Crowdsourcing Online Handwriting Acquisition to Develop and Deploy a Unicode Character Classifier

Francisco Álvaro and Daniel Martín-Albo
WIRIS Math and Science
Barcelona, Spain
{falvaro,dmas}@wiris.com

***Abstract*—There are thousands of Unicode characters and hence it can be hard to visually find a particular one. For this reason, we aimed at developing a tool that allows to handwrite a character and receive a list of the most similar candidates to that input. This tool will be integrated in a math editor which handles more than 5,000 different Unicode characters. Since no public datasets were found to fit our needs, we crowdsourced the acquisition of online handwritten data for training purposes. We developed a neural network combining convolutional layers with shape-based features to classify online handwritten Unicode characters. To make the model more robust to input variability, we used data augmentation in the form of affine transformations. We achieved a top-20 error rate of 12.64% on validation data and received positive feedback from users, thus validating that crowdsourcing is a proper method for online handwriting acquisition. Finally, we deployed the model wrapped in a JSON-based REST API and released a public demo using it. This way, we present the full development cycle of a Unicode character classifier.**

***Index Terms*—crowdsourcing, online, handwriting, Unicode, character, classification, neural networks**

I. Introduction

Throughout history, humankind has developed a great variety of writing systems and currently many different scripts and symbols are used all over the world. Nowadays, Unicode is the computing dominant standard for internal processing and storage of those symbols.

If we write a love letter, a keyboard is usually enough to enter all the necessary Unicode characters. However, if we write a scientific paper we might need to insert special characters (for example, mathematical symbols) not available on the keyboard. This need is normally solved by using a text editor, but even in that case it can still be hard to find a specific character among hundreds (or thousands) of options.

A much easier and comfortable approach would be to draw the character and get a list of the most similar characters from where we could select the one we want to insert. That is the goal of this work, to develop a tool that helps users to find a character by just drawing it. This tool would be used to edit math and text, targeting more than 5,000 Unicode characters.

The shape of a Unicode character can greatly differ from its handwritten counterpart due to human handwriting variability and style. For example, the handwritten version of $\mathbb{R}$ could be $\mathbb{R}$, or $\mathtt{a}$ could become $\mathcal{a}$. Therefore, in order to train a pattern recognition classifier we need an online handwritten character dataset labeled with their corresponding Unicode number.

Although many public handwritten character datasets are available (e.g. CROHME dataset [1] or IAM Handwriting Database [2]) they do not meet our requirements. For example, these datasets altogether contain around 100 different classes, which corresponds roughly to only 2% of the characters that we intend to account for.

Creating and labeling a dataset can be tedious and time-consuming. For this reason, we employed crowdsourcing to collect the required handwriting samples. Although previous studies have used crowdsourcing to collect data for machine learning tasks [3], to the best of our knowledge, there is no previous work that crowdsourced the creation of online handwriting data. This is one of the contributions of this paper.

Once a collection of training samples was available, we developed (and evaluated) a model able to provide a list of candidates for a given handwritten input. Finally, we explored how to deploy the system in order to be integrated with the future version of MathType editor.[1] Thus, another contribution of this paper is the presentation of the full development cycle (from requirements gathering to deployment) of a handwritten Unicode character recognizer. Experimental results with quantitative evaluations as well as qualitative feedback from users validate the proposed approach.

II. Related Work

Unicode [2] is a computing standard for the consistent encoding of symbols. Currently it contains more than 136K characters covering 139 scripts (modern and historic) as well as multiple symbol sets. However, common applications only use a small subset of this huge set of characters.

In this work we are interested in providing a tool to assist users to find Unicode characters by drawing them. There are tools already available that provide this option. For example, Shapecatcher [3] is a web application that implements a classifier based on shape contexts similarities [4]. This application supports the recognition of about 10K Unicode characters.

