

Revisiting N-Gram Models: Their Impact in Modern Neural Networks for Handwritten Text Recognition

Solène Tarride^[0000-0001-6174-9865] and Christopher
Kermorvant^[0000-0002-7508-4080]

TEKLIA, Paris, France

Abstract. In recent advances in automatic text recognition (ATR), deep neural networks have demonstrated the ability to implicitly capture language statistics, potentially reducing the need for traditional language models. This study directly addresses whether explicit language models, specifically n-gram models, still contribute to the performance of state-of-the-art deep learning architectures in the field of handwriting recognition. We evaluate two prominent neural network architectures, PyLaia [23] and DAN [8], with and without the integration of explicit n-gram language models. Our experiments on three datasets - IAM [19], RIMES [11], and NorHand v2 [2] - at both line and page level, investigate optimal parameters for n-gram models, including their order, weight, smoothing methods and tokenization level. The results show that incorporating character or subword n-gram models significantly improves the performance of ATR models on all datasets, challenging the notion that deep learning models alone are sufficient for optimal performance. In particular, the combination of DAN with a character language model outperforms current benchmarks, confirming the value of hybrid approaches in modern document analysis systems.

Keywords: Handwritten Text Recognition, Neural Networks, Statistical Language Modeling, Tokenization

1 Introduction

Before the era of deep neural networks, handwriting recognition systems [13,5,1], derived from automatic speech recognition, usually combined an optical model, designed to generate character hypotheses from the image, and a language model, responsible for reevaluating these hypotheses using language statistics. The introduction of the statistical language model had significantly reduced the transcription error rates.

As deep neural network performance drastically improved, the need for statistical language models was less prevalent. The rise of transformers and their implicit language modeling capacities [8,17,3] eclipsed even more the need for explicit language models for Automatic Text Recognition (ATR). Today, a key

research area consists in improving models by integrating Large Language Models (LLMs) implicitly into ATR [31] or Automatic Speech Recognition (ASR) [20] models, as LLMs have demonstrated impressive modeling capacities.

However, transformers and LLMs require extensive training data and have primarily shown efficiency in the context of well-resourced, contemporary languages. This efficiency might not readily apply to low-resource languages or historical context. In contrast, n-gram models can find applicability in this context, as they are able to capture language statistics even with limited training data. If n-gram models have already demonstrated their utility in enhancing convolutional recurrent models with CTC [23,10], their impact on transformers remains unexplored.

This study aims to fill this gap by investigating how explicit n-gram language models can complement modern neural network architectures to improve ATR performance on diverse linguistic datasets. We explore different strategies for integrating n-gram models and investigate their impact on the efficiency and accuracy of ATR systems.

This paper is organized as follows. The next Section 2 provides an overview of research related to the topic of language models for ATR. In Section 3, we introduce the datasets used and our methodology. Section 4 explores the optimal parameters for language modeling. Finally, results are presented and discussed in Section 5.

2 Related works

In this section, we present research that addresses the significance of language models in improving Automatic Text Recognition (ATR).

2.1 Tokenization levels in language modeling

The granularity of units used in language modeling is referred to as the tokenization level.

Character-level tokenization treats each character as a separate token and is widely employed for ATR [8,23,3]. Character-level language models are highly valuable for capturing fine-grained patterns, particularly in languages with complex scripts.

On the other hand, word-level tokenization treats entire words as tokens. Word-based language models prove effective in capturing syntactic and semantic relationships in text. Nevertheless, they require substantial training datasets and are prone to out of vocabulary (OOV) issues.

Subword-level tokenization can be viewed as a compromise between the character and word levels, as it involves breaking words into smaller units than words. This level is suited for languages with complex word structures and is also beneficial for handling out-of-vocabulary words. Many tokenization algorithms are

available for subword tokenization, including SentencePiece [15], Byte-Pair Encoding (BPE) [25], and Unigram [14]. Subword tokenization is particularly popular in machine translation and text generation tasks, and it has also become increasingly important for some ATR systems [17].

2.2 Improving Automatic Text Recognition models with language models

Language models play a crucial role in improving the performance of Automatic Text Recognition (ATR) systems, especially in the case of noisy documents or ambiguous writing styles. Language models can help by examining the previous words, subwords, or characters, enabling ATR systems to make more contextually informed decisions.

