Applied Artificial Intelligence

An International Journal

ISSN: 0883-9514 (Print) 1087-6545 (Online) Journal homepage: https://www.tandfonline.com/loi/uaai20

Fuzzy String Matching with a Deep Neural Network

Daniel Shapiro, Nathalie Japkowicz, Mathieu Lemay & Miodrag Bolic

To cite this article: Daniel Shapiro, Nathalie Japkowicz, Mathieu Lemay & Miodrag Bolic (2018) Fuzzy String Matching with a Deep Neural Network, Applied Artificial Intelligence, 32:1, 1-12, DOI: 10.1080/08839514.2018.1448137

To link to this article: https://doi.org/10.1080/08839514.2018.1448137

Published online: 19 Mar 2018.

APPLIED ARTIFICIAL INTELLIGENCE
2018, VOL. 32, NO. 1, 1–12
https://doi.org/10.1080/08839514.2018.1448137

Fuzzy String Matching with a Deep Neural Network

Daniel Shapiro[a,b], Nathalie Japkowicz[a], Mathieu Lemay[b], and Miodrag Bolic[a]

[a]School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Ontario, Canada; [b]Lemay Solutions Consulting Inc, Ottawa, Ontario, Canada

ABSTRACT
A deep learning neural network for character-level text classification is described in this work. The system spots keywords in the text output of an optical character recognition system using memoization and by encoding the text into feature vectors related to letter frequency. Recognizing error messages in a set of generated images, dictionary and spell-check-based approaches achieved 69% to 88% accuracy, while various deep learning approaches achieved 91% to 96% accuracy, and a combination of deep learning with a dictionary achieved 97% accuracy. The contribution of this work to the state of the art is to describe a new approach for character-level deep neural network classification of noisy text.

Introduction

Fuzzy string matching is the process of classifying text that contains added noise in the form of spelling errors. This work is about learning to detect and correct mistakes in optical character recognition (OCR) output for certain keywords by examining character-level text features. The contribution of this work is to describe a new approach using a deep neural network (DNN) and memoization to classify based upon particular text features.

Consider a virtual agent that detects onscreen error messages. The input to the agent is the output from an OCR engine translating fullscreen computer screen images to text. OCR conversion from image to text often includes many misspellings, and so the system described in this work learns a mapping from incorrect OCR output words (e.g., nullpointer3xception) back to the correct words (e.g., nullpointerexception), improving error message recognition. The system is described below, in the section entitled ‘Fuzzy string generation process’. The three approaches evaluated in this work are spelling correction, dictionary, and deep learning. The classification accuracy results are discussed in the ‘Discussion’ section, followed by concluding remarks and thoughts on future work.

CONTACT Daniel Shapiro dcshapiro@gmail.com School of Electrical Engineering and Computer Science, University of Ottawa, 888 Broadview Ave, Ottawa, ON, K2A 2M5, Canada

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/UAAI.

Prior art

A recent survey on the general topic of text detection and extraction from images was conducted by Ye and Doermann (2015), and a survey on fuzzy string matching was conducted by Gomaa and Fahmy (2013). The fuzzy string matching literature enumerates many methods for measuring the similarity of two strings, including edit distance metrics and term-based metrics. In future work, a more comprehensive comparison between this work and these many methods will be carried out.

As described in Kasampalis (2015), memoization is an optimization technique used to avoid recomputing a result when the answer has already been computed. In this work, memoization was used as a dictionary for caching keyword spelling corrections. Similar to the general approach in this work, Silberpfennig et al. (2015) corrected a baseline OCR engine using it as a black box to process a set of unlabeled images. The OCR output text was then calculated as the centroid of a set of many candidate OCR words, measured by edit distance. Jaderberg, Vedaldi, and Zisserman (2014) and Zhang, Zhao, and LeCun (2015) processed noisy text using convolutional neural networks. Zhang, Zhao, and LeCun (2015) used memory units in a network nine layers deep with six convolutional layers and three fully connected layers, sending text character by character into the neural network (multiple vectors per word). In this work, three fully connected layers process one vector per word. Using word vectors produces faster training results and avoids the need to consider memory units. Bissacco et al. (2013) extracted text from smartphone images on a character by character basis. Character-level classification was achieved with a deep learning neural network with five hidden layers. The input layer consisted of histograms of oriented gradients coefficients and three geometry features, while the output layer was a softmax over 99 character classes plus a noise class. Just as this character classification system extracts features using a histogram with bins, the system in this work uses letter frequency encoding at the word level as a feature of the error message text. In our own testing of Bissacco et al. (2013) using 23 onscreen error messages, only 15 were correctly translated to text. This provides further motivation for the development of a specialized tool for error message detection.

