ÉCOLE POLYTECHNIQUE DE L'UNIVERSITE FRANÇOIS RABELAIS DE TOURS
Spécialité Informatique
64 av. Jean Portalis
37200 TOURS, FRANCE
Tél +33 (0)2 47 36 14 31
www polytech univ-tours fr

SPECIFICATION

Project : Deep-Agora	technical specifications		
Emitter:	T. Boisseau	**Owner :** J.Y. Ramel	
Date of issue :	12/12/2022		

Validation

Name	Date	Valid (Y/N)	Comments
Ramel J.Y.	02/12/2022	N	
Ramel J.Y.	07/12/2022	Y	

History of changes

Version	Date	Description of the change
1	05/12/2022	Initial version
2	06/12/2022	Corrections from first validation
3	17/12/2022	Added unit & integration tests

TABLE OF CONTENTS

SPECIFICATIONS

This document aims to raise the technical and methodological requirements and for the project. After putting the project into context, it will expose state-of-the-art tools to use, analyse the solution at each stage, define test plans through scenarios and test sets and thus validate the proposal.

The actors of this project are:

- the client, which here are Centre for Advanced Renaissance Studies (fr. Centre d'études supérieures de la Renaissance) (CESR), for which a contact is Rémi Jimenes, lecturer and researcher.
- the Project/Product Owner (fr. Maître d'ouvrage) (fr. MOA), who is Jean-Yves Ramel, professor of computer science, director of Laboratory of Fundamental and Applied Computer Science of Tours (fr. Laboratoire d'Informatique Fondamentale et Appliquée de Tours) (LIFAT) and academic tutor for this project.
- the Project Manager / Scrum Master (fr. Maître d'œuvre) (fr. MOE), Théo Boisseau, an engineering student in his final year of study. He decide on the technical means used to design the product by what was defined by the product owner.

The product owner is responsible for representing the client by ensuring that the deadlines are met and that the product conforms. Thus, he is in charge to review documents such as this one.

The implementation phase starts on 4 January and ends with a final presentation around 3 April.

1. Context of the project

The client expressed the need for easy-to-use interactive software so that its users, CESR researchers and historians, could create their own scenarios for extracting elements of content (EOCs) from images of historical documents. These historical documents are mainly Renaissance corpora, accessible from the CESR database, and contain mainly printed or manuscript text, illustrations and page ornaments.

To convert these historical books into accessible digital libraries, LIFAT is developing software that participates in a complete processing chain, including layout analysis and text/illustration separation (i.e. segmentation of elements of content).

1.1. Objectives

This project aims to build a prototype of an optimisation software based on deep learning neural networks capable of segmenting textual and decorative elements of content from images of historical documents.

The user should not be responsible for training neural network models. Therefore, several deep learning models can be created and trained to extract the elements of content required in the different use cases of the software.

Due to its nature as a prototype, the system will need to be composed of computational documents combining scripts and good documentation. It must also provide access to training datasets and parameter storage files to reproduce the deep learning models created.

If the objective is achieved, the project can be continued and a scenario creation subsystem can be implemented to deploy the models created within it.

1.2. Hypotheses

For this project, we suppose there are no different typefaces in a single line of text. However, there may be, so it will only be taken into account in future versions of this project.

State-of-the-art DL frameworks are not good enough to segment handwritten characters in images of historical documents. If one appears during the development in the deep learning lab, it should be used in the project.

The elements of content to extract are:

- Blocks of texts
- Printed and handwritten text-lines
- Handwritten annotation
- Initial capitals
- Banners
- Figures with (or without) their caption
- Decorations

Ideally, there should be a model for each element of content. Otherwise, models can extract groups of elements of content, as few as possible.

No other methods than grouping connected black pixels exist in frameworks to post-process the binary mask of predictions. Because of that, the segmentation of characters is not possible for handwritten text and it has been removed from the list of elements of content. If state-of-the-art frameworks actually enable that, then it could be a solution to segment characters in images of historical documents. Appropriate new data sets with each character labelled individually should be used to train new neural network models. ALTO files could also identify each character by a Glyph tag.

The DL frameworks do not use binarization algorithms as a pre-processing step. If they actually do, the efficiency will not be as good.