Implicit language modeling with neural networks In recent architectures, language modeling is implicitly learned by the decoder. Recurrent Neural Networks handle sequential data by maintaining hidden states that capture context from previously predicted tokens. However, they suffer from vanishing gradient problems and limited context capture. To address these issues, more advanced architectures, such as Long Short-Term Memory (LSTM) [23,30] and Gated Recurrent Units (GRUs) [21], have been introduced. These models are designed to overcome the vanishing gradient problem and enhance the capture of long-range dependencies, making them particularly effective in scenarios where context over extended sequences is crucial for accurate processing and understanding, for example in ATR [23]. Recently, the inclusion of attention mechanisms by Transformer models has led to a significant change in NLP. They can effectively model long sequences and now serve as the basis for top-performing models in various applications, such as ATR [3,8,17].

Explicit language modeling with statistical language models Since ATR often encounters challenges on distorted or noisy documents, researchers have also relied on explicit statistical language models to improve the performance of CTC-based neural networks [16,23,35]. The most common form of explicit models used in this context are n-gram models, which are probabilistic models that capture the statistical relationships between sequences of tokens in natural language. They are based on the assumption that the probability of the next token in a sequence depends only on a fixed-sized window of previous tokens. Smoothing strategies are often applied to avoid assigning a zero probability to tokens that have not been previously encountered. These statistical models have proven their worth in other language-related tasks, such as ASR [33]. Explicit language models consider the likelihood of text sequences, and can be used as prior in the decoding process to improve recognition. This context-awareness allows ATR systems to choose more contextually appropriate interpretations, resulting in improvements in recognition. So far, n-gram models have been combined to ATR models at character [10,35,29] and word [23,29] levels.

Error correction as post-processing Finally, another way to improve recognition consists in correcting errors as a post-processing step [22]. Different strategies have been proposed over the years, either based on isolated word correction (merging OCR output, lexical approaches, ...) or based on contextual language models (statistical language models, sequence-to-sequence models [26]). If error correction techniques can be helpful during post-processing, it is often preferable to rescore hypotheses during decoding in order to preserve all hypothesis probability distributions from the original ATR model.

2.3 Discussion

After reviewing the literature, two observations can be made.

First, explicit language modeling in ATR has been gradually replaced by models with implicit language modeling capabilities, such as transformers [8,3]. However, the potential integration of explicit n-gram models with transformers remains unexplored, although similar combinations have been investigated in other domains [7,24]. In their study, Diaz et al. [10] found that using CTC-based decoders with explicit n-gram models outperformed transformer decoders, and suggested that combining transformer models with n-gram models could further improve ATR results.

Second, ATR models traditionally operate at the character or word level [23]. However, there is a current trend to explore the integration of intermediate tokenization levels, such as subwords [17,15] or multiple tokenization levels [29]. The integration of explicit language models trained at different text granularities could improve ATR systems, traditionally working at character-level.

3 Datasets and models

In this section, we describe the datasets and models used for our experiments.

3.1 Datasets

We evaluate our systems on three datasets, each covering a different language: IAM, RIMES and NorHand. Figure 1 presents an example of each dataset, and Table 1 details the dataset splits.

IAM The IAM dataset [19] is composed of modern documents in English, written by 500 writers. It includes 747 training pages with corresponding transcriptions. For this dataset, volunteers were asked to write a short text extracted from a book. For our experiments, we use the RWTH split. We perform experiments at two levels: text lines and paragraphs. Since our focus is on handwriting recognition, we use paragraphs instead of pages to exclude the printed instructions in the header.

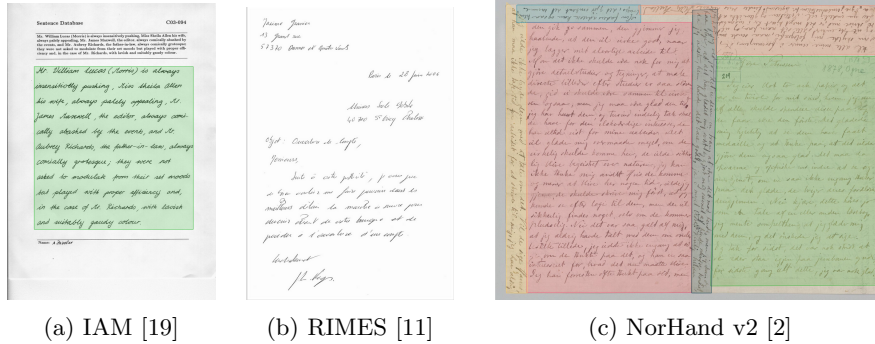


Fig. 1: Examples of pages from the three datasets used in this work. Note that we use full pages for RIMES and paragraphs for IAM (to exclude the printed header) and NorHand v2 (to simplify reading order). Paragraphs are highlighted.

