UNIVERSITY OF ST ANDREWS

MASTERS THESIS

Automated Optical Detection and Recognition of Greek Letters on Degraded Papyri

Author:
Carson BROWN

Supervisor:
Dr. Mark-Jan NEDERHOF

University of
St Andrews

March 21, 2023

Abstract

Annotating and transcribing papyri is a time-consuming and complicated process that requires expertise in both language and papyrus to correctly transcribe damaged texts. This report proposes a fully autonomous pipeline that takes digital images of papyrus and outputs bounding boxes and classifications for each glyph, as well as grouping glyphs into potential lines. The pipeline consists of a binarization step, where glyphs are digitally separated from the papyrus using various methods, a glyph bounding step, a line grouping step, and finally a classification step, each with options that can be used to increase recall or precision. The processes for generating the pipeline and various alternative approaches are discussed, along with an evaluation of the pipeline and its component modules.

Acknowledgements

I would like to thank my project supervisor, Dr. Mark-Jan Nederhof, for his insights on language and character recognition and his help on this project. I would also like to thank Dr. Isabelle Marthot-Santaniello for her assistance with understanding the dataset. Finally, I would like to thank my family and friends who have supported me throughout my education; your support is invaluable.

Declaration of Authorship

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 12,782 words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Signed: *Carson Brown*

Date: *March 21, 2023*

Contents

List of Figures

List of Abbreviations

CC	**C**onnected **C**omponent
CER	**C**haracter **E**rror **R**ate
CNN	**C**onvolutional **N**eural **N**etwork
COCO	**C**ommon **O**bjects in **CO**ntext
CSV	**C**omma **S**eparated **V**alues
DBSCAN	**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise
DIBCO	**D**ocument **I**mage **B**inarization **CO**ntest
DOG	**D**ifference **O**f **G**aussians
GMM	**G**aussian **M**ixture **M**odel
HOG	**H**istogram of **O**rdered **G**radients
HSV	**H**ue **S**aturation **V**alue
IOU	**I**ntersection **O**ver **U**nion
k-NN	*k*-**N**earest **N**eighbors
MNIST	**M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology
NN	**N**eural **N**etwork
OCR	**O**ptical **C**haracter **R**ecognition
POI	**P**oint **O**f **I**nterest
RCNN	**R**egion Based **C**onvolutional **N**eural **N**etwork
RGB	**R**ed **G**reen **B**lue
SVM	**S**upport **V**ector **M**achine
SIFT	**S**cale **I**nvarient **F**eature **T**ransform
VOC	**V**isual **O**bject **C**lasses
YOLO	**Y**ou **O**nly **L**ook **O**nce

List of Symbols

tp	True Positive	correct positive prediction
fp	False Positive	incorrect positive prediction
fn	False Negative	incorrect negative prediction
tn	True Negative	correct negative prediction

Equation Definitions

Precision	$p = \frac{tp}{tp+fp}$
Recall	$r = \frac{tp}{tp+fn}$
F_1 Score (FScore)	$f_1 score = \frac{2 \cdot p \cdot r}{p+r} = \frac{2 \cdot tp}{2 \cdot tp+fp+fn}$

Chapter 1

Introduction

Papyrus was used in the ancient world as early as the fourth millennium BCE as a writing medium, behaving similarly to a thick paper [28]. Constructed from the pith of the papyrus plant, multiple sheets were often joined together to form scrolls. Although originally from Egypt, ancient papyri have been recovered from areas throughout the Mediterranean region containing writings detailing topics from the Gospel of John to remedial methods for scorpion poison and descriptions of demonic and mystical spells [12, 3, 10]. The texts, containing a vast set of information providing insights into daily life, culture, religion, and politics were written in a variety of languages, including Greek Coptic, and Hebrew in addition to Egyptian Hieroglyphic, Hieratic, and Demotic.

With such a wide array of topics and languages, it takes significant amounts of time, resources, and expertise to accurately transcribe and catalog the information contained on even a single sheet of papyrus. With hundreds of thousands, if not millions, of surviving papyrus artifacts contained within both museums and private collections, a pipeline that could augment existing resources to assist papyrologists with this task would allow for "major steps forward" in the field [40]. With papyrus being a "very challenging type of historical document" to digitally transcribe, there are not yet any reliable tools that address this problem, which this project, inspired by the contest organized by Dr. Isabelle Marthot-Santaniello [40], seeks to remedy.

This project is focused on automating two separate but connected tasks: the localization of glyphs and the classification of glyphs. Using a dataset of images of papyrus from multiple collections, representative of the diversity of Greek papyri artifacts, this project hopes to create a general solution to the problem of automating the transcribing of ancient Greek papyri samples [40]. This project makes use of optical character recognition and natural language processing techniques in addition to exploring the use of convolutional neural networks, among other techniques, to classify and localize glyphs.

Chapter 2

Context Survey

2.1 Binarization

As many of the most common, commercial, and competitive optical character recognition (OCR) models and methods require or work better with a binarized image [25, 42, 41, 5, 6] (also called a bi-level image), it is important to explore binarization as a method to improve the results of the pipeline. Various methods are utilized for binarization of historical manuscripts, including thresholding [5, 6], pixel clustering [8], and convolutional neural networks (CNNs) [17, 18, 51]. In each of these methods, the goal is the same, to separate the original markings (ink) on the document from the rest of the image. This usually involves generating a second image where the ink is marked by a black pixel, with the rest of the image being white.

In the case of papyri, this is not a trivial task, requiring techniques to have both high precision and recall to generate useful bi-level images to pass forward to further steps of the OCR pipeline. The strongest methods for binarization are able to not only differentiate ink from papyrus, but also identify extraneous text markings such as ruler marks and annotations added by historians, frames or cases containing the papyri, as well as elements in the image that are either edited in after the image is taken.

Bar-Yosef [5] and Bar-Yosef et al.[5] both utilize a simple thresholding algorithm to generate an initial binarized image to use for further classification.

Bera et al.[8] propose an approach to binarization that involves using the fuzzy C-means, K-medoids and K-means++ clustering algorithms to form clusters of pixels into groups representing foreground and background elements. This method also makes use of normalization, to remove noise and shadow, and thresholding, maximizing inter-cluster distance and minimizing intra-cluster distance, thus generating very specific clusters. Pixels are labeled by the clusters and then the final output is determined by a decision algorithm that takes in the labels from the three clustering algorithms. This approach generates an F-measure result of 76.84 and a peak signal-to-noise ratio of 15.31, both measured against the 2018 DIBCO (Document Image Binarization COntest) dataset[39].

Dhali et al.[17] and Xiong et al.[51] both utilize CNNs for this task using manually annotated data in combination with transfer learning to retrain existing models to generate new networks with F-measure results of 86.7 and 92.81, respectively, and peak signal-to-noise ratios of 21.3 and 17.56, respectively (all measured against DIBCO'18). While it has a lower F-measure on the whole of the DIBCO'18 dataset,

the network proposed by Dhali et al., BiNet, was trained more specifically on degraded manuscripts and is shown to also be able to remove the extraneous elements from the image including ruler marks, machine-printed color-calibrators, and picture frames.

By combining and contrasting these methods against each other, a high quality, bi-level representation can be generated that can be passed to the next step of an OCR pipeline to bound and classify each glyph with potentially more accuracy than would be possible with an unaltered image.

2.2 Glyph Grouping / Bounding Boxes

Precise bounding boxes are key to the success of every classification method, as removing noise from the edges of the glyphs to increase the signal-to-noise ratio of the image being input into the classification algorithm. Similarly, to utilize a language model to help assist in classification, or to generate document level transcriptions, a method for converting a list of bounding boxes into pairs or larger collections of neighboring glyphs is required.

A method proposed by Garz et al.[22, 23] utilizes points of interest to find the spaces between lines, which requires no binarization, and can be performed on the source image without first generating bounding boxes, which allows this method to generate potential bounding boxes by detecting clusters of points of interest (generated using Difference of Gaussian [34]). These points of interest are then used to generate bounding boxes using Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [19], which in the work proposed by Garz et al.[22, 23] requires a manual estimation of neighborhood size to create clusters. However, it could be possible to automatically estimate this distance using a more basic bounding-box finding algorithm or the analytical approach suggested by Daszykowski et al. [14]. Points of interest are also utilized to generate lines by splitting the image on the lines of lowest energy (farthest from the points of interest), weighting horizonal movement as lower energy than vertical movement. This method, analyzed on hand-written text from all 60 pages of the *Saint Gall* database, has a line accuracy of 97.97% over 1431 lines of text, some of which contain background noise. This level of accuracy may allow this method to be successfully adpadted to papyrus.

Williams [49] also proposes a method for line detection based on finding gaps in the vertical coordinates of the glyphs. By finding the potential centers of each glyph and then sorting all the potential centers by their vertical coordinates, significant gaps in the list are used to separate lines. For each line, Williams then takes the line of best fit, using regression, to determine a potential line. A second pass then separates glyphs that are outside of the standard distance by a certain amount. This method appears to work quite well for glyphs where many potential centers are known, such as for crowdsourced data or binarized data with centroids. However, it fails significantly when a line has enough curvature that two lines occupy the same vertical coordinates over the horizontal span of the image.

In contrast to the algorithmic approaches above, Ezra et al.[7] propose a supervised learning approach for transcription alignment and bounding. Using a CNN-RNN trained to classify glyphs on lines, this method was able to generate results with an accuracy of 90.3%. They also achieved a mean bounding box overlap of 81.0% using the same model.

Another potential bounding method that could be explored is utilizing binarization to generate blobs for each glyph and bounding each blob. Combined with any of the above methods, this could create tight bounding boxes for use in classification as the bounding boxes could either be kept binarized or converted back to the full color image, depending on the classification method being used.

Other potential sources for line finding are Tesseract[42] and Ocular[9]. Tesseract is a popular OCR engine with the ability to perform line finding [41] using blobs on images that can be skewed, meaning the image quality is not lost due to warping the images the images to allow for more sensitive techniques to be performed.

2.3 Classification

Once the glyphs are bounded, they need to be classified. This can be done utilizing many methods, some of which could be combined to improve the results. These methods include utilizing transfer learning and image similarity [46, 54], CNN based classification [53, 26, 43], freeman chains [2], probabilistic models [9], and image deformation [35, 31, 44, 24]. While not all of the methods require binarized images to work, these methods may still be improved by precise binarization, as it would remove the need for the model to handle the noise added by the papyrus itself and any degradation of the papyrus.

Bar-Yosef et al.[6] also expand on the work done by Bar-Yosef by eroding characters to generate 'structuring elements' for each glyph, unique elements of glyphs that can be overlaid over the binarized text to identify characters of each glyph class. Using this method, Bar-Yosef et al. were able to achieve an accuracy of 94.65% on a test set of 1477 aleph characters over 22 Hebrew documents. By combining the more advanced binarization methods proposed by Xiong et al.[51] and Dhali et al.[17] it may be possible to generate similar results on the more degraded, ancient Greek documents targeted by this research.

Using image similarity as proposed by Vadicamo [46] and Yuan et al. [54] or image distortion proposed by Keysers et al. [31] could also allow for classification based on referencing known characters, which would allow for the pipeline to be expanded to any glyph set with little effort. Both methods rely on finding glyphs from a dataset that are the closest to the found glyph from the text being transcribed. While image distortion uses algorithmic approaches, image similarity uses CNNs to perform feature extraction from the image before classifying the image, which could utilize distance, clustering, or other unsupervised learning techniques such as random forests. These methods could both be performed on binarized and non-binarized images, but it is likely that they would perform better on high-quality bi-level images than the full color, noisy images. While these methods do not provide any accuracy measures for classifcation using image similarity, Nederhof [35] reports an accuracy of 91.3% with accuracy increasing to 95.5% if the two most likely classes are considered.

Similar to the above methods, using Freeman chain encoding could allow for found glyphs to be encoded in a way that would allow for finding similar glyphs from a database of known characters, using either a distance measure or unsupervised learning techniques. In the method proposed by Althobaiti et al. [2], binarized characters are converted to lists of numbers by tracing the outline of the glyph and recording the slope at each pixel as a number from one to eight based on the direction

of the next pixel in the outline of the glyph. Using this method on handwritten Arabic text, the authors achieved an average accuracy between 92% and 97% depending on the character. Like the methods described previously, this method would also enable the pipeline to be easily translated to another glyph set without needing to retrain the model.

Finally, several approaches to CNN based classifcation are suggested by Yousefi et al.[53], Haliassos et al.[26], and Swindall et al.[43]. Yousefi et al. suggest the use of LSTM Networks on non-binarized images, achieving accuracies of less than 1% on greyscale images. However, Yousefi et al. worked with images with very low noise, and thus had significantly more well-formed letters than could be expected on papyrus. For results on datasets closer to what could be expected from ancient papyrus, Haliassos et al.[26] suggest using Support Vector Machines (SVMs) to classify images of glyphs using Histogram of Ordered Gradients (HOG) features and CNNs to classify glyphs on ancient papyrus, achieving accuracies of 88.71% and 95.77% respectively, with Swindall et al.[43] achieving similar CNN-based results on papyrus with an accuracy of 92.73%

2.4 Crowdsourced Transcriptions

A large portion of the research in ancient Greek manuscript OCR has been done utilizing crowdsourced citizen science, allowing papyrologists and non-papyrologists alike to annotate ancient Greek manuscripts utilizing various pipelines that allow users to detect and/or classify Greek letters.[49, 48, 44, 4] These methods, while not helpful in generating a fully automated pipeline, are valuable for generating data for both training and testing of fully automated pipelines.

Like Dhali et al.[17], Tabin[44] utilizes a manual facsimile generation process that allows a user to manually annotate pixels in images, which they use to perform OCR on directly, but that could also be utilized in generating more data to train on as Dhali et al.[17] proposed.

Another usage for crowdsourced OCR is proposed by Williams [49], in which the consensus transcriptions are compared to known Greek texts to determine if the crowdsourced transcription may be a part of a known text. This allows for error correction and filling in the gaps, as well as the potential for creation of more accurate training data, as damaged or partially-missing glyphs could still be classified correctly if they can be matched to known glyphs from other texts.

This method for transcription, while not able to be directly integrated into a fully automated pipeline for OCR, is a potentially valuable source for training and/or testing data, and could be utilized to verify or improve the results of a future pipeline.

Chapter 3

Ethical Considerations

This project raised no ethical concerns. The self-assessment form submitted at the beginning of the project remained valid throughout the lifetime of the project and can be found in the appendix.

