Bruce Robertson

Optical Character Recognition for Classical Philology

Abstract: This paper explains the technology behind recent improvements in optical character recognition and how it can be attuned to produce highly accurate texts of scholarly value, especially when dealing with difficult scripts like ancient Greek. Drawing upon several practical experiments using the Ciaconna OCR system (itself based on OCRopus), it shows: the impact of Unicode normalized forms on recognition accuracy; the importance of removing ambiguously encoded characters from training material; the advantage of using separate classifiers for different scripts; the helpful effects of image augmentation; and the effects of binarization levels. It also describes how Ciaconna embeds information about spell-check and dehyphenation within its output.

Introduction

Classical philologists may have noticed in the past years a remarkable expansion of texts, especially Greek ones, available online as open data. Throughout much of the 1990s and 2000s, the venerable Perseus collection may have offered an excellent foundational collection of canonical texts in history, poetry and philosophy; but open texts pertaining to the history of science, scholia, and minor philosophical works were lacking. Today, in contrast, the First Thousand Years of Greek project, whose data can be viewed and visualized within the Scaife Digital Reader, offers a far more extensive corpus of open texts, adding at latest count 22 million new words of Greek. New optical character recognition (OCR) techniques have made some of this expansion possible: whereas almost all Greek in the original Perseus collection was generated through expensive manual double-key entry, much of the new data in the Scaife Viewer began as high-quality OCR output whose errors were manually corrected, a far more affordable prospect if the OCR is accurate enough. The latter approach opens new avenues for digitizing scholarly works, including journal articles, monographs and ancient texts for which open editions are not available.

This chapter aims to explain to the digital philologist the conceptual and computational foundations of OCR, especially as it pertains to scholarly

Bruce Robertson, Mount Allison University

https://doi.org/10.1515/9783110599572-008

materials. It is based in my eight years of working on Greek and Latin-script OCR, most recently within the First Thousand Years of Greek project. In this consortium, I wrote all the specialized OCR code, supervised the high performance computing environment in which the OCR took place and provided web-based editing environment to improve the correction task. The following guide should help today's philologist understand the leading edge of OCR, whether she or he merely wants to make use of these data knowledgeably or perhaps wants to participate in an extensive OCR-based project.

Scholarly OCR

The term "Optical Character Recognition" is commonly used to denote the process of transforming a computer image of text into a digital text file, usually so that the latter can be subjected to all the digital tools that manipulate and analyze text. The words in a page image cannot, in isolation, be treebanked, searched, sorted or subjected to n-gram analysis; OCR makes this possible. As a term OCR is, at best, synecdoche, since recognizing characters *per se* is only one small step in a much broader sequence of processes necessary to complete this transformation. (Indeed, as we will see, the best algorithms today do not really even perform 'character recognition' but rather something more like 'line recognition'.)

The approach one takes to each of these steps depends in large part on the intended use of the textual output. It might be surprising to know that for many purposes, even academic ones, relatively low quality OCR output has many uses: often such output can be corrected or subjected to sufficiently clever fuzzy search algorithms so that the corresponding page images can be shown. Services such as JSTOR use this 'image-fronted' search technique. With this approach, also, the exact reading order of the words on the page image need not be accurately determined. It is also clear that a highly useful service like Google n-grams can be developed without perfect OCR.

In contrast, the OCR results for projects that are preparing complete renderings of texts, what I am calling here 'OCR for scholars', must be highly accurate, since scholars have a very low tolerance for errors in their texts. For this reason, the First Thousand Years of Greek project paid commercial editors to correct our OCR to 99.95% character accuracy, or no more than 5 mistakes per every 10,000 characters. Every correction adds to the labour cost and therefore, within a fixed budget, reduces the number of words produced by the project. For example, output with 95% accuracy will need 495 changes per 10,000 characters; whereas

output with 97% accuracy requires only 295, a reduction by 40.4%. This two percent improvement in OCR accuracy allows for 40% more material to be generated with the same budget! In fact, if the reading order of the page is even slightly misrepresented or if more than, say, 5 characters per 100 are incorrectly recognized, then the time it takes to correct such results becomes greater than the time it would take to transcribe the text manually. Thus, scholarly OCR is far more demanding than many other of its uses, but there is one mitigating aspect: OCR for scholarly use usually involves a relatively limited corpus of texts, perhaps thousands of texts, but nothing like the volume of commercial applications. In this context, it is worthwhile carefully to optimize each possible aspect of the process.

Initial steps in OCR

Corpus OCR begins by acquiring or taking digital images of pages that share similar (ideally, identical) fonts and layouts, so that when we have trained the OCR engine to recognize a small set of these pages, it can operate on a great number at once. Of course, the best results come from the best images. If lower quality images are available, for instance on Google Books, it is tempting to begin with them; but this is often a bad decision because the labour required to correct these results might be much more than the relatively short amount of time it takes to simply re-scan the books, ensuring better, and therefore less time-consuming, raw output. A flat-bed scanner ensures the images are in-focus and evenly illuminated.

These images are then 'cleaned' either manually or automatically, using a program like ScanTailor, which separates two-up scans into separate pages, straightens the images, dewarps them and removes minor artefacts and blank margins. Following this, the page is (usually) binarized – that is, made into a black-and-white image. The operator should inspect the output of the binarization stage, ensuring that the binarization level produces a human-readable output.[1] The next computational step divides the page into separate lines while attempting to follow the proper reading order of the document. The algorithms that do this processing are often devised for modern texts and for Latin script. A layout of ancient Greek with little space between lines may

1 If one begins with black-and-white images, binarization is not necessary, but this is usually a poorer approach because it allows the scanner or camera effectively to decide upon the binarization level, and these are rarely optimized for character recognition.

require that the algorithm's parameters be altered. At times a complex page layout might require a separate computational pre-process, as in Robertson, Dalitz, and Schmitt (2014).