[1] Available at: http://www.wiris.com/mathtype/
[2] Official website at: http://www.unicode.org/versions/Unicode10.0.0/
[3] Available at: http://shapecatcher.com/

Moreover, Detexify[4] is a web application that focuses on LaTeX, currently accounting for about 1,000 characters. This tool uses a nearest-neighbour classifier with dynamic time warping as distance measure. Finally, Google provides a tool[5] integrated with Google Docs to insert special characters that accepts handwritten input. Unfortunately, the handled character set and its underlying engine are unknown.

In order to develop an application like those discussed above two things are needed: labeled data and a model able to account for it. There are many datasets containing online handwritten characters (e.g. CROHME dataset [1], CASIA database [5] or IAM Database [2]) but they only cover specific domains or scripts which are not enough for our task requirements.

Crowdsourcing is a sourcing model in which goods are obtained from a large group of Internet users; dividing work between participants to achieve a cumulative result. Lately, many research papers have relied on this approach to create data for different machine learning related tasks [3], such as machine translation, speech recognition, computer vision, sentiment analysis or information retrieval.

Regarding handwriting data, several tasks have used crowdsourcing to transcribe offline handwritten text [6], i.e. they had a set of images of handwriting and collected their transcriptions. On the other hand, Le et al. [7] used crowdsourcing to create images containing handwriting and their corresponding transcripts. In that task, they asked users to write down some text in a piece of paper and then upload a photo or scan of that paper. In a subsequent task, users had to transcribe the images previously collected. However, we have not found any publication that used crowdsourcing to create an online handwriting dataset.

Once labeled data is available we can use a machine learning algorithm to *learn* how to recognize online handwritten Unicode characters. Many approaches have been presented in the literature for character recognition [8], such as hidden Markov Models [9], support vector machines [10] or neural networks [11]. Converting online patterns into offline patterns makes possible to apply offline techniques for recognizing online characters [12]. Moreover, many previous works have opted for combining both modalities, by fusing them [13] or by using a representation like path iterated-integral signature [14] that produces an image where each pixel also encodes information regarding the input trajectory.

Generally, online systems achieve better performance than offline systems [15], because they can use more information from the trajectory of the handwriting. However, offline methods are less dependent on input order or writing styles such that in some cases they can produce better results [12]. Current state-of-the-art approaches to offline recognition make essential use of Convolutional Neural Networks [16,17] (CNNs). They have been successfully used for solving many image-based tasks [18]. In this work, we will explore the application of CNNs to handwritten Unicode character recognition.

III. Crowdsourcing Data

Since there was no available dataset that met our requirements, we decided to create our own dataset using crowdsourcing to train a Unicode character classifier.

Amazon Mechanical Turk[6] (MTurk) is probably the most well-known crowdsourcing platform. Employers (or requesters) are able to post jobs, known as Human Intelligence Tasks (HITs). Workers can browse among existing jobs and complete them in exchange for a monetary payment set by the requester. HITs are commonly short and their reward is as low as $0.01. Requesters have several HIT templates available at MTurk, but none of them is suitable to accept handwritten input. We had to design an *external question* such that the HIT contains a frame to an external website. When designing a HIT one has to consider that workers are probably non-experts in the task and low payments do not enforce high quality results [3]. A HIT could also have specific requirements, for instance, being a native writer or using a touchscreen device.

Crowdsourcing is quite appropriate for collecting handwritten Unicode characters for our task. The use case we address is that a user (probably non-expert) wants to find a particular character so s/he handwrites it using any input device (mouse, touchscreen, etc.) in any scenario. They could even not know the symbol because they are copying it from another source.

