International Journal on Document Analysis and Recognition (IJDAR) (2020) 23:89–102
https://doi.org/10.1007/s10032-020-00350-4

ORIGINAL PAPER

Fast multi-language LSTM-based online handwriting recognition

Victor Carbune[1] · Pedro Gonnet[1] · Thomas Deselaers[1] · Henry A. Rowley[2] · Alexander Daryin[1] · Marcos Calvo[1] · Li-Lun Wang[2] · Daniel Keysers[1] · Sandro Feuz[1] · Philippe Gervais[1]

Received: 8 August 2019 / Revised: 20 December 2019 / Accepted: 24 January 2020 / Published online: 8 February 2020

Abstract

We describe an online handwriting system that is able to support 102 languages using a deep neural network architecture. This new system has completely replaced our previous segment-and-decode-based system and reduced the error rate by 20–40% relative for most languages. Further, we report new state-of-the-art results on IAM-OnDB for both the open and closed dataset setting. The system combines methods from sequence recognition with a new input encoding using Bézier curves. This leads to up to 10× faster recognition times compared to our previous system. Through a series of experiments, we determine the optimal configuration of our models and report the results of our setup on a number of additional public datasets.

1 Introduction

In this paper, we discuss *online* handwriting recognition: Given a user input in the form of an *ink*, i.e., a list of touch or pen *strokes*, output the textual interpretation of this input. A stroke is a sequence of *points* (x, y, t) with position (x, y) and timestamp t.

✉ Victor Carbune
vcarbune@google.com

Pedro Gonnet
gonnet@google.com

Thomas Deselaers
deselaers@google.com

Henry A. Rowley
har@google.com

Alexander Daryin
shurick@google.com

Marcos Calvo
marcoscalvo@google.com

Li-Lun Wang
llwang@google.com

Daniel Keysers
keysers@google.com

Sandro Feuz
sfeuz@google.com

Philippe Gervais
pgervais@google.com

[1] Google, Zurich, Switzerland

[2] Mountain View, CA, USA

Figure 1 illustrates example inputs to our online handwriting recognition system in different languages and scripts. The left column shows examples in English with different writing styles, with different types of content, and that may be written on one or multiple lines. The center column shows examples from five different alphabetic languages similar in structure to English: German, Russian, Vietnamese, Greek, and Georgian. The right column shows scripts that are significantly different from English: Chinese has a much larger set of more complex characters, and users often overlap characters with one another. Korean, while an alphabetic language, groups letters in syllables leading to a large "alphabet" of syllables. Hindi writing often contains a connecting "Shirorekha" line, and characters can form larger structures (grapheme clusters) which influence the written shape of the components. Arabic is written right-to-left (with embedded left-to-right sequences used for numbers or English names), and characters change shape depending on their position within a word. Emoji are non-text Unicode symbols that we also recognize.

Online handwriting recognition has recently been gaining importance for multiple reasons: (a) An increasing number of people in emerging markets are obtaining access to computing devices, many exclusively using mobile devices with touchscreens. Many of these users have native languages and scripts that are not as easily typed as English, e.g., due to the size of the alphabet or the use of grapheme clusters which makes it difficult to design an intuitive keyboard layout [10]. (b) More and more large mobile devices with styluses are

Fig. 1 Example inputs for online handwriting recognition in different languages. See text for details

becoming available, such as the iPad Pro,[1] Microsoft Surface devices,[2] and Chromebooks with styluses.[3]

Early work in online handwriting recognition looked at segment-and-decode classifiers, such as the Newton [48]. Another line of work [38] focused on solving online handwriting recognition by making use of hidden Markov models (HMMs) [20] or hybrid approaches combining HMMs and Feed-forward Neural Networks [2]. The first HMM-free models were based on time delay neural networks (TDNNs) [5,22,37], and more recent work focuses on recurrent neural network (RNN) variants such as long short-term memory networks (LSTMs) [6,7,14].

How to represent online handwriting data has been a research topic for a long time. Early approaches were feature-based, where each point is represented using a set of features [22,23,48], or using global features to represent entire characters [22]. More recently, the deep learning revolution has swept away most feature engineering efforts and replaced them with learned representations in many domains, e.g., speech [17], computer vision [44], and natural language processing [33].

Together with architecture changes, training methodologies also changed, moving from relying on explicit segmentation [25,37,48] to implicit segmentation using the connectionist temporal classification (CTC) loss [12], or encoder–decoder approaches trained with Maximum Likelihood Estimation [51]. Further recent work is also described in [26].

The transition to more complex network architectures and end-to-end training can be associated with breakthroughs in related fields focused on sequence understanding where deep learning methods have outperformed "traditional" pattern recognition methods, e.g., in speech recognition [40,41], OCR [8,47], offline handwriting recognition [16], and computer vision [45].

In this paper, we describe our new online handwriting recognition system based on deep learning methods. It replaces our previous segment-and-decode system [25], which first over-segments the ink, then groups the segments into character hypotheses, and computes features for each character hypothesis which are then classified as characters using a rather shallow neural network. The recognition result is then obtained using a best path search decoding algorithm on the lattice of hypotheses incorporating additional knowledge sources such as language models. This system relies on numerous preprocessing, segmentation, and feature extraction heuristics which are no longer present in our new system. The new system reduces the amount of customization required, and consists of a simple stack of bidirectional LSTMs (BLSTMs), a single Logits layer, and the CTC loss [15] (Sect. 2). We train a separate model for each *script* (Sect. 3). To support potentially many languages per script (see Table 1), language-specific language models and feature functions are used during decoding (Sect. 2.5). For example, we have a single recognition model for Arabic script which is combined with specific language models and feature functions for our Arabic, Persian, and Urdu language recognizers. Table 1 shows the full list of scripts and languages that we currently support.

The new models are more accurate (Sect. 4), smaller, and faster (Table 2) than our previous segment-and-decode models and eliminate the need for a large number of engineered features and heuristics.

We present an extensive comparison of the differences in recognition accuracy for eight languages (Sect. 5) and compare the accuracy of models trained on publicly available datasets where available (Sect. 4). In addition, we propose a new standard experimental protocol for the IBM-UB-1 dataset [43] to enable easier comparison between approaches in the future (Sect. 4.2).

The main contributions of our paper are as follows:

- We describe in detail our recurrent neural network-based recognition stack and provide a description of how we tuned the model. We also provide a detailed experimental comparison with the previous segment-and-decode-based stack [25] on the supported languages.
- We describe a novel input representation based on Bézier curve interpolation, which produces shorter input sequences, which results in faster recognitions.
- Our system achieves a new state of the art on the IAM-OnDB dataset, both for open and closed training sets.
- We introduce an evaluation protocol for the less commonly used English IBM-UB-1 query dataset. We provide experimental results that quantify the structural

[1] https://www.apple.com/ipad-pro/.

[2] https://www.microsoft.com/en-us/store/b/surface.

[3] https://store.google.com/product/google_pixelbook.

