Extracting Toponyms from Maps of Jerusalem

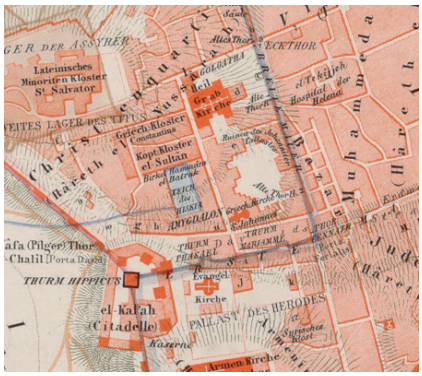
Contents

Introduction

We aim to programmatically and accurately extract toponyms (place names) from historical maps of Jerusalem. Building from mapKurator, a scene text recognition pipeline developed specifically for toponym extraction by the University of Minnesota's Knowledge Computing Lab, we develop a novel label extraction and processing pipeline capable of significant accuracy improvements relative to mapKurator's spotter module alone. We then explore the success of our pipeline during generalization, describe the limitations of our approach, and suggest possibilities for future progress in accuracy or end-user interactivity.

Motivation

Development of an accurate, accessible, and generally applicable tool for extraction of map toponyms is an unrealized goal. Indeed, despite the deep-learning induced boon to scene text recognition (STR) seen in the past decade, STR in general remains difficult due to the sheer variety of shapes, fonts, colors, languages, and text backgrounds encountered.[1] In the context of historical maps and Jerusalem especially, STR issues are compounded by the fact that there is not necessarily agreement between cartographers over toponym spellings (in the case of toponyms transliterated from arabic, for example) or, as is true among biblical toponyms, even the precise location of certain places[2]. Properly collecting toponym locations and spellings from historical maps of Old Jerusalem would thus permit researchers to understand how such ambiguities have manifested as agreement or disagreement in geographic depiction throughout time.



A sample of the linguistic, geometrical, and typographical diversity of toponyms in 19th-century maps of Jerusalem.

In an effort to automate text recognition on historical map archives - our exact use case - the University of Minnesota's Knowledge Computing Lab has developed an end-to-end toponym extraction and rectification pipeline dubbed mapKurator. [3] While we do build on their methods, we identify three issues with simply implementing mapKurator on historical maps of Jerusalem to achieve our intended result. First, some elements of the full mapKurator pipeline (namely the use of Open Street Map to correct extracted labels) are not suitable for historical maps of Jerusalem due to the ambiguities described above. Second, the success of the chief tool used to extract text in mapKurator depends heavily on its model weights - weights that users may not have the computational power or ground truth labels to train. Third, mapKurator's text spotting tool only extracts individual words, requiring further processing to capture multi-word toponyms. We therefore decide to augment mapKurator's text spotting tool with a pipeline involving repeated sampling and methods for recovering proper toponyms from said sampling.

The contributions of this project are thus inroads into the collection of accurate toponyms from historical maps of Jerusalem and a general, accessible pipeline that can be mounted on a pre-trained STR label extractor to improve accuracy.

Deliverables

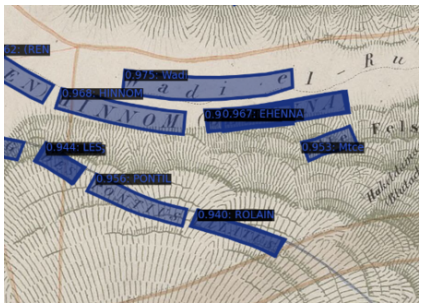
- A novel, general-use, and computationally accessible toponym extraction pipeline capable of improving accuracy for pre-trained SOTA neural network models.
- A set of 266 ground truth toponyms (341 single-line components) across 3 historical maps of Jerusalem, with multi-line groups indicated.
- A set of 3,392 extracted toponyms across 3 historical maps of Jerusalem.
- A set of 2,469 labels denoting font similarity in images of toponyms.

These deliverables are available via the Git repository linked at the bottom of this page.

Methodology

MapKurator Spotter

MapKurator's Spotter module is a neural-network-based method for extracting text instances on scanned historical maps. Trained on human-annotated data with synthetically generated datasets, the spotter module was developed under a Deformable DETR architecture for the mapKurator team's two text extraction networks: TESTR and spotter-v2. We decide to implement as a baseline the spotter-v2 model with a set of pre-trained model weights supplied by the mapKurator team (indeed, in the Results section below, "baseline" refers to a set of labels extracted following patch size tuning). Extracted labels are defined as a tuple of Bounding Polygon, Text, and Confidence Score.

