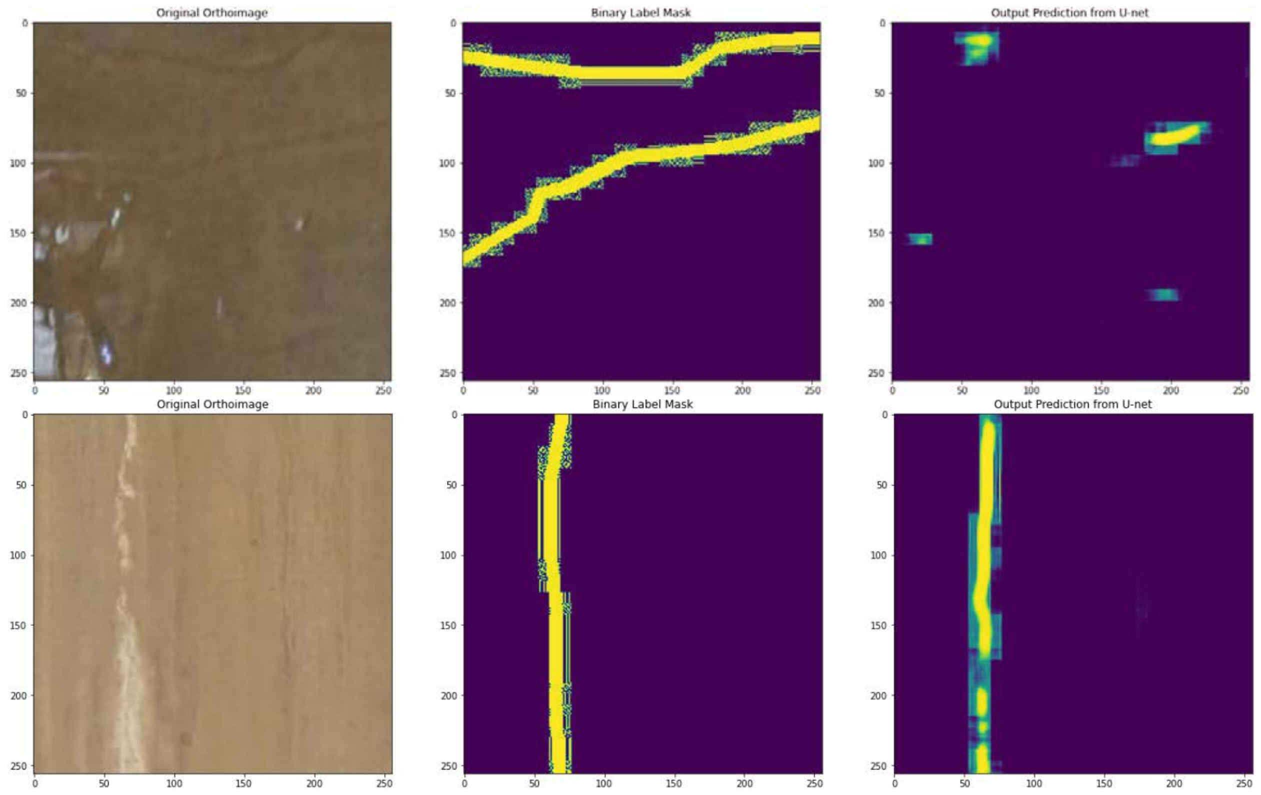
Identifying Cracks in Concrete from Previously Collected UAS Data Using Deep Learning

Science and Technology Program
Research and Development Office
Final Report No. ST-2020-20105-01



U.S. Department of the Interior September 29, 2020

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE 09/15/2020	**2. REPORT TYPE** Research		**3. DATES COVERED** *(From - To)* 2020
4. TITLE AND SUBTITLE Identifying Cracks in Concrete from Previously Collected UAS Data Using Deep Learning			**5a. CONTRACT NUMBER** 20XR0680A1-RY15412020WI20105/X0105
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER 1541 (S&T)
6. AUTHOR(S) Matthew Klein, Civil Engineer; Zackary Leady, Modeler, CGB Region			**5d. PROJECT NUMBER** Final Report ST-2020-20105-01
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Concrete and Structural Laboratory; Technical Service Center; Bureau of Reclamation; U.S. Department of the Interior; Denver Federal Center; PO Box 25007 (86-68530); Denver, CO 80225			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Science and Technology Program; Research and Development Office; Bureau of Reclamation; U.S. Department of the Interior; Denver Federal Center; PO Box 25007, Denver, CO 80225-0007			**10. SPONSOR/MONITOR'S ACRONYM(S)** Reclamation
			11. SPONSOR/MONITOR'S REPORT NUMBER(S) Final Report ST-2020-20105-01
12. DISTRIBUTION/AVAILABILITY STATEMENT Final Report may be downloaded from https://www.usbr.gov/research/projects/index.html			
13. SUPPLEMENTARY NOTES			
14. ABSTRACT Crack mapping concrete structures is a way to document and monitor cracks. In the past, crack mapping has been very labor intensive from data collection to documentation. The use of UAS and photogrammetry has allowed for faster and more comprehensive data collection and products including high-resolution orthoimages used to identify and document cracks. In addition, deep learning models can be used to automatically identify cracks from the orthoimages. This paper presents the process used to develop a deep learning model for automatic crack detection from data collected by UAS.			
15. SUBJECT TERMS Deep learning, machine learning, UAS, crack mapping			
16. SECURITY CLASSIFICATION OF:	**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES** 58	**19a. NAME OF RESPONSIBLE PERSON** Matthew Klein
a. REPORT U / **b. ABSTRACT** U / **THIS PAGE** U			**19b. TELEPHONE NUMBER** *(Include area code)* (303) 445-2368

Acknowledgements

The Science and Technology Program, Bureau of Reclamation, sponsored this research.

Identifying Cracks in Concrete from Previously Collected UAS Data Using Deep Learning

Final Report No. ST-2020-20105-01

prepared by

Technical Service Center
Concrete and Structural Laboratory
Matthew Klein, Civil Engineer

California Great Basin Region
Division of Planning
Zackary Leady, Modeler

Cover Photo: Example inputs and outputs from the automated crack mapping deep learning model.

Peer Review

Bureau of Reclamation
Research and Development Office
Science and Technology Program

Final Report ST-2020-20105-01

Identifying Cracks in Concrete from Previously Collected UAS Data Using Deep Learning

ZACKARY LEADY Digitally signed by ZACKARY LEADY Date: 2020.10.01 09:01:50 -07'00'

Prepared by: Zackary Leady
Modeler, Division of Planning, MP-740

MATTHEW KLEIN Digitally signed by MATTHEW KLEIN Date: 2020.09.30 07:02:50 -06'00'

Prepared/Technical Approval by: Matthew Klein, P.E., Ph.D.
Civil Engineer, Concrete and Structural Laboratory, 86-68530

TREVOR STOCKTON-SALAS Digitally signed by TREVOR STOCKTON-SALAS Date: 2020.09.30 13:38:15 -06'00'

Checked by: Trevor Stockton-Salas
Civil Engineer, Concrete and Structural Laboratory, 86-68530

KATIE BARTOJAY Digitally signed by KATIE BARTOJAY Date: 2020.09.30 15:40:10 -06'00'

Peer Review by: Katie Bartojay, P.E.
Manager, Concrete and Structural Laboratory, 86-68530

Acronyms and Abbreviations

3D	three-dimensional
3DR	3D Robotics
APS-C	advanced photo system type-C
CAD	computer aided design
CFR	Code of Federal Regulations
CGB	California Great Basin
CNN	convoluted neural network
CPN	Cascade Pacific Northwest
CPU	central processing unit
CSL	Concrete and Structural Laboratory
DOI	Department of the Interior
DOI	digital object identifier
etc.	and so forth
FAA	Federal Aviation Administration
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPU	graphics processing unit
GSD	ground sampling distance
i.e.	that is
KL	Kullback-Leibler [divergence]
LiDAR	light detecting and ranging
mIoU	mean-Intersection over Union
MP	megapixel
NUPO	National UAS Project Office
Reclamation	Bureau of Reclamation
RGB	red green blue
SGD	stochastic gradient descent
SOTA	state-of-the-art
T2D2	Thornton Thomasetti Damage Detector
TSC	Technical Service Center
USACE	United States Army Corps of Engineers
USGS	United States Geologic Survey
UAS	unmanned aerial system

Contents

Tables

Figures

Executive Summary

Concrete cracking can often be used to monitor a concrete structure's health and can indicate the cause of the damage. Concrete crack maps are sometimes developed to document the patterns or growth of the cracks to determine the severity of the problem.

Crack maps can be hand drawn based on close-up observations, using a telescope device, or from high-resolution pictures and images. These methods are rough estimates of the cracks and in order to document the exact location, length, width of the cracks other methods must be used. One of these methods is the use of photogrammetry. Photogrammetry can be used to generate an orthomosaic or orthoimage of the surface that is corrected for perspective skew. This orthoimage then contains the exact location and can be used to quantify the lengths and widths of the crack. In addition, these highly accurate crack maps can be used to compare the growth and movement of the crack over time.

Photogrammetric data collection requires access to perpendicular locations of the surface – something that is not easy with tall vertical surfaces like the downstream face of a dam, which can be obstructed by a powerhouse, canyon walls and hard to access due to the downstream water course. However, with the recent adoption of unmanned aerial systems (UAS) at the Bureau of Reclamation, collecting data required for photogrammetric generation of orthoimages is easier than before. UAS can be used to position the camera at the exact location required for high-resolution data collection and can operate to collect hundreds and even thousands of images of the surface.

The actual crack maps are usually manually drawn by engineers in a computer aided design (CAD) software. Lines are drawn in over imported orthoimages and then saved as drawings. This process is somewhat subjective and labor intensive. Advances in computing hardware and a subcategory of artificial intelligence called deep learning can be used to develop a model that can automatically detect cracks from a database of digital images.

This report describes the proof-of-concept procedure for automated crack identification developed through a collaboration between Reclamation's Technical Service Center's (TSC) Concrete and Structural Laboratory (CSL) and one of Reclamation's experts on machine and deep learning located in the California Great Basin (CGB) Region. The procedure utilized UAS data that had already been processed for manual crack mapping to build a deep learning model to find and identify cracks autonomously. It summarizes data collection, preparation, processing, and analysis. In addition, the report also recaps a couple of third-party crack/defect mapping solutions. In the discussion and conclusions, several suggestions and recommendations are given for future work to improve the robustness and accuracy of the procedure and results.

Introduction

Concrete design has long acknowledged the presence of cracks in concrete and engineering practices are in place to limit the cracks to appropriate dimensions based on the loading condition. Most of these cracks are small and barely perceivable with the naked eye. However, cracks may develop that are detrimental to the performance of the structure. In addition, the style, type, extent, and size of the crack can indicate the nature of the deterioration which, in turn, helps to determine the appropriate repair.

For this reason, when conducting a condition assessment of a concrete structure, engineers will detect and mark cracks, often on the structure, if it is accessible, and then document the cracks in a crack map. The crack map is a drawing that will place the location, length and sometimes the width of the cracks within the plan or elevation views of different components of the structure. This crack map can be analyzed for the source of the deterioration and archived for future comparison to determine if the crack(s) have changed. Photographs of the surface can accompany the crack maps if the cracks are large enough or if the photographs have a great enough resolution that can allow identification of the cracks.

Identification of cracks require close access to the surface as the cracks and other defects are usually only detectible by the naked eye at about 24 inches [1]. Once the cracks and other defects are identified, that are marked on the concrete sometimes with a red or colored crayon for easy detection when photographing or sketching them on a drawing.

At Reclamation, cracks have been identified from the ground, using telephoto lenses on high-resolution film and digital cameras, utilizing rope access teams to reach inaccessible features such as inclined and vertical dam faces, and by deploying UAS to capture images. Documentation of the cracked surfaces has been from hand sketches, photos, and digital images as well as by combining a series of high-resolution images into a composite orthorectified image called an orthoimage.

For crack mapping large areas such as the face of an entire face of dam, the orthoimages are imported into CAD software and lines are drawn over the cracks for emphasis and to allow for computation of total number of cracks. This also allows for future time-based quantitative comparisons.

Background

For this research, several areas of specialized expertise were assembled. The first was using UAS collected data. This data was previously collected for manual crack mapping on other projects. The second was working with data that was processed photogrammetrically to produce composite orthoimages. The third was leveraging deep learning expertise to develop a model that could identify cracks from the UAS data.

Unmanned Aerial Systems

Reclamation has been operating UAS commercially since 2012, though UAS usage increased shortly after the Federal Aviation Administration (FAA) released its 14 CFR (Code of Federal Regulations) Part 107 rules which were incorporated into the Department of the Interior (DOI) in the late summer of 2016 [2]. Initially, UAS data collection was limited to aerial photography and videography but it quickly grew into a reliable tool for photogrammetric data collection given that UAS can be pre-programmed with Global Navigation Satellite System or GNSS waypoint navigation.

The Reclamation Technical Service Center (TSC) in Denver acquired UAS shortly after the FAA Part 107 rules were released. Because of its expertise in concrete inspection, geographical information systems (GIS), and photogrammetry, methods were quickly developed to collect close-range, high-resolution imagery. Within about 2-1/2 years, the TSC had five certified UAS pilots that had completed 22 missions, 20 of which were in support of photogrammetric data collection.

The most common airframe used by DOI and Reclamation has been the 3DR (3D Robotics) Solo quadcopter. The Solo features custom payloads and sensors (with a maximum payload capacity of about 1 pound), supports real-time video downlink, has a 10-minute battery life, can be operated autonomously using pre-programmed waypoint navigation, and has a 0.5-mile range [3].

The sensor most often paired with the 3DR Solo to collect photogrammetric data is the Ricoh GR II. The GR II has a 16-megapixel (MP), APS-C (advanced photo system type-C) sensor with a global shutter and fixed focal length lens [4]. It also features a time-lapse mode that is used to capture overlapping photogrammetric images synchronized with the UAS velocity. The camera offers full control over all the exposure settings as well as can capture images in a raw format for postprocessing the exposure if required. Since the Solo was not originally intended for use with the GR II, custom, three-dimensional (3D)-printed fixed mounts were designed by the United States Geological Survey (USGS) National Unmanned Aircraft Systems Project Office (NUPO) in Denver (see Figure 1).

Figure 1. —3DR Solo and Ricoh GR II with 3D printed mount [2].

Photogrammetry

Photogrammetry has been used by Reclamation for many years specifically for mapping. Within the past 15 years or so, photogrammetry has been explored for developing 3D models of foundations, rock faces, and most recently, for facilities. With the development of high-resolution digital cameras and faster computing, photogrammetry can generate millions of points for a subject similar to the digital models that LiDAR (Light Detecting and Ranging) scanners can produce. These models are capable of producing products like point clouds, meshed surfaces, topography, maps, and high-resolution orthoimages as shown in Figures 2 to 8 below.

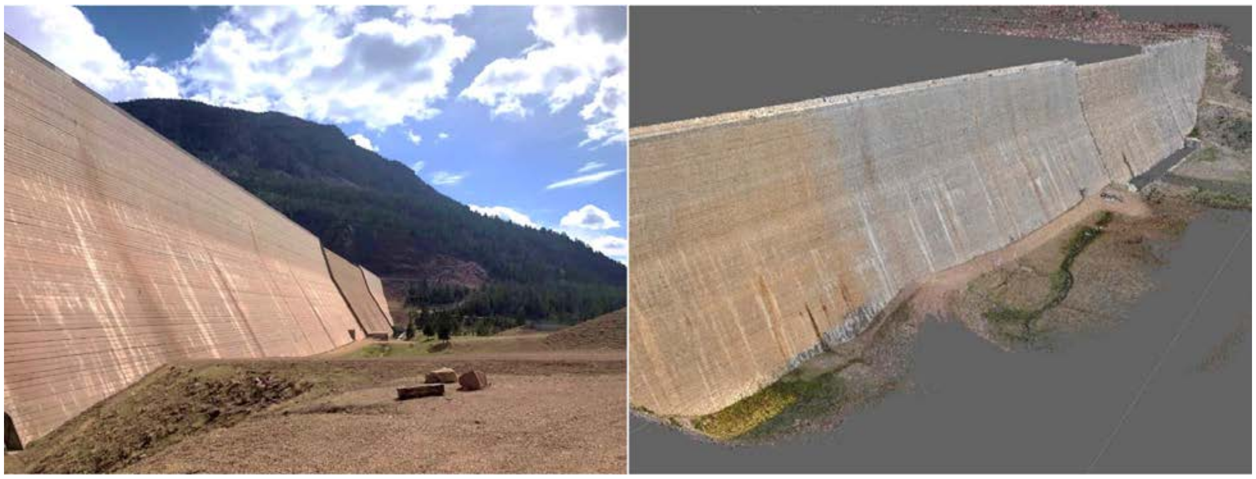


Figure 2. —Image of Upper Stillwater Dam in Utah (left) and a screenshot of a photogrammetric point cloud of the same dam (right).

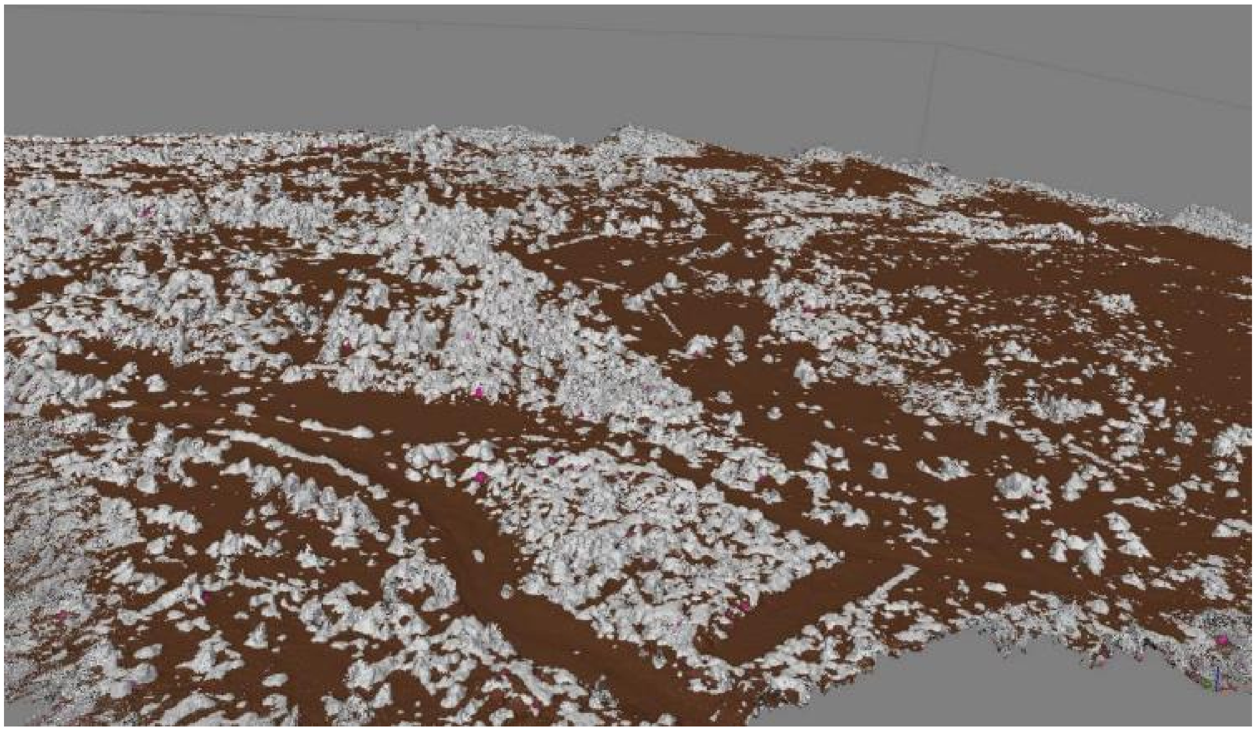


Figure 3. —Screenshot of a photogrammetric point cloud at Altus Reservoir. The ground points have been classified in brown and the vegetation are seen in white [2].