Table 1: Number of lines, paragraphs and pages used for training, validation and testing for each dataset.

Dataset	Lines			Paragraphs			Pages		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
IAM	6,482	976	2,915	747	116	336	747	116	336
RIMES	10,195	1,138	778	-	-	-	3,700	249	100
NorHand	146,422	15,089	1,573	8,666	885	90	6,551	708	126

RIMES The RIMES dataset [11] is composed of administrative documents written in French. We use the *Letters* subset that includes 3700 training pages with corresponding transcriptions. For this subset, volunteers were instructed to write an administrative letter in their own words. For lines, we use the RIMES-2011 version which includes only text lines from the letters’ body (address, reference numbers, signatures are excluded).

NorHand v2 The NorHand [18] dataset consists of Norwegian letters from the 19th and early 20th century. For the experiments, we train models on the version 2 of the NorHand dataset¹ [2], recently released on Zenodo. On this dataset, the test set is designed to maximize diversity, with each test page written by a different author. Note that the paragraphs on a page may have varying orientations, as illustrated in Figure 1c. As a result, various reading orders can appear within a single page. We decided to train our model on paragraphs instead of pages to ensure that the reading order is consistent in a single image. Since we have no information on the orientation, we include all paragraphs in the training and validation phases for paragraph-level training. However, during evaluation, we restrict our assessment to paragraphs with the expected reading order. For

¹ <https://zenodo.org/records/10555698>

line-level training, we include all horizontal lines, which does not prevent us from training on upside-down text-lines.

3.2 Models

Our experiments are carried out using two models: PyLaia [23] and DAN [8]. These models were chosen because they are open-source, and yield state-of-the-art results at line-level [18] and page-level [8,27].

PyLaia² is an open source model for handwritten text recognition. It combines 4 convolutional layers and 3 recurrent layers, and is trained with the CTC loss function. The last layer is a linear layer with a softmax activation function that computes probabilities associated with each character of the vocabulary.

PyLaia is trained on text line images resized with a fixed height of 128 pixels and a batch size of 8. We use early stopping to avoid overfitting: the training is stopped after 80 epochs without improvement. PyLaia can be trained in a few hours on a NVIDIA GeForce RTX 3080 Ti (12Go), depending on the dataset size.

DAN³ is an open source hybrid model, consisting of a CNN encoder followed by an attention-based Transformer decoder. It is designed for handwritten text recognition, and can work directly on images of paragraphs or pages. DAN is trained with the cross-entropy loss function. The last layer is a linear layer with a softmax activation function that computes probabilities associated with each character of the vocabulary.

For each dataset, we train DAN on text lines and on pages (or paragraphs) to study the impact of language models at different scales. When training on lines, we resize images to a fixed height of 128 pixels, and use a batch size of 8. When training on paragraphs or pages, we use a batch size of 2 due to memory constraints, and rescale images so that they do not exceed 1250 pixels x 2500 pixels. DAN can be trained in a few days on a NVIDIA TESLA V100 (32Go), depending on the dataset size.

3.3 Explicit language modeling

In this section, we describe how language models can be built and combined with PyLaia and DAN.

Building n-gram language models Two prominent libraries for constructing n-gram language models are:

² <https://github.com/jpuigcerver/PyLaia>

³ <https://github.com/FactoDeepLearning/DAN>

KenLM is a widely-used and efficient library for creating and using n-gram language models. It offers tools for training models on large text corpora and features fast and memory-efficient decoding. KenLM supports various tokenization levels, making it versatile for different ATR and ASR applications.

The SRILM toolkit is a comprehensive solution for language modeling, including n-gram models. SRILM provides a range of utilities for building, training, and applying language models. It is extensively used in academia and industry for ASR, machine translation, and other language processing tasks, offering flexibility and robust performance. Both toolkits are compatible with Flashlight/TorchAudio CTCDecoder.

Language model integration Once trained, PyLaia and DAN are used to decode with and without an explicit language model.

Decoding without any language model consists in a greedy decoding, where the character with the highest probability is selected at each time step. For DAN, the decoding stops when the end-of-sequence token is predicted (seq2seq model), while for PyLaia there is no such token (CTC model).

In contrast, decoding with a language model relies on the beam search decoding algorithm. This method combines the probability matrix from the optical model with the conditional probabilities of the language model, taking into account lexicon constraints. In our implementation, language models are built at character, subword and word levels using the KenLM library. We rely on torchaudio’s `ctc_decoder`⁴.