Figure 1 shows the Graphical User Interface (GUI) displayed to crowdsourcing workers with simple and clear instructions. The GUI includes options to clear the canvas and to undo the last action. In addition to the frontend, we needed to develop a

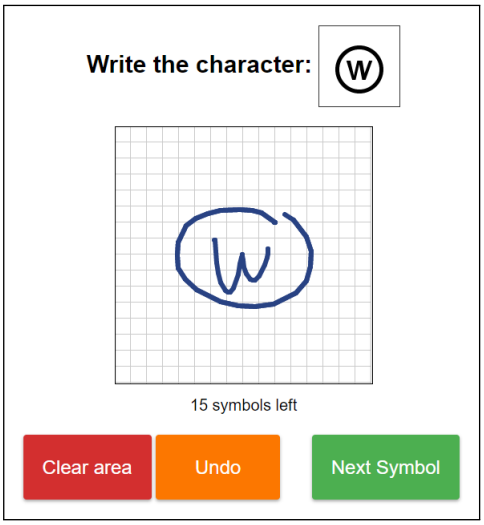

Fig. 1: GUI displayed to crowdsourcing users for collecting online handwritten Unicode characters.

[4]Available at: http://detexify.kirelabs.org

[5]Open a document at Google Docs and '*Insert* > *Special characters*'.

[6]Official website at: https://www.mturk.com/

backend server in charge of proposing new characters such that we collect samples from a list of $5{,}456$ Unicode characters.

Once the collecting pipeline was available, we created HITs that consisted on handwriting 15 characters for a reward of \$0.05. We created $3{,}638$ HITs at MTurk to collect about 10 samples per class for a total of $54{,}570$ samples at the cost of \$0.004 per sample (since Amazon's fees are 20% of the HIT reward). Workers completed all HITs in less than 24 hours, creating a total of $56{,}487$ samples. It should be noted that this number is greater than expected because some workers left a HIT before finishing it, but those samples were also collected.

Given that MTurk does not provide any direct quality control about the task result, we had to assess the correctness of the collected samples. Initially, we tried to use statistical methods for outliers detection such as z-score with robust estimators [19], but this technique did not yield conclusive results given the low number of samples per class. For this reason, we created a simple tool to manually validate the data, given that it is easy to spot a bad sample among samples of the same class. As a result of this validation, 629 samples were found to be incorrect, leaving a total of $55{,}858$ correct samples. In this case, manual validation made sense due to the low complexity and small size of the task. For bigger tasks, another approach would be to also use crowdsourcing to validate the collected data, which is known as iterative tasks [20].

After including samples collected during development and testing of the data acquisition pipeline, the number of samples amounted to $57{,}703$. Each class had 10 samples on average, where the distribution of the classes is not exactly balanced due to the high number of concurrent requests during collection.

IV. Proposed System

A. Data Representation

Online samples were represented as raster images enriched with a set of global features.

1) Rasterization: First, the (x,y)-coordinates of each sample are normalized in size such that its bounding box has a size of 64×64. This normalization process provides a representation robust to scale and translation. Then, coordinates are properly rasterized into a 64×64 matrix.

2) Global Features Extraction: Size normalization performed during rasterization discards information of the original shape useful to distinguish between symbols that are very small (e.g. a dot, or '$\circ$' versus '0'), or very thin (e.g. '_', or '/' versus '|'). For this reason, we compute a set of global features in order to provide additional information about the original input.

Given the original sequence of points with a bounding box of size $w_s \times h_s$ that was drawn on a canvas of size $w_c \times h_c$, we compute the following 8 features:

$$\left[\frac{w_s}{h_s}, \frac{w_s + h_s}{w_c + h_c}, \frac{w_s}{w_c}, \frac{h_s}{h_c}, \frac{\frac{\mu_{10}}{m_{00}}}{w_s}, \frac{\frac{\mu_{01}}{m_{00}}}{h_s}, 2\frac{\sqrt{\frac{\mu_{20}}{m_{00}}}}{w_s}, 2\frac{\sqrt{\frac{\mu_{02}}{m_{00}}}}{h_s} \right]$$

being the last four values image moments [21], where m_{pq} and μ_{pq} represent the geometric moment and central moment of order $(p + q)$, respectively.

B. Neural Network Architecture

We use an architecture inspired by the VGG ConvNet configuration [18], composed of blocks with very small receptive field convolutions followed by a pooling operation.

