INCAD: 2D VECTOR DRAWINGS CREATION USING INSTANCE SEGMENTATION

Thodoris Betsas*[1], Andreas Georgopoulos[1], Anastasios Doulamis[1]

[1] National Technical University of Athens, School of Rural, Surveying and Geoinformatics Engineering, Lab of Photogrammetry, Zografou Campus, Heroon Polytechniou 9, 15780, Zografou, Athens, Greece
betsasth@mail.ntua.gr, drag@central.ntua.gr, adoulam@cs.ntua.gr

Commission II

KEY WORDS: 2D instance segmentation, 2D vector drawings, cultural heritage, YOLOv8, orthophotos

ABSTRACT:

Orthoimages are a common product used as a base in CAD software for vectorization purposes. In fact, vectorization of orthoimages constitutes a tedious and labour-intensive process which should be supervised by experts e.g. architects, chemical engineers etc. On the one hand, deep learning algorithms are used extensively nowadays achieving high quality results. On the other hand, deep learning algorithms require a huge amount of manually annotated data to be trained on, which is a very difficult process especially at pixel level applications like semantic segmentation and instance segmentation. However, the transformation of 2D CAD drawings into a suitable deep learning dataset (CAD2DLD) is underexplored ignoring a large source of data, created by experts. In this effort, the InCAD algorithm is proposed, which aims to automatically create 2D vector drawings using the YOLOv8 instance segmentation algorithm which was trained on CAD2DLD data. Additionally, a methodology for transforming 2D CAD drawings into a suitable deep learning dataset for instance segmentation, is presented. Finally, the proposed workflow is evaluated on the creation of 2D vector drawings of stones of a fortification wall achieving promising results (78.34 mIoU).

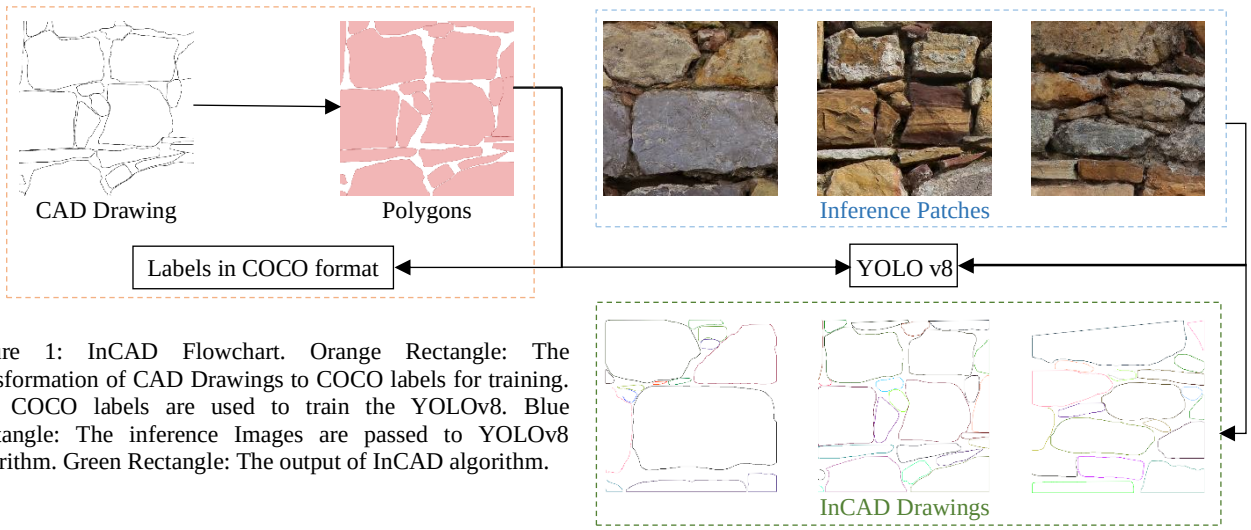


Figure 1: InCAD Flowchart. Orange Rectangle: The transformation of CAD Drawings to COCO labels for training. The COCO labels are used to train the YOLOv8. Blue Rectangle: The inference Images are passed to YOLOv8 algorithm. Green Rectangle: The output of InCAD algorithm.

1. INTRODUCTION

Geometric documentation of cultural heritage buildings could be achieved using either conventional or modern photogrammetric approaches in combination with post-processing steps, providing many products to the users like textured 3D models, orthophotos and 2D architectural vector drawings. There are many researches investigating the use of deep learning techniques on cultural heritage data for 3D documentation (Agrafiotis, Talaveros and Georgopoulos, 2023), and inspection (Tzortzis *et al.*, 2022) among others, using either conventional RGB images or orthophotographies.

Orthophotography combines the valuable information of texture with the ability to perform accurate measurements. However, they are commonly used as raster maps for the creation of 2D vector drawings e.g., architectural, material etc. More precisely, the 2D vector drawings are created by manually vectorizing the orthophotos using Computer Aided Design (CAD) software, which is a time-consuming and labour-intensive process. The output of this process is a CAD drawing file which contains the vectorized objects assigned to different layers. Consequently, the CAD drawings could provide both geometric i.e., vector like lines, polygons etc. and semantic information i.e., the layer names. To be more specific, each CAD drawing has different layers representing the available data and the vectorized objects. For instance, the available data could be an orthophoto of a fortification wall and the vectorized objects could be the "stones", "mortar", "wood" etc. parts of the wall. The main idea of this paper is to investigate the use of CAD drawings as a training dataset for deep learning instance segmentation algorithms in order to automate the creation of 2D architectural vector drawings. Deep learning instance segmentation algorithms seek a huge amount of ground truth data to be trained on. Well known deep learning datasets e.g. COCO (Lin *et al.*, 2015), provide a large number of images accompanied with their labels in a ".txt" file. Thus, the semantic and geometric information of CAD drawings should be transformed into a suitable structure in order to be used as a training dataset of instance segmentation algorithms. To be more specific, the visual information of the orthophotos should be combined with the geometric and semantic

information of the CAD layers to create a suitable dataset for training deep learning algorithms.

