



Version 1.1



Automatic Detection of Archaeological Features – ADAF User Manual

Version Control Table

Title	Automatic Detection of Archaeological Features – ADAF			
Description	User manual for the Automatic Detection of Archaeological Features (ADAF) software tool.			
Created By	<p>Nejc Čož*, Žiga Kokalj*, Ana Kostovska**, Dragi Kocev**, Anthony Corns*** and Susan Curran***</p> <p>* Research Centre of the Slovenian Academy of Sciences and Arts</p> <p>** Bias Variance Labs d.o.o.</p> <p>*** The Discovery Programme</p>			
Date Created	01/12/2022			
Maintained By	Nejc Čož, nejc.coz@zrc-sazu.si			
Rights Statement	<p>Copyright © 2023 The Discovery Programme, Research Centre of the Slovenian Academy of Sciences and Arts (ZRC SAZU) and Bias Variance Labs.</p> <p>This work is openly licensed via CC BY 4.0.</p>			
License Agreement	<p>This software is distributed without any warranty and without even the implied warranty of merchantability or fitness for a particular purpose.</p> <p>ADAF is open source. The code is available on GitHub, with an Apache License 2.0.</p>			
Citation	<p>When using the toolbox, please cite:</p> <p>Čož, Nejc, Žiga Kokalj, Dragi Kocev, and Ana Kostovska. 2024. "ADAF - Automatic Detection of Archaeological Features." Python. Ljubljana, Slovenia: ZRC SAZU, BVL. https://github.com/EarthObservation/adaf.</p>			
Version Number	Modified By	Modifications Made	Date Modified	Status
0.1	Nejc Čož	Additional instructions based on new GUI.	12/12/2023	
0.2	Anthony Corns	Instructions adjusted based on user testing and glossary Added.	18/12/2023	
0.3	Anthony Corns	New URL added for software download	18/12/2023	
0.4	Ana Kostovska	Instructions for training new ML models (Appendix)	14/3/2024	
1.0	Nejc Čož	GUI update	18/3/2024	

1.1	Nejc Čož	Update installation instructions	9/7/2025	

Contents

Automatic Detection of Archaeological Features – ADAF User Manual	2
Version Control Table	2
Contents	4
General information	5
Online resource	5
Acknowledgements	5
Glossary	6
Installation.....	7
Running ADAF.....	10
Input data options	13
Machine learning options.....	15
Post processing options.....	17
Output	19
Known issues	21
Appendix.....	22
ADAF models	22
SLRM visualisation.....	22
Notebooks for creating training data	23
Notebooks for retraining MLMS.....	25

General information

The tool for Automatic Detection of Archaeological Features (ADAF) has been developed to provide user-friendly software that uses machine learning models (in particular convolutional neural networks) to enable the automatic detection of archaeological features from airborne laser scanning (ALS) data. The software requires minimal interaction and no prior user knowledge of machine learning techniques, greatly improving its accessibility to the archaeological community. The underlying machine learning models have been trained on an extensive archive of ALS datasets in Ireland, labelled by experts with three types of archaeological features (enclosures, ringforts, barrows). The core components of the tool are the Relief Visualisation Toolbox (RVT) and the Artificial Intelligence Toolbox for Earth Observation (AiTLAS), both of which are actively used in the field of aerial archaeology. RVT is indispensable for processing input data (for training and inference) by converting digital elevation models into machine learning-friendly visualisations, while AiTLAS provides access to the machine learning models.

Online resource

All installation instructions and files are available in the [GitHub repository](#).

(Deprecated – old video instructions <https://youtu.be/pqYsD3lqBfU>)

Acknowledgements

The development of the Automatic Detection of Archaeological Features (ADAF) tool was funded by Transport Infrastructure Ireland (TII) Open Research Call 2021

Glossary

ADAF	Automatic Detection of Archaeological Features
AiTLAS	Artificial Intelligence Toolbox for Earth Observation
ALS	Airborne Laser Scanning
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DEM	Digital Elevation Model
DFM	Digital Feature Model
FasterRCNN	Faster Region-based Convolutional Neural Network
GDAL	Geospatial Data Abstraction Library
GIS	Geographic Information System
GPU	Graphics Processing Units
HRNet	High-Resolution Net
ML	Machine Learning
PIP	Package Installer Python
RVT	Relief Visualisation Toolbox
SLRM	Simple Local Relief Model

Installation

The installation is currently only supported on Windows 64-bit machines. The application is compatible with machines equipped with CUDA-enabled graphics cards, but will also work on a standard CPU where GPU processing is not possible.

Requirements

The ADAF tool requires **Python version 3.9** or higher:

- We recommend to use **conda virtual environment** is available through [Miniconda](#) or [Anaconda](#) distributions (installation instructions below).
- For experienced Python users, ADAF can be set up in any Python virtual environment.

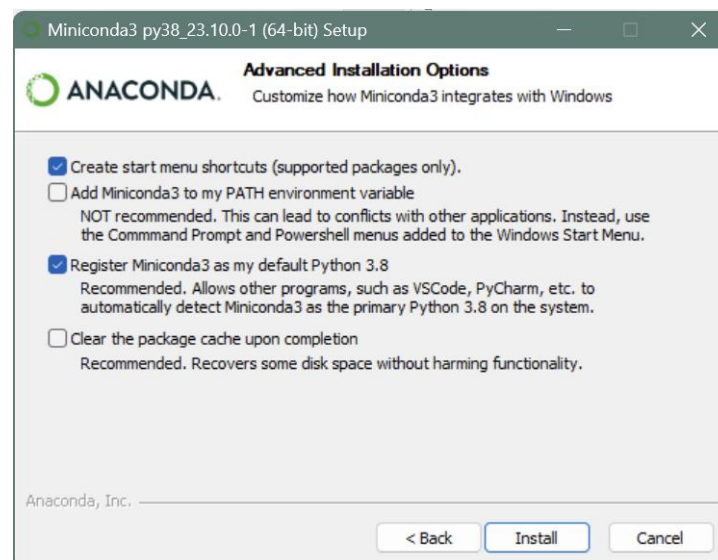
Download the ML models:

- Files with trained Machine learning models. Download from Dropbox:
<https://www.dropbox.com/t/QRVtxUVTPRVSnYKK>

Warning: Download contains data with large file size ~5GB in total. This includes 8 pretrained ML models saved as TAR files: 4 for semantic segmentation and 4 for object detection.

