From Seq2Seq Recognition to Handwritten Word Embeddings: *Supplementary Material*

George Retsinas[1]
gretsinas@central.ntua.gr
Giorgos Sfikas[2]
sfikas@cs.uoi.gr
Christophoros Nikou[2]
cnikou@cs.uoi.gr
Petros Maragos[1]
maragos@cs.ntua.gr

[1] School of Electrical and Computer Engineering
National Technical University of Athens, Greece
[2] Department of Computer Science and Engineering
University of Ioannina, Greece

A Architecture Design and Implementation Details

A.1 Overall Architecture

Figure 1 contains the entire architecture with a detailed per-module description. In the following, we briefly describe the architecture of each basic submodule.

Convolutional Backbone consists of four convolutional stacks ($conv1 - conv4$), comprising $1, 2, 4$ and 4 layers respectively, form residual blocks (all but $conv1$) topped by ReLU nonlinearities preceded by Batch Normalization and dropout layers. Convolution window sizes on stacks 1 to 4 are 7×7, 3×3, 3×3, 3×3, with output channels equal to $32, 64, 128, 256$ respectively. Max-pooling is performed on 2×2 windows of stride 2 between the convolutional stacks. The convolutional backbone is followed a column-wise max-pooling.

CTC branch consists of three 1D convolutional layers with kernel size equal to 5, along with BN, ReLU and dropout are used. The first two have 256 output channels, while the last layer's output channel size is equal to the number of characters ($n\_classes$). A softmax activation is used on the output of the CTC branch in order to provide character predictions. Note that CTC branch is only used in training as an assistive module and discarded during evaluation.

Seq2Seq Encoder uses a bidirectional GRU [3] of 3 layers with 256 hidden size. The $3 \times 2 \times 256$ dimensional output vector of its hidden response (3 layers, 2 directions, 256 hidden size) is then compressed into the desired word embedding (of size $feat\_size$) with a fully connected (FC) linear layer.

Seq2Seq Decoder uses a unidirectional GRU of 1 layer and $feat\_size$ hidden size and takes as input the generated word embedding, as extracted by Seq2Seq Encoder. At each step the GRU takes as input the previous character and the hidden vector computed so far and predict the next character. To this end, the character predictions (represented by one-hot vectors) are transformed by a liner embedding layer into feature vectors equal to the hidden size, while the he output at each step is transformed by a linear (FC) layer into $n_{classes}$ size in order to predict the next character. Since this formulation leads to a classification problem per step, a softmax is applied on each output and the cross entropy loss is selected.

More formally, let g_d be the GRU cell of the decoder module which takes as input a character c_{i-1} and the current hidden state vector $\mathbf{h}_{i-1}$ and outputs the updated hidden state vector $\mathbf{h}_i$ and the next character c_i. According to our formulation, $c_0 = \text{SP}$, where SP stands for the space token. Also, let $\mathbf{x}_{enc}$ the output of the encoder module which would be the input to the decoder module as the initial hidden state vector. Each decoding step can then be written as:

$$\begin{gathered} c_0 = \text{SP} \quad \& \quad \mathbf{h}_0 = \mathbf{x}_{enc} \\ c_i, \mathbf{h}_i = g_d(c_{i-1}, \mathbf{h}_{i-1}), \quad i > 0 \end{gathered} \tag{1}$$

Specifically, the next predicted character c_i is given by selecting the character with the highest probability/score from a vector of characters' probabilities generated as follows:

$$\mathbf{p}_i = softmax(\mathbf{W}^p \mathbf{h}_i) \tag{2}$$

, where $\mathbf{W}^p$ is a weight matrix belonging to the decoder's trainable parameters. The decoding process terminates when a space token is predicted for $i > 0$. The predicted sequence of characters is the string $s = c_0 c_1 \cdots c_{K-1}$, where $c_0 = c_{K-1} = \text{SP}$. Decoder training is described in the following section.

String Encoder consists of a bidirectional GRU of 2 layers with 256 hidden size, where its hidden vector output ($2 \times 2 \times 256$) is projected to the word embedding space by a fully connected layer of $feat\_size$ output size. A linear embedding layer is also used before GRU, which transforms the one-hot representation of characters into feature vectors of size 256, forming sequences of vectors.

Remark: State of the art sequence-to-sequence approaches successfully employ attention mechanisms [9]. As we have already explained, attention is not helpful in our work, since it bypasses the word embedding and therefore it cannot generate discriminative feature vectors. In fact preliminary tests with the presented pipeline along with an attention mechanism indicated a discouraging QbS performance of less than 50% MAP. Nonetheless, attention approaches can notably increase the recognition performance [6, 12].