InCAD (Figure 1) is a simple yet efficient approach which automatically creates 2D vector drawings using an instance segmentation algorithm e.g., YOLOv8 (Jocher, Chaurasia and Qiu, 2023) which has been trained exploiting the layers created by experts in CAD format. Although InCAD uses YOLO v8 for instance segmentation, the presented methodology is compatible with any instance segmentation algorithm.

In fact, deep learning algorithms require vast amount of annotated data to be trained on. Meanwhile, orthophotographs are manually vectorized by experts, to create architectural drawings, especially in the cultural heritage domain. However, the 2D CAD data remains underexplored as a source of training data for deep learning algorithms.

In this paper, the use of 2D CAD drawings as training data for automated creation of architectural vector drawings is investigated. InCAD uses the YOLOv8 algorithm to create vector drawings in .dxf format. YOLOv8 was trained on CAD data created by experts, which was first transformed into a suitable dataset for training deep learning algorithms.

To sum up the contributions of this paper are:

- The proposed methodology which transforms 2D CAD drawings to deep learning datasets (CAD2DLD).
- The automatic creation of 2D vector drawings using instance segmentation algorithms trained on CAD2DLD data, with promising results.

2. RELATED WORK

In this effort we investigate the creation of 2D vector drawings using instance segmentation deep learning algorithms trained using CAD data. Many efforts presented so far aim to automate the creation of 2D-3D vector drawings. In fact, many of them incorporate 2D and 3D edge detection techniques, because the edges could be easily transformed into vectors. Hence, the related work is organized as follows. Firstly, a brief analysis of instance segmentation algorithms is presented. Finally, a brief analysis of 2D – 3D vector drawing creation, using edges and novel computer vision techniques, is included.

2.1 2D Instance Segmentation

Image pixels could be grouped into sets of pixels with similar characteristics performing a segmentation of the image. However, segmenting an image does not include a semantic meaning about the groups. Semantic segmentation techniques (Long, Shelhamer and Darrell, 2015; Chen *et al.*, 2018) have been proposed to assign a meaning to pixels using informative labels like "car", "person" etc. In fact, some of the semantic categories, for example the "car", could have multiple instances in the image while others like "sky" could not. Semantic segmentation techniques cannot distinguish the instances of each category. To achieve this, object detection techniques (Girshick *et al.*, 2013; Ren *et al.*, 2017), are combined with semantic segmentation techniques to mask the instances of each class, performing an instance segmentation of the image (Hariharan *et al.*, 2014). Hafiz and Bhat, 2020 present a review of instance segmentation algorithms. In this effort, the YOLOv8 (Jocher, Chaurasia and Qiu, 2023) algorithm, created by Ultralitics is used. YOLO is a family of algorithms used for object detection, instance segmentation etc. (Bochkovskiy, Wang and Liao, 2020; Jocher, Chaurasia and Qiu, 2023). Redmon *et al.*, 2016 presented the first YOLOv1 algorithm for object detection. Specifically, YOLOv1 divides the given image into a grid. Then, for each grid cell a set of bounding boxes is predicted. Each bounding box has a confidence score. The confidence score is crucial for YOLO because based on that and the class probabilities the algorithm assesses the predicted boxes.

2.2 2D – 3D vector drawing creation

Various 2D edge detection operators have been proposed so far e.g., Canny (Canny, 1983), Prewitt (Prewitt et.al, 1970) etc. These methods exploit the convolution operation to extract the 2D edges. Nowadays, deep learning algorithms for 2D edge detection have been proposed, solving some of the limitations of the traditional methods.

Xie and Tu, 2015 proposed HED a convolutional neural network for 2D edge detection, build upon a modified VGGNet (Simonyan and Zisserman, 2015) architecture. For example, they "trimmed" the last pooling and fully connected layers of VGG to create fine 2D edge maps and to reduce the complexity of the algorithm. Poma, Riba and Sappa, 2020 proposed DexiNed an architecture composed of multiple learning layers for 2D edges detection. During inference DexiNed creates a series of edge maps which are finally combined to the final output.

Bazazian, Casas and Ruiz-Hidalgo, (2015) presented a 3D edge detection technique method which uses the covariance matrix of point neighborhoods. Firstly, the kNN algorithm is applied to define the neighborhood of each 3D point. Afterwards the covariance matrix of each neighborhood is calculated. At the end, the points are classified as "edge points" or "other" based on the eigenvectors and eigenvalues of the covariance matrix of their neighborhood. The performance of the method was evaluated using different metrics e.g., Precision, Recall and F1-score. Dolapsaki and Georgopoulos, (2021) proposed a method which detects 3D edges using digital images with known exterior orientation. More precisely, they first define the 2D edge on the digital images and define the plane on which the perspective center of the image and the 2D edge lay. Inevitably, the 3D edge points of the 2D edge lie on the same plane and hence could be extracted. Betsas and Georgopoulos, (2022) proposed 3DPlan a 3D edge detection and vectorization framework. 3DPlan first segments the given images using 2D edge detection techniques creating 2D edge maps. Then, the RGB images are enriched with the 2D edge maps creating four-channel images. Afterwards, the enriched images are fed into SfM-MVS algorithms to extract the 3D edges points. The output of 3DPlan is a .dxf file with the vectorized 3D edges.

