 78. Loss curve at epoch 20 and optimizer “SGD”

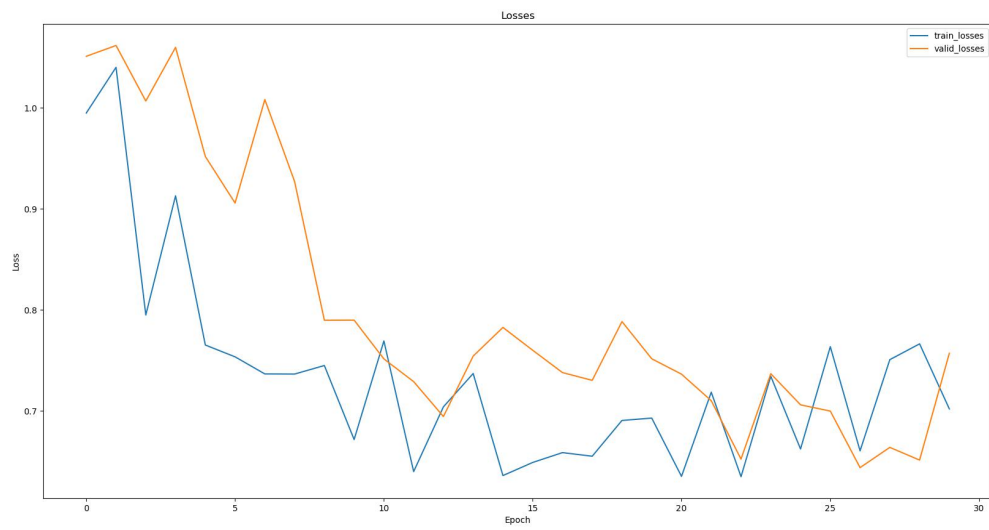


Figure 79. Loss curve at epoch 30 and optimizer “Adam”

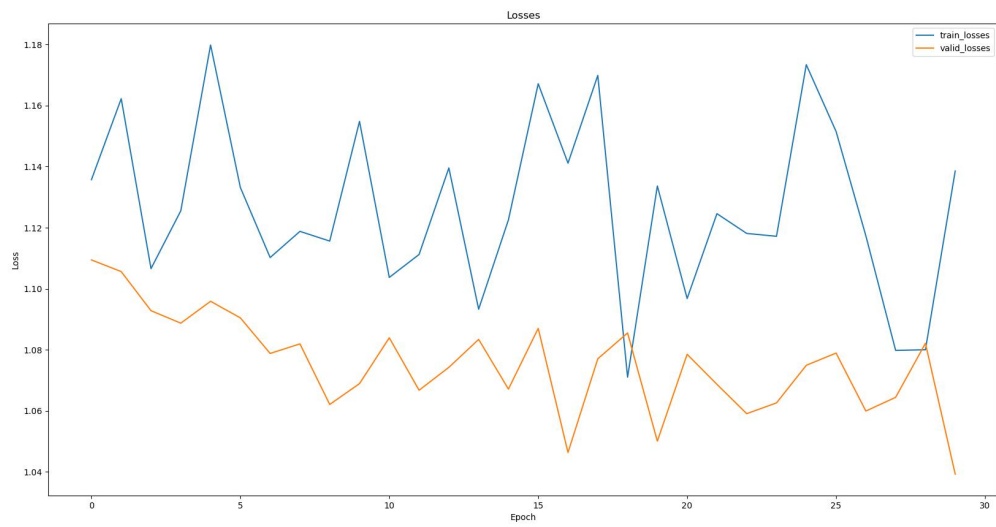


Figure 80. Loss curve at epoch 30 and optimizer “SGD”

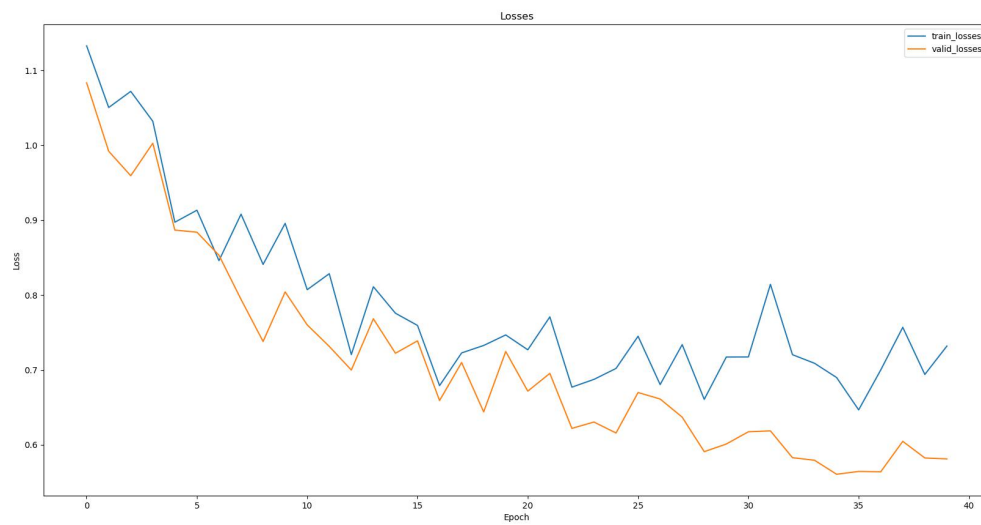


Figure 81. Loss curve at epoch 40 at optimizer “Adam”

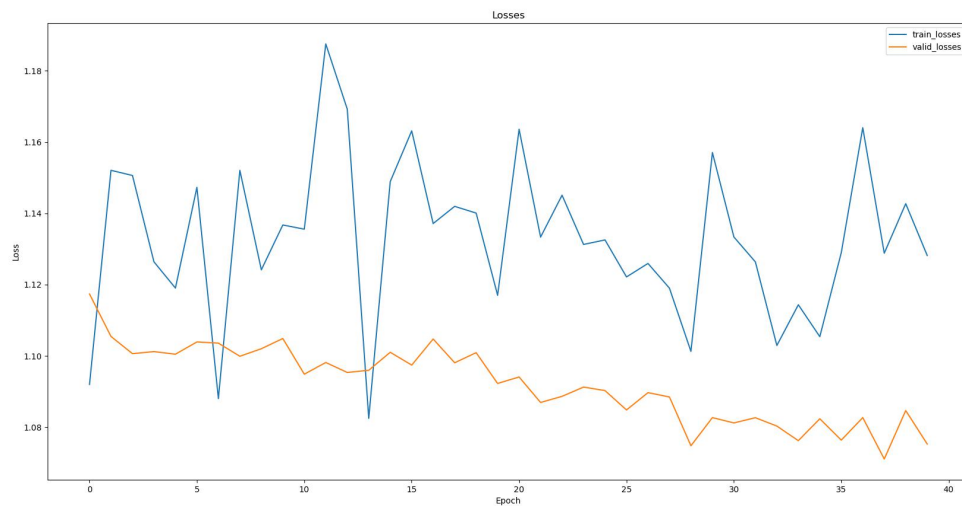


Figure 82. Loss curve at epoch 40 and optimizer “SGD”

