CS230

Optical Character Recognition on Arabic Medieval Manuscripts With Attention-Based Sequence to Sequence Model

Ammar Alqatari
ammarq@stanford.edu

Nawaf Alnaji
alnajina@stanford.edu

Abstract

OCR focused on historical transcription has been rarely applied on Arabic historical manuscripts. The VML-HD dataset [5] provides a dataset of 600 medieval manuscript pages annotated with sub-word bounding boxes and transcriptions. We build an attention-based sequence to sequence model which can recognize sequences of characters in a given token image from this dataset with relatively high accuracy.

1 Introduction

There is a rich tradition of medieval manuscripts written in Arabic script ranging over more than 1000 years of textual production. Unlike medieval manuscripts in Latin script, this wealth of manuscripts has not yet been completely digitized and very little of it is machine-readable. There is no doubt that digitizing such a collection opens vast opportunities for scholarship in the humanities fields, yet it remains a field which is largely untapped. Part of the problem is the non-uniformity of the scripts; there is a wide range of Arabic scripts, fonts, and page layouts coming from the different periods and regions of text production. However, the main issue is that there simply hasn't been much effort dedicated to the problem.

In this project, we construct a deep learning algorithm to learn optical-character recognition on a subset of medieval Arabic manuscripts with high accuracy. We focus on a specific part of the problem. In particular, given images of subword tokens from a medieval manuscript, we use a CNN and sequence-to-sequence model to output the sequence of characters in that token. We also make some progress on transcribing an entire page by segmenting it into lines, then transcribing each line.

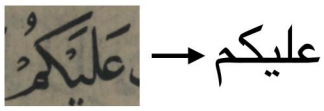

Figure 1: Example of a subword input image and its transcription

2 Related works

Many approaches to OCR (such as [7]) focus on character segmentation and character classification. This approach works quite efficiently on Latin script languages written, but character segmentation is especially difficult on Arabic text, since letters in a single word are all connected, and it's usually

not clear where one letter starts and ends. This is also asserted in [2], which notes that because Arabic script differs greatly from Latin and Chinese scripts, almost all successful approaches fail when applied to Arabic OCR. The survey lists several works which have developed architectures specifically for Arabic handwritten OCR, though most deal with modern handwritten scripts, which greatly differ from medieval handwritten manuscript. Approaches included use of MDLSTMs, deep belief networks, and probabilistic graphical models, the most accurate of which achieved 90% accuracy on a dataset of handwritten Tunisian city names.

There's been a variety of RNN-based OCR implementations, the best performing of which all use CNN feature extraction. Of these models, models with attention-based decoders have been shown to outperform all lexicon-free competitors [6]. This is the approach our work is most similar to, which is more of a pure deep learning approach than the methods listed in [2].

There has also been useful work on historical OCR specifically. The biggest issue with historical OCR is transcribing labels, which must be done by hand. Error estimation is thus difficult when models are used on unlabeled data. Frameworks for error estimation have been proposed [4] that rely mainly on existing modern lexicons, but which are also able to continually learn historical linguistic information from the manuscripts in order to more accurately estimate the error.

3 Dataset and Pre-Processing

We use the VML-HD dataset [5]. It is a dataset, compiled in 2017, which is composed of 5 books handwritten between the 11th and 15th centuries, with a total of 680 pages. Each page is annotated at the subword level, with the transcription of the subword and its location in the page image. Since Arabic is a cursive script, a subword here refers to a connected component of a word. In total, there are approximately 150,000 subwords, and 12,000 lines. Though the different books don't use identical scripts, they all use similar styles.

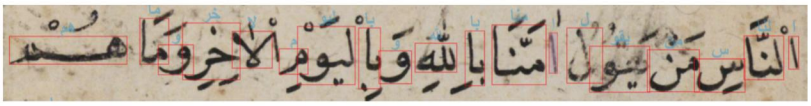

Figure 2: Line annotated with bounding boxes around subwords. Note that the bounding boxes aren't added to the manuscript images; this is only a visual representation of the data

.

Since our project focuses on recognizing subwords sequences of characters, we processed the data to extract each subword from each image and convert it to a grayscale matrix. We then resized each subword image to a fixed height and width (which results in some slightly distorted images). We've also tried resizing each subword image to have a fixed height that allows it to and a variable width which preserves the image's aspect ratio. Both resulting dataset are each composed of 156,383 subwords.

We've also used subword coordinates and transcripts to generate line data. Each line was represented as a larger version of a subword, with coordinates representing the line location in the image, and a transcript composed of concatenating the transcripts of each subword in that line. Due to the nature of this dataset, this was essentially the only way to generate transcripts for entire lines; it is also then quite simple to generate transcripts for entire pages, by simply concatenating the lines, but we did not find any reason to do that within our project scope. These line coordinates are then processed through the aforementioned image processing step to result in a dataset of fixed-height line images and transcripts, of around 6,313 lines.