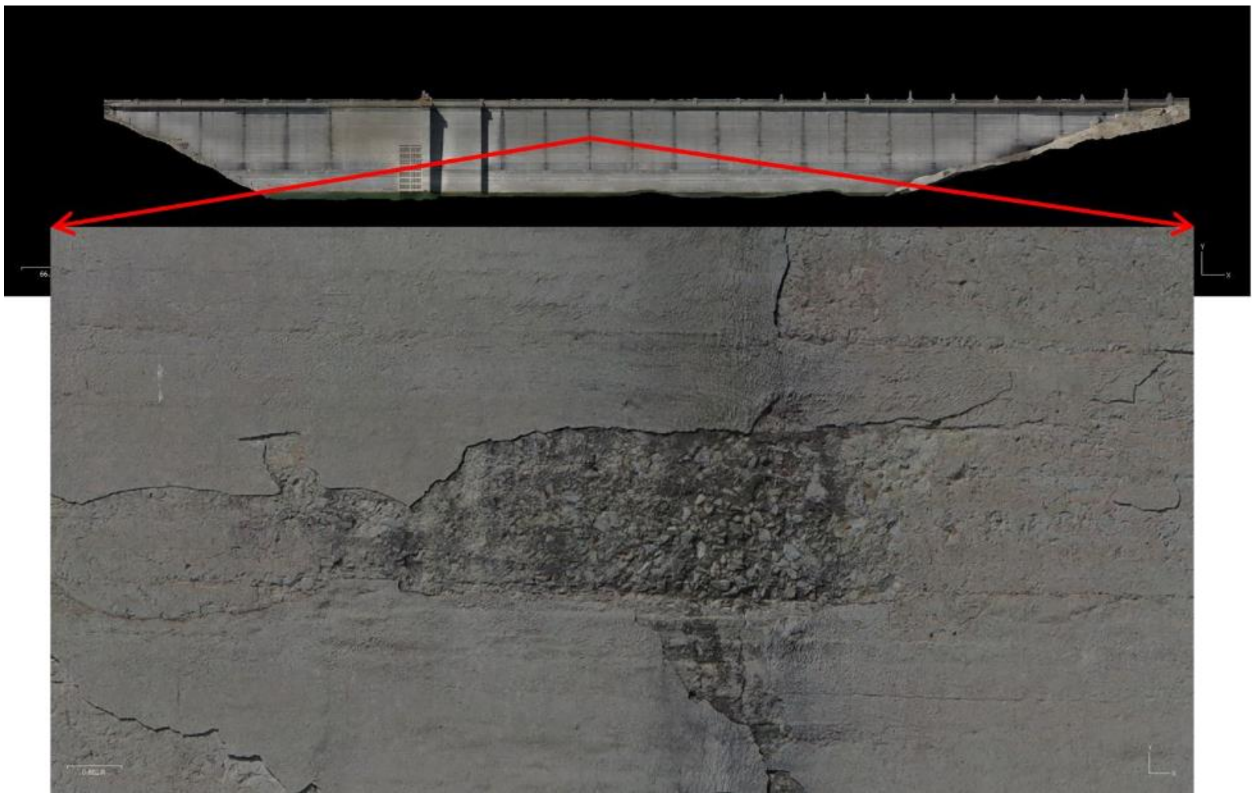


Figure 4. —Screenshot of the upstream face orthoimage at Elephant Butte Dam (above) and an example of the orthoimage resolution (inset) [2].

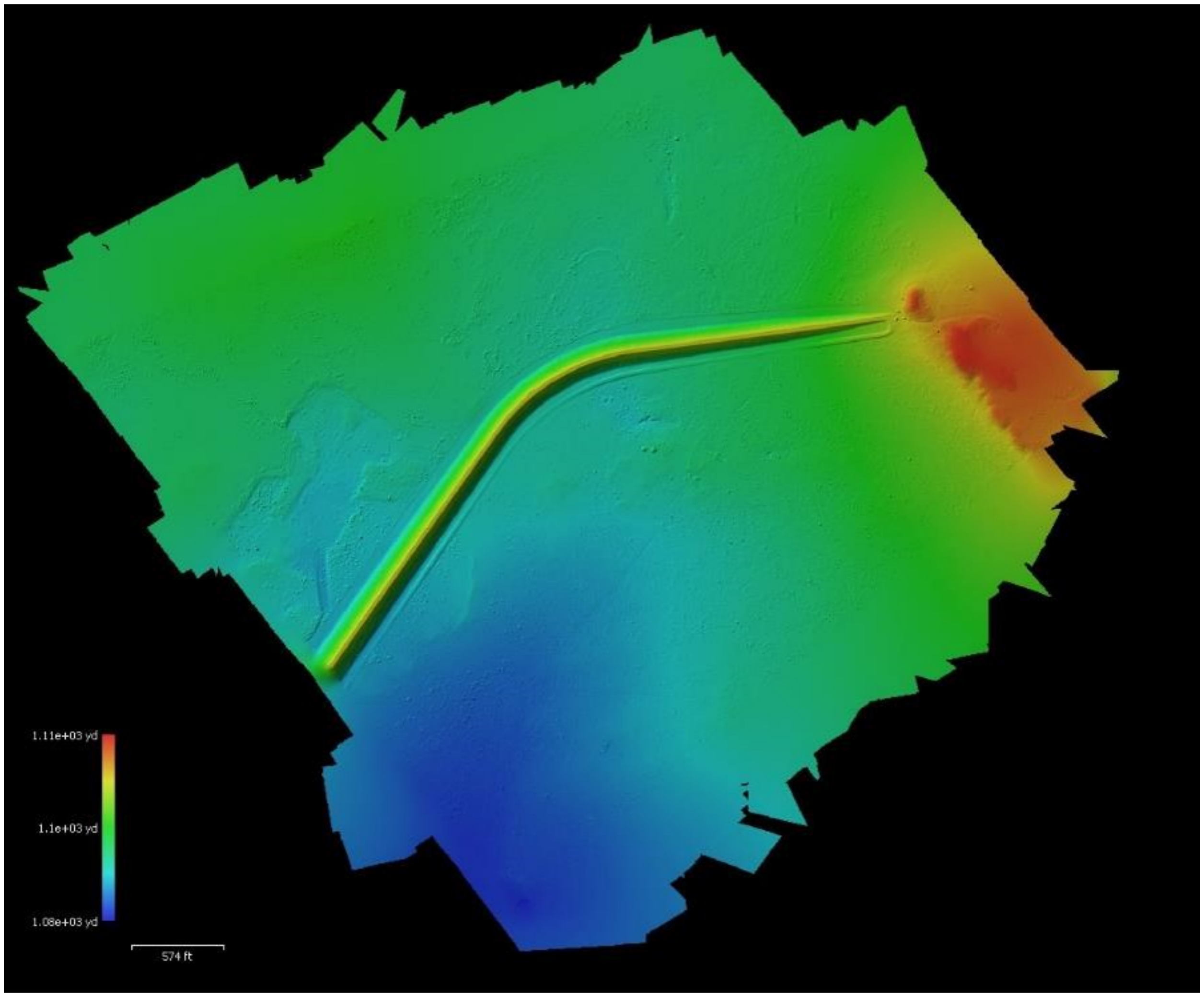


Figure 5. —Screenshot of the digital surface model (DSM) of the left abutment at Brantley Dam. The area shown is about 1 square mile [2].

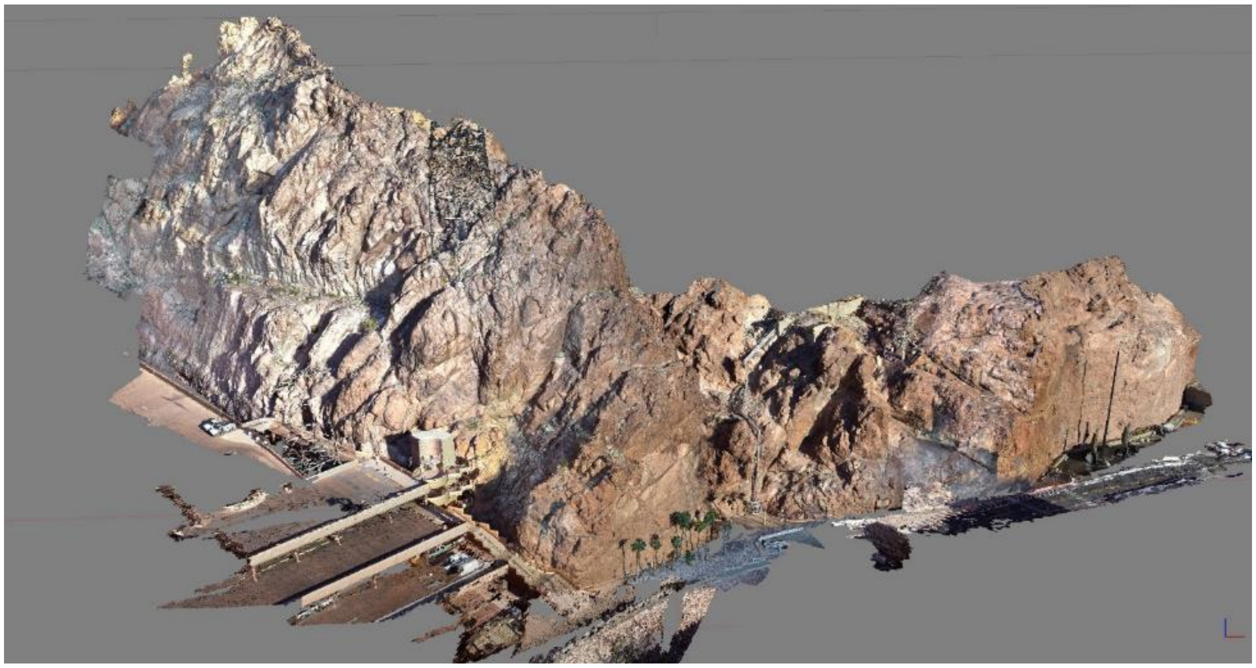


Figure 6. —Screenshot of the point cloud model of the geologic features above the parking garage and visitor center at Hoover Dam [2].

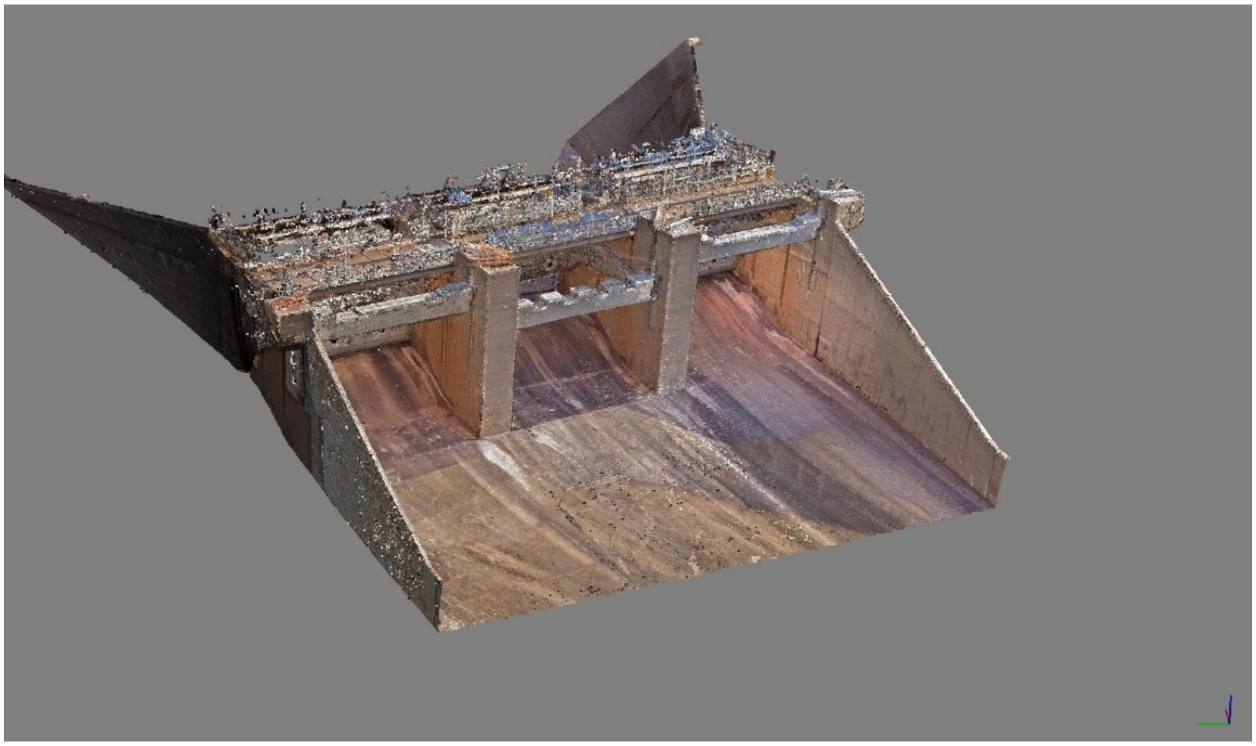


Figure 7. —Screenshot of the photogrammetric model of the gate structure at Vallecito Dam [5].

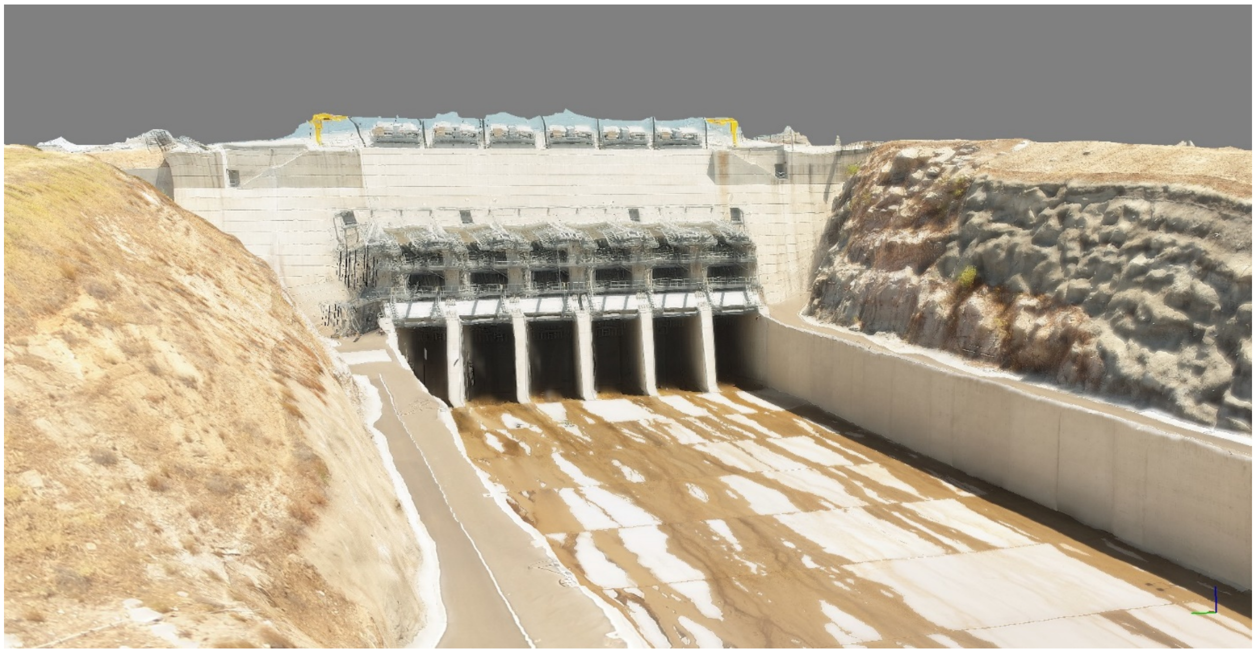


Figure 8. —Screenshot of the photorealistic textured mesh of the Folsom Dam Auxiliary Spillway Control Structure.

Crack Mapping

Crack and deterioration mapping had limited use from terrestrial data collection and photogrammetry in a several instances in Reclamation [6] [7]. However, once techniques for close-range, high-resolution UAS data collection and photogrammetry had been developed, that knowledge was applied to crack mapping as early as 2018 [8].

The current process of crack/deterioration mapping involves importing the orthoimage into CAD software. Then the cracks are observed visually at the resolution of the orthoimage. This observation is systematic throughout the orthoimage and often, gridlines and blocks are added and numbered to aid in the observation process. Once a crack is identified, a line is drawn over the crack. When the crack is jagged but not varying from the overall trendline, a single line is used to represent the crack. Due to the time required and to limit bias, this process is conducted usually by one to two engineers with a background and training in concrete deterioration mechanisms. Review of the cracks mapping is informal and occur on subsets of the data also by an engineer with a background in concrete deterioration mechanisms.

Using this method, crack mapping is much more precise and easier to compare than before with less room for error and is less subjective. However, it is time consuming and can take up to a month to complete. Reclamation projects that used the high-resolution crack mapping method include Elephant Butte Dam, Seminoe Dam, Brock Reservoir, and Folsom Dam [8] [9] [10] [11] (see Figure 9, below, for an example).

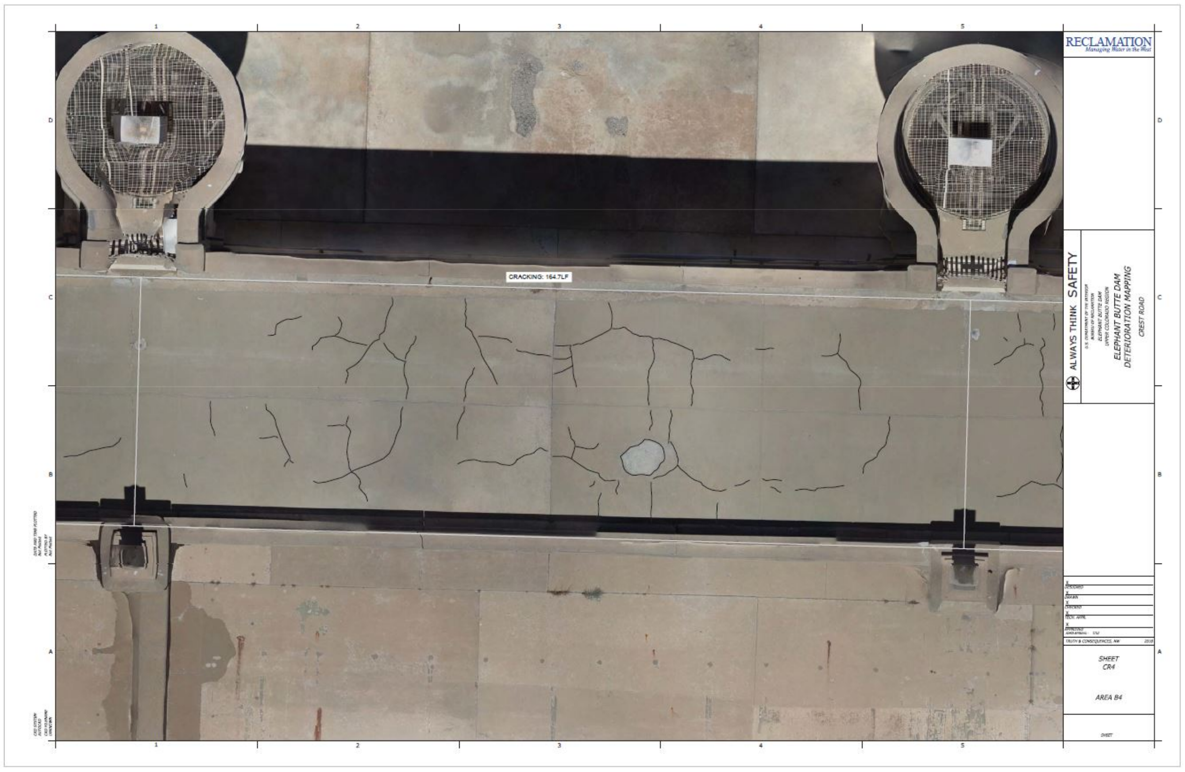


Figure 9. —Crack map drawing example at Elephant Butte Dam [8].

Deep Learning

Machine learning is a subset of artificial intelligence specialized in automatically improving computer algorithms through experience. By providing more data, the computer can actually improve how the task is performed. Deep learning is a subset of machine learning that leverages deep neural networks which are capable of automatically detecting differences in computer vision tasks such finding cracks in digital images. While Reclamation has been exploring machine learning applications since 2016, it has recently gained more traction as greater expertise on the subject has been gained. Specifically, Zackary Leady, in the CGB Region, has expertise in setting up and coding deep learning methodologies beginning with a 3-year research project started in 2018, Seasonal Wetland and Temporary Floodplain Delineation using High-Resolution Sensing and Machine Learning. Leady used the procedures developed and applied them to detecting cracks in this project.

Deep Learning is a branch of machine learning that is focused on the use of many layers in a single neural network to achieve a “deep” neural network that is capable of various tasks. For the purpose of this discussion the focus is primarily on deep convolutional neural networks (CNNs) which have become the state-of-the-art (SOTA) methodology for image-based tasks such as image classification and semantic segmentation. The project developed herein is a binary semantic segmentation task, where each pixel is labeled as “crack” or “no crack”. This type of task differs from image classification which attempts to label an entire picture of an object as that object such as a cat with the label “cat” or a picture of a dog with the label “dog”. Over the last decade, as deep learning has quickly become an efficient way to process and analyze computer vision-related tasks, many architectures or ways of combining the layers in a deep learning neural network have been researched and tested. One of the most significant break-through architectures for CNNs has been

U-net [12]. U-net is named after its U-like structure wherein the encoder, the downward, first half of the “U”, learns to compress the input image into a representative form useful for determining pixel-wise labeling, which is then decoded in the decoder or the upward, second half of the “U” to incrementally up-sample the data back to the original shape [12]. A typical U-net architecture is displayed if Figure 10 below.

