English Handwriting Recognition and Segmentation Utilizing Deep Learning in Keras

Michael F. Smith III
Electrical Engineering Undergrad,
Rochester Institute of Technology
Rochester, NY, USA
mfs1036@rit.edu

***Abstract*—The use of Neural Networks and Deep Learning has become an integral component of handwriting character recognition (HCR) and the larger field of optical character recognition (OCR). The ability to convert a handwritten document to a machine-encoded text document involves the use of not only HCR, but also word segmentation. Word segmentation is the ability for a network to identify the words found within an image. The focus of this paper details the process of utilizing Deep Learning to implement both recognition and segmentation which will allow for the parsing of words and sentences from the read characters of a handwritten text document.**

***Index Terms*—Handwriting Character Recognition, Word Segmentation, Deep Learning, Convolutional Neural Networks, EMNIST Dataset**

I. Introduction

Handwritten word and character recognition usage spans various fields and continues to grow. This recognition and conversion of handwritten or printed text into a machine-encoded version of text is known as Optical Character Recognition (OCR). OCR has applications in automation, banking, handwritten language translations, and many more fields. OCR has two types, online and offline handwriting recognition [3]. Online recognition is recognition that takes place while a writer is making the text (while the user is writing) [3]. Offline recognition, on the other hand, is recognition performed on an image of a document. Many researchers in the field of Artificial Intelligence have implemented a variety of ways to recognize characters both online and offline utilizing large datasets such as MNIST or EMNIST. The dataset MNIST stands for Modified National Institute of Standards & Technology. Utilizing these datasets as well as the power of Convolutional Neural Networks (CNN's), the identification of individual digits, uppercase & lowercase English letters, and various other scripts and languages is quite possible. Additionally, many other researchers have done extensive work in the field of word segmentation. Word segmentation involves the identification of individual words within a handwritten text file as well as the lines between those words (line segmentation) [4].

This paper proposes a method of "Recognition-With-Segmentation" handwriting recognition which will be implemented in Keras [1]. Keras is a Python library which will be used to implement classification of the EMNIST dataset [2]. In addition to Keras, OpenCV will be utilized for preprocessing and Tensorflow will be utilized to access the EMNIST dataset. The EMNIST dataset, which stands for "Extended MNIST," since it extends MNIST to include handwritten letters, is a "set of handwritten character digits derived from the NIST Special Database 19" which were "converted to a 28x28 pixel image format and dataset structure [which] directly matches the MNIST dataset" [7] and [11].

The goal of this method is to input a handwritten text document, preprocess it using OpenCV, have the program break it down into individual lines, and then work through each line, splitting off a word, identifying / recognizing the characters (that make up this word) using a CNN and then moving on to the next word and repeating this until the entire document has been recognized. Section II. of this paper will focus on the comparison of this method to research that has been done previously. Section III. will discuss the means of implementing the proposed method. Preprocessing, line and word segmentation, and CNN implementation and architecture will be detailed. Section IV. details how the method will be validated and deemed successful.

II. Literature Survey

Much research has been conducted within the field of Deep Learning and handwriting recognition. Many papers exist that implement handwriting recognition in a variety of ways along with a variety of datasets. Literature article 2 details the process of implementing recognition of handwritten digits using Keras. The method detailed in that paper utilizes the MNIST dataset to train their CNN. Since this dataset is used, only the identification of digits will be trained on their neural network. The CNN the article implements is described to be multi-layered, featuring convolutional layers, pooling layers, non-linear layers, and fully connected layers as the final layers [2]. The article also details that the author chose a Stochastic Gradient Descent as the optimization of weights,

since it "checks on a single training example and optimizes the weights" and therefore makes it converge faster [2].

My implementation of the character recognition will be similar to a method discussed in the second literature article concerning the use of Keras and CNN's for OCR of the MNIST dataset. Extensive research was conducted in order to find a dataset which could be utilized to implement not only handwritten digit recognition, but also handwritten letter recognition. Through this research, the EMNIST dataset was selected. Literature article 11 further details this dataset. Since the EMNIST dataset was designed in a format that directly matches the MNIST dataset, a similar process can be taken in regards to the recognition of characters. Therefore the architecture that I will be implementing for the CNN will be similar to that depicted in Figure 1 which is graphical representation of a CNN from the second literature article, where the activation function that I have chosen will be the ReLu (Rectified Linear Unit) function and weight optimization will be determined by Stochastic Gradient Descent.

