Does Handwriting Text Recognition Work for Damaged Archives?

Marco Roling MA
E-mail: info@marcoroling.nl

Abstract

Handwriting Text Recognition (HTR) is used on a large scale for digitized archives, but so far experiments have focused on manuscripts with a high standard of preservation and legibility. This paper describes some controlled experiments done on text samples with various types and degrees of archival damage, in order to assess their suitability for HTR. Also some ideas are expressed about how to predict the success of HTR when it is applied to large volumes of scans. Lastly, it is suggested to enhance scans before subjecting them to the HTR process, with the intention to further improve the overall quality of automated transcriptions.

Published: *March 2020*
Keywords: *HTR, Ink corrosion, Manuscript, Image processing, Machine learning, Archival damage, Automated transcription*

1. HTR and the promise of automated transcription

In recent years, Handwriting Text Recognition (HTR) of manuscripts has dramatically improved. The machine learning capabilities of computers, using neural networking, pattern recognition and probability algorithms, have almost reached maturity and are starting to be applicable to larger volumes of scans of handwritten documents. There are several academic research groups around the globe who are active in this field. One of them is READ-coop, a European cooperative institution that originated from the University of Innsbruck, and that has developed the end-user tool called Transkribus over the last years.[1] Transkribus runs on a home computer with the real processing done remotely on an Austrian server. Scanned images of handwritten documents can be uploaded and transcriptions automatically generated for these pages using a previously trained HTR model (figure 1).

Automated transcription results come close to perfection, with only some margin of error. Results can be used immediately in a full text search engine, can be analysed linguistically or processed for named entity recognition. In this way manuscript archives can become more accessible and usable to scholars without them having to take extensive palaeography classes first and to read page by page to locate historically relevant information.

Besides using automated transcription, anyone with manuscript scans can start and train an HTR model from scratch or on top of an already existing one. Scans with matching ground truth transcriptions can be fed into the computer learning engine and the resulting trained model can again be shared with others. A model can be based on any number and any type of documents and can be any language. Even printed text can be used (until recently the sole domain of Optical Character Recognition, OCR), which is useful as printing goes back to the 16th century and old fonts might not be recognized in modern OCR so easily. Consistent character writing such as in medieval texts, or Arabic or Japanese scripts, are other examples of texts that can be used for training an HTR model. The engine is robust and can handle some amount of image noise, and lines do not need to be perfectly straight and consistently written. Several writers can be in the same collection of manuscripts, which can improve the model even further making it better trained and applicable to a larger variety of sources. Nowadays, HTR can be regarded as a promising tool for automated conversion of handwritten text into digital text.

[1] https://read.transkribus.eu/

© 2020 mroling

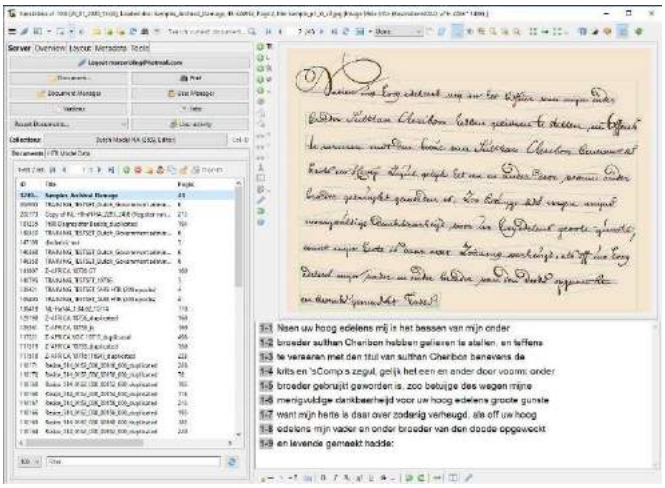


Figure 1 – HTR interface showing a text sample and automated transcription

1.1 The challenge of transcribing damaged archives

Today, more and more institutions with manuscript archives are eager to upscale their digitization because automated transcription is now within reach, and publication online with full digital text search is the next stage in providing access to the general public. Some digitization projects, like for example the ones of the National Archives of the Netherlands and the City Archives of Amsterdam, have already resulted in millions of scanned documents suitable for applying HTR, and transcription has already begun. Projects to fully disclose the texts are progressing, sometimes with the aid of dozens of enthusiastic crowd sourcing volunteers to make the outcome even better.

The archives used for training the models have mostly been the well preserved ones. This makes sense, as new technology at first needs to be proven under optimal conditions before starting to experiment with sources that are of a lesser quality. Some manuscript archives may have suffered from all kinds of damage over the past centuries, even leading to partial loss because of flooding, fire, insects or ink corrosion. While HTR obviously can't recreate lost archives, some forms of archival damage may still yield an acceptable digitized text.