We shuffle our data and split it into train, dev, and test sets, with ratios of 90%, 5%, and 5%, respectively, for subword datasets, and ratios 80%, 10%, and 10% for the line dataset.

4 Model

Our model is based on Qi Guo and Yuntian Deng's Attention-OCR model [3]. At a high level, the model is composed of a CNN component which flattens the input image (which is of fixed height, but variable width) into a 1-dimensional sequence of features. This encoding is then passed to a sequence

to sequence model (with LSTM units) and attention. We use beam search and a character-level language model we trained on the Quran to obtain the output from the decoder.

4.1 Extracting Image Features

To feed the image into a sequence to sequence model, we must first encode it as a sequence. To do this, we use a convolutional neural network (CNN) to extract features from the image and flatten it to a sequence.

The network is composed of 4 convolutional steps, each composed of one or two convolutional layers, followed by a maxpool layer, then a 5th convolutional layer and a dropout layer. Each maxpool layer reduces height by half. As our original image height was set to 32, 4 such layers result in a 1-dimensional sequence.

4.2 Sequence to Sequence Model

We use a sequence to sequence model based on Google's implemented Seq2Seq model in Tensorflow [1].

4.2.1 Encoder

For the encoder, we use a bi-directional recurrent neural network (RNN) with LSTM units. We clip gradients to prevent vanishing gradients. The inputs are the feature vectors generated by the CNN. The encoder (as well as decoder) also makes use of bucketing, which allows variable-length by creating different buckets of lengths and padding each input sequence to fit into a bucket.

4.2.2 Decoder

Our decoder uses attention, also using Tensorflow's existing implementation, so that different parts of the input sequence can continue to influence later decoded characters. Attention works by computing a context vector which weighs each part of the input by its importance to a particular output time-step. Attention is especially helpful in our model, because it can recombine information, which is spatially adjacent in the original input image, but gets separated when it is flattened into a 1-dimensional sequence.

Another feature we add to the decoder is using beam search and a language model to construct the sequence from the logits outputted by the decoder. We trained a small character-based language model locally using data from the Quran, which has a character distribution similar to the texts in our dataset which come from the early Islamic period and are all either religious or linguistic texts.

4.3 Line Segmentation

We perform line segmentation through statistical image processing rather than deep learning. Our algorithm is a simplified implementation of [2], implemented as follows:

1. Binarize image into black and white using Otsu thresholding, which finds the optimal threshold value to split the pixels of an image into two classes.
2. Find the mean number of white pixels per row in the page
3. Find all rows with number of white pixels above mean
4. Run mean shift algorithm to find clusters of white rows
5. Set line dividers to cluster centers

5 Experiments and Results

5.1 Evaluation Metric

Although we use cross-entropy loss to train the model, our objective is to maximize our output string similarity to the ground-truth transcript. We use Levenshtein distance (LD) to evaluate our model's performance, which is a commonly used metric for measuring string similarity. The Levenshtein

distance between two strings is the minimum number of insertions, deletions, or substitutions that can transform one string into the other. For an example with model outputs o and ground-truth string g with length l, the accuracy is equal to $A = 1 - \frac{LD(o,g)}{l}$.

A is equal to 0 when o is the same length as g but shares no characters with it, and equal to 1 when o exactly matches g. It can also be negative if o is a different length from g without sharing any characters. The accuracy of an overall batch of examples is simply the average accuracy of each example.

5.2 Main Results

Dataset	Train Accuracy	Test Accuracy
Lines	0.51	0.18
Variable-width subwords	0.71	0.59
Fixed-width subwords	0.83	0.77

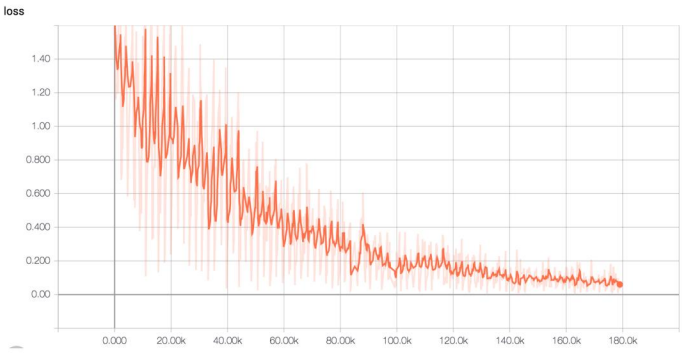

Figure 3: Loss graph of basic model

5.3 Hyper-parameter Tuning

We compare different pairs of parameters by fixing all parameters of the basic model and changing only the specific parameter we are comparing. We train each model for 5 epochs and then evaluate its LD accuracy on the dev test. Our basic model uses LSTM units, embeddings of size 10 of the encoder output, 2 hidden layers for attention, and gradient clipping.