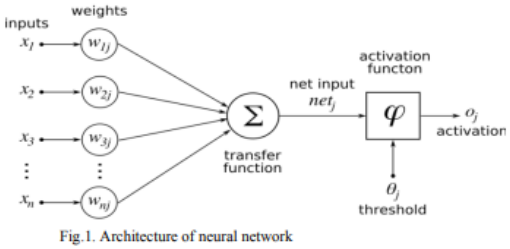


Fig. 1: Architecture of Neural Network [2]

Literature article 10 also details the use of CNN's to implement HCR. This paper documented the method used to identify Arabic characters using a CNN and the CMATERDB 3.3.1 Arabic handwritten digit dataset [10]. This CNN was made up of two convolutional layers, two pooling layers, a Dropout layer, a Flatten layer ("which converts the two-dimensional matrix into a vector"), and finally an output layer. The two convolutional layers include 30 and 15 feature maps, respectively; a kernel size of 55 pixels and 3x3 pixels, respectively; as well as a ReLu activation function for both [10]. The pooling layers are configured to take the maximum value and were therefore configured to a pool size of 22. The Dropout layer was configured to randomly exclude 25% of neurons to reduce overfitting, and the output layer was configured to have 10 neurons due to the 10 classes or digits [10].

My implementation of this project will have to differ from the method proposed in the second literature article, since my project requires the implementation of preprocessing and word segmentation in order to interpret an image of a handwritten document and convert it into a text document. Prior to recognizing characters with my CNN, the image which will be used as the input to the project will require preprocessing and word segmentation. Literature articles 1, 4, 5, and 8 all focus on various techniques to implement word segmentation. Article 1 focuses on two algorithms for segmentation: "recognition-with-segmentation" and "recognition-after-segmentation" [1]. The first algorithm applies recognition directly after a word is segmented out, while the latter algorithm waits to apply recognition until after segmentation has identified all words. The actual segmentation is conducted based on the spacing between the center region of each character. This article also details the use of slant detection and compensation in order to account for characters written on a page which may be in varying degrees of orientation, size and shape [1].

ALGORITHM OF RECOGNITION-WITH-SEGMENTATION

Step 1	if recursive conditions can not be met, return;
Step 2	get recognition results for N: resG1; get recognition results for N,N+1: resG2; get recognition results for N,N+2,N+3: resG3;
Step 3	compare resG1,resG2,resG3, and get the best result R;

Step 4	if R==resG1, go step 5;
	if R==resG2, go step 6;
	if R==resG3, go step 7;

Step 5	for each result, dynamic pruning, get result: res; if $res == failure$ skip current result; else N=N+1; go step 1;
Step 6	for each result, dynamic pruning, get result: res; if $res == failure$ skip current result; else N=N+2; go step 1;
Step 7	for each result, dynamic pruning, get result: res; if $res == failure$ skip current result; else N=N+3; go step 1;

Fig. 2: Algorithm of Recognition With Segmentation from Article 1

Article 4 focuses on line and word segmentation by utilizing the projection profile technique and "tracing the white runs along the base region of the text line" [4]. This means that it uses a technique that observes the spaces in between letters and determines whether they are large enough to be a space between words. This implementation requires compression of the characters which I feel adds a level of complexity which is not needed for my project. Article 5 details the segmentation of unconstrained handwritten strings of digits utilizing a method of "extracting the foreground and background features" of an image and creating points that are turned into segmentation paths, which are then evaluated based on components connected to the left and right of the character being analyzed [5]. Again this implementation of word segmentation seems much more complicated than what is needed for this project implementation. Article 8 focuses on word segmentation identification based on the "oriented sliding window." This paper explains that it observes "the interconnection points between adjacent digits and the cutting path" (the path between these two digits where they can be separated) [8]. Seeing the proposed method in this paper, I could try to emulate a version of this method in order to

implement word segmentation.

I noticed in my research that many conference articles when utilizing the MNIST dataset would seem to focus on the character recognition portion of OCR, but some would never attempt the word segmentation that is required to split up read text into words instead of one long string of characters. My project will attempt to combine both word segmentation and handwritten character recognition utilizing OpenCV, Keras, and Tensorflow.

