Introduction to Machine Learning and Artificial Intelligence in Archaeology (MLA)

Detecting Archaeological artefacts in video stream data

Personal Project Diary - Report

Definition of the problem

In this project, an initial, simplistic approach to the challenge would be that we need to detect archaeological artefacts in video stream data. To clearly define the scope of this project, the problem was broken down into key aspects that shape the overall approach: How detection should be interpretated in this context, what are the archaeological artefacts that need to be identified, and what technical challenges inherent in video stream data need to be addressed to achieve detection.

Detection interpretation

The concept of detection can take different forms, as already established in this course. Initially, a binary classification approach was considered, a classification of images as either containing an artefact (1) or not (0). However, after some initial experimentation, this approach was abandoned because it lacked the ability to detect artefacts within an image and focused more on classification. The focus then shifted on detecting artefacts in video frames and classifying them. A Machine Learning (ML) object detection model was prioritized, one that not only identifies artefacts but also draws bounding boxes around them based on a confidence threshold. The confidence threshold sets the minimum score needed for the model to classify an object. Overall, this method provides a more practical solution for archaeological applications, enabling identification and spatial positioning of artefacts within video frames.

Objects of interest/Classification

The primary objects to be identified in the videos are sherds of archaeological artefacts, specifically ceramic and lithic fragments. While both categories are present in the dataset, not all videos contain lithic artefacts. Initially, the idea was to focus on differentiating between ceramics and lithics. However, due to the limited amount of training data, a decision was made to classify all artefacts under a single category (ceramics). This would improve detection performance by focusing on maximizing artefact identification rather than distinguishing between subcategories.

Technical requirements

Videos consist of multiple frames, each containing artefacts scattered across various natural backgrounds. Several technical challenges had to be addressed:

- Video Processing: Videos need to be broken down into individual frames.

- Artefact Variability: Sherds come in different shapes, sizes, and colors, making detection more complex.
- Dynamic Backgrounds: The environment changes between videos, affecting visibility and contrast.
- Computational Power: Since all model training and implementation were conducted on a personal computer, a solution that does not require excessive computational resources was necessary.

Ultimately, the definition of the problem fell into this interpretation: How can I develop a computationally "light" model that identifies and classifies ceramics in video frames from different backgrounds, based on a limited number of data extracted from the videos?

Project Objectives

The objectives were clearly defined to guide the development of the model and ensure its application:

- Computational efficiency: The development of a computationally efficient and lightweight model, capable of running on a standard commercial computer. This would make the model accessible for practical use in real-world scenarios, even with limited hardware capabilities.
- Generalization: A model that successfully identifies ceramic sherds in video frames, regardless of background complexity, that ranged from static to more complex environments. In simple terms, a model that can generalize across varied backgrounds.
- Avoid overfitting: Develop a model that "learns" from the data rather than memorizing them. Most importantly, avoid overfitting and underfitting, allowing the model to generalize effectively to unseen data.
- Local handling: Manage all aspects of data and code handling, from training/test/validation splits to selecting frames, manually on a local level. This meant no use of external tools to gather or divide the data. Keeping everything local ensures that the project remains self-contained, with complete control over data preparation and model development. It also gives me a better grip on potential issues and model performance.
- Detection: A model that prioritizes detection. The goal was to maximize recall to ensure that as many artefacts as possible were detected, even at the risk of more false identifications of ceramics (False Positives).

Environment

All implementations and code executions for this project were done in Python using Spyder, a user-friendly program for writing and running Python code. For this project, new virtual environments were created, one for the detection model (YOLOv8n) and the frame extraction tool (OpenCV), and another for the annotation tool (LabelImg). The purpose of this separation was to prevent conflicts between libraries, newly installed and pre-existing, ensuring stable environments for each library. Without this differentiation, versions of libraries could interfere with each other, causing compatibility issues and errors. By using dedicated virtual environments, these risks were minimized, keeping installed libraries isolated from each other.

Methodology

Model - YOLOv8n

For the selection of the model that will localize and classify the ceramics in video, I sought an approach that could effectively perform both tasks while running smoothly on an average commercial computer. Inspired by an insightful implementation by Merel Penterman (2025), who applied the YOLOv8 algorithm in her research master's thesis at Leiden University to detect Bronze and Iron Age granaries from archaeological excavation maps, I explored YOLOv8 as a potential solution for my project.

YOLOv8 (You Only Look Once) is an object detection algorithm designed for speed and efficiency. Unlike other models like Faster R-CNN, which first identify regions of interest and then classify them, YOLOv8 processes the whole image in one step. This reduces processing time, which makes the algorithm "lightweight", allowing it to run on standard personal computers without requiring high-end GPUs. As a regression-based detection algorithm, YOLOv8 can not only detect ceramics but also classify them and generate bounding boxes around them. For the project, these classifications were made based on the probability of an object being a ceramic, using a confidence threshold for predictions. To reduce computing power requirements, I used the YOLOv8n model, a smaller and lighter version of YOLOv8.