Since the DL modules to develop cannot segment characters, ALTO is not the best format to export results. This is because the ALTO format assumes that the bounding boxes are rectangular and either vertical or horizontal, which is not what the DL models return. If it would be possible to segment characters, then it would not be that much of an issue. The client expressed that they need an ALTO output, so it must be used. A simpler version can be used if it wastes time on the project.

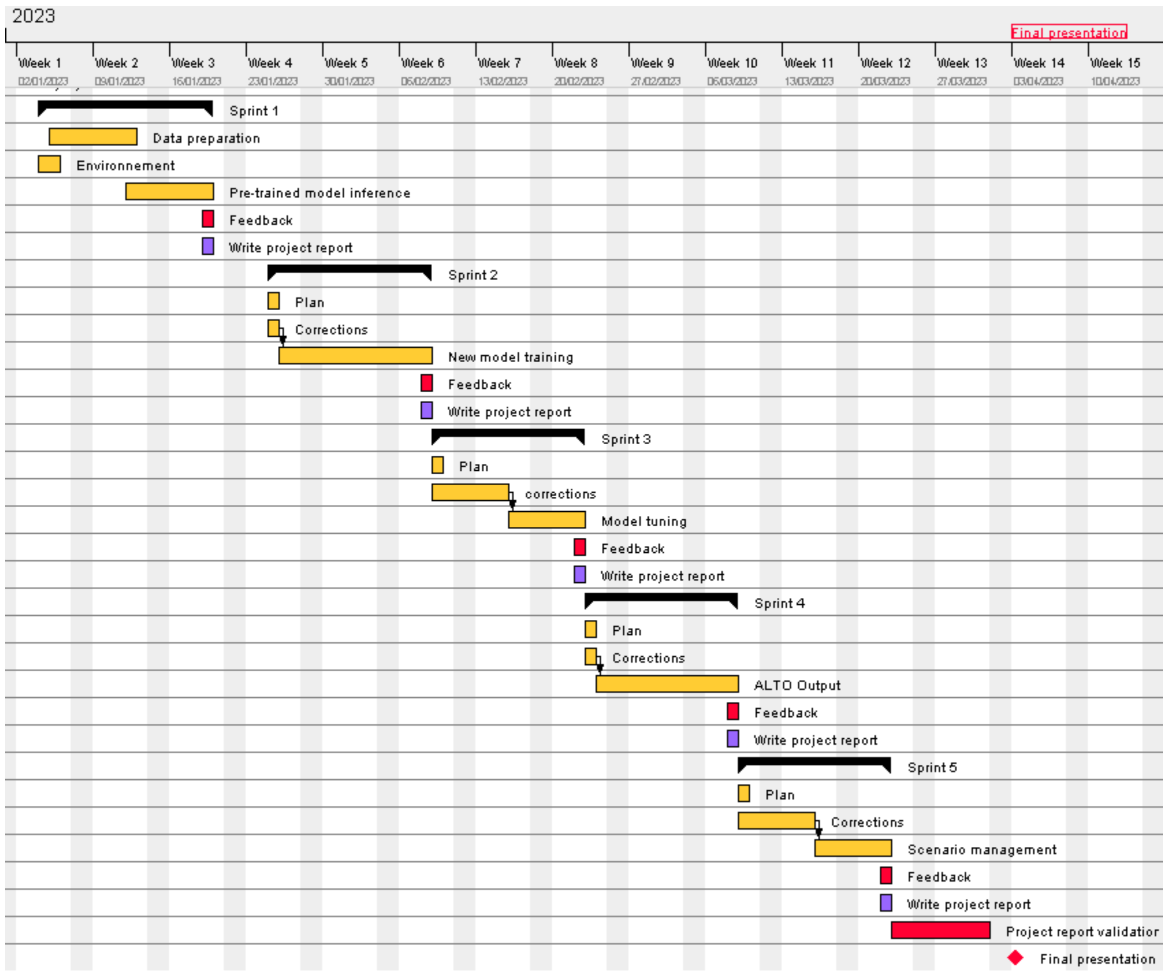
Agora will continue to evolve over the years and new needs may arise. Thus, documentation should be very good to ensure a successful takeover of the project.

A long period of time will be devoted to understanding the frameworks, working on the data and training models. If it wastes time on the project, the subject should be referred to the product owner.

1.3. Methodological

Contact with the product owner should be maintained as much as possible, as it will help me to improve and learn considerably as the project progresses.

