Ecole Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Research & Development Project

2022-2023

Deep-Agora

Incremental segmentation of images of old documents

Company
Centre d'études supérieures de la Renaissance

Student
Théo BOISSEAU (DI5)

Academic supervisor
Jean-Yves RAMEL

Industrial supervisor
Rémi JIMENES

4th April 2023

List of contributors

Company

Centre d'études supérieures de la Renaissance
59, rue Néricault Destouches
37020 Tours, France
cesr.univ-tours.fr

Name	Email	Quality
Théo BOISSEAU	theo.boisseau@etu.univ-tours.fr	Student DI5
Jean-Yves RAMEL	jean-yves.ramel@univ-tours.fr	Academic supervisor, Computer Science Department
Rémi JIMENES	remi.jimenes@univ-tours.fr	Industrial supervisor

Warning

This document was written by Théo BOISSEAU hereinafter referred to as the author.

The company Centre d'études supérieures de la Renaissance is represented by Rémi Jimenes hereinafter referred to as the industrial supervisor.

The Polytechnic School of the University of Tours is represented by Jean-Yves RAMEL hereinafter referred to as the academic supervisor.

By the use of this document model, all parties of the project accept the conditions defined hereunder.

The author acknowledges taking full responsibility for the content of the document, as well as any judicial consequences that might ensue as a result of non-compliance with laws or copyright.

The author certifies that the statements of the document are truthful and takes full responsibility for the veracity of the statements.

The author certifies not to appropriate the work of others and that the document contains no plagiarism.

The author certifies that the document contains no statements that are defamatory or reprehensible under the law.

The author acknowledges that they may not publish this document in part or in whole in any form without prior approval of the academic supervisor and the industrial supervisor.

The author authorizes the Polytechnic School of the University of Tours to publish this document in whole or in part in any form, including after modification with due acknowledgement of the source. These publications must be free of charge and come along with this warning.

To cite this document

Théo BOISSEAU, *Deep-Agora: Incremental segmentation of images of old documents*, Research & Development Project, Ecole Polytechnique de l'Université de Tours, Tours, France, 2022-2023.

```
@mastersthesis{
    author={BOISSEAU, Théo},
    title={Deep-Agora: Incremental segmentation of images of old documents},
    type={Research & Development Project},
    school={Ecole Polytechnique de l'Université de Tours},
    address={Tours, France},
    year={2022-2023}
}
```

This document has been formatted with LaTeX
thanks to the document class `polytech.cls` V2.5.2(2019/08/07)

Contents

List of Figures

1 Introduction

1 Actors, issues and context

The Research & Development Project (R&D project) is the final work that a student engineer must complete to obtain his diploma. It places the future engineer in a project situation by making him/her produce personal work and invites him/her to show initiative and maturity regarding a specific high-level problem. The R&D project is the subject of a dissertation and an oral presentation to a jury each semester, which lasts at least two days a week throughout the fifth year, i.e. 26 weeks.

This report aims to provide both the main document that everyone can read and all the technical and methodological elements. It consists mainly of complete sections of the different documents produced, with the technical sections in the appendix.

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. He is responsible for representing the client by ensuring that the deadlines are met and that the product conforms.
- the Project Manager / Scrum Master (fr. Maître d'œuvre) (fr. MOE), Théo Boisseau (that is me), an engineering student in his final year of study. I decide on the technical means used to design the product by what was defined by the product owner.

The client expressed the need for easy-to-use interactive software so that its users, 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.

The simplicity of creating extraction scenarios, their reuse and their adaptation to different documents are essential dimensions of the requirement.

However, this simplicity should not unduly compromise the reliability and performance of the software. Image processing of historical documents is a particularly difficult task notably because of broken characters, stains, and poor paper quality.

To convert historical books into accessible digital libraries, LIFAT is developing image processing software that participates in a complete processing chain, including layout analysis, text/illustration separation (i.e. segmentation of elements of content), optical character recognition (i.e. OCR) and text transcription. This project focuses on layout analysis and segmentation of elements of content of historical documents.

In recent years, the performance of some deep learning techniques has surpassed that of shallow methods established by experts on various image processing tasks. As this progress has made many computer vision tools available, it now seems possible to meet this need with a completely new approach.

2 Objectives

This project aims to propose a new approach based on deep learning neural networks to solve this image processing problem.

To this end, the Deep-Agora R&D project aims to build a prototype of an optimisation software capable of extracting textual and decorative elements of content from images of historical documents.

The user should not be responsible for training the 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 needs 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.

3 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.

The end users will only look for these elements of content:

- Blocks of texts
- Printed and handwritten text-lines
- Handwritten annotations
- Initial capitals
- Banners
- Figures
- Decorations

And will not look for more modern or scientific ornaments, such as:

- photographs
- tables
- graphics
- formulas

Either way, new data sets should be used to train new neural network models.

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

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 tools for deep learning, working on the data and training models. If it wastes time on the project, the issue should be referred to the product owner.

4 Methodological bases

An Agile project management method is used to create learning loops to quickly gather and integrate feedback. Therefore, the Scrum method should be preferred in which ideology is to:

- learn from experience
- to self-organise and prioritise
- to reflect on gains and losses to continuously improve

Contact with the product owner should be maintained as much as possible, as it helps 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. The implementation phase starts on 4 January and ends with a final presentation around 3 April.

GitHub is 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 or Jira 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

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 networks, but it has 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 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.

3 System features

Users 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, label them, split them and iteratively merge them. They can then save their scenarios and run them on other collections.

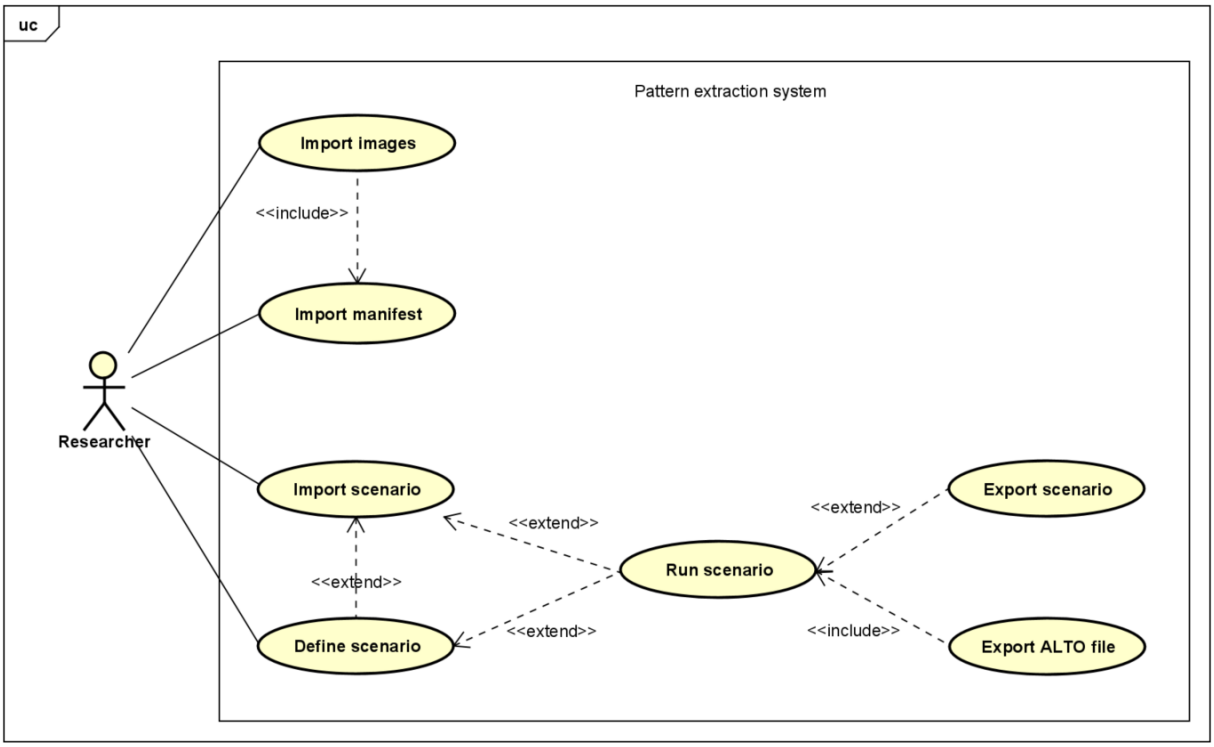


Figure 2.1: *Use cases diagram*

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.

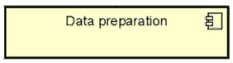
The software itself, Deep-Agora, simply receives trained neural network models and uses them as operations in the scenarios to extract elements of content.

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 provides 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)

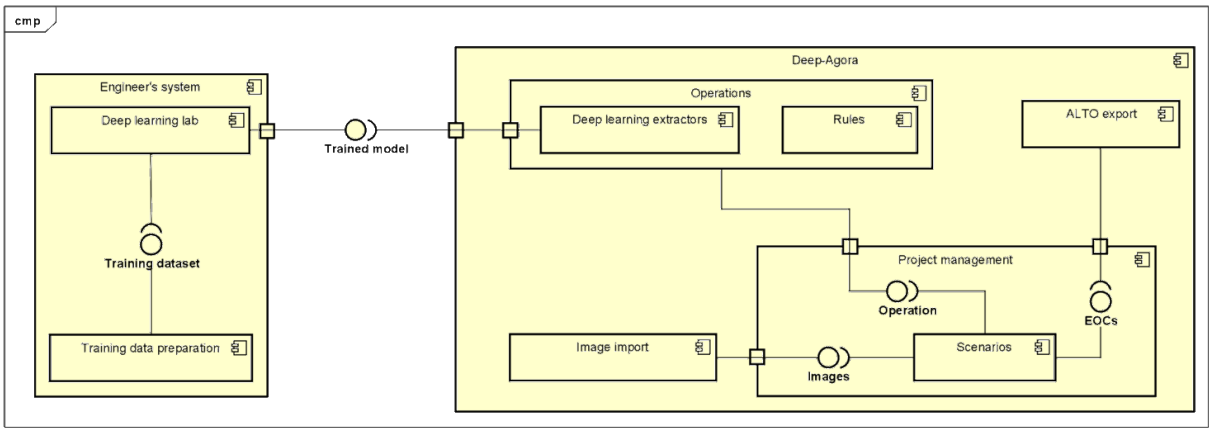


Figure 2.2: *Component diagram*

3 State of the art / Technology watch

1 Existing system

Ten years ago, LIFAT developed and published Agora to segment images of historical documents.

It works page by page by structuring them into elements of content called EOCs. For each page, the resulting structure is a tree of EOCs, some of which are the parents of others, just as text boxes are the parents of text lines. The input of the software is a list of images to be segmented, and the output is composed of an ALTO file and vignettes of EOCs.

The software is not automatic. It uses a strong interaction with the user. Among all the images in a document/collection, the expert (user) selects a typical one on which to build and test specific scenarios, which are then applied to all others.

The software was typically used for 2 cases:

- to extract all content items of a document. These items can then be used by the Retro software to recognise characters for transcription or .
- to extract only figures such as initial capitals and banners. These figures can be stored in a database such as the CESR's Base de Typographie de la Renaissance (BaTyR) for renaissance typography, in order to study their history and thus that of their creators.

Definition of rules

After the document image has been binarised with the possibility to choose between different binarisation algorithms, the black pixels are processed by scenarios. Scenarios are sets of steps to build the tree of EOCs. Each step is a configurable operation on the EOC tree. Operations can consist of deleting or editing an element of content or of creating new ones by setting up rules.

These rules are either labelling rules or merger rules. The labelling rules concern the position, size and neighbourhood of the content items. They give a label to regions of the image. For example, "if black pixels touch each other, they are connected components". The merger rules concern only the neighbourhood of the content items. For example, "if the distance between two elements of content identified as characters is less than 3 pixels horizontally, they are merged into a single character".

Export to ALTO format

ALTO (Analyzed Layout and Text Object), a standardised XML format, is used to save page layout information and OCR-recognized text from printed documents, including books, journals, and newspapers.

The Metadata Encoding and Transmission Standard (METS) provides metadata and structural information, while ALTO contains content and physical information. This extension schema is intended to be used in conjunction with METS. However, ALTO instances can also be used outside of METS as a stand-alone document.

There is one ALTO file per image. ALTO files contain a style section where labels are listed. The layout section contains what is on the page. A page is divided into several regions:

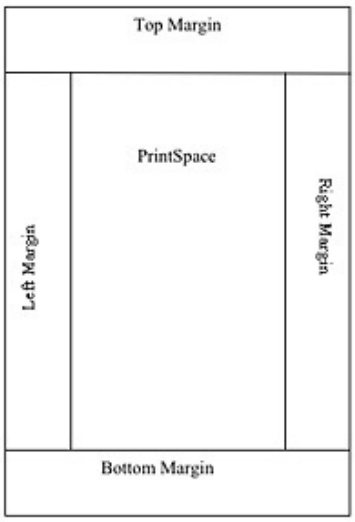


Figure 3.1: *Regions of a page*

To change

The software interface is complex and therefore too difficult for the end user to use. Indeed, most of the rules require a lot of parameters and without being an expert user of AGORA, it can be very easy to get lost in the interface of complex scenarios.

Historical documents are very often damaged in some way: characters are broken, there is the presence of stains, or the paper is of poor quality. As a result, binarisation is problematic as multiple algorithms have already been implemented that cannot satisfy sufficient efficiency on this type of document.

Deep learning could be a solution to binarisation and to this complexity of rules by avoiding all these configurations. DL modules could then be used iteratively with simpler scenarios to extract specifically defined elements of content (EOCs).

2 Segmentation using deep learning approaches

It is not necessarily the case that deep learning is the best way to segment historical documents. The best approach will depend on the specific characteristics of the documents and the goals of the segmentation.

Deep learning approaches, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), can be effective for solving the segmentation problem of historical documents because they are able to learn complex patterns in the data and can handle large amounts of unstructured data. This can be especially useful for segmenting historical documents, which may

have various types of formatting and layout, and may use old or archaic language and writing conventions.