For the implementation of the model in this project, I first ran the command `` `pip install ultralytics` `` in the Anaconda prompt to install the necessary library in my Spyder environment. Once the installation was complete, `yolov8n.pt` model was utilized, a pre-trained model originally designed to detect objects from COCO images. COCO (Common Objects In Context) is a large image repository used for training object detection models, including YOLOv8. This model was ideal for fine-tuning and training to detect ceramics on the ground for the specific needs of this project.

Data - OpenCV library

To address one of the challenges of the project, I used OpenCV, an open-source tool for computer vision and image processing. I installed OpenCV by running `` `conda install -c conda-forge opencv` `` in the Anaconda prompt. Using this library, I developed a function, with the help of ChatGPT, that allowed me to extract frames from multiple videos and save them to specific folders. By specifying the video and destination folder paths, I could extract the frames and name them based on their video type. I used different sets of frames for different model implementations, with distinct files for each video type (ranging from A to D). Frames for training, validation, and testing data were manually selected from each video type, creating distinct datasets for the model. This manual approach allowed for greater control over data quality, helping to identify potential issues and better assess the model's performance.

Annotation

The YOLOv8n model requires annotated data for training and validation. To create these annotations, I installed LabelImg, an annotation tool used for labeling images by drawing bounding boxes around objects of interest. I installed LabelImg locally using the following command in the Anaconda: `conda install -c conda-forge labelimg`. After installation, I used LabelImg to manually annotate training and

validation frames by drawing boxes around ceramics and labeling them. These annotations were then saved in YOLO format, making them compatible with the YOLOv8n model for training and validation.

Data split

The frames used for validation and training were annotated, while the testing frames were manually selected without annotations. All these frames were organized in a directory called "*dataset*". This directory contained two main subdirectories: "*images*" and "*labels*," which stored the data from the manual selection and annotation processes.

images Directory: Contains 3 folders

a) *train* – Contains the frames/data used for training the model, which were annotated with bounding boxes and class labels.
b) *val* – Contains the frames/data used for validating the model, also annotated.
c) *test* – Contains the frames/data used for testing each run of the model. These frames were manually selected.

labels Directory: Contains 2 folders

a) *train* - Contains the annotations (generated using LabelImg) for the training frames.
b) val - Contains the annotations (generated using LabelImg) for the validation frames.

YAML file

The model requires a YAML file to define the paths to the dataset images and annotations for training and validation. It basically works as an intermediate between the model and the data. It helps the model locate the paths for the training and validation datasets. It also includes important information like the number of classes and the class names, ensuring the model can understand and use the data for training.

Each YAML file was created manually for each dataset using Notepad, where the paths for the training and validation data were specified.

Training & Validation

For the training and validation process, a `yolov8n_train_val` function was implemented. This function is responsible for training the YOLOv8n model on the labeled dataset and then validating its performance. The training process involved setting key hyperparameters such as learning rate, number of epochs, and batch size, which are essential for fine-tuning the model. The `data_path` is another crucial parameter, specifying the path to the YAML file that contains the dataset configuration. To ensure compatibility, the `data_path` should be provided as a raw string. After the function is called and the run is over, the training and validation results are saved as `train_run` and `val_run` in the specified `save_path` directory. This directory, which should be provided before training as a raw string, stores logs, metrics, and model checkpoints from each run, ensuring proper tracking and reproducibility of results.

Testing

For the testing of the model, I developed a function defined as `yolov8n_pred_model`, which predicts the presence of ceramics in new, unseen images. The function takes the path to the trained model and the test images as raw strings, applies the model to the test data, and detects objects within the images. At the same time, it classifies them as ceramics, drawing bounding boxes with confidence level indications. To ensure reliability, predictions are filtered based on a confidence threshold, retaining only the most certain detections. Each prediction run made is saved as a `test` file in the `save_path` directory, which must be set before calling the function.

Predictions were made for each train run, using test images manually selected from each video type (A to D). The final test was conducted using at least 20 frames from each video type that the model had not been trained, tested or validated on, applying the weights of the best-performing model run for each dataset. This ensured that the predictions were based on the most optimal model weights obtained during each training process. The confidence threshold for every prediction was set at 0.2, with the risk of getting more misclassifications. This trade-off was chosen to prioritize detection over precision, ensuring that as many artefacts as possible are identified, even at the risk of occasional misclassification.