To this end, we set sprints with a fixed duration of 2 weeks, which means there are 5 sprints. At least one deliverable, containing an e-mail, should be sent to the product owner at least every two weeks and preferably once a week. During the implementation phase, a meeting to get feedback about the product should be scheduled at the end of each sprint.



Agile planning of the implementation phase

All these sprints aim to prioritise and propose different versions of Deep-Agora:

1. The first minor release is expected to offer a prototype deep learning module that prepares a training dataset and uses a pre-trained model that semantically segments the page layout and returns certain elements of content from the list above.
2. After some corrections if necessary, the second minor release should offer another deep learning module targeting other elements of content. Most of them will most likely be trained on different data sets.
3. After some corrections if necessary, the third minor release should allow the previous models to be evaluated and tuned for better results.
4. A secondary fourth release should offer a solution to export the outputs of the previously trained models to vignettes and ALTO files.
5. An optional major release is to develop the functionality for end-users to import images from manifests and manage scenarios so that they can refine the elements of content they want.

The two last sprints being optional, it is envisaged that they will only cover the remaining backlog from previous sprints.

GitHub will be used for configuration management, by creating two different repositories:

- Deep-Agora, which contains the source code of the project
- Deep-Agora_DOC, which contains all the deliverables of the specification, analysis and modelling part of the semester 9

GitHub can also be used as a project management tool. It offers a similar feature to Trello called Projects, an adaptable spreadsheet that can also integrate with my issues and pull requests on GitHub to help me plan and track my work efficiently.

Files of elements of content and their vignettes have an explicit naming convention to locate them by name. It should indicate their encapsulation in other elements of content, hierarchically and separated by dots. For example: 1.10.5 (<page>.<paragraph>.<line>) or 1.1 (<page>.<illustration>).

2. General description

2.1. Project environment

This project is part of a larger research project between CESR and LIFAT. It is currently being carried out as part of a programme for the regional valorisation of old books (mainly dating from the Renaissance), namely the *Humanist Virtual Libraries* controlled by the CESR.
Within this programme, projects such as TypoRef which aims to identify specimens of similar typical characters, and BaTyr, a database of illustrations extracted, need software that meets the requirements of this project.

CESR does not have powerful computing machines capable of training deep neural network models, but it has internet, several machines and a large amount of remote and on-premises storage.

Agora, the software developed and published ten years ago by LIFAT to process images of historical documents, is undergoing a complete overhaul in this project. Its technologies need to be updated and, above all, its overhaul should meet the previously unattainable need for simplicity in scenario creation.

Therefore, no takeover of the existing system is planned, as it has to be completely redesigned.

2.2. User characteristics

End users of Deep-Agora are all historians of CESR. They have a sufficient but moderate command of computer tools. They often use them but need extensive training or solid documentation to use them in the case of advanced tools with complex functions. They did not have a satisfactory experience with Agora, as its interface was too complex. They do not need user access rights to use Agora.