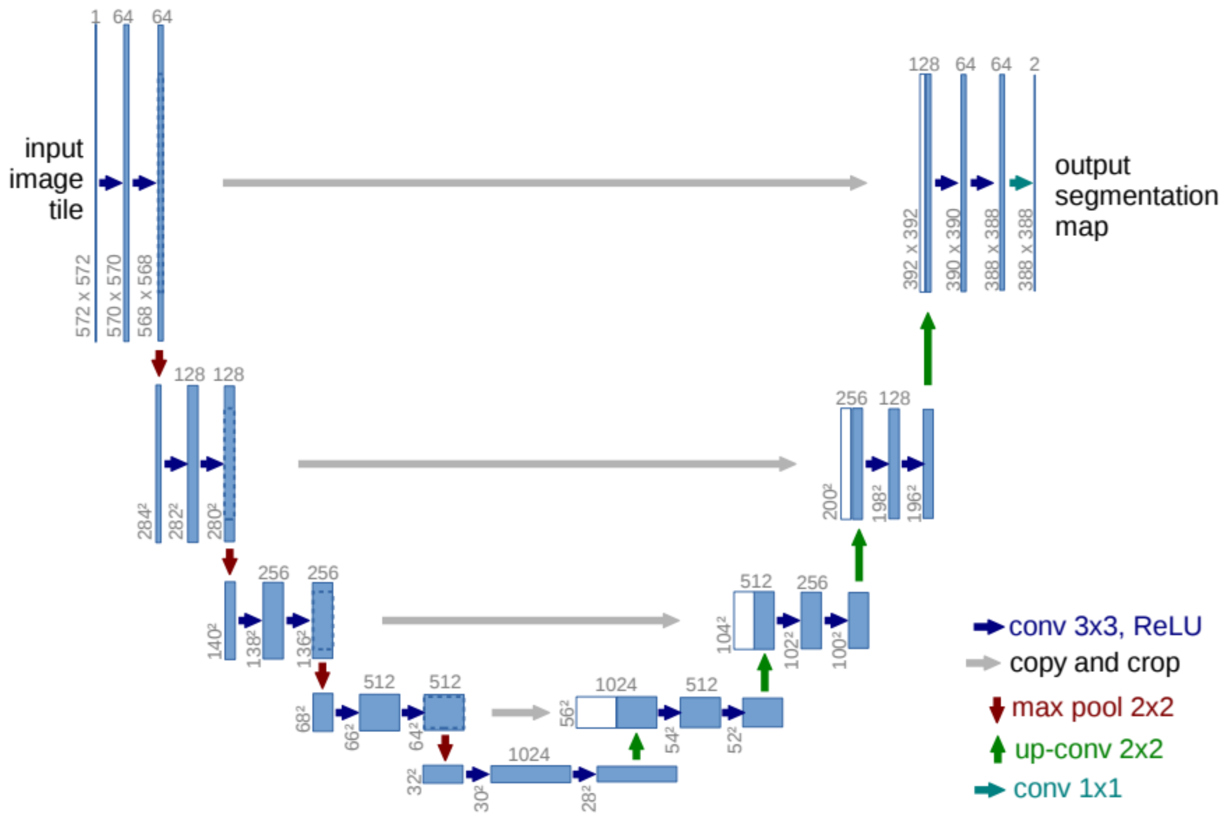


Figure 10. —An original U-Net architecture diagram [12].

U-net and the multitude of U-net improved architectures have consistently made progress in expanding achievable digital image tasks, especially in the biomedical imagery semantic segmentation field [13].

As a result of the CGB Region’s prior experience with U-nets which focuses on applying U-nets to satellite imagery, it was a natural decision and progression to choose a U-net architecture for the scope of this proof-of-concept project. The team is aware of several newer architectures which could lead to a significant improvement such as Mask R-CNN, Unet++, or the DeepLab family of networks and plans to explore these improved architectures in future work [14] [15] [16].

Problem

There are several issues related to traditional crack mapping operations. These include access to the surfaces of the structure, inaccuracies that develop from sketching the cracks, inconsistency in identifying cracks, overlooked or missed cracks, distortion in photographs from the perspective and angle of the camera, weather conditions affecting the surface such as water, and that the operation is time consuming – for some large data sets, crack mapping can require up to a month to complete.

Though initially time consuming, computer-based deep learning can be used to teach a computer to analyze data and, in this case, the computer is taught to analyze images for cracks. In addition, once the computer is taught how to find data, it can be improved by providing more data. Eventually, the computer can find cracks in data with greater precision and in less time than previous methods. The computer can operate all times of the day without needing a break.

Objectives

The objectives of the research are as follows:

- Define a data set that can be used for automatic concrete crack detection
- Organize a dataset into a training set, validation set, and test set.
- Develop code that can use the training dataset to train a concrete crack detection U-net. The training set includes example pairs of input images and labelled cracks
- Prove the deep learning neural network can identify previously unseen concrete cracks using the test dataset

Method

The methods employed in this project include a review of commercially available software and the development of in-house algorithms. The in-house procedure required several steps including data selection and preparation followed by training, validation, and testing the model using the deep learning pipeline.

Third Party Software Review

Review of third-party software capable of detecting cracks and defects in concrete were limited to two companies: Thornton Tomasetti's T2D2 applications and Niricson damage assessment services. For the T2D2 and Niricson evaluation, company representatives gave virtual presentations of the capabilities. In both cases, the offerings are still fairly new, and each company is still working on their commercial availability and pricing.

T2D2 stands for Thornton Tomasetti Damage Detector [17]. It is an artificial intelligence technology that can be used to detect defects within a number of structural types and materials including concrete that was developed by Thornton Tomasetti's CORE Studio [18]. The system is primarily promoted as a real-time damage detector that analyzes live video like that from a UAS or a smartphone. The system is programmed to detect defects from a wide variety of structural materials including concrete. The application will draw a box identifying the damage and will give a rating to help organize and prioritize the defects (see Figure 11 below). Finally, the application would automatically generate a report that summarizes the detected defects. The system is intended to be cloud based and would require a subscription though the company indicated that they may be able to install their system locally. In addition, CORE Studio is positioned to develop custom applications like the ones that are more interesting to Reclamation – find and identify defects in orthoimagery and 3D models.

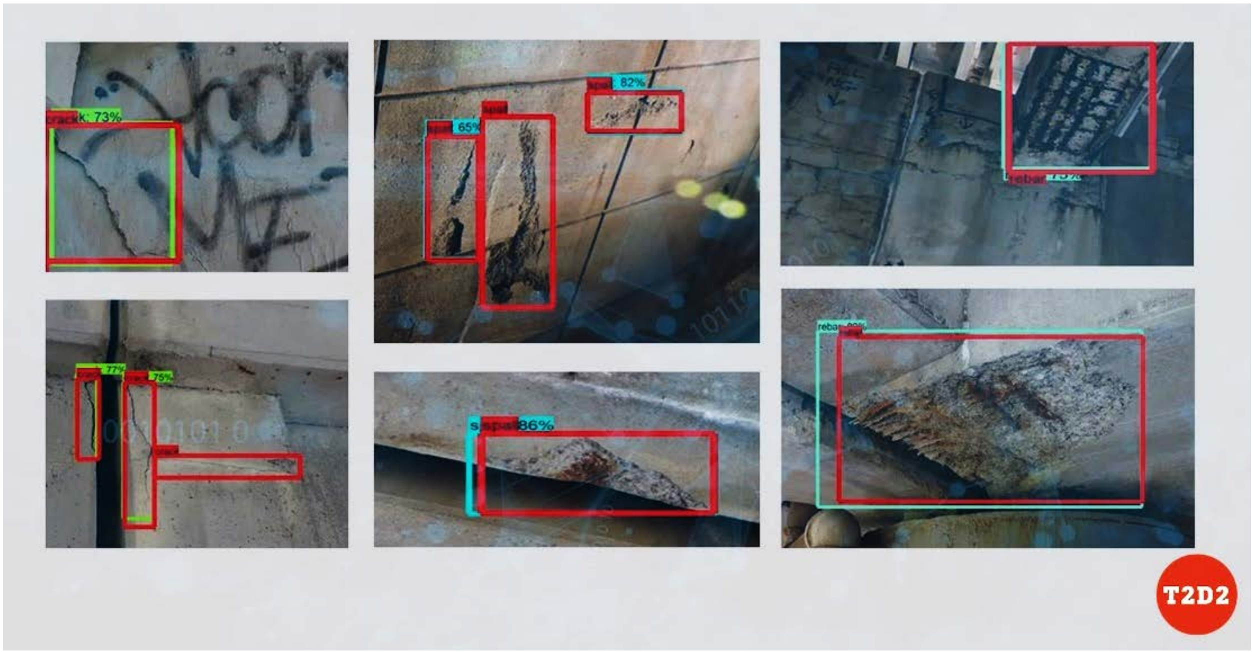


Figure 11. —Examples of the T2D2 deterioration detection system (red boxes highlight damage in real-time) [17].

Niricson Software Inc. is a company that is developing solutions for infrastructure condition assessment and risk management. Their system combines robotics and computer vision to inspect concrete civil infrastructure. Instead of just a software solution, the company deploys UAS flight navigation systems to autonomously collect optical and thermal images as well as acoustic data (from sensors lowered onto the concrete surface), detects and quantifies the damage using machine learning, and then incorporates the data into a report with drawings, orthoimages, and lists of observed defects. The system is optimized for concrete damage including cracks, delaminations, voids, and corrosion, among others. In addition to pinning the location of the damage on an orthoimage and drawing, the report tabulates the estimated cause or type of damage, width, length, and even depth (see Figure 12 below). Niricson team members have demonstrated the software on several dams and water control structure features.

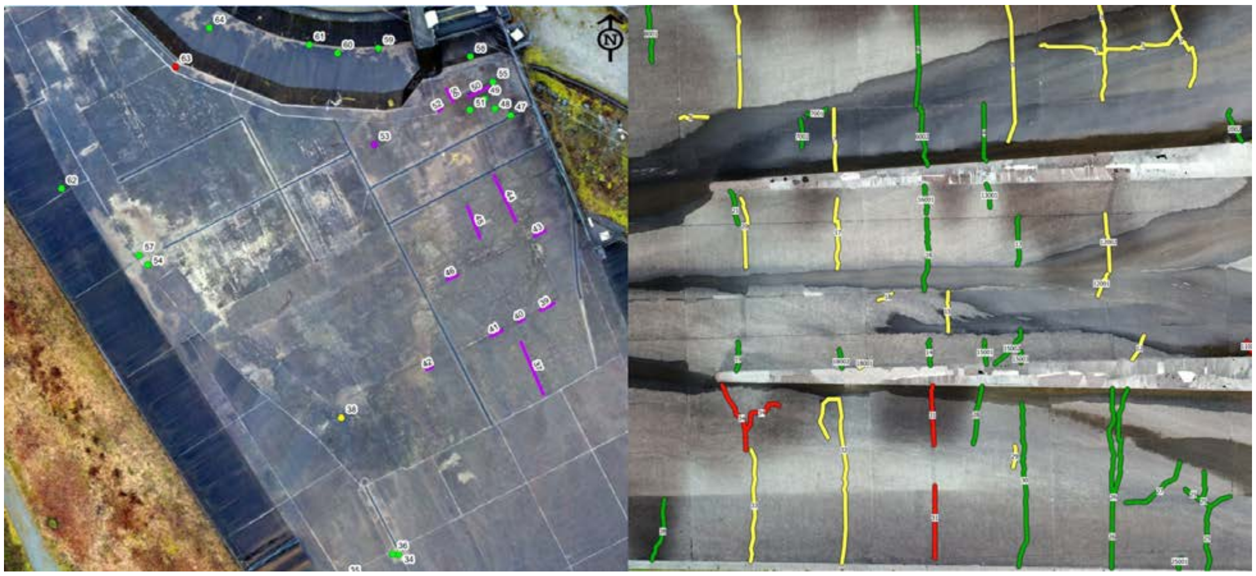


Figure 12. —Examples of the Niricson reporting system where cracks and damage are identified by different color lines, points, and labels (Photos Courtesy of Niricson Software).

Data Preparation

The original data used in this study came from previously existing UAS data. The data was collected for crack mapping of the lower portions of the gate piers and inverts in each of the six bays of Folsom Dam Auxiliary Spillway Gate Structure (see Figure 8). Twelve walls and six inverts were captured; both the wall and floor areas were about 24 feet high by 70 feet long (from the pier nose to the radial gate).

The data was collected between Tuesday, July 16 and Wednesday, July 17, 2019. The data consisted of individual 16.2 MP images captured from a Ricoh GR II camera mounted on a 3DR Solo quadrocopter UAS. Since the images were captured within the gate structure, GNSS positioning and waypoint navigation were not able to be used to collect the data. The UAS was operated from a nominal 30 feet from each surface in a technique to allow for about 80% image overlap for photogrammetric processing into high-resolution orthoimages. Camera settings were set manually to limit shadowing though raw images were captured to allow for exposure adjustments in post-processing. Targets were placed throughout each subject area for scaling and accurate measurements within the orthoimages.

An average of 168 images were collected of the inverts and an average of 314 images were collected of each wall. The average ground sampling distance or GSD (based on the offset) of the images was about 0.06 inches per pixel.

The data was processed photogrammetrically using Agisoft PhotoScan V.1.2.6 to produce the scaled orthoimages. The photogrammetric software was fed the images for each wall or floor separately. The software matched adjacent images using uniquely identified key points and position of each camera location was solved. Once the three-dimensional camera locations were known, the images were corrected for skew along each x, y, and z rotation axis and combined to create a total of 18 orthoimages (three for each bay: North wall, South wall, and invert). Scale was added by providing measurements between each target.

Once the orthoimages were produced, they were imported into AutoDesk Civil 3D 2018. Then lines were drawn over the cracks which could be visibly observed (to about 1/16 inch). The resulting orthimages and cracks were saved as a drawing. An example of the crack mapped drawing is shown in Figure 13 below.

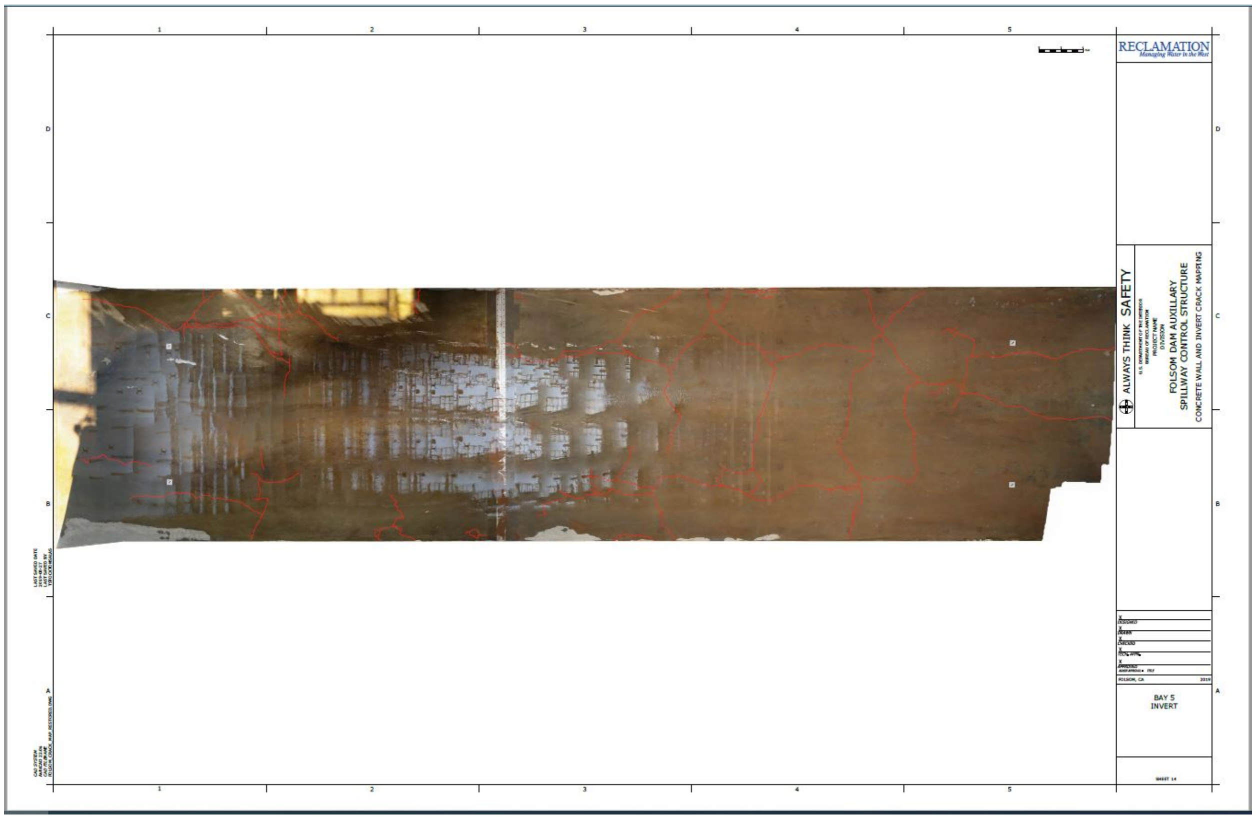


Figure 13. —Example crack map drawing with cracks in red.

However, the deep learning method required the orthoimage and exported image format of the cracks both in the same size so that the crack image could be layered over the orthoimage as representative of the cracks. The crack image was referred to as the binary orthoimage label mask. While the orthoimage was exported from the photogrammetric software as an image file, the crack mask was exported from the CAD software in an image format with the same dimensions or resolution as the orthoimage.

Data Sorting

The initial deep learning process was to identify images with cracks but was changed to identify the cracks in the images. In the case of identifying images with cracks, the orthoimages would need to be divided into small segments and the computer would determine if there was a crack in each segment and would categorize the segments as "crack" or "no crack". (This methodology is still being considered as preliminary step to reduce processing time, see the Discussion section for more details).

In order to train the computer to identify the segments "crack" or "no crack", it would need to be exposed to a training set of data. This data would need to be sorted into the two folders. This step would be time consuming, since the data would need to be manually sorted into appropriate folders, so a "Sorter" program was written at the TSC to help reduce the sorting time. The program was not completely automated but reduced time required by displaying an image that the operator could mark as "crack" or "no crack" and use a mapped keyboard input to place the image in the appropriate folder and advance the display to the next image. The program was built in Microsoft Visual Basic and controlled Microsoft Photos to display the image and used keyboard shortcuts to drop the image into the predetermined folder path. However, as mentioned above, a different approach was utilized that didn't require the Sorter program after all.

Processing

All the orthoimages were in a 3-channel RGB (red green blue) *.tif format typical of raster data. This data is easily read into Python using libraries such as rasterio along with Python's defacto array library: numpy [19] [20]. Each orthoimage has an accompanying concrete crack mask file which is in a *.jpeg format. For the purpose of this project the *.jpeg crack mask file was converted into a binary mask of "crack or "no crack" using a Boolean array (true/false). The orthoimage and the binary label mask are of the exact same size and thus every pixel has a discrete label. It is important to note that the orthoimages are 8-bit images with pixel values in each of the 3 channels ranging from 0-255. No preprocessing in terms of exposure, color correction, etc. was utilized. In order to be used in a deep learning pipeline capable of being trained on a laptop, the orthoimages and accompanying label masks had to be reduced to 256- by 256-pixel patches, which is a customary input size for U-nets. An example of the orthoimage for the Bay 6 Invert along with its accompanying binary label mask is displayed in Figure 14 and 15 below:

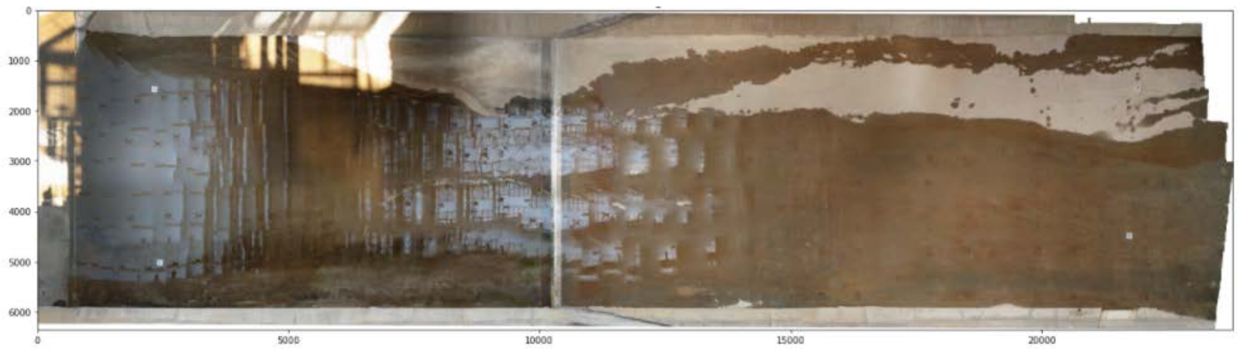


Figure 14. —Original Bay 6 Invert orthoimage

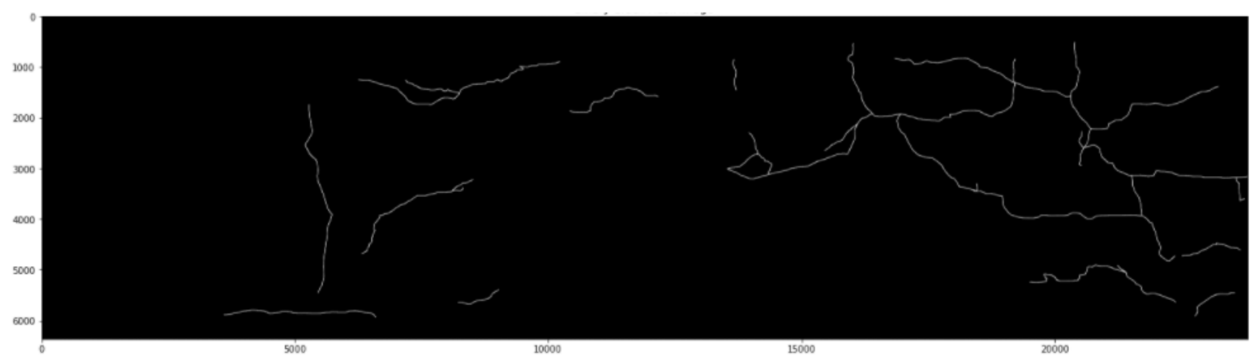


Figure 15. —Original Bay 6 Invert binary label mask

The orthoimages and label masks were extracted into 256- by 256-pixel image patches using n-dimensional array slicing of non-overlapping portions. Originally this resulted in approximately 15,000 patches; however, due to a massive class imbalance with less than 1 percent of the pixels labeled as “crack”, the extraction process had to be modified to only extract 256- by 256-pixel patches that contained at least 200 pixels labeled as “crack”. This led to a significant reduction in the number of patches to approximately 2,000.

