Handwriting Recognition 2022-2023.2B (WMAI019-05)

Project Instructions

Starting date: 19 April 2023

Due date (Project/Code): 23:59 hrs, **11 June 2023**
Due date (Group report): 23:59 hrs, **16 June 2023**
Due date (Individual reports): 23:59 hrs, **18 June 2023**

You will be working on three different handwriting recognition tasks for the group project. Tasks 1 and 2 are connected and based on the Dead Sea Scrolls data, and task 3 will use the IAM database for English handwriting. The datasets and task instructions will be described below. You are allowed to divide the work of implementing your solutions between group members, but make sure everyone knows which choices are being made and why. Furthermore, you can use existing code for parts of your codebase, as long as you properly cite where it came from and understand what it does.

Good Luck!

Contents of this instruction manual:

Datasets

Dead Sea Scrolls (DSS) images for Task 1 & 2

This collection of ancient manuscripts has immense historical, religious, and linguistic significance. Most texts of the DSS collection use Hebrew, with some written in Aramaic and a few in Greek. However, we will keep it restricted to only Hebrew characters for this project.

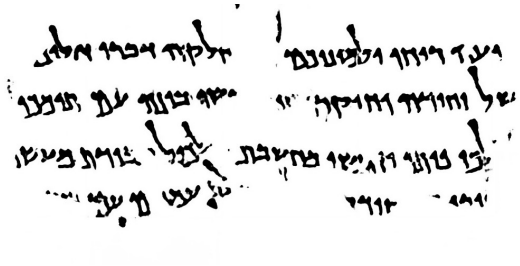

Figure 1: A binarized image of a Dead Sea Scrolls fragment

The character set (labeled data) can be downloaded from here:
https://unishare.nl/index.php/s/AosweMNSy9PGywR

Note that the labeled images are in PGM format (lowest common denominator grayscale file format). In Linux/Unix system, it should not cause any trouble. However, if you are using a Windows system, your OS may be missing the NetPBM package to read the PGM format. Keep this in mind! Just in case, here are the JPEG images of the same format:
https://unishare.nl/index.php/s/BkmkDBmq9SFQrbi

You will find 27 different directories containing character images for one of the letters from the Hebrew alphabet. Please note that the Hebrew alphabet has 22 letters. It does not have an upper or lower case, but five letters (Kaf, Mem, Nun, Pe, Tsadi) have different forms depending on their position within words. For simplicity, we will consider 27 different classes for the project. The name of the directory is the label for each class.

Here is a list of those 27 characters, with their average images. This will help you to get an idea on how they look like:
https://unishare.nl/index.php/s/ZmSYKpPzFbEP6Ys

Example test images can be downloaded from here:
https://unishare.nl/index.php/s/AKsCbZ4WqtGgi25

In this directory, you will get 60 images (20 RGB, 20 gray-scaled, and 20 binarized). Each RGB, gray, and binarized image set corresponds to the same physical fragment. The gray images are fused from multiple spectral-band images of each fragment. As a result, they have better visual quality in ink background separation than the colored ones. The binarized images are automatically generated using a deep encoder-decoder network (BiNet). For final testing, we will use images similar to binarized images. The colored and gray images are there for your reference only, and they will give you an idea of what the original fragments look like (and if you want to improve the binarization process). The images are already rotated to have the correct orientation for reading.

(If you have a system that works directly on the coloured images then that will be an add-on, but not required. Please discuss and mention this with us before approaching with the coloured images.)

To have an overall idea of how the Dead Sea Scrolls data look like, visit this site:
https://www.deadseascrolls.org.il/

For those of you who want to pre-train your model, we recommend using the Habbakuk font (recommended, but not a must!). A copy can be downloaded from here:
https://unishare.nl/index.php/s/PiLApMsSgtzA4pt

We have also attached a small Python script that provides you with an example of how to create images, similar to your training data, using the font. The script can be downloaded from here:
https://unishare.nl/index.php/s/wFDpktzbcj5fibQ

IAM dataset for Task 3

The IAM database [1] is publicly available for writer-independent handwritten text recognition and writer-identification tasks. It was created by asking 657 different writers to copy a small English text (about five lines) in their natural handwriting. The handwritten texts have been scanned and segmented at the line and word level. If you want to know more about the reasoning behind the creation of this database, there is a paper describing it in more detail [2], which you can also cite in your report.