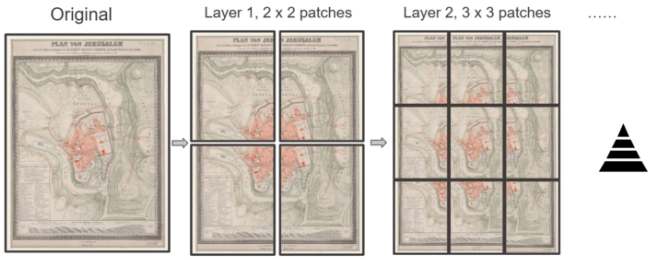


A sample of results from spotter-v2 network on a map of Jerusalem.

Upon extracting labels via the spotter-v2 module we noticed the mapKurator tool (1) had generally low accuracy, (2) had difficulty with overly small or large text, and (3) mishandled toponyms when crops divided text. We sought to address these issues and more with our own pyramid scanning regime and subsequent processing pipeline.

Pyramid Scan

Cutting the original maps into small patches and feeding these patches to the spotter-v2 model is a economic and effective way of getting text instances from a larger map. While the original mapKurator team used an image cropping step in their pipeline, this method is rather naive and crude. Dividing crops with a fixed pixel resolution means the pipeline suffers from an inability to adapt to the different resolutions that might be encountered in an archive of scanned maps, resulting in greater-than-necessary instances of omitted ground-truth text.

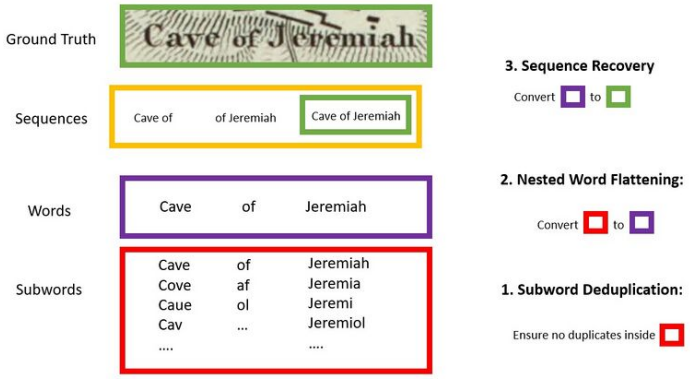


The illustration of the pyramid scan approach.

We proposed a scanning regime that resembles a pyramid through its various layers of cropping. The original map images are cut into n-by-n sets of patches, with n starting at 1 and increasing to 5, and crops overlapping by 30% of geometric area. This regime ensures we gradually sample the same toponyms multiple times and with different coverage, ensuring a healthy mix of detail and complete coverage is represented in our samples. Our experiments suggest 5 layers guarantees a stable increase in all evaluation metrics while keeping a reasonable computational cost. The major benefit of our method is the ability to sample each toponym on the map multiple times, allowing us a better chance at recovering ground truth after processing.

Extracted Label Processing

Let 'word' refer to a series of consecutive non-space characters. For a given word on a given map, let W = (P,T,C) represent a word extracted with mapKurator's spotter tool, where P is composed of the x- and y-coordinates which define the geometric location of W's bounding box, T is recognized text composed entirely of non-space characters, and C is a confidence score for T.



The illustration of Extracted Label Processing pipeline.

Define each ground truth label G = (P, T). Each ground truth label G is a sequence composed of one or more words, and each of these constituent words is composed of one more subwords - meaning either a subset of the word or a misspelling of the word. The related image depicts instances of subwords, words, and sequences for a ground truth label.

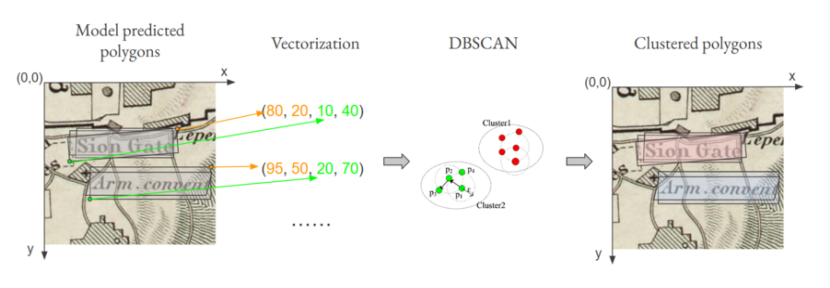
In practice, we have more extracted words than ground truth labels. This is firstly because every piece of text can be sampled more than once in our pyramid context, yielding either fully duplicated word

labels or nested word labels upon extraction. Furthermore, our spotter is only equipped to extract one word per label, precluding proper extraction of multi-word and multiline toponyms. The goal of our extracted label processing, then, is to to convert the set of extracted words into the set of ground truth toponyms $\boldsymbol{G}$.