Metrics

The performance of the models was primarily evaluated using four key metrics: Recall, Accuracy, Loss, and mAP50. Recall measures the model's ability to detect actual artefacts. It is the ratio of correctly detected artefacts to the total number of artefacts present in the dataset. A higher recall indicates that fewer artefacts are missed. Accuracy represents the proportion of correctly classified artefacts among all predictions. Loss reflects how well the model is learning during training. Lower loss values and a downward trend indicate better model performance. map50 represents a general measure of how well an object detection model performs in classification. It helps determine how accurately the model identifies and localizes objects.

Hyperparameters

In machine learning, hyperparameters are settings that influence how a model learns from data. These are set before training begins and can significantly impact the model's performance. For the project's model, key hyperparameters were chosen to optimize model training. These included epochs, batch size, image size, learning rate and augmentation. Epochs indicate the number of times the model passes through the entire training dataset. Batch size is the number of training samples processed before the model updates its internal parameters. Image size represents the resolution of input images "fed" into the model. Higher resolutions provide more detail but increase computational cost. The learning rate controls how much the model's weights are updated during training. Lastly, augmentation techniques are applied to training images to improve generalization to unseen data. For this project, the image size was kept at 640x640 pixels across all model implementations (except for a few specific experimental runs, which are indicated at the results part). This means that all input images were resized to a consistent resolution of 640 pixels by 640 pixels. This ensured a balance between retaining sufficient detail for the model to learn from while minimizing the

computational cost of processing larger images. Other hyperparameters, such as the number of epochs, batch size, learning rate, and augmentation methods, were adjusted during different runs to assess their impact on the model's performance.

Ideal scenario

From a metrics perspective, the optimal model implementation would involve a steady downward trend in loss metrics, indicating effective learning, alongside high recall and accuracy. Since detection is prioritized over precision, recall takes precedence over accuracy, ensuring that as many artefacts as possible are identified. A strong mAP50 score would further confirm the model's ability to accurately detect and localize ceramics within video frames.

From a prediction standpoint, an ideal scenario would mean consistently accurate ceramic detections across all video types, regardless of background complexity or environmental variations.

Model implementations

Overall, four distinct model implementations were developed for training, validation, and testing:

Initial implementations

1. A model trained on frames from all video types, primarily A and B, with two classes – Located in the `Previous attempts` directory, the file named *Dataset with frames from all types (2 classes)*.
2. A model trained mainly on frames from A-type videos – Found in the `Previous attempts` directory, the file named *Dataset with frames from A videos*.
3. A model trained with frames from all video types, equally selected, with a single class (ceramic) – Available in the `Previous attempts` directory, the file named *Dataset with frames from all types (1 class)*.

Final implementation

4. A final model trained with frames from all video types, focusing on those with less complex backgrounds – Found in the `*Final implementation*` directory.

Each of these files contains the following components:

- A `*dataset*` file with a YAML file, images (training, validation and testing data) and labels (their annotations) used for training, validation, and testing each run.
- A `*results*` file that logs each run of the model, including training, validation results and tests for each run.
- A manually created PDF file with detailed metrics and performance analysis of each model run, including a breakdown of the results after each training session.

A `*Predictions*` directory was also developed, containing predictions for each model implementation. These predictions are applied to at least 20 frames from each video type that the model hasn't been trained, validated or tested on before, using the weights from the best-performing model run.

Data were split into 80-10-10 for training, validations and testing. Initially, one model was trained to detect two different classes. However, this approach was abandoned after an initial experimentation.

Each dataset was developed to evaluate different aspects of the model's performance, enabling comparative analysis based on data types and complexity.

Results

Dataset with frames from all types (2 classes)

This stage was an initial experiment to explore how the model could be implemented for the project. The model was trained to identify two classes (ceramics and lithics). A total of 160 annotated frames were used for training, 20 for validation and 20 for testing each run, all manually selected from the extracted frames of the videos. Three runs were conducted to assess the model's performance.

Training – validation metrics

In the first run, 20 epochs were selected, with a batch size of 8 and a learning rate of 0.0001. Augmentation code was also applied. The metrics showed significant fluctuation in the validation loss, even though precision and recall remained high. Notably, the validation loss did not show a downward trend, indicating instability in the model's learning process. The second run yielded better results, with the number of epochs increased to 30 while keeping the batch size and learning rate the same. This adjustment led to a downward trend in the validation loss metrics, and both precision and recall improved. However, the model still exhibited fluctuation.

Overall, while recall and precision metrics showed improvement across the three runs, the model experienced considerable fluctuation. The mAP50 metric consistently stayed above 0.4, which was a relatively positive result given the instability of the validation loss metrics.

