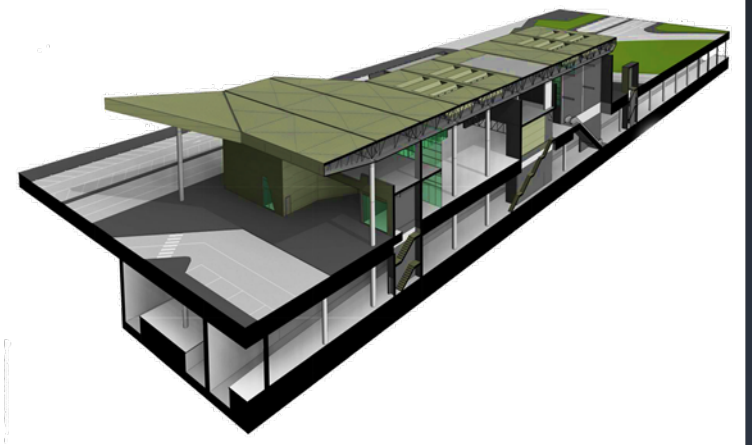
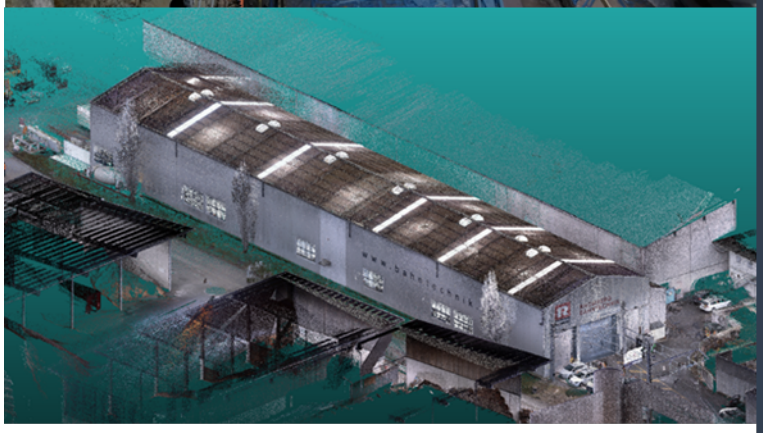




COGITO

CONSTRUCTION PHASE DIGITAL TWIN MODEL

cogito-project.eu

D5.3 – Deep - Learning -based Visual QC component v1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310

D5.3 – Deep-Learning -based Visual QC component v1

Dissemination Level:	Public
Deliverable Type:	Demonstrator
Lead Partner:	CERTH
Contributing Partners:	Hypertech, UCL
Due date:	31-03-2022
Actual submission date:	31-03-2022

Authors

Name	Beneficiary	Email
Athanasios Tsakiris	CERTH	atsakir@iti.gr
Apostolia Gounaridou	CERTH	agounaridou@iti.gr
Vasileios Karkanis	CERTH	epantrak@iti.gr
Tasos Sinanis	CERTH	sinanis@iti.gr
Vasileios Dimitriadis	CERTH	vdimitriadis@iti.gr
Anastasia Tsita	CERTH	atsita@iti.gr
Evangelia Pantraki	CERTH	epantrak@iti.gr
Apostolos Papafragkakis	Hypertech	a.papafragkakis@hypertech.gr
Kyriakos Katsigarakis	UCL	k.katsigarakis@ucl.ac.uk
Georgios Lilis	UCL	g.lilis@ucl.ac.uk

Reviewers

Name	Beneficiary	Email
Jochen	AU	teizer@cae.au.dk
Martin Straka	NT	straka@novitechgroup.sk
Giorgos Giannakis	Hypertech	g.giannakis@hypertech.gr

Version History

Version	Editors	Date	Comment
0.1	CERTH	04.02.2022	Table of Contents
0.2	CERTH	25.02.2022	Sections 1, 2, 3, 5.1, 5.2, 5.5, 5.6, 5.8
0.3	CERTH	11.03.2022	Update Sections 1, 2, 3, 5.1, 5.2, 5.5, 5.6, 5.8, 5.9
0.4	CERTH	19.03.2022	Sections 3.1.4, 3.3, 4, 5.3, 5.4, 5.7, 6
0.6	CERTH	21.03.2022	Final draft for internal review
0.8	AU, NT, Hypertech	28.03.2022	Deliverable internal review
0.9	CERTH	30.03.2022	Review comments addressed
1.0	CERTH, Hypertech	31.03.2022	Submission to the EC Portal

Disclaimer

Executive Summary

The COGITO Deliverable D5.3 “Deep-Learning -based Visual QC component v1” documents the first version of the COGITO Visual Quality Control (QC) component and presents the development activities concerning the T5.2 “Deep Learning Image Processing for Visual Quality Control”. Overall, the as-built data (2D images) acquired on site, is processed by the Visual Data Pre-processing Module and then sent to the Digital Twin (DT) Platform for digesting by other COGITO components. The Visual QC component receives the processed as-built data (2D images) from the DT Platform, analyse it and detect potential defective areas of the construction.

The Visual QC component is a backend service developed using a set of well-known open-source libraries. It is composed of four sub-components: the DT Platform connector, the Structural type-based selector, the Defect detector and the QC report generator. The DT Platform connector manages the communication and the data exchange with the DT Platform. The Structural type-based selector aims at loading the appropriate trained model for defect detection based on the type of the structure (concrete or steel) while the Defect detector classifies the image into specific categories. Finally, the QC report generator returns the results (predicted class and confidence level) to the DT Platform for further exploitation from other COGITO visualization components.

The present documentation of the COGITO Visual QC component, along with its sub-components, is oriented towards the functionalities of the tool, the technology stacks, the inputs, outputs and implemented APIs, the installation instructions, the assumptions and restrictions, the applications examples, the development and integration status, and the requirements coverage. In this first release, the COGITO Visual QC component implements a set of functionalities (i.e., structural type-based selection, defect detection etc.) focusing on concrete surfaces. Its usage is illustrated and evaluated with examples obtained during the development of the sub-components. In the second release, more functionalities will be implemented (i.e., defect detection on steel surfaces) or existing functionalities will be refined.

Table of contents

Executive Summary 3
Table of contents 4
List of Figures 6
List of Tables 7
List of Acronyms 8
1 Introduction 9
1.1 Scope and Objectives of the Deliverable 9
1.2 Relation to other Tasks and Deliverables 9
1.3 Structure of the Deliverable 9
2 Methodology 11
2.1 Theoretical Background 11
2.1.1 Image Classification 11
2.1.2 Object Detection 11
2.1.3 Deep Learning 11
2.1.4 Transfer Learning 12
2.2 Typical Types of Defects on Site 13
2.3 State of the Art 14
3 Dataset Construction 16
3.1 Concrete Defects Dataset 16
3.1.1 Existing Datasets 16
3.1.2 Additional Data 18
3.1.3 Data Augmentation Technique 19
3.1.4 Final Concrete Defects Dataset Properties 20
3.2 Steel Defects Dataset 21
3.2.1 Existing Datasets 21
3.2.2 Additional Data and Final Dataset 22
3.3 Structure Type Dataset 22
4 Deep Learning Model 24
4.1 Training 24
4.1.1 1st stage: Structure Type Classifier 24
4.1.2 2nd stage: Multiclass Concrete Defect Classifier 25
4.2 Evaluation 26
5 Deep -Learning -based Visual QC component 28
5.1 Prototype Overview 28
5.1.1 DT Platform connector sub-component 29
5.1.2 Structural type-based selector sub-component 29

5.1.3 Defect detector sub-component 29
5.1.4 QC report generator sub-component 29
5.2 Technology Stack and Implementation Tools 29
5.3 Input, Output and API Documentation 30
5.3.1 Input Data 30
5.3.2 Output Data 30
5.3.3 API Documentation 30
5.4 Application Example 32
5.4.1 Structure Type Classifier 32
5.4.2 Multiclass Concrete Defect Classifier 33
5.5 Licensing 38
5.6 Installation Instructions 38
5.7 Development and integration status 38
5.8 Requirements Coverage 38
5.9 Assumptions and Restrictions 39
6 Conclusions 40
References 41

List of Figures

Figure 1 - Categorizing general computer vision methods (top) and specific methods to defect detection, classification and assessment of civil infrastructure [4] 12
Figure 2 - The architecture of a Transfer Learning model 13
Figure 3 - Dataset examples. From left to right: Blistering, Crack, Honeycomb, Moss, Normal class 16
Figure 4 - Dataset examples. Top row from left to right: Efflorescence, Crack, Non-defective Background. Bottom row from left to right: Spalling, Corrosion, Exposed Rebar 17
Figure 5 - Dataset examples. Top row from left to right: Crack, General defect, No defect, Scaling, Spalling. Bottom row from left to right: Efflorescence, Exposed Rebar, Corrosion 17
Figure 6 - Dataset examples. From left to right:2 healthy and 2 potentially unhealthy 18
Figure 7 - Segmentation of the large image in smaller and labelling the smaller pieces 18
Figure 8 - Expanding an existing dataset using offline data augmentation technique 19
Figure 9 - Offline data augmentation- exposed rebar. Left: Original image. Right: augmented images artificially produced 20
Figure 10 - Offline data augmentation- honeycomb. Left: Original image. Right: augmented images artificially produced 20
Figure 11 - Final Concrete Defect Dataset. Top row from left to right: 3 blistering, 3 cracks. Bottom row: 3 efflorescence 21
Figure 12 - Final Concrete Defect Dataset. Top row from left to right: 1 exposed rebar, 2 honeycomb, and 2 no defect. Bottom row from left to right: 1 exposed rebar, 1 no defect 21
Figure 13 – NEU Dataset examples. From left to right: Rolled-in scale, Patches, Crazing, Pitted surface, Inclusion and Scratches 22
Figure 14 - Severstal Dataset examples. From left to right: Non-defective, defective steel surface 22
Figure 15 - Final Structure Type Dataset. Top row from left to right: 2 concrete surfaces, and 1 steel surfaces. Bottom row from left to right: 2 steel surfaces 23
Figure 16 - On the fly data augmentation technique 25
Figure 17- Visual QC component overall architecture 28
Figure 18 - Workflow in the Visual QC tool 28
Figure 19 - API endpoint for defect detection- request 31
Figure 20 - API endpoint for defect detection- response 31
Figure 21 – (a) Accuracy training/ Validation accuracy, and (b) Loss training/ Validation loss curves of InceptionV3 32
Figure 22 - Classification report of InceptionV3 33
Figure 23- Confusion matrix of Inception V3 33
Figure 24 - (a) Accuracy training, (b) Validation accuracy, (c) Loss training, and (d) Validation loss curves of the four models 34
Figure 25 - Training/ Validation accuracy and Training/ Validation loss curves of (a) InceptionV3, (b) MobileNet, (c) VGG19 and (d) ResNet50 34
Figure 26 - Classification report - Top row from left to right: (a) InceptionV3, (b) MobileNet. Bottom row from left to right: (c) VGG19, (d) ResNet50 35
Figure 27 - Confusion matrix - Top row from left to right: (a) InceptionV3, (b) MobileNet. Bottom row from left to right: (c) VGG19, (d) ResNet50 35
Figure 28- Performance of InceptionV3 for 9 unseen images 36
Figure 29 - Performance of MobileNet for 9 unseen images 36
Figure 30 - Performance of VGG19 for 9 unseen images 37
Figure 31 - Performance of ResNet50 for 9 unseen images 37

List of Tables

Table 1 – Famous CNN architectures used as pre-trained models 13
Table 2 - Typical defects based on the type of the structure 14
Table 3 – Final concrete defects dataset's summary 21
Table 4 - Final structure type dataset's summary 23
Table 5 - Customized layers architecture for the type classifier 24
Table 6 - Hyper-parameters in training the type of classifier 25
Table 7 - Customized layers architecture for the concrete classifier 26
Table 8 - Hyper-parameters in training the multiclass concrete classifier 26
Table 9- Confusion matrix of the multiclass problem 27
Table 10 – Libraries and Technologies used in Visual QC component 30
Table 11 – Visual QC component: Stakeholders' Requirements coverage from D2.1 38
Table 12 – Visual QC component: Functional and Non-Functional Requirements coverage from D2.4 39

List of Acronyms

Term	Description
API	Application Programming Interface
CNN	Convolutional Neural Network
COGITO	Construction Phase diGItal Twin mOdel
CPU	Central Processing Unit
DCC	Digital Command Centre
DigiTAR	Digital Twin visualisation with Augmented Reality
DL	Deep Learning
DNN	Deep Neural Network
DT	Digital Twin
FC	Fully Connected
GPU	Graphics Processing Unit
ML	Machine Learning
QC	Quality Control
UAV	Unmanned Aerial Vehicle
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

1 Introduction

1.1 Scope and Objectives of the Deliverable

This deliverable reports on the work conducted from M9 to M17 on the Deep Learning Image Processing for Visual Quality Control component that is developed as part of T5.2. The scope of this component is to perform an automatic Quality Control compliance and detect construction defects by expanding on the state-of-the-art deep learning-based detection and recognition models. The detection includes basic structural construction types such as concrete and steel surfaces, as well as various defect types (e.g. cracks, blistering etc.) that usually occur in new constructions.

More specifically, this deliverable reports on the development of the first release of the Visual Quality Control component focusing on its main sub-components listed below:

- **DT Platform connector sub-component:** manages the communication and the data exchange with the DT Platform.
- **Structural type-based selector sub-component:** is in charge of recognizing the type of the structure (concrete or steel) in order to load the appropriate trained model for detecting the defects.
- **Defect detector sub-component:** processes the as-built data (2D image) and classifies it into a defect category. In addition, it returns the confidence of the model's decision for this specific data.
- **QC report generator:** returns the results to the DT Platform for further exploitation.