Apart from the edge detection techniques, novel computer vision algorithms could be used for automatic architectural vector drawing creation. Agrafiotis, Talaveros and Georgopoulos, (2023) presented a method for automated creation of architectural drawings using conditional generative adversarial networks (cGANs). The input of their approach is an orthophoto. Then cGAN uses the given orthophoto and tries to create the 2D architectural drawings by mimicking the CAD drawing. The trained algorithm was evaluated diversly. Firstly, a visual evaluation using different orthophotos was made with promising results. Also, informative conclusions were drawn by sending a questionary to end user experts in which they tried to distinguish the ground truth from the cGAN, drawings.

3. METHODOLOGY

Creating an instance segmentation dataset using CAD drawings has multiple benefits for many applications. In this paper the created dataset is used to train the YOLOv8 algorithm to automatically generate 2D CAD drawings of stones. Firstly, a CAD drawing to Deep Learning Dataset (CAD2DLD) pipeline is presented, which is used to create the training dataset. At the inference step i.e., when the trained algorithm is applied on new images, the generated instance masks should be transformed to

CAD drawings (IS2CAD), creating the final products of InCAD algorithm. The following paragraphs describe the CAD2DLD and IS2CAD workflows.

The scope of CAD2DLD step is to transform the CAD drawing to a suitable training dataset for deep learning algorithms. First and foremost, the CAD drawing is stored in ".dxf" format. Then the orthophoto and the CAD vectors are imported to a GIS software using the local coordinate system of the CAD drawing. In this effort the QGIS software is used. Afterwards the line vectors are transformed to polygons using the built-in QGIS algorithm "lines2polygons" creating a new file in ".shp" format.

InCAD aims to use the created polygons as instance segmentation masks of the underlying orthophoto. However, the coordinates of the polygons which are referenced to the local projective coordinate system should be transformed to pixel coordinates in order to be used as masks of the orthophoto. Additionally, the created orthophoto is a 74022 x 11194 pixels image which cannot be used directly into the training and inference procedure of the deep learning algorithm because most of them are fed with smaller images, for example 640x640 pixels. Thus, the created polygons and the orthophoto should be cropped into patches and each polygon transformed to the coordinate system of each patch.

Firstly, the boundary of the orthophoto is calculated using the QGIS software. Afterwards, a grid with the desired image dimensions e.g., 512x512, 640x640 etc. is created. Finally, the orthophoto and the polygons are cropped to multiple patches simultaneously, using each cell of the grid as crop boundary. The cropped image patches are stored in ".tif" while the cropped polygons patches to ".geojson", format. The polygons' coordinates into the "geojson" files, are referenced to the local coordinate system of the CAD drawing and so they should be transformed to pixel values using the following equations:

$$X^p = \frac{X^C - X^R}{pixel\ size};\ Y^p = \frac{Y^R - Y^C}{pixel\ size} \quad (1)$$

where X^p, Y^p = Polygons Pixel Coordinates.
X^C, Y^C = Polygons Local Coordinates.
X^R, Y^R = Local Coordinates of the center of the patch upper left pixel.
pixel size = Orthophoto pixel size.

InCAD uses the YOLOv8 instance segmentation algorithm which requires mask coordinates from 0 to 1. So, a coordinates normalization procedure is followed the transformation in pixel coordinates step, using the following equations:

$$X^N = \frac{X^P}{w};\ Y^N = \frac{Y^P}{h} \quad (2)$$

where X^N, Y^N= Normalized Coordinates.
X^P, Y^P= Pixel Coordinates.
w , h = Image Width and Height.

Finally, the normalized values should be stored in COCO format to be accessible from the YOLOv8 instance segmentation algorithm. To be more specific, each created patch is accompanied by a ".txt" file. Each row of the file contains a label-mask pair in "$L\ X^{N1}, Y^{N1}, X^{N2}, Y^{N2}, \dots,\ X^{Nn}, Y^{Nn}, X^{N1}, Y^{N1}$" format, where L is the label number and the rest is the normalized values of the mask.

In fact, the creation of CAD drawings using the instance segmentation masks, during inference, is the opposite workflow of the CAD2DLD workflow. Firstly, the orthophotos are fed into the InCAD algorithm resulting in a set of instance masks for each image which finally transformed to the 2D vector drawings. To be more specific, the normalized coordinates of each mask are firstly transformed to pixels and then to the local projective coordinate system of the CAD drawing, using the equations included in eq. 1 and eq. 2. Thus, 2D drawings for each orthophoto are created in ".dxf" format which is the main product of the InCAD algorithm. A detailed analysis of the performance of the algorithm under different training schemes using an orthophoto of a fortification wall is presented in Section 4.