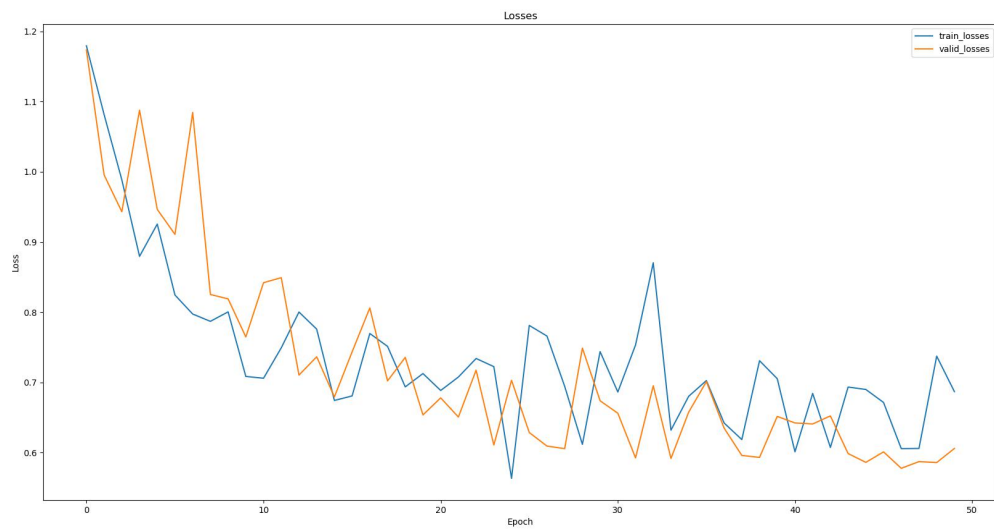


Figure 83. Loss curve at epoch 50 and optimizer “Adam”

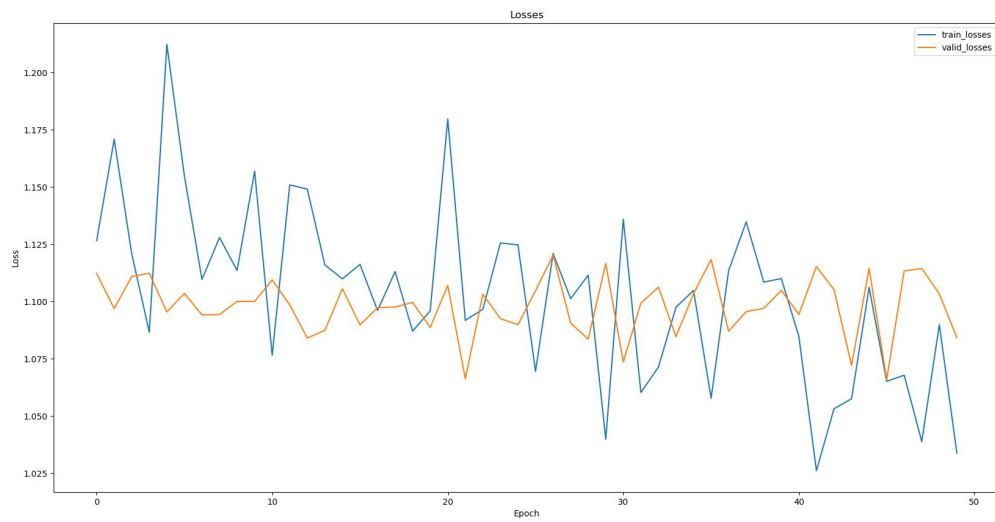


Figure 84. Loss curve at epoch 50 and optimizer “SGD”

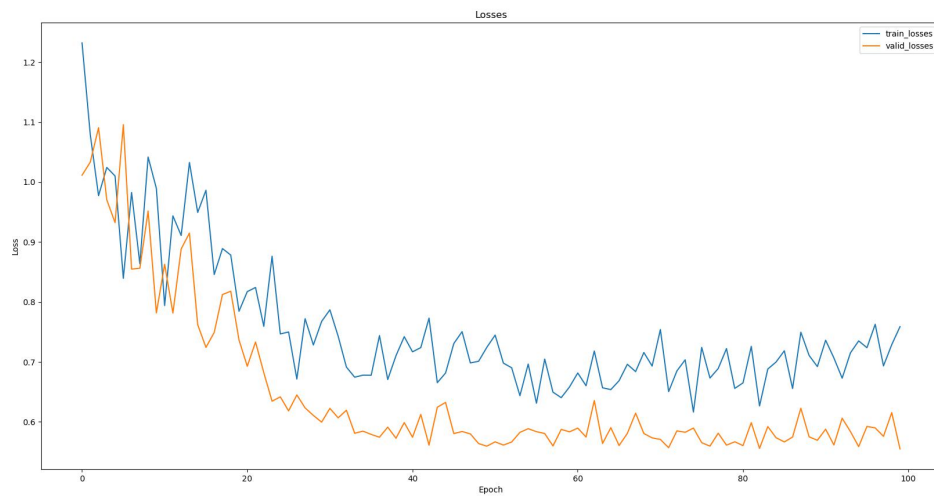


Figure 85. Loss curve at epoch 100 and optimizer “Adam”

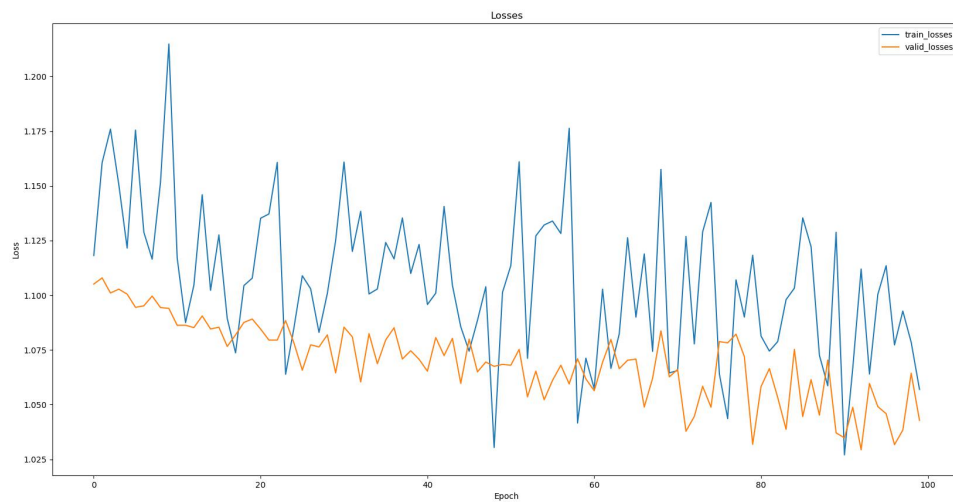


Figure 86. Loss curve at epoch 100 and optimizer “SGD”

