DEEP LEARNING-BASED VISUAL CRACK DETECTION USING GOOGLE STREET VIEW IMAGES

Mohsen Maniat

DEEP LEARNING-BASED VISUAL CRACK DETECTION USING GOOGLE STREET VIEW IMAGES

by

Mohsen Maniat

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Engineering

The University of Memphis

December 2019

Acknowledgements

I sincerely appreciate all the supports that I have received during the completion

of this dissertation from Civil Engineering Department and Herff College of Engineering at the University of Memphis. I especially would like to thank my main advisor Dr. Charles Camp who his continuous support and invaluable mentoring. He gave me the freedom to explore and always encouraged me to be creative. I am very grateful for having the chance to be his student. I would like to express my sincere appreciation to Dr. Roger Meier for his helpful suggestions and comments to improve this research. I would like to thank Dr. Shahram Pezeshk, Dr. Masoud Sanayei, Dr. Mihalis Golias, and Dr. Adel Abdelnaby for their help and advice. I would like to thank my good friends Nima Nazemi and Mohammad Farshchin for supporting me whenever I needed it. I Also like to thank Christie Moore and Melish Kayastha for their help in this study.

Abstract

The need for developing an economical and efficient quality assessment system for pavement motivates this study to take advantage of available new technologies and provide a novel approach to address this need. In this study, the utility of using Google Street View (GSV) for evaluating the quality of pavement is investigated. GSV is a technology featured in Google Maps and Google Earth that provides interactive panoramas along many streets throughout the world. This technology provides a large data set of pavement images that can be used for pavement evaluation. Advanced deep learning algorithms are utilized to automate the pavement assessment process of these GSV images. These algorithms autonomously learn to find the important features in a data set to perform a particular task. A convolutional neural network (CNN) is one of the deep learning algorithms that has been shown to be very effective in learning from digital images. Several CNNs are used in this study to perform image classification on GSV pavement images. Training an effective CNN with many learning parameters requires a large image data set. To provide the required data for training a CNN, a large number of pavement images are extracted from GSV are then divided to smaller image patches to form a larger data set. Each image patch is visually classified into different categories of pavement cracks based on the standard practice. A comparative study of pavement quality assessment is conducted between the results of the CNN classified images patches obtained from GSV and those from a sophisticated commercial visual inspection company. The result of the comparison indicates the feasibility and effectiveness of using GSV images for pavement evaluation. An effective CNN is designed and trained on the image data set to automate the crack detection process. The trained network is then tested on a new data set. The results of this study show that the designed CNN is effective in classifying the pavement images into different defined crack categories.

Table of Contents

List of Figures

1. Chapter 1

Introduction

It is crucial to monitor and inspect infrastructure systems to ensure safety and reduce the maintenance cost. Roads are among the most visible and familiar forms of infrastructure with over four million miles crisscrossing the United States. Based on 2017 Infrastructure Report Card, one out of every five miles of highway pavement is in poor condition and have a major and increasing backlog of rehabilitation needs [1]. The first step in improving the current condition is to provide a comprehensive assessment of the roads. This assessment would help the decision makers to effectively plan short and long-term goals. Performing a large-scale monitoring and assessment on roads with conventional methods is a time-consuming process and requires large investments of time and money. Thus, it is crucial to invest in developing assessment systems that can be applied efficiently on a large scale. In this study, a novel, time efficient, and economic approach is proposed for pavement assessment.

Among different nondestructive evaluation (NDE) techniques, visual inspection is the most common method and often serves as a baseline to confirm observations obtained from other NDE methods [2,3]. In this method, a trained inspector looks over a system using naked eye to search for flaws. The inspector may be equipped with measuring devices to evaluate the severity of the flaws (see Figure 1). A visual inspection of pavement surface can provide valuable data that could be utilized to estimate the current and future pavement performance and to determine and prioritize pavement maintenance and rehabilitation necessities [4]. Although visual inspection conducted by humans is among one of the lowest-cost and most reliable NDE methods, performing this type of inspection for large systems such as roads has several drawbacks. The U.S.

Federal Highway Administration conducted a comprehensive study of reliability of visual inspection of highway bridges which discovered a great discrepancy in the results of several inspectors for the same structure [2]. Although, to some extent, this discrepancy is attributed to negligence or improper training of inspectors, there are many factors that cannot be overcome by humans in the current visual inspection process. Also, visual inspection poses safety risks to both inspectors and the general public [5]. For example, when inspecting highway infrastructure inspectors are often exposed to traffic or dangerous climbing scenarios (see Figure 2) which also can be a distraction to passing drivers. Furthermore, visual inspection of large areas such as roads is a time-consuming and expensive process. These factors have led many researches to propose and develop various partially automated systems for visual inspection. The automation can happen in both image acquisition and image processing.

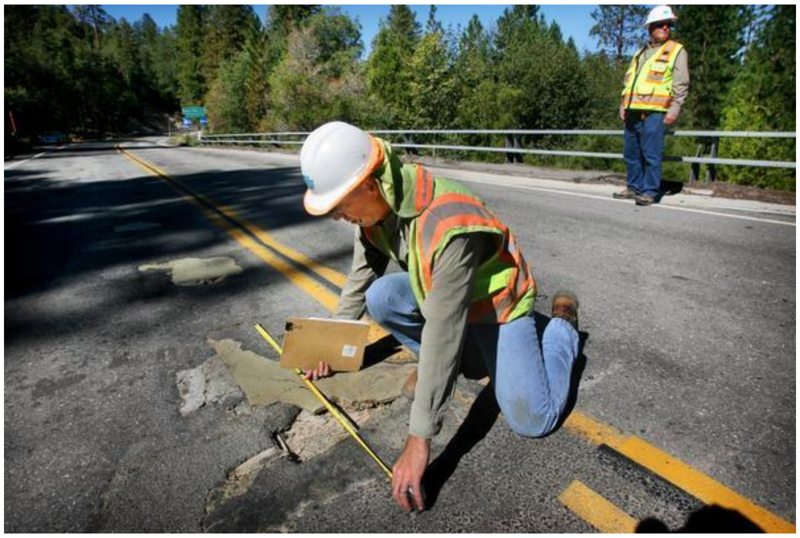


Figure 1. Human conducted visual inspection with simple equipment [6].

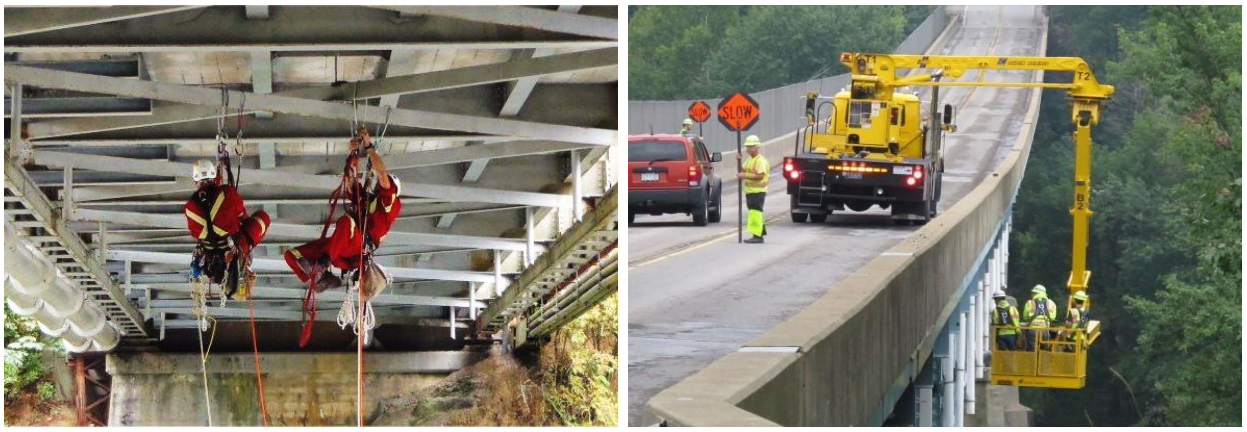


Figure 2. Human conducted visual inspection and safety risks [7,8].

1.1 Literature Review on Automation in Image Acquisition

Advancements and availability of optical devices and vision-based sensing technology gained great attention of many researchers in damage detection [9,10]. Typically, image acquisition systems involve using a camera to capture images of the surface of an object. The camera sensor may work in the visible light or beyond (vision-based sensing is usually based on visible light). Jahanshahi et al. [11] developed a visual monitoring system by mounting several inexpensive digital cameras (capable of zooming and rotation in three directions) to collect images (see Figure 3). This system allows an inspector to compare the current situation of the structure with the results of previous inspections, which would help to evaluate the changes of the structure at different locations. They also provide a panoramic reconstruction of various view cases by using an image stitching algorithm.

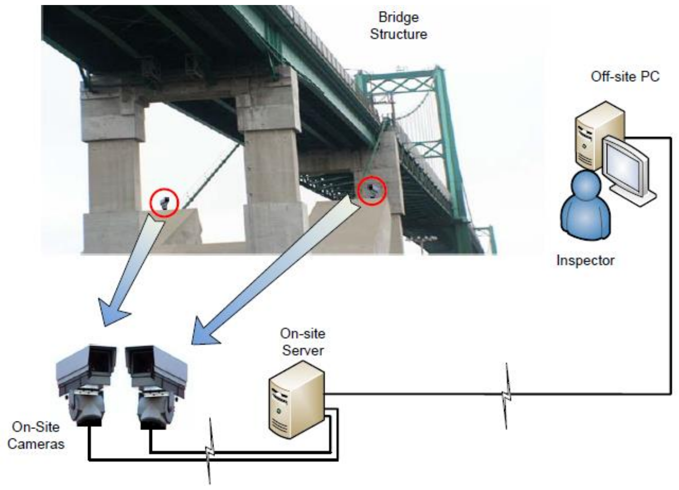


Figure 3. Schematic hardware configuration of the image-based inspection system [11].

To improve the accessibility to large bridges, Lee et al. developed devices for operating from the underside of superstructure during inspection [12]. This system is similar to an under bridge inspection vehicle but replaces a bucket with several cameras (see Figure 4) [13]. They used commercial digital cameras with auto focusing functionality to collect images. The device has an adjustable boom which could be employed for bridges with 2 lanes (each way).

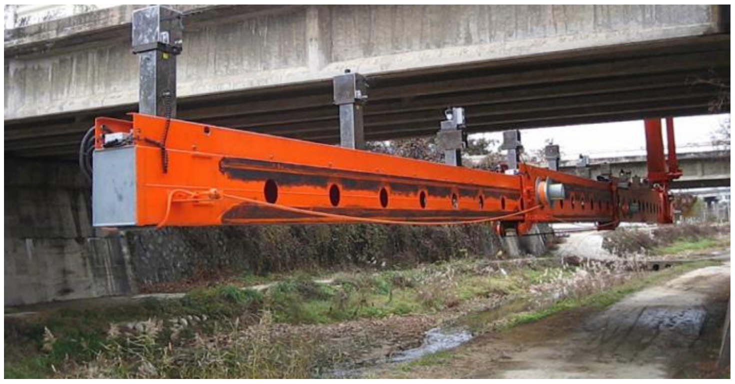


Figure 4. The adjustable boom for image acquisition under bridges [12].

The widespread availability of commercial unmanned aerial vehicles (UAV) (particularly, quadcopter drones) has been a key driver of inspection robotics research [5,14]. In these systems quadcopter drones are used to capture a video of a structure. In some methods, the system will also record the location of the drone and will use it to reconstruct a 3D model of the structure. Lattanzi and Miller provided a review of robotic infrastructure inspection and image acquisition methods over the past two decades [5]. Dorafshan and Maguire were also reviewed the state of practice of U.S. bridge inspection programs and summarized current and future capabilities of unmanned aerial systems in automated bridge inspections [15]. In their review paper, they discussed the challenges of using UAV for bridge inspection and concluded that the recent advances of UAV could potentially shift the bridge inspection paradigm by providing low cost options for image acquisition.

For pavement image acquisition, using a digital camera with strobe or halogen lights which are mounted on a mobile laboratory is very common (see Figure 5) [16–19]. Lopes et al. developed a system for scanning road surface called Road-Kill for surveying mortality of amphibians in Portuguese roads [20]. They used a camera with 35mm lens and a LED lighting system in their image acquisition system (see Figure 6). Their system can scan a 1 m width of roadway in every pass at 30 km/h and with a 250 μm/pixel resolution.

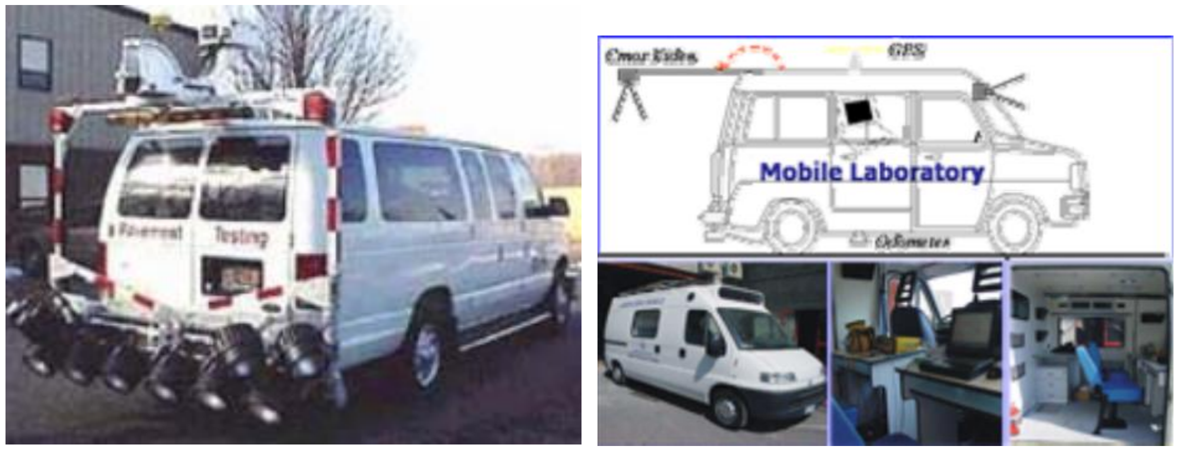


Figure 5. Mobile laboratory for pavement image acquisition [16].

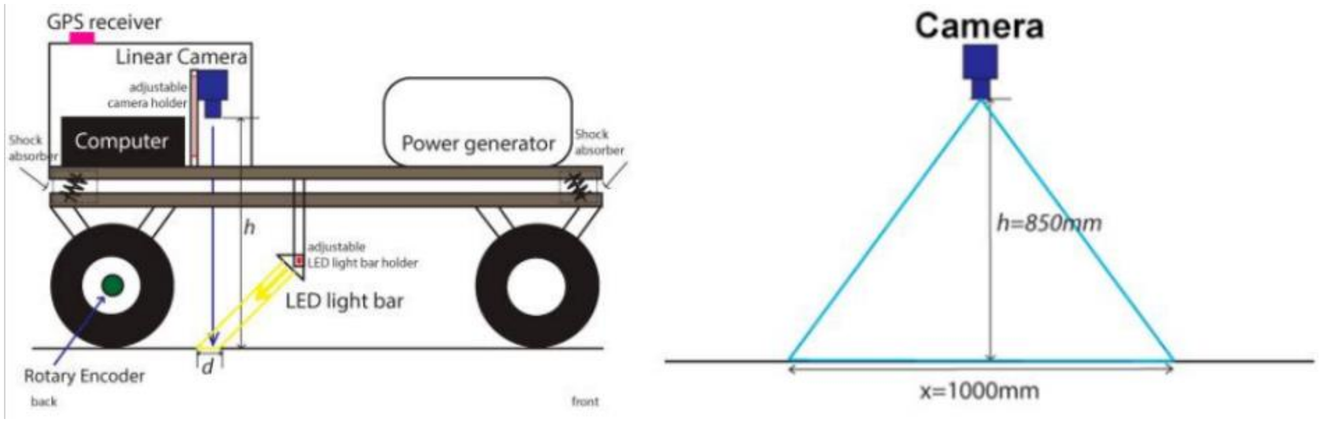


Figure 6. Diagrams of the developed scanner, Road-Kill [20].

Due to the advances in smart phone cameras and their widespread availability, several researches have used this technology for data acquisition [21–23]. Zhang et al. used smart phones to collect more than 500 pavement pictures of size 3,264×2,448 pixels at the Temple University campus for pavement evaluation [22]. Maeda et al. installed a smart phone on the dashboard of a car, as shown in Figure 7, and drove the car to capture images of 600×600 pixels once per second

[23]. The average speed of the car was 40 km/h. The images were captured in a wide variety of weather and illuminance conditions. Varadharajan et al. used a Samsung Galaxy Camera for image acquisition that almost has all functionalities of a smartphone [24]. The camera was mounted on the windshield of a personal vehicle (see Figure 8). They collected more than 100 hours of video with 1080p and 10 Hz (about 4 million images), over a period of one year in Pittsburgh area.

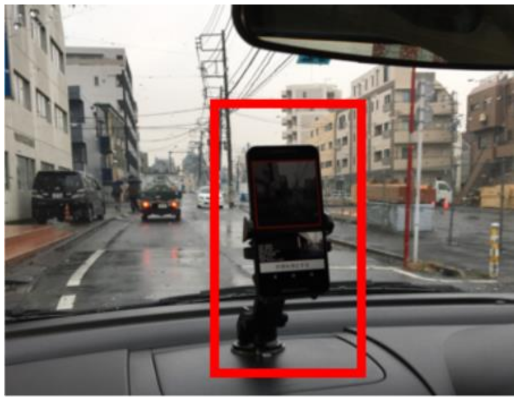


Figure 7. Using smart phones for collecting pavement images [23].

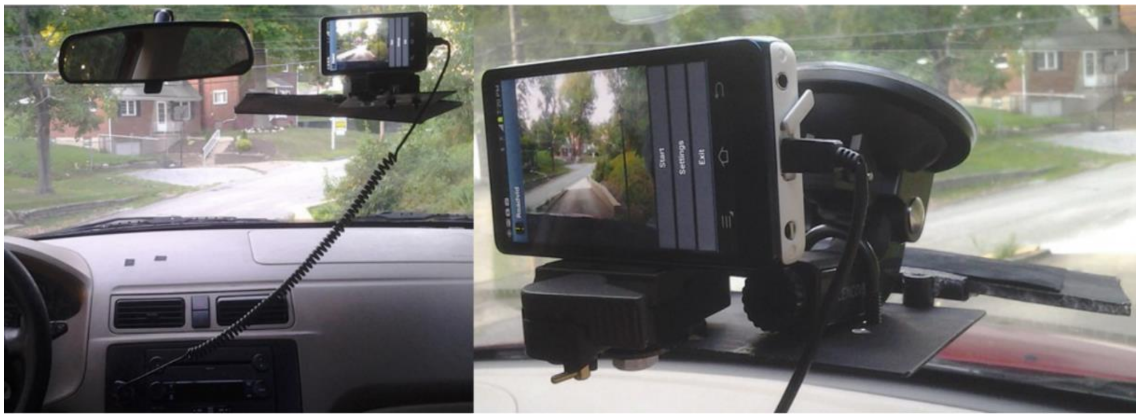


Figure 8. Using smart camera for data collection [24].

1.2 Literature Review on Automation in Image Processing

Many researchers have developed and applied different computer vision techniques for damage detection in civil infrastructure. Most of the early works in crack detection were mainly based on thresholding intensity values of pixels [25–28]. In these methods, the pixels are partitioned depending on their intensity value. The main assumption of these methods is that the cracked areas are darker. Although these methods are effective in some applications, they are generally too simplistic to be applicable to images with variety of artifacts. Shi et al. [28] classified the most recent studies in crack detection into five categories: methods based on saliency detection, textured-analysis, wavelet transform, minimal path, and machine learning (ML). Critical assessment of some of these categories can be found in [29,30]. Salient detection is more visible due to the contrast of a crack with its surroundings [31,32], but has poor performance on assessing the completeness and continuity of a detected crack. Some researchers used textured-analysis methods for road crack detection since the pavement images are often highly textured [33–35]. For detecting cracked area, these methods use a local binary pattern operator. Since the local neighbor information is not considered, the cracks with intensity inhomogeneity will not be detected with high accuracy [28]. Wavelet transform is applied to pavement images for noise reduction [36] and crack detection [37]. Because wavelets have anisotropic characteristic, wavelet transform methods may not work well in detection of the cracks with high curvature or with low continuity [38]. In minimal path methods, simple open curves in images can be extracted by providing the endpoints of the curve [39]. Several researchers have applied these methods in crack detection [40–42]. The availability of large datasets encouraged several researchers to apply ML methods to crack detection [43–48]. Many of the studies in crack detection use edge detection techniques to extract useful information form the images [13,49–51]. Over the history of digital image processing, a variety of edge detectors have been developed which differ in their mathematical and algorithmic properties [52–55]. Abdel-Qader at al. compared four different edge detection methods and showed that the Fast Haar transform was more reliable than the other three edge-detection techniques in identifying cracks [56]. Additionally, Zalama et al. utilized Gabor filters to detect the longitudinal and transverse cracks in road images [57]. They used 4 Gabor filters with different orientations to detect cracks. Figure 9 shows two of the Gabor filters that they used in their study. They reasoned that the drastic differences will help to cover the widest possible range with the filters.

The images used in these studies were typically high-resolution and clear images with minimum artifacts. Edge detection requires smoothing and differentiation of the image using different filters. Differentiation is an ill-conditioned problem so the presence of artifacts in images significantly alter the result. Additionally, smoothing results in a loss of information, which may lead to total loss of useful information in low-resolution images. Considering these issues, it is difficult to design a general filter which performs well in many contexts [13]. Instead of explicitly engineering a filter that may work well under certain conditions, many researchers, in the recent years, have attempted to use convolutional neural network (CNN) architectures to train layers of filters that can extract useful information from the images.

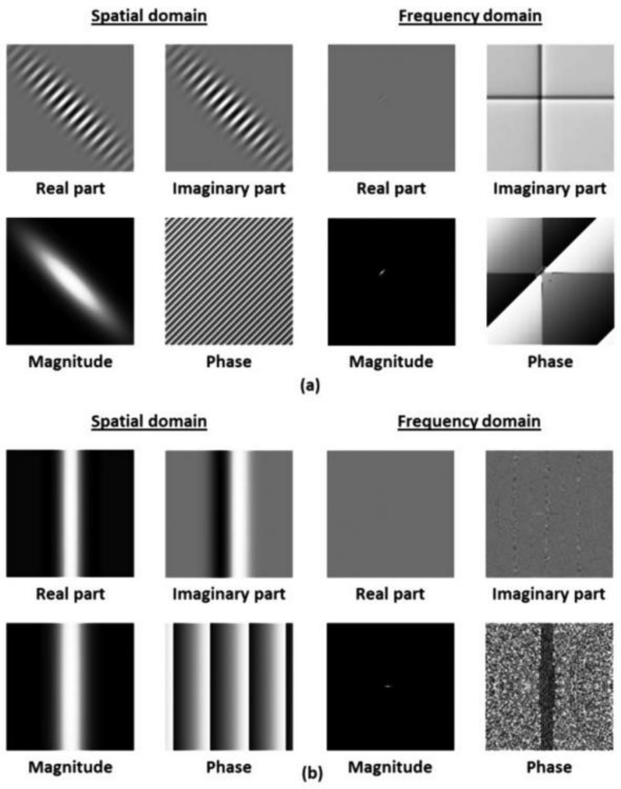


Figure 9. Spatial and frequency domain of two Gabor filters (a and b). Real and imaginary part range is from –1 to 1, magnitude range is from 0 to 1, and phase range is from –180° to 180°. (a) Central frequency is 0.010 and orientation is 45°. (b) Central frequency is 0.010 and orientation is 45°. (b) Central frequency is 0.013 and orientation is 0° [57].

1.3 Literature Review on Deep Learning and Crack Detection

In recent years, deep learning (DL) methods have been proven to be very effective in solving many practical problems [58–61]. By relying more on automatic learning and less on heuristics, LeCun at al. showed that better pattern recognition systems can be built [62]. By introducing AlexNet in 2012, Krizhevsky et al. achieved record-breaking results in an image classification contest (ImageNet challenge [63]) and demonstrated the power of CNN architectures

[64]. Since then, several researchers have applied AlexNet and other CNN architectures to damage detection of civil infrastructure. Cha et al. developed a classic CNN for detecting concrete cracks and compared their results with Canny and Sobel edge detection methods [65]. They used 40,000 images with 256×256 pixel resolutions for training the network and 55 images of 5,888×3,584 pixel resolutions for testing. They have showed that CNN performs better in finding concrete cracks in realistic situations [65]. In another study, Cha et al. applied Faster Region-based CNN for detecting multiple damage types [66]. Huang et al. used fully convolutional network for semantic segmentation of crack and leakage defects on inner surface of concrete tunnels [67]. Chen and Jahanshahi proposed Naive Bayes CNN to analyze individual video frames for crack detection on nuclear power plant components [68]. Wang et al. proposed a CNN architecture for pavement crack detection on 3D asphalt surfaces that removed pooling layers in typical CNNs to ensure pixel-perfect accuracy [69]. Zhang et al. utilized a simple CNN with three convolutional layers to detect pavement cracks [22]. They evaluated their method on 500 images (size 3,264×2,448 pixels) collected by a low-cost smart phone and showed the superiority of DL framework when compared to existing hand engineered methods. Maeda et al. collected 9,053 road damage images captured with a smartphone installed on a car, and applied Single shot multibox detector (SSD) Inception V2 and SSD MobileNet (two CNN architectures) for detecting the location and type of cracks in road images [23]. Eisenbach et al. evaluated both computer vision and DL crack detection approaches with the German Asphalt Pavement Distress data set [70]. Pauly et al. investigated the effectiveness of having more layers in CNN architectures for pavement crack detection [71]. They also showed how variations in location of training and testing data sets affect the performance of the DL.

1.4 Problem Statement

To train a deep CNN with many hidden layers, a large data set is required. In fact, the benefit of using a deep network are only revealed when a large data set is available for training. In the crack detection literature, there are few large labeled image data sets which are suitable for DL applications. Most of the pavement crack detection studies use road images taken directly from above the road [24]. It is difficult to reproduce these images and it is costly to maintain a dedicated car for taking road images [24]. In 2017, Eisenbach et al. stated that there were only three different data sets available for crack detection of pavement images, all of which have less than 300 total images [70]. In the summer of 2018, the need for a large data set of images for classification of structural objects, inspired Pacific Earthquake Engineering Research Center (PEER) to organize the first image-based structural damage identification competition, namely PEER Hub ImageNet (PHI) Challenge [72]. All these factors indicate that there is an immediate need for building a proper data set of images for damage detection of civil infrastructure.

In this study, the main objective is to develop a reliable, inexpensive, accurate and automated system for identifying cracks in a surface. Due to the importance of identifying cracks in pavements, this study will focus on crack detection in asphalt pavements. However, the developed system will readily be applicable for crack detection on concrete surfaces in a structure. By reviewing the recent literature of visual inspection automation, it is easy to observe a growing interest in using neural network (NN) algorithms in civil infrastructure applications. While NN algorithms have proven themselves to be reliable and efficient in solving complex problems, the affordability of powerful graphics processing units (GPUs) have advanced the research by providing the computational speed required for training a deep network. In this study, two different deep CNN architectures are developed and applied to pavement image classification. To train and test the network, a data set is collected containing pavement images captured from Google Street View (GSV). GSV provides panoramic 360-degree views from positions along many streets in the world and forms a massive data set that is well-suited for DL applications. GSV has been successfully used in many fields of research [73–75]; however, this study is the first attempt to use GSV technology for damage detection. The main challenge in using the GSV data are the presence of many artifacts and low quality of the images. Therefore, an effective crack detection method is implemented to address these challenges.

The chapters of this dissertation are structured in a way that provide an overview of the important concept and methods in this study. Chapter 2 provides an extensive discussion of the data collection process and the nature of the data. In this chapter, the practical challenges are explained when working with data collected from GSV. In Chapter 3, the fundamental concepts of DL are introduced along with the building blocks of the CNN used for solving the image classification problem. Chapter 4 provides a detailed discussion of the designed experiments for crack detection and their results. Lastly, in Chapter 5, a summary is provided on the overall crack detection process, conclusions, and some comments on topics for future study.

2. Chapter 2

Data Collection

This study is the first attempt in using Google Street View (GSV) images for civil engineering applications. The data collection portion of this study posed multiple challenges that that were addressed with innovative and unique solutions. In this section, the details are presented for data acquisition, data representation, data preprocessing, data labeling, complexity analysis, and some of the pragmatic challenges in collecting the data.

2.1 Data Acquisition

There are four million miles of roads in the United States and collecting data for pavement evaluation is a time-consuming and expensive task which makes it almost impractical to perform in a large scale for the whole network. To overcome this challenge, GSV is used to collect the required data. In this subsection, information is presented on GSV and methods for collecting data using GSV.

The Sandford City Block project [76,77] was the origin of GSV. The purpose of the project was to build a technology for multi-perspective panoramas from sideways-looking video taken from a vehicle driving on a street (Figure 10). This project was folded into Google Street View project to provide an interactive panorama from positions along many streets and roads in the world. Most of the photography is done by google cars as shown in the Figure 11. For each single location GSV provides a photo sphere, or set of images, that provides a full 360-degree view. The resulting 360-degree panoramic image defines a projection on a sphere with the image wrapped to the two-dimensional surface of that sphere [78]. Since its launch in 2007 [79], the

GSV project has captured billions of photos across many countries [80]. In 2012, Google announced that it has captured 20 petabytes of data for GSV. By increasing the quality of images in 2017 and innovation of variety of methods for capturing images, the size of the GSV database has been exponentially increased. GSV data has been used by the computer vision community for testing different methods [81,82] and a source from which data is extracted and analyzed [83–85]. A large portion of the GSV database is images of pavements. Tapping this massive resource of data can provide a unique tool for solving one of the challenging problems in monitoring civil infrastructure.

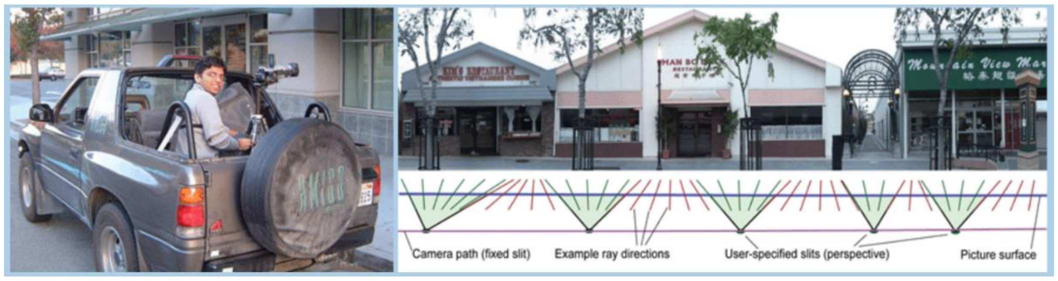


Figure 10. Shown at left, the camera mounted in the back of a slowly-moving car, and at right is the constructed panorama [86].

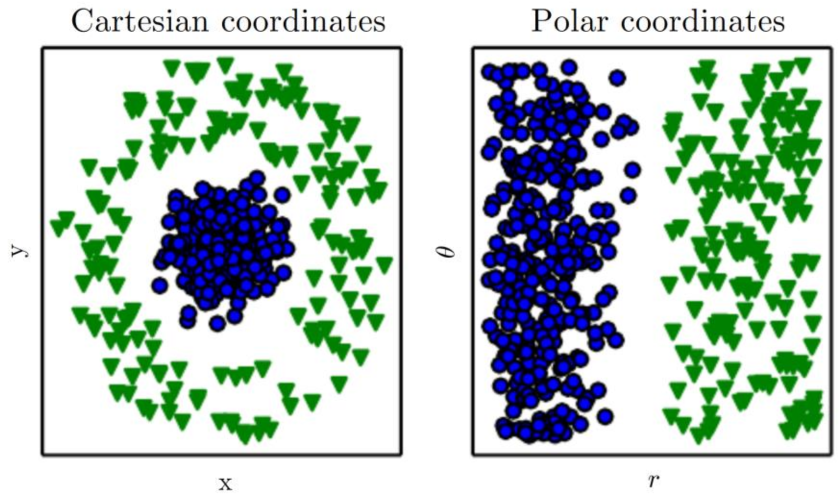
Figure 11. Google street view cars [87].