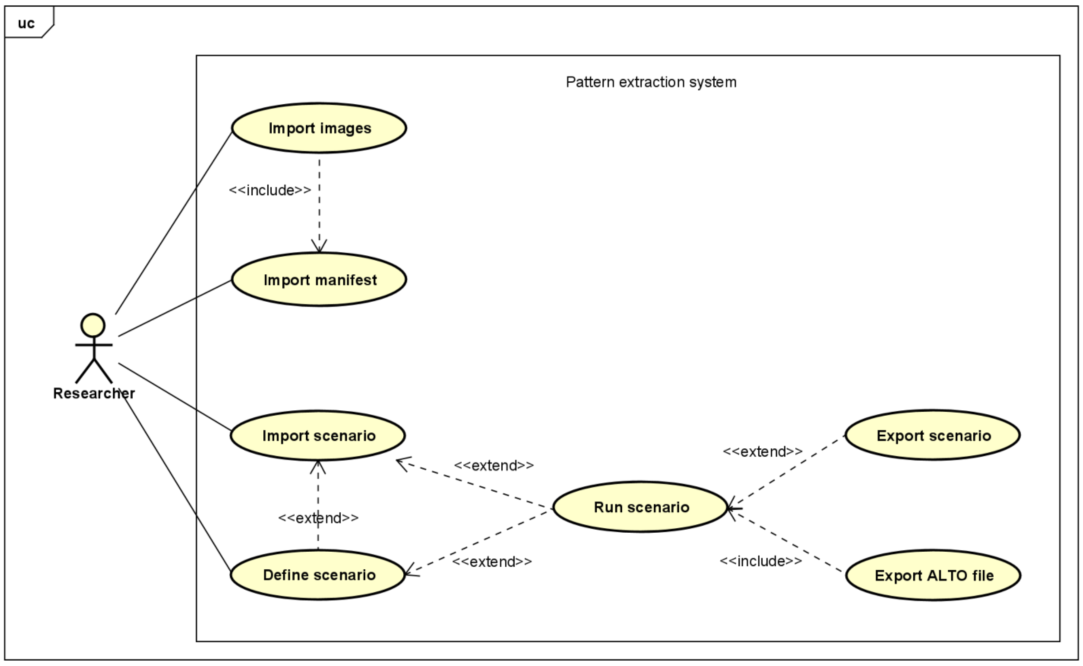
2.3. System features

Users will use this software to extract patterns.
For this purpose, they should:

- import a manifest (redirecting to a collection of images)
- import images directly
- import an existing scenario from their file system

- define a scenario by defining iterative operations
- run the scenario to view the extracted content items
- export the results to an ALTO file
- export the scenario to the file system, making it available for import.

In practice, from all the images in a collection, users select a typical one on which they build and test their scenarios to extract elements of content. They can then save their scenarios and run them on other collections.



Use case diagram of Deep-Agora (pattern extraction system) by a CESR researcher

2.4. General structure of the system

Training deep neural network models is not a task intended for Deep-Agora end users. This part of the project is to be carried out outside the software system, but within the environment, as the engineer's system. It includes training data preparation of the datasets found on the internet and the deep learning laboratory where the neural network models are trained.

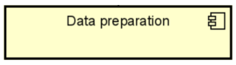
The software itself, Deep-Agora, simply receives trained neural network models and uses them as operations in the scenarios to extract elements of content.

Rules are another type of operation that can complete the scenarios with a more descriptive approach, to specifically label or merge elements. This type of operation exists and will be part of Deep-Agora but is not the subject of this project.

The scenarios are managed by projects that deliver the images, provide them with available operations and save their results.

Image importation will provide projects with usable images that are either directly provided or whose IIIF links allow them to be found from a manifest.

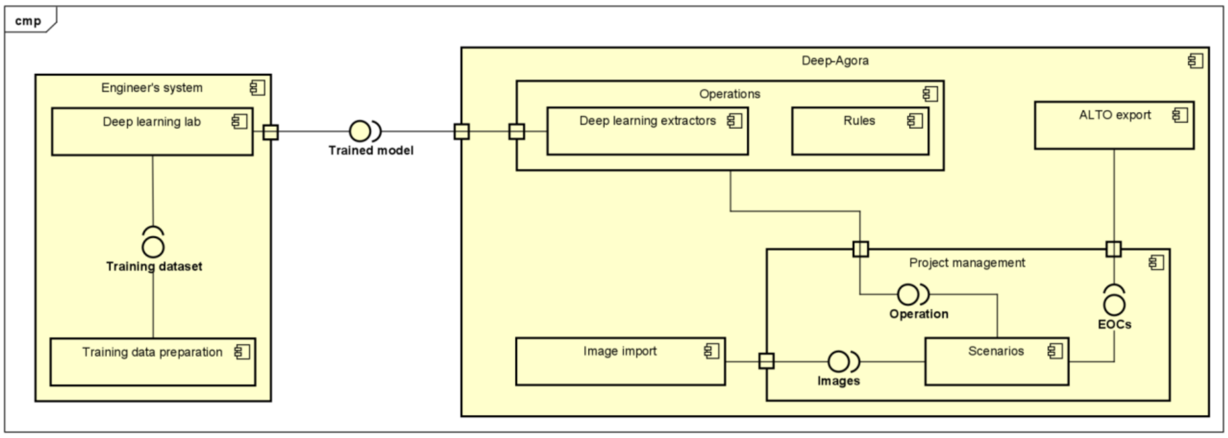
ALTO export converts the results of the scenarios to an ALTO XML data structure and saves them to ALTO files.



