

# Vision-based Automated Crack Detection using Convolutional Neural Networks for Condition Assessment of Infrastructure

Journal Title  
XX(X):1-13  
©The Author(s) 2020  
Reprints and permission:  
[sagepub.co.uk/journalsPermissions.nav](https://sagepub.co.uk/journalsPermissions.nav)  
DOI: 10.1177/ToBeAssigned  
[www.sagepub.com/](https://www.sagepub.com/)  
SAGE

Aravinda S. Rao<sup>1</sup>, Tuan Nguyen<sup>2</sup>, Marimuthu Palaniswami<sup>1</sup> and Tuan Ngo<sup>2</sup>

## Abstract

With the growing number of aging infrastructure across the world, there is a high demand for a more effective inspection method to assess its conditions. Routine assessment of structural conditions is a necessity to ensure the safety and operation of critical infrastructure. However, the current practice to detect structural damages such as cracks depends on human visual observation methods, which are prone to efficiency, cost and safety concerns. In this article, we present an automated detection method, which is based on Convolutional Neural Network (CNN) models and a non-overlapping window-based approach, to detect crack/non-crack conditions of concrete structures from images. To this end, we construct a dataset of crack/non-crack concrete structures, comprising 32,704 training patches, 2,074 validation patches and 6,032 test patches. We evaluate the performance of our approach using 15 state-of-the-art CNN models in terms of number of parameters required to train the models, Area Under the Curve (AUC), and inference time. Our approach provides over 95% accuracy, over 87% precision in detecting the cracks for most of the CNN models. We also show that our approach outperforms existing models in literature in terms of accuracy and inference time. The best performance in terms of AUC was achieved by Visual Geometry Group's VGG-16 model (AUC=0.9805) and best inference time was provided by AlexNet (0.32 seconds per image of size 256 x 256 x 3). Our evaluation shows that deeper CNN models have higher detection accuracies, however, they also require more parameters and have higher inference time. We believe this study would act as a benchmark for real-time, automated crack detection for condition assessment of infrastructure.

## Keywords

Structural Health Monitoring, Automated Assessment, Crack Detection, Deep Learning, Convolutional Neural Network (CNN)

## 1 Introduction

The growing number of aging critical infrastructure (e.g. bridges, roads) around the world has increased concerns for the operational efficiency and safety of these structures. The capacity of infrastructure can be deteriorated during their service life due to the presence and development of structural damages such as cracks. Current mainstream methods of infrastructure assessment involve performing visual inspection periodically to inform management agencies the current stage of infrastructure. Hence maintenance and strengthening works can be carried out timely to assure the operational efficiency and safety of critical infrastructure. For example, the current level-1 and level-2 inspection guidelines heavily rely on visual inspection carried out by qualified inspectors to detect visible cracks on the

surface of structures<sup>1</sup>. However, the current visual inspection practice has been identified as a costly, time-consuming and subjective method<sup>2,3</sup>. Manual inspection of large infrastructure such as long-span bridges requires inspectors to enter hazardous areas or inaccessible to physical location limits, which not only affects the reliability and efficiency of the inspection but is also a safety concern for inspector<sup>4</sup>.

<sup>1</sup>Department of Electrical and Electronic Engineering, The University of Melbourne, Melbourne, Australia

<sup>2</sup>Department of Infrastructure Engineering, The University of Melbourne, Melbourne, Australia

### Corresponding author:

Tuan Ngo, Department of Infrastructure Engineering, The University of Melbourne, Melbourne, Australia.

Email: [dtngo@unimelb.edu.au](mailto:dtngo@unimelb.edu.au)

With rapid advancements in automation technologies, there is an increasing trend in managing critical infrastructure using autonomous and intelligent inspection systems<sup>5,6</sup>. An autonomous inspection system is usually equipped with a visual camera for taking high resolution images, and therefore, it requires an automated detection for structural damages to maximize the benefits of the automated inspection system<sup>6</sup>. Recently vision-based systems appear to be a promising solution for an autonomous inspection system to analyze images and detect cracks on structures. Applications of vision-based systems could be found in detecting cracks in: dams<sup>7</sup>, bridges<sup>8</sup>, road surfaces<sup>9</sup>, concrete surfaces<sup>10</sup>, concrete bridges<sup>11</sup> and also detecting potholes<sup>12</sup>. The term “automated detection” or “autonomous” refers to the process in which, whenever the system is provide with an image or a video, the system will then highlight the defective surfaces without any human intervention or human input other than the input data (images or videos). In other words, the trained system automatically detects the defects (such as cracks) from images or videos without any additional input.

The advantages of vision-based methods is that they capture 2D/3D information of the structures. This will enable automated systems to detect superficial defects (cracks, corrosion) as well as add comprehensive information about the structures. In addition, vision-based systems provide accurate information compared to manual inspection and crack detection using contact-based sensors<sup>13</sup>. Recently, deep learning (DL) algorithms have accelerated many vision-based systems to provide better detection accuracy. DL algorithms utilize neural networks to build a deeper network architecture to hierarchically extract important features automatically. DL<sup>14</sup> has been extensively used in object detection and classification tasks, human action recognition, face recognition, natural language processing, medical image processing, pedestrian detection and tracking<sup>15</sup>, safety of construction works<sup>16</sup> and also in the detection of cracks<sup>17</sup>.

Broadly, vision-based methods can be classified into four categories: (1) image processing methods use signal processing tools to detect cracks, including edge filters and then extract features manually to detect cracks using machine learning; (2) region-based classification methods aim to detect cracks by localizing the cracks in the image regions—this is done by creating patches of images and then classifying (using traditional machine learning or the recent deep learning approaches) whether or not a patch contains a crack; (3) object-detection methods using generic objection schemes to detect cracks along with other objects, usually use deep learning object detectors; and (4) segmentation methods detect cracks by classifying whether

each pixel belongs to a crack or any other object—this approach requires more computational power. Each method has its own advantages and disadvantages, but region-based crack detection is highly sought-after to localize the cracked regions, especially on concrete surfaces for automated inspection. Existing region-based methods use Convolutional Neural Networks (CNNs)<sup>18</sup> incorporating image patches (of sizes 256 x 256, 520 x 520) such that they can detect cracks from large images (5888 x 3584, 4096 x 4800) with the similar patch size used for training. However, one of the drawbacks of these approaches is that they are designed to identify cracks in a coarse manner—cracks are assumed to be of the same size as that of patch. To elaborate, suppose we have a crack of only 16 x 16 in an image of size 256 x 256, then the entire image will be classified as containing crack. Identification of patches by such coarse methods is not suitable if we want to detect cracks at finer levels, say 16 x 16. The existing methods also need to be retrained and would be computationally more expensive because of multiple scans required to detect cracks on the edges of scanning windows.

To address these issues, in this article, we present a non-overlapping window-based approach to detect concrete cracks from images at finer level (windows size of 64 x 64) on smaller images mainly targeted toward real-time applications. The proposed approach can be extended to larger images as well (keeping the same window size) without re-training or scanning the images multiple times. We do performance evaluation of our approach using 15 state-of-the-art CNN models and any of these models can be used to detect cracks on concrete surfaces depending on the user needs while balancing accuracy vs. inference time.

The article is organized as follows: Section 2 provides a detailed review of the existing works; Section 3 presents our approach using 15 CNN models and explains each of the models; Section 4 includes details of dataset, implementation, handling imbalanced data, loss function, network optimization, online learning and training the models. Section 5 includes results of the CNN models in terms of Area Under the Curve (AUC), optimal operating thresholds and inference time, and also comparison of the results with discussion; and Section 6 concludes the work.

## 2 Related Work

In this section, we review the existing work related to crack detection in Structural Health Monitoring (SHM) under four subsections: (1) Image Processing Methods, (2) Region-based Classification Methods, (3) Object Detection Methods,

and (4) Semantic Segmentation Methods. The following subsection reviews each approach in detail.

## 2.1 Image Processing Methods

Image processing techniques are quite attractive to visually inspect critical infrastructure. For example, histogram of pixels were employed to detect cracks using an expert system to automatically detect spalling and transverse cracks<sup>19</sup>. Sobel, Canny, Fourier Transform and Fast Haar Transform, which are some of the classical signal processing techniques, were also used to detect edges, and subsequently cracks in the images<sup>20</sup>. Region growing methods<sup>21</sup>, which rely on the connectivity of pixels in the cracked regions, first convert the images to binary images (0 or 1 values of each pixel) using a threshold, set the initial seed for percolation of crack using the edge information in the local windows and then grow the crack regions for detecting the cracks. A combination of edge detection and region filling techniques by connecting the 8-neighborhood pixels with Dijkstra's shortest path search algorithm<sup>10</sup> were also used to detect cracks in concrete structures. Furthermore, edge detection and morphological dilation was used to detect cracks using Unmanned Aerial Vehicles (UAVs)<sup>22</sup>.

With the advancements in machine learning, such as the superior performance of Support Vector Machines (SVMs)<sup>23</sup> and Neural Networks (NN)<sup>24</sup> for classification tasks in the early 2000s, SVMs were employed to classify image patches containing cracks by extracting features from Hough Transform<sup>25</sup>. Texture features, such as Gray Level Coefficient Matrix (GLCM) features with NN classifiers were also employed<sup>26</sup>. Furthermore, Laplacian of Gaussian (LoG) weighted Haar-like (edge) features were extracted and fed to Adaptive Boosting (AdaBoost) learning algorithm, which will keep the best predictive features to detect cracks<sup>27</sup>. To handle illuminations, a semi-automated approach was proposed by manually cropping crack regions, filtering noise based on wavelet filters and then segmenting the image using energy functionals into background and crack<sup>28</sup>. A heuristics approach was proposed to localize the cracks in images using hierarchical clustering<sup>29</sup>. Spatially tuned multifeature (STRUM) uses localized line segments to detect cracks and provides an accuracy of 95%<sup>11</sup>. We find that manually extracting features from images (or patches) and then training a classifier is often not on par with deep learning models.

## 2.2 Region-based Classification Methods

The main challenges in detecting cracks from images (or videos) are that the features must be invariant to scale, translation, noise, lighting conditions and shadows. The manual feature extraction methods, which uses handcrafted features are suitable for a specific case and often fail to perform well when tested in real-world conditions. Recently, DL algorithms, such as the convolution neural networks (CNNs) have shown promising results for real world applications<sup>30</sup>. With the availability of large amounts of training data, the CNN-based architectures usually outperform their shallow counterparts due to generalization of features through hierarchical learning of features at different abstract levels.

In this aspect, detecting cracks/non-cracks is an important problem in SHM research. For this, both SURF-based and CNN-based classification approaches were tested<sup>18</sup>. Compared with traditional edge detection methods in classifying each image patch as "crack" or "intact", a CNN-based sliding-window architecture with 8 layers trained using 40k images of  $256 \times 256$  pixels has shown superior performance of 98% accuracy<sup>17</sup>.

CNN-based Residual Neural Network (ResNet), which tries to address the problem of vanishing gradient in deeper networks by making use of skip connections, showed a slightly lower performance (of 87.5% accuracy with 35 parameter layers) when classifying cracks, deposit and water leakage<sup>31</sup>.

To reduce false positives, we also see the utilization of infrared images in conjunction with visual images to train a CNN model [GoogleNet<sup>32</sup> with 22 layers]<sup>33</sup>. Bayesian data fusion has also been explored for detecting cracks. For example, crack detection using Local Binary Pattern (LBP), SVM and Bayesian fusion delivered an accuracy of 85%. An improvement in the performance (with a sensitivity of 98.3%) was achieved with Naive Bayes (NB)-CNN architecture<sup>34</sup>. NB-CNN uses 11-layer with overlapping image patches ( $120 \times 120$ ) to detect cracks in different video frames.