III. Method

A. Task Environment

The task environment chosen to implement character recognition is the EMNIST dataset. The EMNIST dataset stands for Extended MNIST dataset [11]. This dataset is very similar to the MNIST dataset, but instead of only consisting of the digits 0-9; EMNIST also includes all uppercase and lowercase letters found in the English alphabet. The EMNIST dataset was taken from the NIST Special Database 19 dataset which consists of over 800,000 images [6 and 9]. Like the MNIST dataset, the EMNIST dataset is made up of png's with a resolution of 28 x 28 pixels. More information regarding the Datasets within EMNIST can be seen within Figures 3 and 4. The dataset used to train the model was split into training images and testing images by_Class.

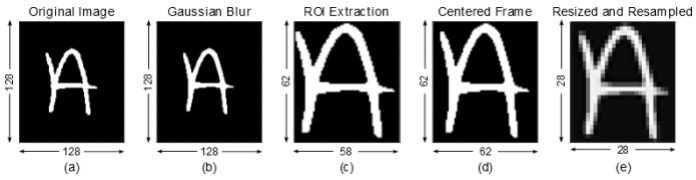


Fig. 3: Image a) is one of the images in the NIST SD 19 dataset and Image e) is the same image as it is in the EMNIST dataset from Article 11

Name	Classes	No. Training	No. Testing	Validation	Total
By_Class	62	697,932	116,323	No	814,255
By_Merge	47	697,932	116,323	No	814,255
Balanced	47	112,800	18,800	Yes	131,600
Digits	10	240,000	40,000	Yes	280,000
Letters	37	88,800	14,800	Yes	103,600
MNIST	10	60,000	10,000	Yes	70,000

Fig. 4: EMNIST Datasets (Graphic from Article 11)

B. Methodology

Initially, the proposed method for implementing this project was to take an image of a handwritten document, preprocess it using OpenCV, break it up into lines, split those lines up into words, split those words up into letters (by finding interletter spaces), and then preprocess those letters into a png with a resolution of 28 x 28 which matches the resolution of the training values from EMNIST. The actual methodology involved a very similar process, although there were a few differences. The inputted handwritten image was imported into the program utilizing OpenCV. The same python library was then used to convert the image to a grayscale image which was then thresholded to black and white using Otsu's Binarization and Binary Thresholding. Otsu Binarization is a function which automatically determines the optimum threshold value based on the pixel location in an image [13]. Any pixel in the image above the threshold was converted to the max value (black) and anything below was converted to white. Once the grayscale image was thresholded, this new image was inverted using a bitwise_not function. This inverted image was then passed through the findContours function built into OpenCV. This function identifies the contours or "curves joining all the continuous points (along a boundary), having [the] same color or intensity" [12]. By finding the contours, the individual characters on the input image are able to be identified if they are properly spaced. If there is any overlap between the characters on the input image, the findContours function could possibly give incorrect character detection. Once the contours are found, they are sorted top-to-bottom using the imutils image processing python library. Each image is then resized to 28 x 28 pixels to match the EMNIST dataset images and then saved as individual character images. Each character/contour image is saved as a png with the pixel location of the top left corner of the character saved in the image name. This coordinate location is used later on to implement both line and word segmentation.

The line and word segmentation was implemented using the recognition-after-segmentation method. Using this method, the characters are first segmented into lines and words before the CNN model predicts the characters. The line segmentation is implemented by extracting the coordinates of the top left corner of the saved character images and comparing this coordinate with the coordinates of the next image. If they are within a specified threshold, the names of the images are added to a list representing the same row. If they are different then the second image is added to a new row list. Since the imutils sortContours function was used as mentioned above, the following image should always be greater than or equal to the vertical coordinate of the previous image. Once the images were sorted by line, each line was individually sorted by column (x-value) in ascending order using the built-in python sort function. The word segmentation was implemented in a method very similarly to that mentioned in Article 4 which detects the white space between characters. Since the coordinates of each character were known, the space between each of the now sorted characters could be calculated. If this calculated space was greater than the specified threshold, then a space between words was found and a space was added to the list which was outputted to the predicted text file. This predicted text file is the text file which is compared to the answer key text file.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 128)       663680
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3211392
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
gaussian_noise (GaussianNois (None, 64)                0
_________________________________________________________________
activation (Activation)      (None, 64)                0
_________________________________________________________________
preds (Dense)                (None, 62)                4030
=================================================================
Total params: 3,906,014
Trainable params: 3,906,014
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5: CNN Architecture