4. EXPERIMENTS AND RESULTS

InCAD algorithm was trained on CAD2DLD data in order to automate the creation of 2D vector drawings of stones. In general, instance segmentation algorithms are not pretrained on classes like "stones", "mortar", "wood", "columns" etc. but in completely different ones like "cars", "person" etc. and so a retraining exploiting the new dataset should be performed by initializing the YOLOv8 with the provided weights. YOLOv8 algorithm has five different versions (n, s, m, l, x). Each version has a different number of parameters, execution time, metrics and floating-point operations per second (FLOPs). Hence, the first step is to select the appropriate version for the current application. In this effort the "m" version is selected, as the version with average complexity. There is no need for real-time execution of the InCAD algorithm and also by testing the "m" version we could roughly estimate if the application needs a more complicated i.e., "l", "x" or a simpler i.e., "n". "s" version of YOLOv8. The following subsections describe the CAD dataset used during the experiments and the assessment of the YOLOv8 and InCAD algorithms.

4.1 CAD and Training Datasets

In our experiments we use a dataset of a fortification wall from Chios, Greece, acquired during its geometric documentation by the Laboratory of Photogrammetry SRSGE NTUA in 2016 (Tapinaki *et al.*, 2019). The CAD dataset contains an orthophoto 74022 x 11194 pixels and a drawing in ".dwg" format (Figure 2).

Figure 2: The CAD dataset used for training and testing the InCAD algorithm. (a) Orthoimage, (b) Close view of the Orthoimage, (c) CAD drawing, (d) Close view of the CAD drawing

Vectorizing an orthophoto is a tedious process which could be performed only by the supervision of specialists like architects. Each category in the created orthophoto is represented using a different layer e.g., "stones", "mortar" etc. in the CAD drawing. In this experiment we only use the "stones" layer and so the

InCAD algorithm automatically creates the 2D drawing of them. As described in Section 3 the CAD dataset should be transformed to the training dataset.

Firstly, the orthophoto and the polygons are cropped simultaneously, to create the patches. Based on the size of the grid, the number of the images varies in the generated dataset. In this effort 640x640 images are created for training. which is the default image dimensions of YOLO. The idea behind the selection of 640x640 grid is the assessment of InCAD performance using the default dimensions and altering only some of the model hyperparameters like the learning rate (LR), the training epochs and the optimizer e.g., Adam (Kingma and Ba, 2014) or Stochastic Gradient Descent SGD (Ruder, 2017). Almost 2000 patches were created by cropping the given orthophoto, using its boundary, into patches of 640x640 pixels. Some of them do not have valuable information e.g., only background info, due to the orthophoto boundary calculation. Hence, around 1700 images were finally used into the training dataset. After the patches' generation, the 90%, 5% and 5% i.e., 1530, 85 and 85, of them were grouped randomly into the "train", "validation" and "test" sets, respectively (Table 1).

	Images	Train	Validation	Test
Percentage (%)	100	90	5	5
Numb. of Imgs.	1700	1530	85	85

Table 1: Train, Validation and Test sets of the created Dataset.

Finally, the computers' specifications used for training are presented in Table 2.

	CPU	GPU	RAM
Asus ROG Zephyrus G15	Ryzen 9 6900HS Z3+ 3,3 GHz	NVDIA GeForce RTX 3060 (6GB)	16 GB DDR5

Table 2: The Computer Specifications

4.2 Evaluation Metrics

After the training, using different schemes, the YOLOV8 algorithm was applied on the test set to assess its performance using Accuracy, Precision, Recall, F1-score and intersection over union (IoU) metrics using the following equations:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + TN} \quad (5)$$

$$F1\ score = 2\frac{Precision\ x\ Recall}{Precision + Recall} \quad (6)$$

$$IoU = \frac{Intersection}{Union} \quad (7)$$

4.1 YOLOv8 Training Scenarios for Instance Segmentation

In this subsection, the training scenarios of the YOLOv8 algorithm using different combinations of hyperparameters are presented. Although Ultralitics, the creators of YOLOv8, provide many hyperparameters for tuning purposes, the assessment conducted during this endeavor was made only by changing the optimizer, the number of epochs and the learning rate (lr) ones which are some of the "main" hyperparameters for training purposes. The performance of the instance segmentation algorithm is in relationship to the effectiveness of InCAD algorithm. However, an in-depth analysis of YOLOv8 performance it is out of the scope of this paper. So, YOLOv8 and other instance segmentation algorithms will be further analyzed in future in combination with modern computer vision techniques. In this paper, six different scenarios were evaluated to improve the products of InCAD algorithm and to investigate the future improvement of YOLOv8 performance. Table 3 displays the different scenarios used during training assessment of the YOLOv8 algorithm using the CAD2DLD data described in Subsection 4.1.

Scenario	Optimizer	Epochs	Learning Rate
1 (default)		100	0.01
2	SGD	100	0.003
3		100	0.001
4 (default)		100	0.001
5	ADAM	100	0.003
6		130	0.003

Table 3: The Different Training Scenarios

Firstly, YOLOv8 is trained on the "train" set of images using the ground truth annotation data and then it is evaluated on the "validation" set of images to estimate its performance. Based on the calculated metrics of each epoch, YOLOv8 tries to improve the next epoch performance. When the algorithm trained for the entire set of epochs, the weights of the best epoch are stored as the final model for each scenario. The metrics calculated during training are good indicators in order to improve the performance of YOLOv8, by tuning the hyperparameters and not assessing the performance of the model to unseen images. Thus, an objective assessment of the performance of YOLOv8 should be performed.