1.2 Relation to other Tasks and Deliverables

T5.2 "Deep Learning Image Processing for Visual Quality Control" and consequently D5.3 "Deep-Learning-based Visual QC component v1" are related to the following COGITO tasks and deliverables:

- The first version of the COGITO architecture in the corresponding deliverable "D2.4 COGITO System Architecture v1" provided an overview on the Visual QC component, its requirements and the communication with other components.
- The Visual QC component, similarly to all components, relies on a shared ontology and a common data model developed within T3.2 "COGITO Data Model, Ontology Definition and Interoperability Design"; the first version of COGITO ontologies and data models have been documented in D3.2 "COGITO Data Model, Ontology Definition and Interoperability Design v1".
- The Visual Data Pre-processing Module provides, via the DT Platform, the as-built data (2D images) for Visual Quality Control to the Visual QC component (D3.7 "Visual Data Pre-processing Module v1").
- The DigiTAR module (D5.7 "User interface for Construction Quality Control v1") uses as input the results generated by the Visual QC component to visualize and confirm on site the detected defects and their location.
- The DCC module (D7.7 "Construction Digital Twin 3D Visualisation module v1") uses as input the results generated by the Visual QC component to visualize the detected defects within the BIM model.

1.3 Structure of the Deliverable

The rest of the deliverable is organised as follows:

- Section 2 presents the methodology followed, the theoretical background, typical defects occurred on site, and the relevant state of the art.
- Section 3 describes the dataset preparation, introduces already existing datasets, analyses the process of collecting additional data and generating new data to construct a new dataset for both concrete and steel defects.
- Section 4 notes the implementation and network training.

- Section 5 includes the technology stack, implementation tools, the input/output data, and API documentation, application examples, licensing, installation instructions, development and integration status, requirements coverage, as well as assumptions and restrictions of the Visual QC component.
- The document concludes with Section 6, where the progress, the next steps and the contribution to the overall COGITO objectives are being reported.

2 Methodology

This section presents the basic concepts of computer vision such as classification, object detection, Deep Learning etc. In addition, the typical types of defects are noted, based on the literature review, as well as the current state of the art regarding Machine and Deep Learning techniques for defect detection.

2.1 Theoretical Background

2.1.1 Image Classification

Image classification problem requires determining the category (class) that it belongs to. The problem is considerably complicated with the growth of categories' count: if several objects of different classes are present at the image, then the image could belong to several categories simultaneously [1].

2.1.2 Object Detection

Detection problem is more general in sense that it requires not only to determine whether the object of interest is present in image but also define where all its instances are located. Object detection is still a challenging problem due to several factors that must be handled: variety of possible objects' forms and colours, occlusions, lighting conditions, perspective etc. [1].

2.1.3 Deep Learning

Deep Learning (DL) considers being the wider part of the Machine Learning (ML) and becomes more popular day by day. It takes a lot of data and then can make decisions about new data. The data is passed through Neural Networks, known as Deep Neural Networks (DNN). The Convolutional Neural Networks (CNN) is a popular type of DNN. CNNs are very popular and wide-used in DL, while they eliminate the need for manual feature extraction like traditional features extraction algorithms (Figure 1).

A CNN architecture typically consists of several convolutional blocks and a Fully Connected (FC) layer. Each convolutional block is composed of a convolutional layer, an activation unit, and a pooling layer. A convolutional layer performs convolution operation over the output of the preceding layers using a set of filters or kernels to extract the features that are important for classification [2].

The CNN directly extract the features from a set of raw image data. Related features are not pre-trained; they learn when the networks are on the train on a group of images. This automated way of feature extraction is the most accurate learning models for computer vision tasks such as object detection, classification, recognition. To conclude, in DL approaches the network itself extracts the features and classifies the objects without user interpretation, while in traditional ML approaches the feature extraction is done manually and the classification algorithm classifies the objects separately [3].

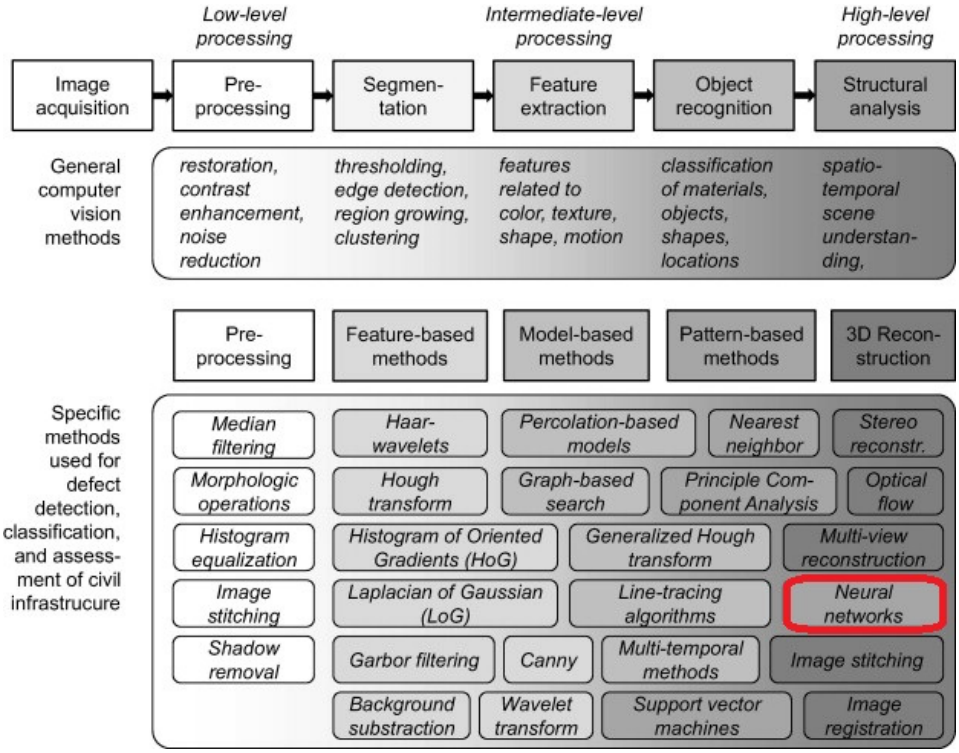


Figure 1 - Categorizing general computer vision methods (top) and specific methods to defect detection, classification and assessment of civil infrastructure [4]

2.1.4 Transfer Learning

Although the development of a Deep Learning model for defect detection in construction sites is the key part of this task, training a model from scratch takes a considerable amount of time even with computers with parallel CPUs. This long training time prevents quick validation of the trained classifier with various training options. While in Deep Learning the model is trained with a large volume of data and learns model weight and bias during training, these weights can be transferred to other network models for testing. Hence, the new network model can start with pre-trained weight (Transfer Learning) [3].

Transfer Learning is a ML technique to solve the fundamental problem of insufficient training data. As illustrated in Figure 2, a model is trained and developed for one task (Source Domain). It is then reused on a second related task (Target Domain) by freezing some layers, cutting the last ones and replacing them with new ones, focusing on specific classes. Transfer Learning refers to the situation whereby what has been learned in one setting is exploited to improve optimization in another environment.

Transfer Learning is usually applied if there is a new dataset smaller than the original dataset used to train the pre-trained model [5]. The Transfer Learning technique is considered an effective approach to reduce training time, while a deep learning model that has previously been trained for a similar purpose is fine-tuned and re-trained on a new specific dataset. Fine-tuning is currently very popular in the field of DL because it enables training DNNs with comparatively little data [2].

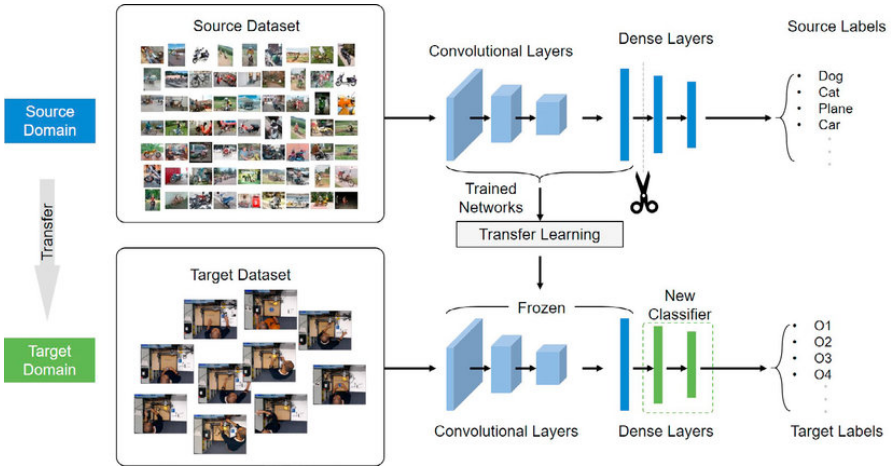


Figure 2 - The architecture of a Transfer Learning model[1]

Using a pre-trained model requires less computational power and less time than training the huge models on large datasets from scratch. Thus, training the new network with pre-trained weights can speed up the learning process. In Table 1, famous CNN architectures usually used as pre-trained models for image classification and the number of their parameters while training time are presented [3], [6].

Table 1 – Famous CNN architectures used as pre-trained models

CNN architecture	Year	Developed by	No. of Parameters
LeNet	1998	YannLeCun et al.	60000
AlexNet	2012	Alex K et al	62.3 million
VGGNet	2014	Simonyan, Zisserman	138 million
Inception[2]	2014	Google	4 million
ResNet	2015	Kaiming He	25 million

2.2 Typical Types of Defects on Site

In COGITO solution, the Visual Quality Control tool will be in charge of detecting defects on site, in both concrete and steel structures. Therefore, depending on the type of the structure, different defects will be detected in the visual data acquired by the Visual Pre-processing tool [7].

Based on [4] the defects related to general bridge elements are the following: Delamination/ Spallation/ Patched area, Exposed Rebar, Efflorescence/Rust Staining, Cracking, Abrasion/Wear, Distortion, Settlement, and Scouring. In addition, common civil/structural defects of concrete tunnels are the following: Scaling, Cracking, Spalling/Joint, Spall, Pop-Outs (holes), Leakage [4].

Combining the relevant literature review ([4], [8], [9], [10]) and the information provided by the industry partners, we managed to list the typical defects for concrete surfaces, which are presented in Table 2. It is worth noting that only typical defects that can occur in new constructions were taken into consideration. Defects such as corrosion stain and spalling were disregarded because they cannot be found in new constructions. In addition, geometry-based defects (such as distortion and settlement) were also ignored because these types of defects will be examined in another COGITO component, the Geometric QC [11]. Defects concerning steel surfaces will be investigated and presented in the second release of the Visual QC component (M24).

[1] https://medium.com/@lorenzofamiglini/transfer-learning-with-deep-learning-machine-learning-techniques-b4052befe7e2

[2] Inception is also known as GoogLeNet

Table 2 - Typical defects based on the type of the structure

Concrete Surface
Blistering
Crack
Efflorescence
Exposed Rebar
Honeycomb

2.3 State of the Art

CNNs stand out as an effective method for solving image-based object detection and classification problems [2]. Much of the work on defect classification has concentrated on detecting a single defect class (actually two classes: defect and background [8]. In addition, plenty of surveys have been conducted focusing largely the specific topic of crack detection on concrete surfaces [9]. In [12] a machine learning-based model was developed to detect cracks on concrete surfaces. A transfer-learning methodology was applied and the influence of the model's parameters (such as learning rate, number of nodes in the last fully connected layer and training dataset size) were investigated. In [13] an application for crack detection in buildings where is difficult to access or would endanger human life was presented. The architecture was based on CNNs and in this study three different approaches were described and compared. Bu et al. in [14] proposed an automatic bridge inspection approach employing wavelet-based image features along with support vector machines (SVM) for automatic detection of cracks in bridge images. In [10] a new method was proposed to recognize healthy and potentially unhealthy areas in concrete bridges, and thus to increase inspection efficiency by reducing the search space for a bridge inspector as well as guiding the inspector directly to the regions of interest. In addition, in this work, the Cambridge Bridge Inspection Dataset was composed, containing 1028 images belonging to two classes; healthy/ potentially unhealthy. [15] proposed an autonomous crack detection algorithm based on CNNs to solve the problem in the real world conditions, due to noise and undesired artifacts. In [16] a deep hierarchical CNN was proposed, to predict pixel-wise crack segmentation in an end-to-end method.

Furthermore, numerous studies exist, regarding the detection of a single defect in infrastructure. Wei et al. [17] proposed an instance-level recognition and quantification approach based on Mask R-CNN to recognize bugholes on concrete surface rapidly, accurately and quantify their area and maximum diameter. Gao et al. [18] proposed a method for pothole detection and segmentation based on cement concrete pavement that integrates the grayscale and texture features. However, the Visual QC component refers to a multi-class classification issue, while different types of defects should be detected on the construction sites.

Several studies have applied CNN-based algorithms to detect multiple types of defects on concrete's surfaces structures. In Hung et al. [2] a method for classifying damaged concrete surfaces based on machine vision and deep learning technologies was proposed. A new dataset with labelled classes (Normal, Cracked, Honeycomb, Blistering, and Moss) was created and published on GitHub. Based on the image dataset obtained, the Transfer Learning approach was used with three pre-trained models (VGG19, InceptionV3 and InceptionResnetV2). Customized FC layers were defined and associated to the pre-trained models to perform classification. The result indicated that both the models perform well (accuracy ~ 90%) for damage concrete classification. Hüthwohl et al. [8] presented a multi-classifier that can assign none, one, or multiple classification labels to a defect concrete image. A hierarchical classification approach was presented that consists of three stacked image classifiers. These classifiers were trained using manually labelled training data. A new dataset was constructed as a combination of the authors' own data collection and authority image sets. It consisted of 6 classes (Crack, Efflorescence, Scaling, Spalling, General defect, and No defect) for the first stage, a sub-set of 2 classes (Exposed reinforcement/ No exposed reinforcement) for the second stage and a sub-set of 2 classes (Rust staining/ No rust staining) for the third stage of classification. Inception V3 was used as pre-trained network and fine-tuned on the specific defect dataset. The model was afterwards assessed using unseen validation data. Experimental results showed that the three-staged hierarchical multi-classifier can reliably assign class labels to an image from a variety of classes with (F1 score ~ 83.5%) and detect potentially unhealthy concrete surface. Mundt et al. [9] introduced a novel multi-

class dataset (CODEBRIM) for multi-target classification of five commonly appearing concrete defects. Furthermore, the authors investigated and compared two recent meta-learning approaches, to identify suitable convolutional neural network architectures for this challenging multi-class multi-target task. It was shown that these architectures feature fewer overall parameters, fewer layers and are more accurate than their human designed counterparts on the presented multi-target classification task. Feng et al. [19] proposed a method to detect four different defect types (cracks, deposit, water leakage, and combination of the former three) using a deep neural network. Four classifiers were trained separately. A deep residual network was firstly designed for defect detection and classification in an image and was trained following an active learning strategy. Experiments demonstrated an efficient performance (accuracy ~87.5%). However, the classification results of these four defect-type-dependent classifiers could not be combined.