The preprocessing step which included some manual cropping of the orthoimages and binary mask labels along with the extraction process reduced the 18 original orthoimages to 15 orthoimages and label masks suitable for use in the deep learning pipeline. The 15 orthoimages filenames, the binary label mask filenames, their y, x pixel size and the number of extracted pixel patches are summarized in Table 1 below.

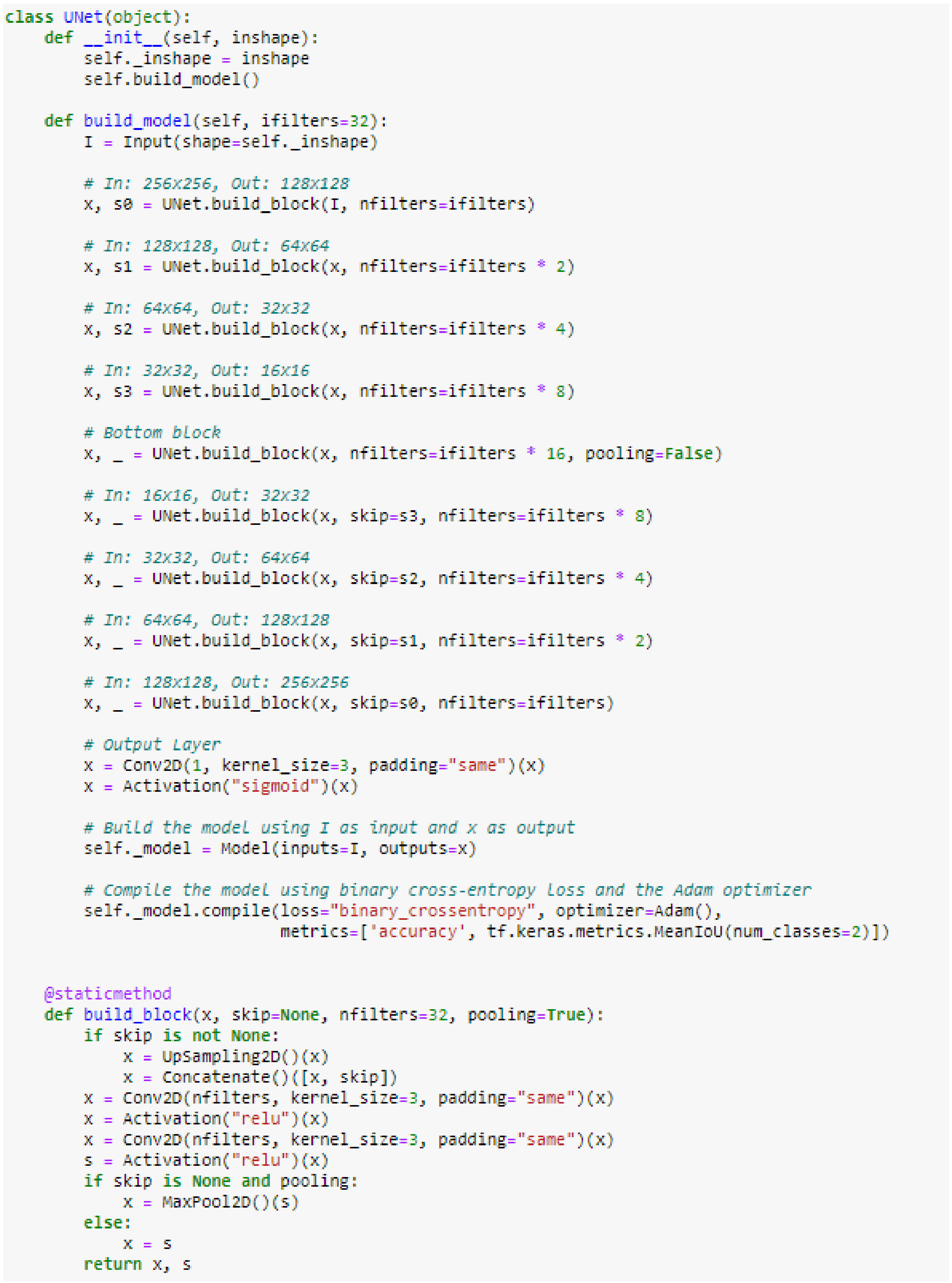
Table 1. —The organization of the 15 orthoimages and labeled binary crack masks used in creating the deep learning pipeline along with the number of non-overlapping 256- by 256-pixel patches extracted from each.

Number	Orthoimage Filename	Mask Filename	Orthoimage Size (y, x pixels)	Number of extracted patches
1	Bay 1 North.tif	B1N.jpg	7959, 23849	103
2	Bay 1 South.tif	B1S.jpg	5512, 18750	95
3	Bay 2 North.tif	B2N.jpg	8411, 24250	147
4	Bay 2 South.tif	B2S.jpg	7664, 23341	99
5	Bay 3 North.tif	B3N.jpg	6723, 20195	112
6	Bay 3 South.tif	B3S.jpg	5871, 20840	52
7	Bay 3 Invert.tif	B3I.jpg	6052, 21601	26
8	Bay 4 North.tif	B4N.jpg	5215, 15553	45
9	Bay 4 South.tif	B4S.jpg	7366, 25500	80
10	Bay 5 North.tif	B5N.jpg	5121, 16290	67
11	Bay 5 South.tif	B5S.jpg	6397, 21590	80
12	Bay 5 Invert.tif	B5I.jpg	5928, 22872	365
13	Bay 6 North.tif	B6N.jpg	6748, 22760	218
14	Bay 6 South.tif	B6S.jpg	8435, 36369	96
15	Bay 6 Invert.tif	B6I.jpg	5858, 23606	265

Once the matching pairs of 256- by 256-pixel image patches were extracted they were written out as new individual picture files as *.jpegs split into training and test datasets. The training dataset included all the extracted patches except patches from Bay 5 North, South and Invert. The extracted patches from Bay 5 North, South and Invert were written to a test dataset folder as to fully exclude the Bay from training to limit picture autocorrelation effects. It is important to note that this led to 1,283 image patches in the training dataset and 509 image patches in the test dataset. The training to test dataset split was 72% training and 28% test. While this is a fairly low number of non-overlapping 256- by 256-pixel image patches extracted for each of the training and test datasets, it was deemed sufficient for a proof-of-concept U-net to demonstrate the potential of deep learning.

In order to build an original U-net, the deep learning library Tensorflow was used – specifically, Tensorflow 2.0 with the integrated Keras library from Google [21]. It is important to note that PyTorch or MXNet could have also been used as the deep learning methodology as it is not locked into or particular to a specific deep learning library in Python. Furthermore, a different programming language could have been used such as C++ or Julia to achieve a similar outcome.

For the sake of reproducibility, the class structure of the U-net model used for this proof-of-concept effort is provided in Figure 16 below. The majority of this class code was modified from commonly available non-original code available on the internet in various locations.



```python
class UNet(object):
    def __init__(self, inshape):
        self._inshape = inshape
        self.build_model()

    def build_model(self, ifilters=32):
        I = Input(shape=self._inshape)

        # In: 256x256, Out: 128x128
        x, s0 = UNet.build_block(I, nfilters=ifilters)

        # In: 128x128, Out: 64x64
        x, s1 = UNet.build_block(x, nfilters=ifilters * 2)

        # In: 64x64, Out: 32x32
        x, s2 = UNet.build_block(x, nfilters=ifilters * 4)

        # In: 32x32, Out: 16x16
        x, s3 = UNet.build_block(x, nfilters=ifilters * 8)

        # Bottom block
        x, _ = UNet.build_block(x, nfilters=ifilters * 16, pooling=False)

        # In: 16x16, Out: 32x32
        x, _ = UNet.build_block(x, skip=s3, nfilters=ifilters * 8)

        # In: 32x32, Out: 64x64
        x, _ = UNet.build_block(x, skip=s2, nfilters=ifilters * 4)

        # In: 64x64, Out: 128x128
        x, _ = UNet.build_block(x, skip=s1, nfilters=ifilters * 2)

        # In: 128x128, Out: 256x256
        x, _ = UNet.build_block(x, skip=s0, nfilters=ifilters)

        # Output Layer
        x = Conv2D(1, kernel_size=3, padding="same")(x)
        x = Activation("sigmoid")(x)

        # Build the model using I as input and x as output
        self._model = Model(inputs=I, outputs=x)

        # Compile the model using binary cross-entropy loss and the Adam optimizer
        self._model.compile(loss="binary_crossentropy", optimizer=Adam(),
                            metrics=['accuracy', tf.keras.metrics.MeanIoU(num_classes=2)])


    @staticmethod
    def build_block(x, skip=None, nfilters=32, pooling=True):
        if skip is not None:
            x = UpSampling2D()(x)
            x = Concatenate()([x, skip])
        x = Conv2D(nfilters, kernel_size=3, padding="same")(x)
        x = Activation("relu")(x)
        x = Conv2D(nfilters, kernel_size=3, padding="same")(x)
        s = Activation("relu")(x)
        if skip is None and pooling:
            x = MaxPool2D()(s)
        else:
            x = s
        return x, s
```

Figure 16. —The U-net Python code utilizing the Keras API of Tensorflow 2.0 for the proof-of-concept effort

It is critical to note that the original implementation of U-net used Convolution Transposing rather than UpSampling; however, research since the original publication of U-net has demonstrated that convolution transposing can introduce artifacts into the output predictions of the U-net, and thus it has been ever so slightly modified to use UpSampling instead.

The U-net model was trained on an 8 GB (gigabyte) EVGA Nvidia GeForce GTX 1070 GPU (graphics processing unit) card using a batch-size of 32 image patches for approximately 128 epochs, which took only about two hours to train.

It is not practical to fully include every code block utilized in this proof-of-concept effort. Therefore, the entire workflow has been preserved in Jupyter Notebooks which is a common application for evaluating and archiving Python code and can be provided upon request.

After the U-net training was completed, a Python visualization utility like Matplotlib was utilized to assess the performance of the proof-of-concept U-net by direct comparison [22].

Results

The proof-of-concept U-net described in this section was accessed both with accuracy metrics and through randomized visual inspection on both the training and test datasets.

The proof-of-concept U-net is a very crude and preliminary effort and as such the accuracy metrics demonstrated here should be interpreted with the understanding that the datasets are too limited, they possibly contain autocorrelation bleed-through and more time should have been taken to rigorously test and demonstrate the outcomes provided here. Nonetheless, to describe what has been achieved, the accuracy metrics on the training and test datasets are provided.

The loss value is the outcome from the minimized objective or loss function at every iteration of training. In general, when training a deep learning neural network, the goal is to lower the loss in the training phase to demonstrate the neural network is first learning something, but then to also demonstrate that the neural network has not been overfitted when applied to a test set.

The accuracy metric is defined as the frequency with which the prediction of the neural network matches the mask labels. This metric is very sensitive to class imbalance problems.

The U-net was trained using a binary cross-entropy loss function, whose correct accuracy metric during the training process is the mean-Intersection over Union (mIoU) accuracy metric. For semantic segmentation, the mIoU accuracy metric is more informative as it penalizes and captures both false positive and false negatives in a positional informed manner and as such it has also been utilized in this proof-of-concept methodology. This is because mIoU seeks to directly quantify how much of a perfect overlap match the prediction is to the label. Generally, mIoU is a perfect match at 1 and becomes a poorer match or has less overlap as it approaches 0.

Once the proof-of-concept U-net was trained, the evaluate() function from Tensorflow was utilized to calculate the performance over the entire training or the entire test datasets depending on which was provided to the evaluate() function, and thus the metric scores provided are not based on randomly selected subsets. The loss, accuracy, and mIoU performance of the training and test datasets are shown in Table 2 below.

Table 2. —The loss, accuracy, and mIoU metric of the proof-of-concept U-net's performance on the training and test datasets

Dataset	**Loss**	**Accuracy**[1]	**mIoU**[1]
Training	0.0200	0.96	0.490
Test	0.5337	0.93	0.487

The final loss values provided demonstrate that the neural network at least during training can "learn" or "improve" and is important as a quality check. The final loss on the test dataset is also much higher than the training dataset likely demonstrating problems with the limited dataset as well as possible "overfitting" of the model to the training dataset again due to the limited amount of data processed.

The accuracy metric is perhaps the most misleading as the datasets have a class imbalance problem. The class imbalance of around 5 percent of the pixels identified as cracks means that this neural network could have a 0.95 accuracy by just always guessing that every pixel is "no crack". However, this also means that the expected behavior that the test dataset performs worse than the training dataset is correct and demonstrates that the neural network is learning.

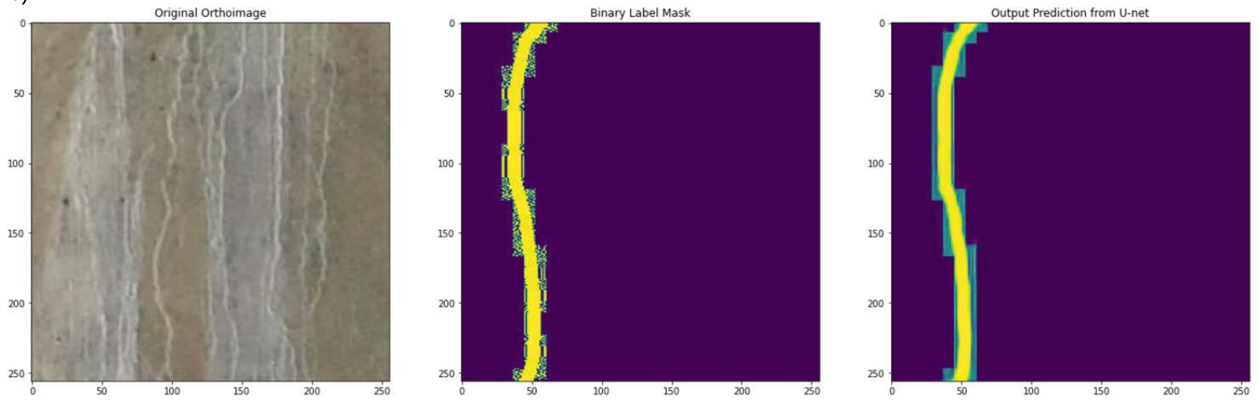
The mIoU value around 0.49 means that the prediction is only matching half the data. While this isn't a great result it does mean that the neural network is able to learn as shown by the low loss metrics and because the test dataset score is lower than the training dataset. This indicates that more data will improve the model and results.

A more representative way to access the current proof-of-concept U-net's performance is through visual inspection of randomly chosen (i.e. selected by a pseudo-random number generator) inputs/target pairs and to examine the proof-of-concept U-net's prediction. This was done for both the training and test datasets.

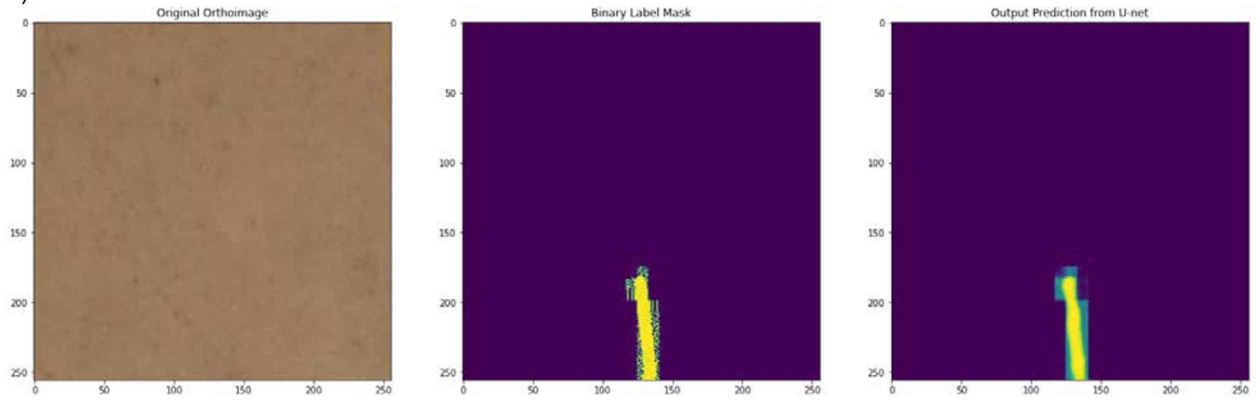
Four examples from each of the training and test datasets including the orthoimage input, binary label mask targets, and the output prediction are provided in sequence in Figures 17.1-4 and 18.1-4 below. The full selections of 32 orthoimage inputs, binary label mask targets, and the output prediction of the U-net are provided in Appendix A.

[1] The accuracy or mIoU performance stated is not representative of any type of "real-world" or even "academic-level" performance. The metrics are reported for transparency. The produced proof-of-concept U-net still needs significant work and improvement before a real product can be produced or evaluated.

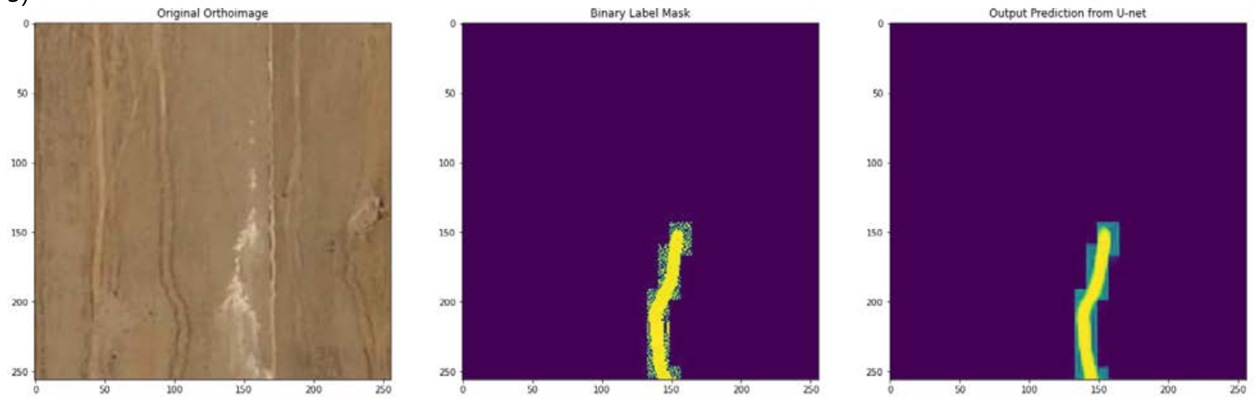
1)


Original Orthoimage
Binary Label Mask
Output Prediction from U-net

2)


Original Orthoimage
Binary Label Mask
Output Prediction from U-net

3)


Original Orthoimage
Binary Label Mask
Output Prediction from U-net

4)