A good example of the differences in preservation between parts of the same archive is that of the Dutch East India Company, formed during the 17th and 18th century. This large archive was not only created in Holland, but also in several colonies and trading places around the world like present day South Africa, India and Indonesia. The archives in Indonesia, for example, were stored for centuries under tropical humid conditions and this led to all imaginable forms of archival damage. Not everything suffered, and some parts are surprisingly well preserved, but the legibility of the texts is sometimes severely reduced because deterioration processes continue to affect the original sources. Three of these processes are discussed here in relation to HTR. Paper discoloration from almost white to dark brown; the fading of the ink from dark brown to almost invisible; and ink corrosion that leads to blurring of the text (figure 2). In combination they can be really devastating, causing the text to become fully lost to the eye, or completely bleed into the paper as a big ink stain.

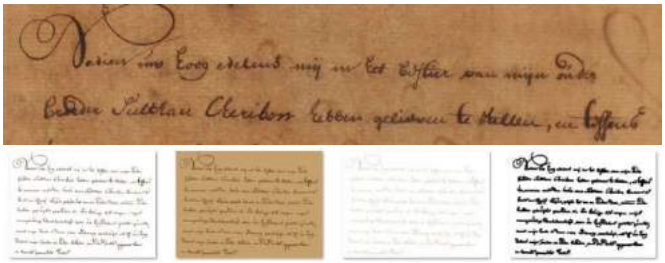


Figure 2 – Original text and constructed samples with archival damage

Much historical and also practical research has been done on the physics of paper and ink, and how they interact[2]. The specialized craft of paper making was done only in few places in the Dutch Republic, using wind and water powered mills, and linen textiles as raw materials. Holland, and especially Amsterdam, were very active in printing books, newspapers, maps and flyers and this demanded a constant influx of paper. Local production volumes were insufficient most of the times and even paper mills in France and Germany were commissioned to supply for the Dutch market.[3]

Besides printing, writing and packaging also required paper, but of a different texture and density. It can be expected that all these different kinds of paper were also transported to Batavia, in the former Dutch East Indies, and used for writing and archiving. Paper sent to Batavia was not always of the best quality, and it took a long time to get there under less than optimal conditions. The ink was most likely also transported, as the ingredients for making iron-gall ink originate from different places in Europe and the Levant, and ink-making and selling was a thriving line of business. Bird feathers used for writing may have been locally produced around Batavia, but this is unknown. In Europe there was a specialized trade in goose and raven feathers[4]. One can imagine that the quality of paper, ink and writing feathers all had an influence on the handwriting and the possible susceptibility of the finished manuscript to various forms of decay over time. The present-day state of these manuscripts is the result and has to be dealt with when applying modern HTR.

[2] https://irongallink.org/igi_index.html

[3] The author attending a lecture (2020) by Birgit Reissland and Frank Ligterink on the drawings by Rembrandt.

[4] See also: https://www.nicas-research.nl/

The question arises if acceptable digital transcription results are achievable even if these archives have suffered archival damage.

1.2 Using a Dutch language model for HTR

For a number of years, several institutions in the Netherlands and Belgium have been working on HTR models based on the Dutch language and Dutch archival sources. The National Archives of the Netherlands (NAN) published a public HTR model as recently as January 2020. NAN holds a large part of the archives of the Dutch East India Company, and has used a selection of 4810 scanned pages from the inventory numbers 7528-9540 for training an HTR model.[5] This model is known as 'NAN_GT_M11+' in Transkribus. It is based on 17th and 18th century source material and is expected to give acceptable results, suitable for application to similar manuscripts housed at the Indonesian counterpart. The so-called Character Error Rate (CER[6]) of the model, measured on the ground truth examples, was reported to be 5.3% and 7.3% on a separate random sample set. This means that well over 90% of the text is correctly transcribed automatically, which is acceptable for immediate full text search. This model was used in the HTR experiment conducted by the author.

The experiment focused on three different types of archival damage, namely paper discoloration, ink fading and ink corrosion. These were chosen because they were expected to have a measurable effect on the outcome of HTR processing. This hypothesis was tested using the mentioned model with a known quality and performance level against a series of constructed samples that reflect progressive archival damage. A base sample was first manually created and by varying different image layers and aspects, another 42 test samples were created to reflect different stages of degradation. Images were created using regular end user software. Six categories were defined to scale the degree of degradation, from 'some', 'mild', 'moderate', 'serious', and 'severe' to 'extreme' archival damage. These categories were based on the Metamorfoze archive damage atlas in which many more types are described.[7]

The base sample (figure 3, first sample on the left) was recreated using a snippet of text copied from the published archives of the Dutch East India Company on the website of 'Sejarah Nusantara'[8] of the National Archives of Indonesia in collaboration with The Corts Foundation[9]. This collaboration led to the online publication of 1.1 million scans, all from the 17th and 18th century. The sample snippet used for this research was originally copied from inventory number 2567, page 304, of the sub archive of the Daily Journals of Batavia Castle by the High Government.[10] The snippet, dated May 3, 1735, is actually a Dutch translation of an originally Javanese letter from the Sultan of Cheribon to the Governor General of Batavia, with a very polite thanks for granting him the title of Sultan after succeeding his father and older brother. The nine lines of text contain 85 words and 435 characters and although short, this is regarded sufficient for the experiment as the comparison between the outcomes of all samples is the focus of the analysis.