In case of steel surfaces, Abu et al. [5] developed deep learning models that can perform steel defect detection and evaluated the potential of transfer learning for this task. Four types of transfer learning methods (ResNet, VGG, MobileNet, and DenseNet) were experimentally evaluated to develop models for steel surface defect detection. The models were developed for binary classification (defect and no-defect) using the Severstal dataset and then they were also assessed for multiclass classification using NEU dataset. Image pre-processing was also included to improve the result of steel defects detection, while different parameters can affect the prediction outcome. The experimental results showed that MobileNet performed the best. In [20] a new approach was proposed. Part of pre-trained VGG16 was used as a feature extractor and a new CNN neural network was added as a classifier to recognize the defect of steel surface based on the feature maps created by the feature extractor. The method achieved a very high accuracy, compared with Deep Learning methods. While this study focused on extremely small datasets, an extra experiment was performed, where each class in the dataset only contained 10 images. The authors achieved a deeper optimization of the model, using different data augmentation and initialization methods. In [21] the authors, focusing on the problem of steel defect detection, explored three deep learning methods: Xception, U-Net and Mask R-CNN. They compared the aforementioned networks' performance on the Severstal dataset. Wang et al. [22] combined improved ResNet50 and enhanced faster R-CNN to reduce the average running time and improve the accuracy. Within the improved ResNet50 the sample is classified as with defect or without defect. In case of a probability lower than a 0.3 threshold, the output is directly marked as without defect. Otherwise, the sample is further input into the improved faster R-CNN to detect the specific type of defect. The final output is the location and classification of the defect in the sample or without defect in the sample. The authors in this study also used the Severstal dataset, after modifying it. Finally, Cha et al. [23] proposed a method to provide quasi real-time simultaneous detection of multiple types of damages (not only specific types of damage such as concrete or steel cracks). For this paper, a new dataset including 2,366 images (with 500 × 375 pixels) labelled for five types of damages (concrete crack, medium steel corrosion, high steel corrosion, bolt corrosion, and steel delamination) was developed; however, the dataset is not public. The Faster R-CNN was modified, trained, validated, and tested using this dataset, achieving both a high performance of the model and remarkably fast speed. Therefore, a framework for quasi real-time damage detection on video using the trained networks was developed.

In summary, the majority of the existing studies regarding concrete surfaces are focused on binary classification problems (crack/ non-crack etc.). Studies, which concern a multiclass classification issue, focus mostly on long-term concrete defects and mainly refer to bridge deterioration; in the best of our knowledge, no considerable studies exist for automatic visual inspection on steel structures. In COGITO, the intention is to exploit the Visual Quality Control component for automating visual inspection on several new construction sites (not only on bridges). In the final version of the Visual QC module, a two-staged defect classifier will be developed; in the first stage, the image will be classified as either a concrete or steel structure (binary classification problem). Depending on this decision (concrete or steel structural type), in the second stage potential defects will be identified in the same image in case of an unhealthy area. Our aim is to combine the information collected from existing datasets with real data to increase the variety during training and focus on the defects that occur both in concrete and steel new structures.

3 Dataset Construction

Preparing data for machine learning projects is a crucial first step. This Section describes the image gathering process in order to create the relevant datasets used for training the models to detect defects on site. The first sub-section describes the existing datasets that refer to the potential concrete defects. The second sub-section refers to existing datasets that contain defects on steel surfaces. Thus far, the Visual QC component has focused on the concrete defects; therefore, the case of steel structures will be examined and presented in detail in the second release of the tool (M24).

3.1 Concrete Defects Dataset

3.1.1 Existing Datasets

In the first step, extensive research was conducted through literature review regarding existing relevant defect datasets. The majority of the existing concrete defect classification datasets (SDNET, BCD, ICCD) mainly focus on crack detection tasks [6]. While this COGITO task concerns a multi-classification problem, in this section, datasets for damage classification in a multi-label setting will be presented.

Based on the literature review and relevant publications, three already existing (multi-class classification) concrete defect datasets, which were found online and used in our case, are presented below.

- **Concrete Damage Classification master**
  The original dataset comprised of five classes: Blistering, Crack, Honeycomb, Normal and Moss. 636 images were acquired and divided into 2 groups, training and testing, containing 80 and 20 percent of the images respectively. The offline image augmentation technique was implemented on both groups to get more training and testing samples. In that way, 4,200 images were available for training and 1,050 for testing (5,250 images in total). Finally, to avoid unbalanced data, the number of images in each class was made equal [2].All the images are 227 x 227 pixels. Examples of this dataset are illustrated in Figure 3.

Figure 3 - Dataset examples. From left to right: Blistering, Crack, Honeycomb, Moss, Normal class

- **COncrete DEfect BRidge Image (CODEBRIM)**
  CODEBRIM features six mutually non-exclusive classes: Crack, Spallation, Efflorescence, Exposed Bars, Corrosion (stains) and Non-defective Background. 1,590 high-resolution images were acquired from 30 unique bridges, at different scales and resolutions. After annotation process, the images were divided in 3 groups, training, validation and testing, containing 80, 10 and 10 percent of the images respectively. The final dataset is composed by 7,860 images. The images were acquired at high-resolution, partially using an unmanned aerial vehicle (UAV) to gain close-range access and feature varying scale and context. The bridges were chosen according to varying overall deterioration, defect extent, severity and surface appearance (e.g., roughness and colour). Images were taken under changing weather conditions to include wet/stained surfaces with multiple cameras at varying scales [9]. Examples of this dataset are depicted in Figure 4.

Figure 4 - Dataset examples. Top row from left to right: Efflorescence, Crack, Non-defective Background. Bottom row from left to right: Spalling, Corrosion, Exposed Rebar

- **Multi classifier data**
  For this dataset, 38,408 raw images were used in total, but after a processing and labelling step, only 3,607 single defect images were extracted, given that many of the images collected only show intact concrete surfaces without a defect. The final dataset consists of five classes: Cracks, Efflorescence, General defect, No defect, Scaling and Spalling. In addition two smaller binary datasets are included, for Exposed Reinforcement or not, as well as for Corrosion and Rust Stain or not, while in case of Efflorescence, Scaling and Spalling, a further investigation for the existence or not of Exposed Reinforcement, as well as for the existence of Corrosion and Rust Staining or not occurs [8]. Examples of this dataset are presented in Figure 5.

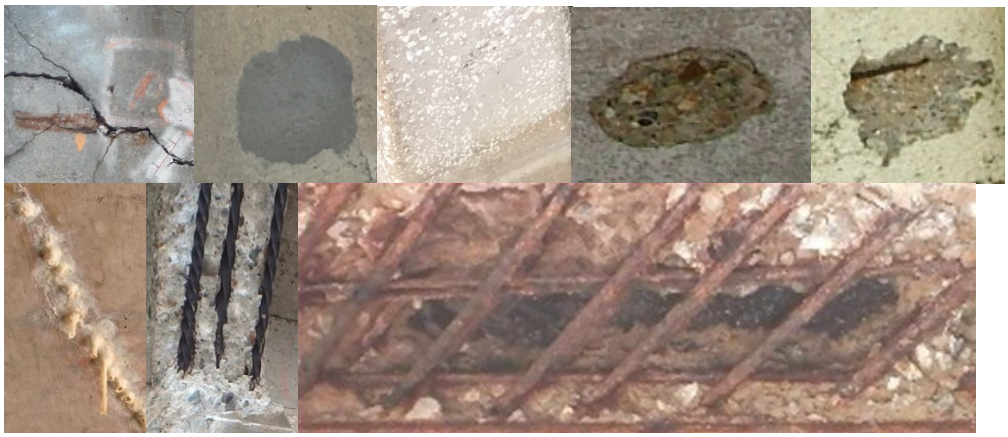


Figure 5 - Dataset examples. Top row from left to right: Crack, General defect, No defect, Scaling, Spalling. Bottom row from left to right: Efflorescence, Exposed Rebar, Corrosion

In most of the cases, the datasets contained useful classes of defects; however, some classes referred to defects that occur after a long time (such as moss and corrosion stains). Therefore, these classes were excluded, as well as the images, which depict two or more type of defects concurrently (multi-label classification[3]). Based on the above, a large number of existing images was exploited to construct our initial dataset.

[3] Class labels are not mutually exclusive - there is no constraint on how many of the classes the instance can be assigned to.

- **Cambridge Bridge Inspection Dataset**

The Cambridge Bridge Inspection dataset is a small bridge inspection dataset that seeks to detect concrete defects. The samples are divided into healthy and potentially unhealthy (691 and 337 images respectively), hence, it concerns a binary classification problem. Besides different luminous conditions on all images, the unhealthy class consists of concrete damage such as cracking, graffiti, vegetation, and blistering [24]. Examples of this dataset are presented in Figure 6.

Figure 6 - Dataset examples. From left to right:2 healthy and 2 potentially unhealthy

3.1.2 Additional Data

After the collection of the existing datasets, many online meetings were arranged with the industry partners to finalise the lists of defects. In addition, a large amount of raw data acquired during quality control process of elder projects and construction sites was provided for further processing. The provided data was captured by UAVs; therefore, the image size and resolution were quite high. However, the required data for training a model should have fixed dimensions (i.e., 224x224 pixels) and the available data was much greater. For that reason, the original raw data had to be processed, to be in consistency with the existing datasets. The original large images were divided into several smaller images (224×224 pixels) using Python. This way, only images with dimensions of 224×224 pixels were created. The new images constitute parts of an original image corresponding to concrete surface with many types of defects. They were then checked and labelled manually from scratch. Only the segmented images, which depicted clearly only one type of defect, were selected to be included in the final dataset. Abstract or blurry images that may confuse the model during training were excluded. Figure 7 depicts the segmentation and labelling of the new raw data.

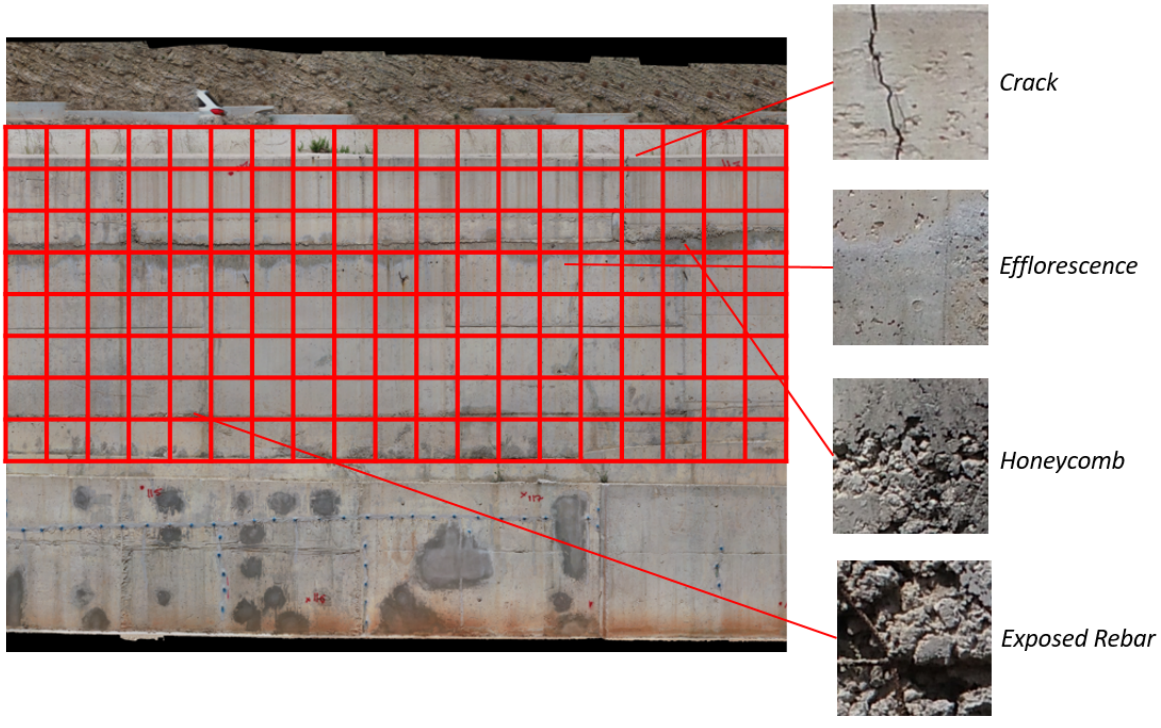


Figure 7 - Segmentation of the large image in smaller and labelling the smaller pieces

3.1.3 Data Augmentation Technique

As aforementioned in the previous sections, a new dataset was created combining all the relevant data found in existing datasets in the literature with the new raw data provided by the industry partners. Unfortunately, the class distribution of the constructed dataset was not equal or close to equal; instead, it was biased. The constructed dataset considered imbalanced, while in some classes, the number of samples was sufficient (i.e., cracks) but some classes lacked in samples (i.e., exposed rebar). An imbalanced dataset can create problems in the classification task, during the evaluation of the model's performance. Therefore, it was necessary to balance the dataset by virtually replicating the under-represented class examples such that the overall defect number per class is on the same scale ensuring that defect classes are sampled equally during training.

The data augmentation technique can be defined as the technique used to improve the diversity of the data by slightly modifying copies of already existing data or newly create synthetic data from the existing data [25]. It is a great way to expand the size of the dataset, as new transformed images come up from the original dataset. During this process, different transformations are applied to original images and multiple transformed copies of the same image are produced. Each copy, however, is different from the others (depending on the augmentation techniques i.e., shifting, rotating, flipping, etc.) Applying these small amounts of variations on the original image provides a new perspective of the object but does not change its target class practically. In addition, applying the data augmentation technique, allows the model to become more robust and to generalize better on unseen data, as incorporates a level of variation in the dataset. The artificial increase in our case was achieved by implementing the offline data augmentation technique, using the PIL library (Figure 8).