Note that for now, torchaudio supports the Connectionist Temporal Classification (CTC) mode, however the sequence-to-sequence (S2S) mode is not available in Python bindings yet. As a result, the integration in PyLaia was straightforward, but more challenging with DAN. To make DAN compatible with the CTC mode, we had to add fake CTC frames between frames to avoid the removal of duplicate characters. This simple trick allowed the use of language models with DAN without impacting the optimal decoding path. Note that in both case, the optical part of the decoding can be performed on GPU. However, the language model re-scoring is always performed on CPU.

Our implementation of PyLaia combined with n-gram models is now open-source⁵ and is described in [28].

4 Language model optimization

In this section, we explore the optimal parameters to use for language modeling. More specifically, we explore the tokenization algorithms, the n-gram order, the weight of the language model, smoothing, and the impact of using additional text data. Optimal parameters are optimized on the NorHand dataset.

⁴ https://pytorch.org/audio/main/generated/torchaudio.models.decoder.ctc_decoder.html

⁵ <https://gitlab.teklia.com/atr/pylaia>

4.1 Tokenization level

N-gram language models can be built at different granularities. Representing language as a sequence of words provides a broader context, but presents challenges in dealing with unseen words. In contrast, character modeling eliminates the problem of out-of-vocabulary words, but provides a narrow context because the model only considers the preceding n characters. Choosing subword modeling is a useful compromise that breaks down rare words into more common subwords, reducing the impact of the out-of-vocabulary problem while maintaining a broader context. Since PyLaia and DAN predict sequences of characters, it may be interesting to combine them with n-gram models at a different level of granularity to enrich the language context.

Three levels of tokenization are explored for language modeling: character-level, subword-level, and word-level. An example of tokenization at different levels is presented in Table 2.

Table 2: An example of text tokenized at different levels

Level	Tokenized text
Characters	T h e _ _ n u m e r i c a l l y _ _ l a r g e s t _ _ g r o u p
Subwords	The _ _ numer ic ally _ _ large st _ _ gro up
Words	The _ _ numerically _ _ largest _ _ group

Character tokenization is straightforward, as each token represents a character. For word tokenization, we rely on the `wordpunct_tokenize` function from the NLTK Python Package, which considers punctuation as separate words. Subword tokenization can be achieved with different tokenization algorithms. In this study, we trained SentencePiece [15] with a vocabulary size of 1000 subwords. This value was chosen as a compromise between the number of distinct characters and the number of distinct words. However, further experiments could be performed to determine the optimal value for the vocabulary size. Two different representations were initially tested for subword tokenization.

- **SentencePiece representation.** The first representation follows the SentencePiece mode, and includes spaces in tokens, e.g. `_ _numer ic ally`. In this case, a single subword has two distinct representations based on whether it appears at the beginning of a word or not.
- **SeparateSpaces representation.** The second representation handles spaces as separate tokens, e.g. `_ _ numer ic ally`. In this case, each subword has a unique representation that does not depend on its position in a word.

Our initial experiments demonstrated a marginal improvement in results (1% WER on Norhand) with the second representation method. Consequently, we adopted this method for future experiments.

4.2 Order

N-gram models compute the probability of a token based on the preceding sequence of $n-1$ tokens. In our evaluation, we compared various values for the order n , spanning from 1 to 6 since SRILM and KenLM do not support higher orders at the moment.

Our experiments show that the optimal order depends on the tokenization level of the language model. Notably, for character and subword language models, performance consistently improved with the increase in order. Consequently, the peak performance was observed at $n = 6$, suggesting that further increases in order would likely lead to continued performance improvement. In contrast, the best performance was obtained with $n = 3$ for word-based language models.

4.3 Language weight

When combining PyLaia and DAN with n-gram language models, the weight assigned to the language model can be defined.

A one-dimensional grid search, spanning from 0 to 5 with a step of 0.5, was applied to determine the optimal weight to give language modeling. Our experiments show that the optimal weight depends on the tokenization level of the language model. For character-level and subword-level language models, the most effective weight was determined to be 1.5 for both DAN and PyLaia. On the other hand, the optimal weight for word-level language models was found to be lower, equal to 0.5 for both DAN and PyLaia.

4.4 Smoothing

To keep the language model from assigning zero probability to unseen tokens, it is recommended to apply smoothing on the probability distribution [12]. We conducted experiments to determine the best smoothing method among no smoothing, Kneser-Ney smoothing, and Witten-Bell smoothing.

Our experiments show that Kneser-Ney smoothing [6] leads to the best performance by a thin margin. Although it does not have a huge impact on performance, it avoids errors on unseen sequences. Note that the Kneser-Ney smoothing is the default smoothing method used in KenLM.