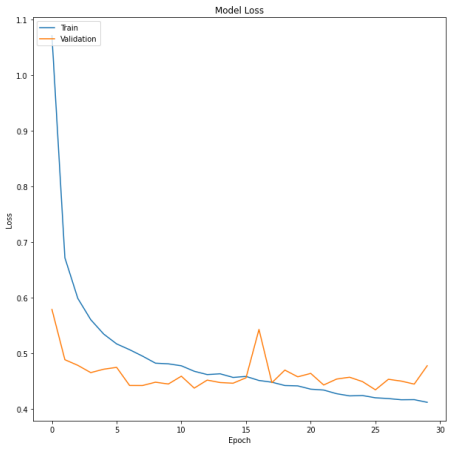


Fig. 6: Model Loss Plot of Model 21

The CNN model was implemented using Keras and Tensorflow in a Jupyter Notebook. The CNN architecture was similar to the ones mentioned in literature articles 10 and 2. It can be seen in Figure 5. The architecture consisted of an input layer which was a 2D convolution layer with 32 feature maps, a kernel size of 2x2 and an input shape of (28,28,1). After this a max_pooling2d layer with a pool size of 2x2 was added. Next another convolution layer with 64 feature maps and a kernel size of 3x3 was used. After this a 25% dropout layer was added to help reduce overfitting. Following this a third and final 2d convolution layer with a feature map size of 128 and a kernel size of 7x7 was added. Following this, a flatten layer was needed to turn the 2D matrix from the output of the convolution layer into a vector which could to fit a dense layer. This dense layer accepted 128 units and then fed into another dropout with a 50% dropout. This layer was followed by another dense layer with 64 units. To help reduce overfitting even further, a GaussianNoise layer with an input of 0.1 was added. Following this an activation layer with a ReLU function was added. The final layer was a dense function with 62 neurons, since there are 62 different classes or the combination of digits and uppercase and lowercase letters in the EMNIST dataset. The activation function used throughout this model was the ReLU (rectified linear unit). The ReLU function was chosen, since it helps deal with the issue of vanishing gradients [2]. Overall, this created a model with over 3.9 million parameters. This model yielded the best results of the 32 models trained and tested. In addition to this model, the adam optimizer as well as the categorical_crossentropy loss function were utilized when compiling it and the model was ran for 30 epochs.

The CNN model was trained on 20% of the train split of the EMNIST dataset in order to cut down on the total train time. This 20% was 139,586 images used for training and 23,265 images for testing. The trained model had a model loss as seen in Figure 6. Based on this model, there was very little overfitting at this number of epochs.

Each of the character images was then fed through the CNN developed in Keras and then the character's value was outputted as either a digit value, or a letter value (uppercase or lowercase).

C. Validation

The method of implementing this project is deemed successful if it can take in an image / scan of a handwritten document written in English and convert it to a text document. Lines and words must be segmented properly. In order to validate that the program was able to convert the image to text properly through character recognition and word segmentation, a text document containing the same notes as written on the handwritten document will act as an answer key. Both the text document generated through character recognition as well as the answer key will be compared and based on differences between the two, a score will be generated defining the accuracy of the program.

I hypothesize that the program will have an accuracy no greater than 95%. The character recognition portion of the program should have an accuracy greater than 90%, but when including word segmentation, that may drop the accuracy to below 90%. Based on papers detailing research focusing solely on the use of Keras and CNN's to implement OCR for digit recognition using the MNIST dataset, accuracy for that was between 95 and 99%, which is a much easier problem to solve, since only digits were being recognized. The goal of this project requires recognition of digits and upper and lowercase letters, as well as word and line segmentation, therefore the accuracy will be lower.

IV. Results

Through implementing this HCR program, over 32 different models were built and trained in order to fine-tune the model's parameters. The corresponding predicted values of these 32 models can be seen in Figure 7. Of all the models trained and tested, the best one was Model 21 as detailed in Figure 5.

