applied sciences

MDPI

*Article*

# InstaDam: Open-Source Platform for Rapid Semantic Segmentation of Structural Damage

**Vedhus Hoskere [1,*], Fouad Amer [2], Doug Friedel [3,4], Wanxian Yang [5], Yu Tang [5], Yasutaka Narazaki [2], Matthew D. Smith [6], Mani Golparvar-Fard [2,5] and Billie F. Spencer, Jr. [2]**

1 Department of Civil and Environmental Engineering, University of Houston, Houston, TX 77204, USA
2 Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA; famer2@illinois.edu (F.A.); narazak2@illinois.edu (Y.N.); mgolpar@illinois.edu (M.G.-F.); bfs@illinois.edu (B.F.S.J.)
3 National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA; friedel@illinois.edu
4 Department of Astronomy, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
5 Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA; wyang52@illinois.edu (W.Y.); yutang2@illinois.edu (Y.T.)
6 US Army Corps of Engineers, Engineering Research and Development Center, Vicksburg, MS 39180, USA; Matthew.D.Smith@erdc.dren.mil
* Correspondence: vhoskere@uh.edu

**Abstract:** The tremendous success of automated methods for the detection of damage in images of civil infrastructure has been fueled by exponential advances in deep learning over the past decade. In particular, many efforts have taken place in academia and more recently in industry that demonstrate the success of supervised deep learning methods for semantic segmentation of damage (i.e., the pixel-wise identification of damage in images). However, in graduating from the detection of damage to applications such as inspection automation, efforts have been limited by the lack of large open datasets of real-world images with annotations for multiple types of damage, and other related information such as material and component types. Such datasets for structural inspections are difficult to develop because annotating the complex and amorphous shapes taken by damage patterns remains a tedious task (requiring too many clicks and careful selection of points), even with state-of-the art annotation software. In this work, InstaDam—an open source software platform for fast pixel-wise annotation of damage—is presented. By utilizing binary masks to aid user input, InstaDam greatly speeds up the annotation process and improves the consistency of annotations. The masks are generated by applying established image processing techniques (IPTs) to the images being annotated. Several different tunable IPTs are implemented to allow for rapid annotation of a wide variety of damage types. The paper first describes details of InstaDam's software architecture and presents some of its key features. Then, the benefits of InstaDam are explored by comparing it to the Image Labeler app in Matlab. Experiments are conducted where two employed student annotators are given the task of annotating damage in a small dataset of images using Matlab, InstaDam without IPTs, and InstaDam. Comparisons are made, quantifying the improvements in annotation speed and annotation consistency across annotators. A description of the statistics of the different IPTs used for different annotated classes is presented. The gains in annotation consistency and efficiency from using InstaDam will facilitate the development of datasets that can help to advance research into automation of visual inspections.

**Keywords:** supervised learning; deep learning; image processing; structural inspections; damage identification; computer vision; software engineering; user interface design; usability testing; software analysis



**Citation:** Hoskere, V.; Amer, F.; Friedel, D.; Yang, W.; Tang, Y.; Narazaki, Y.; Smith, M.D.; Golparvar-Fard, M.; Spencer, B.F., Jr. InstaDam: Open-Source Platform for Rapid Semantic Segmentation of Structural Damage. *Appl. Sci.* **2021**, *11*, 520. https://doi.org/10.3390/app11020520

Received: 9 November 2020
Accepted: 23 December 2020
Published: 7 January 2021



## 1. Introduction

Visual inspections to assess the condition of civil infrastructure are time-consuming, repetitive, and put inspectors at high levels of risk. The automation of visual inspection tasks

can greatly enhance the overall efficacy and efficiency of condition assessment by allowing inspectors to focus on higher level decisions rather than repetitive tasks. Supervised deep learning methods have proven successful in a key step towards this goal of automation, namely the automated identification of multiple types of damage [1–4]. To realize a fully automated system for inspections, however, higher fidelity information needs to be extracted from images, including, for example, the material type, as well as other damage-like features such as cables, shadows, etc. Different approaches have been examined for these inspection tasks, including (i) object detection, where a bounding-box is drawn around the damaged region [5–9], and (ii) semantic segmentation [1,10–13], where each pixel is classified as a certain damage type. While these methods have shown tremendous success, researchers often develop their own private datasets for each new study, making rigorous benchmarking and comparisons of advances in deep learning architectures difficult. For other applications of semantic segmentation, such as self-driving, the availability of large and open datasets such as Cityscapes [14], SYNTHIA [15], KiTTy [16], BDD100K [17], etc. for scene understanding has helped advance research into deep learning methods to a point where they are now being directly applied in the self-driving industry at scale. The availability of similar large open datasets for semantic segmentation of damage in civil infrastructure will facilitate advances towards automation of civil infrastructure inspection.

To annotate datasets for image segmentation, researchers [18–20] often use software such as LabelMe [21], Matlab [22], and Photoshop [23]. More recently, many commercially available and web-based annotation packages for image segmentation have sprung up, such as Labelbox [24], Prodigy [25], Hasty [26], and Kili Technologies [27], among others. These solutions offer a variety of general labeling tools such as dots, lines, polygons, bounding boxes, pen (lasso), and superpixels. Additionally, the available applications support different degrees of user and project management, ranging from locally hosted applications with no notion of user or project management to full-fledged web-based applications that support full team collaboration. Most of these general-purpose annotation software packages have been designed for use in applications such as scene segmentation, where buildings, roads, pedestrians, signs, etc. are the main items of interest. For the annotation of amorphous shapes like cracks or spalling, the methods available in state-of-the-art software require too many clicks from the user, all of which must be extremely precise, making the annotation process very difficult, prone to inaccuracies, and time consuming. A more efficient software platform is thus needed that will enable rapid development of large and complex annotated datasets for damage.

In this paper, we present an open source software platform, termed InstaDam, for rapid annotation of damage for automation of visual inspections. By incorporating binary masks to aid user input for annotation of different damage types, InstaDam helps to increase the efficiency and consistency of annotations. These binary masks are generated by applying image processing techniques (IPT) to the image being annotated. Several IPTs that researchers have identified for automated detection of damage are implemented for use in InstaDam, allowing for annotation of a wide variety of damage types. The parameters of IPTs can be quickly modified to fine tune the mask edges. InstaDam includes a suite of standard annotation tools found in most annotation software. Once the mask is generated, the user can use different input tools such as the brush or the lasso tool to make sub-selections that correspond to the required segments. InstaDam's front end is developed using Qt C++ which provides cross-platform compatibility. The software also incorporates a modular cloud-based data management framework developed using Flask. The novelty of the research is in the integration of the components into a seamless framework that is open source and is available to the research community for continued development. No other software frameworks are available that address this need.

The paper is organized as follows. Section 2 first describes the problems with current software that slow the annotation process down. Following a brief assessment of the impediments to rapid labeling of images of structural damage, the paper describes details of InstaDam's software architecture and presents some of its key features that help enable

rapid annotation. Section 3 explores the benefits of InstaDam in a quantitative manner by comparing it to the Image Labeler app in Matlab. Improvements are quantified through experiments where two employed student annotators are given the task of annotating damage in a small dataset of 30 images using Matlab, InstaDam without IPTs, and InstaDam. Comparisons are made, quantifying the improvements in annotation speed and annotation consistency across annotators. Finally, a larger dataset of 300 images is annotated for damage and materials by the student annotators using InstaDam and the user actions are recorded. The dataset has been made available for use by researchers. A description of the statistics of the different IPTs used for different annotated classes are presented based on data recorded during the annotation of the large dataset. The gains in annotation consistency and efficiency from using InstaDam will facilitate development of datasets that can help advance research into automation of visual inspections. Section 4 presents the main conclusions, followed by the references.