Figure 3 – Constructed test samples with increased paper colouring

2. Running HTR on constructed samples

Testing the model first against the base sample gave a benchmark CER to be used as a reference for the other constructed samples. The model had an outstanding performance, with a low CER of 1.6%. Only seven characters were not recognized properly (differences in capitals were not regarded as errors and ignored). This first test proved that the model was well suitable for the experiment. Next, samples reflecting degradation through paper discoloration from almost white to brown were tested (figure 4, column 'browning'). The results showed some variation in CER, a little higher than the benchmark, but this could not be regarded as really significant. It seemed that the browning of the paper hardly influenced HTR.

To continue, samples with progressive stages of ink fading were tested. Results were similar and fading did not seem to have a real negative effect on HTR. But when paper discoloration and ink fading were combined, an interesting break point showed up. Perhaps not surprisingly, when the paper had become severely brown and the ink severely faded the difference in contrast between the two was reduced to almost zero and HTR failed completely. It can be concluded though that HTR is robust as long as there is some difference in contrast between paper and ink. It is probably fair to say that if the human eye can still distinguish between the text and

[5] Keijser [2]

[6] CER is the percentage of characters that differ from the known perfect ground truth transcription, initially used in the model training. Besides CER there is also a WER (Word Error Rate) that reflects the number of words that differ. Both indicate a quality measure for the accuracy of the HTR model

[7] See: schadeatlas-2010, https://www.metamorfoze.nl/

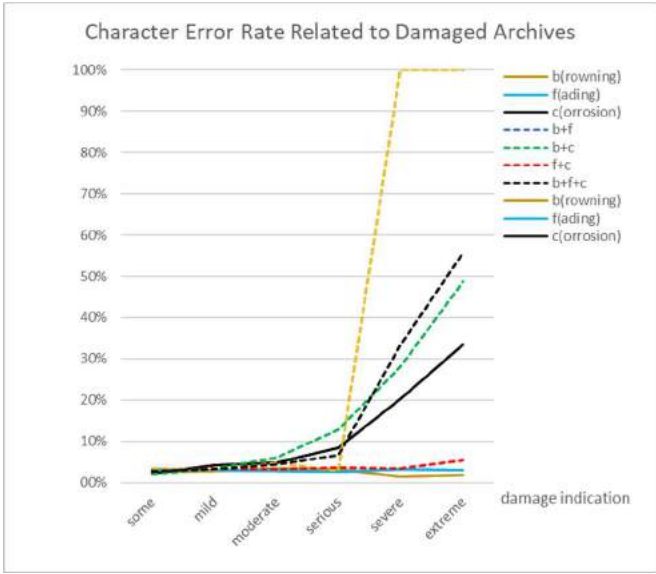
[8] https://sejarah-nusantara.anri.go.id/

[9] https://www.cortsfoundation.org/

[10] Original scan can be found on: https://sejarah-nusantara.anri.go.id/marginalia_search/

the paper background, HTR will also be able to recognize the text without significant loss compared to the original.

When testing samples with worsening ink corrosion, HTR results showed a dramatic increase in CER when a 'serious' degree of damage was present in the sample (figure 4). The CER graph started to show a steeply rising curve and HTR yielded poor results. It needs to be pointed out, however, that the used model was not trained to recognize blurred characters and words. When taking this into account, the results are actually not that bad, and up to a 'moderate' degree of ink corrosion HTR still leads to automated transcriptions that may be acceptable for full text searching.



CER result	b(rowning)	f(ading)	c(orrosion)	b+f	b+c	f+c	b+f+c
some	2.5%	3.2%	2.3%	3.4%	2.1%	2.8%	2.8%
mild	2.8%	3.0%	4.1%	3.0%	3.4%	3.4%	3.2%
moderate	3.4%	2.8%	4.8%	4.8%	6.0%	3.2%	4.4%
serious	3.2%	2.5%	8.5%	2.8%	12.9%	3.7%	6.7%
severe	1.4%	3.2%	20.2%	100.0%	28.0%	3.4%	33.3%
extreme	1.8%	3.0%	33.6%	100.0%	48.7%	5.5%	55.6%

Figure 4 – CER results of HTR on scans with combined archival damage in different stages

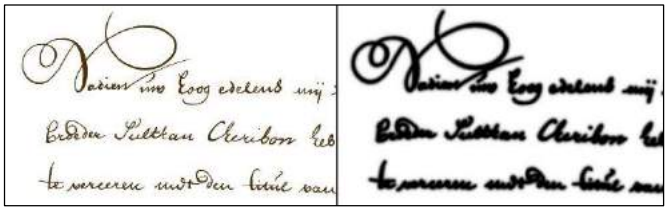


Figure 5 – Constructed samples (extreme ink corrosion on the right)

Combining paper discoloration and ink fading with ink corrosion led to more difficulties. Creating the test samples in the first place was a challenge, and applying HTR gave results that needed some additional analysis. In any case it appeared that a rise in ink corrosion accelerated the increase of the CER. In fact, the ink corrosion factor dominated the simultaneous effects of paper discoloration and ink fading in the results. The worst test sample (see 'b+f+c' column with 'extreme' archival damage) eventually gave a CER of 55.6%. This result cannot be used for full text search. The initial ground truth transcription and matching HTR results are given here to illustrate this.