Chapter 4

Requirement Specification

4.1 Primary Objectives

Investigate the following:

- taking in an image of a single sheet of ancient papyrus with Greek glyphs and returning to the user the Greek glyphs that appear and the bounding boxes of the Greek glyphs.
- separating the ink in the image from the rest of the image (binarize) and saving the resultant image.

4.1.1 Secondary Objectives

Investigate and compare some or all of the following to generate a pipeline:

- CNN-based methods for glyph bounding, separation, and/or classification
- image distortion methods for glyph classification
- stochastic model-based methods for glyph classification
- algorithmic-based methods for glyph bounding and/or separation
- NN-based methods for text line grouping
- algorithmic-based methods for text line grouping

The following was another potential task, but the dataset did not support it, so it could not be explored in the time restraints of the project:

- investigate bounding of diacritic signs of the characters and see if these can be automatically classified as well.

Chapter 5

Software Engineering Methodology

This project was primarily written in Python 3.10 utilizing Conda for dependency management. While Python is not the fastest language, it is optimized for prototyping and can construct and running neural networks, which makes it an optimal choice for this project as there were no speed or space-based performance requirements defined in either the original contest briefing, or the requirements defined for this project.

Given the exploratory nature of the project, agile development methodologies were utilized to quickly evaluate methods and their combinations to yield the best results. With a cycle length of one week, ideas were considered, implemented, and then evaluated to determine which methods were generating helpful results and which could be abandoned in favor of other approaches. By using this short cycle time, ideas that did not work were quickly discarded, allowing time to be spent efficiently while still allowing multiple methods to be explored. As the requirements for the project did not change cycle to cycle, determining the per-cycle requirements of any given cycle could be done by either working to optimize an existing method that showed promise or implementing a new method. In addition, by keeping the requirements the same week-to-week, it was possible to quickly integrate and test newly implemented techniques against their existing counterparts, as old metrics were able to be reused for comparison without redefining new goals and retesting each existing method on a new metric.

For the same reason, the results of previous experiments were saved and made accessible to future experiments, vastly decreasing the time spent on each experiment. This was done using methods such as using a modular pipeline architecture, pickle [38] (python object serialization) for large datasets, and storing intermediate and reusable images to disk, instead of regenerating them for each experiment. Pickling datasets and storing images were most useful for storing the output of CNNs where the processing time was magnitudes larger than the time it took to read the information from disk. These methods also guaranteed that all experiments were performed with the same parameters by ensuring that accidental changes to an already implemented technique would not alter the results of future experiments.

Even with the protections provided by the methods described above, care was taken to maintain code quality and validity when making changes, ensuring that results could be replicated or improved on after each change was made. This also involved maintaining comments and non-code elements of the project, such as file structures

and dependency versions. The latter is also being assisted by caching potentially vulnerable dependencies, such as neural network structures and weights.

Chapter 6

Design

6.1 Design Constraints

6.1.1 Time Efficiency

In addition to the project goals, there were also some more technical design restrictions that were either self-imposed or implied. Given that this project was based on a contest, it was decided that the restrictions explicitly given, or implied, by the contest could be utilized as design guidelines. As the contest gave a week between the release of the test data and the deadline for test data classification and bounding, the same restriction on computation time was given for this project. This meant that any result of the pipeline would need to utilize up to a week of processing time for an input of 31 images, allowing for 5.4 hours per image, a time restriction so generous that it could be ignored as a constraint.

6.1.2 Input

The contest also supplied images in various formats, such as `.png` and `.jpg`/`.jpeg`. As such, the pipeline can take any of the common image formats, such as PNG, JPEG, TIF, and BMP, but not GIF due to restrictions imposed by the cv2 and PIL. As approximately 2% of the training data given by the contest contained malformed images, requiring that the pipeline gracefully manage such files allowed for more training data.

In addition to accepting images as input for processing, the program accepts bounding box and glyph category information for the purposes of training and evaluation. This information is accepted, as in the contest, in the COCO image format which formats information in a JSON structure (Figure A.2). Each COCO file contains lists of images, classes, and annotations, with each object being defined by a single annotation that references an image and a class and provides a bounding box, along with any metadata that the dataset may provide. In the case of the competition, additional information about the classes of the glyphs were provided, grouping each class of glyphs into sub-types, and providing a rating of how well-formed each glyph is.

6.1.3 Output

To evaluate submissions, the contest required COCO-formatted output, so in addition to a simple csv output format (Figure A.1), the pipeline supports a COCO-formatted output.

6.2 Pipeline

The project is designed with a pipeline type structure with a single input and output. With this structure, the pipeline takes all the images in a directory through the pipeline one at a time, allowing the program to only need to load one image at a time, while keeping any neural networks in memory, reducing the memory load while keeping time complexity down, especially for bulk processing jobs.

6.3 Modular Components

With such a pipeline structure, the importance of allowing internal modularity becomes clear, especially when evaluating the attempted method. Each method implemented is constructed with modularity as a core focus, allowing methods to be slotted together without additional design complexity. This approach to the design of individual components of the pipeline allowed for quick prototyping of the pipeline and efficient evaluation of individual modules without needing to construct a novel information flow for each method that was attempted. By designing the pipeline components in this way, the pipeline is also already optimized for future expansion or reconfiguration.

6.4 User Interface

Due to being designed in the spirit of generating results for a contest, the user interface of the project is limited to the command line. While potentially not optimal for most users, this design is the most compact and easy to use for the purpose of generating results.

Chapter 7

Implementation

7.1 Binarization

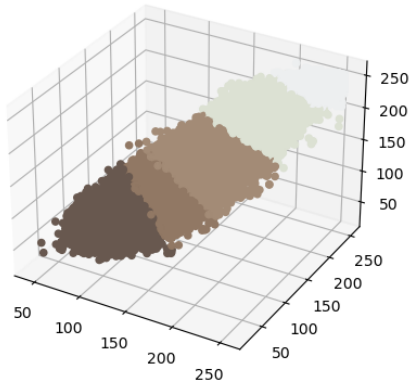
A perfect binarized version of a papyrus image is defined by a background color and ink color, where the ink is black, and the background is white. Each pixel in an image is altered so that it is either black or white, depending on if it is ink or anything else (background). As no ground truth for ink was given to train on, all of the methods for binarization are based on either pre-trained CNNs or algorithmic solutions utilized in similar problems.

7.1.1 Clustering

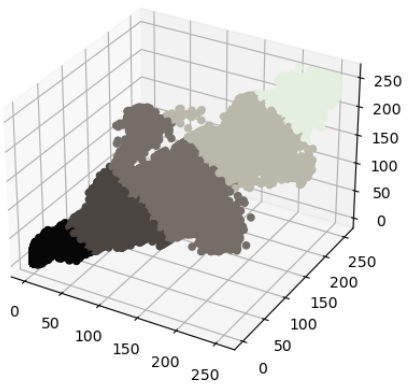
Clustering-based binarization is performed with k-means clustering. For each image, the pixels of the image were placed in a 3D space with their red, green, and blue values being used to determine their location (Figure 7.1). Pixels were then clustered in groups using k-means clustering. The pixel cluster which contained the value closest to black was then utilized as the ink cluster, with all pixels not in that cluster being labeled as the background (Figure 8.2).

FIGURE 7.1: RGB Pixel Clustering In 3D Space

(A) Pixel Clustering on an Image With Low Color Variation



(B) Pixel Clustering on an Image With High Color Variation



Two examples of clustering pixels using RGB color representation to place each pixel in space. These clusterings were generated using K-means Clustering with $k = 5$, with each cluster being displayed by the color represented by the location in RGB space of the cluster's centroid.

7.1.2 Gabor Filter

Filtering-based binarizations are generated utilizing the Gabor filtering [20, 21] to find clusters of pixels that are not like those around them, which could be glyphs. Gabor filtering is based on a linear filter, but when multiple linear filters are grouped together with a set rotation applied to each subsequent filter, a 2D filter can be generated to create an approximation of Gabor wavelets [20, 21]. Using these 2D filters, this method of binarization seeks to find features in an image that have a specific frequency, which in the case of glyphs might be the width or height of a single line in a glyph.

This method requires an assumption on the wave frequency that may generate the glyphs, so a manual heuristic method is utilized to generate the function that most accurately generated the correct wave for the images in the training dataset.

To utilize the approximation on a Gabor wavelet on the input, the input is read in and converted to a greyscale image before having the collection of Gabor filters applied. This reduces the channels from three to one and allows the Gabor wavelet approximation to work on the image. The outputs of each filter in the wavelet approximation are then averaged together to generate a new greyscale image, which can be binarized using naive thresholding.

7.1.3 Input Invarient Thresholding

In a similar fashion to utilizing the Gabor filter or Gabor wavelet, thresholding utilizing an input invariant method, such as the one proposed by Bar-Yosef et al. [5, 6] requires consistency in the input or a strong method for determining the correct threshold. Using clustering, as discussed above, could be a potential method to find a threshold; however, the method then becomes a variant of clustering. Therefore, this method was not implemented past the initial stage of picking an arbitrarily chosen threshold value and binarizing the input by converting it to greyscale and then binarizing the greyscale image by converting all values darker than the threshold to black and all other values to white.

7.1.4 DP-Linknet

Although it was out of the scope of this project to train a CNN to binarize an image with manually generated training data, Convolutional Neural Networks have already been generated and trained on the problem of binarization of historical texts. DP-Linknet [51] is one such CNN, trained on historical texts and implemented in Python utilizing common deep learning tools such as PyTorch [36] and Scikit-Learn [37]. As the Python source code for DP-Linknet is publicly available [52], it was a simple task to implement it into the project and add a simple API that integrates the CNN into the pipeline.

Note: All code contained in the `src/binarization/dp_linknet` *directory is either directly from the GitHub* [52] *reposity or highly based on that code. I take no credit for this code, as minimal work was needed to integrate the work of Xiong et al. into this project.*

7.2 Glyph Bounding

7.2.1 Bounding Boxes

The rectangular areas that contain glyphs (bounding boxes) are represented internally by an object that holds the PASCAL Visual Object Classes (VOC) representation of a bounding box (Figure 7.2). Using this method of representation, compared to Python's Tuple or List objects, allows for operations on bounding boxes to be standardized and for bounding boxes to be represented in a way that is abstracted from their format. Allowing this abstract bounding box format assists with the problem of having multiple constraints on how bounding boxes are utilized in the pipeline, such as importing and exporting COCO-formatted bounding boxes (Figure 7.3), calculating metrics based on PASCAL VOC-formatted bounding boxes (Figure 7.2), and utilizing tools and methods that require YOLO-formatted bounding boxes (Figure 7.4). Bounding box objects are then able to utilize internal methods for operations such as finding the IOU (intersection over union) (Figure 7.5) between two bounding boxes, calculating the one dimensional IOU (Figure 7.6), and generating cropped images from lists of bounding boxes.

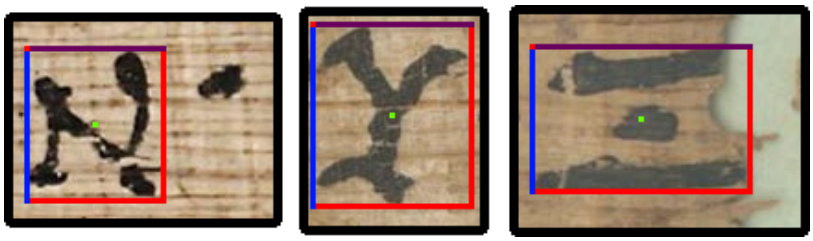
FIGURE 7.2: PASCAL VOC Bounding Box Examples

Three examples of bounding boxes, on a N, Y, and a Ξ (left to right). For these examples, the red lines denote the pixels that are inside of the bounding boxes. In the PASCAL VOC bounding box format, bounding boxes are described using the top left corner (marked in green) of the box in absolute pixels, and the bottom right corner of the box (marked in blue) in absolute pixels. In this example, the boxes shown above would be denoted in the PASCAL VOC-format as follows: $N : (6, 13, 82, 97)$, $\text{Y} : (1, 3, 89, 103)$, $\Xi : (6, 16, 126, 96)$.

FIGURE 7.3: COCO-Format Bounding Box Examples

Three examples of bounding boxes, on a N, Y, and a Ξ (left to right). In the COCO bounding box format, bounding boxes are described using the top left corner (marked in green) of the box in absolute pixels and the box's width (purple) and height (blue) in absolute pixels. In this example, the boxes shown above would be denoted in the COCO-format as follows: $N : (6, 13, 76, 84)$, $\text{Y} : (1, 3, 88, 100)$, $\Xi : (6, 16, 120, 80)$.

FIGURE 7.4: YOLO-format Bounding Box Examples

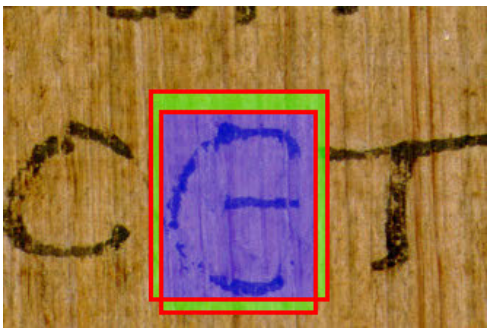


Three examples of bounding boxes, on a N, Y, and a Ξ (left to right). In the YOLO bounding box format, bounding boxes are described using the center of the box (marked in green) in relative coordinates (based on image width and height) and the box's width (purple) and height (blue) also in relative coordinates. In this example, the boxes shown above would be denoted in the YOLO-format as follows (rounded to three decimal places): $N : (0.319, 0.519, , 0.532, 0.739)$, Y : $(0.469, 0.469, 0.917, 0.885)$, $\Xi : (0.431, 0.487, 0.784, 0.696)$.

FIGURE 7.5: 2D Intersection Over Union (IOU) Examples

(A) Bounding Boxes With A Low IOU Value

(B) Bounding Boxes With A High IOU Value



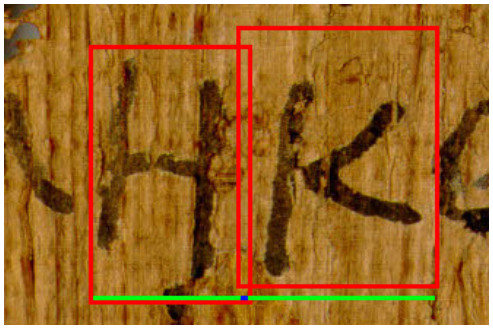
The IOU of two bounding boxes is a measure of how similar they are (with a range denoted by the interval $[0, 1]$). It is calculated using the following formula, where a and b are bounding boxes represented by sets of pixels.

$$IOU(a, b) = \frac{a \cap b}{a \cup b} \tag{7.1}$$

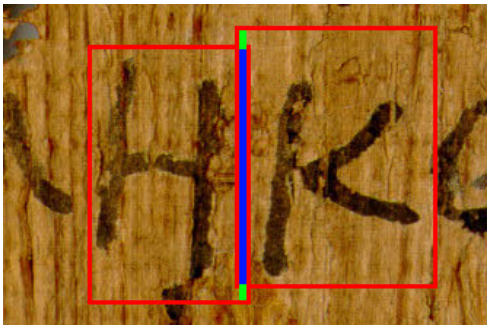
Figure 7.5a is an example of two bounding boxes with a low IOU value as the union of the sets (green and blue regions) is much larger than the intersection of the sets (blue region). Figure 7.5b is an example of two bounding boxes with a high IOU value as the union of the sets (green and blue regions) is similar to the intersection of the sets (blue region).