## 2. InstaDam: Design and Features

This section presents an overview of the design and features of the software developed. First, damage annotation for semantic segmentation is discussed to motivate the design of InstaDam. Then, details about the choice of frameworks for the front end architecture are presented followed by a section on the user interface design. Finally, the cloud-based data management capabilities of the software are discussed.

### *2.1. Damage Annotation for Semantic Segmentation*

Damage patterns occurring on civil infrastructure often take amorphous, non-regular shapes. This property makes annotating these patterns using existing tools exceedingly difficult, as very careful attention is required during the annotation process. While annotating these patterns, researchers often use annotation software such as LabelMe [21], Matlab [22], Labelbox [24], Prodigy [25], Hasty [26], and Kili Technologies [27], to name a few. These software packages have primarily been designed for scene annotation and do not lend themselves well to annotating complex the amorphous shapes taken by damage, some examples of which are presented in Figure 1. For example, some commonly available tools on most of these platforms are the polygonal select tool—where vertices of a polygon are manually selected on the image to define a polygonal region, and the brush tool—where pixels that fall inside the brush are selected. Selecting the boundary pixels for damage shapes with a polygonal tool is not feasible and highly inefficient. While the brush tool is more convenient for selecting damage, for large area damage such as corrosion, the selection of boundaries becomes tedious. For defects such as cracks, varying thickness and shape of cracks create similar problems. For other patterns such as cables, which need to be annotated as they otherwise result in false positive detection, annotating a large number of cables with these tools can make the annotation process time-consuming and inefficient.

### *2.2. Choice of Software Frameworks*

The selection of frameworks and libraries for development of the InstaDam was a critical design decision. An overview of the front end frameworks and libraries used is shown in Figure 2. Qt [28] is an open-source C++ framework for creating applications and was chosen to build the front end layer for InstaDam. Applications developed with Qt are cross-platform and can be built for different operating systems with little or no changes to the main code base. Qt also supports WebAssembly [24], using which native C++ applications can be compiled with emscripten [29] to run directly on a browser. Our approach was to build a desktop application that could be used as a standalone tool, and then to use WebAssembly to transform that desktop application into a web application. In addition, Qt provides extensive support in terms of documentation and has a widespread, active developer community. The performance of the application is also key consideration to ensure a high-quality user experience that can aid the rapid generation of

annotations. For image display and on-screen annotation management, the Graphics View (GV) [30] framework was employed. GV provides an extremely efficient view querying mechanism for smooth zooming. Furthermore, GV uses a binary space partitioning tree to provide very fast item discovery, and it can visualize large numbers of items in real-time. In addition, OpenCV was used as the image processing library to enable fast and efficient filtering operations on images.

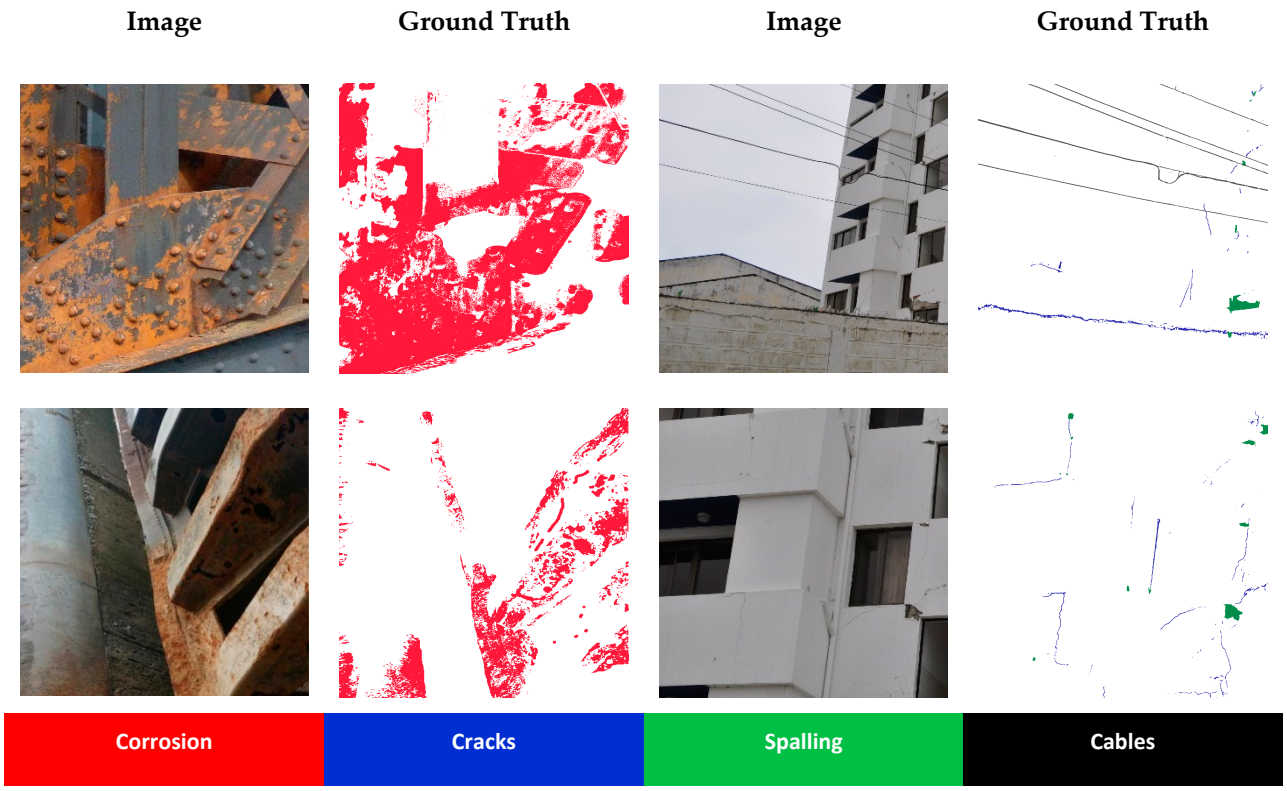


**Figure 1.** Examples of annotations for semantic segmentation of damage in civil infrastructure. It can be seen how the damage patterns are amorphous, complex, and hard to capture using regular annotation tools.