: Module (Sub-system)

: Output of one module and input of another (Interface)

: Encapsulation of module inputs/outputs (Port)



Component diagram of the engineer's system and the Deep-Agora software system

3. Description of the external interfaces of the software

3.1. Hardware/software interfaces

IIIF links require an Internet connection to send HTTP requests to online virtual libraries.

The machine on which the engineer's system deep learning lab will be run should have a GPU to process neural network training faster.

Data sets for training will be stored in the engineer's system.

3.2. Human/machine interfaces

The prototype should be made of computational documents combining scripts and good documentation, such as Jupyter Notebooks.

The HMI of the software should display at least 4 panels:

- Scenario: different operations in iterative order
- Tree of EOC: elements of content organised structurally in a tree
- Existing label: a list of extracted labels
- Current image: a picture of the image being analysed

To build scenarios, operations can be accessed through different dedicated tabs. A File tab is dedicated to the management of the user's project. A project tab is dedicated to configuring it. A scenario Tab is dedicated to clearing it or undoing the last operation performed.

The simplicity of the HIM to create scenarios, reuse them and adapt them to different documents is essential.

3.3. Software/software interfaces

To import images at the beginning of a project, databases are indirectly requested through the use of IIIF links. IIIF links are URLs that return images in response to a standard HTTP or HTTPS request. These links can redirect to internal or external networks.

Trained neural network models are implemented in Deep-Agora manually, by restoring their parameters from storage files.

In the engineer’s system, datasets are downloaded manually through websites, then transformed, and models are trained using a state-of-the-art generic framework for historical document processing.

4. Functional specifications

🟥 : Deep-Agora software system

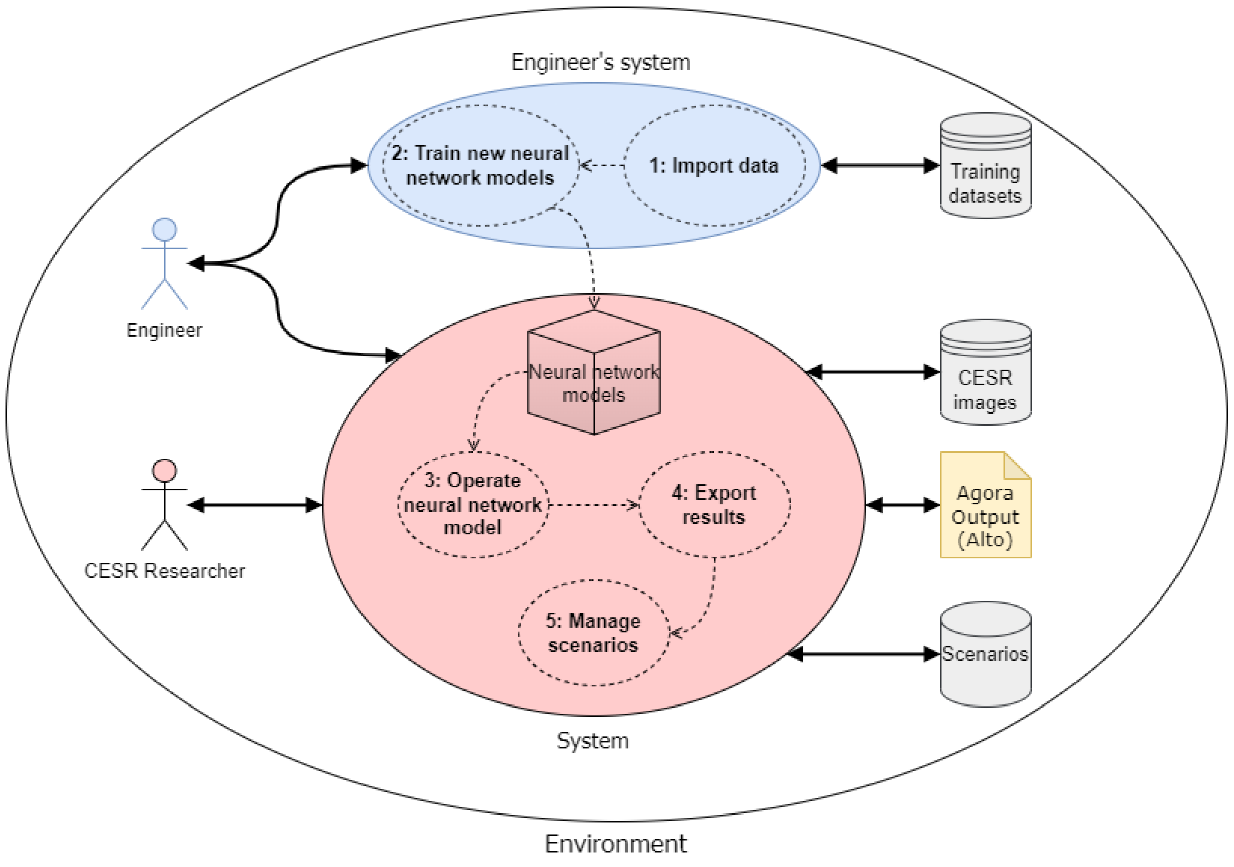
🟦 : External to the Deep-Agora software system