We implement the following pipeline to address the effects of repeated sampling and our extraction tool's word-based (rather than toponym-based) extraction capabilities: subword deduplication, nested word flattening, and sequence recovery.

Subword Deduplication

Let the set $\boldsymbol{ES}^{SG}$ refer to the set of extracted word labels that correspond to ground-truth subword SG in its entirety. The goal of Subword Deduplication is to retain the single most accurate extracted word W in a given $\boldsymbol{ES}^{SG}$ and exclude the rest - to filter $\boldsymbol{ES}^{SG}$ such that there remains just one extracted `representative' for SG, in other words. Let σ^{*} be the single extracted label remaining in $\boldsymbol{ES}^{SG}$ after subword deduplication has occurred.



The illustration of DBSCAN.

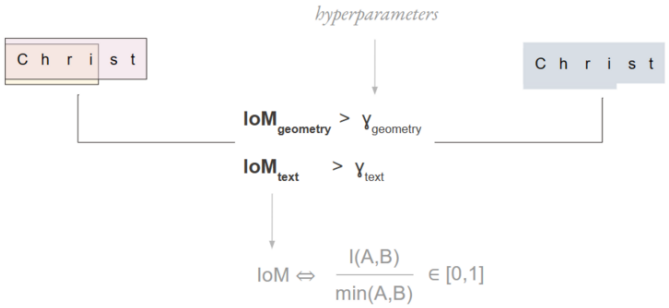
We begin by performing unsupervised clustering of extracted labels' bounding boxes into appropriate sets. To do so, we vectorize each extracted bounding box P with its bottom left and top right cartesian coordinates and implement DBSCAN on the resulting set of four-dimensional vectors. Regarding hyperparameters, we set the maximum distance possible between neighbors intra-cluster to 50 and the number of intra-neighborhood observations defining a core observation to be 3. Outliers identified by DBSCAN are excluded from further processing under the assumption that actual text (as opposed to noise) would be extracted multiple times under our pyramid approach and thus be slotted into a cluster.

With our set of geometric clusters available (meaning we have $\boldsymbol{ES}^{SG}$ for all SG apparently captured during extraction), we proceed to filter each set down to its most appropriate representative by observing extracted text (T) and confidence scores (C). If T is unique for all extracted words in $\boldsymbol{ES}^{SG}$, then σ^{*} earns the T from the label for which C is highest. If T is not unique for all extracted words in ES, we first deduplicate instances of T by choosing one confidence score from a given text's associated confidence scores via a RANSAC search, then proceed as specified above.

We end this stage with a single σ^{*} for each SG apparently recognized by our model; call this set $\boldsymbol{\Sigma}$.

Nested Word Flattening

Let $\boldsymbol{\Sigma}^{WG}$ be a subset of $\boldsymbol{\Sigma}$ that represents, for a given ground truth word WG, the set of extracted and deduplicated word labels amounting to all relevant unique subwords identified by our model. Note that for a given ground-truth word, the set of subwords captured by our model may or may not cover all possible subwords that can be derived. The goal of Nested Word Flattening is to retain a single label from each $\boldsymbol{\Sigma}^{WG}$: the label ψ^{*} = WG.



The illustration of intersection over minimum.

This process is performed iteratively. The first step in the sequence consists of computing pairwise geometric and textual intersection over minimum (IoM) values between all labels in $\boldsymbol{\Sigma}$. For example, suppose we are comparing σ_1 and σ_2. In this case, geometric IoM equals $P_1 \cap P_2$ (the intersection of the associated bounding box polygons) divided by the area of the smaller polygon. Textual IoM, meanwhile, equals the number of non-unique shared characters in label texts T_1 and T_2 divided by the length of the longer string. Those pairs with geometric IoM value $\gamma_{geom} > 0.75$ and textual IoM value $\gamma_{text} > 0.5$ are considered to exhibit a subset-parent relationship. They are therefore 'flattened', or amalgamated, meaning (1) a new word label σ_α is added to $\boldsymbol{\Sigma}$ with $P_\alpha = P_1 \cup P_2$ and T_α equal to the longer string from T_1 and T_2, and (2) both σ_1 and σ_2 are dropped from $\boldsymbol{\Sigma}$. When all possible amalgamations have been made based on the group of pairwise combinations satisfying our γ_{geom} and γ_{text} conditions, the sequence begins anew with updated $\boldsymbol{\Sigma}$, and ultimately terminates when $\boldsymbol{\Sigma}$ ceases to yield further possible amalgamations. In this way we accomplish our goal of recovering one representative per $\boldsymbol{\Sigma}^{WG}$ in an unsupervised way (i.e. without actually defining the sets $\boldsymbol{\Sigma}^{WG}$ and instead recovering their representatives from the global set $\boldsymbol{\Sigma}$).