Recognizing characters

The next step, recognition of the characters in these lines, is the most critical in OCR, and it is here that the greatest advances have taken place in the past years. For these reasons, the limitations and potential of character recognition algorithms must be comprehended in order to achieve high-quality results.

To understand these, imagine a writing system comprising only two glyphs: one, like Latin 'O' or Greek majuscule omicron, is a black circle on a white background; the other, like Latin 'I' or Greek majuscule iota, is a black vertical line on a white background. The computer recognizes as a possible glyph every blob of black pixels that is surrounded by white. How could a computer be programmed to tell the difference between these two types of blobs? One obvious approach would be to find the rectangle or 'bounding box' around them. If this box's profile is, say four times taller than it is wide, then we say it has the 'is-tall' *feature*. We can apply this feature to have a robust way of choosing between, or *classifying,* glyphs in this imaginary writing system. A blob that has the 'is-tall' feature is classified as an 'I'; otherwise, it is classified as an 'O'.

Unfortunately, someone adds a third character to this writing system, one that looks like Latin 'M' or Greek majuscule mu. If we use the feature and classifier we described above, we can be certain that this new character won't be classified as 'M', of course, since the classifier knows nothing of that character. It will probably be classified as 'O', since its bounding box is squarish. To make an engine that recognizes all three characters we need to extract at least one more feature from all our blobs and then make a new classifier that is based on it and the old one. Let us say this new feature will be horizontal symmetry: 'I' and 'O' will have this feature, but 'M' will not because its top half is not symmetrical with its bottom half. Our new classifier might work like this: if a blob has the is-tall feature and the horizontal symmetry feature, it is an 'I'; if doesn't have the is-tall feature and has the horizontal symmetry feature, it is an 'O'; and if it doesn't have the is-tall feature and doesn't have the horizontal symmetry feature, it is an 'M'. This leaves one other possibility, that a blob has the is-tall feature but isn't horizontally symmetrical. We might want to indicate that this is a result that confounds our classifier.

This example illustrates the fundamental elements of all OCR engines: they test for features, and they somehow integrate the results of those tests to make a classifier. But as we add more characters say 'E', 'H', 'N' and 'P', it's clear that many more features will be needed and that the classifier's means of integrating these becomes more complex.

Adding code that tests for features – such as the volume of black space in various regions, or vertical symmetry – is a one-off challenge, and such code can automatically be applied to every dark region. In contrast, as the number of glyphs and feature tests increase, the classifier becomes very hard to produce by hand, especially one that is optimized for the best results. OCR engines like Tesseract 3 and Gamera therefore test connected components for features that are defined by human-created code, and they then use a class of machine learning algorithms called 'supervised learning algorithms' to discover the best classifier. This is the one that best matches the results of the feature tests with a set of human-verified corresponding characters known as 'ground truth.' These algorithms scores the output for one certain weighting of the importance of the features. They then move on to a slightly different weighting compare the score and, based on this information, try to come up with an even better setting, and so on.

This optimization can only do so much: it seeks to get the best possible result given the features extracted, where *best* is defined by the given ground truth. If the ground truth is not representative of the images eventually to be processed, the optimized classifier could be worse at OCR'ing those images than another classifier using the same features and glyphs! For instance, if the volume to be OCR'd is a critical edition, and the classifier is optimized only using the body of the text and not the apparatus criticus, the optimization will never take into account the high frequency of the usual letters and sigla that appear in the apparatus. (In fact, it probably would never get the chance to test itself on symbols like '‖'.) Similarly, if some pages use different fonts or stylistic variations (bold, italics, etc.), then these will be well processed only if their features are extracted and they are included in the ground truth used to generate the optimized classifier. Therefore one cannot omit training for certain parts and then 'ignore' the resulting bad output because to ignore something, you first have to classify it; otherwise, because the classifier has been trained only on the desirable parts, it will tend to produce output that resembles those parts, and distinguishing them as 'to be ignored' is a new problem, and solving these problems in sequence is no easier.[2]

2 This is a corollary of the so-called 'no free lunch theorem' in optimization (Wolpert and Macready 1997).

This simplified overview makes clear one of the reasons that OCR for Greek texts is particularly challenging. We noted that when a new glyph was added, the task of the classifier became more complicated. Now since it is almost always the case that predominantly Greek pages still include some Latin glyphs, we cannot ignore these, or we will get lines of Greek output and have a difficult time knowing what is useless. Thus, whereas a Latin critical edition might be require a classifier to distinguish between 120 glyphs, Greek editions usually double this number, and thereby they make a much greater demand of the classifier. Additionally, each human writing system provides a set of characters that are easily distinguishable amongst themselves; but Latin script and Greek comprise many characters, especially upper-case ones, that are nearly identical. If our initial writing system contained Latin 'O' *and* Greek majuscule omicron along with Latin 'I' *and* Greek majuscule iota, the feature extraction would need to be much more subtle, and even with all that the results would depend on the typeface involved.

Assuming careful selection of the ground truth characters, the approach described so far has a number of advantages. It requires little training – maybe only three pages' worth – to become adept. Secondly, it performs well with rarely occurring glyphs because each glyph is subject to the same feature tests and then becomes part of the classifier. Finally, with this system it is possible for a programer to modify the code, noting the position and ascribed characters of each connected component. For this reason, previous Greek OCR systems were built using supervised learning systems, such as Rigaudon,[3] based on the Gamera OCR library,[4] and White (2013), based on the Tesseract 3 engine. Both of these use the k-NN supervised learning algorithm.[5]

But there are also obvious problems with this approach. First, although it is based in a one-to-one correspondence between connected components and glyphs, in practice the situation often is messier. The letter 'i' ideally comprises two connected components, but often the letter's superscript dot is not separated from its vertical stroke. Also, imperfect scanning sometimes causes two characters to join together into a single connected component. Even more common for classical philologists is the situation where diacritical accents join with each other or with their combining letter. Two solutions are possible:

3 (Robertson and Boschetti 2017).
4 (Droettboom et al. 2003).
5 (Kononenko and Kukar 2013).

either the system can recognize these combined glyphs are representing a sequence of characters, or it can recognize them as a class of characters that should be divided (or 'cut'), allowing the resulting two connected components to be further recognized. Both solutions are imperfect, since the first greatly increases the number of glyphs to be identified and the second results in strangely shaped glyphs which are hard to identify as their own connected components.