Each convolutional block (ConvBlock) contains a 3×3-sized kernel convolutional layer. Dropout [22] is applied at the input of the block in order to reduce overfitting. Moreover, Batch Normalization [23] is used to normalize the nonlinear activation function inputs. Exponential Linear Units (ELUs) [24] are employed as activation functions. The output of the activation is connected to a Pooling Layer with maximum operation (Maxpool) and nonoverlapping 2×2-sized kernels, in order to reduce the input size.

The network deals with two different inputs for each sample: a raster image and a set of global features. Three ConvBlocks provide an encoded representation of the raster image. This representation is concatenated with the set of global features that finally goes through a linear layer with softmax activation that maps its output to the number of classes to be recognized. As a summary, Figure 2 shows the chosen network architecture. Our toolkit was built upon Apache MXNet,[7] a well-known deep learning framework that supports both CPU and GPU devices.

C. Learning

We trained the network to minimize the cross-entropy objective function. We performed the optimization with stochastic gradient descent employing the RMSProp method [25] to incrementally update the parameters of the network on each batch of 256 samples. We used a learning rate of 0.001 and training was stopped when the top-20 error rate on the validation set did not improve for 20 epochs.

In order to reduce overfitting as well as to make the model robust to rotation and shear variations, we performed data augmentation on each training sample. We applied affine transformations (rotation and shearing) dynamically and independently for each training sample. Thus, the exact same sample is virtually never observed twice during training. The parameters controlling the distortions, i.e. rotation and shear angles, are sampled from a fixed distribution. These parameters as well as all the hyperparameters of the network were automatically tuned using Bayesian optimization [26].

D. Deploying to Production

The neural network that was finally deployed to production was trained using the hyperparameters found during experimentation and using all data available (both training and validation sets).

[7] Official website at: https://mxnet.incubator.apache.org/

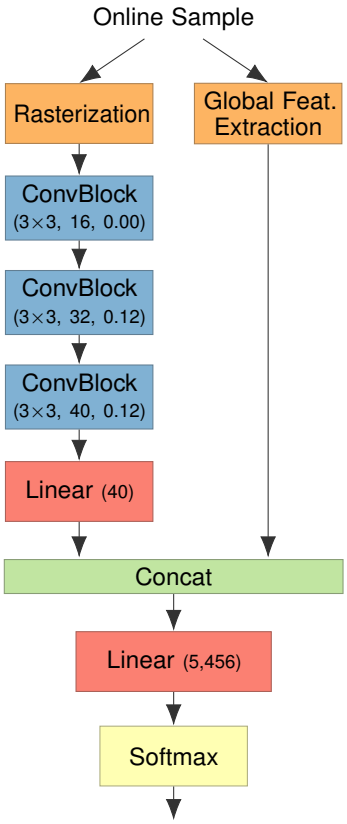

Fig. 2: Architecture of the neural network used in this work. Each ConvBlock displays, from left to right: the kernel size, the number of filters and the value of dropout applied.

We relied on Model Server for Apache MXNet[8] (MMS) to deal with the deployment of this model. MMS not only accepts MXNet models but also Open Neural Network Exchange[9] (ONNX) models, which is an open format to represent deep learning models supported by the most popular frameworks (e.g. PyTorch, MXNet, Tensorflow, Caffe2, Chainer, or CNTK).

MMS wraps a deep learning model inside a JSON-based REST API to become backend agnostic. This way, we can use any technology stack to integrate it on websites or native mobile apps. Our API provides a service that receives a sequence of points representing a character, as well as the input canvas size where it was written. The response is a JSON containing a list of 20 Unicode number candidates for the input provided.

Moreover, we implemented a web application that consumes the web service as a publicly accessible demo.[10]

V. Evaluation

We conducted experiments to evaluate 1) the accuracy of our recognizer for predicting the label of handwritten Unicode characters and 2) the utility of the developed tool.