## 2.3 Object Detection Methods Applied to Crack Detection

Object detection methods use region proposal methods<sup>35</sup> and region-based CNN (R-CNN)<sup>36</sup> to detect objects. Region proposal methods are computationally expensive and uses Selective Search<sup>35</sup> greedily to generate possible locations of objects. However, the R-CNN<sup>36</sup> approach had higher accuracy, but computationally expensive as R-CNN

architecture performs ConvNet for each object proposal in the forward pass. Later, Fast R-CNN<sup>37</sup> was introduced to address the drawbacks of R-CNN by training end-to-end and showing higher accuracy. However, Fast R-CNN has limitations in generating object proposals as it is dependent on selective search<sup>35</sup> object proposals, which is time-consuming and acts as a bottleneck. To take advantages of both region proposals and Fast R-CNN, Region Proposal Network (RPN) was introduced. Faster R-CNN<sup>38</sup> combines RPN and Fast R-CNN to achieve state-of-the-art object detection results.

In structural assessments, Faster R-CNN (region-based CNN)<sup>38</sup> was utilized to classify multiple types of structural damage from images, such as concrete crack, steel corrosion (medium and high), bolt corrosion, and steel delamination with a mean average precision (mAP) of 89.7% and average precision (AP) of 94.7% for detecting concrete cracks<sup>39</sup>. It is to be noted that object detection methods such as Faster R-CNN perform well when detecting objects of different classes, but it performs poorly when one wants to only detect cracks - we show this in Section 5. In<sup>39</sup>, the good AP score is because there were other distinguishing objects other than cracks, which helped to develop feature maps that made it possible to detect cracks. Faster R-CNN is also used to detect concrete spalling with a mAP of 90.79%<sup>40</sup>. Faster R-CNN with ResNet-101 provided a mAP of 90% for detecting spalling on historic masonry buildings. A modified version of the faster R-CNN, called *CrackDN*<sup>41</sup>, integrates sensitivity detection network and Region Proposal Refinement Network (RPRN) to detect sealed and unsealed cracks. CrackDN provides a mAP of 0.9, better than Faster R-CNN and *Single Shot Detector* (SSD).

## 2.4 Semantic Segmentation Methods

Semantic segmentation approaches endeavor to classify each pixel into one of the pre-determined classes (for example, the pixels could either belong to “crack” or “no crack” class). In other words, semantic segmentation is a natural progress from coarser inference (such as, patch-based, region-based, object-detection) to finer inference. A study consisting of six edge detectors (Roberts, Prewitt, Sobel, Laplacian of Gaussian, Butterworth, and Gaussian) and CNN [AlexNet<sup>42</sup>] to classify pixels into “crack” and “non-crack” showed that the edge-based techniques for classifying pixels is sub-optimal when compared with CNN approaches<sup>43</sup>. This result reinforces our previous view that edge-based methods are sub-optimal in detecting cracks. *CrackNet*<sup>44</sup> is a CNN-based 5-layer architecture designed for automated pavement crack detection on 3D asphalt surfaces with a precision of

90.13% and recall of 87.63%. The results are superior when compared with the Pixel-SVM<sup>45</sup>, which extracts features from non-overlapping pavement image patches and uses SVM to classify the patches.

Alternatively, Fully Convolutional Network (FCN) has shown remarkable progress in classifying pixels<sup>46</sup>. FCN has encoder-decoder network to encode (extract features from input images along with one of the backbone architectures, such as VGG16, VGG-19, ResNet, etc to classify, but without the final output layer) and decode (deconvolve and upsample layers to reconstruct segmented images). FCN with VGG-16 as encoder architecture<sup>47</sup> trained on 40k images with  $227 \times 227$  pixels and tested on 500 images provided an accuracy of 90% in classifying pixels. In addition, FCN with DenseNet-121 as encoder provides pixel accuracy of 98.61%<sup>48</sup>. However, FCN with VGG-19 as backbone<sup>49</sup>, did not improve the maximum accuracy (81.73%), with precision and recall of 78.97% and 79.95%, respectively), but FCN reduced the training time required for training from several days (CrackNet) to hours. We also see the variants of CNN such as Mask R-CNN<sup>50</sup> being used for crack detection. DeepCrack<sup>51</sup> extends FCN by combining FCN and Deeply-Supervised Nets (DSN)<sup>52</sup>, and applies both Conditional Random Fields (CRFs) and guided filtering to improve prediction of pixel-wise semantic segmentation of crack with a mean Intersection of Union (IoU) of 0.86.

More recently, U-Net trained network was proposed to detect concrete cracks<sup>53</sup>. Like FCN, U-Net uses encoder-decoder network but with modifications, including: (1) U-Net is symmetric in network structure consisting of contracting and expansive paths (*i.e.*, the shape of the network from input to output look “U”, hence the name “U-Net”), (2) U-Net uses skip connections between upsampling and downsampling paths, and (3) the pooling operators in the expansive path are replaced by upsampling operators. One of the key features of U-Net is its ability to learn from limited training data. In<sup>53</sup>, only 57 images were used to train and it provided an F1-score of 90%. Depending on the end-user application, semantic segmentation approaches can be useful. For example, U-Net is agnostic to input image size, whereas it requires considerable amount of training time for relatively larger image sizes.

Table 1 provides a summary of image processing and region-based classification methods with their advantages and disadvantages. Likewise, Table 2 provides a summary of object detection and semantic segmentation methods applied for crack detection.

### 3 Our Approach

In this work, we present a patch-based approach using 15 deep learning classification models to identify the image patches with cracks. We use the existing state-of-the-art CNN models to compare their effectiveness in detecting the cracked regions. We approach this by creating non-overlapping patches of images and then classifying whether a given image patch contains a crack or otherwise.

#### 3.1 Overview of the proposed approach

Figure 1 shows the overview of the crack detection approach used in this work. Figure 1 (a) shows the training phase and Figure 1 (b) shows the testing (or the inference) phase. In both phases, input image of size  $256 \times 256$  and 24-bit depth (three channels) is divided into 16 patches (each patch of size  $64 \times 64$  - the number of channels remain the same). Then, the image patch is fed into one of the CNN models by resizing the patch and normalizing each patch). Resizing of patch is done such that it matches the specific CNN model input size. Please refer to Table 3 for more information about the input size for each of the models used in this work. During training, we use image patches belonging to “crack” and “no crack”, and test the model against validation image patches. During testing, the test image is first divided into patches, normalized and then the trained model is used for inference (to predict) whether or not there is a crack in each patch. We highlight the patch depending on whether or not there was a crack.

In the following subsections, we briefly discuss each of the 15 CNN architectures as these are the current state-of-the-art models. Interested readers are encouraged to refer to the references for more detailed description of the CNN models. Later in the Section 4, we provide details of how we trained the models and evaluated the performance of each of these models in detecting cracks.

#### 3.2 AlexNet

AlexNet<sup>42</sup> was trained on Imagenet data consisting of 1.2 million images with 1000 classes. It won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)-2010 and ILSVRC 2012, with the best ever reported results. In this network, the authors used Rectified Linear Units (ReLU) as activation functions. We use this network architecture and modify the final fully connected (FC) layer to have only two classes (“crack” or “no crack”), resulting in approximately 57M parameters (Table 3). Figure 2(a) shows the modified network architecture.

#### 3.3 Visual Geometry Group (VGG) Networks

Visual Geometry Group (VGG) from Oxford University proposed CNN networks with 16 and 19 layers, popularly known as VGG-16 and VGG-19 architectures<sup>54</sup>. We used these networks with the output layer changed to two classes (instead of 1000 in the original VGG architecture) as shown in Figure 2 (b) and Figure 2 (c), respectively. VGG-16 and VGG-19 with two output classes have close to 134.2 million and 139.5 million, respectively (Table 3). In both the cases, the networks use  $3 \times 3$  convolution layers stacked on one another, increasing the depth of the networks.

#### 3.4 Residual Networks (ResNets)

Increasing the depth of the networks *i.e.*, adding more layers to the network, such as in VGG-16 and VGG-19, it was shown that networks could learn well. However, it also exposed one of the important problems in training deeper networks: *degradation* of training accuracy—it is not easy and also not the same to optimize deeper networks<sup>55</sup>. To overcome this problem of degradation, *deep residual learning* framework was introduced. The *residual networks* add *identity* mapping between a group of stacked layers, which do not add any extra parameter for learning. The blocks of network layers along with identity mappings form residual mapping and is easier to optimize the learning with deeper networks. Let  $x$  and  $y$  represent input and output vectors of layers. Suppose, if we have  $\mathcal{H}(x)$  as the mapping to be fit by a few stacked layers, then the residual function is given by  $\mathcal{F}(x) : \mathcal{H}(x) - x$ . With this formulation, if the identity mappings are optimal, then the weights of the multiple non-linear weights will be driven towards zero, to approximate identity mappings. Residual networks (ResNets) are inspired by VGG networks, but have lower complexities. ResNet-50, ResNet-101 and Resnet-152 have 50, 101 and 152 layers, and 23.5, 58.1 and 235.1 million parameters, respectively (Table 3). Figures 2 (c)–(e) show the network architectures of ResNet-50, ResNet-101 and Resnet-152 used in this work, with the last FC layer modified to two outputs for detecting cracks.

#### 3.5 Inception Networks

The Inception architecture (also called as “GoogLeNet”) is based on the idea of representing dense components by optimal local sparse structure in a convolutional network. It also applies dimensionality reduction (*i.e.*, low-dimensional embeddings) and projections wherever computational resources are limited. In other words, Inception network combines the two ideas using stacked

layers with occasional max pooling layers<sup>32</sup>. Inception networks use 12x fewer parameters than AlexNet, while maintaining high accuracies. One of the advantages of Inception networks is that it allows to increase the number of hidden units at each stage without significantly increasing the computational complexity of the network. With dimensionality reduction at each layer, practically it allows to use the improved computational resources to increase the width and depth at each stage.

Inception-v3<sup>56</sup> uses (1) the idea of factorizing larger spatial filter into smaller ones, and (2) replacing a symmetric spatial convolutional filter with multiple asymmetric filters. For example, (1) instead of using  $5 \times 5$  filter, one could use two  $3 \times 3$  filter, which would reduce the computational load and also increase the non-linearity; and (2) instead of using a single  $3 \times 3$  convolutional layer, one could go for two layers with  $3 \times 1$  followed by  $1 \times 3$  layers. This approach could be generalized to replace any  $n \times n$  convolution by a  $1 \times n$  convolution, followed by a  $n \times 1$  convolution, which would increase the computational cost savings as  $n$  increases. Inception-v3 has 42 layers and costs about 2.5x GoogLeNet, but less than VGG networks. Figure 3 (a) shows the Inception-v3 network architecture used for detecting cracks. Inception-v3 for crack detection has 25.1 million parameters (Table 3).

Inception-v4<sup>57</sup> improves on the architecture of Inception-v3 by adding more layers and a more simplified and uniform architecture. Figure 3 (b) shows the Inception-v4 architecture with simplified architecture. The “stem” block in the Inception-v4 (Figure 3 (b)) describes the early stage convolution, pooling and normalization operations. To answer the question of whether adding residual connections would improve the performance of inception networks, Inception-ResNets, such as the Inception-ResNet-v1 and Inception-ResNet-v2, were introduced<sup>57</sup>. Inception-ResNet-v1 has similar computational cost as Inception-v3. On the other hand, Inception-ResNet-v2 combines Inception-v4 and ResNet, which is hybrid and has costlier computational cost, but with improved recognition performance. In this work, we have used Inception-ResNet-v2 for detecting cracks and Figure 3 (c) shows the network architecture. Inception-v4 and Inception-Resnet-v2 have 41.1 and 54.3 million parameters, respectively (see Table 3).