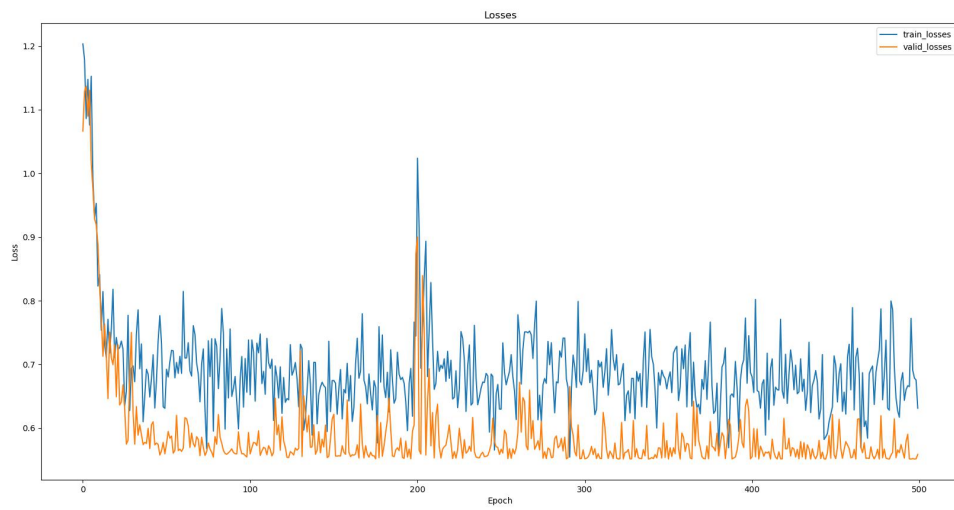


Figure 87. Loss curve at epoch 500 and optimizer “Adam”

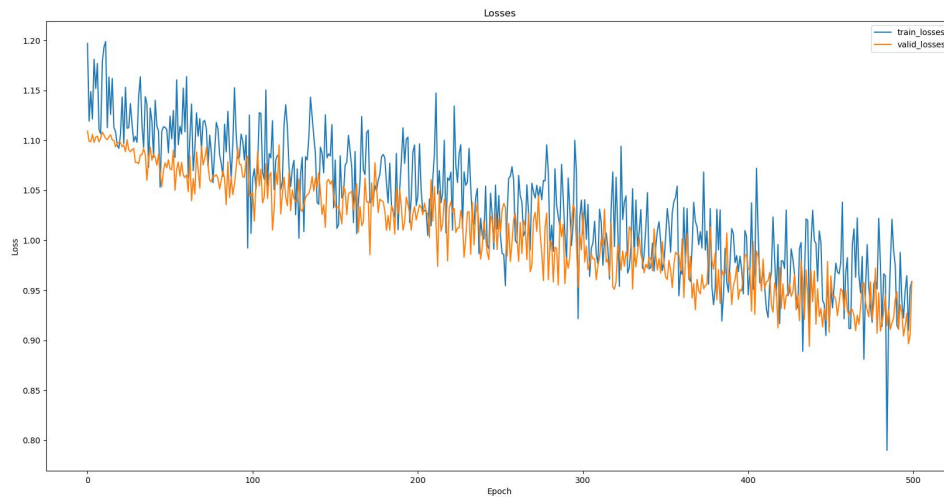


Figure 88. Loss curve at epoch 500 and optimizer “SGD”

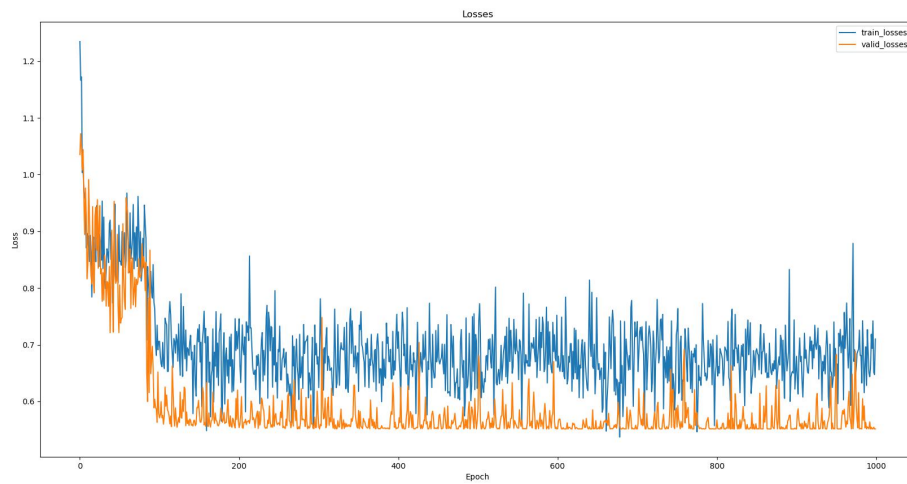


Figure 89. Loss curve at epoch 1000 and optimizer “Adam”

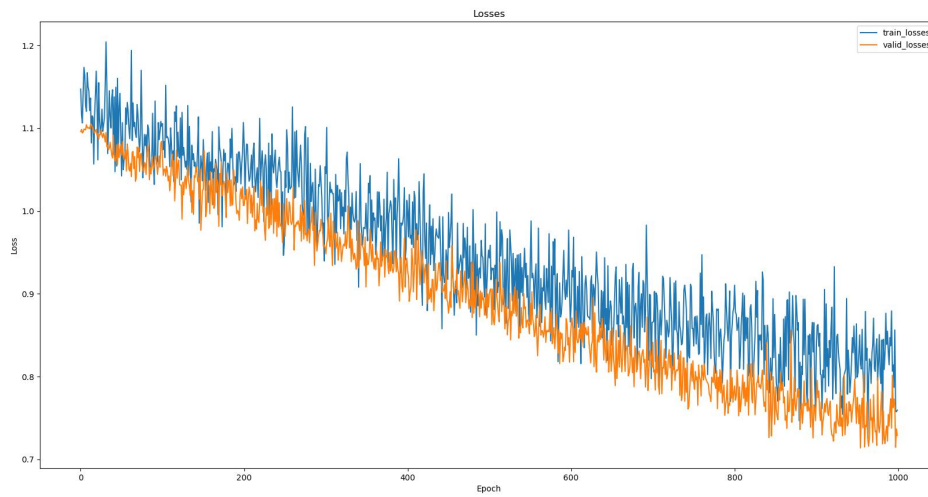


Figure 90. Loss curve at epoch 1000 and optimizer “SGD”

Appendix E: Shell script and run time email for transferred-ResNet 50

```
#!/bin/sh
#SBATCH --job-name=tamimadnan # create a short name for your job
#SBATCH --partition=Andromeda # partition
#SBATCH --nodes=1 # node count
#SBATCH --ntasks=2 # total number of tasks across all nodes
#SBATCH --cpus-per-task=16 # cpu-cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=30gb # memory per cpu-core (4G is default)
#SBATCH --time=48:00:00 # maximum time needed (HH:MM:SS)
#SBATCH --mail-type=begin # send email when job begins
#SBATCH --mail-type=end # send email when job ends
#SBATCH --mail-user=tadnan@uncc.edu

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

## the above is multithreaded job
module load pytorch

epochs=(10 20 30 40 50 100 500 1000)

optimizers=("Adam" "SGD") # add the optimizers to be used as an array

for epoch in "${epochs[@]}"
do
  for optimizer in "${optimizers[@]}" # add a loop for the optimizers
  do
    srun --ntasks 1 --cpus-per-task $SLURM_CPUS_PER_TASK python
    ResNet50_tranfered_pavement.py "$epoch" "$optimizer">
    output_epoch_optimizers_${epoch}_${optimizer}.txt & # launch each inner loop iteration as
    a separate task in the background
  done
done
wait # wait for all the tasks to complete before exiting
```

Slurm Job_id=5127899 Name=tamimadnan Ended, Run time 04:21:49, COMPLETED, ExitCode 0