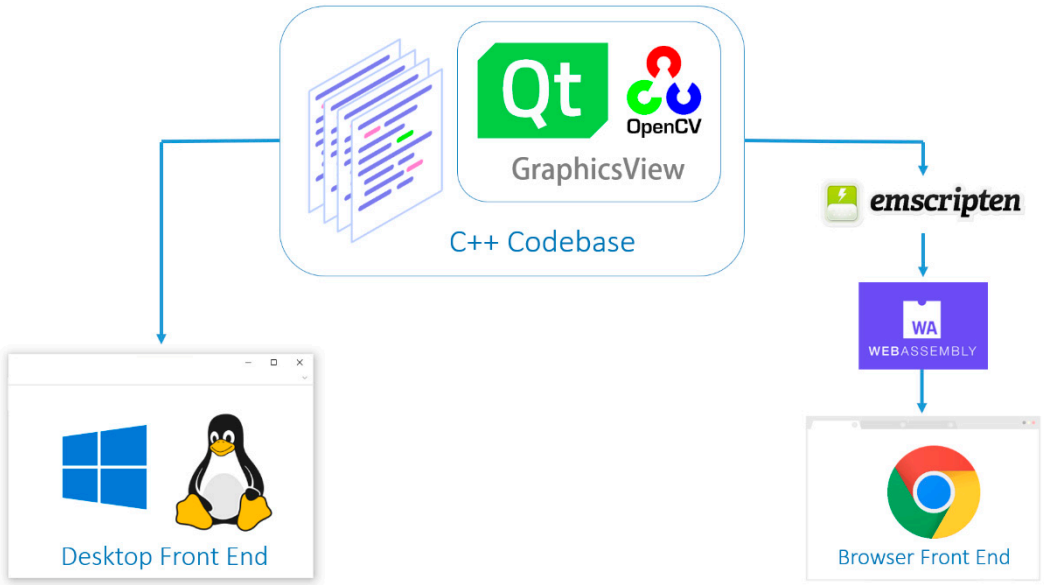


**Figure 2.** InstaDam front end frameworks and libraries.

### 2.3. User Interface Design

The InstaDam user interface was designed to address the described challenges for damage annotation. The front end user interface consists of five main parts as shown in Figure 3, namely the photo viewer, mask viewer, image processing bar, tool bar, and label bar. The photo viewer is where the user can view the original image with overlays of any annotations made by the user. The label bar lists the different classes of annotations, i.e., labels, that the user has created. The mask viewer provides a view of the selected image processing filter from the image processing bar at the bottom. The tool bar allows the user to pick from one of several pixel annotation tools implemented and provides access to commands such as opening and saving new projects or annotations, changing the label class, changing user permissions, and undoing/redoing actions.

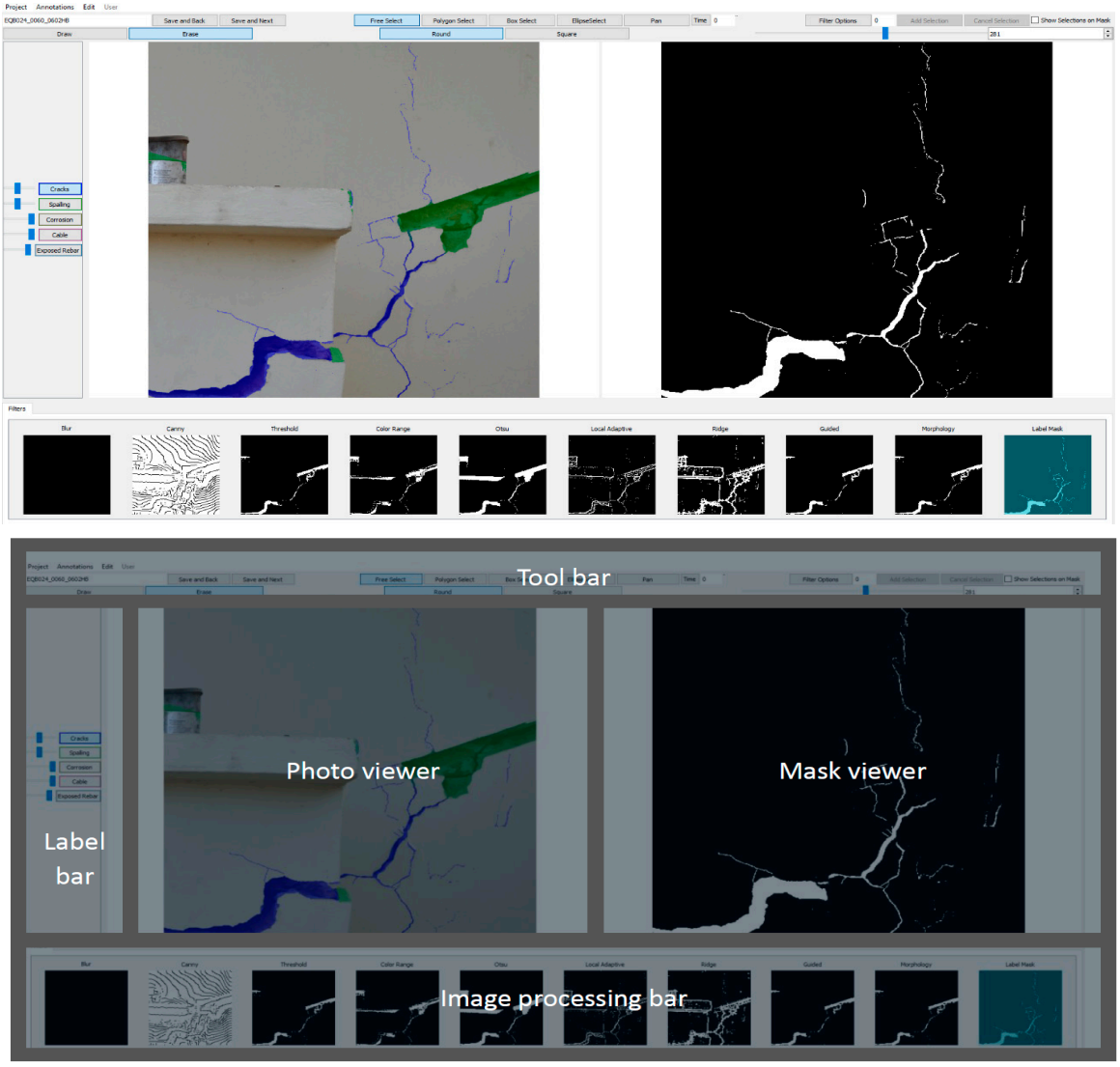


**Figure 3.** InstaDam user interface.

Annotations can be made through actions using one of the annotation tools either directly on the photo or on the mask. When annotations are made on the mask, the intersection between the user selected area and the mask appears as an annotation on the photo viewer. To better visualize the user interface, readers may refer to this short 30 s video [31]. The mask viewer and the photo viewer were designed such that any changes in the viewport due to zooming or panning are instantaneously reflected in the adjacent view as well. The final annotation is the intersection of the annotations on the photo viewer

and the mask viewer and is always visible on the photo viewer. This allows for seamless annotation of detailed shapes with high precision using the masks. The split view also helps with instantaneous feedback as the user modifies the tunable parameters of the image processing techniques to modify the mask.

#### 2.3.1. Annotation Tools

Common annotation tools are implemented to use directly or in conjunction with the generated masks. These include the Shape Selection tools (Box Select, for selecting rectangular regions, Ellipse Select for ellipsoid regions), Polygon Select for irregular regions that can be constrained by vertices, and the Free Select for selecting irregular regions on a pixel by pixel basis. Once made, annotations can be edited to fine tune any issues during the initial pass. Figure 4 shows examples of images annotated using the annotation tools directly on the photo viewer. The annotation tools can also be used in combination with the mask viewer to greatly reduce the number of user actions required per annotation, as shown in Figure 5.

**Figure 4.** Examples of annotated images where the tools are used directly on the photo using (i) ellipse tool, (ii) rectangle tool, (iii) polygonal tool (from left to right).