FIGURE 7.6: 1D Intersection Over Union (IOU) Examples

(A) 1D Horizontal IOU Calculation



(B) 1D Vertical IOU Calculation

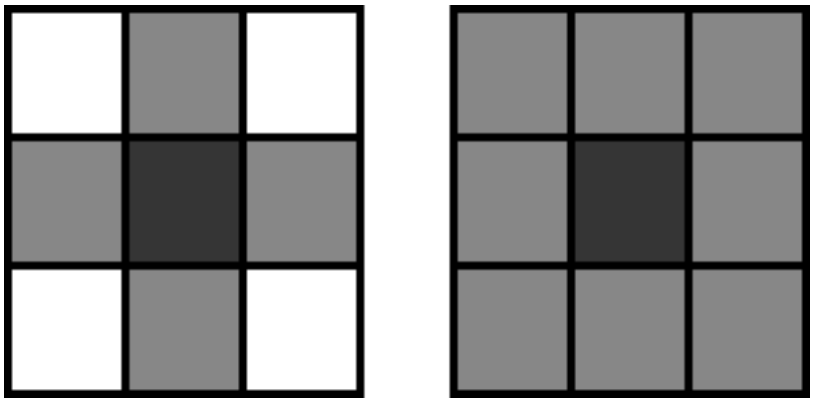


To find the one-dimensional IOUs of a pair of bounding boxes, both boxes are treated as a pair of one dimensional objects (lines) and the IOU is found as normal, calculating the intersection and union for the horizontal and vertical lines of the bounding boxes separately. In Figure 7.6a the horizonal IOU is low, with the intersection of the sets in 1D (blue region) being divided by the union of the sets (green and blue regions). Figure 7.6b has a high vertical IOU, with the intersection of the sets (blue region) being divided by the union of the sets (green and blue regions).

7.2.2 Connected Component Labeling

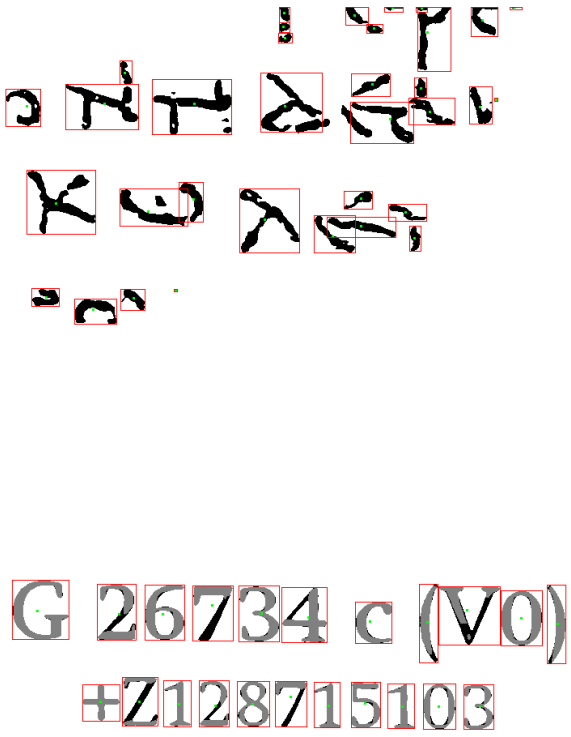
Bounding boxes can be generated by the pipeline utilizing a method called connected-component labeling. In this method, which is implemented by OpenCV [1], adjacent (8-connectivity) (Figure 7.7), similarly colored pixels are grouped together to form blobs in a recursive manner until no more blobs can be grouped. Each blob is given a label and a minimal bounding box (in PASCAL VOC-format) that contains the entire blob (Figure 7.8). These bounding boxes are then converted to a bounding box object, which allows them to be utilized by the rest of the pipeline.

FIGURE 7.7: Pixel Connectivity Types



In the left figure (which represents 4-connectivity), the center pixel (dark grey) is considered adjacent to the pixels that share an edge with it (light grey) and not the pixels that share a corner with it (white). In the right figure (which represents 8-connectivity), the center pixel (dark grey) is considered adjacent to the pixels that share an edge with it and the pixels and that share a corner with it (light grey). Note that the n in n-connectivity refers to how many pixels can be considered adjacent to any given pixel.

FIGURE 7.8: Connected Component Bounding Example



An example of a binarized file bounding using the connected component method using 8-connectivity. The red boxes denote the edges of bounding boxes (with the red pixels inside the bounding box). The green dots renote the centroid of the blob being bound for each box.

7.2.3 Region-Based Convolutional Neural Networks

Another method for bounding in the pipeline utilizes a region-based convolutional neural networks (RCNN) called YOLOv5 [29]. The RCNN was trained on the ground truth glyph images in two ways. In the first, YOLOv5 was trained on all twenty-four glyph classes, to allow YOLO to both classify and bound the glyphs. For the second training attempt, the RCNN was trained on glyphs as a single class, preventing YOLO from learning about the separate classes, but allowing it to much more efficiently learn how to bound glyphs and separate them from the noisy background that is papyrus.

Training was performed utilizing the native YOLOv5 code, downloaded from their GitHub page [30], using the largest network architecture provided (called YOLOv5x). The trained network consists of 86.7 million parameters and was trained on mosaic images of glyphs resized to 640px squares over the course of over 600 epochs. After training, the epoch with the best results was extracted and exported for use by PyTorch, which allows the network to be loaded and utilized in the pipeline with no external code required.

Note: YOLOv5 is the product of work by Glenn Jocher (Ultralytics) and hundreds of contributors. I take no credit for the implementation of this network, as all credit for this work should go to Jocher et al. However, all training and hyperparameter tuning was done in the course of the project, as well as integrating the network into this project.

RCNN Sliding Window

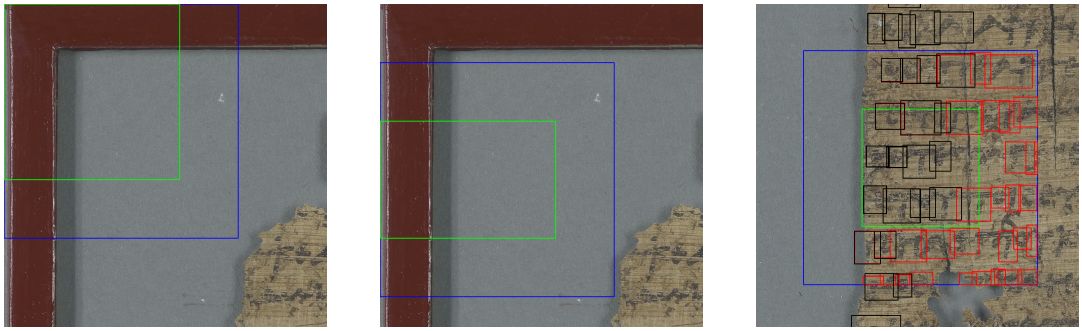
Utilizing the YOLOv5x-trained network and a sliding window approach to generating smaller images, glyphs can be detected in smaller groups. This works by creating a window that moves around the image, creating a series of overlapping grids that create smaller images that can be individually input into YOLO. Using an overlapping sliding window with an internal window (Figure 7.9), full images are split up into smaller images that allow YOLO to bound glyphs. The bounded glyphs are then accepted if they are in the internal window and added to the list of all the glyph bounding boxes as bounding box objects. Clear outliers and invalid glyph bounding boxes are then removed by a series of heuristic approaches that clean the list of bounding boxes to generate a list of high-quality potential glyph bounding boxes. There is also an option to utilize the sliding window without an internal window, which operates similarly to the main option, keeping bounding boxes if the bounding boxes do not touch the edges of the window, which prevents partial boxes from being kept.

FIGURE 7.9: Sliding Window Region Examples

(A) Corner Sliding Window Region

(B) Edge Sliding Window Region

(C) Center Sliding Window Region



Examples of the three cases of sliding window regions. In each image, the blue bordered region represents the region of the image that is given to the RCNN to locate glyphs in, with the green region representing the region of the image that a bounding box must be fully within to be accepted as a valid bounding box. Bounding boxes that are found, but not accepted as valid, are marked in red, with valid bounding boxes being marked in black. Bounding boxes marked in black outside of the green bounded regions are boxes that have already been accepted by previous windows. In the left-most figure (7.9a), the region of acceptance is expanded to fit to the corner, as there would otherwise need to be an image that is smaller than the rest to pass into the CNN. For the same reason, the middle figure (7.9b) shows a region of acceptance that is expanded to fit to the edge of the image. The region in the right-most figure (7.9c) is an example of a normal region, with no expansion required to ensure full coverage of the image.

RCNN Duplicate Removal

Duplicate bounding boxes are removed using an IOU threshold and the confidence of each bounding box. When two bounding boxes have an IOU over the threshold (which can be configured to modify both bounding box F1 score and classification F1 score), the bounding boxes with the lower confidence scores (generated by YOLO) are discarded and the bounding box with the highest confidence score is kept.

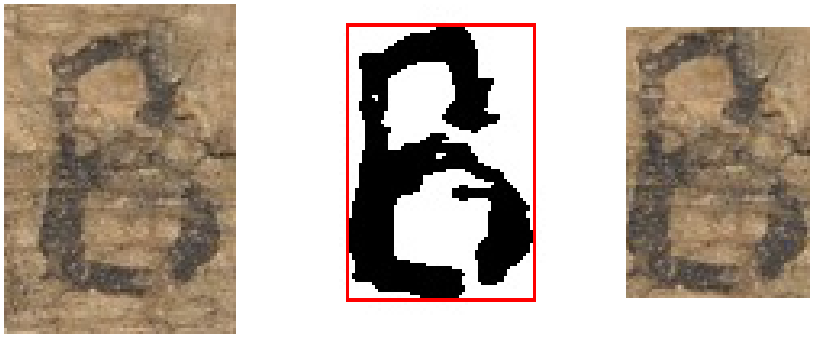
RCNN Malformed Bounding Box Removal

Malformed bounding boxes are defined as bounding boxes that are overly large or small, or bounding boxes that cover multiple glyphs or cover only a portion of a glyph. Bounding boxes that touch the edges of the sliding window are removed, as they have a high chance of being partial bounding boxes. Bounding boxes that cover multiple glyphs are cleaned by removing all bounding boxes that hold at least two other bounding boxes within them. This method also removes bounding boxes that contain two partial bounding boxes that were not correctly removed before this step.

Binarization-Assisted RCNN

The close cropping approach, based on connected component labeling on binarized images, can be done in conjunction with the larger bounding boxes generated from the RCNN-based methods. This allows the bounding boxes generated by the RCNN to be used as initial potential regions, which can be cropped down to closely fit the glyph they contain (Figure 7.10). This approach to creating tight bounding boxes on binarized images is able to approximate minimal bounding boxes for the raw input images, which would otherwise be difficult to generate tight-to-ink bounding boxes for, as the ground-truth training data is not tightly bound.

FIGURE 7.10: Binarization-Assisted RCNN Example



An example of a *B* bounding box that was generated and then tightly bounded using the binarized image as a guide. On the left is the original bounding box that was generated by the RCNN. This image has empty space around all sides of the glyph. In the middle is the binarized image with the generated tight bounding box drawn in red. On the right is the final generated bounding box, which has been cropped down to tightly bound the glyph.

7.3 Line Bounding

Once bounding boxes for individual glyphs are generated, they can be grouped together to form lines, which allow the language models in the classification step to have the context in which glyphs appear.

7.3.1 Point-Of-Interest Clustering

The first approach to line bounding is based on the work of Garz et al. [22, 23], which utilizes points of interest to group lines together. In this approach, gaussians and the SIFT algorithm are utilized to find points of interest, which are clustered using Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [19], an

algorithm which uses distance to recursively group points together, with an adaption that allows for the algorithm to be size invariant by dynamically generating the parameters for the algorithm using random noise [14].

Difference of Gaussians

The first step in this algorithm requires that features be extracted from the images in a scale-invariant way, such that any image size may be utilized without the need for scaling, which could cause information to be lost. To do this, the Difference of Gaussians (DOG) method is utilized. Using the implementation of this algorithm provided by OpenCV [1], a collection of gaussian kernels is generated and used to blur a greyscale image to extract high frequency spatial data. As multiple gaussian blurs are performed, only signals in the image that are between the frequencies of the gaussian kernels remain after the differences of all the gaussian blurred images are taken and used to construct a new image. This results in an image that contains only the high frequency data of the original image, regardless of the size of the original image.

Points of Interest & SIFT

Using the scale-invariant feature transform algorithm (SIFT) [34], on the high-frequency information extracted by DOG, key points are extracted from the image which are transform, scale, and rotation invariant.

Adaptive DBSCAN

Once points of interest are generated using SIFT, they are clustered by a scale invariant version of DBSCAN proposed by Daszykowski et al. [14]. This method works by first generating n random points in a region with the same size as the image, where n is the number of points of interest that need to be clustered. Then the distance to the mth closest point is found for each point, where $m = \frac{n}{25}$. The 95th percentile is then taken for these distances, which is used as the ϵ parameter for DBSCAN (the maximum distance for a point to be added to a cluster), with m acting as the minimum number of points required to allow a cluster to not be considered noise. The clusters produced by this method can be considered to be lines, while points outside of clusters are noise or glyphs that are not within lines.