 Inbox x

 slurm@uncc.edu
to me ▼

Wed, Jul 19, 7:16 PM ☆

Appendix F: Python code for classification in transferred-ResNet50

```
#!/usr/bin/env python
# coding: utf-8
#!/usr/bin/env python
# coding: utf-8
# License: BSD
# Author: Sasank Chilamkurthy [71]
# The code is modified as needed by the # researcher of this thesis.
# Therefore its not from only one resource
# the referenced resources are from [71 - [74]
from __future__ import print_function, division
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import random_split
import matplotlib.pyplot as plt
import time
import os
import copy
cudnn.benchmark = True
plt.ion() # interactive mode
from random import *
from tqdm.notebook import tqdm, trange
from time import sleep
from pathlib import Path
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import transforms, models
from torchvision.datasets import ImageFolder
from warnings import filterwarnings
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from tqdm import tqdm
import sys
filterwarnings('ignore')
device = torch.device("cpu")
print(device)
## codes for data augmentation
# define transformations to apply to the data
## Normalization converts the PIL image with a pixel range of [0, 255]
```

```

#to a PyTorch FloatTensor of shape (C, H, W) with a range [0.0, 1.0].
# Source for normalization:
https://www.geeksforgeeks.org/how-to-normalize-images-in-pytorch/.
# so this study randomly chooses 0.5 meand and standatdard devivation considering the
medium values between 0 and 1
train_trans = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.FiveCrop(224),
    transforms.Lambda(lambda crops: crops[0]),
    transforms.RandomHorizontalFlip(p=0.5), ## tamim: image will move left and right
    transforms.RandomVerticalFlip(p=0.5), ## tamim: image will come to eye vertically
    transforms.RandomRotation(degrees=(.5, 5)), ## very small rotation of the cracks
    transforms.ToTensor(),
    transforms.Normalize(0.5, 0.5)
])
test_trans = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(0.5, 0.5)
])
## Load data
from torchvision.datasets import ImageFolder
data = ImageFolder("../Data/Data_Structure(Annotated)", transform=train_trans , )
test_folder= ImageFolder("../Data/Data_Structure(Raw images)", transform=test_trans, )
## hyperparameters
batch_size = 32
num_classes = 3
learning_rate = 0.001
# read the epoch parameter from the command line / shellscript
num_epochs = int(sys.argv[1])
# read the optimizers from the command line / shellscript
optimizers = str(sys.argv[2])
# functions to show an image
def imshow(img):
    img = img / 2 + 0.5 # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
    plt.savefig(f"labels_{num_epochs}_{optimizers}.JPG")
# your code for training the model with the given epoch
print("Follwing classes are there : \n",data.classes)
print("data length:", len(data))
classes = ('Alligator Cracks', 'Longitudinal Cracks', 'Transverse Cracks')
##Splitting Data and Prepare Batches:
## Source: https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48
val_size = 40 ## Tamim:30% data for validation ##
train_size = len(data) - val_size

```

```

## To randomly split the images into training and testing, PyTorch provides random_split()
train_data, val_data = random_split(data,[train_size,val_size])
print(f'Length of Train Data : {len(train_data)}')    ## changed the folder names
print(f'Length of Validation Data : {len(val_data)}')
# Splitting train and validation data on batches
train_loader = torch.utils.data.DataLoader(train_data, shuffle=True, batch_size=batch_size)
## defined train data & val data
val_loader = torch.utils.data.DataLoader(val_data, shuffle=True, batch_size=batch_size)
test_loader = torch.utils.data.DataLoader(test_folder, shuffle=False, batch_size=batch_size)
# visualize images of a single batch
dataiter = iter(train_loader)
images, labels = next(dataiter)
# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
model = models.resnet50(pretrained=True).to(device)
model.eval().to(device)
# model.fc = nn.Linear(2048, 4) # original model has outputs for 1000 classes. Tamim
# changed it to 4
print(model)
# https://www.kaggle.com/code/pmigdal/transfer-learning-with-resnet-50-in-pytorch
# Freeze all layers except the last fc layer
for param in model.parameters():
    param.requires_grad = False
model.fc = nn.Sequential(
    nn.Linear(2048, 512), # reduce hidden units to 512
    nn.ReLU(inplace=True),
    nn.Linear(512, 3),
    nn.Softmax(dim=1),
    nn.Dropout(p=0.5))
# Print the modified ResNet50 architecture
print(model)
## Defining model optimizer and loss function
loss_fn = nn.CrossEntropyLoss()
if optimizers == 'Adam':
    opt = optim.Adam(model.parameters(), lr=1e-3)
elif optimizers == 'SGD':
    opt = optim.SGD(model.parameters(), lr=1e-3)
else:
    raise ValueError('Invalid optimizer name: {}'.format(optimizer_name))
def mean(l: list):
    return sum(l) / len(l)
def plot_losses_and_acc(train_losses, train_accuracies, valid_losses, valid_accuracies):
    fig, axes = plt.subplots(1, 2, figsize=(15, 10))
    axes[0].plot(train_losses, label='train_losses')
    axes[0].plot(valid_losses, label='valid_losses')

```



```

axes[0].set_title('Losses')
axes[0].legend()
axes[1].plot(train_accuracies, label='train accuracies')
axes[1].plot(valid_accuracies, label='valid accuracies')
axes[1].set_title('Accuracy')
axes[1].legend()
plt.savefig(f"learning_curve_{num_epochs}_{optimizers}.JPG")
def plot_losses(train_losses, valid_losses):
    fig, ax = plt.subplots(figsize=(20, 10))
    ax.plot(train_losses, label='train losses')
    ax.plot(valid_losses, label='valid losses')
    ax.set_title('Losses')
    ax.set_xlabel('Epoch')
    ax.set_ylabel('Loss')
    ax.legend()
    plt.savefig(f"learning_curve_{num_epochs}_{optimizers}.jpg")
def validate(model, valid_data, loss_fn):
    valid_losses, valid_accuracies = [], []
    model.eval()
    with torch.no_grad():
        for X_batch, y_batch in tqdm(valid_data, leave=False):
            X_batch, y_batch = X_batch.float(), y_batch.long()
            logits = model(X_batch)
            loss = loss_fn(logits, y_batch)
            valid_losses.append(loss.item())
            preds = torch.argmax(logits, axis=1)
            valid_accuracies.append(((preds == y_batch).sum() / len(preds)).item())
    return mean(valid_losses), mean(valid_accuracies)
def train(model, train_data, valid_data, loss_fn, opt, epochs):
    train_losses, valid_losses = [], []
    train_accuracies, valid_accuracies = [], []
    for epoch in tqdm(range(epochs)):
        train_loss = []
        train_acc = []
        model.train()
        for X_batch, y_batch in tqdm(train_data, leave=False):
            opt.zero_grad()
            X_batch, y_batch = X_batch.float(), y_batch.long()
            logits = model(X_batch)
            loss = loss_fn(logits, y_batch)
            train_loss.append(loss.item())
            pred = torch.argmax(logits, dim=1)
            train_acc.append(((pred == y_batch).sum() / len(pred)).item())
            loss.backward()
            opt.step()
        valid_loss, valid_accuracy = validate(model, valid_data, loss_fn)
        train_accuracies.append(mean(train_acc))

