EUROGRAPHICS Workshop on Graphics and Cultural Heritage (2023)
A. Bucciero, B. Fanini, H. Graf, S. Pescarin, and S. Rizvic (Editors)

R-CNN based Polygonal Wedge Detection Learned from Annotated 3D Renderings and Mapped Photographs of Open Data Cuneiform Tablets

E. Stötzner, T. Homburg, J.P. Bullenkamp and H. Mara

Abstract

Motivated by the demands of Digital Assyriology and the challenges of detecting cuneiform signs, we propose a new approach using R-CNN architecture to classify and localize wedges. We utilize the 3D models of 1977 cuneiform tablets from the Frau Professor Hilprecht Collection *available as pen data. About 500 of these tablets have a transcription available in the* Cuneiform Digital Library Initiative *(CDLI) database. We annotated 21.000 cuneiform signs as well as 4.700 wedges resulting in the new open data* Mainz Cuneiform Benchmark Dataset *(MaiCuBeDa), including metadata, cropped signs, and partially wedges. The latter is also a good basis for manual paleography. Our inputs are MSII renderings computed using the* GigaMesh Software Framework *and photographs having the annotations automatically transferred from the renderings.*

Our approach consists of a pipeline with two components: a sign detector and a wedge detector. The sign detector uses a RepPoints model with a ResNet18 backbone to locate individual cuneiform characters in the tablet segment image. The signs are then cropped based on the sign locations and fed into the wedge detector. The wedge detector is based on the idea of Point RCNN approach. It uses a Feature Pyramid Network (FPN) and RoI Align to predict the positions and classes of the wedges. The method is evaluated using different hyperparameters, and post-processing techniques such as Non-Maximum Suppression (NMS) are applied for refinement. The proposed method shows promising results in cuneiform wedge detection. Our detector was evaluated using the Gottstein system and with the PaleoCodage encoding.

Our results show that the sign detector performs better when trained on 3D renderings than photographs. We showed that detectors trained on photographs are usually less accurate. The accuracy on photographs improves when trained, including 3D renderings. Overall, our pipeline achieves decent results, with some limitations due to the relatively small amount of data. However, even small amounts of high-quality renderings of 3D datasets with expert annotations dramatically improved sign detection.

CCS Concepts

• Computing methodologies** → **Object detection; Machine learning; • Applied computing** → **Archaeology;

1. Introduction

Accessing the earliest written texts has long been a challenging task that has fascinated computer scientists since the dawn of computing, including the era of Zuse's computers.[BM22] Developing effective optical character recognition (OCR) approaches for cuneiform tablets is both a formidable challenge and a source of inspiration. Because cuneiform is a script that can only be deciphered in 3D with proper illumination, working with photographs poses additional difficulties due to limitations such as distracting colors and inadequate illumination caused by the curved shapes of the tablets. However, recent advances in 3D capture techniques have made it more accessible and affordable to obtain 3D models of cuneiform tablets, although few of these models are currently available as open data. By using high-quality curvature rendering techniques based on *Multi-Scale Integral Invariant* (MSII) filtering within, e.g., the open-source *GigaMesh Software Framework*[†], researchers can provide valuable support to the human experts in this field. In this contribution to Digital Assyriology, we use a combination of raster image data, including both photographs and renderings, and we have begun to explore the potential applications of artificial intelligence systems. It should be noted that applying these systems directly to the meshes of 3D datasets can be challenging, but they have shown promising results in areas such as period classification of tablets.[BM20]

Detecting cuneiform signs from images is an essential first step in an automated analysis of cuneiform texts in a natural language processing setup as another important research topic of Digital Assyriology. Correct recognition of cuneiform signs paves the way for further computational analysis of cuneiform textual content,

[†] https://gigamesh.eu

DOI: 10.2312/gch.20231157

such as transliteration assignments, part of speech tagging classifications, and automated machine translation tasks. The main challenge in this OCR task is the segmentation of cuneiform signs, which has been examined in previous work [DKMO20], and in the precise detection of cuneiform wedges that are visible on the clay tablet itself. A missing or slightly differently positioned cuneiform wedge could, depending on spatio-temporal considerations of the circumstances of the cuneiform tablet or depending on the context, be interpreted as a different sign variant [Hom21] and thus, yield a different interpretation of the tablet's content by a machine learning algorithm. In addition, because expert image annotations are typically unavailable, virtually all previous approaches suffered from a scarcity of expert-annotated training data. As [DKMO20] points out, deep learning of cuneiform signs requires a substantially large amount of training data in the form of bounding boxes, which experts should annotate at best. To our knowledge, the currently ongoing *Cune-IIIF-orm* project of the Ghent University in Belgium is the only one collecting annotations using polygonal selections from images of the *Portable Light Dome* [VVP*18], as inspired by [HZBM22].

Most of the previous approaches have used photographs to train machine-learning models [DKMO20; HFP*22; RFW*22; SAP*23] A weak supervision approach by applying a large dataset of transliterations in addition to the image annotations for Cuneiform sign detection was introduced by [DKMO20]. [RFW*22] is the closest related work using 63 annotated images of 13 Hittite Cuneiform tablet fragments. Also, using renderings of 3D models, they have shown that illumination augmentation by using 3D renderings has the potential to improve the results of Cuneiform sign classification. However, they used only one 3D rendering for data augmentation, while we used only 3D renderings for our training. To our knowledge, the wedges' detection has only been investigated for horizontal wedges.[HFP*22] Their basic dataset contains photographs of eight tablets from different languages, cropped into 823 labeled images with 7355 horizontal wedge annotations, and achieved a precision of 0.745 and a recall of 0.705. However, the article lacks clarity on whether the training, validation, and test sets were entirely separated, which may lead to unreliable model performance results. The merging of subsets in [HFP*22] may occur if overlapping areas are inadvertently included in different subsets, highlighting the need for careful attention to dataset construction. We extended this task to a horizontal, vertical, and oblique wedge detection task. Furthermore, we discuss the use of the Gottstein system [Got12] and the PaleoCodage encoding [Hom21]. None of the previous approaches have provided a fully manually annotated dataset. In most cases, only a subset of the training data has been annotated and made available. However, it is important to note that these datasets served only as examples and were not easily reusable. In addition, it is worth mentioning that the open access material provided by [DKMO20] has unfortunately disappeared from the web, further limiting the availability of valuable resources. In this work, we focus on the following novel approaches:

- We use and provide a large corpus of fully manually annotated 3D renderings as machine-learning training data.
- We apply an R-CNN architecture to classify wedges and predict the location of irregular quadrilaterals.
- We discuss the impact of the Gottstein system and PaleoCodage encoding on the wedge detector.
- We introduce a pipeline approach for detecting wedges on an entire segment.
- Evaluation providing results for the sign localization, the global performance of the wedge detection, and separated by classes, and the quality of the pipeline results.

