Handwriting Teaching Program

Jorden Jolley

Columbia University, Visual Interfaces Final Project.

Abstract

Paper motivates the need for learning good penmanship and handwriting and describes the implementation of a Python program for teaching a user good handwriting by evaluating and providing feedback on images of their handwriting, following prompts.

Keywords: Learning Development, OpenCV, Python, Tesseract, Handwriting

1 Introduction

1.1 The Phenomena

Although we spend so much time on keyboards, the importance of learning handwriting goes beyond penmanship. Learning to write letters helps children with their letter recognition skills and with quicker reading. In addition, there is evidence that learning to write by hand helps students retain information by activating neural pathways associated with strong reading skills [1]. Additionally, handwriting can also help with memory retention and reading comprehension [1].

When learning new words, one research study showed that participants who handwrite the words performed better on the recognition test compared to those who typed the words. The N400 component of the EEG (an index of semantic processing in the brain) was larger and more sustained for handwritten words. This suggests that handwritten words are processed more deeply and are easier to retrieve from memory [2].

We have established the usefulness of handwriting in a modern world. But does the quality of penmanship matter? Research studies such as "The Effects of Handwriting Practice on Functional Brain Development in Pre-Literate Children" by James and Engelhardt [3] have shown that teaching children to write neatly and legibly does have positive effects on their brain development. For adults, the Journal of Learning

disabilities found that improving handwriting skills could also lead to improved reading skills [4].

1.2 Learning Handwriting and Program Approach

How might one learn handwriting effectively? While incorporating multisensory activities (tracing the letter with a finger, writing in the air, or speaking the letter out loud) are suggested, a key part of learning letters is repetition and muscle memory, in addition to receiving feedback [5]. Useful feedback includes suggestions on size, stroke order, spacing, and alignment with other letters.

Finally, the best order to teach letters in handwriting is grouped by their similarity [6].

In the modern age, computers can automate the feedback process for users seeking to improve their handwriting skills. This paper describes a program which gives writing instructions ("Write the letter A five times on a piece of paper"), analyzes the user's image, and returns feedback based on research-based evaluation metrics such as size, slant, spacing, identifiability, and thickness of the input letters. The user is instructed to repeat the process, receiving feedback on their best letters and letter confidence scores, until they reach a score above the set threshold of their choice.

The program is an example of visual interfaces by way of analyzing the user-generated images and comparing them to set images and extracting textual elements from the images in addition to generating results and feedback based on the images' similarities. In addition, the program will need to identify each letter from a page and compare sizing and spacing between the letters on the page.

Generally, the approach this program took is to use real-life strategy for teaching handwriting (including suggested learning order and evaluation types) and automate the feedback.

2 Limits and Assumptions

2.1 Font and Case

Discussions surrounding cursive often come hand-in-hand with penmanship and the importance of handwriting. However, this initial iteration of the program does not focus on non-cursive language in a simple font similar to those of handwriting books for children in upper and lowercase. The letters will be in English. The entire English alphabet, uppercase and lowercase, is included. Generally, font size and spacing is calculated relative to the other letters; the program does not provide feedback on if the letters are reasonable in comparison to the size of the page, for example, since how large or small one might want their letters to be is relative to the context, but having good size and spacing is ubiquitous.

2.2 Environmental Setup and GUI

The images are not in real-time, but rather uploaded to the program aligning with prompting from the user. This is in part due to the complications for users to record

well whilst handwriting, versus just taking a photo after. The user will interact with the program via a terminal interface for this iteration of the project.

In addition, the user environment should be a controlled domain: a jpg photo of their drawn letters in dark ink on an unlined white paper. This is in order to control the quality of the input for the best possible program recommendations. It should be relatively easy for most users to produce the required materials and domain. Poor sizing and letter distance will be included in program feedback, so major issues in quality of writing should be addressed quickly. HEIC iphone photos must be converted to jpg.

For this iteration of the project, this process can be time consuming, so a real product launch would need a better user GUI. For example, an app that allows users to take photos in-app and receive feedback immediately (converting to jpg on their behalf) would be ideal. However, app development is not in scope for this course, so I focused on building the feedback algorithm.

The product performs best on a blank sheet of paper with a sharpie marker. Future development should include lined paper.