```

```

    train_losses.append(mean(train_loss))
    valid_losses.append(valid_loss)
    valid_accuracies.append(valid_accuracy)
    print(f'epoch: {epoch}: train_loss: {mean(train_losses)}, train_acc: {mean(train_acc)},
val_loss: {valid_loss}, val_acc: {valid_accuracy}')
    plot_losses(train_losses, valid_losses)
    return model, train_losses, train_accuracies, valid_losses, valid_accuracies
model, train_losses, train_accuracies, valid_losses, valid_accuracies = train(model,
train_loader, val_loader, loss_fn, opt, epoches=num_epochs)
# resource: #https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
## loss calculation
valid_loss, valid_acc = validate(model, test_loader, loss_fn)
print(valid_loss, valid_acc)
##Testing
# Set the random seed for reproducing the previous results.
seed = 100 #randomly chosen for arbitrary values
torch.manual_seed(seed)
np.random.seed(seed)
dataiter = iter(test_loader)
images, labels = next(dataiter)
# resource: #https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
# Save model
PATH = f'./New_{num_epochs}_{optimizers}.pth'
torch.save(model.state_dict(), PATH)
##Let us look at how the network performs on the whole dataset.
correct = 0
total = 0
# Calculate F1 score, precision, and recall for the whole dataset
## we need true labels and predicted labels
true_labels = []
predicted_labels = []
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = model(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        ## calculate the value for true labels and predicted labels
        true_labels.extend(labels.tolist())
        predicted_labels.extend(predicted.tolist())
print(f' Accuracy of the network on the test images: {100 * correct // total}%')
# using Accuracy metrics from Sklearn
print(f'F1 Score: {f1_score(true_labels, predicted_labels, average="weighted")}')

```

```

print(f'precision_score: {precision_score(true_labels, predicted_labels,
average="weighted"}}')
print(f'recall_total: {recall_score(true_labels, predicted_labels, average="weighted"}}')
# prepare to count predictions for each class on iphone image data
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}
# again no gradients needed because we already trained
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = model(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
                total_pred[classes[label]] += 1
# print accuracy for each class
for classname, correct_count in correct_pred.items():
    total_count = total_pred[classname]
    if total_count != 0:
        accuracy = 100 * float(correct_count) / total_count
        print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
    else:
        print(f'Accuracy for class: {classname:5s} is N/A (No predictions)')
## for Accuracy score, F1, precision and recall for each class
true_labels_class = [label for label, pred in zip(true_labels, predicted_labels) if
classes[pred] == classname]
predicted_labels_class = [pred for label, pred in zip(true_labels, predicted_labels) if
classes[pred] == classname]
print(f'F1 score: {f1_score(true_labels_class, predicted_labels_class,
average="weighted"}}')
print(f'precision_score: {precision_score(true_labels_class, predicted_labels_class,
average="weighted"}}')
print(f'recall_score: {recall_score(true_labels_class, predicted_labels_class,
average="weighted"}}')

```

Appendix G: Shell script and run time email for UNet for EdmCrack600 dataset

```
#!/bin/sh

#SBATCH --job-name=tamimadnan # create a short name for your job
#SBATCH --partition=Andromeda # partition
#SBATCH --nodes=1 # node count
#SBATCH --ntasks=2 # total number of tasks across all nodes
#SBATCH --cpus-per-task=16 # cpu-cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=30gb # memory per cpu-core (4G is default)
#SBATCH --time=48:00:00 # maximum time needed (HH:MM:SS)
#SBATCH --mail-type=begin # send email when job begins
#SBATCH --mail-type=end # send email when job ends
#SBATCH --mail-user=tadnan@uncc.edu

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

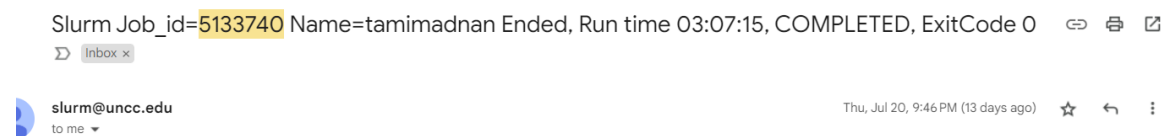
## the above is multithreaded job

module load pytorch

epochs=30

for epoch in "${epochs[@]}"
do
    srun --nodes=1 --ntasks=2 --cpus-per-task=$SLURM_CPUS_PER_TASK python
    UNet_thesis_EdmCrack.py "$epoch" > output_epoch_${epoch}.txt & # launch each inner
    loop iteration as a separate task in the background
done

wait # wait for all the tasks to complete before exiting
```



Appendix H: Python code for segmentation in modified-UNet for EdmCrack470 dataset

```
#!/usr/bin/env python
# coding: utf-8
# ## inspired code:
https://www.kaggle.com/code/gokulkarthik/image-segmentation-with-unet-pytorch/notebook.
I took UNet model from but,
# their data was integrated, so they splitted. But I didn't do that. I defined my dataset to load
my data for this model.
import os
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt
import os
from tqdm.notebook import tqdm
torch.cuda.is_available = lambda : False
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
train_images_dir = "./dataset_EdmCrack600_center_cropped/train/cropped_images_thesis"
train_masks_dir = "./dataset_EdmCrack600_center_cropped/train/cropped_masks_thesis"

val_images_dir = "./dataset_EdmCrack600_center_cropped/test/cropped_images_thesis"
val_masks_dir = "./dataset_EdmCrack600_center_cropped/test/cropped_masks_thesis"
class ImageMaskDataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.image_files = os.listdir(image_dir)
    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_name = self.image_files[idx]
        img_path = os.path.join(self.image_dir, img_name)
        mask_name = img_name.split('.')[0] + '.png'
        mask_path = os.path.join(self.mask_dir, mask_name)
```

```

image = Image.open(img_path).convert('RGB')
mask = Image.open(mask_path).convert('L')
if self.transform:
    image = self.transform(image)
    mask = self.transform(mask)
    # remove the first dimension of the mask tensor
    mask = mask.long() # convert the mask tensor to Long data type
    return image, mask
# define transformations to apply to the data
## Normalization converts the PIL image with a pixel range of [0, 255]
# to a PyTorch FloatTensor of shape (C, H, W) with a range [0.0, 1.0].
# Source: https://www.geeksforgeeks.org/how-to-normalize-images-in-pytorch/. so this study
# randomly chooses 0.5 mean and standard deviation considering the medium values
# between 0 and 1
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.GaussianBlur(kernel_size=(7, 13), sigma=(0.1, 0.2)), ## making the blur
conditions
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5))
])
transform_valid = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5))
])
# create dataset and dataloader for training data
train_dataset = ImageMaskDataset(train_images_dir, train_masks_dir, transform=transform)
print(len(train_dataset))
## use data loader to create train dataset
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
# create dataset and dataloader for validation data
val_dataset = ImageMaskDataset(val_images_dir, val_masks_dir,
transform=transform_valid)
print(len(val_dataset))
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False)
# get a batch of data from the train_loader
images, masks = next(iter(train_loader))
print(images.shape, masks.shape)
class UNet(nn.Module):
    def __init__(self, num_classes):
        super(UNet, self).__init__()
        self.num_classes = num_classes
        self.contracting_11 = self.conv_block(in_channels=3, out_channels=64)
        self.contracting_12 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_21 = self.conv_block(in_channels=64, out_channels=128)
        self.contracting_22 = nn.MaxPool2d(kernel_size=2, stride=2)