2: Train new neural network models : Main project tasks

Datastore (data under various formats) : CESR images

Database (structured data) : Scenarios

Deep learning modules : Neural network models



System-environment boundary diagram and main project tasks

4.1. Definition of module 1

Name	Training data import module
Summary	Implement a pipeline to prepare a training dataset for neural network model.
Priority	Primary
Inputs	State-of-the-art training datasets, with images and different labels of EOCs.
Preconditions	Available datasets are downloaded on local storage. Their images are in different formats, such as TIFF or JPEG. Their labels are stored under PAGE XML format.
Outputs	A training data folder containing a class file and test, train and validation subfolders. Each of the subfolders contains an image folder and a label folder of mask images.
Postconditions	The original datasets were selected and merged for the pipeline. Dataset is split into test, train and validation folders. Dataset themselves divided into pairs (images, labels) of images with the same name (excluding the extension). The images in the image folder are in JPEG format. The images in the label folder are in PNG format and are RGB masks of the regions to segment with a different colour for each EOC. The class file is multi-label and has one row for each combination of EOC and each one has 3 RGB values and an attribution code.
Interacting components	• Deep learning lab component • File system • Datasets from the file system • Engineer's system
Prioritized features list	1. Acquire datasets from the file system 2. Distribute them in according to the EOCs targeted by the neural network models 3. Create a training data folder containing a class file and test, train and validation subfolders. For each subfolder, create an image folder and a label folder. 4. Convert the images to JPEG format and put them in the image folders 5. Convert PAGE XML label files to PNG mask images with colours associated with the class file and put them in the label folders
Error handling and implementation	• Some datasets are not available for download ➔ Delete them from the list and evaluate again the feasibility of the model
Unit tests	• Unknown requested EOCs → raise an unknown parameter exception • There are X % of the pairs (images, labels) in the train folder, Y % in the test folder, Z % in the validation folder, and X+Y+Z=100% of the pairs from the datasets • Each image in the image subfolder → has the JPEG format • Each image in the image subfolder → has a PNG mask with the same name in the label subfolder • Each PNG mask in the label subfolder → uses the colours from the class file
Integration tests	• Importing datasets from the file system → no memory allocation error (data fits into memory) • The module is run before the deep learning lab functions

4.2. Definition of module 2

Name	Deep learning lab component
Summary	Computational documents that implement frameworks for historical document

	processing to train new neural network models to semantically segment images into targeted EOCs.
Priority	Primary
Inputs	A training data folder containing a class file and test, train and validation subfolders. Each of the subfolders contains an image folder and a label folder of mask images.
Preconditions	Subfolders are themselves divided into pairs (images, labels) of images with the same name (excluding the extension). The images in the image folder are in JPEG format. The images in the label folder are in PNG format and are RGB masks of the regions to segment with a different colour for each EOC. The class file is multi-label and has one row for each combination of EOC and each one has 3 RGB values and an attribution code.
Outputs	Neural network model objects stored in a file. If inference, probability map of each attribution code (combination of EOCs).
Postconditions	Neural network model objects have been saved using the framework defined for loading and saving models in the non-functional specifications. The probability map is pixel-wise labelled and its dimensions are image_height X image_length X number_of_attribution_codes.
Interacting components	• Training data import module • Deep learning lab component • File system • Engineer's system
Prioritized features list	1. For each neural network model, acquire the right pipeline according to the EOCs targeted 2. Declare the training parameters of the generalist deep learning framework. 3. Train the model 4. Validate it or reiterate from 2.
Error handling and implementation	• The dataset does not dispose of enough samples or is not balanced enough. ➔ Communicate the error to the product owner ➔ Or pass to another model
Unit tests	• Training parameters→ are stored in a JSON file • Predicted mask → use the attribution codes of the class file • For any image in the test folder → a probability map is predicted by the model for each EOC it was trained to extract
Integration tests	• Regarding requested EOCs to Training data import module → a pipeline for these EOCs has been developed • Models' variables → are serialised in the file system