There are several methods for extracting images from GSV. The first method is to save a view that covers an area of pavement under investigation. For this method, simply go to Google Map website and find the location that of interest. GSV can be navigated by dragging and dropping the pegman icon on the street, selecting a view that covers the area of the pavement, and storing the viewed image on a hard disk. There are other websites that might help to perform this process [88,89]. These websites try to provide a fast and user-friendly interface to work with GSV. While manually navigating through GSV and saving images is a time consuming and impractical method for evaluating a whole network of roadways, for research purposes this method can provide adequate data for training and testing a DL model. Web browser interactions can be automated using a Python library called Selenium [90]. Although Selenium can help to extract data from a straight road, it was found to be ineffective for streets with multiple turns (unless the GPS coordinates are given). The practical option is to use GSV Static API [91]. The location, camera angle, and the size of the image can be selected, and the API will provide the image (Google charges $0.0056 for each image). For this study, images were manually extracted directly form the GSV website. The images are cropped and spliced into smaller patches to form both the training and test data sets.

2.2 **Data Representation**

A digital image is a collection of picture elements or pixels that have been organized in a grid-shape format with fixed number of rows and columns. Digital images in machines are stored in a simple 2D array (for gray images) of numbers which represent the intensity of light at the pixel. A typical color image consists of three layers (channels) of 2D arrays for red, green, and blue light for the additive RGB color model. The intensity values in each array can be digitized in 8 bits (256 intensity levels). This forms a typical 32-bit images. Although images have a well-defined structure, in data science they will be considered as unstructured data for the classification task.

Many ML problems can be solved by identifying the correct set of features. These features will be provided to a ML algorithm to perform the required task. For example, a useful feature for identifying a speaker from their sound is the speaker's vocal tract. This feature of the sound provides a good indicator for identifying if the speaker is a man, a woman, or a child. However, for many tasks it is difficult to know what features should be extracted. For example, it is difficult to describe what a crack looks like in terms of pixel values. One solution to this problem is to utilize ML to not only learn the mapping form input to output but also learn how to represent the data to make it possible to map. To illustrate the importance of data representation, consider the following example: classifying data presented in cartesian coordinates with a linear classifier. In Figure 12(a), there is no linear line that can correctly separate the data points into two unique subsets. In Figure 12(b) the same data is presented in polar coordinates and a linear classifier can easily separate the data points.



(a) (b)

Figure 12. Data representation [92].

A deep network model learns how to represent data in different forms in each layer of the network and make it possible for the model to correctly learn the general pattern. Particularly, in CNN, the network learns to represent the raw pixel values into different forms. DL breaks down a complicated mapping into series of nested simple mappings (Figure 13). In other words, the network learns to put a structure on the unstructured data. Figure 13 shows a graphically simplified DL model that has been trained for object classification. Since it is not possible to directly extract all the important features of a complex object from pixel values, the network breaks a complex feature down to series of simpler features in each layer. In the first layer, the network learns to transform the raw pixel values to a space defined by edges. The output of the first layer is edges of an input image. In the next layer, the output of the previous layer is transformed to a space defined by corners and contours. Then, in the third layer it learns to represent the corners and contours in a space defined by object parts and from this last layer the model can decide about the class of the image.

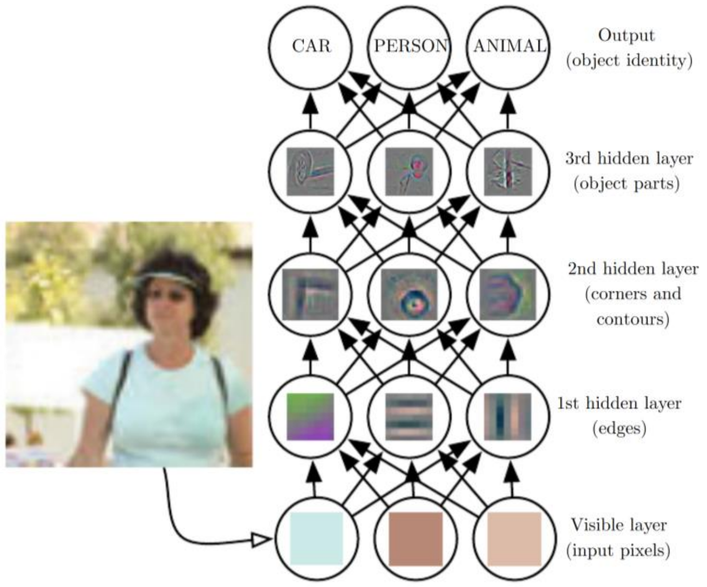


Figure 13. Illustration of a DL model [92].

2.3 Labeling Images

The first part of this study was to identify if there is a crack in the image or not. This is a binary classification problem. For solving this problem, 1,500 images of 1,800×800 pixels were extracted from GSV for several roads in Memphis, TN. These images were than split into patches of 200×200 pixels (see Figure 14) and labeled as “cracked” or “not cracked” as is illustrated in Figure 15. Overall, 48,000 images were manually labeled and cleaned. The number of images in each class is in the same range for both classes.

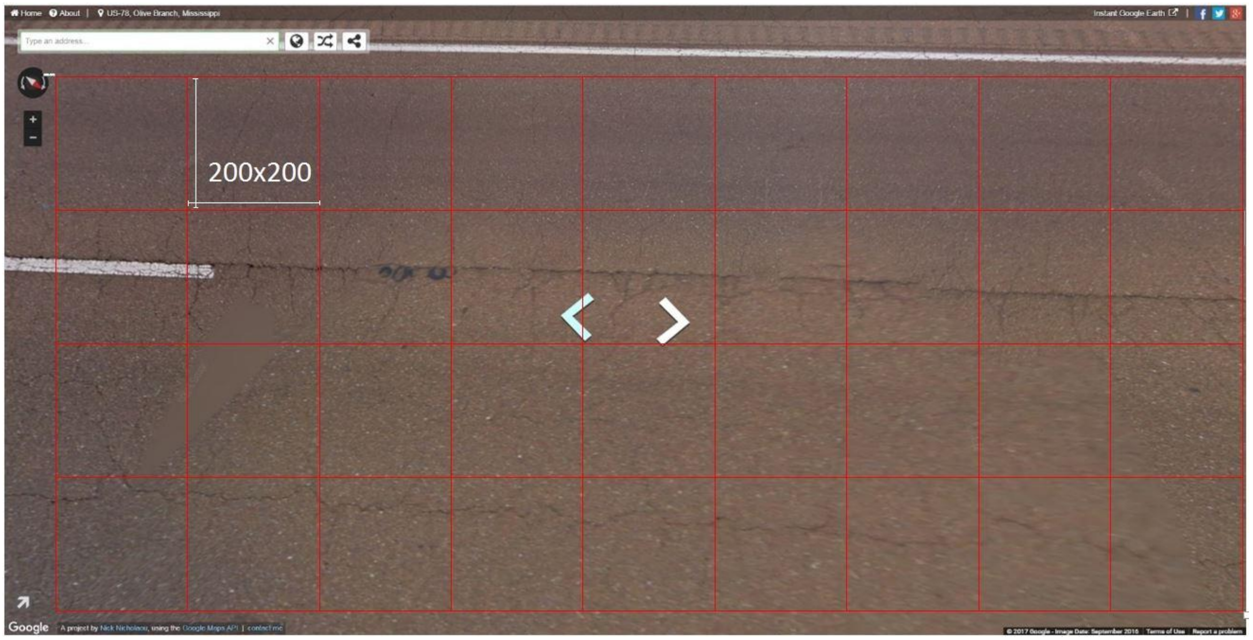


Figure 14. Splitting one image into small patches.

The second part of this study is a multiclass classification problem where the different types of pavement cracks are identified. For this classification problem 2,346 GSV images of 3,750×500 pixels were collected from a part of Route 28 in Virginia. The length of the road is around 13 miles. Figure 16 depicted the locations of the captured images of Route 28. The crack evaluation for this part of the road was performed by a commercial company (ARRB [93]) and was available for this study. The company uses intelligent Pavement Assessment Vehicle (iPAVe), fully automated crack detection system that uses 3D sensors combining lasers and high-speed 3D cameras, to provide data on the quality and quantity cracks in pavement. To be able to compare the result of GSV with the result of the company, the images were split into patches of 250×250 pixels, and labeled into 5 categories with the following simplified description:

- Not cracked, when there is no crack in the image
- Longitudinal crack, when there is just one horizontal crack in the image
- Transverse crack, when there is just one vertical crack in the image
- Alligator crack, when there is more than one crack or when the shape of the crack is similar to alligator crack
- Not pavement, when the image is not pavement

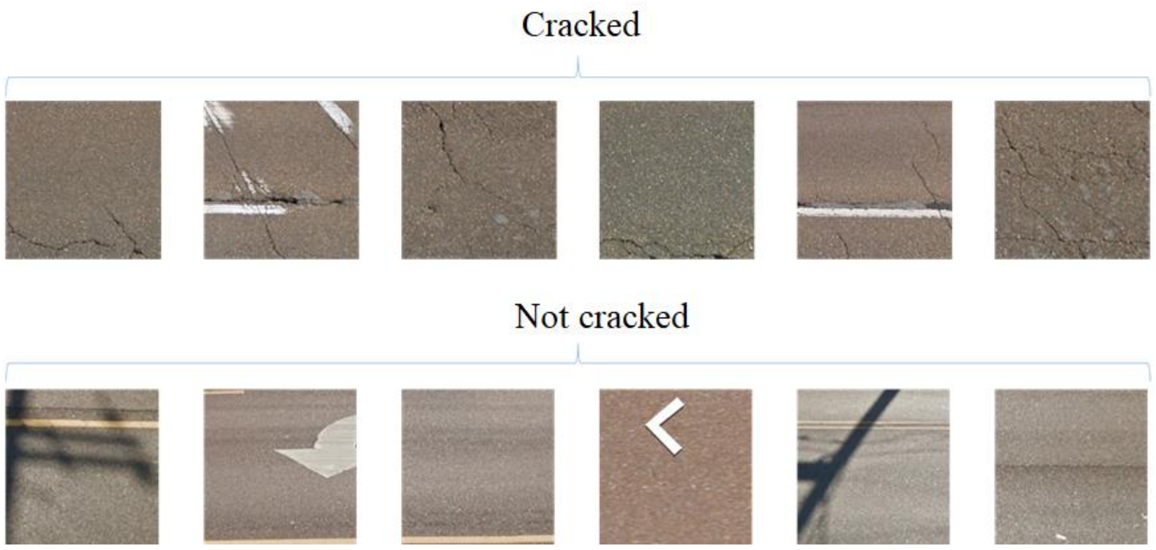


Figure 15. Labeling image patches into cracked or not cracked.

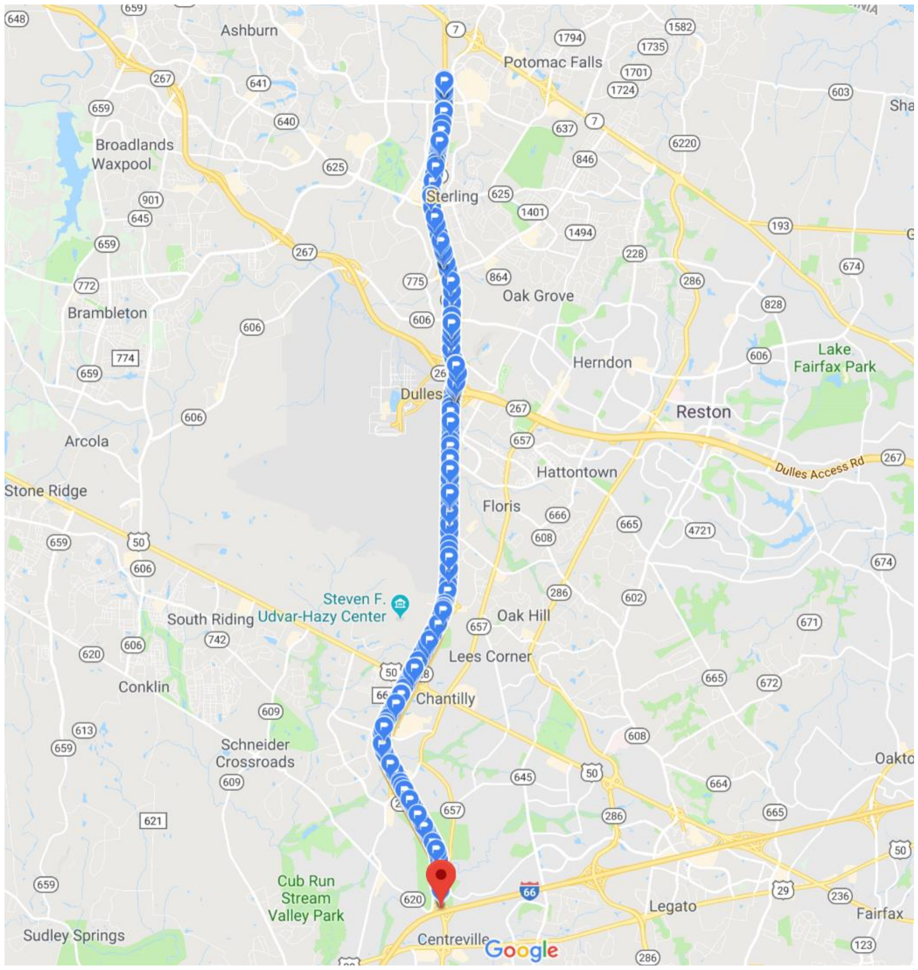


Figure 16. Data points on the Route 28.

Figure 17 shows samples of images of the five classes. An interactive tool was developed in MATLAB and used to speed up the labeling process and minimize the human error (see Figure 18). For each image, the MATLAB program starts from the top-left corner the image and one by one draws a red rectangular around border of each patch. The user can zoom on the image to observe more details. Also, a magnified view of the current patch is also plotted alongside of the whole image to help the user decide which category best classifies the pavement conditions of the presented patch. Each patch was labeled with an integer number (0, 1, 2, 3, 5) cracks conditions. By pressing each of these numbers on the keyboard the selected number is stored and the red rectangular slides to the next patch. After labeling all the patches in an image, the MATLAB program shows the entire image with the selected labels for all patches in order to allow the user to double check the labels (see Figure 19). Patches are then color coded to help spot errors in the labeling. The user will be asked to rate their level of certainty about their selections. In this study, two civil engineering graduate students to perform the labeling process. Each student was given the same instructions, data, and MATLAB program for labeling the data. In addition, to help them perform the task correctly a relativity long lead time (about a month) was given for the labeling process. Based on rough estimates, each student spent around 20-30 hours to complete the entire labeling process.

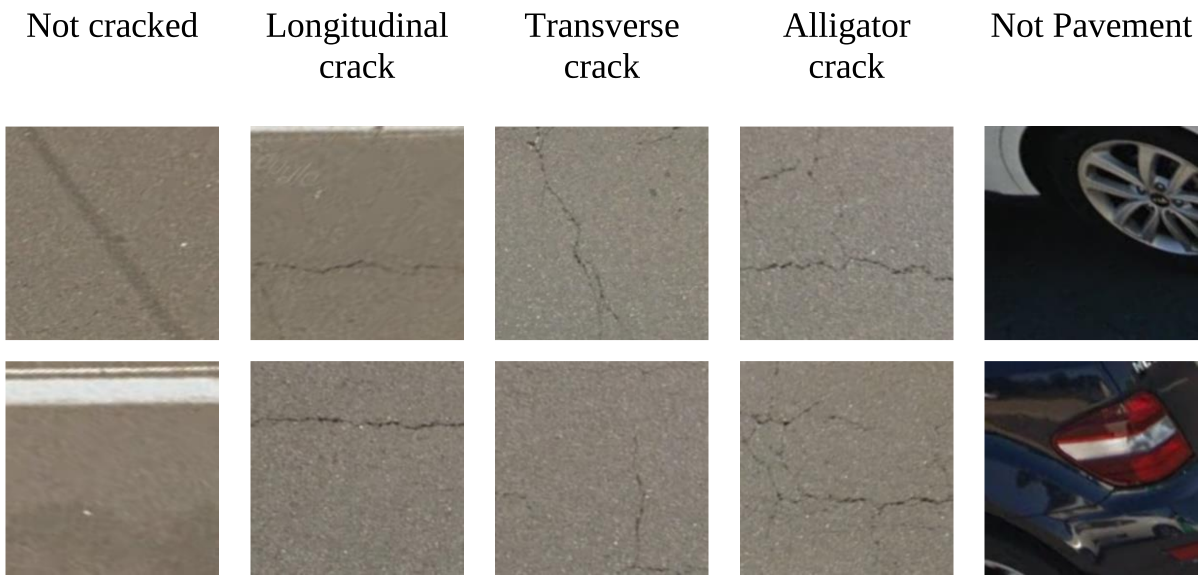


Figure 17. Sample of 5 different classes.

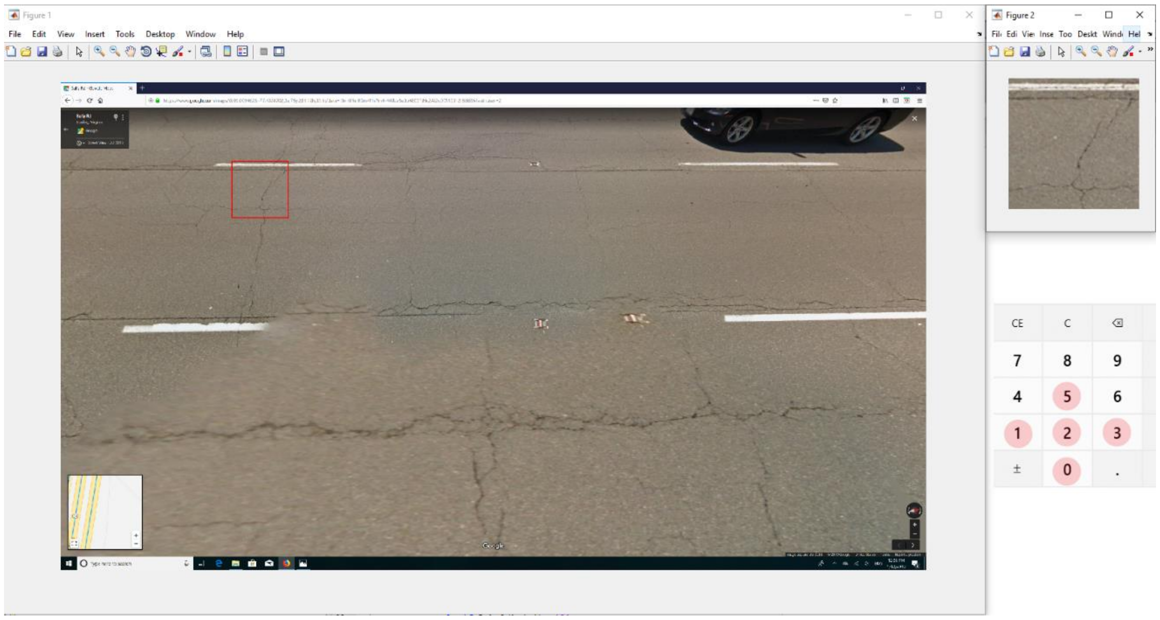


Figure 18. The interactive program for labeling images.

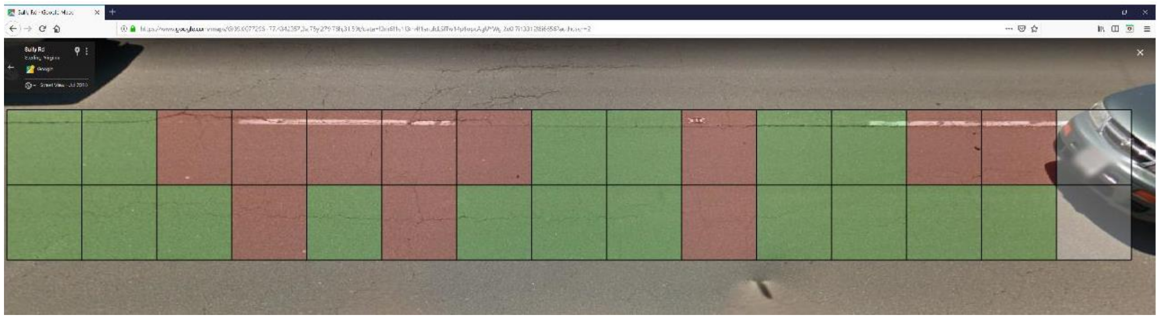


Figure 19. A color-coded image for double checking the labels.

After comparing the labels identified by each of the two students, the frequency of mismatches for each class are analyzed. Overall 8,546 mismatches were indentified which is around 12% of the entire data (70,380 patches). This is an important number which is a rough estimate of human error in classifying Google Street View images. Figure 20 shows the distribution of the mismatches. In this figure, the number of patches with incongruent labeling is plotted based on the different ten possible combinations ($C(5, 2)=\frac{5!}{(5-2)!2!}=10$). Since the number of samples in each class is different, the percentage of the discrepancy is presented for each combination of mismatches in Figure 21. The highest discrepancies are associated with the longitudinal-alligator cracks and transverse-alligator cracks, with percent discrepancies of 22.4% and 13.3%, respectively. In practice many longitudinal and transverse cracks will develop into alligator cracks, and based on severity, environmental factor, and other information, the class of the crack can be estimated. Here, the information is limited to one picture which result in high discrepancy.

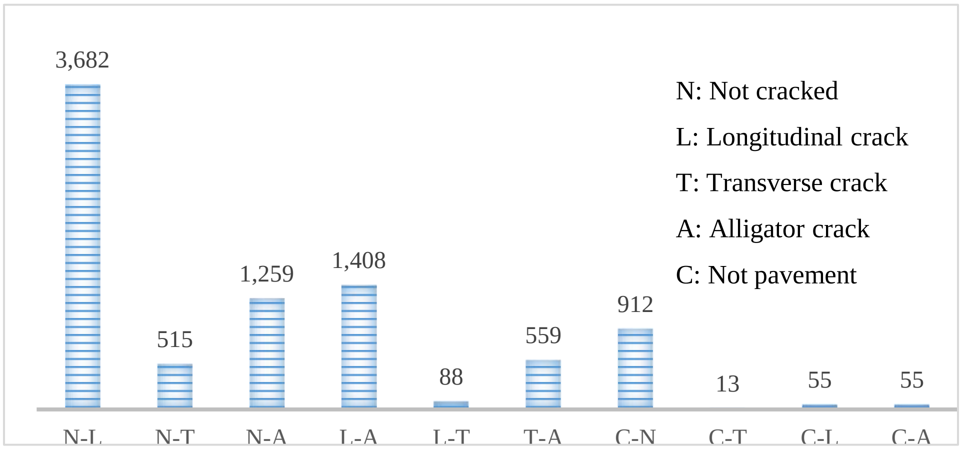


Figure 20. Distribution of discrepancies between the labels of the two students.

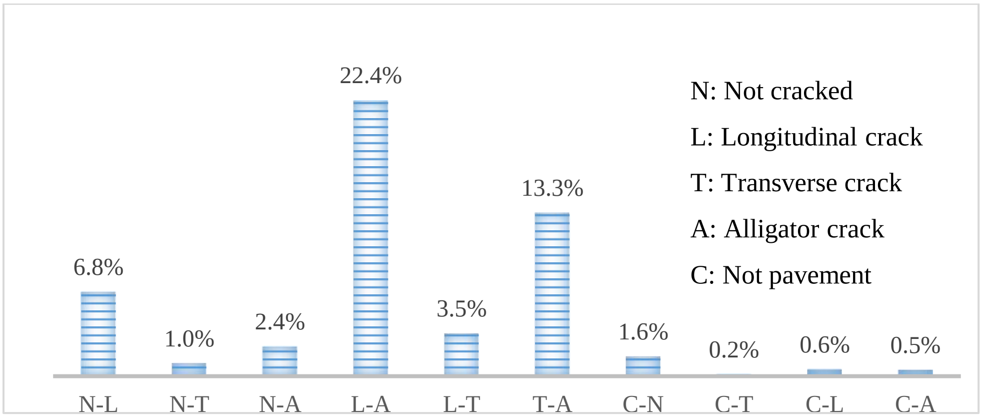


Figure 21. Percentage of labeling discrepancies among different classes.

In this study, the mismatched images are eliminated from the training data set. Figure 22 shows the number of images reaming in each of the five classes. It appears that in this data set that most of the images have no cracks. This is not an unexpected result since Route 28 is an important highway that has been routinely maintained. Since the labeled data distribution is not balanced; therefore, several techniques will be applied to that help with imbalanced data sets.

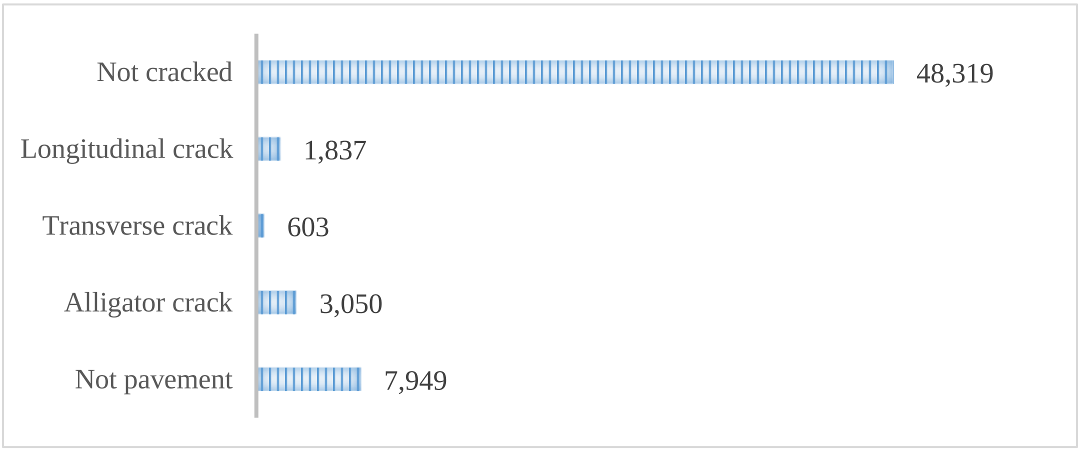


Figure 22. Number of samples in each class.

2.4 **Working with Imbalanced Data**

Most ML models are designed to maximize accuracy and reduce error; therefore, they work best when the number of samples in each class are in the same range. The first step in managing imbalanced data is to choose a proper metric for measuring the performance of the model.

2.4.1. Singular Assessment Metrics

The most frequently used metrics are accuracy and error rate (1-accuracy). Accuracy is defined as the number of correct predictions over the total number of predictions. To illustrate the shortcoming of accuracy in working with imbalanced data consider this example: a data set with 1,000 samples labeled A and 10 samples labeled B. If a model predicts every samples as A, the accuracy of the model is $\frac{1000}{1010}$=0.99. However, this model is not accurate at predicting B and accuracy is not a good metric to measure the performance of the overall model. In this situation, metrics such as precision, recall, and F_β are better metrics to represent the performance of the model. For the previous example, both precision and recall for the class B is zero. The formulations of these metrics are as follows [94]

$$\text{precision}=\frac{\text{TP}}{\text{TP+FP}} \tag{1}$$

$$\text{recall}=\frac{\text{TP}}{\text{TP+FN}} \tag{2}$$

$$F_\beta=\left(1+\beta^2\right)\frac{\text{Precision*Recall}}{\left(\beta^2\text{*Precision}\right)\text{+Recall}} \tag{3}$$

where TP, FP, and FN are true positive, false positive, and false negative, respectively. Model performance is analyzed using the Receiver Operating Characteristics (ROC) curve

[94,95]. By plotting true positive rate (recall) against false positive rate for varying prediction threshold, a ROC curve visualizes the ability of the model to discriminate the positive class from the rest of the data. False positive rate is defined as:

$$\text{false positive rate} = \frac{\text{TN}}{\text{TN+FP}} \tag{4}$$

ROC plots along with the value of area under the curve (AUC) are a good way to analyze the performance of a model, especially for a binary classification.

2.4.2. Resampling

One effective technique to help overcome imbalancment in data is resampling, which includes over-sampling and under-sampling. In over-sampling, samples are collected or synthesized to increase the number of samples in the minority class. In under-sampling, some samples are randomly eliminated in the majority class to reduce the number of samples. Figure 23 illustrate these resampling concepts. For this study the difference between the number of samples in minority and majority class is significant. For example, there are 75 times more samples in not-cracked class than in the transverse class. By performing under-sampling, a significant portion of the labeled data will be removed. Since NN models need large data set to train the model, this option is not considered in this study. Additionally, collecting more data is time consuming and there is no guarantee that the new data will increase the minority class significantly. Thus, in this study more samples were synthesized from the minority classes to increase the number of samples.

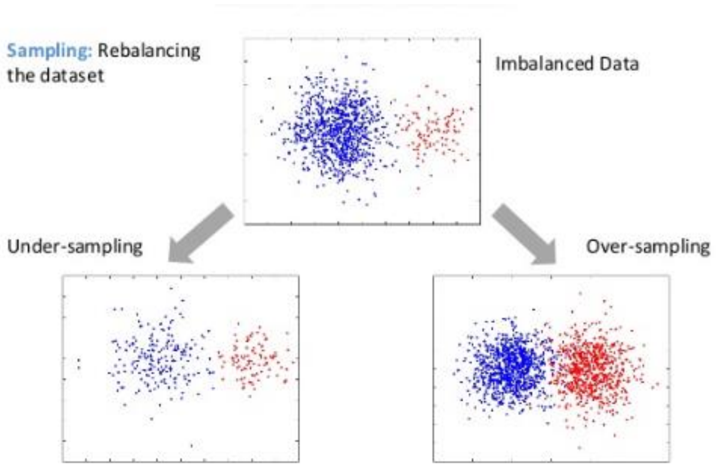


Figure 23. Over-sampling and under-sampling techniques to overcome data imbalancement [96].

There are several methods to synthesize new images from a minority class [97–101]. Typically, in these methods the original image is transformed with a combination of affine transformations to generate a new image. In this process, the arrangement or the values of pixels in the new image is slightly changed, without losing the important features of the original image. Some of these methods are flipping, rotating, scaling, cropping, translating, shearing, zooming, distorting, shading with a hue, and applying gaussian noise. Since the color of most of images are close to gray, changing the hue will not generate a new image. Cropping and translating, may remove a crack from the image. The GSV images are highly noisy and adding more noise to the images could reduce the performance of the classifier. Thus, a combination of shearing, rotating, and flipping (on both side) is used to generate new images. Figure 24 shows a sample set of the generated images based on the original image.

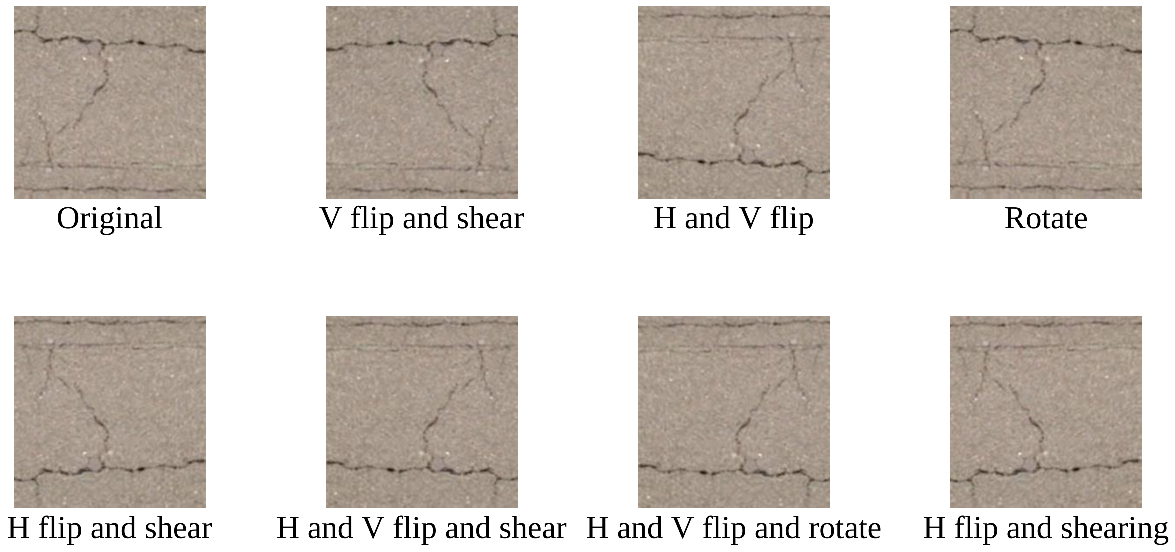


Figure 24. A sample of synthesized images (H: horizontal, V: vertical).