Table 1 List of languages supported in our system grouped by script

Script	Languages
Latin	Afrikaans, Azerbaijani, Bosnian, Catalan, Cebuano, Corsican, Czech, Welsh, Danish, German, English, Esperanto, Spanish, Estonian, Basque, Finnish, Filipino, French, Western Frisian, Irish, Scottish Gaelic, Galician, Hawaiian, Hmong, Croatian, Haitian Creole, Hungarian, Indonesian, Icelandic, Italian, Javanese, Kurdish, Latin, Luxembourgish, Lao, Lithuanian, Latvian, Malagasy, Maori, Malay, Maltese, Norwegian, Dutch, Nyanja, Polish, Portuguese, Romanian, Slovak, Slovenian, Samoan, Shona, Somali, Albanian, Sundanese, Swedish, Swahili, Turkish, Xhosa, Zulu
Cyrillic	Russian, Belarusian, Bulgarian, Kazakh, Mongolian, Serbian, Ukrainian, Uzbek, Macedonian, Kyrgyz, Tajik
Chinese	Simplified Chinese, Traditional Chinese, Cantonese
Arabic	Arabic, Persian, Urdu
Devanagari	Hindi, Marathi, Nepali
Bengali	Bangla, Assamese
Ethiopic	Amharic, Tigrinya
Languages with distinct scripts: Armenian, Burmese, Georgian, Greek, Gujarati, Hebrew, Japanese, Kannada, Khmer, Korean, Lao, Malayalam, Odia, Punjabi, Sinhala, Tamil, Telugu, Thai, Tibetan, Vietnamese[a]	

[a]While Vietnamese is a Latin-script language, we have a dedicated model for it because of the large amount of diacritics not used in other Latin-script languages

Table 2 Character error rates on the validation data using successively more of the system components described above for English (en), Spanish (es), German (de), Arabic (ar), Korean (ko), Thai (th), Hindi (hi), and Chinese (zh) along with the respective number of items and characters in the test and training datasets. Average latencies for all languages and models were computed on an Intel Xeon E5-2690 CPU running at 2.6 GHz

Language	en	es	de	ar	ko	th	hi	zh
Internal test data (per language)								
Items	32,645	7136	14,408	11,617	22,951	23,608	9030	197,547
Characters	162,367	40,302	83,231	84,017	55,654	109,793	36,726	312,478
Internal training data (per script)								
Items		3,293,421		570,375	3,495,877	207,833	1,004,814	5,969,179
Characters		15,850,724		4,597,255	4,770,486	989,520	5,575,552	7,548,434
Unique supported characters		295		337	3524	195	197	12,726
System					CER (%)			
Segment-and-decode [25]	7.5	7.2	6.0	14.8	13.8	4.1	15.7	3.76
BLSTM (comparison) [25]	10.2	12.4	9.5	18.2	44.2	3.9	15.4	–
Model architecture (this work)		5 × 224		5 × 160	5 × 160	5 × 128	5 × 192	4 × 192
(2) BLSTM-CTC baseline curves	8.00	6.38	7.12	12.29	6.87	2.41	7.65	1.54
(3) + n-gram LM	6.54	4.64	5.43	8.10	6.90	1.82	7.00	1.38
(4) + character classes	6.60	4.59	5.36	7.93	6.79	1.78	7.32	1.39
(5) + word LM	6.48	4.56	5.40	7.87	–	–	7.42	–
Avg. latency per item					(ms)			
Segment-and-decode [25]	315	359	372	221	389	165	139	208
This work	23	25	26	14	20	13	19	30
Number of parameters (per script)								
Segment-and-decode [25]		5,281,061		5,342,561	8,381,686	6,318,361	9,721,361	–
This work		5,386,170		2,776,937	3,746,999	1,769,668	3,927,736	7,729,994

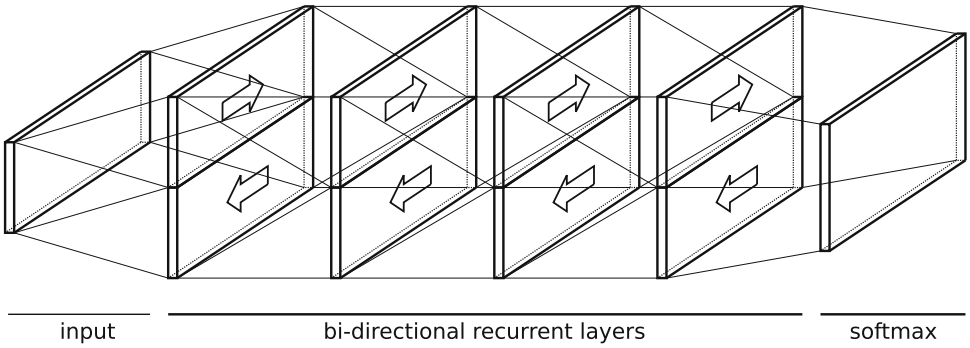

Fig. 2 An overview our recognition models. In our architecture, the input representation is passed through one or more bidirectional LSTM layers, and a final softmax layer makes a classification decision for the output at each time step

difference between IBM-UB-1, IAM-OnDB, and our internal dataset.
– We perform ablation studies and report results on numerous experiments highlighting the contributions of the individual components of the new recognition stack on our internal datasets.

2 End-to-end model architecture

Our handwriting recognition model draws its inspiration from research aimed at building end-to-end transcription models in the context of handwriting recognition [15], optical character recognition [8], and acoustic modeling in speech recognition [40]. The model architecture is constructed from common neural network blocks, i.e., bidirectional LSTMs and fully connected layers (Fig. 2). It is trained in an end-to-end manner using the CTC loss [15].

Our architecture is similar to what is often used in the context of acoustic modeling for speech recognition [41], in which it is referred to as a CLDNN (Convolutions, LSTMs, and DNNs), yet we differ from it in four points. Firstly, we do not use convolution layers, which in our own experience do not add value for large networks trained on large datasets of relatively short (compared to speech input) sequences typically seen in handwriting recognition. Secondly, we use *bidirectional* LSTMs, which due to latency constraints is not feasible in speech recognition systems. Thirdly, our architecture does not make use of additional fully connected layers before and after the bidirectional LSTM layers. And finally, we train our system using the CTC loss, as opposed to the HMMs used in [41].

This structure makes many components of our previous system [25] unnecessary, e.g., feature extraction and segmentation. The heuristics that were hard-coded into our previous system, e.g., stroke-reordering and character hypothesis building, are now implicitly learned from the training data.

The model takes as input a time series $(v_1, \ldots, v_T)$ of length T encoding the user input (Sect. 2.1) and passes it through several bidirectional LSTM layers [42] which learn the structure of characters (Sect. 2.2).

The output of the final LSTM layer is passed through a softmax layer (Sect. 2.3) leading to a sequence of probability distributions over characters for each time step.

For CTC decoding (Sect. 3.1), we use beam search to combine the softmax outputs with character-based language models, word-based language models, and information about language-specific characters as in our previous system [25].