```

```

self.contracting_31 = self.conv_block(in_channels=128, out_channels=256)
self.contracting_32 = nn.MaxPool2d(kernel_size=2, stride=2)
self.contracting_41 = self.conv_block(in_channels=256, out_channels=512)
self.contracting_42 = nn.MaxPool2d(kernel_size=2, stride=2)
self.middle = self.conv_block(in_channels=512, out_channels=1024)
self.expansive_11 = nn.ConvTranspose2d(in_channels=1024, out_channels=512,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_12 = self.conv_block(in_channels=1024, out_channels=512)
self.expansive_21 = nn.ConvTranspose2d(in_channels=512, out_channels=256,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_22 = self.conv_block(in_channels=512, out_channels=256)
self.expansive_31 = nn.ConvTranspose2d(in_channels=256, out_channels=128,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_32 = self.conv_block(in_channels=256, out_channels=128)
self.expansive_41 = nn.ConvTranspose2d(in_channels=128, out_channels=64,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_42 = self.conv_block(in_channels=128, out_channels=64)
self.output = nn.Conv2d(in_channels=64, out_channels=num_classes, kernel_size=3,
stride=1, padding=1)

def conv_block(self, in_channels, out_channels):
    block = nn.Sequential(nn.Conv2d(in_channels=in_channels,
out_channels=out_channels, kernel_size=3, stride=1, padding=1),
nn.ReLU(),
nn.BatchNorm2d(num_features=out_channels),
nn.Conv2d(in_channels=out_channels, out_channels=out_channels,
kernel_size=3, stride=1, padding=1),
nn.ReLU(),
nn.BatchNorm2d(num_features=out_channels))

    return block

def forward(self, X):
    contracting_11_out = self.contracting_11(X) # [-1, 64, 256, 256]
    contracting_12_out = self.contracting_12(contracting_11_out) # [-1, 64, 128, 128]
    contracting_21_out = self.contracting_21(contracting_12_out) # [-1, 128, 128, 128]
    contracting_22_out = self.contracting_22(contracting_21_out) # [-1, 128, 64, 64]
    contracting_31_out = self.contracting_31(contracting_22_out) # [-1, 256, 64, 64]
    contracting_32_out = self.contracting_32(contracting_31_out) # [-1, 256, 32, 32]
    contracting_41_out = self.contracting_41(contracting_32_out) # [-1, 512, 32, 32]
    contracting_42_out = self.contracting_42(contracting_41_out) # [-1, 512, 16, 16]
    middle_out = self.middle(contracting_42_out) # [-1, 1024, 16, 16]
    expansive_11_out = self.expansive_11(middle_out) # [-1, 512, 32, 32]
    expansive_12_out = self.expansive_12(torch.cat((expansive_11_out,
contracting_41_out), dim=1)) # [-1, 1024, 32, 32] -> [-1, 512, 32, 32]
    expansive_21_out = self.expansive_21(expansive_12_out) # [-1, 256, 64, 64]
    expansive_22_out = self.expansive_22(torch.cat((expansive_21_out,
contracting_31_out), dim=1)) # [-1, 512, 64, 64] -> [-1, 256, 64, 64]
    expansive_31_out = self.expansive_31(expansive_22_out) # [-1, 128, 128, 128]

```

```

        expansive_32_out = self.expansive_32(torch.cat((expansive_31_out,
contracting_21_out), dim=1)) # [-1, 256, 128, 128] -> [-1, 128, 128, 128]
        expansive_41_out = self.expansive_41(expansive_32_out) # [-1, 64, 256, 256]
        expansive_42_out = self.expansive_42(torch.cat((expansive_41_out,
contracting_11_out), dim=1)) # [-1, 128, 256, 256] -> [-1, 64, 256, 256]
        output_out = self.output(expansive_42_out) # [-1, num_classes, 256, 256]
        return output_out
num_classes = 2
model = UNet(num_classes=num_classes)
import sys
epochs = int(sys.argv[1])
lr = 0.001
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
train_losses = []
val_losses = []
best_val_loss = float('inf')
for epoch in tqdm(range(epochs)):
    epoch_train_loss = 0
    epoch_val_loss = 0
    # Train loop
    model.train()
    for X, Y in tqdm(train_loader, total=len(train_loader), leave=False):
        X, Y = X.to(device), Y.to(device)
        Y = torch.argmax(Y, dim=1)
        optimizer.zero_grad()
        Y_pred = model(X)
        loss = criterion(Y_pred, Y)
        loss.backward()
        optimizer.step()
        epoch_train_loss += loss.item()
    train_losses.append(epoch_train_loss/len(train_loader))
    # Validation loop
    model.eval()
    with torch.no_grad():
        for X_val, Y_val in tqdm(val_loader, total=len(val_loader), leave=False):
            X_val, Y_val = X_val.to(device), Y_val.to(device)
            Y_val = torch.argmax(Y_val, dim=1)
            Y_val_pred = model(X_val)
            val_loss = criterion(Y_val_pred, Y_val)
            epoch_val_loss += val_loss.item()
        val_losses.append(epoch_val_loss/len(val_loader))
    # Save best model based on validation loss
    if val_losses[-1] < best_val_loss:
        best_val_loss = val_losses[-1]
        torch.save(model.state_dict(), f"best_model_{epochs}.pt")
    # Print and update progress bar

```



```

    tqdm.write(f"Epoch {epoch+1}/{epochs} - Train loss: {train_losses[-1]:.4f} - Val loss:
{val_losses[-1]:.4f}")
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.savefig(f"learning_curve_{epochs}.png")
model_name = f'U-Net_{epochs}.pth'
torch.save(model.state_dict(), model_name)
model_path = f"U-Net_{epochs}.pth"
model_ = UNet(num_classes=num_classes).to(device)
model_.load_state_dict(torch.load(model_path))
# create a folder to save the images
os.makedirs(f'results_{epochs}', exist_ok=True)
# Set the random seed for reproducing the previous results.
seed = 40
torch.manual_seed(seed)
np.random.seed(seed)
#### Test the model now
with torch.no_grad():
    for X_val, Y_val in tqdm(val_loader, total=len(val_loader), leave=False):
        X_val = X_val.to(device)
        Y_pred_val = model(X_val)
        Y_pred_val = torch.sigmoid(Y_pred_val)
        # print(gt_pred_val) ## check if needed
        predicted_masks = Y_pred_val.cpu().numpy()
        # Threshold predicted masks to obtain binary masks
        threshold = 0.81 ## as needed
        predicted_masks = np.where(predicted_masks > threshold, 1, 0)
        # Intersection and Union calculation
        ## ref:
https://medium.com/mlearning-ai/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f
        intersection = np.sum(predicted_masks * Y_val.cpu().numpy())
        intersection = np.abs(intersection)
        union = np.sum(predicted_masks) + np.sum(Y_val.cpu().numpy()) - intersection
        union = np.abs(union)
        # total of predicted mask and ground truth to get Precision and Recall calculation
        total_pixel_pred = np.sum(predicted_masks)
        total_pixel_pred = np.abs(total_pixel_pred)
        # print('total_pixel_pred:', total_pixel_pred) ## only for check

        total_ground_truth = np.sum(Y_val.cpu().numpy())
        total_ground_truth = np.abs(total_ground_truth)
        # print('total_ground_truth:', total_ground_truth) ## only check
        # Plot the original image, target, and predicted segmentation mask