You will not be using the entire database but only the *text lines* and their transcriptions, without using information about which line was written by which writer. This writer's information is not essential for the task you will be working on. You can download the data as an archive containing the line strip images and the ground-truth text. You do not need to register for access on the IAM website since we will provide the data directly.
You can download the IAM data from here (338 MB):
https://unishare.nl/index.php/s/kC5SzzjqjAj6eKT

The IAM archive we provide to you contains two files: an `img` folder, which contains all the line images, and `iam_lines_gt.txt`, which contains the ground truth transcriptions for the line images. The text file contains the file names of the images and their transcriptions, separated by two newlines:

```
line1.png
Text for line 1

line2.png
Text for line 2

...
```

Note: We will not provide you with all the data from the IAM database, since we reserve a subset for testing your final submission. Nevertheless, the IAM database is public, and anyone can access it in principle. However, you are expected to use *only* the data that we provide to you for training your system. If we suspect that the IAM test data is used to train your model, it will be considered cheating, and the prevalent academic consequences will follow. So, please, refrain from improving your model's accuracy by using the test data.

[1] `https://fki.tic.heia-fr.ch/databases/iam-handwriting-database`
[2] The IAM-database: an English sentence database for offline handwriting recognition

Note: Both of the provided datasets' folders, files, and examples have been prepared and structured so that you may solely focus on solving the tasks. However, you should also never assume that the files and datasets you will work on in real-life will be perfectly reliable. For this reason, we highly recommend familiarizing yourself with the provided files and data. We do not require you to include sections in the report or to do a full-blown statistical Exploratory Data Analysis. Instead, it will support your understanding of the tasks and be aware early enough if there is anything you would naturally not assume, e.g., file formats not supported by your system, corrupted or duplicated images, etc.

Tasks

1. Preprocessing and character segmentation (DSS dataset):

   In order to recognize each character, the first step is to segment them from the test images. The output of the segmentation should be similar to the ones in the training set. The performance can be measured by the overlap (Intersection over Union IoU) between your box and the ground truth. (one example can be found in page-8 of this paper). In this project, we will not make any evaluation of the bounding box, this is just for your own knowledge of how things work.

   Keywords for literature: line segmentation, center of gravity (for character segmentation), horizontal/vertical bound, etc.

2. Character recognition (DSS dataset):
   *(The reading direction for Hebrew is from **right to left.**)*
   Your recognizer should recognize all the characters segmented from the first step. You can use the printed characters (font/s) to pre-train your model and then fine-tune using the provided training data set. You can try different models. Note that the total number of characters in actual test images is unknown (for now!) but will be similar to the sample images.
   Some data augment methods might be used if you do not have enough training samples. You can use the image-morph tool from Lambert:
   https://github.com/GrHound/imagemorph.c
   Keywords: Character morphing, Elastic morphing.

   Using a sliding window strategy, you can train a character recognizer (step 2) and apply it to detect the character box (step 1). Your main task is designing a character recognizer, irrespective of how you want to do that. In case you want to integrate the linguistic context, we are providing here the n-grams (bi-gram, tri-gram, etc.):
   https://unishare.nl/index.php/s/Hy2ZYbnwNYKQde8

   (The easiest way to explain these bi-grams or tri-grams is as follows:
   If there are three instances of the word CAT, five instances of CAPE, and two instances of CANADA in a document, the count will be as follows:
   CAT = 3
   CAP = 5
   CA = 2
   You can see here that 'when it is possible' to count the tri-grams or more, it has been done following 'CA-', else it was taken as bi-grams of 'CA'. It is obvious that you can find CA a total of ten times; that is true. However, this is not how the n-grams were calculated on the provided list.
   For instance: in the case of Kaf_Waw, you can find the combination in both Kaf_ Waw and Kaf_ Waw_Lamed (so the total number would be the sum of both the counts).)

   Anticipatory buzzwords: Binarization, OTSU, HMM, CNN, LSTM.
   Packages: Tensorflow, Theano, Keras, Caffe, etc.