The following section describes our approach to cuneiform wedge detection on the corpus described in Section 3. The described networks are trained from scratch, i.e., we only used our dataset and did not work with any pre-trained backbones.

2. Method

First, we describe the setup of our pipeline consisting of a sign detector described in Section 2.2, which feeds into a wedge detector Section 2.3.

2.1. Pipeline

Our approach to detecting and classifying wedges in an entire tablet segment image is a pipeline with two components (see Figure 1). First, the sign detector predicts the sign locations on the entire tablet segment. Based on this information, each sign is cropped in a single image.

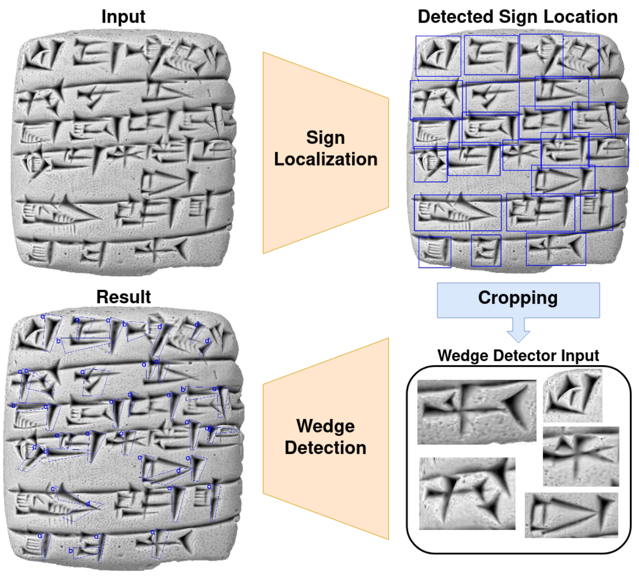


Figure 1: *Wedge Detection Pipeline*

Due to our setup, the network could benefit from a more focused training approach emphasizing wedge shapes. By utilizing a two-stage pipeline, we can present the network with individual cutouts based on sign localization, allowing it to focus on learning the specific characteristics of a wedge without the added complexity of dealing with hundreds of wedges simultaneously.

The need for this pipeline arises from the fact that attempts to detect wedges using the entire tablet segment as input resulted in inferior results compared to our approach. This is primarily due to the loss of information caused by resizing the input image, as tablet images are typically rendered at high resolutions (e.g., 600 *DPI*) with side lengths well over 1000 pixels. Technical limitations prevent us from using such large images directly as input for the CNNs.

In order to obtain the final detection result for the entire tablet, the wedge detector takes the predicted wedge positions and classes from the cutouts and calculates the global wedge positions based on the sign locations and the relative wedge positions within the cropped image.

By using this pipeline, we address the challenges posed by information loss during resizing and technical limitations in image size, ultimately achieving better results in wedge detection. Additionally, the focused training on wedge shapes improves the network's ability to learn and recognize the distinctive features of individual wedges.

2.2. Sign Localization

The Sign Localization is a single-class object detection task, where each character location is represented by a bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$. Due to the reusability of our Wedge Detector model (see Section 2.3), we decided to use *RepPoints* [YLH*19] as detection model with *ResNet18*[HZRS16] as a backbone network, where we used the layer $c4$ which results to a resolution of 64×64 of the feature map f. The use of a *Feature Pyramid Network (FPN)* [LDG*17] in the Sign Detector led to a worse result in our experiments. *RepPoints* is a one-stage anchor-free object detector that, for each object, predicts a set of k representation points and confidence values for $c+1$ classes, where c is the number of classes, and one is added for the background. The prediction is based on each position (x_f, y_f) of the feature map returned by the backbone. With the application of two non-shared subnets, the localization consists of two point sets $P_1(x_f, y_f)$ and $P_2(x_f, y_f)$, and the classification is performed. The set $P_1(x_f, y_f)$ is a result of k offsets Δx_f and Δy_f for each feature map point (cf. Equation (1))

$$P_1(x_f, y_f) = \{(x_f + \Delta x_{f_i}, y_f + \Delta y_{f_i})\}_{i=1}^{k} \tag{1}$$

These offsets are also used as input offsets of the deformable convolution layer in the classification and localization subnets. To refine the point positions, the localization subnet predicts the set $P_2(x_f, y_f)$ by k offsets $\Delta x'_f$ and $\Delta y'_f$, based on the points in P_1 (cf. Equation (2)).

$$\begin{aligned} P_2(x_f, y_f) = &\{(x_{p_1 i} + \Delta x'_{f_i}, y_{p_1 i} + \Delta y'_{f_i})\}_{i=1}^{k}, \\ &(x_{p_1 i}, y_{p_1 i}) \in P_1(x_f, y_f) \end{aligned} \tag{2}$$

For each experiment, $k = 9$ is used, following [YLH*19]. To train the network, these k points of the point set P_j for each feature map position (x_f, y_f) must be converted to a pseudo bounding box, where the min-max function is used for both dimensions, as shown in Equation (3).

$$\begin{aligned} \hat{x}_{min} &= \min_{x_p \in P_j(x_f, y_f)} (x_p) \\ \hat{y}_{min} &= \min_{y_p \in P_j(x_f, y_f)} (y_p) \\ \hat{x}_{max} &= \max_{x_p \in P_j(x_f, y_f)} (x_p) \\ \hat{y}_{max} &= \max_{y_p \in P_j(x_f, y_f)} (y_p) \end{aligned} \tag{3}$$