Once both Subword Deduplication and Nested Word Flattening have occurred, the set of extracted words has been condensed to the set $\boldsymbol{\Psi}$ composed of each unique word label ψ^{*} considered equivalent to some ground truth word in the ground truth set.

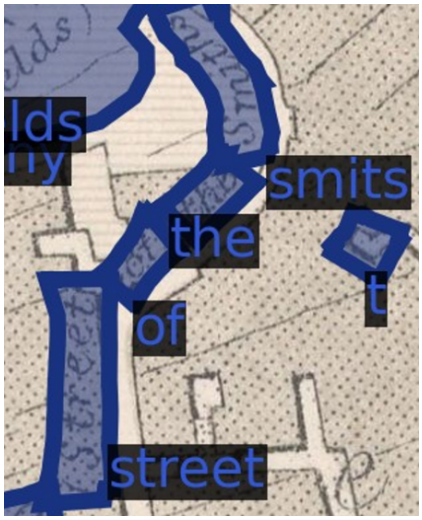
Sequence Recovery

One large and as yet unaddressed issue is the fact that our extracted labels still exist only at the word level with the processing steps defined above. With sequence recovery, we seek to recover ground truth sequences using ***geometric*** and ***typographical*** features from word labels extracted and processed in previous steps.

Identified toponym sequences are "Single Lines", "Multiple Lines", and "Curved Lines". In this project, we do not consider multi-line sequence recovery.

Single Line

Multiple Lines

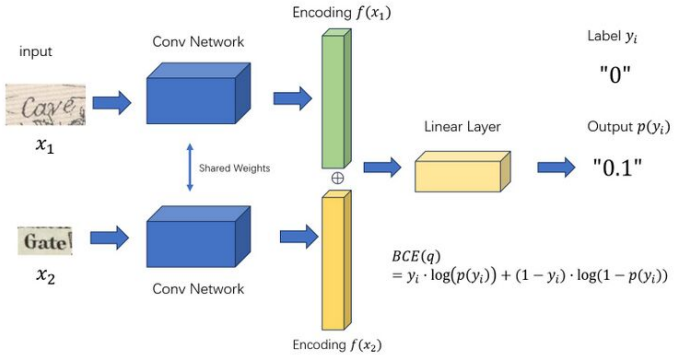


Curved Line

Font Similarity

We assume that word labels which belong to a sequence are identical in their fonts. To implement this assumption, we train a neural network which takes as input two cropped images of text outputs a font similarity score between 0 and 1, i.e., the probability of two texts having the same font.

To train our model, we first crop rectangular images for all labels in $\boldsymbol{G}$. By manually annotating randomly chosen image pairs and padding annotations with pairs that appear in multiline toponyms (assumed to have a similarity of 1), we create 447 matching-font labels and 2,022 non-matching-font labels. We used 80% of these labels to train our model, retaining the rest for testing. Our model achieved 92.9% recall and 89.7% precision on the test set.



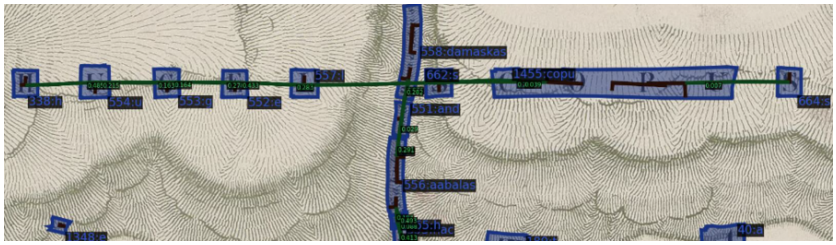
The archetecture of our Font Descriminator.

As the image shows on the right shows, we implement a Siamese network which encodes two input images of text on a map (resized as 128*128) with the same two-layer convolution neural network, and use two linear layers and a Sigmoid function to map the concatenated vector of images to a scalar value ranging from 0 to 1. We use Binary Cross Entropy and the Adam optimizer with learning rate 0.01 for training, and retain the model weights obtained at epoch 40 for later use.

Once the model is trained, we calculate font similarity for all pairwise combinations of subword-deduplicated and nested-word-flattened word labels in $\boldsymbol{\Psi}$.