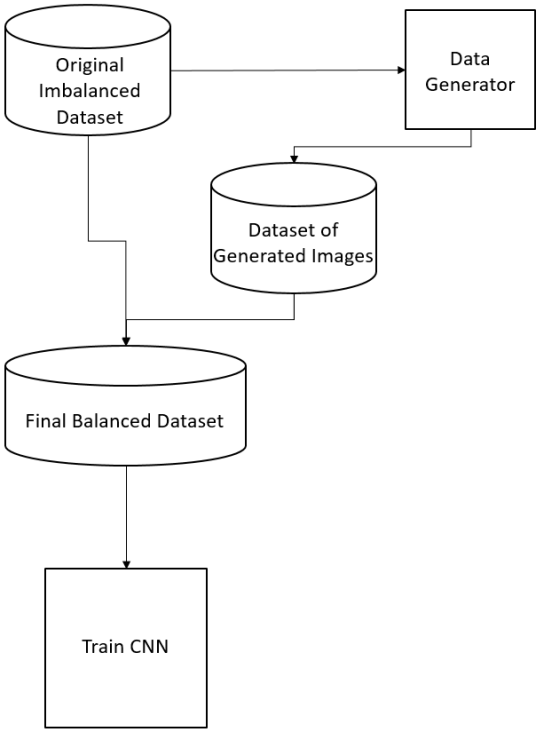


Figure 8 - Expanding an existing dataset using offline data augmentation technique

Two artificial augmented images were produced from the original one. In the first augmented image, 4 transformations were implemented: rotation 90 degrees clockwise, blur filter, reducing of brightness and contrast. In the second augmented image, 3 transformations were implemented: vertical flip, sharpness and increasing of brightness. Examples of the data augmentation implemented for the exposed rebar and the honeycomb class are illustrated in Figure 9 and Figure 10, respectively.

Figure 9 - Offline data augmentation- exposed rebar. Left: Original image. Right: augmented images artificially produced

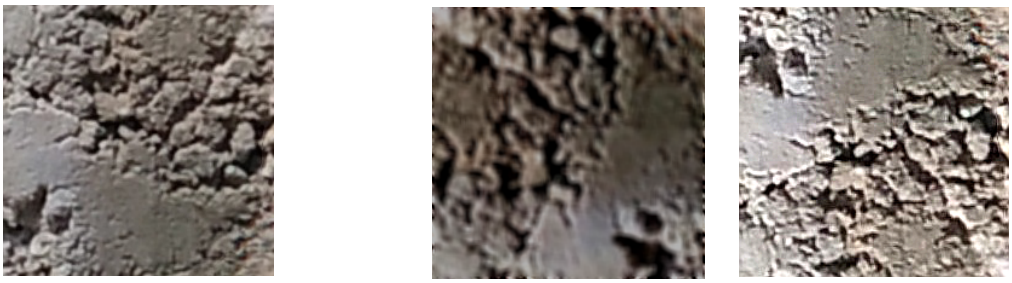


Figure 10 - Offline data augmentation- honeycomb. Left: Original image. Right: augmented images artificially produced

3.1.4 Final Concrete Defects Dataset Properties

The final concrete dataset was created to detect 5 types of concrete defects: blistering, crack, efflorescence, exposed rebar and honeycomb. However, an extra class was included to cover the case of a non-defective area. Hence, the final dataset contains 6 classes and 8,225 images; the 80% of the samples was used for training the model while the 20% of the samples was used for validation and testing sets (10% and 10% respectively). More specifically, 6,578 images were used for training, while the rest 1,647 (820 and 827) for validation and testing respectively as illustrated in Table 3. Over 1,000 images belong to each class. The images were collected from the existing datasets analysed in Section 3.1.1. We were focused only in the final defect types presented in Section 2.2; therefore, some classes (such as spalling) were disregarded while this type of defects cannot occur in new constructions. Images from existing datasets were mixed to comprise the final dataset. The data was also cleaned up to avoid confusing samples, which include more than one defects. While some of the existing datasets are based only on ideal laboratory conditions, contain only ideal cracks and surfaces, excluding the specific types of damage to real structures in different environmental conditions, the new processed data provided by the industry partners were also exploited at this step. By combining several images from different existing datasets and new real data (which however belong to the same category), we tried to create a more representative dataset containing different instances of defects. While the variety of samples depicting defects is increased significantly (i.e., image size, camera resolution, weather conditions etc.), it is very likely that the model generalizes and performs better in unknown data. This fact can lead to model's optimization and the improvement of its performance. In addition, as aforementioned in Section 3.1.3, some classes (such as exposed rebar and honeycomb) lacked in samples. The offline data augmentation approach was implemented in these cases, to balance the dataset and avoid the biases. The final concrete defects dataset's details are presented in Table 3.

Table 3 – Final concrete defects dataset's summary

No .of images / Classes	Original	Augmented	Total	Training	Validation	Testing
Blistering	1,050	-	1,050	840	105	105
Crack	1,394	-	1,394	1,115	139	140
Efflorescence	1,397	-	1,397	1,117	139	141
Exposed Rebar	562	1,124	1,686	1,348	168	170
Honeycomb	990	400	1,390	1,112	139	139
No defect	1,308	-	1,308	1,046	130	132
Total	**6,701**	**1,524**	**8,225**	**6,578**	**820**	**827**

Some representative examples of the data collected from existing datasets mixed with the raw data provided by the industry partners and used for training, validation and testing process are illustrated in Figure 11 and Figure 12.

Figure 11 - Final Concrete Defect Dataset. Top row from left to right: 3 blistering, 3 cracks. Bottom row: 3 efflorescence

Figure 12 - Final Concrete Defect Dataset. Top row from left to right: 1 exposed rebar, 2 honeycomb, and 2 no defect. Bottom row from left to right: 1 exposed rebar, 1 no defect

3.2 Steel Defects Dataset

3.2.1 Existing Datasets

Based on the literature review and relevant publications, two existing (multi-class classification) steel defect datasets found online.

- **NEU Dataset**

The NEU Dataset[4] includes 1,800 greyscale images (200 x 200 pixels) of six different kinds of typical surface defects of the hot-rolled steel strip (rolled-in scale, patches, crazing, pitted surface, inclusion and scratches). The dataset is balanced; thus, each class consists of 300 samples. The 300 samples per class are then split and the 240 images can be used for training while the 60 images for testing [5]. Examples of this dataset are presented in Figure 13.

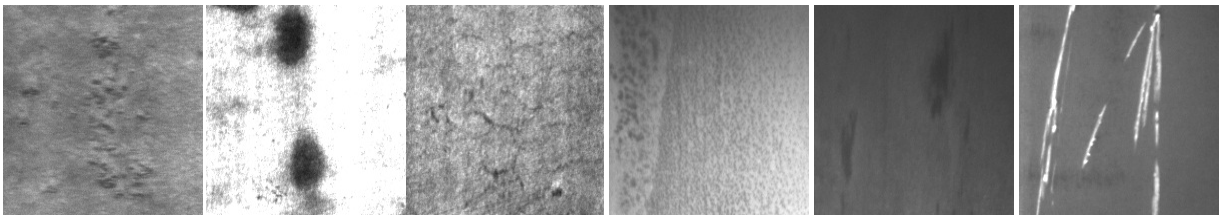


Figure 13 – NEU Dataset examples. From left to right: Rolled-in scale, Patches, Crazing, Pitted surface, Inclusion and Scratches

- **Severstal Dataset**

The Severstal[5] steel defect dataset contains 12,568 grey scale images (1600 x 256 pixels) of steel surface with and without defects (four types: pitted surface, crazing, scrapes, and patches) [5]. Examples of this dataset are presented in Figure 14.

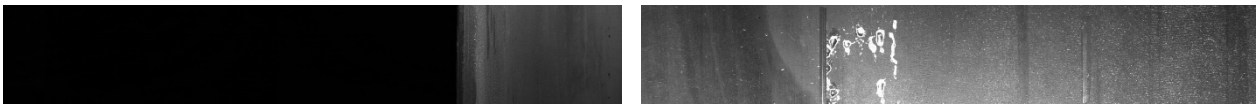


Figure 14 - Severstal Dataset examples. From left to right: Non-defective, defective steel surface

3.2.2 Additional Data and Final Dataset

In case of steel, it is obvious that literature is not as extensive as concrete surface defects and these datasets do not seem to be representative of the steel constructions real case scenarios. Thus, additional data should be collected from the COGITO industry partners, in order to create a dataset that which corresponds to specific types of damage in real structures under different environmental conditions. The collection of the appropriate data will take place in the next months and the final multiclass steel dataset will be presented in the second release of the Visual QC component (M24).

3.3 Structure Type Dataset

As it was already mentioned, at the first stage of the process, a binary classifier identifies if the image sent to the Visual Quality Control tool concerns a concrete or a steel structure surface. As presented in Table 4, the first version of the component, a binary dataset finally containing 509 images (300 and 209) belonging to two classes (concrete/ steel surfaces respectively) was created, exploiting the offline data augmentation technique (Section 3.1.3). The concrete surface images were selected from the created concrete dataset (Section 3.1.4). The images depicting steel structures were collected from the web, exploiting an open-source command line python tool, google-images-download[6]. This tool is used to scrape images from search engines (e.g., Google) by keywords (i.e., rusty bolts, steel structures etc.) and download them to the computer. Some additional samples from the NEU Dataset were also exploited. It is worth mentioning that at this point, the focus is on whether the image depicts a concrete or a steel structure (not a healthy or an

[4] https://www.kaggle.com/kaustubhdikshit/neu-surface-defect-database
[5] https://www.kaggle.com/iafoss/severstal-256x256-images-with-defects
[6] Available at: https://pypi.org/project/google_images_download/

unhealthy area). In the second version of the tool, once steel defects data will be available, the binary dataset will be enriched and refined. Examples of this dataset are presented in Figure 15.

Table 4 - Final structure type dataset's summary

No. of images / Classes	Original	Augmented	Total	Training	Validation	Testing
Concrete	100	200	300	240	30	30
Steel	99	110	209	167	20	22
Total	**199**	**310**	**509**	**407**	**50**	**52**

Figure 15 - Final Structure Type Dataset. Top row from left to right: 2 concrete surfaces, and 1 steel surfaces. Bottom row from left to right: 2 steel surfaces

4 Deep Learning Model

The Visual Quality Control works as follows: at the first stage, a binary classifier was implemented to define the type of the structure; it detects whether the image uploaded for visual inspection depicts a concrete or a steel surface. At the second stage, another multiclass model was trained in order to detect whether a concrete defect is identified in this specific image. In the second version of the component an additional model, which detects steel defects, will be created. The appropriate model will be uploaded based on the first stage decision. The first sub-section describes the proposed method to implement the Transfer Learning technique for detecting the structural type (binary classifier) and the defects on site (multiclass classifier). The second sub-section presents the metrics used for evaluating the models.

4.1 Training

For the implementation of both classifiers (binary and multiclass), a Transfer Learning strategy was applied, while the number of collected defect samples was not sufficient to train a Deep Neural Network (DNN) from scratch.

4.1.1 1st stage: Structure Type Classifier

At the first stage, primary experiments were done regarding the identification of the type of the structure illustrated in the image data. Therefore, we implemented the Transfer Learning strategy exploiting the InceptionV3 pre-trained CNN model. According to the Transfer Learning approach (Section 2.1.4) and as illustrated in Figure 2, the pre-trained layers were frozen; changes were made only to the final additional layer. Regarding the network weights, just updating the final layer of the network requires less training time and memory while updating the whole network leads to a higher classification reliability [8]. Hence, only the weights of the new customized layers of the network were updated, while the rest were set as non-trainable.

Instead of the last layer, we added custom new layers to create a new binary classifier able to handle this binary problem (concrete/ steel). On top of the existing layers, a customized Flatten layer was added, followed by a Dense layer with 512 hidden units and ReLu activation function. Furthermore, a Dropout layer was added to avoid overfitting. Last, an extra Dense layer implemented to classify the image in the two potential categories: concrete or steel. Table 5 shows the customized layers architecture. The last output layer consisted of a single unit with Sigmoid activation function.

Table 5 - Customized layers architecture for the type classifier

Layer (type)	Output Shape	No. of Parameters
Flatten	(None, 131072)	0
Dense	(None, 512)	67,109,376
Dropout	(None, 512)	0
Dense	(None, 1)	513
Total Params:	67,109,889	
Trainable Params:	67,109,889	
Non-trainable Params:	0	

Next, the model was compiled on the few last layers of the network in order to adjust the pre-trained weights using the Binary Cross-entropy loss function and the Adam optimizer. The model was trained with the following hyper-parameters Table 6.

Table 6 - Hyper-parameters in training the type of classifier

Hyperparameter	Value
Epochs	20
Learning rate	0.001
Decay	0.01
Batch size	32
Dropout	0.001
Optimizer	Adam

Finally, it worth mentioning that the dataset was loaded for training and evaluating the model using the Tensorflow ImageDataGenerator[7]. The ImageDataGenerator generates batches of tensor image data with real-time data augmentation. It is a data augmentation technique that allows to randomly load and transform the data on the fly while training process. The transformations applied during training were rotation, zoom, horizontal and vertical flip, width and height shift, as well as brightness range.