4.2 YOLOv8 Objective and Subjective Evaluation and Best Model Selection for the InCAD algorithm.

In this subsection, an assessment of the best models of each scenario of the YOLOv8 algorithm is presented, applied on the test set of the created dataset to select the optimal model for the InCAD experiments. Stone instance segmentation is a binary class segmentation problem i.e., the pixels should be categorized as stones or background. Thus, the mean value of each metric in Table **4**, e.g. mAcc, is the mean value of the metrics calculated for the images of the test set, and not the mean metric e.g., mAcc between the different classes. Table 4 presents the mean Accuracy (mAcc), mean Precision (mPrecision), mean Recall (mRecall), mean F1-Score and mean Intersection Over Union (mIoU) metrics for each scenario applied on the images of the test set.

Sc.	mAcc	mPrecision	mRecall	mF1	mIoU
1	82.40	**84.75**	88.02	**87.85**	76.84
2	82.38	84.44	89.33	85.68	77.18
3	**83.17**	84.28	**90.95**	86.42	**78.34**
4	78.67	83.95	82.48	84.42	71.95
5	78.95	83.88	83.38	82.42	72.58
6	81.29	84.65	86.16	86.78	75.44
mScs.	81.14	84.32	86.72	85.59	75.38

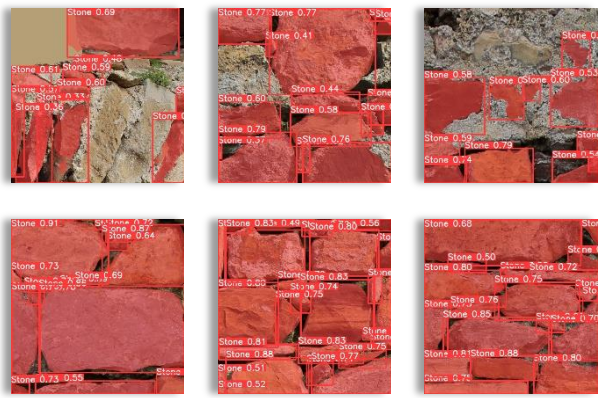
Table 4: The Evaluation Metrics of YOLOv8 algorithm applied on the Test set for each Scenario. Where mScs: mean value for each metric taking into account all the scenarios, Scenario 3 was

selected as the instance segmentation model of InCAD algorithm (Light Gray)

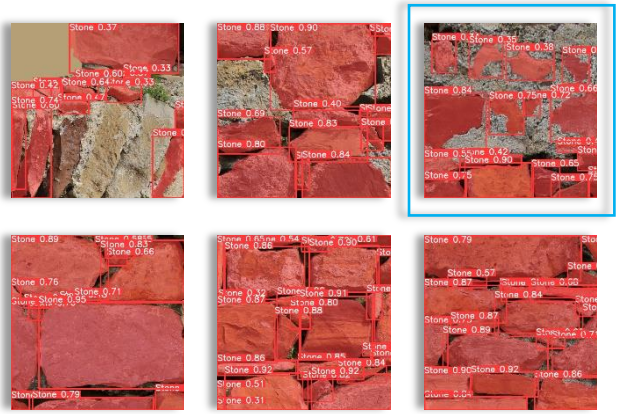
Often, a safe conclusion about the trained model could be drawn by examine each calculated metric. In fact, the assessment of the performance of a model it is a more complicated process including objective and subjective analysis i.e., the metrics calculations and visual comparisons, among others. Also, the assessment process should take into account the specific application with its particularities. For example, in this application most of the pixels of the patches included in the created dataset are “stone” rather than “background” pixels and thus the accuracy, precision, recall and F1-score metrics may not be a true indicator of the model’s performance. However, the calculated metrics are not so high i.e., over 90%, hence the complexity involved in the automatic creation of 2D architectural vector drawings of stones and in general, is confirmed. Furthermore, the intersection over union (IoU), which is the main evaluation metric for semantic, instance and panoptic segmentation tasks due to the “spatial” criteria involved in the calculation, gives promising results. Apart from the objective analysis of each scenario a subjective analysis i.e., visual comparisons, is included to select the optimal instance segmentation model for the InCAD algorithm. For a fair comparison between the scenarios, some of the worst (first line) and some of the best (second line) predictions for each scenario are displayed in Figure 3.