However, there are also several potential drawbacks to using deep learning for segmenting historical documents. One disadvantage is that deep learning approaches often require a large amount of labelled training data, which may not be available for historical documents. Additionally, deep learning approaches can be computationally intensive, which may be a concern for large datasets or for documents with complex layouts.

Overall, it is important to consider the specific characteristics of the historical documents being analyzed, as well as the goals of the segmentation, when deciding whether to use deep learning or another approach for segmenting these documents. This is why other approaches, such as rule-based approaches may be used on demand by the Deep-Agora users.

2.1 U-Net

A U-Net is a type of convolutional neural network (CNN) that takes an image as input for image segmentation tasks, developed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015. It is designed to be efficient and easy to train, making it a popular choice for image segmentation tasks. Its architecture is shaped like a U by consisting of an encoder network and a decoder network connected by a series of skip connections.

The encoder network processes the input image and extracts features from it. It consists of a series of convolutional layers and max pooling layers. The convolutional layers are responsible for learning features from the input image, while the max pooling layers downsize the feature maps and reduce the computational complexity of the network.

The U-Net uses skip connections, also known as shortcut connections, to connect the encoder and decoder networks. These connections allow the U-Net to preserve information about the spatial context of the image, which is important for accurate image segmentation.

The decoder network uses the features from the encoder to generate a segmentation mask that indicates the boundaries between different regions of elements of content in the image. It typically consists of a series of transposed convolutional layers. These layers apply a transposed convolution operation to the input, which upsamples the feature maps and increases the spatial resolution. The transposed convolutional layers use the skip connections from the encoder network to incorporate information about the spatial context of the image.

Finally, the output of the U-Net is a segmentation mask where each pixel value corresponds to the probability of a specific label, such as an element of content or a combination of elements of content. The U-Net can then be trained to predict multiple labels, so different elements of content.

2.2 Transfer learning

Transfer learning can be a useful technique for deep learning historical document processing, particularly when there is a limited amount of labelled training data available. By using a pre-trained model as a starting point and fine-tuning it on a new task or dataset, it is possible to leverage the knowledge learned from the pre-trained model to improve the performance of the model on the new task.

For example, if a pre-trained model has already been trained on a large dataset of modern documents, it may be possible to use transfer learning to fine-tune the model on a smaller dataset of historical documents. This could allow the model to learn important features and

patterns specific to historical documents, improving its ability to classify, transcribe, or extract information from them.

Transfer learning can also be useful for segmentation in historical documents, as it allows the model to leverage the knowledge learned from modern images and objects to identify similar features in historical documents.

The VGG (Visual Geometry Group) network is a CNN architecture that was initially developed for image classification tasks. In the case of segmentation tasks, the VGG network can be used as a starting point for developing a CNN architecture that is specifically designed for segmentation.

For example, the VGG network could be modified and extended by adding additional convolutional and pooling layers, as well as skip connections, to better handle the complexity and spatial resolution of the segmentation task. The modified VGG network could then be trained on a large dataset of labelled images, where the boundaries between different objects or regions of interest have been manually annotated.

In the case of transfer learning, the VGG network can be used as a pre-trained model to initialize the weights of a U-Net architecture that is being trained for a different task.

Therefore, in state-of-the-art DL frameworks for historical documents, models have a U-Net network in which the first layers are from a pre-trained VGG network. The actual U-Net network can then be trained on a smaller dataset of historical documents.

3 Frameworks for historical document processing

Frameworks are platforms that provide a foundation for developing applications by providing generic functionality. These functionalities can be selected or edited to provide a more specific application.

In our case, we need a framework for Historical Document Processing. Its generic approach should allow to segment various elements of content and extract them from different documents. For example, a framework using deep learning can help to train convolutional neural networks (CNN) or recurrent neural networks (RNN) on a dataset of labelled historical documents. The operation using a trained network model could then be used to predict the regions of elements of content in new, unseen documents.

Frameworks using deep learning approaches can be very effective for solving the segmentation problem of historical documents, but they also require a large amount of labelled training data and may be computationally intensive.

Existing state-of-the-art training datasets for historical document processing have been created:

Dataset	Year	Handwritten/ printed	Competition	Layout	Text-Line	Table	Graphics
IMPACT [129]	2013	Both		x	x	x	x
ICFHR18 RASM2018 [34]	2018	Handwritten	x	x	x		x
ICDAR19 RASM2019	**2019**	**Handwritten**	**x**	**x**	**x**		**x**
HORAE [15]	**2019**	**Handwritten**		**x**	**x**		**x**
GERMANA [141]	2009	Handwritten		x	x		
RODRIGO [159]	2010	Handwritten		x	x		
ESPOSALLES [151]	**2013**	**Handwritten**	**x**	**x**	**x**		
BH2M [51]	2014	Handwritten		x	x		
FCR [143]	**2020**	**Handwritten**		**x**	**x**	**x**	
HisClima [150]	2021	Handwritten		x	x	x	
DIVA-HisDB [168]	2016	Handwritten	x	x	x		
Pinkas [93]	**2019**	**Handwritten**		**x**	**x**		

Figure 3.2: *Sample of state-of-the-art datasets for historical document processing*

For training a model, training data and configuration must be provided to the framework:

- Training data which can be a directory containing the images, a list of image filenames, or a path to a csv file.
- Evaluation data which is used for validation, under the same format as training data.
- A directory for model output.
- A class file coding each region to segment.
- Additional parameters for training, such as:
  - Data augmentation to scale, rotate or editing the images
  - Batch size
  - Make patches by cropping image in smaller pieces
  - Number of epochs to cycle trough data
  - GPU

After training, the inference of a model can be operated on a directory of input data, that returns a probability maps for each label

To choose the right framework, certain features should be taken into account.

It must explicitly handle historical documents. Indeed, frameworks designed for historical document layout analysis usually use additional features than others, in order to take into account broken characters, stains, poor paper quality, and so on.

No binarisation algorithm should be used by the framework as preprocessing. It has been revealed that binarisation algorithms were not efficient enough in previous versions of Agora, so they should avoided as a pre-processing requirement.

It must allow at least a segmentation at the line level. Agora must extract text lines in text blocks. As historical document analysis frameworks are usually designed for further OCR operations, they very often extract text lines.

It must allow decoration segmentation. In historical documents are all types of visual EOC that are extremely valuable for their amount of historical information.

It should be compatible with handwritten text. In the case of handwritten documents, one or more characters, words or even lines tend to touch each other and are treated as the same content items. Therefore, the extraction of handwritten content items is impossible because too many of them touch each other.

The framework should learn from annotated masks. Each training sample consists in an image of a document and its corresponding parts to be predicted. Additionally, a text file encoding the RGB values of the classes needs to be provided. In this case if we want the classes "background", "document" and "photograph" to be respectively classes 0, 1, and 2 we need to encode their color line-by-line:

0	255	0
255	0	0
0	0	255

The user expressed the need to have an ALTO file as output of Agora. So it would be even better if the framework as an output well structured.

The Deep-Agora project should be continued. Therefore, it would be better if the tools it uses were also pursued.

Good documentation of the tool would also help to shorten the developer's adaptation phase.

Because of these criteria, two good frameworks could be used: dhSegment and Kraken.

dhSegment

Even if dhSegment seems to be the most suitable framework, its documentation is limited, especially for the last versions, and there is no ALTO conversion routine for its results.

Kraken

Kraken offers great documentation and an ALTO conversion routine for its results. However, it is not designed to segment visual elements of content and it seems that no good substitute for binarisation as a pre-processing operation has been found.

4 Analysis and design

1 Analysis

1.1 Assumptions used

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 would 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.

1.2 Specifications

1.2.1 System

The Deep-Agora software takes images of the CESR as inputs and output ALTO files and vignettes of the elements of content extracted. To operate, it uses trained neural network models in operations. Scenarios can also be saved and restored. Neural network models must be trained outside of the software system: in the engineer's system. Only after new models have been trained on prepared training datasets, they can be deployed in the software system.

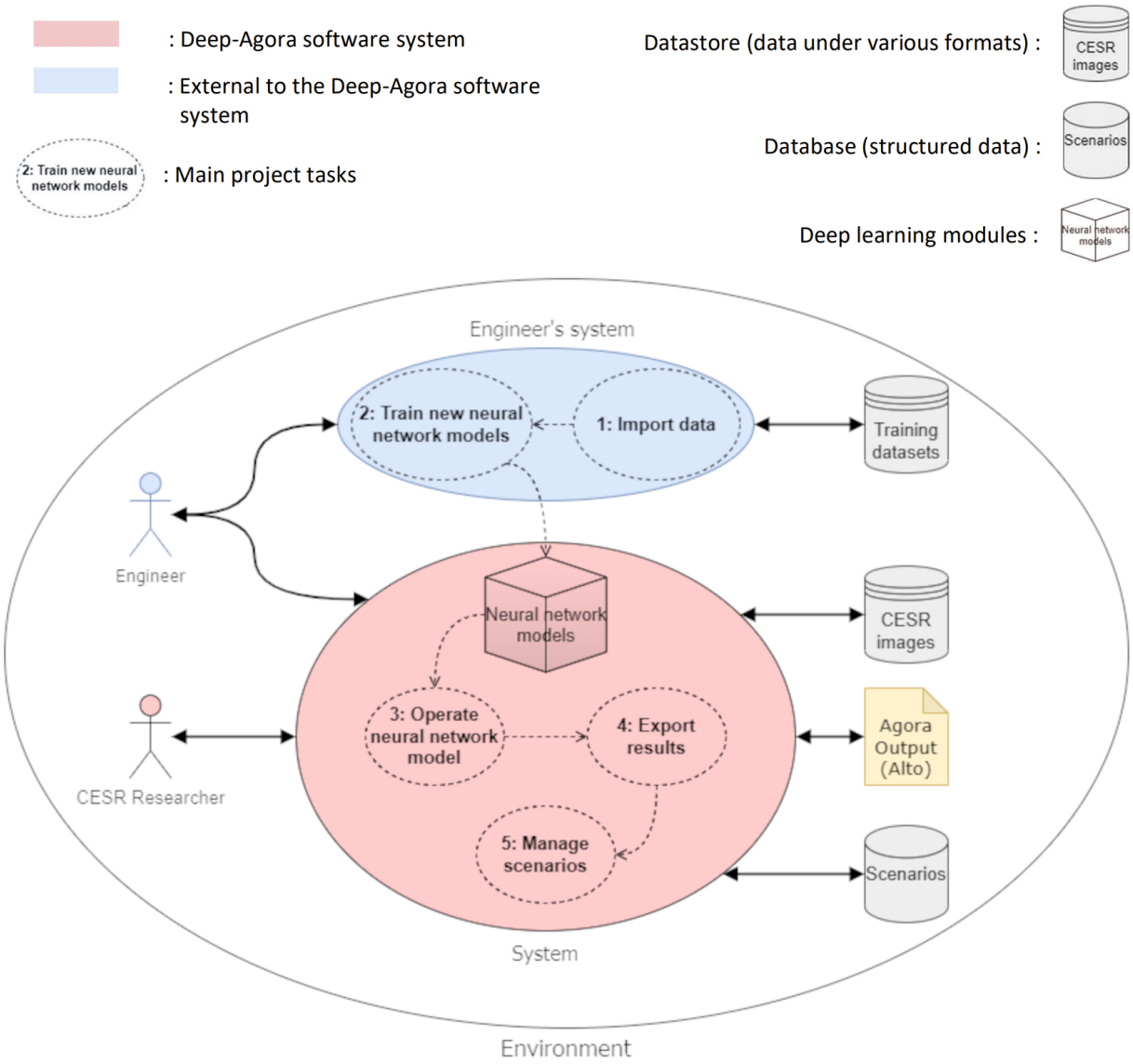


Figure 4.1: *Tasks in the context of the system*

Import data

Importing data consists of implementing a pipeline that acquires and prepares a dataset for each neural network model. This way, downloaded datasets from the internet under different formats

and containing different elements of content can be selected and merged to provide correctly structured ones.

Train new neural network models

Training new neural network models consists of implementing a generic framework for historical document processing to segment images into targeted elements of content. After acquiring the right prepared dataset, the model can be trained until it passes tests.

Operate neural network model

Operating neural network model consists of implementing a module to operate a trained neural network model as an operation in scenarios. For an image and an element of content to extract, the right model is restored to infer the image and returns a probability map for the element of content. It can then be thresholded to segment the image.

Export results

Exporting results consists of editing the outputs of scenarios to convert them into ALTO files and vignettes. When multiple elements of content are extracted from the same image, they are first structured hierarchically. Then, their name following the naming convention, their label and their coordinates can be altogether either written in an ALTO file or used to build vignettes extracted from the original image.

Manage scenarios

Managing scenarios consists of easing the end users' projects. Beyond importing images via manifests, the end users can manage their projects and refine the elements of content they want. Scenarios can be saved or loaded, and run on a directory of images.

1.2.2 Data

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.

2 Proposed modelling

2.1 Data pipeline

To train our models on appropriate data, we need to develop a series of processes that are used to extract, transform, and load data from one or more sources to a destination.

We first need to extract state-of-the-art training datasets by downloading them into the engineer's system. Once the data has been extracted, we need to transform it to make it more suitable for

the models. In our case, this involves converting data formats. Images must be converted to JPEG format and be of the same size. Labels of EOCs written in PAGE XML files must be used to build mask images. Mask images are RGB images in which each colour is associated with a different EOC. These colours are associated with EOCs in a class file.

After the different datasets have been transformed, they are ready to be selected and merged into a single one. This final dataset is then divided and saved in a training data folder and test, train and validation subfolders. For each subfolder, there is an image folder and a label folder.

This pipeline is implemented in the Import data module of the engineer's system, and this module interacts with the deep learning lab component.

2.2 Multilabel semantic segmentation

Regarding the class file in the previous subsection, it is important to note that in some cases, a pixel may belong to multiple classes or labels at the same time. In this case, the pixel is referred to as a "multilabel pixel." This case must be taken into account when developing a model capable of semantically segmenting several regions at once. For example, a pixel in the image might belong to both the "text line" class and the "text block" class.