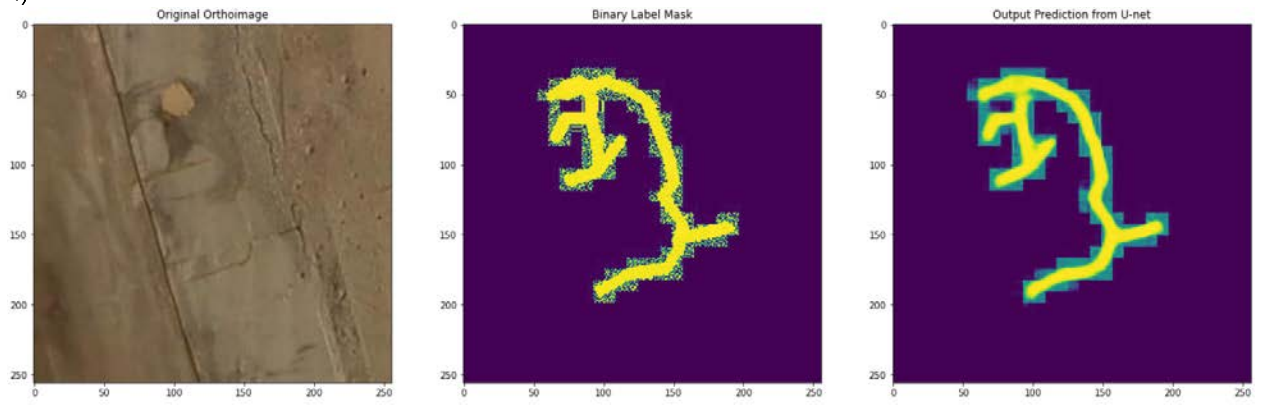
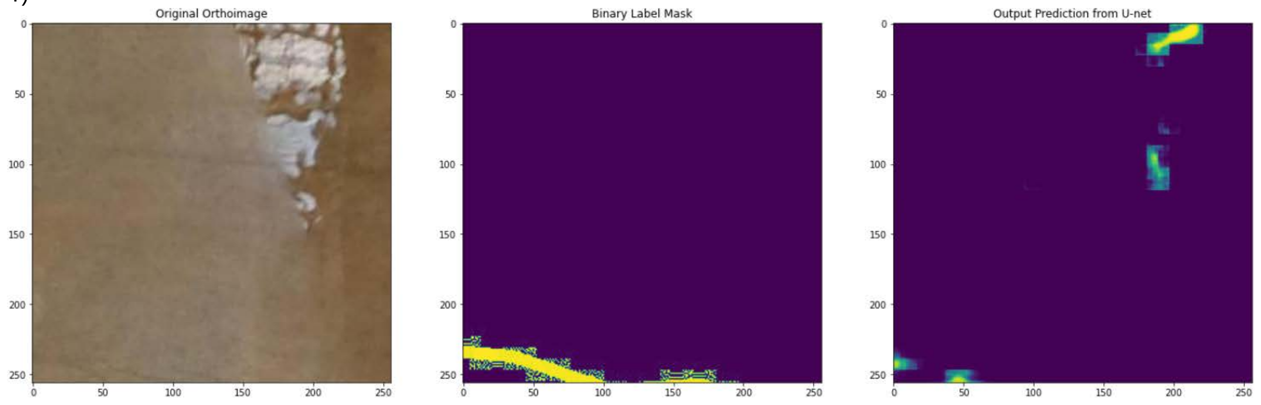
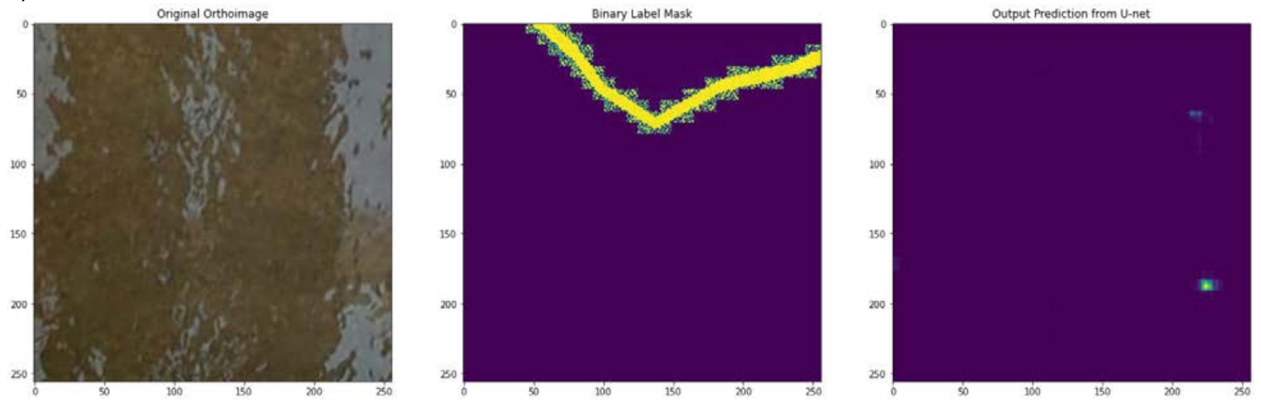


Figure 17.1 - 4. —Four examples from the **Training** dataset including the original orthoimage input (left), the binary label mask target (center), and the output prediction mask from the U-net (right) where "purple" is no crack detected and "yellow" is crack detected

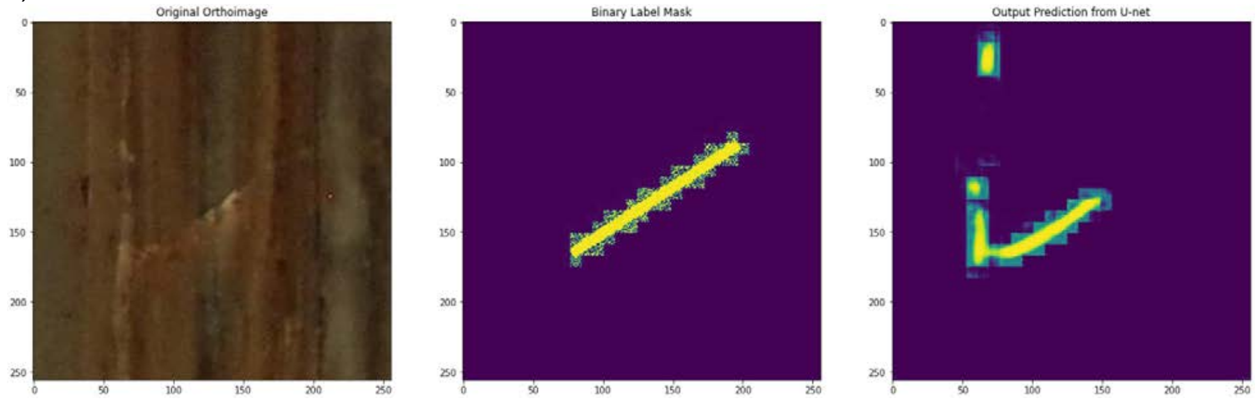
1)



2)



3)



4)

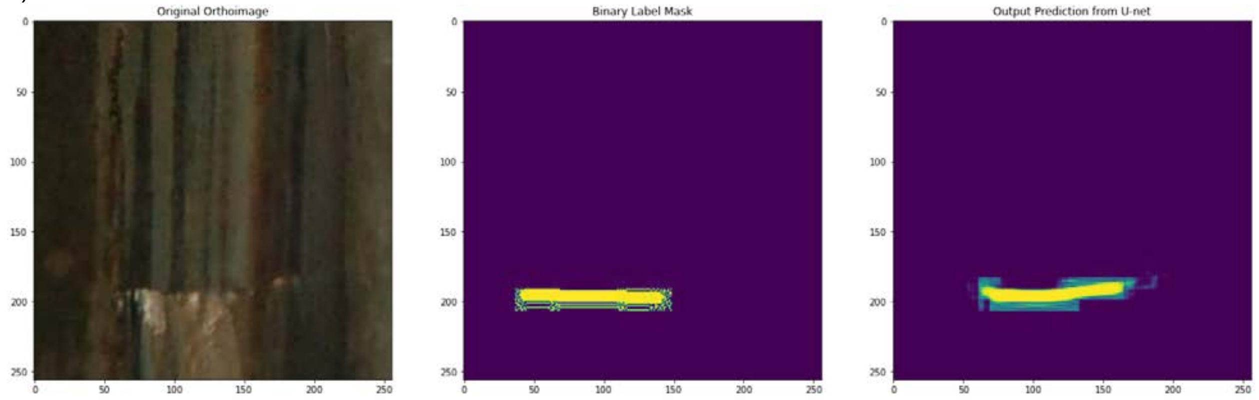


Figure 18.1-4. —Four examples from the **Test** dataset including the original orthoimage input (left), the binary label mask target (center), and the output prediction mask from the U-net (right) where “purple” is no crack detected and “yellow” is crack detected

Based on the current proof-of-concept U-net, the U-net is capable of learning as evident by the good visual performance on the training dataset. However, the performance on the test dataset is not as good as the training dataset and major improvements in generation will be necessary to have any type of future production system for automated concrete crack detection. The next section will highlight the path forward for enhancing this U-net methodology.

Discussion

The proof-of-concept U-net has demonstrated clear promise for future work to expand on. An outline of the next steps is discussed within this specific domain of deep learning applications for the task of detecting cracks in concrete structures such as dams. It should be highlighted that detecting cracks in concrete infrastructure is not a task limited to Reclamation as an agency and it is likely the future value and collaboration should be explored with other agencies with facilities like Reclamation such as the United States Corps of Engineers (USACE), water districts and other federal, state and local municipalities. In addition, this work may be expanded to agencies with other concrete infrastructure such as canals, bridges, freeways, etc.

Dataset Improvements

The single most important future improvements are related to direct and indirect expansions of the training, validation and test datasets. The currently used dataset only contained fifteen orthoimages at a single location. The next step would be to utilize additional orthoimages already collected and processed by the TSC UAS team as well as other UAS teams in the regions such as the Nebraska/Kansas, Oklahoma/Texas, Provo and Cascade Pacific Northwest (CPN) teams. This should expand not only the number of images and lighting conditions, but also the number of locations. Having a diverse dataset improves the capability of the deep learning neural network performance.

Once existing orthoimage data is further integrated into the existing dataset, the following suggested efforts should be explored:

1. Performing lighting and camera exposure corrections processes for decreasing the variance within the data samples either through post-processing or by enforcing stricter data collection methodologies that compensate for light magnitude and light direction.
2. Performing an extensive statistical analysis on the textures, shapes, and patch autocorrelation among the imagery and cracks available.
3. Establish a program and criterion for collecting additional concrete crack data such as deploying more UAS flights, use of non-aerial camera methods, and/or the integration of terrestrial and aerial LiDAR to create a 3D point cloud and surfaces of the structure.
4. Establish a review system for concrete crack labeling to be performed by a panel of independent engineers with expertise in detecting and categorizing concrete cracks. Each crack should be manually labeled and annotated and then reviewed. This labeled dataset can then be utilized to train deep learning methodologies. This type of labeling methodology is commonly used in life/safety-critical applications like biomedical image segmentation.
5. Collaborate with other significant partner agencies for dataset labeling, collection, and domain expansion.

A significant effort should also be made to sterilize the data enough to allow Reclamation to publish the dataset with an instructional paper and digital object identifier (DOI). If Reclamation releases a high-quality dataset and publicizes it, it may be that there will be significant interest and value derived as currently there are no publicly available concrete crack detection datasets for semantic segmentation.

Neural Network Structure Improvements

While improving performance of deep learning methodologies tends towards improvements of the dataset yielding the highest returns, there are several developments that can be made to the deep learning methodology itself. First the U-net itself is now a somewhat outdated architecture and newer architecture such as Mask R-CNN, Unet++, or the DeepLab family of networks are all freely available to be implemented in Tensorflow, PyTorch, MXnet, or Julia-based deep learning libraries. Substantial testing and investment by large tech companies such as Google and other research groups have demonstrated significant performance improvements over the original U-net implementation for semantic segmentation.

Additionally, U-net and the improved, newer architectures mentioned above all utilize an encoder component (i.e. the first downward part of the "U"). Instead of training this encoder from scratch like was done in this proof-of-concept U-net, a previously trained encoder could be utilized and implemented in what is termed "transfer learning". This process is quite simple and allows the neural network to start from previous image training efforts performed on massive image datasets that are simply impractical for Reclamation resources to reproduce, and thus by leveraging these previously trained encoders (which are also free/open-source) and undergoing a process of fine-tuning it is possible that performance improvements can be gained at a fairly low-cost.

Neural Network Training and Metric Improvements

The current methodology does not take advantage of any data augmentation methods or transforms which are commonly used in deep learning semantic segmentation tasks/pipelines. A range of data augmentation procedures should be explored and integrated into the deep learning workflow to enhance performance and generalization capability.

No validation dataset was provided separately during training. A validation dataset is separate from both the training datasets and the test datasets and is used to track the performance metric progress of the neural network as it trains and is often utilized to determine if to keep training or to halting training. It was decided to not provide a validation dataset due to the already limited size of the training dataset and the limited GPU computational resources available at the time of the project. Both of these factors can be easily enhanced at limited cost and would then facilitate the development of an integrated validation dataset to be used during training of the neural network.

The field of view provided to the neural network at the time of training and the time of prediction was an image patch of 256- by 256-pixels, while the orthoimage sizes where often in the 10,000+ pixel range. It was noted in back and forth conversations between the engineering labeling team and the deep learning team that often the engineering labeling team was utilizing the entire context of the orthoimages in order to first identify "true" concrete cracks and then to trace their complete shape/effect through the rather large orthoimage. This amount of context utilized by humans could simply not be provided to the neural network during training due to two important issues. First, the dataset is simply too small to allow for larger contexts as the number of instances would decrease in relation to the field of view size. Second, the GPU has a separate memory component than the RAM (random access memory) that the CPU has access to for the entire computer system. This GPU memory was limited to 6 of the available 8 GB during the training process utilized for this project due to other demands on the GPU such as generating the display. As such, increasing the field of view size was not realistic. Therefore future efforts should be made to expand Reclamation's GPU computational resources as well as increasing the dataset size so that the context size (i.e. the size of a single extracted image patch) can be increased to possibly 384 by 384, 512 by 512 or 1024 by 1024 as this would likely leading to improved performance as typically observed in studies for satellite imagery semantic segmentation [23].

Another possible improvement is the utilization of two separate but connected neural networks. The first neural network would determine if any part of an image patch contains a crack. For example, a simple "yes, the image patch contains a crack somewhere" or "no, the image patch doesn't contain a crack anywhere". Then if a crack is detected somewhere within the image patch the image patch could be sent to the second neural network which is trained/designed for semantic segmentation to determine the exact pixels that form the crack. This rather simple idea actually has three powerful components. First, cracks often span multiple connected/adjacent extracted image patches and it is likely that the probability of an adjacent patch also containing a crack is higher if a previously adjacent patch was detected as positive. Second, the occurrence of a concrete crack is an anomaly almost by definition as the balance of crack vs. no crack pixels is heavily weighted towards the no crack pixels bucket, and thus being able to structure a deep learning pipeline around this known fact will likely have improvements. In fact, evidence for this is already demonstrated in this proof-of-concept U-net work as there were limitations implemented in the training and test datasets to achieve an occurrence rate of crack pixels in the datasets to get the U-net to train initially. Thirdly, by using the two separate task neural networks, it is likely that the computation will be faster as the majority of the image will not need to be semantically segmented which takes longer to compute than a simple "yes" or "no" designation. (Note that this is different from the code that was developed to sort images as this sorting would be automated).

The last set of improvements pertain to changing and/or modifying the loss function, optimizer, learning rate, and accuracy metrics used. This part of a neural network can often lead to many different directions, but even simpler explorations in changing the loss function from a binary cross-entropy to a soft-dice loss would likely see a performance enhancement. Additionally, the choice of an Adam optimizer as opposed to a stochastic gradient descent (SGD) can have impacts on the learning rate, that is, how well the model can generalize and perform on unseen or out-of-sample data. Furthermore, the accuracy metrics all impact how progress is measured and made for overall improvements and a few simple experiments in this direction would also likely yield performance enhancements.

Visualizations and Post-Processing Improvements

Currently no visualization or post-processing methods were designed under the proof-of-concept U-net workflow. Future efforts should be made to produce visualization and post-processing improvements such as forward and backward visual reconstruction of the complete orthoimage after prediction; the use of image patch cropping so that only the center of an image patch (i.e. 128 by 128 of a 256 by 246 image) is utilized from the prediction (as predictions are more accurate at the center of the patch); and the development of an interactive workflow in Jupyter Notebooks or internal web platform to allow very simple processing of orthoimages once the neural network is trained. These improvements will move a trained neural network from a research product to actually being used in production/deployment environments by end-users. Furthermore, nothing prevents the deployment of the trained neural network onto smartphone devices or tablets, which could even be utilized by the collectors of the original data to guide additional flight paths or to highlight areas where higher resolution data should be collected.

Conclusion and Summary

This research made significant progress in producing a deep learning pipeline that is capable of detecting or identifying concrete cracks within Reclamation structures such as dams. However, this project is only the beginning of a very powerful and economically valuable tool that could reduce future costs of concrete monitoring efforts, while enhancing the safety and maintenance procedures around concrete infrastructure. Additionally, Reclamation stands well poised to capitalize on an integrative and collaborative approach with other agencies for expanding a future deep learning method to concrete canals, bridges, roads, highways, and buildings. The following conclusions are made:

- Deep learning crack identification proof-of-concept was successfully demonstrated using previously collected UAS data though with limitations.
- Methods should be developed to collect and produce optimal data for deep learning datasets including strict controls on the sensor exposure and image output as discussed in the Dataset Improvements section.
- The training, validation, and test sets should be expanded with diverse examples to improve the accuracy of the identification. These examples could include more previously collected UAS data as well as new UAS data collected specifically for deep learning. This data collection effort should include UAS teams from across Reclamation and other agencies that have similar infrastructure.
- Datasets should include more cracks to improve crack recognition.
- Additional work should be conducted to improve the confidence in the testing method. In addition, research should be conducted to reassemble the crack segments or chips into the crack mapped orthoimage as the final data product.
- Other neural net structures should be investigated to improve accuracy and reduce processing times.
- Deep learning crack identification hardware should be upgraded with GPU with greater memory and faster processing.
- Formal crack mapping methods and review processes should be discussed and applied to gage the effectiveness and accuracy of the crack map.
- Third-party automated crack mapping solutions should be further investigated once they become more established, however, having a Reclamation database for evaluation will be more accurate.
- Crack mapping processes should be expanded to evaluate the change of a crack over time.

Data

The data used in this project can be made available upon request.

References

[1] American Concrete Institute, *ACI Manual of Concrete Inspection, ACI SP-2(07),* Farmington Hills: ACI, 2007, p. 87.

[2] M. Klein, "Unmanned Aerial System (UAS) Data Collection at Reclamation Sites," Bureau of Reclamation, Denver, 2019.

[3] 3DR Inc, "3DR Solo," Google, 1 September 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.o3dr.solo.android&hl=en_US. [Accessed 18 September 2019].

[4] Ricoh Imaging Company, LTD, "GR II," NA. [Online]. Available: http://www.ricoh-imaging.co.jp/english/products/gr-2/spec/index.html. [Accessed 17 September 2019].

[5] M. Klein, "Photogrammetric Analysis of the Left Gate Abutment at Vallecito Dam," Bureau of Reclamation, Denver, 2018.

[6] J. Kottenstette, "Use of Photogrammetric Measurements in a Concrete Damage Survey: Guernsey Dam South Spillway," in *Biennial Geotechnical Seminar 2012*, Denver, 2012.

[7] E. Schlosser, "Seminoe Dam Photo-Mapping and Photogrammetry Report," Bureau of Reclamation, Denver, 2013.

[8] M. Klein, "Elephant Butte Dam UAS Inspection, ST-2018-1817-01," Bureau of Reclamation, Denver, 2018.

[9] M. Klein, "Deformation Monitoring using Unmanned Aerial Systems (UAS) and Photogrammetry - Seminoe Dam; DSO-2019-03," Bureau of Reclamation, Denver, 2019.

[10] K. Bartojay, "W.H. Brock Reservoir Soil-Cement Crack and Damage Mapping," Bureau of Reclamation, Denver, 2019.

[11] M. Klein and J. Prickett, "Site Visit to Folsom Auxiliary Spillway to Collect UAS Data for Crack Mapping," Bureau of Reclamation, Denver, 2019.

[12] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proceedings of the 18th International Conference on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - Volume 9349*, Berlin, 2015.

[13] T. Liabacher, T. Weyde and S. Jalali, "M2U-Net: Effective and Efficient Retinal Vessel Segmentation for Real-World Applications," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Long Beach, 2019.

[14] K. He, G. Gkioxari, P. Dollar and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, 2017.

[15] Z. Zhou, M. R. Siddiquee, N. Tajbakhsh and J. Liang, "UNet++: A Nested U-Net Architecture for Medical Image Segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, Granada, 2018.

[16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 40, no. 4, pp. 834-848, 2017.

[17] Thornton Tomasetti, "T2D2," Thornton Tomasetti, [Online]. Available: https://www.thorntontomasetti.com/capability/t2d2. [Accessed 20 September 2020].

[18] Thornton Thomasetti, "CORE Studio," [Online]. Available: https://www.thorntontomasetti.com/core-studio. [Accessed 20 September 2020].

[19] S. Gillies and others, "Rasterio: geospatial raster I/O for Python programmers," NA, 2013. [Online]. Available: https://github.com/mapbox/rasterio. [Accessed 28 September 202].

[20] T. E. Oliphant, "Python for Scientific Computing," *Computing in Science & Engineering,* vol. 9, no. 3, pp. 10-20, 2007.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng and Google Brain, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX Symposium on Operating System Design and Implementation (OSDI '16)*, Savannah, 2016.

[22] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering,* vol. 9, no. 3, pp. 90-95, 2007.

[23] V. Iglovikov, S. Seferbekow, A. Buslaev and A. Shvets, "TernausNetV2: Fully Convolutional Network for Instance Segmentation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Salt Lake City, 2018.

Glossary

Adam optimizer. The Adam optimizer is a very specific and complicated algorithm which differentiates itself from a standard optimizer like "stochastic gradient descent" by using an adaptive learning rate. The Adam optimizer works specifically with cross-entropy. It determines how the neural network model attempts to lower the specific loss function in every round of training by adjusting weights, parameters, and/or coefficients.

Binary. A separation of two classes. For example, heads and tails is a binary outcome of flipping a coin. Or more commonly used in computer science, binary is 0 or 1, true or false. In the case of deep learning models, it is the classification of the objective, that is, a pixel is a "crack" or "not a crack".

Convolution Transposing. Convolution Transposing is routine layer used to upsample the input by allowing all adjacent cells influence the expanded cells.

Cross-entropy. Cross-entropy is a type of loss function commonly used in machine/deep learning which essentially calculates the total entropy difference between a set of distributions. Cross-entropy is specifically used with an Adam optimizer. This should not be confused with KL (Kullback-Leibler) divergence, logistic loss, or log loss.

Learning rate. In deep learning, a learning rate is defined as the magnitude of the step an optimizer takes in each round of training. The learning rate impacts the generalization and performance of the final neural network produced.

Loss function. A loss function is the term used in deep learning for the name of the objective function that is being minimized.

Optimizer. An optimizer is the algorithmic routine of how the model attempts to optimize or minimize the objective function or in this case the "loss function".