Installing Anaconda

1. Download the Miniconda installer: https://repo.anaconda.com/miniconda/Miniconda3-py38_23.11.0-2-Windows-x86_64.exe
2. Start installation by running the exe file.
3. Use the default settings in the installation wizard.



Setting up additional software

1. Clone the GitHub repository to your local drive


GitHub repository: <https://github.com/EarthObservation/adaf>

For example: C:\Users\<username>\adaf-main

This is now the location of ADAF repository

2. Move the downloaded TAR files to **ml_models** folder, which is located inside the **adaf** folder (**Important:** do not extract the individual TAR files!)

In our example: C:\Users\<username> \adaf-main\adaf\ml_models

3. Run Anaconda Prompt (press Win key () and type "anaconda prompt").
4. In the Anaconda Prompt, navigate to the **repository** folder by running command:
cd <path-to-repository-folder>, i.e. where you have downloaded and unzipped the installation files
to e.g. cd C:\temp\adaf-main
5. Create and activate a conda environment. Run commands:
conda create -n adaf python=3.9 -c conda-forge
conda activate adaf

6. Install PyTorch for CUDA (**ONLY FOR CUDA COMPLIANT GPUS**)

Skip step 6 if you don't have a CUDA enabled device!

When installing on a PC which has a CUDA enabled graphics card (check [here](#) for NVIDIA compliant cards) the GPU can be used to reduce processing times. If your card is compliant (also requires installation of CUDA software that is not covered in this manual) install compatible PyTorch version by running:

torch_cuda_installation.bat

If *.bat script fails (this may happen because of version upgrades of CUDA), please install pytorch manually by following the instructions on the official website:

<https://pytorch.org/get-started/locally/>

7. Install required Python packages

conda install -c conda-forge gdal


pip install ./installation/aitlas-0.0.1-py3-none-any.whl

8. Enable the use of the **adaf** virtual environment in Jupyter notebooks by running:

python -m ipykernel install --name adaf

9. Close the Anaconda Prompt window. The installation is now complete.

Reinstalling or uninstalling the environment

1. Open Anaconda Prompt (press Win key () and type “anaconda prompt”)
2. Remove aitlas virtual environment by executing the following commands:


```
conda env remove -n adaf
```

```
jupyter kernelspec uninstall adaf
```
3. Delete the main ADAF folder. ADAF is now uninstalled.

To reinstall the software, follow the installation steps from the start.

Running ADAF


Starting ADAF

1. Open Anaconda Prompt (press Win key () and type "anaconda prompt"
2. Click and drop the shortcut called **ADAF.Ink** onto the Anaconda Prompt window and press enter



Starting ADAF manually*

* Users who are familiar with Python and/or Jupyter Notebooks can use ADAF notebooks used in their preferred way.

- Open Anaconda Prompt (press Win key () and type "anaconda prompt")
- Activate the atlas environment

```
conda activate adaf
```

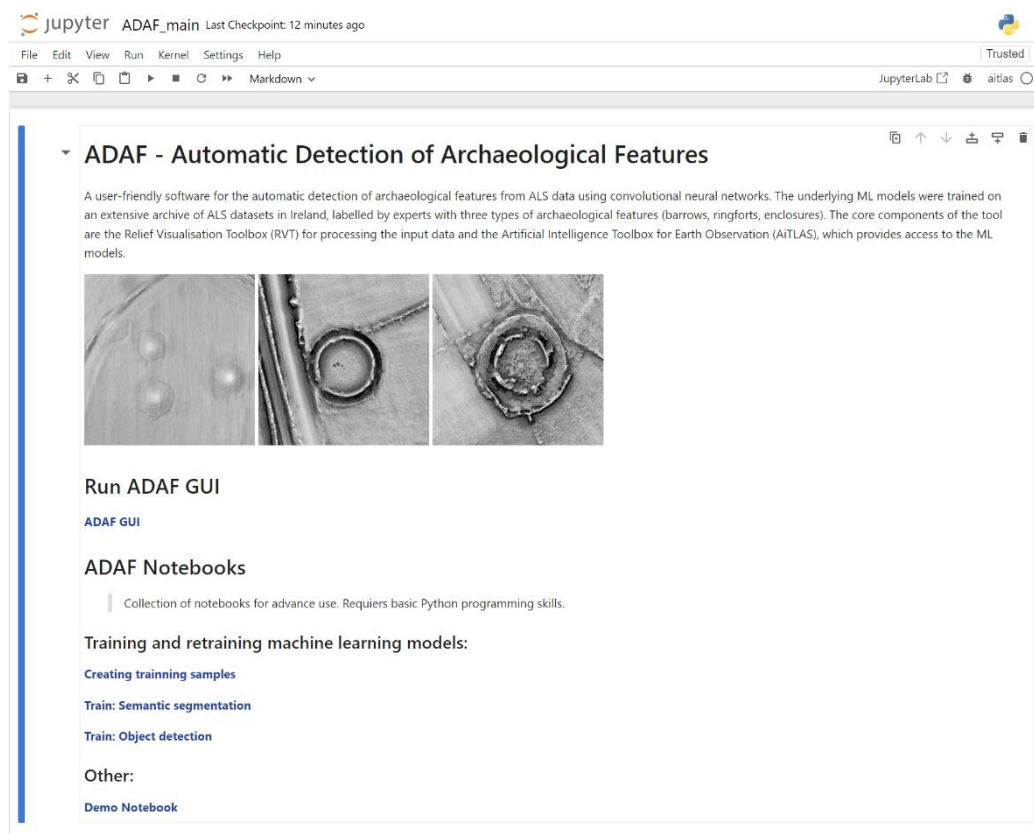
- Navigate to the location of ADAF

```
cd <path-to-adaf-folder>
```

- Run command to open Jupyter notebook:

```
jupyter notebook ADAF_main.ipynb
```

Once the Jupyter notebook has loaded an introduction screen will be displayed in your chosen web browser



Here you can select several ways to use the tool.