2.3 Major Data Structures

`practice_letters`	a List of all the letters to practice with, in the following order according to suggested research: L, F, E, H, T, I, D, B, P, U, J, C, O, G, Q, S, R, A, K, M, N, V, W, X, Y, Z, l, t, i, c, k, o, p, s, v, u, w, x, z, h, n, m, r, b, a, d, g, q, e, f, j, k, y
`img`	The NumPy version of the current image uploaded by the user that the program is providing feedback on.
`sorted_bounding_rectangles`	Dictionary of sorted bounding rectangle data for each of the 5 letters. Sorted by letter order, left to right.
`written_letter_information`	Dictionary containing information for each letter, such as their bounding rectangle data, area, and number.
`spacing_feedback_scores`	List of spacing feedback scores for each letter.
`size_feedback_scores`	List of size feedback scores for each letter.
`skew_feedback_scores`	List of skew feedback scores for each letter.
`readability_feedback_scores`	List of identifiability feedback scores for each letter.
`thickness_feedback_scores`	List of line thickness feedback scores for each letter.
`total_feedback_scores`	List of sum of feedback scores for each letter.

2.4 Major Algorithm Overviews

Overall, the program prompts user for an letter acceptance threshold. The user enters a threshold number. The program will prompt the user to write the letter on paper. After uploading the image as a jpeg, the program will analyze the letters and provide

feedback for each one, with each type of feedback (size, spacing, thickness, identifiability, and slant) generating a point each. The user is provided highlighted images demonstrating feedback areas as well, with red/orange/green highlighting according to the score for the letter.

The algorithm will continue on the same letter until the user meets their threshold. The algorithm will move to the next letter until all letters meet the threshold, or the user chooses to stop.

Algorithm 1 Base Image Analysis Algorithm Overview

image uploaded by user as jpg
process with cv into numpy array
binary threshold image
pull contours
filter out smallest contours
generate bounding rectangles around 5 letters
create dictionary with sorted letters and some metadata

Algorithm 2 Size Feedback Algorithm Overview

Require: image with letters identified in bounding boxes
while letters to identify > 0 **do**
Calculate area of the letter
Store area of the letter
end while
Calculate median area
Identify outliers $> 1/3$ median
Return feedback

Algorithm 3 Spacing Feedback Algorithm Overview

Require: image with letters identified in bounding boxes
while letters to identify > 0 **do**
Calculate distance to next letter
Store distance of the letters
end while
Calculate median spacing
Identify outliers > 50 pixels apart
Return feedback

Algorithm 4 Slant Feedback Algorithm Overview

Require: image with letters identified in bounding boxes
  while letters to identify > 0 **do**
    Reduce image size by scaling
    Gaussian blur to smooth image
    Binary threshold
    Invert image colors
    Calculates coordinates of non-zero pixels in image
    Calculate area of minimum rectangle WRT horizontal axis with minAreaRect()
    Determine if slant in range for skew angle acceptance
  end while
  Return feedback

Algorithm 5 Identification Algorithm Overview

Require: image with letters identified in bounding boxes
  while letters to identify > 0 **do**
    Cut out each letter's individual image bounding box
    Increase letter thickness with erosion after making image colors opposite
    Tesseract evaluation of image letter: check for match with expected letter
  end while
  Return feedback

Algorithm 6 Line Thickness Algorithm Overview

Require: image with letters identified in bounding boxes
  while letters to identify > 0 **do**
    Cut out each letter's individual image bounding box
    bitwise not the image NumPy
    Create image kernel
    Erode the image using kernel and cv.erode
    Calculate unique values and counts for 0s and 1s
    Generate 0s and 1s percentage
    Check percentage within threshold (2 - 30 percent)
  end while
  Return feedback

3 Testing and Development Process

3.1 Personal Imagery Testing and User Testing

To develop and evaluate the system, I conducted training runs on various letters and tested multiple types of feedback, including size, spacing, slant, identifiability, and line thickness. To evaluate the system's performance, I created "bad" examples of each feedback type that would challenge the algorithm, such as letters that were too large, disconnected, slanted, or not thick enough. This iterative process was crucial

in determining the appropriate values for magic numbers, such as how much distance between pixels is considered too far apart or how thick a line should be. While the algorithm may be somewhat overfit to my personal handwriting, user testing helped adjust the numbers to be more generalizable. For instance, adjustments were made to the slant and thickness algorithms after user testing to address cases where acceptable letters were incorrectly marked as unacceptable due to overfitting.

3.2 Size and Spacing Feedback Algorithm Development

For size and spacing feedback, I kept relatively close to the original idea for the algorithm: simply identifying outliers in size and spacing.