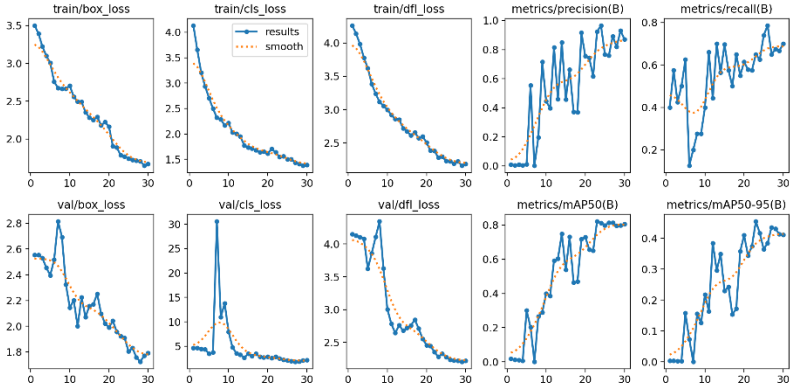

Figure 1. Metrics from the second run of the model

Predictions

Predictions were employed with the model weights of the second run (the best overall in this dataset). On (at least) 20 frames from the A-type videos, the model demonstrated quite consistent identification of ceramics and lithics on the ground. It still missed quite a few though that were not in the training images. In the frames from the B, C and D video types the model's performance was notably poor, successfully identifying only a few ceramics on the ground of B-type frames and almost none in the C and D type frames.

Figure 2. Predictions using the model weights from the second run on A-type frames (left), B-type frames (middle) and D-type frames (right)

Dataset with frames from A videos

This was also an experimental attempt to figure out the optimal implementation of YOLOv8n. In this case, a custom YOLOv8n model was used without pre-training on the COCO dataset. Instead of using the pre-trained `yolov8n.pt` model, the `yolov8n.yaml` configuration file was used for the training. This approach was chosen to train the model from scratch, providing more flexibility to adapt it specifically to the project's dataset.

A total of 123 annotated frames were used for training, 16 for validation and 16 for testing each run, all manually selected from the extracted frames of the videos. Overall, nine runs of the model were performed. It is also the only case where the image size was altered for the first three runs to 240x240 pixels.

Training – validation metrics

The training process began with 10 epochs, a batch size of 8, and a learning rate of 0.0001. Although the validation loss was relatively high, it showed a downward trend. In run 5 (the best of the model overall), I increased both the image size and the number of epochs, which appeared to improve the model's

performance, as evidenced by higher mAP50 and recall metrics. However, the validation loss remained high.

Overall, throughout the various runs, I experimented with more epoch values and lower learning rates to reduce the validation loss. This model demonstrated better trends in the metrics compared to the previous one, but fluctuations were still present. The validation loss remained higher than that of the model trained on the COCO dataset.

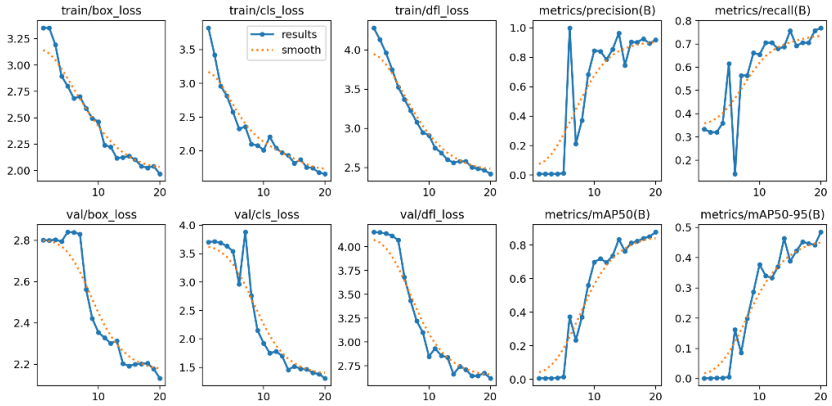

Figure 3. Metrics from the fifth run of the model

Predictions

Predictions were made using the model weights from the fifth run, which performed the best overall in this dataset. On frames from the A-type videos, the model showed strong performance, correctly identifying ceramics in all but one frame. However, in the frames from the B, C, and D video types, the model struggled to detect ceramics, although it performed slightly better than the previous model implementation, which had to identify two classes in the frames. Overall, it showed improved performance, particularly in the A-type videos, compared to the earlier model version.

Figure 4. Predictions using the model weights from the fifth run on A-type frames (left), B-type frames (middle) and C-type frames (right)

Dataset with frames from all types (1 class)

In this implementation, the pre-trained YOLOv8n model was used (`yolov8n.pt`) in all runs, as the training data would become more complex than in the previous implementations. For each run, 20 images from each video type were used for training, 5 images from each type for validation, and 5 images from each type for testing. In total, 160 annotated frames were used for training, with 20 frames each for validation and testing. Five runs of the model were conducted.