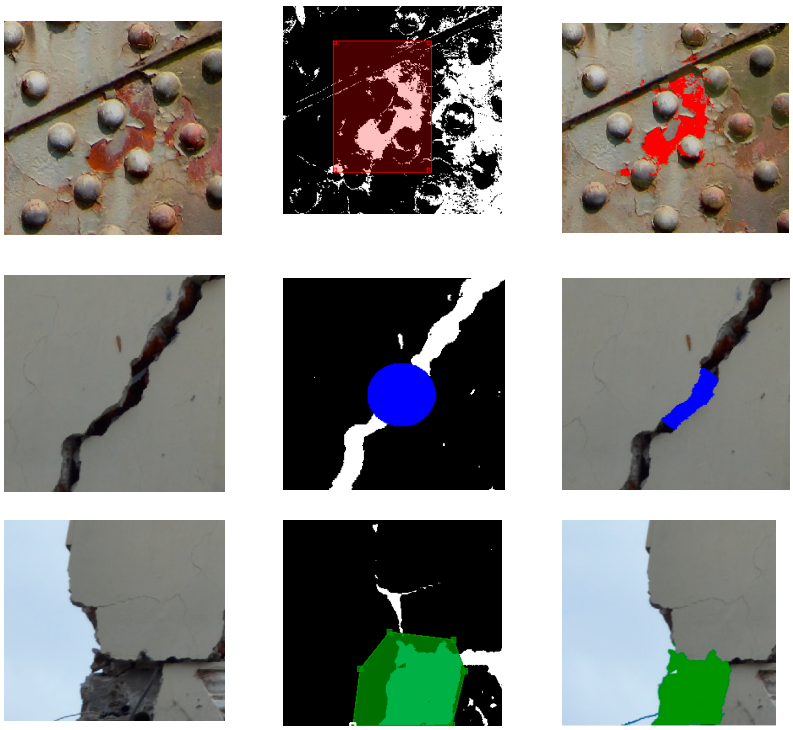


**Figure 5.** Examples of annotated images where the tools are used on the mask. The first column shows the image, the second shows the mask and the user action, and the final column shows the resulting annotation.

#### 2.3.2. Image Processing Techniques Implemented

Several different image processing techniques (IPTs) were implemented in InstaDam for mask generation. The IPTs were selected from the literature based on their relevance for automated detection of damage and the processing time. Two simple operations of direct thresholding and thresholding following a Gaussian blur [32] were implemented for cases where the contrast between the damage pattern and the background is high. Otsu thresholding [33], which is also commonly used in various image processing applications, and morphology operations [34] that allow the users to dilate and erode selected regions, were incorporated as well. The canny edge detection method [32], proposed for use in damage identification by Abdel-Qader et al. [35], was implemented for fine edges. Distance filtering in the hue-saturation-intensity color space was implemented for corrosion detection based on the method proposed by Medeiros et al. [36]. The ridge filter is often used in the medical field for segmentation of veins and blood vessels [37], and is incorporated for detecting fine lines such as cracks. Additionally, the guided filter [38] and local adaptive thresholding [39] available in opencv were incorporated. The time taken by each of the implementations to process one image of size 1MP on a Windows PC with an i7 processor is provided in Table 1. From the standpoint of usability, the processing time is very important, as delays in generation of the mask will likely result in the user avoiding use of the particular IPT. Table 1 also provides details about the tunable parameters for each of the IPTs that have been implemented in InstaDam. In addition to the IPTs mentioned, another option available is to create a mask out of existing annotations. This mask option is called the label mask and is useful when annotations are to be made by excluding other annotations.

**Table 1.** Computation time for different image processing techniques.

| IPT | Processing Time for 1 MP Image (ms) | IPT Parameters |
|---|---|---|
| Common parameters | 0.65 | invert |
| Threshold | 1.24 | threshold |
| Otsu | 1.85 | NA |
| Morphology | 2.77 | erode, dilate, open close |
| Local Adaptive Threshold | 3.06 | strength, detail |
| Gaussian blur | 5.25 | kernel size, threshold, |
| Canny edge | 21.32 | threshold min, threshold max, kernel size |
| Color distance | 25.26 | R, G, B, fuzziness |
| Ridge filter | 50.02 | scale |
| Guided filter | 50.44 | threshold, diameter, sigma |

### *2.4. Data Management*

InstaDam can be used with two modes of operation for data management: local and server modes. In the former, the application is run completely locally (illustrated in Figure 6), where data and annotations are stored on the user's local machine, whereas in the latter (illustrated in Figure 7), the front end communicates directly with the back end to store the data and annotations on the server-side database. To accommodate the divergence of these modes of operation while maximizing code reuse, the read and write operations are abstracted away in the front end code base. This flexibility allows the user to decide how they would like to store their data, given the size of their project and availability of servers.

The back end of InstaDam consists of a set of Web APIs that provides accessibility to the database for end-users. The back end can be seen as the middleware that hosts communication between multiple annotator-clients and the server database. Flask [40] is used as the backbone of the InstaDam back end. Flask is a micro web framework written in Python, which provides great usability and extensibility, along with a thriving community. It supports fast templates, strong WSGI features and thorough unit testability. On top of the Flask backbone, we implement our APIs, which adhere to the REST architectural constraints. SQLAlchemy is used as the object-relational mapper (ORM) for IntaDam's underlying

relational database. SQLAlchemy considers the database to be a relational algebra engine, providing abstractions from relational tables and entries to python objects. The usage of SQLAlchemy in the development of InstaDam creates the decoupling between the object model and database schema and helps to enable further modifications and development to the codebase.

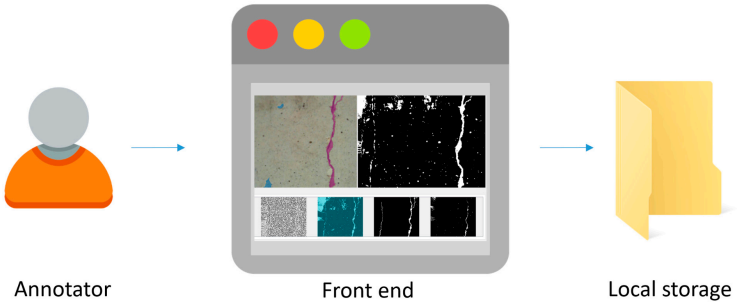


**Figure 6.** Local operation of InstaDam.

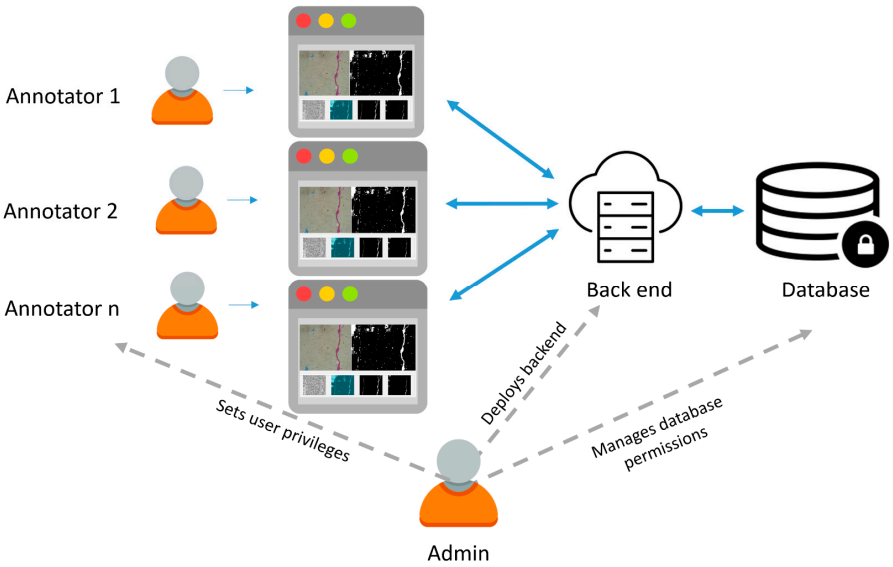


**Figure 7.** Server operation of InstaDam.

## 3. Experiments and Software Analysis

A data logging framework was incorporated in InstaDam, where user actions could be captured by (i) measuring the time the user spent at different parts of the UI, and (ii) recording the software state at every user click. The software state includes information such as the location of the cursor, the button clicked, the tools selected, the IPT properties, etc. To quantitatively evaluate the benefits of InstaDam, two experiments were conducted and discussed in this section.