4.3. Definition of module 3

Name	Import module
Summary	Import images from a manifest.
Priority	Optional
Inputs	A manifest file
Preconditions	The manifest is in JSON format and contains page image URLs.
Outputs	A page folder containing images
Postconditions	The images in the page folder are in JPEG format and were downloaded from the manifest file.
Interacting components	• File system • Manifests • Deep-Agora system
Prioritized features list	1. Load manifest 2. For each page of a corpus in the manifest, extract its IIIF links and store them in a list 3. Download images from IIIF links 4. Convert them to JPEG format 5. Save them in the page folder
Error handling and implementation	• The IIIF link does not refer to an available image. ➔ Pass to the next one
Unit tests	• For each IIIF links → there is a JPEG image in the page folder
Integration tests	• The module connects to the Internet

4.4. Definition of module 4

Name	Conversion of EOCs to tree
Summary	Interpret the outputs of the neural network models to construct EOC overlap trees from the attribution codes.
Priority	Secondary
Inputs	Probability map of each attribution code (combination of EOCs) and class file.
Preconditions	The probability map is pixel-wise labelled and its dimensions are image_height X image_length X number_of_attribution_codes. The class file is multi-label and has one row for each combination of EOC and each one has 3 RGB values and an attribution code.
Outputs	An XML tree structuring EOC regions
Postconditions	Each node is an EOC that is associated with its coordinates in the original image and a name according to the naming convention for vignettes.
Interacting components	• Deep learning extractors • Scenarios • Deep-Agora system
Prioritized features list	1. Threshold the probability map to obtain the matrix of attribution codes for each pixel 2. Draw bounding boxes of each label 3. Structure EOC regions in an XML tree 4. Associate coordinates of the bounding boxes and a name to each node according to the naming convention for vignettes
Error handling and implementation	-
Unit tests	• For XML trees → illustrations are outside any text block, and lines of text, as well as annotations, are inside text blocks. • Nodes of the XML tree → have a label, coordinates and a name following the naming convention.
Integration tests	-

4.5. Definition of module 5

Name	Export module
Summary	Export vignettes of elements of content and export their location and coordinates into ALTO files for each page.
Priority	Optional
Inputs	An XML tree structuring EOC regions
Preconditions	Each node is an EOC that is associated with its coordinates in the original image and a name according to the naming convention for vignettes.
Outputs	Vignettes of each element of content, structured in a results folder and an ALTO file.
Postconditions	The vignettes in the vignette folder have their names hierarchically structured and they have the same content as the corresponding bounding boxes in the original image. The ALTO file structures all the input elements of content in a tree so that each EOC is included in its overlapping EOC regions.
Interacting components	• Scenarios • File system • Deep-Agora system
Prioritized features list	1. For each node of the XML tree, extract vignettes from the original image and save them in the vignette folder 2. Convert the XML tree to ALTO and save it in a file
Error handling and implementation	-
Unit tests	• Nodes of the XML tree → have a corresponding vignette with the same name in the vignette folder • Tags of the ALTO file from the XML tree → have an id (name), coordinates and a style (label) • For each node of the XML tree that has a parent node, the corresponding tag of the ALTO file → has the corresponding parent tag
Integration tests	• ALTO file is stored in the file system

4.6. Definition of module 6

Name	Manage scenarios
Summary	End users can manage their project and refine the elements of content they want.
Priority	Optional
Inputs	HMI instructions
Preconditions	The other modules have been implemented.
Outputs	-
Postconditions	-
Interacting components	• Project management component • File system • Deep-Agora system
Prioritized features list	1. Enable end-user to organise their project 2. Enable saving and loading of scenarios 3. Enable scenarios to be run on multiple images 4. View results of scenarios on HMI panels
Error handling and implementation	-
Unit tests	• User can select images and a current image • Operations can be selected by tabs • Operations can be added to and removed from scenarios • Scenarios are saved in XML files
Integration tests	• Operation interface can be implemented by Deep Learning Extractors and Rules • Calling the Import module fills the image folder of the project • Results are an XML tree handled by the Export module

4.7. Definition of training datasets

The structure of a document refers to the organization of every element within it. The organization of these elements in specific places of the document constitutes the layout of the document. Detecting and extracting information is essential to get the geometry presented in a document. A document may consist of several blocks of text such as title, paragraphs, main body text, text lines, graphics, tables and more. Many datasets are publicly available to promote research that deals with the structure of documents.