7.3.2 Adjacent and Overlapping Glyph Clustering

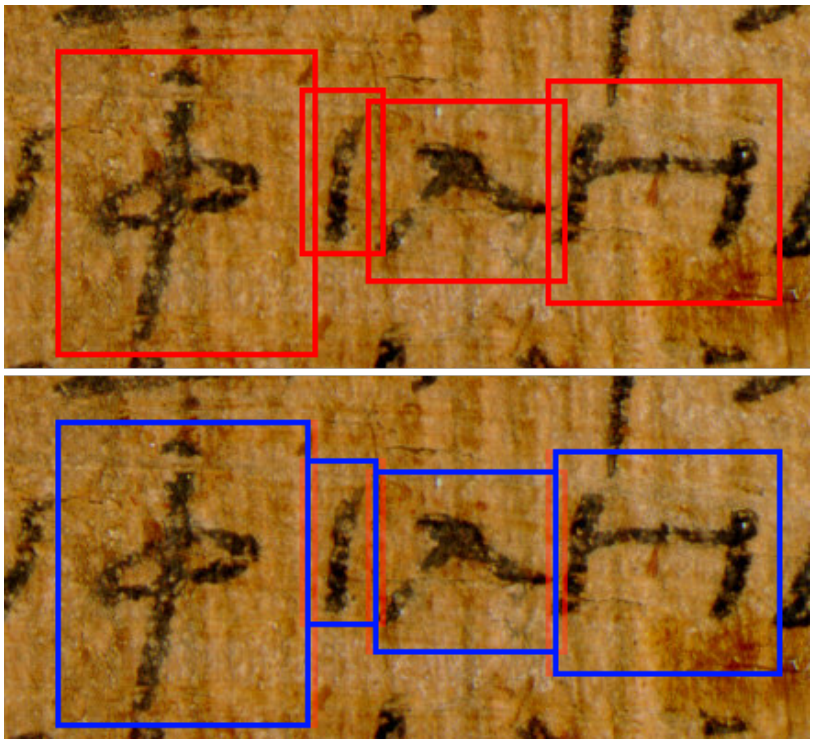
Another method for clustering glyphs is to utilize the oversized nature of the bounding boxes in the ground truth (and thus generated bounding boxes) to link glyph bounding boxes horizontally using their overlap. Using this method, glyphs are linked into lines based on proximity in the x and y directions. Horizontally overlapping bounding boxes are linked if they overlap and they each contain the y coordinate at the center of the other bounding box. Boxes are similarly linked if they are relatively close to each other, based on the average width of the two glyph bounding boxes.

Once rough grouping are generated using this method, the lines are cleaned of extranious and overlarge bounding boxes by removing bounding boxes that have no effect on the validity of the line based on this linking method. Lines are then joined together, if possible, to form longer lines.

7.3.3 Line Based Bounding Box Cropping

Once glyphs are linked into lines, the bounding boxes of the glyphs in each line can be cut down such that there is no overlap between bounding boxes. For each pair of overlapping bounding boxes, the center of the overlapping region is set as the edge for both bounding boxes, such that the bounding boxes touch, but do not overlap (Figure 7.11).

FIGURE 7.11: Overlapping Bounding Box Cropping



An example of a line with the overlapping sections of bounding boxes being removed, such that the boxes still touch, but do not overlap. In the above image, bounding boxes are shown in red. In the bottom image, the new bounding boxes are shown in blue, with a red, partially-transparent layer showing the old bounding box dimensions.

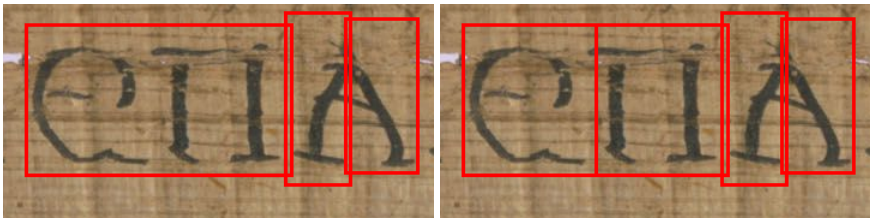
7.3.4 Line Variation Generation

Once lines are formed, they can also be used to modify bounding boxes to fix issues such as a bounding box covering two glyphs, or one glyph being covered by two bounding boxes (Figure 7.12).

Using lines and the probability of glyph classifications based on language models, the least likely bounding boxes are altered to generate lines with a higher likelihood. Starting with the lowest likelihood bounding box (based on classification certainty), bounding boxes are merged with their neighbors and split to generate two bounding boxes to attempt to increase the total probability of the line. This method can be done with individual bounding boxes or to whole lines, but as it generates exponentially more potential lines for each bounding box that can be altered, it is recommended that it be capped to only work on the least likely bounding boxes with an exact count being limited by the processing time available.

FIGURE 7.12: Line Bounding Variations

(A) Original Bounding Boxes (B) Potential Line Variant



(C) Potential Line Variant (D) Potential Line Variant

An example of bounding errors in a line and three of the potential variations that are made to the line to generate a more correct list of bounding boxes. Image 7.12a contains three bounding boxes generated by the RCNN (marked in red). Image 7.12b shows an example of the first bounding box being split, which would increase the accuracy of the bounding boxes for the *E* and Π. Image 7.12c shows an example of the second and third bounding boxes being merged, which allows the *A* to be more correctly bound. Image 7.12d shows a variation with both the merged and split bounding boxes, which has the highest accuracy and should have the highest total confidence in the classification.

7.4 Train/Test Data Extraction

7.4.1 Ground Truth Glyph Examples

The ground truth data provided by the contest contained image files, each with multiple bounding boxes and glyphs. To extract this data for the purposes of training and evaluation of classification and bounding methods, each original image was copied and cropped to contain only one glyph, generating a single cropped image, cut to the exact bounding box given, for each glyph (Figure 7.13). To extract the isolated glyph images for training, the following method was used. For each image in the training data, the image was loaded, and the COCO file provided by the contest was utilized to find each annotated glyph. For each glyph, the loaded image was duplicated and cropped to the bounding box provided in the COCO file. The cropped image was then saved to disk either separated by glyph class, or without any class label. Each cropped glyph file was saved with information on which image the glyph was cropped from, the style it was written in (footmark type) (Figure 7.14), and the preservation of the glyph (base type) (Figure 7.15).

These methods were used to extract glyph images from both the binarized images and unaltered images. Similar processes were also used to generate training data for the various CNNs and external tools utilized in the project.

FIGURE 7.13: Ground Truth Glyph Examples

Four examples of the bounded glyphs provided by the contest training data. All of the glyphs provided in the training and test data are uppercase and bounded by hand-drawn bounding boxes, which appear here. The first glyph, a Γ, is well-formed, but has the noise from another glyph on the left side and is truncated by the bounding box on the right side. Similarly, the second glyph, an *A*, is well-formed, but has the noise from another glyph on the left side and is truncated by the bounding box on the right and bottom sides. The third glyph, an *I*, is not cut off, but does have a line from a neighboring glyph as well. It also shows the edge of the papyrus. The fourth glyph, another *A*, is an example of the varying image quality of the training data, showing obvious artifacting or distortion.

FIGURE 7.14: Styles of Ξ in the Training Data

Examples of the six unique styles for the letter Ξ, as determined by the authors of the contest this project is based on. Numbered from 1-6 left to right. The furthest right image of a Ξ was resized for the purpose of display in the figure, which has increased the blurriness of the image.

FIGURE 7.15: Examples of Preservation Levels in Ωs in Training Data

Examples of the three levels of preservation for the letter Ω (appearing as the unical manuscript form, which looks similar to a larger lowercase modern ω), as determined by the authors of the contest this project is based on. Numbered from 1-3, left to right. There are three examples of each level to display three examples of the various degradations that result in assignment to each level. The least-degraded level (1) is defined as a complete or nearly complete glyph. The second least-degraded level (2) is defined as an incomplete letter that unambiguously belongs to one class. The most degraded level in the dataset (3), requires context to determine its actual class, as the glyph may appear to potentially be one of multiple classes when viewed in isolation.

7.4.2 Tightly Bound Glyph Examples

Using the methods discussed in subsection 7.2.3, tightly bound glyph images (Figure 7.16) were able to be generated and stored for use in training and evaluation using automatically generated bounding boxes, which contain less padding between the glyphs and the bounding box borders. After generating bounding boxes which are automatically cropped more tightly to the glyph, the ground truth bounding boxes were able to be paired with the generated bounding boxes and used to assign the known class to the generated bounding box. This allowed for generated bounding boxes to be stored with known classes so they could be utilized for training and evaluation of classification models on real data generated by the pipeline, instead of using the models trained on the ground truth bounding boxes.

FIGURE 7.16: Generated Cropped Glyph Examples

Four examples of the cropped glyphs generated by the pipeline. The first glyph, a *T*, is well-formed and is tightly bounded on all sides except the top. Similarly, the second glyph, an *O*, is well-formed, but has empty space on the bottom and right side. The third glyph, a *M*, is tightly bounded on all sides except the bottom and has degradation. The fourth glyph, an *H*, is poorly formed, with significant amounts of smudging, but is tightly bound on all sides. Glyphs are not reliably tightly bound on all sides, as the center is maintained when cropping, causing up to two sides of a well-formed glyph to have padding. Poorly formed glyphs are sometimes less tightly bound due to issues in their edges.

7.4.3 Greek Language Corpus

To form the training data for a language model, the entirety of the Greek section of the Perseus Digital Library [13] dataset was downloaded. The data was converted from XML to plain text and cleaned of all non-standard Greek letters utilizing a mix of regex, Python string functions, and a list of the glyphs that appear in the training set of images. When removing invalid characters, the passages that they appeared in were also removed to ensure the integrity of the training data. To ensure the language data is as close to the image training data as possible, all diacritics and punctuation marks were removed, and the letters were converted to their uppercase variant. The remaining data contains $46,400,000$ glyphs. This data was used to generate five smaller files for the purposes of training language models by randomly selecting continuous strings of characters from from the data.

7.5 Classification

7.5.1 Transfer Learning

The first form of classification attempted was image similarity using pretrained networks in a form of transfer learning to find the ground truth glyph image (and thus class) closest to an unseen glyph image. Due to the potentially time-consuming nature of developing and training CNNs to classify images, a section of a pre-trained CNN was utilized to convert individual glyph images to feature vectors of size $9,216$ to utilize in supervised and unsupervised learning. AlexNet [32] is used as it is a proven network that has shown promise as an engine for image similarity research [46, 54]. By taking all of the convolutional layers of AlexNet and the first fully-connected layer and utilizing the output of those layers as a feature vector, various unsupervised learning methods and later neural networks can be used to classify images without the ability to handle images directly. Images are first shrunk such that their smaller dimension is 240 and then a center crop of size 224 is taken before the pixels are normalized to the distributions expected by AlexNet. All unsupervised methods are based on the implementations by Scikit-Learn [37]. The supervised learning is implemented with PyTorch neural networks [36], partially based

on the code provided in this PyTorch RCNN training example [45] and the starter code provided as a part of the contest [40].

GMM Clustering

Using the feature vectors generated by AlexNet, Gaussian Mixture Model (GMM) clustering on the training data was utilized to generate a gaussian distribution in N-dimensional space for each class of glyph. Using this method, clusters were trained on feature vectors generated from both binarized and unaltered images. The trained clusters can be utilized to classify unseen feature vectors using the list of probabilities that a feature vector is the result of each gaussian probability distribution being used to classify each unseen feature vector.

Random Forest Clustering

Similarly, using the feature vectors generated by AlexNet, Random Forest Clustering was utilized to generate decision trees in N-dimensional space. Using this method, binary decision trees of varied sizes were trained on feature vectors generated from both binarized and unaltered images. The trained trees were then combined and utilized to classify unseen feature vectors where the chance of a feature vector being in a given class being proportional to the percent of trained decision trees that classify the feature vector to be that class.

K Nearest Neighbors

Finally, a K-Nearest Neighbors network was also generated on the training data. Using this method, feature vectors are classified based on how close they are to feature vectors of known classes. Feature vectors were classified for both binarized and unaltered images. The built networks are utilized to classify unseen feature vectors such that the chance of the feature vector being a given class is proportional to the number of feature vectors in the k-nearest neighbors that are from that class.

Distance in n-Dimensional Space

In the above classification techniques, the way that the distance between two points in n-dimensional space is calculated produces different results. Therefore, multiple distance metrics were used to increase the accuracy of the models. The first distance measurement is Euclidean distance, where the distance D between two points p and q in n-dimensional space is defined as

$$D_{Euclidean}(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{7.2}$$

Another distance measurement is Manhattan distance (also called taxicab distance), which is defined as

$$D_{Manhattan}(p,q) = \sum_{i=1}^{n}|p_i - q_i| \tag{7.3}$$

The final distance measurement used is the Cosine distance, a variant of Cosine similarity, which is defined as

$$D_{Cosine}(p,q) = 1 - \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}} \tag{7.4}$$

Neural Networks

With the feature vectors output by the modified AlexNet network, it is also possible to train deep neural networks to classify glyphs. Using the size of the feature vector as the input size $(9,216)$ and the number of glyphs (24) as the output, several dense NNs (See Figures B.1, B.2, B.3, & B.4) were generated to classify glyphs. Each takes a single feature vector as input and outputs a vector that can be put through the softmax function σ (Equation 7.5) to generate a percentage likelihood of the feature vector being from an image of each class of glyph.

$$\sigma(x_i) = \frac{e^{x_i}}{\sum^{j} e^{x_j}} \tag{7.5}$$

7.5.2 Convolutional Neural Networks

Once it was clear that neural networks were outperforming unsupervised learning approaches by a significant margin, CNNs were generated and trained to explore the possibility of further increasing the accuracy of the pipeline. These CNNs are mostly based on the convolutional layers of pre-trained and high-performing image classification CNNs, starting with AlexNet [32] (Figure C.1) and moving to ResNeXt [50], an expansion of ResNet [27] (see Figures C.2 & C.3, C.4, C.5). By extracting the convolutional and pooling layers from these neural networks and attaching a densely-connected network on the end with the correct number of output classes, these networks can be adapted to the problem of glyph classification quickly and with minimal training required. In addition to these pre-trained networks, several smaller networks based on the work of Deotte [16] were generated and trained from scratch (see Figures C.6, C.7, & C.8). The network constructed by Deotte was considered despite its smaller size more specific nature as it was specifically optimized for character classification on the Modified National Institute of Standards and Technology (MNIST) Digits dataset [15] with an accuracy of 99.75% on the MNIST data.

7.5.3 LSTM-Based Neural Networks

LSTM layers in a neural network allow the model to utilize knowledge of previous and potentially latter elements in a batch for the classification (or regression) for each input to the network. This is done by saving an internal hidden state that is altered by and utilized in the output of the layer after each forward step. These layers can be trained to discard their hidden state or alter their hidden state to increase the accuracy of a model that interacts with ordered information such as language data. These layers can be one-directional or bi-directional. Bi-directional LSTM layers work by duplicating and mirroring their input, generating twice the number of outputs, where half of the output of the layer is assuming the batch was passed in backwards.