2.1 Input representation

In our earlier paper [25], we presented results on our datasets with a model similar to the one proposed in [15]. In that model, we used 23 per-point features (similar to [22]) as described in our segment-and-decode system to represent the input. In further experimentation, we found that in substantially deeper and wider models, engineered features are unnecessary and their removal leads to better results. This confirms the observation that learned representations often outperform handcrafted features in scenarios in which sufficient training data are available, e.g., in computer vision [28] and in speech recognition [46]. In the experiments presented here, we use two representations:

2.1.1 Raw touch points

The simplest representation of stroke data is as a sequence of touch points. In our current system, we use a sequence of 5-dimensional points $(x_i, y_i, t_i, p_i, n_i)$ where (x_i, y_i) are the coordinates of the ith touchpoint, t_i is the timestamp of the touchpoint since the first touch point in the current observation in seconds, p_i indicates whether the point corresponds to a pen-up ($p_i = 0$) or pen-down ($p_i = 1$) stroke, and $n_i = 1$ indicates the start of a new stroke ($n_i = 0$ otherwise).[4]

In order to keep the system as flexible as possible with respect to differences in the writing surface, e.g., area shape,

[4] We acknowledge a redundancy between the features p_i and n_i which evolved over time from experimenting with explicit pressure data. We did not perform additional experiments to avoid this redundancy at this time but do not expect a large change in results when dropping either of these features.

size, spatial resolution, and sampling rate, we perform some minimal preprocessing:

- Normalization of x_i and y_i coordinates, by shifting in x such that $x_0 = 0$, and shifting and scaling the writing area isometrically such that the y coordinate spans the range between 0 and 1. In cases where the bounding box of the writing area is unknown, we use a surrogate area 20% larger than the observed range of touch points.
- Equidistant linear resampling along the strokes with $\delta = 0.05$, i.e., a line of length 1 will have 20 points.

We do not assume that words are written on a fixed baseline or that the input is horizontal. As in [15], we use the differences between consecutive points for the (x, y) coordinates and the time t such that our input sequence is $(x_i - x_{i-1}, y_i - y_{i-1}, t_i - t_{i-1}, p_i, n_i)$ for $i > 0$, and $(0, 0, 0, p_0, n_0)$ for $i = 0$.

2.1.2 Bézier curves

However simple, the raw input data have some drawbacks, i.e.,

- Resolution: Not all input devices sample inputs at the same rate, resulting in different point densities along the input strokes, requiring resampling which may inadvertently normalize-out details in the input.
- Length: We choose the (re-)sampling rate such as to represent the smallest features well, which leads to oversampling in less interesting parts of the stroke, e.g., in straight lines.
- Model complexity: The model has to learn to map small consecutive steps to larger global features.

Bézier curves are a natural way to describe trajectories in space, and have been used to represent online handwriting data in the past, yet mostly as a means of removing outliers in the input data [21], up-sampling sparse data [22], or for rendering handwriting data smoothly on a screen [35]. Since a sequence of Bézier curves can represent a potentially long point sequence compactly, irrespective of the original sampling rate, we propose to represent a sequence of input points as a sequence of parametric cubic polynomials, and to use these as inputs to the recognition model.

These Bézier curves for x, y, and t are cubic polynomials in $s \in [0, 1]$:

$$\begin{aligned} x(s) &= \alpha_0 + \alpha_1 s + \alpha_2 s^2 + \alpha_3 s^3 \\ y(s) &= \beta_0 + \beta_1 s + \beta_2 s^2 + \beta_3 s^3 \\ t(s) &= \gamma_0 + \gamma_1 s + \gamma_2 s^2 + \gamma_3 s^3 \end{aligned} \tag{1}$$

We start by normalizing the size of the entire ink such that the y values are within the range $[0, 1]$, similar to how we process it for raw points. The time values are scaled linearly to match the length of the ink such that

$$t_{N-1} - t_0 = \sum_{i=1}^{N-1} \left[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \right]^{1/2}. \tag{2}$$

in order to obtain values in the same numerical range as x and y. This sets the time difference between the first and last points of the stroke to be equal to the total spatial length of the stroke.

For each stroke in an ink, the coefficients α, β, and γ are computed by minimizing the sum of squared errors (SSE) between each observed point i and its corresponding closest point (defined by s_i) on the Bézier curve:

$$\sum_{i=0}^{N-1} (x_i - x(s_i))^2 + (y_i - y(s_i))^2 + (t_i - t(s_i))^2 . \tag{3}$$

where N is the number of points in the stroke. Given a set of coordinates s_i, computing the coefficients corresponds to solving the following linear system of equations:

$$\underbrace{\begin{bmatrix} x_1 & y_1 & t_1 \\ x_2 & y_2 & t_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & t_N \end{bmatrix}}_{Z} = \underbrace{\begin{bmatrix} 1 & s_1 & s_1^2 & s_1^3 \\ 1 & s_2 & s_2^2 & s_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & s_N & s_N^2 & s_N^3 \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} \alpha_0 & \beta_0 & \gamma_0 \\ \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \\ \alpha_3 & \beta_3 & \gamma_3 \end{bmatrix}}_{\Omega} \tag{4}$$

which can be solved exactly for $N \leq 4$, and in the least-squares sense otherwise, e.g., by solving the normalized equations

$$V^{\mathsf{T}} Z = V^{\mathsf{T}} V \Omega. \tag{5}$$

for the coefficients Ω. We alternate between minimizing the SSE in Eq. (3) and finding the corresponding points s_i, until convergence. The coordinates s_i are updated using a Newton step on

$$x'(s_i)(x_i - x(s_i)) + y'(s_i)(y_i - y(s_i)) = 0, \tag{6}$$

which is zero when $(x_i - x(s_i), y_i - y(s_i))$ is orthogonal to the direction of the curve $(x'(s_i), y'(s_i))$.

If (a) the curve cannot fit the points well (SSE error is too large) or if (b) the curve has too sharp bends (arc length longer than 3 times the endpoint distance), we split the curve into two parts. We determine the split point in case (a) by finding the triplet of consecutive points with the smallest

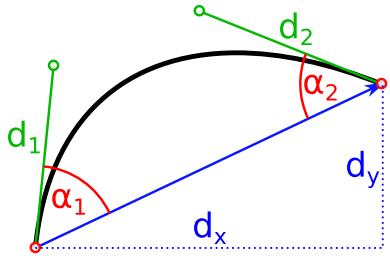

Fig. 3 Parameterization of each Bézier curve used to feed the network. Namely: vector between the endpoints (blue), distance between the control points and the endpoints (green dashed lines, 2 values), and the two angles between each control point and the endpoints (green arcs, 2 values) (color figure online)

angle, and in case (b) as the point closest to the maximum local curvature along the entire Bézier curve. This heuristic is applied recursively until both the curve matching criteria are met.

As a final step, to remove spurious breakpoints, consecutive curves that can be represented by a single curve are stitched back together, resulting in a compact set of Bézier curves representing the data within the above constraints. For each consecutive pair of curves, we try to fit a single curve using the combined set of underlying points. If the fit agrees with the above criteria, we replace the two curves by the new one. This is applied repeatedly until no merging happens anymore.

Since the Bézier coefficients α, β, and γ may vary significantly in range, each curve is fed to the network as a 10-dimensional vector $(d_x, d_y, d_1, d_2, \alpha_1, \alpha_2, \gamma_1, \gamma_2, \gamma_3, p)$, with:

- d_x, d_y: the vector between the endpoints (cp. Fig. 3)
- d_1, d_2: the distance between the control points and the endpoints relative to the distance between the endpoints (cp. Fig. 3),
- α_1, α_2: the angles between control points and endpoints in radians (cp. Fig. 3),
- γ_1, γ_2 and γ_3: the time coefficients from Eq. 1,
- p: a Boolean value indicating whether this is a pen-up or pen-down curve.

Due to the normalization of the x, y, and t coordinates, as well as the constraints on the curves themselves, most of the resulting values are in the range $[-1, 1]$.