A.2 Training Details

In this section we describe in more detail several training related aspects of our work.

Multi-task Loss: Training the proposed architecture requires a multi-task loss as Equation 1 suggests. Concerning the multi-task loss weight λ in Eq. 1, we found that applying a larger weight to the Seq2Seq branch ($\lambda = 10$) was beneficial, since CTC branch can be trained more easily and its aim is to assist the overall convergence.

Seq2Seq Decoder: Training of Seq2Seq decoder is performed using the per-character cross entropy loss. Essentially, we try to enforce per-character predictions to match the target sequence $s_t = c'_0 c'_1 \cdots c'_{K-1}$, where $c'_0 = c'_{K-1} = \text{SP}$. Moreover, the target sequence is padded with SP tokens (using a predefined length - larger than any possible predicted word) in order to take into account the potential difference in length of predictions. Formally, the Seq2Seq

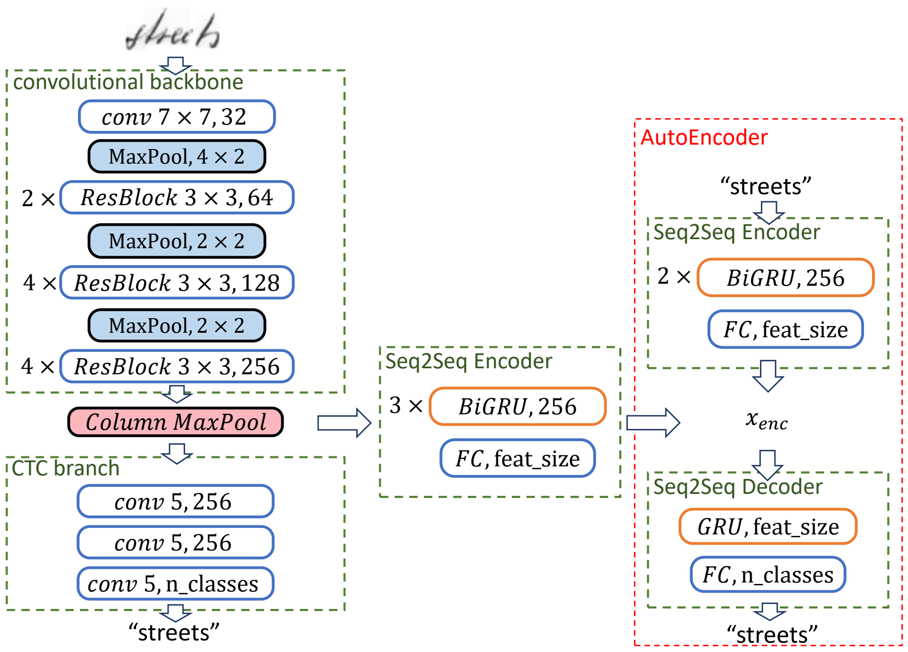

Figure 1: Proposed architecture design. We distinguish four discrete modules: 1) convolutional backbone 2) CTC branch 3) Seq2Seq Encoder 4)Seq2Seq Decoder and 5) String Encoder. Seq2Seq Encoder and Seq2Seq Decoder comprise the Seq2Seq branch, while String Encoder and Seq2Seq Decoder form an autoencoder.

loss is calculated as follows:

$$L_{S2S}(\mathbf{P}_s, s_t) = \sum_{i=0}^{k-1} L_{CE}(\mathbf{W}^p \mathbf{h}_i, c'_i) \tag{3}$$

,where $\mathbf{P}_s$ corresponds to a 2D matrix of the characters' probabilities for each prediction step, i.e. $\mathbf{P}_s = \{\mathbf{p_i}\} = \{\mathbf{W}^p \mathbf{h}_i\}, i = 0 \dots K-1$. This matrix is the output of the whole decoding procedure (the per-step application of g_d), represented by the function f_{dec}. Therefore, we have that $\mathbf{P}_s = f_{dec}(\mathbf{x}_{enc})$.

To assist the training procedure, we employ the *teacher forcing scheme* [2]. Specifically, we randomly select the decoder input at step i to be either the predicted character of the previous step c_{i-1} or the real character c'_{i-1} from the target sequence. In this manner we avoid frequent error propagation from a miss-predicted character during step-by-step decoding.