1. Run ADAF GUI

Selecting **ADAF GUI** will take you into the standard default interface to use the ADAF tool and will be the one mostly used.

2. ADAF Notebooks

a. Training and retraining machine learning models

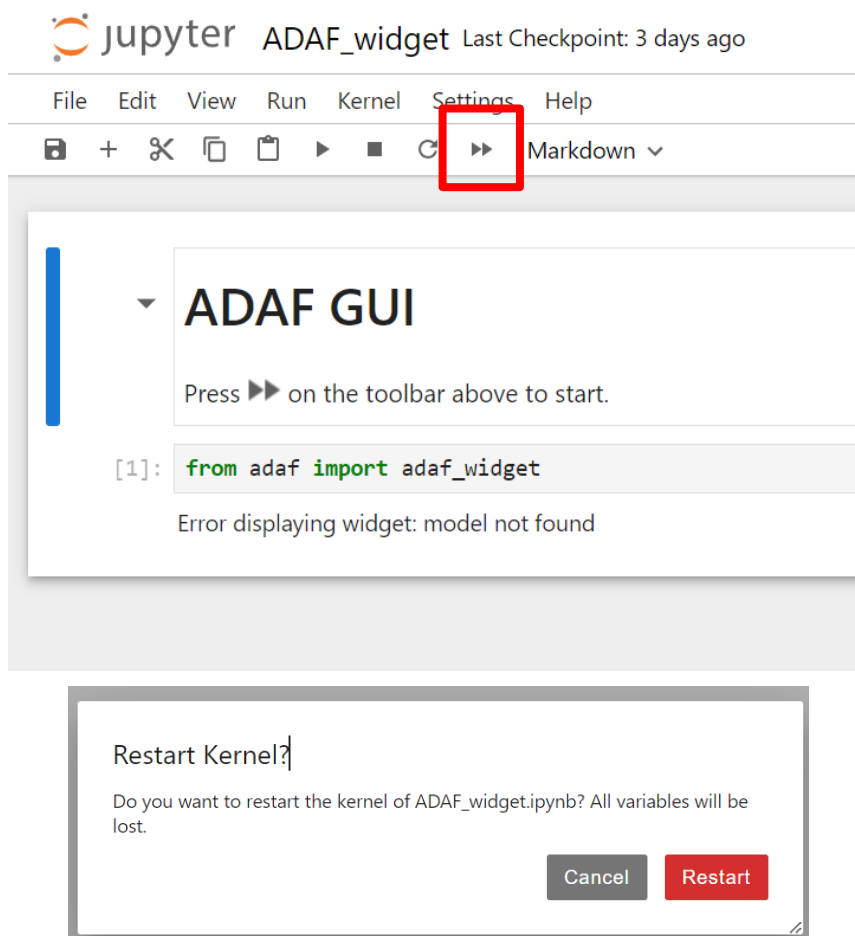
For the more experienced user this option will take you into the notebook without a GUI enabling you to observe the processing and parameter details in Python.

b. Other

Any other useful Jupyter notebooks will be made available here.

Starting the ADAF tool notebook

Once you have selected your chosen notebook you will need to restart it to load the tool interface. This can be done by pressing the restart button (▶▶) on the Jupyter toolbar.



It will ask you to confirm that you want to **restart the kernel**. Select **Restart** and wait for the interface to load. This process is only required for your first use of the ADAF tool in a session.

GUI

The graphical user interface leverages interactive Jupyter Notebook widgets, allowing users to intuitively set parameters for running the provided ML models.

Input data options:

Select input file:

- ☒ DEM (*.tif / *.vrt)
☐ Visualization (*.tif / *.vrt)

Select file

☐ Tiles are from same dataset (create VRT)

Output folder: **not selected**

Change output folder

☐ Save visualizations

Machine learning options:

Select machine learning method:

- ☒ segmentation
☐ object detection

Select model:

ADAF model



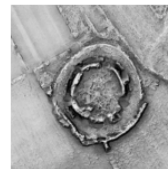
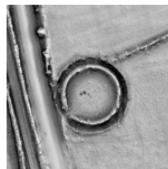
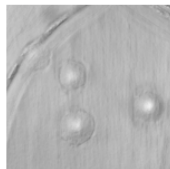
Select classes for inference:

☒ All archaeology

☐ Barrow

☐ Ringfort

☐ Enclosure



Post-processing options:

Select min area [m²]:



40

Select min roundness:



0.50

Roundness examples:

0.50



0.95



☐ Keep probability masks (raw segmentation results)

Run ADAF

Input data options

A description of all available options for the Input data module of ADAF.