Fuzzy string generation process

Consider the block diagram in Figure 1 for recording OCR inputs and outputs. An image generator (B) generates many images of various font settings (C) for one specific keyword (A). The system then submits the images for OCR processing (D), optionally performs corrective operations on the OCR output (E), and then compares the accuracy of the output with

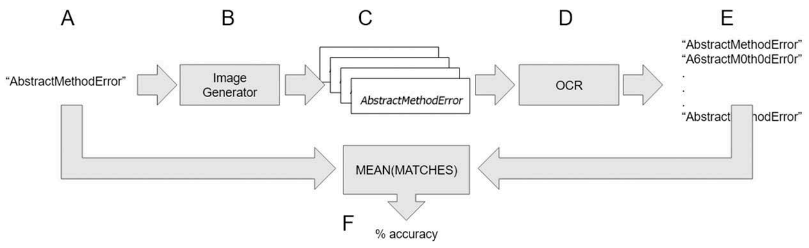

Figure 1. System for recording and correcting OCR mistakes.

the original text input (F). OCR at (D) uses the tesseract-OCR engine Smith (2007) to convert the image back into a string of text (E). OCR is treated as a black box (D), and the behavior of this black box is characterized through the recording of its input (A) and output (E). The image generation subroutine depicted in Figure 1 (B) takes in a keyword (e.g. AbstractMethodError) and outputs a 1024 × 786 grayscale Portable Network Graphics (PNG) image containing that keyword with a white background and black text. The system selects a random vertical and random horizontal position on the screen to display the word, ensuring that the word is not cut off. The word style is selected randomly as either bold, italic, or plain. The font size is selected randomly between 10 and 40 pt, and the font is selected randomly from a set of 18 common fonts.

Testing and training datasets

A Java Error List (JEL) was created to model onscreen error messages. JEL contained a set of 248 keywords such as "indexoutofboundsexception" cast to lowercase. Testing and training datasets were created using JEL according to the procedure described in 'the Fuzzy String Generation Process', above. For the TESTING dataset, 10 images were generated for each keyword in JEL, while for the TRAINING dataset 100 images per JEL keyword were created. TESTING contained a total of 2480 images and corresponding keywords, while TRAINING contained a total of 24,800 images and corresponding keywords. Each dataset contained the original text that each picture was based upon, as well as the text produced by the OCR process. TRAINING contained 5269 distinct OCR results associated to the 248 distinct keywords in JEL. There were therefore, on average, 21.2 variants of each keyword produced by the OCR output. Some common substrings in JEL include "illegal" (10 keywords), "error" (23 keywords), and "exception" (189 keywords). Note that 14 data points (0.56%) in TESTING had blank OCR text as a result of the OCR software detecting no text in the image. The theoretical maximum possible classification accuracy was therefore 99.44%. When the deep learning system described in 'Combining Deep Learning and

Memoization' below was both trained and tested using the TESTING dataset, the classification accuracy was in fact 99.35%.

A baseline measurement for the detection accuracy of keywords is to measure exact matches between OCR input and output text, and to calculate the resulting accuracy. The default OCR configuration had 61% accuracy processing the TESTING dataset. The tesseract-OCR software configuration can be tuned in various ways to improve OCR accuracy on non-dictionary text (Morris et al. 2016). Disabling the dictionaries did not improve the OCR accuracy (60%). Disabling OCR dictionaries was not an effective strategy for improving OCR accuracy when processing error message text from TESTING. These results leave significant room for other approaches to provide improvements in accuracy.