### *3.1. User Testing with InstaDam*

During the experiments, the users were given some nominal introduction about the software use but were not briefed with the details about these IPTs. The users were still able to effectively utilize the software, as there is instant feedback between changing of the IPT parameters and the corresponding mask generation. As a result, the users were directly able to see the effect of changing the parameters and develop an intuition for the role of different parameters in a short span of time. Additionally, dynamic thumbnails for each of the filters is available at the bottom. These thumbnails reflect the chosen parameters and allow users to quickly understand the type of mask that would be generated by different

IPTs. Thus, while the authors cannot make guarantees about optimal selection by the users, the instant feedback and dynamic thumbnails make good IPT selection straightforward.

### *3.2. Comparison with Existing Annotation Software*

The first experiment aimed to compare InstaDam with existing annotation software. The image labeler app available in Matlab was chosen for comparison, as it is often used by researchers to create computer vision datasets and is easily accessible. Two student annotators were employed to annotate a dataset of 25 images. Each student was given the same set of instructions where they were introduced to the different classes to be annotated (cracks, spalling, corrosion, cables). The students were first allowed to become familiar with both software and then were asked to annotate the images using (i) Matlab, (ii) InstaDam but without using IPTs, and (iii) InstaDam. The time taken for each annotation was recorded by the data logger in InstaDam and recorded manually for the annotations created in Matlab using a stopwatch. The time comparison is presented in Figure 8.

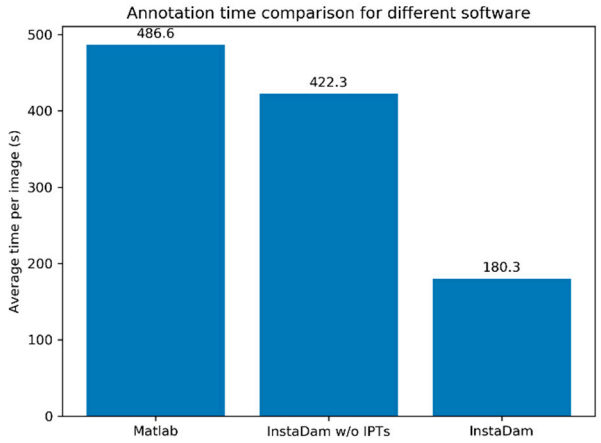


**Figure 8.** Annotation time comparison for different software.

The generated data were also used to objectively evaluate the quality of annotations and the consistency across image annotators. Different masks can be used for different portions of the image; for example, a user may utilize an IPT such as the ridge filter for fine cracks and utilize the guided filter for thicker cracks. The final annotation will be a combination of different masks at different portions, and manual input, directly on the photo. As the final goal is to have good quality annotations, an objective evaluation of the masks themselves is a challenging problem. The authors resolve this issue by evaluating the consistency of final annotations across the users instead of directly evaluating masks. The users will mostly likely take different paths to get their final annotated result. The selection of masks at different stages will be influenced by user judgement of the damage shape and border. Thus, if the annotations are consistent across users, a reasonable conclusion would be that the software facilitates accurate annotating. The annotation consistency is measured using the intersection-over-union (IoU) between annotations by different annotators. The IoU between two annotations is given by the number of pixels of intersection of the two annotations divided by number of pixels in the union of the two annotations. The comparison of inter-annotator IoU for the three annotation software are provided in Figure 9.

### *3.3. Annotating Damage and Materials*

As a second experiment, the annotators were given two sets of images from the Madnet dataset [4] and were a sked to annotate five classes of damage (cracks, spalling, corrosion, cables and rebar) and five classes of materials (steel, concrete, asphalt, masonry, other). The data from the annotation process for every image were recorded and a summary of the data is presented here. Figure 10 shows the time users spend on different parts of the

user interface while annotating different classes. Figures 11 and 12 show the distribution of time spent using each of the filters and tools, respectively.

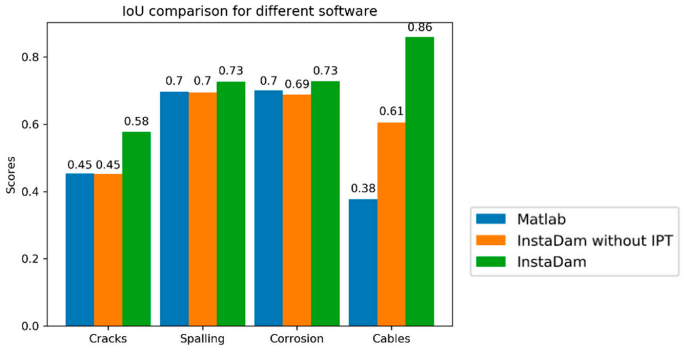


**Figure 9.** Intersection-over-union comparison for different software.

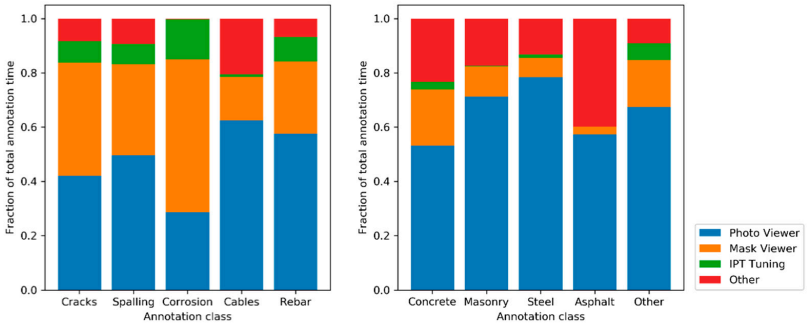


**Figure 10.** Annotation time spent using different parts of the user interface.

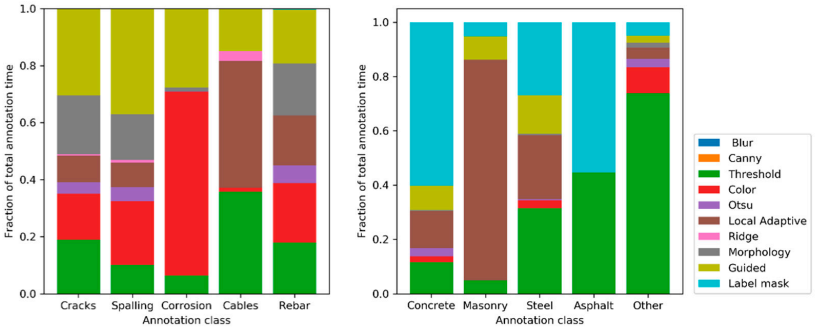


**Figure 11.** Annotation time per filter.

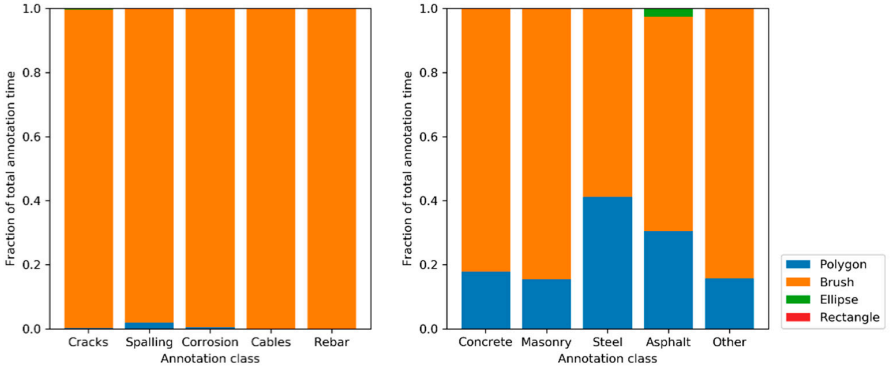


**Figure 12.** Annotation time spent using different tools.

## 4. Discussion of Results

The data generated from the conducted experiments is analyzed in this section and additional qualitative improvements of the use of InstaDam are presented.