2.4.3. Adjusting Class Weights

Another method for mitigating imbalanced data is to consider class weights. By considering higher weights for minority class more value is placed on these samples, and if the predicted value is not correct the cost function will get more penalized.

2.5 Data Visualization

One of the effective ways of understanding the important features of a dataset is to represent the data in a graphical form. Data visualization can help in understating difficult concepts and discovering new patterns. While there are many ways to visualize structured data with a limited number of variables, visualization of unstructured data remains a challenging problem. In this study, the database is a set of images. Although an image is a graphical representation by itself, considering thousands of images in a data set requires an effective method to visualize the image data set. One way to plot image data in a graph is to consider each image as a high dimensional vector. Each pixel value is a variable in the vector. Using this method, if the images only consisted of three pixels, one can plot them in the conventional 3D cartesian system. However, in reality the number of pixels is much higher than three, so the dimensions of the data need to be reduced. There are different ways to reduce dimensionality. In this study, the Principal component analysis (PCA) is used to reduce the dimensionality of the data while minimizing information loss by projecting the data on its principal components (eigenvectors) [102]. This method allows us to visualize the image data set in a cartesian system.

In Figure 25, a sample of the first image data set (binary classification images) is plotted. Each point in the figure is a 200×200 pixels gray image where the dimensionality has been reduced from 40,000 dimensions to two dimensions. For this data, first and second principal components explain 36.4% and 5.6% of the variation in the data. It can be seen in Figure 25 that the points with colored cyan and red are mix together and there is no clear pattern to classify the points. In Figure 26, a sample of the second image data set is plotted where all three cracked classes in the dataset are combined to form a general class for cracked images, and the not pavement class is excluded. Based on these results, the multiclass data with 5 classes are changed to a binary class data set containing only with cracked and not cracked. In this data set, the images are 250×250 pixels, and the overall dimensionality of gray images is 62,500. For this data, first and second principal components explain 62.4% and 5.9% of the variation in the data which is higher than the first data set. The reason for the higher values is the variation in collecting the data. In the first data set, images were collected from different roads and with different features; however, the second data set is collected from one specific roadway. Thus, the variation of data in the first data set is higher and the first principal component is only able explain 36.4% of the variation. Similar to the binary classification shown Figure 25, in the Figure 26 points are mixed together and there is no clear pattern to classify the points. By training a DL model, the model learns how to transform (represent) the data into a new space that the classifier can effectively classifies the data.

Figure 28 shows a visualization of a sample (same number of points in each class) of the second image data set. For this data, first, second, and third principal components explain 64.8%, 8.3%, and 3.6% of the variation in the data. Figure 28 shows the same data in a 3D graph with three principal components as the axes. As expected, there is no clear pattern in the points, and a DL model is needed to learn the complicated patterns in the image data set.

2.6 **Image Pre-processing**

Since the GSV images are captured in different times of year and in different hours, the color and the luminosity of images may have drastic change. Time of day and the color of asphalt are some features that should not affect the classification results. Additionally, for optimization and stability considerations, the dataset is normalized such that the mean value of each image would be equal to zero. The mean value is calculated across the whole image and subtracted from each pixel value.

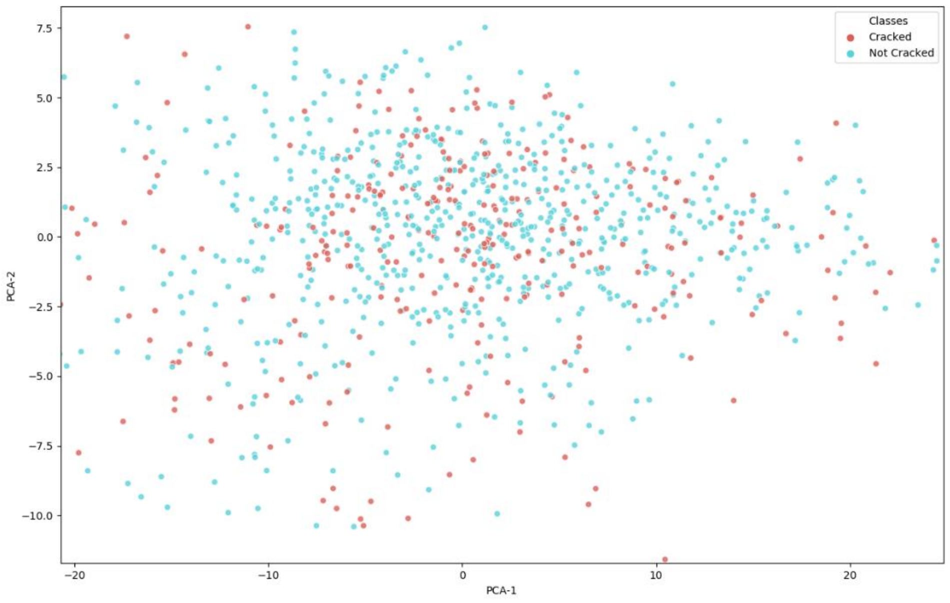


Figure 25. Visualizing 1,024 images of the first data set (with two classes).

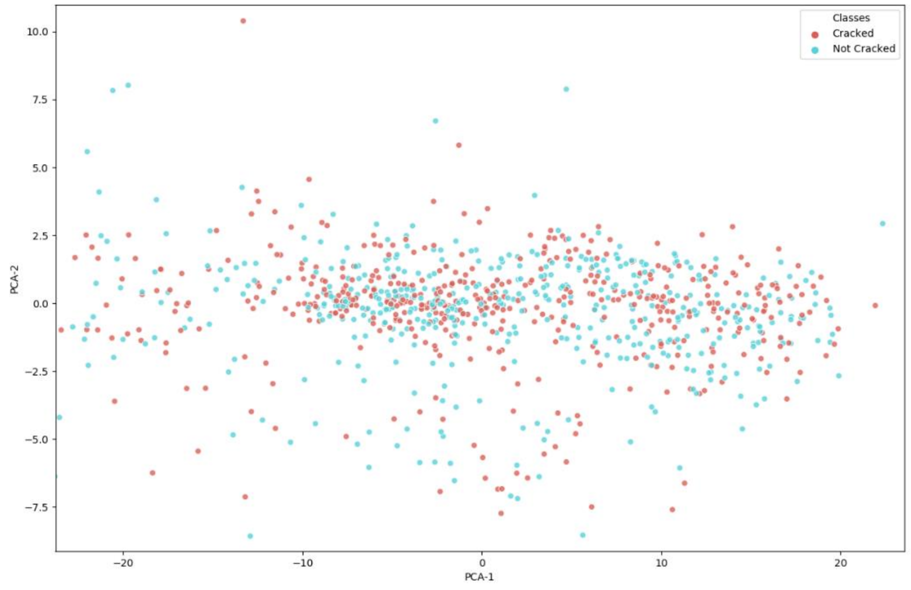


Figure 26. Visualizing 1024 images of the second data set (with two classes).

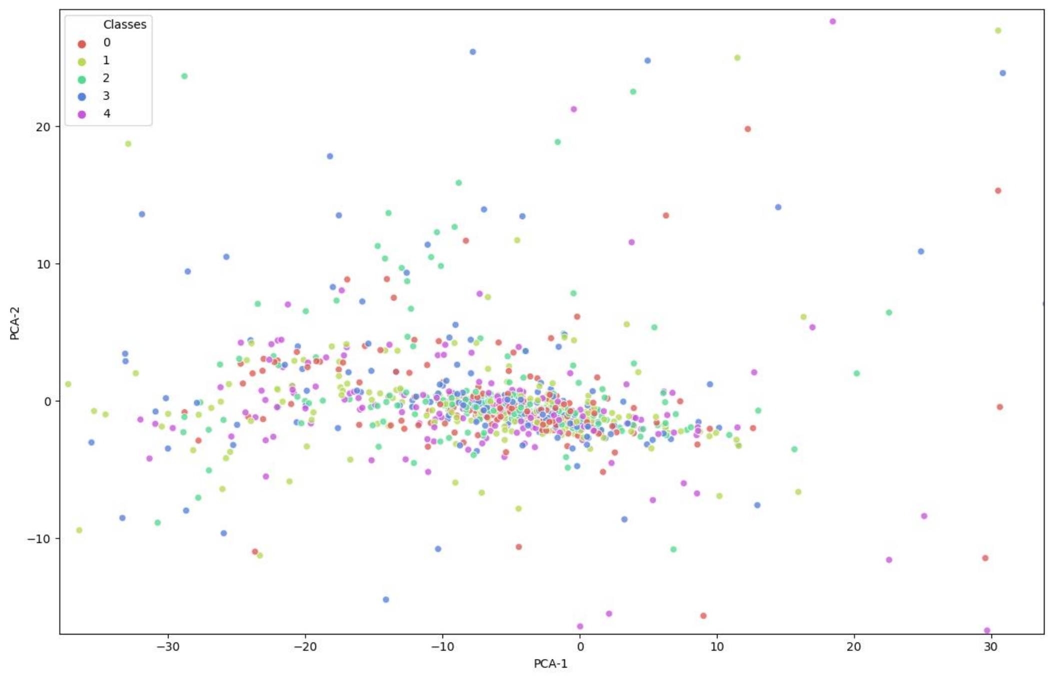


Figure 27. Visualizing 1,024 images of the second data set (0: alligator crack, 1: not pavement, 2: longitudinal crack, 3: transverse crack, 4: no crack).

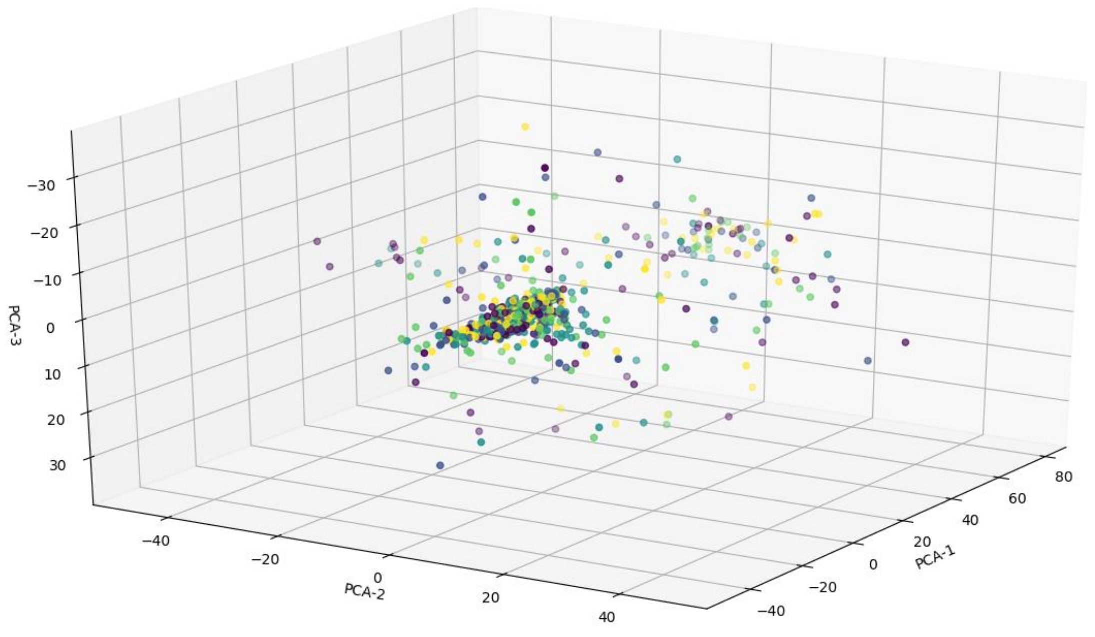


Figure 28. Visualizing 1,024 images of the second data set with three principal components.

3. Chapter 3

Deep Learning

Today, many aspects of modern society are enabled by ML methods. A massive amount of data is being constantly generated (see Figure 29), and this amount will become even larger in the future. A large portion of the available data (80%-90%) is not structured enough for most tasks (unstructured data). Traditional ML methods (such as logistic regression, support vector machine, decision tree, and k-nearest neighbors) were limited in their ability to process unstructured data. For decades, building a ML system required careful engineering and considerable domain expertise to transform the unstructured data (such as the pixel values of an image) into a suitable internal representation from which the ML algorithm could perform a task on the input data [61]. Representation learning is a set of methods that allows a machine to explore unstructured data and to automatically discover the representations needed to perform a specific task. DL algorithms are representation learning methods with multiple levels of representation [61]. This happens by combining simple but non-linear units that each transforms the input from one representation into another representation at slightly more abstract level. With the arrangement of enough such units, very complex functions can be learned [61]. Therefore, DL has gained a growing interest among many researchers in different fields. In this study, these new technologies are utilized to solve a challenging problem in civil engineering. In this section, the fundamental concepts in DL are discussed and different building blocks are presented that are required to build an effective DL model. From these concepts and tools, a model is developed for crack classification in asphalt pavements.

Company	Size of daily processed data
eBay	100 PB*
Google	100 PB
Facebook	30+ PB
Twitter	0.1 PB
Spotify	0.064 PB

Figure 29. An estimation generated data by few companies [103].

3.1 History of Deep Learning

In the early days of making machines intelligent, the field of artificial intelligence (AI) rapidly attempted to solve problems that were intellectually difficult for a human; problems defined by a of list of formal and mathematical rules, but relatively straight-forward for machine computation. The true challenge to AI is in solving tasks that are easy for a human, tasks that are intuitive, but difficult to describe in a formal machine language [104]. DL is a powerful approach to solve these challenges. DL not only attempts to discover the mapping from data representation to the output but also to learn the representation itself [104].

The term “artificial neural network” or “neural network” have been used interchangeably with DL since some of the earliest learning algorithms were intended to be models of how learning happens in the brain. In the early 1960s, Rosenblatt popularized neural network (NN) by describing many different kinds of perceptron networks [105]. Later in 1969, Minsky and Papert analyzed the limitations of perceptrons [106]. Many people overgeneralized these limitations to all NN models which led to a major drop in NN popularity. Many methods in DL were developed between 1980s–1990s, such as the long short-term memory [38] and back-propagation algorithm [39]. In mid-1990s, the AI community began to make unrealistic claims, which led to disappointments when AI research did not satisfy these unreasonable expectations. Simultaneously, other fields of ML such as Kernel machines and graphical models achieved good results on many important problems. These two factors led to another decline in the popularity of NN that lasted up until 2007 [104]. In 2006, Hinton et al. showed that deep belief network could be efficiently trained using greedy layer-wise pretraining [107]. Other researchers implement the same strategy to train other kinds of deep networks [108,109]. These studies helped bring DL out of dormancy. Today, by outperforming other ML methods in many AI challenges, DL has placed itself among the most successful methods in supervised, unsupervised, and reinforcement learning.

3.2 Machine Learning

Since DL is a part of a broader family of ML methods, it is necessary to discuss some of the fundamental concepts in ML. The algorithms and models in ML have been used in many fields, and thus, there are multiple definition of ML. The term “machine learning” was coined in 1959 [110], and it can be defined as the scientific study of algorithms and statistical models that computer systems use to perform specific tasks by learning from experience (data) [111]. Learning from data is used in situations where there is a pattern in the data, and an analytical solution does not exist or is too complicated to be derived. In these situations, ML provides some tools to explore the data and learn the pattern. ML problems are often categorized in three general classes: supervised learning, unsupervised learning, and reinforcement learning (see Figure 30).

In supervised learning, an algorithm learns from a set of data (learning set) that contains both the inputs and the outputs and builds a mathematical model for estimating a desired output for a new input (test set). For instance, if the task were determining whether an image contained a particular object, the training data would include images with and without that object (the input), and each image would have a label (the output) entitling whether it contained the object or not[111]. In contrast, unsupervised learning methods are used when the outputs are not available. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. Reinforcement learning is a type of ML technique that enables a software agent to learn in an interactive environment by trial-and-error using feedback from its own actions and experiences. The agent automatically determines the ideal behavior within a specific context to maximize its performance.

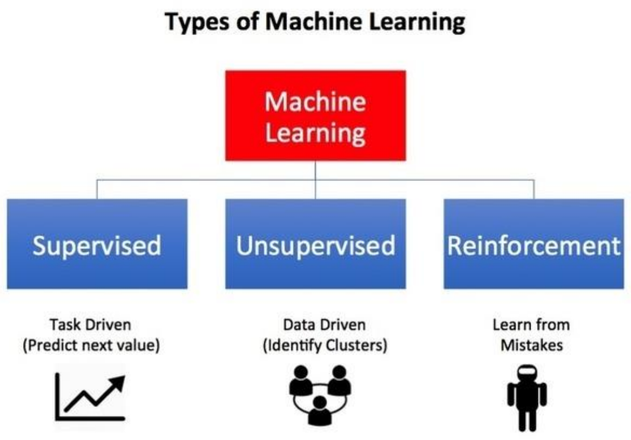

Types of Machine Learning
Machine Learning
Supervised
Unsupervised
Reinforcement
Task Driven (Predict next value)
Data Driven (Identify Clusters)
Learn from Mistakes

Figure 30. Different categories of ML problems [112].

3.3 What is learning?

Traditional statistical frameworks explain many aspects of learning algorithms, and provides mathematical proves for the feasibility of learning [113,114]. Shai Shalev-Shwartz and Shai Ben-David [115] presented examples to explain the basics of the learning process and some of the most fundamental issues in ML. The first example is how rats learn to avoid poisonous food. Rats, when they encounter a new food with a new look and smell, will first eat very small amount of the food and will process the physiological effect of it. If the food results a negative effect the new food will be associated with the illness, and the rats will avoid the food. The animal used experience to detect a safe food. If the experience was negatively labeled, the animal predicts that it will also have a negative effect. Now, consider writing a program to detect spam emails. A naive solution is to memorize all previous spam emails labeled by user, and when a new email arrives the machine will search the spam set to find a match. If there is a match the new email is a spam, otherwise it will be moved to the inbox folder. While learning by memorization is sometimes useful, it lacks an important aspect of learning – the ability to generalize. A successful intelligent learner should be able to achieve a broader generalization from individual examples. The ability to generalize, in that sense, refers to the abstract term of intelligence. One of the factors that makes humans more intelligent than other animals is their exceptional ability to generalize. Humans, as young as two years old, have the capacity to appreciate features in one object and to generalize it to other instances. For example, a child who has been showed a picture of a real elephant, can easily identify the abstract image of an elephant, although the two pictures were substantially different (see Figure 31).

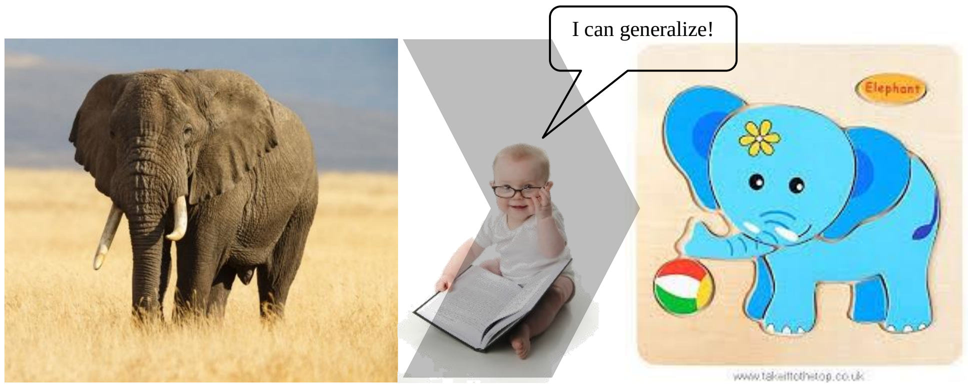


Figure 31. Concept of generalization and intelligence.

Another issue arises when the learner has a false conclusion. Pigeon superstition experiments by Skinner is a good example to illustrate this concept [116]. Skinner placed a series of hungry pigeons in a cage attached to an automatic machine that delivered food to the pigeon “at regular intervals with no reference whatsoever to the bird's behavior.” He discovered that the pigeons associated the delivery of the food with whatever chance actions they had been performing as it was delivered, and they subsequently continued to perform these same actions. “One bird was conditioned to turn counter-clockwise about the cage, making two or three turns between reinforcements. Another repeatedly thrust its head into one of the upper corners of the cage. A third developed a 'tossing' response, as if placing its head beneath an invisible bar and lifting it repeatedly. Two birds developed a pendulum motion of the head and body, in which the head was extended forward and swung from right to left with a sharp movement followed by a somewhat slower return” [117]. While humans rely on common sense to filter out random meaningless learning conclusions or patterns, well defined principles are needed to guide a machine out of reaching meaningless conclusions in the learning process. In other words, a algorithm should be able to learn the pattern in the data while ignores the pattern in the noise. Bias and variance are two concepts that help to reach this goal.

3.4 Bias-Variance Tradeoff

The goal of a learning model is to find a function $\hat{f}(x)$ that approximates a target function f(x) as well as possible. If the available data has zero mean noise with variance σ^2 (irreducible error), for real-valued targets and using mean squared error, expected error on an unseen sample (x, y) can be decomposed as follow:

$$E\left[\left(y-\hat{f}(x)\right)^2\right]=\left(E\left[\hat{f}(x)\right]-f(x)\right)^2+E\left[\hat{f}(x)^2\right]-E\left[\hat{f}(x)\right]^2+\sigma^2 \tag{4}$$

The left side of the equation is the expected error between the hypothesis $\hat{f}(x)$ and the target values. In the learning process the expected error will be minimized. The right side of the equation consist of three parts, bias $\left(E\left[\hat{f}(x)\right]-f(x)\right)^2$, variance $E\left[\hat{f}(x)^2\right]-E\left[\hat{f}(x)\right]^2$, and irreducible noise. Figures 32 and 33 illustrate the concept of bias and variance. High bias in a model indicates that it is diverging from the target like the superstitious pigeons who have learned patterns that are far from the target. Low bias is the symptoms of a too simplistic model. High variance indicates that the model is unstable, and instead of learning the general patterns, memorizes each data points which is a symptom of a model that is too complex.

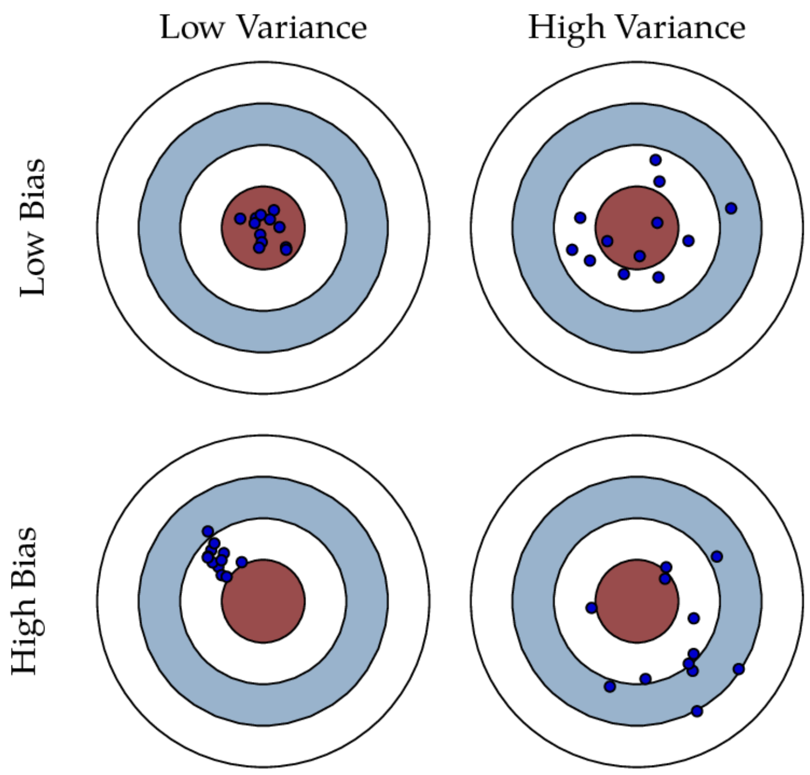


Figure 32. Bias and variance illustration.

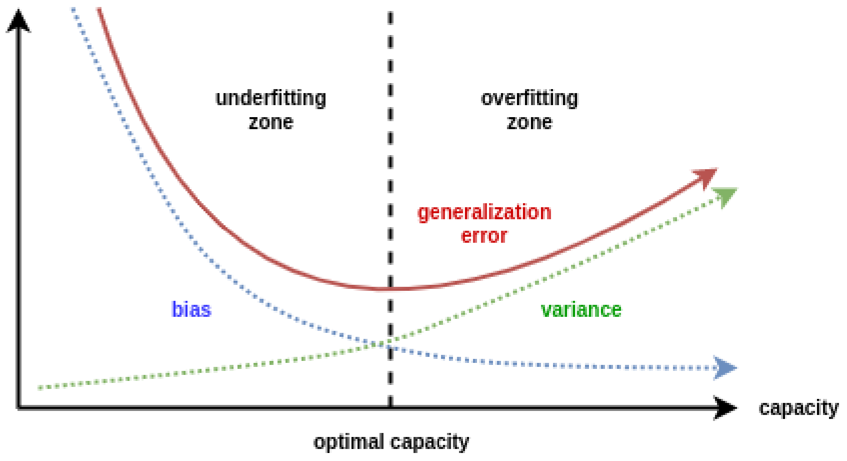


Figure 33. Bias and variance tradeoff [118].

As mentioned in the previous subsections, the purpose of learning from experience is to learn to generalize. By increasing the complexity of a model, more sample are needed to effectively introduce a function that can generalize well. Having over complex models will lead to high variance and over-fitting. Such models will not generalize well and will have low performance in during testing. On the other hand, a model which is too simple for a given data set, does not have the ability to learn the complex structures that might be represented in the training data. The amount of available data that one can use is a critical factor for choosing the complexity of a model. To illustrate this point, consider the following example [113]. Assume that the target function is $f(x)=\sin(\pi x)$, and just two data points are given for training the model and approximating the function. Next, consider two models with different complexities, constant ($\mathcal{H}_0$) and linear ($\mathcal{H}_1$). Here the question is which of these two models will provide a better approximation based on the given data. Depending on where the two points are located, different models can be proposed. Figure 34 shows two of the possible models. Considering the possibility of the two points anywhere on the target function, a set of hypotheses can be generated for constant and linear models (see Figures 35 and 36 where $\bar{g}(x)=E\left[\hat{f}(x)\right]$). To compare the two model, bias and variance values are calculated and shown in Figure 37. It can be observed that the bias of the linear model is less than constant model, but its variance is significantly larger. Thus, the constant model is a better model for approximating the sinusoidal function given two data points. This may sound counterintuitive, since the linear model is better for approximating the sinusoidal function; however, when the number of given training data points is limited to two points, the linear model overfit the data and consequently result is higher level error. By increasing the number of data points the linear model becomes outperforms the constant model.

The notion of using the simplest model come from the famous principle of Occam's razor, which states: that “among competing hypotheses, the simplest is the best” [119]. One of the methods that reduces the complexity of a model and the variance is regularization. Regularization put some extra constraints on the parameters of the model and, consequently, reduces the complexity of the model. There are different ways of applying regularization: early stopping, L1 regularization, L2 regularization, elastic net, max norm, and dropout are some of the methods that have a regulatory effect [120]. In this study, the L2 and dropout regularizations are implemented.

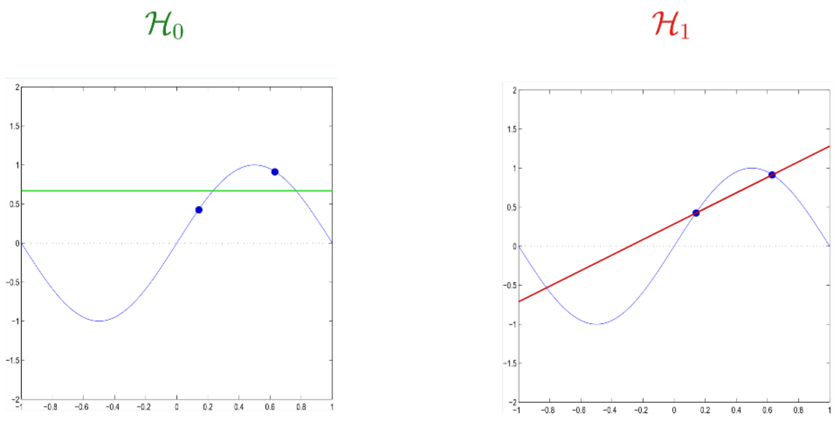


Figure 34. Approximating the sinusoidal function based on two learning data points [113].

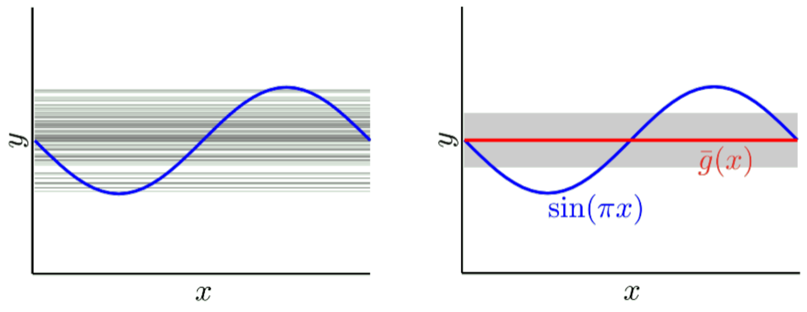


Figure 35. Constant approximation of the target function based on two learning data points [113].

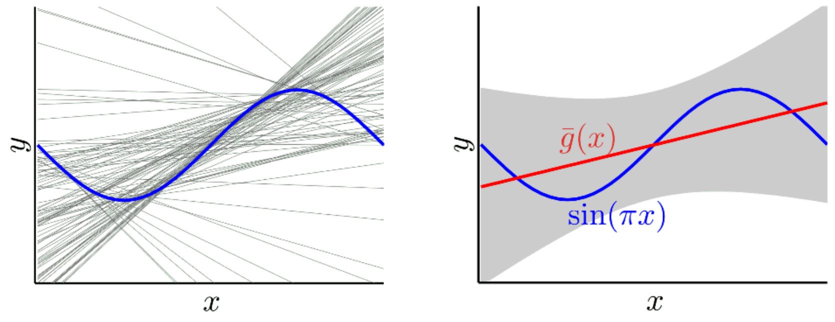


Figure 36. Linear approximation of the target function based on two learning data points [113].

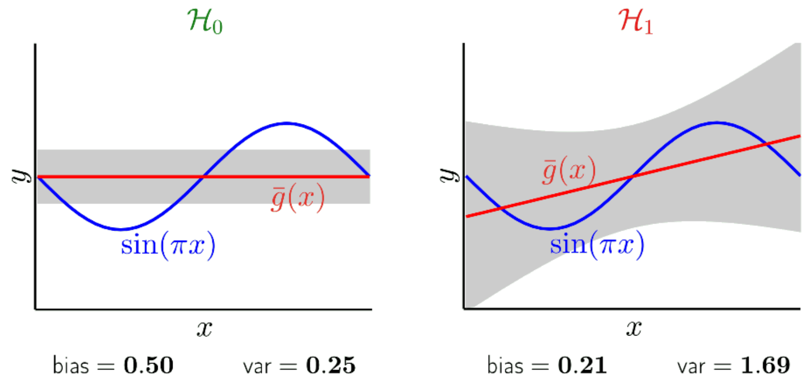


Figure 37. Bias and variance of constant and linear model [113].