Spell-check approach

An initial OCR output correction system was developed for this work based upon the ideas in Bassil and Alwani (2012) to correct OCR output text using the Google search engine's built-in spell-check correction. The OCR output text was submitted to the search engine, and if the "Showing results for" field appeared in the results page, the term that the search engine expected replaced the OCR output text. This system correctly classified 1721 keywords out of 2480 in TESTING (69.40% accuracy). A second OCR output correction system was developed for this work to further evaluate the effectiveness of correcting spelling mistakes in OCR output text. The spell-check module (McCallum, 2014–2016) selects the best candidate correction from a variety of ranked options including known misspellings of words, dictionaries of words, word lists generated by parsing natural language documents, and word snippets. The ranking is based on word use frequency in the reference document, preferring more common terms over less common ones. The autocorrect dictionary was updated to include the corpus of keywords from TRAINING into its database. After this upgrade, the system correctly classified 1993 keywords out of 2480 (80.36% accuracy) when processing TESTING.

Memoization

A dictionary-based approach called memoization was used to first learn a set of OCR output correction rules based upon the TRAINING dataset, and then tested against the TESTING dataset. Using TRAINING, a dictionary D was trained to store outputs from the OCR process as keys, and text inputs to the image generation process as values. These key/value pairs map incorrect OCR outputs back to the correct output word. The benefit of this approach is that the learning algorithm executes very quickly, whereas the drawback of this approach is that it cannot generalize to identify new relationships that were not observed during the training phase. This approach learns which spelling mistakes the OCR module is prone to make, and stores them in relation to the correct spelling for use after the training phase.

The dictionary was trained using the TRAINING dataset, and then TESTING was processed. For TESTING, the system corrected the output of the OCR tool in cases where the OCR output matched a key in the dictionary. When there was a key match, the corresponding dictionary value was substituted for the OCR output. The resulting accuracy was 88% (296 fails and 2184 passes). This 88% accuracy is an improvement over the 61% baseline. This result indicates that most OCR output errors are repeatable as a given keyword is most likely to be misinterpreted in a particular way. Furthermore, it is clear that more than 1 in 10 results from the OCR module is generated by rare or unpredictable circumstances that a dictionary-based approach cannot hope to solve. For example, OCR errors can be generated when a characters is touching the edge of the image, or is encoded in an unusual font. Another interesting result was the presence of dictionary key collisions. Even with TESTING's small 248-keyword lexicon, there were several cases where two input keywords resulted in the same OCR results in the training dataset, causing a key conflict between two keywords. Both keywords expect to use the same key, but keys cannot be shared between keywords.

Combining deep learning and memoization

The three-layer deep learning neural network used in this work is a branch of the image processing code (Radford 2014), originally designed to recognize handwritten characters in the MNSIT dataset (LeCun, Cortes, and Burges 1998). Radford (2014) was modified for this work to accept text rather than image data, and the width of the output layer was expanded to 5000 neurons, where each output neuron encodes for a specific keyword. Radford (2014) included numerically stable softmax for the output layer, gradient scaling, improved training with dropout, improved training with noise injection, and more. The DNN input vector representations considered in this section are Naive ASCII encoding (Ascii), Binary ASCII encoding (BinAscii), Morse encoding (Morse), Morse with letter frequency encoding (MorseFreq), and Binary ASCII with letter frequency encoding (BinAsciiFreq). Finally, the memoization (dictionary) approach was combined with the vector encoding of BinAsciiFreq (BinAsciiFreqDict). Output vectors specified in TESTING and TRAINING are one-hot vectors. This gives the network the ability to map many millions of possible input vectors (representing OCR text) onto up to 5000 keywords. The input layer contains 784 neurons, and the hidden layer contains 625 neurons. Training of the network is accomplished over 150 epochs. During each epoch, the system is trained on TRAINING and then tested against TESTING. Many experiments in this work were performed at different levels of uniform random sampling from TRAINING: 10%, 20%, or 100%. Using less of the dataset in training helps the model to complete the training phase faster at the cost of predictive performance after the training. Using too much training causes the model overfit to the training data and then under perform when classifying the TESTING dataset.