Training – validation metrics

I began training the model with 20 epochs, a batch size of 12, and a learning rate of 0.0001. However, the large batch size seemed to hinder performance, resulting in low mAP50 metrics and fluctuating trends across all metrics. As the runs progressed, I focused on reducing the batch size while maintaining a low learning rate. By run 5, the model showed less fluctuation in the metrics, with slightly improved mAP50 values. However, recall and precision appeared to plateau, indicating limited further improvement.

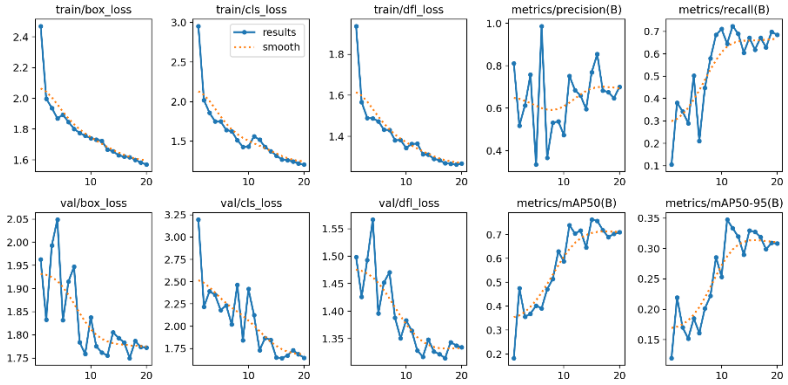

Figure 5. Metrics from the fifth run of the model

Predictions

Testing in this implementation of the model showed significant improvement compared to previous versions. Ceramics in the A-type frames were almost all correctly identified, with only a few false positives. Predictions for frames from types B, C, and D were also generally accurate, although there were many false positives, particularly in the D-type frames.

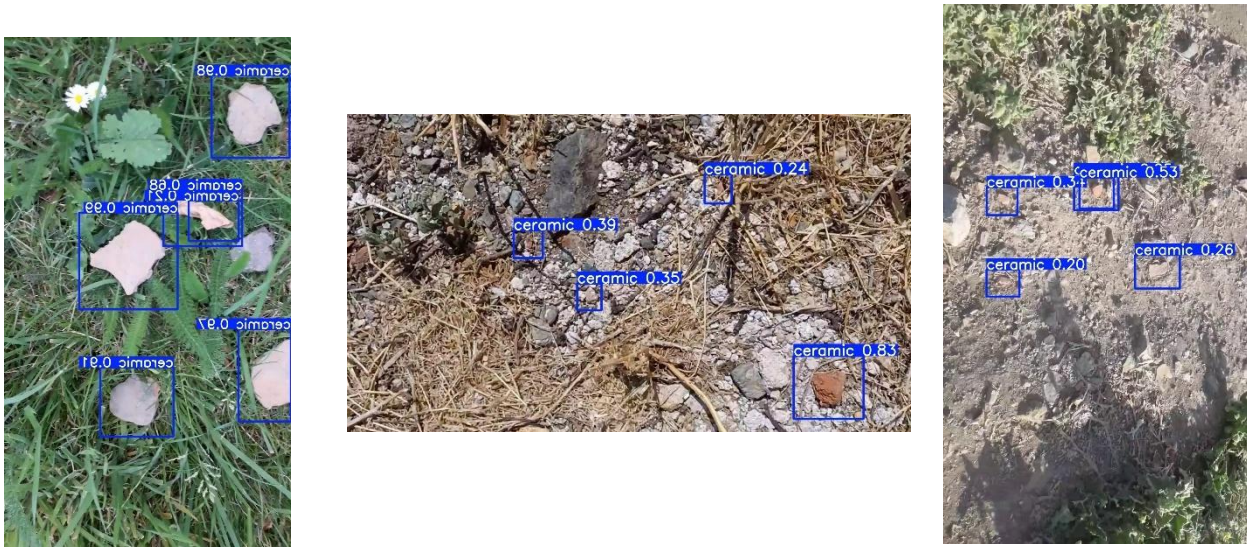

Figure 6. Predictions using the model weights from the fifth run on A-type frames (left), B-type frames (middle) and D-type frames (right)

Final Implementation

Since the predictions from the previous model were the best so far, I attempted to improve them further by reducing the training and validation images. I kept only the frames and annotations where ceramics were more easily detectable with the naked eye. For training 130 annotated images were used and 16 annotated for validation. Eleven runs of the model were conducted.

Training – validation metrics

The training and validation metrics clearly indicated that the model was overfitting. Although the fluctuation remained strong, the mAP50 metrics and recall were higher than in any previous model. In run eight, I used 20 epochs, a batch size of 8, a learning rate of 0.00005, and applied augmentation. This run emerged as the best, as it showed a higher recall and a downward trend in validation loss, although significant fluctuation in all metrics persisted.