For these reasons, a new approach to character recognition has become increasingly popular. In a recent presentation on scholarly OCR, an audience member asked presciently, "if the computer is so smart, why doesn't *it* figure out the best features for character recognition?" In fact, that is exactly what happens in this new approach, which is based on so-called Recurrent Neural Networks, a class of artificial neural networks that can take credit for the expanding role of Machine Learning in everyday life, from the ever-improving voice recognition of smart speakers in the home to facial recognition. RNN-based OCR engines (like Ocropus or Tesseract 4) replace the supervised learning algorithms described above with a kind of learning algorithm that has no need of human-determined features: its classifier takes as input *simply lines of the characters*, not a set of feature scores, and as it trains over a great number of iterations it forms a neural net classifier that can transform the image of a line of text into the corresponding characters. Strictly speaking, then, they do not perform optical *character* recognition, but rather optical *line* recognition: the context of a character in its line becomes pertinent information. (Although there are now many RNN-based open source OCR projects to chose from, this paper will explore this technology using the longstanding Ocropus engine.[6])

Thus overall these classifiers usually perform better than those based on supervised learning algorithms. They are also typically more robust and agile when handling poorer data: their results degrade more slowly when confronted with characters or diacritics that are combined. Furthermore, they automatically manage the reordering of characters that diacritics sometimes make necessary. Their only drawback is that they require copious, accurate training data.

6 (Breuel 2008). Tesseract's version 4 offers an RNN mode, while Calamari (Wick 2019), Kraken (Kiessling 2019) and Ocropus3 (Breuel 2019) offer much speedier line recognition and classifier generation. Calamari notably integrates an image augmentation process as discussed below.

Choosing ground truth

OCR ground truth is unicode plain text: no sort of markup can be used to indicate italicized text, superscripts or text layout. Despite this, there is a trade-off to be made here between the number of different characters in the training set and the accuracy of the output. It should be understood that for training purposes not every separate glyph must be represented with a different code point: all methods of training can learn, for instance, to represent both italicized and upright 'a' with the same output. Nevertheless, special attention must be paid to each and every character's encoding in the ground truth, with consideration for how it might be used in the future. For instance, quotation marks: should left and right variants be encoded differently? The same is true for various glyphs that can be used in Greek texts: if a single glyph in the page images is represented in ground truth by two different Unicode characters, the classifier optimizes for an illusory distinction, in the process very likely becoming somewhat worse at distinguishing between actual characters.

The effect of data ambiguity

This can be demonstrated if we note the degraded performance caused by intentionally confusing a set of ground truth that otherwise produces well-performing classifier. Classifier performance is measured through character accuracy, the percentage of characters that are 'right'.[7] For example, a classifier was trained on 1113 lines of Loeb text that comprised 82 unique Greek characters. When tested against a different page of Loeb Greek – proper procedure dictates that the test document not be used for training purposes – this classifier scored a 99.1% character accuracy after 19600 lines of training. Half of these lines were then altered: the "middle dot" (U+00B7) character replaced the "Greek ano teleia" (U+0387) character; "double quotation mark" was substituted for the characters that had been carefully encoded as either "right double quotation mark" or "left double quotation mark"; and the text's interrogative punctuation, previously indicated with the "semicolon" character, was swapped with the "Greek question mark" (U+037E) character. The result is a slightly confused ground truth, in which a small number of glyphs were

7 More formally, this is calculated as $(n - e) / n$, where n equals the number of characters in the ground truth and e equals the number of errors as determined by Levenshtein distance (Rice et al. 1993).

represented in two different ways, a common situation when more than one editor provides the training text.[8] This classifier could only muster a far poorer maximum character accuracy of 95.5% after 26700 lines of training, a drop of 3.6%. It is important to note that the additional errors did not in fact pertain to the characters that our experiment altered: many other letters and diacritics were mis-identified or not produced in the output. So automated search-and-replace should be used to disambiguate the ground truth to one or the other, always with the goal of reducing the number of characters in the training set to a minimum necessary number.

This is not to say that every aspect of character encoding should be simplified. Distinctions between glyphs certainly should be preserved if they will matter to the text's users. Superscript letters and numbers, for example, might be omitted or normalized as plain letters or numbers in a business context, but in scholarly works they often play a crucial role in understanding the reading order of a text by linking footnotes with the pertinent section of the text's body. Similarly, where Gothic letters are used as sigla, for example, the 𝔓 (Gothic 'P') used for New Testament papyrus numbers, it is worthwhile to use a separate code point for this character, such as the "Fraktur P" designated by the Unicode consortium for mathematical use (U+1D513).

It is important to check that the OCR engine does not normalize the various styles of quotation marks, apostrophes, etc. in a manner that undoes this careful decision-making. Indeed, unless one tests the character accuracy herself, transformations like these will give the appearance of higher-scoring results, since they simplify the training and testing material. Grepping the source code for the string 'normal' or for the "right double quotation mark" character is a good way to search for and, if necessary, remove or alter these parts of the OCR engine.