Spline Connection Cost

We assume that word labels which belong to a sequence should be connectable via relatively short and non-curved splines. To implement this assumption, for each subword-deduplicated and nested-word-flattened word label ψ in $\boldsymbol{\Psi}$, we first apply PCA to the points constituting bounding box P_ψ to recover unit vector d_ψ, the principal component basis for a given label's bounding box. Furthermore, for each P_ψ, we create the vectors $e_{\psi,p}$ between successive points p in $\mathbb{Z}$ along P_ψ, recovering directional vectors along each label's bounding box. We then use these bounding-box vectors to adjust the principal component direction vector in preparation for spline calculation as follows:


Visualization of Bezier Splines on text instances.

Define $RES_p = |d_\psi * e_{\psi,p}| (1 - |d_\psi * e_{\psi,p}|)$. For bounding-box vectors parallel or orthogonal to d_ψ, this value will be 0. Our goal is to maximize the number of vectors (nearly or exactly) parallel or orthogonal to d_ψ, as it would represent a more suitable principal direction for a given bounding box. Thus, our specific objective function is minimize the sum across all p of $w_p*(RES_p)^2$, where w_p is the length of $e_{\psi,p}$ (it is more important for longer bounding box vectors to agree with the principal component direction than shorter bounding box vectors). We retrieve the optimal principal component direction d_ψ^* via the Gauss-Newton algorithm for solving non-linear least squares optimizations.

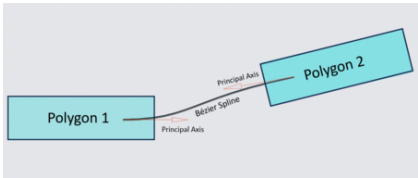


Illustration of Bezier Splines.

Next, for each d_ψ^*, we define a primary and secondary axis relative to the centroid of the associated bounding box P_ψ (the latter axis being orthogonal to d_ψ). We call this local axis the bounding box's anchor. If the variance of the principal component for a given bounding box is six times larger than the variance of the secondary component, we assume the polygon may have sufficient size to warrant different local axes at

different locations (for example, the word may be curved), and thus, we divide the bounding box to permit multiple anchor points. These anchor points are points through which splines are drawn.

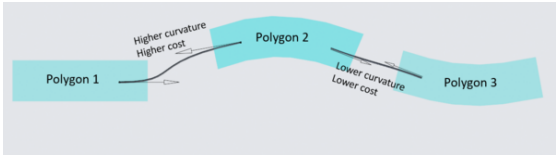


Illustration of Spline Connection Cost.

We create Cubic Bezier splines to measure the cost of connecting word labels. Assume we are drawing a spline from anchor point ξ_1 to anchor point ξ_2 in distinct word labels wl_1 and wl_2. The first control point cp_1 is the centroid of ξ_1. The second control point is cp_1 plus the product of (1) the furthest point from cp_1 on wl_1's bounding box in ξ_1's principal direction and (2) ξ_1's principal direction vector. The fourth point cp_4 is of course the centroid of ξ_2, meaning the third control point is ξ_2 minus the product of (1) the furthest point from cp_4 on wl_2's bounding box in ξ_2's principal direction and (2) ξ_2's principal direction vector. . We are, in effect, drawing a curve between two polygons that respects their orientation. After drawing such splines between all pairwise combinations of word labels (with distant geometric pairs excluded to reduce computational complexity), we can finally calculate the cost of connecting word labels via spline.

For a given spline, let C equal its maximum curvature, AD equal the euclidean distance between the two connected anchor points' centroids, GD equal the minimum euclidean distance between two connected bounding boxes, and FS equal the average font size of the two connected polygons (as measured by the expansion along the non-principal component direction of the polygons). The greater a spline's maximium curvature and the greater the distance between two polgyons, the less likely it is two polygons deserve to be grouped into a sequence. However, one should scale the former by centroid distance and the latter by font size to achieve scale invariance. Hence, the Spline Connection cost is equal to C * AD * GD / FS, which ranges from 0 to infinity. The smaller this value, the more it makes sense for two word labels to belong to the same sequence.

Combining Word Labels Using Font and Spline Metrics

With both font similarity and spline connection cost metrics in hand, we identify those word label pairs for which spline connection cost is less than 1.5. We then commence pairwise combinations by prioritizing the smallest spline connection costs; combined labels inherit maximum font similarity scores and minimum spline connection to constituent labels' neighbors, while text is concatenated starting with the leftmost polygon's text in a given pair. This process is performed iteratively until no combinations can be made, at which point the resultant set $\boldsymbol{\Theta}$ is assumed to be equivalent to $\boldsymbol{G}$ (some issues with this assumption are discussed in the limitations section below).