Input data options:

Select input file:

☒ DEM (*.tif / *.vrt)
☐ Visualization (*.tif / *.vrt)

Select file
☐ Tiles are from same dataset (create VRT)

Output folder: **not selected**

Change output folder

☐ Save visualizations

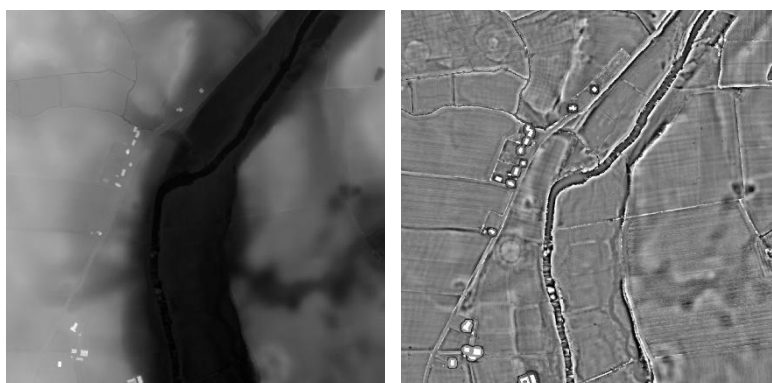
Select input file

The user has the option to select an ALS DFM or derived visualisation raster as the input data where they wish to identify potential monuments.

DEM – the user can enter any DEM in a GeoTIFF (*.tif) or a GDAL virtual format (*.vrt) format. The ADAF app creates the visualisation and splits the image into smaller tiles. The size of the tiles is pre-set and has been selected to optimize processing performance.

Visualisation – the user can enter the visualisation that has already been created. This speeds up processing as the step of creating the visualisation is skipped. It is up to the user to ensure that the correct visualisation type is specified. For the built-in ADAF ML models, the required visualisation is the normalized SLRM (can be calculated externally with the RVT tool). For the parameters for creating the SLRM visualisation, see Appendix.

Note the processing time if a DEM is selected will be slightly longer than if a visualisation is input



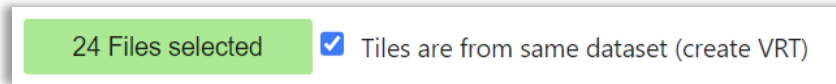
Example of input datasets DEM (left) and Visualisation (right)

Input file(s)

Select the input file(s) by clicking on the [Select file](#) button. The files are browsed and selected through a dialog window.

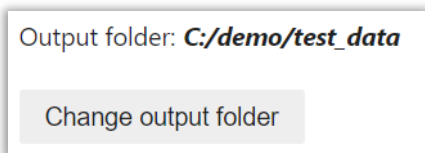
The app will allow to select any file in *.tif or *.vrt format. The best results will be achieved with input files of similar raster quality to the datasets used in training. This model is trained on digital feature models (DFM, i.e. DEM of terrain and buildings) with spatial resolution of 0.5m.

If more than one file is selected, the software will assume each file is a separate dataset and will run inference sequentially on every file. If files are in fact tiles of the same dataset, [check the box](#) next to the file selection button and virtual mosaic (VRT) will be built on-the-fly.



Output folder

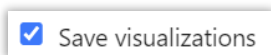
By default, the output folder is set to be the parent folder of the input file(s). The path to the output folder will displayed bellow the "Select input file" button once the input file is selected.



A different output folder can be selected by clicking the [Change output folder](#) button.

Save visualisation

If checked, the visualisation files generated from the DEM will be saved in the output folder as a virtual mosaic (*.vrt). All files generated by the tool are saved in the same root folder as the source DEMs or visualisation files (See Output).



Machine learning options

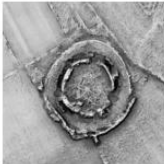
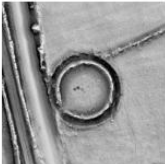
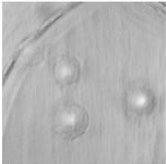
A description of all available options for machine learning module of ADAF.

Machine learning options:

Select machine learning method:
☒ segmentation
☐ object detection

Select model: ADAF model ▼

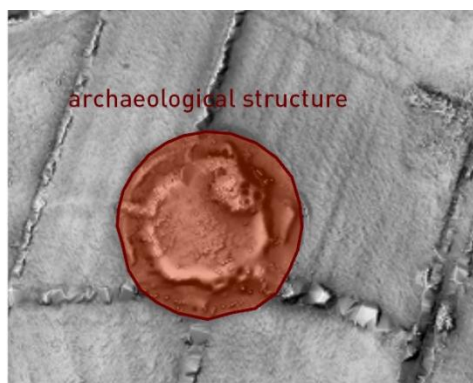
Select classes for inference:
☒ All archaeology ☐ Barrow ☐ Ringfort ☐ Enclosure



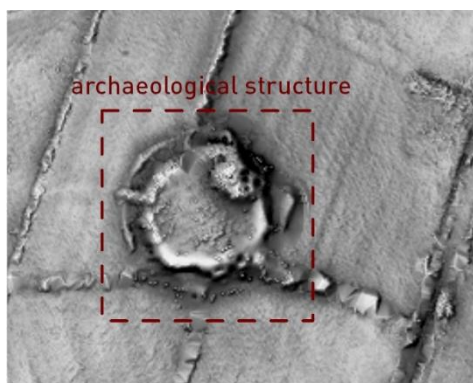
Select ML method

The user has the option to select which ML method they wish to use on their input data either *segmentation* or *object detection*.

Object segmentation: a labelled footprint that locates and delineates the object of interest within the image.



Object detection: a labelled bounding box that locates the object of interest within the image.