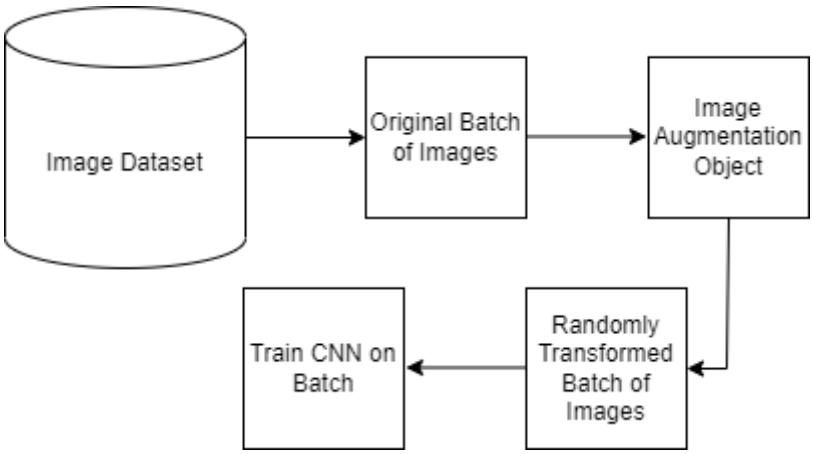


Figure 16 - On the fly data augmentation technique

4.1.2 2nd stage: Multiclass Concrete Defect Classifier

Following the assumption that the above binary classification result was correct, the appropriate model will be loaded to identify potential defects on the surface (concrete or steel). Thus far, the implementation has focused on the case of concrete multiclass problem. We experimented with different state of the art image classification pre-trained CNN models including VGG19, InceptionV3, ResNet50 and MobileNet to compare them and to evaluate their performance. As in the binary classifier, the existing layers were frozen; changes were made only to the final layer. Actually, instead of the last layer (these models are built to handle up to 1000 classes) we added custom new layers to create a new classifier able to handle the target classes (i.e., blistering, crack, efflorescence, exposed rebar and honeycomb) as illustrated in Figure 2. Regarding the network weights, just updating the final layer of the network requires less training time and memory while updating the whole network leads to a higher classification reliability [8]. Hence, only the weights of the new customized layers of the network were updated, while the rest were set as non-trainable.

As mentioned above, in order to create a new classifier after last layer's removal, a few customized layers were added on top of these output features. More specifically, the output of the last convolutional layer was first flattened (by adding a Flatten layer) and connected to a Dense layer with 512 hidden units and ReLU activation function. In addition, a Dropout layer was added to avoid overfitting. The last output layer consisted of a single unit with Softmax activation function. Softmax is an activation function that turns numbers aka logits into probabilities that sum to one. It outputs a vector that represents the probability

[7] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

distributions of a list of potential outcomes (in this case are the six classes: blistering, crack, efflorescence, exposed rebar, honeycomb and no defect. Table 7 shows the customized layers architecture and the number of trainable parameters in each tested pre-trained model.

Table 7 - Customized layers architecture for the concrete classifier

Output Shape / Layer (type)	InceptionV3	MobileNet	VGG19	ResNet50
Flatten	(None, 131072)	(None, 50176)	(None, 25088)	(None, 100352)
Dense	(None, 512)	(None, 512)	(None, 512)	(None, 512)
Dropout	(None, 512)	(None, 512)	(None, 512)	(None, 512)
Dense	(None, 6)	(None, 6)	(None, 6)	(None, 6)
Total Params:	67,112,454	25,693,702	32,873,030	76,971,526
Trainable Params:	67,112,454	25,693,702	32,873,030	76,971,526
Non-trainable Params:	0	0	0	0

Next, the model was compiled on the few last layers of the network in order to adjust the pre-trained weights using the Categorical Cross-entropy loss function and the Adam optimizer. The model was trained with the following hyper-parameters (Table 8).

Table 8 - Hyper-parameters in training the multiclass concrete classifier

Hyperparameter	Value
Epochs	30
Learning rate	0.001
Decay	0.0001
Batch size	32
Dropout	0.01
Optimizer	Adam

The dataset was loaded for training and evaluating the model using the Tensorflow ImageDataGenerator. The data augmentation technique on the fly was implemented as before.

4.2 Evaluation

The robustness of the network is measured using classification quality measures such as precision, recall, accuracy and F1 score. For the case of a non-binary classifier, all samples belonging to a corresponding group are summed up. Quality measures can then be defined using these four groups of labels.

Precision is the fraction of samples classified as positive that are actually positive.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Recall[8] is the ratio of the Actual Positives the model managed to identify (True Positive - TP).

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{TN}}$$

Accuracy is the fraction of a sample being classified correctly, independently from its class.

[8] Recall is also known as sensitivity.

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}}$$

In general, accuracy is a reliable measurement for assessing the overall performance of a classifier. However, it is biased towards the predominant class if classes are unbalanced. For this reason, the F1 score is introduced which takes both false positives and false negatives into account. F1 score is the harmonic mean of precision and recall [8].

$$F1\ score = 2 * \frac{\mathrm{precision} * \mathrm{sensitivity}}{\mathrm{precision} + \ \mathrm{sensitivity}}$$

For the Visual Quality Control component evaluation, we focused on the recall and F1 score. Recall reflects the percentage of samples belonging to a class that the model correctly classified; it actually calculates how many relevant items were retrieved. The higher the recall, the more defects were correctly detected. Thus, the model can be trusted in its ability to detect defects. In addition, F1 score has an advantage in unbalanced datasets such as in our case.

During the evaluation for the multiclass problem, the trained network was asked to predict the label for each image in the testing set. During this process, the predictions were compared to the ground-truth labels, the category of the actual images of the testing set. The results were summed up in a confusion matrix. A confusion matrix allows us to measure the aforementioned metrics. It is a specific table layout, which allows visualization of the performance of the algorithm. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. An example of a confusion matrix is depicted in Table 9. The yellow cell (True Positive - TP) represents the samples that were actually cracks and were also predicted as cracks by the model. The purple cells (False Negative - FN) represent the samples that were actually cracks but they were predicted to belong in other classes (such as blistering). Finally, the cyan cells (False Positive - FP) represent the samples that despite the fact that they belong to another class (such as honeycomb) were wrongly predicted as cracks.

Table 9- Confusion matrix of the multiclass problem

		PREDICTED class				
ACTUAL class	**Classes**	Blistering	Crack	Efflorescence	Exposed Rebar	Honeycomb
	Blistering	**TN**	FP	TN	TN	TN
	Crack	FN	**TP**	FN	FN	FN
	Efflorescence	TN	FP	**TN**	TN	TN
	Exposed Rebar	TN	FP	TN	**TN**	TN
	Honeycomb	TN	FP	TN	TN	**TN**

5 Deep -Learning -based Visual QC component

In this section, the Deep-Learning-based Visual QC component is presented in detail. The overall architecture of the component is presented, as well as the different sub-components. In addition, relevant information about the implementation tools, the current status etc. are described in the next sub-sections.

5.1 Prototype Overview

The Visual QC tool is organized into four submodules, which are presented in the following sub-sections. The overall architecture of the tool is illustrated in Figure 17.

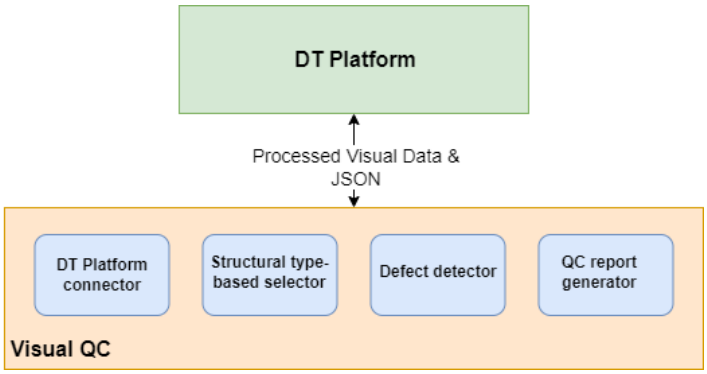


Figure 17- Visual QC component overall architecture

In Figure 18, the basic workflow of the automated visual QC process is described. The input to the Visual QC tool is the processed data, as well as relevant metadata regarding the involved building components of the BIM model, provided through the DT Platform. The Structural type-based selector first decides which deep learning algorithm should be loaded for the defect detection process, based on the type of the structure (steel or concrete). Based on its decision, the Defect detector sub-component practically performs the defect detection, exploiting the respective trained model (i.e., Inception V3 pre-trained model). The predicted result is then forwarded to the QC report generator, which creates a new report regarding the defect type and the relevant metadata. The results are finally sent to the DT Platform in a JSON format file.

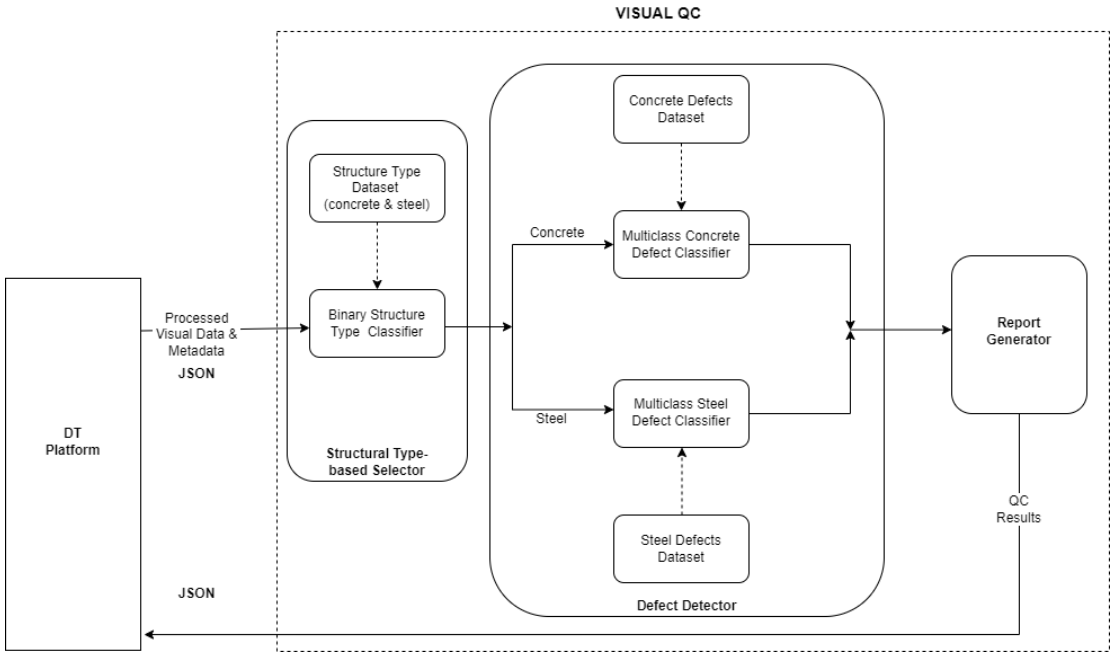


Figure 18 - Workflow in the Visual QC tool

5.1.1 DT Platform connector sub-component

The DT Platform connector enables the communication of the Visual QC tool with the DT Platform. It is in charge of data exchanging with the DT Platform through a provided API endpoint.

5.1.2 Structural type-based selector sub-component

The Structural type-based selector selects the appropriate trained model depending on the structural type of the visual data (concrete or steel). Practically it is the binary, trained model, which predicts weather the image depicts a surface of a concrete or a steel structure. Based on this decision, the appropriate multiclass model will be loaded to be used for detecting the defects.

5.1.3 Defect detector sub-component

The Defect detector carries out the Visual QC using the processed visual data and the appropriate trained model. Practically it is the multiclass, trained model, which is loaded and predicts the defect class for each image sent to the Visual QC tool.

5.1.4 QC report generator sub-component

The QC report generator creates the Visual QC reports that are going to be delivered to the DT Platform, based on the results of the Defect detector. It actually returns the result in order to be further exploited by the relevant COGITO visualization components (DigiTAR and DCC).

5.2 Technology Stack and Implementation Tools

The programming language used in this tool was Python 3.8. All the experiments with neural networks were carried out on an NVIDIA GeForce GTX 1660 GPU running on the CUDA[9] platform. For the Deep Learning (DL) models training, an i5-9600K CPU at 3.70 GHz with 16 GB RAM was used.

Furthermore, the training of the DL model was performed using the Tensorflow 2.6.0 Keras library.

TensorFlow is an end-to-end open-source platform for machine learning. It is a comprehensive and flexible ecosystem of tools, libraries and other resources, providing workflows with high-level APIs. In addition, it gives flexibility and control with features like the Keras Functional API [26].

Keras is a high-level neural networks library that is running on the top of TensorFlow. It is designed for deep neural networks and works as a wrapper to TensorFlow framework. Using Keras in deep learning allows for easy and fast prototyping. It is written in Python code, which is easy to debug and allows ease for extensibility. The main advantages of Keras are described below:

- User-Friendly: Keras has a simple, consistent interface optimized for common use cases, which provides clear and actionable feedback for user errors.
- Modular and Composable: Keras models are made by connecting configurable building blocks together, with few restrictions.
- Easy To Extend: With the help of Keras, it is easy to write custom building blocks for new ideas and researches.
- Easy To Use: Keras offers consistent & simple APIs, which helps in minimizing the number of user actions required for common use cases and provides clear and actionable feedback upon user error [26].

NumPy[10] is an open source project aiming to enable numerical computing with Python. It is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

[9] CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to speed up dramatically the computing applications by harnessing the power of GPUs.
[10] https://numpy.org/

PIL[11] (Python Imaging Library) is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

Flask[12] is a Python web framework, used for developing web applications using Python. More specifically, Flask provides the user with tools, libraries and technologies that allow building a web application. Using the Flask framework has several advantages, such as the built-in development server and the fast debugger, the lightweight of the framework and the supported secure cookies.

All the libraries and technologies used for developing the Visual QC component are summarised in Table 10.

Table 10 – Libraries and Technologies used in Visual QC component

Library/Technology Name	Version	License
Python	3.8	Python Software Foundation
Tensorflow	2.6.0	Apache License 2.0
Keras	2.6.0	Apache License 2.0
NumPy	1.19.5	Berkeley Software Distribution (BSD)
PIL	8.2.0	HPND License
Flask	1.1.2	Berkeley Software Distribution (BSD)

5.3 Input, Output and API Documentation

The Visual Quality Control component will interact only with the DT Platform for the data exchange. It will receive metadata regarding a created job and structural elements that need to be assessed, as well as the respective 2D image depicting the area of interest and the involved structural components. In addition, it will forward to the DT Platform, the generated results (predicted class and confidence level) for each image sent for quality control.

5.3.1 Input Data

The Visual QC component would require as input the following information:

- **Processed as-built 2D image data:** processed image (i.e. .png, .jpeg) of the as-built data provided by the Visual Quality Control tool.
- **Metadata:** JSON file containing all the relevant information accompanying the processed image (e.g. Project_ID, Job_ID, etc.).