3. Line recognition (IAM dataset):

   You will perform text recognition online images from the IAM dataset. For example, given an input image of a handwritten line of text, the task is to transcribe the text in the image. As an example, see Fig. 2. The expected output would be "Susan Hayward plays the wife sharply".

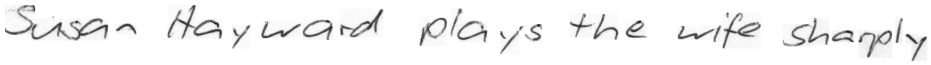

Figure 2: A line image from the IAM dataset.

Rather than segmenting words or characters as a preprocessing step, you may want to consider an end-to-end deep learning solution without initial segmentation. Modern deep learning architectures for handwriting recognition can deal with varying image sizes, e.g., using an *attention* mechanism. It is your job to find the most suitable solution for the task.

Data Augmentation:
(*This is not a separate task, but rather useful for both tasks 2 and 3.*)

Since deep learning methods thrive on large volumes of data, you may again want to consider using data augmentation methods to increase the number of training samples. There are a lot of off-the-shelf augmentation techniques based on various color and geometric transforms. You can use also elastic morphing-based ones as references in task 2 above, which is more targeted for handwritten characters. Finally, you can try the GAN-based techniques for augmentation. Whatever the case, you should use at least one augmentation technique in your tasks and explain the implications in your report.

Note on data augmentation based on generative adversarial networks (GANs):
Generative Adversarial Networks (GANs) have been successfully used for data augmentation in various computer-vision tasks. GANs consist of two neural networks, a generator, and a discriminator, that are trained together in a process where the generator learns to yield realistic pictures, and the discriminator learns to distinguish between real and fake samples.

Please keep in mind that training a good GAN model is a challenging task that often requires significant time and computational resources. For example, it will take about seven days to train a GAN model on the data set with size 80,000 on a single V100 GPU. To build a genuinely exceptional GAN model, one must be willing to invest the time and effort required to achieve success.

Keywords: encoder-decoder, CNN, RNN, Transformer, CTC loss, attention, Word Error Rate, Character Error Rate, etc.

Presentations

Students are expected to give an oral (+powerpoint) presentation as a progress update during each lecture session, starting from the second lecture. We will randomly assign each group an identifying number (1,2,3,..). Groups with an odd number will present in the second lecture, with an even number in the third lecture, and so on. We will put a schedule once all the groups are formed. The progress update should at least have the following contents:

1. Overview of articles you have read (including a total number read so far);
2. Amount of text written for literature review & methodology section of the paper;
3. Programming progress (how many modules will the system have, how far is each module completed?);
4. Empirical evaluation progress (both technical and theoretical);
5. Is the overall progress on schedule?

Each component needs to be properly documented and supported by either references or tables and graphs. Show that you read the articles (i.e., show what the article was about and the conclusions), and keep a list; you need it for the References section of your paper.

Each group member should have had the opportunity to show their presentational skills: divide all tasks, including the presentations, between the group members equally. For example, appoint a person to maintain the progress and update PPT slides, a person overlooking overall system architecture, a person to design the empirical evaluation (test scripts), etc. Each group will have 5-6 minutes for presentation and 2 minutes for Q&A. The presentations will be graded and count for 10% of the final course grade.

Evaluation of recognition systems

At the end of the course, you are expected to have developed two handwriting recognition systems, one for Hebrew characters on the Dead Sea Scrolls data and one for line-level text recognition on the IAM data. You will hand in both systems in the form of a zip archive (or a separate archive for each task if you prefer) containing the code and saved models if applicable. We will run each method on a test dataset different from the ones provided to you in order to measure how they perform on new data. Note that the performance is only a part of the grade, and *it is possible to get a good grade even if the final performance is not as good as you would like.* You must understand how your method works and why it behaves the way it does.

README and dependencies

Make sure to include a `README.md` file in your submission with clear instructions on how to set up dependencies and compile and run your code. Ideally, you should provide a list of commands that we can run in a Linux environment to install all the dependencies. Your list of dependencies should be appropriately updated once you finish the project. If necessary, test your code on the system of one of your group members to make sure it works on systems besides your own. If you use Python, working with a virtual environment for managing packages is highly recommended. If you want, you can also use a Docker container to manage your dependencies. In that case, please describe clearly how we can start your container and give it access to the input and output directories in the README.