```

```

for i in range(len(predicted_masks)):
    fig, axs = plt.subplots(1, 3, figsize=(10,10))
    axs[0].imshow(X_val[i].cpu().numpy().transpose(1,2,0), cmap=None)
    axs[0].set_title("Original Image")
    axs[1].imshow(Y_val[i][0].cpu().numpy(), cmap='gray')
    axs[1].set_title("Target Mask")
    axs[2].imshow(predicted_masks[i][0], cmap='gray')
    axs[2].set_title("Predicted Mask")
    # save the figure as an image
    plt.savefig(f'results_{epochs}/image_{i}.jpg')
    # close the figure to free up memory
    plt.close(fig)
print('intersection:', intersection) ## check
print('union:', union) ## check
# IoU calculation
IoU = intersection / (union + 1e-7)
## precision calculation
precision = intersection / (total_pixel_pred + 1e-7)
## print("precision:", precision) ## only for check
## recall calculation
recall = intersection / (total_ground_truth + 1e-7)
## print("recall:", recall) ## only for check
# Calculate F1 score
## ref: https://deeptai.org/machine-learning-glossary-and-terms/f-score
F1 = (2 * precision * recall) / (precision + recall + 1e-7)
print(f'IoU: {IoU:.4f}')
print(f'precision: {precision:.4f}')
print(f'recall: {recall:.4f}')
print(f'F1: {F1:.4f}')

```

Appendix I: Python code for segmentation in modified-UNet for CRACKTREE206 dataset

```
#!/usr/bin/env python
# coding: utf-8

# ## inspired code:
# https://www.kaggle.com/code/gokulkarthik/image-segmentation-with-unet-pytorch/notebook.
# I took UNet model from but,
# their data was integrated, so they splitted. But I didn't do that. I defined my dataset to load
# my data for this model.
import os
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt
import os
from tqdm.notebook import tqdm
torch.cuda.is_available = lambda : False
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device

train_images_dir = "./CRACKTREE206/train/images"
train_masks_dir = "./CRACKTREE206/train/masks"
val_images_dir = "./CRACKTREE206/test/images"
val_masks_dir = "./CRACKTREE206/test/masks"
class ImageMaskDataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.image_files = os.listdir(image_dir)
    def __len__(self):
        return len(self.image_files)
    def __getitem__(self, idx):
        img_name = self.image_files[idx]
        img_path = os.path.join(self.image_dir, img_name)
        mask_name = img_name.split('.')[0] + '.jpg'
        mask_path = os.path.join(self.mask_dir, mask_name)
```

```

image = Image.open(img_path).convert('RGB')
mask = Image.open(mask_path).convert('L')
if self.transform:
    image = self.transform(image)
    mask = self.transform(mask)
    # remove the first dimension of the mask tensor
    mask = mask.long() # convert the mask tensor to Long data type
    return image, mask
# define transformations to apply to the data
## Normalization converts the PIL image with a pixel range of [0, 255]
# to a PyTorch FloatTensor of shape (C, H, W) with a range [0.0, 1.0].
# Source: https://www.geeksforgeeks.org/how-to-normalize-images-in-pytorch/. so this study
# randomly chooses 0.5 meand and standatdard deviation considering the medium values
# between 0 and 1
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.GaussianBlur(kernel_size=(7, 13), sigma=(0.1, 0.2)), ## making the blur
conditions
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5))
])
transform_valid = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize((0.5), (0.5))
])
# create dataset and dataloader for training data
train_dataset = ImageMaskDataset(train_images_dir, train_masks_dir, transform=transform)
print(len(train_dataset))
## use data lodaer to create train dataset
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
# create dataset and dataloader for validation data
val_dataset = ImageMaskDataset(val_images_dir, val_masks_dir,
transform=transform_valid)
print(len(val_dataset))
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False)
# get a batch of data from the train_loader
images, masks = next(iter(train_loader))
print(images.shape, masks.shape)
class UNet(nn.Module):
    def __init__(self, num_classes):
        super(UNet, self).__init__()
        self.num_classes = num_classes
        self.contracting_11 = self.conv_block(in_channels=3, out_channels=64)
        self.contracting_12 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_21 = self.conv_block(in_channels=64, out_channels=128)
        self.contracting_22 = nn.MaxPool2d(kernel_size=2, stride=2)

```

```

self.contracting_31 = self.conv_block(in_channels=128, out_channels=256)
self.contracting_32 = nn.MaxPool2d(kernel_size=2, stride=2)
self.contracting_41 = self.conv_block(in_channels=256, out_channels=512)
self.contracting_42 = nn.MaxPool2d(kernel_size=2, stride=2)
self.middle = self.conv_block(in_channels=512, out_channels=1024)
self.expansive_11 = nn.ConvTranspose2d(in_channels=1024, out_channels=512,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_12 = self.conv_block(in_channels=1024, out_channels=512)
self.expansive_21 = nn.ConvTranspose2d(in_channels=512, out_channels=256,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_22 = self.conv_block(in_channels=512, out_channels=256)
self.expansive_31 = nn.ConvTranspose2d(in_channels=256, out_channels=128,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_32 = self.conv_block(in_channels=256, out_channels=128)
self.expansive_41 = nn.ConvTranspose2d(in_channels=128, out_channels=64,
kernel_size=3, stride=2, padding=1, output_padding=1)
self.expansive_42 = self.conv_block(in_channels=128, out_channels=64)
self.output = nn.Conv2d(in_channels=64, out_channels=num_classes, kernel_size=3,
stride=1, padding=1)

def conv_block(self, in_channels, out_channels):
    block = nn.Sequential(nn.Conv2d(in_channels=in_channels,
out_channels=out_channels, kernel_size=3, stride=1, padding=1),
nn.ReLU(),
nn.BatchNorm2d(num_features=out_channels),
nn.Conv2d(in_channels=out_channels, out_channels=out_channels,
kernel_size=3, stride=1, padding=1),
nn.ReLU(),
nn.BatchNorm2d(num_features=out_channels))

    return block

def forward(self, X):
    contracting_11_out = self.contracting_11(X) # [-1, 64, 256, 256]
    contracting_12_out = self.contracting_12(contracting_11_out) # [-1, 64, 128, 128]
    contracting_21_out = self.contracting_21(contracting_12_out) # [-1, 128, 128, 128]
    contracting_22_out = self.contracting_22(contracting_21_out) # [-1, 128, 64, 64]
    contracting_31_out = self.contracting_31(contracting_22_out) # [-1, 256, 64, 64]
    contracting_32_out = self.contracting_32(contracting_31_out) # [-1, 256, 32, 32]
    contracting_41_out = self.contracting_41(contracting_32_out) # [-1, 512, 32, 32]
    contracting_42_out = self.contracting_42(contracting_41_out) # [-1, 512, 16, 16]
    middle_out = self.middle(contracting_42_out) # [-1, 1024, 16, 16]
    expansive_11_out = self.expansive_11(middle_out) # [-1, 512, 32, 32]
    expansive_12_out = self.expansive_12(torch.cat((expansive_11_out,
contracting_41_out), dim=1)) # [-1, 1024, 32, 32] -> [-1, 512, 32, 32]
    expansive_21_out = self.expansive_21(expansive_12_out) # [-1, 256, 64, 64]
    expansive_22_out = self.expansive_22(torch.cat((expansive_21_out,
contracting_31_out), dim=1)) # [-1, 512, 64, 64] -> [-1, 256, 64, 64]
    expansive_31_out = self.expansive_31(expansive_22_out) # [-1, 128, 128, 128]