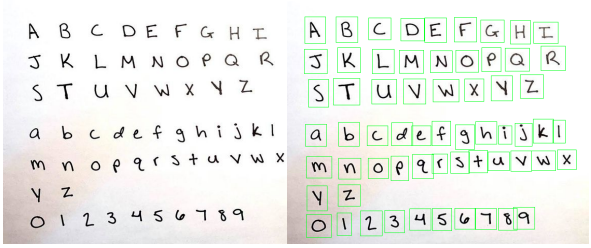
This model was able to predict 37.09677% of the handwritten input characters correctly. Originally I had thought that it was only able to predict 12.90% of the characters correctly, but upon further inspection I remembered that all of the EMNIST dataset images were rotated and therefore I needed the handwritten input images to also be rotated. Once that was implemented properly, there was significant improvement in the predictions. Upon analysis, it appears that many of the lowercase letters are mistakenly predicted as uppercase ones. In the cases where the letters are incorrectly predicted, the letters are often mistaken for ones that are fairly similar: i.e. ”A and R”, ”D and O”, ”W and w”, ”5 and S”. Characters ”i” and ”j” seem to be some of the few letters that had predictions which were very dissimilar. This is probably due to the fact that they are the only letters with dots, which probably make predicting more difficult. The handwritten input image can be seen in Figure 8. The handwritten characters and their corresponding predictions can be seen in Figures 9 and 10. Figure 10 shows what was predicted and compares that to what the character actually was.

Models	Test Loss	Test Accuracy	TestImagePercentage	TI_Correct	TI_Incorrect	PredictedCharsPercentage	Epochs	Parameters
Save1			78.97%	91855	24468	3.23%		
Save2								
Save3			85.73%	99728	16595	6.45%		
Save4								
Save5								
Save6	0.459255546	0.994974673	83.93%	97632	18691	3.23%		
Save7								
Save8	0.431581586	0.995189786	84.77%	98605	17718			
Save9	3.679705	0.98388344	5.44%	6330	109993	1.61%		
Save10	0.434203178	0.99520576						
Save11	0.421948373	0.99531132	85.08%	98965	17358	6.45%	5	
Save12	0.421948373	0.99531132	84.78%	98620	17703	4.84%	10	
Save13								
Save14								
Save15			84.16%	97897	18426	4.84%	2	
Save16	0.440481752	0.995226443	84.82%	98663	17660	9.68%		
Save17	0.45881018	0.995030463	84.22%	97963	18360	8.06%		3,906,014
Save18	0.461376369	0.994991839	84.07%	9779	1853	9.68%	20	
Save19	0.461376369	0.994991839	5.29%	615	11017	1.61%		
Save20	**0.476704925**	**0.99501276**	**83.91%**	**9760**	**1872**	**12.90%**	30	3,906,014
Save21	0.472225368	0.995184958	84.66%	98478	17845	12.90%	30	3,906,014
Save22	0.472592354	0.995015085	84.02%	19548	3717	3.23%	20	3,906,014
Save23	0.498704582	0.994955063	83.80%	9748	1884	4.84%	20	7,125,598
Save24	0.473213732	0.994853139	83.56%	97194	19129	9.68%		
Save25	3.6858239	0.983870387	5.52%	1284	21981	1.61%	100	
Save26	0.425023496	0.995323002	85.13%	99021	17302	8.06%	20	2,046,558
Save27	0.470564455	0.995122373	84.50%	98292	18031	4.84%	120	2,046,558
Save28	0.407821566	0.995368242	85.33%	99253	17070	9.68%		2,046,558
Save29	0.392757058	0.995523393	85.83%	99844	16479	12.90%		2,046,558
Save30	0.488137662	0.994810224	83.42%	97031	19292	9.68%		3,668,830
Save31	0.468271022	0.9949366	83.82%	97503	18820	9.68%	100	
Save32	0.016084807	0.967846394	0.27%	317	116006	1.65%		

Fig. 7: Model Overview with Corresponding Accuracy

V. Discussion

Looking back on the hypotheses made in the Validation section of this paper, they were not supported by the results. Using the CNN model described above, the percentage of characters predicted correctly was not within an accuracy greater than 90%. Additionally, based on the prediction percentage of the test images, the percentage of 84.65909%



(a) Original Image (b) With Contours

Fig. 8: Handwritten Image

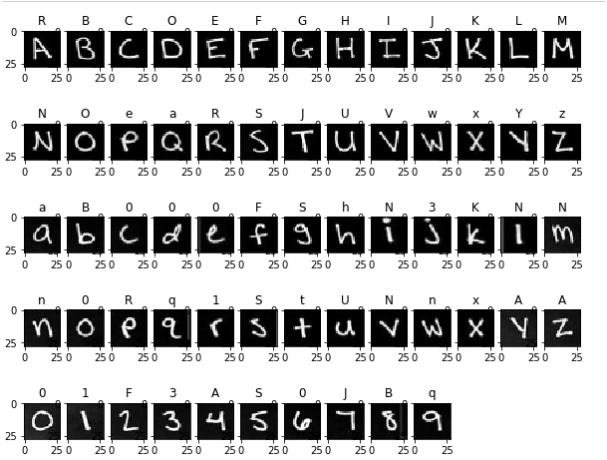


Fig. 9: Predicted Characters with Images

```
Percentage similar:  37.096774193548384 %
Predicted :  RBCOEFGHIJKLMNOeaRSJUVwxYzaB000FShN3KNNn0Rq1StUNnxAA01F3AS0JBq
Answer Key:  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
```