Scenario 1
The Predicted Instance Segmentation Masks

Scenario 2
The Predicted Instance Segmentation Masks

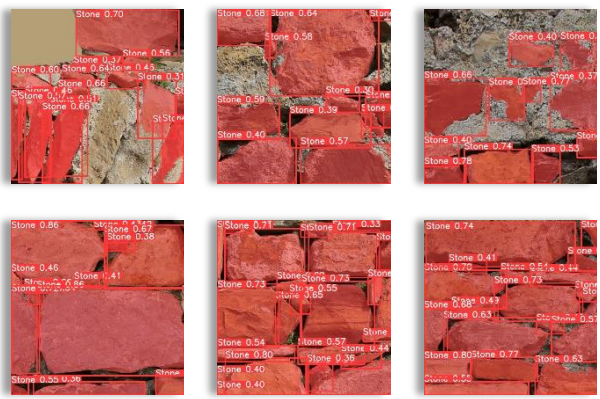


Scenario 3
The Predicted Instance Segmentation Masks



Scenario 4
The Predicted Instance Segmentation Masks

Scenario 5
The Predicted Instance Segmentation Masks



Scenario 6
The Predicted Instance Segmentation Masks

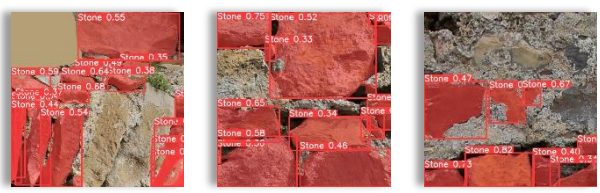
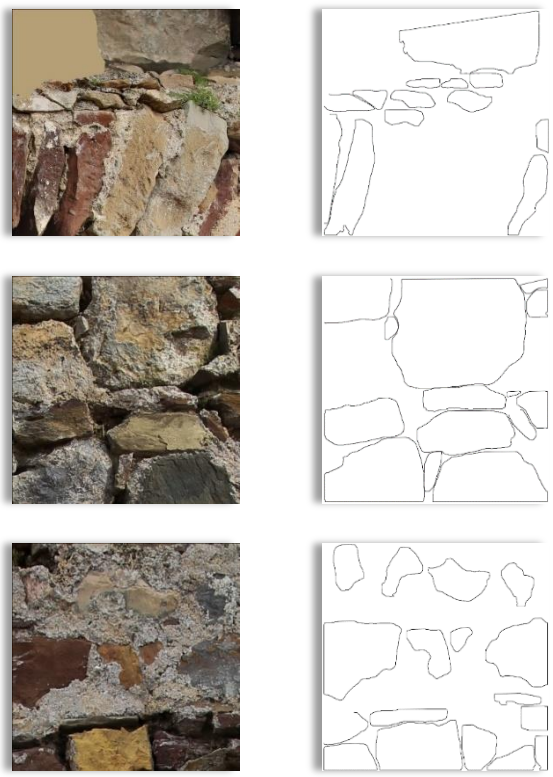


Figure 3: Subjective Evaluation of Training Scenarios. The first row presents some of the worst instance segmentation results while the second some of the best instance segmentation results.

In general, the most difficult predictions are those including mortar areas and small stones. On the one hand the instance segmentation algorithm struggles to find the stones that are surrounded by mortar areas and especially the small ones. On the other hand, it achieves high quality results in demanding images as depicted in the second row of Figure 3. To sum up, after the quantitative and qualitative analysis of the training scenarios the scenario 3 gives the most promising results, especially at the most difficult cases (blue rectangle Figure **3**). Thus, the best model of scenario 3 was chosen as the YOLOv8 instance segmentation model of InCAD algorithm.

In fact, the instance segmentation algorithm is a crucial part of InCAD algorithm. However, a subjective and objective analysis of the InCAD main products should be performed, using the selected YOLOv8 model. In Figure 4 the architectural vector drawings of some of the worst and best predictions of the training scenario 3 are presented.

InCAD Architectural Vector Drawings



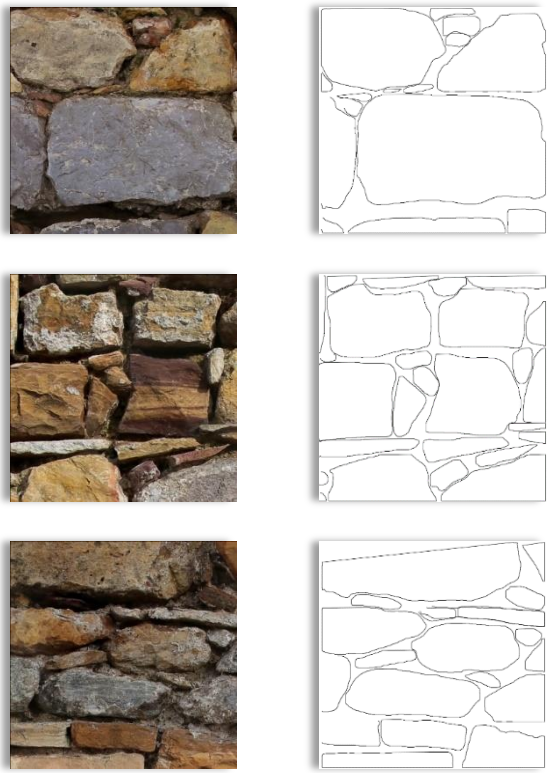


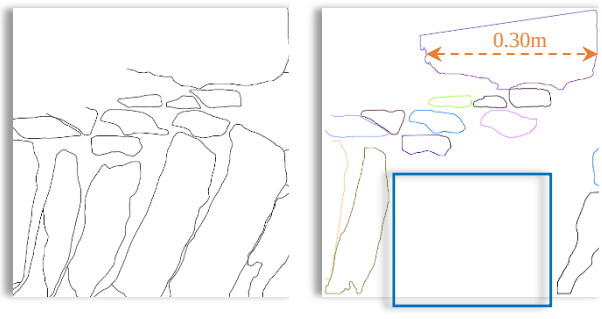
Figure 4: 2D Vector Drawings in .dxf format created using the InCAD algorithm. Orthophotos patches (left), generated 2D vectors in CAD format (right).