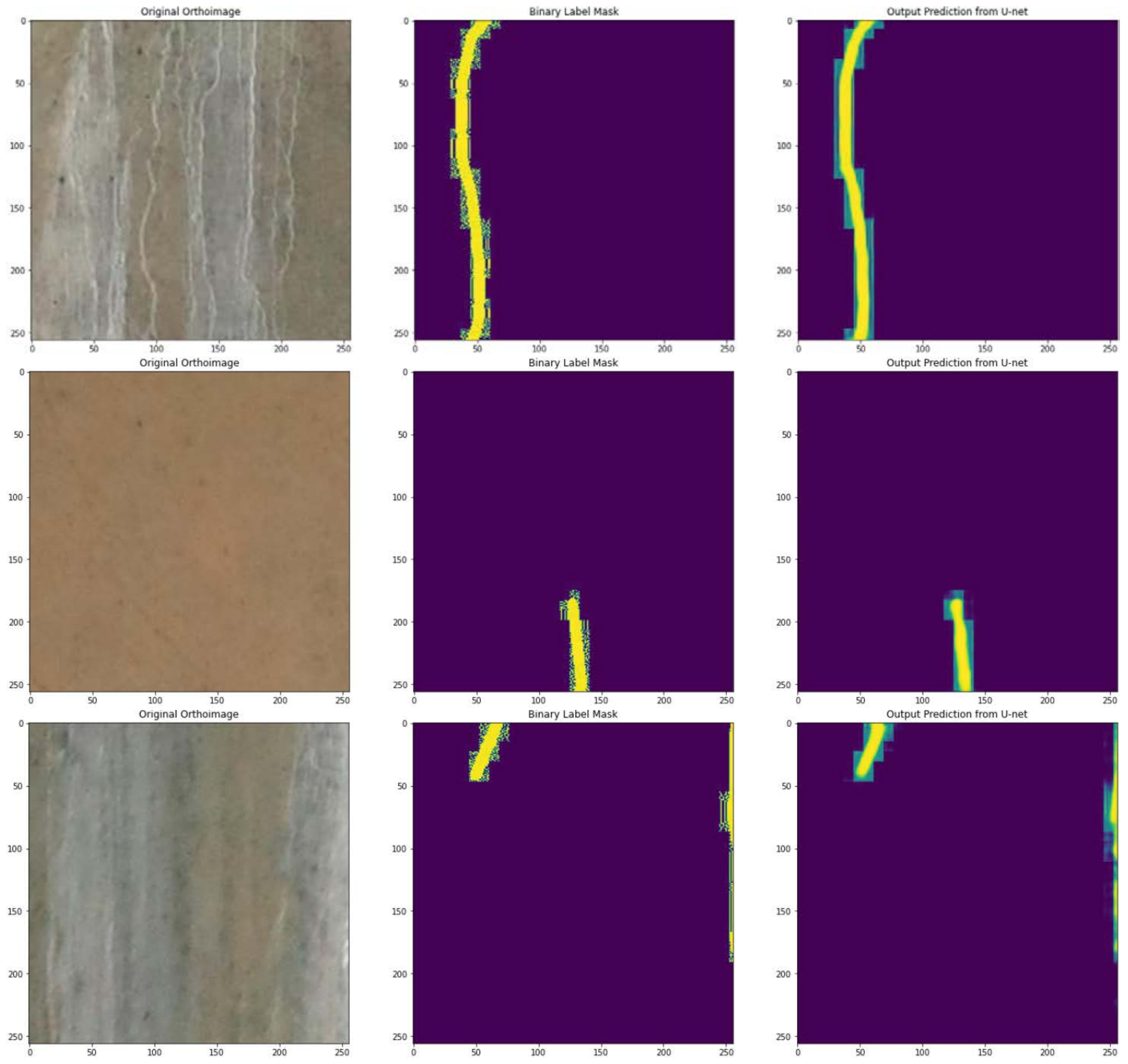
Soft-dice loss. Soft-dice loss is the dice-loss metric that has been modified and approximated to make it differentiable for use as a loss function rather than a pure metric. The soft-dice loss function sometimes performs better than cross-entropy for serving as the objective function to be minimized rather than an evaluation metric of performance.

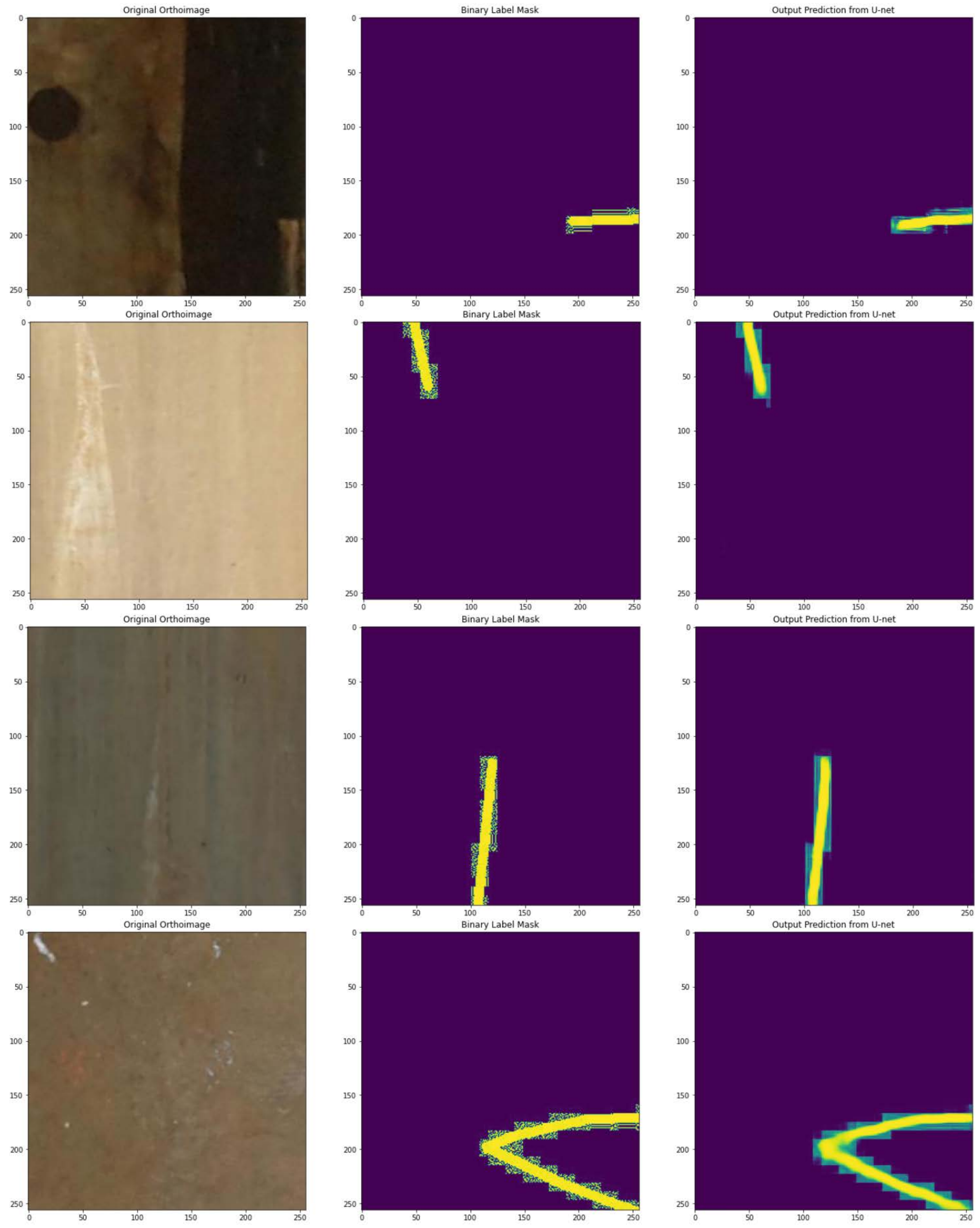
Stochastic gradient descent (SGD). SGD is the default optimization routine for deep learning neural networks.

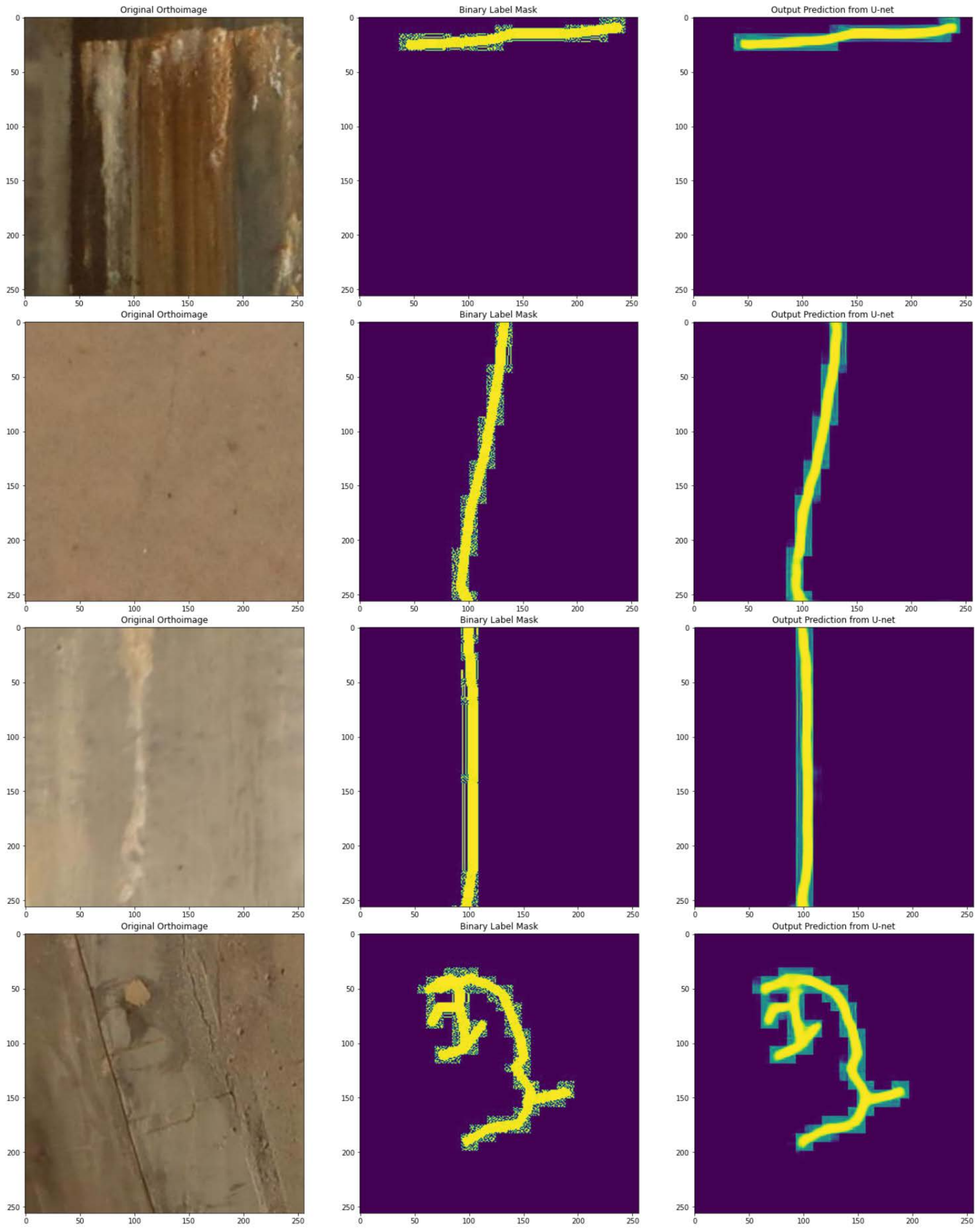
Upsampling. Upsampling is a generic routine that increases the dimensions of the input.

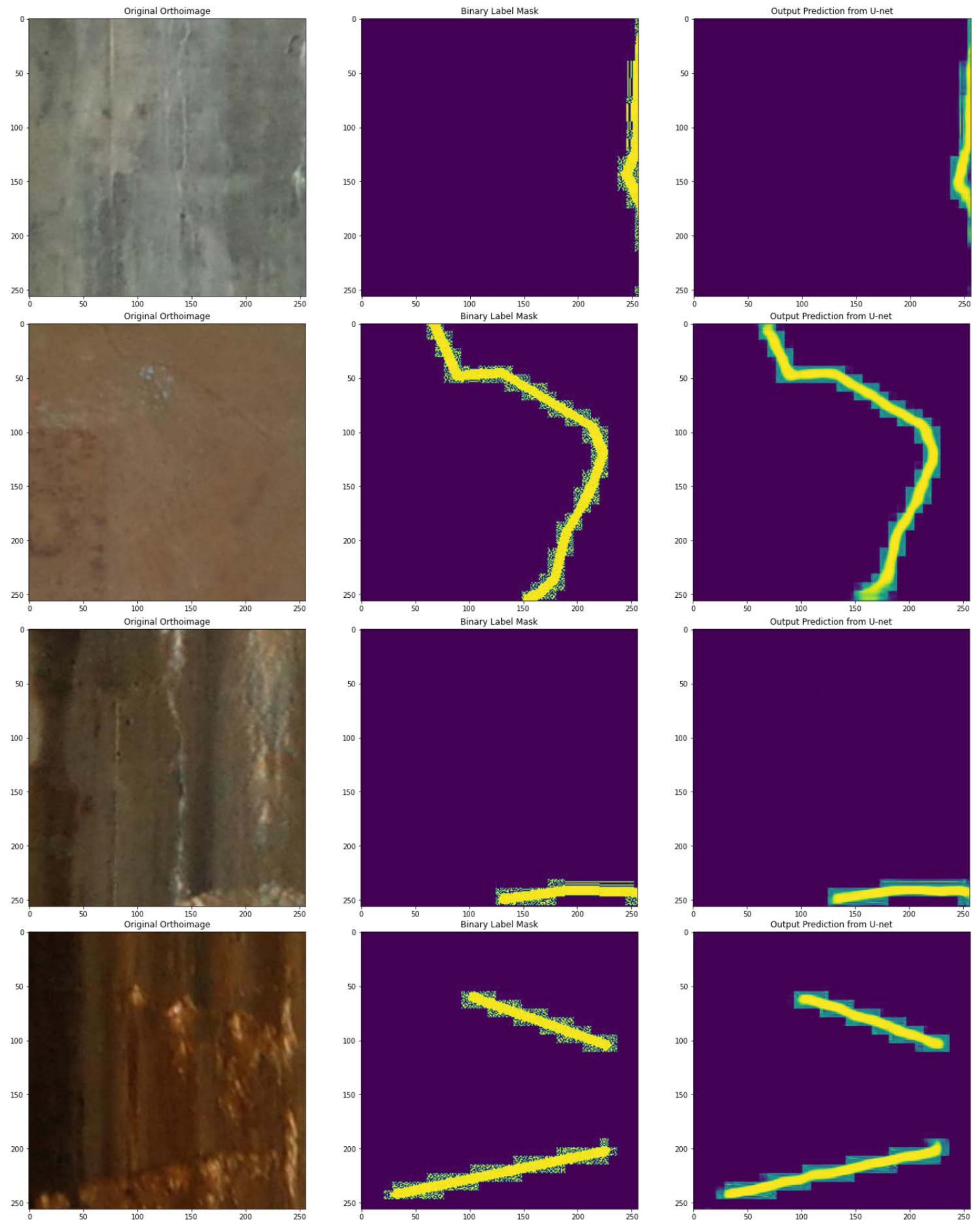
Appendix A – U-net Performance Verification Samples

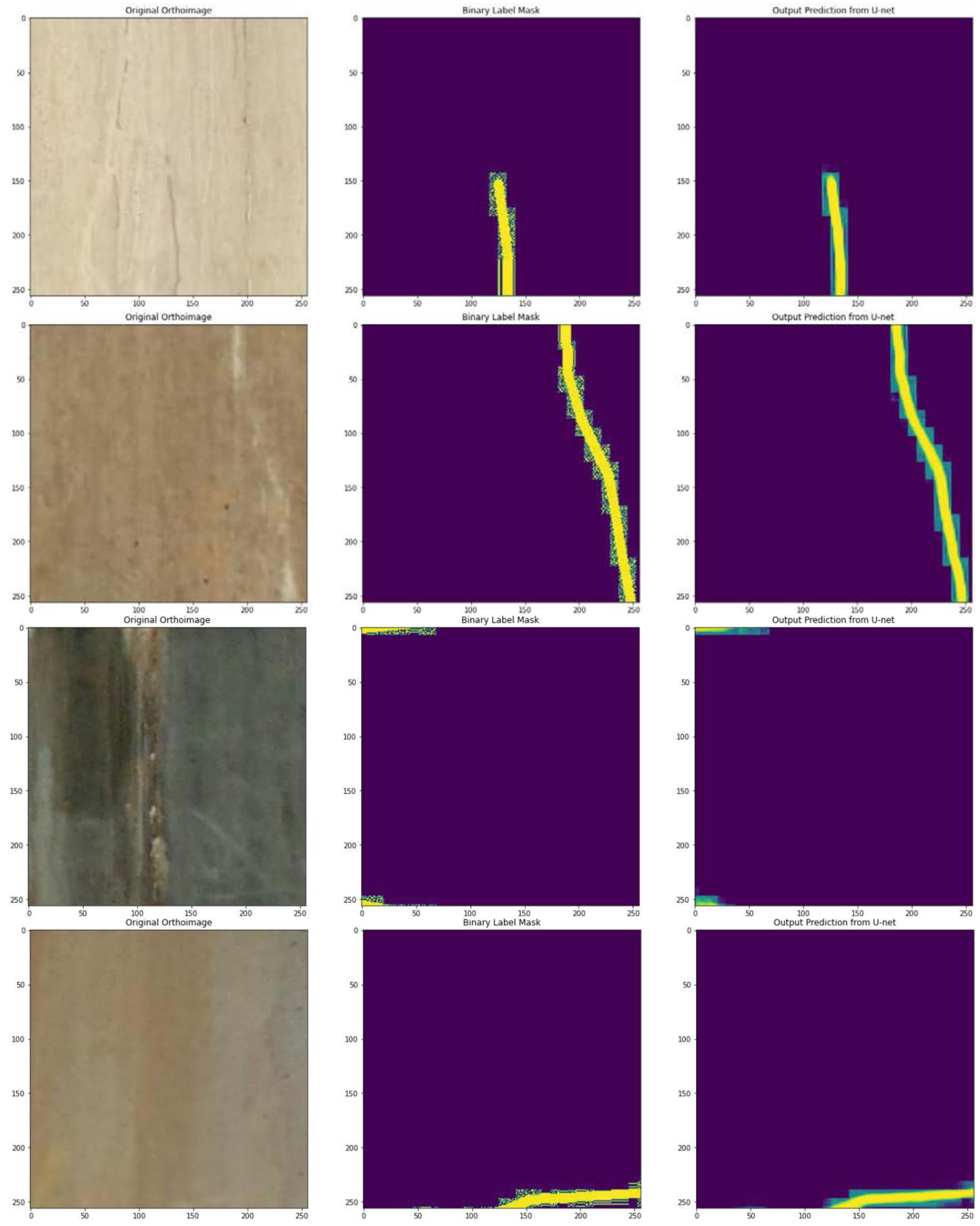
Thirty-two triplet image groups from the **Training** Dataset, which were randomly chosen. Each triplet group is composed of the original orthoimage input (left), the binary label mask target (center), and the output prediction mask from the U-net (right) where “purple” is no crack detected and “yellow” is crack detected.