### 3.6 Dense Convolutional Networks (DenseNets)

Traditional convolutional networks with  $L$  layers have  $L$  connections, whereas Dense Convolutional Network

(DenseNet) connects each layer to every layer in a feed-forward approach<sup>58</sup>. Thus, DenseNets have  $\frac{L(L+1)}{2}$  connections, resulting in fewer parameters than traditional convolutional networks. ResNets do not combine feature through summations before being passed into the next layer, whereas DenseNets combine them by concatenating the inputs from the preceding layers as well as its own feature-maps. As a result, there is also an improved flow of information and gradients throughout the network, which helps in training deeper networks. In addition, DenseNet network architectures have regularization effects on small training sets, reducing overfitting during training. Figures 3 (d) and (e) show the DenseNet with a depth of 121 and 169 layers, respectively. From Table 3, we see that DenseNet-121 and DenseNet-169 require only 6.9 and 12.4 million parameters for detecting cracks, which are significantly fewer than the rest of the networks.

### 3.7 Aggregated Residual Transformations

In the previous subsections, we saw that Inception models<sup>56,57</sup> have a common property of *split-transform-merge* strategy, wherein the input is split into a few lower-dimensional embeddings, transformed by specialized filter and merged by concatenating them. However, it will be challenging to adapt this strategy to new data sets. To address this issue, aggregated residual transformations<sup>59</sup> were introduced by adopting VGG/ResNets’ scheme of repeating layers and utilizing split-transform-merge of Inception modules within the same topology. This redesign results in a homogenous, multi-branch architecture with a fewer set of parameters. It also enables a new dimension called “cardinality,” which represents the size of transformations. This new dimension is an additional dimension that enables to describe the networks along with size and width. In this work, we have included ResNeXt-50 (32x4d) and ResNeXt-101 (32x8d) as shown in Figures 4 (a) and (b). In the ResNeXt-50 (32x4d), 50 indicates the network with a depth of 50 layers, 32 represents the cardinality of transformations and 4d represents the bottleneck width, which is part of ResNet. Similar expansion applies to ResNeXt-101 (32x8d). ResNeXt-50 (32x4d) has 22.9 million parameters (see Table 3), which is slightly lesser than ResNet-50, whereas ResNeXt-101 (32x8d) has 86.7 million parameters (an additional 28.6 million parameters when compared with ResNet-101).

### 3.8 Wide Residual Networks (Wide ResNets)

As we have seen in the previous subsections, ResNets use identity mappings to train very deep networks. However, this also has a disadvantage: there is no guarantee that the network would make the gradient flow through the residual weights. So, only some of the blocks could learn representation or very little information could be shared among blocks. To increase a fraction of accuracy requires nearly doubling the number of layers, indicating the problem of diminishing feature re-use. To address this issue, Wide Residual Networks (Wide ResNets)<sup>60</sup> were introduced. Wide ResNets decrease the depth of network while increasing the width of residual networks. In this work, we have included Wide-ResNet-50-2 and Wide-ResNet-101-2 for comparison with other models and the network architectures for detecting cracks are shown in Figures 4 (c) and (d). In the naming convention, Wide-ResNet-50-2 represents a wide ResNet with 50 layers deep and 2x width of the original ResNet-50 architecture (similar naming convention applies to Wide-ResNet-101-2). In comparison to the original ResNet-50 network, the Wide-ResNet-50-2 has almost 3x the parameters (*i.e.*,  $\approx 66.8$  million), whereas Wide-ResNet-101-2 has 2x the parameters of the original ResNet-101 (*i.e.*, Wide-ResNet-101-2 has nearly 124.8 million parameters).

## 4 Training and Evaluation

### 4.1 Dataset

The dataset gathered consists of 2,173 training images (2044 train + 129 validation) and 377 test images of size  $256 \times 256$  and 24-bit depth (three channels). The cracks in the training and test dataset were annotated manually using LabelImg\*. Further, each image was divided into  $64 \times 64$  non-overlapping image patches, resulting in 16 patches per image. Using the annotation information for each image, labels were generated for each patch based on whether there was a “crack” or “no crack”, and manually verified for each patch. Table 4 provides the details of number of patches used for training, validation and testing the models. Our dataset, trained models and codes are available publicly for future development in this research area †.

### 4.2 Implementation

All the models were implemented using Python 3.6 and PyTorch 1.1. We used a cloud-based service consisting of Nvidia K80 Graphics Processing Unit (GPU) card with 11 GB graphics memory and Intel Xeon® (with 2 cores) Central Processing Unit (CPU) to train the models. To test the

performance of the trained models, we used a laptop with 64-bit Ubuntu Operating System (OS), 16 GB RAM and 8<sup>th</sup> generation Intel® Core™ i7-8550 CPU with a clock speed of 1.80 GHz.

### 4.3 Handling Data Imbalance

As can be seen from Table 4, the number of training patches for class 0 is significantly higher (*i.e.*, the “no crack” training patches accounts for about 72.76% of the entire training data), whereas the number of training patches for class 1 (“crack”) is only 27.24%, a highly skewed dataset which forces CNN models to be biased towards majority class. As a result, if we do not manage this data imbalance, we are likely to have incorrect classification results on the test samples. However, the data imbalance is a common phenomenon that we observe in everyday practical applications. To ensure that the CNN models learn meaningful discriminating representations of both the classes (0 and 1), we provide a weight tensor during training as:

$$w = [\text{weight for class 0, weight for class 1}] \quad (1)$$

$$= \left[ \frac{\# \text{ of training samples in class 1}}{\text{Total \# of training samples}}, \frac{\# \text{ of training samples in class 0}}{\text{Total \# of training samples}} \right] \quad (2)$$

$$= [0.27, 0.73] \quad (3)$$

The expression (3) indicates that the classifier learns the thresholding boundary between two classes such that it counteracts the majority class by adding more weight (0.73) to class 1 (“crack”) samples and less weight (0.27) to class 0 (“no crack”). This ensures that the CNN models learn a balanced threshold in the sample space and provide an unbiased prediction in test cases.

### 4.4 Loss Function

In this work, the problem of detecting cracks in an image has been formulated as a binary (two-class) classification problem by dividing the images into patches, classifying each patch depending on whether or not there is a crack and subsequently labeling the patch containing crack. For this binary classification problem, the negative log likelihood (NLL) function is given by:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{n} \sum_{i=1}^n y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i), \quad (4)$$

\*<https://github.com/tzutalin/labelImg>

†[https://drive.google.com/open?id=1m3AE\\_sghjfSb7SkzygVKHUqzABpUtr7m](https://drive.google.com/open?id=1m3AE_sghjfSb7SkzygVKHUqzABpUtr7m)

where  $y$  is the actual label,  $\hat{y}$  is the predicted label and  $n$  is the number of samples. This function is also called as the *binary cross entropy loss* as it is a special case of cross entropy (measure of distance between two probabilistic distributions  $\mathbf{y}$  and  $\hat{\mathbf{y}}$ ) function. It is to be noted that as NLL is a convex function, it provides a unique minimum. Here,  $\mathbf{y}$  can be treated as the distribution of input class provided to the model and  $\hat{\mathbf{y}}$  is the distribution of class predicted by the model. From information theory standpoint, entropy is referred to the randomness in an event and it is represented in terms of information bits. Then, the binary cross entropy is representing the additional bits required to represent the target class because of randomness in the data in place of true class. In the equation (4), we can consider the term  $y_i \log(\hat{y}_i)$  referring to the cross entropy of “crack” class and the second term  $(1 - y_i) \log(1 - \hat{y}_i)$  referring to the cross entropy of “no-crack” class. These terms are summed and averaged over the input datapoints (images in our case). A value of  $l(\mathbf{y}, \hat{\mathbf{y}}) = 0$  represents zero loss, implying the model was able to predict perfectly. A value greater than zero indicates that there is a room for improving the model’s performance by tuning model parameters. The Bias-Variance is a well-known trade-off in machine learning that aims balance the model underfitting or overfitting whenever we try to train and test the model<sup>61</sup>. Considering this trade-off, the goal in our approach is to drive the loss function error  $l(\mathbf{y}, \hat{\mathbf{y}})$  to as low as possible while ensuring that the model does not underfit or overfit for the input data. To improve the model’s performance by reducing this error in each target class, we use optimization algorithms, such as, the stochastic gradient descent (SGD) algorithm described in the next subsection.

#### 4.5 Network Optimization and Online Learning

The objective of neural network optimization is to update the (weight of) parameters of the neurons in conjunction with activation functions such that the neural network achieves the global minimum loss, resulting in a reduced generalization error with optimized network for a particular application. In other words, the optimization problem is to try to find the global minimum on the surface of loss function. For this we need large amounts of data and as the size of the (image) dataset grows, we are limited by the capacity of data that can be processed at a time. This calls for *online learning* in which we consider a batch of  $B$  samples and update our parameter estimates as the new batch of input data arrives rather than updating the parameters once after all the samples have been observed. This is also applicable to streaming data such as video and one-dimensional sensor data. We can define the loss  $L(z, \theta)$  incurred on sample  $z$  when the parameter

takes on  $\theta$ , where the gradient associated with the sample is  $\frac{\partial L(z, \theta)}{\partial \theta}$ . For a mini-batch of size  $B$ , the gradients at time  $t$  using stochastic mini-batch gradient descent is given by<sup>62</sup>:

$$\theta_t \leftarrow \theta_{t-1} - \epsilon_t \frac{1}{|B|} \sum_{t'} \frac{\partial L(z_{t'}, \theta)}{\partial \theta} \quad (5)$$

$$\theta_t \leftarrow \theta_{t-1} - \epsilon_t \frac{1}{|B|} \nabla F(\theta_t) \quad (6)$$

where  $z_{t'}$  is the sample in a batch of size  $B$ ,  $\epsilon_t$  is the learning rate,  $|\cdot|$  is the cardinality of the set and  $\nabla F(\theta_t)$  is the short-form notation of the gradient vector. We use a mini-batch size of 32 and SGD was used as it has been shown that SGD finds a flatter minima than other optimizers. With a constant learning rate, SGD may not always be able to find the global minimum. Hence, momentum-based approaches consider also the velocity of gradients, which have shown to perform better in training deeper networks. This can be formally defined as<sup>63</sup>:

$$v_{t+1} = \mu v_t - \epsilon_t \frac{1}{|B|} \nabla F(\theta_t) \quad (7)$$

$$\theta_{t+1} = \theta_t + v_{t+1}, \quad (8)$$

where  $\mu \in [0, 1]$  is the momentum decay coefficient. We use Step Learning Rate scheduler with a learning rate  $\epsilon_t = 0.001$  and  $\mu = 0.9$ . Further, our implementation considers step learning decay of 4 epochs, and learning rate decay ( $\gamma$ ) of 0.1. During training, all layers were allowed to update gradients and we have used 50 epochs for comparing the efficacy of the 15 CNN models. Although we ran 50 epochs for each models, only the best network weights were stored and used as trained models for evaluating the performance on the test images. The total number of training iterations with a mini-batch of size of 32 and 32,704 patches, is 1,022 and the total number of iterations for 50 epochs is 51100. Our implementation of the 15 CNN models considers patches of “no-crack” as class 0 and “crack” patches as class 1.

#### 4.6 Classifier Performance Metrics

We define a prediction function  $\delta : \mathcal{X} \mapsto \mathcal{Y}$  (where  $x \in \mathcal{X}$  is the input and  $y \in \mathcal{Y}$  is the target) and the cost function  $L(\mathbf{y}, \hat{\mathbf{y}})$  as given in (4). Suppose we have a decision function  $\delta(x) = \mathbb{I}(f(x) > \tau)$ , where  $f(x)$  measures the confidence that  $y = 1$  and  $\tau$  is the threshold parameter<sup>64</sup>, then for the binary classification problem of detecting the presence of cracks in a given image patch, the performance metrics are defined<sup>65</sup> by varying the threshold ( $\tau$ ) and calculating the confusion matrix is defined in Table 5.