The localization loss is calculated by the smooth l_1 distance between the four bounding boxes describing values $x_{min}, y_{min}, x_{max}$ and y_{max} of the ground truth (GT) bounding boxes and the predicted pseudo bounding boxes. As a classification loss function, the Focal Loss [LGG*17] is used. As described in [YLH*19], the center of the GT bounding boxes are projected to the feature map position, and only for these feature map points the location loss of pseudo bounding boxes $(\hat{x}_{min}, \hat{y}_{min}, \hat{x}_{max}, \hat{y}_{max})$ by the min-max function based on P_1 is calculated. The second location loss, based on the P_2, is only calculated for those feature map points where the pseudo bounding box of P_1 has an intersection-over-union (IoU) value with the GT bounding box above the threshold θ_{TP}. Similarly, the classification loss is determined by the thresholds θ_{TP} and θ_{FP}, where all predicted boxes with an IoU value above θ_{FP} but below θ_{TP} belong to the GT 'background', and if the IoU value is above θ_{TP} the GT class corresponding to the bounding box is used for the classification loss calculation. Different from [YLH*19], we used $\theta_{FP} = 0.6$ and $\theta_{TP} = 0.7$ because the signs and wedges are densely placed. The original values $\theta_{FP} = 0.4$ and $\theta_{TP} = 0.5$ led to worse results for our task. Another difference is that our architecture contains dropout [SHK*14] with an extinction probability of 0.2 for the input and the first convolution of the backbone and dropout with a probability of 0.5 for each further convolution layer in the backbone and for the first-three convolution layers in the *RepPoints* architecture. As post-processing, we applied Non-Maximum Suppression (NMS) with a threshold of 0.1 IoU to keep boxes. After the hyperparameter optimization of the Stochastic Gradient Descent (SGD) optimizer, we set 0.001 as the learning rate, 0.9 as momentum, and no weight decay.

2.3. Wedge Detection

The Wedge Detection is defined as a multi-class object detection task, where the wedges are represented as irregular quadrilaterals specified by four corner points $(x_1, y_1, \ldots, x_4, y_4)$, hereafter called polygons, and an assigned class. Our method is oriented to *Point RCNN* [ZY22], a Region-based Convolutional Neural Network (R-CNN) approach to predicting rotated bounding boxes without the use of an angle parameter. As you can see in Figure 2, the wedge detector is divided into two stages, the Region Proposal Network (RPN), which provides a set of object proposals, and, further, the classification and position refinement stage.

To predict the potential locations of wedges, also called the regions of interest (RoI), the *RepPoints* object detector described in Section 2.2 is used. However, to deal with different sizes and overlapping wedges, a *FPN* is applied to the *ResNet18* [HZRS16] backbone. Different from [ZY22], the RPN is trained using bounding boxes calculated by the min-max function for each point of the

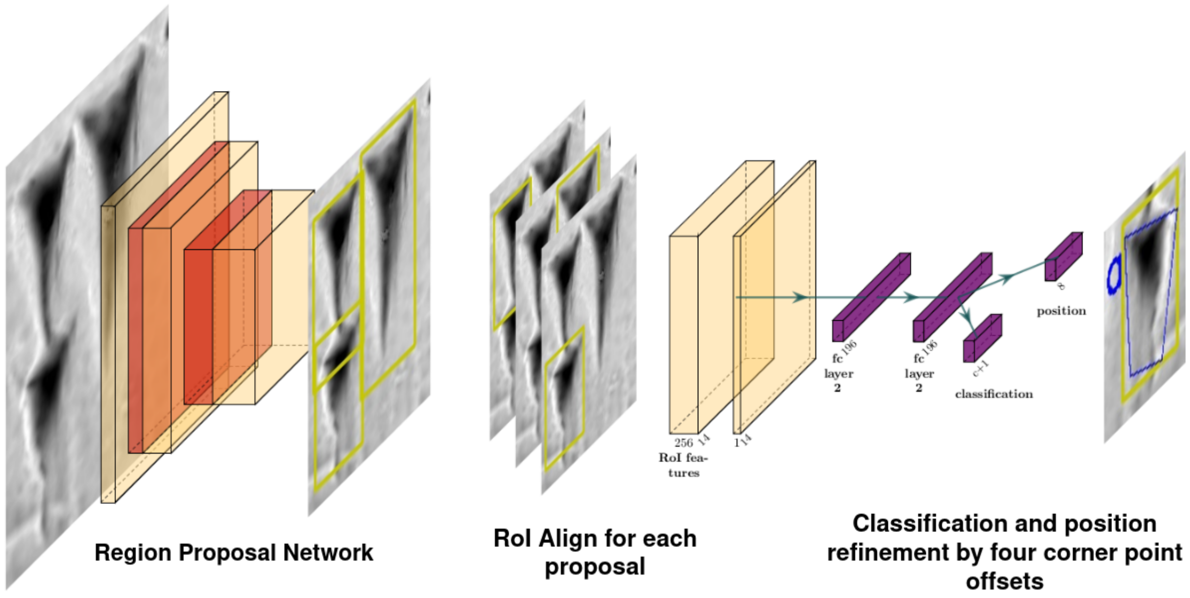

256
14
RoI features
1
14
fc
196
layer
2
fc
196
layer
2
c+1
classification
8
position
Region Proposal Network
RoI Align for each proposal
Classification and position refinement by four corner point offsets

Figure 2: *Wedge Detection architecture*

polygon. To assign the GT bounding box i for the first location loss and the classification loss to a feature map level l, we apply

$$l_i = \left\lfloor \log_2\left(\sqrt{\frac{w_i \cdot h_i}{s}}\right) \right\rfloor \tag{4}$$

where w_i is the width, h_i is the height, and s is a hyperparameter set to six for each of our experiments.

Based on the result of the RPN after a NMS with a threshold of 0.5, a RoI Align [HGDG17] is performed to extract a part of the feature map that is the input of the classification and position refinement stage. Due to the memory limitation and the training performance during the first epochs, only the best 500, but at least with a confidence of 0.5, proposals are used. Figure 2 shows the refinement stage of one *FPN* level, where the yellow-green boxes represent the RoIs. First, to reduce the extracted $14 \times 14 \times 256$ features to $14 \times 14 \times 1$ features, a 3×3 convolution with one-pixel padding is applied, followed by two fully connected layers. Then, the net is split into the classification output layer that returns the class probability of the c classes and the background, and the position output layer that returns eight offset values $\{\Delta x_i, \Delta y_i\}_{i=1}^{4}$ of the four corresponding proposed bounding box corner points B. Therefore, the resulting polygon corner points

$$\begin{aligned} R = &\{(x_{b_i} + \Delta y_i, y_{b_i} + \Delta y_i)\}_{i=1}^{4}, \\ &(x_{b_i}, y_{b_i}) \in B \end{aligned} \tag{5}$$

Those points adapt to the wedge shape, e.g., the head of the wedge, and to the rotation. The polygon annotations are sorted clockwise to learn these offsets, starting with the point in the upper left corner so that the dataset has a consistent point order.