Select model

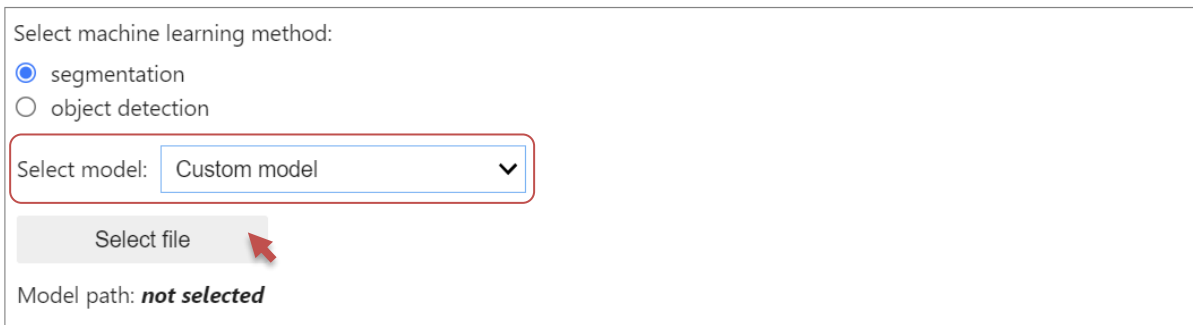
The user must select which ML model is used to identify potential archaeology sites in the input data. The two options that can be selected from the dropdown menu are *ADAF model* and *Custom model*.

ADAF model uses the default ML models that were trained specifically for the task of detecting the selected archaeological features in Ireland and the UK. The user has an option to run the detection for the following archaeological classes:

- All archaeology (all three classes listed below are treated as a single class)
- Barrows
- Ringforts
- Enclosures

Custom model allows the user to run detection using a user-provided ML model. The ML model is loaded into the ADAF by selecting the *Custom model* option from the "Select model" dropdown list. The import model button will appear. Click the button and select the desired model *.tar file.

Machine learning options:



Select machine learning method:

☒ segmentation
☐ object detection

Select model: Custom model ▼

Select file

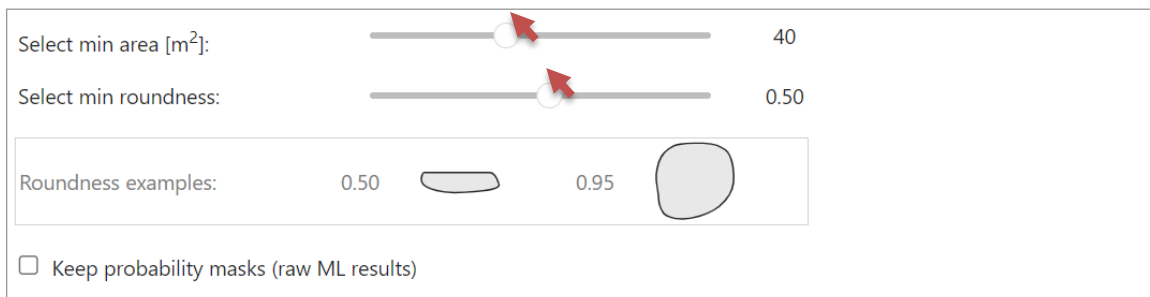
Model path: **not selected**

The user-provided ML model has to use the same model specifications as the default ADAF models. Instructions for creating a custom model are available in Appendix. This include appendices: Notebooks for retraining MLMs and Notebooks for creating training data.

Post processing options

A description of all available options for post processing module of ADAF.

Post-processing options:



Select min area [m²]: 40

Select min roundness: 0.50

Roundness examples: 0.50 0.95

☐ Keep probability masks (raw ML results)

Once the ML model has identified and segmented the potential monuments within the input DEM data, you can use post processing options to filter out segmentation polygons based upon their geometric shape and size.

Select minimum area

To filter out smaller segmentation polygons you can adjust the minimum area slider to remove detections smaller than the threshold value. The default setting for this is 40m², however if you are finding that too many small polygons are being created which don't represent archaeological features lower the threshold area and rerun the process.

Select minimum roundness

The ADAF has been trained to detect have a regular circle shaped footprint. Removing irregular shapes from the detected footprints can help to reduce the number of false positives. To filter out segmentation objects which are less rounded you can adjust the minimum roundness slider. Rounder objects would have a value closer to 1, whilst longer thin flat objects will have a roundness value closer to 0. The default setting for this is 0.5, however if you are finding that too many long thin polygons are being created which don't represent archaeological features increase the roundness threshold area and rerun the process.

Minimum Roundness index, which is calculated as:

$$\text{roundness} = 4\pi \cdot \text{area} / (\text{convex perimeter})^2$$

Roundness is expressed mathematically by dividing the area by the square of the perimeter of the polygon and normalising by multiplying the value by 4π . Shape index of a perfect circle is 1 and for a very elongated polygon it is close to zero.

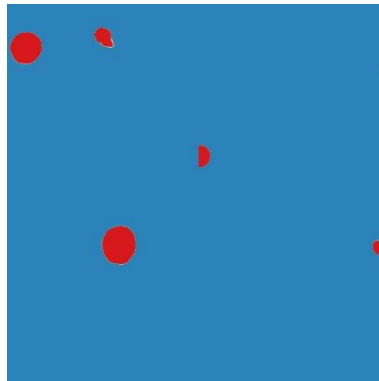
Saving raw predictions files

You can keep temporary files that are created during processing (normally they are deleted after processing) depending on the ML options chosen.

Probability maps

☒ Keep probability masks (raw segmentation results)

If you are using the semantic segmentation ML method, you can save the probability maps in GeoTIF format. These are raster files where every pixel is labelled by probability score predicted by the ML model.



Example of a probability map with probability values ranging from 0 (blue) to 1 (red)

Bounding box files

☒ Keep bounding box txt files (raw object detection results)

If you are using the object detection ML method, bounding box text files are created. These consist of a text file containing relative coordinates of the detected bounding boxes with probability score and metadata for georeferencing.

The following is a template for an individual result as outputted by object detection model:

$x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4$ <label> <probability score> <EPSG code> <resolution> <x min> <y max>

Where $x_n y_n$ are pixel (relative) coordinates, label is the detected archaeological class, probability score is ML output and is self-explanatory, and the rest (EPSG, resolution, x min and y max) are image metadata that are needed to transform relative coordinates into georeferenced polygons.