The resulting sequences of 10-dimensional curve representations are roughly $4\times$ shorter than the corresponding 5-dimensional raw representation (Sect.2.1.1) because each Bezier curve represents multiple points. This leads to faster recognition and thus better latency.

In most of the cases, as highlighted through the experimental sections in this paper, the curve representations do not have a big impact on accuracy but contribute to faster speed of our models.

2.2 Bidirectional long short-term memory recurrent neural networks

LSTMs [19] have become one of the most commonly used RNN cells because they are easy to train and give good results [24]. In all experiments, we use bidirectional LSTMs [6,12], i.e., we process the input sequence forward and backward and merge the output states of each layer before feeding them to the next layer. The exact number of layers and nodes is determined empirically for each script. We give an overview of the impact of the number of nodes and layers in Sect. 4. We also list the configurations for several scripts in our production system, as of this writing.

2.3 Softmax layer

The output of the LSTM layers at each timestep is fed into a softmax layer to get a probability distribution over the C possible characters in the script (including spaces, punctuation marks, numbers or other special characters), plus the blank label required by the CTC loss and decoder.

2.4 Decoding

The output of the softmax layer is a sequence of T time steps of $(C + 1)$ classes that we decode using CTC decoding [12]. The logits from the softmax layer are combined with language-specific prior knowledge (cp. Sect. 2.5). For each of these additional knowledge sources, we learn a weight (called "decoder weight" in the following) and combine them linearly (cp. Sect. 3). The learned combination is used as described in [13] to guide the beam search during decoding. [5]

This combination of different knowledge sources allows us to train one recognition model per script (e.g., Latin script, or Cyrillic script) and then use it to serve multiple languages (see Table 1).

2.5 Feature functions: language models and character classes

Similar to our previous work [25], we define several scoring functions, which we refer to as *feature functions*. The goal of these feature functions is to introduce prior knowledge about the underlying language into the system. The introduction of

[5] We implement this as a `BaseBeamScorer` (https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/util/ctc/ctc_beam_scorer.h) which is passed to the `CTCBeamSearchDecoder` implementation in TensorFlow [1].

recurrent neural networks has reduced the need for many of them, and we now use only the following three:

- Character language models: For each language we support, we build a 7-gram language model over Unicode codepoints from a large web-mined text corpus using Stupid back-off [3]. The final files are pruned to 10 million 7-grams each. Compared to our previous system [25], we found that language model size has a smaller impact on the recognition accuracy, which is likely due to the capability of recurrent neural networks to capture dependencies between consecutive characters. We therefore use smaller language models over shorter contexts.
- Word language models: For languages using spaces to separate words, we also use a word-based language model trained on a similar corpus as the character language models [4,39], using 3-grams pruned to between 1.25 million and 1.5 million entries.
- Character classes: We add a scoring heuristic which boosts the score of characters from the language's alphabet. This feature function provides a strong signal for rare characters that may not be recognized confidently by the LSTM, and which the other language models might not weigh heavily enough to be recognized. This feature function was inspired by our previous system [25].

In Sect. 4, we provide an experimental evaluation of how much each of these feature functions contributes to the final result for several languages.

3 Training

The training of our system happens in two stages, on two different datasets:

1. End-to-end training of the neural network model using the CTC loss using a large training dataset
2. Tuning of the decoder weights using Bayesian optimization through Gaussian Processes in Vizier [11], using a much smaller and distinct dataset.

Using separate datasets is important because the neural network learns the local appearance as well as an implicit language model from the training data. It will be overconfident on its training data, and thus, learning the decoder weights on the same dataset could result in weights biased toward the neural network model.

3.1 Connectionist temporal classification loss

As our training data do not contain frame-aligned labels, we rely on the CTC loss [12] for training which treats the alignment between inputs and labels as a hidden variable. CTC training introduces an additional blank label which is used internally for learning alignments jointly with character hypotheses, as described in [12].

We train all neural network weights jointly using the standard TensorFlow [1] implementation of CTC training using the Adam Optimizer [27] with a batch size of 8, a learning rate of 10^{-4}, and gradient clipping such that the gradient L_2-norm is ≤ 9. Additionally, to improve the robustness of our models and prevent overfitting, we train our models using random dropout [18,36] after each LSTM layer with a dropout rate of 0.5. We train until the error rate on the evaluation dataset no longer improves for 5 million steps.

3.2 Bayesian optimization for tuning decoder weights

To optimize the decoder weights, we rely on the Google Vizier service and its default algorithm, specifically batched Gaussian process bandits, and expected improvement as the acquisition function [11].

For each recognizer training, we start 7 Vizier studies, each performing 500 individual trials, and then we pick the configuration that performed best across all of these trials. We experimentally found that using 7 separate studies with different random initializations regularly leads to better results than running a single study once. We found that using more than 500 trials per study does not lead to any additional improvement.

For each script, we train these weights on a subset of the languages for which we have sufficient data, and transfer the weights to all the other languages. For example, for the Latin-script languages, we train the decoder weights on English and German, and use the resulting weights for all languages in the first row of Table 1.

4 Experimental evaluation

In the following, where possible, we present results for public datasets in a *closed data* scenario, i.e., training and testing models on the public dataset using a standard protocol. In addition we present evaluation results for public datasets in an *open data* scenario against our production setup, i.e., in which the model is trained on our own data. Finally, we show experimental results for some of the major languages on our internal datasets. Whenever possible we compare these results to the state of the art and to our previous system [25].

4.1 IAM-OnDB

The IAM-OnDB dataset [31] is probably the most used evaluation dataset for online handwriting recognition. It consists

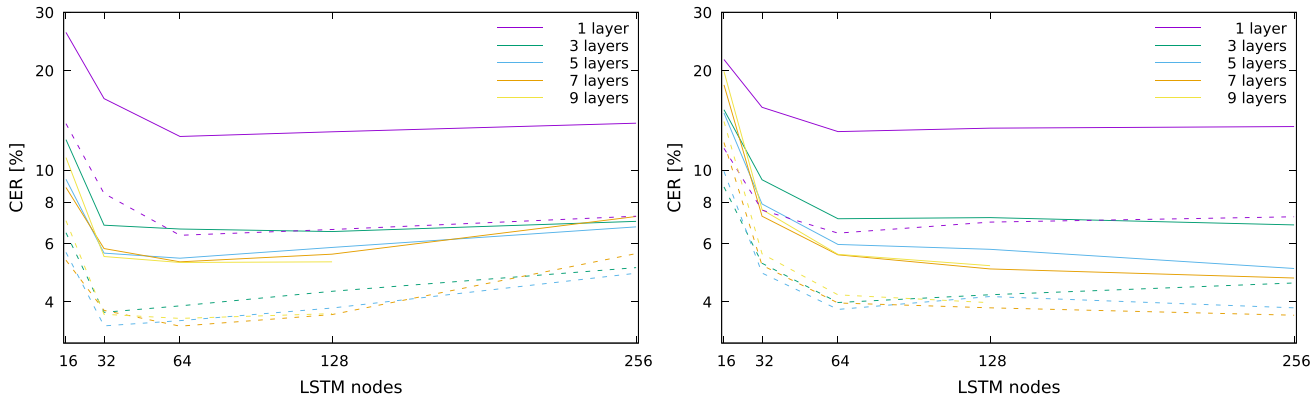

Fig. 4 CER of models trained on the IAM-OnDB dataset with different numbers of LSTM layers and LSTM nodes using raw (left) and curve (right) inputs. Solid lines indicate results without any language models or feature functions in decoding, and dashed lines indicate results with the fully tuned system