In addition to the *RepPoints* loss functions, a classification loss and a location loss are added. These two losses are calculated if the IoU value between the GT bounding box and the proposal is above θ_{TP}. While the RPN classification loss is only relevant for foreground and background, the refinement classification loss is a Focal Loss for all the c classes. The refined position is penalized by the smooth l_1-distance between the GT polygon points and the predicted points. As a post-processing, we applied NMS with the same threshold as in the Sign Detector but on polygons and separated by class. To calculate the IoU between polygons to process NMS and evaluate the model, we used the Python library *shapely* [GvdWV*23]. The hyperparameter optimization resulted in a 10^{-4} learning rate, 0.9 momentum, and 10^{-7} weight decay.

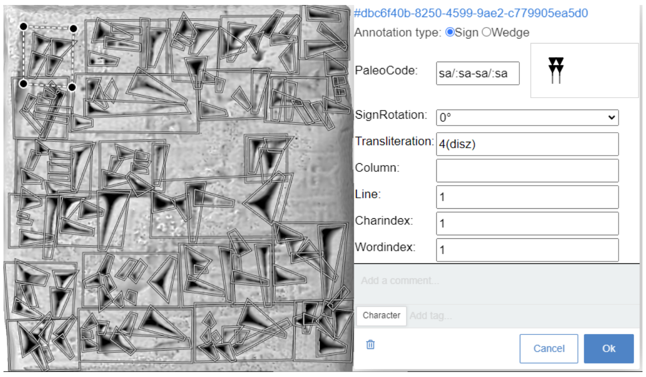
3. Data

We use training data from the *Frau Professor Hilprecht Collection of Babylonian antiquities* at the University of Jena. This collection was captured in 3D with the assistance of the *Max-Planck-Institut für Wissenschaftsgeschichte* (MPIWG) in Berlin and provided using a CC-BY license. The cleaned, orientated, cleaned, and filtered 3D data together with high-resolution renderings were published as *Heidelberg Cuneiform Benchmark Dataset* (HeiCuBeDa) [Mar19] for machine learning tasks with a CC-BY-SA license.

3.1. Data preparation

This dataset provides 3D models of 1977 cuneiform tablets in Jena spanning a significant amount of time periods of cuneiform sign history. At the time of writing, about 500 texts had a transliteration available at the Cuneiform Digital Library Initiative (CDLI) [Eng16] available. The CDLI is one central online resource in *Digital Ancient Near Eastern Studies* (DANES) and has been used in related digital cuneiform projects such as [BCE*].

We used these transliterated cuneiform texts as the basis to conduct the annotation on 3D renderings of the different written surfaces (front, back, left, right, top, bottom) of the cuneiform tablets in the Cuneiform Annotator application[‡] also known as *Cuneur* shown in Figure 3.


#dbc6f40b-8250-4599-9ae2-c779905ea5d0

Annotation type: [✓ Sign] Wedge

PaleoCode:	sa/:sa-sa/:sa
SignRotation:	0°
Transliteration:	4(disz)
Column:	
Line:	1
Charindex:	1
Wordindex:	1

Figure 3: *Sign annotations as created in the Cuneiform Annotator for the corpus used in this publication, as exemplified on tablet HS 1001. The PaleoCode on the picture is shown for illustration purposes and has not been annotated for every cuneiform sign.*

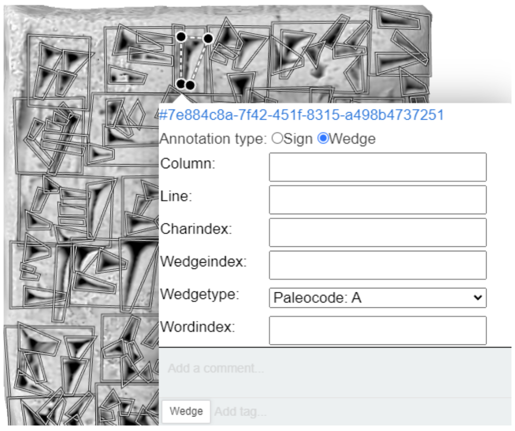
Each annotation comprises an index (`TabletID_surface_column_line_charindex`) that uniquely identifies every annotated cuneiform sign and even cuneiform wedge on the tablet and refers it to its position in the respective transliteration. In addition, the reading of the cuneiform sign is annotated. A reading of a cuneiform sign identifies the cuneiform sign uniquely, as it can be mapped to its sign name and its Unicode representation. We use the Nuolenna sign list[§] of about 11.000 sign readings to map the image annotation to Unicode code point and dismiss signs which cannot be determined this way from any classifications.

Furthermore, in addition to the sign annotations for 45 tablet segments of 27 Sumerian tablets and one Akkadian tablet, we provide the annotations of the associated wedges (cf. Figure 4), indexed as (TabletID_surface_column_line_charindex_wedgeindex). For this purpose, we marked the position as polygons, more precisely as irregular quadrilaterals, and we labeled the class using the PaleoCodage [Hom21] encoding.

Annotations are saved using the W3C Web Annotation Data Model [YSC17] in JSON-LD [CLK20], and besides cropped images of the 3D renderings comprise the dataset.

3.2. Dataset description

Using the aforementioned annotated data, we present the machine-learning dataset for this publication: The *Mainz Cuneiform Benchmark Dataset* (MaiCuBeDa)[¶], which consists of the following image components:


#7e884c8a-7f42-451f-8315-a498b4737251

Annotation type: Sign [✓ Wedge]

Column:	
Line:	
Charindex:	
Wedgeindex:	
Wedgetype:	Paleocode: A
Wordindex:	

Figure 4: *Wedge annotations as created in the Cuneiform Annotator for the corpus used in this publication, as exemplified on tablet HS 1001. The annotated PaleoCode signifies a vertical cuneiform wedge and can be reused as a Gottstein code in classifications as well*

- 27696 Cuneiform sign images and 5947 wedge images as PNG in two versions:
  - Polygon cropping: The exact polygon of the annotation is cropped, remaining parts become transparent
  - Image cropping: The bounding box around the annotation is cropped
- Line croppings: Lines in which the cuneiform signs appear
- Word croppings: Words in which the cuneiform signs appear

All aforementioned annotations have been generated for three different 3D rendering types of the HilprechtCollection tablets, the VirtualLight rendering, the MSII Filter rendering, and a mixture of both rendering types.