Ground truth original transcription text

Nadien uw hoog edelens mij in het bestier van mijn ouder broeder Sulthan Cheribon hebben gelieven te stellen , en teffens te vereeren met den titul van Sulthan Cheribon benevens dese krits en sComp: Zegul, gelijk het een en ander door voorm: ouder broeder gebruijkt geworden is, Zoo betuijge des wegen mijne menigvuldige dankbaarheijd voor uw hoogedelens groote gunste, want mijn herte is daar over Zodanig verheugd, als off uw hoog edelens mijn vader en ouder broeder van den doode opgeweckt en levende gemaekt hadde:

HTR result transcription text (CER 55.6%)

Naen zen hoog elens wij ten het teste van mijen aeden
beln sutlan Clerilora hebten geteenen te sallen, : te
te twaaan achb den : waar alulem: k s
kner vn pe keijent ggel het n no zindar dara ame m
boeken getangis: geerha is, zer bnge ds wegem
vatrige 20 akaaahejjd en Cosheekens geente geemt:
waar mijt loot son vaer Petnij verlnge, as en kog
Eete mijn an Er beelden van de dts opg
vonde ccasil haer

2.1 HTR predictive modelling

In the previous section, we have seen how HTR performs when applied to constructed samples of handwritten texts with increasing archival damage. It appeared that the colour of the ink and paper separately hardly influenced HTR but the combination did, as decreasing contrast between ink and paper led to them becoming indistinguishable. The complex colour gradients in ink corrosion also turned out to have a major impact. In order to assess if scans of manuscripts lead to acceptable results in HTR processing, the next step would be to find some automated computational way to perform measurements on individual scans.

Predicting the success of HTR requires more detailed understanding of the image characteristics in terms of colour, contrast and gradients. In a first attempt, colour and grayscale histograms were computer generated. In the colour histogram the image colours were separated in three basic channels of red, green and blue and plotted in a 3D cube visualization with their original colours. Before visualizing the sample image this way, the colour data was filtered to reduce noise, some of which originated from JPG compression in the image source file. HTR has already been proven to be robust to some degree of noise, and filtering made the resulting graph more interpretable. For the second grayscale histogram the unfiltered image data was used to show all colour gradients.

The first sample discussed here was manually constructed. The sample showed some paper discoloration and slight ink fading, but no noticeable ink corrosion (figure 6). The colour histogram of this sample showed distinctly separated clusters of a light and darker brown. The light brown cluster included a large marker that clearly indicated the discoloured paper background. In the most ideal case, when the paper colour would have been white and the ink colour black, these clusters would be even further apart in the histogram towards the extreme left lower and right upper corner. When converting the original colour image into grayscale, it showed distinct peaks in the histogram, but the expected bimodality turned out to be trimodal. probably as a result of JPG compression noise being present in the image.

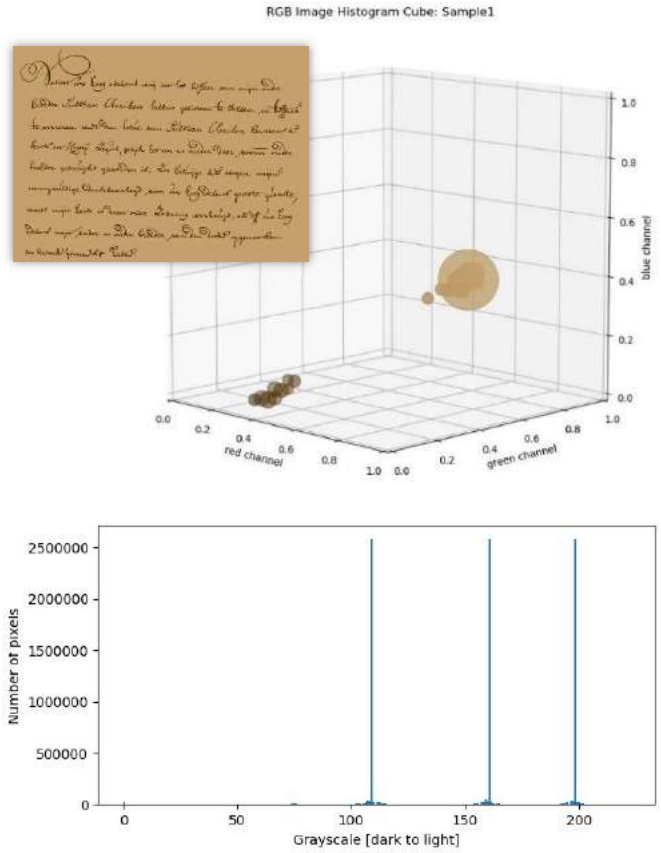


Figure 6 – Constructed sample with mild archival damage, with colour and grayscale histograms

It can be argued that samples with distinct colour clusters are at least theoretically ideal for HTR, as the histogram indicates a clear distinction between ink and paper and suggests optimal legibility for man and machine.