The one-hot vectors in TRAINING and TESTING are the classes into which the neural network is trained to classify inputs. The images based upon these keywords are the inputs to the OCR. Because of the sparse nature of one-hot vectors, they are stored as integers signifying the integer index into the one-hot vector. Input vectors in TRAINING are created using the encoding of the OCR output. Data in the vector representation of a keyword are left aligned onto the input vector. Various alignment schemes for the input vectors were attempted with poor results, including random alignment and random amounts of 0-padding both sides of any keyword less than 784 elements long. The initial approach of left alignment of the data produced the best accuracy. Data less than 784 characters long are right-padded with 0s length 784, and data longer than 784 characters are truncated to length 784. Input vectors can be encoded to increase the number of array elements encoded by a character (increased number of neurons activated) and to increase the contrast of the input vectors during training (array elements contain only the values 0 or 255).

Naive ASCII encoding (Ascii)

This encoding involves assigning each element in an array of length 784 and height 1 with the ASCII value for a character. The encoding at each element maps the ASCII number of the corresponding character to the element in the array at the same index. The naive ASCII encoding scheme was formatted in a way that the deep learning neural network was not able to learn well (14% accuracy after training on a 10% sample of TRAINING).

Binary ASCII encoding (BinAscii*)*

To improve the perceptron processing of the information contained in the ASCII input to the neural network, the data in each ASCII character were converted into binary and then into elements. Figure 2 shows an example of this encoding scheme. For example, the letter “a” has ASCII code 55, which is “110111” in binary. This binary string is then converted into the following string: “255, 255, 0, 255, 255, 255.” These element strings were then concatenated into a vector representing the original text coming from the OCR module (Figure 3). The classification accuracy of binary ASCII encoding is reported in Table 1 and Figure 4 under the labels BinAscii10, BinAscii20, and

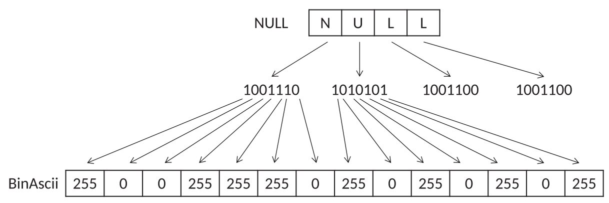

Figure 2. Conversion from text into input vectors using binary ASCII encoding.

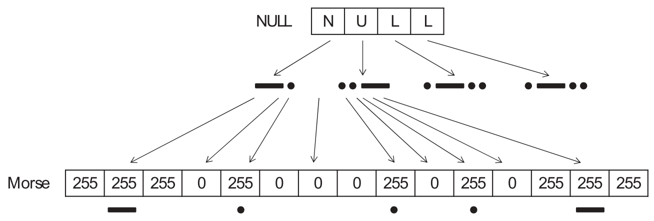

Figure 3. Conversion from text into input vectors using Morse encoding.

Table 1. Classification accuracy for deep learning neural network using various input vector encoding schemes and classifying the TESTING dataset.

Training effort	Classification accuracy	Learning graph label in Figure 4
None	61.09%	Baseline (not in Figure 4)
Training on 10% sample of TRAINING	88.26%	Morse10
Training on 20% sample of TRAINING	90.84%	Morse20
Training on 100% of TRAINING	88.91%	Morse100
Training on 10% sample of TRAINING	90.28%	BinAscii10
Training on 20% sample of TRAINING	92.01%	BinAscii20
Training on 100% of TRAINING	89.43%	BinAscii100
Training on 10% sample of TRAINING	95.97%	Morse2Freq10
Training on 20% sample of TRAINING	95.85%	Morse2Freq20
Training on 100% of TRAINING	94.03%	Morse2Freq100
Training on 10% sample of TRAINING	96.09%	BinAscii2Freq10
Training on 20% sample of TRAINING	95.97%	BinAscii2Freq20
Training on 100% of TRAINING	94.03%	BinAscii2Freq100
Training on 20% sample of TRAINING	96.29%	BinAscii10Freq20
Training on 20% sample of TRAINING	96.97%	BinAscii10Freq20Dict

BinAscii100. The binary ASCII encoding scheme was very good at training the deep learning neural network to classify OCR output text. The maximum classification accuracy achieved with binary ASCII encoding was 92%.