Unfortunately, we find that our font similarity score is too restrictive at present, and do not use it to filter candidates for pairwise combination. That said, we believe it will play an important role for future improvements in this project.

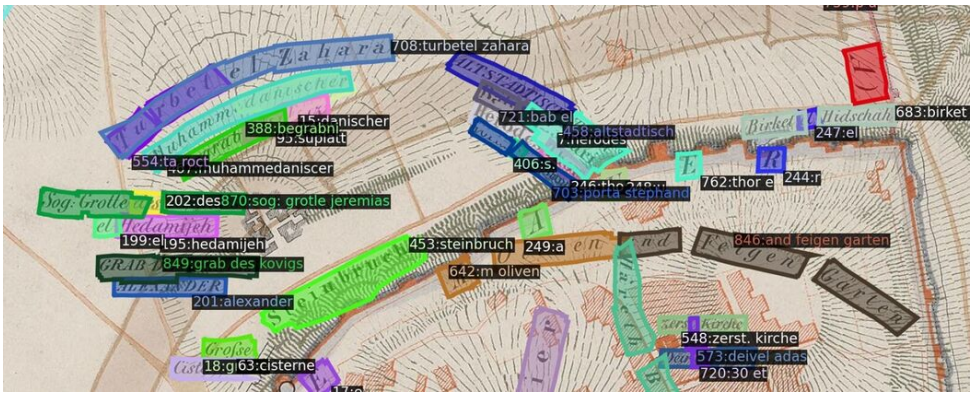
Evaluation Metrics

Unmentioned thus far is how we develop our ground truth label set $\boldsymbol{G}$. Using the online image annotating software VIA (https://www.robots.ox.ac.uk/~vgg/software/via/via.html), we manually label 341 labels across 3 maps. These 3 maps were chosen for their high resolution, their preponderance of toponyms located directly over place locations (as opposed to inside legends), their visual diversity, and their linguistic diversity. Ground truth labels are annotated at the line level rather than the toponym level. For example, if "Church of" appears directly above "the Flagellation", we annotate each as a separate ground truth label - though we manually indicate that they belong to the same toponym by assigning a group id. Though we report accuracy statistics at the line-level below, line-level annotations with the option for multiline combination will allow us to explore multiline accuracy going forward, shedding light on the power of our pipeline in different contexts.

To evaluate $\boldsymbol{\Theta}$ for a given map, we first retain only those extracted and processed labels located in the same pixel space as a crop of the map which has all possible ground truth toponyms in $\boldsymbol{G}$. We then calculate all pairwise geometrical IoUs between these subsets of $\boldsymbol{\Theta}$ and $\boldsymbol{G}$, find the 1-to-1 matched set that maximizes the sum of said IoUs, and divide this sum by the number of extracted labels (in the case of precision) or the number of ground truth labels (in the case of recall). Text-based accuracy statistics, meanwhile, report the sum of normalized Levenshtein edit distances from the 1-to-1 **geometrically** matched set divided by the pertinent number of labels.

Results

As mentioned above, "baseline" in this section refers to a set of raw extracted labels from mapKurator's spotter-v2 module. We observed that baseline precision and recall are tightly correlated with the resolution of input patches (recall that crop size is an input into the mapKurator team's spotter-v2 module). We ultimately selected a crop size that ensured a sufficient level detail based on accuracy statistics; note that one major advantage of our pipeline is the performance is not sensitive to a user-selected crop size parameter, ensuring a level of scale invariance which means better generalization and less hyperparameter tuning.



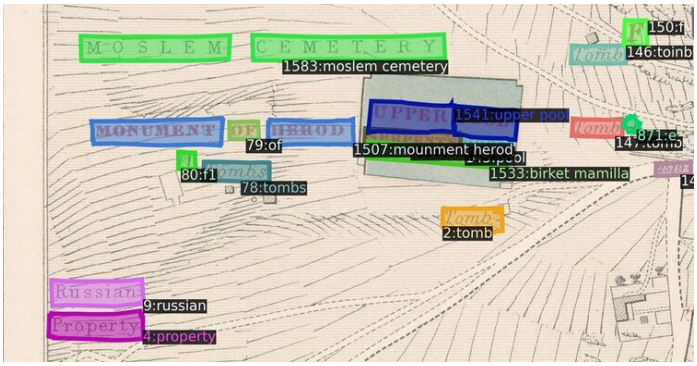
A cropped image illustrating the final result for Kiepert 1845. Text instances from the same sequence have the same color.