Unicode normalization

An even more pervasive problem arises with the so-called normalization forms of Unicode text. Unicode offers four normalized forms, 'decomposed', 'composed' and the 'compatibility' variants of both of these. 'Composed' normalization uses the smallest number of characters to represent a set of glyphs; decomposed uses the maximum, and it orders them according to a set of rules. For instance, the grapheme ā can be represented as a single character, U+0101, or as a sequence of 'a' (U+0061) plus 'macron above' (U+0304), though it cannot be represented with

8 In total 342 characters were changed, 1.4% of all the ground truth characters.

sequence 'macron above' followed by 'a'. The former is the 'composed' form; the latter, 'decomposed'. A great number of Greek letters with their diacritics can be represented in either form. Therefore mixing composed and decomposed forms result in ground truth subject to the problem of ambiguity described above. To avoid this, OCR programs will apply normalization. However the form of normalization might not be best for our purposes, and often it is preferable to change the OCR code to use normalization that is to our advantage.

First we might wonder if there is a difference in performance between decomposed and composed forms, especially in ancient Greek texts, where this issue frequently arises. To study this and other questions, ground truth for OCRing Loeb editions was produced, comprising 3428 lines of text, both Greek and English, drawn from a variety of sources.[9] Its ground truth comprised 254 unique characters when using the Unicode composed normalized form ('NFC'), but this reduced to unique 155 characters when that ground truth was represented with the decomposed normalized form ('NFD'). (This is because each combination of a vowel and sequence of diacritics is represented by a different character in the composed form, multiplying the number of characters.) According to our reasoning above, it seems likely that the smaller character set will give better results; but perhaps in this instance the odd positioning of the characters, superimposed as they are, will make it more difficult for an RNN-based classifier to recognize them.

We can visualize the results with the chart in Figure 1. For the two training sets described above (differing only in their the normalization forms), classifiers were saved after each 100 steps, or iterations, of training. These 300 classifiers were used to generate results on two test pages: one that mainly contained Greek lines and another that mainly contained Latin-script (English) lines. The chart plots the character accuracy of these results on the y axis with the training iteration on the x axis, showing the difference in training progress between the two training sets for each image.[10] It can be seen that the decomposed form caused the training to progress more rapidly and reached a higher point of accuracy overall for both page images. It can also be seen that the composed form ground truth caused the training algorithm to twice dip down to a very poor

9 The page images, taken from the Internet Archive, Google Books and scanned by hand, varied in resolution from 300 to 600 pixels per inch.

10 This and the following charts use the logit scale on the vertical axis, where logit(a) = log(a/(1-a)). This is chosen to represent the greater challenge and importance of improvements as results increase towards 100%.

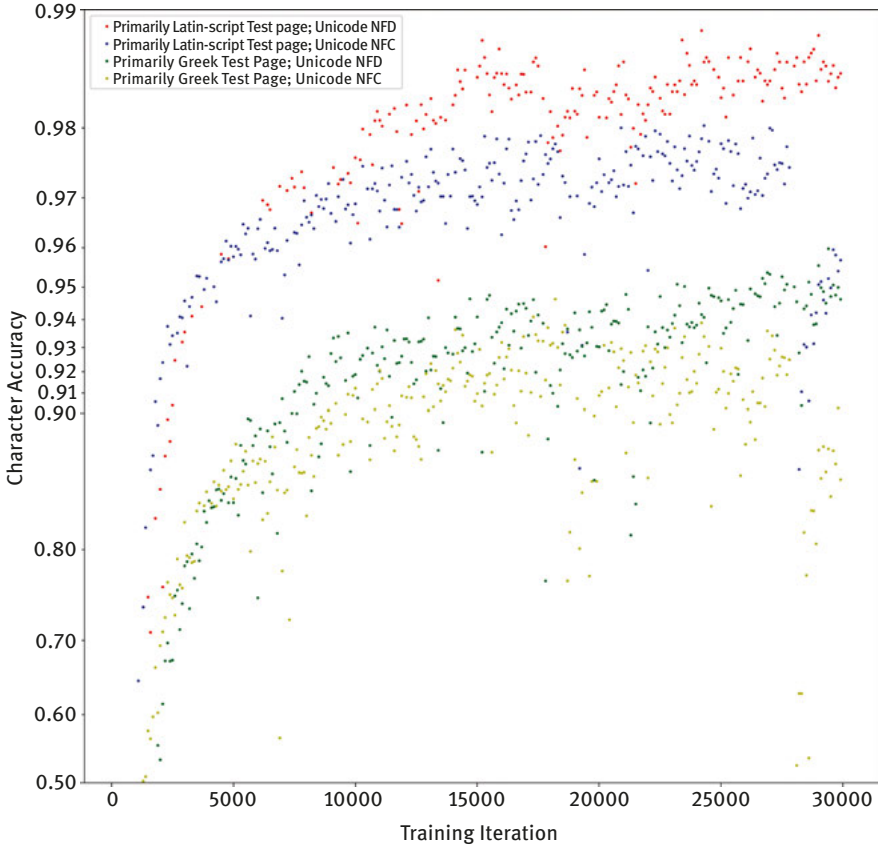

Figure 1: Scatter plots of training progress for ground truth in Unicode composed and decomposed normalized forms, tested against predominantly Latin-script and Greek-script pages. Progress is represented by character recognition accuracy versus training iteration.

score and consequently climb back from these. It also had greater volatility, with a standard deviation of accuracy at 0.25 compared 0.13 for the decomposed dataset. Finally, and most importantly, the highest character accuracy score of the decomposed training data operating on the primarily Greek page was 96.3%; the composed dataset only reached 94.7%. With the primarily English page, the decomposed training data reached a score of 99.3%, while the composed training data was 1.3% worse at 98%. (The relative improvement matters here, and these scores should not be compared to those mentioned earlier, since those used a far smaller character set.) Clearly, in all respects the decomposed normalization form is preferable, yet most OCR engines will normalize to the composed form and their code needs to be rewritten to do

decomposed normalization instead. Because of the already very high scores of the Latin-script results, the remainder of the paper will concentrate on pages that primarily contain Greek characters, though the observations should be understood as pertaining to both.