The datasets to use should therefore contain labels such as layout, text-line and graphics.

The percentage of training and test data of the datasets should be defined during the project, accordingly to the amount of data available for selected elements of content and during the evaluation of the models.

5. Non-functional specifications

5.1. Development constraints and design

The state-of-the-art framework for training the neural network model is dhSegment. The programming language is Python and the computational documents are made of Jupyter Notebooks. Jupyter Notebooks can be made of any IDE or Jupyter Lab, however, they use the Conda environment. IIIF links are requested by HTTP or HTTPS protocols.

5.2. Functional and operational constraints

5.2.1. Performance

There is no specific time limit for processing multiple images. However, the use of the HMI must be reactive as the construction of scenarios requires a great deal of experimentation by the user.

5.2.2. Capabilities

The software runs on a single computer. It takes 3 different types of neural network models: text lines, ornaments and figures. They are implemented manually and on demand. A model can process only one image at a time. The data from outside the system can consume significant storage.

The software itself should be light. However, memory constraints can become a risk for neural network models. This risk will be evaluated by making the prototype, and the right specifications will be detailed in the final report.

5.2.3. Operating modes

As a prototype, it can be started with a Jupyter Notebook file after starting a Jupyter server.
After implementing the HIM, it can be started with a python script. It remains on until the user closes the window.

5.2.4. Controllability

The data import should display data samples before and after pre-processing.
The deep learning lab should display the training parameters, the learning curves, the number of live epochs and a graphic of the learning curves at the end of the training.
During the prototype part, the results of the deep learning extractor should display the bounding boxes encapsulating the targeted elements of content on the image.
The ALTO export and the scenario management respectively display the ALTO file and the serialised scenario produced.

5.2.5. Security

The level of confidentiality of the system is non-existent: there is no user access control, and no keywords or passwords.

5.2.6. Integrity

ALTO files and serialised scenarios are not protected. The end user can save them wherever they want.
The software only connects to the Internet when a manifest requires it. There is no protection.

5.3. Maintenance and development of the system

Maintenance of the HIM is palliative (fr. curative), which means it should only be done punctually on specific issues.

Maintenance of the operations and scenarios is curative, which means they should be restored if there is an issue. It should also be perfective to improve efficiency and evolutive since new needs can appear.

GLOSSARY

A	
Agile	A set of software development practices designed to create and respond to changes as the project progresses
ALTO	Analyzed Layout and Text Object is an open XML Schema to describe the layout and text of a document image
B	
Binarization	Conversion of a picture to only black and white
C	
CESR	Centre d’études supérieures de la Renaissance
D	
Deep Learning	A type of artificial intelligence where the machine learns by itself using neural networks inspired by the human brain
E	
EOC	Element Of Content
F	
Framework	A platform that provides a foundation for developing applications
I	
IIIF	A standardised method of describing and delivering images over the web
M	
Manifest	A file containing metadata and URLs for a group of images
P	
Pipeline	A series of data processing steps
S	
Sprint	A time-bound effort, i.e. the duration is agreed and fixed in advance for each sprint

BIBLIOGRAPHY

Meeting minutes deliverables

20221012SpecsOutputs, Output specifications deliverable

20221020SpecsFrameworks, Framework specifications deliverable

20221110SpecsDataSets, Data set specifications deliverable

20221115Planning, Planning deliverable

20221116DiagComponent, Component diagram deliverable

20221116DiagUseCase, Use cases diagram deliverable

PARADIIT Project, https://sites.google.com/site/paradiitproject/

dhSegment paper, https://arxiv.org/abs/1804.10371

GitHub dhSegment, https://github.com/dhlab-epfl/dhSegment

INDEX