3.5 **Interpretation vs. Performance**

Although the primary interest of most practical ML examples is to improve the model performance, a secondary interest may be to interpret the model and understand why the model works. For instance, in the case of choosing treatment therapies for a cancer patient, the doctor and the patient might like to discuss other factors such as potential side effects and survival rates. In this case, not being able to interpret the model may be considered as unethical. The unfortunate reality is that in a quest to have higher performance, the complexity of the resulting models increase and their interpretability becomes challenging [121]. In most of the real-word problems the primary goal is to have a better prediction; therefore, if a complex model can be validated, the interpretability will be sacrificed.

3.6 **Convolutional Neural Network (CNN)**

In this section, the basic concepts of NNs are presented, then different components of CNNs are discussed, and the advantages of each architecture are explained. Figure 38 shows a typical NN where an input *i* is a single vector of the features x_k. The input is fed into a sequence of hidden layers to predict an output $\hat{y}$. Each hidden layer consists of a set of nodes (neurons) where each node is fully connected to all nodes in the previous and next layers. Each node in a layer functions independently and does not share any information with other nodes. At each node, the output of the previous layer $a_k^{[l-1]}$ is multiplied by a weight $\omega_{jk}^{[l]}$ and added to a bias term $b_j^{[l]}$. Then, the result is fed to an activation function $g^{[l]}$ to determine the output of the node $a_j^{[l]}$. The general formulation for output of each node is

$$a_j^{[l]}=g^{[l]}\left(\sum\nolimits_k \omega_{jk}^{[l]}a_k^{[l-1]}+b_j^{[l]}\right) \tag{5}$$

where $a^{[0]}$ is the input vector. In this example, the last fully-connected layer $a^{[3]}$ is called the "output layer", and in classification problems it represents the class probabilities. Both the weights $\omega_{jk}^{[l]}$ and the bias terms $b_j^{[l]}$ are the parameters of the model that are determined during training.

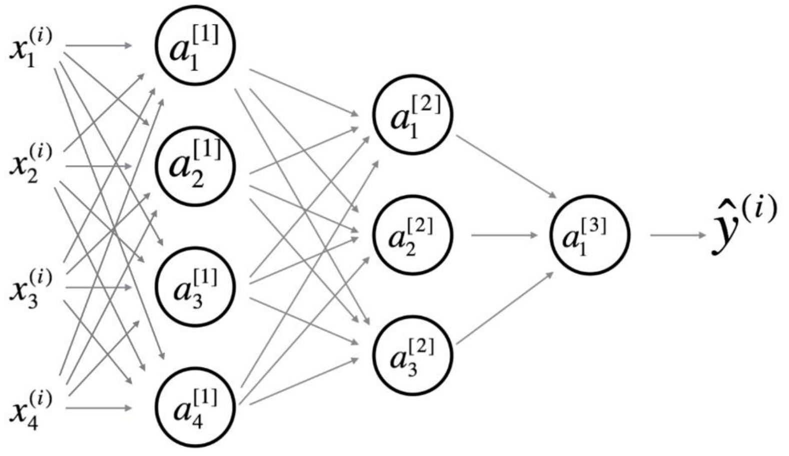


Figure 38. A typical Neural Network with two hidden layers.

3.6.1. Working with Digital Images

To process a digital image with a typical network a tensor with the order of 3 (a matrix with 3 channels) is converted to a tensor with the order of one (a vector). For instance, an image with 100×100 pixel resolution and 3 channels (red, green, and blue) turns to a vector with 30,000 elements. Each element is an input feature or variable. For building the NN model, 30,000 weight parameters would be required for each node in the first layer of the network. It follows that the number of parameters will increase when using larger images or by adding extra nodes to the first layer. This framework is not an efficient way of developing NN models for images. A CNN is another class of NN that takes advantage of the shape of the inputs and designs an architecture that uses the weights more efficiently. CNNs leverages two important ideas to help improve the performance of the network: sparse interactions and parameter sharing.

In a typical NN, every output unit $a_j^{[l]}$ interacts with every input unit $a_k^{[l-1]}$; however, CNNs typically have sparse interactions. This is accomplished by choosing a smaller filter size than the input. For instance, the input image might have thousands of pixels, but small, meaningful features such as edges can be detected with filters that sample only tens or hundreds of pixels. This reduces both the number network parameters and the required memory while improving statistical efficiency [104]. Unlike a traditional NN where weights are used exactly once in one forward pass, in CNNs, weights apply to different part of an input (parameter sharing) [104]. This strategy is based on the reasonable assumption that if a filter (feature detector) is useful in one part of an image, then it may also be useful in a different part of the image.

A deep CNN architecture is developed by assembling (stacking) several layers, such as input, convolution, pooling, fully connected, and output layers. There are other techniques such as a dropout layer that that can enhance performance and avoid overfitting of the data. Details on each of the layers and their configuration in the CNN are explained in the following sections.

3.6.2. Convolution Layers

In CNNs the main computational elements are the convolution layers. Each convolutional block includes a set of filters with learnable weights. These filters convolve with the output of the previous layer and search for a useful pattern or feature in the entire image. The network designs each filter in a way that minimizes an error function (objective function). A convolution operation in a CNN is the same as the cross-correlation operation (convolution operation without flipping the filter) in 2D signal processing (image processing). Figure 39 illustrates the convolution operation on a 2D image I of 5×5 pixels, with a filter K of size 3×3 pixels. The result of the convolution operation when passing the filter one pixel in each step to compute the next pixel in the output (which is called stride of one), is smaller than the input image. To have consistent size, a zero-padding technique is used on the edges of the input (see Figure 39). The result of the convolution operation is added to a bias b and passed through an activation function a to compute the output of the convolutional layer. The formula of the convolutional layer $\mathrm{Conv}(I, K)_{xy}$ for a pixel in (x, y) coordinate is

$$\mathrm{Conv}(I,K)_{xy}=a\left(b+\sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d}K_{ijk}*\;I_{x+i-1,y+j-1,k}\right) \tag{6}$$

where h and w are the size of the filter and d is the number of channels in the input.

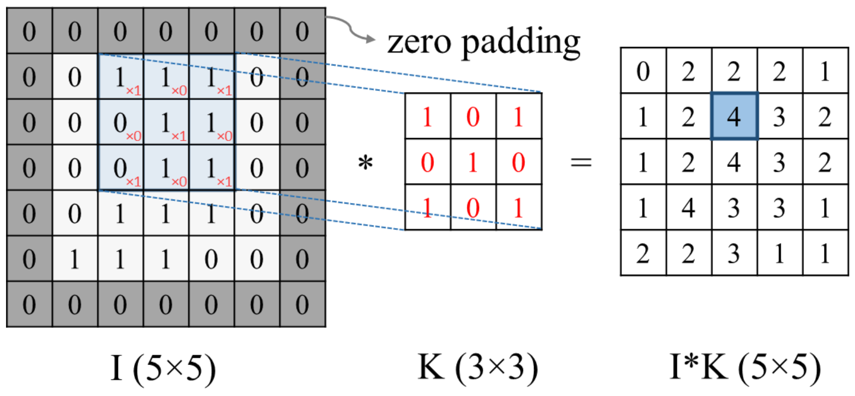


Figure 39. Convolution operation in CNN.

3.6.3. Activation Function

To introduce nonlinearity, a nonlinear activation function should be implemented in the network. Figure 40 shows three common activation functions used in DL. In the early days of DL, the sigmoid function was very popular but now the tanh function has been shown to have better performance [122]. One drawback of these two functions is that they saturate at the tail of the function and the gradient at these regions is almost zero which significantly slows down the learning process when a gradient based optimizer is used. In the last few years, the rectified linear unit (ReLU) function (a non-saturating function) has become very popular. Using the ReLU function has been shown to improve the performance of the network [123,124]. In this study, the ReLU function is used for all activation functions except for the activation of the last layer of the network. To classify the input data, a softmax activation function will be used in the last layer of the CNN network. The softmax function $s_i(\vec{x})$ for class i which returns the probabilities of the input belonging to each of the classes is given as

$$s_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{2} e^{x_j}} \tag{7}$$

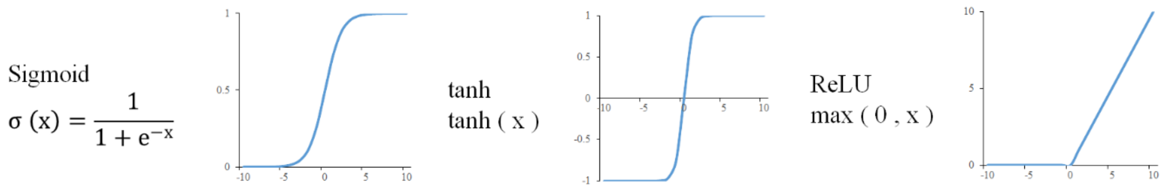


Figure 40. Common activation functions in DL.

3.6.4. Pooling Layers

CNNs typically use pooling layers to reduce the size of the input layers which will speed up the computation and increase the robustness of feature detection. Among different pooling options, max-pooling and average-pooling are common in DL. Max-pooling has been shown to be vastly superior for image-like data [125]. In this study, all the pooling layers are max-pooling unless otherwise stated. Figure 41 illustrates the max-pooling mechanism using a 2×2 window and stride of 2. As shown in the figure, the maximum value is selected within the 2×2 window as it passes through the input data. The 2×2 window shifts by two pixels, and the process is repeated over the whole input. By performing this operation, the size of the input data is reduced (in this example, the output data is half the size of the input data).

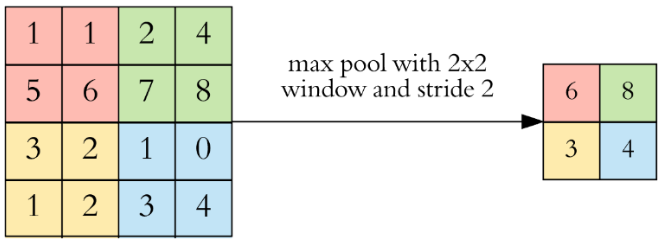
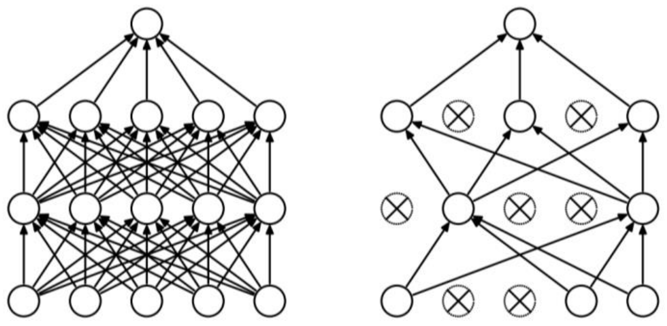


Figure 41. Max pooling mechanism [126].

3.6.5. Dropout

Dropout is a technique that helps to prevent overfitting and provides a way of combining many different neural network architectures [127]. The term "dropout" refers to randomly dropping out neurons in a NN. Figure 42 shows how dropped-out neurons are temporarily removed from the network, along with all their incoming and outgoing connections. In this study, dropout will be implemented at each fully connected layer using the recommended probability of 0.5 [127].



Standard NN (b) After applying dropout

Figure 42. Dropout Neural Net Model. (a) Standard neural net with 2 hidden layers. (b) An example of applying dropout. Crossed units have been dropped [127].

3.6.6. Cost Function

The main objective of training a CNN is to find a set of weights and biases (parameters) which minimizes the error between prediction and the actual value. A loss function is defined to quantitively measure the error. Here, categorical cross entropy (Equation 8) will be used as the loss function L_i to estimate the difference between the true class y and the probability distribution of predicted class ŷ for one image. The probability distribution of the predicted class is calculated by softmax function.

$$L_i(\hat{y}_i, y_i) = \sum_{i=1}^{k} -y_i \ln \hat{y}_i \tag{8}$$

It is typical to use one-hot encoding for introducing the image labels to the network. For example, to encode binary classification, (0, 1) and (1, 0) are used for class one and two. The output of the network is the probability of each class, $(\hat{y}_1, \hat{y}_2)$. For instance, with this definition, the output of (0.3, 0.7) for an image means that 30% chance the image is class one and %70 chance the image is class two. Assuming the actual class is one, the loss value for this example based on Equation (8) is 0.36 (-0* $\ln(0.3)$ -1*ln(0.7)). For the same example, a bad prediction such as (0.6, 0.4) will result in a loss value of 0.92 and for a good prediction such as (0.05, 0.95) the loss is 0.05. The cost function which is the average of the loss function applied to all images (N is the number of images) is

$$\text{Cost} = \frac{1}{N} \sum_{i}^{N} L_i(\hat{y}_i, y_i) \tag{9}$$

To apply regularization in the model, the L2 regularization formula, which defines as the sum of the squares of the feature weights, is added to the cost function with an associated coefficient.

3.6.7. Optimization

In general, the learning problem is an optimization problem. The objective of the optimization is to find the best parameters (weights and biases) that minimizes the cost function. For large NNs there is no closed form optimization solution, so the optimization problem is solved with iterative algorithms that use a variety of methods, such as gradient descent. The search space of a common NN is non-convex, and it is reasonable to use a variation of a stochastic gradient descent algorithm. The proposed CNNs will have millions of parameters that need to be adjusted to minimize the cost function. In this study, the Adam (derived from adaptive moment estimation) algorithm is utilized to minimize the cost function. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions. This optimization algorithm is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for non-convex optimization problems in ML that are large in terms of data and/or parameters [128,129]. Thus, Adam optimizer is implemented as the optimization algorithm in the training phase [128,130]. Since a gradient based optimizer is being used, calculations are required for the gradient of the learning parameters. Backpropagation is an effective algorithm for calculating the gradients of the parameters using a recursive application of the chain rule along with a computational graph. After each forward pass through the network, the cost function is calculated. Based on the value of the cost and the inputs, derivatives of the learning parameters are calculated using back propagation. These derivatives are used in the Adam optimizer to update the learning parameters.

Vectorization increase the computational parallelism which results in faster computation when using graphics processing unit (GPU) processors. However, training on a large data set not only requires a large amount of memory for vectorization, it slows down the computation. To solve the issue, the training data is split into smaller mini-batches [131]. While the use of large mini-batches increases the available computational parallelism, small batch training has been shown to provide improved generalization performance and allows for a significantly smaller memory footprint, which might also be exploited to improve machine throughput [132]. Masters and Luschi showed that smaller mini-batch sizes provide more up-to-date gradient calculations, which yields more stable and reliable training [132]. Thus, this study will use mini-batches of 32 images (N in Equation (9) will be 32 instead of number of all images).

3.7 The Overall CNN Architectures

There are different ways of assembling convolutional layers to build a CNN model. The most straightforward way of improving the performance of a network is to increase the size of the network by increasing the depth (number of layers) and width (number of neurons) of the network [133]. However, a larger network typically requires not only more computational resources, but also increases the number of parameters which makes the network more prone to overfitting, especially if the number of labeled examples is limited. To deal with these issues researchers have proposed different CNN architectures [62,64,133–143]. Canziani et al. analyzed and compared some of the popular CNN architectures for practical applications [144]. Figure 43 compares different CNN models for image classification for ImageNet challenge [145]. This graph shows the accuracy of the first prediction (Top-1) versus the number of operations and number of parameters in the models. In this study, two CNN models were developed based on the architectures introduced by the Visual Geometry Group (VGG) [135] and Google Inc. [133]. These two architectures are explained in the following sections.

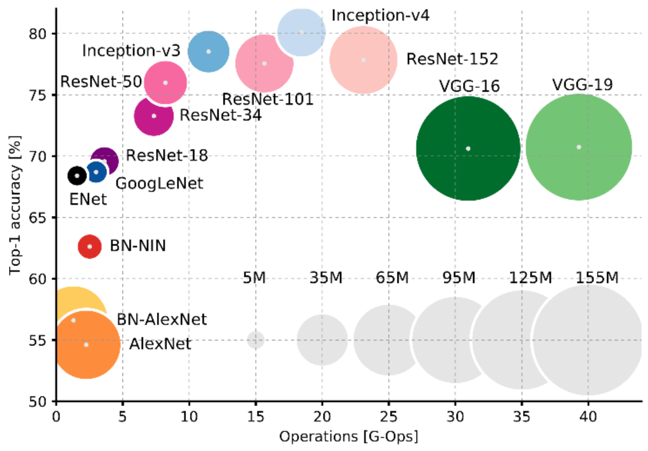


Figure 43. Top-1 accuracy vs. operations (in giga), and size of the model (number of parameters) [144].

3.7.1. VGG Network

In 2015, the VGG at University of Oxford proposed several networks and investigated the effect of depth of CNNs on their accuracy for a the ImageNet Challenge image classification problem [135]. VGG examined six CNNs with different depths (number of layers). Instead of using relatively large filter sizes (receptive fields), they used small 3×3 receptive fields throughout the whole network. They showed that stacking two or three 3×3 convolutional layers (without pooling layer in between) has an effective receptive field of 5×5 and 7×7 which makes the network more flexible and discriminative while decreases the number of parameters. This concept is illustrated in Figure 44. Applying a 5×5 filter (with no zero-padding) to a 6×6 pixel image will result in a 2×2 output. In this layer one 5×5 filter with 25 parameters is used to generate the result, and the number of operations is 100 (25*4). Now, instead of one 5×5 filter, the layer is broken to two layers of applying 3×3 filters. Applying a 3×3 filter to a 6×6 pixel image will result in a 4×4 output, and applying another 3×3 filter will result in a 2×2 output. With this method the number of parameters is reduced to 18 (2 filter with 9 parameters). The number of operations for the second method is 180 (9*16+9*4). In this method, by stacking two layers of 3×3 convolutional layers, the similar effect of a 5×5 convolutional layer is achieved, while the number of parameters is reduced. VVG also showed that their deepest networks with 16 and 19 weight layers achieved the best accuracy. In this study, a network with 16 learnable layers based on VGG-16 is designed as shown in the Figure 45. The network has 13 convolutional layers denoted as "conv <receptive field size> - <number of channels> - <stride length>", 5 max pooling layers denoted as "maxpool", and 4 fully connected layers denoted as "FC- <number of nodes>" (the last fully connected layer calculates the probability of each class and is not considered as a weighted layer). The original VGG-16 presented in [135] has more than 138 million parameters. Since the data set in the present study is not large enough to properly train a network with this number of parameters, the last layers of the original network are adjusted to reduce the number of parameters to less than 20 million parameters. While this network has a simple structure, it performs well in image classification. The number of parameters in each layer along with other details are presented in Appendix 1. For multiclass classification the last layer needs to be adjusted based on the number of classes.

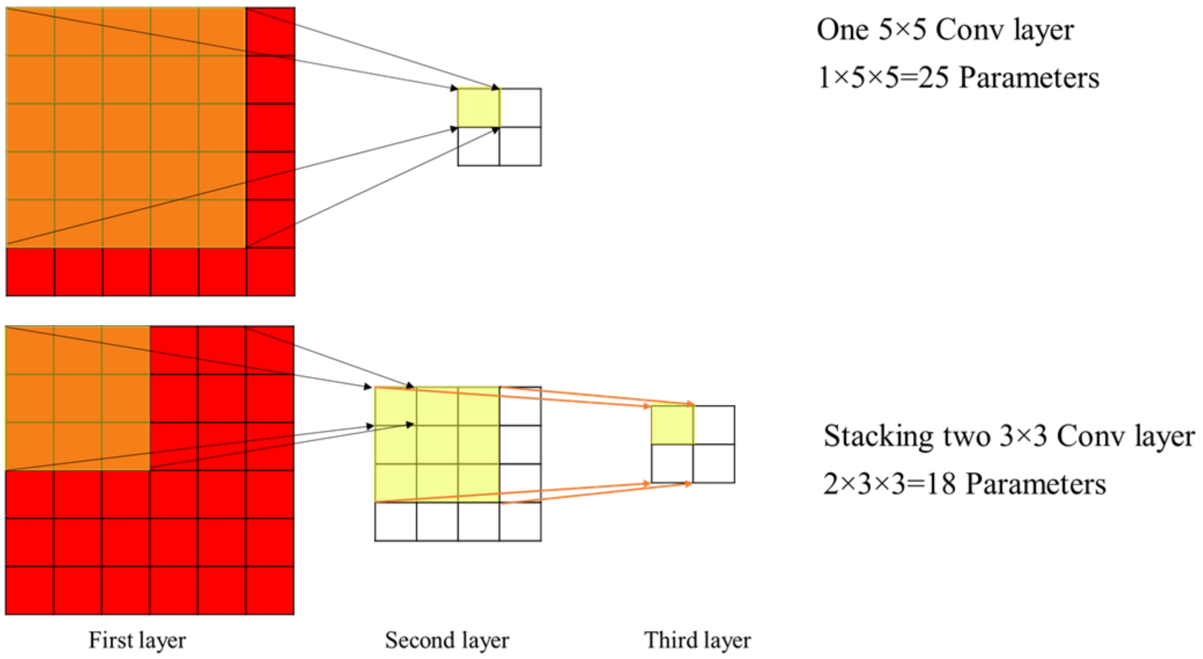


Figure 44. The effect of stacking convolutional layers.

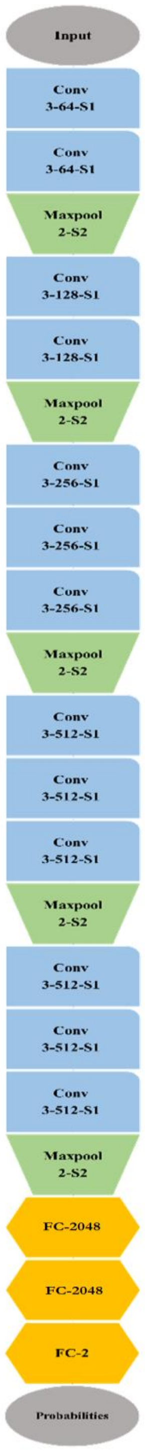


Figure 45. VGG-16 based model.

3.7.2. GoogLeNet

The core of GoogLeNet architecture is the Inception module [133]. In the Inception module, instead of selecting one specific filter size for the convolutional layers, the convolutional layer is applied with filter sizes of 1 and 3 and 5 to an input and the results are concatenated to build an output of the module. Figure 46 illustrates an example of the concepts of the Inception module [146]. Using 1×1 filter size enables the model to change the depth of the output and reduces the number of operations. Based on the GoogLeNet architecture (see Figure 47), a network is designed by stacking 3 convolutional layers and two pooling layers at the beginning, followed by nine inception modules with two pooling layers between Inception modules 2 and 3, and another between 7 and 8. The output of last inception module is a 7×7×1,024 tensor which feeds to an average pooling layer and then to a fully connected layer. This network, unlike the VGG network, does not have fully connected layers at the end of the network which reduces the number of parameters to 5.4 million parameters. Szegedy et. al modified GoogLeNet architecture by incorporating the idea of stacking smaller convolutional layers from VGG group and a technique of normalizing output of each layer [147]. They called the new network Inception-V3 (version 3 of Inception). This network is modified by adding two fully connected layers at the end of the Inception-V3 which increased the number of parameters to 30 million. The number of parameters in each layer along with other details of the designed network are presented in Appendix 1.

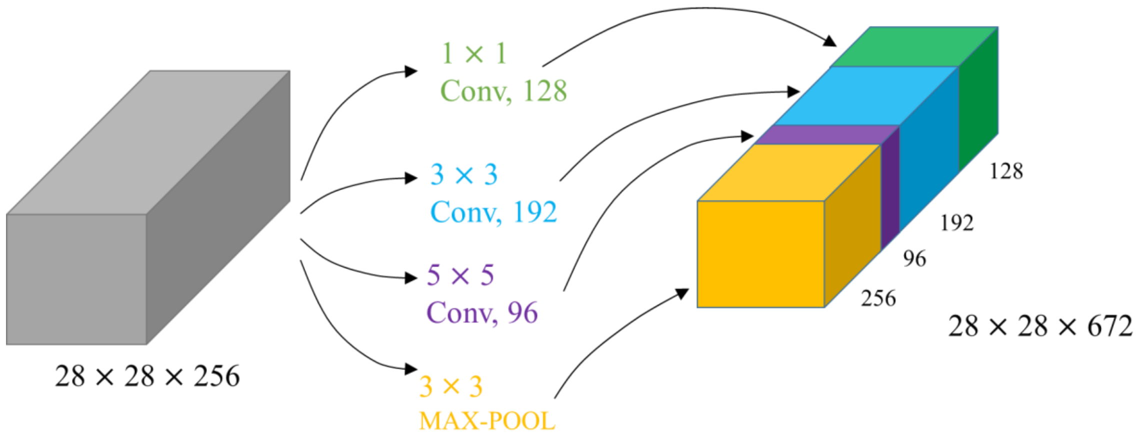


Figure 46. An example of Inception module for an input size of 28×28×256.

3.1 Transfer Learning

In the first step of training, the weights in the model need to be initialized. There are different ways for generating the initial values. Since in learning process a gradient based optimizer is used, selecting proper initialization can mitigate the chance of exploding or vanishing gradients [148,149]. Instead of randomly generating the initial weights, it is possible to transfer weights from a pretrained network on different data set. In this method the knowledge that has been gain through training the model on a large data set can be transferred to the new model. This way, the model will have a good starting point for learning. Gao and Mosalam investigated this method for image-based structural damage recognition [150].

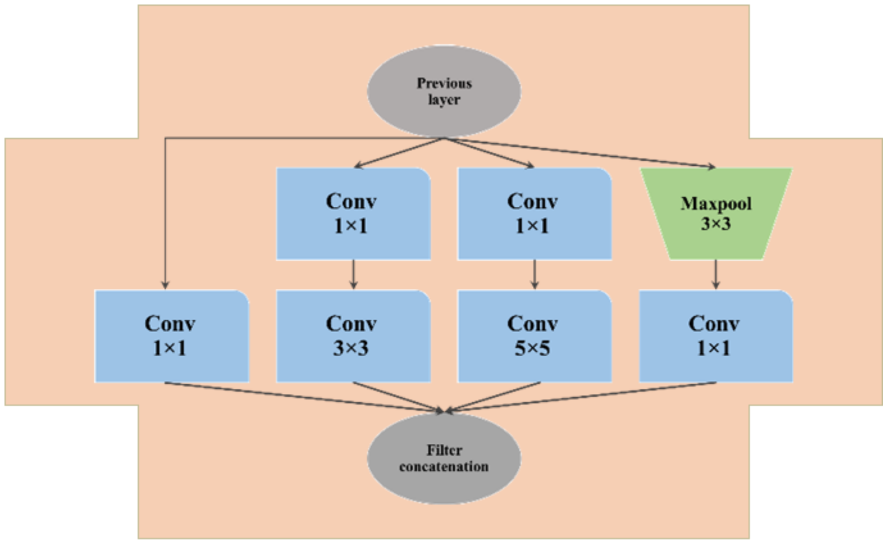


Figure 47. Inception module with 1 by 1 filter.

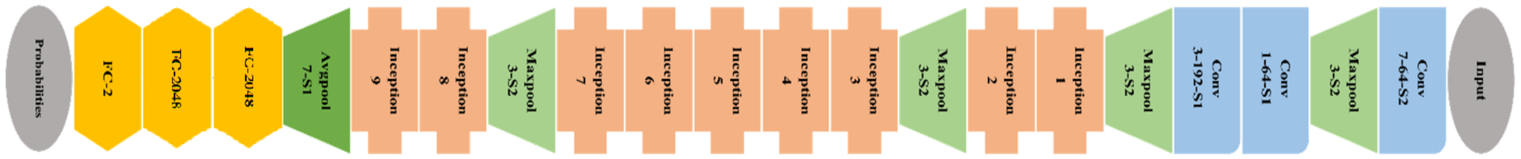


Figure 48. GoogLeNet based model.

In this study, the weights that are publicly available for image classification models are transferred to the model [151]. These weights are the result of training on large image database (ImageNet) with more than 14 million images of different objects [145]. Depending on the number and the nature of an image data set, there are several common methods for implementing transfer learning. In one method, after transferring the pretrained weights, the whole model is freezed (the learnable weights are disabled) except the last few layers. The model is not changing its weights in the freezed layers. This method works well when the number of data points are very limited, and they have the same characteristics of the data the model has been pretrained. Therefore, with this method the number of learnable parameters are limited to the parameters of the few last layers which reduces the chance of overfitting. By getting access to more data more layers can be unfreezed. In working with a data set that is very different than the pretrained model data set, it is better to train the whole network. In this study, since the nature of the images of pavements are drastically different from pictures of objects and animals (ImageNet images) the model exhibits a better performance when the whole network considered for training after transferring the weights. Thus, the pretrained weights are used as the initial weight for the whole network except the last layers which initialized with random weights.

3.2 Implementation

Training a DL model is a computationally expensive task and requires a large input data set. Additionally, building a complex DL model is challenging to program. Data management, computational power, and software framework are three important aspects of implementing a DL model. Some of the challenges to developed DL are discussed in the section.

3.2.1. Data Management

In practice, DL models require massive amounts of data for the training process. Storing and reading data for large data sets (more than the size of a hard disk) is a challenging problem. In this study, the size of the data set is not large enough to raise any practical issues (less than 500 GB). However, this could easily be a big data problem if this method is implemented to the large system of roadways. In general, data is considered to be “big” when it cannot be stored on a typical data storage system (typically more than 10 TB). To store GSV images for the 4 million miles of road in the US would require 4 PB (4,000 TB) of data storage capacity. In addition to the hardware, data management software should be implemented to make parallel writing and reading of the data possible.

3.2.2. Computing Power

In last decade, there has been a great advancement in the capacity of computing power . There are several companies that provide processing units such as Intel, AMD and NVIDIA. In the past, the main processing unit for computation was a central processing unit (CPU) which is able to carry out the instructions of a computer program by performing basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions. IN general, a CPU can do any form of computation. A graphic computing unit (GPU) (see Figure 49) is a processing unit specially designed and optimized to perform single-instruction-multiple-data operations needed to display (render) graphics much faster than a regular CPU. In a simple terms, GPUs are suitable for processing simple operations in parallel. Most of the operations in DL are matrix multiplication which can be easily parallelized. This ability makes the GPU processors much faster in training a deep network. Johnson performed simple experiments on different CNN benchmark problems and showed how GPUs can speed up the training process. [152]. Figure 50 provides a comparative summary of his result. For these benchmark problems, the performance of a GPU is 60-70 times faster than a CPU.

Figure 49. A typical graphic computing unit [153].

Parallel programming could be challenging to optimize; thus, several GPU companies has developed platforms to help this process. CUDA (Compute Unified Device Architecture) is an extension of the C programming language and was created by NVIDIA to help perform parallel computing on NVIDIA GPUs. The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for DL programming. The cuDNN library provides highly optimized implementations for many standard layers such as convolution, pooling, normalization, and activation [154]. In the comparison showed in Figure 50, network models implementing cuDNN showed a speedup of 2 to 3.

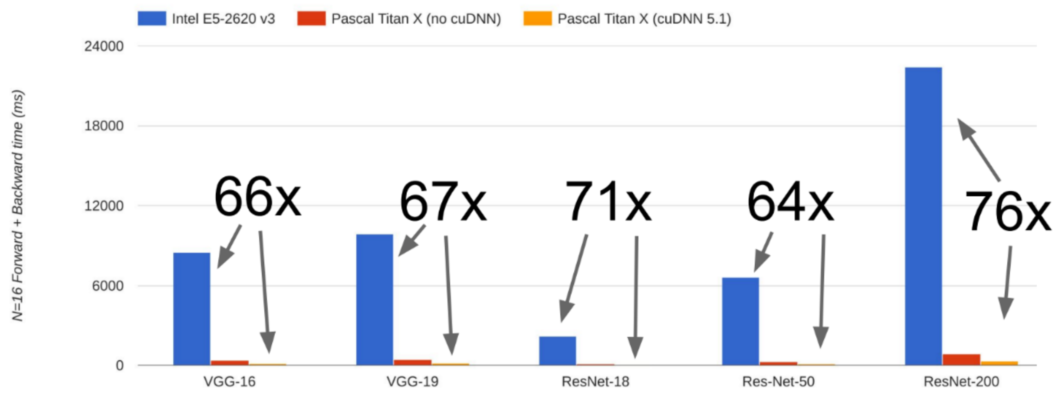


Figure 50. Comparison of CPU and GPU processing time in DL applications [152].