Similar to the CNNs discussed above, bi-directional LSTM layers can be added to NNs by extracting the convolutional layers from the front of the models and inserting an LSTM layer between the convolutional and the final densely-connected layer

that generates the output of the model. In addition, densely-connected layers can be inserted between the LSTM layer and the convolutional and pooling layers to add an initial classification step before the memory layer and the final classification layer. By utilizing bi-directional LSTM layers, the networks can use the context on both sides of the glyph it is classifying to increase accuracy.

Training Methodology

To train these neural networks, techniques to augment the limited ground truth data are utilized. In training, each glyph image is passed into the CNN or NN with a random rotation ($\pm 15^{\circ}$), perspective (20%), color jitter (brightness $< 50\%$, contrast $< 50\%$, saturation $< 50\%$, hue $< 30\%$), and blur ($0 - 20\%$) applied. When training, randomly selected images from the training image set are also grouped into batches using the text in the Perseus dataset to randomly pick glyph images from each class in the order they appear in the ancient Greek language dataset. Models are then trained with dropout $0 - 50\%$ and ReLU layers between the densely-connected layers.

7.5.4 Stochastic Language Models

Using Markov chains, more traditional language models are also trained on the Perseus dataset. Based on the occurrence of glyphs in the dataset, Markov chains are generated to be used in predicting the occurrence of glyphs after or before other glyphs, using log-probabilities to minimize the number of floating-point math errors. These language models were attached to the CNN classifiers by adding their probabilities together using weighted combinations and by utilizing the probabilities to manage uncertainty in the classifiers. In the weighted combinations, probabilities are weighted by a factor, α, such that the total probability of the n^{th} glyph (g_n) is given by

$$\mathbb{P}(g_n) = \frac{\mathbb{P}_{cnn}(g_n) + \alpha \mathbb{P}_{markov}(g_n)}{\sum_{i=0}^{n} \mathbb{P}_{cnn}(g_n) + \alpha \mathbb{P}_{markov}(g_n)} \tag{7.6}$$

such that both α and g_n are in the range $[0, 1]$.

Uncertainty in CNN classification can be resolved using two methods with the language models. The first option is to consider the top n choices of a CNN and then utilize the above method to interpolate the probabilities of the CNN and language models together to generate new probabilities for the top n choices of the CNN. This method ignores the exact certainty of the classifier for the top n choices. The other method is to set an uncertainty threshold u such that the top k choices of the CNN with probabilities within u of each other are considered. The class fis then chosen from the top k glyphs using the probabilities given by the language model. This method takes the exact certainty of the classifier into account, and only breaks ties between classes that the classifier thinks are nearly equally likely, with an assumed uncertainty of u in the classifier.

Chapter 8

Evaluation

8.1 Experiment Design

To ensure experimental validity, all intermediate evaluation and hyperparameter tuning was performed on 23 randomly selected images that were separate from both the training image set of 106 images and the final test set of 23 images. This split generated 27,623 glyph images for training with 7,759 glyph images being available for evaluation and 6,835 glyph images for testing. All examples and images, and some intermediate evaluations, are taken from the evaluation set, with final metrics being taken on the test set. No results are taken from the training set, as they would have little meaning.

8.2 Existing Pipelines

To attempt to generate a baseline to assess new methods against, Tesseract [42], a popular OCR engine sponsored by Google [47] and Ocular [9], another OCR engine based on stochastic models, were integrated into the pipeline using Pytesseract [33] and by calling jar files from python respectively.

8.2.1 Evaluation Metrics

As these existing pipelines were assessed in the early phases of the project, a full ground truth to evaluate them on was not yet generated as the bounding of lines was not fully implemented. Therefore, they were assessed on how well they were able to detect the quantity of glyphs in an image and how visually close the transcriptions looked when the images were looked at by a human. These metrics proved to be enough to evaluate the methods as both methods were far enough away from an acceptable solution that a more exact metric for comparison was not needed.

8.2.2 Tesseract

With Tesseract, using the ancient Greek corpus, and using both raw and binarized images, the output was comparable to noise, with the transcription lengths being an order of magnitude different from the ground truths.

8.2.3 Ocular

Similarly, training Ocular on both raw and binarized full images and individual glyph images, the output was comparable to noise, with the transcription lengths

being an order of magnitude different from the ground truths and A and Λ making up over 95% of the returned glyphs. This result happened regardless of the dataset that Ocular was trained on, with binarized and non-binarized images performing the same and singular glyph images performing the same as whole images.

8.3 Binarization

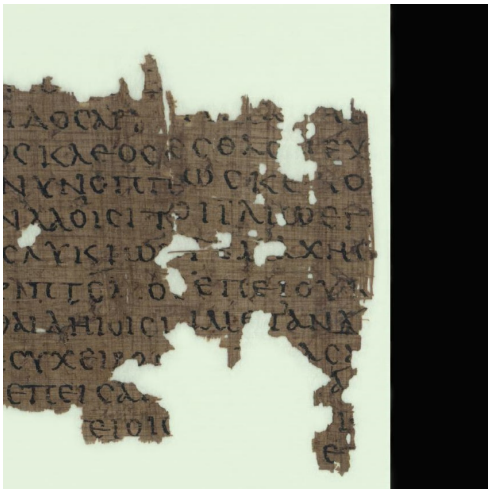
8.3.1 Evaluation Metrics

As there was no ground-truth information provided to compare binarizations against, visual comparison was utilized to determine which binarization method was providing the highest quality binarizations. Evaluation was performed on four different images, which contained various features that would allow for quick visual evidence of the effectiveness of the methods (Figure 8.1). Images from the training image set were utilized in hyperparameter optimization for the binarization methods, which included generating binarizations of more than four images. The four images are of assorted sizes, from different collections, and are degraded to different degrees, allowing them to function as a valuable quick assessment tool without introducing bias into the evaluation of the methods using visually approximated signal-to-noise ratio as a metric.

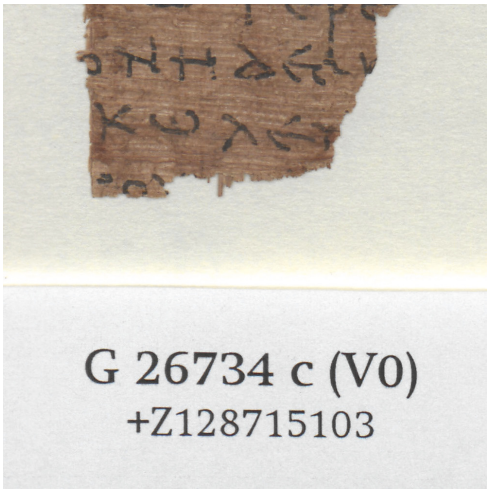
To assess each binarization method, these images, containing a total of approximately 300 glyphs, were used to determine how much of the ink on the papyrus was being included, and if noise was being added to the image in the form of background pixels being black or ink pixels being white. This metric was designed to ensure that the signal-to-noise ratio would be low and that the recall and precision would be high, even if these metrics were not directly measurable without an expert-generated ground truth for each pixel.

FIGURE 8.1: Four Images Utilized to Visually Assess Binarization Quality

(A) Example File A

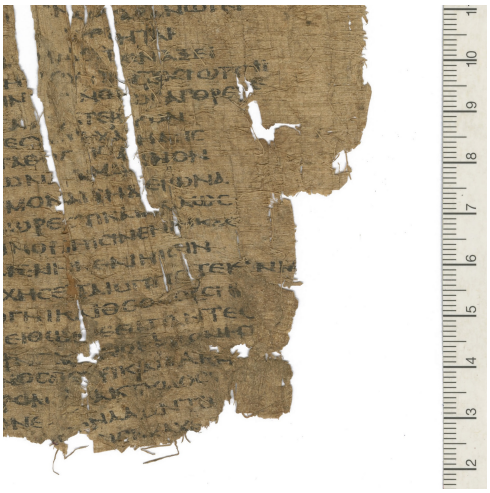


(B) Example File B



(C) Example File C

(D) Example File D



The four images used to assess binarization methods, cropped to squares so that are easy to tile to make it easy to assess visually.

8.3.2 Evaluation of Methods

The results of k-means clustering on the evaluation images are included for reference (Figure 8.2). These images clearly show that the signal-to-noise ratio is far too high to be utilized as the general binarization solution for the pipeline, as the approximated signal-to-noise ratio is sometimes less than 1, with significantly more noise than signal being generated.

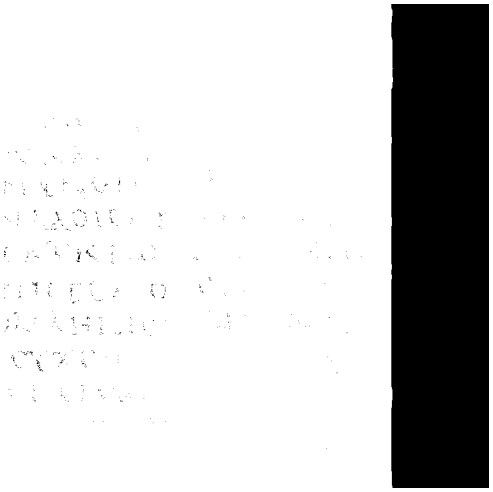
The results generated by clustering can be directly compared to those of a neural network to see how poor they are. The results generated by DP-Linknet (Figure 8.3) are visibly much better at including whole glyphs and less of the noise included in the image, especially the papyrus noise.

Using the results of the Gabor wavelet approximation (Figure 8.4), a binary mask can be created (Figure 8.5) using DP-Linknet. This works using the assumption that large amounts of noise in the binary images generated by DP-Linknet are due to intentionally added objects in the original images, which are often more regular in

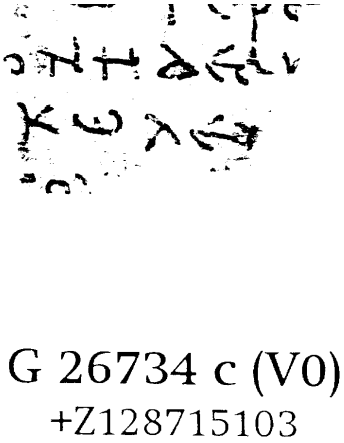
color or shape than the glyphs. This means the Gabor-filtered images do not lose this noise, allowing the CNN to pick up on this information to create an image that can be used as a mask to remove these details from the final binarizations (Figure 8.6).

FIGURE 8.2: Four Example Binarizations Generated with Clustering

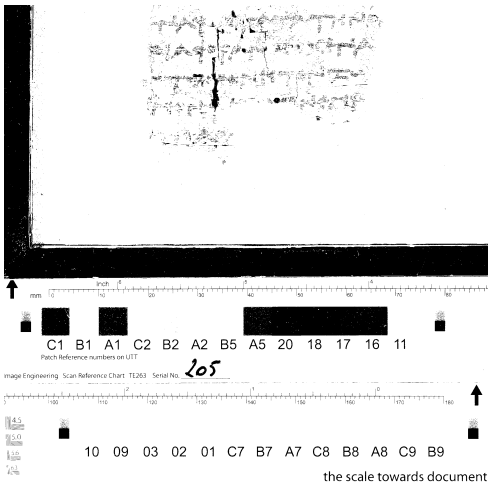
(A) Example File A



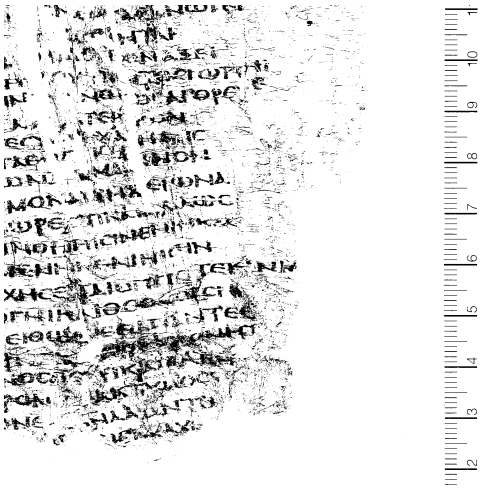
(B) Example File B



(C) Example File C



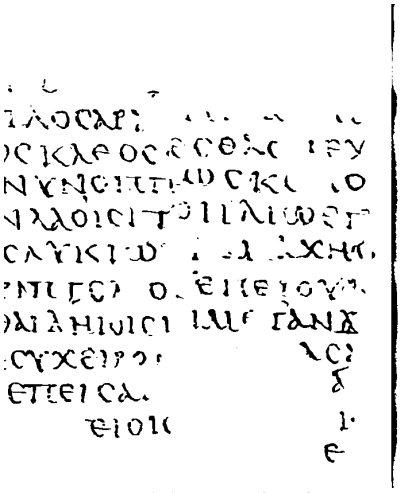
(D) Example File D



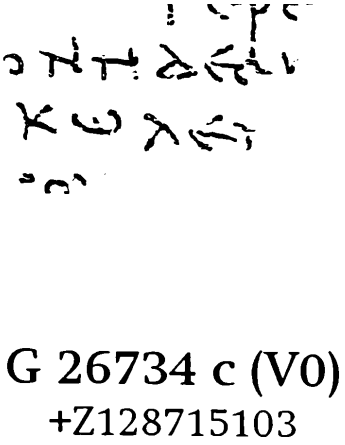
The four images were used to assess binarization methods, binarized using k-means with $k = 5$. Image A has a poor signal-to-noise ratio. Image B has a much better signal-to-noise ratio for the glyphs, but also contains text that should not be included near the bottom of the image. Images l and D also contain glyph-signal, but are both noisy around the glyphs and include non-glyph information such as a ruler in Image D and the border and color scale in image C.