### 4.1. Improvements in Efficiency

A key advantage of InstaDam is the significant time savings afforded by using the software. The data summary reported in Figure 8 shows that compared with Matlab, the annotation time using InstaDam reduces from an average of 486.6 s per image to 180.3 s per image which is a 62.9% reduction in annotation time. When InstaDam is used without IPTs, the annotation speed is slightly faster than Matlab, suggesting that the software functionality is on par with existing annotation software. The availability of IPT masks drastically reduces the annotation time as expected. These savings are significant, and they will not only help produce annotations faster, but they will also play a major role in developing higher quality annotations.

### 4.2. Improvements in Consistency

Having a high consistency across annotations for different images is an important necessity for high quality datasets. When producing large datasets, multiple annotators are often employed and thus consistency across annotators is important to improve annotation quality. The inter-annotator IoUs shown in Figure 9 demonstrate that using InstaDam with IPTs improves the IoUs for all the annotation classes as compared to InstaDam without IPTs or Matlab. The improvements are most pronounced for annotations of cracks and cables. This is easily explained by the high contrast between these features and the background, making the use of IPTs ideal. Figure 13 provides two examples of images where this difference is illustrated. In the first image, crack annotations are shown. Annotator 1 misses the fine crack on the right when using Matlab. However, with InstaDam, the IPTs make the presence of the fine crack very obvious and make the annotation a lot easier. The crack widths are also more consistent with InstaDam. Similarly, for the cables shown in the next figure, the thickness of cables is more consistent with the use of InstaDam. The IPTs guide different users to have more consistent decisions regarding the location of the pixels that need annotation especially around the edges of the investigated damages.

### 4.3. Analysis of Feature Use

Figure 10 shows the use of different parts of the software by the user. For damage annotation, a significant amount of the annotation time is spent by users on the mask viewer, further indicating that the IPTs are used extensively. As damage usually takes amorphous and complex shapes the IPTs are tremendously useful in delineating boundaries and speeding up damage annotation. On the other hand, for material annotation, the users prefer to use the photo viewer the most. The IPTs do not provide the same increase in efficiency for material annotations as they do for damage annotations due to the different nature of boundaries between materials. Figure 11 shows the filters used most often for different classes. The guided filter is most frequently used for both cracks and spalling and is especially effective in filtering out edge features. The morphology operations allow users to control the thickness of the crack annotated and is also used extensively for crack annotation. The color filter is clearly the most used for corrosion annotation. As the color of corrosion is usually distinctly reddish hues, color-based filtering is highly effective. The local adaptive threshold and simple threshold are most effective for annotating cables. There is no clear choice by the annotators for the annotation of rebar. The mask type used most often for material annotation is the label mask. This is because materials are mutually exclusive and material annotations fill up the entire image. Thus, the option to create annotations excluding previously created annotations avoids duplicating work describing the edges of materials. Finally, Figure 12 shows the distribution of different tools used. The brush tool is almost exclusively used for annotating damage types, and the most commonly used tool even for material annotation. The polygonal tool is also used

heavily for material annotations. The analysis of the user data indicates that the features incorporated into InstaDam are used extensively and that the mask viewer is a valuable addition to improve the efficiency of annotation.

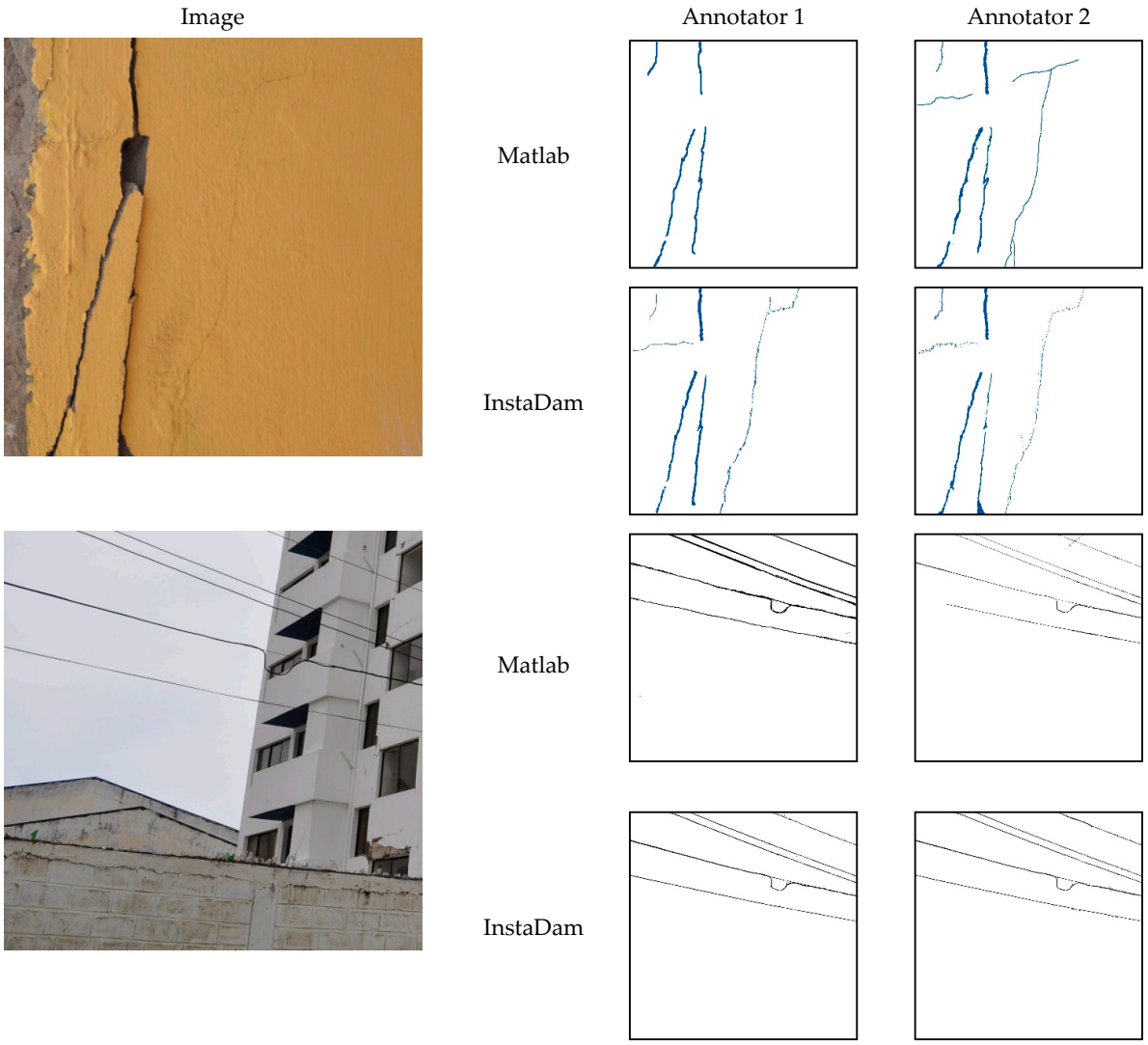


**Figure 13.** Annotation consistency for different software.

### *4.4. Open Source Data Development Platform*

Several commercial software applications have recently become available for creating semantic segmentation datasets [24–27]. While research on using deep learning for the identification of damage constitutes a large and growing community, the number of users is not yet significant enough for these companies to modify their offerings for the use of researchers in this field. An open source alternative such as InstaDam offers researchers the option to tailor the annotation platform to their own needs and requirements. InstaDam provides documentation and developer guides for quick modification and addition to the source code.