Character Encoder: The problem that arises from the addition of the extra character encoder module is the simultaneous training along with the visual encoder of the Seq2Seq component. If we train only one path (auto-encoder or Seq2Seq) and we constrain the output of the character encoder to be similar to the visual encoder output, e.g. using MSE loss, there is no guarantee that both encodings can be decoded successfully, since even a small Euclidean divergence between the encoding may result into two different decodings. We overcome this problem by randomly choosing to decode one of the two information flows at each iteration, while constraining them to be close to each other. More formally, let f_{cnn}

be the backbone CNN, f_{enc}/f_{dec} the encoder/decoder of Seq2Seq and f_{cenc} the character encoder. Given an image I and its corresponding groundtruth string $s_t = c'_0 c'_1 \dots c'_{K-1}$, the two encodings can be written as follows:

$$\mathbf{x}_{enc} = f_{enc}(f_{cnn}(I; \mathbf{w}_{cnn}); \mathbf{w}_{enc}) \quad \& \quad \mathbf{x}_{cenc} = f_{cenc}(s_t; \mathbf{w}_{cenc}) \tag{4}$$

The loss for the Seq2Seq branch is then extended according to the following formula (replacing the corresponding term in the full model multi-task loss of Eq.1):

$$L_{S2S}(\mathbf{P}_s, s_t; \mathbf{w}_{cnn}, \mathbf{w}_{s2s}) + d(\mathbf{x}_{enc}, \mathbf{x}_{cenc}) \tag{5}$$

where $d(\cdot,\cdot)$ stands for an extra distance loss (e.g. cosine or euclidean) and the prediction probability matrix $\mathbf{P}_s$ (defined below Eq. 3) corresponds to the decoding of either $\mathbf{x}_{enc}$ or $\mathbf{x}_{cenc}$, randomly selected, as follows:

$$\mathbf{P}_s = f_{dec}(b\mathbf{x}_{enc} + (1-b)\mathbf{x}_{cenc}; \mathbf{w}_{dec}), \quad b \sim Bernoulli(0.5) \tag{6}$$

The distance metric $d(\cdot,\cdot)$ enforces word representations, which are generated by the two different encoders, to be close to each other. Since these word representations are directly used for spotting by employing cosine distance, we used a distance metric that includes cosine distance. However, we do not want these representations of different encoders to drift apart (with respect to L2 norm) since the decoder will undertake the task of projecting different representations into the same string prediction, adding increased complexity to the problem. To this end, we empirically set the following distance function: $d(x,y) = 0.1||x-y||_2 + 1 - cos(x,y)$, where $cos(x,y)$ denotes the cosine similarity function. Note that, alternatively, we could normalize the encoder outputs (with an appropriate layer) and apply a euclidean distance loss, simulating the cosine distance function.

Language Model: Finally, the implicit Language Model variation requires training only the auto-encoder path using word strings, drawn from a corpus. To this end, after each iteration of the main system and backpropagation of the aforementioned multitask loss, we separately train the auto-encoder path with an extra forward-backward pass. We use 128 randomly drawn words at each extra iteration and a lower learning rate (10^{-4}) in order to train them. The lower learning rate was selected in order to apply small "correction" steps to the encoder/decoder submodules towards learning an implicit LM without affecting the performance of the proposed pipeline.

A.3 Inference Details

In this section we explore the details of the evaluation processes, such as force alignment and binarization.

Force-Alignment: Keyword Spotting is performed on the encoding space of word embeddings by straightforwardly comparing words (either images or strings) using the cosine distance. The force-alignment variant follows a rather different rationale since it employs the decoder during evaluation in order to compare the predicted characters against the expected characters of a specific query string. This process can be viewed as employing the training loss in teacher forcing mode in order to compute the score of the query string $s_t = c'_0 c'_1 \dots c'_{K-1}$ against the predicted. Formally, following the decoder definition, the query-constrained scoring can be efficiently performed as the following equations suggest:

$$\begin{aligned} c'_0 = \mathrm{SP} \quad & \& \quad \mathbf{h}_0 = \mathbf{x}_{enc} \\ h_i = g_d(c'_{i-1}, h_{i-1}), & \quad i = 1, \dots, K-1 \\ \text{score} = & \sum_{j=1}^{K-1} L_{CE}(\mathbf{W}^p \mathbf{h}_i, c'_i) \end{aligned} \tag{7}$$

Specifically, we assume that the input of the decoder is the requested query $s_t = c'_0 c'_1 \dots c'_{K-1}$ and thus it is straightforward to predict the next character, given the previous one and the hidden vector computed this far. Consequently, the score is the average cross entropy loss of the predictions and thus if the score is low, the given query was in line with the word representation $\mathbf{x}_{enc}$. This forced-alignment approach can be considerably more compact compared to the typical CTC-based case due to the length of output sequences.