Kiepert 1845 (87 ground truth labels)

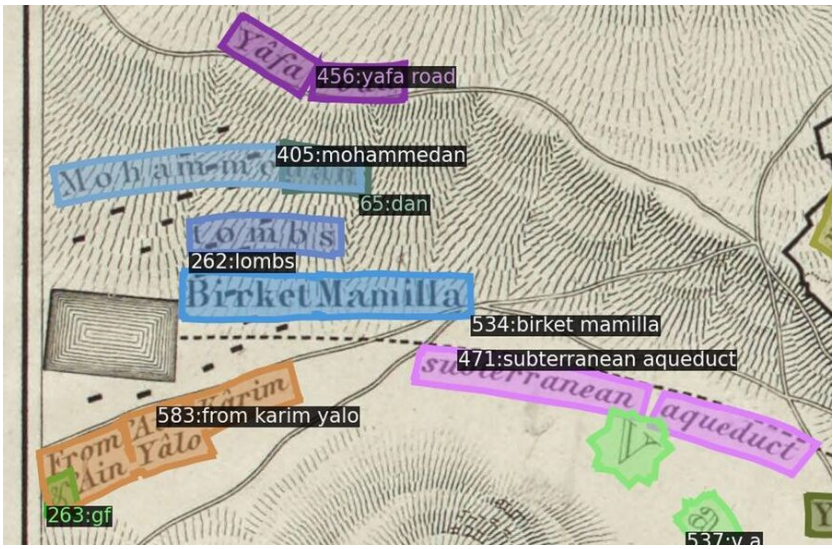
	Precision		Recall	
	Geometric	**Textual**	**Geometric**	**Textual**
Baseline	12.87%	10.34%	48.69%	39.10%
Pyramid + Processing	47.68%	50.19%	60.83%	64.03%

Saunders 1874 (136 ground truth labels)

	Precision		Recall	
	Geometric	**Textual**	**Geometric**	**Textual**
Baseline	14.48%	13.59%	34.82%	32.68%
Pyramid + Processing	52.70%	49.50%	46.12%	43.31%



A cropped image illustrating the final result for Saunders 1874. Text instances from the same sequence have the same color.



A cropped image illustrating the final result for Vandevelde 1846. Text instances from the same sequence have the same color.

Vandevelde 1846 (94 ground truth labels)

	Precision		Recall	
	Geometric	**Textual**	**Geometric**	**Textual**
Baseline	11.80%	12.26%	48.96%	50.87%
Pyramid + Processing	53.79%	58.58%	63.52%	69.17%

We see both precision and recall improvements at the text- and geometric-level across three distinct maps. Additionally, we find that, with but one exception, each stage of our processing pipeline provides a marginal boost to accuracy statistics across all three maps (these marginal statistics are not pictured). We are cautiously enthusiastic about our results, and use the next to section to discuss potential areas of weakness.

Limitations and Future Work

One chief limitation of our project is the limited evidence pointing toward its generalizability. With only 341 ground truth labels developed from 3 maps – maps chosen in part for their relatively high resolution – and marked inter-map variation in accuracy boosts, it is not evident that our pipeline would assuredly yield significant benefits across other historical maps of Jerusalem, not to mention other maps in general. Furthermore, defining our baseline as the accuracy of mapKurator's pre-trained extraction module on historical maps of Jerusalem is problematic given that it was developed using a different map archive. A more appropriate evaluation regime would involve applying our pipeline to the same set of maps and labels the mapKurator team used to create and evaluate their model, thus expanding the number and diversity of ground truth labels, decoupling the marginal effect of our pipeline from the marginal effect of substituting the mapKurator's map archive with another, and (ideally) making a stronger case for our pipeline's generalizability. Though this evaluation regime is outside the scope of this Jerusalem-based project, we consider it to be the most important step toward the further validation of our approach.

Certain stages of our pipeline could benefit from further tuning. Our spline-based approach toward single line sequence recovery does not accommodate cases where the centroid of two consecutive words is vertically mismatched (such as when a small, short word follows a larger word and both are bottom-justified, for example). We neglect neglect multiline sequences in our current pipeline. One of our chosen maps uses a pervasive graphical element to represent trees that is consistently mistaken by the mapKurator spotter tool for a letter of text. Many such unresolved corner cases persist due to time limitations; addressing them would boost accuracy statistics. So too would developing further labels for font discriminator tool, which still misclassifies a small set of label pairs despite thousands of manually labelled comparisons, likely due to imbalance in the number of matching fonts in the label set. In fact, given the narrow map coverage and imbalanced label set used to train the discriminator, it may still take considerable signal from the map the text is pulled from rather than the type of font used alone. Identifying and tackling each processing stage's distinct imperfections is a necessary if time-intensive path toward making our pipeline production ready.