The Unicode 'compatibility' forms pose a different problem. These are intended to simplify processes like indexing and search, and so they convert unusual characters to ones that are similar but more familiar. For instance, all superscript numbers and letters are converted into their ordinary Arabic equivalents and the Gothic 'P' used for New Testament papyrus numbers becomes a plain Latin-script one. This can easily erase the decisions made regarding ground truth and its careful editing as discussed above. It can be difficult to detect this problem because one might assume that the errors are caused by misrecognition, not altered training data. Once again it is worthwhile to scan OCR source code for the pertinent keywords, in this case "NFKC" or "NFKD", to ensure that the engine does not perform this transformation at a low level.[11]

Improving training images

So far, we have considered the effect that ground truth has on our training results. But training involves both lines of text and images of those lines, and the proper manipulation of the latter can also improve recognition results. RNN approaches to OCR make the neural network responsible for choosing as well as integrating the features employed effectively to detect the characters. One effect of this is that the neural net can easily rely on features that are incidentally specific to the input images but not truly dispositive for other images.[12] The best way to avoid this is to increase the amount of training data, but generating ground truth is costly. Another approach is to slightly alter the training images, matching these to the already-produced ground truth. Among people working on image analysis and classification this is known as 'data augmentation' and a widely studied practice.[13] Many programming libraries exist for augmentation; I used the Python Augmentor library.[14] Because augmentation is particularly used in

11 As of this writing, the Ocropus engine performs a NFKC transformation on all training data and Tesseract 4 strongly defaults to the NFC form.

12 For example, if, by sheer coincidence, every line with an 'e' in the third position had a black pixel in the top left of the line image, a neural net's classifier might heavily weight this feature even though it will not prove effective.

13 (Mikołajczyk and Grochowski 2018).

14 (Bloice et al. 2017).

image analysis, it is important to use these libraries carefully, since many of their capabilities pertain to image recognition, but not to OCR. When training a neural net to recognize a car or a cat, for example, they rotate or crop the image drastically, since these objects should still be recognized despite these distortions. The line images for OCR training should not, of course, be changed these ways, but it can help training to add fuzzier or slightly sheared copies to the training set. I have found the `random_distortion` function of Augmentor to be effective.

Image augmentation is particularly effective when working with a small set of training data, but it can help in all circumstances. Figure 2 shows two scatterplots of character accuracy for each training iteration. Both use the best ground truth from Figure 1, namely the decomposed unicode normalization form, The series labeled 'without augmentation' matches each line of this

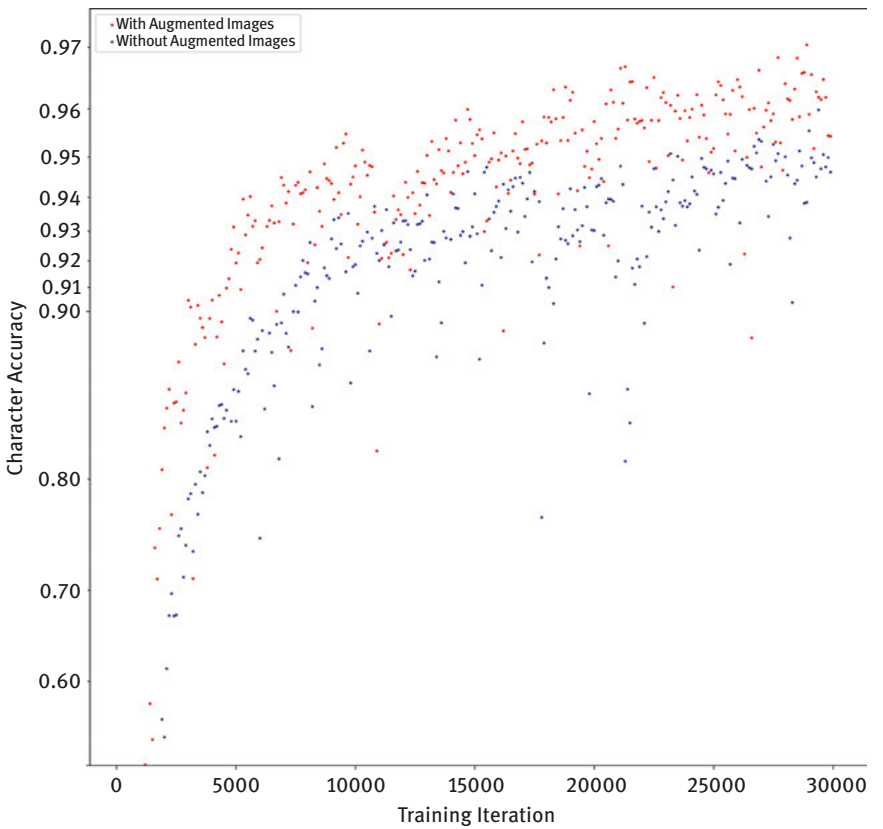

Figure 2: Scatter plots of training progress using unaugmented and augmented training images. Progress is represented by character recognition accuracy versus training iteration.

ground truth with a single image; that labelled 'with augmentation' matches each line with an additional three images, each randomly distorted copies of the original. The improves the maximum character accuracy score from the previous 96.2% to 97.0%.