However, a decoder forward pass is required at each step of a character-per-character comparison with respect to the query, which adds additional on-the-fly computational overhead. In other words, implementation-wise, performing a forced alignment as described above is time consuming and cannot be parallelized in order to fully utilize accelerators. We can however overcome the computational cost by organizing multiple queries into a character trie: given a single intermediate feature, we decode over the character trie in a breadth-first traversal manner, making use of multiple nodes for a parallelized, fast implementation on the GPU. Note that we did not implement such an optimal evaluation strategy in this work.

Binarization: The proposed binarization scheme relies on training well-performing binarized embeddings with the use of STE. The binarization is effectively performed by a *sign* operation, which can be straightforwardly translated to a binary representation. Here, we describe implementation-related issues and how such binary embeddings can significantly reduce both storage and time requirements.

The storage reduction of binary descriptos is evident: a float can be replaced by a binary value and thus storage requirements are reduced by a factor of 32. For example, if we have a collection comprised of $1,000$ words (assuming perfect word segmentation), the proposed representation has a dimensionality of 512 with floating-point values, which amounts to 512×4 bytes and overall almost 2.5 GB for the whole collection. On the other hand, by assuming a binary 512-dimensional representation, the overall storage requirements are only $64MB$, namely $32\times$ less storage.

The proposed binarization variation has the exact network inference time with the initial framework, since the same modules are used. Note that all the signed vectors have the same magnitude and therefore, since we care about comparing an image representation with a reference one, the cosine similarity can be efficiently computed by *XNOR* and *bincount* operations (effectively computing a Hamming distance). In other words, using binary embeddings and operations can accelerate the comparison step of the KWS pipeline. However, this speed-up has not been evaluated in this work, since it requires low-level operations which were not available in Python.

A.4 Further Enhancements

The proposed system consists of several sub-modules crafted to address weaknesses of the main sequence-to-sequence pipeline. For example, CTC branch assists convergence, character encoder enables QbS, while the autoencoder path (consisted of the String Encoder and the Seq2Seq Decoder) can be trained using lexicon words to further enhance the learnt implicit language model. Nonetheless we can further assist our system, with the following two approaches:

1) Adding an extra loss term which ensures that different words have representations that differ significantly (e.g. [11]). This could be implemented by using triplets of words: two of the exact same transcription and one of a neighboring transcription, which could be potentially confused. The required triplet loss function tries to bring together the embedding of images with the same text, while driving away - to the greatest extent possible - images that correspond to different texts.

2) Adding a DNN head of fully connected layers, which transforms the encoded representation, generated by Seq2Seq module, to PHOC embeddings. This extra flow of information can be seen as a complementary way to enforce a well-behaved word representation. The binary cross entropy loss, appropriate for PHOC training, would be added as an extra term to the multitask loss of Eq. 1. This way, we expect that the generated word representation can be efficiently decoded to either a PHOC representation or fully decoded to a target string. In the context of this work, the PHOC DNN estimator would be omitted at inference. It only acts as a assistive training module, akin to CTC branch.

These extensions were not evaluated in this paper in order to keep the reasoning of each module as simple as possible and avoid overloading the proposed pipeline.

B Experimental Setup

B.1 Training Protocol

We train the proposed architecture using the Adam optimizer for 80 epochs along with a cosine annealing scheduler restarted every 20 epochs [8]. The initial learning rate after each restart is set to 10^{-3}.

The entire training procedure is repeated 5 times (with random seed) and the mean values are reported. Note that we report the performance of the network at the end of its training procedure and no validation set is used in order to selecting the best performing network (with respect to the validation set) across different epochs or repetitions. In fact, we observed that the system may have slightly better performance few epochs before the end of training or in a previous state, before a scheduler restarting step.

B.2 Keyword Spotting Protocol

As in previous word spotting works, we follow the setting of Almazan et al. [1], therefore we consider queries that include only digits and lowercase letters. We do not distinguish between lowercase and uppercase characters, i.e. 'AND' and 'and' correspond to a single word.