Fig. 10: Predicted Results

would be the highest percentage the current model could predict of a set of data correctly. Further fine-tuning of the hyper-parameters would be required to reach a percentage of correctness higher than 37%.

If this work were to be continued in the future, other possible word segmentation implementations could be used or pursued. The current word separation technique relies heavily on the handwritten characters being the same size and spacing apart. Other techniques such as end punctuation identification or entire word recognition / identification could be more robust.

VI. Conclusion

In conclusion, implementation of this project was conducted in Jupyter Notebook utilizing Keras, OpenCV, and Tensorflow in order to take advantage of the EMNIST dataset. The first component of this project was to preprocess a handwritten image and then implement word segementation. Once this was done, a CNN model was designed and implemented to perform the handwritten character recognition using the EMNIST dataset. Once the model validated on EMNIST, the handwritten characters were sent into the CNN and predicted. In the end, all portions of the project were achieved, and the program was able to accept an image of a text document and convert it into a machine-encoded text file.

Acknowledgment

The method proposed and investigated within this paper will be conducted on the EMNIST dataset. This portion of the paper is dedicated to acknowledge the creator of this dataset as well as all who worked on it.

REFERENCES

[1] Aiquan Yuan, Gang Bai, Po Yang, Yanni Guo, and Xinting Zhao. "Handwritten English Word Recognition Based on Convolutional Neural Networks." 2012 International Conference on Frontiers in Handwriting Recognition, 2012, doi:10.1109/icfhr.2012.210.

[2] Arora, Shefali, and M. P. S Bhatia. "Handwriting Recognition Using Deep Learning in Keras." 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), 2018, doi:10.1109/icacccn.2018.8748540.

[3] Rohan Vaidya, Darsha Trivedi, Sagar Satra, and Mrunalini Pimpale. "Handwritten Character Recognition Using Deep-Learning." 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, doi:10.1109/icicct.2018.8473291.

[4] Mohammed Javed, P. Nagabhushan, and B.B. Chaudhuri. "A Direct Approach for Word and Character Segmentation in Run-Length Compressed Documents with an Application to Word Spotting." 2015 13th International Conference on Document Analysis and Recognition (ICDAR), 2015, doi:10.1109/icdar.2015.7333755.

[5] Sadri, J., et al. "Automatic Segmentation of Unconstrained Handwritten Numeral Strings." Ninth International Workshop on Frontiers in Handwriting Recognition, doi:10.1109/iwfhr.2004.21.

[6] Sherena.johnson@nist.gov. "NIST Special Database 19." NIST, 27 Apr. 2019, www.nist.gov/srd/nist-special-database-19.

[7] "Emnist : TensorFlow Datasets." TensorFlow, www.tensorflow.org/datasets/catalog/emnist.

[8] Gattal, Abdeljalil, and Youcef Chibani. "Segmentation and Recognition Strategy of Handwritten Connected Digits Based on the Oriented Sliding Window." 2012 International Conference on Frontiers in Handwriting Recognition, 2012, doi:10.1109/icfhr.2012.265.

[9] Patricia.flanagan@nist.gov. "The EMNIST Dataset." NIST, 28 Mar. 2019, www.nist.gov/itl/products-and-services/emnist-dataset.

[10] Ashiquzzaman, Akm, and Abdul Kawsar Tushar. "Handwritten Arabic Numeral Recognition Using Deep Learning Neural Networks." 2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (IcIVPR), 2017, doi:10.1109/icivpr.2017.7890866.

[11] Cohen, Gregory, et al. "EMNIST: Extending MNIST to Handwritten Letters." 2017 International Joint Conference on Neural Networks (IJCNN), 2017, doi:10.1109/ijcnn.2017.7966217.

[12] "Contour Approximation Method." OpenCV, docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

[13] "Image Thresholding." OpenCV, docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.

[14] "Imutils." PyPI, pypi.org/project/imutils/.