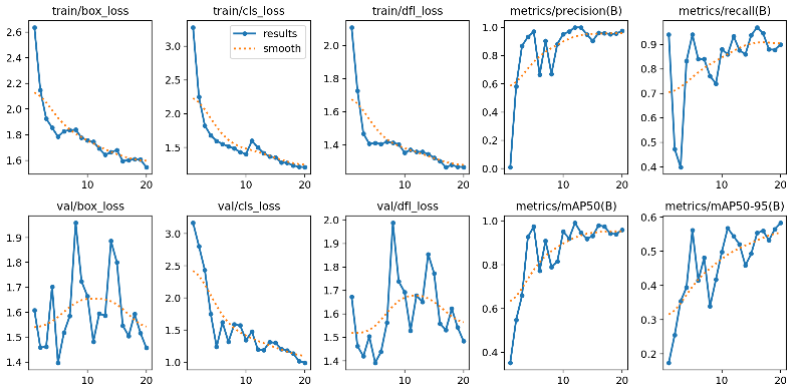

Figure 7. Metrics from the eighth run of the model

Predictions

This model implementation (run 8) achieved the best overall performance. Nearly all ceramics in A-type frames were correctly identified, with a few false classifications. Predictions in B- and C-type frames were also highly accurate, with minimal false positives or missed ceramics. In D-type frames, ceramic detection was strong, though some false positives were present, and some ceramics were missed.

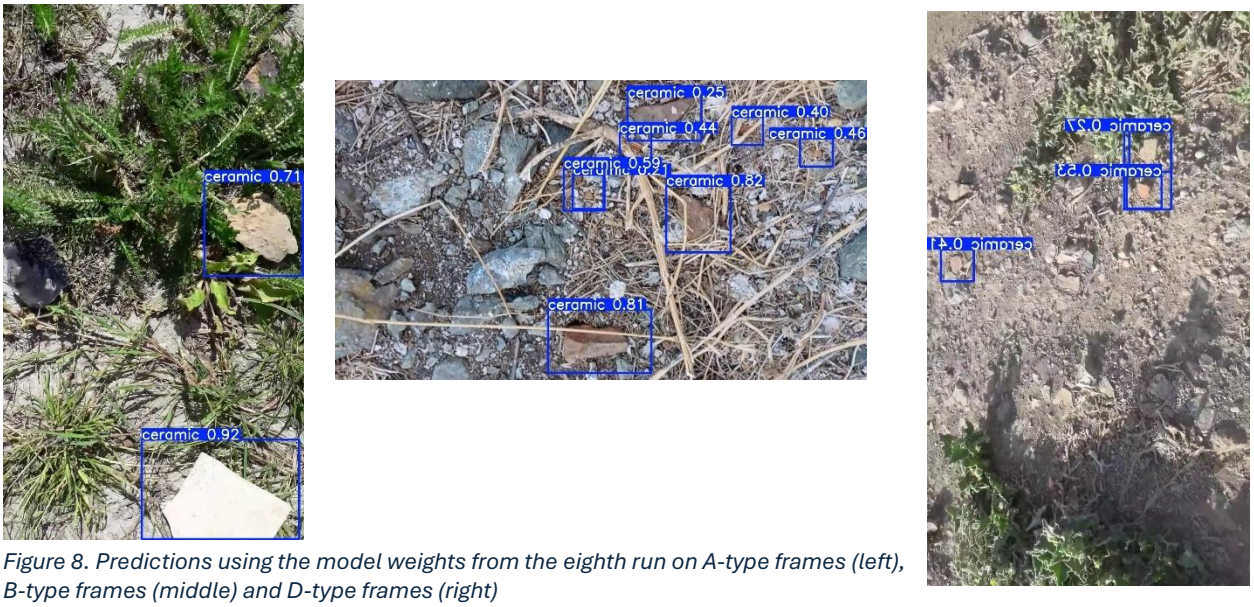

Figure 8. Predictions using the model weights from the eighth run on A-type frames (left), B-type frames (middle) and D-type frames (right)

Discussion

In this project, the aim was to develop a model capable of accurately detecting ceramics on the ground across various backgrounds and environments. Exploring the YOLOv8n model for this purpose proved to be more challenging than expected, but it provided valuable insights and promising results for future refinements.

Interpretating the results - Overfitting

Based on the results from each model, it is clear that the models performed best on the frames they were trained on. The initial objective was to transition from simpler, static backgrounds to more complex ones, progressively refining the model's ability to detect ceramics across diverse environments. The first model was trained on different backgrounds, but mainly on A- and B-type videos, to assess its ability to accurately detect ceramics and lithics under static conditions. This model faced challenges also due to having less information for each class, which hindered its ability to perform well in any environment. The second model, which was primarily trained on A-type videos, effectively "memorized" the ceramics in this simpler background. This model performed adequately in detecting ceramics on static backgrounds but struggled when applied to more complex environments. To address this, the dataset was altered to include annotated frames from more challenging backgrounds, aiming to improve the model's performance. The next iterations incorporated a broader range of equal distribution frames from all video types. When tested, the third model performed well in the more complex backgrounds and adequately in the simple backgrounds.