Finally, we recognize that certain elements of the mapKurator pipeline which we disregard due to their inability to directly apply to our map archive would be beneficial if adapted for our project. We refer primarily to the PostOCR module, in which extracted text is mapped to verified toponyms from OpenStreetMap using a fuzzy query function (the issue with directly using this module is described in the Motivation section above). One can imagine emulating this process with a different set of "valid" toponyms: simply run the spotter module over a vast set of Jerusalem maps, apply our label processing pipeline, retain a set of the most prolific toponyms (or perhaps words) across all maps as the substitute for OpenStreetMap's dictionary, and revisit all extracted toponyms to modify or at least annotate any which exhibit minor spelling changes from this novel dictionary. Such a process would invoke the same sampling justification behind the pyramid scanning regime to create a global, archive-wide dictionary in an unsupervised manner, offer a lens through which to view potential spelling disagreements, and facilitate an environment in which our pipeline actually improves as more unlabeled data is processed.

We believe the pipeline we have produced to be useful and sound despite the limitations listed above, and are eager to see how resolving said limitations might further improve the extraction of toponyms from historical maps.

Project Timeline

Timeframe	Task	Completion
Week 4	▪ Finalize and present project proposals. ▪ **Toponym extraction project selected.**	✓
Week 5	▪ Survey SOTA toponym extraction tools.	✓
Week 6	▪ Port MapKurator's Spotter tool and model weights into Windows-based Python. ▪ Select two (later three) maps to use when implementing, evaluating, and fine-tuning MapKurator's model.	✓
Week 7	▪ Create ground truth labels for first map with VIA's online interface.	✓
Week 8	▪ Create ground truth labels for second map. ▪ Implement 1:1-matched precision and recall via IoU (geometry) and normalized Levenshtein (text). ▪ Calculate baseline accuracy statistics.	✓
Week 9	▪ Implement multi-layer pyramid application of MapKurator's Spotter.	✓
Week 10	▪ Create ground truth labels for third map. ▪ Implement subword deduplication and nested word flattening on pyramid-derived toponyms.	✓
Week 11	▪ Calculate pyramid accuracy statistics. ▪ Fine-tune subword deduplication and nested word flattening. ▪ Deliver Midterm presentation.	✓
Week 12	▪ Launch Wiki. ▪ Implement PCA on extracted label polygons to assist in font discriminator crops and Bezier splines. ▪ Create manual labels for font discriminator. ▪ Train font discriminator.	✓
Week 13	▪ Add ground truth labels to our three maps. ▪ Implement Bezier spline ▪ Implement iterative sequence recovery using font and Bezier scores	✓

Week 14	▪ Fine-tune Sequence Recovery.
	▪ Calculate final accuracy statistics.
	▪ Finalize Wiki and final presentation.
	▪ Postponed for future work:
	  ▪ Hierarchize final toponyms and develop Voronoi map.
	  ▪ Prototype toponym-disagreement visualizer.
		✓

Github Repository

Jerusalem Maps EPFL DH405 (https://github.com/kaedejohnson/jerusalem_maps_epfl_dh405)

Acknowledgements

We thank Professor Frédéric Kaplan, Sven Najem-Meyer, and Beatrice Vaienti of the DHLAB for their valuable guidance over the course of this project.

References

Webpages

▪ MapKurator GitRepo (https://github.com/machines-reading-maps/map-kurator)

- MapKurator Introduction (https://knowledge-computing.github.io/mapkurator-doc/#/)
- VIA Image Annotator (https://www.robots.ox.ac.uk/~vgg/software/via/via.html)

Chief Literature

- Kim, Jina, et al. "The mapKurator System: A Complete Pipeline for Extracting and Linking Text from Historical Maps." arXiv preprint arXiv:2306.17059 (2023).
- Li, Zekun, et al. "An automatic approach for generating rich, linked geo-metadata from historical map images." Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020

Footnotes

1. *Liao, Minghui, et al. "Scene text recognition from two-dimensional perspective." Proceedings of the AAAI conference on artificial intelligence. Vol. 33. No. 01. 2019.*
2. Smith, George Adam. Jerusalem: The Topography, Economics and History from the Earliest Times to AD 70. Vol. 2. Cambridge University Press, 2013.
3. Kim, Jina, et al. "The mapKurator System: A Complete Pipeline for Extracting and Linking Text from Historical Maps." arXiv preprint arXiv:2306.17059 (2023).