All the computations in this study were performed on a computer system with the following configuration:

CPUs: 2 processors of Intel (R) Xenon (R) Gold 6130 @ 2.10 GHz

RAM: 64 GB DDR4

GPU: NVIDIA, Quadro P6000 with 3,840 cores and 24 GB DDR5

3.2.3. Deep Learning Software Platform

Implementing a complex DL model can be a time consuming and complicated process. The are several software platforms that provide high level abstractions for implementing different blocks of a DL model. TensorFlow is an open source software library for high performance numerical computation [155]. TensorFlow has a comprehensive, flexible system of tools, libraries, and community resources that allows researcher to easily build their ML and DL models in many languages. It also provides a visualization toolkit for inspecting and understanding models and their performance. Keras is an open-source high-level DL API that uses TensorFlow as the backend. Keras is written in Python programming language, and allows for easy and fast prototyping of complex DL models for researchers and practitioners [156].

4. Chapter 4

Vision-Based Pavement Crack Detection

The objective of this study is to provide an economical, reliable, and accurate automated system for identifying and classifying cracks in pavements. In this study GSV is used as a source of locating and extracting images of pavements. Each image extracted from GSV is split into a series of small patches and examined as whether it displays a crack in the pavement. Since these GSV images are low resolution and are contaminated with significant levels of noise, a DL model is implemented to analyze these patches for cracks. Results of the DL model should provide information like a pavement condition index [149]. Two experiments were designed to investigate the proposed method. In the first experiment, a binary classification problem was designed to identify whether a patch had a crack or not. In the second experiment, the problem is extended to a multiclass classification to identify not only if a patch has a crack, but the type of the crack. In this section, the result of these two experiments are discussed and validated.

4.1 GSV Images for Crack Detection

One of the fundamental questions in this study was the feasibility of using GSV images for crack detection on pavement. In attempt to answer this question, the following concerns were addressed: first, will GSV images be updated in future to provide the required images? and second, will these GSV images provide a good representative of cracks in pavements?

To answer the first question, an analysis was performed on the sequence intervals Google used to capture images. After examining visiting several road sections, no organized pattern was found in the sequence of intervals. However, for high traffic and important roads such as interstate highways Google updated their images more frequently than low traffic roads. Figures 51 and 52 show different GSV images at the same location for different times. With the advances in camera devices and the expansion of the GSV project, it highly likely that Google or other companies will continue this imaging project in future.

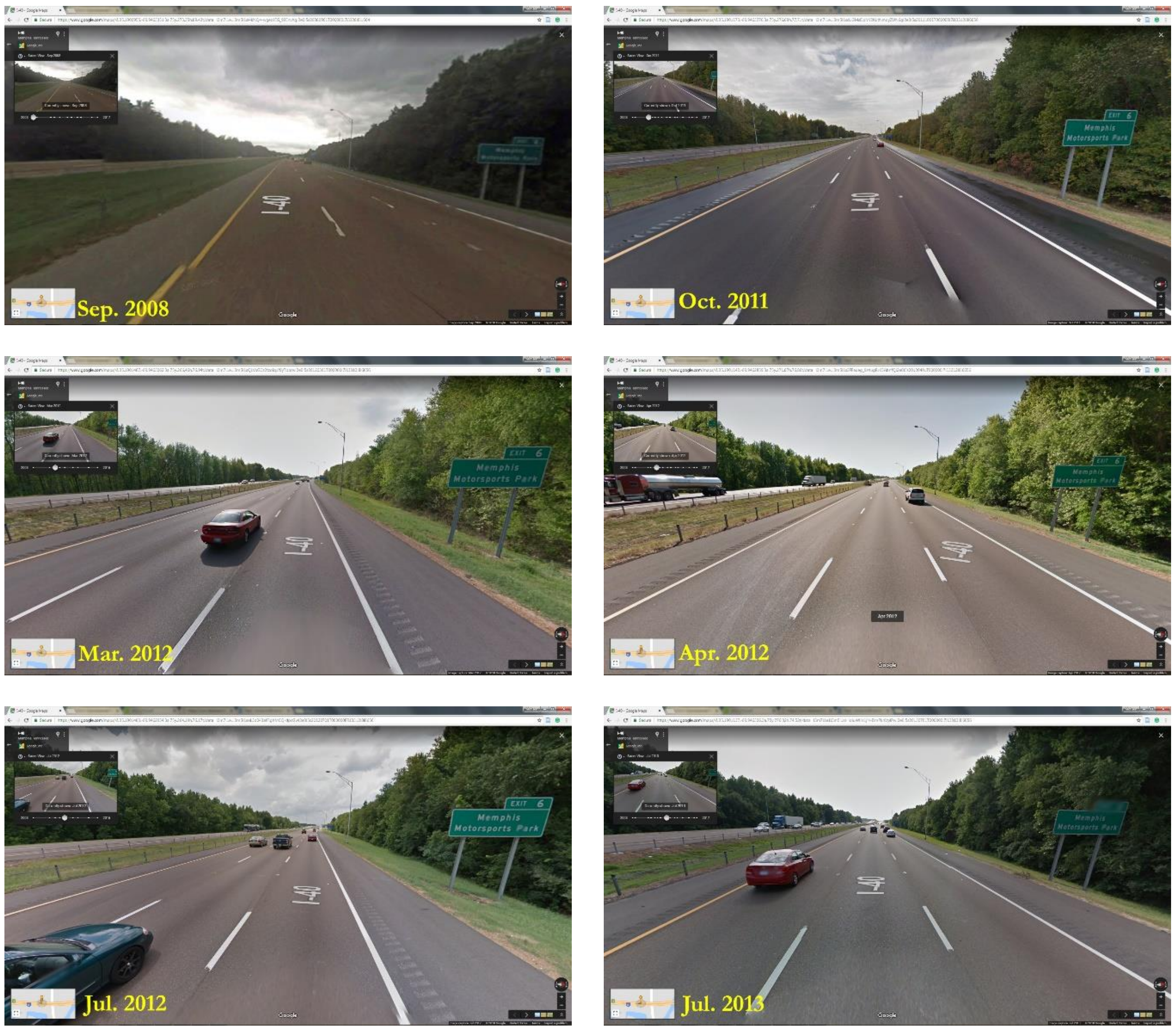


Figure 51. GSV images form the same location on I-40 at different times (part one).

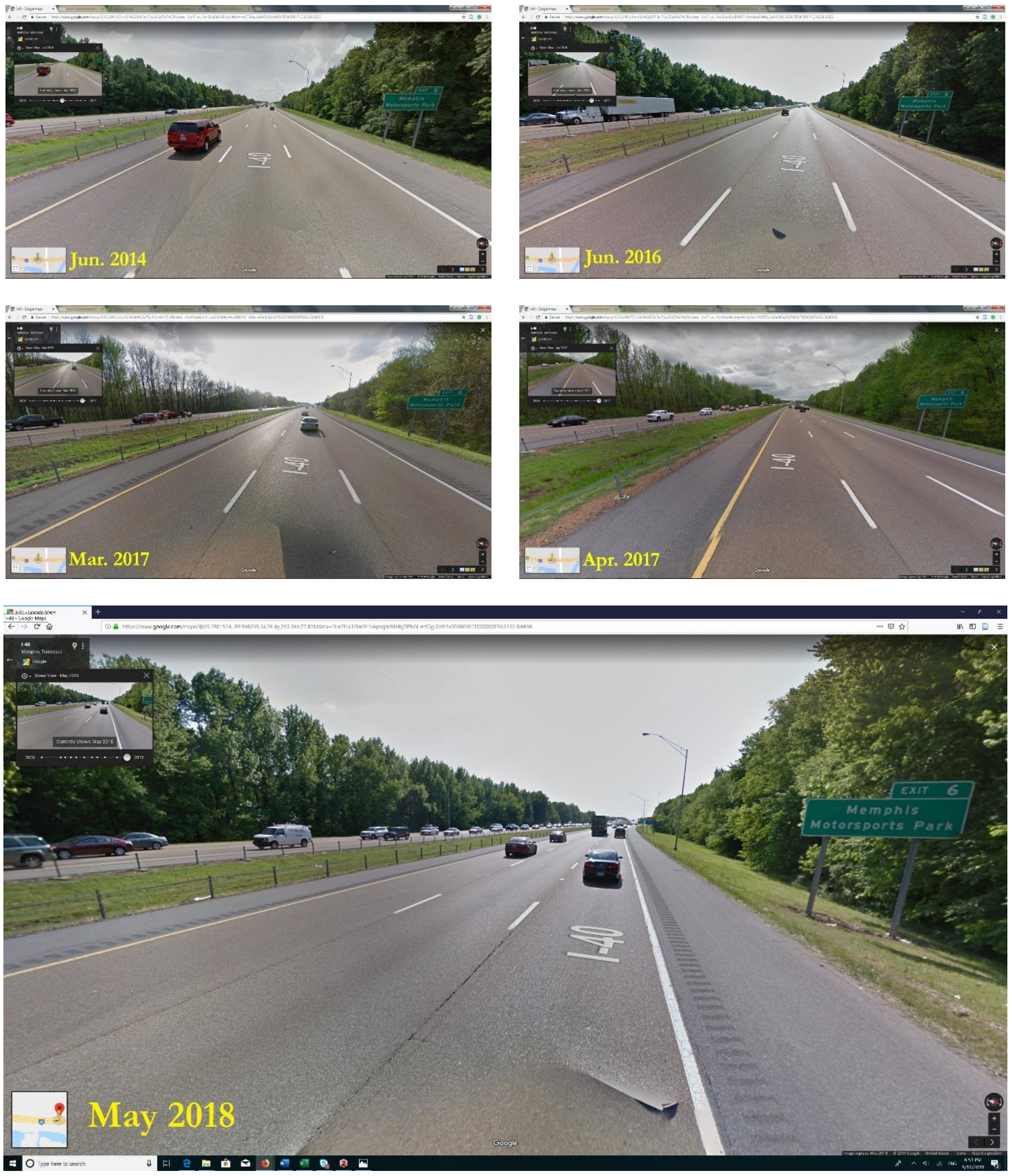


Figure 52. GSV images form the same location on I-40 at different times (part two).

The second question is investigated by considering the crack detection data developed by the ARRB Group using their iPAVe survey method [93]. ARRB has a fully automated crack detection system that uses 3D sensors combining lasers and high-speed 3D cameras to provide data on the quality and quantity cracks in pavement. In their system, each section of a roadway is divided into segments about 31-37 feet long (sometimes smaller depending on the geometry of the road) and then each segment is subdivided into10 cells (each cell is about 3 feet or longer). From their survey of the pavement, cracks in each cell are categorized as either longitudinal, transverse, or alligator cracks. The percentage of the crack in a segment was calculated by counting the number of cells with cracks divided by the total number of cells. This data was valuable in validating the results using GSV images. Although a GSV image and an iPAVe segment are not exactly the same size (in average each image covers 33.8 feet of the roads), each GSV image was split into a number of patches that closely matched the iPAVe data. Figure 53 shows the distribution of the cracked cells in 6.5 miles of Route 28 in near Sterling, Virginia. Since the changes from one image to another image or from one segment to another segment are drastic, the data was averaged, and the curve was smoothed to make the trend more visible. As shown in Figure 53, there is a high correlation between the iPAVe results and results based on GSV images. Some of the dissimilarities can be attributed to differences in pavement coverage contained in the GSV images.

Figures 54, 55, and 56 compare the iPAVe results with data from the GSV images for different types of cracks. It appears that there is a meaningful correlation between the two graphs especially for longitudinal and alligator cracks. In Figure 55, although the number of identified transverse cracked cells of the GSV images are lower than those based on the iPAVe estimate, the shapes of the graphs are meaningfully correlated. The lower estimate may be attributed to the difference in the definition of a transverse crack. Additionally, there is discrepancy in scale of the cells and the segments used in iPAVe and the patches and images from GSV.

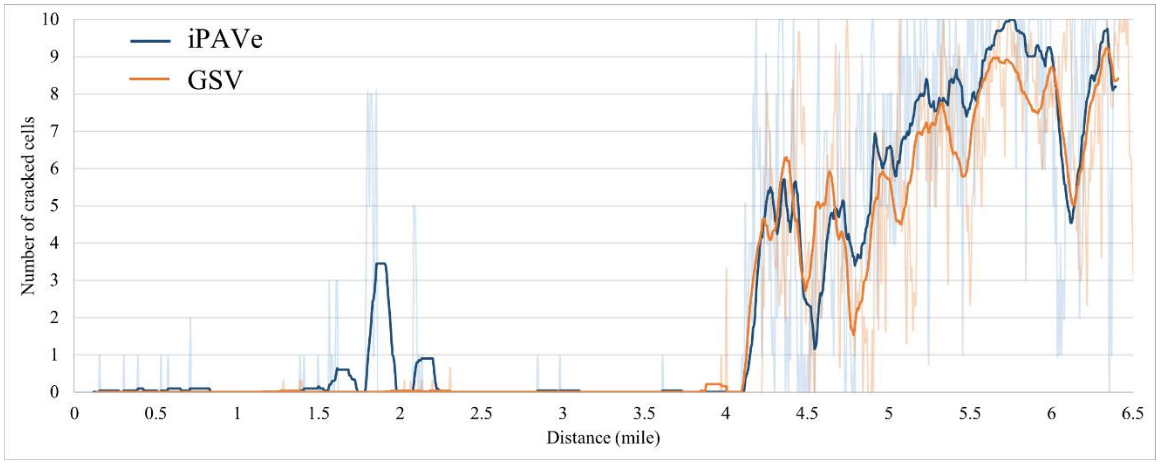


Figure 53. Comparison of total cracked cells evaluated by iPAVe and by GSV images.

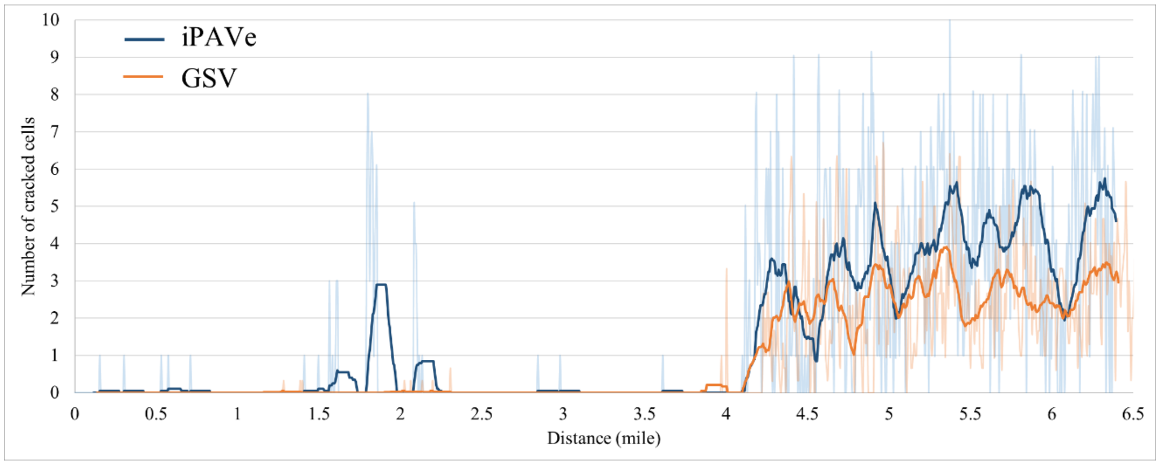


Figure 54. Comparison of longitudinal cracked cells evaluated iPAVe and by GSV images.

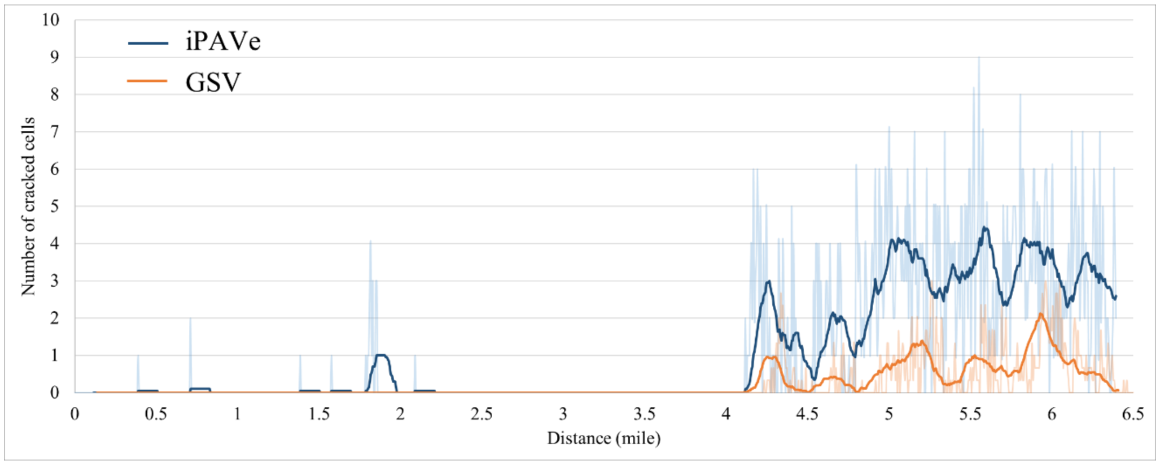


Figure 55. Comparison of transverse cracked cells evaluated by iPAVe and by GSV images.

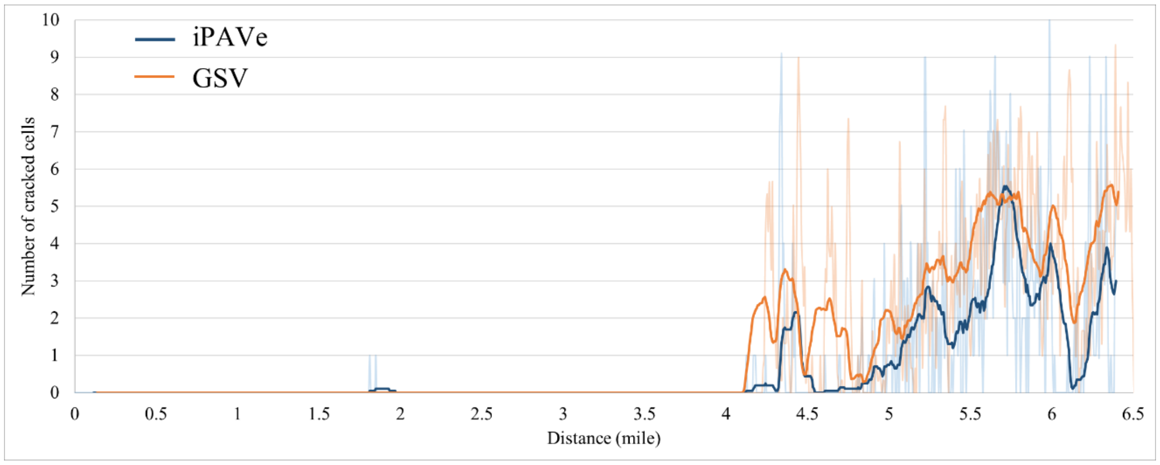


Figure 56. Comparison of alligator cracked cells evaluated by iPAVe and GSV images.

4.2 Binary Crack Classification Experiment

The first experiment is a supervised binary image classification problem designed to determine if there is a crack in a pavement image or not. Two CCN architectures were designed for this binary image classification problem. In general, designing an effective deep CNN network is a complex process and requires theoretical and practical knowledge in different fields such as statistic, programming, ML, computer engineering, optimization, data science, and computer vision. There are several hyperparameters of the networks that needs to be determined for each specific project (such as number of epochs, learning rate, optimization parameters, number of layers, number of nodes, combination of layers, dropout rate, type of regularization, regularization rate, batch size, and activation function). These hyperparameters were selected based on numerous try-and-error experiments, recommendations of experts in the field, and data published in the current literature.

The first data set with 48,000 image patches is used to perform the binary classification. After cleaning the data from unwanted images, the data set is reduced to 27,000. The data set is divided to testing, training, and developing sets each containing 5,000, 17,000, and 5,000 randomly selected images respectively. Since the number of samples in each class is in the same range (12,000 not-cracked and 15,000 cracked), accuracy can be considered as a metric to compare performance of the models.

A VGG-16 model as described in the previous chapter is used to solve the classification problem. The model for is trained for 100 epochs. Figure 57 shows the gradual increase of accuracy of the model after each epoch. During the first few steps, the accuracy is close to 50% which indicate the model is randomly classifying the images. Most like this is due to the fact that the model weights were initialized with the pretrained weights from ImageNet problem; however, after few steps the model reach 90% accuracy in the first epoch. After few epochs the model learned from the input data and the accuracy increases. The final accuracy of the model on the training set is close to 99.9%, and on the test set it is 98.9%.

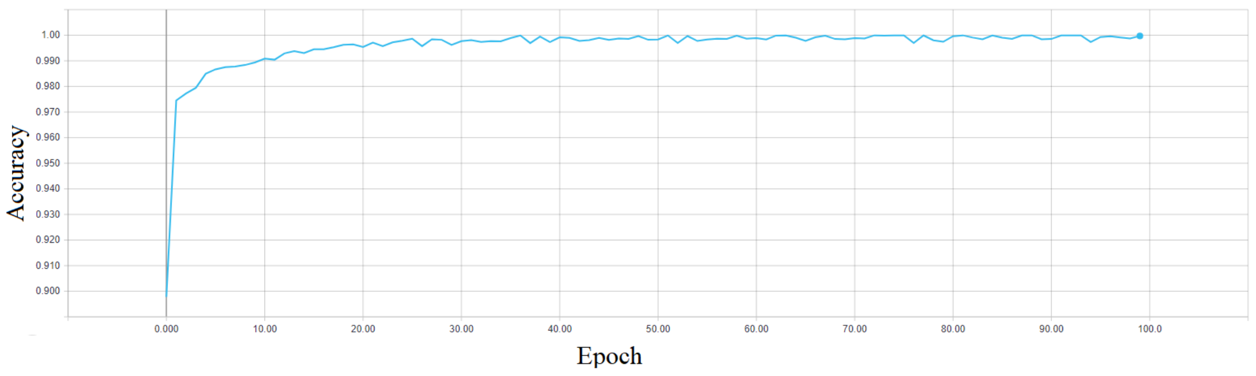


Figure 57. Accuracy of the model on the leaning set versus each epoch (binary VGG-16).

Figure 58 shows the color-coded confusion matrix [157] (error matrix) for the binary classification based on the VGG-16 model. Each row of the matrix represents the number of instances in a predicted class and each column represents the number of instances in a true class. It can be observed that the majority of the data is concentrated at the main diagonal of the matrix which are true positive and true negative. The model has only 55 mistakes out of 5,000 patches which result in 1.1% error rate. Figure 59 shows some of misclassified patches. The existence and nonexistence of crack in most of these patches is very subtle. In some of the images, the image is significantly distorted, and it has lost its continuity. There are some images that were mislabeled the data acquisition phase by the human operator (for example, two patches indicated by a red border in Figure 59). It is very common to have a level of human error in the data set. While the accuracy of the model will increase by reducing the human error, DL models are not too sensitive to these errors as long as they are provided with large data set.

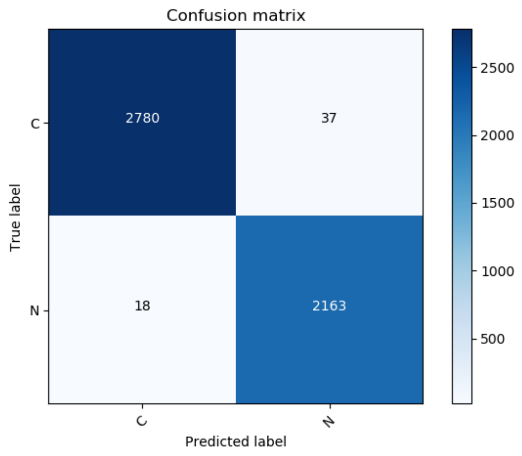


Figure 58. Confusion matrix for binary classification (C: Cracked, N: Not-cracked) (VGG-16).

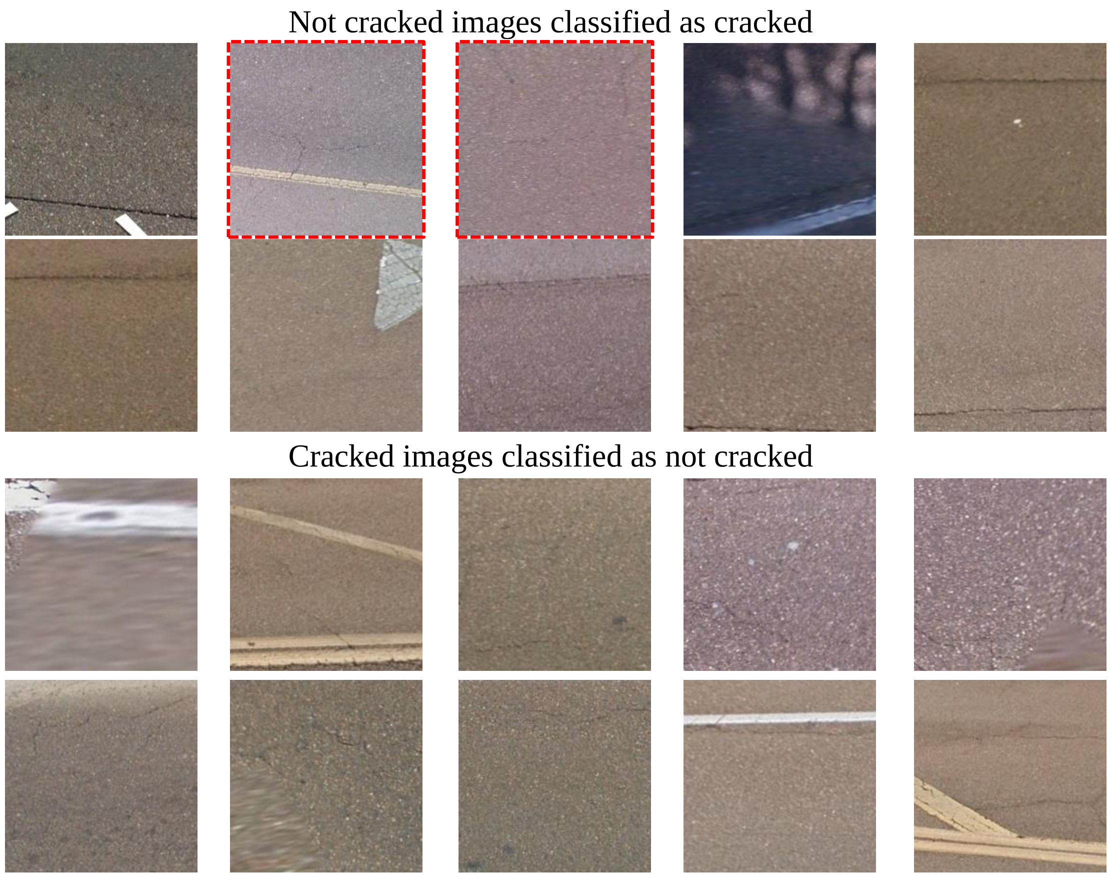

Not cracked images classified as cracked
Cracked images classified as not cracked

Figure 59. A sample of misclassified images.

Table 1 lists a summary of the performance of the VVG-16 model. In the test data set, 2,817 of the images are not-cracked and 2,181 images are cracked. The model has a high performance based on the values of different metrics. The number of false negative and false positive or recall and precision can be changed by considering different threshold for classification. As mentioned in the previous section the output of the SoftMax function is a value between zero and one which represent the probability of each class. It is typical to consider 0.5 as a threshold for binary classification; however, for some applications it is important to minimize the false positive as much as possible. In these situations, a threshold other than 0.5 can be selected to meet the need. ROC curves can be useful to choose a proper threshold based on the true positive and false positive rates. Additionally, the area under the ROC curve is helpful metric when comparing the performance of models. Figure 59 shows the ROC curve for the VGG-16 model. In this study, the goal is to minimize both false positive and false negative to provide a fair evaluation of a road. Thus, the threshold of 0.5 is selected. The AUC value is a good indicator of the performance of the model, and it is typically used for comparing two models. The AUC for VGG-16 model is close to one which is an indication of high performance.

Table 1. Summary of the performance of the binary classification (VGG-16).

	Precision	Recall	F1-Score	# of images
Not cracked	0.99	0.99	0.99	2,817
Cracked	0.98	0.99	0.99	2,181
Overall	0.99	0.99	0.99	4,998

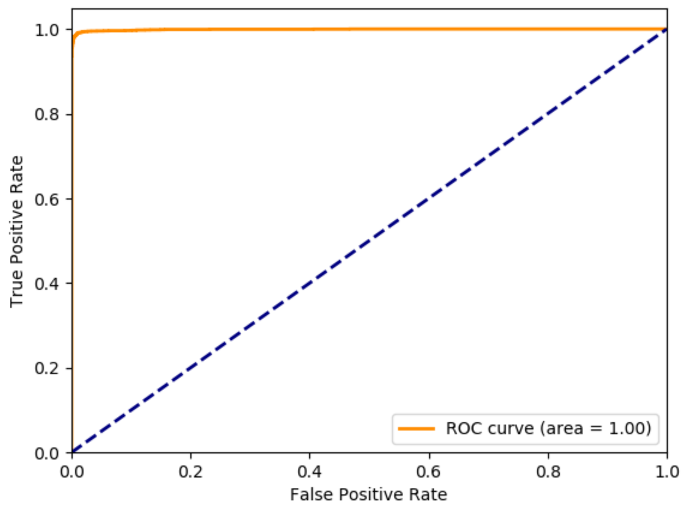


Figure 59. ROC for binary classification (VGG-16).

As mentioned in the previous chapter, DL is a representational learning algorithm. In Figure 60, the output of the second fully connected layer with 2,048 neurons is visualized for the same images presented in Figure 25. PCA is used to reduce 2,048 dimensions to 3 dimensions to show the points in a 3D graph. Each axes of the graph represent a principal component. The first three principal components describe 99.7%, 0.28%, and 0.02% of the variation in the data. As originally shown in Figure 25, there was no detectable pattern to distinguish the two classes. Using DL, the VGG-16 model has learned to present the data in a space that a simple classifier can easily categorize the images. In Figure 60, it can be seen that after representing the data into the new space, there is a distinguishable pattern that can be defined by a simple function.

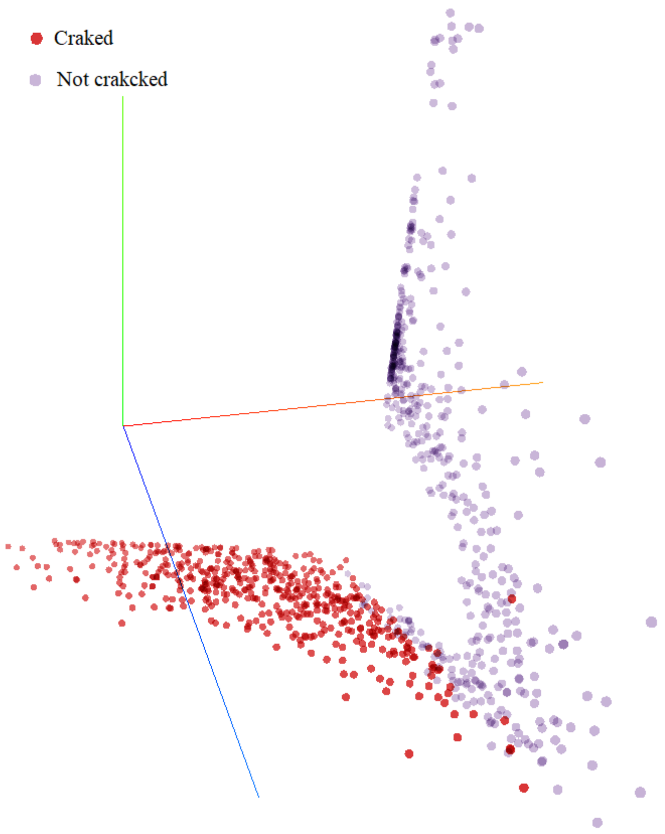


Figure 60. Visualizing 1,024 images of the first data set in a transformed space.