FIGURE 8.3: Four Example Binarizations Generated with DP-Linknet

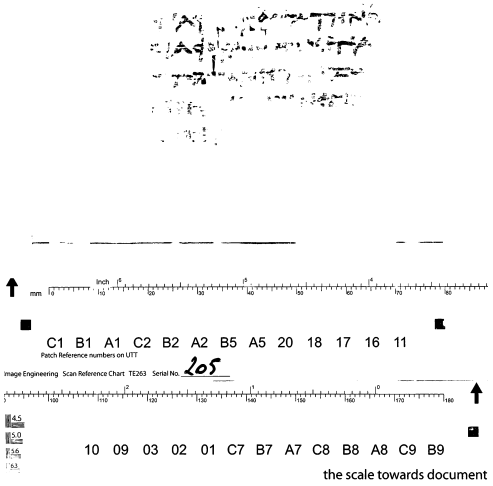
(A) Example File A



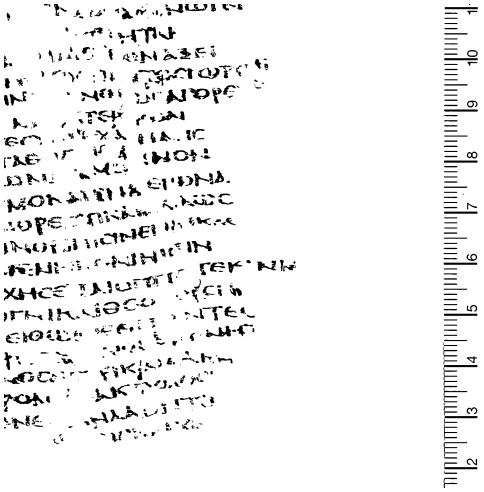
(B) Example File B



(C) Example File C



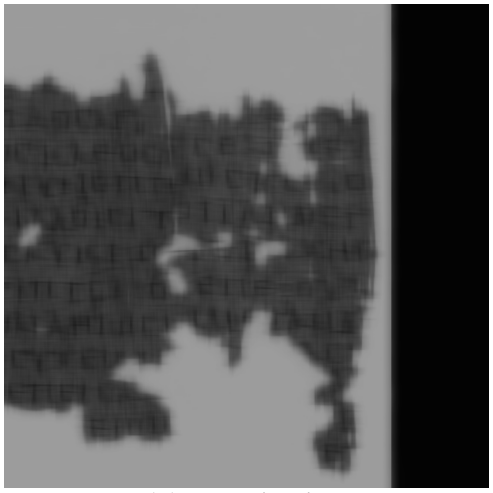
(D) Example File D



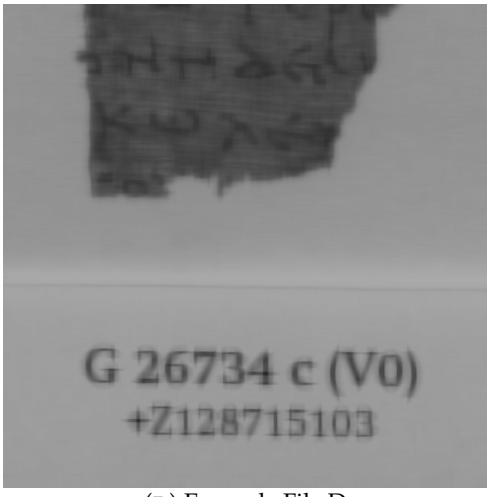
The four images used to assess binarization methods, binarized using DP-Linknet [51] with confidence value of 2. Image A has clearly defined glyphs, especially when compared to the results of clustering. Image B has a similar signal-to-noise ratio for the glyphs to the clustering, with the extra text near the bottom of the image still being included. Images l and D also contain glyph-signal, and are less noisy around the glyphs and include non-glyph information such as a ruler in Image D and the border and color scale in image C.

FIGURE 8.4: Four Example Gabor-Filtered Images

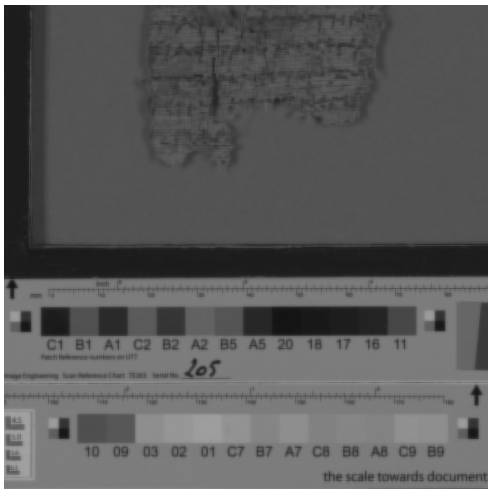
(A) Example File A



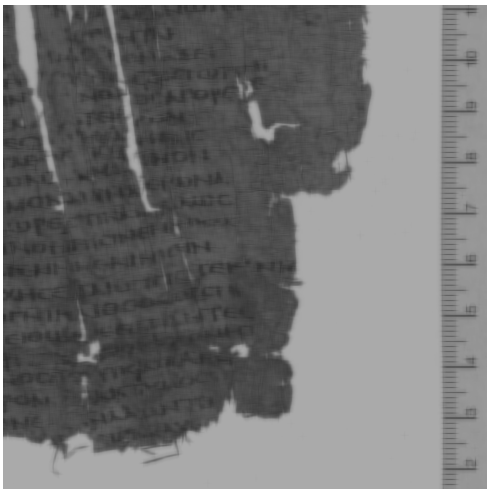
(B) Example File B



(C) Example File C



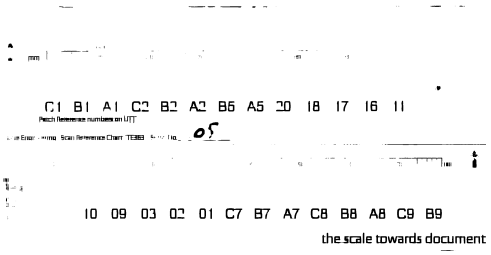
(D) Example File D



The four images used to assess binarization methods, passed through the Gabor wavelet approximation function.

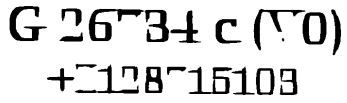
FIGURE 8.5: Four Example Gabor Wavelet Binarized Images

(A) Example File A



(B) Example File B

(C) Example File C

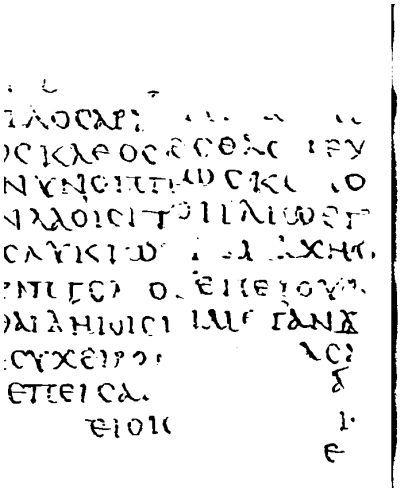


(D) Example File D

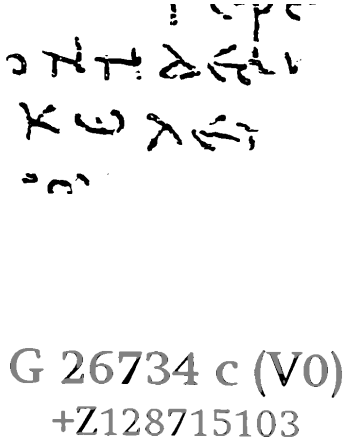
The four images used to assess binarization methods, passed through the Gabor wavelet approximation function and then passed through DP-Linknet to create a rudimentary mask that contains some non-glyph information.

FIGURE 8.6: Four Example CNN Binarized Images (Masked)

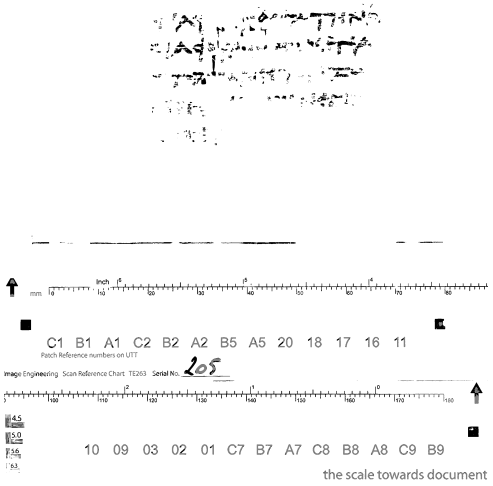
(A) Example File A



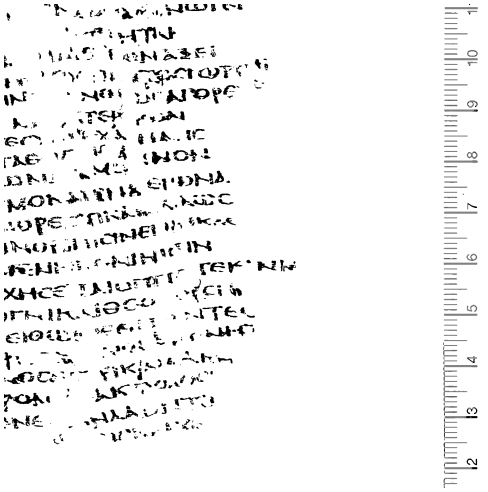
(B) Example File B



(C) Example File C



(D) Example File D



The four images used to assess binarization methods, binarized using DP-Linknet [51] with confidence value of 2 and then masked by the binarizations of the Gabor-filtered images with the removed black parts of the image shown in grey. No changes were made to image A, as there was no detail that the CNN could recognize left in the Gabor-filtered image. The extra text from image B is mostly removed, with the glyph information intact. Similarly, images l and D have their glyph information intact, with the artificial noise in the image being partially removed.

8.4 Glyph Bounding

8.4.1 Evaluation Metrics

As the ground-truth information provided for bounding boxes is not tight, perfectly accurate metrics on bounding are not possible without manually tightly rebounding each glyph. Therefore, IOU is used, along with glyph precision and recall measuring the approximate efficacy of methods, with the caveat that a method may have a lower IOU if it generates tighter bounding boxes than if the bounding box was less tight.

Precision and recall are measured on glyphs, ignoring the exact pixels and sizes of the bounding boxes. A prediction is defined as a match for a known bounding box if the IOU between the two bounding boxes is greater than zero. Each ground truth bounding box is then paired with the matched prediction bounding box with the highest IOU value such that no bounding box is paired with more than one other bounding box. The mean IOU is also recorded for all true positives as a method to determine how close the prediction bounding boxes are to the ground truth values.

FIGURE 8.7: Test Results for Glyph Bounding Methods

Bounding Method	Precision	Recall	F1 Score	Mean IOU
Connected Components (CC)	0.2006	0.8923	0.3276	0.3182
CC w/ Best Cleaning	0.2335	0.8093	0.3624	0.3142

8.4.2 Connected Components

Glyph bounding using connected components fails to precisely generate bounding boxes due to the flaws in the binarization methods used. Otherwise, the method can tightly bound boxes with a high amount of recall. By removing regions with centroids in other regions or by removing regions fully inside other regions (which includes the before method), the precision and F1 score can be increased, at the cost of recall (Figure 8.8).

FIGURE 8.8: Test Results for Connected Component Bounding and Cleaning

Cleaning Method	Precision	Recall	F1 Score	Mean IOU
None	0.2006	0.8923	0.3276	0.3182
Internal Centroid Removal	0.2256	0.8124	0.3531	0.3145
Internal Region Removal	0.2335	0.8093	0.3624	0.3142

8.4.3 Region-Based Convolution Neural Networks

YOLO, when trained on color images, performs better than when trained on binarized images. In addition, by training YOLO to recognize all glyphs as one class instead of each glyph class as its own class the accuracy can be significantly increased (Figure 8.9). In addition, networks were trained with and without freezing the backbone layers, which are the layers defined as the initial convolutional layers (layers 0-9) [**YOLOBackbone**] that process the input before the main convolutions are performed. This allowed for faster training with a potential for lost accuracy.

The accuracy metrics taken for each YOLO model were generated on the evaluation set to ensure that the final test set evaluation could be done on unseen data. Therefore, the classification metrics in this sub-section will not map directly to the results of the same classification methods when utilized in the final pipeline.

FIGURE 8.9: Evaluation Results for YOLO Bounding

Training Set	Backbone Layers	Image Type	Model Size	F1 Score
Multi-Class	Frozen	Color	YOLOv5x (largest)	0.07
Mono-Class	Frozen	Binary	YOLOv5l (large)	0.63
Mono-Class	Frozen	Color	YOLOv5l (large)	0.68
Mono-Class	Frozen	Color	YOLOv5x (largest)	0.69
Mono-Class	Unfrozen	Color	YOLOv5x (largest)	0.74
Mono-Class	Unfrozen	Color	YOLOv5x (largest)	0.72*

All the models were trained for 300 epochs, with the exception of the Model with a * next to the F1 score, which was trained for 600 epochs, with the F1 score being taken on only the last 300 epochs.

RCCN Sliding Window

By using the model with the highest base F1 score and introducing a sliding window to the RCNN, the peak F1 score can be increased. This effect is amplified by introducing an inner sliding window, which reduces the recall for the benefit of more precision (Figure 8.10).

FIGURE 8.10: Evaluation Results for Sliding Window Assisted YOLO Bounding

Inner Window	Type	Precision	Recall	F1 Score
False	Mean	0.934	0.851	0.874
False	Max F1 Score	0.926	0.936	0.931
True	Mean	0.953	0.830	0.870
True	Max F1 Score	0.929	0.942	0.936

All results are taken with a sliding window of 800 pixels and a step size of 200 pixels. The results are averaged over the confidence thresholds t required for the bounding boxes to be kept, where $0.1 \leq t \leq 0.85$ with a step size of 0.05.

RCNN Duplicate Removal

YOLO generates a confidence value for each bounding box. Using these confidence levels, ties between overlapping bounding boxes are broken where bounding boxes are defined as having an IOU above a threshold. Using this method to break ties, in addition to a minimum confidence level for bounding boxes, duplicates can be much more effectively removed, increasing the maximum F1 score of the bounding boxes (Figure 8.11).

FIGURE 8.11: Evaluation Results for Maximal F1 Score for Sliding Window Duplicate Removal

Inner Window	Confidence	Duplicate IOU	Precision	Recall	F1 Score
False	0.47	0.33	0.940	0.928	0.934
True	0.29	0.45	0.931	0.944	0.937

All results are taken with a sliding window of 800 pixels and a step size of 200 pixels. The maximum F1 score values were found by narrowing the hyperparameter search window and decreasing step size until the 3 significant figures of the F1 score no longer changed.

Binarization-Assisted CNN

Using the binary images to tightly crop the YOLO-generated bounding boxes to the ink generates bounding boxes that have a mean IOU of 71.5% compared to a mean IOU of 73.0% without cropping the bounding boxes. This difference in bounding box size and shape results in a difference of less an a 0.01% in the mean precision, recall, and F1 scores. Cropping bounding boxes using the binarized images also reduces the mean classification F1 score by 1.8%.

8.5 Line Bounding

As in glyph bounding, no ground truth information was provided for which glyphs where in the same line, so a visual approximation must be used to determine which methods are best, at least when comparing results of each metric without taking the rest of the pipeline into account.

8.5.1 Point-Of-Interest Clustering

Using point-of-interest clustering fails to generate valid lines as the noise in both the binarized and color images preventing the glyphs from being clustered correctly, which causes higher frequency noise to be made into lines, instead of the glyphs. Given more time, the application of de-noising techniques could be explored to make this method more effective.