After your code is set up, we should be able to run each method by executing a command and providing paths to the input and output directories as command-line arguments. Again, make sure you describe how to do this in your README and avoid using hard-coded paths.

Evaluation of task 1 & 2 (DSS data)

To test your first recognizer on the Dead Sea Scrolls data, we will (compile and) run your code on a separate, secret test set. The output should be given by creating a separate text file containing the recognized text for each input image. During the third week's tutorial session, we will provide a couple of sample outputs from the test data. To measure the performance, we will use a simplified version of Levenshtein distance.

Evaluation of task 3 (IAM data)

We will run your code on a separate test set to test your second recognizer on the IAM data. The input to your program should be a path to the directory that contains the line images. Your code should write the recognized text for each line to a txt file with the same structure as we used to provide the ground-truth data (two lines per prediction: one for the image name, one for the text prediction, where two lines separate predictions). To estimate the performance, character-level and word-level accuracy will be measured.

Report

The report needs to be a scientific paper about the handwriting recognition system(s) that you built during the course. It should be around 4000-5000 words long, written in English (roughly 8-10 pages).

The report should be structured like a scientific article and consist of the following sections:

- Title + Group number and members' names & student numbers
- Abstract
- Introduction and Literature review
- Methods
- Results and detailed descriptions
- Discussion and conclusions (*submit individually*)
- References
- Appendix, if any
- Note on **individual contribution** to the project (*must*)

The report should be submitted **as a group** except for the 'Discussions and conclusions section. Every member should contribute extensively to the group report. Coordinate the distribution of focus points among your team and describe which group members contribute to which topic.

The 'Discussions and conclusions' should be written and handed in **individually** (please mention your name, student number, and group number at the top). It should be a separate (PDF) document of around 800-1200 words (around 1-2 pages) where individual students reflect and discuss the project work they did as a group. You can use the discussion to demonstrate that you understand the project and reflect on the methods you used and the results you obtained. You can also compare the methods you used for both datasets and their preconditions, advantages, and disadvantages. Your discussion can contain a small bibliography if necessary.

Grading

Assessment Criteria

You will be finally graded on your **participation in the group**, on your **presentations**, **empirical evaluation and programming**, and on your group & individual **reports**. The code and the group report (without a discussion section) should be handed in via Brightspace by one member of the group. In addition, each group member individually submits a separate document via Brightspace containing their version of the discussion.
The final grade appears on Progress. There is **no exam** other than the final recognizer and written report. The final course grade is determined as follows:

- **Presentation (10%)**
- **Recognition system (45%)**
  - Code quality
  - Documentation
  - Performance on the test set
  - Ease of set-up
- **Report (45% = 100 points)**
  - Title + Abstract: **5**
  - Introduction + literature review: **20**
  - Methods:
    - Task 1 & 2 (DSS): **15**
    - Task 3 (IAM): **15**
    - Use/choice of augmentation (can be included in tasks 1-3): **10**
  - Results representation:
    - Task 2 (DSS): **5**
    - Task 3 (IAM): **5**
  - Discussion and conclusions: **25 (submit individually)**

Contacts

You can use the **discussion section** on Brightspace to ask general questions about the project and report, and you can ask questions about the project during the tutorial sessions.

You can also send questions to the *course helpdesk* (that are not covered in the discussion board or in tutorial sessions) about the assignment outside of the scheduled lab hours. For our automatic filtering, you must include "Group-N" in the e-mail subject, where N is the number of your project group.

For example: "[Group-9] Question about segmentation task".

Course helpdesk:
handwriting.recognition.ai@rug.nl

Instructors:
Lambert Schomaker, Professor (l.r.b.schomaker[at]rug.nl)
Maruf A. Dhali, Lecturer (m.a.dhali[at]rug.nl)

Moderator:
Zhenxing Zhang, post-doc (z.zhang[at]rug.nl)

Student Assistants:
Andrei Krauss
Shi Qiu
Abhishek Satyanarayana

*The use of the Dead Sea Scrolls images used is **restricted**. Please avoid any kind of unwanted distribution of both the labels and the images. Always contact the course instructors before producing any results in the public domain.*