For ICFHR2016 datasets, Botany and Konzilsprotokole [10], the queries and their corresponding matches are provided as groundtruth both for QbS and QbE scenarios.

For GW and IAM datasets, we follow a specific rationale (as in [1]) for selecting the queries: • Omit words consisted of single characters • Consider only words with more than one occurrence in the test set. • Any word that contains punctuation cannot be considered as query. • **IAM only:** ignore stop-words as defined in [1]. All omitted words are retained in the test set and act as "distraction" words.

B.3 Word Recognition Protocol

Word-level recognition protocol, with respect to the character set used, is not clearly stated in the majority of the existing literature. Many approaches [4, 7] seem to follow the paradigm of [1] and compare to it. Specifically, the authors of [1] used a a "workaround" for performing recognition by comparing predicted PHOC embeddings to lexicon word embeddings and thus only a subset of possible characters were considered: digits and lowercase letters, following the keyword spotting setting. Even though we do not focus on the recognition task in this work, it plays a crucial role to the proposed pipeline. Therefore, we shall describe both the training and evaluation protocols that we use in our work.

Since the proposed recognition system is used to train word embeddings, ideal for spotting, we also follow similar settings to keyword spotting. Apart from digits and lowercase characters (after transforming any uppercase to its lowercase counterpart), we also include:

- the space character SP, which denotes the start or the end of a word and is required for Seq2Seq training. The SP character is added before and after every transcription in the train set. During evaluation, all spaces are removed.
- the blank character '_', used for CTC training according to [5].
- a wildcard character '$\star$', which denotes any other character not considered so far (punctuation). We consider that such a wildcard is useful for training, since the system should learn the existence of any character, even though it may be ignored during evaluation.

Evaluation of recognition task is performed as follows. First, we ignore words that contain any wildcard character. Then, blank and space characters are omitted both from the groundtruth and the predicted strings. Note that, following the aforementioned protocol, the wildcard character can appear in the predicted strings. Obviously, if a wildcard character appears in any prediction it would count as an error.

C Additional Experimental Results

C.1 KL Divergence between Representations

The efficiency of the proposed word representations is also evaluated with an alternative method, trying to understand how faithfully they can represent the manifold corresponding to the original word strings. We have computed statistics over the edit distance between words in our corpus (taken from IAM test set) and the corresponding statistics when using cosine distance on either the proposed *binary representation* or the *PHOC* representations ($level5$ unigrams). We used the Kullback-Leibler divergence (KL) to express the statistics' correlation between the reference edit distance and the considered representations, and found the two divergences to be 0.0224 (proposed) and 0.0417 (PHOC). Hence, the proposed binary representation is closer to the desirable statistics, as expressed by edit distances between corpus words, compared to the state-of-the-art PHOC descriptor.

References

[1] J. Almazán, A. Gordo, A. Fornés, and E. Valveny. Word spotting and recognition with embedded attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(12):2552–2566, Dec 2014.

[2] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] Kartik Dutta, Praveen Krishnan, Minesh Mathew, and C.V. Jawahar. Improving CNN-RNN hybrid networks for handwriting recognition. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 80–85. IEEE, 2018.

[5] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[6] Lei Kang, Pau Riba, Marçal Rusinol, Alicia Fornés, and Mauricio Villegas. Distilling content from style for handwritten word recognition. In *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 139–144. IEEE, 2020.

[7] Praveen Krishnan, Kartik Dutta, and C.V. Jawahar. Word spotting and recognition using deep embedding. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 1–6. IEEE, 2018.

[8] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[9] Rohit Prabhavalkar, Tara N Sainath, Bo Li, Kanishka Rao, and Navdeep Jaitly. An analysis of attention in sequence-to-sequence models. In *Interspeech*, pages 3702–3706, 2017.

[10] Ioannis Pratikakis, Konstantinos Zagoris, Basilis Gatos, Joan Puigcerver, Alejandro H. Toselli, and Enrique Vidal. Icfhr2016 handwritten keyword spotting competition (H-KWS 2016). In *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 613–618, 2016.

[11] George Retsinas, Georgios Louloudis, Nikolaos Stamatopoulos, Giorgos Sfikas, and Basilis Gatos. An alternative deep feature approach to line level keyword spotting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12658–12666, 2019.

[12] Jorge Sueiras, Victoria Ruiz, Angel Sanchez, and Jose F Velez. Offline continuous handwriting recognition using sequence to sequence neural networks. *Neurocomputing*, 289:119–128, 2018.