Image augmentation takes place after binarization, the first step in an OCR process, which converts a grayscale or colour image into a binary, or black-and-white, one. In this step, a threshold of image darkness is set, beyond which a pixel is represented as black, not white. Because one part of a page can be more illuminated than another, a good binarization algorithm, such as `ocropus-nlbin` (a component of the Ocropus system), will vary this threshold depending on the general darkness of separate regions within the page image. Nevertheless, it is still possible to set an over-all binarization level, and this has an important impact on the appearance of glyphs. Darker binarization will cause glyphs to take up more space and possibly cause adjacent ones to join together into one connected component. This is especially true with Greek diacritics and their combining characters. Lighter binarization will make the strokes of glyphs thinner, eventually causing them to break into multiple connected components. Training should be performed at a binarization level that ensures easy reading and reasonable separation of glyphs. In the examples shown here, a level of 0.7 (70%) was used.

However, as Figure 3 shows, the result of this training did not perform optimally when evaluated against a test document binarized at the same 0.7 threshold! Instead, the three lighter steps, 0.4–0.6 all performed slightly better, with the 0.4 threshold yielding an improved maximum character accuracy of 98.2%, an improvement over the 0.7 threshold of 0.8%.

By comparing the binarized images and the resulting documents, we can see why. Figure 4 shows one accented character as it appears at the 0.4 and 0.7 binarization levels. The smooth breathing mark failed to be identified at the latter level. It appears that the significantly greater definition between the acute accent and the smooth breathing mark at the 0.4 level was an aid to the classifier, even though it had been trained at a higher binarization level.

It was noted above that neural networks require large volumes of training material, but we might wonder how much is enough. Figure 5 shows the results of four rounds of training 151 Greek and English characters, all subject to 30,000 rounds of training and evaluated against the same (primarily Greek-text) test document. In the first round, though, only ten pages of training data were used.[15] In this case the maximum character accuracy is 94.6%, and it can be seen that on four occasions the training accuracy dropped out. With three times as

15 Each page in this set comprises about 30 lines, or 1900 characters, of ground truth.

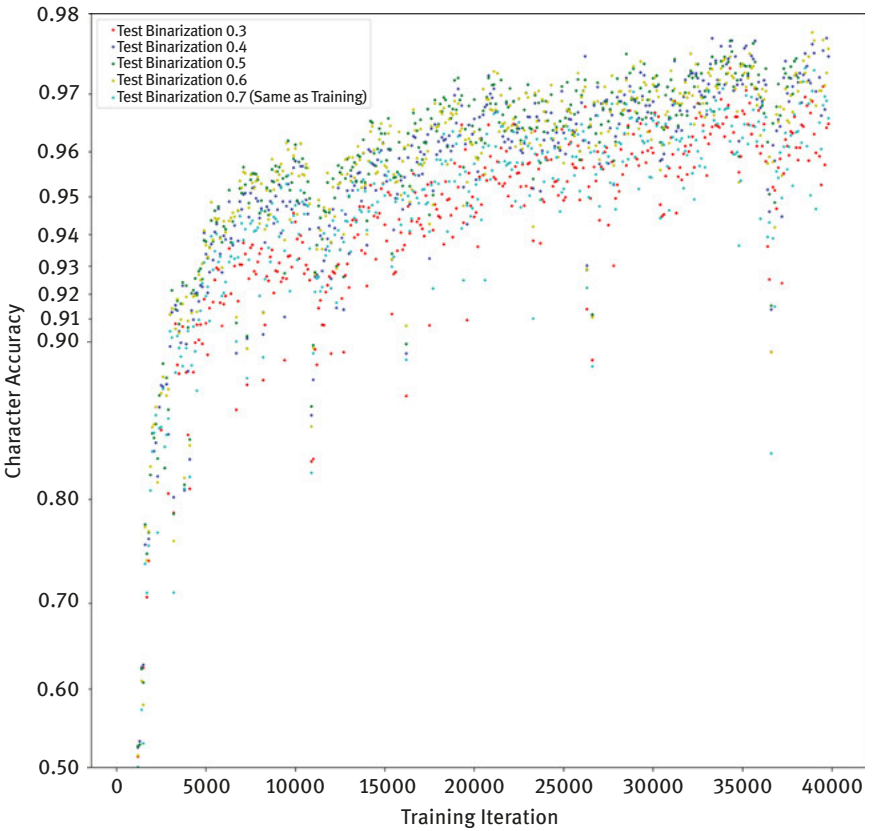

Figure 3: Scatter plots of training results using test images of varying binarization (darkness) and classifiers trained on images at a 0.7 level. Progress is represented by character recognition accuracy versus training iteration.

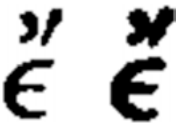

Figure 4: An accented Greek character at the 0.4 (left) and 0.7 (right) binarization level.

many pages of training material, the results improve considerably, up to 97% accuracy. Interestingly, the next increase, to fifty pages, does not provide a corresponding increase in accuracy: this session's maximum accuracy is 95.4%. Finally, with seventy pages of training material, an excellent training session occurs and the accuracy peaks at 98.8%. Considering that 96% accuracy is

our benchmark for profitable editing, it seems that one should seek out minimally around twenty to twenty-five pages of training material. (Data augmentation, as described above, might help improve results with minimal training material.)

A practical approach to amass large volumes of training material is as follows: begin with a few pages of transcribed ground truth and with its resulting classifier generate additional pages which can be corrected by editing. Iterating in this way often means that the last thirty pages of new training material is easier to come by than the first five. However, as Figure 5 shows, any arbitrary increase in training volume does not always guarantee improved results with any given page.