In addition, we provide metadata about the respective signs derived from tablet metadata descriptions by the Cuneiform Digital Library Initiative, CDLI. Together with the aforementioned annotations in JSON-LD, the metadata and image references create a linked open data graph, which we publish in the TTL format [Bec08].

For this wedge and character detection task, we use only the mixture 3D renderings of the front and back segments of the tablets, the annotated sign and wedge positions, and the corresponding labeled wedge classes in PaleoCodage. Fortunately, the labels can be easily converted to the Gottstein system by mapping: $e \to c, f \to d$, and $w \to c$. Although the seal wedges (x and y) are not part of the Gottstein system, we also add them to our dataset as Gottstein classes. The chart in Figure 5 provides the wedge class distribution of the complete dataset in PaleoCodage and also visualizes the shape and the alignment of the wedges per class. So our annotations do not have any instances of y.

At this point in our research, the sign labels were irrelevant, so we ignored them. As described in Section 3.1, the sign annotations refer directly to the transliteration of the segment. Unfortunately, this results in some visible signs in the image not being annotated

[‡] *Cuneur* – Cuneiform Annotator Showcase https://fcgl.gitlab.io/annotator-showcase/

[§] https://github.com/tosaja/Nuolenna

[¶] MaiCuBeDa: https://doi.org/10.11588/data/QSNIQ2 because they are not part of the transliteration. To reduce the annotation error within the dataset for sign detection, we manually sorted out the instances with many missed signs by subjective decision. However, there are still many segments containing not annotated signs because we did not want to minimize the dataset extremely. Due to the indispensability of the sign annotations for cutting the signs to train our model described in Section 2.3, each tablet is fully annotated with sign annotations.

annotations	# tablets
Sign	490
Sign + Wedge	28

(a) *Number of tablets with annotations*

annotations	# segments
Sign	873
Sign + Wedge	45

(b) *Number of segments with annotations*

	Total annotations
Signs	21228
Wedges	5556
Cropped images by sign annotations	864

(c) *Total number of used annotations*

Table 1: *Our Dataset in numbers*

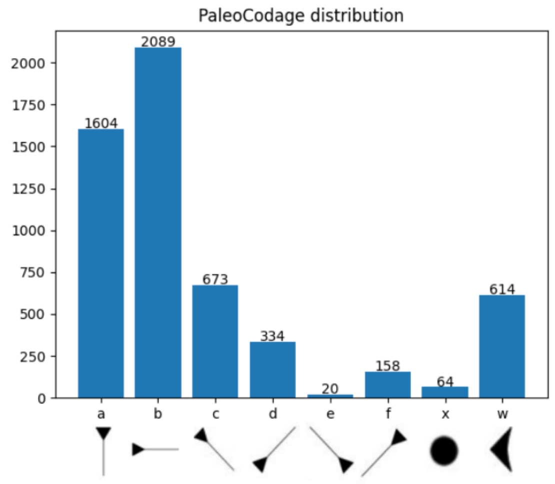


Figure 5: *Our class distribution in the PaleoCodage*

Table 1 provides an overview of the applied data, which is based on Sumerian tablets, apart from one Akkadian tablet, to train the neural networks described in Section 2. In addition to the annotated images of the 3D renderings, we extended our image data with the corresponding photographs published at the CDLI, which we mapped with the *Cuneur Transformer*[‖] in such a way that we can re-use the annotations, created on the 3D renderings, to train our machine learning models.

Furthermore, we created a subset with cropped images by using the sign bounding boxes, and for each cropped image, an additional annotation file containing the location and class of the wedges in that cropped image. As shown in Figure 5, the wedge classes have imbalanced distributions. Therefore during the training, the cutouts containing wedges of the minority class x in both systems were repeated by the factor 5. Since the 3D renderings are grayscale images, we converted the photographs to grayscale. As further preprocessing, we perform a normalization of the grayscale values and resize the images to 512×512 for the use of the Sign Detector and to 224×224 for the use of the Wedge Detector.

4. Results

This section introduces our evaluation methods and presents the results of sign localization and wedge detection that we achieved. We compare the training using 3D renderings as well as photographs and elaborate on the differences between these two media.

4.1. Evaluation

For our evaluation, we split our dataset by segments into training, validation, and test sets with a ratio of $2:1:1$, while we ensured that the wedge minority class of Gottstein and PaleoCodage x also satisfies this ratio. The crops for the wedge detection training were created based on the same dataset split, therefore the segments are separated as entire images and as cropped images. For character localization evaluation, we consider a prediction to be a true positive, if the intersection-over-union (IoU) between the predicted bounding box and GT bounding box is at least 0.5. True positives in wedge detection are predictions that reach at least an IoU between polygons of 0.4 and match the label of the ground truth. As the measurement to compare models during the optimization of the sign localization is the *mean* $F_1 - Score(mF_1)$, which is defined as the mean of the $F_1 - Score$ F_{1_i} for each image i of the set with the size n:

$$mF_1 = \frac{1}{n}\sum_{i=1}^{n} F_{1_i}$$

In the evaluation of the sign localization, the measurements *precision* and *recall* are also mean values. For each image, the measurement is calculated separately, and the mean is formed. The wedge detection is evaluated by the precision P of each class and by the mean of all precision values (mP) for the performance of the entire wedge detector.

4.2. Sign Detector

The evaluation results of the sign detector per dataset combination are shown in Table 2. As described in Section 3.1, the partial annotation of signs is a challenge of this dataset. This also leads to a difficult comparison with results from a fully annotated dataset, as correct predictions may be evaluated as false due to a missing ground truth annotation. Therefore, the actual results are better than the numbers. The model trained only on 3D renderings leads to the best results for 3D renderings in sign localization and predicts with a *mean recall* of 0.66 and a *mean precision* of 0.60.