## 5. Future Research

InstaDam offers decoupled front and back ends, providing great extensibility. Each part can be tailored easily to fit different usage scenarios. Further components, such as a cloud-native front end, can also be easily developed without modification to the user APIs. This section discusses extensions to the InstaDam codebase for future research.

### 5.1. Machine Learning Enhanced Annotations

Two methods to enhance the annotation methods for structural inspection using machine learning will be considered. The first involves automated tuning of the image processing techniques (IPTs). The feature analysis of InstaDam showed that 10–15% of user time is spent on tuning IPTs depending on the particular class being annotated. Based on the user data recorded, a smart suggestion filter will be implemented whereby the properties of the IPTs can be automatically suggested. A mechanism will need to be developed where the user can easily accept or reject the suggestion. The second approach is a more straightforward approach where the annotations are directly suggested by a deep network based on the chosen class. Similar options are available in other software such as Amazon Sagemaker and Labelbox at a premium. However, even on those platforms, a large number of annotated images are required before the suggestions are effective, as there are no pretrained models available for damage features. Incorporating models pretrained on the developed datasets as well as the generated synthetic data for typical damage features into InstaDam will help to speed up the annotation process even further.

### 5.2. Web-Based Platform

Using WebAssembly (WASM) for Qt, a front end version that could be run remotely via a web page can be generated from the existing InstaDam codebase. WebAssembly for Qt is a very new technology and has some notable restrictions. Many of these have to do with security restrictions applied by browsers to apps running in the JavaScript sandbox. Several of these restrictions were known ahead of time and workarounds have been incorporated into the InstaDam software architecture. Other restrictions were discovered after the start of development and require further design and analysis. The primary restriction accounted for is that WASM apps do not have direct access to the local file system. The default Qt deployment creates a temporary file system in memory, but that is not permanent. A workaround for this was to use the browser's native JavaScript open file dialog to open projects and images, and use the browser's download feature to save outputs, but only as a single zip file. Another restriction is that due to WASM running in a JavaScript sandbox, security restrictions prevent it from having a DNS lookup, or contacting the back end on any other machine that the one hosting the WASM web page. Both of these can be worked around using static IP addresses and hosting the browser front end on the same server as the back end. Future development of this code can take advantage of cutting-edge Qt 5.14 and the necessary patches.

### 5.3. Extension to Crowdsourced Annotation Platform

In this study, InstaDam has been used as an annotation tool by trained annotators that were selected after careful review. The experiments conducted demonstrated the benefits of InstaDam for the efficiency and consistency of annotations. The developed framework for InstaDam however allows for further scaling for use as a crowdsourced annotation tool. Crowdsourcing platforms such as Amazon Mechanical Turk [41] have millions of users, many of whom will likely produce low-quality annotations due to poor or malicious judgment. While crowdsourcing annotations in such a scenario, two important steps that can contribute to quality assurance and quality control include (i) identifying which annotators produce reliable annotations through automated protocols, and (ii) subdividing large tasks into microtasks to reduce annotator fatigue. Lu et al. [42] proposed methods for quality control and quality assurance for construction video annotations involving preassessment and postassessment filtering of annotators. The same approach can be adapted for use in InstaDam. In the preassessment stage, the performance of annotators is tested against a gold standard annotated dataset and then used for filtering. In the postassessment stage, filtering is done using repeated labeling by different annotators combined with a vote-based metric. For the second point of creating microtasks, regions of interest can first be selected, and the segmentation can then be done on the selected region of interest. Splitting these two tasks will enable annotators to focus on damage search and

careful annotations separately, rather than at the same time. When pre-existing training data are available, the selection of regions of interest can be done more intelligently using trained deep learning models. The extension of InstaDam to a crowdsource annotation tool will help in the development of more comprehensive datasets for inspection applications.

### 5.4. User Interface Testing

To the authors' knowledge, software platforms comparable to InstaDam for annotating civil infrastructure damage are not currently available. The objective of the paper is to present a new software framework for this purpose. The developed software and code are released as an open source offering that can be utilized by the research community at large. While some initial investment of time may be required, the returns allow for much faster labeling as the number of annotated images increases. Future work will involve a detailed study on the time for users to understand the filters. Additionally, the authors have done their best with the available resources to make a fair comparison with existing software. While the experiments illustrate the advantages of using InstaDam, future research will involve testing alternate designs with a larger number of participants.

## 6. Conclusions

In this paper, InstaDam—an open source software platform for fast pixel-wise annotation of damage—was presented. InstaDam allows the user to choose from a variety of image processing techniques (IPTs) to generate masks that greatly speed up the annotation process. The software includes annotation tools developed in Qt C++ and a cloud-based data management framework developed in python using Flask. Experiments were conducted to analyze the advantages of InstaDam over existing annotation software and validate the user interface design. The experiments showed the InstaDam provides a 63% reduction in the annotation time while also enhancing the annotator consistency for several commonly annotated defects such as cracks and cables. The analysis of the user data captured showed that the implemented IPTs are employed extensively by the users indicating the need for such tools. A discussion of the different IPTs used most often for different classes was presented as well as a discussion of use of the different tools and features implemented. The combination of these features enables efficient and consistent annotations using InstaDam and will help enable the development of large datasets for structural inspections.

**Author Contributions:** Conceptualization, V.H., F.A., Y.N. and B.F.S.J.; methodology, V.H., D.F., F.A., and M.G.-F.; software, V.H., F.A., D.F., W.Y. and Y.T.; validation, V.H. and F.A.; resources, M.D.S., M.G.-F. and B.F.S.J.; writing—original draft preparation, V.H., F.A., D.F., W.Y. and Y.T.; writing—review and editing, M.G.-F., Y.N., B.F.S.J., M.D.S., and V.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** Funding for this work was provided by the United States Army Corps of Engineers through the U.S. Army Engineer Research and Development Center Research Cooperative Agreement W912HZ-17-2-0024.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy restrictions.

**Acknowledgments:** The authors would like to acknowledge the help of Shengyi Wang and Shannon Chen for help with the image annotations.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References