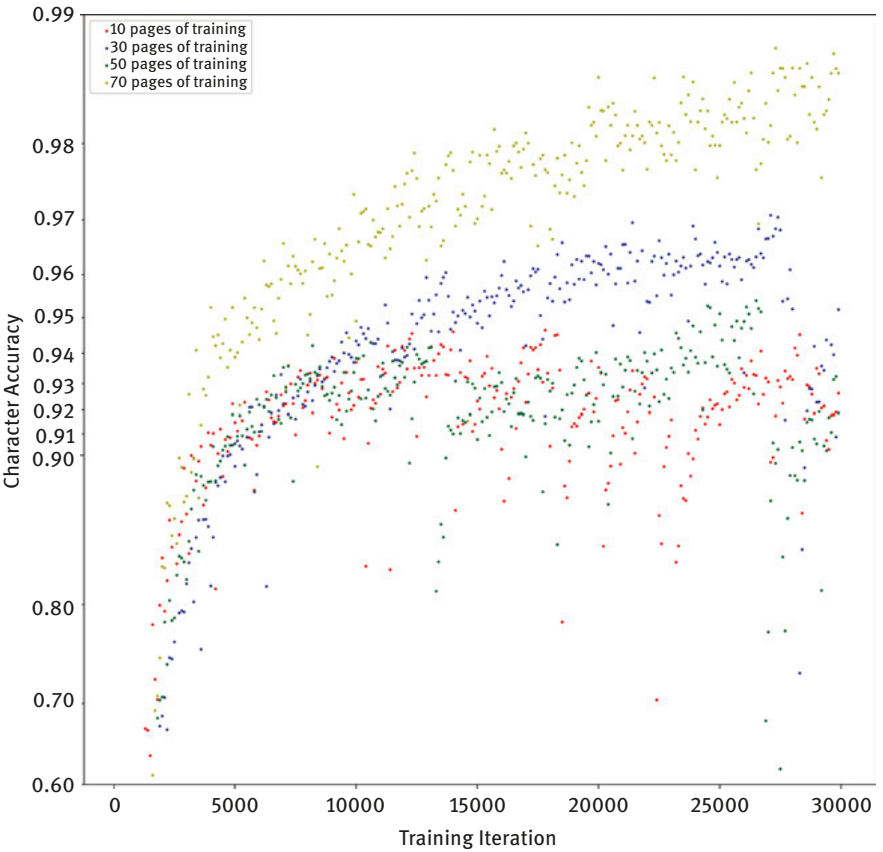

Figure 5: Scatter plots of training results using different volumes of training material. Progress is represented by character recognition accuracy versus training iteration.

Post-processing

Ocropus, like other OCR engines, outputs its text in a format identifying the region of the page image corresponding to the text's words and lines. In this case, the hOCR format is used, a variant of HTML.[16] The first two 'span' elements in Example 1 show how this format uses the HTML 'class' and 'title' attributes to indicate the type of text range and the coordinates of the corresponding rectangle on the page image. I have programmed a sequence of further post-processing steps as part of the Ciaconna OCR system that attempts to make OCR output as useful as possible for scholarly purposes.[17] These steps further modify the hOCR output, embedding additional information in HTML5-compliant custom data attributes. These begin with the string `data-` ("HTML 5.2" n.d.).

Often scholarly editions, like other texts, are laid out in a compact justified format, requiring the use of hyphens to split words between lines. Such hyphenation impedes some uses of a resulting digital text, such as indexing and search, as well as the production of new marked-up texts, the purpose of the First Thousand Years of Greek project. The first step of post-processing, then, in the Ciaconna system aims to reassemble hyphenated forms, adding information about the hyphenation as attributes on the two pertinent words' `<span>` elements. The program, named `dehyphenate.py` identifies hyphenated pairs in hOCR output even if the first word of the second line is preceded by a line number, a common event in scholarly editions.[18] As shown in Example 1, the `data-dehyphenatedform` attribute on the first of the two words provides the reassembled form, and the `data-hyphenendpair` and `data-hyphenstartpair` provide a unique and matching number for this hyphenation instance on the page. Finally, the `data-hyphenposition` attribute indicates after which character the hyphen appears. This information is useful because if the word is automatically corrected by a later process, the corrected parts can be applied properly to the two halves of the hyphenation.

Example 1: Output from dehyphenation and spellcheck routines of Ciaconna. Spellcheck output is indicated in bold face. This is generated from Herodotus et al. ([1908]). *Herodoti Historiae, recognovit breviqve adnotatione critica instrvxit Carolus Hu.* 2, V.86.2

```
<span class="ocr_line" title="bbox 89 1230 1815 1302">
    ...
    <span class="ocr_word" title="bbox 1296 1235 1815 1297" id="_47100321129176"
```

16 (Breuel and Kaiserslautern 2007).

17 (Robertson 2019).

18 At present, however, it does not properly handle the case where marginal text follows the last, hyphenated word of the first line.

```
     data-manually-confirmed="false" data-dehyphenatedform="ἀπαλλάχθησαν·" data-
     hyphenposition="7" data-hyphenendpair="2" data-spellcheck-mode="True"
     data-selected-form="ἀπαλλά-">
          Ἀπαλλά-
      </span>
</span>
<br />
<span class="ocr_line" title="bbox 150 1314 1818 1388">
     <span class="ocr_word" title="bbox 150 1319 412 1383"
     id="_47100321131408" data-manually-confirmed="false" data-
     dehyphenatedform="" data-hyphenstartpair="2" data-spellcheck-
     mode="True">
          χθησαν·
     </span>
     ....
</span><!-- end of 'ocr_line' →
```

Despite all efforts to improve raw OCR results, errors inevitably occur. In the examples above, the mixed-language classifiers dealing with around 155 glyphs produced a maximum accuracy of slightly above 98%, which means that nearly two out of every characters is misidentified. Often these are substitutions of similarly-shaped upper-case Latin-script and Greek letters or they are substitutions of one combination of diacritics for a slightly different combination, given how small these are printed. Some sort of post-processing, therefore, can be a very powerful tool to correct these obvious errors. In all cases this involves applying a so-called language model to the raw OCR output, and that output is made to conform to the model in some way. However, this is yet another step in the OCR process where we must be careful to attune our tools to the nature of the materials we are processing.