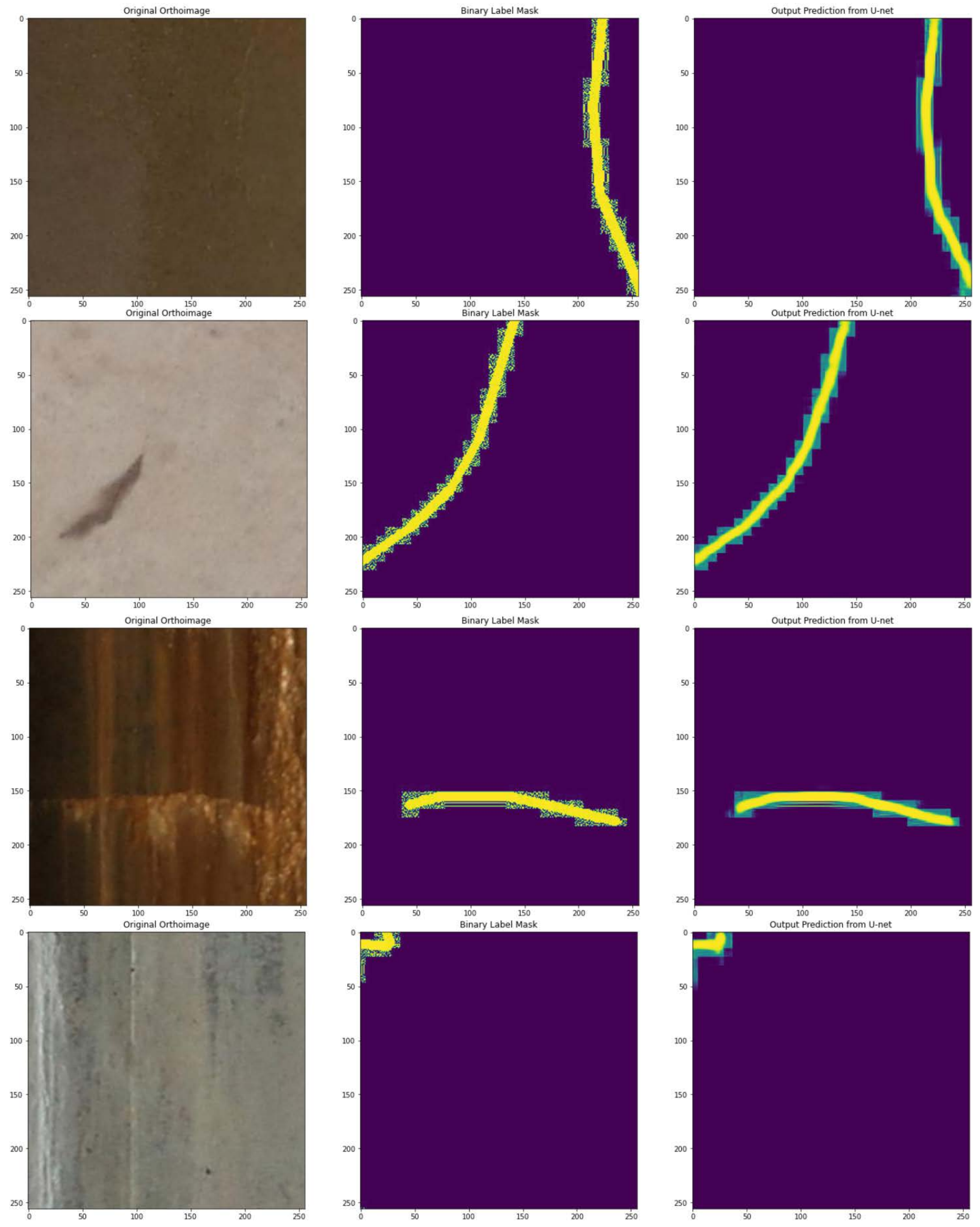


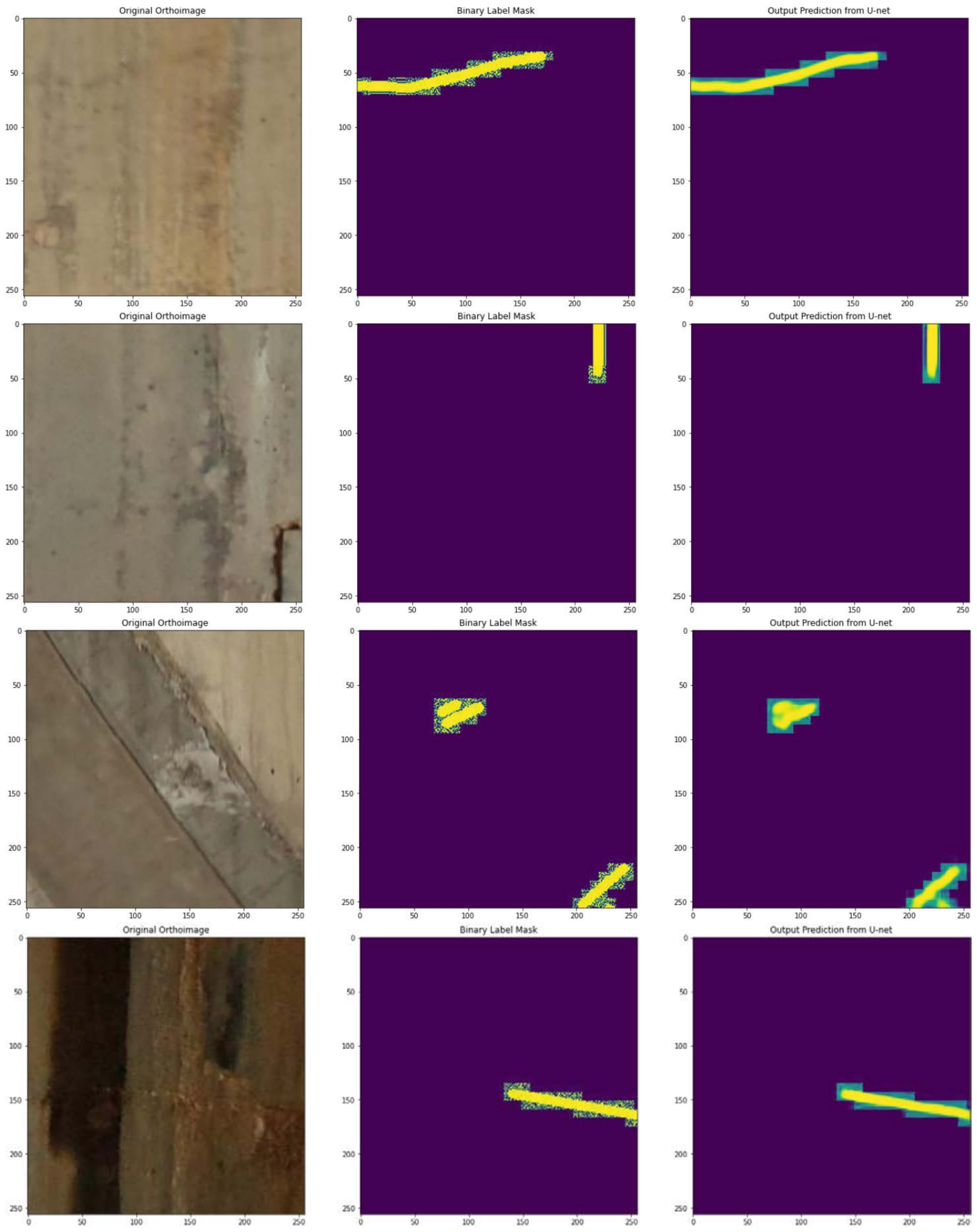

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net

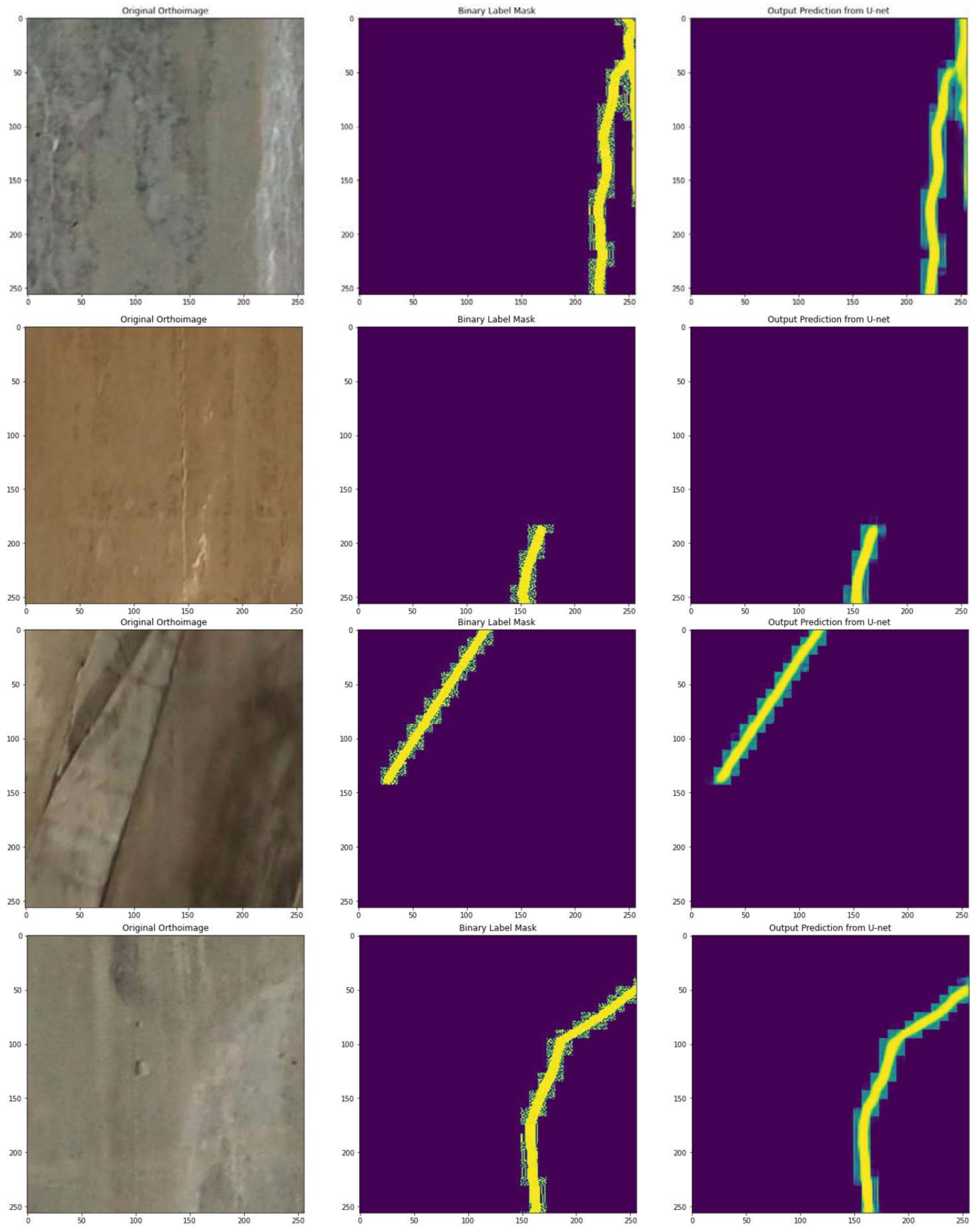

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250

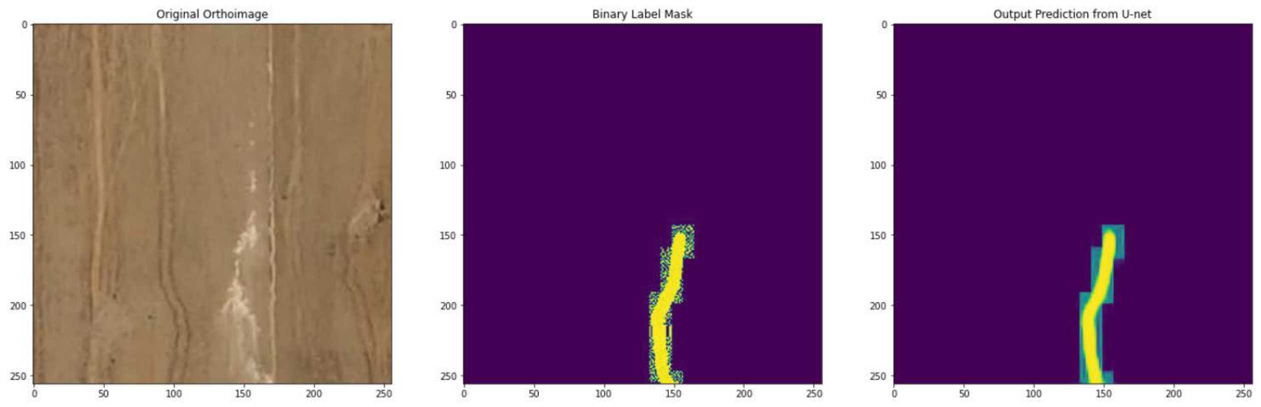

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250


Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250

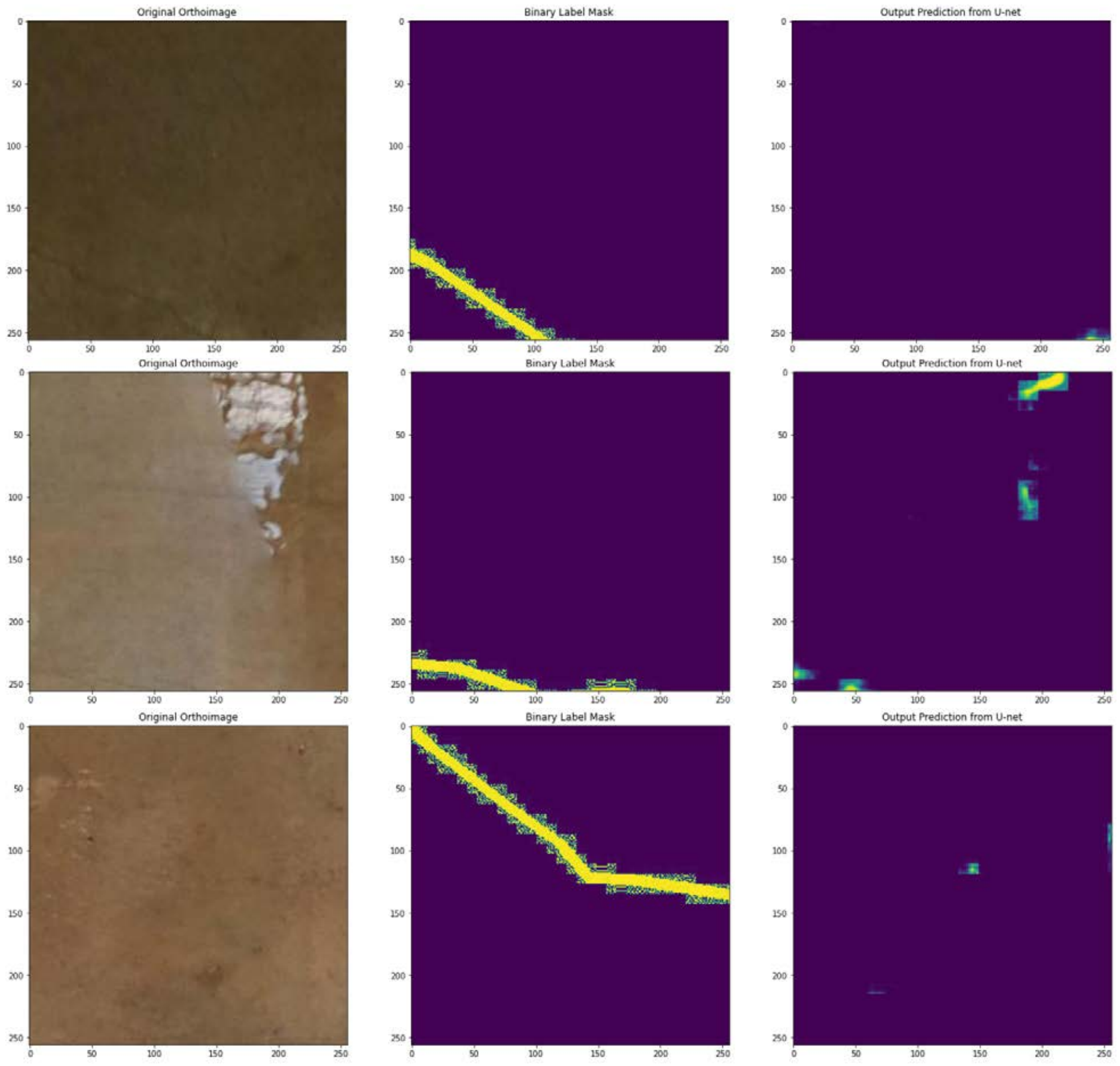

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net

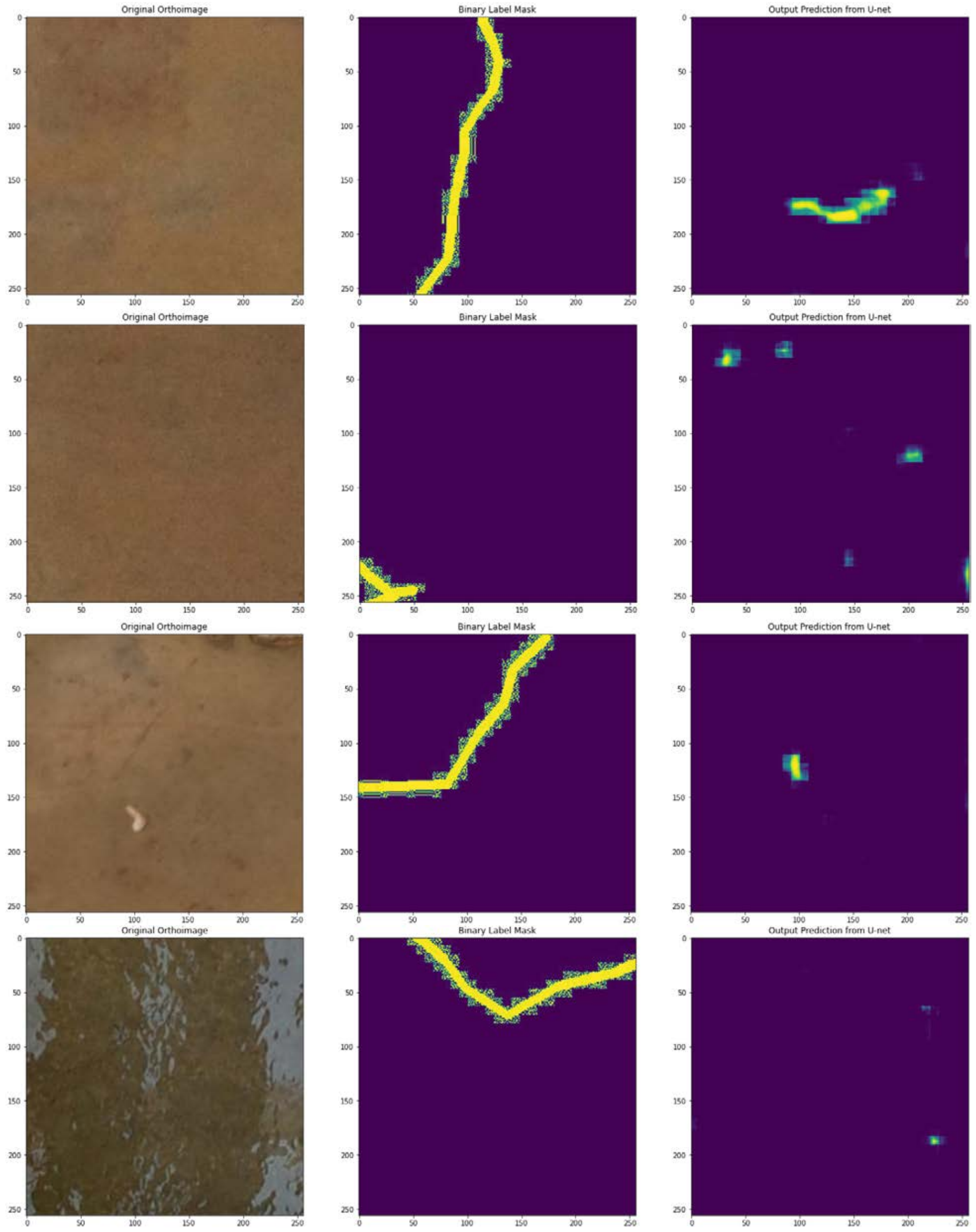

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250

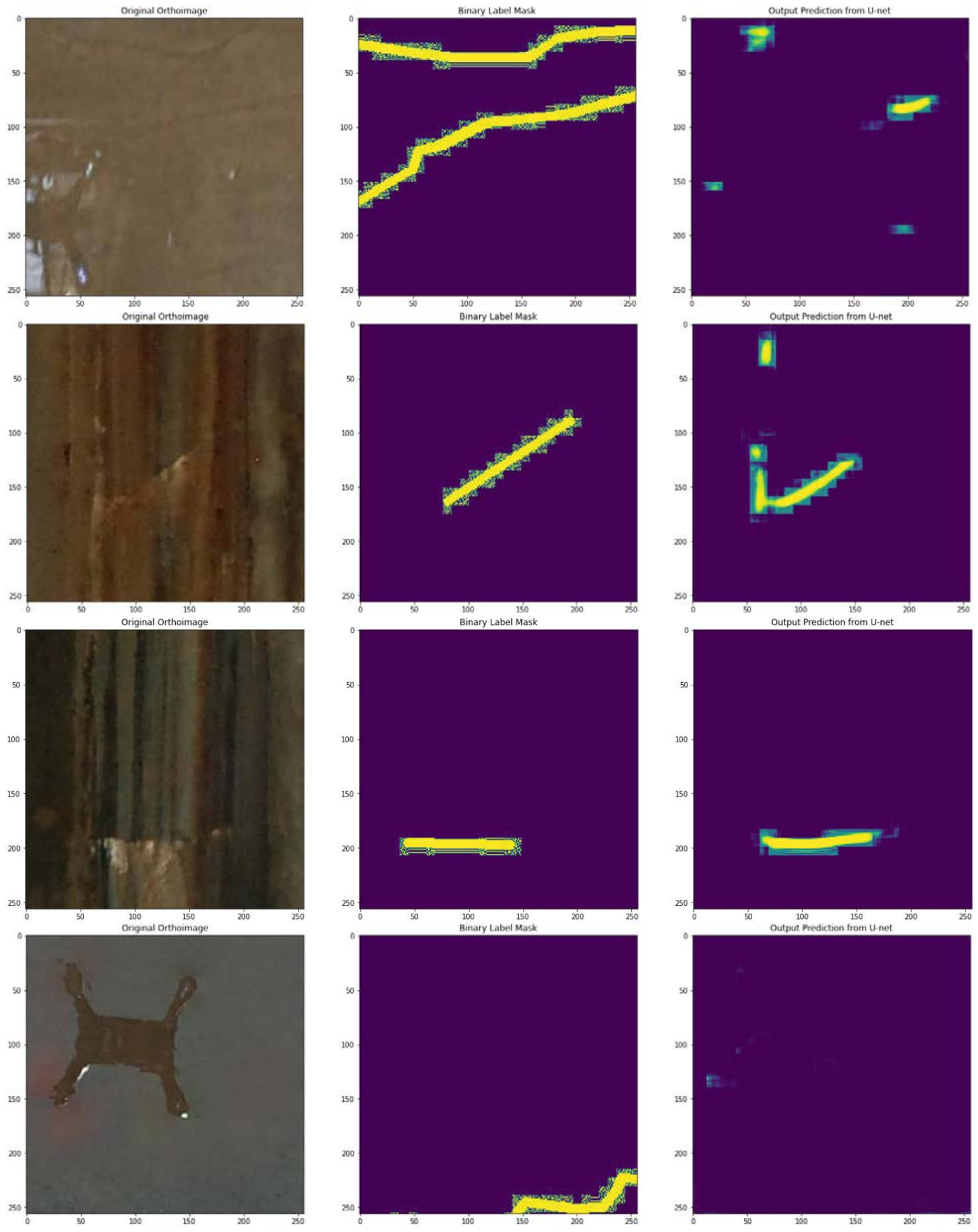

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250

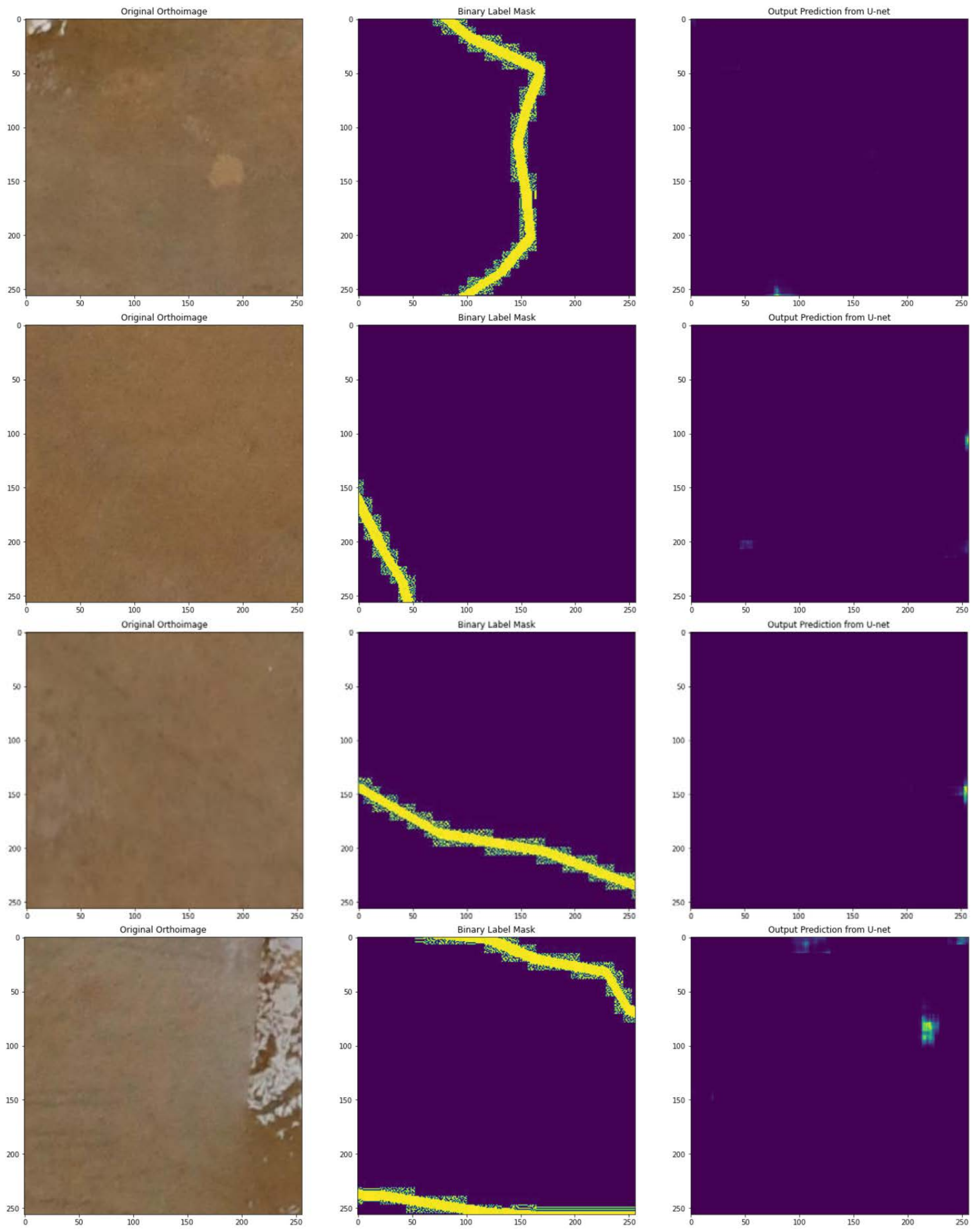

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250

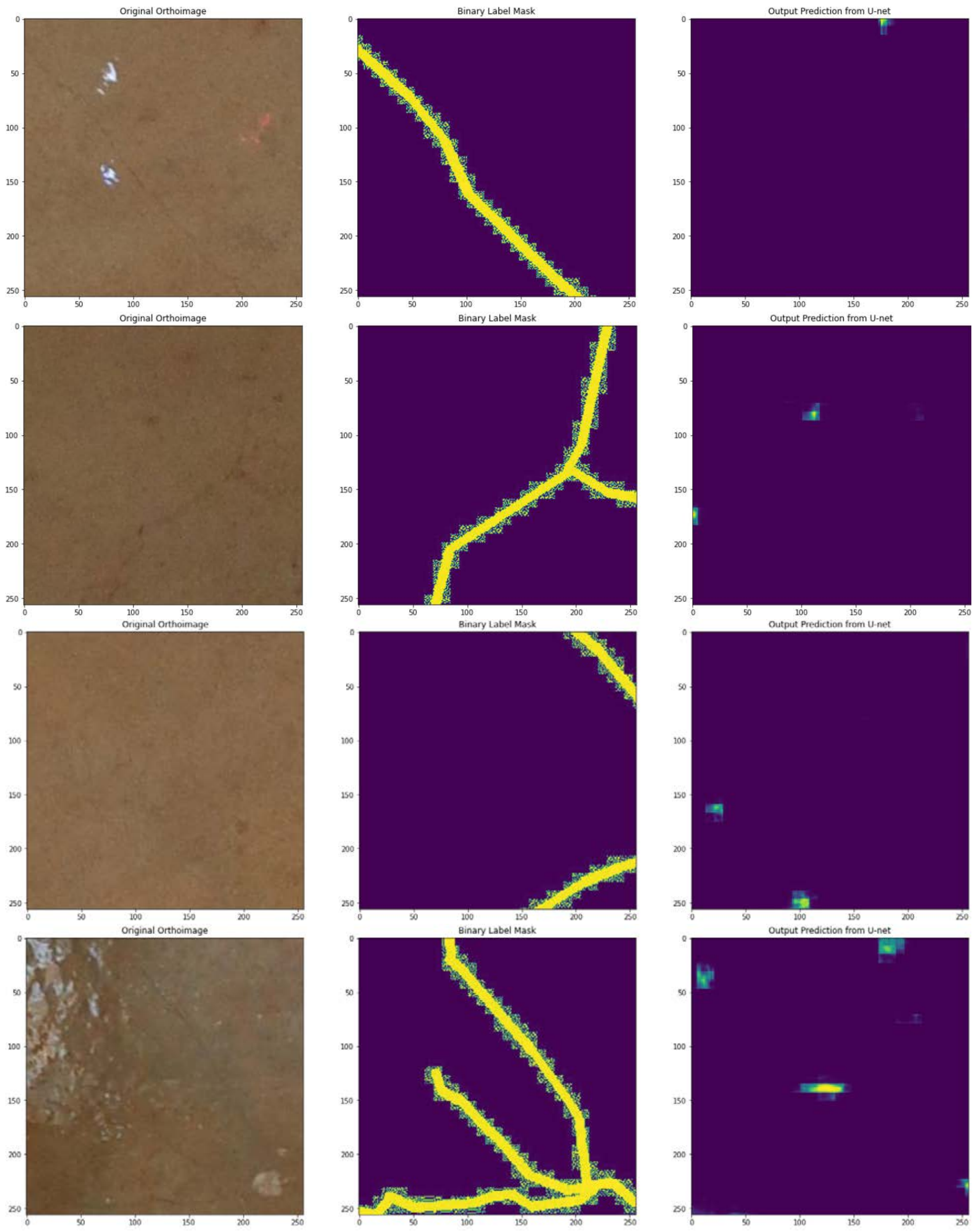