1. Hoskere, V.; Narazaki, Y.; Hoang, T.A.; Spencer, B.F., Jr. Vision-based Structural Inspection using Multiscale Deep Convolutional Neural Networks. In Proceedings of the 3rd Huixian International Forum on Earthquake Engineering for Young Researchers, Champaign, IL, USA, 12 August 2017.
2. Spencer, B.F., Jr.; Hoskere, V.; Narazaki, Y. Advances in Computer Vision–based Civil Infrastructure Inspection and Monitoring. *Engineering* **2019**. [CrossRef]
3. Kim, B.; Cho, S. Automated Vision-Based Detection of Cracks on Concrete Surfaces Using a Deep Learning Technique. *Sensors* **2018**, *18*, 3452. [CrossRef] [PubMed]
4. Hoskere, V.; Narazaki, Y.; Hoang, T.A.; Spencer, B.F., Jr. MaDnet: Multi-task Semantic Segmentation of Multiple types of Structural Materials and Damage in Images of Civil Infrastructure. *J. Civ. Struct. Health Monit.* **2020**, *10*, 757–773. [CrossRef]
5. Chen, F.-C.; Jahanshahi, R.M.R. NB-CNN: Deep Learning-based Crack Detection Using Convolutional Neural Network and Naïve Bayes Data Fusion. *IEEE Trans. Ind. Electron.* **2017**, *65*, 1. [CrossRef]
6. 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. Civ. Infrastruct. Eng.* **2018**, *33*, 731–747. [CrossRef]
7. Carr, T.A.; Jenkins, M.D.; Iglesias, M.I.; Buggy, T.; Morison, D.G. Road crack detection using a single stage detector based deep neural network. In Proceedings of the 2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS), Salerno, Italy, 21–22 June 2018; pp. 1–5. [CrossRef]
8. Yeum, C.M. Computer Vision-Based Structural Assessment Exploiting Large Volumes of Images. Ph.D. Thesis, Purdue University, West Lafayette, IN, USA, 2016. Available online: https://docs.lib.purdue.edu/open_access_dissertations/1036 (accessed on 20 March 2020).
9. Kim, B.; Cho, S. Automated crack detection from large volume of concrete images using deep learning. In Proceedings of the 7th World Conference on Structural Control and Monitoring, Qingdao, China, 22–25 July 2018.
10. Narazaki, Y.; Hoskere, V.; Hoang, T.A.; Fujino, Y.; Sakurai, A.; Spencer, B.F. Vision-based automated bridge component recognition with high-level scene consistency. *Comput. Civ. Infrastruct. Eng.* **2019**, 12505. [CrossRef]
11. Alipour, M.; Harris, D.K.; Miller, G.R. Robust Pixel-Level Crack Detection Using Deep Fully Convolutional Neural Networks. *J. Comput. Civ. Eng.* **2019**, *33*, 04019040. [CrossRef]
12. Dung, C.V.; Anh, L.D. Autonomous concrete crack detection using deep fully convolutional neural network. *Autom. Constr.* **2019**, *99*, 52–58. [CrossRef]
13. Liang, X. Image-based post-disaster inspection of reinforced concrete bridge systems using deep learning with Bayesian optimization. *Comput. Civ. Infrastruct. Eng.* **2018**. [CrossRef]
14. Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The Cityscapes Dataset for Semantic Urban Scene Understanding. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 3213–3223.
15. Ros, G.; Sellart, L.; Materzynska, J.; Vazquez, D.; Lopez, A.M. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 3234–3243.
16. Menze, M.; Geiger, A. Object Scene Flow for Autonomous Vehicles. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 3061–3070. [CrossRef]
17. Yu, F.; Chen, H.; Wang, X.; Xian, W.; Chen, Y.; Liu, F.; Madhavan, V.; Darrell, T. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. Available online: https://bair.berkeley.edu/blog/2018/05/30/bdd/ (accessed on 12 March 2020).
18. Li, Y.; Li, H.; Wang, H. Pixel-Wise Crack Detection Using Deep Local Pattern Predictor for Robot Application. *Sensors* **2018**, *18*, 3042. [CrossRef] [PubMed]
19. Xu, Y.; Li, S.; Zhang, D.; Jin, Y.; Zhang, F.; Li, N.; Li, H. Identification framework for cracks on a steel structure surface by a restricted Boltzmann machines algorithm based on consumer-grade camera images. *Struct. Control Health Monit.* **2018**, *25*, e2075. [CrossRef]
20. Hoskere, V.; Narazaki, Y.; Spencer, B.F.; Smith, M.D. Deep learning-based damage detection of miter gates using synthetic imagery from computer graphics. In Proceedings of the 12th International Workshop on Structural Health Monitoring, Stanford, CA, USA, 10–12 September 2019; Volume 2, pp. 3073–3080. [CrossRef]
21. LabelMe: The Open Annotation Tool. Available online: http://labelme.csail.mit.edu/Release3.0/ (accessed on 1 August 2018).
22. MATLAB-MathWorks, Natick, MA, USA. Available online: https://www.mathworks.com/products/matlab.html (accessed on 19 March 2020).
23. Adobe, San Jose, California, United States | Adobe Photoshop | Photo, Image, and Design Editing Software. Available online: https://www.adobe.com/products/photoshop.htm (accessed on 19 March 2020).
24. Labelbox, San Francisco, CA, USA. Available online: https://labelbox.com/ (accessed on 12 March 2020).
25. Computer Vision Prodigy. An Annotation Tool for AI, Machine Learning & NLP. Available online: https://prodi.gy/features/computer-vision (accessed on 12 March 2020).
26. Instance Segmentation Assistant–Hasty.ai Documentation. Available online: https://hasty.gitbook.io/documentation/annotating-environment/instance-segmentation-tool (accessed on 12 March 2020).

27. Radically Efficient Annotation Platform to Speed up AI Projects–Kili Technology. Available online: https://kili-technology.com/ (accessed on 12 March 2020).
28. Qt | Cross-Platform Software Development for Embedded & Desktop. Available online: https://www.qt.io/ (accessed on 12 March 2020).
29. Main—Emscripten 1.39.8 Documentation. Available online: https://emscripten.org/ (accessed on 12 March 2020).
30. Graphics View Framework | Qt Widgets 5.14.1. Available online: https://doc.qt.io/qt-5/graphicsview.html (accessed on 13 March 2020).
31. InstaDam. Available online: https://youtu.be/N3z1YUMr-ME (accessed on 23 December 2020).
32. Szeliski, R. Computer Vision. In *Texts in Computer Science*; Springer: London, UK, 2011; ISBN 978-1-84882-934-3.
33. Otsu, N. A threshold selection method from gray-level histograms. *IEEE Trans. Syst. Man Cybern.* **1979**, *9*, 62–66. [CrossRef]
34. Jahanshahi, M.R.; Masri, S.F.; Padgett, C.W.; Sukhatme, G.S. An innovative methodology for detection and quantification of cracks through incorporation of depth perception. *Mach. Vis. Appl.* **2013**, *24*, 227–241. [CrossRef]
35. Abdel-Qader, I.; Abudayyeh, O.; Kelly, M.E. Analysis of Edge-Detection Techniques for Crack Identification in Bridges. *J. Comput. Civ. Eng.* **2003**, *17*, 255–263. [CrossRef]
36. Medeiros, F.N.S.; Ramalho, G.L.B.; Bento, M.P.; Medeiros, L.C.L. On the evaluation of texture and color features for nondestructive corrosion detection. *EURASIP J. Adv. Signal Process.* **2010**, *2010*, 817473. [CrossRef]
37. Staal, J.; Abràmoff, M.D.; Niemeijer, M.; Viergever, M.A.; Van Ginneken, B. Ridge-Based Vessel Segmentation in Color Images of the Retina. *IEEE Trans. Med. Imaging* **2004**, *23*, 501. [CrossRef] [PubMed]
38. The Robotics Institute Carnegie Mellon University. Robust Crack Detection in Concrete Structures Images Using Multi-Scale Enhancement and Visual Features. Available online: https://www.ri.cmu.edu/publications/robust-crack-detection-in-concrete-structures-images-using-multi-scale-enhancement-and-visual-features/ (accessed on 31 July 2020).
39. OpenCV. Available online: https://opencv.org/ (accessed on 12 March 2020).
40. Flask (1.1.x). Available online: https://flask.palletsprojects.com/en/1.1.x/ (accessed on 19 March 2020).
41. Amazon Mechanical Turk. Available online: https://www.mturk.com/ (accessed on 2 August 2020).
42. Liu, K.; Golparvar-Fard, M. Crowdsourcing Construction Activity Analysis from Jobsite Video Streams. *J. Constr. Eng. Manag.* **2015**, *141*, 04015035. [CrossRef]