```

```

        expansive_32_out = self.expansive_32(torch.cat((expansive_31_out,
contracting_21_out), dim=1)) # [-1, 256, 128, 128] -> [-1, 128, 128, 128]
        expansive_41_out = self.expansive_41(expansive_32_out) # [-1, 64, 256, 256]
        expansive_42_out = self.expansive_42(torch.cat((expansive_41_out,
contracting_11_out), dim=1)) # [-1, 128, 256, 256] -> [-1, 64, 256, 256]
        output_out = self.output(expansive_42_out) # [-1, num_classes, 256, 256]
        return output_out
num_classes = 2
model = UNet(num_classes=num_classes)
import sys
epochs = int(sys.argv[1])
lr = 0.001
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
train_losses = []
val_losses = []
best_val_loss = float('inf')
for epoch in tqdm(range(epochs)):
    epoch_train_loss = 0
    epoch_val_loss = 0
    # Train loop
    model.train()
    for X, Y in tqdm(train_loader, total=len(train_loader), leave=False):
        X, Y = X.to(device), Y.to(device)
        Y = torch.argmax(Y, dim=1)
        optimizer.zero_grad()
        Y_pred = model(X)
        loss = criterion(Y_pred, Y)
        loss.backward()
        optimizer.step()
        epoch_train_loss += loss.item()
    train_losses.append(epoch_train_loss/len(train_loader))
    # Validation loop
    model.eval()
    with torch.no_grad():
        for X_val, Y_val in tqdm(val_loader, total=len(val_loader), leave=False):
            X_val, Y_val = X_val.to(device), Y_val.to(device)
            Y_val = torch.argmax(Y_val, dim=1)
            Y_val_pred = model(X_val)
            val_loss = criterion(Y_val_pred, Y_val)
            epoch_val_loss += val_loss.item()
        val_losses.append(epoch_val_loss/len(val_loader))
    # Save best model based on validation loss
    if val_losses[-1] < best_val_loss:
        best_val_loss = val_losses[-1]
        torch.save(model.state_dict(), f"best_model_{epochs}.pt")
    # Print and update progress bar

```

```

    tqdm.write(f"Epoch {epoch+1}/{epochs} - Train loss: {train_losses[-1]:.4f} - Val loss:
{val_losses[-1]:.4f}")
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.savefig(f"learning_curve_{epochs}.png")
model_name = f'U-Net_{epochs}.pth'
torch.save(model.state_dict(), model_name)
model_path = f"U-Net_{epochs}.pth"
model_ = UNet(num_classes=num_classes).to(device)
model_.load_state_dict(torch.load(model_path))
# create a folder to save the images
os.makedirs(f'results_{epochs}', exist_ok=True)
# Set the random seed for reproducing the previous results.
seed = 400
torch.manual_seed(seed)
np.random.seed(seed)
#### Test the model now
with torch.no_grad():
    for X_val, Y_val in tqdm(val_loader, total=len(val_loader), leave=False):
        X_val = X_val.to(device)
        Y_pred_val = model(X_val)
        Y_pred_val = torch.sigmoid(Y_pred_val)
        # print(gt_pred_val) ## check if needed
        predicted_masks = Y_pred_val.cpu().numpy()
        # Threshold predicted masks to obtain binary masks
        threshold = 0.66 ## as needed
        predicted_masks = np.where(predicted_masks > threshold, 1, 0)
        # Intersection and Union calculation
        ## ref:
https://medium.com/mlearning-ai/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f
        intersection = np.sum(predicted_masks * Y_val.cpu().numpy())
        intersection = np.abs(intersection)
        union = np.sum(predicted_masks) + np.sum(Y_val.cpu().numpy()) - intersection
        union = np.abs(union)
        # total of predicted mask and ground truth to get Precision and Recall calculation
        total_pixel_pred = np.sum(predicted_masks)
        total_pixel_pred = np.abs(total_pixel_pred)
        # print('total_pixel_pred:', total_pixel_pred) ## only for check
        total_ground_truth = np.sum(Y_val.cpu().numpy())
        total_ground_truth = np.abs(total_ground_truth)
        # print('total_ground_truth:', total_ground_truth) ## only check

        # Plot the original image, target, and predicted segmentation mask

```

```

for i in range(len(predicted_masks)):
    fig, axs = plt.subplots(1, 3, figsize=(10,10))
    axs[0].imshow(X_val[i].cpu().numpy().transpose(1,2,0), cmap=None)
    axs[0].set_title("Original Image")
    axs[1].imshow(Y_val[i][0].cpu().numpy(), cmap='gray')
    axs[1].set_title("Target Mask")
    axs[2].imshow(predicted_masks[i][0], cmap='gray')
    axs[2].set_title("Predicted Mask")
    # save the figure as an image
    plt.savefig(f'results_{epochs}/image_{i}.jpg')
    # close the figure to free up memory
    plt.close(fig)
print('intersection:', intersection) ## check
print('union:', union) ## check
# IoU calculation
IoU = intersection / (union + 1e-7)
## precision calculation
precision = intersection / (total_pixel_pred + 1e-7)
## print("precision:", precision) ## only for check
## recall calculation
recall = intersection / (total_ground_truth + 1e-7)
## print("recall:", recall) ## only for check
# Calculate F1 score
## ref: https://deeptai.org/machine-learning-glossary-and-terms/f-score
F1 = (2 * precision * recall) / (precision + recall + 1e-7)
print(f'IoU: {IoU:.4f}')
print(f'precision: {precision:.4f}')
print(f'recall: {recall:.4f}')
print(f'F1: {F1:.4f}')

```


Appendix J: Shell script and run time email for UNet for CRACKTREE260 dataset.

```
#!/bin/sh
#SBATCH --job-name=tamimadnan # create a short name for your job
#SBATCH --partition=Andromeda # partition
#SBATCH --nodes=1 # node count
#SBATCH --ntasks=2 # total number of tasks across all nodes
#SBATCH --cpus-per-task=16 # cpu-cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=30gb # memory per cpu-core (4G is default)
#SBATCH --time=48:00:00 # maximum time needed (HH:MM:SS)
#SBATCH --mail-type=begin # send email when job begins
#SBATCH --mail-type=end # send email when job ends
#SBATCH --mail-user=tadnan@uncc.edu

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

## the above is multithreaded job


module load pytorch


epochs=30

for epoch in "${epochs[@]}"
do
    srun --nodes=1 --ntasks=2 --cpus-per-task=$SLURM_CPUS_PER_TASK python
    UNet_thesis_CRACKTREE.py "$epoch" > output_epoch_${epoch}.txt & # launch each
    inner loop iteration as a separate task in the background
done

wait # wait for all the tasks to complete before exiting
```

Slurm Job_id=5129464 Name=tamimadnan Ended, Run time 01:22:29, COMPLETED, ExitCode 0

 Inbox x

 slurm@uncc.edu
to me ▼

Thu, Jul 20, 8:08AM (13 days ago)

Appendix K: Link of codes and datasets for classification, segmentation in GitHub and Google Drive

Drive:

https://drive.google.com/drive/folders/1OVIw-WSsvaM3axqjB-_iQCk39oytVNII?usp=drive_link

GitHub: <https://github.com/tamimdnan6240/Master-s-Thesis/tree/main>