Copied from a scan of the original manuscript, the second snippet turned out to be much more complex than the manually constructed one discussed before. The snippet also showed other signs of archival damage processes. For example, the text on the back was shining slightly through to the front side due to the ink corrosion process that had progressed through the paper (so called: ink bleeding). Although still well readable for the trained historian, the colour histogram of this sample (figure 7) showed a continuous gradient in the colour spectrum from ink to paper, very different from the previous example. The colour histogram revealed no clear distinction between ink and paper. The grayscale histogram again showed a trimodal graph, but with the peaks much more connected. The width of the farthest left one (indicating the dark text part) cannot be simply interpreted as a grade for the ink corrosion. Based on these first observations it cannot be concluded that histogram data is a good candidate for use in HTR predictive modelling.

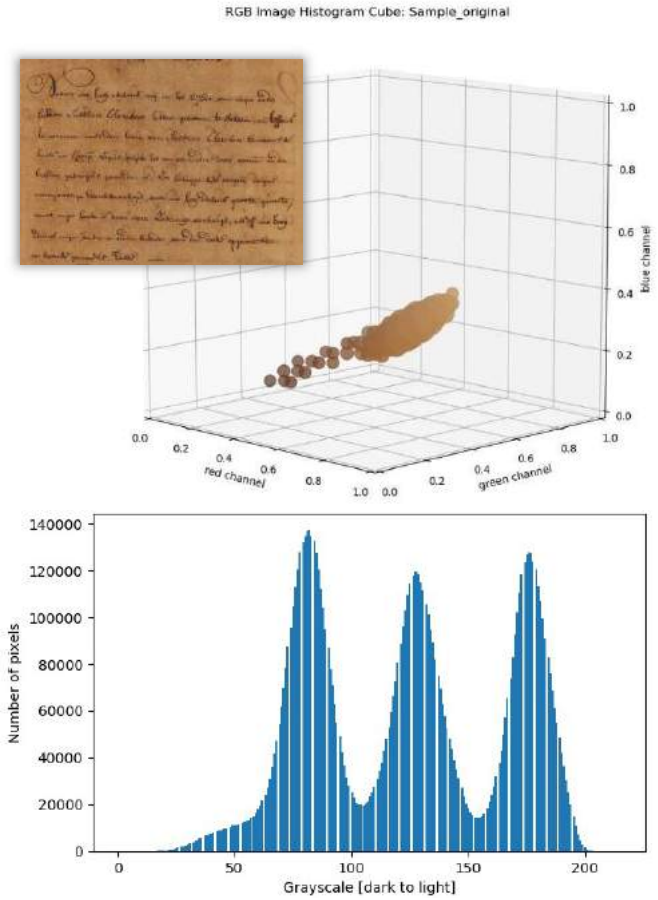
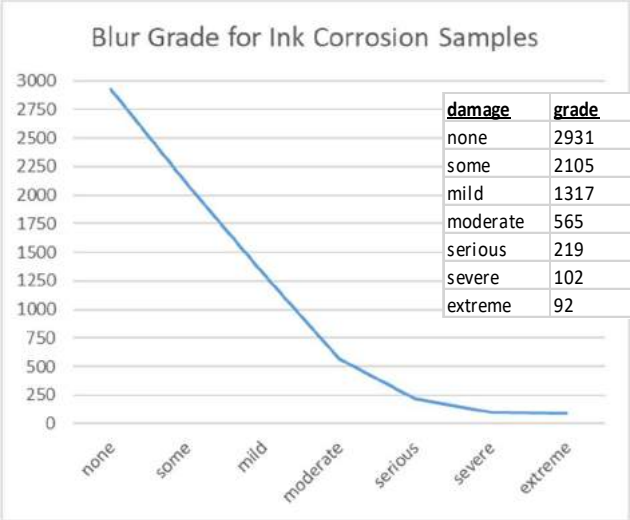


Figure 7 – Real-life sample with more complex archival damage, with colour and grayscale histograms

Alternatively, ink corrosion might be approached similar to image blurring. The handwritten text can be regarded as a picture that is somewhat out of focus. The blurring of an image can be assessed with a computated value, using existing mathematical calculations (the one used here is known as 'Lapacian'). Calculation was done for all the constructed ink corrosion examples. The plotted graph showed a clear downward trend line in which high values are to be interpreted as a low corrosion grade and vice versa (see figure 8).