This improvement was due to its training data, which contained more complex environments. The final model was fine-tuned using frames from every video type but with reduced complexity, prioritizing frames where ceramics were clearly visible. This version showed the best performance overall, considering the testing and prediction aspect of the project. While the model performed well on both types of environments, it still showed signs of overfitting, as it tended to "memorize" the training data. Overfitting remained a significant challenge across all model versions. The models demonstrated strong recall but exhibited signs of memorization rather than “learning”. This issue became particularly evident when tested on unrelated images. As seen in Figure 9, the “best” model performed poorly when applied to random images, reinforcing the concern that it was overfitting to specific training patterns rather than developing true generalization capabilities. Of course, given the limitations of my implementations and the specific reason this model was trained, I did not expect it to accurately identify ceramics in completely unrelated images with no connection to archaeology or ceramic detection whatsoever.

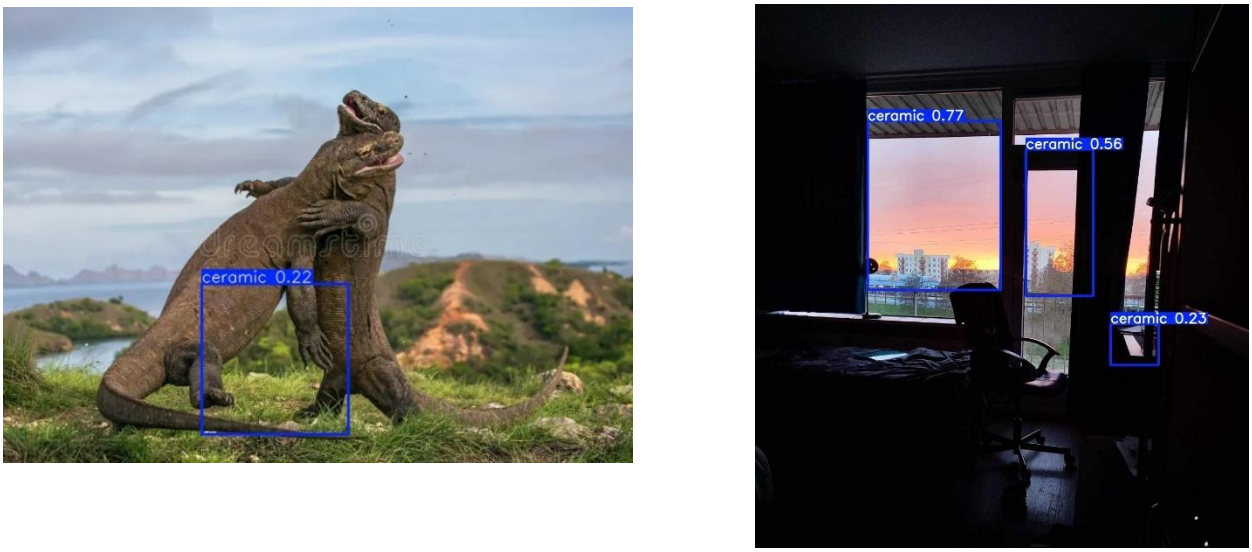

Figure 9. Predictions on unrelated to archaeology imagery using the weights of the best model developed in this project

Metrics

Throughout all implementations, metric evaluations revealed strong fluctuations, particularly in validation loss, which is often an indicator of overfitting. Various hyperparameter adjustments were tested in an attempt to stabilize these fluctuations, but none fully addressed the issue. However, as the number of epochs increased and both learning rate and batch size were kept lower, the model exhibited improved performance in both metrics and predictions. This suggests that, with limited training data and computational resources, a slow and steady training process is necessary to optimize results. Nonetheless, in this case, the model ultimately memorized the training data rather than learning to generalize effectively.

Detection

Detection was prioritized over precision, as reflected in the confidence threshold being set at 0.2. Lowering the threshold allowed for more detections, even at the cost of increased false positives. The final model was

able to identify ceramics across most frames, demonstrating a reasonable level of generalization. However, while the model successfully detected ceramics in multiple scenarios, confidence levels varied, and false positives - though fewer than initially expected - remained an issue. A higher threshold value could be implemented, once the training process of the models is better developed.

Computational efficiency

All training, validation, and testing processes were executed on a personal computer with limited computational power. Despite these constraints, the model was successfully developed and tested, proving that YOLOv8n can be trained and fine-tuned without access to high-end hardware. This highlights the feasibility of deploying similar models in resource-limited environments.