Morse encoding (morse)

The 784-element Morse input vectors are populated by converting characters from letters and numbers in the OCR input/output into the dots and dashes of the International Morse Code (Morse) (ITU 2009). Although Morse standards require special characters to be mapped to specific patterns, that component of the standard was not followed for this work. In Morse, a dot is

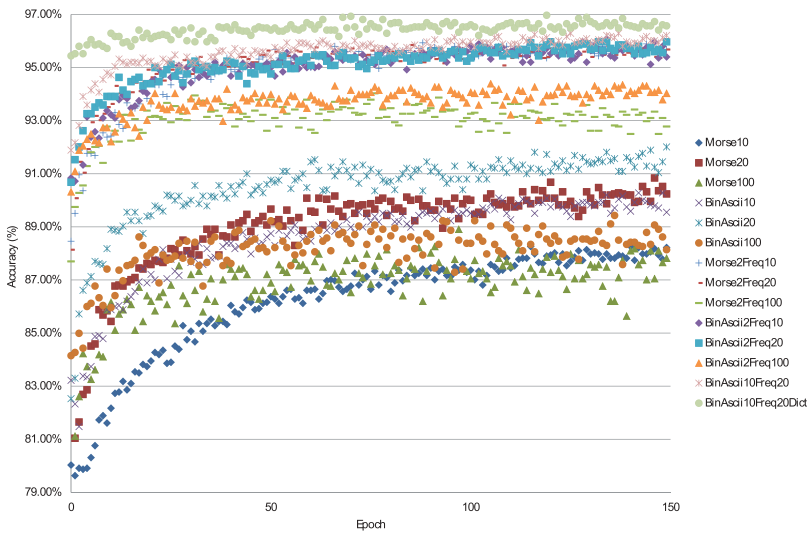

Figure 4. Classification accuracy (%) for various deep learning input vector encoding schemes at each training iteration (epoch) during the training phase.

occupied by one element with value 255 followed by a one element with the value 0, a dash occupies three elements with value 255 followed by one element with the value 0, the space between letters occupies three elements with value 0, and the space character is represented by seven consecutive elements with value 0. The dot and dash representation of each letter and number is specified in the Morse system, and this work maps special characters to existing Morse codes to retain the information they contain. For example, the double quote is treated as a U character, and the comma is treated as the number 4.

The classification accuracy of Morse encoding is reported in Table 1 and Figure 4 under the labels Morse10, Morse20, and Morse100. The Morse encoding scheme was very good at training the deep learning neural network to classify OCR output text. The maximum classification accuracy achieved with Morse encoding was 91%.

Morse with letter frequency encoding (MorseFreq*)*

Morse encoding of input vectors resulted in 91% classification accuracy. The 9% of TESTING that was incorrectly classified was analyzed to uncover the possible reasons for learning failures. It became clear from looking through the data that one major problem is frame shifts. Frame shifting in the OCR output text (e.g., the "M" character interpreted as two consecutive lowercase "l" characters) throws off the neural network by shifting all subsequent letters from their usual position. These shifts can occur multiple times in the same text. Two such shifts confuse the network too much to be resolved correctly.

One solution is to encode letter frequency into the final elements of the input vector to preserve the information contained in a keyword when characters are added and the information in the keyword is shifted. This encoding region gives the network hints about the keyword in a frame shift invariant way using letter frequency encoding. As shown in Algorithm 1, 52 elements are used in a 2-bin letter frequency vector. Only the letters "a" through "z" (26 characters) are represented in the encoding, and each letter is represented by two elements in the vector. The encoding is simple: for each letter, one input represents the presence of a character (e.g., "a" in the word "apple" is present and therefore "255"), while the adjacent input represents the detection of more than one of a letter being detected. And so, the two elements encoding the frequency of "a" in "apple" will contain "255," "0,", while the elements for "p" contain "255," "255," and the elements for the letter "z" will contain "0," "0."