For comparison, the Inception model presented in the previous chapter is applied to to the binary image classification problem., After 100 epochs of training, the accuracy of the Inception model on the test data set is 97.2%. Table 2 lists a summary of the performance of the model and shows lower performance metrics when compared to the VGG-16. Figure 61 shows the ROC graphs for inception model. The AUC value showed at the bottom of the figure indicates the performance of the model. By comparing the results of VGG-16 and Inception model, it can be concluded that VGG-16 model significantly out performs the Inception model when solving this classification problem. Although Inception architects has been shown to have a great performance in image classification, when considering the nature and number of the images in this problem, the complexity of the Inception architecture lead to the model overfitting the data. It is worth mentioning that by adjusting the hyperparameters of the Inception model, the performance may improve. However, since transfer learning from the pretrained model on ImageNet was utilized in this study, changing some the hyperparameters of the network associated with the configuration of the layers was not possible.

Table 2. Summary of the performance of the binary classification (Inception).

	Precision	Recall	F1-Score	# of images
Not cracked	0.96	0.98	0.97	2,817
Cracked	0.97	0.95	0.96	2,181
Overall	0.96	0.96	0.96	4,998

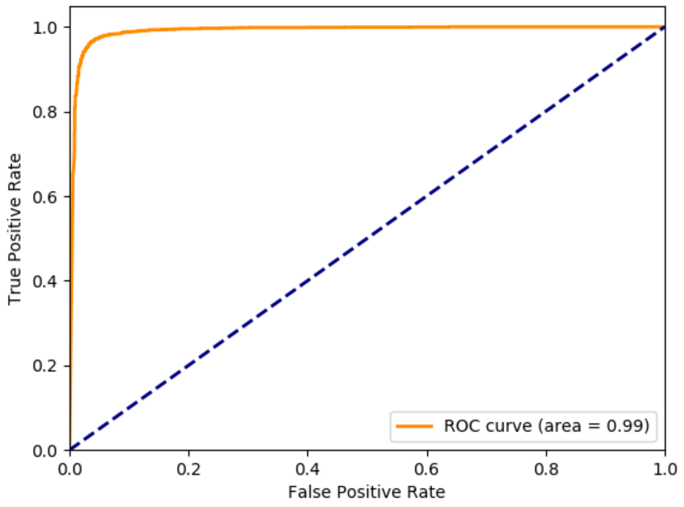


Figure 61. ROC for binary classification (Inception).

Figure 62 shows a sample of 4 images analyzed by the VGG-16 model. The patches colored green indicate correctly identified cracked images (true positive), red patches are false negatives, yellow patches are false positives, and patches with no color are true negatives (images that correctly identified as not cracked).

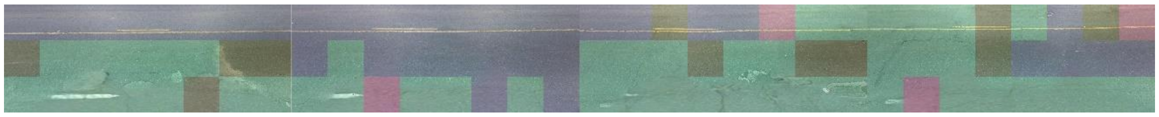


Figure 62. An example of binary crack classification.

4.3 Multi-crack classification experiment

The second experiment is a supervised multiclass classification on the second data set. In this experiment, a classifier is designed to identify 5 classes typically used to categorize pavements: alligator crack (A); longitudinal crack (L); transverse crack (T); not cracked (N); not pavement (C). Since the VGG-16 model in the previous experiment had a better performance, the same network is modified to be used for multiclass image classification. The second data set has around 75,000 image patches of 250×250 pixels. Two graduate students labeled the data and due to discrepancies between the labels 8,546 of the images were removed to develop a consistent data set. For this subset, 12,332 images are considered for testing, 37,170 images for training, and 12,332 images for validation. Since the number of images in each of the five classes was different, the up-sampling method was applied to the data. Since the N class constitutes the majority of the data, the number of samples for each of the other classes is increased to 10,000 images. In addition, a weight proportion is applied to provide more value to samples in the minority classes. Figure 63 shows the accuracy of the VVG-16 model after each epoch of training. The final accuracy on the training set is 99% and on the testing set is 97.2%.

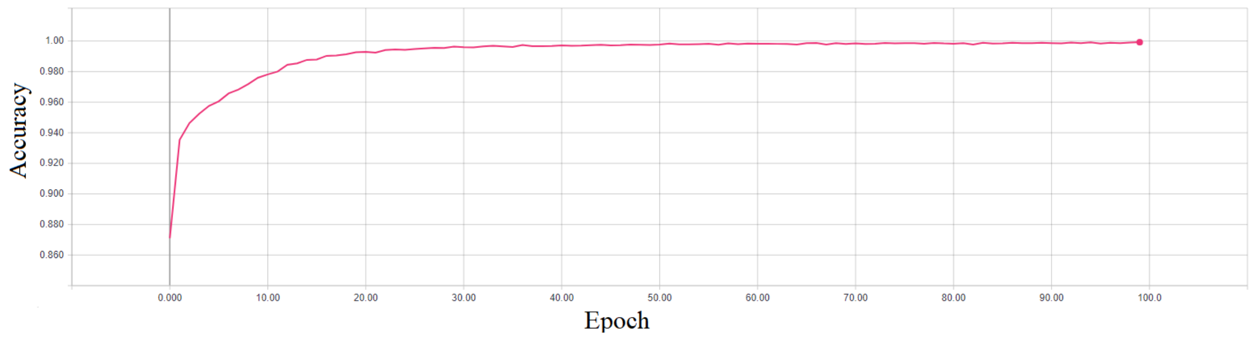


Figure 63. Accuracy of the model on the leaning set versus each step (multiclass VGG-16).

As discussed before, accuracy is not a good metric for evaluating the performance of models using highly imbalanced data. ROC curves are typically plotted for binary classification; however, they become too complicated for evaluating the performance of multiclassification models. Figure 64 shows the color-coded confusion matrix for the crack classification. It can be seen that most of the images are concentrated at the main diagonal of the matrix which represent a correct classification. In addition, this figure illustrates that the classifier can successfully distinguish between images from the N and C classes. From the 1,589 patches of images of the C class, 38 samples are incorrectly classified as N and 1,551 sample correctly classified. There is no instance of misclassifying a C class as one of the classes of cracked images, and there is only one example of misclassifying a L crack as a C image. In order to better visualize the performance on the three cracked classes, the confusion matrix related to these classes is isolated in a separate figure (see Figure 65). It can be seen from this figure that the most misclassified images are L images classified in the A class. There are two reasons for these misclassifications. First, as it is showed in Figure 20 and 21, there is a large discrepancy in labeling longitudinal cracks to alligator crack and vice versa. Although the mislabeled images were removed from the test and training data, there may have been addition images that were mislabeled. The second reason is the similarity of the two image classes which result in short distances between the images of these two classes. The short distance of samples makes it more difficult for the model to classify the images correctly. This point will be examined further when an analysis of the PCA figures is presented later in this section.

N: Not cracked

L: Longitudinal crack

T: Transverse crack

A: Alligator crack

C: Not pavement

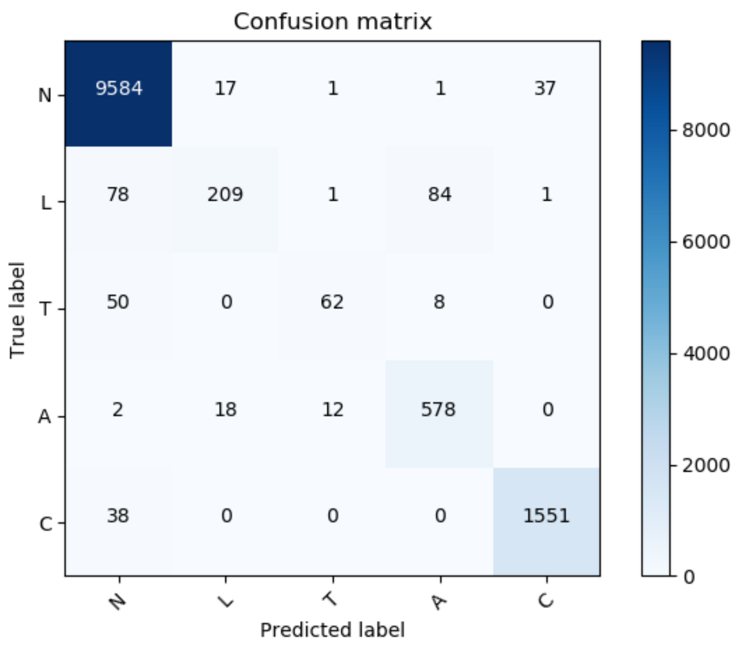


Figure 64. Confusion matrix for multiclass crack classification (5 classes).

L: Longitudinal crack

T: Transverse crack

A: Alligator crack

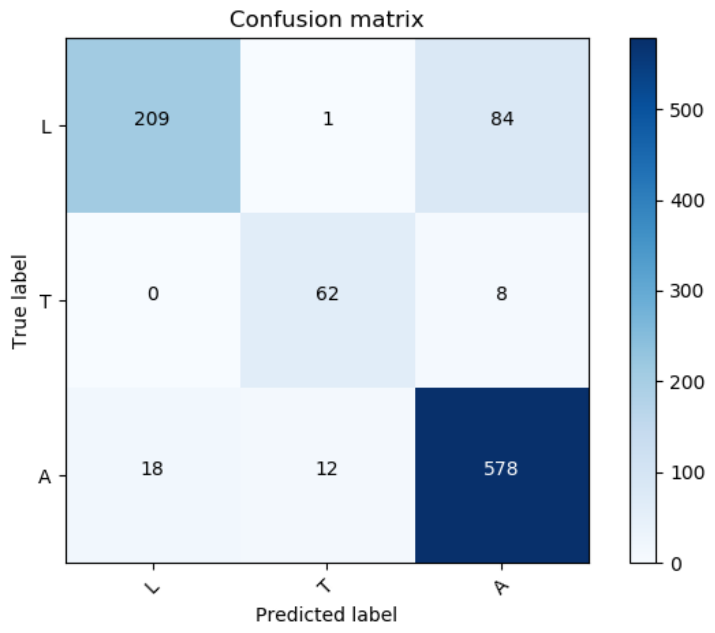


Figure 65. Confusion matrix for multiclass crack classification (3 classes).

Table 3 lists the performance of the VVG-16 model. For this multiclassification model, a macro-average metric is computed independently for each class as well as an average where all classes are considered equal. A micro-average will aggregate the contributions of all classes to compute the average metric (preferred for imbalanced data). In Table 3, it can be seen that all of the metrics for both the N and C classes are outstanding. Since these two classes are the majority of the test set, micro weighted average values are also high. The lowest performance is associated with T class and is associated with the low number of samples in the training set.

In Chapter 2, the human level error is approximated to be around 12%. Although, one experiment is not statistically significant to have a conclusion on human error rate (which is not the aim of this study) this number provides a rough approximation of the level of error that a person might have in a similar classification. The error rate of the proposed model is 2.8% which is significantly lower than the estimated human error rate. It is worth mentioning that, the graduate students who labeled the images had the advantage of looking at each patch in a context of the whole image while the model considered each patch out of the context.

Table 3. Summary of the performance of the multiclass crack classification (VGG-16).

	Precision	Recall	F1-Score	# of images
Not cracked (N)	0.98	0.99	0.99	9,640
Longitudinal crack (L)	0.86	0.56	0.68	373
Transverse crack (T)	0.82	0.52	0.63	120
Alligator crack (A)	0.86	0.95	0.9	610
Not pavement (C)	0.98	0.98	0.98	1,589
Micro Average	**0.97**	**0.97**	**0.97**	**12,332**
Macro Average	0.9	0.8	0.84	12,332
Weighted Average	0.97	0.97	0.97	12,332

The power of DL is the ability to learn a correct representation. As illustrated in Figure 27 and 28 there is no clear pattern in the raw data at the pixel level. In Figures 66 and 67 the same 1,024 images are transformed by the network and are represented in a space with 2,048 dimensions in the last fully connected layer with 2,048 neurons. PCA is used to plot the transformed images in a 2D and 3D graphs. Each axes of the graph represent a principal component. The first three principal components describe 53%, 32%, and 7.4% of the variation in the data. Both Figures 66 and 67 show a clear separation of images of each class. The images associated with the C class have a considerable distance from the other points. This distance is due to the fact that these images typically have different texture and color. For example, many of the samples in the C class are patches that include portions of a car which present a very different look than other pavement images.

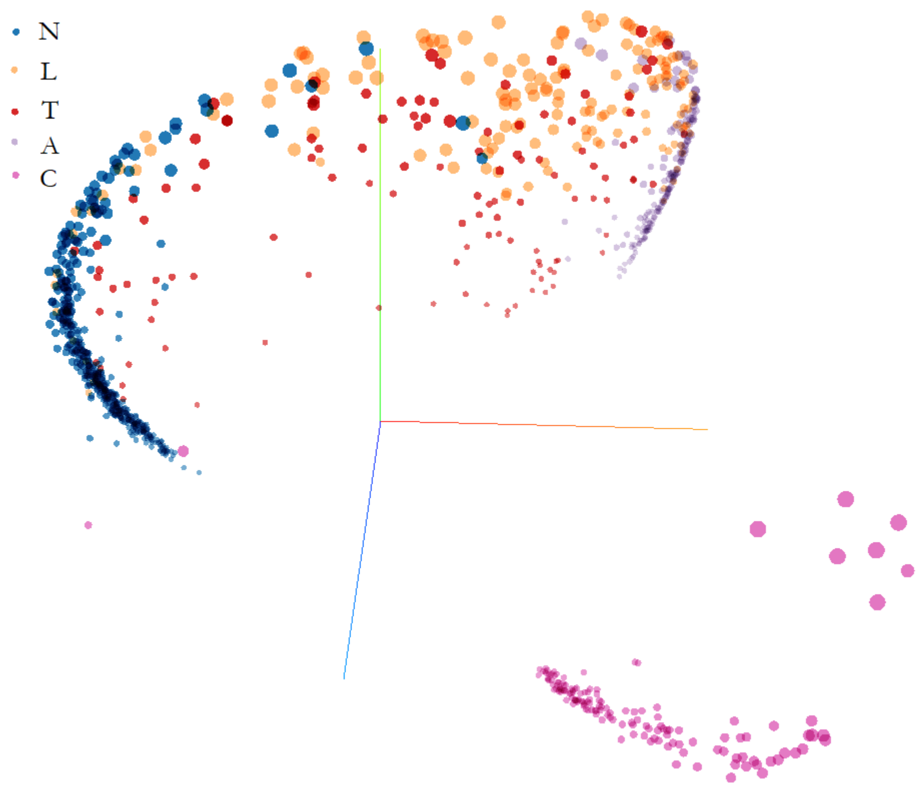


Figure 66. Visualizing 1,024 images of the second data set in a 3D transformed space using PCA.

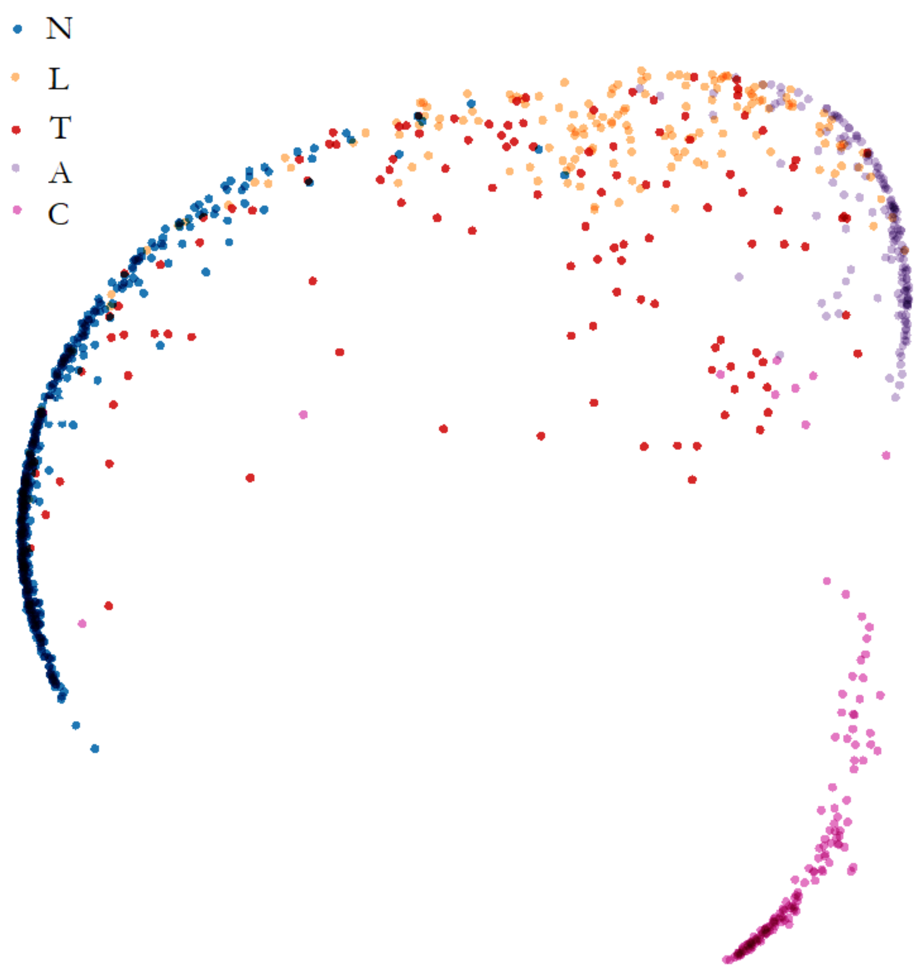


Figure 67. Visualizing 1,024 images of the second data set in a 2D transformed space using PCA.

To explore the details of these graphs, 850 of the patches that are in the upper swarm (mostly pavement images) are selected and replotted in 2D and 3D PCA graphs (see Figures 68 and 69). These graphs show that there is a significant distance between the points associated with N class and the other cracked classes. In order to see the patterns in the crack classes, a sub-set of 300 the data points are selected and replotted in a 2D PCA graph (see Figure 70). In this graph, which is mostly consist of images with cracks, it can be seem the model has learned to represent the pixel values into a space where these images can be classified. As it is described in the confusion matrix, some of the L images are classified in the A class. This misclassification can be observed in the Figure 70 as well, as there is a region where some orange points (L class cracks) are mixed with purple points (A class cracks).

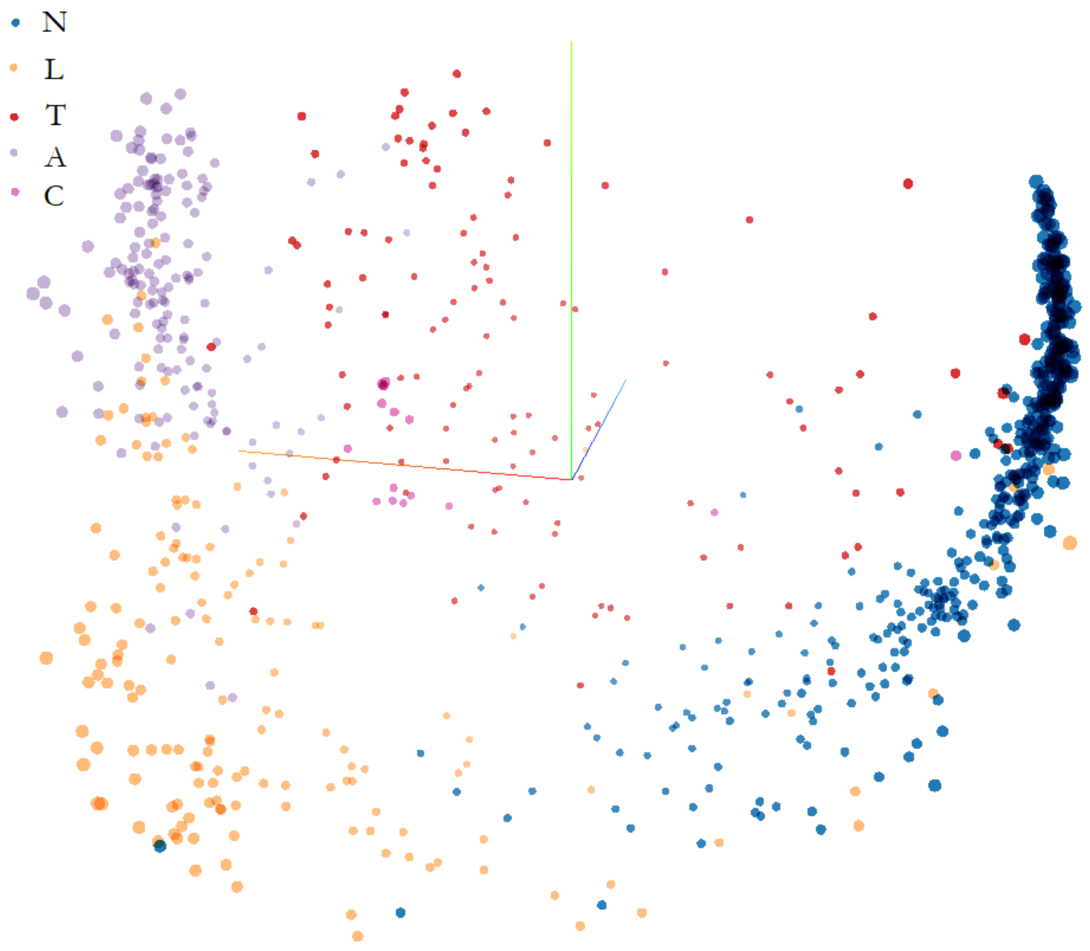


Figure 68. Visualizing 850 images of the second data set in a 3D transformed space using PCA.

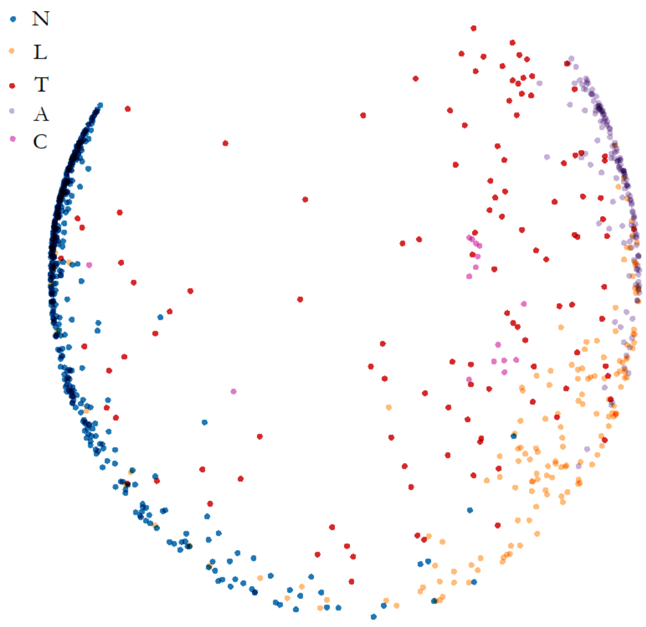


Figure 69. Visualizing 850 images of the second data set in a 2D transformed space using PCA.

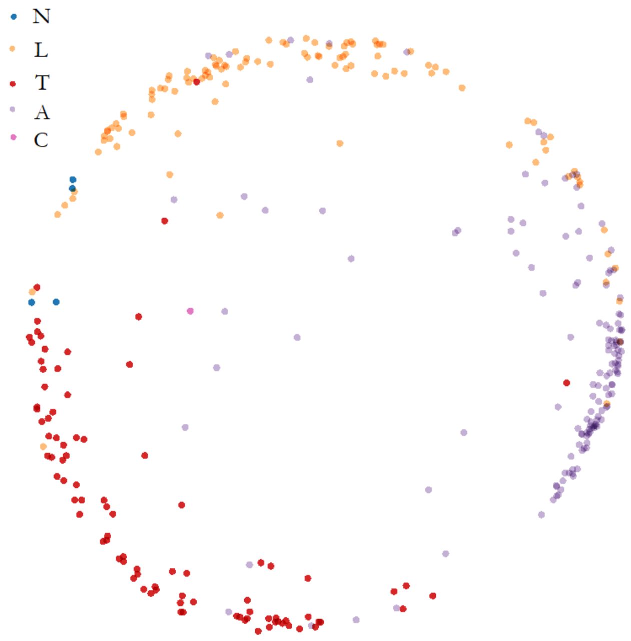


Figure 70. Visualizing 300 images of the second data set in a 2D transformed space using PCA.

5. Chapter 5

Conclusion and Future work

5.1 Conclusion

The importance of pavement evaluation motivates this study to develop a novel technique for identifying and classifying cracks in pavement. The developed method provides an efficient and economical alternative for evaluating pavement quality on a large scale. In this method Google Street View (GSV) technology is used to extract images of pavement. The images were divided into small patches to increase the level of accuracy. Based on the GSV image patches, an image classification was designed to identify the existence of a crack in a small patch. The existence of high level of noise in the images lead to use of deep learning (DL) models for image classification. Two convolutional neural network (CNN) models were developed for classifying the image patches. The available image patches were preprocessed and manually labeled to form a supervised learning problem. The labeled patches were divided into training, developing, and testing sets. The CNN models were then trained by using developing and training sets and were validated on the testing set.

A comparative investigation is performed on the result of the labeled images from GSV with the result from the iPAVe automated crack detection system developed by ARRB for 6.5 miles of the Route 28 near Sterling, Virginia. By approximately scaling the crack detection on GSV images to the results obtained from iPAVe, a highly correlated result in the overall crack detection was observed. Although the ARRA did not provide any additional information about their crack detection procedure and its level of its accuracy, their data gave enough information to conclude that images of the GSV are great resource for estimating the quality of pavement based on crack detection.

In order to automate the crack detection on GSV images, two experiments were performed with two classes (cracked and not cracked) and five classes (not cracked, longitudinal crack, transverse crack, alligator crack, not pavement). In the first experiment, two models were developed for solving the image classification problem. The first model was based on VGG-16 architecture and the second model was based on a version of Inception architecture. For binary classification, both types of models showed exceptional accuracy (98.9% for VGG-16 and 97.2% for Inception). Considering the low resolution of the images and high level of noise in the images, these are outstanding results. By plotting the output of the fully connected layer for each image, the behavior of the model as a representational learning method was analyzed. In the second experiment, the problem was expanded to multiclass crack classification by considering five classes. The number of samples in each class obtained from GSV were drastically different. Thus, several techniques were implemented to overcome the imbalances in the data. For solving the image classification problem, a model based on VGG-16 architecture was developed and trained on an augmented training set. A high level of performance for multiclass crack detection was achieved with the accuracy of 97.2% on the testing data set. By providing the output of the final hidden layer, the transformed image patches using PCA graphs were presented. A clear separation of the data points of the five classes in the transformed space was observed that illustrate the ability of the DL in learning the correct representation.

After in depth review of the result of the proposed models, it is concluded that the DL models presented in this study are effective in solving crack classification problem on pavement images extracted from GSV.

5.2 **Future work**

This study is the first attempt in using GSV in a civil engineering application, and the exceptional result of this study should rightfully motivate many researchers to implement this great resource of data in their work. Additionally, DL methods have been significantly advanced in the past few years and are provide powerful techniques to solve many problems in civil engineering.

In this study, two civil engineering graduate students performed the labeling task. Although they both had a good understanding of the types of cracks in pavement, identifying a crack type based on an observation of a single image, requires years of experience. In order to reduce the error rate and increase the reliability of the result, labeling should be performed by a group of experienced visual inspectors. These inspectors not only can provide useful information about types of cracks, but also can estimate the severity of cracks based on the common standard [158]. One other way to increase the reliability of result is to compare it with other computer vision techniques on pavement. As mentioned in the introduction, there are many research projects and companies that use video cameras along with other sensors for pavement crack detection (which is very expensive to perform). These videos could be used to be matched with GSV images and provide an accurate prediction of cracks and a reliable ground truth data set for training and testing a deep learning model.

In this study, each GSV image was divided into small patches for image classification. Instead of defining crack detection as an image classification problem, the problem can be defined as an object detection or an instance segmentation problem. The aim of these problems is to identify locations as well as types of cracks by drawing several rectangular shapes (object detection) or pixel-accuracy masks (instance segmentation) around each crack. Results of these problems provide more details about cracks which is useful to evaluate the overall quality of pavement. The challenge in using these methods is that they typically require much more data and the labeling process could be more time-consuming than classification problem.

As it was discussed, the labeling process is time-consuming and requires an expert knowledge to be reliable for pavement evaluation. Instead of defining the problem as a supervised learning problem, the crack detection problem can be defined as an unsupervised learning problem (without the need of labeled data) or a semi-supervised learning problem (with small labeled data and large unlabeled data). In an unsupervised method, a DL model will be used to cluster images to a limited number of classes based on features that it can find in the images. By performing clustering, the model may cluster the data into similar standard crack classes, or it may introduce a new system of crack classification. The new system can be scaled to the conventional standard system to provide a practical and meaningful evaluation. In a semi-supervised learning problem the correct label of a given unlabeled data may be inferred based on the labeled data (transductive learning) or the correct mapping from input to output may be inferred (inductive learning).

References

[1] Roads. ASCEs 2017 Infrastruct Rep Card n.d. https://www.infrastructurereport-card.org/cat-item/roads/ (accessed August 3, 2018).

[2] Phares BM, Rolander DD, Graybeal BA, Washer GA. RELIABILITY OF VISUAL BRIDGE INSPECTION. Public Roads 2001;64.

[3] Lenz H, Weichers B. Applications of Specialized Visual Inspection Techniques on Nuclear Components n.d.

[4] Broten M, De Sombre R. The airfield pavement condition index (PCI) evaluation procedure: Advantages, common misapplications, and potential pitfalls. Fifth Int. Conf. Manag. PavementsWashington State Dep. Transp. Pavement Preserv. Soc. Asph. PavementsFederal Highw. Adm. Res. Board, 2001.

[5] Lattanzi D, Miller G. Review of robotic infrastructure inspection systems. J Infrastruct Syst 2017;23:04017004.

[6] TRANSPORTATION: Ratings focus attention on problem bridges. Press Enterp 2014. https://www.pe.com/2014/11/23/transportation-ratings-focus-attention-on-problem-bridges/ (accessed May 17, 2019).

[7] EMA Bridge Inspection Florida, Tampa, Orlando, Jacksonville. Struct Forensic Eng Houst Corpus Christi Austin Forensic Eng Struct Eng Beaumont Forensic Eng Beaumont Struct Eng n.d. http://www.emaengineers.com/building-condition-surveys/bridge-inspection/ (accessed May 17, 2019).

[8] Georgia Dept. of Transportation Makes Successful Early Delivery of Bridge Data to FHWA. AgileAssets 2015. https://www.agileassets.com/blog/georgia-dept-of-transportation-makes-successful-early-delivery-of-bridge-data-to-fhwa/ (accessed May 17, 2019).

[9] Ye XW, Dong CZ, Liu T. A Review of Machine Vision-Based Structural Health Monitoring: Methodologies and Applications. J Sens 2016. doi:10.1155/2016/7103039.

[10] Review of machine-vision based methodologies for displacement measurement in civil structures | SpringerLink n.d. https://link.springer.com/article/10.1007/s13349-017-0261-4 (accessed January 8, 2019).

[11] Jahanshahi MR, Masri SF, Sukhatme GS. Multi-image stitching and scene reconstruction for evaluating defect evolution in structures. Struct Health Monit 2011;10:643–657.

[12] Lee BJ, Shin DH, Seo JW, Jung JD, Lee JY. Intelligent bridge inspection using remote controlled robot and image processing technique. Int. Symp. Autom. Robot. Constr. ISARC Seoul Korea, 2011, p. 1426–1431.