[8]Official website at: https://github.com/awslabs/mxnet-model-server
[9]Official website at: https://onnx.ai/
[10]Available at: https://www.wiris.net/demo/hand/tests/en/test-unicode.html

A. Dataset

We carried out experiments using a private dataset collected following the procedure described in Section III. This dataset comprises 5,456 Unicode characters performed by hundreds of users for a total of 57,703 online handwritten samples.

Each participant was asked to handwrite a Unicode character that was displayed as a typographic character (see Figure 1). Figure 3 depicts several Unicode characters performed by users.

Fig. 3: Examples of Unicode characters collected from users.

B. Design and Procedure

We defined a validation set, composed of one sample per class (5,456 samples in total). In order to evaluate the influence of the amount of training data, we defined four different training partitions containing 25%, 50%, 75% and 100% of the remaining samples (52,247 samples).

For each sample in the validation set, we compared the labels predicted by the system (described in Section IV) with the true label using different top-k error rates, where $k \in \{1, 10, 20\}$. Top-k error rate considers the prediction incorrect if the ground-truth label is not in the best k predictions.

C. Results

Figure 4 displays the validation results, where top-1, top-10 and top-20 error rates are reported for the different train partitions. The best top-1, top-10 and top-20 error rates are 60.37%, 20.49% and 12.64%, respectively. In all three cases, the best results are achieved when 100% partition is used. The best error rate for top-20 was achieved after training for 65 epochs.

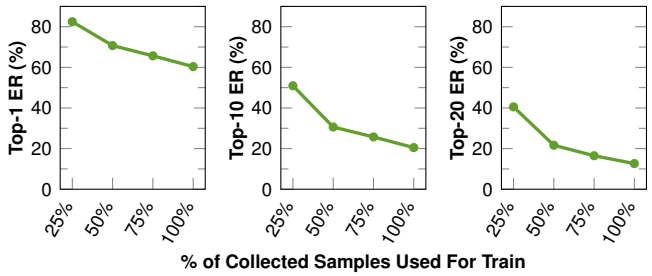

Fig. 4: From left to right: top-1, top-10, top-20 error rates for the different training partitions.

Moreover, we can observe that all top-k errors decrease when the amount of training data is bigger. The improvement achieved by using more data is also reduced when the model has more samples to properly learn the classes involved.

D. Informal User Tests

We informally evaluated the tool with 21 users (5 females) aged 22-41 (M=28.9, SD=4.8). Participants had many different backgrounds (e.g. Engineering, Compute Science, Humanities or Mathematics) and most of them had used a math editor at some point, although nobody had technical background in handwriting recognition. There was no economic compensation for the participants.

Each participant had to handwrite ten Unicode characters extracted from a list chosen at random. No restrictions were applied to the input device used. After completing the task, users were asked to score the Unicode character recognition tool in a 5-point Likert scale in terms of usefulness, accuracy and efficiency. More precisely, users had to express their level of agreement/disagreement with the following statements:

- **Accuracy**: I could find the symbol I was looking for.
- **Usefulness**: The candidates proposed were visually similar to my handwritten input.
- **Efficiency**: I think I could find a symbol faster than with an editor/toolbar.

The feedback obtained from users is summarized in Table I, where the mean subjectivity score and the standard deviation with 95% confidence are displayed. Overall, users expressed a very positive perception of the tool developed. The Unicode character candidates are visually similar to the provided input and the method would represent an efficient way to insert special characters with respect to a traditional toolbar.

TABLE I: Mean subjectivity scores (higher is better) of users testing the Unicode characters recognizer.

Perception	Mean Subjectivity Score ($\pm$ SD)
Usefulness	4.76 ± 0.18
Accuracy	4.76 ± 0.22
Efficiency	4.57 ± 0.41
Overall	4.70 ± 0.17

VI. Discussion

In this section we discuss several topics found during the development of this work, and we point out some limitations, but also opportunities for future work.

Manual verification of data and experimental results validate crowdsourcing as a good resource for collecting this type of data. Nevertheless, we would like to remark some aspects to take into account for future online handwriting acquisitions.