Algorithm 1: Encoding letter frequency of OCR text output into a vector

```
Input: Array of 26 letters [a-z]: letter; text output from OCR module:
       ocrText;
       generator of 0-filled array: zeroes(length); parser of occurrences of
       letter in text: count(letter, text); Number of elements used to
       represent the frequency of each letter: numBuckets
Output: Array representation of letter frequency frequencyArr
frequencyArr = zeroes(26 * numBuckets)
for i in range(0, 26) do
    occurrences = count(letter[i],ocrT ext)
    for j in range(0, numBuckets) do
        if occurrences > j then
            frequencyArr[numBuckets * i + j] = 255
        end
    end
end
```

The classification accuracy of Morse with 2-bin frequency encoding is reported in Table 1 and Figure 4 under the labels Morse2Freq10, Morse2Freq20, and Morse2Freq100. The maximum classification accuracy achieved with Morse with 2-bin frequency encoding was 96%.

Binary ASCII with letter frequency encoding (BinAsciiFreq)

Binary ASCII character encoding of BinAscii was combined with letter frequency encoding of MorseFreq to produce another input vector encoding scheme.

The classification accuracy of binary ASCII with 2-bin frequency encoding is reported in Table 1 and Figure 4 under the labels BinAscii2Freq10, BinAscii2Freq20, and BinAscii2Freq100. The maximum classification accuracy achieved with binary ASCII with 2-bin frequency encoding was 96%. Results for binary ASCII with 10-bin frequency encoding are reported as BinAscii10Freq20, where the highest classification accuracy was also 96%.

Binary ASCII with letter frequency encoding and memoization

To further increase the classification accuracy, the dictionary approach was combined with the input vector encoding from BinAscii10Freq20. The dictionary was programmed to replace the output from the neural network when it identified an exact match, and otherwise to defer to the result from the neural network. This approach (BinAscii10Freq20Dict) resulted in 97% as the highest classification accuracy. More detailed results are presented in Figure 4 and Table 1.

This approach resulted in accurate identification of 2403 out of 2480 images in TESTING. The remaining 77 images from TESTING that failed to be correctly classified were: 13 images where the OCR output was blank, 48 images where key collisions occurred, and the remaining 16 images contained OCR output that TRAINING did not contain. One example of these rare OCR output mistakes is the OCR output for an image containing the keyword "cannotundoexception" where the OCR output was mammogram.

Discussion

A summary of the results for each OCR approach is presented in Figure 5. The results for the naive ASCII encoding were poor (14%), but clearly some learning did take place as the classification was better than chance. Without optimization, the baseline system was able to detect 61% of the onscreen keywords. Using spelling correction improved the accuracy (80%), picking

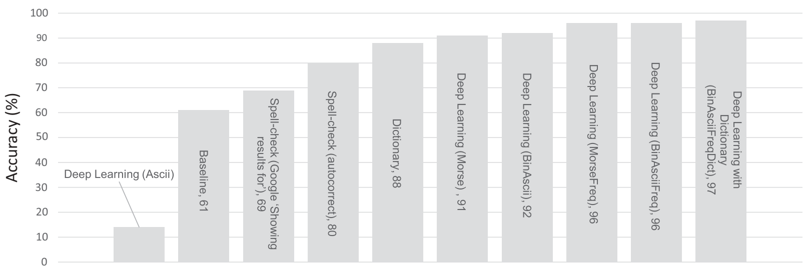

Figure 5. Classification accuracy on TESTING dataset for OCR approaches.

up gains in cases with single-letter substitutions and/or added spaces. Using memoization to remember common OCR errors further improved accuracy (88%). Results for the DNN trained to recognize binary ASCII and Morse encodings slightly outperformed the previously mentioned methods (91% and 92%), while including letter frequency in the feature vectors further improved accuracy (96%). Combining memoization with the most successful DNN approach brought accuracy up to 97%, close to the maximum possible accuracy of 99.44%.