To represent the labels for each pixel in an image using dense encoding, we create a fixed-length vector for each pixel, with each element of the vector corresponding to a particular label. The value of each element in the vector indicates the presence or absence of the corresponding label at that pixel. If a pixel belongs to multiple labels at the same time, the corresponding elements in the vector would be set to 1. For example, if a pixel contains both a line and a text block, the "text line" and "text block" elements of the vector would both be set to 1, while the "foot" element would be set to 0.

Therefore, in the class file, the RGB code of each colour is associated with an attribution code. The class file could then look like this:

255	0	0	1	0	0
0	255	0	0	1	0
0	0	255	0	0	1
0	255	255	0	1	1

2.3 Tree of EOCs

The output of the neural network model is equivalent to an attribution probability for each pixel of the image. Through the framework, we get a probability map of the image for each EOC. In order to output results for the end user, we need to turn the regions into a hierarchically structured tree of EOCs.

Based on the class file and the types of EOCs, we know which regions are included in the others. Then, the whole region of the text line is included in the region of the text block. We can summarise this logic in this scheme:

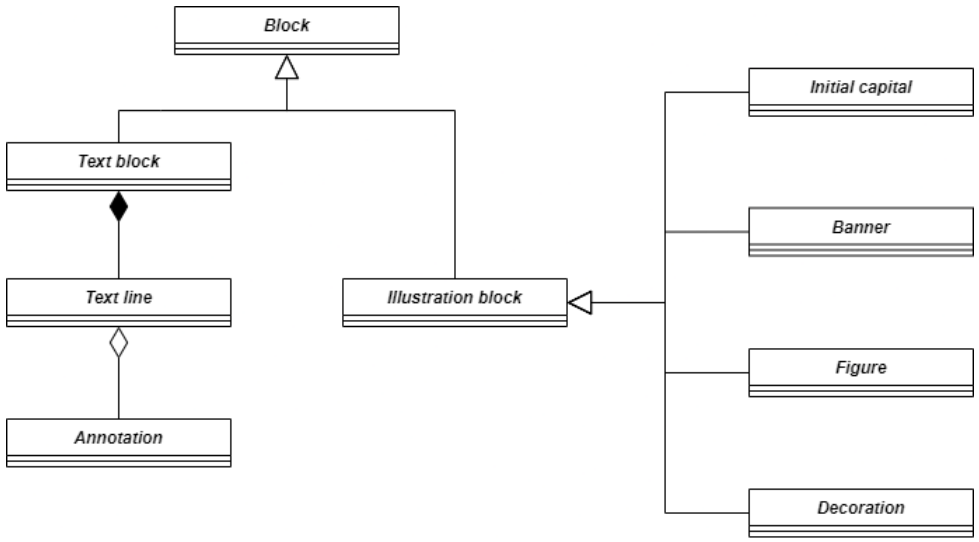


Figure 4.2: *Class diagram of the EOC hierarchy*

The objects of these classes can be structured following the exact same hierarchy. For example, in the case where a pixel both belongs to a text line and a text block, we know the text line is included in the text block. In the case of a banner, it is considered as belonging to an illustration block.

Finally, this hierarchically structured tree can be used to name each element appropriately and to build the ALTO file. An example of an ALTO file with the three types of EOCs "Block of text", "Text line" and "Banner" would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<alto xmlns="http://www.loc.gov/standards/alto/ns-v4#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.loc.gov/standards/alto/ns-v4#
     file:alto-4-3.xsd" SCHEMAVERSION=""
>
    <Description>
        <MeasurementUnit>pixel</MeasurementUnit>
        <sourceImageInformation>
            <fileName>[filename .e.g XXXXX.png]</fileName>
            <fileIdentifier fileIdentifierLocation="[path
             .e.g ../images]"/>
        </sourceImageInformation>
        <Processing ID="Agora"/>
    </Description>
    <Styles>
        <ParagraphStyle ID="[idString .e.g BLOCK]"/>
        <ParagraphStyle ID="[idString .e.g BANN]"/>
        <ParagraphStyle ID="[idString .e.g LINE]"/>
        <ParagraphStyle .../>
    </Styles>

    <Layout>
        <Page ID="agora.[uniqueString .e.g 0]" HEIGHT="[integer]"
         WIDTH="[integer]" PHYSICAL_IMG_NR="0">
            <TopMargin HEIGHT="[integer]" WIDTH="[integer]"
             HPOS="[integer]" VPOS="[integer]">
```

```
                ...
            </TopMargin>
            <LeftMargin ...>
                ...
            </LeftMargin>
            <RightMargin ...>
                ...
            </RightMargin>
            <BottomMargin ...>
                ...
            </BottomMargin>
            <PrintSpace ...>
                <ComposedBlock ID="agora.[uniqueString .e.g 0.0]"...>
                    <TextBlock ID="agora.[uniqueString .e.g 0.0.0]"
                     STYLEREFS="[idString]"
                          HEIGHT="[integer]" WIDTH="[integer]"
                          HPOS="[integer]" VPOS="[integer]">