From Table 5, we can define the following performance metrics:

$$\text{True Positive Rate (TPR)} = \frac{TP}{TP + FN} \quad (9)$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} \quad (10)$$

$$\text{Specificity} = \frac{TN}{FP + TN} \quad (11)$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \quad (12)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (13)$$

We use the above metrics to measure the performance of each model and compare them objectively to understand the suitability of each model for real-time applications.

#### 4.7 Training CNN Models

For training the models to classify each patch, we use ImageNet pre-trained models as our initial model weights and then train our models through *transfer learning* to detect cracks. Figure 5 shows the sample training and validation loss curves along with model accuracies for AlexNet [(a) and (b)], and VGG-16 [(c) and (d)], respectively. Being mindful of the space and readability of the article, we do not present the loss and accuracy graphs for the remaining 13 models in this paper; however, these can be accessed from the public folder<sup>‡</sup>. From the graphs, we observe that the training loss is lower and the validation accuracy is also lower, indicating this small error is common in practical applications (as expected). Figure 6 shows the sample results of VGG-16 from the training step. In Figure 6, (a)–(h) represent the manually annotated ground truths, and (i)–(p) show the corresponding predicted output of VGG-16 model with a probability (decision,  $\tau$ ) threshold of 0.6. We see that the VGG-16 model was able to correctly predict all 16 patches containing cracks in all images.

### 5 Results and Discussion

#### 5.1 Receiver Operating Characteristics (ROC)

We compare the efficacy of the 15 models using ROC-AUC. AUC is a well-known metric used in machine learning to evaluate the performance of classifiers with the highest being 1. The ROC AUC are generated by varying the threshold parameter  $\tau$  (see Section 4.6). Figure 7 shows the comparison of all 15 models. We notice that VGG-16 has the highest AUC (0.9805), followed by ResNet-152 (0.9788) and AlexNet (0.9780). The AUC results reinforce the established fact that deeper networks (such as VGG and ResNet) provide

better classification results. The result also indicates that despite AlexNet being comparatively shallower network, it provides comparable results in detecting cracks of concrete structures. Table 6 lists the best operating threshold ( $\tau$ ) that were found from the ROC analysis. In addition, Table 6 also lists the performance metrics (sensitivity, specificity and accuracy) achieved by the CNN models for the corresponding threshold. The operating threshold listed was chosen such that the CNN model provides the best specificity, followed by sensitivity. In many cases, we see the models are optimized for both.

It can be observed from Table 6 that VGG-16 and VGG-19 both have higher sensitivity (0.95) and specificity (0.95). In addition, VGG-19 has higher precision (0.90) along with ResNet-152, Inception-v3, Inception-v4 and Wide-ResNet-50-2; and VGG-19 achieved the highest accuracy (of 0.96). It is important to notice that Cha et al.<sup>17</sup> obtained about 98% accuracy for detecting concrete cracks using CNN. However, their approach would cause misclassification if the edges were to be present on the edge of the sliding windows and hence they used overlapping windows to detect cracks<sup>17</sup>. In this work, we have used non-overlapping windows (in other words, we use patches), which are not only computationally efficient than overlapping windows, but also our approach was able to detect cracks that were present on the edges of patches. Moreover, Feng et al.<sup>31</sup> used ResNet model and achieved about 87.5% accuracy. However, the results show that we could reach over 95% accuracy with ResNet model from Table 6. Another study conducted by Jang et al.<sup>33</sup> used GoogLeNet model and reported the precision value of 59.84%, which is far below when compared to our patch-based approach as can be seen in Table 6 – our approach achieves higher precision for the two GoogLeNet models (0.90 for Inception-v3 and 0.89 for Inception-v4).

From Figure 6, we also note that our proposed approach is robust to different concrete texture surfaces. The results of our approach remain unchanged even if the concrete surface texture have varied textures and complex cracks because of the inherent nature of the layered approaches of CNN wherein the lower layers capture edges, orientations, and higher layers capture shapes and texture. Our training examples include a variety of concrete images with different texture and hence the CNN models automatically learn these features over training time. Furthermore, our approach is invariant to translation and rotation of input images as observed during testing. These key observations of

<sup>‡</sup>[https://drive.google.com/open?id=1m3AE\\_sghjSb7SkzygVKHUqzABpUtr7m](https://drive.google.com/open?id=1m3AE_sghjSb7SkzygVKHUqzABpUtr7m)

our approach are primarily attributed to the fundamental properties of CNN. More detailed invariant properties of the CNN, such as translation and rotation are available here<sup>66</sup>. Our proposed approach is also invariant to scale and texture to an extent as our models are trained by randomly cropping and flipping the patches from input images during training<sup>67</sup>. However, if the scale of the image is such that, for example, a crack in a patch covers almost 80% of the patch, then the models may struggle to classify the patch correctly. We recommend using image scales such that the cracks in each patch do not exceed over 60-70% of the patch size. To achieve this, one can create a larger patch size (greater than 64 x 64) and then resize the patch when feeding to the CNN models, ensuring the scale of the cracks are accounted for. However, one can also look at scale invariant CNNs, such as the RetinaNet<sup>68</sup>, where pyramidal CNN architecture considers different image scales. The RetinaNet<sup>68</sup> was designed primarily towards handling multiple scales in general object detection scenarios. Nevertheless, our experience shows that cracks generally do not suffer from scaling issues; however, these can be solved by including pyramidal CNN architecture.

## 5.2 Inference time

Inference time is also an important factor to be considered in addition to the model accuracy and number of parameters when choosing the models for real-time applications. For example, Kim et al.<sup>69</sup> reported that automated crack-detection using UAVs took 1.6 seconds to detect cracks in an image of concrete structures. Figure 8 shows the comparison of inference time and ROC AUC for all the 15 models. To ascertain the performance of our approach with regard to detection accuracy and inference time, we compare our results with those presented in Chen and Jahanshahi<sup>34</sup> using a CNN model and a Naive Bayes data fusion scheme (NB-CNN). NB-CNN approach had a sensitivity of 98.3%, while the AUC was around 96%<sup>34</sup> and took about 2.55 seconds to detect cracks on a 750 x 540 image. From Table 6, we see that our approach provides a comparable sensitivity of 95-96% while also providing 97-98% AUC. Also, our AlexNet-based approach (which requires 0.0205 seconds to process a patch - see Inference time Figure 8) is 1.3x faster (takes 0.328 seconds for 256 x 256 and accordingly 0.328 x 6 = 1.96 seconds to process 750 x 540) than NB-CNN<sup>34</sup>.

Table 7 shows the comparison of 15 CNN models investigated in this paper in terms of model size (*i.e.*, numbers of parameters), AUC and inference time. It is evident that although AlexNet was third in AUC (from

highest to lowest), it requires the least inference time (about 20 ms) per patch. This is followed by DenseNet-121 and ResNet-50 with 86ms and 90ms, respectively. VGG-16, which had the highest AUC, comes at 10<sup>th</sup> place in inference time (0.22 s per patch). These results indicate that as the networks get deeper, the inference time also increases. So, we have to choose models that fit the applications in hand not only based on accuracy, but also considering the inference time. If the inference time is too high, then those models may not be suited for real-time crack detection of concrete structures, but they can be used for offline assessments. Real-time crack detection for infrastructure inspection require models to be light-weight as the devices used will usually have limited processing time, such as the Edge Computing devices.

## 6 Conclusion

Detecting cracks on concrete surfaces is crucial for the inspection and management of infrastructure as they indicate the possibility of underlying structural damage due to defect or aging, and may affect the safety, durability and serviceability of infrastructure. Automatic crack detection models are imperative to address the drawbacks of manual inspections such as labor and time-intensity, high cost and safety. Existing methods that use window-based scanning use larger window size, resulting coarsely identified patches of cracks, with less precise localization of cracks in smaller images. In addition, window-based methods use multiple scans (at least 50% overlap in patches) to detect the cracks that appear on the edges of the scanning windows. In this paper, we presented an automated crack detection approach using Convolutional Neural Networks (CNN) and non-overlapping windows to detect crack/non-crack conditions of concrete structures from images. We constructed a publicly available dataset of crack/non-crack concrete images, consisting of 32,704 training patches, 2,074 validation patches and 6,032 test patches. We evaluated the performance of our approach using 15 state-of-the-art CNN models. The performance of these models were evaluated based on the number of parameters required to train the models, ROC AUC metric and inference time. The results showed that our approach outperformed existing models in literature for both accuracy and efficiency (*i.e.*, inference time). Our evaluation also shows that deeper models have higher detection accuracies, however, they also require more parameters and have higher inference time. Therefore, for real-time applications, one has to choose models which provide a balance between accuracy and inference time.

Our proposed approach is suitable for SHM applications involving drones for real-time inspections.

## Acknowledgments

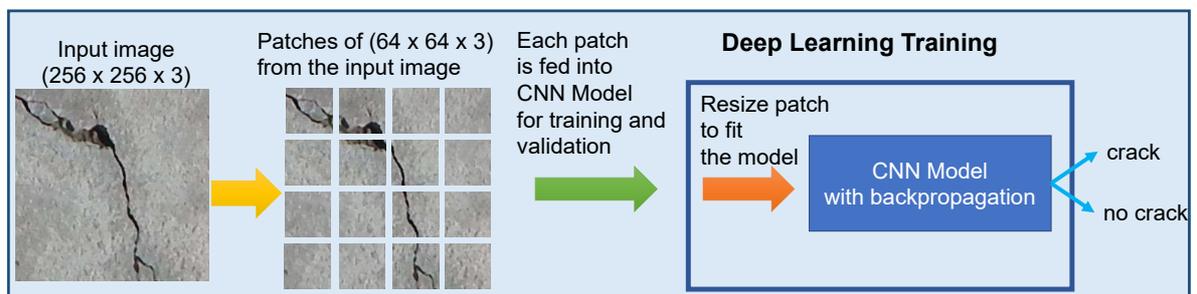
This work was supported by the CRC-P for Advanced Manufacturing of High Performance Building Envelope project, funded by the CRC-P program of the Department of Industry, Innovation and Science, Australia, and the Asia Pacific Research Network for Resilient and Affordable Housing (APRAH) grant, funded by the Australian Academy of Science, Australia.