Calculating the blurring for the original image, with even less than 'some' ink corrosion, resulted in a grade 207. This value seems to conflict with the 'serious' ink corrosion grading reflected in the graph. This could indicate that the constructed samples do not accurately represent real samples, which tend to be much more complex. Testing a few other real manuscript snippets with different grades of ink corrosion did suggest, however, that the trend line could be similar, although the actual value range differs, starting at a much lower maximum than shown in the graph below. For now it seems promising that blur grading is a candidate for use in HTR predictive modelling. More research is to be done in order to investigate other existing algorithms that might be suitable as well.[11]



damage	grade
none	2931
some	2105
mild	1317
moderate	565
serious	219
severe	102
extreme	92

Figure 8 – Blur calculation for constructed examples of ink corrosion

2.2 The effect of image enhancement on HTR

It is expected that digital elimination of the paper colour and any traces of back side text will increase the automated transcription quality of a scan image. Cleaning up scanned images is possible to a large extent, as Schomaker and He (2019) have already proved. Neural network deep learning algorithms can already reduce most of the image noise as an intermediate enhancement step, thus leaving the frontal texts with much better contrast and less colour gradient.[12]

In addition, digital reduction of ink corrosion effects in particular can be tested as well. In order to illustrate its potential, one of the constructed examples with 'extreme' ink corrosion (shown before in figure 5) was converted into grayscale and subsequently subdued to so-called binarization with a manually chosen threshold (figure 9).[13] The threshold used for this example was lowered as much as possible to maximize the reduction of ink corrosion (eventually chosen to be only 3 on a scale to 255). In the HTR result, the CER was down from 33.6% to 2.8%, an overwhelming improvement and with an acceptable automated transcription. Although the constructed example used here may not fully represent real examples with extreme corrosion, it is still worthwhile to further investigate the use and implication of this method.[14]

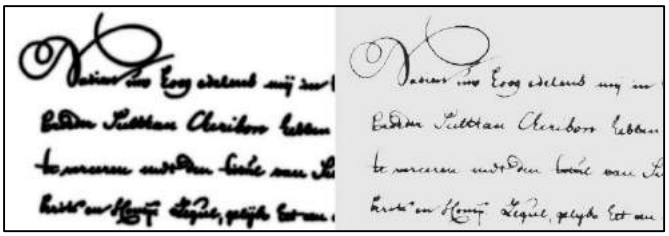


Figure 9 – Original sample (left) and image-enhanced sample (right)

Summary

Handwriting Text Recognition (HTR) has developed into a useful and applicable machine learning tool to automate the transcription of manuscripts into digital texts. Although many archives are well preserved and highly suitable for HTR, some archives have suffered damage for centuries because of suboptimal preservation conditions and lower quality materials like paper and ink. It is important to assess whether these archives are also suitable for HTR. Digitizing millions of pages is time and resource consuming and only meaningful if the resulting texts are legible. This raises the question to what extent HTR still produces acceptable results for damaged archives. To investigate this, controlled experiments were conducted by the author using an excellent HTR model created by the National Archives of the Netherlands, and a series of manually constructed test images that reflect the increasing damage from paper discoloration, ink fading, and ink corrosion. Combinations of these processes were included in the sample set comprising a total of 42 images.

From the results, it can be concluded that ink corrosion has by far the most negative influence on HTR, when compared to paper discoloration and ink fading. Actually, the last two hardly influence HTR results individually. However, when they are combined, there is a clear break point beyond which there is almost zero contrast between the ink and the paper, and the text is no longer visible. On the positive side, HTR is sufficiently robust to handle a moderate degree of ink corrosion, even though the used model was not originally trained with degraded archival material. It may therefore be possible to improve the model by training it with additional scans that have a higher degree of corrosion. But it cannot be

[11] Rosebrock, A. 2015, see: https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/

[12] Schomaker, He [1]

[13] https://en.wikipedia.org/wiki/Thresholding_(image_processing)

[14] See appendix 2 for the code used in this experiment

assessed here if this would substantially improve the model, lowering the CER on the corrosion samples.

The author has some doubt about the accuracy of the constructed samples that combine two or three types of archival damage. An attempt could be made to improve the test samples, but at the moment it is not regarded likely that this would change the conclusions drawn so far. Additional research is advised focusing on other forms of archival damage that has affected manuscripts partially, such as burn marks, water stains, and small insect holes.

Prediction of possible HTR success over a large volume of scanned manuscripts requires some form of computation that deals with recognizing the contrast difference between ink and paper on the one hand, and ink corrosion effects on the other hand. At first glance, using colour histograms appears to be promising, but only seems to be working for the constructed sample images with limited colour gradients. Real samples are much more complex and even extreme reduction of colour noise from the image data does not lead to a clear difference in the graph between the frontal text and paper background. Calculating blur grades does, however, seem to be effective and exact, although there is a difference between the constructed and real samples in terms of the range of grading values. Additional research with many more real samples is needed to verify if this method is indeed useful in practice. Further research is advised here in general to try and find other reliable computational methods for use in HTR predictive modelling.