Crowdsourcing workers are in general non-experts and one of the effects we noticed is that when they are confronted with unknown characters they tend to reproduce them exactly as requested. Figure 5 shows some examples of requested characters and their collected samples, were we can see the difference of variability for different characters (some of them more common than others).

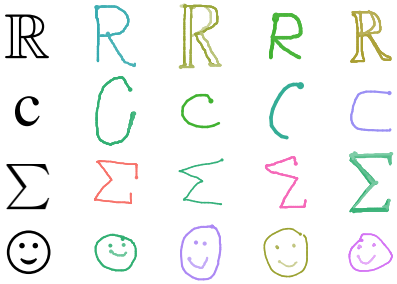

Fig. 5: Examples of Unicode characters (leftmost) and some handwritten samples collected from users.

Bad samples can be produced due to unintentional (user makes a mistake) or intentional (users tries to cheat on the task) errors. It is recommended to include some automatic metrics to assess whether the task has been properly completed. For instance, if we wanted to crowdsource more samples, we could compute the top-20 error rate of each new sample and reject them based on a preset (conservative) threshold. If we detect a worker intentionally producing bad samples, MTurk provides a way for rejecting a user task (such that they are not paid) or even for blocking that particular worker (such that they will not be able to see our future tasks).

Moreover, one could argue that there is a directly proportional relationship between the amount of compensation for a task and the quality of the obtained samples. However, (counter-intuitively) this does not seem to be true [27] and in some cases there could even be an inverse relationship since it is more tempting to cheat on tasks with higher rewards. The interested reader may refer to Callison et al. [3] for more recommendations on crowdsourcing quality control.

Although the scenario we target provides online information and the dataset we collected contains online samples, we decided to use mainly their offline representation for recognition. We made this decision given that characters can be potentially written in many ways and we were collecting just 10 samples per class (we targeted 5,500 classes). The network combines an offline representation with global features extracted from the online information. The choice for this architecture along with data augmentation applied to train data, yield a system robust to affine transformations.

We selected the top-20 error rate as our main evaluation metric because it seems a good number of samples to display to the user (in a 4×5 matrix). Furthermore, many Unicode characters are very similar or even their handwritten version is virtually identical (see Figure 6). This is why top-1 error rate is high, but top-10 and top-20 present a much better performance. Also, given the system's performance tendency when train data is increased, it seems worth collecting more data for further developing the tool. Another interesting option would be to explore the generation of synthetic human-like samples [28].

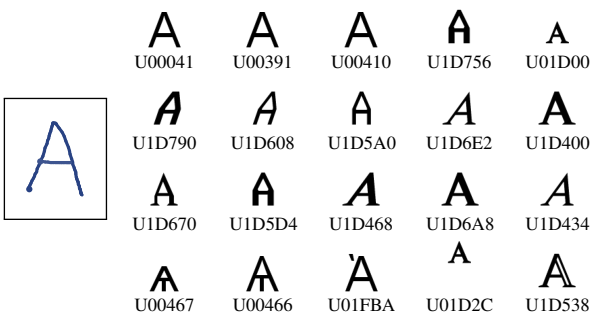

Fig. 6: Example of the tool displaying very similar Unicode characters for a given handwritten input.

Moreover, we would like to add that, even if the crowdsourced data contained some errors, deep learning models have proved to successfully learn with noisy training data [29].

Finally, comparative user tests with other similar tools (see Section II) were discarded given that comparison would not have been fair since systems did not use the same training data and none of them target the same set of characters.

VII. Conclusion

In this work we have presented the full development cycle of a tool for inserting Unicode characters by drawing them. After analyzing the problem and gathering the requirements, we crowdsourced the creation of a dataset with $57{,}703$ online handwritten Unicode characters distributed in $5{,}456$ classes. Using these data, we developed a machine learning model able to classify handwritten Unicode characters. Experimentation with both data and users validate the proposed approach. In order to enhance future developments in this field, we discussed the results obtained regarding the data collected and the system accuracy on validation data. Finally we deployed the trained model and published an online demo.