A visual analysis of the results has revealed some instances of false classifications made by the detector. These errors can be attributed to various factors. First, the presence of damaged surface

[‖] https://gitlab.com/fcgl/cuneur-transformer parts and seals on the tablet often leads to incorrect detections. We can see that the detector performs better on smaller signs and is slightly worse on wider signs. Another notable observation, particularly prevalent in photographs, is that lateral signs are more poorly detected compared to signs located at the center of the fragment. This discrepancy could be due to variations in lighting conditions or the angle at which the photograph was taken. In addition, it has been noted in the evaluation of photographs that the written identification number of the tablet is sometimes mistakenly detected as a sign. Overall, it has been observed that the detector's predictions tend to be more accurate for smaller tablets, regardless of whether they are in the form of renderings or photographs. This suggests that the detector performs better when dealing with tablets that have a smaller size, likely due to the reduced complexity and clearer visibility of signs on such tablets. Although the sign detector has some weaknesses in the objective evaluation, the model shows applicable results in the visual examination and could be used in our pipeline to cut out signs from a complete segment as input of the wedge detector.

Figure 6 shows an example of the sign localization for a photo. This instance represents most of the false classifications. First, in lines two and three of the segment, you can see that a sign is detected as a false positive evaluated even though there is a sign. This is based on only partially annotated signs and also makes the evaluation more difficult. There is also a wide sign in line four, where the detector only predicted the first part of the sign. Furthermore, you can see two undetected lateral signs at the bottom of the image, which are not fully visible, and the obvious false positive detection on the fractured surface at the top of the segment. But overall, we were able to correctly detect most of the signs, including the two which are correct but are considered as false in this example.

4.3. Wedge Detector

The evaluation of the wedge detection task is divided into the application of the Wedge Detector (see Section 2.3) directly with cropped images by the annotations as described in Section 3.2 and in the utilization of the pipeline (see Section 2.1). In Table 3, the different precision per class for the best, determined by *mP*, PaleoCodage detector, and the Gottstein system detector, are shown. We can see that the wedge detector struggles, especially with the less common wedge classes, e.g., *e* and *x*, while it detects more common wedges like *a* and *b* better. The wedge detector results in Table 2, therefore, seem worse than they are, as the mean precision for the classes is not weighted according to the occurrence of the classes. Both systems were trained on mapped photographs and 3D renderings. It is noticeable that the class *a* has much higher precision than the other classes in both systems. For the other classes, which are part of both systems, the models have shown different behaviors. The prediction of *c* wedges is more accurate when trained and evaluated in the Gottstein system than in the PaleoCodage encoding. Furthermore, the minority class *x* is noticeable because the detector can only detect some instances within the test set when PaleoCodage is applied. It is important to note that the visual investigation of the results has shown that in most cases for the same wedge, an *a* or *b* is also predicted. In some cases, it is a challenge to detect only one class for a wedge. Especially in the PaleoCodage

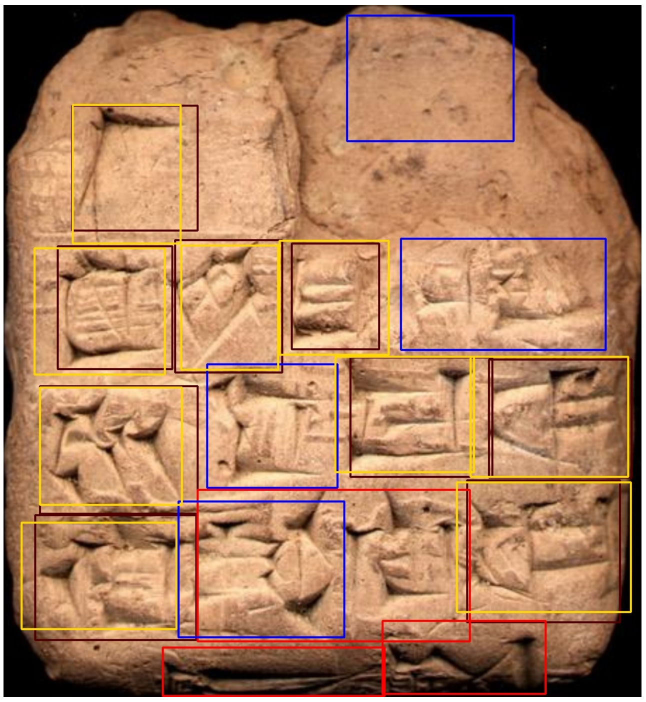


Figure 6: *Result of sign localization for a mapped photograph. This is an example of the false classifications with lateral signs, fragments, not annotated signs, and wide signs. Yellow: true positive, Blue: false positive, Red: not found signs (GT), Black: GT of correct detected Signs*

output, the mirrored class pair *d* and *f* are confounded. The other wedge classes that exist only in PaleoCodage are not predicted.

A weakness of the wedge detector is the low recall. The best PaleoCodage model achieves a $recall = 0.19$, and the best Gottstein model achieves a $recall = 0.15$. This also affects the pipeline results (see Figure 7). As seen in Table 4, the *mP* differs only slightly from the evaluation on cropped images, though the recall fell to 0.005. It is important to note that the *mP* measurement also disadvantages the precision of classes with a high occurrence. The visual analysis has shown that the detected wedges are distributed over the entire segment. This confirms that the sign localization within the pipeline provides cutouts that are usable input for the wedge detector.

4.4. Discussion of results

All of our detection models, even those trained only with photographs, perform better or equally well on 3D renderings (see Table 2). One reason for the performance gap between photographs and 3D renderings is based on the mapping between them. The transformation to process the mapping by the *Cuneur Transformer* can lead to small deviations in the annotated location, and a few lateral signs are generally not visible in the photographs. Especially for wedge annotations, these deviations could lead to inaccuracies. Due to the size and dense location of the wedges, a small transformation error led to incorrect location annotations. The gap could

Train set	**Test set**	**Sign Localization** mF_1	**Wedge Detector (Gottstein) mP**	**Wedge Detector (PaleoCodage) mP**
Incl. photos	Photos	0.45	0.39	0.33
	3D renderings	0.59	**0.52**	**0.43**
Only 3D renderings	Photos	0.10	0.28	0.25
	3D renderings	**0.61**	0.51	0.37
Only photos	Photos	0.41	0.21	0.18
	3D renderings	0.44	0.28	0.19

Table 2: *Training impact of the different dataset compositions. The training set inclusive photos means that the 3D renderings training set is expanded by the mapped photographs. The results of the wedge detector are obtained by evaluating the detector directly with cropped images.*

Wedge class	**PaleoCodage P**	**Gottstein system P**
a	0.76	0.76
b	0.59	0.61
c	0.52	0.45
d	0.61	0.8
e	0.0	—
f	0.75	—
w	0.0	—
x	0.22	0.0

Table 3: *Precision of the wedge detector for the Gottstein system and the PaleoCodage encoding by class. These results are based on applying the cropped images of the renderings by the sign annotations.*

also be explained by the much higher contrast, increased by the MSII filtering [Mar19], between the pressed wedges and the clay in 3D renderings than in photographs. It is also consistent with human perception of cuneiform tablets, as 3D renderings after MSII filtering are much more legible than photographs.