Imagine the case where we were certain that every word in the raw output was represented in a 'dictionary' file that comprised hundreds of thousands of unique words. With that certainty, we could produce extraordinarily effective post-processing with the following simple algorithm: output every input word that is in this dictionary, and for every input word that isn't, output the dictionary word that is 'closest' to it. This might be suitable for certain OCR proposes, but for ours, the digitization of textual editions, it would be very inappropriate. This is because editions include important forms that do *not* conform to such a dictionary but must not be corrected, such as words in obelized regions or in the apparatus criticus. Not only would such an algorithm increase the error rate in such instances, once a word has been erroneously 'corrected' to a form permitted in the language, it is much harder for an editor who knows the language

to identify and correct the error manually. Accordingly, our OCR process applies spellcheck with a very light touch, applying a list of common substitutions, such as the Latin-for-Greek errors described above, and it records the status of the spellcheck in the `data-spellcheck-mode` attribute shown in Example 1. The data encoded in this augmented hOCR format informs our OCR editing webapp, called Lace, a detailed description of which is beyond the scope of this paper. Lace assists the editor by colour-coding the spell-check status of each word, and most importantly it stores the manually corrected results so that these can be repurposed as training data.

The future

Thus far this system has generated 52,938,168 editable words of ancient texts, of which 10,237,171 are manually verified, providing an excellent basis for further classifier training. This paper suggests two approaches that will improve results further. The first is to train three classifiers for every font: one that recognizes Latin script and Greek together, and one each for only the Latin and only the Greek words in the ground truth. (We have the advantage here that in most scholarly material Greek and Latin letters are not combined in the same word. The exception is certain symbols in apparatus critici.) Each line would be recognized first with classifier trained with both scripts. Then the "Greek" words that don't pass spellcheck would be re-recognized by the Greek-only classifier, and the dubious "Latin" words re-recognized by the Latin-script-only one. We have seen that since each of these classifiers has to contend with only half as many glyphs as the comprehensive one, they are more accurate, with even the Greek classifier rising above 99% accuracy. Of course, it is essential that the algorithm distinguishing Greek words from Latin ones be very accurate; one mistake at this level will probably add many additional erroneous characters. The second improvement will come from OCRing each line at a variety of binarization levels and selecting the best results from each, a technique that was applied even more aggressively in the Rigaudon OCR suite described in Robertson and Boschetti (2017).

These improvements and others will be necessary if the texts of the so-called *second* thousand years of Greek are to be digitized satisfactorily. The much greater quantity of these texts makes it unlikely that commercial manual editing will be financially possible. Instead, we hope that interested scholars will undertake to correct this material manually, thereby establishing a corrected text for others and providing additional training material for improved classifiers. One hopes that

a character accuracy of better than 99% will encourage philologists to participate in such a project.

Bibliography

Bloice, M.D.; Stocker, C.; Holzinger, A. (2017): "Augmentor: An Image Augmentation Library for Machine Learning". arXiv [cs.CV]. http://arxiv.org/abs/1708.04680.

Breuel, T.M. (2008): "The OCRopus Open Source OCR System". Electronic Imaging 2008, 68150F – 68150F – 15. International Society for Optics and Photonics.

Breuel, T.M. (2019): Ocropus3. GitHub. https://github.com/NVlabs/ocropus3 (last access 2019.01.31).

Breuel, T.M.; Kaiserslautern, U. (2007): "The hOCR Microformat for OCR Workflow and Results". In: ICDAR 2007. Proceedings of the Ninth International Conference on Document Analysis and Recognition. Volume 2. Washington DC: IEEE Computer Society, 1063–1067.

Droettboom, M.; MacMillan, K.; Fujinaga, I. (2003): "The Gamera Framework for Building Custom Recognition Systems". JScholarship. Sheridan Libraries Staff Research. https://jscholarship.library.jhu.edu/handle/1774.2/44378 (last access 2019.01.31).

"HTML 5.2." n.d. https://www.w3.org/TR/html52/ (last access 2019.01.31).

Kiessling, B. (2019): Kraken. GitHub. https://github.com/mittagessen/kraken (last access 2019.01.31).

Kononenko, I.; Kukar, M. (2013): Machine Learning and Data Mining Introduction to Principles and Algorithms. Oxford: Woodhead Publ.

Mark, D.; Whistler, K. (2018): "UAX #15: Unicode Normalization Forms". May 10, 2018. http://unicode.org/reports/tr15/ (last access 2019.01.31).

Mikołajczyk, A.; Grochowski, M. (2018): "Data Augmentation for Improving Deep Learning in Image Classification Problem." In: 2018 International Interdisciplinary PhD Workshop (IIPhDW). Institute of Electrical and Electronics Engineers (IEEE), 117–122.

Rice, S.V.; Kanai, J.; Nartker, T.A. (1993): "An Evaluation of OCR Accuracy". Information Science Research Institute. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.7878&rep=rep1&type=pdf#page=9 (last access 2019.01.31).

Robertson, B. (2019): Ciaconna. GitHub. https://github.com/brobertson/ciaconna (last access 2019.01.31).

Robertson, B.; Boschetti, F. (2017): "Large-Scale Optical Character Recognition of Ancient Greek". Mouseion 14:3, 341–359.

White, N. (2013): "Training Tesseract for Ancient Greek OCR." The Eutypon 28–29, 1–11.

Wick, C. (2019): Calamari. GitHub. https://github.com/Calamari-OCR/calamari (last access 2019.01.31).

Wolpert, D.H.; Macready, W.G. (1997): "No Free Lunch Theorems for Optimization". In: IEEE Transactions on Evolutionary Computation 1:1, 67–82.