5. DISCUSSION OF RESULTS

In general, the InCAD algorithm gives promising results as depicted in Figure **3** and Figure **4**, and analyzed in detail in the previous sections. However, a critical analysis of the results and the development of InCAD algorithm should be performed for a fair assessment of the algorithm. First and foremost, YOLOv8 algorithm was trained using the CAD data depicted in Figure 2. For a fair comparison a test set of orthophotos was created. During training the test of images was not involved in any step and thus the quantitative (Table **4**) and qualitative (Figure **3** and Figure **4**) evaluation provide a fair assessment of the workflow. Figure **4** depicts some of the worst and the best results of InCAD algorithm applied on the orthophotos of the test set. However, a critical visual comparison between the created 2D vector drawings (Figure **4** (right column)) and the ground truth CAD drawings (Figure 2c, d) should be conducted (

Figure **5**).

Ground Truth Drawings VS InCAD Drawings



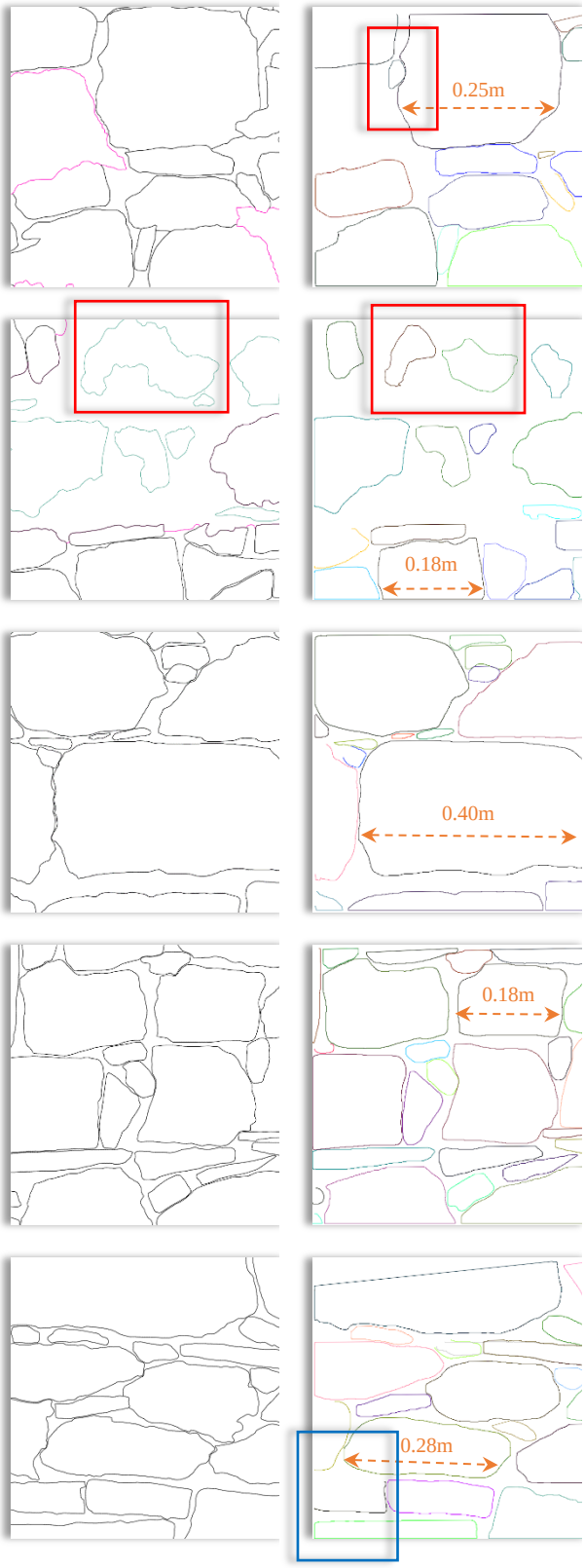


Figure 5: Comparison of ground truth vector drawings (left) with the InCAD drawings (right). The different colours in the left columns indicates different layers in CAD i.e., mortar (pink) and stones which are covered by mortar or are farther than most of the stones (green). Each stone is depicted with different colour (right). The scale of the depicted objects is given with orange.

By comparing the created 2D architectural vector drawings with the ground truth vector drawings created by experts, informative outcomes can be drawn. On the one hand the InCAD algorithm miss to identify and vectorize stones (

Figure **5** blue rectangles). Additionally, misdetections in which the InCAD algorithm identifies two stones instead of one, are also occur (

Figure **5** red rectangles). On the other hand, the InCAD algorithm gives very good results in many easy but also in very demanding cases as depicted in

Figure **5**. Finally, the execution time of InCAD algorithm, without including the training phase, is less than 50 seconds for 80 drawings on a CPU. To sum up the InCAD algorithm gives very fast high quality results.

6. CONCLUSION AND FUTURE WORK

In this paper, the InCAD algorithm is introduced, which aims to contribute to the creation of 2D architectural vector drawings automatically, using deep learning instance segmentation algorithms e.g., YOLOv8. On the one hand deep learning algorithms seek huge amount of data for training purposes. On the other hand, the 2D CAD drawings created by vectorizing orthophotographs are underexplored as training data for deep learning algorithms. Hence, a methodology to transform 2D CAD drawings into suitable deep learning datasets (CAD2DLD), is presented.