8.5.2 Adjacent and Overlapping Glyph Clustering

Grouping glyph bounding boxes into lines based on adjacency and overlapping produces significantly better results than would otherwise be generated, with this method improving the mean classification F1 score by 2.6% on the best performing LSTM-based CNN classifier.

8.5.3 Line-Based Bounding Box Cropping

Using line-based bounding box cropping to prevent bounding boxes from overlapping has a negative effect on glyph classification accuracy, but a small positive effect on bounding accuracy (Figure 8.12). When combined with bounding box cropping, this effect is amplified, further reducing classification and bounding box accuracies.

FIGURE 8.12: Evaluation Results for Bounding Box Intersection Removal on Bounding Box and Classification F1 Scores

Intersection Removed	Tight Bounding Boxes	Bounding Box F1 Score	Classification F1 Score
True	True	0.872	0.797
True	False	0.873	0.807
False	True	0.872	0.805
False	False	0.872	0.809

8.5.4 Line Variation Generation

Using lines to generate bounding box variations produced negative results on both the bounding and classification accuracy of the pipeline (Figure 8.13). Variations are made based on the likelihood of the classifications, which, depending on the classifier used, could lead to worse line variants being accepted over the original line. This is because partial glyphs, such as the two halves of a Π, may closely resemble and be classified as other glyphs, such as a Γ and a T. If this happens, then the line variant with the split Π could be considered more likely, especially if the Π is poorly formed or has noise in the upper right corner, for example.

FIGURE 8.13: Evaluation Results for Line-Based Bounding Box Variation

Line Variants Generated	Bounding F1 Score	Classification F1 Score	Combined F1 Score
True	0.885	0.795	0.707
False	0.910	0.811	0.741

8.6 Classification

The accuracy metrics taken for the classification methods were generated on the evaluation set to ensure that the final test set evaluation could be done on unseen data. Therefore, the classification metrics in this section will not map directly to the results of the same classification methods when utilized in the final pipeline.

8.6.1 Transfer Learning

Using the feature vectors generated with AlexNet on the non-NN methods, the best results were obtained using the binarizations of the images generated by DP-Linknet. GMMs, Random Forests, and K-Nearest Neighbors (k-NN) classifiers were generated with the full color and binarized images and Euclidean distance calculations (Equation 7.2), with the best results for each method being shown (Figure 8.14), where F1 score is the average of the per-glyph F1 score weighted by glyph class occurrence frequency.

FIGURE 8.14: Evaluation Results for Euclidean Non-NN Transfer Learning Classification

Classification Method	Color F1 Score	Binary F1 Score
GMM	0.07	0.08
Random Forest	0.11	0.19
KNN	0.08	0.14

By generating a new set of training data by training the classifications on exactly one randomly chosen template for each class and footmark type, the results were mostly unchanged, but were significantly increased by manually cleaning the templates to remove noise and other small imperfections in the characters (Figure 8.15). These results were generated using only a k-NN classifier as it was the fastest to train and did not require as many example images to train an accurate model. These results show that the k-NN classifier can be trained on very few images and generate comparable results to a dataset that is magnitudes larger, which shows promise for classification on similar problems with significantly less training data.

FIGURE 8.15: Evaluation Results for Templated k-NN Transfer Learning Classification

Template Type	Distance	Best k Value	Weighted F1 Score
Color	Euclidean	5	0.08
Cleaned Binary	Euclidean	5	0.27
Cleaned Binary	Manhattan	4	0.34
Cleaned Binary	Cosine	5	0.39

Using pretrained neural networks to classify the data generated better results, both on binarized and color images of glyphs (Figure 8.16). Using these networks a higher classification accuracy was attainable with unaltered training data, producing better results with less manual work being required.

FIGURE 8.16: Evaluation Results for NN Transfer Learning Classification

Network	Image Type	Weighted F1 Score
Linear B.1	Binary	0.666
Linear B.1	Color	0.626
Simple-LSTM B.2	Color	0.593
LSTM-to-Linear B.3	Color	0.662
Linear-to-LSTM B.4	Binary	0.710
Linear-to-LSTM B.4	Color	0.668

8.6.2 Convolutional Neural Networks

By training the convolutional layers of AlexNet [32], ResNeXt [50], and MNIST9975 [16], the accuracy of the classifiers were able to be further increased. The convolutional network based on AlexNet generates similar results to those generated by the classifiers based on the feature vectors generated by AlexNet, but has a higher weighted F1 score. The networks based on ResNeXt perform even better, at the cost of time, taking much longer to train and evaluate (Figure 8.17). In these larger,

more complex models, the binarized images performed worse than the color images across all the models, so only the peak color image F1 score is used for evaluation. Each network was trained and evaluated for approximately 100 epochs on the same train and evaluation image and word sets.

FIGURE 8.17: Evaluation Results for CNN Classification

Base Network	Modified Network Name	Peak Weighted F1 Score
AlexNet	AlexNet LSTM	0.754
ResNeXt50	ResNeXt50 Simple-LSTM	0.786
ResNeXt50	ResNeXt50 Classify-to-LSTM	0.700
ResNeXt50	ResNeXt50 LSTM-To-Tall-Linear	0.835
ResNeXt50	ResNeXt50 LSTM-To-Linear	0.836
MNIST9975	MNISTCNN (Black Padding)	0.634
MNIST9975	MNISTCNN (White Padding)	0.644
MNIST9975	MNISTCNN-LSTM (White Padding)	0.716
MNIST9975	MNISTCNN-Deep-LSTM (White Padding)	0.794

8.6.3 Stochastic Language Models

Utilizing a trained ResNeXt50 CNN with no LSTM layers to evaluate the stochastic language models resulted in results consistently worse than those of any of the CNNs with an LSTM. By evaluating all parameters α in $[0, 1]$ with a step size of 0.05 and then 0.01 around the peak, the best value for α was 0.03 with an F1 score of 0.681, compared to the baseline F1 score of 0.680 using an α of 0. With an α of 1, the F1 score dropped even further to 0.636. In addition, the language models were used to break ties in LSTM-based CNNs, which resulted in similar results when breaking ties between the top n choices of the CNN (Figure 8.18) and when breaking ties between the top choices when there was a difference between the top choices of less than a set uncertainty threshold u (Figure 8.19). Therefore, it can be concluded that while a stochastic language model can be utilized to augment a non-LSTM classifier, a single LSTM layer generates results far better without requiring extra algorithmic steps in the classification process. However, utilizing an additional language model to assist the LSTM-based classifier could potentially increase accuracy in approximately 0.02% of the classifications.

FIGURE 8.18: Evaluation Results for Top n Tie-Breaking Using Stochastic Language Models

n Choices Considered	Weighted F1 Score
1	0.832
2	0.632
3	0.479

FIGURE 8.19: Evaluation Results for Uncertainty Tie-Breaking Using Stochastic Language Models

Uncertainty Threshold	Weighted F1 Score
0.0	0.8320
0.01	0.8322
0.02	0.8319
0.03	0.8318
0.04	0.8316
0.05	0.8317

8.6.4 Neural Network Language Models

As is partially demonstrated in Figures 8.16 and 8.17, adding a LSTM layer to the classifiers increases the accuracy over a classifier without the ability to remember. This can also be shown by shuffling the glyphs to make random sequences of glyphs instead of the expected glyph orderings patterns that the language models can learn (Figure 8.20). In addition, the number of glyphs in a line has a significant effect on classification accuracy, with longer lines (larger batch size) generating a higher average F1 score than shorter lines (smaller batch size) (Figure 8.21).

FIGURE 8.20: Evaluation Results for Shuffled Batches

Shuffled	n	Top n Accuracy
True	1	0.761
True	2	0.844
True	3	0.889
True	4	0.909
True	5	0.923
False	1	0.818
False	2	0.902
False	3	0.929
False	4	0.950
False	5	0.959

FIGURE 8.21: Evaluation Results for the Effect of Batch Size on Weighted F1 Score

Batch Size	Precision	Recall	F1 Score
1	0.772	0.772	0.772
2	0.792	0.793	0.792
3	0.805	0.802	0.799
4	0.821	0.806	0.803
5	0.837	0.811	0.806

8.6.5 Training on Tightly Bound Glyphs

Using tightly bound glyphs, and the un-modified bounding boxes generated by YOLO generated better results on average for the CNNs based on the MNIST9975 network, but generated worse results for the larger CNNs. This is likely due to the input size of the larger CNNs favoring images with more resolution, especially as

they accept images that have been cropped, while the MNIST9975 networks accept padded images. This difference in input makes direct numerical comparison relatively meaningless, as it introduces too many extra factors to the comparison, such as input resolution, padding color, cropping rules, etc.

8.7 Pipeline

Using the product of the classification F1 score and the bounding F1 score we can get a metric for the pipeline that, when optimized, generates hyperparameters that favor bounding and classification equally. Using this metric, the best results on the evaluation data were generated with a bounding box confidence threshold of 0.635 and a duplicate threshold of 0.185 without using an inner sliding window, cropping the bounding boxes to the binarized images, or removing the intersecting regions between bounding boxes. These parameters were used on the unseen testing set to generate evaluation results for the entire pipeline as a whole (Figure 8.22).

FIGURE 8.22: Final Pipeline Evaluation and Test Results

Data Set	Task	Precision	Recall	Weighted F1 Score
Evaluation	Bounding	0.963	0.869	0.914
Test	Bounding	0.958	0.811	0.878
Evaluation	Classification	0.84	0.833	0.832
Test	Classification	0.822	0.815	0.814

Chapter 9

Conclusion

In conclusion, an automatic pipeline for annotating glyphs on papyrus can help to more accurately and quickly interpret the texts in which they appear and to learn more about the history and beliefs of these ancient civilizations. Additionally, digitally annotating glyphs can also aid in the preservation of these fragile documents. This will allow future generations to study and appreciate their significance and for more automatic tools to have access to these records, which could further expand the ability to research these sources. Overall, the process of annotating glyphs on papyrus is an important and valuable endeavor for anyone interested in learning more about the ancient civilizations of the past.

Chapter 10

Proposed Future Work

Further research on automatically annotating glyphs on papyrus could focus on three different areas. One potential direction for future study could be to develop more sophisticated methods for binarization by using an RCNN-type approach to locate and bound the glyphs before utilizing binarizing techniques on them. This could decrease the amount of noise that would need to be managed by the binarization and could allow for a higher signal-to-noise ratio in the resultant binarizations. Composite images could then be reconstructed from these glyph binarizations, which would allow later parts of the pipeline to process complete images as normal.

Another area for further research could be to expand the scope of the study to include more diverse examples of glyphs and papyrus texts. This could involve examining a wider range of texts from different regions or time periods or studying texts in different languages or writing systems. This could allow for a more robust glyph bounding system and may allow for more features of the glyphs to be classified, such as diacritics, case, and preservation level.

Finally, existing texts could be utilized further by allowing the pipeline to find similar words or full lines of text to the ones being annotated to correct glyph classification errors.

Appendix A

File Fragments

FIGURE A.1: Minimal Example of the CSV Output Format

```
glyph,  certainty, min_x,  min_y,  max_x,  max_y,        image_path
     Δ,      0.253,   100,     50,     20,     30, images/image.png
```

The minimum required information for a single image with a single instance of the Δ glyph in the CSV output file format. The bounding boxes in this format are output in the Pascal VOC bounding box format (Figure 7.2), in which bounding boxes are described by their maximum and minimum x and y coordinates.

FIGURE A.2: Minimal Example of the COCO Annotation Format

```
{
  "annotations": [
    {
      "area": 600,
      "bbox": [
        100,
        50,
        20,
        30
      ],
      "category_id": 186,
      "id": 1,
      "image_id": 1,
      "iscrowd": 1
      "tags": {...}
    }
  ],
  "categories": [
    {
      "id": 186,
      "name": "Δ",
      "supercategory": "Greek"
    }
  ],
  "images": [
    {
      "bln_id": 1,
      "date_captured": null,
      "file_name": "images/image.png",
      "img_url": "images/image.png",
      "width": 1000,
      "height": 2000,
      "id": 1,
      "license": 2
    },
  ],
  "licenses": [
    {
      "id": 2,
      "name": "CC-BY-NC 3.0.",
      "url": "https://creativecommons.org/licenses/by/3.0/"
    },
  ]
}
```

The minimum required information for a single image with a single instance of the Δ glyph in the COCO file format. The bounding boxes in this format are output in the COCO bounding box format (Figure 7.3), in which bounding boxes are described by their minimum x and y coordinates and their width and height.

Appendix B

Neural Network Architecture

The following figures are psuedocode simplifications of the implementations of the non-convolutional neural networks utilized in the project. The full code for these networks is supplied in the github repo [11] of the project, but these should act as quick references for the differences in depth and layers between the networks.

FIGURE B.1: Linear Classifier

```
Linear(9216, 200),
ReLU(),
Dropout(p=.1),
Linear(200, 200),
ReLU(),
Dropout(p=.1),
Linear(200, 100),
ReLU(),
Dropout(p=.1),
Linear(100, 24)
```

FIGURE B.2: Simple LSTM Classifier

```
self.lstm = LSTM(9216, 50, num_layers=1, bidirectional=True),
Dropout(p=.1),
Linear(100, 24)
```

FIGURE B.3: Linear-to-LSTM Classifier

```
Linear(9216, 100),
ReLU(),
Dropout(p=.1),
Linear(100, 100),
ReLU(),
Dropout(p=.1),
self.lstm = LSTM(100, 100, num_layers=4, bidirectional=True),
ReLU(),
Linear(200, 24)
```

FIGURE B.4: LSTM-to-Linear Classifier

```
LSTM(9216, 100, num_layers=4, bidirectional=True),
ReLU(),
Dropout(p=.1),
Linear(200, 100),
ReLU(),
Linear(100, 24)
```

Appendix C

Convolutional Neural Network Architecture

The following figures are psuedocode simplifications of the implementations of the convolutional neural networks utilized in the project. The full code for these networks is supplied in the github repo [11] of the project, but these should act as quick references for the differences in depth and layers between the networks.