Table 3 Comparison of character error rates (lower is better) on the IAM-OnDB test set for different LSTM layers configurations

Input	lstm	64 Nodes	128 Nodes	256 Nodes
Raw	1 Layer	6.1	5.95	5.56
	3 Layers	**4.03**	4.73	4.34
	5 Layers	4.34	**4.20**	**4.17**
Curves	1 Layer	6.57	6.38	6.98
	3 Layers	4.16	**4.16**	4.83
	5 Layers	**4.02**	4.22	**4.11**

For each LSTM width and input type, we show the best result in bold

Table 4 Error rates on the IAM-OnDB test set in comparison with the state of the art and our previous system [25]

System	CER (%)	WER (%)
Frinken et al. BLSTM [7]	12.3	25.0
Graves et al. BLSTM [15]	11.5	20.3
Liwicki et al. LSTM [32]	–	18.9
This work (curve, 5x64, no FF)	5.9	18.6
This work (curve, 5x64, FF)	**4.0**	**10.6**
Our previous BLSTM [25]*	8.8	26.7
Combination [32]*	-	13.8
Our segment-and-decode [25]*	4.3	10.4
This work (production system)*	**2.5**	**6.5**

A "*" in the "system" column indicates the use of an open training set. "FF" stands for "feature functions" as described in Sect. 2.4

of 298,523 characters in 86,272 word instances from a dictionary of 11,059 words written by 221 writers. We use the standard IAM-OnDB dataset separation: one training set, two validations sets and a test set containing 5363, 1438, 1518 and 3859 written lines, respectively. We tune the decoder weights using the validation set with 1438 items and report error rates on the test set.

We perform a more extensive study of the number of layers and nodes per layer for both the raw and curve input formats to determine the optimal size of the bidirectional LSTM network (see Fig. 4, Table 3). We first run experiments without additional feature functions (Fig. 4, solid lines), then re-compute the results with tuned weights for language models and character classes (Fig. 4, dashed lines). We observe that for both input formats, using 3 or 5 layers outperforms more shallow networks, and using more layers gives hardly any improvement. Furthermore, using 64 nodes per layer is sufficient, as wider networks give only small improvements, if at all. We see no significant difference in the accuracy between the raw and the curve representation.

Finally, we show a comparison of our old and new systems with the literature on the IAM-OnDB dataset in Table 4. Our method establishes a new state-of-the-art result when relying on closed data using IAM-OnDB, as well as when relying on our in-house data that we use for our production system, which was not tuned for the IAM-OnDB data and for which none of the IAM-OnDB data were used for training.

To better understand where the improvements come from, we discuss the differences between the previous state-of-the-art system (Graves et al. BLSTM [15]) and this work across four dimensions: input preprocessing and feature extraction, neural network architecture, CTC training and decoding, and model training methodology.

Our input preprocessing (Sect. 2.1) differs only in minor ways: The x-coordinate used is not first transformed using a high-pass filter, we do not split text-lines using gaps and we do not remove delayed strokes, nor do we do any skew and slant correction or other preprocessing.

The major difference comes from feature extraction. In contrast to the 25 features per point used in [15], we use either 5 features (raw) or 10 features (curves). While the 25 features included both temporal (position in the time series) and spatial features (offline representation), our work uses

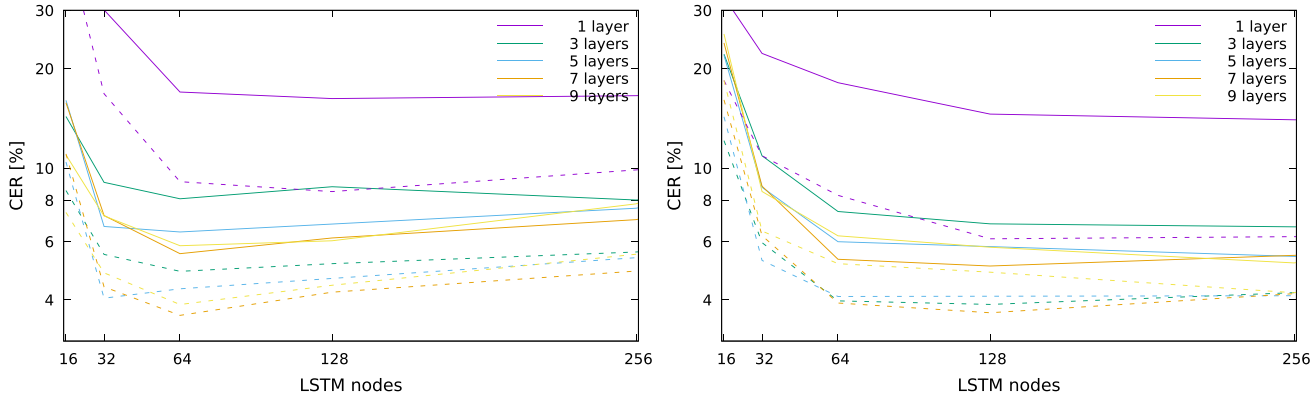

Fig. 5 CER of models trained on the IBM-UB-1 dataset with different numbers of LSTM layers and LSTM nodes using raw (left) and curve (right) inputs. Solid lines indicate results without any language models or feature functions in decoding, and dashed lines indicate results with the fully tuned system

only the temporal structure. In contrast also to our previous system [25], using a more compact representation (and reducing the number of points for curves) allows a feature representation, including spatial structure, to be learned in the first or upper layers of the neural network.

The neural network architecture differs both in internal structure of the LSTM cell as well as in the architecture configuration. Our internal structure differs only in that we do not use peephole connections [9].

As opposed to relying on a single bidirectional LSTM layer of width 100, we experiment with a number of configuration variants as detailed in Fig. 4. We note that it is particularly important to have more than one layer in order to learn a meaningful representation without feature extraction.

We use the CTC forward–backward training algorithm as described in [15], and implemented in TensorFlow. The training hyperparameters are described in Sect. 3.1.

The CTC decoding algorithm incorporates feature functions similarly to how the dictionary is incorporated in the previous state-of-the-art system. However, we use more feature functions, our language models are trained on a different corpus, and the combination weights are optimized separately as described in Sec 3.2.

4.2 IBM-UB-1

Another publicly accessible English-language dataset is the IBM-UB-1 dataset [43]. From the available datasets therein, we use the English query dataset, which consists of 63,268 handwritten English words. As this dataset has not been used often in the academic literature, we propose an evaluation protocol. We split this dataset into 4 parts with non-overlapping writer IDs: 47,108 items for training, 4690 for decoder weight tuning, 6134 for validation and 5336 for testing.[6]

Table 5 Error rates on IBM-UB-1 test set in comparison with our previous system [25]

System	CER (%)	WER (%)
This work (curve, 5x64, no FF)	6.0	25.1
This work (curve, 5x64, FF)	4.1	15.1
Segment-and-decode from [25]	6.7	22.2
This work (production system) (Sect. 5)*	4.1	15.3

A "*" in the "system" column indicates the use of an open training set

We perform a similar set of experiments as we did for IAM-OnDB to determine the right depth and width of our neural network architecture. The results of these experiments are shown in Fig. 5. The conclusion for this dataset is similar to the conclusions we drew for the IAM-OnDB: using networks with 5 layers of bidirectional LSTMs with 64 cells each is sufficient for good accuracy. Less deep and less wide networks perform substantially worse, but larger networks only give small improvements. This is true regardless of the input processing method chosen and again, we do not see a significant difference in the accuracy between the raw and curve representation in accuracy.