There are two most likely reasons for the weak prediction of lateral signs. First, the loss of information content due to the distortion of the sign by the curved writing surface. Second, most of the missed signs in the annotations are lateral signs, so the model may have learned not to predict signs on the side of the segment.

Test set	**PaleoCodage mP**	**Gottstein mP**
Photographs	0.12	0.22
3D renderings	0.34	0.36

Table 4: *Results Pipeline. Both components of the pipeline, the sign detector and wedge detector are trained on mapped photographs and 3D renderings.*

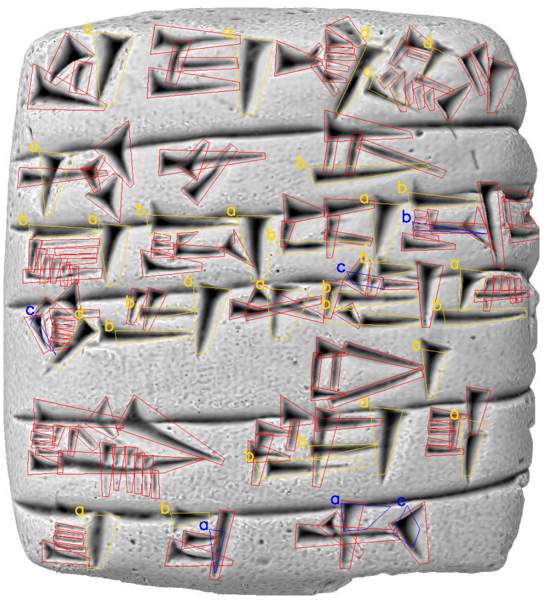


Figure 7: *Result of the Wedge Detection Pipeline in Gottstein system. Yellow: true positive, Blue: false positive, Red: not found wedges*

As described in Section 4.3, the wedge detector has a variance between the class precision and different behavior depending on the wedge classification system. Although the class *a* is not the most represented in the data set (see Figure 5), it is the best-predicted wedge type. One explanation may be that vertical wedges are distinguished from the others due to frequently defined prominent heads, and to locate these wedges, the polygons must not be rotated. On the other hand, class *b* also has this rotation advantage and is more represented in the dataset, but the precision is worse than the precision of *a*. This could be due to the type of occurrences on the tablets. Most occurrences are tiny; sometimes, they are not even more than a short stroke (see Figure 7). Therefore, annotating them accurately without overlap is difficult and even harder to detect them precisely. A similar problem exists with the Winkelhaken (*w*). They are also often tiny and extremely densely placed, so the detector failed to detect them. The mass of Winkelhaken was added to the class *c* in the Gottstein system, and this might be the reason for the less precise performance of this class of the Gottstein wedge detector. Moreover, because of the higher similarity within the classes, the fusion of classes *d* and *f* results in higher precision in the Gottstein system. The oblique wedge *e* is the minority in PaleoCodage, and the detector cannot predict them. To keep the experiments between Gottstein and PaleoCodage comparable, the cutout with *e*, which always contains also other wedges, was not repeated. In summary, there is a high precision, especially for the Gottstein system, for the wedges with a high occurrence in the dataset. A larger dataset can also improve the precision also for the other classes. A high precision of wedge predictions is indispensable for a sign recognition based on detected wedges in future work. Our approach provides a wedge detector with high precision, however, it is a challenging task to improve the recall.

The result of the pipeline confirmed that the general idea of dividing between sign localization and wedge detection seems to be a potential approach to detect wedges on an entire tablet. Although the sign detector is theoretically the bottleneck of the pipeline, because it is not possible to detect wedges without detected signs, it did not result in the hardest challenge. However, as discussed before, the wedge detector is improvable and turns out to be the main bottleneck in our pipeline.

As can be observed in Table 2, the training with 3D renderings and mapped photographs improves the quality for both types of test sets. An explanation for this behavior could be that the different information in the renderings and photographs supports the model to generalize. The improvement by this fusion of datasets provides an indication that 3D data and photographs in combination are necessary to achieve good results in Cuneiform OCR. Additionally, the results have shown that the detectors trained on photographs and 3D renderings are applicable for both kinds of media without a performance loss of one of them.

5. Conclusion and Outlook

We introduced a pipeline approach to localize and detect wedges on an entire Cuneiform tablet and evaluated besides the pipeline both, the sign localization and wedge detection, components separately. The applied dataset has been annotated by us and is available at `https://doi.org/10.11588/data/QSNIQ2`. Our investigations on sign localization have shown that the dataset might not be suitable for sign detection tasks, but the signs can be cut out and could be a good base for a classification task without localization. Despite the challenge of incomplete annotations, the results of the sign detector are reasonably good and even better than the results using only photographs.

With smaller images the quality of the results for both the sign and the coupled wedge detector increases, so one approach to improve the results can be to split the images of an entire tablet into smaller squares, similar to [HFP*22].

The classification and localization of wedges is another challenging task in Cuneiform sign recognition. To improve the quality of the model in future work, the architecture could be more oriented towards the method for rotated objects in [ZY22]. The original approach in [ZY22] introduced the usage of the convex-hull of the predicted points by *RepPoints*, instead of pseudo bounding boxes to calculate the location loss, and the usage of Rotated Position Sensitive RoI Align (RRoI) [DXL*19] to extract the feature map of a rotated object. Furthermore, in our approach, only one backbone was applied, so a different number of *ResNet* [HZRS16] layers or a different backbone architecture could be tested. We have shown that the extension of the 3D renderings dataset with mapped photographs increases the quality of both models.

A further approach might be to omit the wedge localization. For example, to train a model that detects only the number of the occurring wedges within a sign, but not their positions. In the future, the detected wedges could help to classify Cuneiform signs. Using the introduced pipeline, the detected wedges are assigned to a sign. Based on this information, it might be possible, especially with a more accurate wedge detector, to identify the sign with the Gottstein system or PaleoCodage encoding.

Acknowledgements

This work was partially funded by DFG Project "Die digitale Edition der Keilschrifttexte aus Haft Tappeh" under project reference number 424957759 [**]

References

[BCE*] Baker, Heather D, Chiarcos, Christian, Englund, Robert K, et al. “Machine Translation and Automated Analysis of Cuneiform Languages (MTAAC)”. () 4.