## References

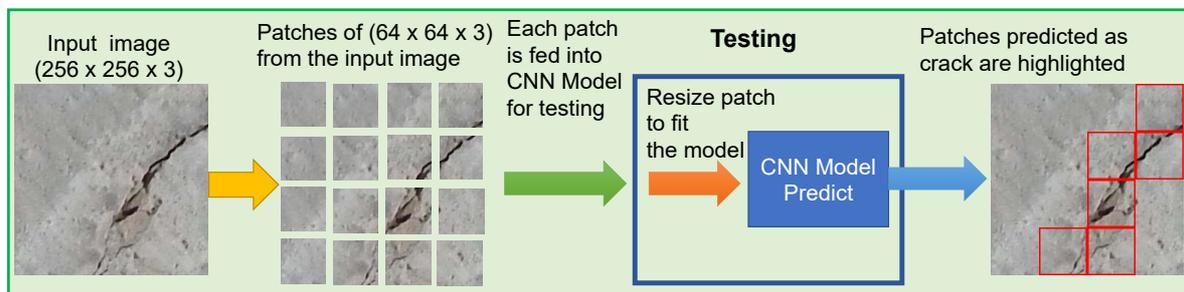
1. VicRoads. Road structures inspection manual, 2018.
2. Kim H, Ahn E, Cho S et al. Comparative analysis of image binarization methods for crack identification in concrete structures. *Cement and Concrete Research* 2017; 99: 53–61.
3. Lynch JP, Farrar CR and Michaels JE. Structural health monitoring: technological advances to practical implementations. *Proceedings of the IEEE* 2016; 104(8): 1508–1512.
4. Peter C, Alison F and Liu S. Review paper: health monitoring of civil infrastructure. *Structural health monitoring* 2003; 2(3): 0257–267.
5. Rakha T and Gorodetsky A. Review of unmanned aerial system (uas) applications in the built environment: Towards automated building inspection procedures using drones. *Automation in Construction* 2018; 93: 252–264.
6. Dorafshan S and Maguire M. Bridge inspection: human performance, unmanned aerial systems and automation. *Journal of Civil Structural Health Monitoring* 2018; 8(3): 443–476.
7. Shi P, Fan X, Ni J et al. A detection and classification approach for underwater dam cracks. *Structural Health Monitoring* 2016; 15(5): 541–554.
8. Oh JK, Jang G, Oh S et al. Bridge inspection robot system with machine vision. *Automation in Construction* 2009; 18(7): 929–941.
9. Gavilán M, Balcones D, Marcos O et al. Adaptive road crack detection system by pavement classification. *Sensors* 2011; 11(10): 9628–9657.
10. Yu SN, Jang JH and Han CS. Auto inspection system using a mobile robot for detecting concrete cracks in a tunnel. *Automation in Construction* 2007; 16(3): 255–261.
11. Prasanna P, Dana KJ, Gucunski N et al. Automated crack detection on concrete bridges. *IEEE Transactions on automation science and engineering* 2014; 13(2): 591–599.
12. Rao AS, Gubbi J, Palaniswami M et al. A vision-based system to detect potholes and uneven surfaces for assisting blind people. In *2016 IEEE International Conference on Communications (ICC)*. IEEE, pp. 1–6.
13. Broberg P. Surface crack detection in welds using thermography. *NDT & E International* 2013; 57: 69–73.
14. LeCun Y, Bengio Y and Hinton G. Deep learning. *nature* 2015; 521(7553): 436.
15. Rao AS, Gubbi J, Marusic S et al. Crowd event detection on optical flow manifolds. *IEEE transactions on cybernetics* 2015; 46(7): 1524–1537.
16. Mneymneh BE, Abbas M and Khoury H. Evaluation of computer vision techniques for automated hardhat detection in indoor construction safety applications. *Frontiers of Engineering Management* 2018; 5(2): 227–239.
17. Cha YJ, Choi W and Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering* 2017; 32(5): 361–378.
18. Kim H, Ahn E, Shin M et al. Crack and noncrack classification from concrete surface images using machine learning. *Structural Health Monitoring* 2019; 18(3): 725–738.
19. Tsao S, Kehtarnavaz N, Chan P et al. Image-based expert-system approach to distress detection on crc pavement. *Journal of Transportation Engineering* 1994; 120(1): 52–64.
20. Abdel-Qader I, Abudayyeh O and Kelly ME. Analysis of edge-detection techniques for crack identification in bridges. *Journal of Computing in Civil Engineering* 2003; 17(4): 255–263. DOI: doi:10.1061/(ASCE)0887-3801(2003)17:4(255).
21. Yamaguchi T and Hashimoto S. Improved percolation-based method for crack detection in concrete surface images. In *2008 19th International Conference on Pattern Recognition*. ISBN 1051-4651, pp. 1–4. DOI:10.1109/ICPR.2008.4761627.
22. Zhong X, Peng X, Yan S et al. Assessment of the feasibility of detecting concrete cracks in images acquired by unmanned aerial vehicles. *Automation in Construction* 2018; 89: 49–57. DOI:https://doi.org/10.1016/j.autcon.2018.01.005.
23. Shilton A, Palaniswami M, Ralph D et al. Incremental training of support vector machines. *IEEE transactions on neural networks* 2005; 16(1): 114–131.
24. Moselhi O and Shehab-Eldeen T. Automated detection of surface defects in water and sewer pipes. *Automation in Construction* 1999; 8(5): 581–588.
25. Hu H, Gu Q and Zhou J. Htf: a novel feature for general crack detection. In *2010 IEEE International Conference on Image Processing*. ISBN 2381-8549, pp. 1633–1636. DOI: 10.1109/ICIP.2010.5653171.
26. Chen Z, Derakhshani R, Halmen C et al. A texture-based method for classifying cracked concrete surfaces from digital images using neural networks. In *The 2011 International Joint Conference on Neural Networks*. IEEE. ISBN 1424496373, pp.

- 2632–2637.
27. Ruan C and Ruan Q. An effective feature for crack detection on train wheel surface. In *2012 IEEE 11th International Conference on Signal Processing*, volume 2. ISBN 2164-523X, pp. 865–868. DOI:10.1109/ICoSP.2012.6491717.
28. Li G, He S, Ju Y et al. Long-distance precision inspection method for bridge cracks with image processing. *Automation in Construction* 2014; 41: 83–95.
29. Rimkus A, Podvieszko A and Gribniak V. Processing digital images for crack localization in reinforced concrete members. *Procedia Engineering* 2015; 122: 239–243.
30. Nguyen T, Kashani A, Ngo T et al. Deep neural network with high-order neuron for the prediction of foamed concrete strength. *Computer-Aided Civil and Infrastructure Engineering* 2019; 34(4): 316–332.
31. Feng C, Liu MY, Kao CC et al. Deep active learning for civil infrastructure defect detection and classification. In *Computing in Civil Engineering 2017*. American Society of Civil Engineers, 2017. pp. 298–306. DOI:10.1061/9780784480823.036.
32. Szegedy C, Liu W, Jia Y et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1–9.
33. Jang K, Kim N and An YK. Deep learning-based autonomous concrete crack evaluation through hybrid image scanning. *Structural Health Monitoring* 2018; : 1475921718821719.
34. Chen FC and Jahanshahi MR. Nb-cnn: deep learning-based crack detection using convolutional neural network and naïve bayes data fusion. *IEEE Transactions on Industrial Electronics* 2017; 65(5): 4392–4400.
35. Uijlings JR, Van De Sande KE, Gevers T et al. Selective search for object recognition. *International journal of computer vision* 2013; 104(2): 154–171.
36. Girshick R, Donahue J, Darrell T et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 580–587.
37. Girshick R. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. pp. 1440–1448.
38. Ren S, He K, Girshick R et al. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems*. pp. 91–99.
39. Cha Y, Choi W, Suh G et al. Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Computer-Aided Civil and Infrastructure Engineering* 2018; 33(9): 731–747.
40. Beckman GH, Polyzois D and Cha YJ. Deep learning-based automatic volumetric damage quantification using depth camera. *Automation in Construction* 2019; 99: 114–124. DOI: <https://doi.org/10.1016/j.autcon.2018.12.006>.
41. Huyan J, Li W, Tighe S et al. Detection of sealed and unsealed cracks with complex backgrounds using deep convolutional neural network. *Automation in Construction* 2019; 107: 102946. DOI:<https://doi.org/10.1016/j.autcon.2019.102946>.
42. Krizhevsky A, Sutskever I and Hinton GE. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. pp. 1097–1105.
43. Dorafshan S, Thomas RJ and Maguire M. Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete. *Construction and Building Materials* 2018; 186: 1031–1045. DOI:<https://doi.org/10.1016/j.conbuildmat.2018.08.011>.
44. Zhang A, Wang KC, Li B et al. Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network. *Computer-Aided Civil and Infrastructure Engineering* 2017; 32(10): 805–819.
45. Marques AGCS. *Automatic road pavement crack detection using SVM*. Master of science degree dissertation, Electrical and Computer Engineering, Instituto Superior Técnico, Lisbon, Portugal, 2012.
46. Long J, Shelhamer E and Darrell T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 3431–3440.
47. Dung CV and Anh LD. Autonomous concrete crack detection using deep fully convolutional neural network. *Automation in Construction* 2019; 99: 52–58. DOI:<https://doi.org/10.1016/j.autcon.2018.11.028>.
48. Li S, Zhao X and Zhou G. Automatic pixel-level multiple damage detection of concrete structure using fully convolutional network. *Computer-Aided Civil and Infrastructure Engineering* 2019; 34(7): 616–634. DOI:10.1111/mice.12433.
49. Yang X, Li H, Yu Y et al. Automatic pixel-level crack detection and measurement using fully convolutional network. *Computer-Aided Civil and Infrastructure Engineering* 2018; 33(12): 1090–1109.
50. Kim B and Cho S. Image-based concrete crack assessment using mask and region-based convolutional neural network. *Structural Control and Health Monitoring* 2019; 26(8): e2381. DOI:10.1002/stc.2381.
51. Liu Y, Yao J, Lu X et al. Deepcrack: A deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing* 2019; 338: 139–153. DOI:<https://doi.org/10.1016/j.neucom.2019.01.036>.

52. Lee CY, Xie S, Gallagher P et al. Deeply-supervised nets. In *Artificial intelligence and statistics*. pp. 562–570.
53. Liu Z, Cao Y, Wang Y et al. Computer vision-based concrete crack detection using u-net fully convolutional networks. *Automation in Construction* 2019; 104: 129–139. DOI:<https://doi.org/10.1016/j.autcon.2019.04.005>.
54. Simonyan K and Zisserman A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*. pp. 1–10.
55. He K, Zhang X, Ren S et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778.
56. Szegedy C, Vanhoucke V, Ioffe S et al. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 2818–2826.
57. Szegedy C, Ioffe S, Vanhoucke V et al. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*. pp. 4278–4284.
58. Huang G, Liu Z, Van Der Maaten L et al. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 4700–4708.
59. Xie S, Girshick R, Dollár P et al. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 1492–1500.
60. Zagoruyko S and Komodakis N. Wide residual networks. In Richard C Wilson ERH and Smith WAP (eds.) *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press. ISBN 1-901725-59-6, pp. 87.1–87.12. DOI:10.5244/C.30.87. URL <https://dx.doi.org/10.5244/C.30.87>.
61. Belkin M, Hsu D, Ma S et al. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* 2019; 116(32): 15849–15854.
62. Bengio Y. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 2012. pp. 437–478.
63. Sutskever I. *Training recurrent neural networks*. PhD Thesis, University of Toronto Toronto, Ontario, Canada, 2013.
64. Murphy KP. *Machine learning: a probabilistic perspective*. MIT press, 2012.
65. Fawcett T. An introduction to roc analysis. *Pattern recognition letters* 2006; 27(8): 861–874.
66. LeCun Y, Bengio Y et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 1995; 3361(10): 1995.
67. Marcos D, Volpi M and Tuia D. Learning rotation invariant convolutional filters for texture classification. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 2012–2017.
68. Lin TY, Goyal P, Girshick R et al. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. pp. 2980–2988.
69. Kim B and Cho S. Automated vision-based detection of cracks on concrete surfaces using a deep learning technique. *Sensors* 2018; 18(10): 3452.