We give some exemplary results and a comparison with our current production system as well as results for our previous system in Table 5. We note that our current system is about 38% and 32% better (relative) in CER and WER, respectively, when compared to the previous segment-and-decode approach. The lack of improvement in error rate when evaluating on our production system is due to the fact that our datasets contain spaces while the same setup trained solely on IBM-UB-1 does not.

[6] Information about the exact experimental protocol is available at https://arxiv.org/src/1902.10525v1/anc.

4.3 Additional public datasets

We provide an evaluation of our production system trained on our in-house datasets applied to a number of publicly available benchmark datasets from the literature. More details about our in-house datasets are available from Table 2. Note that for all experiments presented in this section, we evaluate our current live system without any tuning specific to the tasks at hand.

4.3.1 Chinese isolated characters (ICDAR 2013 competition)

The ICDAR-2013 Competition for Online Handwriting Chinese Character Recognition [50] introduced a dataset for classifying the most common Chinese characters. We report the error rates in comparison with published results from the competition and more recent work done by others in Table 6.

We evaluate our live production system on this dataset. Our system was not tuned to the task at hand and was trained as a multi-character recognizer, thus it is not even aware that each sample only contains a single character. Further, our system supports 12,363 different characters, while the competition data only contain 3,755 characters. Note that our system did not have access to the training data for this task at all.

Whenever our system returns more than one character for a sample, we count this as an error. (This happened twice on the entire test set of 224,590 samples.) Despite supporting almost four times as many characters than needed for the CASIA data and not having been tuned to the task, the accuracy of our system is still competitive with systems that were tuned for this data specifically.

4.3.2 Vietnamese online handwriting recognition (ICFHR 2018 competition)

In the ICFHR2018 Competition on Vietnamese Online Handwritten Text Recognition using VNOnDB [34], our production system was evaluated against other systems. The system used in the competition is the one reported and described in this paper. Due to licensing restrictions, we were unable to do any experiments on the competition training data, or specific tuning for the competition, which was not the case for the other systems mentioned here.

Table 6 Error rates on ICDAR-2013 Competition Database of Online Handwritten Chinese Character Recognition

System	ER (%)
Human performance [50]	4.8
Traditional benchmark [30]	4.7
ICDAR-2011 winner [29]	4.2
This work (production system) Sect. 5	3.2
ICDAR-2013 winner: UWarwick [50]	2.6
RNN: NET4 [52]	2.2
100LSTM-512LSTM-512FC-3755FC [49]	2.2
RNN: ensemble-NET123456 [52]	1.9

Our system was trained with an open training set, including a mix of characters, words, and phrases

Table 7 Results on the VNONDB-Word dataset

System	Public test set		Secret test set	
	CER (%)	WER (%)	CER (%)	WER (%)
This work (Sect. 5)	6.1	13.2	9.8	20.5
IVTOVTask1	2.9	6.5	7.3	15.3
MyScriptTask1	2.9	6.5	6.0	12.7

Table 8 Results on the VNONDB-Line dataset

System	Public test set		Secret test set	
	CER (%)	WER (%)	CER (%)	WER (%)
This work (Sect. 5)	6.9	19.0	10.3	27.0
IVTOVTask2	3.2	14.1	5.6	21.0
MyScriptTask2_1	1.0	2.0	1.0	3.4
MyScriptTask2_2	1.6	4.0	1.7	5.1

We participated in the two tasks that best suited the purpose of our system, specifically the "Word" (ref. Table 7) and the "Text line" (ref. Table 8) recognition levels. Even though we can technically process paragraph level inputs, our system was not built with this goal in mind.

In contrast to us, the other teams used the training and validation sets to tune their systems:

- The IVTOV team's system is very similar to our system. It makes use of bidirectional LSTM layers trained end-to-end with the CTC loss. The inputs used are delta x and y coordinates, together with pen-up strokes (Boolean feature quantifying whether a stroke has ended or not). They report using a two-layer network of 100 cells each and additional preprocessing for better handling the dataset.
- The MyScript team submitted two systems. The first system has an explicit segmentation component along with a feed-forward network for recognizing character hypotheses, similar in formulation to our previous system [25]. In addition, they also make use of a bidirectional LSTM system trained end-to-end with the CTC loss. They do not provide additional details on which system is which.

We note that the modeling stacks of the systems outperforming ours in this competition are not fundamentally different (to the best of our knowledge, according to released descriptions). We therefore believe that our system might

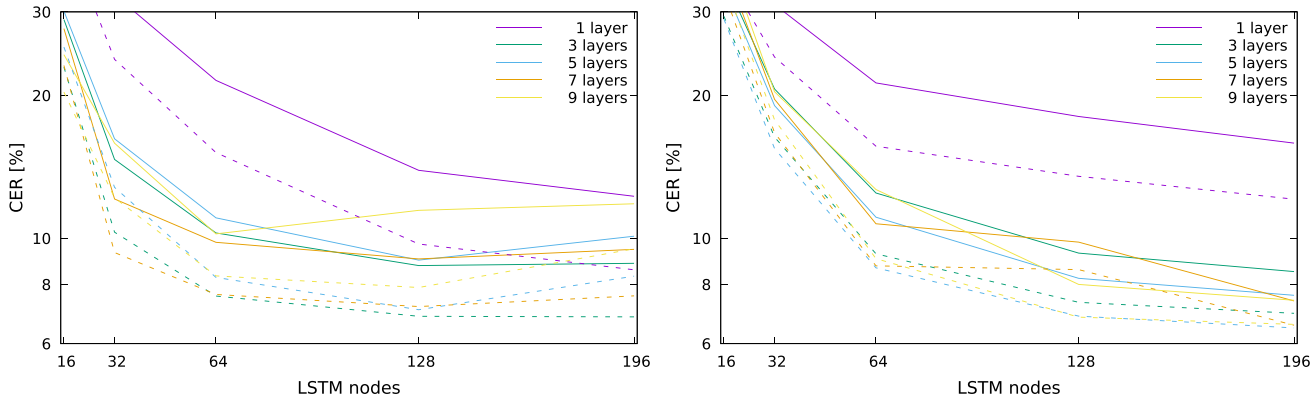

Fig. 6 CER of models trained on our internal datasets evaluated on our English-language validation set with different numbers of LSTM layers and LSTM nodes using raw (left) and curve (right) inputs. Solid lines indicate results without any language models or feature functions in decoding, and dashed lines indicate results with the fully tuned system

perform comparably if trained on the competition training dataset as well.

On our internal test set of Vietnamese data, our new system obtains a CER of 3.3% which is 54% relative better than the old Segment-and-Decode system which had a CER of 7.2% (see also Fig. 7).