[Bec08] Beckett, David. *Turtle-terse RDF triple language*. Tech. rep. 2008. url: `https://www.w3.org/TeamSubmission/turtle/` 5.

[BM20] Bogacz, Bartosz and Mara, Hubert. “Period Classification of 3D Cuneiform Tablets with Geometric Neural Networks”. *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2020, 246–251. doi: `10.1109/ICFHR2020.2020.00053` 1.

[BM22] Bogacz, Bartosz and Mara, Hubert. “Digital Assyriology—Advances in Visual Cuneiform Analysis”. *J. Comput. Cult. Herit.* 15.2 (May 2022). issn: 1556-4673. doi: `10.1145/3491239` 1.

[CLK20] Champin, Pierre-Antoine, Longley, Dave, and Kellogg, Gregg. *JSON-LD 1.1*. W3C Recommendation. https://www.w3.org/TR/2020/REC-json-ld11-20200716/. W3C, July 2020 5.

[DKMO20] Dencker, Tobias, Klinkisch, Pablo, Maul, Stefan M, and Ommer, Björn. “Deep learning of cuneiform sign detection with weak supervision using transliteration alignment”. *PLoS ONE* 15 (Dec. 2020). doi: `https://doi.org/10.1371/journal.pone.0243039` 2.

[DXL*19] Ding, Jian, Xue, Nan, Long, Yang, et al. “Learning RoI Transformer for Oriented Object Detection in Aerial Images”. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, 2844–2853. doi: `10.1109/CVPR.2019.00296` 9.

[Eng16] Englund, Robert K. “The Cuneiform Digital Library Initiative: DL in DH”. (2016) 4.

[Got12] Gottstein, Norbert. “Ein stringentes Identifikations- und Suchsystem für Keilschriftzeichen”. Berlin, Germany: Deutsche Orient-gesellschaft, 2012 2.

[GvdWV*23] Gillies, Sean, van der Wel, Casper, Van den Bossche, Joris, et al. *Shapely*. Version 2.0.1. Jan. 2023. doi: `10.5281/zenodo.7583915` 4.

[**] `https://gepris.dfg.de/gepris/projekt/424957759`

[HFP*22] Hamplová, Adéla, Franc, David, Pavlíček, Josef, et al. "Cuneiform Reading Using Computer Vision Algorithms". *Proceedings of the 2022 5th International Conference on Signal Processing and Machine Learning*. SPML '22. Dalian, China: Association for Computing Machinery, 2022, 242–245. isbn: 9781450396912. doi: 10.1145/3556384.3556421 2, 9.

[HGDG17] He, Kaiming, Gkioxari, Georgia, Dollár, Piotr, and Girshick, Ross. "Mask R-CNN". *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, 2980–2988. doi: 10.1109/ICCV.2017.322 4.

[Hom21] Homburg, Timo. "PaleoCodage—Enhancing machine-readable cuneiform descriptions using a machine-readable paleographic encoding". *Digital Scholarship in the Humanities* 36.Supplement_2 (2021), ii127–ii154. doi: 10.1093/llc/fqab038 2, 5.

[HZBM22] Homburg, Timo, Zwick, Robert, Bruhn, Kai-Christian, and Mara, Hubert. "3D Data Derivatives of the Haft Tappeh Processing Pipeline". *Cuneiform Digital Library Journal* 2022.1 (July 2022). [Online; accessed 2023-05-30] 2.

[HZRS16] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. "Deep Residual Learning for Image Recognition". *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 770–778. doi: 10.1109/CVPR.2016.90 3, 9.

[LDG*17] Lin, Tsung-Yi, Dollár, Piotr, Girshick, Ross, et al. "Feature Pyramid Networks for Object Detection". *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, 936–944. doi: 10.1109/CVPR.2017.106 3.

[LGG*17] Lin, Tsung-Yi, Goyal, Priya, Girshick, Ross, et al. "Focal Loss for Dense Object Detection". *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, 2999–3007. doi: 10.1109/ICCV.2017.324 3.

[Mar19] Mara, Hubert. "HeiCuBeDa Hilprecht–Heidelberg Cuneiform Benchmark Dataset for the Hilprecht Collection". *Version V2. heiDATA* (2019). doi: 10.11588/data/IE8CCN 4, 8.

[RFW*22] Rest, Christopher, Fisseler, Denis, Weichert, Frank, et al. "Illumination-Based Augmentation for Cuneiform Deep Neural Sign Classification". *J. Comput. Cult. Herit.* 15.3 (Sept. 2022). issn: 1556-4673. doi: 10.1145/3495263 2.

[SAP*23] Sommerschield, Thea, Assael, Yannis, Pavlopoulos, John, et al. "Machine Learning for Ancient Languages: A Survey". *Computational Linguistics* (2023), 1–44 2.

[SHK*14] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *J. Mach. Learn. Res.* 15.1 (Jan. 2014), 1929–1958. issn: 1532-4435 3.

[VVP*18] Vanweddingen, Vincent, Vastenhoud, Chris, Proesmans, Marc, et al. "A Status Quaestionis and Future Solutions for Using Multi-light Reflectance Imaging Approaches for Preserving Cultural Heritage Artifacts". *Digital Heritage. Progress in Cultural Heritage: Documentation, Preservation, and Protection*. Ed. by Ioannides, Marinos, Fink, Eleanor, Brumana, Raffaella, et al. Cham: Springer International Publishing, 2018, 204–211. isbn: 978-3-030-01765-1. doi: 10.1007/978-3-030-01765-1_23 2.

[YLH*19] Yang, Ze, Liu, Shaohui, Hu, Han, et al. "RepPoints: Point Set Representation for Object Detection". *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Aug. 2019, 9656–9665. doi: 10.1109/ICCV.2019.00975 3.

[YSC17] Young, Benjamin, Sanderson, Robert, and Ciccarese, Paolo. *Web Annotation Data Model*. W3C Recommendation. https://www.w3.org/TR/2017/REC-annotation-model-20170223/. W3C, Feb. 2017 5.

[ZY22] Zhou, Qiang and Yu, Chaohui. "Point RCNN: An Angle-Free Framework for Rotated Object Detection". *Remote Sensing* 14.11 (2022). issn: 2072-4292. doi: 10.3390/rs14112605 3, 9.