The deep learning system appears to have generalized more than the strict memoization approach. OCR tuning, dictionary-based classification, and spelling correction provided lower accuracy than the deep learning approach. Deep learning has a long initial training phase which becomes a problem when retraining the whole network to remove (unlearn) a keyword. 5000 keywords in the one-hot output vector are initially available to be added to the network's lexicon without incurring a heavy training penalty. However, removing keywords required the network to be completely retrained. In the approaches without deep learning, removing keywords is comparatively computationally cheap.

Training on a 20% sample of the Morse data resulted in higher accuracy than using 100% of the data. The model is likely overfitting to the training data and then getting stuck on the novel testing data. This is a common problem with deep learning systems that is mostly solved by dropout, which was used in this work by Srivastava et al. (2014). A gradual increase in accuracy from 88% to 97% can be observed in Figure 4 as the input vector encoding scheme was tuned to maximize accuracy. Comparing ASCII and Morse input vector encoding schemes, the binary encoding of ASCII characters performed equally or slightly better than Morse encoding, even though the length of the data in Morse vectors is typically longer than ASCII vectors. Perhaps, these two encoding schemes express equivalent amounts of pattern information for the neural network to learn from.

Conclusion and future work

The systems discussed in this work learned to detect and correct mistakes in OCR output. A combination of deep learning and memoization achieved 97% classification accuracy. This work is a component in a virtual agent tasked with identifying onscreen error messages. The plan for future work is to compare this technique to many others, to apply this system beyond the domain of computer error messages, and to include unsupervised learning and context awareness. Sets of keywords such as gene names, case law identifiers, and other domain-specific identifiers that are not processed by OCR with high accuracy could be applied to this system in order to detect their presence onscreen and generate insights for the computer user.

ORCID

Miodrag Bolic http://orcid.org/0000-0002-8013-8645

References

Bassil, Y., and M. Alwani. 2012. OCR post-processing error correction algorithm using google online spelling suggestion. *arXiv Preprint* arXiv:1204.0191.

Bissacco, A., M. Cummins, Y. Netzer, and H. Neven. 2013. PhotoOCR: Reading text in uncontrolled conditions. In Proceedings of the IEEE International Conference on Computer Vision, Sydney, Australia. pp. 785–92.

Gomaa, W. H., and A. A. Fahmy. 2013. A survey of text similarity approaches. *International Journal of Computer Applications* 68 (13). doi:10.5120/11638-7118.

International Telecommunication Union (ITU). (2009). International Morse code. ITU-R M.1677-1. Accessed October, 2009. https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1677-1-200910-I!!PDF-E.pdf.

Jaderberg, M., A. Vedaldi, and A. Zisserman. 2014. *Deep Features for Text Spotting*, 512–28. Cham: Springer International Publishing.

Kasampalis, S. 2015. *Mastering Python design patterns*. Packt Publishing Ltd, Birmingham, UK.

LeCun, Y., C. Cortes, and C. J. Burges. 1998. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

McCallum, J. 2014–2016. Autocorrect: 0.2.0. https://pypi.python.org/pypi/autocorrect/0.2.0.

Morris, T., A. Dovev, S. Weil, and Zdenop. 2016. ImproveQuality tesseract-ocr/tesseract Wiki GitHub. Accessed June 07, 2016. https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality.

Radford, A. 2014. Introduction to deep learning with python. http://www.slideshare.net/indicods/deep-learning-with-python-and-the-theano-library.

Silberpfennig, A., L. Wolf, N. Dershowitz, S. Bhagesh, and B. B. Chaudhuri. 2015, August. Improving OCR for an under-resourced script using unsupervised word-spotting. In Document Analysis and Recognition (ICDAR), 2015 13th International Conference on, Nancy, France. pp. 706–10.

Smith, R. 2007. An overview of the tesseract OCR engine. 2013 12th International Conference on Document Analysis and Recognition 2, Washington D.C., USA 629–33.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15 (1):1929–58.

Ye, Q., and D. Doermann. 2015. Text detection and recognition in imagery: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (7, July):1480–500. doi:10.1109/TPAMI.2014.2366765.

Zhang, X., J. Zhao, and Y. LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems 28*, Eds. C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, 649–57. Curran Associates, Inc., Montreal, Canada.