5.3.2 Output Data

- **Metadata:** JSON file containing information generated by the Visual QC component (i.e. Predicted_Label, Confidence_Level, and Status).

5.3.3 API Documentation

Regarding the API, a first version is implemented in this first release of the Visual QC component. The API is used for storing, retrieving, updating and deleting data. The requests were tested in Postman API Platform[13]. In the second release, the component will expose an execution interaction with the DT Platform in order to be consistent with the rest of the COGITO solution. Thus far, tests were done involving a POST request and a URL for getting the image data.

Once the web service is up and running, the Visual QC is accessible through the following endpoint:

<local_IP>:<defined_port>/QualityControl/Type_Defect_Identification

[11] https://python-pillow.org/
[12] https://flask.palletsprojects.com/en/2.0.x/
[13] https://www.postman.com/

where the <local_IP> is the IP of the local machine on which the Visual Quality Control component has been deployed, whereas the <defined_port> is the port of the server that is defined by the user.

As aforementioned, via a POST request the Visual QC receives information such as Project_ID, WorkOrderID and the image URL (Figure 19). The workflow begins, the appropriate model is loaded and makes a prediction. In Figure 20, the response is illustrated; extra fields such as Structure_Type, Confidence_Level, Detected_Defect and Status are generated to be forwarded to the DT Platform.

Figure 19 - API endpoint for defect detection- request

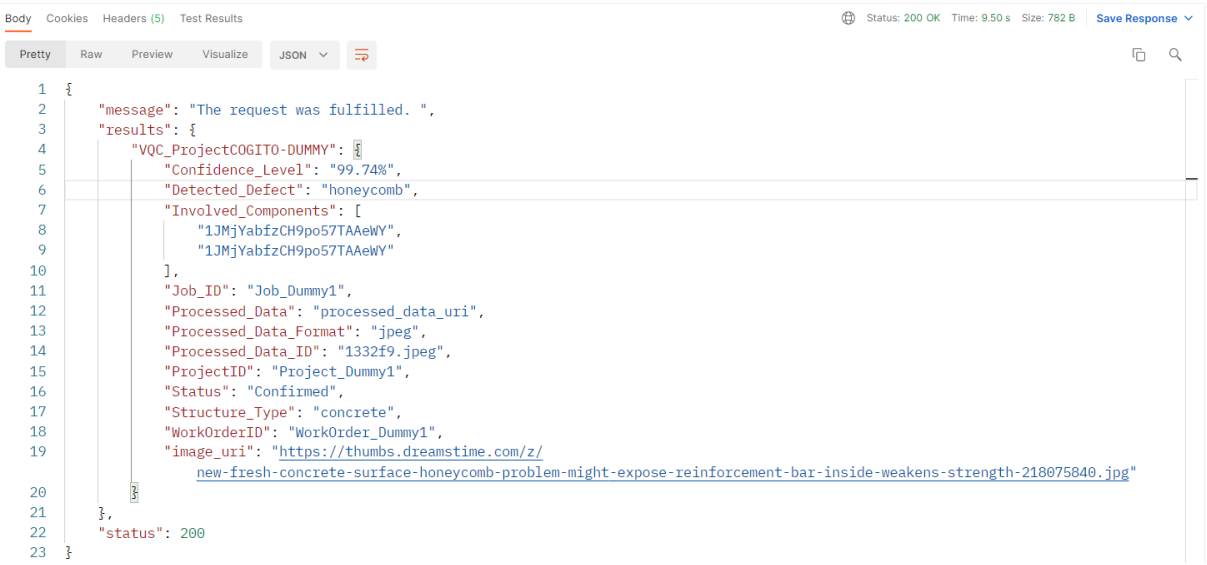


Figure 20 - API endpoint for defect detection- response

As can be seen from the above figure, the enriched JSON contains the following entries:

- **Structure_Type:** The structural type (concrete/ steel) of the specific image, generated by the 1st stage classifier.
- **Detected_Defect:** The predicted defect category of the specific image, generated by the 2nd stage classifier.
- **Confidence_Level:** The percentage that the model is confident that the actual value occurs.
- **Status:** Depending on the Confidence_Level value. At this step, if the confidence level is higher than 75% the status is defined as “Confirmed”. If not, the status is defined as “Unconfirmed” and needs to be confirmed by the stakeholder on site (DigiTAR tool).

5.4 Application Example

5.4.1 Structure Type Classifier

Thus far, the binary dataset (concrete/ steel) and the type identification is implemented in a preliminary level to ensure that the service is available and the basic pipeline is achieved. As already mentioned, the dataset will be enriched (with extra steel surfaces images) and refined in the second version of the component; hence, results regarding the binary classification (concrete or steel structure) will be presented in summary in this deliverable.

Based on the trained process as described in Section 4.1.1, the model was tested in 52 samples (30 and 22 concrete and steel samples respectively). By implementing the Early Stopping[14] technique to avoid overfitting, the training process has been interrupted after 13 epochs while the accuracy had not further improved for 5 epochs. However, the accuracy of the model seems to be high (98 % accuracy) and considers a very satisfactory outcome. The performance of the model (accuracy and loss) is illustrated in Figure 21.

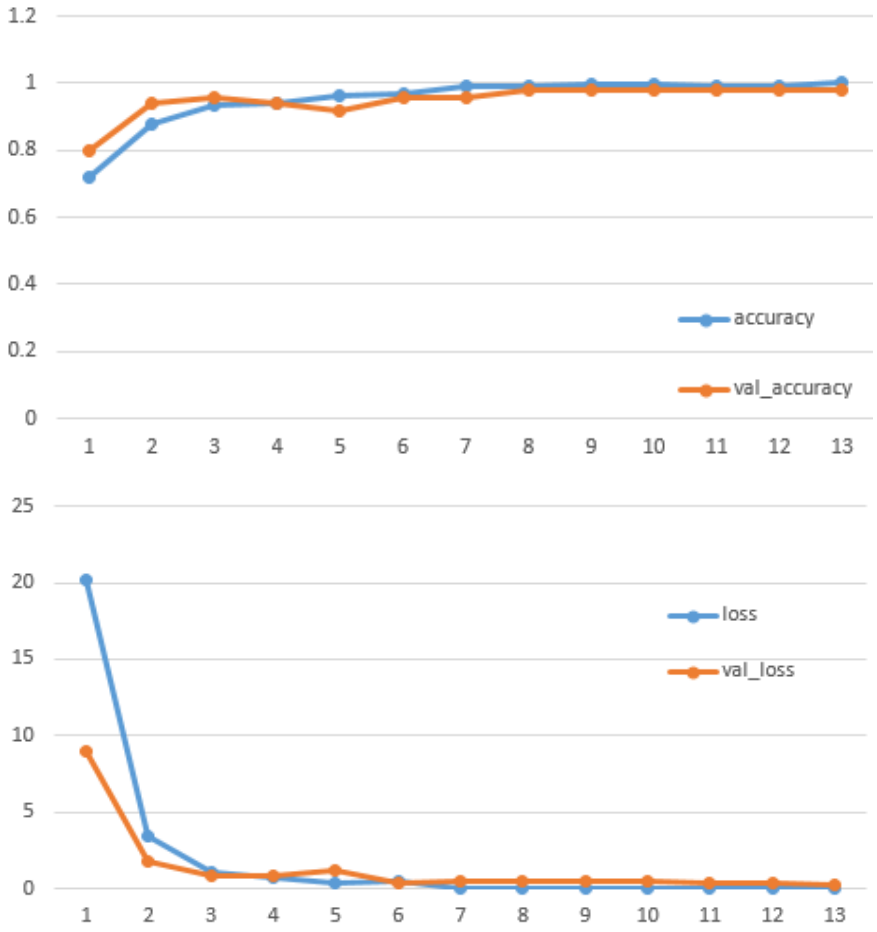


Figure 21 – (a) Accuracy training/ Validation accuracy, and (b) Loss training/ Validation loss curves of InceptionV3

The classification report of the model is depicted in Figure 22. Both the metrics calculated have a great value (> 97%) which makes the model very accurate and reliable.

[14] https://keras.io/api/callbacks/early_stopping/

	precision	recall	f1-score	support
Concrete (Class 0)	1.00	0.97	0.98	30
Steel (Class 1)	0.96	1.00	0.98	22
accuracy			0.98	52
macro avg	0.98	0.98	0.98	52
weighted avg	0.98	0.98	0.98	52

Figure 22 - Classification report of InceptionV3

The confusion matrix of the model is illustrated in Figure 23. As can be seen, only one sample was misclassified as steel structure instead of concrete. All the other predictions were correct.

		Predicted Label	
	Classes	Concrete	Steel
Actual Label	Concrete	29	1
	Steel	0	22

Figure 23- Confusion matrix of Inception V3

5.4.2 Multiclass Concrete Defect Classifier

The results of experiments done regarding the multiclass problem of detecting defects on concrete surfaces are presented in this sub-section. Four pre-trained CNN models, InceptionV3, MobileNet, VGG19 and ResNet were compared using the same dataset, pipeline and hyper parameters. The performance (accuracy and loss) of the models is presented in Figure 24. The results indicate that ResNet50 and MobileNet perform better than VGG19 and InceptionV3 in this multiclass concrete dataset. After 30 epochs, the accuracy reaches 91%, and 92% respectively, compared to 88 % of VGG19 and InceptionV3. During testing, ResNet50, MobileNet and InceptionV3 are almost equal. In term of loss value, in validation set VGG19's loss is again higher than the other Networks. It is worth mentioning that VGG19 was stopped training earlier (25 epochs) while the validation loss had not improved for 10 epochs.

However, it seems that MobileNet, ResNet50 and VGG19 tend to be overfitting whereas InceptionV3 has normal curves. In Figure 25, it can be seen that the three aforementioned models, could have stopped training earlier (less than 10 epochs) to avoid overfitting. In this case, the accuracy would have been lower (but not significantly) than the final. InceptionV3 requires more time (30 epochs) to reach its highest values. For all these reasons, ResNet50 has shown more stable and reliable results to solve the multi-classification concrete defects problem. The high accuracy of this model, 91%, also confirms that transferring learning approach is effective while saving training time.

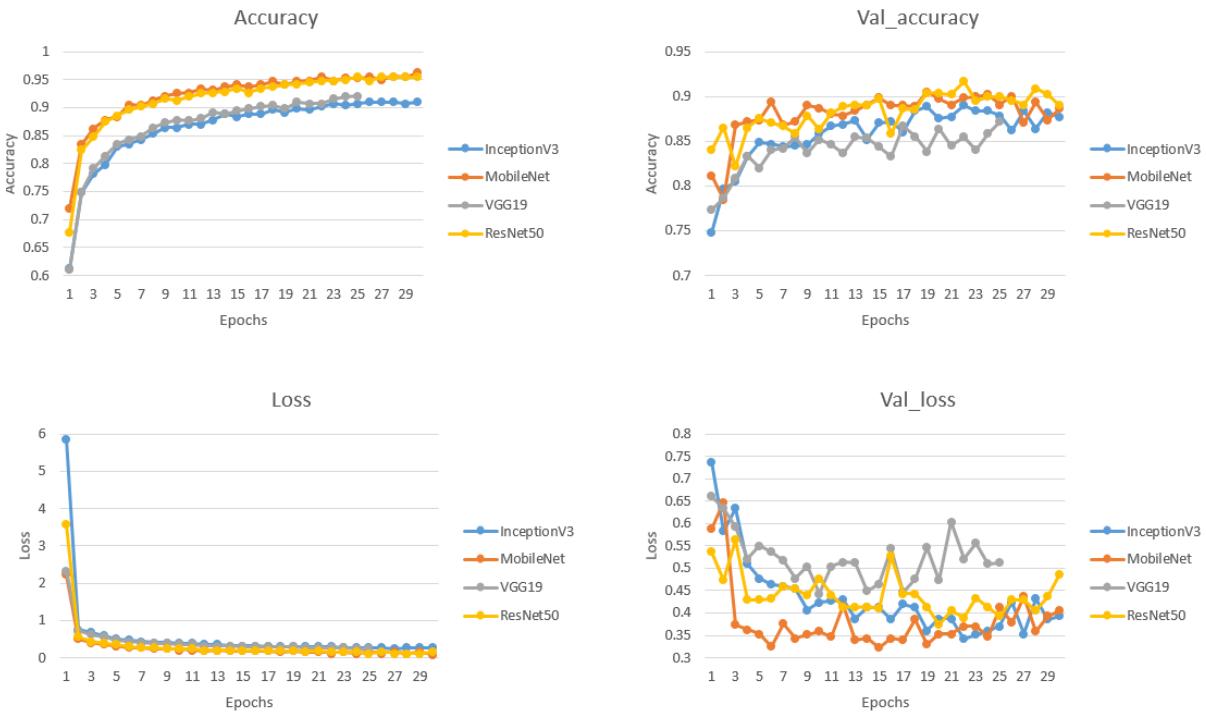


Figure 24 - (a) Accuracy training, (b) Validation accuracy, (c) Loss training, and (d) Validation loss curves of the four models