[13] Yeum CM, Dyke SJ. Vision-based automated crack detection for bridge inspection. Comput-Aided Civ Infrastruct Eng 2015;30:759–770.

[14] Mosly I. Applications and Issues of Unmanned Aerial Systems in the Construction Industry. Int J Constr Eng Manag 2017;6:235–239.

[15] Dorafshan S, Maguire M. Bridge inspection: human performance, unmanned aerial systems and automation. J Civ Struct Health Monit 2018;8:443–76. doi:10.1007/s13349-018-0285-4.

[16] Cafiso S, Graziano RD, Battiato S. Evaluation Of Pavement Surface Distress Using Digital Image Collection And Analysis. n.d.

[17] Gavilán M, Balcones D, Marcos O, Llorca DF, Sotelo MA, Parra I, et al. Adaptive Road Crack Detection System by Pavement Classification. Sensors 2011;11:9628–57. doi:10.3390/s111009628.
[18] Roadware n.d. https://www.fugro.com/our-services/asset-integrity/roadware (accessed August 16, 2018).
[19] Road Crack Detection. Quant Imaging n.d. https://research.csiro.au/qi/road-crack-detection/ (accessed August 16, 2018).
[20] Lopes G, Ribeiro AF, Sillero N, Gonçalves-Seco L, Silva C, Franch M, et al. High Resolution Trichromatic Road Surface Scanning with a Line Scan Camera and Light Emitting Diode Lighting for Road-Kill Detection. Sensors 2016;16. doi:10.3390/s16040558.
[21] Tedeschi A, Benedetto F. A real-time automatic pavement crack and pothole recognition system for mobile Android-based devices. Adv Eng Inform 2017;32:11–25. doi:10.1016/j.aei.2016.12.004.
[22] Zhang L, Yang F, Zhang YD, Zhu YJ. Road crack detection using deep convolutional neural network. Image Process. ICIP 2016 IEEE Int. Conf. On, IEEE; 2016, p. 3708–3712.
[23] Maeda H, Sekimoto Y, Seto T, Kashiyama T, Omata H. Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone. ArXiv Prepr ArXiv180109454 2018.
[24] Varadharajan S, Jose S, Sharma K, Wander L, Mertz C. Vision for road inspection. IEEE Winter Conf. Appl. Comput. Vis., Steamboat Springs, CO, USA: IEEE; 2014, p. 115–22. doi:10.1109/WACV.2014.6836111.
[25] Oliveira H, Correia PL. Automatic road crack segmentation using entropy and image dynamic thresholding. 2009 17th Eur. Signal Process. Conf., 2009, p. 622–6.
[26] Cheng HD, Miyojim M. Automatic pavement distress detection system. Inf Sci 1998;108:219–40. doi:10.1016/S0020-0255(97)10062-7.
[27] Zhao H, Qin G, Wang X. Improvement of canny algorithm based on pavement edge detection. 2010 3rd Int. Congr. Image Signal Process., vol. 2, 2010, p. 964–7. doi:10.1109/CISP.2010.5646923.
[28] Shi Y, Cui L, Qi Z, Meng F, Chen Z. Automatic Road Crack Detection Using Random Structured Forests. Undefined 2016. /paper/Automatic-Road-Crack-Detection-Using-Random-Forests-Shi-Cui/ce711e917b9f6a4abd2d3555714a90a280c9fa44 (accessed August 17, 2018).
[29] Tsai Yi-Chang, Kaul Vivek, Mersereau Russell M. Critical Assessment of Pavement Distress Segmentation Methods. J Transp Eng 2010;136:11–9. doi:10.1061/(ASCE)TE.1943-5436.0000051.
[30] Chambon S, Moliard J-M. Automatic Road Pavement Assessment with Image Processing: Review and Comparison. Int J Geophys 2011. doi:10.1155/2011/989354.
[31] Achanta R, Estrada F, Wils P, Süsstrunk S. Salient Region Detection and Segmentation. Comput. Vis. Syst., Springer, Berlin, Heidelberg; 2008, p. 66–75. doi:10.1007/978-3-540-79547-6_7.
[32] Achanta R, Hemami S, Estrada F, Susstrunk S. Frequency-tuned salient region detection. 2009 IEEE Conf. Comput. Vis. Pattern Recognit., 2009, p. 1597–604. doi:10.1109/CVPR.2009.5206596.

[33] Petrou M, Kittler J, Song KY. Automatic surface crack detection on textured materials. J Mater Process Technol 1996;56:158–67. doi:10.1016/0924-0136(95)01831-X.
[34] Song KY, Petrou M, Kittler J. Texture crack detection. Mach Vis Appl 1995;8:63–75. doi:10.1007/BF01213639.
[35] Hu Y, Zhao C. A Novel LBP Based Methods for Pavement Crack Detection. J Pattern Recognit Res 2010;5:140–7. doi:10.13176/11.167.
[36] Zhou J, Huang PS, Chiang F-P. Wavelet-based pavement distress detection and evaluation. Opt Eng 2006;45. doi:10.1117/1.2172917.
[37] Subirats P, Dumoulin J, Legeay V, Barba D. Automation of Pavement Surface Crack Detection using the Continuous Wavelet Transform. 2006 Int. Conf. Image Process., 2006, p. 3037–40. doi:10.1109/ICIP.2006.313007.
[38] Zou Q, Cao Y, Li Q, Mao Q, Wang S. CrackTree: Automatic crack detection from pavement images. Pattern Recognit Lett 2012;33:227–38. doi:10.1016/j.patrec.2011.11.004.
[39] Kass M, Witkin A, Terzopoulos D. Snakes: Active contour models. Int J Comput Vis 1988;1:321–31. doi:10.1007/BF00133570.
[40] Amhaz R, Chambon S, Idier J, Baltazart V. A new minimal path selection algorithm for automatic crack detection on pavement images. 2014 IEEE Int. Conf. Image Process. ICIP, 2014, p. 788–92. doi:10.1109/ICIP.2014.7025158.
[41] Amhaz R, Chambon S, Idier J, Baltazart V. Automatic Crack Detection on Two-Dimensional Pavement Images: An Algorithm Based on Minimal Path Selection. IEEE Trans Intell Transp Syst 2016;17:2718–29. doi:10.1109/TITS.2015.2477675.
[42] Nguyen TS, Begot S, Duculty F, Avila M. Free-form anisotropy: A new method for crack detection on pavement surface images. 2011 18th IEEE Int. Conf. Image Process., 2011, p. 1069–72. doi:10.1109/ICIP.2011.6115610.
[43] Cord A, Chambon S. Automatic Road Defect Detection by Textural Pattern Recognition Based on AdaBoost. Comput-Aided Civ Infrastruct Eng 2012;27:244–59. doi:10.1111/j.1467-8667.2011.00736.x.
[44] Lee BJ, Lee H "David." Position-Invariant Neural Network for Digital Pavement Crack Analysis. Comput-Aided Civ Infrastruct Eng 2004;19:105–18. doi:10.1111/j.1467-8667.2004.00341.x.
[45] Delagnes P, Barba D. A Markov random field for rectilinear structure extraction in pavement distress image analysis. Proc. Int. Conf. Image Process., vol. 1, 1995, p. 446–9 vol.1. doi:10.1109/ICIP.1995.529742.
[46] Oliveira H, Correia PL. Supervised strategies for cracks detection in images of road pavement flexible surfaces. 2008 16th Eur. Signal Process. Conf., 2008, p. 1–5.
[47] Nguyen TS, Avila M, Begot S. Automatic detection and classification of defect on road pavement using anisotropy measure. 2009 17th Eur. Signal Process. Conf., 2009, p. 617–21.
[48] Cheng H, Wang J, Hu Y, Glazier C, Shi X, Chen X. Novel Approach to Pavement Cracking Detection Based on Neural Network. Transp Res Rec J Transp Res Board 2001;1764:119–27. doi:10.3141/1764-13.
[49] Anand RS, Kumar P. Flaw detection in radiographic weldment images using morphological watershed segmentation technique. Ndt E Int 2009;42:2–8.

[50] Nishikawa T, Yoshida J, Sugiyama T, Fujino Y. Concrete crack detection by multiple sequential image filtering. Comput-Aided Civ Infrastruct Eng 2012;27:29–47.
[51] Yamaguchi T, Nakamura S, Saegusa R, Hashimoto S. Image-based crack detection for real concrete surfaces. IEEJ Trans Electr Electron Eng 2008;3:128–135.
[52] Ziou D, Tabbone S. Edge detection techniques-an overview. Pattern Recognit Image Anal CC Raspoznavaniye Obraz Anal Izobr 1998;8:537–559.
[53] Peli T, Malah D. A study of edge detection al—gorithms. Comput Graph Image Process 2009;20:1–21.
[54] Chandrakar N, Bhonsle D. Study and comparison of various image edge detection techniques. Int J Manag IT Eng 2012;2:499–509.
[55] Sharifi M, Fathy M, Mahmoudi MT. A classified and comparative study of edge detection algorithms. Inf. Technol. Coding Comput. 2002 Proc. Int. Conf. On, IEEE; 2002, p. 117–120.
[56] Abdel-Qader I, Abudayyeh O, Kelly ME. Analysis of edge-detection techniques for crack identification in bridges. J Comput Civ Eng 2003;17:255–263.
[57] Zalama E, Gómez-García-Bermejo J, Medina R, Llamas J. Road Crack Detection Using Visual Features Extracted by Gabor Filters. Comput-Aided Civ Infrastruct Eng 2014;29:342–58. doi:10.1111/mice.12042.
[58] Lee S, Ha J, Zokhirova M, Moon H, Lee J. Background Information of Deep Learning for Structural Engineering. Arch Comput Methods Eng 2018;25:121–9. doi:10.1007/s11831-017-9237-0.
[59] Liu R, Yang B, Zio E, Chen X. Artificial intelligence for fault diagnosis of rotating machinery: A review. Mech Syst Signal Process 2018;108:33–47. doi:10.1016/j.ymssp.2018.02.016.
[60] Ross ZE, Meier M-A, Hauksson E, Heaton TH. Generalized Seismic Phase Detection with Deep Learning. Bull Seismol Soc Am 2018;108:2894–901. doi:10.1785/0120180080.
[61] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;521:436–44. doi:10.1038/nature14539.
[62] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE 1998;86:2278–2324.
[63] Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. ImageNet: A large-scale hierarchical image database. 2009 IEEE Conf. Comput. Vis. Pattern Recognit., 2009, p. 248–55. doi:10.1109/CVPR.2009.5206848.
[64] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst., 2012, p. 1097–1105.
[65] Cha Y-J, Choi W, Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. Comput-Aided Civ Infrastruct Eng 2017;32:361–378.
[66] Cha Y-J, Choi W, Suh G, Mahmoudkhani S, Büyüköztürk O. Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. Comput-Aided Civ Infrastruct Eng 2017.
[67] Huang H, Li Q, Zhang D. Deep learning based image recognition for crack and leakage defects of metro shield tunnel. Tunn Undergr Space Technol 2018;77:166–176.

[68] Chen F-C, Jahanshahi MR. NB-CNN: deep learning-based crack detection using convolutional neural network and naive Bayes data fusion. IEEE Trans Ind Electron 2018;65:4392–4400.
[69] Zhang A, Wang KC, Li B, Yang E, Dai X, Peng Y, et al. Automated pixel-level pavement crack detection on 3D asphalt surfaces using a deep-learning network. Comput-Aided Civ Infrastruct Eng 2017;32:805–819.
[70] Eisenbach M, Stricker R, Debes K, Gross H-M. Crack Detection with an Interactive and Adaptive Video Inspection System. Arbeitsgruppentagung Infrastrukturmanagement 2017:94–103.
[71] Pauly L, Hogg D, Fuentes R, Peel H. Deeper networks for pavement crack detection. Proc. 34th ISARC, IAARC; 2017, p. 479–485.
[72] PEER Hub ImageNet n.d. http://apps.peer.berkeley.edu/spo/.
[73] Shapiro A. Street-level: Google Street View’s abstraction by datafication. New Media Soc 2018;20:1201–1219.
[74] Rundle AG, Bader MD, Richards CA, Neckerman KM, Teitler JO. Using Google Street View to audit neighborhood environments. Am J Prev Med 2011;40:94–100.
[75] Torii A, Havlena M, Pajdla T. From google street view to 3d city models. Comput. Vis. Workshop ICCV Workshop 2009 IEEE 12th Int. Conf. On, IEEE; 2009, p. 2188–2195.
[76] Roman A, Garg G, Levoy M. Interactive design of multi-perspective images for visualizing urban landscapes. IEEE Vis. 2004, Austin, TX, USA: IEEE Comput. Soc; 2004, p. 537–44. doi:10.1109/VISUAL.2004.50.
[77] Román A, Lensch HPA. Automatic Multiperspective Images n.d.:10.
[78] Street View for Mobile | Street View. Google Dev n.d. https://developers.google.com/streetview/android (accessed May 8, 2019).
[79] Anguelov D, Dulong C, Filip D, Frueh C, Lafon S, Lyon R, et al. Google street view: Capturing the world at street level. Computer 2010;43:32–38.
[80] Vincent L. Taking online maps down to street level. Computer 2007;40.
[81] Xiao J, Quan L. Multiple view semantic segmentation for street view images. Comput. Vis. 2009 IEEE 12th Int. Conf. On, IEEE; 2009, p. 686–693.
[82] Jae Lee Y, Efros AA, Hebert M. Style-aware mid-level representation for discovering visual connections in space and time. Proc. IEEE Int. Conf. Comput. Vis., 2013, p. 1857–1864.
[83] Goodfellow IJ, Bulatov Y, Ibarz J, Arnoud S, Shet V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. ArXiv Prepr ArXiv13126082 2013.
[84] Zamir AR, Shah M. Accurate image localization based on google maps street view. Eur. Conf. Comput. Vis., Springer; 2010, p. 255–268.
[85] Movshovitz-Attias Y, Yu Q, Stumpe MC, Shet V, Arnoud S, Yatziv L. Ontological supervision for fine grained classification of street view storefronts. Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, p. 1693–1702.
[86] The Stanford CityBlock Project n.d. http://graphics.stanford.edu/projects/cityblock/ (accessed May 8, 2019).

[87] Į Lietuvos kelius grįžta „Google Street View“ automobiliai. Tv3Lt n.d. https://www.tv3.lt/naujiena/lietuva/967514/i-lietuvos-kelius-grizta-google-street-view-automobiliai?utm_source=facebook.com&utm_medium=recommend&utm_campaign=naujiena (accessed May 8, 2019).
[88] Instant Google Street View. Instant Google Str View n.d. http://www.instantstreetview.com (accessed May 8, 2019).
[89] showmystreet.com - fast & easy street viewing n.d. https://showmystreet.com/ (accessed May 23, 2019).
[90] Selenium - Web Browser Automation n.d. https://www.seleniumhq.org/ (accessed May 8, 2019).
[91] Developer Guide | Street View Static API. Google Dev n.d. https://developers.google.com/maps/documentation/streetview/intro (accessed May 8, 2019).
[92] Deep Learning Book n.d. https://www.deeplearningbook.org/contents/intro.html (accessed May 8, 2019).
[93] ARRB Group Inc. - Road Survey Equipment. ARRB Group n.d. http://arrbgroup.net/ (accessed May 24, 2019).
[94] He H, Garcia EA. Learning from Imbalanced Data. IEEE Trans Knowl Data Eng 2009;21:1263–84. doi:10.1109/TKDE.2008.239.
[95] Chawla NV. Data mining for imbalanced datasets: An overview. Data Min. Knowl. Discov. Handb., Springer; 2009, p. 875–886.
[96] Karagod V. How to Handle Imbalanced Data: An Overview n.d. https://www.datascience.com/blog/imbalanced-data (accessed May 10, 2019).
[97] Wu R, Yan S, Shan Y, Dang Q, Sun G. Deep Image: Scaling up Image Recognition n.d.:12.
[98] Inoue H. Data Augmentation by Pairing Samples for Images Classification. ArXiv180102929 Cs Stat 2018.
[99] Frid-Adar M, Klang E, Amitai M, Goldberger J, Greenspan H. Synthetic data augmentation using GAN for improved liver lesion classification. 2018 IEEE 15th Int. Symp. Biomed. Imaging ISBI 2018, 2018, p. 289–93. doi:10.1109/ISBI.2018.8363576.
[100] Perez L, Wang J. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. ArXiv171204621 Cs 2017.
[101] Miko\lajczyk A, Grochowski M. Data augmentation for improving deep learning in image classification problem. 2018 Int. Interdiscip. PhD Workshop IIPhDW, IEEE; 2018, p. 117–122.
[102] Abdi H, Williams LJ. Principal component analysis. Wiley Interdiscip Rev Comput Stat 2010;2:433–59. doi:10.1002/wics.101.
[103] Introduction to Big Data. Coursera n.d. https://www.coursera.org/learn/big-data-introduction/ (accessed May 13, 2019).
[104] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning. vol. 1. MIT press Cambridge; 2016.
[105] Rosenblatt F. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. CORNELL AERONAUTICAL LAB INC BUFFALO NY; 1961.

[106] Minsky M, Papert SA. Perceptrons: An introduction to computational geometry. MIT press; 2017.
[107] Hinton GE, Osindero S, Teh Y-W. A fast learning algorithm for deep belief nets. Neural Comput 2006;18:1527–1554.
[108] Bengio Y, LeCun Y. Scaling learning algorithms towards AI. Large-Scale Kernel Mach 2007;34:1–41.
[109] Ranzato MA, Poultney C, Chopra S, Cun YL. Efficient learning of sparse representations with an energy-based model. Adv. Neural Inf. Process. Syst., 2007, p. 1137–1144.
[110] Samuel AL. Some Studies in Machine Learning Using the Game of Checkers. IBM J Res Dev 1959;3:210–29. doi:10.1147/rd.33.0210.
[111] Machine learning. Wikipedia 2019.
[112] Heidenreich H. What are the types of machine learning? Data Sci 2018. https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f (accessed May 5, 2019).
[113] Abu-Mostafa YS, Magdon-Ismail M, Lin H-T. Learning from data. vol. 4. AMLBook New York, NY, USA:; 2012.
[114] Cherkassky V, Mulier FM. Learning from data: concepts, theory, and methods. John Wiley & Sons; 2007.
[115] Shalev-Shwartz S, Ben-David S. Understanding machine learning: From theory to algorithms. Cambridge university press; 2014.
[116] B. F. Skinner. Wikipedia 2019.
[117] Skinner BF. 'Superstition'in the pigeon. J Exp Psychol 1948;38:168.
[118] Saunders D. The Bias-Variance Tradeoff. Minds Brains Programs 2017. https://djsaunde.wordpress.com/2017/07/17/the-bias-variance-tradeoff/ (accessed May 6, 2019).
[119] Occam's razor. Wikipedia 2019.
[120] Noble B&. Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies|Hardcover. Barnes Noble n.d. https://www.barnesandnoble.com/p/fundamentals-of-machine-learning-for-predictive-data-analytics-john-d-kelleher/1126353364/2661627131683 (accessed May 18, 2019).
[121] Kuhn M, Johnson K. Applied predictive modeling. vol. 26. Springer; 2013.
[122] Activation functions - deeplearning.ai. Coursera n.d. https://www.coursera.org/learn/neural-networks-deep-learning/lecture/4dDC1/activation-functions (accessed July 26, 2018).
[123] Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. Proc. 27th Int. Conf. Mach. Learn. ICML-10, 2010, p. 807–814.
[124] Zeiler MD, Ranzato M, Monga R, Mao M, Yang K, Le QV, et al. On rectified linear units for speech processing. 2013 IEEE Int. Conf. Acoust. Speech Signal Process., 2013, p. 3517–21. doi:10.1109/ICASSP.2013.6638312.
[125] Scherer D, Müller A, Behnke S. Evaluation of pooling operations in convolutional architectures for object recognition. Artif. Neural Networks–ICANN 2010, Springer; 2010, p. 92–101.

[126] Spark C. Deep learning for complete beginners: convolutional neural networks with keras n.d. http://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html (accessed July 26, 2018).
[127] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 2014;15:1929–1958.
[128] Kingma DP, Ba J. Adam: A method for stochastic optimization. ArXiv Prepr ArXiv14126980 2014.
[129] Reddi SJ, Kale S, Kumar S. ON THE CONVERGENCE OF ADAM AND BEYOND 2018:23.
[130] An Open Source Machine Learning Framework for Everyone: tensorflow/tensorflow. tensorflow; 2019.
[131] Bengio Y. Practical recommendations for gradient-based training of deep architectures. Neural Netw. Tricks Trade, Springer; 2012, p. 437–478.
[132] Masters D, Luschi C. Revisiting Small Batch Training for Deep Neural Networks. ArXiv Prepr ArXiv180407612 2018.
[133] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, p. 1–9.
[134] Zeiler MD, Fergus R. Visualizing and Understanding Convolutional Networks. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T, editors. Comput. Vis. – ECCV 2014, vol. 8689, Cham: Springer International Publishing; 2014, p. 818–33. doi:10.1007/978-3-319-10590-1_53.
[135] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. ArXiv Prepr ArXiv14091556 2014.
[136] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. ArXiv151203385 Cs 2015.
[137] Lin M, Chen Q, Yan S. Network In Network. ArXiv13124400 Cs 2013.
[138] Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. ArXiv160207360 Cs 2016.
[139] Zagoruyko S, Komodakis N. Wide Residual Networks. ArXiv160507146 Cs 2016.
[140] Xie S, Girshick R, Dollár P, Tu Z, He K. Aggregated Residual Transformations for Deep Neural Networks. ArXiv161105431 Cs 2016.
[141] Huang G, Sun Y, Liu Z, Sedra D, Weinberger K. Deep Networks with Stochastic Depth. ArXiv160309382 Cs 2016.
[142] Larsson G, Maire M, Shakhnarovich G. FractalNet: Ultra-Deep Neural Networks without Residuals. ArXiv160507648 Cs 2016.
[143] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely Connected Convolutional Networks. CVPR, vol. 1, 2017, p. 3.
[144] Canziani A, Paszke A, Culurciello E. An Analysis of Deep Neural Network Models for Practical Applications. ArXiv160507678 Cs 2016.
[145] ImageNet n.d. http://www.image-net.org/ (accessed May 10, 2019).
[146] Convolutional Neural Networks. Coursera n.d. https://www.coursera.org/learn/machine-learning (accessed May 10, 2019).

[147] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the Inception Architecture for Computer Vision. ArXiv151200567 Cs 2015.
[148] Fernandez-Redondo M, Hernandez-Espinosa C. A comparison among weight initialization methods for multilayer feedforward networks. Proc. IEEE-INNS-ENNS Int. Jt. Conf. Neural Netw. IJCNN 2000 Neural Comput. New Chall. Perspect. New Millenn., vol. 4, 2000, p. 543–8 vol.4. doi:10.1109/IJCNN.2000.860828.
[149] Thimm G, Fiesler E. Neural network initialization. In: Mira J, Sandoval F, editors. Nat. Artif. Neural Comput., Springer Berlin Heidelberg; 1995, p. 535–42.
[150] Gao Y, Mosalam KM. Deep Transfer Learning for Image-Based Structural Damage Recognition. Comput Civ Infrastruct Eng 2018;33:748–68. doi:10.1111/mice.12363.
[151] vgg16_weights.h5. Google Docs n.d. https://drive.google.com/file/d/0Bz7KyqmuGsilT0J5dmRCM0ROVHc/view?usp=sharing&usp=embed_facebook (accessed May 19, 2019).
[152] Li F-F, Johnson J, Yeung S. Convolutional Neural Networks for Visual Recognition, Lecture 8 n.d.:http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture8.pdf.
[153] Introducing The GeForce GTX 1080 Ti, The World’s Fastest Gaming GPU n.d. http://www.geforce.co.uk/whats-new/articles/nvidia-geforce-gtx-1080-ti (accessed May 13, 2019).
[154] NVIDIA cuDNN. NVIDIA Dev 2014. https://developer.nvidia.com/cudnn (accessed May 13, 2019).
[155] TensorFlow White Papers. TensorFlow n.d. https://www.tensorflow.org/about/bib (accessed July 19, 2018).
[156] Home - Keras Documentation n.d. https://keras.io/ (accessed May 13, 2019).
[157] Glossary of Terms Journal of Machine Learning n.d. http://ai.stanford.edu/~ronnyk/glossary.html (accessed May 28, 2019).
[158] E17 Committee. Practice for Roads and Parking Lots Pavement Condition Index Surveys. ASTM International; n.d. doi:10.1520/D6433-18.

Appendix A – Details of the VGG-16 network

```
Layer (type)                 Output Shape              Param #
=================================================================
Image (InputLayer)           (None, 200, 200, 3)       0

block1_conv1 (Conv2D)        (None, 200, 200, 64)      1792

block1_conv2 (Conv2D)        (None, 200, 200, 64)      36928

block1_pool (MaxPooling2D)   (None, 100, 100, 64)      0

block2_conv1 (Conv2D)        (None, 100, 100, 128)     73856

block2_conv2 (Conv2D)        (None, 100, 100, 128)     147584

block2_pool (MaxPooling2D)   (None, 50, 50, 128)       0

block3_conv1 (Conv2D)        (None, 50, 50, 256)       295168

block3_conv2 (Conv2D)        (None, 50, 50, 256)       590080

block3_conv3 (Conv2D)        (None, 50, 50, 256)       590080

block3_pool (MaxPooling2D)   (None, 25, 25, 256)       0

block4_conv1 (Conv2D)        (None, 25, 25, 512)       1180160

block4_conv2 (Conv2D)        (None, 25, 25, 512)       2359808

block4_conv3 (Conv2D)        (None, 25, 25, 512)       2359808

block4_pool (MaxPooling2D)   (None, 12, 12, 512)       0

block5_conv1 (Conv2D)        (None, 12, 12, 512)       2359808

block5_conv2 (Conv2D)        (None, 12, 12, 512)       2359808

block5_conv3 (Conv2D)        (None, 12, 12, 512)       2359808
```

```
block5_pool (MaxPooling2D)   (None, 6, 6, 512)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)              0
_________________________________________________________________
features1 (Dense)            (None, 2048)             1050624
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)             0
_________________________________________________________________
features2 (Dense)            (None, 2048)             4196352
_________________________________________________________________
dropout_2 (Dropout)          (None, 2048)             0
_________________________________________________________________
Prediction (Dense)           (None, 2)                4098
=================================================================
Total params: 19,965,762
Trainable params: 19,965,762
Non-trainable params: 0
_________________________________________________________________
```

Appendix B – Details of the Inception network

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Image (InputLayer)              (None, 200, 200, 3)  0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 99, 99, 32)   864         Image[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 99, 99, 32)   96          conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 99, 99, 32)   0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 97, 97, 32)   9216        activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 97, 97, 32)   96          conv2d_2[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 97, 97, 32)   0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 97, 97, 64)   18432       activation_2[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 97, 97, 64)   192         conv2d_3[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 97, 97, 64)   0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 48, 48, 64)   0           activation_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 48, 48, 80)   5120        max_pooling2d_1[0][0]
```

```
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 48, 48, 80)   240         conv2d_4[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 48, 48, 80)   0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 46, 46, 192)  138240      activation_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 46, 46, 192)  576         conv2d_5[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 46, 46, 192)  0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 22, 22, 192)  0           activation_5[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 22, 22, 64)   12288       max_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 22, 22, 64)   192         conv2d_9[0][0]
__________________________________________________________________________________________________
activation_9 (Activation)       (None, 22, 22, 64)   0           batch_normalization_9[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 22, 22, 48)   9216        max_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 22, 22, 96)   55296       activation_9[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 22, 22, 48)   144         conv2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 22, 22, 96)   288         conv2d_10[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 22, 22, 48)   0           batch_normalization_7[0][0]
```

```
________________________________________________________________________________
________________________
activation_10 (Activation)     (None, 22, 22, 96)   0          batch_normalization_10[0][0]
________________________________________________________________________________
________________________
average_pooling2d_1 (AveragePoo (None, 22, 22, 192)  0          max_pooling2d_2[0][0]
________________________________________________________________________________
________________________
conv2d_6 (Conv2D)              (None, 22, 22, 64)   12288       max_pooling2d_2[0][0]
________________________________________________________________________________
________________________
conv2d_8 (Conv2D)              (None, 22, 22, 64)   76800       activation_7[0][0]
________________________________________________________________________________
________________________
conv2d_11 (Conv2D)             (None, 22, 22, 96)   82944       activation_10[0][0]
________________________________________________________________________________
________________________
conv2d_12 (Conv2D)             (None, 22, 22, 32)   6144        average_pooling2d_1[0][0]
________________________________________________________________________________
________________________
batch_normalization_6 (BatchNor (None, 22, 22, 64)   192         conv2d_6[0][0]
________________________________________________________________________________
________________________
batch_normalization_8 (BatchNor (None, 22, 22, 64)   192         conv2d_8[0][0]
________________________________________________________________________________
________________________
batch_normalization_11 (BatchNo (None, 22, 22, 96)   288         conv2d_11[0][0]
________________________________________________________________________________
________________________
batch_normalization_12 (BatchNo (None, 22, 22, 32)   96          conv2d_12[0][0]
________________________________________________________________________________
________________________
activation_6 (Activation)      (None, 22, 22, 64)   0          batch_normalization_6[0][0]
________________________________________________________________________________
________________________
activation_8 (Activation)      (None, 22, 22, 64)   0          batch_normalization_8[0][0]
________________________________________________________________________________
________________________
activation_11 (Activation)     (None, 22, 22, 96)   0          batch_normalization_11[0][0]
________________________________________________________________________________
________________________
activation_12 (Activation)     (None, 22, 22, 32)   0          batch_normalization_12[0][0]
```

```
__________________________________________________________________________________________________
mixed0 (Concatenate)            (None, 22, 22, 256)  0           activation_6[0][0]
                                                                 activation_8[0][0]
                                                                 activation_11[0][0]
                                                                 activation_12[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 22, 22, 64)   16384       mixed0[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 22, 22, 64)   192         conv2d_16[0][0]
__________________________________________________________________________________________________
activation_16 (Activation)      (None, 22, 22, 64)   0           batch_normalization_16[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 22, 22, 48)   12288       mixed0[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 22, 22, 96)   55296       activation_16[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 22, 22, 48)   144         conv2d_14[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 22, 22, 96)   288         conv2d_17[0][0]
__________________________________________________________________________________________________
activation_14 (Activation)      (None, 22, 22, 48)   0           batch_normalization_14[0][0]
__________________________________________________________________________________________________
activation_17 (Activation)      (None, 22, 22, 96)   0           batch_normalization_17[0][0]
__________________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, 22, 22, 256)  0           mixed0[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 22, 22, 64)   16384       mixed0[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 22, 22, 64)   76800       activation_14[0][0]
```