FIGURE C.1: AlexNet LSTM Classifier

```
Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
ReLU(),
MaxPool2d(kernel_size=3, stride=2),
Conv2d(64, 192, kernel_size=5, padding=2),
ReLU(),
MaxPool2d(kernel_size=3, stride=2),
Conv2d(192, 384, kernel_size=3, padding=1),
ReLU(),
Conv2d(384, 256, kernel_size=3, padding=1),
ReLU(),
Conv2d(256, 256, kernel_size=3, padding=1),
ReLU(),
MaxPool2d(kernel_size=3, stride=2),
self.avgpool = AdaptiveAvgPool2d((6, 6))
Dropout( bidirectional.1),
Linear(256 * 6 * 6, 4096),
ReLU(),
Dropout( bidirectional.1),
Linear(4096, 4096),
ReLU(),
Linear(4096, 2048),
LSTM(2048, 24, bidirectional=True)
Linear(48, 24)
```

FIGURE C.2: ResNeXt Simple-LSTM Classifier

```
self.resnext = resnext50_32x4d(ResNeXt50_32X4D_Weights.IMAGENET1K_V2)
self.resnext.fc = LSTM(self.resnext.fc.in_features, 24, bidirectional=True)
self.classifier = Linear(48, 24)
```

FIGURE C.3: ResNeXt50 Linear-to-LSTM Classifier

```
self.resnext = resnext50_32x4d(ResNeXt50_32X4D_Weights.IMAGENET1K_V2)
self.resnext.fc = Sequential(
    Linear(self.resnext.fc.in_features, 200),
    Dropout(0.1),
    ReLU(),
    Linear(200, 100),
    Dropout(0.1),
    ReLU(),
    LSTM(100, 48, bidirectional=True)
),
Linear(48, 24)
```

FIGURE C.4: ResNeXt50 LSTM-to-Linear Classifier

```
self.resnext = resnext50_32x4d(ResNeXt50_32X4D_Weights.IMAGENET1K_V2)
self.resnext.fc = LSTM(self.resnext.fc.in_features, 48, bidirectional=True)
Dropout(.2),
ReLU(),
Linear(96, 96),
Dropout(.2),
ReLU(),
Linear(96, 24)
```

FIGURE C.5: ResNeXt50 LSTM-to-Tall-Linear Classifier

```
self.resnext = resnext50_32x4d(ResNeXt50_32X4D_Weights.IMAGENET1K_V2)
self.resnext.fc = LSTM(self.resnext.fc.in_features, 96, bidirectional=True)
Dropout(0.2),
ReLU(),
Linear(192, 96),
Dropout(0.2),
ReLU(),
Linear(96, 24)
```

FIGURE C.6: MNIST9975 CNN Classifier

```
Conv2d(3, 32, 3, padding=1),
ReLU(),
BatchNorm2d(32),
Conv2d(32, 32, 3, padding=1),
ReLU(),
BatchNorm2d(32),
Conv2d(32, 32, 3, stride=2, padding=1),
ReLU(),
BatchNorm2d(32),
MaxPool2d(2, 2),
Dropout(.5),
Conv2d(32, 64, 3, padding=1),
ReLU(),
BatchNorm2d(64),
Conv2d(64, 64, 3, padding=1),
ReLU(),
BatchNorm2d(64),
Conv2d(64, 64, 3, stride=2, padding=1),
ReLU(),
BatchNorm2d(64),
MaxPool2d(2, 2),
Dropout(.5),
Conv2d(64, 128, 3, padding=1),
ReLU(),
BatchNorm2d(128),
MaxPool2d(2, 2),
Dropout(.5),
Linear(128, 24)
```

FIGURE C.7: MNIST9975 LSTM Classifier

```
Conv2d(3, 32, 3, padding=1),
ReLU(),
BatchNorm2d(32),
Conv2d(32, 32, 3, padding=1),
ReLU(),
BatchNorm2d(32),
Conv2d(32, 32, 3, stride=2, padding=1),
ReLU(),
BatchNorm2d(32),
MaxPool2d(2, 2),
Dropout(.5),
Conv2d(32, 64, 3, padding=1),
ReLU(),
BatchNorm2d(64),
Conv2d(64, 64, 3, padding=1),
ReLU(),
BatchNorm2d(64),
Conv2d(64, 64, 3, stride=2, padding=1),
ReLU(),
BatchNorm2d(64),
MaxPool2d(2, 2),
Dropout(.5),
Conv2d(64, 128, 3, padding=1),
ReLU(),
BatchNorm2d(128),
MaxPool2d(2, 2),
Dropout(.5),
LSTM(128, 24, bidirectional=True),
nn.Linear(48, 24)
```

FIGURE C.8: MNIST9975 LSTM-to-Linear Classifier

```
Conv2d(3, 32, 3, padding=1),
ReLU(),
BatchNorm2d(32),
Conv2d(32, 32, 3, padding=1),
ReLU(),
BatchNorm2d(32),
Conv2d(32, 32, 3, stride=2, padding=1),
ReLU(),
BatchNorm2d(32),
MaxPool2d(2, 2),
Dropout(.5),
Conv2d(32, 64, 3, padding=1),
ReLU(),
BatchNorm2d(64),
Conv2d(64, 64, 3, padding=1),
ReLU(),
BatchNorm2d(64),
Conv2d(64, 64, 3, stride=2, padding=1),
ReLU(),
BatchNorm2d(64),
MaxPool2d(2, 2),
Dropout(.5),
Conv2d(64, 128, 3, padding=1),
ReLU(),
BatchNorm2d(128),
MaxPool2d(2, 2),
Dropout(.5),
LSTM(128, 24, bidirectional=True),
nn.Linear(48, 48),
nn.Linear(48, 48),
nn.Linear(48, 24)
```

Bibliography

[1] Nov. 2022. URL: https://opencv.org/.

[2] Hassan Althobaiti and Chao Lu. "A survey on Arabic Optical Character Recognition and an isolated handwritten Arabic Character Recognition algorithm using encoded freeman chain code". In: *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. 2017, pp. 1–6. DOI: 10.1109/CISS.2017.7926062.

[3] *Ancient Egyptian Medical Papyri*. URL: http://indigo.ie/~marrya/papyri.html.

[4] Vlad Atanasiu and Isabelle Marthot-Santaniello. "Personalizing image enhancement for critical visual tasks: improved legibility of papyri using color processing and visual illusions". In: *International Journal on Document Analysis and Recognition (IJDAR)* 25.2 (2022), pp. 129–160.

[5] Itay Bar-Yosef. "Input sensitive thresholding for ancient Hebrew manuscript". In: *Pattern Recognition Letters* 26 (June 2005), pp. 1168–1173. DOI: 10.1016/j.patrec.2004.07.014.

[6] Itay Bar-Yosef et al. "Binarization, character extraction, and writer identification of historical Hebrew calligraphy documents". In: *International Journal of Document Analysis and Recognition (IJDAR)* 9 (Apr. 2007), pp. 89–99. DOI: 10.1007/s10032-007-0041-5.

[7] Daniel Stökl Ben Ezra et al. "Transcription Alignment for Highly Fragmentary Historical Manuscripts: The Dead Sea Scrolls". In: *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2020, pp. 361–366. DOI: 10.1109/ICFHR2020.2020.00072.

[8] Suman Kumar Bera et al. "A Non-Parametric Binarization Method Based on Ensemble of Clustering Algorithms". In: *Multimedia Tools Appl.* 80.5 (Feb. 2021), pp. 7653–7673. ISSN: 1380-7501. DOI: 10.1007/s11042-020-09836-z. URL: https://doi.org/10.1007/s11042-020-09836-z.

[9] Taylor Berg-Kirkpatrick and Dan Klein. "Improved Typesetting Models for Historical OCR". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 118–123. DOI: 10.3115/v1/P14-2020. URL: https://aclanthology.org/P14-2020.

[10] Hans Dieter Betz. *The Greek Magical Papyri in translation, including the demotic spells*. University of Chicago Press, 1996.

[11] Carson Brown. *Github Repo*. URL: https://github.com/CarsonMBrown/glyphs.

[12] Philip Wesley Comfort and David P. Barrett. *The text of the earliest New testament greek manuscripts*. Tyndale House, 2001.

[13] Gregory R. Crane, ed. URL: https://www.perseus.tufts.edu/.

[14] M Daszykowski, B Walczak, and D.L Massart. "Looking for natural patterns in data: Part 1. Density-based approach". In: *Chemometrics and Intelligent Laboratory Systems* 56.2 (2001), pp. 83–92. ISSN: 0169-7439. DOI: https://doi.org/

10.1016/S0169-7439(01)00111-3. URL: https://www.sciencedirect.com/science/article/pii/S0169743901001113.

[15] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[16] Chris Deotte. *Cdeotte/MNIST-CNN-99.75*. URL: https://github.com/cdeotte/MNIST-CNN-99.75.

[17] Maruf A. Dhali, Jan Willem de Wit, and Lambert Schomaker. "BiNet: Degraded-Manuscript Binarization in Diverse Document Textures and Layouts using Deep Encoder-Decoder Networks". In: *arXiv e-prints* (2007), arXiv:1911.07930. ISSN: 0031-3203. DOI: https://doi.org/10.48550/arXiv.1911.07930. URL: https://arxiv.org/abs/1911.07930.

[18] Maruf A. Dhali et al. "Feature-extraction methods for historical manuscript dating based on writing style development". In: *Pattern Recognition Letters* 131 (2020), pp. 413–420. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2020.01.027. URL: https://www.sciencedirect.com/science/article/pii/S0167865520300386.

[19] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[20] Dennis Gabor. "Microscopy by reconstructed wave-fronts". In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 197.1051 (1949), pp. 454–487.

[21] Dennis Gabor. "Holography, 1948-1971". In: *Science* 177.4046 (1972), pp. 299–313. DOI: 10.1126/science.177.4046.299. eprint: https://www.science.org/doi/pdf/10.1126/science.177.4046.299. URL: https://www.science.org/doi/abs/10.1126/science.177.4046.299.

[22] Angelika Garz, Robert Sablatnig, and Markus Diem. "Layout Analysis for Historical Manuscripts Using Sift Features". In: *2011 International Conference on Document Analysis and Recognition*. 2011, pp. 508–512. DOI: 10.1109/ICDAR.2011.108.

[23] Angelika Garz et al. "Binarization-Free Text Line Segmentation for Historical Documents Based on Interest Point Clustering". In: *2012 10th IAPR International Workshop on Document Analysis Systems*. 2012, pp. 95–99. DOI: 10.1109/DAS.2012.23.

[24] Tobias Gass. "Deformations and Discriminative Models for Image Recognition". In: URL: https://thomas.deselaers.de/teaching/files/gass_diploma.pdf.

[25] Maya R. Gupta, Nathaniel P. Jacobson, and Eric K. Garcia. "OCR binarization and image pre-processing for searching historical documents". In: *Pattern Recognition* 40.2 (2007), pp. 389–397. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2006.04.043. URL: https://www.sciencedirect.com/science/article/pii/S0031320306002202.

[26] Alexandros Haliassos et al. "Classification and Detection of Symbols in Ancient Papyri". In: Apr. 2020, pp. 121–140. ISBN: 978-3-030-37190-6. DOI: 10.1007/978-3-030-37191-3_7.

[27] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[28] Keith Houston. In: *The book: A cover-to-cover exploration of the most powerful object of Our time*. W.W. Norton & Company, 2016.

[29] Glenn Jocher et al. *Ultralytics/yolov5*. URL: https://zenodo.org/record/7002879.

[30] Glenn Jocher et al. *Yolov5 GitHub*. URL: https://github.com/ultralytics/yolov5.

[31] Daniel Keysers et al. "Deformation Models for Image Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.8 (2007), pp. 1422–1435. DOI: 10.1109/TPAMI.2007.1153.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[33] Matthias A Lee. *Madmaze/pytesseract: A Python wrapper for google tesseract*. URL: https://github.com/madmaze/pytesseract.

[34] David Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. DOI: 10.1023/B:VISI.0000029664.99615.94.

[35] Mark-Jan Nederhof. "OCR of handwritten transcriptions of Ancient Egyptian hieroglyphic text". In: 2015. URL: https://mjn.host.cs.st-andrews.ac.uk/egyptian/align/dhegypt2015.pdf.

[36] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[37] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[38] *Pickle - Python object serialization*. URL: https://docs.python.org/3/library/pickle.html.

[39] Ioannis Pratikakis et al. "ICFHR 2018 Competition on Handwritten Document Image Binarization (H-DIBCO 2018)". In: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2018, pp. 489–493. DOI: 10.1109/ICFHR-2018.2018.00091.

[40] Mathias Seuret et al. *ICFHR2022 Competition on Detection and Recognition of Greek Letters on Papyri*. July 2022. URL: https://lme.tf.fau.de/competitions/icfhr2022-competition-on-detection-and-recognition-of-greek-letters-on-papyri/.

[41] R. Smith. "A simple and efficient skew detection algorithm via text row accumulation". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 2. 1995, 1145–1148 vol.2. DOI: 10.1109/ICDAR.1995.602124.

[42] Ray Smith. "An Overview of the Tesseract OCR Engine". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. 2007, pp. 629–633. DOI: 10.1109/ICDAR.2007.4376991.

[43] Matthew I. Swindall et al. "Exploring Learning Approaches for Ancient Greek Character Recognition with Citizen Science Data". In: *2021 IEEE 17th International Conference on eScience (eScience)*. 2021, pp. 128–137. DOI: 10.1109/eScience51609.2021.00023.

[44] Julius Tabin. *From Papyrus to Pixels: Optical Character Recognition Applied to Ancient Egyptian Hieratic*. Tech. rep. University of Chicago, 2022.

[45] *Torchvision object detection finetuning tutorial*. URL: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html.

[46] Lucia Vadicamo. "Enhancing Content Based Image Retrieval Using Aggregation Of Binary Features Deep Learning And Supermetric Search". In: (2018).

[47] Luc Vincent. *Announcing tesseract OCR*. URL: http://googlecode.blogspot.com/2006/08/announcing-tesseract-ocr.html.

[48] Alex C Williams. "Computationally accelerated papyrology". PhD thesis. Middle Tennessee State University, 2015.

[49] Alex C Williams et al. "A computational pipeline for crowdsourced transcriptions of Ancient Greek papyrus fragments". In: *2014 IEEE International Conference on Big Data (Big Data)*. IEEE. 2014, pp. 100–105.

[50] Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". In: *arXiv preprint arXiv:1611.05431* (2016).

[51] Wei Xiong et al. "DP-LinkNet: A convolutional network for historical document image binarization". In: *KSII Transactions on Internet and Information Systems* 15.5 (May 2021), pp. 1778–1797. DOI: 10.3837/tiis.2021.05.011.

[52] Wei Xiong et al. *Beargolden/DP-LinkNet: DP-linknet: A convolutional network for historical document image binarization*. URL: https://github.com/beargolden/DP-LinkNet.

[53] Mohammad Reza Yousefi et al. "Binarization-free OCR for historical documents using LSTM networks". In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. 2015, pp. 1121–1125. DOI: 10.1109/ICDAR.2015.7333935.

[54] Zhe Yuan and Jun Zhang. "Feature extraction and image retrieval based on AlexNet". In: *International Conference on Digital Image Processing*. 2016.