4.4 Tuning neural network parameters on our internal data

Our in-house datasets consist of various types of training data, the amount of which varies by script. Sources of training data include data collected through prompting, commercially available data, artificially inflated data, and labeled/self-labeled anonymized recognition requests (see [25] for a more detailed description). This leads to more heterogeneous datasets than academic datasets such as IBM-UB-1 or IAM-OnDB which were collected under standardized conditions. The number of training samples varies from tens of thousands to several million per script, depending on the complexity and usage. We provide more information about the size of our internal training and tests datasets in Table 2.

The best configuration for our system was identified by running multiple experiments over a range of layer depths and widths on our internal datasets. For the Latin-script experiments shown in Fig. 6, the training set we used was a mixture of data from all the Latin-script languages we support and evaluation is done on an English validation dataset, also used for the English evaluation in Table 2.

Similar to experiments depicted in Figs. 4 and 5, increasing the depth and width of the network architecture brings diminishing returns fairly quickly. However, overfitting is less pronounced probably because our datasets are substantially larger than the publicly available datasets.

For the experiments with our production datasets, we are using the Bézier curve inputs which perform slightly better in terms of accuracy than the raw input encoding but are much faster to train and evaluate because of the shorter sequence lengths.

5 System performance and discussion

The setup described throughout this paper that obtained the best results relies on input processing with Bézier spline interpolation (Sect. 2.1.2), followed by 4–5 layers of varying width bidirectional LSTMs, followed by a final softmax layer. For each script, we experimentally determined the best configuration through multiple training runs.

We performed an ablation study with the best configurations for each of the six most important scripts[7] by number of users and compare the results with our previous work [25] (Table 2). The largest relative improvement comes from the overall network architecture stack, followed by the use of the character language model and the other feature functions.

In addition, we show the relative improvement in error rates on the languages for which we have evaluation datasets of more than 2 000 items (Fig. 7). The new architecture performs between 20 and 40% (relative) better over almost all languages.

5.1 Differences between IAM-OnDB, IBM-UB-1 and Our internal datasets

To understand how the different datasets relate to each other, we performed a set of experiments and evaluations with the goal of better characterizing the differences between the datasets.

We trained a recognizer on each of the three training sets separately, then evaluated each system on all three test sets

[7] For Latin script, we report results for 3 languages.

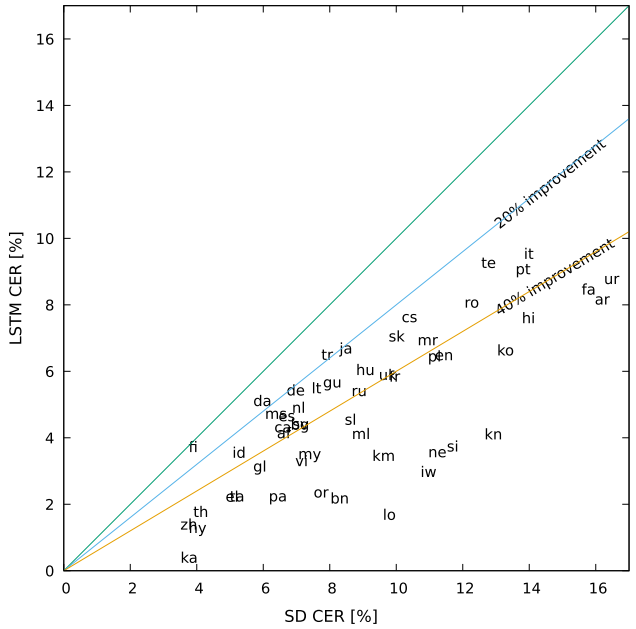

Fig. 7 A comparison of the CERs for the LSTM and segment-and-decode (SD) system for all languages on our internal test sets with more than 2000 items. The scatter plot shows the ISO language code at a position corresponding to the CER for the SD system (x-axis) and LSTM system (y-axis). Points below the diagonal are improvements of LSTM over SD. The plot also shows the lines of 20% and 40% relative improvement

Table 9 CER comparison when training and evaluating IAM-OnDB, IBM-UB-1 and our Latin training/eval set

Train/test	IAM-OnDB	IBM-UB-1	Own dataset
IAM-OnDB	3.8	17.7	31.2
IBM-UB-1	35.1	**4.1**	32.9
Own dataset	**3.3**	4.8	**8.7**

We want to highlight the fundamental differences between the different datasets

(Table 9). The neural network architecture is the same as the one we determined earlier (5 layers bidirectional LSTMs of 64 cells each) with the same feature functions, with weights tuned on the corresponding tuning dataset. The inputs are processed using Bézier curves.

To better understand the source of discrepancy when training on IAM-OnDB and evaluating on IBM-UB-1, we note the different characteristics of the datasets:

- IBM-UB-1 has predominantly cursive writing, while IAM-OnDB has mostly printed writing
- IBM-UB-1 contains single words, while IAM-OnDB has lines of space-separated words

This results in models trained on the IBM-UB-1 dataset not being able to predict spaces as they are not present in the dataset's alphabet. In addition, the printed writing style of IAM-OnDB makes recognition harder when evaluating cursive writing from IBM-UB-1. It is likely that the lack of structure through words-only data makes recognizing IAM-OnDB on a system trained on IBM-UB-1 harder than vice versa.

Systems trained on IBM-UB-1 or IAM-OnDB alone perform significantly worse on our internal datasets, as our data distribution covers a wide range of use-cases not necessarily relevant to, or present, in the two academic datasets: sloppy handwriting, overlapping characters for handling writing on small input surfaces, non-uniform sampling rates, and partially rotated inputs.

The network trained on the internal dataset performs well on all three datasets. It performs better on IAM-OnDB than the system trained only thereon, but worse for IBM-UB-1. We believe that using only cursive words when training allows the network to better learn the sample characteristics, than when learning about space separation and other structure properties not present in IBM-UB-1.

6 Conclusion

We describe the online handwriting recognition system that we currently use at Google for 102 languages in 26 scripts. The system is based on an end-to-end trained neural network and replaces our old segment-and-decode system. Recognition accuracy of the new system improves by 20–40% relative depending on the language while using smaller and faster models. We encode the touch inputs using a Bézier curve representation which performs at least as well as raw touch inputs but which also allows for a faster recognition because the input sequence representation is shorter.

We further compare the performance of our system to the state of the art on publicly available datasets such as IAM-OnDB, IBM-UB-1, and CASIA and improve over the previous best published result on IAM-OnDB.