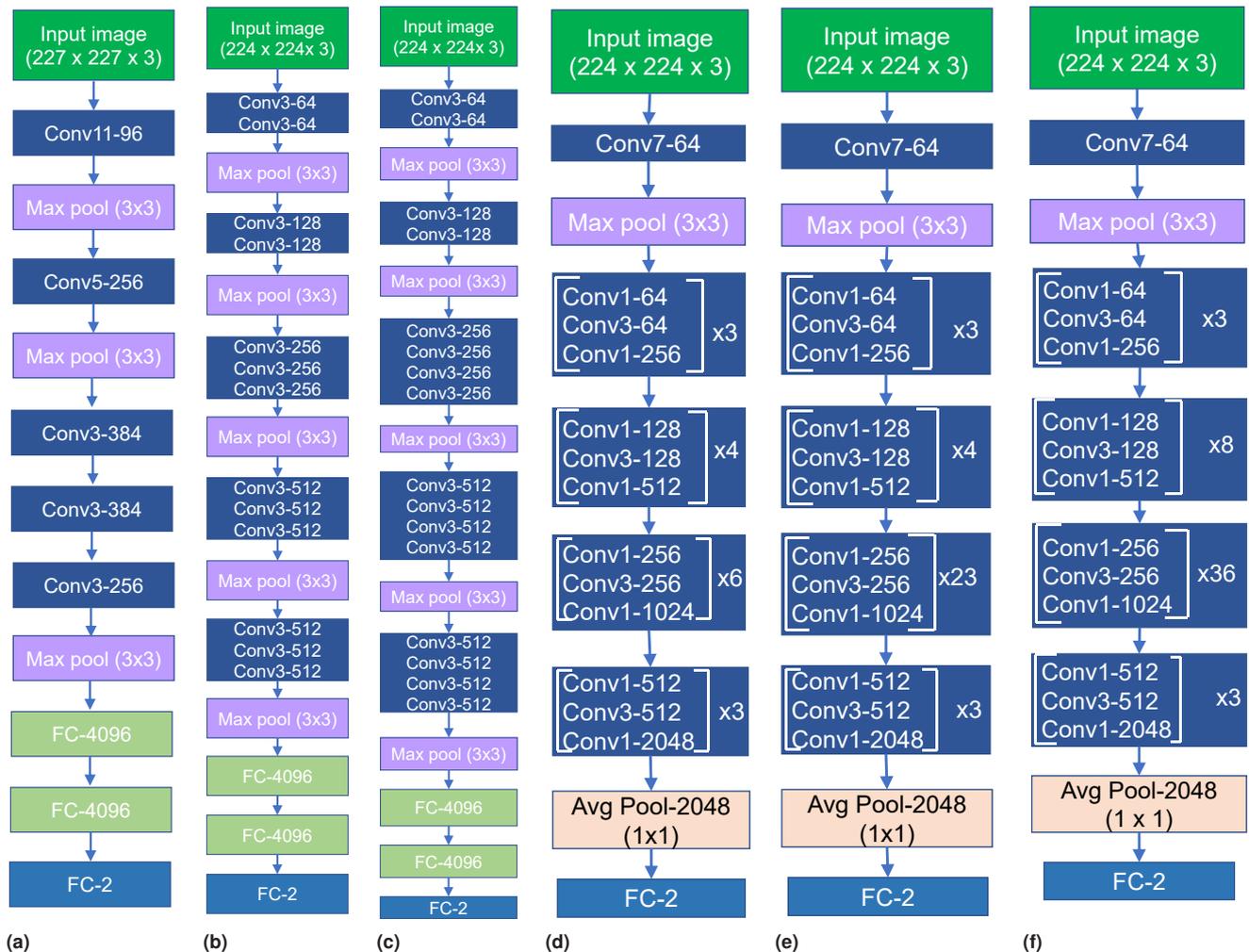


(a) Training phase

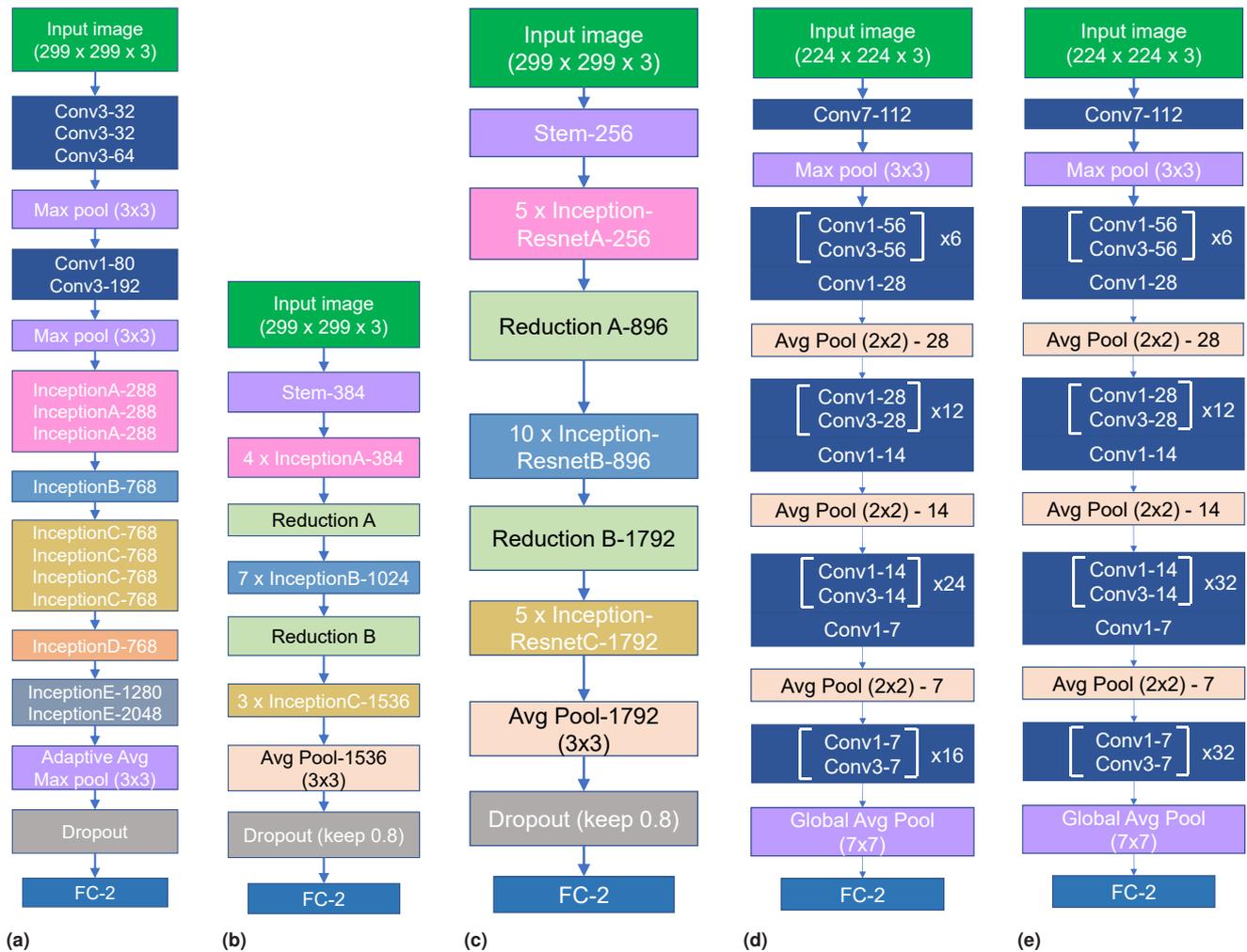


(b) Testing phase

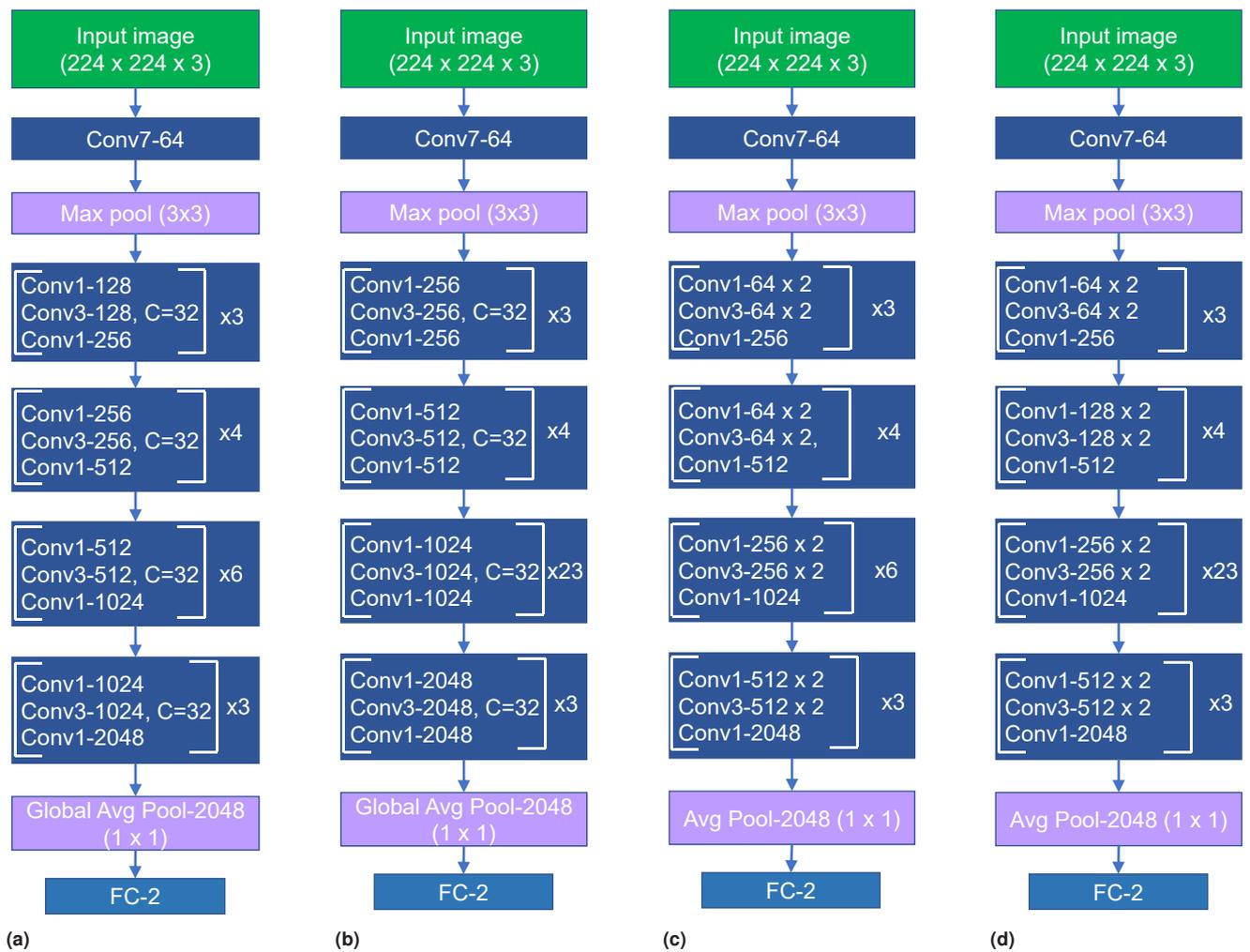
**Figure 1.** Overview of the crack detection approach used in this work.