                        <TextLine ID="agora.[uniqueString
                         .e.g 0.0.0.0]" .../>
                        ...
                    </TextBlock>
                    ...
                    <Illustration FILEID="agora.[uniqueString
                          .e.g 0.0.1.jpg]"
                     ID="agora.[uniqueString .e.g 0.0.1]" .../>
                    <TextBlock ID="agora.[uniqueString
                          .e.g 0.0.2]" ...>
                        ...
                    </TextBlock>
                    ...
                </ComposedBlock>
            </PrintSpace>
        </Page>
    </Layout>
</alto>
```

5 Implementation

We focused on developing deep learning models that will be employed in future software development. We implemented the *deep_learning_lab* package that allows a data scientist to prepare data, train deep neural networks and use them for inference on images. It resulted in the creation of deep neural networks, each specialising in the semantic segmentation of a specific element of content. Therefore, we only worked on single labels, not multilabels.

1 Tools and library used

We decided to use the **dhSegment-torch** framework. dhSegment-torch is an external Deep-Learning framework for historical document processing cloned from the GitHub repository dhSegment-torch. We chose it because of its generic approach that allows to segmenting various types of regions and extracting content from different types of documents. Note that dhSegment-torch is currently under development and is the PyTorch version of the no longer developed dhSegment Tensorflow version. However, only the Tensorflow version comes with documentation.

We used a Linux machine with a GPU as the processing time of training can be very long. The machine must have CUDA installed and the rest of the dependencies can be installed by Conda from the *dependencies* directory at the root of the project. The environment files have been edited to adapt to sm_86 CUDA architecture.

Besides dhSegment-torch, some very basic libraries are used such as *numpy* and *PIL* for image processing, *pandas* for CSV files processing and *xml* for XML files parsing. The project's use of external dependencies and sub-modules, such as conda environment files and setup files, highlights the importance of managing dependencies and environment configurations to ensure consistency and reproducibility across different machines and environments.

In addition, integration tests are managed by *pytest*. The use of tests for integration testing further enhances its reliability. The tests ensure that the code behaves as expected under different scenarios and edge cases, providing confidence in the code's correctness and reducing the likelihood of errors and bugs.

The use of these dependencies also enables the project to leverage existing, well-tested, and reliable libraries and frameworks, reducing development time and enabling developers to focus on higher-level tasks.

2 Implementation elements, technical choices

The project structure consists of multiple working directories, including the primary focus *deep_learning*. The *deep_learning* working directory provides a framework for data preparation, training, and inference for deep learning models. The *deep_learning_lab* package provides tools for data preparation, including patching and selecting labels, as well as training and inference tools. The dhSegment library provides the trainer and predictor classes for semantic segmentation. The project dependencies are contained in the *dependencies/* folder.

By organizing the project into working directories for data science, future software development, and dependencies, the project enables a clear separation of concerns and promotes modularity, making it easier to maintain, test, and update the code. Additionally, the separation of raw datasets into a dedicated folder and the use of sub-packages for data preparation and model training further enhances modularity and enables developers to use parts of the codebase in other projects.

The *deep_learning_lab* package included in the project follows software engineering principles by adopting a clear module structure and defining clear interfaces between different components. The package also makes use of logging and configuration files, improving the project's maintainability and scalability.

Future work can include implementing multi-label support for data preparation and further customization of the trainer and predictor classes.

The *segmentation.ipynb* Jupyter Notebook acts as an application and demonstration of the use of the *deep_learning_lab* package. First, it patches raw data sets into the **Training data preparation** section and demonstrates how to use the Orchestrator class. Next comes **Deep learning lab** which shows how to use the Trainer class by training a model on a single label, and then how to use the Predictor class by using inference from the same model, on the same label and on test data. Note that we did always used the same parameters for our models and did not tune them for better training:

- batch size is 4,
- number of epochs is 100,
- learning rate is 1e-4,
- decay rate for the learning rate is 0.9995,
- evaluate the model on the validation set every 5 epochs,
- 4 epochs to wait for improvement in validation loss before early stopping,
- repeat 4 times the dataset,
- 1e6 pixels in the images (size of the input layer)

Finally, the section **Tests** shows an example of how to use the inference results and allows the user to test the performance of the model.

3 Analysis of results, evaluation, quality

We trained four models, one for each of the following labels: TextLine, TextRegion, Word, ImageRegion.

Here are the most useful metrics to know for our problem of semantic segmentation:

- The intersection-over-union (IoU) and mean intersection-over-union (mIoU) measures the overlap between the predicted segmentation and the ground truth segmentation by computing the ratio of the intersection between the two regions to their union. It varies from 0 to 1, where 1 indicates a perfect overlap between the predicted and ground truth segmentations.

- mIoU, on the other hand, calculates the average IoU across all classes in the dataset. It provides a more comprehensive measure of the overall performance of the segmentation model.

3.1 TextLine

TextLine has the best trained model we have. Here is a good example of its performance, with the regions drawn on the image and the first 3 vignettes we extracted:

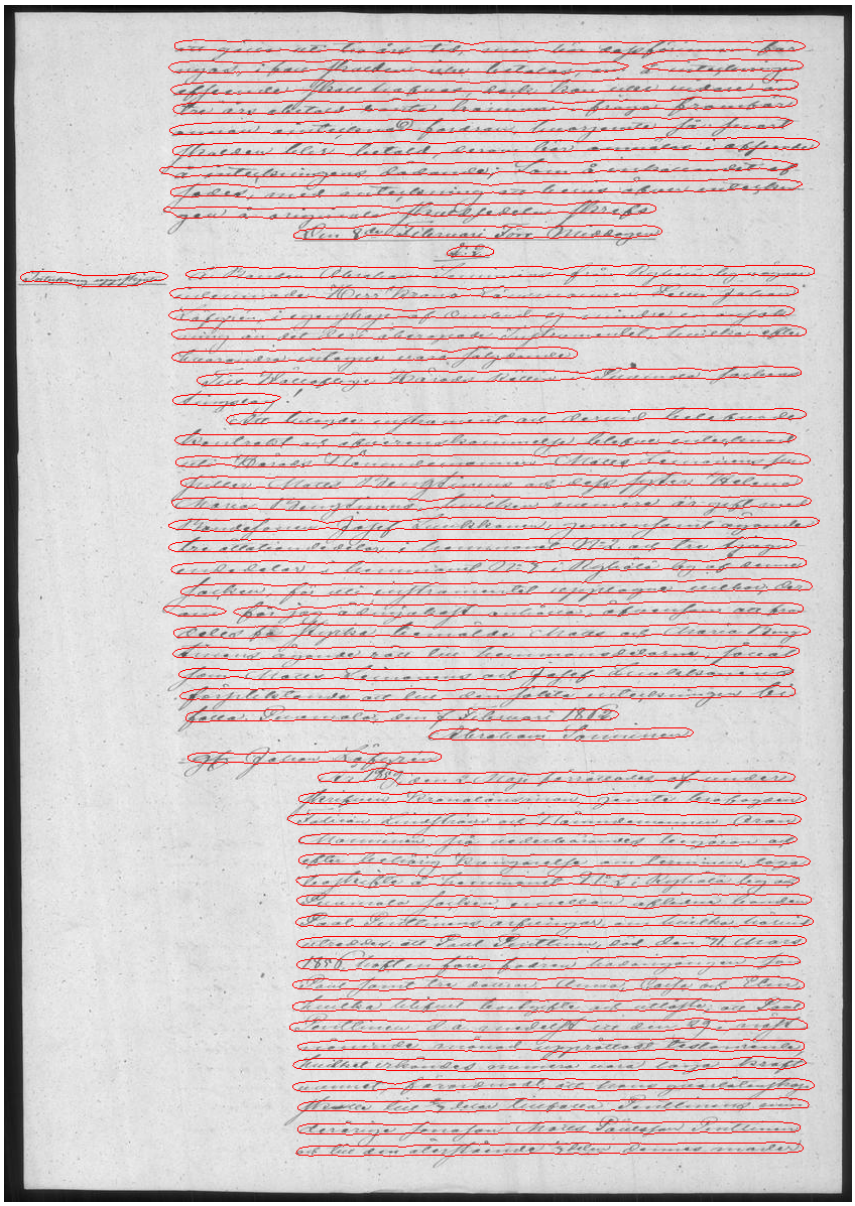


Figure 5.1: *Good TextLine predictions on an image*

Figure 5.2: *Predicted TextLine vignette 1*

Figure 5.3: *Predicted TextLine vignette 2*

Figure 5.4: *Predicted TextLine vignette 3*

And here is an example of bad performance:

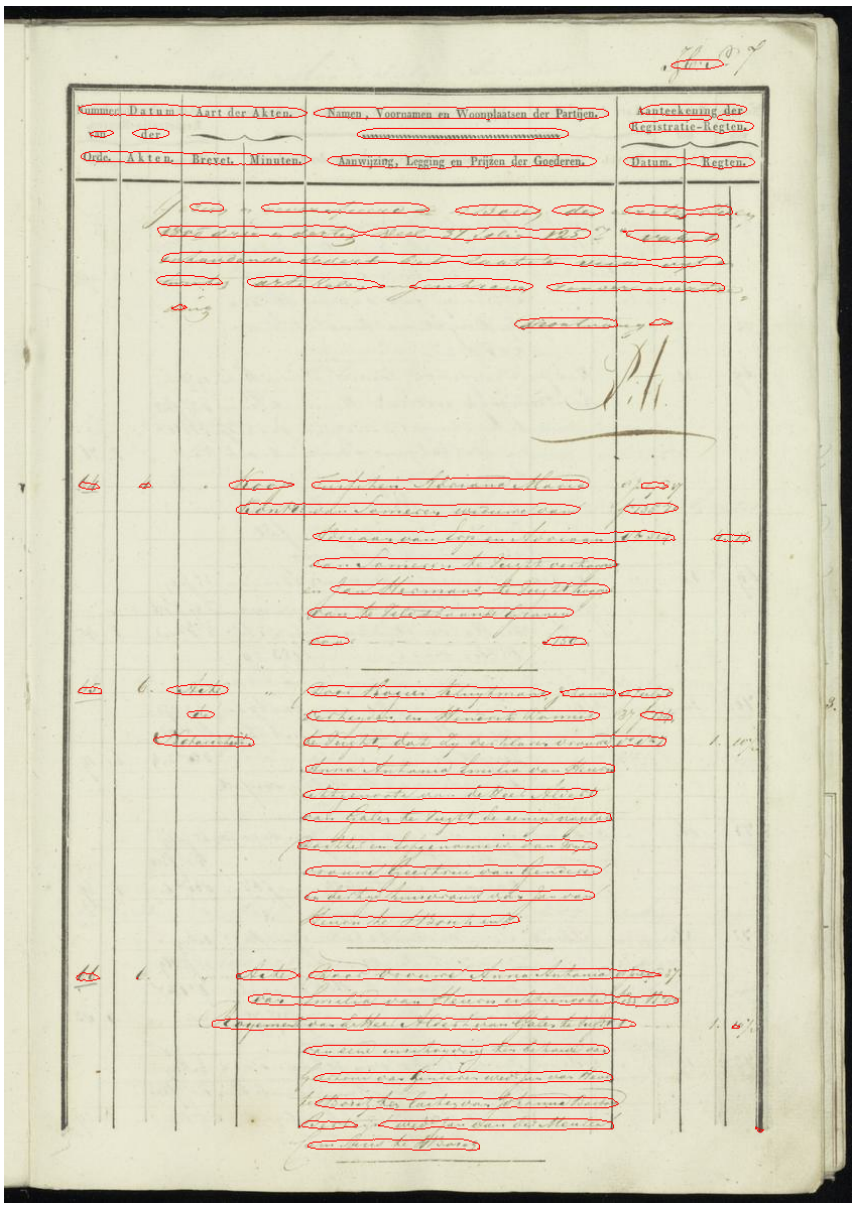


Figure 5.5: *Bad TextLine predictions on an image*

In order to get a better understanding of the quality of the model, here come the metrics we measured using Tensorboard.

We use this learning rate decay based on the exponential law:

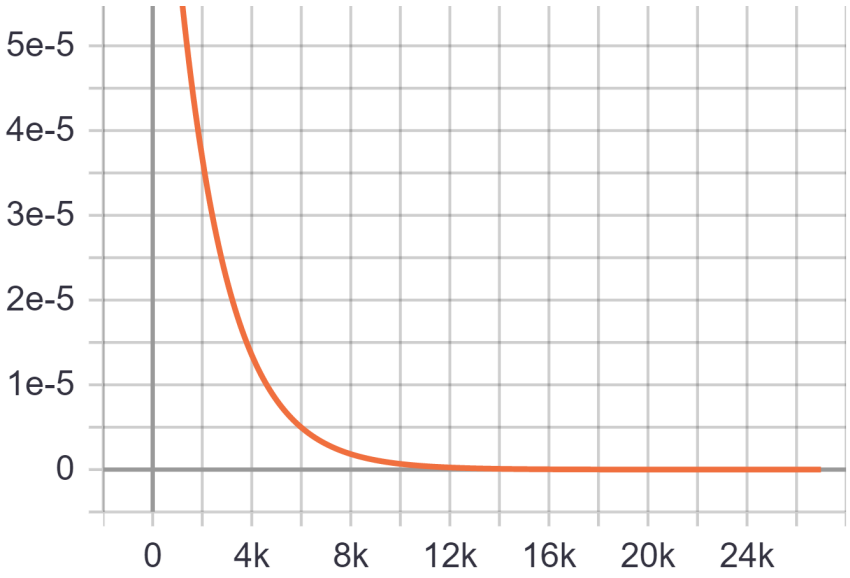


Figure 5.6: *Learning rate of TextLine model on training set*

We observe a fairly good loss curve on the training set:

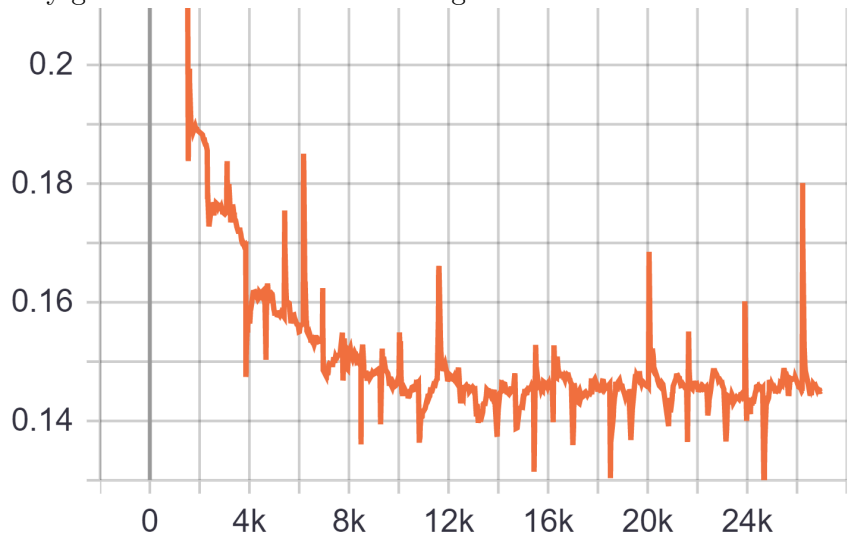


Figure 5.7: *Loss curve of TextLine model on training set*

We have the same metrics for the validation set:

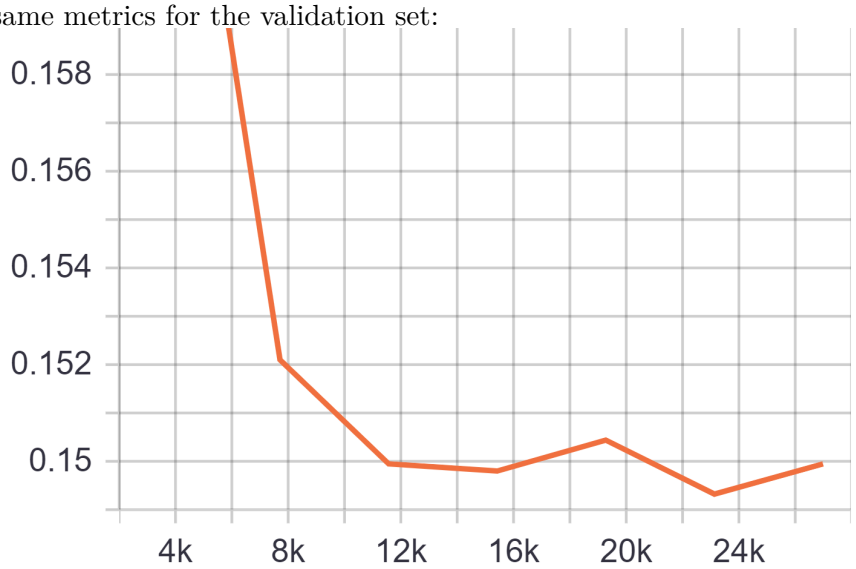


Figure 5.8: *Loss curve of TextLine model on validation set*

However, we get much more information about the quality of training by looking at the following metrics on the validation set:

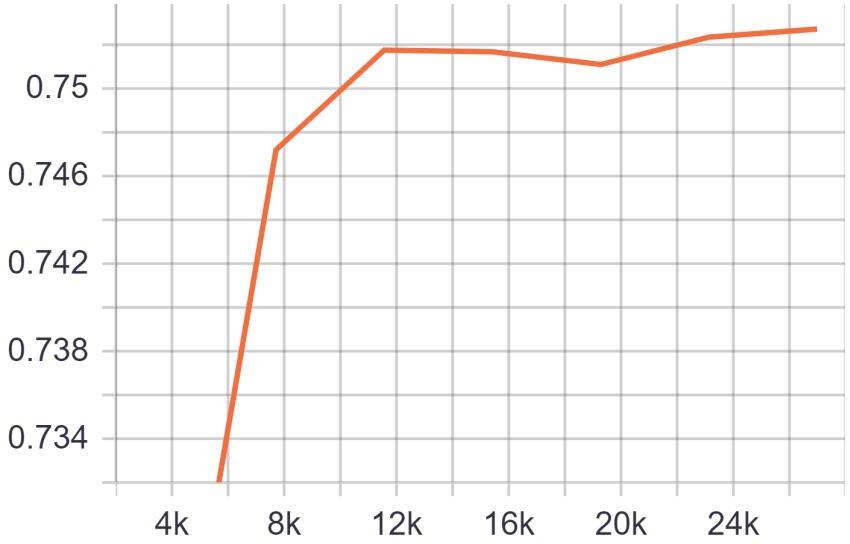


Figure 5.9: *IoU metric for TextLine of TextLine model on the validation set*

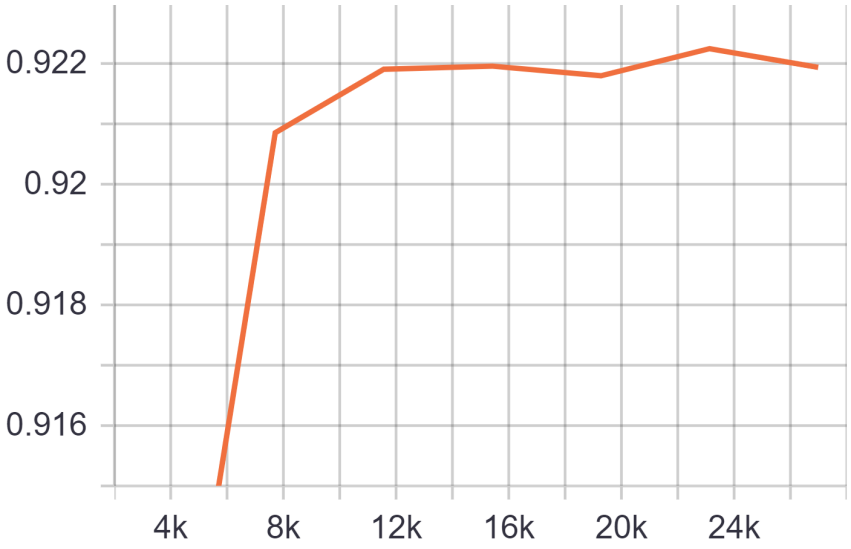


Figure 5.10: *IoU metric for Background of TextLine model on the validation set*

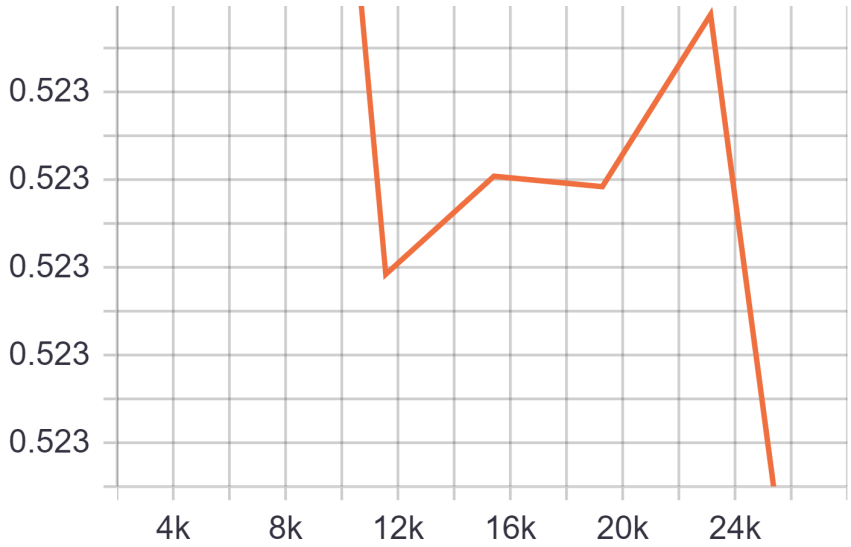


Figure 5.11: *mIoU metric of TextLine model on the validation set*

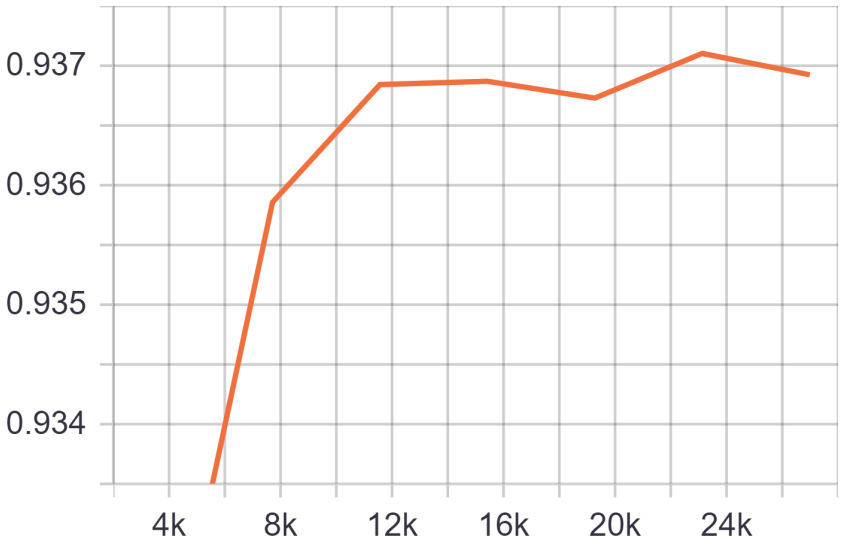


Figure 5.12: *Precision metric of TextLine model on the validation set*

3.2 TextRegion

The TextRegion model always finds a region of text, but it is generally not fine enough and tends to create unnecessary or too few regions. Here is a good example of its performance, with the regions drawn on the image and the first 3 vignettes we extracted:

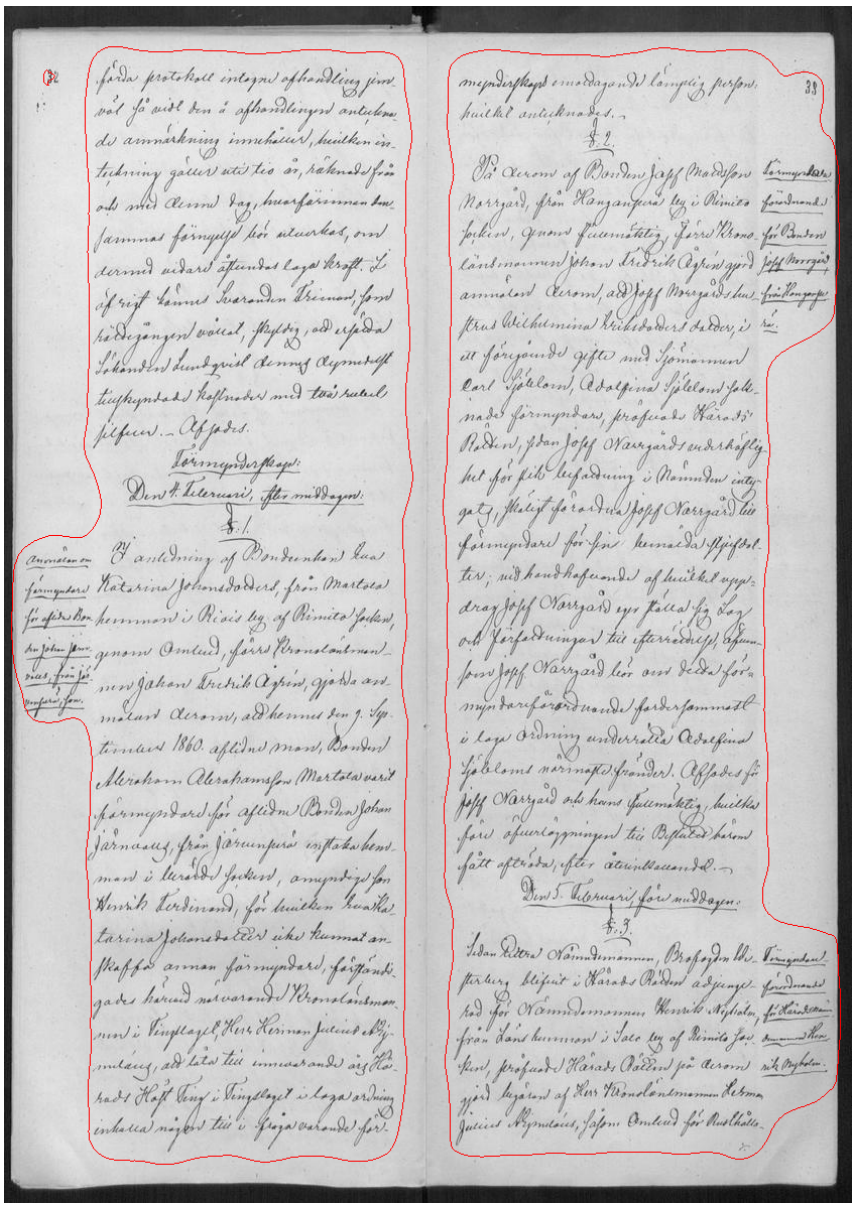


Figure 5.13: *Good TextRegion predictions on an image*

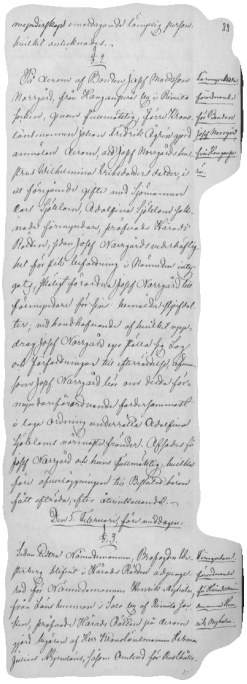


Figure 5.14: *Predicted TextRegion vignette 1*

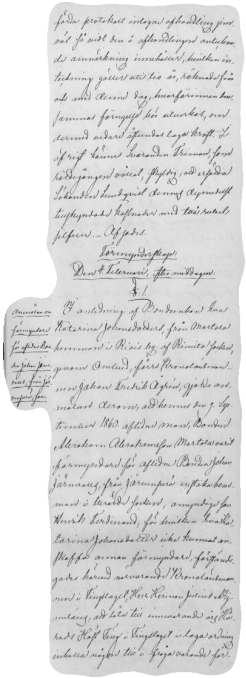


Figure 5.15: *Predicted TextRegion vignette 2*

And here is an example of bad performance:

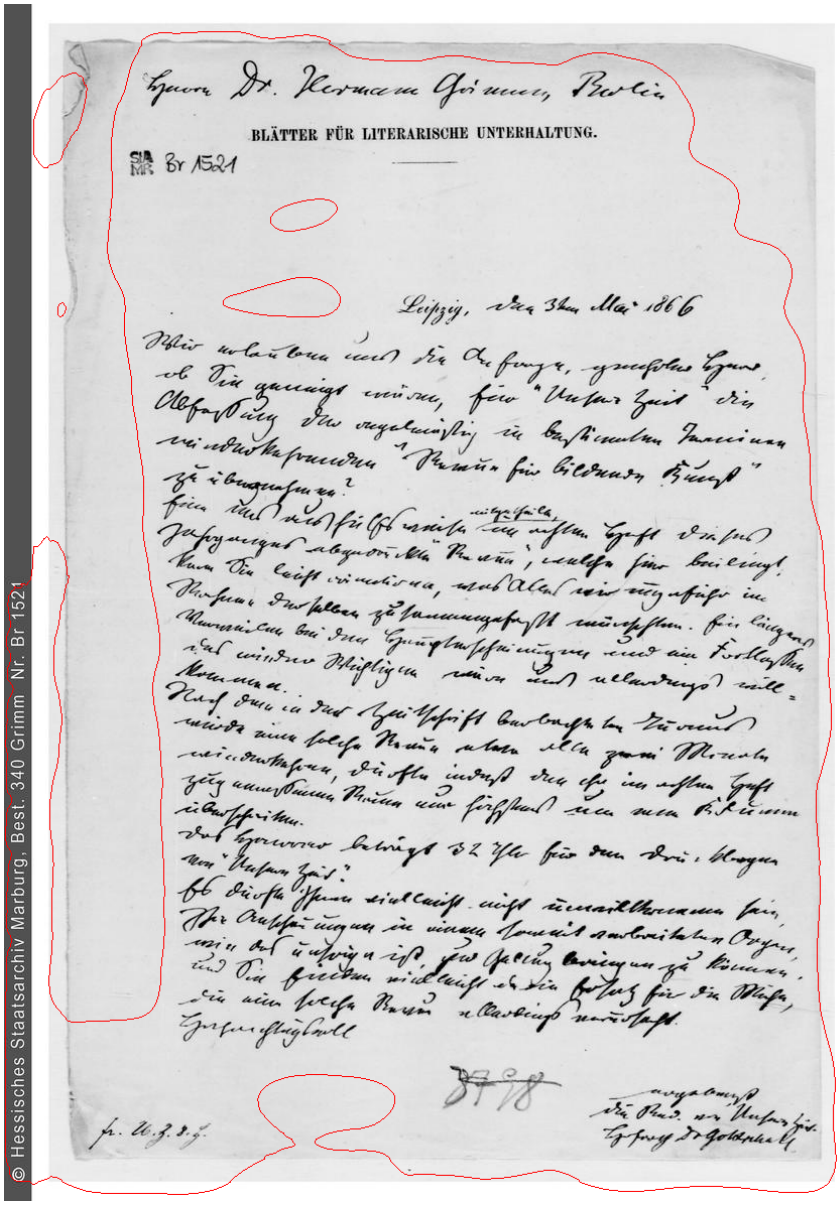


Figure 5.16: *Bad TextRegion predictions on an image*

With the same learning rate decay, here are its loss curve on training and validation datasets:

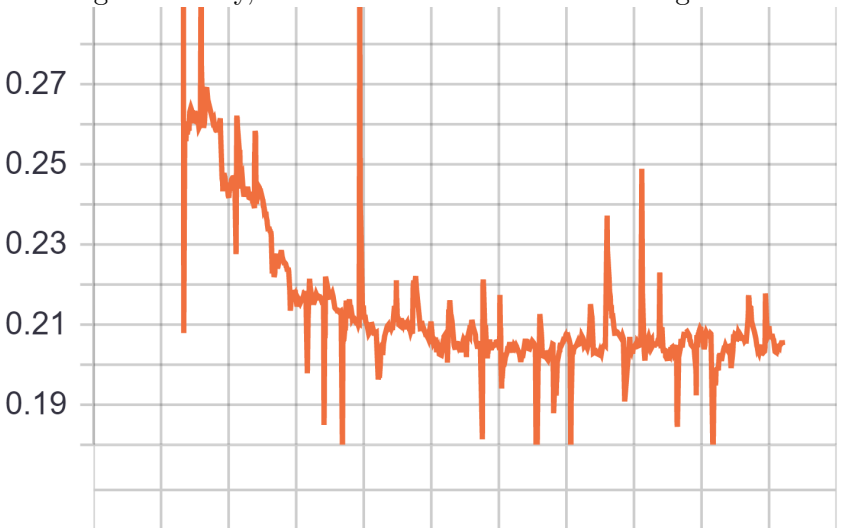


Figure 5.17: *Loss curve of TextRegion model on training set*

Figure 5.18: *Loss curve of TextRegion model on validation set*

Here are its other metrics on the validation set:

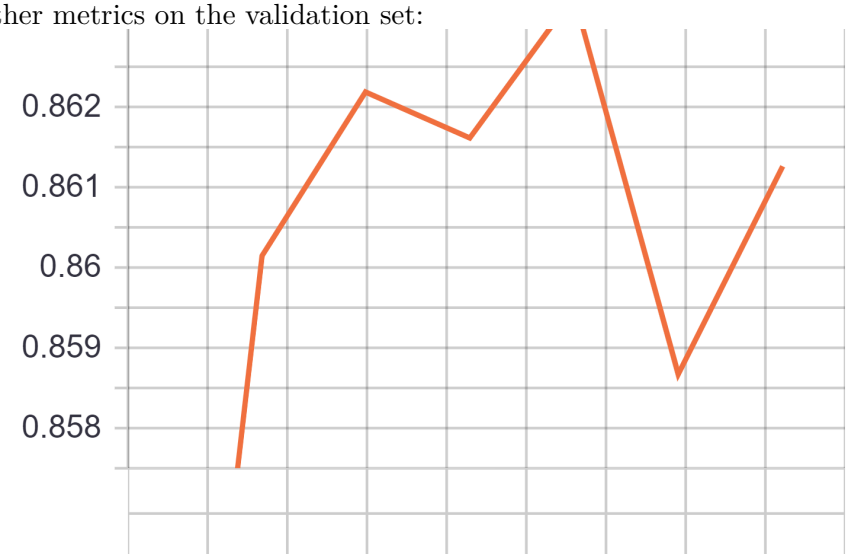


Figure 5.19: *IoU metric for TextRegion of TextRegion model on the validation set*

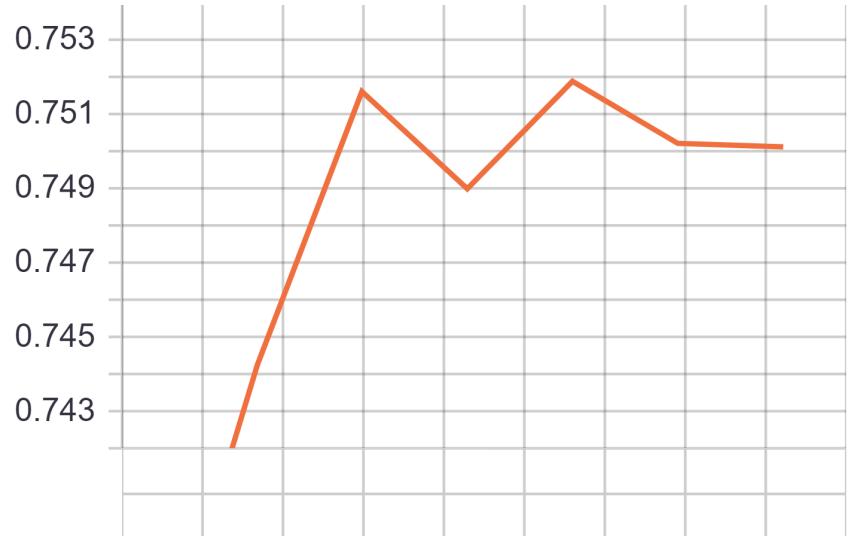


Figure 5.20: *IoU metric for Background of TextRegion model on the validation set*

Figure 5.21: *mIoU metric of TextRegion model on the validation set*

Figure 5.22: *Precision metric of TextRegion model on the validation set*

As we can see, all the metrics vary a lot. We did not have enough time to tune the models since we focused on the quality of the code. A solution could have been to increase the number of epochs or the number of validation steps without increase to tolerate (the early stopping). This way, the model could train more and maybe the loss on the validation set would start decreasing.

However, the result is unlikely to be much better, as the TextRegion masks in the training data actually often overlap. It comes from annotation files that often make poor quality masks. Here is an example:

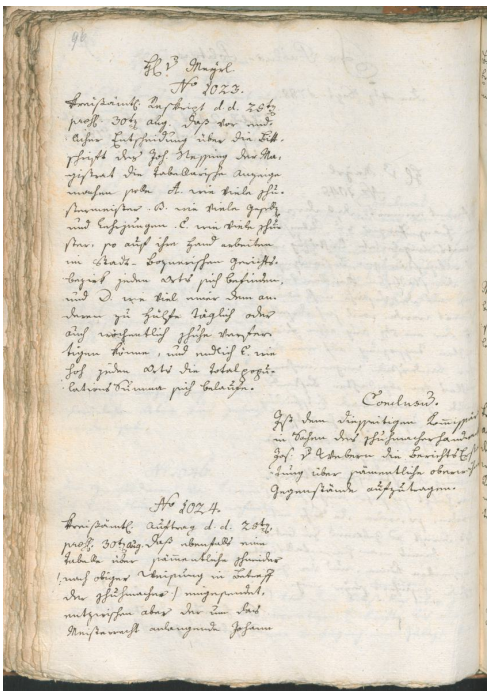


Figure 5.23: *Image of HS_107_192_object_132921 from "Baseline Competition - Complex Documents/Bohisto_Bozen_SetP" dataset*

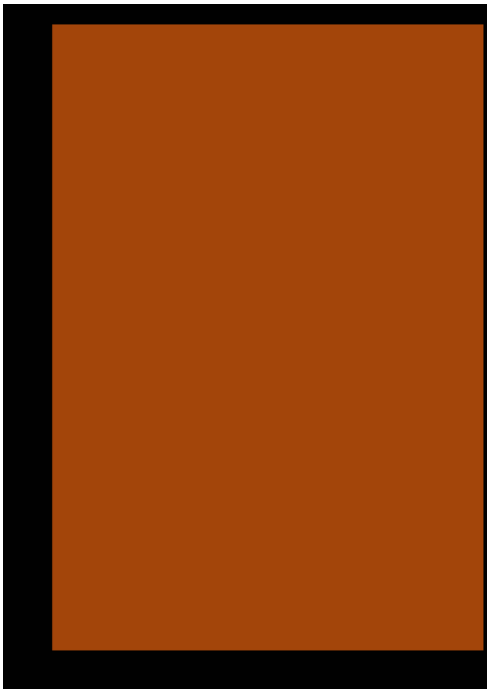


Figure 5.24: *Mask of HS_107_192_object_132921 from "Baseline Competition - Complex Documents/Bohisto_Bozen_SetP" dataset*

A solution could be to ignore the annotation files that only contain 1 label TextRegion when patching the datasets. They can easily be counted with the Orchestrator.validate method and verbose parameter greater or equal to 4.

3.3 ImageRegion and Word

Their models have very bad performances. They simply do not find any ImageRegion or Word in the test data.

The IoU metric for ImageRegion clearly shows that the ImageRegion model has not learned to recognise any of these elements:

Figure 5.25: *IoU metric for ImageRegion of ImageRegion model on the validation set*

This bad performance comes from a lack of data: we only have 3 labelled ImageRegion in all of our datasets

Concerning the Word model, it is not as obvious:

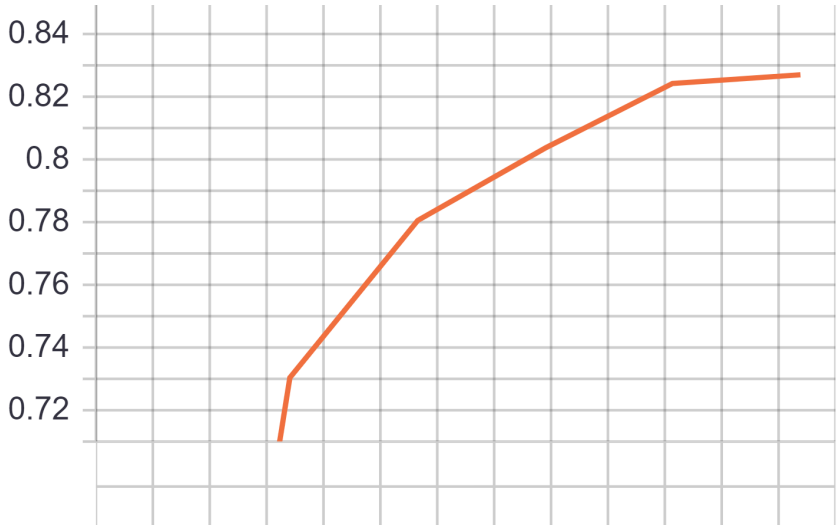


Figure 5.26: *IoU metric for Word of Word model on the validation set*

However, although we have about 40,000-word regions in all our datasets, they belong to only 135 images in total, of which 35 are from the pinkas dataset and 100 from the IEHHR dataset. Therefore, we have a very bad diversity of our data for the Word regions.

In addition, the annotations from the IEHHR dataset, which constitutes most of the Word training data, makes poor quality masks that overlap:

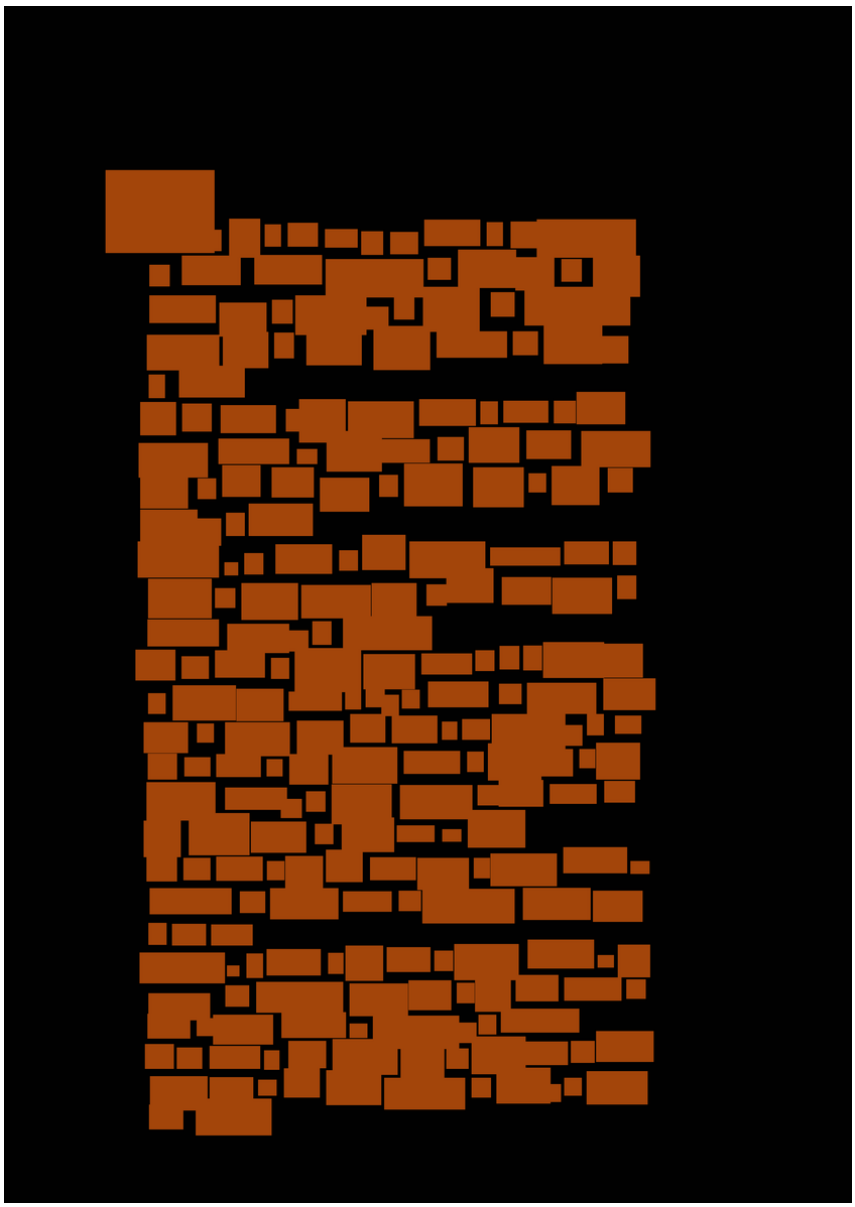


Figure 5.27: *Mask of Volum_069_Registres_0004 from "IEHHR-XMLpages" dataset*

6 Assessment and conclusion

1 Semester 9 review

This report took more time than excepted to be completed. Indeed, even after validation, it has been corrected and improved several times during holidays.

Tasks done

The following tasks have been completed:

1. Study existing system and state-of-the-art approaches
2. Specify I/O
3. Specify datasets
4. Set up planning
5. Design diagrams
6. Write specification document
7. Review and validate specification deliverables

Tasks in progress

1. Set up the environment

This task is particularly time-consuming if I want to configure my computer for using my GPU. I have an NVIDIA GeForce RTX 3050 Laptop GPU, and I am experiencing trouble using it for development purposes. If it persists, I will contact my supervisor and try to find a solution together. The rest of the environment is operational.

Tasks to do

1. Finish setting up the environment
2. Prepare dataset

As soon as the environment is ready, the data pipeline will have to be started in order to soon try a pre-trained model of dhSegment.

2 Semester 10 review

The environment of the project has been hard to configure due to the incompatibility between the CUDA version of our GPU server and the requirements of the dgSegment-torch library. In