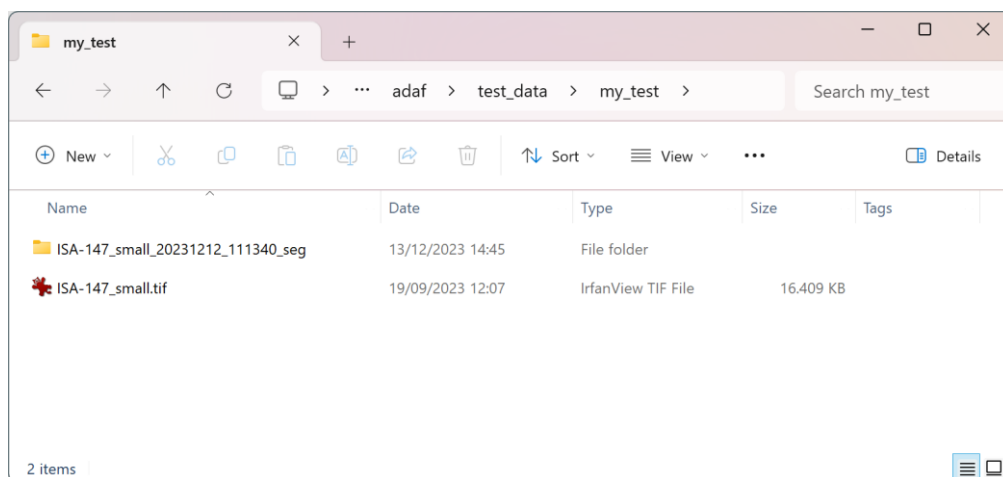
Example of a bounding box file created during object detection

```
927 259 999 339 AO 0.7047 29903 0.5 244112.0 169414.5
198 522 544 1022 AO 0.3502 29903 0.5 244112.0 169414.5
0 214 140 412 AO 0.2279 29903 0.5 244112.0 169414.5
466 453 817 779 AO 0.2147 29903 0.5 244112.0 169414.5
158 0 789 497 AO 0.0608 29903 0.5 244112.0 169414.5
1 447 88 616 AO 0.0572 29903 0.5 244112.0 169414.5
```


Output

For each run, ADAF saves the results in a separate folder located in the same parent directory as the input raster file. The name of the output folder is a combination of the name of the input file, the date/time stamp and the ML model type (seg for semantic segmentation, obj for object detection).

The results folder always contains the ML results in vector format (*.gpkg), which can be opened with most GIS tools and a corresponding log file which records all tool settings and processing parameters and processing times. If more than one ML model has been selected for processing (i.e. AO, barrow, enclosure, ringfort), the results are all saved in the same vector file with the "label" attribute to distinguish between the different detected archaeological classes. If the user has selected to save intermediate results, these will also be saved here.



Once created the output files can be imported into your GIS for checking and validation.



Example of detection bounding box output viewed on top of the input ALS data with the associated attribute table viewed in QGIS. This output was generated with only the AO ML model class selected



Example of segmentation polygon output viewed on top of the input ALS data with the associated attribute table viewed in QGIS. This output was generated with only the AO ML model class selected



Example of segmentation polygon output viewed on top of the input ALS data with the associated attribute table viewed in QGIS. This output was generated using all the ML model classes selected

Known issues

n/a

Appendix

ADAF models

The detection in ADAF employs 8 different ML models that were trained on an extensive dataset of ALS data with annotations for archaeological sites in Ireland, which include barrows, ringforts and enclosures.

We divided these models into two categories: four for object detection and four for semantic segmentation. For each type of archaeological site - barrows, enclosures, ringforts - there is a separate model. There is also a model that detects all archaeology and treats the three classes as one.

For the semantic segmentation models, we use the HRNet architecture, which is available in the AiTLAS toolbox. These models are trained on data of quality levels 1 (excellent) and 2 (some triangulation; enclosing feature identifiable) as specified by the domain experts, and on patches containing only the specific objects for which the models are trained. The other two quality levels are 3 (complete or almost complete triangulation) and 4 (monument barely visible). For example, if we train a model for barrows, the training data only contains data about barrows with quality levels 1 and 2. For validation and testing, we extend the data to include annotation qualities 1, 2 and 4 and use patches with different object types. The validation data for semantic segmentation incorporates feedback from domain experts based on analysing the predictions of our previous models.

For the object detection models, we use the FasterRCNN architecture from the AiTLAS toolbox. These models are trained on quality level 1 and 2 data, as specified by the domain experts, and on patches containing only the specific objects for which the models were trained. For example, if we train a model for barrows, the training data only contains data about barrows with quality levels 1 and 2. For validation and testing, we broaden the data to include annotation qualities 1, 2 and 4 and use patches containing only the specific objects for which the models were trained. The validation data for object detection does not contain any feedback from domain experts.

Finally, in both learning scenarios (object detection and semantic segmentation), all our data is sourced from the SLRM visualisation, using a patch size of 512 px. We perform data transformations, including random horizontal and vertical flips and rotations. The learning rate is set to 10^{-4} for optimal results.

SLRM visualisation

While the primary input data is DEM, the ML models were trained on simple local relief model (SLRM) visualisations. The SLRM images were created using the SLRM visualisation function available in the RVT toolbox. The parameter for radius for trend assessment was set to 20 pixels for images with a spatial resolution of 0.5 m and normalized with a linear histogram stretch between -0.5 and 0.5 and normalised to values between 0 and 1.

Radius for trend assessment	10 m (20 px for 0.5m image)
Min/max normalisation	between -0.5 and 0.5

The same settings are used in ADAF when preparing the input data for detection with ML models.