```
__________________________________________________________________________________________________
conv2d_18 (Conv2D)          (None, 22, 22, 96)   82944       activation_17[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)          (None, 22, 22, 64)   16384       average_pooling2d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 22, 22, 64)   192         conv2d_13[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 22, 22, 64)   192         conv2d_15[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 22, 22, 96)   288         conv2d_18[0][0]
__________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 22, 22, 64)   192         conv2d_19[0][0]
__________________________________________________________________________________________________
activation_13 (Activation)      (None, 22, 22, 64)   0           batch_normalization_13[0][0]
__________________________________________________________________________________________________
activation_15 (Activation)      (None, 22, 22, 64)   0           batch_normalization_15[0][0]
__________________________________________________________________________________________________
activation_18 (Activation)      (None, 22, 22, 96)   0           batch_normalization_18[0][0]
__________________________________________________________________________________________________
activation_19 (Activation)      (None, 22, 22, 64)   0           batch_normalization_19[0][0]
__________________________________________________________________________________________________
mixed1 (Concatenate)            (None, 22, 22, 288)  0           activation_13[0][0]
                                                                 activation_15[0][0]
                                                                 activation_18[0][0]
                                                                 activation_19[0][0]
__________________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 22, 22, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 22, 22, 64)   192         conv2d_23[0][0]
__________________________________________________________________________________________________
______________________
activation_23 (Activation)      (None, 22, 22, 64)   0           batch_normalization_23[0][0]
__________________________________________________________________________________________________
______________________
conv2d_21 (Conv2D)              (None, 22, 22, 48)   13824       mixed1[0][0]
__________________________________________________________________________________________________
______________________
conv2d_24 (Conv2D)              (None, 22, 22, 96)   55296       activation_23[0][0]
__________________________________________________________________________________________________
______________________
batch_normalization_21 (BatchNo (None, 22, 22, 48)   144         conv2d_21[0][0]
__________________________________________________________________________________________________
______________________
batch_normalization_24 (BatchNo (None, 22, 22, 96)   288         conv2d_24[0][0]
__________________________________________________________________________________________________
______________________
activation_21 (Activation)      (None, 22, 22, 48)   0           batch_normalization_21[0][0]
__________________________________________________________________________________________________
______________________
activation_24 (Activation)      (None, 22, 22, 96)   0           batch_normalization_24[0][0]
__________________________________________________________________________________________________
______________________
average_pooling2d_3 (AveragePoo (None, 22, 22, 288)  0           mixed1[0][0]
__________________________________________________________________________________________________
______________________
conv2d_20 (Conv2D)              (None, 22, 22, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________________
______________________
conv2d_22 (Conv2D)              (None, 22, 22, 64)   76800       activation_21[0][0]
__________________________________________________________________________________________________
______________________
conv2d_25 (Conv2D)              (None, 22, 22, 96)   82944       activation_24[0][0]
__________________________________________________________________________________________________
______________________
conv2d_26 (Conv2D)              (None, 22, 22, 64)   18432       average_pooling2d_3[0][0]
__________________________________________________________________________________________________
______________________
batch_normalization_20 (BatchNo (None, 22, 22, 64)   192         conv2d_20[0][0]
__________________________________________________________________________________________________
______________________
batch_normalization_22 (BatchNo (None, 22, 22, 64)   192         conv2d_22[0][0]
__________________________________________________________________________________________
batch_normalization_25 (BatchNo (None, 22, 22, 96)   288         conv2d_25[0][0]
__________________________________________________________________________________________
batch_normalization_26 (BatchNo (None, 22, 22, 64)   192         conv2d_26[0][0]
__________________________________________________________________________________________
activation_20 (Activation)      (None, 22, 22, 64)   0           batch_normalization_20[0][0]
__________________________________________________________________________________________
activation_22 (Activation)      (None, 22, 22, 64)   0           batch_normalization_22[0][0]
__________________________________________________________________________________________
activation_25 (Activation)      (None, 22, 22, 96)   0           batch_normalization_25[0][0]
__________________________________________________________________________________________
activation_26 (Activation)      (None, 22, 22, 64)   0           batch_normalization_26[0][0]
__________________________________________________________________________________________
mixed2 (Concatenate)            (None, 22, 22, 288)  0           activation_20[0][0]
                                                                 activation_22[0][0]
                                                                 activation_25[0][0]
                                                                 activation_26[0][0]
__________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 22, 22, 64)   18432       mixed2[0][0]
__________________________________________________________________________________________
batch_normalization_28 (BatchNo (None, 22, 22, 64)   192         conv2d_28[0][0]
__________________________________________________________________________________________
activation_28 (Activation)      (None, 22, 22, 64)   0           batch_normalization_28[0][0]
__________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 22, 22, 96)   55296       activation_28[0][0]
__________________________________________________________________________________________
batch_normalization_29 (BatchNo (None, 22, 22, 96)   288         conv2d_29[0][0]
__________________________________________________________________________________________
activation_29 (Activation)      (None, 22, 22, 96)   0           batch_normalization_29[0][0]
```

```
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 10, 10, 384)  995328      mixed2[0][0]
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 10, 10, 96)   82944       activation_29[0][0]
__________________________________________________________________________________________________
batch_normalization_27 (BatchNo (None, 10, 10, 384)  1152        conv2d_27[0][0]
__________________________________________________________________________________________________
batch_normalization_30 (BatchNo (None, 10, 10, 96)   288         conv2d_30[0][0]
__________________________________________________________________________________________________
activation_27 (Activation)      (None, 10, 10, 384)  0           batch_normalization_27[0][0]
__________________________________________________________________________________________________
activation_30 (Activation)      (None, 10, 10, 96)   0           batch_normalization_30[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 10, 10, 288)  0           mixed2[0][0]
__________________________________________________________________________________________________
mixed3 (Concatenate)            (None, 10, 10, 768)  0           activation_27[0][0]
                                                                 activation_30[0][0]
                                                                 max_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 10, 10, 128)  98304       mixed3[0][0]
__________________________________________________________________________________________________
batch_normalization_35 (BatchNo (None, 10, 10, 128)  384         conv2d_35[0][0]
__________________________________________________________________________________________________
activation_35 (Activation)      (None, 10, 10, 128)  0           batch_normalization_35[0][0]
__________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, 10, 10, 128)  114688      activation_35[0][0]
__________________________________________________________________________________________________
batch_normalization_36 (BatchNo (None, 10, 10, 128)  384         conv2d_36[0][0]
__________________________________________________________________________________________________
```

```
activation_36 (Activation)      (None, 10, 10, 128)  0           batch_normalization_36[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 10, 10, 128)  98304       mixed3[0][0]
__________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, 10, 10, 128)  114688      activation_36[0][0]
__________________________________________________________________________________________________
batch_normalization_32 (BatchNo (None, 10, 10, 128)  384         conv2d_32[0][0]
__________________________________________________________________________________________________
batch_normalization_37 (BatchNo (None, 10, 10, 128)  384         conv2d_37[0][0]
__________________________________________________________________________________________________
activation_32 (Activation)      (None, 10, 10, 128)  0           batch_normalization_32[0][0]
__________________________________________________________________________________________________
activation_37 (Activation)      (None, 10, 10, 128)  0           batch_normalization_37[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 10, 10, 128)  114688      activation_32[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 10, 10, 128)  114688      activation_37[0][0]
__________________________________________________________________________________________________
batch_normalization_33 (BatchNo (None, 10, 10, 128)  384         conv2d_33[0][0]
__________________________________________________________________________________________________
batch_normalization_38 (BatchNo (None, 10, 10, 128)  384         conv2d_38[0][0]
__________________________________________________________________________________________________
activation_33 (Activation)      (None, 10, 10, 128)  0           batch_normalization_33[0][0]
__________________________________________________________________________________________________
activation_38 (Activation)      (None, 10, 10, 128)  0           batch_normalization_38[0][0]
__________________________________________________________________________________________________
average_pooling2d_4 (AveragePoo (None, 10, 10, 768)  0           mixed3[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 10, 10, 192)  147456      mixed3[0][0]
```

```
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 10, 10, 192)  172032      activation_33[0][0]
__________________________________________________________________________________________________
conv2d_39 (Conv2D)              (None, 10, 10, 192)  172032      activation_38[0][0]
__________________________________________________________________________________________________
conv2d_40 (Conv2D)              (None, 10, 10, 192)  147456      average_pooling2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, 10, 10, 192)  576         conv2d_31[0][0]
__________________________________________________________________________________________________
batch_normalization_34 (BatchNo (None, 10, 10, 192)  576         conv2d_34[0][0]
__________________________________________________________________________________________________
batch_normalization_39 (BatchNo (None, 10, 10, 192)  576         conv2d_39[0][0]
__________________________________________________________________________________________________
batch_normalization_40 (BatchNo (None, 10, 10, 192)  576         conv2d_40[0][0]
__________________________________________________________________________________________________
activation_31 (Activation)      (None, 10, 10, 192)  0           batch_normalization_31[0][0]
__________________________________________________________________________________________________
activation_34 (Activation)      (None, 10, 10, 192)  0           batch_normalization_34[0][0]
__________________________________________________________________________________________________
activation_39 (Activation)      (None, 10, 10, 192)  0           batch_normalization_39[0][0]
__________________________________________________________________________________________________
activation_40 (Activation)      (None, 10, 10, 192)  0           batch_normalization_40[0][0]
__________________________________________________________________________________________________
mixed4 (Concatenate)            (None, 10, 10, 768)  0           activation_31[0][0]
                                                                 activation_34[0][0]
                                                                 activation_39[0][0]
                                                                 activation_40[0][0]
__________________________________________________________________________________________________
conv2d_45 (Conv2D)              (None, 10, 10, 160)  122880      mixed4[0][0]
```

```
__________________________________________________________________________________________________
batch_normalization_45 (BatchNo (None, 10, 10, 160)  480         conv2d_45[0][0]
__________________________________________________________________________________________________
activation_45 (Activation)      (None, 10, 10, 160)  0           batch_normalization_45[0][0]
__________________________________________________________________________________________________
conv2d_46 (Conv2D)              (None, 10, 10, 160)  179200      activation_45[0][0]
__________________________________________________________________________________________________
batch_normalization_46 (BatchNo (None, 10, 10, 160)  480         conv2d_46[0][0]
__________________________________________________________________________________________________
activation_46 (Activation)      (None, 10, 10, 160)  0           batch_normalization_46[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 10, 10, 160)  122880      mixed4[0][0]
__________________________________________________________________________________________________
conv2d_47 (Conv2D)              (None, 10, 10, 160)  179200      activation_46[0][0]
__________________________________________________________________________________________________
batch_normalization_42 (BatchNo (None, 10, 10, 160)  480         conv2d_42[0][0]
__________________________________________________________________________________________________
batch_normalization_47 (BatchNo (None, 10, 10, 160)  480         conv2d_47[0][0]
__________________________________________________________________________________________________
activation_42 (Activation)      (None, 10, 10, 160)  0           batch_normalization_42[0][0]
__________________________________________________________________________________________________
activation_47 (Activation)      (None, 10, 10, 160)  0           batch_normalization_47[0][0]
__________________________________________________________________________________________________
conv2d_43 (Conv2D)              (None, 10, 10, 160)  179200      activation_42[0][0]
__________________________________________________________________________________________________
conv2d_48 (Conv2D)              (None, 10, 10, 160)  179200      activation_47[0][0]
__________________________________________________________________________________________________
batch_normalization_43 (BatchNo (None, 10, 10, 160)  480         conv2d_43[0][0]
__________________________________________________________________________________________________
batch_normalization_48 (BatchNo (None, 10, 10, 160)  480         conv2d_48[0][0]
__________________________________________________________________________________________________
activation_43 (Activation)      (None, 10, 10, 160)  0           batch_normalization_43[0][0]
__________________________________________________________________________________________________
activation_48 (Activation)      (None, 10, 10, 160)  0           batch_normalization_48[0][0]
__________________________________________________________________________________________________
average_pooling2d_5 (AveragePoo (None, 10, 10, 768)  0           mixed4[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 10, 10, 192)  147456      mixed4[0][0]
__________________________________________________________________________________________________
conv2d_44 (Conv2D)              (None, 10, 10, 192)  215040      activation_43[0][0]
__________________________________________________________________________________________________
conv2d_49 (Conv2D)              (None, 10, 10, 192)  215040      activation_48[0][0]
__________________________________________________________________________________________________
conv2d_50 (Conv2D)              (None, 10, 10, 192)  147456      average_pooling2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_41 (BatchNo (None, 10, 10, 192)  576         conv2d_41[0][0]
__________________________________________________________________________________________________
batch_normalization_44 (BatchNo (None, 10, 10, 192)  576         conv2d_44[0][0]
__________________________________________________________________________________________________
batch_normalization_49 (BatchNo (None, 10, 10, 192)  576         conv2d_49[0][0]
__________________________________________________________________________________________________
batch_normalization_50 (BatchNo (None, 10, 10, 192)  576         conv2d_50[0][0]
__________________________________________________________________________________________________
activation_41 (Activation)      (None, 10, 10, 192)  0           batch_normalization_41[0][0]
__________________________________________________________________________________________________
activation_44 (Activation)      (None, 10, 10, 192)  0           batch_normalization_44[0][0]
```

```
__________________________________________________________________________________________________
activation_49 (Activation)      (None, 10, 10, 192)  0           batch_normalization_49[0][0]
__________________________________________________________________________________________________
activation_50 (Activation)      (None, 10, 10, 192)  0           batch_normalization_50[0][0]
__________________________________________________________________________________________________
mixed5 (Concatenate)            (None, 10, 10, 768)  0           activation_41[0][0]
                                                                 activation_44[0][0]
                                                                 activation_49[0][0]
                                                                 activation_50[0][0]
__________________________________________________________________________________________________
conv2d_55 (Conv2D)              (None, 10, 10, 160)  122880      mixed5[0][0]
__________________________________________________________________________________________________
batch_normalization_55 (BatchNo (None, 10, 10, 160)  480         conv2d_55[0][0]
__________________________________________________________________________________________________
activation_55 (Activation)      (None, 10, 10, 160)  0           batch_normalization_55[0][0]
__________________________________________________________________________________________________
conv2d_56 (Conv2D)              (None, 10, 10, 160)  179200      activation_55[0][0]
__________________________________________________________________________________________________
batch_normalization_56 (BatchNo (None, 10, 10, 160)  480         conv2d_56[0][0]
__________________________________________________________________________________________________
activation_56 (Activation)      (None, 10, 10, 160)  0           batch_normalization_56[0][0]
__________________________________________________________________________________________________
conv2d_52 (Conv2D)              (None, 10, 10, 160)  122880      mixed5[0][0]
__________________________________________________________________________________________________
conv2d_57 (Conv2D)              (None, 10, 10, 160)  179200      activation_56[0][0]
__________________________________________________________________________________________________
batch_normalization_52 (BatchNo (None, 10, 10, 160)  480         conv2d_52[0][0]
__________________________________________________________________________________________________
batch_normalization_57 (BatchNo (None, 10, 10, 160)  480         conv2d_57[0][0]
__________________________________________________________________________________________
__________________________
activation_52 (Activation)      (None, 10, 10, 160)  0           batch_normalization_52[0][0]
__________________________________________________________________________________________
__________________________
activation_57 (Activation)      (None, 10, 10, 160)  0           batch_normalization_57[0][0]
__________________________________________________________________________________________
__________________________
conv2d_53 (Conv2D)              (None, 10, 10, 160)  179200      activation_52[0][0]
__________________________________________________________________________________________
__________________________
conv2d_58 (Conv2D)              (None, 10, 10, 160)  179200      activation_57[0][0]
__________________________________________________________________________________________
__________________________
batch_normalization_53 (BatchNo (None, 10, 10, 160)  480         conv2d_53[0][0]
__________________________________________________________________________________________
__________________________
batch_normalization_58 (BatchNo (None, 10, 10, 160)  480         conv2d_58[0][0]
__________________________________________________________________________________________
__________________________
activation_53 (Activation)      (None, 10, 10, 160)  0           batch_normalization_53[0][0]
__________________________________________________________________________________________
__________________________
activation_58 (Activation)      (None, 10, 10, 160)  0           batch_normalization_58[0][0]
__________________________________________________________________________________________
__________________________
average_pooling2d_6 (AveragePoo (None, 10, 10, 768)  0           mixed5[0][0]
__________________________________________________________________________________________
__________________________
conv2d_51 (Conv2D)              (None, 10, 10, 192)  147456      mixed5[0][0]
__________________________________________________________________________________________
__________________________
conv2d_54 (Conv2D)              (None, 10, 10, 192)  215040      activation_53[0][0]
__________________________________________________________________________________________
__________________________
conv2d_59 (Conv2D)              (None, 10, 10, 192)  215040      activation_58[0][0]
__________________________________________________________________________________________
__________________________
conv2d_60 (Conv2D)              (None, 10, 10, 192)  147456      average_pooling2d_6[0][0]
__________________________________________________________________________________________
__________________________
batch_normalization_51 (BatchNo (None, 10, 10, 192)  576         conv2d_51[0][0]
__________________________________________________________________________________________________
batch_normalization_54 (BatchNo (None, 10, 10, 192)  576         conv2d_54[0][0]
__________________________________________________________________________________________________
batch_normalization_59 (BatchNo (None, 10, 10, 192)  576         conv2d_59[0][0]
__________________________________________________________________________________________________
batch_normalization_60 (BatchNo (None, 10, 10, 192)  576         conv2d_60[0][0]
__________________________________________________________________________________________________
activation_51 (Activation)      (None, 10, 10, 192)  0           batch_normalization_51[0][0]
__________________________________________________________________________________________________
activation_54 (Activation)      (None, 10, 10, 192)  0           batch_normalization_54[0][0]
__________________________________________________________________________________________________
activation_59 (Activation)      (None, 10, 10, 192)  0           batch_normalization_59[0][0]
__________________________________________________________________________________________________
activation_60 (Activation)      (None, 10, 10, 192)  0           batch_normalization_60[0][0]
__________________________________________________________________________________________________
mixed6 (Concatenate)            (None, 10, 10, 768)  0           activation_51[0][0]
                                                                 activation_54[0][0]
                                                                 activation_59[0][0]
                                                                 activation_60[0][0]
__________________________________________________________________________________________________
conv2d_65 (Conv2D)              (None, 10, 10, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
batch_normalization_65 (BatchNo (None, 10, 10, 192)  576         conv2d_65[0][0]
__________________________________________________________________________________________________
activation_65 (Activation)      (None, 10, 10, 192)  0           batch_normalization_65[0][0]
__________________________________________________________________________________________________
conv2d_66 (Conv2D)              (None, 10, 10, 192)  258048      activation_65[0][0]
__________________________________________________________________________________________________
batch_normalization_66 (BatchNo (None, 10, 10, 192)  576         conv2d_66[0][0]
________________________________________________________________________________
__________________________
activation_66 (Activation)      (None, 10, 10, 192)  0           batch_normalization_66[0][0]
________________________________________________________________________________
__________________________
conv2d_62 (Conv2D)              (None, 10, 10, 192)  147456      mixed6[0][0]
________________________________________________________________________________
__________________________
conv2d_67 (Conv2D)              (None, 10, 10, 192)  258048      activation_66[0][0]
________________________________________________________________________________
__________________________
batch_normalization_62 (BatchNo (None, 10, 10, 192)  576         conv2d_62[0][0]
________________________________________________________________________________
__________________________
batch_normalization_67 (BatchNo (None, 10, 10, 192)  576         conv2d_67[0][0]
________________________________________________________________________________
__________________________
activation_62 (Activation)      (None, 10, 10, 192)  0           batch_normalization_62[0][0]
________________________________________________________________________________
__________________________
activation_67 (Activation)      (None, 10, 10, 192)  0           batch_normalization_67[0][0]
________________________________________________________________________________
__________________________
conv2d_63 (Conv2D)              (None, 10, 10, 192)  258048      activation_62[0][0]
________________________________________________________________________________
__________________________
conv2d_68 (Conv2D)              (None, 10, 10, 192)  258048      activation_67[0][0]
________________________________________________________________________________
__________________________
batch_normalization_63 (BatchNo (None, 10, 10, 192)  576         conv2d_63[0][0]
________________________________________________________________________________
__________________________
batch_normalization_68 (BatchNo (None, 10, 10, 192)  576         conv2d_68[0][0]
________________________________________________________________________________
__________________________
activation_63 (Activation)      (None, 10, 10, 192)  0           batch_normalization_63[0][0]
________________________________________________________________________________
__________________________
activation_68 (Activation)      (None, 10, 10, 192)  0           batch_normalization_68[0][0]
________________________________________________________________________________
__________________________
average_pooling2d_7 (AveragePoo (None, 10, 10, 768)  0           mixed6[0][0]
conv2d_61 (Conv2D)           (None, 10, 10, 192)  147456      mixed6[0][0]

conv2d_64 (Conv2D)           (None, 10, 10, 192)  258048      activation_63[0][0]

conv2d_69 (Conv2D)           (None, 10, 10, 192)  258048      activation_68[0][0]

conv2d_70 (Conv2D)           (None, 10, 10, 192)  147456      average_pooling2d_7[0][0]

batch_normalization_61 (BatchNo (None, 10, 10, 192)  576        conv2d_61[0][0]

batch_normalization_64 (BatchNo (None, 10, 10, 192)  576        conv2d_64[0][0]

batch_normalization_69 (BatchNo (None, 10, 10, 192)  576        conv2d_69[0][0]

batch_normalization_70 (BatchNo (None, 10, 10, 192)  576        conv2d_70[0][0]

activation_61 (Activation)     (None, 10, 10, 192)  0          batch_normalization_61[0][0]

activation_64 (Activation)     (None, 10, 10, 192)  0          batch_normalization_64[0][0]

activation_69 (Activation)     (None, 10, 10, 192)  0          batch_normalization_69[0][0]

activation_70 (Activation)     (None, 10, 10, 192)  0          batch_normalization_70[0][0]

mixed7 (Concatenate)           (None, 10, 10, 768)  0          activation_61[0][0]
                                        activation_64[0][0]
                                        activation_69[0][0]
                                        activation_70[0][0]
```

```
__________________________________________________________________________________________________
conv2d_73 (Conv2D)              (None, 10, 10, 192)  147456      mixed7[0][0]
__________________________________________________________________________________________________
batch_normalization_73 (BatchNo (None, 10, 10, 192)  576         conv2d_73[0][0]
__________________________________________________________________________________________________
activation_73 (Activation)      (None, 10, 10, 192)  0           batch_normalization_73[0][0]
__________________________________________________________________________________________________
conv2d_74 (Conv2D)              (None, 10, 10, 192)  258048      activation_73[0][0]
__________________________________________________________________________________________________
batch_normalization_74 (BatchNo (None, 10, 10, 192)  576         conv2d_74[0][0]
__________________________________________________________________________________________________
activation_74 (Activation)      (None, 10, 10, 192)  0           batch_normalization_74[0][0]
__________________________________________________________________________________________________
conv2d_71 (Conv2D)              (None, 10, 10, 192)  147456      mixed7[0][0]
__________________________________________________________________________________________________
conv2d_75 (Conv2D)              (None, 10, 10, 192)  258048      activation_74[0][0]
__________________________________________________________________________________________________
batch_normalization_71 (BatchNo (None, 10, 10, 192)  576         conv2d_71[0][0]
__________________________________________________________________________________________________
batch_normalization_75 (BatchNo (None, 10, 10, 192)  576         conv2d_75[0][0]
__________________________________________________________________________________________________
activation_71 (Activation)      (None, 10, 10, 192)  0           batch_normalization_71[0][0]
__________________________________________________________________________________________________
activation_75 (Activation)      (None, 10, 10, 192)  0           batch_normalization_75[0][0]
__________________________________________________________________________________________________
conv2d_72 (Conv2D)              (None, 4, 4, 320)    552960      activation_71[0][0]
__________________________________________________________________________________________________
conv2d_76 (Conv2D)              (None, 4, 4, 192)    331776      activation_75[0][0]
```

```
__________________________________________________________________________________________________
batch_normalization_72 (BatchNo (None, 4, 4, 320)    960         conv2d_72[0][0]
__________________________________________________________________________________________________
batch_normalization_76 (BatchNo (None, 4, 4, 192)    576         conv2d_76[0][0]
__________________________________________________________________________________________________
activation_72 (Activation)      (None, 4, 4, 320)    0           batch_normalization_72[0][0]
__________________________________________________________________________________________________
activation_76 (Activation)      (None, 4, 4, 192)    0           batch_normalization_76[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 4, 4, 768)    0           mixed7[0][0]
__________________________________________________________________________________________________
mixed8 (Concatenate)            (None, 4, 4, 1280)   0           activation_72[0][0]
                                                                 activation_76[0][0]
                                                                 max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_81 (Conv2D)              (None, 4, 4, 448)    573440      mixed8[0][0]
__________________________________________________________________________________________________
batch_normalization_81 (BatchNo (None, 4, 4, 448)    1344        conv2d_81[0][0]
__________________________________________________________________________________________________
activation_81 (Activation)      (None, 4, 4, 448)    0           batch_normalization_81[0][0]
__________________________________________________________________________________________________
conv2d_78 (Conv2D)              (None, 4, 4, 384)    491520      mixed8[0][0]
__________________________________________________________________________________________________
conv2d_82 (Conv2D)              (None, 4, 4, 384)    1548288     activation_81[0][0]
__________________________________________________________________________________________________
batch_normalization_78 (BatchNo (None, 4, 4, 384)    1152        conv2d_78[0][0]
__________________________________________________________________________________________________
batch_normalization_82 (BatchNo (None, 4, 4, 384)    1152        conv2d_82[0][0]
__________________________________________________________________________________________________
```

```
activation_78 (Activation)      (None, 4, 4, 384)    0          batch_normalization_78[0][0]
__________________________________________________________________________________________________
activation_82 (Activation)      (None, 4, 4, 384)    0          batch_normalization_82[0][0]
__________________________________________________________________________________________________
conv2d_79 (Conv2D)              (None, 4, 4, 384)    442368     activation_78[0][0]
__________________________________________________________________________________________________
conv2d_80 (Conv2D)              (None, 4, 4, 384)    442368     activation_78[0][0]
__________________________________________________________________________________________________
conv2d_83 (Conv2D)              (None, 4, 4, 384)    442368     activation_82[0][0]
__________________________________________________________________________________________________
conv2d_84 (Conv2D)              (None, 4, 4, 384)    442368     activation_82[0][0]
__________________________________________________________________________________________________
average_pooling2d_8 (AveragePoo (None, 4, 4, 1280)   0          mixed8[0][0]
__________________________________________________________________________________________________
conv2d_77 (Conv2D)              (None, 4, 4, 320)    409600     mixed8[0][0]
__________________________________________________________________________________________________
batch_normalization_79 (BatchNo (None, 4, 4, 384)    1152       conv2d_79[0][0]
__________________________________________________________________________________________________
batch_normalization_80 (BatchNo (None, 4, 4, 384)    1152       conv2d_80[0][0]
__________________________________________________________________________________________________
batch_normalization_83 (BatchNo (None, 4, 4, 384)    1152       conv2d_83[0][0]
__________________________________________________________________________________________________
batch_normalization_84 (BatchNo (None, 4, 4, 384)    1152       conv2d_84[0][0]
__________________________________________________________________________________________________
conv2d_85 (Conv2D)              (None, 4, 4, 192)    245760     average_pooling2d_8[0][0]
__________________________________________________________________________________________________
batch_normalization_77 (BatchNo (None, 4, 4, 320)    960        conv2d_77[0][0]
__________________________________________________________________________________________________
activation_79 (Activation)      (None, 4, 4, 384)    0          batch_normalization_79[0][0]
```

```
__________________________________________________________________________________________________
activation_80 (Activation)      (None, 4, 4, 384)    0           batch_normalization_80[0][0]
__________________________________________________________________________________________________
activation_83 (Activation)      (None, 4, 4, 384)    0           batch_normalization_83[0][0]
__________________________________________________________________________________________________
activation_84 (Activation)      (None, 4, 4, 384)    0           batch_normalization_84[0][0]
__________________________________________________________________________________________________
batch_normalization_85 (BatchNo (None, 4, 4, 192)    576         conv2d_85[0][0]
__________________________________________________________________________________________________
activation_77 (Activation)      (None, 4, 4, 320)    0           batch_normalization_77[0][0]
__________________________________________________________________________________________________
mixed9_0 (Concatenate)          (None, 4, 4, 768)    0           activation_79[0][0]
                                                                 activation_80[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 4, 4, 768)    0           activation_83[0][0]
                                                                 activation_84[0][0]
__________________________________________________________________________________________________
activation_85 (Activation)      (None, 4, 4, 192)    0           batch_normalization_85[0][0]
__________________________________________________________________________________________________
mixed9 (Concatenate)            (None, 4, 4, 2048)   0           activation_77[0][0]
                                                                 mixed9_0[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_85[0][0]
__________________________________________________________________________________________________
conv2d_90 (Conv2D)              (None, 4, 4, 448)    917504      mixed9[0][0]
__________________________________________________________________________________________________
batch_normalization_90 (BatchNo (None, 4, 4, 448)    1344        conv2d_90[0][0]
__________________________________________________________________________________________________
activation_90 (Activation)      (None, 4, 4, 448)    0           batch_normalization_90[0][0]
__________________________________________________________________________________________________
```

```
conv2d_87 (Conv2D)              (None, 4, 4, 384)    786432      mixed9[0][0]
__________________________________________________________________________________________________
conv2d_91 (Conv2D)              (None, 4, 4, 384)    1548288     activation_90[0][0]
__________________________________________________________________________________________________
batch_normalization_87 (BatchNo (None, 4, 4, 384)    1152        conv2d_87[0][0]
__________________________________________________________________________________________________
batch_normalization_91 (BatchNo (None, 4, 4, 384)    1152        conv2d_91[0][0]
__________________________________________________________________________________________________
activation_87 (Activation)      (None, 4, 4, 384)    0           batch_normalization_87[0][0]
__________________________________________________________________________________________________
activation_91 (Activation)      (None, 4, 4, 384)    0           batch_normalization_91[0][0]
__________________________________________________________________________________________________
conv2d_88 (Conv2D)              (None, 4, 4, 384)    442368      activation_87[0][0]
__________________________________________________________________________________________________
conv2d_89 (Conv2D)              (None, 4, 4, 384)    442368      activation_87[0][0]
__________________________________________________________________________________________________
conv2d_92 (Conv2D)              (None, 4, 4, 384)    442368      activation_91[0][0]
__________________________________________________________________________________________________
conv2d_93 (Conv2D)              (None, 4, 4, 384)    442368      activation_91[0][0]
__________________________________________________________________________________________________
average_pooling2d_9 (AveragePoo (None, 4, 4, 2048)   0           mixed9[0][0]
__________________________________________________________________________________________________
conv2d_86 (Conv2D)              (None, 4, 4, 320)    655360      mixed9[0][0]
__________________________________________________________________________________________________
batch_normalization_88 (BatchNo (None, 4, 4, 384)    1152        conv2d_88[0][0]
__________________________________________________________________________________________________
batch_normalization_89 (BatchNo (None, 4, 4, 384)    1152        conv2d_89[0][0]
__________________________________________________________________________________________________
batch_normalization_92 (BatchNo (None, 4, 4, 384)    1152        conv2d_92[0][0]
__________________________________________________________________________________________________
batch_normalization_93 (BatchNo (None, 4, 4, 384)    1152        conv2d_93[0][0]
__________________________________________________________________________________________________
conv2d_94 (Conv2D)              (None, 4, 4, 192)    393216      average_pooling2d_9[0][0]
__________________________________________________________________________________________________
batch_normalization_86 (BatchNo (None, 4, 4, 320)    960         conv2d_86[0][0]
__________________________________________________________________________________________________
activation_88 (Activation)      (None, 4, 4, 384)    0           batch_normalization_88[0][0]
__________________________________________________________________________________________________
activation_89 (Activation)      (None, 4, 4, 384)    0           batch_normalization_89[0][0]
__________________________________________________________________________________________________
activation_92 (Activation)      (None, 4, 4, 384)    0           batch_normalization_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 4, 4, 384)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
batch_normalization_94 (BatchNo (None, 4, 4, 192)    576         conv2d_94[0][0]
__________________________________________________________________________________________________
activation_86 (Activation)      (None, 4, 4, 320)    0           batch_normalization_86[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 4, 4, 768)    0           activation_88[0][0]
                                                                 activation_89[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 4, 4, 768)    0           activation_92[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
activation_94 (Activation)      (None, 4, 4, 192)    0           batch_normalization_94[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 4, 4, 2048)   0           activation_86[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_2[0][0]
```

```
                                                    activation_94[0][0]
________________________________________________________________________________
__________________________
global_average_pooling2d_1 (Glo (None, 2048)        0          mixed10[0][0]
________________________________________________________________________________
__________________________
features1 (Dense)             (None, 2048)        4196352     global_average_pooling2d_1[0][0]
________________________________________________________________________________
__________________________
dropout_1 (Dropout)           (None, 2048)        0          features1[0][0]
________________________________________________________________________________
__________________________
features2 (Dense)             (None, 2048)        4196352     dropout_1[0][0]
________________________________________________________________________________
__________________________
dropout_2 (Dropout)           (None, 2048)        0          features2[0][0]
________________________________________________________________________________
__________________________
Prediction (Dense)            (None, 2)           4098       dropout_2[0][0]
================================================================================
==========================
Total params: 30,199,586
Trainable params: 30,165,154
Non-trainable params: 34,432
________________________________________________________________________________
__________________________
```