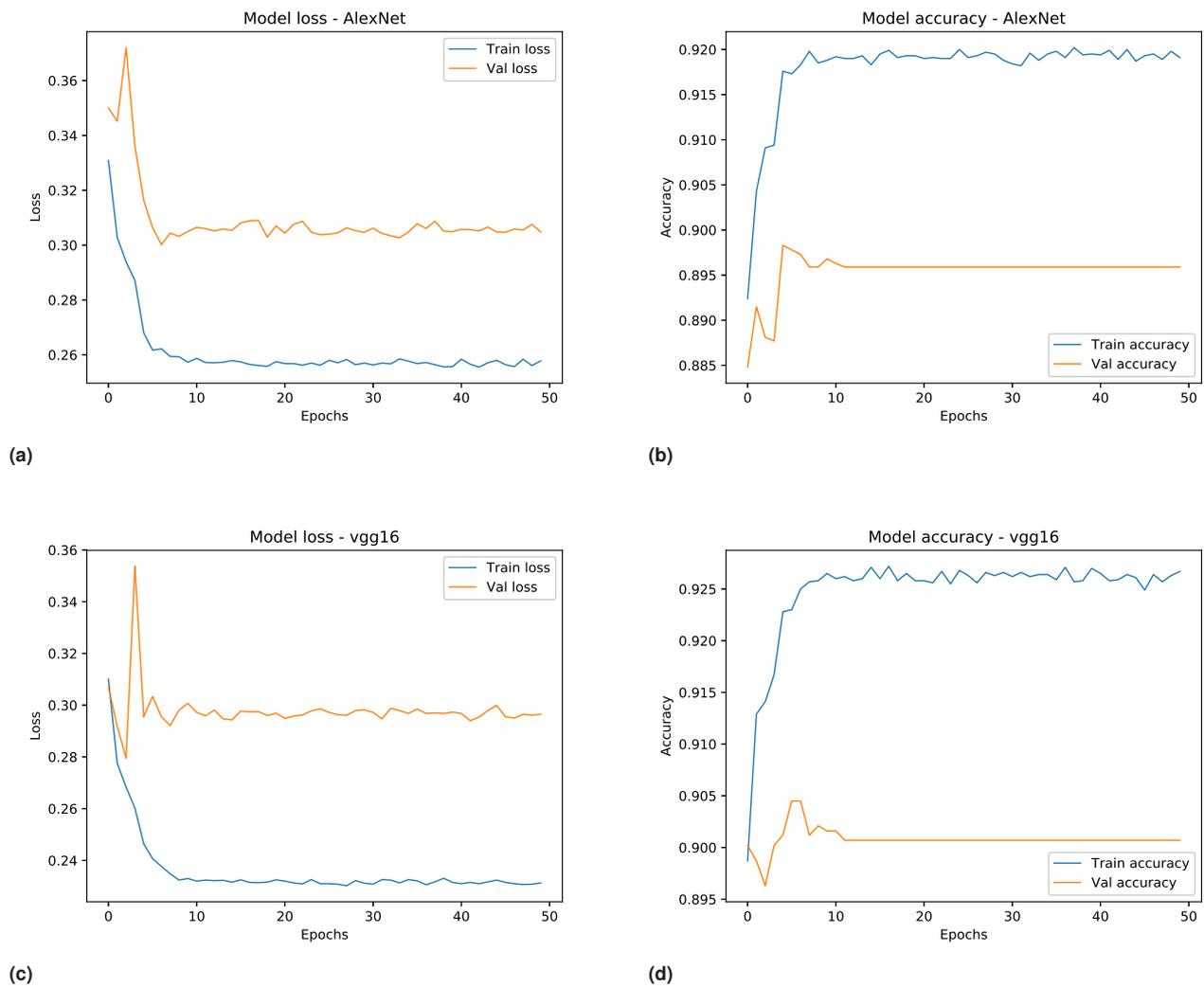
**Figure 2.** CNN architectures of (a) AlexNet, (b) VGG-16, (c) VGG-19, (d) ResNet-50, (e) ResNet-101 and (f) ResNet-152. In this work, we represent convolution layers as “Conv(receptive field size)–number of channels”. For example, Conv5-256 in (a) represents a convolution layer with a receptive field size of 5 x 5 and 256 channels. FC: Fully Connected layer. FC-2 is abbreviation for fully connected layer with two outputs. Max pool (3 x 3) is a short notation for max pooling operation with 3 x 3 kernel. Avg Pool: average pooling. Similar analogies apply to other operations. Readers are recommended to read the original papers for more detailed information about each layer.



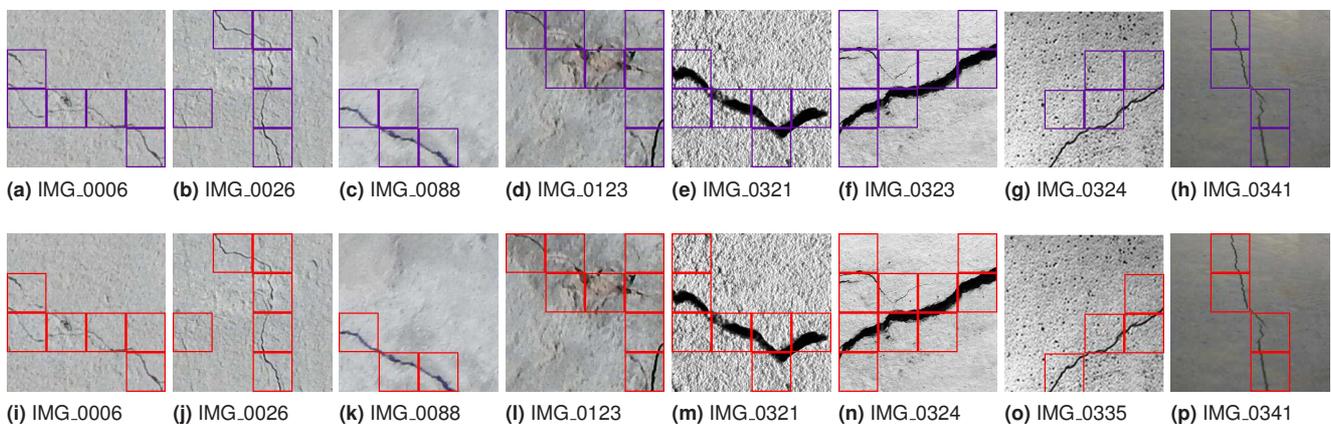
**Figure 3.** CNN architectures of (a) Inception-v3, (b) Inception-v4, (c) Inception-ResNet-v2, (d) DenseNet-121 and (e) DenseNet-169. The “stem” block in the (b) and (c) describes the early stage convolution, pooling and normalization operations. FC: Fully Connected layer. FC-2 is abbreviation for fully connected layer with two outputs. Max pool (3 x 3) is a short notation for max pooling operation with 3 x 3 kernel. Avg Pool: average pooling. Similar analogies apply to other operations. Readers are recommended to read the original papers for more detailed information about each layer.



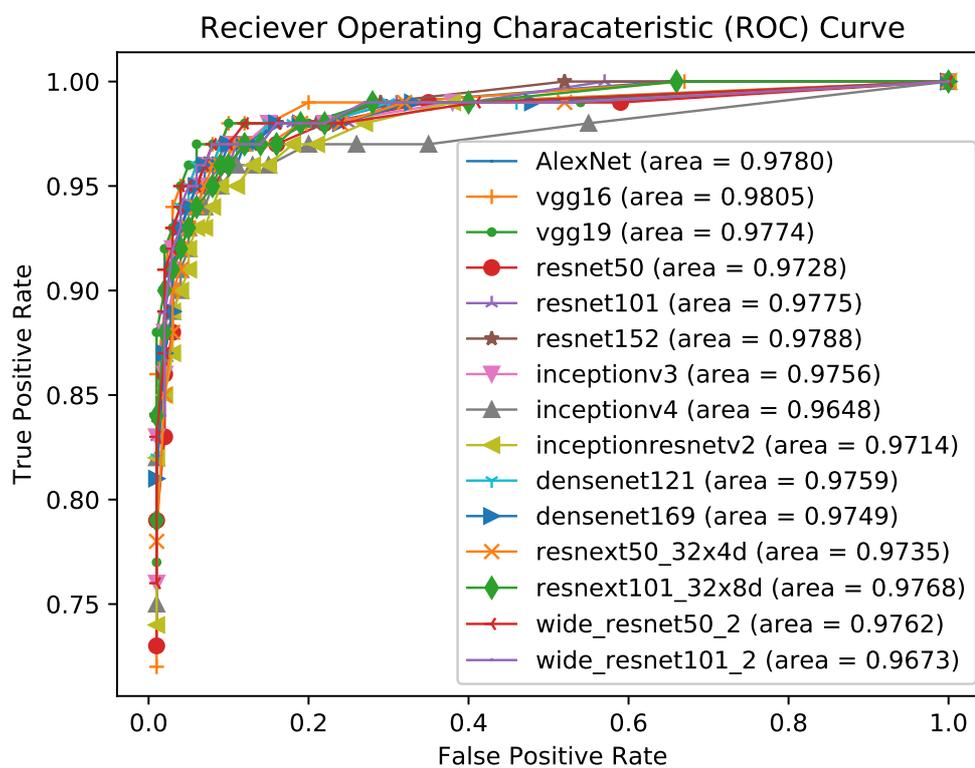
**Figure 4.** (a) ResNeXt-50-32x4d, (b) ResNeXt-101-32x8d, (c) Wide-ResNet-50-2 and (d) Wide-ResNet-101-2. In the ResNeXt-50 (32x4d), 50 indicates the network with a depth of 50 layers, 32 represents the cardinality of transformations and 4d represents the bottleneck width, which is part of ResNet. Wide-ResNet-50-2 represents a wide ResNet with 50 layers deep and 2x with of the original ResNet-50 architecture (similar naming convention applies to Wide-ResNet-101-2). FC: Fully Connected layer. FC-2 is abbreviation for fully connected layer with two outputs. Max pool (3 x 3) is a short notation for max pooling operation with 3 x 3 kernel. Avg Pool: average pooling. Similar analogies apply to other operations. Readers are recommended to read the original papers for more detailed information about each layer.



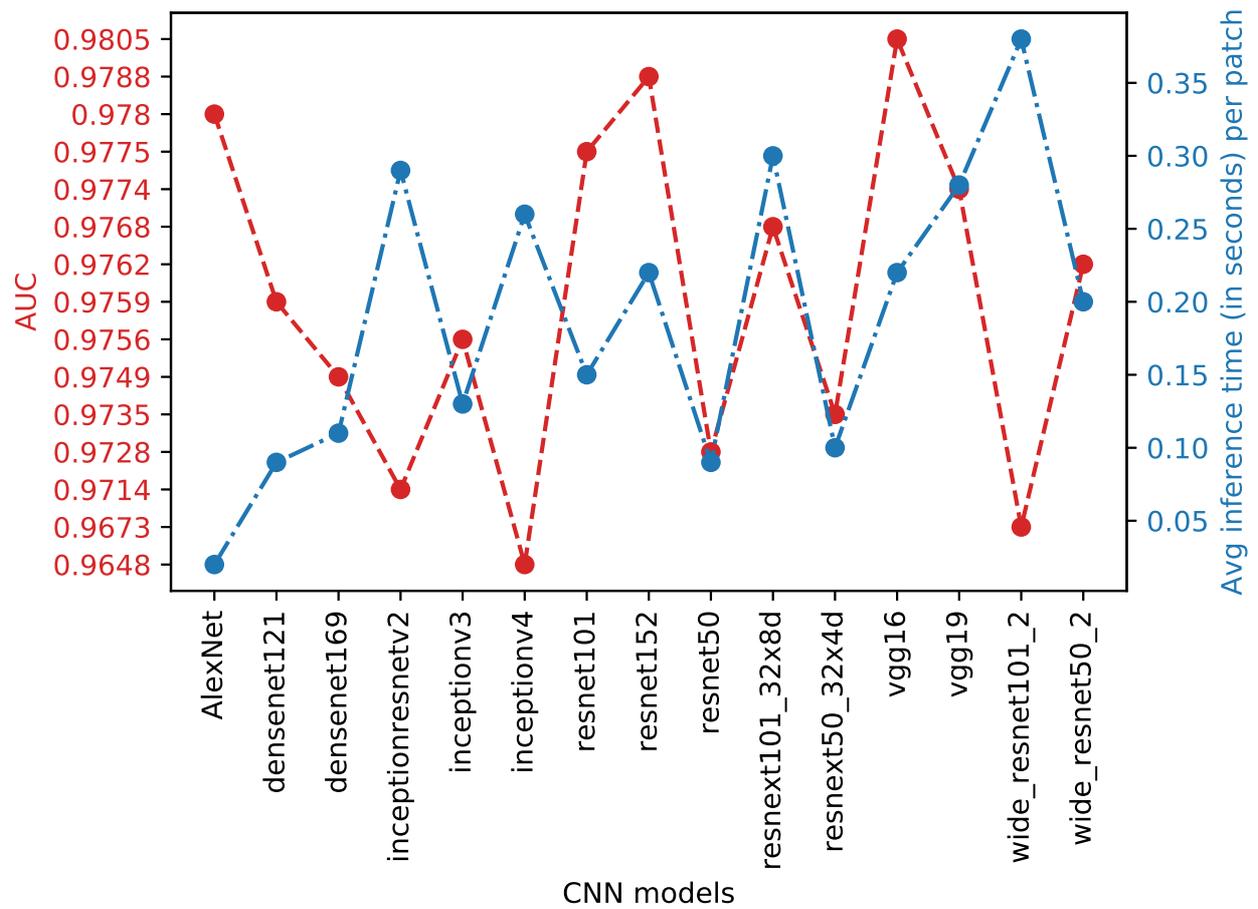
**Figure 5.** Shows (a) training and validation loss of AlexNet training for detecting cracks; (b) accuracy of AlexNet during training and validation; (c) training and validation loss of VGG-16 training; (d) accuracy of VGG-16 during training and validation.