Notebooks for creating training data

From the main ADAF notebook select *Creating training samples* notebook under the “Training and retraining machine learning models” section and follow the workflow in the notebooks.

Introduction

A complete training dataset consists of so-called patches, i.e. tiled images that are provided with a labelled segmentation mask and/or labelled bounding boxes for semantic segmentation or object detection respectively. The patches are systematically extracted from the DFM raster and only patches with positive samples (i.e. with archaeological sites) are retained. Labelled masks and bounding boxes are created from polygons of archaeological sites (i.e. vector files).

Patches are created from the DFM in two steps - creating visualisation from DFM and cutting DFM into ML-readable patches.

Creating visualisations

- INPUT: DFM in GeoTIFF or VRT format
- OUTPUT: visualisation of DFM in GeoTIFF or VRT format

The default visualisation for the ADAF model is SLRM (Simple Local Relief Model). The Irish dataset was processed with the SLRM radius of 10 metres.

The SLRM visualisation with default parameters can be prepared using the Python functions included in ADAF. The same visualisation can be created at your own discretion using external software such as RVT (rvt_py package, RVT plugin for QGIS, RVT desktop app) or other third-party software. To replicate the default visualisation, make sure you use:

- radius 10 m (NOTE: in RVT the radius is given in pixels, convert it accordingly depending on the spatial resolution)
- normalise to between 0 and 1 values
- set all nodata and nan values to 0
- save the visualisation in GeoTIFF or VRT file format

ADAF can also be trained on visualisations other than SLRM. In this case, make sure that the visualisation raster has:

- either 1 or 3 bands
- is normalised between 0 and 1
- contains no nan values

Creating training data

- INPUT:
 - visualisation of DFM in GeoTIFF or VRT format
 - vector file(s) in SHP or GPKG format with labelled segments; one file per label
- OUTPUT:
 - a folder with visualisation tiles (GeoTIFF format),
 - a folder with masks for semantic segmentations (GeoTIFF format),
 - a folder with *labelTxt* for object detection (text file format).

The Python routines for creating patches are included in ADAF. The input visualisations must conform to the above guidelines and a separate vector file is required for each class (polygons with labelled archaeological features).

Users can set the size of the patches (i.e. the size of the image tiles in pixels) and the overlap of the tiles. If the “DFM” attribute is included in the vector file, this information is included in the

segmentation masks and bounding boxes as well. The DFM value was used in the original vector data to indicate the quality of the archaeological features on the DFM and can be used to filter out data by quality during ML training.

Default values for patch creation parameters are:

- tile size of 512 pixels
- overlap of tiles by 0.5 tile (i.e. 256 pixels)

The ADAF is set up for detection of three default classes, namely barrows, ringforts and enclosures. The procedure for creating patches follows this format, which means that masks can contain up to 3 different classes (not necessarily using the default names of labels). At least one vector file must be specified. In this case, the segmentation mask still has 3 bands, but only the first band is filled with the labelled data.

NOTE: When training or retraining the models, only one band from the segmentation mask is used at a time. The user must specify the band ID and the corresponding labelling name.

Examples

For retraining of the existing models, three vector files are given for barrows, ringforts and enclosures. All three classes are saved in the same segmentation mask file. The user first trains the model for barrows by selecting band 1 and naming the class with the "barrow" label. Then he continues with the model for ringforts by selecting band 2 and naming the class "ringfort" and finally he follows the same pattern for enclosures.

Training a completely new class requires a single vector file with labelled data when creating patches. As only one vector file has been provided, only the first band of the segmentation mask file is filled with valid data, the other two bands contain all zeros. When training a new model, the user must specify band 1 and give the class a new name.

The training data must be divided geographically into training, validation and test subsets. This is not covered by any Python script and must be done at your own discretion. As the split is done geographically, it can be performed before or after processing, i.e:

- slice the DFM raster into appropriate regions and create patches for each group separately,
- create patches for the entire DFM and split the resulting patches into 3 groups.

It is recommended to split the dataset in an approximate ratio of 60:20:20 (train:validation:test).

NOTE: If retraining the existing ADAF model, the visualisations have to be SLRM with 10 m radius and patch size of 512 px. If training a completely new ML model from scratch, any visualisation and tile size can be used. Refer to relevant literature or TII-ADAF technical report where we tested other visualisations and patch sizes.

Notebooks for retraining MLMs

Overview

We offer two specialized Jupyter notebooks designed for the tasks of retraining machine learning models (MLMs) for semantic segmentation and object recognition.

Each notebook serves a dual purpose:

- **Updating an existing ADAF model:** using an already trained MLM on the ADAF dataset for further training to improve its predictive power.
- **Training a new ADAF model:** training a new ADAF model without using an existing model. It's important to note that while this approach doesn't build on an existing ADAF model, the model isn't learned from scratch. By default, our methodology uses already trained weights available in the `aitlas` toolbox to have a head start in the training process.

Getting started

When using the notebooks, it's crucial to ensure the correct setup to allow for a smooth and successful retraining. Start by selecting the `aitlas` conda environment as the kernel (similar to the inference task). This environment contains all the necessary dependencies and libraries required for the tasks at hand.

Next, customise the input parameters and paths to your specific data. This step is crucial to ensure that the model is trained on the correct data and that the paths for accessing the input data and storing the output models are set correctly.

Once the environment is set and the parameters are configured correctly, proceed to run each cell in the notebook in turn. This step-by-step execution is designed to guide you through the process, from data preparation and model initialization to training and evaluation, ensuring that each phase is completed without problems before moving on to the next.