- I adjusted the algorithms to each use median instead of mean, since mean was causing too many letters to be identified as incorrect due to the "bad" letters skewing the average.
- Regarding size feedback, the algorithm utilizes a range equivalent to 1/3 of the median. The "magic" number was determined through trial and error, in order to optimize performance.
- In the case of spacing feedback, the algorithm employs a range of 50 pixels around the median, ensuring the algorithm checks that the letters are neither too close nor too far away.

In user testing, the algorithm was found to be effective. The highlighted results were in line with both the users' and the my expectations for optimal size and spacing of all letters.

3.3 Slant Feedback Algorithm Development

Originally, the idea was to use symmetry. However, this would require generating a magic threshold for symmetry for every letter, capitalized and lowercase. This seemed too ripe for overfitting. In addition, it would be hard to scale and difficult to generate considering font variations.

- New idea: While researching all types of handwriting feedback, I found letter slant is a large handwriting issue and decided to determine slant specifically.
- The skew of an image is actually, at heart, the lack of symmetry - thus it is a related idea. I considered many different calculations for generating skew, such as rotating the image until I could identify the letter using the identification algorithm, and calculating how much I had to rotate it.
- In the end, I found a CV function minAreaRect, which can calculate the angle of an area with respect to a horizontal axis. Therefore, I can use the minimum bounding rectangle idea in combination with this function to find the skew.
- Through trial and error, I could now find the bounds for acceptable skew angle.

Skew is the angle at which the text in an image is tilted with respect to the horizontal axis. The algorithm generates this skew angle by cutting out the image using the bounding rectangle to isolate the letter, resizing the input to a smaller size, applying gaussian blurs, and binary thresholding the image.

After these preparation steps are complete, it inverts the image and calculates the coordiantes of the non-zero pixels in the image. Picture a kind of cutout of the bounding rectangle. This shape is sent to minAreaRect, which calculates the angle of the minimum area rectangle with respect to the horizontal axis.

From there, I analyze the angle to determine if it is slanting left or right and the scale of the slant. Through trial and error, I determined the threshold for slant should be within a range of 15 degrees of 0, -90, or 90. However, in user testing, this does appear to be relatively strict.

3.4 Identification Feedback Algorithm Development

One of the first algorithm ideas was to find the letter "similarity" to the ideal letter. I found that this is the meta idea combining all this feedback, but also wanted to have some way of seeing if the program, at its core, could identify the letter the person wrote, since the other feedback could be applied to any range of letters.

- In short, the goal was to determine: did the user really write the requested letter?
- While researching different ways to achieve this goal, I considered training an image recognition algoritmn. However, there are already plentiful resources for recognizing text in images.
- Therefore, I decided to use Tesseract, an optical character recognition engine. Tesseract's image to string function will pull out text from an image.
- Now, the challenge was actually getting Tesseract to correctly identify true positives and not identify true negatives.

To prepare to use Tesseract, I extracted just the individual letter image. This is in order to pull feedback for a single letter at a time. Once complete, I had an extremely difficult time actually getting Tesseract to return valuable feedback: it seemed to find letters in every bump of the page, and would return long strings of nonsense. There are a variety of configurations you can set - I landed on "–psm 10 –oem 3", which translates to treating the image as a single character. However, the image still did not return valuable information until I applied blurring, cleaning, and most importantly: resized the input image to be smaller. Thus, I believe the large size of the input files and sharpness was causing readability issues.

However, the identification is still the most finicky part of the process: through my own testing and user testing, I found that it seems to get the right readings for certain letters - especially uppercase, curved, unqiue letters - and have a harder time with lowercase and straight edge (L or l) letters.

3.5 Line Thickness Feedback Algorithm Development

Line thickness was also not in my original proposal, however, I found through research that thickness is an essential part of good handwriting. Much too thick or think handwriting can cause readability issues.

- The first challenge: How can the algorithm determine the thickness of a line?

- When brainstorming how to actually identify line thickness issues, I considered algorithms to calculate line thickness by using lines cutting across the letters (this an an adaptation of "barcode reader" algorithms.
- Other ideas, such as counting the number of black pixels and the average line width, did not make sense for this context: Letters are not made up of separate lines, but rather lines all joined together, which are often curved and not even straight.
- I finally found the idea of erosion as it relates to thickness. The intuition is this: you can continually erode an image and evaluate how much of the letter is left after erosion.
- The new challenge was simply to determine how much to erode each letter, and what the expected "amount" left of the letter should be.