Acknowledgements We would like to thank the following contributors for fruitful discussions, ideas, and support: Ashok Popat, Yasuhisa Fujii, Dmitriy Genzel, Jake Walker, David Rybach, Daan van Esch, and Eugene Brevdo. We thank Google's OCR team for the numerous collaborations throughout the years that have made this work easier, as well as the speech recognition and machine translation teams at Google for tools and support for some of the components we use in this paper.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/. Software available from tensorflow.org. Accessed 8 Aug 2019
2. Bengio, Y., LeCun, Y., Nohl, C., Burges, C.: Lerec: a NN/HMM hybrid for on-line handwriting recognition. Neural Comput. **7**(6), 1289–1303 (1995)
3. Brants, T., Popat, A.C., Xu, P., Och, F.J., Dean, J.: Large language models in machine translation. In: EMNLP-CoNLL, pp. 858–867 (2007)
4. Chua, M., van Esch, D., Coccaro, N., Cho, E., Bhandari, S., Jia, L.: Text normalization infrastructure that scales to hundreds of language varieties. In: Proceedings of the 11th Edition of the Language Resources and Evaluation Conference (2018)
5. Franzini, M., Lee, K.F., Waibel, A.: Connectionist Viterbi training: a new hybrid method for continuous speech recognition. In: 1990 International Conference on Acoustics, Speech, and Signal Processing, 1990. ICASSP-90, pp. 425–428. IEEE (1990)
6. Frinken, V., Bhattacharya, N., Uchida, S., Pal, U.: Improved blstm neural networks for recognition of on-line bangla complex words. In: S+SSPR (2014)
7. Frinken, V., Uchida, S.: Deep BLSTM neural networks for unconstrained continuous handwritten text recognition. In: ICDAR (2015)
8. Fujii, Y., Driesen, K., Baccash, J., Hurst, A., Popat, A.C.: Sequence-to-label script identification for multilingual OCR. In: ICDAR (2017)
9. Gers, F.A., Schmidhuber, E.: Lstm recurrent networks learn simple context-free and context-sensitive languages. IEEE Trans. Neural Netw. **12**(6), 1333–1340 (2001)
10. Ghosh, S., Joshi, A.: Text entry in indian languages on mobile: user perspectives. In: India HCI (2014)
11. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google vizier: A service for black-box optimization. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1487–1495 (2017)
12. Graves, A., Fernández, S., Gomez, F.J., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: ICML (2006)
13. Graves, A., Jaitly, N.: Towards end-to-end speech recognition with recurrent neural networks. In: ICML (2014)
14. Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J., Fernández, S.: Unconstrained on-line handwriting recognition with recurrent neural networks. In: Advances in Neural Information Processing Systems, pp. 577–584 (2008)
15. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **31**(5), 855–868 (2009)
16. Graves, A., Schmidhuber, J.: Offline handwriting recognition with multidimensional recurrent neural networks. In: Advances in Neural Information Processing Systems, pp. 545–552 (2009)
17. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Magazine **29**(6), 82–97 (2012)
18. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
20. Hu, J., Brown, M.K., Turin, W.: HMM based online handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **18**(10), 1039–1045 (1996)
21. Huang, B.Q., Zhang, Y., Kechadi, M.T.: Preprocessing techniques for online handwriting recognition. In: Seventh International Conference on Intelligent Systems Design and Applications, 2007. ISDA 2007, pp. 793–800. IEEE (2007)
22. Jaeger, S., Manke, S., Reichert, J., Waibel, A.: Online handwriting recognition: the NPen++ recognizer. Int. J. Doc. Anal. Recognit. **3**(3), 169–180 (2001)
23. Jäger, S., Liu, C., Nakagawa, M.: The state of the art in Japanese online handwriting recognition compared to techniques in western handwriting recognition. Int. J. Doc. Anal. Recognit. **6**(2), 75–88 (2003)
24. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: International Conference on Machine Learning, pp. 2342–2350 (2015)
25. Keysers, D., Deselaers, T., Rowley, H., Wang, L.L., Carbune, V.: Multi-language online handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1180–1194 (2017)
26. Kim, J.H., Sin, B.: Online handwriting recognition. In: Handbook of Document Image Processing & Recognition, pp. 887–915. Springer-Verlag London (2014)
27. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: ICLR (2014)
28. LeCun, Y., Bottou, L., andPatrick Haffner, Y.B.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE (1998)
29. Liu, C., Yin, F., Wang, Q., Wang, D.: ICDAR 2011 Chinese handwriting recognition competition. In: ICDAR (2011)
30. Liu, C.L., Yin, F., Wang, D.H., Wang, Q.F.: Online and offline handwritten Chinese character recognition: benchmarking on new databases. Pattern Recognit. **46**(1), 155–162 (2013)
31. Liwicki, M., Bunke, H.: IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard. In: ICDAR, pp. 956–961 (2005)
32. Liwicki, M., Bunke, H., Pittman, J.A., Knerr, S.: Combining diverse systems for handwritten text line recognition. Mach. Vis. Appl. **22**(1), 39–51 (2011)
33. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
34. Nguyen, H.T., Nguyen, C.T., Nakagawa, M.: ICFHR 2018–competition on Vietnamese online handwritten text recognition using HANDS-VNOnDB (VOHTR2018). In: ICFHR (2018)
35. Nuntawisuttiwong, T., Dejdumrong, N.: Approximating online handwritten image by bézier curve. In: CGIV (2012)
36. Pham, V., Bluche, T., Kermorvant, C., Louradour, J.: Dropout improves recurrent neural networks for handwriting recognition. In: 2014 14th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 285–290. IEEE (2014)

37. Pittman, J.A.: Handwriting recognition: Tablet PC text input. IEEE Comput. **40**(9), 49–54 (2007)
38. Plamondon, R., Srihari, S.N.: Online and off-line handwriting recognition: a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. **22**(1), 63–84 (2000)
39. Prasad, M., Breiner, T., van Esch, D.: Mining training data for language modeling across the world's languages. In: Proceedings of the 6th International Workshop on Spoken Language Technologies for Under-resourced Languages (SLTU 2018) (2018)
40. Sainath, T.N., Kingsbury, B., Saon, G., Soltau, H., Mohamed, A.R., Dahl, G., Ramabhadran, B.: Deep convolutional neural networks for large-scale speech tasks. Neural Netw. **64**, 39–48 (2015)
41. Sainath, T.N., Vinyals, O., Senior, A., Sak, H.: Convolutional, long short-term memory, fully connected deep neural networks. In: ICASSP (2015)
42. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Trans. Signal Process. **45**(11), 2673–2681 (1997)
43. Shivram, A., Ramaiah, C., Setlur, S., Govindaraju, V.: IBM_UB_1: A dual mode unconstrained English handwriting dataset. In: ICDAR (2013)
44. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
45. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
46. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.J.: Phoneme recognition using time-delay neural networks. IEEE Trans. Acoust. Speech Signal Process. **37**, 328–339 (1989)
47. Wang, T., Wu, D.J., Coates, A., Ng, A.Y.: End-to-end text recognition with convolutional neural networks. In: ICPR (2012)
48. Yaeger, L., Webb, B., Lyon, R.: Combining neural networks and context-driven search for on-line, printed handwriting recognition in the Newton. AAAI AI Magazine (1998)
49. Yang, Y., Liang, K., Xiao, X., Xie, Z., Jin, L., Sun, J., Zhou, W.: Accelerating and compressing LSTM based model for online handwritten Chinese character recognition. In: ICFHR (2018)
50. Yin, F., Wang, Q.F., Zhang, X.Y., Liu, C.L.: ICDAR 2013 Chinese handwriting recognition competition. In: 2013 12th International Conference on Document Analysis and Recognition (ICDAR), pp. 1464–1470. IEEE (2013)
51. Zhang, J., Du, J., Dai, L.: A gru-based encoderdecoder approach with attention for online handwritten mathematical expression recognition. In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 902–907. IEEE (2017)
52. Zhang, X.-Y., Yin, F., Zhang, Y.-M., Liu, C.-L., Bengio, Y.: Drawing and recognizing chinese characters with recurrent neural network. IEEE Trans. Pattern Anal. Mach. **40**(4), 849–862 (2017)