Finally, some effects of scan image enhancement were tested with HTR. Apart from the automated removal of background colour and noise using deep learning algorithms, the digital reduction of some of the ink corrosion effects was found to contribute much to the quality of the resulting automated transcription. Applying image conversion to grayscale and subsequent binarization with a chosen threshold is promising and can be further optimized, maybe in combination with additional machine learning capabilities. Other computer vision and machine learning algorithms can be researched for their potential use in image enhancement of handwritten documents.

The hands-on experiments as presented in this paper were conducted in order to better understand and predict the potential of HTR for archival manuscripts with some degree of archival damage. The presentation aims to reach a broad audience, in particular archival institutions, museums and libraries. It is clear that much more research needs to be done on this most interesting and challenging topic. Progress is ongoing, hopefully leading to additional end user tools that will assist owners of manuscript archives in their assessment and preparation for successful and large scale application of HTR.

Acknowledgements

Special thanks to Frank Ligterink for providing useful python code samples, Monika Cunnington for reviewing and editing, and finally the READ team for creating the Transkribus HTR tool in the first place.

References

[1] Schomaker, L., He, S. Pattern Recognition Volume 91, July 2019, Pages 379-390, Online: https://www.sciencedirect.com/science/article/abs/pii/S0031320319300330

[2] Keijser, L., NAN. (2020). Scans and transcriptions of the VOC and the Haarlem notarial deeds archives (Version 2.1) Online: http://doi.org/10.5281/zenodo.3672406

Appendices

[1] Computer generated colour histogram – code example
[2] Computer generated grayscale histogram and image enhancement – code example
[3] Computer generated image blur grading – code example

Appendix 1: Computer generated colour histogram – code example

```
###############################################################################
# It shows a histogram, where RGB channels are separated along the axes
# Any JPG image is read and each pixel colour evaluated and counted
# The resulting cube shows the spread of colours, and their relative counts (dots size)
#
# This plot was developed with the aim to support analysis of scans of handwritings (ink on paper)
# to see if a histogram would be useful somehow to mathematically calculate and assess the application of
# automated text recognition.
#
# This code can be run directly in a Jupyter notebook
# Installation prerequisites are: python, pip, ipython jupyter, numpy, matplotlib
###############################################################################

%matplotlib notebook

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from datetime import datetime

# Set basic input parameters here for further processing
file_img_input = 'Sample_original'      # define image file name
nbins = 32                              # define number of bins to be used
thresh_low = 5000                       # define num_pixels minimum per bin for final cube

# Define input and output image
imgname_input = file_img_input + '.jpg'
plot_log = file_img_input + '_log.txt'
plot_output = file_img_input + '_histcube.jpg'

# Open logfile for writing
file = open(plot_log,'w')

# Read image as a 3d numpy array
img = plt.imread(imgname_input)

# transform values as float
img = img.astype(np.float32) /  256

# make a list with rgb values based on the array
rgb_list = img.reshape(-1, 3)

# print and write to log
img_height = img.shape[0]; img_width = img.shape[1]; img_channels = img.shape[2]
line = '[IMAGE]' + '\n' + 'image_name=' + file_img_input + ' / dateTime_processed=' + str(datetime.now()) + '\n'
print(line); file.write(line)
line = 'dimension HeightWidthChannels=' + str(img_height) + '/' + str(img_width) + '/' + str(img_channels) + '\n'
print(line); file.write(line)
line = 'num_pixels=' + str(len(rgb_list)) + ' / upper_left_pixel_rgb=' + str(img[0,0]) + '\n'
print(line); file.write(line)

# create a 3D histogram
hist, edges = np.histogramdd(rgb_list, bins=nbins, range=[[0,1], [0,1], [0,1]])

# make a list of of the 3D histogram
hist_list = hist.flatten()

# get the values of the non-zero bins
indices = np.argwhere(hist_list > 0).flatten()
h = hist_list[indices]

# start cleanup of the histogram, print and write to log
max_num_px_perRGB = int(max(h))
line = '[BEFORE CLEANUP] num_bins=' + str(h.size) + ' / max_num_px_perRGB=' + str(max_num_px_perRGB) +
'\n'
print(line); file.write(line)
```

```
# reduce noise by applying a lower threshold on the bins
thresh_low_indices = h < thresh_low
h[thresh_low_indices] = 0

# assuming that the highest bin contains the background colour, it is reduced here from the plot
# BGcolor_index = h == max_num_px_perRGB
# h[BGcolor_index] = 0

# end cleanup of the histogram, print and write to log
line = '[AFTER CLEANUP] threshold_px_perRGB=' + str(thresh_low) + ' / bins_set_to_zero=' + str((h == 0).sum())
+ '\n'
print(line)
file.write(line)

# fill 3D grid
mgrid = np.mgrid[0:nbins, 0:nbins, 0:nbins] / nbins
mgrid = mgrid.transpose(1, 2, 3, 0)
mgrid = mgrid.reshape(-1, 3)

# mark colors
m_colors = mgrid[indices]

# define the rgb coordinates of each bin
m_r, m_g, m_b = mgrid[indices].T

# redefine marker size, in order to scale down larger markers
m_s = h**(1/1.8)

# prepare to plot the data cube
fig = plt.figure(figsize=[10, 10])                    #set the figure size of plot
ax = fig.add_subplot(111, projection='3d')            #set the type of plot
ax.set_xlabel('red channel')                          #set the label of the x-axis
ax.set_ylabel('green channel')                        #set the label of the x-axis
ax.set_zlabel('blue channel')                         #set the label of the x-axis
ax.set_title('RGB Image Histogram Cube: ' + file_img_input)    #set the label of the x-axis
ax.set_xlim([0,1]); ax.set_ylim([0,1]); ax.set_zlim([0,1])          #set the scale of the x,y,z-axis
ax.view_init(elev=10, azim=-45)                       #set the initial elevation and azimut of the plot

# plot the data cube
ax.scatter(m_r, m_g, m_b, s=m_s, c=m_colors)

# save the data cube image to disk
plt.savefig(plot_output)

# Close file for writing
file.close()
```