Erosion is good for evaluating thickness: if you set a determined number of erosions, you can state how much of the letter you "expect" to still exist after eroding. Therefore while implementing, I thresholded the image into only two values, performed erosion (trial and error for the number of times to erode) and checked the relative percentages of the values - how much "black" is expected to be left? Through testing, I determined between 2 - 30 percent.

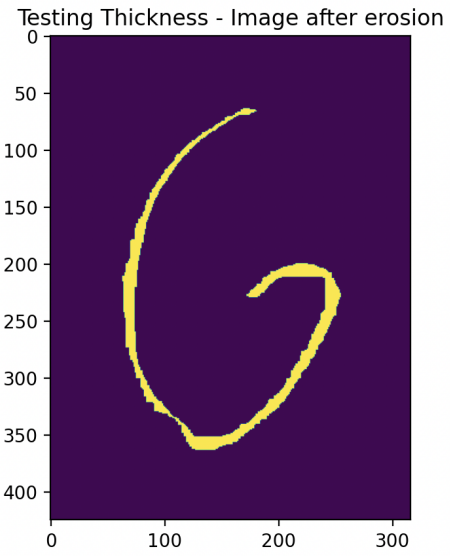

Fig. 1 Example of letter image post-erosion

3.6 Input

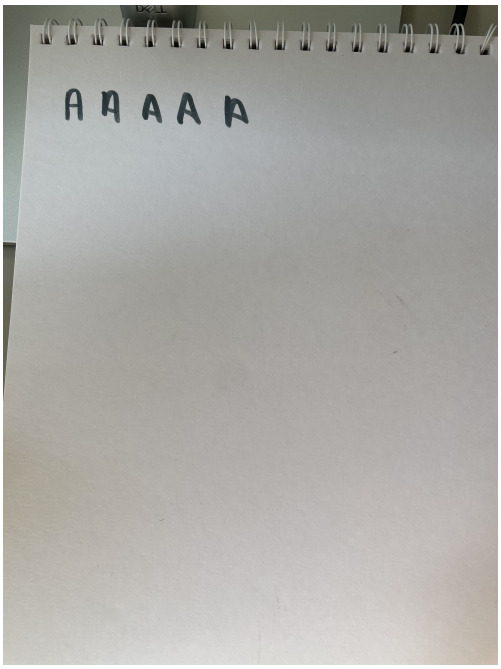

Fig. 2 Example user input image - not processed yet, full size

```
----------F----------

Please upload an image of the letter 'F'
written five times in a row, like this:

F F F F F

Title the image 'F_1.jpg'
Press enter once the image has been uploaded.

Thanks for uploading your image!
Generating feedback.....



 --- Size and Spacing Feedback ---
Letter 1 is too large compared to your other letters. Try writing it smaller.
Letter 5 is too small compared to your other letters. Try writing it larger.
Letter  1  and letter  2  have incorrect relative spacing
Score for size and spacing feedback section:  70.0 %


 --- Slant Feedback ---
Score for slant feedback section:  100.0 %
```

Fig. 3 Example program interface with some feedback

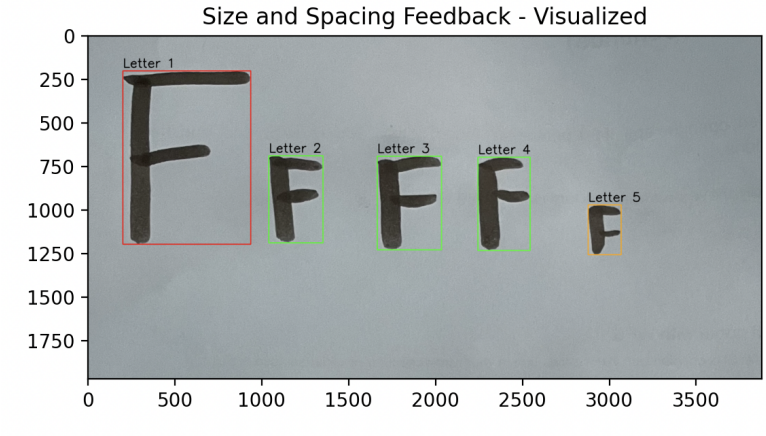

Fig. 4 Size and spacing visual feedback - note the sizes.