Thirty-two triplet image groups from the **Test** Dataset, which were randomly chosen. Each triplet group is composed of the original orthoimage input (left), the binary label mask target (center), and the output prediction mask from the U-net (right) where “purple” is no crack detected and “yellow” is crack detected.

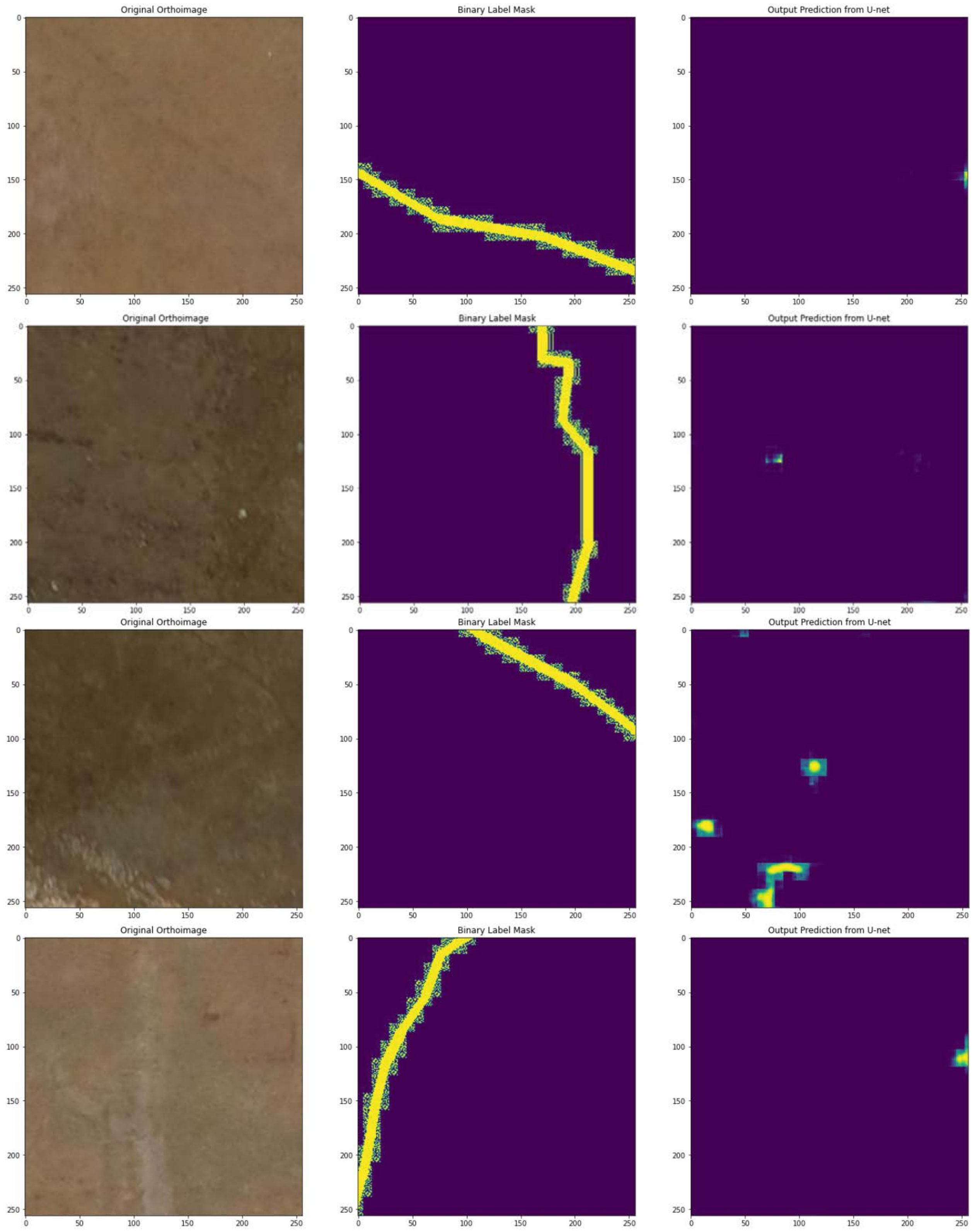


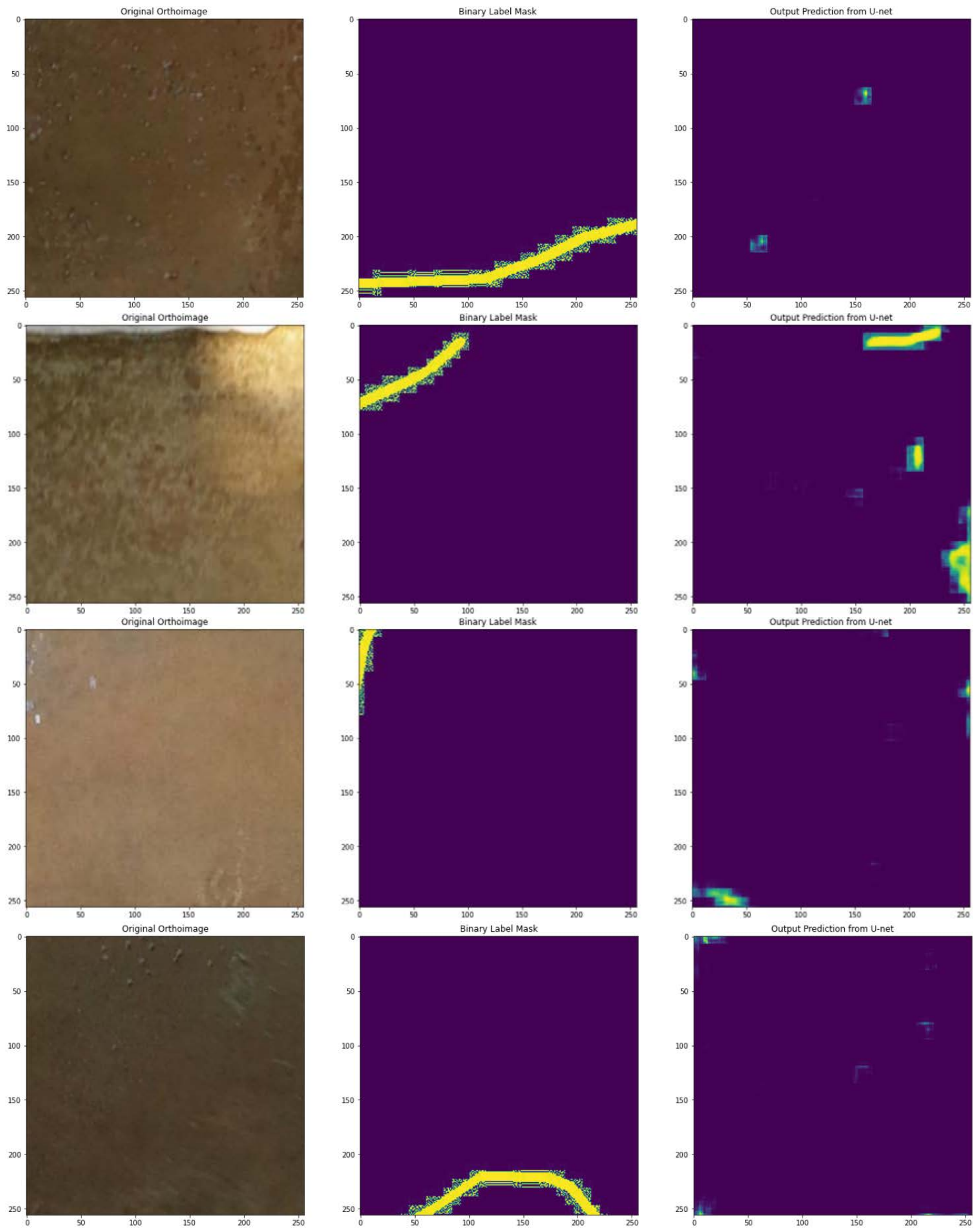

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net

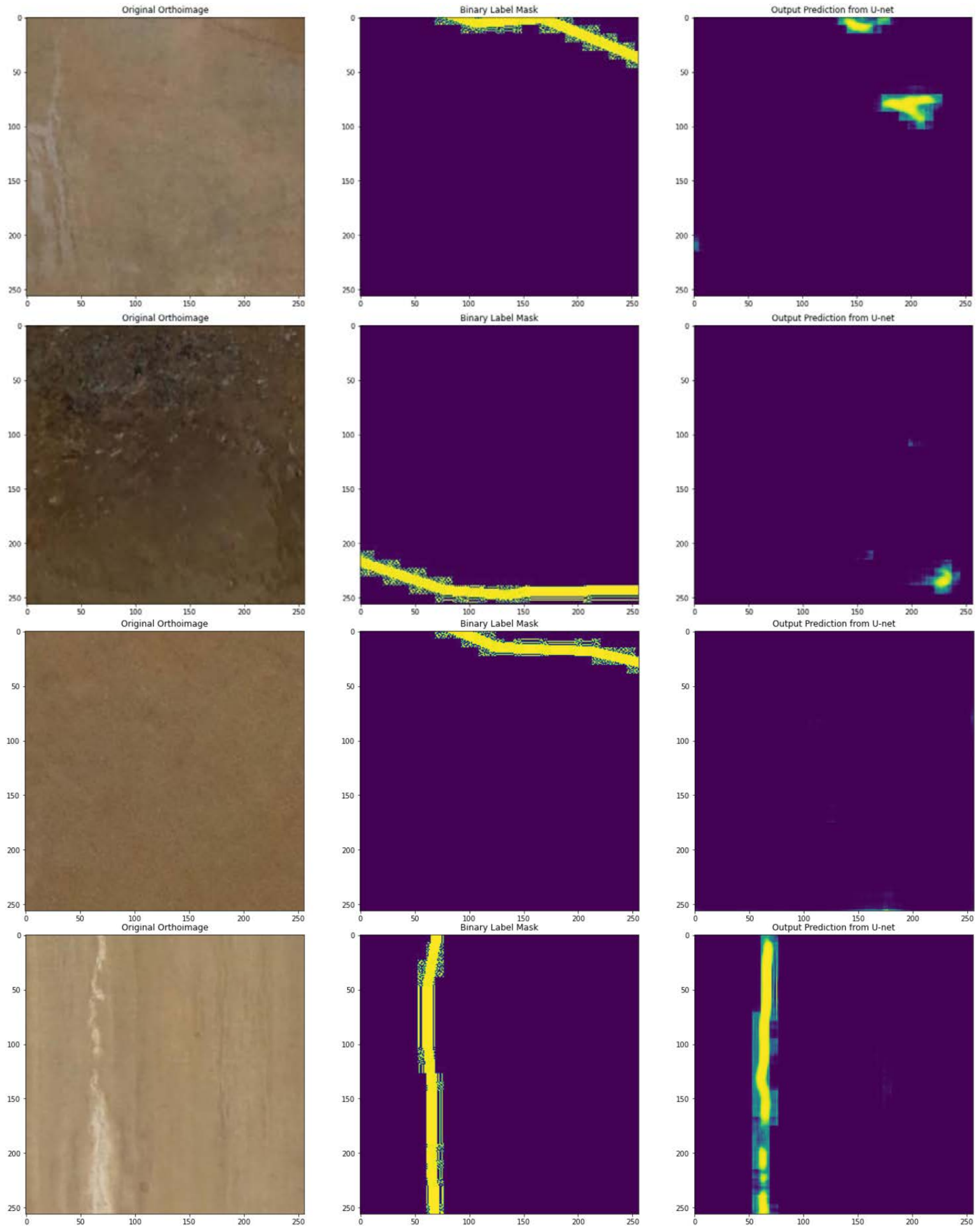

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net

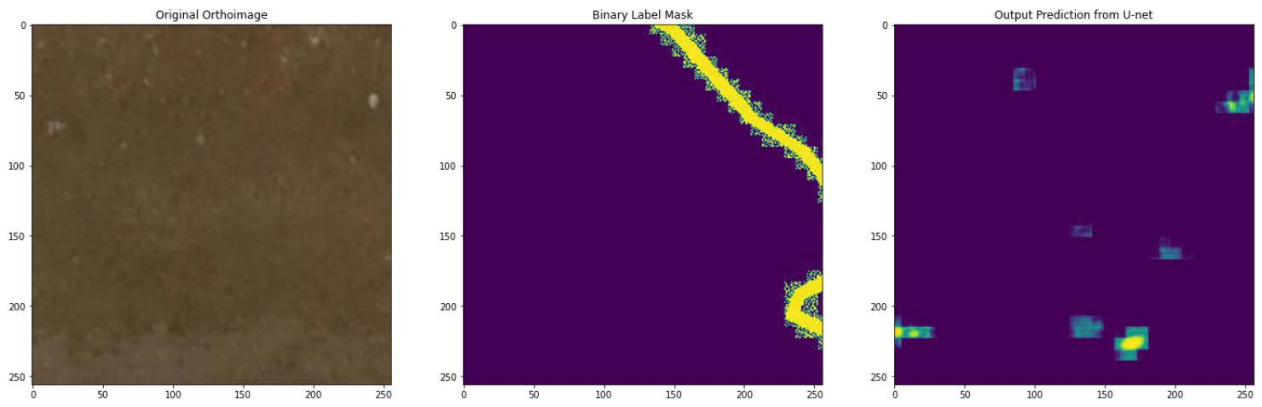

Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net


Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250


Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net


Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net
Original Orthoimage
Binary Label Mask
Output Prediction from U-net


Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250


Original Orthoimage
Binary Label Mask
Output Prediction from U-net
0
50
100
150
200
250