**Figure 6.** Shows the sample results of VGG-16: (a)–(h) represent the manually annotated ground truths (marked in purple color), and (i)–(p) show the corresponding predicted output (marked in red color) of VGG-16 model with a prediction threshold ( $\tau$ ) of 0.6.



**Figure 7.** Comparison of ROC AUC curves for the 15 CNN models. The models were tested on 377 test images, where each image was of the size 256 x 256 (RGB). Each image was divided into 16 64 x 64 patches (total 16 patches per image). The ROC AUC results were generated based on the number of patches that matched the ground truth from all the test images.



**Figure 8.** Shows the comparison of inference time (per patch) and AUC for all the 15 models.

**Table 1.** Summary of image processing and region-based classification methods applied for crack detection along with their advantages and disadvantages.

<b>Image Processing Methods</b>		
<b>Approach</b>	<b>Advantages</b>	<b>Disadvantages</b>
Histogram of pixels <sup>19</sup>	Adapts simple rule-based hierarchical image processing system to detect concrete distress from subimages	One needs to know the rules and hierarchy. Requires rule update if input domain is different.
Edge detection <sup>20</sup>	Automatically localize edges for detecting cracks in concrete structures	Uses image heuristics and lacks robustness to surface changes.
Region Growing <sup>21</sup>	Scalable local crack detection based on neighborhood connectivity	Need to specify the window width and not robust to illumination changes.
Edge and Region Filling <sup>10</sup>	Laplacian of Gaussian 2D filter in detecting edges makes it rotationally invariant to cracks	It is semi-autonomous, not fully robust to small cracks.
Hough Transform <sup>25</sup>	Uses simple and effective Hough Transform for detecting cracks	Assumes cracks are not complex and mostly form straight lines.
Grey Level Coefficient Matrix <sup>26</sup>	Uses textural features (invariant to illumination) to detect cracks	May not work if surface texture is uniformly spread across different regions.
Laplacian of Gaussian (LoG) weighted Haar <sup>27</sup>	Robust feature capturing edges, orientations and textures of cracks	Highly sensitive to small cracks and becomes complex with larger image sizes.
The Chan-Vese (C–V) model and Wavelet Filters <sup>28</sup>	Robust to local image noises while detecting cracks	Detection relies on differentiable crack and intact texture.
Image Correction with Hierarchical Clustering <sup>29</sup>	Provides hierarchical grouping of clusters based on distance for localization	Assumes cracks are aligned in a single direction and may fail in the case of complex cracks.
Spatially tuned multifeature (STRUM) <sup>11</sup>	Robust to spatial noise and invariant to image scale	Highly dependent on the intensity of pixels in crack and non-crack regions.

<b>Region-based Classification Methods</b>		
<b>Approach</b>	<b>Advantages</b>	<b>Disadvantages</b>
Image Binarization and Localization <sup>18</sup>	Two-stage approach to detect cracks. First proposes candidate crack regions (CCR) and then localizes the cracks using CNN	CCR binarization uses pixel statistics assuming certain properties of cracks. This could fail if these properties change because of varying pixel intensities.
CNN-based crack detection <sup>17</sup>	Proposes CNN framework that learns crack/non-crack regions automatically	Can misclassify the crack that are image edges and requires twice scanning of images to detect cracks during testing, making this computationally expensive at inference.
ResNet with Active Learning <sup>31</sup>	Automatically learns to train from less number of labeled samples where annotated dataset is a problem	Requires re-training of the deep learning model to accommodate newly identified training samples when available.
GoogLeNet with Hybrid Images <sup>33</sup>	Combines data from RGB camera and infrared camera to detect micro- and micro-cracks	Calibration of imaging systems and alignment requires expert knowledge. Computationally expensive to train networks because of multiple modalities.
CNN with Naïve Bayes Learning <sup>34</sup>	Maintains spatiotemporal coherence of detected cracks in videos	Computationally expensive for real-time applications because of frame registration and motion estimation between frames.

**Table 2.** Summary of object detection and semantic segmentation methods applied for crack detection along with their advantages and disadvantages.

<b>Object Detection Methods</b>		
<b>Approach</b>	<b>Advantages</b>	<b>Disadvantages</b>
Faster R-CNN <sup>39</sup>	Detects five types of damages including concrete cracks in quasi real-time (1.4 frames per second)	Sensitive to light intensities. May not perform well when designed for only two classes (instead of five).
Faster R-CNN with depth camera <sup>40</sup>	Provides volumetric quantification of concrete spalling using depth information	Requires trial-and-error approach to determine anchor points to detect spalling even before training the model.
Faster R-CNN with sensitivity detection network <sup>41</sup>	Improves the detection of cracks by using linear crack filters in addition to localization using CNN	The addition of linear crack filters and their features limit the generalization of automation of crack detection.

<b>Semantic Segmentation Methods</b>		
<b>Approach</b>	<b>Advantages</b>	<b>Disadvantages</b>
Multi-scale with SVM <sup>45</sup>	Robust to different image scales based on crack neighborhood	Need to explicitly construct probability maps of pixel-level cracks.
3D Crack Detection <sup>44</sup>	Provides pixel-level detection of cracks using CNN	Need 3D data for this to be successful.
FCN with VGG-16 <sup>47</sup>	End-to-end training and segmentation of crack pixels	May not perform well when there are multiple, complex cracks.
FCN with DenseNet-121 <sup>48</sup>	Detects different damage types including crack, spalling, efflorescence and hole	Requires large amounts of training data.
FCN with VGG-19 <sup>49</sup>	Detects cracks at pixel-level with good accuracy and speed	Accuracy deteriorates in the case of insufficient training data and variety.
FCN with Deep Supervised Net <sup>51</sup>	Integrates multi-level and multi-scale features with direct supervision of CNN features for improved detection of cracks	Fails to provide continuous and complete thin crack segments.
U-Net for Crack Detection <sup>53</sup>	Employs focal loss metric during training that balances the ratio of crack and non-crack pixels. Robust to illumination, complex background and width of cracks	The depth of the U-Net can vary depending on the dataset and hence the inference time.

**Table 3.** Lists the CNN network architectures used in this work. In addition, we list the input image size for each network and the number of parameters required to train each network for detecting cracks. Note: the final layer of the networks are modified to output two probabilities corresponding to two classes ("crack" or "no crack"). In the parameters column, we see that the models require millions of parameters to be tuned to detect cracks.

<b>Model</b>	<b>Input size</b>	<b>No. of parameters</b>
AlexNet	224 x 224	57,012,034
VGG-16	224 x 224	134,268,738
VGG-19	224 x 224	139,578,434
ResNet-50	224 x 224	23,512,130
ResNet-101	224 x 224	58,147,906
ResNet-152	224 x 224	235,121,30
Inception-v3	299 x 299	25,116,362
Inception-v4	299 x 299	41,145,890
Inception-ResNet-v2	299 x 299	54,309,538
DenseNet-121	224 x 224	6,955,906
DenseNet-169	224 x 224	12,487,810
ResNeXt-50-32x4d	224 x 224	22,984,002
ResNeXt-101-32x8d	224 x 224	86,746,434
Wide-ResNet-50-2	224 x 224	66,838,338
Wide-ResNet-101-2	224 x 224	124,841,794

**Table 4.** Details of dataset used for training, validation and testing of deep learning models. Table shows the number of images (of size 256 x 256) and the corresponding number of patches (of size 64 x 64) used for training, validation and testing. Crack and non-crack are the number of patches of size 64 x 64.

	Images	no crack (patches)	crack (patches)	Total (patches)
		(class 0)	(class 1)	
Train	2,044	23,797	8,907	32,704
Validation	129	1,032	1,032	2,074
Test	377	4,358	1,674	6,032

**Table 5.** Confusion matrix for a binary classifier. From this matrix, we derive True Positive Rate, False Positive Rate, Specificity and Accuracy for measuring the performance of CNN models in identifying the cracks and classifying each patch.

Predicted Class	True Class	
	True Positives (TP)	False Positives (FP)
	False Negatives (FN)	True Negatives (TN)

**Table 6.** Shows the performance comparison of each CNN model in terms of performance metrics: classifying threshold, sensitivity (= TPR = recall), specificity, and accuracy. Bolded text in each column indicate the best performance measured that metric.

Model	Threshold	Sensitivity	Specificity	Accuracy	Precision
AlexNet	0.55	0.94	0.95	0.94	0.88
VGG-16	0.60	<b>0.95</b>	<b>0.96</b>	0.95	0.89
VGG-19	0.60	<b>0.95</b>	<b>0.96</b>	<b>0.96</b>	<b>0.90</b>
ResNet-50	0.55	0.93	0.95	0.95	0.89
ResNet-101	0.75	0.94	0.95	0.95	0.88
ResNet-152	0.70	0.92	<b>0.96</b>	0.95	<b>0.90</b>
Inception-v3	0.60	0.93	<b>0.96</b>	0.95	<b>0.90</b>
Inception-v4	0.65	0.92	0.95	0.95	0.89
Inception-ResNet-v2	0.60	0.92	0.95	0.94	0.87
DenseNet-121	0.65	0.94	0.95	0.95	0.89
DenseNet-169	0.65	0.94	0.95	0.95	0.88
ResNeXt-50-32x4d	0.60	0.93	0.95	0.95	0.88
ResNeXt-101-32x8d	0.65	0.93	0.95	0.94	0.87
Wide-ResNet-50-2	0.55	0.94	<b>0.96</b>	0.95	<b>0.90</b>
Wide-ResNet-101-2	0.70	<b>0.95</b>	0.95	0.95	0.89

**Table 7.** Shows the performance comparison of each CNN model in terms of number of parameters, AUC and inference time in seconds (per patch). Bolded text in columns indicate the best performers in each aspect.

No.	Model	Parameters (millions)	AUC	Inference time (in seconds)
1	AlexNet	57 M	0.9780	<b>0.0205</b>
2	VGG-16	134.2 M	<b>0.9805</b>	0.2229
3	VGG-19	139.5 M	0.9774	0.2782
4	ResNet-50	23.5 M	0.9728	0.0902
5	ResNet-101	58.1 M	0.9775	0.1534
6	ResNet-152	235.1 M	0.9788	0.2202
7	Inception-v3	25.1 M	0.9756	0.1278
8	Inception-v4	41.1 M	0.9648	0.2563
9	Inception-ResNet-v2	54.3 M	0.9714	0.2861
10	DenseNet-121	<b>6.9 M</b>	0.9759	0.0863
11	DenseNet-169	12.4 M	0.9749	0.1073
12	ResNeXt-50-32x4d	22.9 M	0.9735	0.1043
13	ResNeXt-101-32x8d	86.7 M	0.9768	0.2975
14	Wide-ResNet-50-2	66.8 M	0.9762	0.2020
15	Wide-ResNet-101-2	124.8 M	0.9673	0.3750