Implementation	Train accuracy	Test accuracy
1. Basic	0.76	0.70
2. Size 20 embedding	0.75	0.73
3. 4 hidden layers	0.66	0.658
4. Without gradient clipping	.72	0.69

6 Discussion

A surprising result from our experiments is that the model performs much better when learning on a dataset where all input token images have been resized to a square of the same size, rather than inputs with a variable width that preserves the image's original aspect ratio. This is surprising because we expected resizing tokens to the same size to distort them too much to be learned easily – especially since their size can range from 5 to 250 pixels wide. We believe this might be the case because resizing all inputs to a uniform size has a normalizing effect.

We also hoped that since our model handles different-sized inputs through bucketing, it would be able to learn to recognize entire lines if we feed them as inputs. However, this approach did not meet our desired results. This is expected, however, since we only have around 7,000 lines to train on compared to more than 100,000 subword tokens. We discuss in the next section possible approaches to apply in the future which leverage the parameters our model learned to recognize subwords in order to transcribe entire lines.

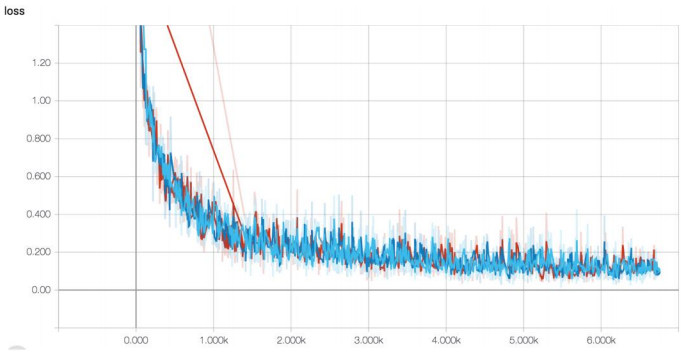

Figure 4: Comparison of loss over 5 epochs with embedding size 20 (dark blue), 4 attention layers (light blue), and no gradient clipping (red)

Our best performing model achieves 83% accuracy on the training set and 77% accuracy on the test set. We do not believe that the model has over-fit the training set because it achieves a similar accuracy on the dev set. The slight discrepancy in performance could be due to slightly different distributions between the two sets, because they might each have different proportions of the five books which compose the dataset.

Finally, we believe that getting better accuracies will require a much greater amount of hand-engineering than what we perform. The data we have is very noisy due to the quality of the manuscripts, the presence of diacritics which are unaccounted for by our transcriptions, many mislabeled tokens, and the inherent difficulty of the script itself which can be somewhat non-linear with overlapping words at times contrary to the model's structure.

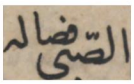

Figure 5: Example of token with completely overlapping words

We've tested the model using Levenshtein-Distance accuracy without discriminating letters that only differ in dots, resulting in the error being reduced by 30%. This indicated that around a third of the error was due to letters that have the same shape.

7 Conclusion and Future Steps

We've deconstructed our transcription problem into several steps in such a way that makes good use of our dataset. The first step is line segmentation, and the second step was intended to be end-to-end line transcription. The transcription model has high performance on both subword datasets, though significantly higher in the fixed-width dataset. The transcription model performed well on subwords, but unfortunately it has not scaled well to lines. There are two potential approaches we believe may solve this issue. The first is to generate more data of variable length, by using sequences composed of parts of lines ranging from one subword to the entire line (for example, a line 'A B C' can be used to generate dataset

$$'A','B','AB','BC','ABC'$$

). The second is by constructing a method to segment lines into subwords, possibly using character segmentation methods.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris

Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Mahmoud Al-Ayyoub, Aya Nuseir, Kholoud Alsmearat, Yaser Jararweh, and B B Gupta. Deep learning for arabic nlp: A survey. 11 2017.

[3] Yuntian Deng, Anssi Kanervisto, and Alexander M. Rush. What you get is what you see: A visual markup decompiler. *CoRR*, abs/1609.04938, 2016.

[4] Florian Fink, Klaus U. Schulz, and Uwe Springmann. Profiling of ocr'ed historical texts revisited. *CoRR*, abs/1701.05377, 2017.

[5] M. Kassis, A. Abdalhaleem, A. Droby, R. Alaasam, and J. El-Sana. Vml-hd: The historical arabic documents dataset for recognition systems. In *2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR)*, pages 11–14, April 2017.

[6] Chen-Yu Lee and Simon Osindero. Recursive recurrent nets with attention modeling for OCR in the wild. *CoRR*, abs/1603.03101, 2016.

[7] H. Zhao, Y. Hu, and J. Zhang. Character recognition via a compact convolutional neural network. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6, Nov 2017.

https://github.com/ammarqat/medieval-manuscript-ocr