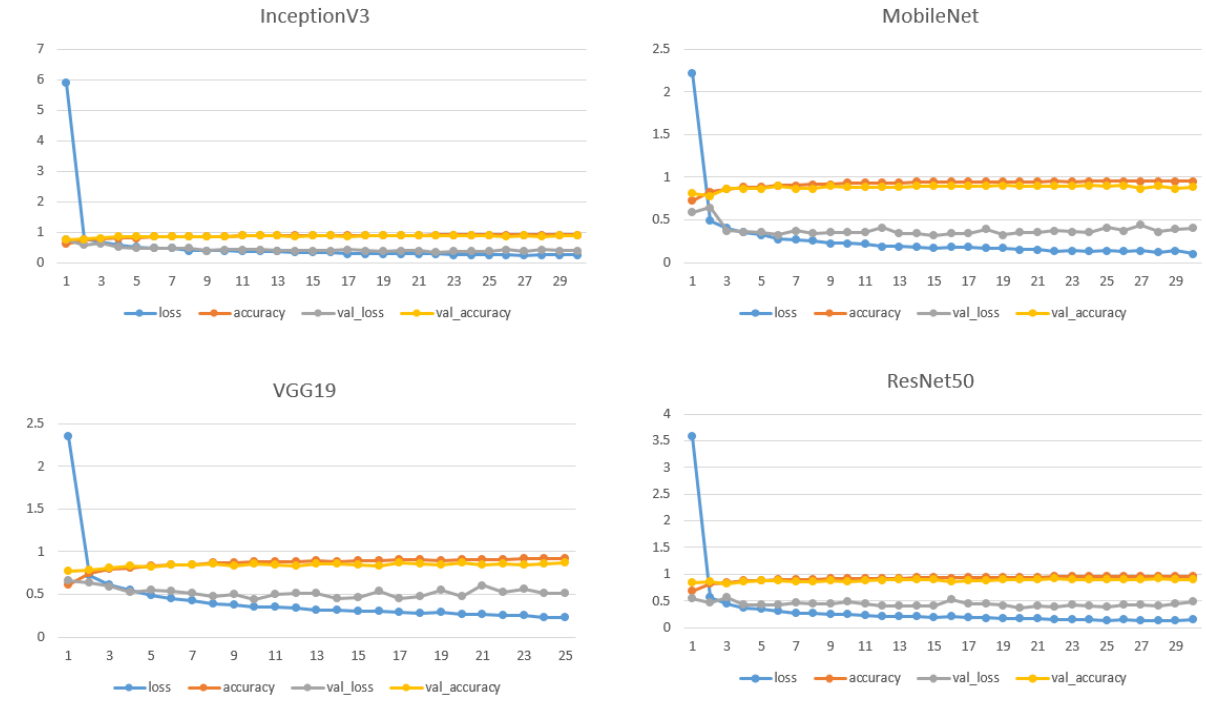


Figure 25 - Training/ Validation accuracy and Training/ Validation loss curves of (a) InceptionV3, (b) MobileNet, (c) VGG19 and (d) ResNet50

Regarding the classification report, in Figure 26, the average F1 score of ResNet (0.91) is almost equal with the MobilNet F1 score (0.92). The VGG19 F1 score is lower (0.88) than the two aforementioned models, as well as InceptionV3 (0.88).

	precision	recall	f1-score	support
Blistering (Class 0)	0.91	1.00	0.95	105
Crack (Class 1)	0.79	0.90	0.84	140
Efflorescence (Class 2)	0.81	0.91	0.86	141
Exposed_Rebar (Class 3)	0.95	0.85	0.90	170
Honeycomb (Class 4)	0.97	0.81	0.89	139
No_defect (Class 5)	0.90	0.86	0.88	132
accuracy			0.88	827
macro avg	0.89	0.89	0.89	827
weighted avg	0.89	0.88	0.88	827

	precision	recall	f1-score	support
Blistering (Class 0)	1.00	0.87	0.93	105
Crack (Class 1)	0.92	0.87	0.90	140
Efflorescence (Class 2)	0.91	0.91	0.91	141
Exposed_Rebar (Class 3)	0.94	0.96	0.95	170
Honeycomb (Class 4)	0.95	0.96	0.95	139
No_defect (Class 5)	0.83	0.94	0.88	132
accuracy			0.92	827
macro avg	0.93	0.92	0.92	827
weighted avg	0.92	0.92	0.92	827

	precision	recall	f1-score	support
Blistering (Class 0)	0.94	0.91	0.93	105
Crack (Class 1)	0.79	0.87	0.83	140
Efflorescence (Class 2)	0.89	0.77	0.82	141
Exposed_Rebar (Class 3)	0.91	0.93	0.92	170
Honeycomb (Class 4)	0.94	0.91	0.92	139
No_defect (Class 5)	0.82	0.89	0.85	132
accuracy			0.88	827
macro avg	0.88	0.88	0.88	827
weighted avg	0.88	0.88	0.88	827

	precision	recall	f1-score	support
Blistering (Class 0)	0.99	0.91	0.95	105
Crack (Class 1)	0.85	0.89	0.87	140
Efflorescence (Class 2)	0.91	0.90	0.90	141
Exposed_Rebar (Class 3)	0.95	0.94	0.94	170
Honeycomb (Class 4)	0.96	0.92	0.94	139
No_defect (Class 5)	0.85	0.92	0.88	132
accuracy			0.91	827
macro avg	0.92	0.91	0.92	827
weighted avg	0.92	0.91	0.91	827

Figure 26 - Classification report - Top row from left to right: (a) InceptionV3, (b) MobileNet. Bottom row from left to right: (c) VGG19, (d) ResNet50

The average number of images predicted correctly as their actual labels by ResNet50 was almost the same as by MobileNet (126 and 127 respectively). Inception and VGG19 have lower; 121 images were predicted correctly. The confusion matrices for InceptionV3, MobileNet, VGG19 and ResNet50 are presented in Figure 27.

PREDICTED class

ACTUAL class	Blistering	Crack	Efflorescence	Exposed Rebar	Honeycomb	Normal Class
Blistering	105	0	0	0	0	0
Crack	0	126	5	2	1	6
Efflorescence	0	7	128	1	2	3
Exposed Rebar	0	9	16	145	0	0
Honeycomb	5	9	4	5	113	3
Normal Class	5	9	5	0	0	113

PREDICTED class

ACTUAL class	Blistering	Crack	Efflorescence	Exposed Rebar	Honeycomb	Normal Class
Blistering	91	1	2	0	3	8
Crack	0	122	5	2	1	10
Efflorescence	0	2	129	3	2	5
Exposed Rebar	0	1	4	163	1	1
Honeycomb	0	2	0	3	133	1
Normal Class	0	4	1	3	0	124

PREDICTED class

ACTUAL class	Blistering	Crack	Efflorescence	Exposed Rebar	Honeycomb	Normal Class
Blistering	96	1	0	1	1	6
Crack	1	122	4	4	2	7
Efflorescence	2	15	108	4	2	10
Exposed Rebar	0	4	3	158	3	2
Honeycomb	1	4	3	4	126	1
Normal Class	2	8	3	2	0	117

PREDICTED class

ACTUAL class	Blistering	Crack	Efflorescence	Exposed Rebar	Honeycomb	Normal Class
Blistering	96	0	0	0	0	9
Crack	0	125	6	2	1	6
Efflorescence	1	6	127	1	0	6
Exposed Rebar	0	5	2	159	4	0
Honeycomb	0	4	3	3	128	1
Normal Class	0	7	2	2	0	121

Figure 27 - Confusion matrix - Top row from left to right: (a) InceptionV3, (b) MobileNet. Bottom row from left to right: (c) VGG19, (d) ResNet50

Finally, the four models were tested in nine unseen images, randomly collected from the web. The results for InceptionV3, MobileNet, VGG19 and ResNet50 are shown in Figure 28, Figure 29, Figure 30 and Figure 31 respectively. ResNet misclassified only one of the nine unknown images, followed by InceptionV3, which misclassified two images. MobileNet and VGG19 predicted wrongly three and four images respectively.

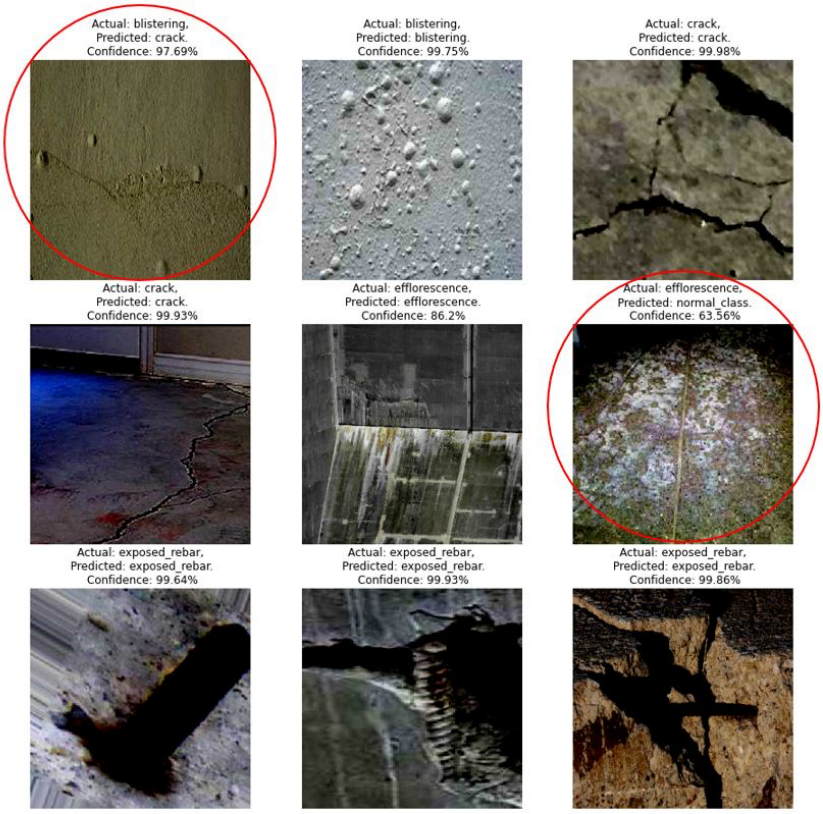


Figure 28- Performance of InceptionV3 for 9 unseen images

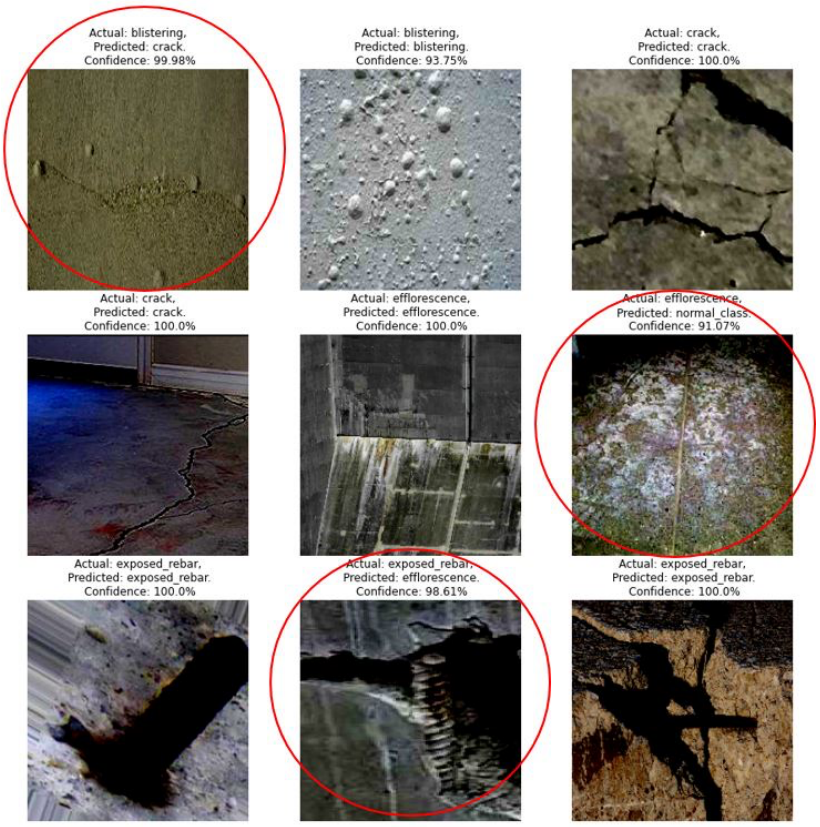


Figure 29 - Performance of MobileNet for 9 unseen images

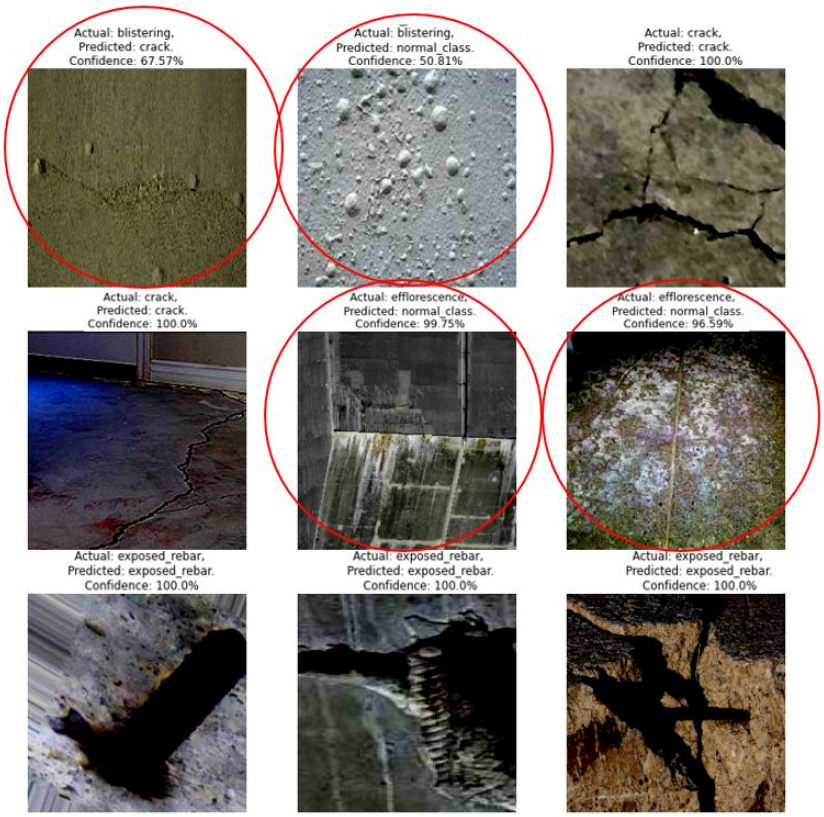


Figure 30 - Performance of VGG19 for 9 unseen images

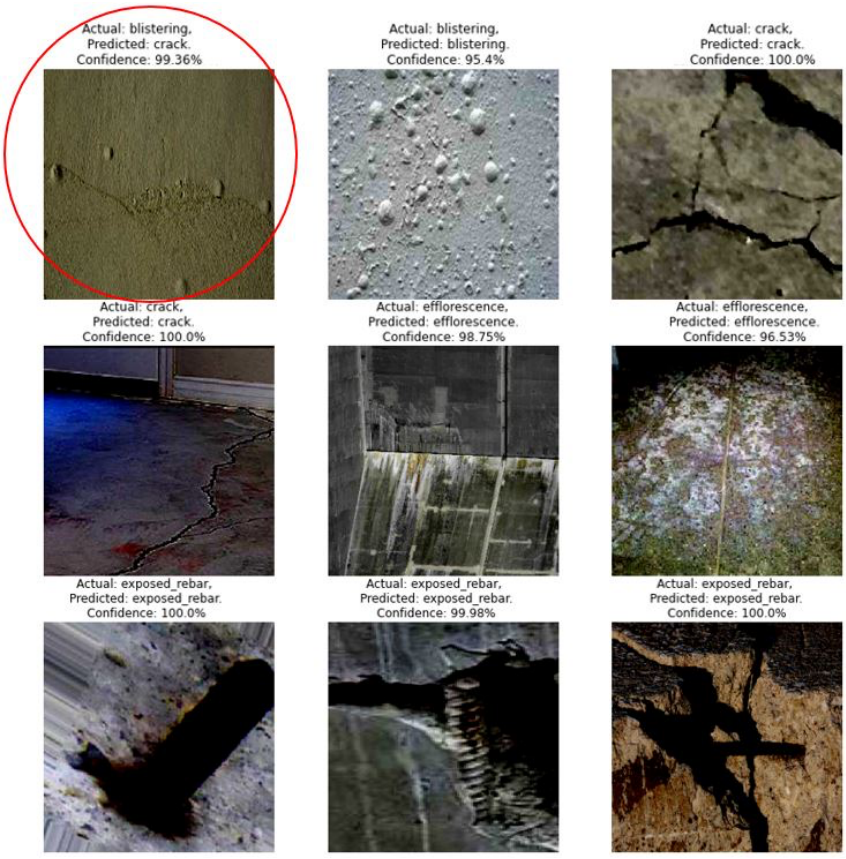


Figure 31 - Performance of ResNet50 for 9 unseen images

5.5 Licensing

The Visual QC component is a closed source component.