order to avoid this complication for the next users of the project, new environment files, solid documentation and an installation guide have been created.

We estimate the delay due to the environment, including research and migration, to be two weeks. The progress of the project has been slowed down by two weeks due to illness. The last two sprints, each lasting two weeks, were already planned as optional.

We have been able to develop a validated model of text line segmentation and analyse its performance. However, we did not tune it. Instead, we really focused on creating new packages for the project and created a modular project structure with a deep learning package. As specified, we focused on the quality of the code and its documentation so that it is easily maintainable and scalable for future developers.

Finally, we offered the first functionality for output export by saving all the vignettes in a directory. However, we did not go further in the development and therefore did not create functionality for building ALTO output files or HMIs (which was specified in the optional sprints).

3 Quality assessment

In terms of the quality of the code and the documentation, we are very satisfied with our achievements and we believe that the next developers working on this project will have all the elements to do a great job.

We also believe that we had good project management and managed to overcome the challenges we faced in our project.

However, we would have liked to have acquired and patched more diverse data, tuned our models and obtained better results for most of them, as only one has really satisfactory results. Many datasets were not available, and we could have realised it earlier if we did not work as much on the specification part.

4 Self-critical review

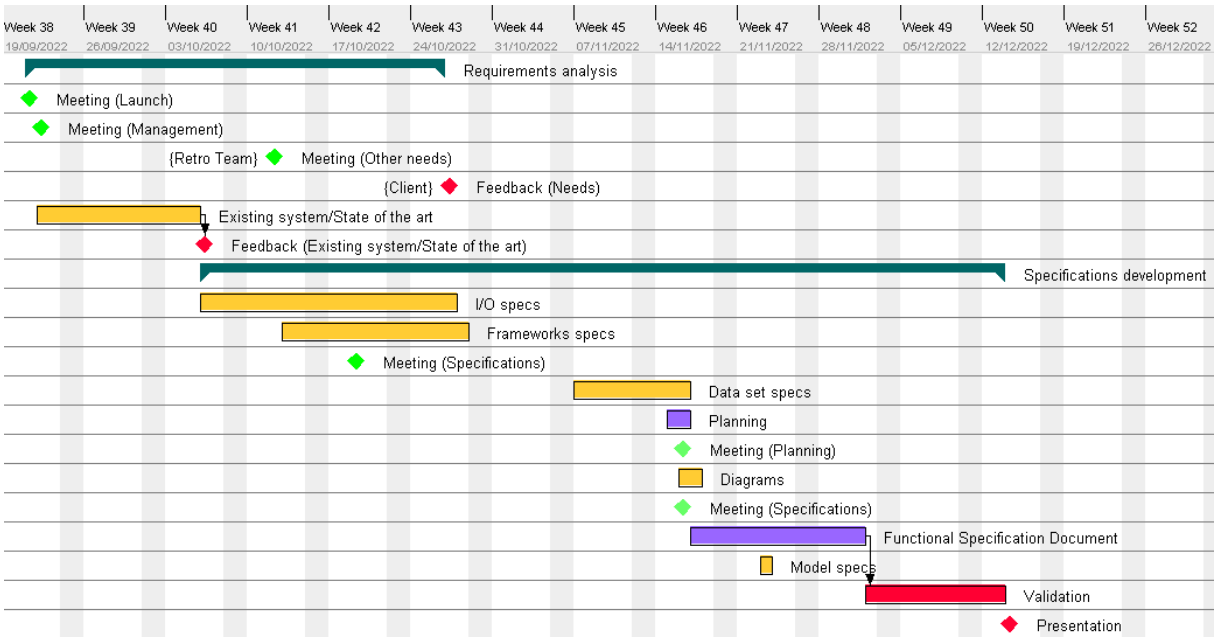
We really enjoyed doing this project and thought it has been very formative. It was a personal goal to work for a long time on a large deep learning project and this R&D project made it possible. We even insisted on getting this project from our supervisor and we are very grateful to have been able to work on it. It reinforces our idea that we want to work as an engineer in the field of machine learning.

Appendices

A Planning, project management

1 Specification phase

1.1 Evolution of the project



?figurename? A.1: *Gantt Chart for planning of the specification phase*

1.2 Job description

Task 1: Study existing system and state-of-the-art approaches

- Start date: 22/09/2022
- End date: 19/01/2022
- Duration: 14 days

- Description: Study and present Agora's problems and what state-of-the-art generic frameworks enable historical document processing.

Task 2: Specify I/O

- Start date: 06/10/2022
- End date: 27/10/2022
- Duration: 22 days
- Description: Identify and interview Deep-Agora future end-users to gather information about the inputs and outputs of the software.

Task 3: Specify datasets

- Start date: 07/11/2022
- End date: 16/11/2022
- Duration: 10 days
- Description: Specify and identify the best state-of-the-art datasets for historical document processing.

Task 4: Set up planning

- Start date: 16/11/2022
- End date: 16/11/2022
- Duration: 1 day
- Description: Set up and review the initial planning of the project.

Task 5: Design diagrams

- Start date: 16/11/2022
- End date: 17/11/2022
- Duration: 2 days
- Description: Model via diagrams the system and the components of the project.

Task 6: Write Specification Document

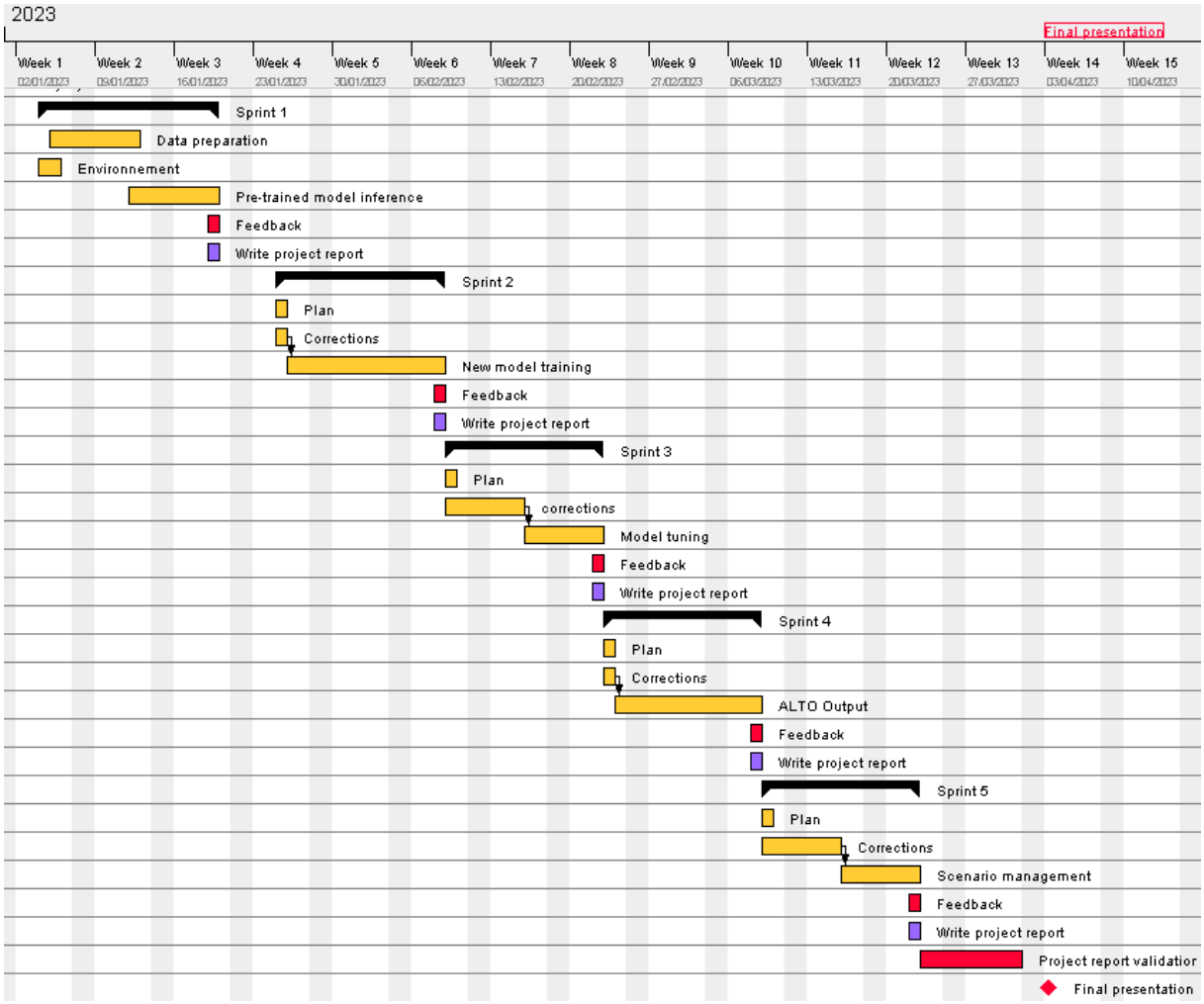
- Start date: 17/11/2022
- End date: 01/12/2022
- Duration: 15 days
- Description: Design and write the specifications of the project with the completion of each part to put in the final report. The completeness of the document should allow simplifying the writing of the final report.

Task 7: Review and validate

- Start date: 02/12/2022
- End date: 13/12/2022
- Duration: 12 days
- Description: Review the specifications with the product owner, present them, complete the specification document and finish the final report.

2 Implementation phase

2.1 Evolution of the project



?figurename? A.2: *Initial Gantt Chart for Agile planning of the implementation phase*

2.2 Job description

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

Minor release 1: Prototype DL module

- Start date: 04/01/2023
- End date: 19/01/2023
- Duration: 15 days
- Description: 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.

Minor release 2: New DL module for other EOCs

- Start date: 25/01/2023

- End date: 08/02/2023
- Duration: 14 days
- Description: 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.

Minor release 3: Evaluation and tuning

- Start date: 09/02/2023
- End date: 22/02/2023
- Duration: 13 days
- Description: After some corrections if necessary, the third minor release should allow the previous models to be evaluated and tuned for better results.

Minor release 4: Output export

- Start date: 23/02/2023
- End date: 08/03/2023
- Duration: 13 days
- Description: A secondary fourth minor release should offer a solution to export the outputs of the previously trained models to vignettes and ALTO files.

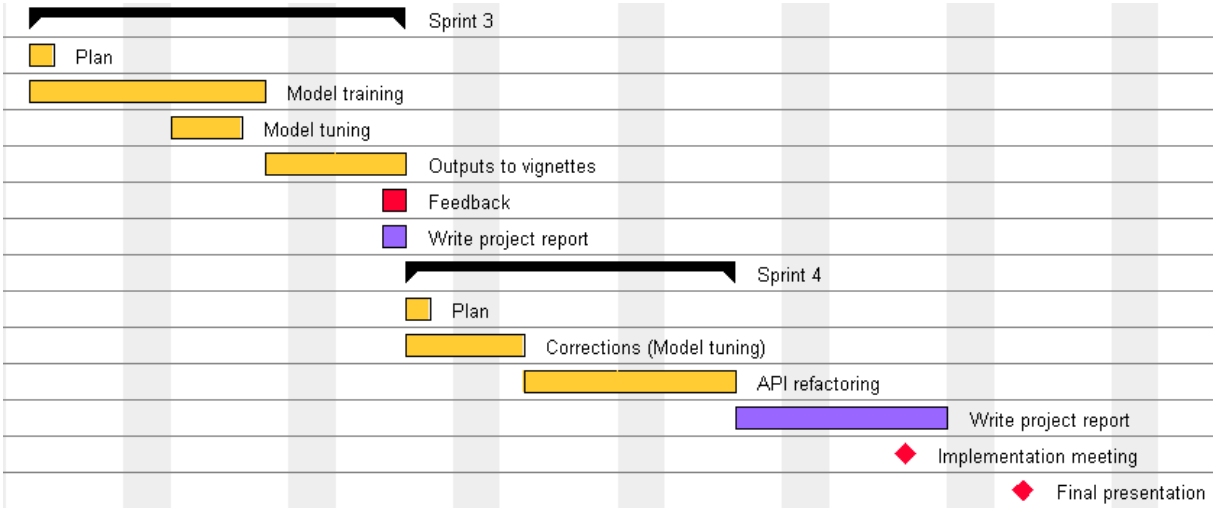
Major release 1: Interface for end-user

- Start date: 09/03/2023
- End date: 22/03/2023
- Duration: 13 days
- Description: 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. It should use all the modules developed in the previous sprints.

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

Here are the final plannings of the project:

?figurename? A.3: *Final Gantt Chart for Agile planning of the implementation phase (part 1)*



?figurename? A.4: *Final Gantt Chart for Agile planning of the implementation phase (part 2)*

B Description of the interfaces

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.

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 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.

C Specification

1 Functional specifications

1.1 Definition of module 1: Import training data

Presentation:

- Name: Training data import module
- Summary: Implement a pipeline to prepare a training dataset for neural network model.
- Priority: Primary
- Interacting components:
  - Deep learning lab component
  - File system
  - Datasets from the file system
  - Engineer's system

Description:

- 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.

Details:

- Prioritized features list:

1. Acquire datasets from the file system
2. Distribute them in different pipelines according to the EOCs targeted by the neural network models