In this effort, the InCAD algorithm was applied on a binary instance segmentation problem. Thus, an application using multiple classes could be performed in the future, to examine the performance of InCAD in a problem with higher complexity. Additionally, experiments using larger images with overlap or using augmentation techniques, will be made in the future to investigate the refinement of instance segmentation process. Furthermore, experiments using different instance segmentation algorithms could be conducted in future to find the best one for the needs of InCAD algorithms. To conclude, many experiments were conducted to evaluate the performance of InCAD algorithm using the CAD2DLD data demonstrating the promising results of the algorithm.

ACKNOWLEDGEMENTS

This work is funded by the European Union Funded project euPOLIS ”Integrated NBS- based Urban Planning Methodology for Enhancing the Health and Well-being of Citizens: the euPOLIS Approach”, under the Horizon 2020 program H2020-EU.3.5.2., grant agreement No. 869448. The content of this publication is the sole responsibility of NTUA and does not necessarily reflect the opinion of the European Union.

REFERENCES

Agrafiotis, P., Talaveros, G. and Georgopoulos, A. (2023) Orthoimage-to-2D Architectural Drawing with Conditional Adversarial Networks’, *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, X-M-1–2023, pp. 11–18. Available at: https://doi.org/10.5194/isprs-annals-X-M-1-2023-11-2023.

Bazazian, D., Casas, J.R. and Ruiz-Hidalgo, J. (2015) ‘Fast and robust edge extraction in unorganized point clouds’, in *2015 international conference on digital image computing: techniques and applications (DICTA)*. IEEE, pp. 1–8.

Betsas, T. and Georgopoulos, A. (2022) 'Point-Cloud Segmentation for 3D Edge Detection and Vectorization', *Heritage*, 5(4), pp. 4037–4060. Available at: https://doi.org/10.3390/heritage5040208.

Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y.M. (2020) 'YOLOv4: Optimal Speed and Accuracy of Object Detection'. arXiv. Available at: https://doi.org/10.48550/arXiv.2004.10934.

Canny, J.F. (1983) *Finding Edges and Lines in Images*. Massachusetts Inst of Tech., Cambridge Artificial Intelligence Lab.

Chen, L.-C. *et al.* (2018) 'DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4), pp. 834–848. Available at: https://doi.org/10.1109/TPAMI.2017.2699184.

Dolapsaki, M.M. and Georgopoulos, A. (2021) 'Edge Detection in 3D Point Clouds Using Digital Images', *ISPRS International Journal of Geo-Information*, 10(4), p. 229.

Girshick, R. *et al.* (2013) 'Rich feature hierarchies for accurate object detection and semantic segmentation'. arXiv. Available at: https://doi.org/10.48550/arXiv.1311.2524.

Hafiz, A.M. and Bhat, G.M. (2020) 'A survey on instance segmentation: state of the art', *International Journal of Multimedia Information Retrieval*, 9(3), pp. 171–189. Available at: https://doi.org/10.1007/s13735-020-00195-x.

Hariharan, B. *et al.* (2014) 'Simultaneous Detection and Segmentation', in D. Fleet et al. (eds) *Computer Vision – ECCV 2014*. Cham: Springer International Publishing (Lecture Notes in Computer Science), pp. 297–312. Available at: https://doi.org/10.1007/978-3-319-10584-0_20.

Jocher, G., Chaurasia, A. and Qiu, J. (2023) 'YOLOv8'. Available at: https://github.com/ultralytics/ultralytics (Accessed: 20 April 2023).

Kingma, D.P. and Ba, J. (2014) 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* [Preprint].

Lin, T.-Y. *et al.* (2015) 'Microsoft COCO: Common Objects in Context'. arXiv. Available at: https://doi.org/10.48550/arXiv.1405.0312.

Long, J., Shelhamer, E. and Darrell, T. (2015) 'Fully convolutional networks for semantic segmentation', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Poma, X.S., Riba, E. and Sappa, A. (2020) 'Dense extreme inception network: Towards a robust cnn model for edge detection', in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1923–1932.

Prewitt, J.M. and others (1970) 'Object enhancement and extraction', *Picture processing and Psychopictorics*, 10(1), pp. 15–19.

Ren, S. *et al.* (2017) 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), pp. 1137–1149. Available at: https://doi.org/10.1109/TPAMI.2016.2577031.

Ruder, S. (2017) 'An overview of gradient descent optimization algorithms'. arXiv. Available at: http://arxiv.org/abs/1609.04747 (Accessed: 11 January 2024).

Simonyan, K. and Zisserman, A. (2015) 'Very Deep Convolutional Networks for Large-Scale Image Recognition', *arXiv:1409.1556 [cs]* [Preprint]. Available at: http://arxiv.org/abs/1409.1556 (Accessed: 22 April 2022).

Tapinaki, S. *et al.* (2019) '3D Image Based Geometric Documentation of a Medieval Fortress', *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W9, pp. 699–705. Available at: https://doi.org/10.5194/isprs-archives-XLII-2-W9-699-2019.

Tzortzis, I.N. *et al.* (2022) 'Automatic Inspection of Cultural Monuments Using Deep and Tensor-Based Learning on Hyperspectral Imagery', in *2022 IEEE International Conference on Image Processing (ICIP)*. *2022 IEEE International Conference on Image Processing (ICIP)*, pp. 3136–3140. Available at: https://doi.org/10.1109/ICIP46576.2022.9897527.

Xie, S. and Tu, Z. (2015) 'Holistically-Nested Edge Detection', In *Proceedings of the IEEE international conference on computer vision* (pp. 1395-1403).