5.6 Installation Instructions

The Visual QC component is accessible via a rest API, thus no installation or downloading of component is required.

5.7 Development and integration status

The Visual Quality Control is currently under development. The alpha version of the tool has already been developed and focuses on the defects related to concrete surfaces. The main functionalities such as processing an image, identifying the type of the structure, loading the appropriate trained model, detecting potential defects and generating the QC results have already been implemented and are ready to be used. In addition, a first version of the API has been developed and its documentation has been presented in this deliverable. Regarding the second release of the component in M24, the intention is to focus on the defects related to steel surfaces as well as to expand the existing concrete-related data (additional data or additional defect types) and the binary classifier, which identifies the type of the structure. Furthermore, while the Visual QC component interacts with the DT Platform, tests will be concentrated to establishing the communication and the correct data exchanging with the DT Platform (i.e. receiving image data through an API).

5.8 Requirements Coverage

The Visual QC component is delivered as a back-end COGITO solution; however, it covers several of the requirements defined in D2.1 and D2.4.

Table 11 presents the Stakeholders Requirements documented in D2.1, which are relevant to the Visual QC component [27]. COGI-CS-1 and COGI-CS-4 are covered simply by the programming language selected for development. COGI-CS-5 is not tested but it is considered to be achieved, however runtimes for Python and Tensorflow exist for MAC-OS. COGI-QC-8, COGI-QC-9 and COGI-QC-11 are considered partially achieved while the as-built data is processed, and potential concrete defects are detected. Finally, COGI-QC-21 is achieved by using the relevant libraries (such as PIL) which handle images in PNG, JPEG format. COGI-QC-22 is not yet supported but it could be handled frame by frame.

Table 11 – Visual QC component: Stakeholders' Requirements coverage from D2.1

ID	Description	Type	Priority	Status
COGI-CS-1	Runs on desktop or laptop PC	• Operational	Must	Achieved
COGI-CS-4	Runs on Windows	• Operational	Must	Achieved
COGI-CS-5	Runs on Mac	• Operational	Could	Untested
COGI-QC-8	Supports systematic quality control on earthworks, substructure, concrete works	• Performance	Should	Partially achieved
COGI-QC-9	Supports systematic quality control on weld points and connection points	• Functional	Should	Partially achieved
COGI-QC-11	Automates QC-related activities	• Functional • Operational	Must	Partially achieved
COGI-QC-21	Handles images in PNG/JPEG format	• Functional	Must	Achieved
COGI-QC-22	Handles videos in XML and	• Functional	Could	Not yet supported

	AVI/MPG/MP4 format			

The functional and non-functional requirements, which are relevant to the Visual QC component, were documented in D2.4 [28] and are presented in Table 12. Req-1.1 is partially achieved, while primary API tests have been already done to connect the Visual QC with other endpoints. However, extra tests must be done to achieve the connection of the component with the specific DT Platform endpoint. Req-1.2 is also partially achieved, while a model for detecting concrete defects has already been trained. Req-1.3 is well covered by this version of the component and Req-1.4 is partially achieved and will be completed in the second release of the component. Req-2.1 is achieved while the alpha version of the application is already developed. Req-2.2 will be covered in the second release of this component. Req-2.3 is partially achieved by this version of the component. Req-2.5 is covered while the code could be modified in order to support additional structural surfaces and defect types. Regarding the Req-2.4 and Req-2.6, a strong GPU is used for training the models and image processing.

Table 12 – Visual QC component: Functional and Non-Functional Requirements coverage from D2.4

ID	Description	Type	Status
Req-1.1	Connect to DT Platform	Functional	Partially achieved
Req-1.2	Defects' detection based on rules	Functional	Partially achieved
Req-1.3	Defects' annotation	Functional	Achieved
Req-1.4	Defects' notification to be used by WODM	Functional	Partially achieved
Req-2.1	Web-based Application	Non-Functional	Achieved
Req-2.2	Security	Non-Functional	Not yet supported
Req-2.3	The component must provide APIs for all expected operations	Non-Functional	Partially achieved
Req-2.4	The component must have low latency	Non-Functional	Achieved
Req-2.5	Scalability	Non-Functional	Partially achieved
Req-2.6	Strong CPU is needed for image processing	Non-Functional	Achieved

5.9 Assumptions and Restrictions

The first version of the Visual Quality Control component is accompanied by certain assumptions and restrictions, which are presented in the following:

- The current version of the component is oriented to concrete surfaces and defects. Therefore, it supports the defect detection only on concrete surfaces and not on steel surfaces. This functionality will be implemented in the second version of the module (M24).
- The current version of the component identifies the type of the structure (concrete/ steel) based on preliminary binary dataset. This will be expanded and refined in the second version of the component (M24).
- The current version of the module's REST API supports queries regarding the metadata exchange with the DT Platform. Additional tests regarding the collection of the processed image will be included in the second version of the Visual QC component.
- The current version of the component is tested on these artificial datasets and not on real case data acquired on site. The intention is to increase the variety and to enrich the testing part of the datasets with random images collected under different weather conditions (rainy, cloudy etc.), cameras parameters (such as resolution, shooting range, lighting), construction elements (e.g., retaining walls, columns, beams etc.) etc.

6 Conclusions

This deliverable introduced the main functional components of Visual Quality Control component and their use. Information regarding the development tools, the data exchange and an alpha version of the API were also presented.

As documented in this deliverable (D5.3), the Visual QC is a web service, which analyses the as-built visual data (2D images) and predicts potential defects on concrete and steel surfaces. It is composed of four sub-components: the DT Platform connector, the Structural type-based selector, the Defect detector and the QC report generator. Within the Visual QC component appropriate trained model is loaded to analyse the processed image provided by the Visual Data Pre-processing module via the DT Platform and detect potential concrete or steel defects in construction. The results are pushed back to the DT Platform for further exploitation from other COGITO visualization components (DCC and DigiTAR).

The work presented here introduces a first version of the Visual Quality Control component. Thus far, the component has been focused on the concrete surfaces and tested only with basic artificial test data. In the future and as COGITO tools evolve, more tests will be executed to improve and refine the component and to support the communication with the other COGITO tools. The second version of the Visual QC component will be provided at M24 and will be tested on the pre-validation (T8.2) and validation (T8.4) sites with real case data during M21-30 and M28-34 respectively.

References

[1] Druzhkov, P. N. , Kustikova, V. D., "A Survey of Deep Learning Methods and Software Tools for Image Classification and Object Detection," *Pattern Recognition and Image Analysis,* vol. 26, pp. 9-15, 2016.

[2] Hung, P. D., Su, N. T., Diep, V. T., "Surface Classification of Damaged Concrete Using Deep Convolutional Neural Network," *Pattern Recognition and Image Analysis,* vol. 29, p. 676–687, 2019.

[3] Krishna, S.T., Kalluri, H.K., "Deep Learning and Transfer Learning Approaches for Image Classification," *International Journal of Recent Technology and Engineering,* vol. 7, no. 5S4, 2019.

[4] Koch, C., Georgieva, K., Kasireddy, V., Akinci, B., Fieguth, P., "A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure," *Advanced Engineering Informatics,* vol. 29, no. 2, pp. 196-210, 2015.

[5] Abu, M., Amir, A., Lean, Y. H., Zahri, N.A.H., Azemi, S. A., "The Performance Analysis of Transfer Learning for Steel Defect Detection by Using Deep Learning," in *5th International Conference on Electronic Design (ICED)*, 2020.

[6] Bukhsh, Z. A., Jansen, N., Saeed, A., "Damage detection using in-domain and cross-domain transfer learning," *Neural Computing and Applications,* vol. 33, pp. 16921-16936, 2021.

[7] D3.7-COGITO, "Deliverable 3.7: Visual Data Pre-processing Module v1," 2022.

[8] Hüthwohl, P., Lu, R., Brilakis, I., "Multi-classifier for reinforced concrete bridge defects," *Automation in Construction,* vol. 105, 2019.

[9] Mundt, M., Majumder, S., Murali, S., Panetsos, P., Ramesh, V., ""Meta-learning Convolutional Neural Architectures for Multi-target Concrete Defect Classification with the COncrete DEfect BRidge IMage Dataset," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[10] Hüthwohl, P., Brilakis, I., "Detecting healthy concrete surfaces," *Advanced Engineering Informatics,* vol. 37, pp. 150-162, 2018.

[11] D5.1- COGITO, "Deliverable 5.1: Innovative Scan-vs-BIM- based Geometric QC component v1," 2022.

[12] Silva, W.R.L.d., Lucena, D.S.d, "Concrete Cracks Detection Based on Deep Learning Image Classification," *Proceedings,* vol. 2, 2018.

[13] Coca, G.L., Romanescu, S.C., Botez, S.M., Iftene, A., "Crack detection system in AWS Cloud using Convolutional neural networks," *Procedia Computer Science,* vol. 176, pp. 400-409, 2020.

[14] Bu, G.P., Chanda, S., Guan, H., Jo, J. , Blumenstein, M., Loo, Y.C., "Crack detection using a texture analysis-based technique for visual bridge inspection," *Electronic Journal of Structural Engineering,* vol. 14, pp. 41-48, 2015.

[15] Lee, J., Kim, H.S., Kim, N., Ryu, E.M., Kang, J.W., "Learning to Detect Cracks on Damaged Concrete Surfaces Using Two-Branched Convolutional Neural Network," *Sensors,* vol. 19, 2019.

[16] Liu, Y., Yao, J., Lu, X., Xie, R., Li, L., "A deep hierarchical feature learning architecture for crack segmentation," *Neurocomputing,* vol. 338, pp. 139-153, 2019.

[17] Wei, F., Yao, G., Yang, Y., Sun, Y. , "Instance-level recognition and quantification for concrete surface bughole based on deep learning," *Automation in Construction,* vol. 107, 2019.

[18] Gao, M., Wang, X., Zhu, S., Guan, P., "Detection and Segmentation of Cement Concrete Pavement Pothole Based on Image Processing Technology," *Mathematical Problems in Engineering,* 2020.

[19] Feng, C., Liu, M.Y., Kao, C.C., Lee, T.Y., "Deep Active Learning for Civil Infrastructure Defect Detection and Classification," in *ASCE International Workshop on Computing in Civil Engineering*, 2017.

[20] Fu, J., Zhu, X., Li, Y., "Recognition Of Surface Defects On Steel Sheet Using Transfer Learning," *CoRR,* vol. abs/1909.03258, 2019.

[21] Li, M., Wang, X., Ma, Z., "Steel defect detection with high-frequency camera images FA 19-20 CS 229 Project".

[22] Wang, S., Xia, X., Ye, L., Yang, B., "Automatic Detection and Classification of Steel Surface Defect Using Deep Convolutional Neural Networks," *Metals,* vol. 11, 2021.

[23] 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.," *Computer-Aided Civil and Infrastructure Engineering,* vol. 33, pp. 731-747, 2018.

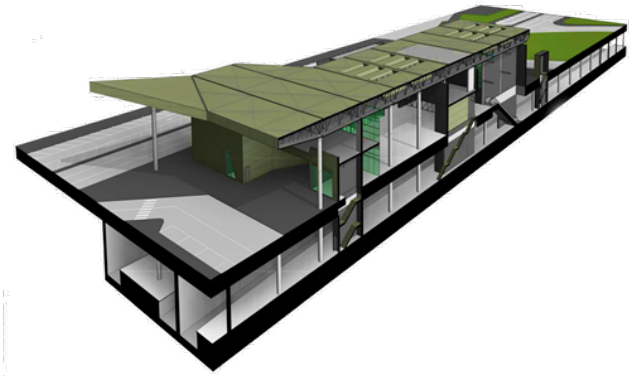
[24] Huethwohl, P., "Cambridge Bridge Inspection Dataset [Dataset]," *https://doi.org/10.17863/CAM.13813,* 2017.

[25] [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/offline-data-augmentation-for-multiple-images/.

[26] [Online]. Available: https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/.

[27] D2.1-COGITO, "Deliverable 2.1: Stakeholder requirements for the COGITO system," 2021.

[28] D2.4-COGITO, "Deliverable 2.4: COGITO System Architecture v1," 2021.



COGITO

CONSTRUCTION PHASE DIGITAL TWIN MODEL

cogito-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310