3. Create a training data folder for each pipeline 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

1.2 Definition of module 2: Deep learning lab

Presentation:

- 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
- Interacting components:
  - Training data import module
  - Deep learning lab component
  - File system
  - Engineer's system

Description:

- 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.

Details:

- 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 and/or pass to another model

1.3 Definition of module 3: Import images

Presentation:

- Name: Import module
- Summary: Import images from a manifest.
- Priority: Optional
- Interacting components:
  - File system
  - Manifests
  - Deep-Agora system

Description:

- 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.

Details:

- 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

1.4 Definition of module 4: EOCs to tree

Presentation:

- 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
- Interacting components:
  - Deep learning extractors
  - Scenarios
  - Deep-Agora system

Description:

- 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.

Details:

- 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:
  –

1.5 Definition of module 5: Export results

Presentation:

- Name: Export module
- Summary: Export vignettes of elements of content and export their location and coordinates into ALTO files for each page.
- Priority: Optional
- Interacting components:
  – Scenarios
  – File system
  – Deep-Agora system

Description:

- 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.

Details:

- 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:
  –

1.6 Definition of module 6: End user interface

Presentation:

- Name: Manage scenarios
- Summary: End users can manage their project and refine the elements of content they want.
- Priority: Optional
- Interacting components:
  - Project management component
  - File system
  - Deep-Agora system

Description:

- Inputs: HMI instructions
- Preconditions: The other modules have been implemented.
- Outputs: -
- Postconditions: -

Details:

- 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:
  -

2 Non-functional specifications

2.1 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.

2.2 Functional and operational constraints

2.2.1 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.

2.2.2 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.

2.2.3 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.

2.2.4 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.

2.2.5 Security

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

2.2.6 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.

2.3 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.

D Developer's Workbook

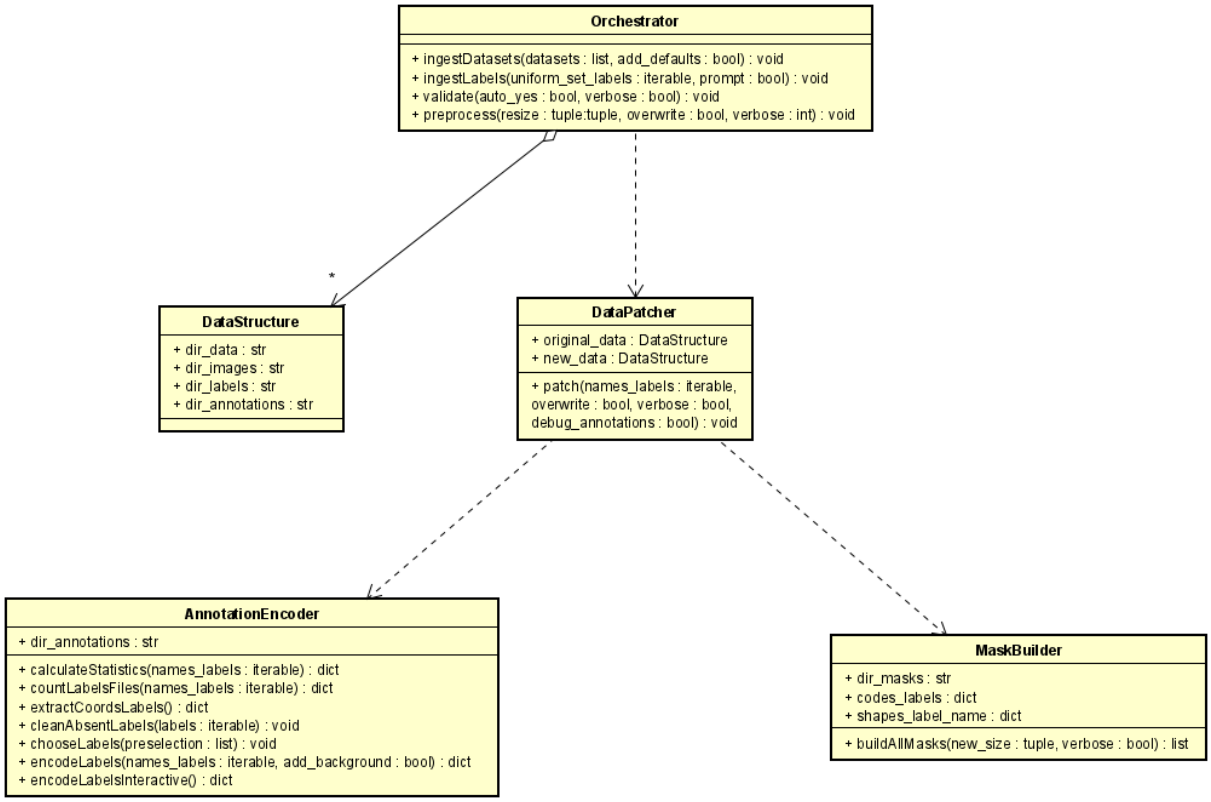
1 Introduction

The package-based structure of the *deep_learning_lab* module promotes code reusability and maintainability. The use of external dependencies, such as the dhSegment-torch framework, allows for faster development and reduces the amount of custom code that needs to be written. Overall, these practices contribute to a more streamlined and scalable project development process.

2 Architectural diagrams and UML

The *deep_learning_lab.data_preparation.patch* module defines the classes DataStructure, DataPatcher, AnnotationEncoder and MaskBuilder:

- Orchestrator is responsible for orchestrating the data preparation and patching process for a given set of datasets.
- DataStructure is a description of a dataset in the file system.
- DataPatcher allows to transform a dataset into a dhSegment framework compatible one containing the masks of the specified labels.
- AnnotationEncoder inspects a directory of annotations files to select, extract, and encode labels and their features.
- MaskBuilder creates masks for each labelled region in the image, given the annotations.



?figurename? D.1: *Class diagram of the deep_learning_lab.data_preparation subpackage*

3 Detailed descriptions of data used

For training our deep neural networks, we used the 6 following datasets (some bring together several data sets):

- FCR
- ESPOSALLES
- ScriptNet
- REID2019
- Pinkas
- ImageCLEF 2016

They constitute around 1,700 pairs (images, annotations). Here are the results of our analysis using the *Orchestrator.validate* method:

Dataset	TextLine	TextRegion	Word	ImageRegion
FCR_500/data	32177	1701	-	-
ABP_FirstTestCollection	961	226	-	-
Bohisto_Bozen_SetP	815	152	-	-
EPFL_VTM_FirstTestCollection	252	38	-	-
HUB_Berlin_Humboldt	693	81	-	-
NAF_FirstTestCollection	930	164	-	-
StAM_Marburg_Grimm_SetP	857	214	-	-
UCL_Bentham_SetP	1024	191	-	-
unibas_e-Manuscripta	848	96	-	-
ABP_FirstTestCollection	4230	30	-	-
Bohisto_Bozen_SetP	910	26	-	-
BHIC_Akten	2339	30	-	-
EPFL_VTM_FirstTestCollection	2790	28	-	-
HUB_Berlin_Humboldt	885	28	-	-
NAF_FirstTestCollection	6147	29	-	-
StAM_Marburg_Grimm_SetP	1064	30	-	-
UCL_Bentham_SetP	2294	31	-	-
unibas_e-Manuscripta	1081	20	-	-
pinkas_dataset	1013	175	13744	-
IEHHR-XMLpages	3070	968	31501	-
ImageCLEF 2016 pages_train_jpg	9645	765	-	-
REID2019	-	454	-	3

4 Detailed descriptions of classes, modules, achievements

4.1 Data Preparation

The raw datasets are expected to be placed in the *raw_datasets/* folder, located in the deep_-learning working directory. These datasets typically contain images and XML files containing their annotations. These annotation files cannot be used directly to train the model because they must be converted to masks. The *deep_learning_lab.data_preparation* package allows developers to select and use labels from their raw datasets to build masks.

To patch a dataset, the developer must specify the main directory, the image directory, and the annotation directory. The *deep_learning_lab.data_preparation.orchestration* module provides default datasets that have been implemented inside the source code. Additional datasets can either be added to the defaults datasets in the source code of the module or via the *Orchestrator.ingestDatasets* method.

The *deep_learning_lab.data_preparation.patch* module, which runs in the backend of the Orchestrator class, has not been validated for multi-labels since it misses the processing of one-hot encoding.

To extract annotations from data sources in a format other than PAGE, the DataPatcher class can be very easily extended to support other extraction methods by using the method *DataPatcher._-*

securelyExtractUsing(self, extraction_fun). Indeed, this method uses the extraction_fun method given as a parameter to parse XML files. Here is a simple example of how to use it:

```
def countLabelsFiles(self, names_labels) -> dict:
    """Counts the number of occurrences of specified label names in
    the given XML page.

    Args:
        names_labels (iterable): An iterable of label names to
                                                search for.
    Returns:
        The number of occurrences of the specified label names in
        the XML page.

    """
    number_labels_name, _ = self._securelyExtractUsing(
        lambda page: xml.countTags(
            page,
            names_labels,
            self.namespaces_label
        )
    )
    return number_labels_name
```

```
def extractCoordsLabels(self) -> dict:
    """Extracts the coordinates of specified label names from the
    XML page.

    Returns:
        A dictionary containing the size of the page and the
        coordinates of the specified labels.
        A list of any anomalies encountered during extraction.

    """
    shapes_label_name, anomalies = self._securelyExtractUsing(
        lambda page: {
            'size': self._getPageSize(page),
            'coords': xml.extractAttributesTag(
                'Coords',
                'points',
                page,
                self.codes_labels.keys(),
                self.namespaces_label
            )
        }
    )
    return shapes_label_name, anomalies
```

As you can see, an operation module called XMLParser used as *xml* has been created to easily parse annotation files.

4.2 Training

Before using the trainer provided by the dhSegment library, the developer must specify if they want to use a GPU and which one. The *deep_learning_lab.gpu_setup* module allows the selection of a GPU/CPU in the backend.

The Trainer class from the *deep_learning_lab.model* module can then be instantiated with a specified set of labels to segment and the dataset to use. The trainer can be configured with many parameters relating to the split of the validation and test sets or to the training of the model itself.

By default, the dataset to be used is *training_data* (results/'specified_labels'/training_data). The model and tensorboard directories should be in the same location. The model directory contains the best serialized models, and the tensorboard directory contains the logs of the metrics acquired during training.

4.3 Inference

The *deep_learning_lab.gpu_setup* module is also used for selection of a GPU/CPU in the backend.

The Predictor class from the *deep_learning_lab.model* module can be instantiated with a specified set of labels to segment. The class can convert the input images from a CSV file into a folder in order to reuse the split test data from the training phase (it is only designed for developers). It can draw regions around the elements of content in an image and cut out vignettes from it. The output directory contains the vignettes and the output of the *'Predictor.start'* method returns additional data such as the original image on which the regions are drawn.

By default, the input data is *inference_data*, and the output directory is not specified.

E Installation document

1 Requirements

To work in the deep_learning directory, it is recommended to have a Linux or WSL machine with a GPU, as the processing time can be very long. You also need to have Conda installed on your machine.

Check if you have a GPU and CUDA installed via the NVIDIA System Management Interface (NVIDIA-SMI) driver by entering in your terminal:

```
nvidia-smi
```

2 Installation

The deep_learning_lab package uses the sub-module and framework dhSegment-torch. First, go to dependencies/ and clone the sub-module(s) as follows:

```
cd dependencies/
git submodule update --init --recursive
```

The environment files environment.yml and setup.py of the dhSegment-torch framework have been edited to adapt to the sm_86 CUDA architecture of our machine. You can match your GPU name to your CUDA architecture on this website. To apply such changes, do as follows:

```
cp environment.yml dhSegment-torch/
cp setup.py dhSegment-torch/
```

To install the package of the sub-module, go to dhSegment-torch/:

```
cd dhSegment-torch
```

dhSegment will not work properly if the dependencies are not respected. In particular, inaccurate dependencies may result in an inability to converge, even if no error is displayed. Therefore, we highly recommend to create a dedicated environment as follows:

```
conda env create --name dhs --file environment.yml
source activate dhs
python setup.py install
```

F Tests

1 Unit testing

1.1 Module 1: Import training data

IDENTIFICATION OF COMPONENT
Request the right EOCs
DESCRIPTION OF THE TEST
Unknown EOCs are requested
EXPECTED RESULTS
Raise an unknown parameter exception
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Datasets are split correctly
DESCRIPTION OF THE TEST
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
EXPECTED RESULTS
X+Y+Z=100 % of the pairs from the datasets
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Image formats are respected
DESCRIPTION OF THE TEST
For each image in the image subfolder
EXPECTED RESULTS
Each has the JPEG format
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Masks have been built
DESCRIPTION OF THE TEST
For each image in the image subfolder
EXPECTED RESULTS
Each has a PNG mask with the same name in the label subfolder
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Masks were built correctly
DESCRIPTION OF THE TEST
For each PNG mask in the label subfolder
EXPECTED RESULTS
Each uses the colours from the class file
OBTAINED RESULTS
-

1.2 Module 2: Deep learning lab

IDENTIFICATION OF COMPONENT
Parameters are stored correctly
DESCRIPTION OF THE TEST
Training parameters configuration
EXPECTED RESULTS
Is stored in a JSON file
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Masks use the class file definition
DESCRIPTION OF THE TEST
Predicted masks
EXPECTED RESULTS
Use the attribution codes of the class file
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Each EOC had a predicted probability map
DESCRIPTION OF THE TEST
For any image in the test folder
EXPECTED RESULTS
A probability map is predicted by the model for each EOC it was trained to extract
OBTAINED RESULTS
-

1.3 Module 3: Import images

IDENTIFICATION OF COMPONENT
Images downloaded successfully
DESCRIPTION OF THE TEST
For each IIIF links
EXPECTED RESULTS
There is a JPEG image in the page folder
OBTAINED RESULTS
-

1.4 Module 4: EOCs to tree

IDENTIFICATION OF COMPONENT
Tree is structured hierarchically
DESCRIPTION OF THE TEST
For XML trees
EXPECTED RESULTS
Illustrations are outside any text block, and lines of text, as well as annotations, are inside text blocks
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Each node got all its information
DESCRIPTION OF THE TEST
Nodes of the XML tree
EXPECTED RESULTS
have a label, coordinates and a name following the naming convention.
OBTAINED RESULTS
-

1.5 Module 5: Export results

IDENTIFICATION OF COMPONENT
Each node of the tree was used for exportation
DESCRIPTION OF THE TEST
Nodes of the XML tree
EXPECTED RESULTS
have a corresponding vignette with the same name in the vignette folder.
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
ALTO file is complete
DESCRIPTION OF THE TEST
Tags of the ALTO file from the XML tree
EXPECTED RESULTS
have an id (name), coordinates and a style (label).
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
The structure is the same as that of the tree
DESCRIPTION OF THE TEST
For each node of the XML tree that has a parent node, the corresponding tag of the ALTO file
EXPECTED RESULTS
has the corresponding parent tag.
OBTAINED RESULTS
-

1.6 Module 6: End user interface

IDENTIFICATION OF COMPONENT
User can select images and a current image
DESCRIPTION OF THE TEST
Click on Import image... button. Select one or more images.
EXPECTED RESULTS
First image is loaded and displayed as the Current image. Other images are loaded in the background.
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Operations can be selected by tabs
DESCRIPTION OF THE TEST
Click on Import image... button. Select an operation.
EXPECTED RESULTS
The current image is processed and displayed.
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Operations can be added to from scenarios
DESCRIPTION OF THE TEST
Click on the tab of the operation. Set up the operation.
EXPECTED RESULTS
The operation is added to the list of the scenario.
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Operations can be removed from scenarios
DESCRIPTION OF THE TEST
Click on an operation in the scenario. Click Remove.
EXPECTED RESULTS
The operation is removed from the list of the scenario.
OBTAINED RESULTS
-

IDENTIFICATION OF COMPONENT
Scenarios are saved in XML files
DESCRIPTION OF THE TEST
Click on the tab of the project. Click on Save Scenario...
EXPECTED RESULTS
The XML of the scenario contains the parameters of the operation.
OBTAINED RESULTS
-

2 Integration testing

2.1 Module 1: Import training data

- Data fits into memory: importing datasets from the file system does not throw a memory allocation error.
- The module is run before the deep learning lab functions.

2.2 Module 2: Deep learning lab

- Regarding requested EOCs to Training data import module, a pipeline for these EOCs has been developed
- Models' variables are serialised in the file system

2.3 Module 3: Import images

- The module connects to the Internet.

2.4 Module 4: EOCs to tree

-

2.5 Module 5: Export results

- ALTO file is stored in the file system.

2.6 Module 6: End user interface

- 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

G Glossary

A	
Agile	A set of software development practices designed to create and respond to ch
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 neura
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

H Bibliography

- S. Ares Oliveira, B.Seguin, and F. Kaplan, “dhSegment: A generic deep-learning approach for document segmentation”, in Frontiers in Handwriting Recognition (ICFHR), 2018 16th International Conference on, pp. 7-12, IEEE, 2018.
- K. Nikolaidou, M. Seuret, H. Mokayed and M. Liwicki, “A survey of historical document image datasets”, IJDAR 25, 305–338 (2022)
- PARADIIT Project
- GitHub deep-agora
- GitHub dhSegment-torch

Research & Development Project

Deep-Agora : Incremental segmentation of images of old documents

Théo BOISSEAU

Supervisor : Jean-Yves RAMEL

université de TOURS

In collaboration with Centre d'études supérieures de la Renaissance

Objectifs

- point 1
- point 2
- point 3

LIFAT EA 6300

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

Mise en œuvre

1. point 1
2. point 2
3. point 3

LIFAT EA 6300

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

Résultats attendus

Voici du texte. Voici du texte. Voici du texte.
Voici du texte. Voici du texte. Voici du texte.

LIFAT EA 6300

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

Ecole Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Computer Science

Research & Development Project

Deep-Agora : Incremental segmentation of images of old documents

Théo BOISSEAU

Supervisor : Jean-Yves RAMEL

In collaboration with Centre d'études supérieures de la Renaissance

Objectifs

- point 1
- point 2
- point 3

Mise en œuvre

1. point 1
2. point 2
3. point 3

Résultats attendus

Voici du texte. Voici du texte. Voici du texte. Voici du texte. Voici du texte. Voici du texte.

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS
LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

Ecole Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Computer Science

Deep-Agora

Incremental segmentation of images of old documents

Résumé

La collaboration avec le CESR a donné naissance au logiciel Agora (issu du projet PaRADIIT) qui réalise simultanément l'analyse de la mise en page, la séparation texte/graphique et l'extraction de motifs. L'objectif de ce projet est de faire une refonte complète d'Agora en utilisant une nouvelle approche orientée vers l'apprentissage profond.

Mots-clés

document, ancien, segmentation, sémantique

Abstract

The collaboration with the CESR resulted in the Agora software (from PaRADIIT Project) which simultaneously performs page layout analysis, text/graphics separation and pattern extraction. The objective of this project is to do a complete overhaul of Agora using a new approach oriented towards deep learning.

Keywords

historical, document, semantic, segmentation, alto

Company
Centre d'études supérieures de la Renaissance

université de TOURS

Student
Théo BOISSEAU (DI5)

Academic supervisor
Jean-Yves RAMEL

Industrial supervisor
Rémi JIMENES