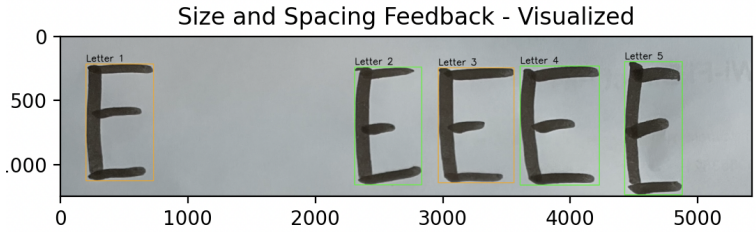

Fig. 5 Size and spacing visual feedback - one is too far, and one is too close.

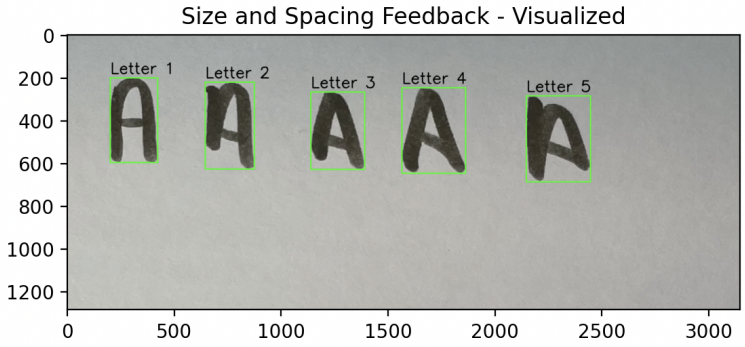

Fig. 6 Size and spacing visual feedback - all good.

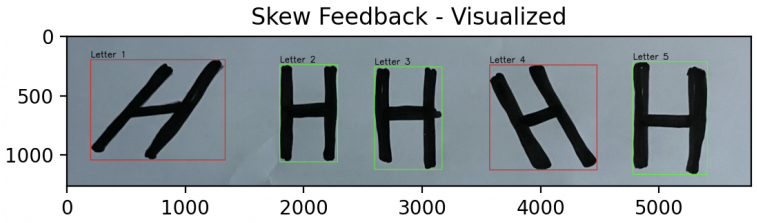

Fig. 7 Skew feedback showing some skew that needs improvement

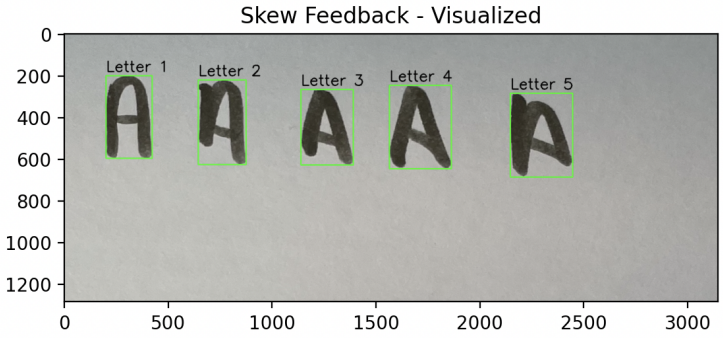

Fig. 8 Skew feedback on 5 "good" letters

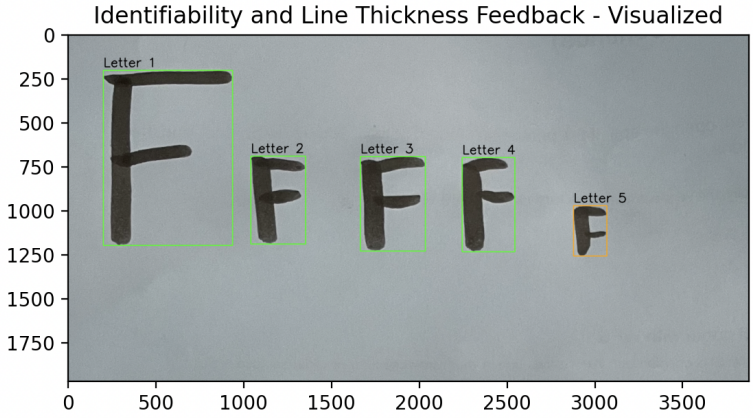

Fig. 9 Trouble with thickness of the last letter.