Acknowledgment

We would like to thank the MTurk workers that completed our tasks as well as the users that carried out the evaluation tests. This work has received funding from the EU's Horizon 2020 programme under grant agreement No 731861 (iMuSciCA).

References

[1] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain, "ICFHR2016 CROHME: Competition on recognition of online handwritten mathematical expressions," in *Proc. Intl. Conf. on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2016, pp. 607–612.

[2] M. Liwicki and H. Bunke, "IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard," in *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2005, pp. 956–961.

[3] C. Callison-Burch and M. Dredze, "Creating speech and language data with Amazon's Mechanical Turk," in *Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010, pp. 1–12.

[4] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE T. Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 509–522, 2002.

[5] D.-H. Wang, C.-L. Liu, J.-L. Yu, and X.-D. Zhou, "CASIA-OLHWDB1: A database of online handwritten chinese characters," in *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2009, pp. 1206–1210.

[6] A. S. Lang and J. Rio-Ross, "Using Amazon mechanical turk to transcribe historical handwritten documents," *The Code4Lib Journal*, vol. 15, 2011.

[7] A. Le, J. Ajot, M. Przybocki, and S. Strassel, "Document image collection using Amazon's Mechanical Turk," in *Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010, pp. 45–52.

[8] Ø. D. Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition - a survey," *Pattern recognition*, vol. 29, no. 4, pp. 641–662, 1996.

[9] S. K. Parui, K. Guin, U. Bhattacharya, and B. B. Chaudhuri, "Online handwritten bangla character recognition using HMM," in *Proc. Intl. Conf. on Pattern Recognition (ICPR)*, 2008, pp. 1–4.

[10] C. Bahlmann, B. Haasdonk, and H. Burkhardt, "On-line handwriting recognition with support vector machines: A kernel approach," in *Intl. Workshop on Frontiers in Handwriting Recognition*, 2001, pp. 49–54.

[11] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.

[12] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí, "Offline features for classifying handwritten math symbols with recurrent neural networks," in *Proc. Intl. Conf. on Pattern Recognition (ICPR)*, 2014, pp. 2944–2949.

[13] R. Rampalli and A. G. Ramakrishnan, "Fusion of complementary online and offline strategies for recognition of handwritten kannada characters," *Journal of Universal Computer Science*, vol. 17, no. 1, pp. 81–93, 2011.

[14] B. Graham, "Sparse arrays of signatures for online character recognition," *arXiv preprint arXiv:1308.0371*, 2013.

[15] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: a comprehensive survey," *IEEE T. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63–84, 2000.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2011, pp. 1135–1139.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014.

[19] P. J. Rousseeuw and M. Hubert, "Robust statistics for outlier detection," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 73–79, 2011.

[20] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller, "Turkit: Tools for iterative tasks on mechanical turk," in *Proc. of Workshop on Human Computation*, 2009, pp. 29–30.

[21] M. Kozielski, J. Forster, and H. Ney, "Moment-based image normalization for handwritten text recognition," in *Proc. Intl. Conf. on Frontiers in Handwriting Recognition (ICFHR)*, 2012, pp. 256–261.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift." in *International Conference on Machine Learning (ICML)*, vol. 37, 2015, pp. 448–456.

[24] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," *CoRR*, 2015.

[25] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[26] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 2951–2959.

[27] W. Mason and D. J. Watts, "Financial incentives and the performance of crowds," *ACM SigKDD Explorations Newsletter*, vol. 11, no. 2, pp. 100–108, 2010.

[28] L. A. Leiva, D. Martín-Albo, and R. Plamondon, "Gestures à go go: authoring synthetic human-like stroke gestures using the kinematic theory of rapid movements," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 7, no. 2, p. 15, 2016.

[29] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep learning is robust to massive label noise," *arXiv preprint arXiv:1705.10694*, 2017.