Local handling

The entire dataset was manually organized, making local data handling a tedious yet essential part of the project. This approach ensured full control over data selection, annotation, and preprocessing. The functions for model implementation were independently developed, with assistance from ChatGPT for frame extraction code. This hands-on process reinforced a localized workflow where every step remained manageable without external computational resources.

Things that proved challenging

Throughout this project, several challenges were encountered that influenced the model's development and implementation.

From a technical perspective, one of the main challenges was the installation and compatibility of the required libraries. Specifically, setting up LabelImg was quite challenging, as it repeatedly crashed until I installed and updated it in a different environment. Ensuring all dependencies functioned correctly across different setups required extensive troubleshooting. Additionally, after several iterations of the model, it took some time to accurately understand where and how the validation and training results were being saved. This emphasized the need for a function that clearly specifies where the results should be stored.

The manual handling of several aspects throughout this process also proved to be quite demanding. For instance, every time the directory paths or names of the training and validation data changed, the YAML file had to be manually edited to reflect these modifications. The file also required forward slashes (/) instead of backslashes (\), which made frequent updates to the YAML file necessary. Furthermore, the manual annotation and selection of the dataset, while necessary for creating a reliable model, proved to be extremely time-consuming and difficult task. While tools like LabelImg helped to locally streamline the process, they still required significant manual oversight to avoid errors. This made the whole process more complex and prone to human error.

From a model implementation standpoint, overfitting remained a persistent issue throughout the project. Despite experimenting with different techniques, such as adjusting learning rates and batch sizes, the model struggled to generalize effectively. The level of overfitting observed made it clear that further work is required to address this issue and improve the model's capabilities.

Lastly, effectively managing time and devising a structured training strategy for the model proved to be significant challenges throughout the project. Training and testing deep learning models required multiple iterations, each demanding considerable time for annotation, model adjustments, and result analysis. The need to troubleshoot unexpected issues - such as dependency conflicts, overfitting, and manual dataset adjustments - often slowed progress.

Critical evaluation & future proposed changes

This project primarily served as an experimental process for developing a functioning Machine Learning model with applicability to archaeology. While key objectives were successfully achieved, others remain open for improvement in future iterations. Reflecting on my initial objectives of this project there are some key takeaways.

One of the main goals was to assess the feasibility of training and deploying a ceramic detection model using local resources with limited computational power. Despite hardware constraints, the model was effectively trained and tested, demonstrating its potential in low-resources settings. A working model capable of detecting ceramics on the ground across various backgrounds was developed, providing a foundation for future refinements. At the same time, two critical challenges persisted, generalization and overfitting. While the model performed well on the specific environments it was trained on, it struggled when applied to new, unseen data. Overfitting remained a limitation, as the model tended to memorize training data rather than develop the ability to generalize.

It is also important to address an oversight in how the data were handled. Although the dataset was initially split into an 80-10-10 ratio, the testing process was conducted separately from the training and validation workflow. A clearer distinction between training, validation, and testing is necessary to ensure a more reliable evaluation of the model's performance. A more effective approach would involve a 80-20 or 90-10 split for training and validation, with a separate set of unseen data reserved exclusively for final testing. This would enable a better training and validation process, subsequently improving the model's ability to detect ceramics.

To address limitations in future developments, data refinement is also essential. While increasing the dataset size may seem like an obvious solution, I believe the model can perform effectively with a relatively small, more refined dataset. More time should be dedicated to improving the quality of the dataset, ensuring that it is well-balanced and representative. This will be crucial for enhancing the model's ability to generalize. Additionally, incorporating manual data augmentation (variations in lighting, angles, and background textures) could further improve performance by helping the model recognize ceramics in different environments. This would shift the focus from memorizing training images to learning key ceramic features.

Lastly, more time could be dedicated to addressing the specific issues of each model. The initial goal of detecting two artefact classes - ceramics and lithics - was quickly abandoned, despite opportunities for improvement. The main issue stemmed from how data were provided to the model, as the lithic dataset was small, sourced mostly from static environments, and contained consecutive frames, limiting its ability to generalize. The second model aimed to simplify detection but struggled to perform in varied environments. The use of consecutive frames also did not help the model to learn meaningful patterns. The third and then the final models were trained on more complex, "crowded" frames, where even annotations were challenging to create. Additionally, although a custom YOLOv8n model was tested and showed promising

performance in both metrics and predictions, it was not refined in later stages. Addressing each model individually, with a more targeted and streamlined strategy, would be a great start for further improvement of the project.

To conclude, while exploring a different model is an option, this would not solve the core issues encountered with the YOLOv8n model. In future iterations, the model has the potential to become a more reliable tool for archaeological application.

Reference list

Penterman, M. (2025). Ancient Storage and AI: Automated object detection of prehistoric granaries on archaeological GIS maps: A Deep Learning approach.