Jupyter Notebook Structure

1. Import packages

In this step, the required dataset classes and the machine learning models are loaded from the `aitlas` Toolbox. This includes:

- `aitlas.datasets.TiiLIDARDatasetSegmentation` for semantic segmentation
- `aitlas.datasets.TiiLIDARDatasetObjectDetection` for object detection
- `aitlas.models.HRNet` model for semantic segmentation
- `aitlas.models.FasterRCNN` model for object detection

This step also includes setting up the model configuration, which determines the number of classes, enables the download of pre-trained weights, sets the learning rate to 0.0001 and determines the evaluation metric (mean average precision for detection and intersection over union for segmentation).

2. Loading train, validation and test data

This part deals with the preparation of the training, validation and test data sets. All relevant input parameters and the data paths are defined and the data sets are loaded. Most of the input parameters for this step are the same for both learning tasks (semantic segmentation and object detection). Specific details are listed below:

- **batch_size:** The number of samples that are processed before the model is updated. A larger batch size can speed up processing, but requires more memory. A large batch size can lead to a runtime error if the computer does not have the required memory to load the data from the batch. The default setting for this parameter is 16, although adjustments to the specific system capabilities may be required.
- **num_workers:** The number of worker processes used to process data. Increasing the number of workers can significantly speed up data processing, but also increases memory and CPU/GPU consumption. The default value is 4.
- **object_class:** A string value indicating the type of archaeological object class you are interested in processing, e.g. 'AO', 'barrow', 'enclosure', 'ringfort'.
- **visualisation_type:** The visualisation type used for the patches, e.g. 'SLRM'.
- **DFM_Quality:** List of DFM qualities that should be included in the processed data, e.g. '1,2'.
- **shuffle:** This parameter controls the randomization of the data sequence before processing. To facilitate learning and prevent the model from memorizing the order of samples, it is recommended to enable shuffling (set to True) for training data. For validation and test datasets, shuffling is normally disabled (set to False).
- **data_dir:** The directory path where the input image data is stored.
- **annotations_dir:** The directory path in which the annotations are stored. This includes segmentation masks for semantic segmentation tasks and bounding box labels for object detection.
- **transforms:** An optional parameter that contains the list of transformations that will be applied to the input image data during processing. Although customization is possible, we recommend using the default settings specified in the notebooks.
- **target_transforms:** An optional parameter that contains the list of transformations that will be applied to the target (annotations – either segmentation masks or bounding boxes) during processing. Although customization is possible, we recommend sticking to the default settings specified in the notebooks.
- **joint_transforms:** An optional parameter that contains the list of transformations that will be applied to both the input image data and the target.

Semantic segmentation datasets have two additional parameters:

- **object_class_band_id:** An integer parameter that specifies the band in which the annotations for a particular object class are located within the segmentation masks. The band ID depends on how the data was pre-processed. An incorrect band ID can lead to a runtime error or loads annotations for a different object class. As this is a parameter with a zero-index, the values 0, 1 or 2 would be permitted for a segmentation mask with three bands. If the `object_class` parameter is set to 'AO', the annotations are loaded from all available bands. In this case, `object_class_band_id` should be set to None.
- **keep_empty_patches:** A Boolean parameter that determines whether empty patches are kept. It should be set to False during training, as "empty" data cannot be used for training. For testing or validation, True can be used to check how the model handles empty patches.

3. Model Creation

Here, the model is initialized with the predefined model configuration and prepared for training by setting up the required configurations and pre-processing steps.

4. Loading Pretrained ADAF Model (Optional)

This step is optional and allows you to load an existing MLM that has been trained on ADAF data. It's intended for users who want to further train an existing ADAF model. The only required input is the file path of the model. When you perform this step, the weights of the model created in step 3 are replaced by the weights of the model stored in the specified model path.

5. Training the Model

In this step, the actual training of the MLM is performed with the loaded training data and its performance is evaluated using a separate validation dataset.

There are three input parameters that the user must specify:

- **epochs:** The total number of training cycles the model will undergo. Each epoch represents a complete run of the training data set through the model.
- **model_directory:** Path to the directory where the trained model and its test points are stored. This is used to save the model during and after training.
- **run_id:** Name of the subdirectory within the <model_directory> directory in which the results of the various runs are saved.

Once this step is completed, the now trained model is saved in the specified directory (i.e. <model_directory>/run_id). The file with the name best_checkpoint_*.tar contains the model weights that achieved the best performance during training.

6. Model Evaluation

The final step is to evaluate the performance of the trained model on unseen test data. The only user input required is the file path to the model trained in the previous step, which is stored in the local file system. Once the evaluation process is complete, the evaluation metrics calculated for the test data are displayed.

Example of the output of a semantic segmentation model in the evaluation step:

```
{
  'IOU mean': 0.863975617602004,
  'IOU per Class': array([0.98613997, 0.74181307])
}
```

Our main focus is on the "IOU per Class" results, which provide specific Intersection over Union (IoU) scores for each class. The first value gives the IoU for the **background**, while the second is the IoU for the **archaeological object class**, as determined by the `object_class` parameter. The aim is to train models that have higher IoU scores for the selected object class (the second value).

Example of the output of an evaluation step for an object detection model:

```
{
  'map': 0.511692225393075,
  'map_50': 0.8819712400436401,
  'map_75': 0.5255807631868335,
  'map_small': -1.0,
  'map_medium': 0.4580733180064815,
  'map_large': 0.5440743028861223,
}
```

```
'mar_1': 0.550572486406042,  
'mar_10': 0.5869565010070801,  
'mar_small': -1.0,  
'mar_medium': 0.5909090904211426,  
'mar_large': 0.623404255032,  
'map_per_class': 0.511692225393075,  
'mar_100_per_class': 0.5869565010070801,  
'classes': 1.0  
}
```

The evaluation metrics presented here reflect the performance for the `object_class` only, without the background class. The mean average precision (mAP) is calculated for different IoU thresholds. It is advisable to optimize the [map_50](#) metric, which represents the mAP at an IoU threshold of 0.5, as this is a commonly used metric for object detection tasks.