Appendix 2: Computer generated grayscale histogram and image enhancement – code example

```
##############################################################################################
# It shows another histogram, where RGB is converted into grayscale
# Next the image is filtered using a threshold, leaving dark text only
#
# This code can be run directly in a Jupyter notebook
# Installation prerequisites are: python, pip, ipython jupyter, numpy, matplotlib
##############################################################################################

%matplotlib notebook

import cv2
import numpy as np
from matplotlib import pyplot as plt

# Set basic input parameters here for further processing
file_img_input = 'Sample1'      # define image file name
threshold = 100                 # define grayscale threshold for cleanup (0=black, 255=white)
```

```
# Define input and output image
imgname_input = file_img_input + '.jpg'
imgcolor = cv2.imread(imgname_input)
imggray = cv2.imread(imgname_input, 0)
imgname_output = file_img_input + '_enhanced.jpg'

#decrease size and show image for reference in a popup window
scale_percent = 25
width = int(imgcolor.shape[1] * scale_percent / 100)
height = int(imgcolor.shape[0] * scale_percent / 100)
dsize = (width, height)
img_resized = cv2.resize(imgcolor, dsize)
cv2.imshow('original color image: ' + imgname_input,img_resized)
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image

#convert to grey scale image, apply binarization and show histogram
ret, imgf = cv2.threshold(imggray, threshold, 255, cv2.THRESH_BINARY)
#ret, imgf = cv2.threshold(imggray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU) #OTSU is not used`here

plt.subplot(3,1,1), plt.hist(imgcolor.ravel(), bins=256)
plt.xlabel('Grayscale [dark to light]')
plt.ylabel('Number of pixels')
plt.axvline(x=ret, color='r', linestyle='dashed', linewidth=1)
plt.title('Histogram')

plt.subplot(3,1,2), plt.imshow(imgcolor,cmap = 'gray')
plt.title('Grayscale image: ' + imgname_input)

plt.subplot(3,1,3), plt.imshow(imgf,cmap = 'gray')
plt.title('With thresholding: ' + imgname_output)

plt.show()
plt.imsave(imgname_output, imgf)
```

Appendix 3: Computer generated image blur grading – code example

```
#################################################################################################
# It shows an image and calculated blur grade, indicating possible ink corrosion
#
# This code can be run directly in a Jupyter notebook
# Installation prerequisites are: python, pip, ipython jupyter, numpy, matplotlib, imutils
#################################################################################################

%matplotlib notebook

# import the necessary packages
import argparse
import cv2
import numpy as np
from imutils import paths
from matplotlib import pyplot as plt

def variance_of_laplacian(image):
        # compute the Laplacian of the image and then return the focus
        # measure, which is simply the variance of the Laplacian
        return cv2.Laplacian(image, cv2.CV_64F).var()

# Set basic input parameters here for further processing
file_img_input = 'Real_Sample3'      # define image file name

# Define input and output image
imgname_input = file_img_input + '.jpg'
imgcolor = cv2.imread(imgname_input)
imggray = cv2.imread(imgname_input, 0)
imgname_output = file_img_input + '_enhanced.jpg'
```

```
#set image
image = cv2.imread(imgname_input)
#decrease size
scale_percent = 50
width = int(imgcolor.shape[1] * scale_percent / 100)
height = int(imgcolor.shape[0] * scale_percent / 100)
dsize = (width, height)
image_resized = cv2.resize(imgcolor, dsize)
#convert to grayscale
grayimage = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY)

# calculate and show blur
fm = variance_of_laplacian(grayimage)
# show the image
cv2.putText(image_resized, "{}: {:.2f}".format("blur grade: ", fm), (10, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 3)
cv2.imshow("Image: " + file_img_input, image_resized)
key = cv2.waitKey(0)#
```