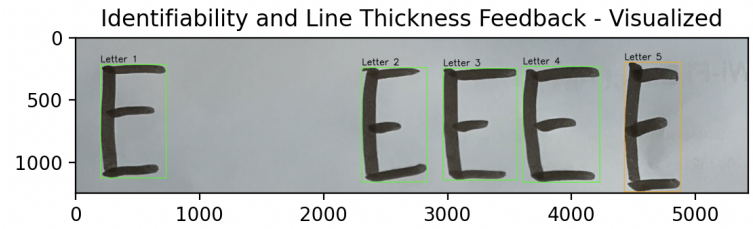

Fig. 10 Trouble identifying the last letter.

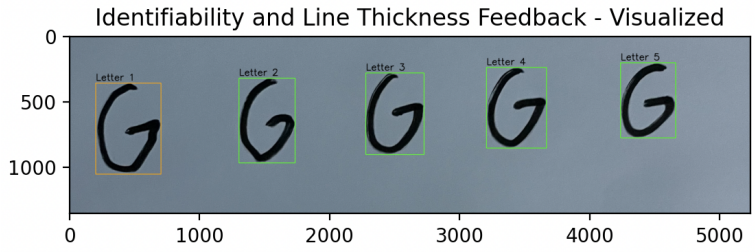

Fig. 11 Mostly good feedback.

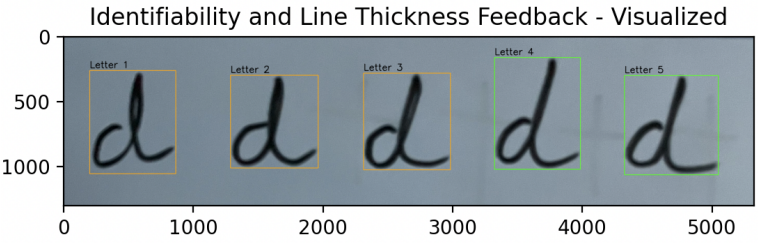

Fig. 12 The last two are the "best" examples of d.

4 Evaluation and Reflection

While testing, improvement scores went up around 20 percent after the initial round of feedback. The value of the program in many ways lies in making people stop to consider the different aspects of their handwriting and see constructive and positive feedback - both visually and written out in the terminal.

Generally, each user has a different difficulty threshold, but I found that 50 percent was reasonable to achieve for most users and letters. To test improvement, it is important to use a lower threshold in order to make the program challenging enough for the user.

However, there are some false negatives - especially relating to the identification algorithm, that make a threshold score too low risky, since the score has some variability in reasonableness due to the identification algorithm sometimes being incorrect.

Next time, I would spend more time testing and adjusting the threshold values for each algorithm. In addition, I would continue adjusting the identifiability algorithm, which is the weakest feedback section of the program. I would also make the program better suited to a variety of domain engineering: for example lined paper and light pencil. The written feedback could be adjusted to be more specific. Finally, a better user interface would be essential for any real applications of the program - especially one that allows users to upload photos in-application instead of having to manually take the photo, upload to the computer, and convert to jpg.

Through developing this program, I gained insights how to translate the theory of teaching good handwriting into practical computer applications. I learned different ideas for determining line thickness, including erosion and barcode algorithms. I learned about skewness and skew angles, and how the angle can be determined using rectangles and horizontal lines. Moreover, I learned about on the distinctions between good and poor handwriting and aimed to apply fundamental learning principles and instructional guidelines to deliver constructive feedback to users.

References

[1] James, V. K. Berninger: Brain research shows why handwriting should be taught in the computer age. LDA Bulletin **512**(1), 25–30 (2019)

[2] A. S. Ihara, A.K.K.I.K.O. Kae Nakajima, Naruse, Y.: Advantage of handwriting over typing on learning words: Evidence from an n400 event-related potential index. Frontiers in Human Neuroscience **15: 679191** (2021)

[3] Karin H. James, L.E.: The effects of handwriting experience on functional brain development in pre-literate children. Trends Neuroscience Education (2014)

[4] Montgomery, D.: The Contribution of Handwriting and Spelling Remediation to Overcoming Dyslexia. Middlesex University, London and Learning Difficulties Research Project, Essex, UK (2012)

[5] Syahputri, D.: The effect of multisensory teaching method on the students' reading achievement. Budapest International Research and Critics in Linguistics and Education (BirLE) Journal (2019)

[6] Engel, L.K.Z.S. C.: Handwriting in early childhood education: Current research and future implication. Journal of Early Childhood Literacy (2015)