5 Results and discussion

In this section, we present results with and without language models on three datasets, using the parameters described in the previous section. The results are presented in Table 3 at line-level and in Table 4 at page-level. We analyze and discuss the conclusions that can be drawn from our experiments.

5.1 Impact of language models on performance

In this section, we explore the effects of tokenization levels on language modeling, investigate the influence of model architecture, and analyze the impact of varying image input levels.

Table 3: Comparison of models trained at line-level with different tokenization levels for language modeling. Evaluation results are presented on the test set (%). Best results for each dataset appear in bold, best results for each model are underlined.

Dataset	LM level	PyLaia		DAN		N lines
		CER	WER	CER	WER	
NorHand	None	9.72	27.78	7.85	22.07	1494
	Character	<u>8.23</u>	21.65	7.23	19.87	1494
	Subword	8.48	<u>21.60</u>	7.66	21.16	1494
	Word	8.79	23.20	9.00	27.02	1494
RIMES	None	4.57	14.72	3.22	10.69	778
	Character	<u>3.79</u>	<u>10.94</u>	3.22	10.83	778
	Subword	4.26	12.11	3.15	10.26	778
	Word	4.28	12.59	3.19	<u>10.08</u>	778
IAM	None	8.44	24.51	4.86	16.39	2,915
	Character	<u>7.50</u>	20.98	4.38	14.40	2,915
	Subword	8.05	21.87	4.58	14.74	2,915
	Word	12.02	27.70	7.56	21.80	2,915

Table 4: Comparison of models trained at page or paragraph-level with different tokenization levels for language modeling. Evaluation results are presented on the test set (%). Best results for each dataset appear in bold.

Dataset	LM level	DAN		N pages
		CER	WER	
NorHand	None	12.28	28.17	81
	Character	11.28	25.04	81
	Subword	12.27	29.29	81
	Word	13.68	33.09	81
RIMES	None	2.22	8.23	100
	Character	2.13	7.96	100
	Subword	2.22	8.44	100
	Word	2.64	10.27	100
IAM	None	4.30	14.29	336
	Character	3.95	12.71	336
	Subword	4.10	13.22	336
	Word	7.56	21.18	336

Tokenization level Below, we discuss how different levels of tokenization affect ATR performance across datasets.

On IAM, character-level language model (LM) shows superior performance, as reflected by a small relative reduction of 13% in word error rate compared to the model without any LM. While subword-based LM contributes marginal improvements, word-based LM tends to show a decline, particularly on IAM, where the word error rate increased by 31% compared to the model without LM. This trend may be due to the relatively smaller size of the dataset, resulting in lower word frequencies compared to NorHand and Rimes.

On RIMES, the character LM proves to be highly effective for PyLaia, with a relative improvement of 26% in terms of word error rates, while its impact on DAN Line and DAN Page is more marginal due to already low error rates. Subword language models show moderate efficiency for PyLaia, with minimal impact on DAN. Surprisingly, word language models show a significant decrease for DAN Page, while performing well for DAN Line and PyLaia.

On NorHand, character-based language modeling emerges as the optimal choice for all models, with an average relative improvement of 15% in terms of WER. The subword language model is also a solid choice for PyLaia, although it shows a marginal improvement for DAN. The word-based language model is ineffective for DAN and completely negative for PyLaia.

Overall, character-based language models are the optimal choice, consistently outperforming subword models. Word-based models, on the other hand, lead to a decline in performance, mainly due to challenges with out-of-vocabulary words.

Model architecture DAN Line outperforms PyLaia on the IAM and RIMES datasets, but surprisingly, PyLaia is better on the NorHand dataset. After analyzing the results, we observe some hallucinations from DAN on NorHand, which is a known issue with Transformer models [8,3], causing a decline in performance.

The results show that explicit language modeling improves both PyLaia and DAN. If the relative improvement is comparable on IAM, our experiments indicate that PyLaia benefits more from explicit language modeling than DAN on NorHand, and even more on RIMES. Character-based language modeling remained the optimal choice for both models, while word-based language modeling generally failed to improve results.

Image level DAN Page and Line showed similar trends. Character-based language models were consistently optimal, while sub-word based language models showed limited positive effects. The use of Word LM consistently resulted in a decline in performance. The improvement on RIMES is low, since error rates without language models are already low. However, on IAM and Norhand, explicit language modeling significantly improved results. It is interesting to note that DAN performs generally better when trained on pages rather than lines, at least for IAM and RIMES. This observation suggests that it benefits from a larger context to predict characters. Surprisingly, this is not the case for NorHand v2, which can be explained by the difficulties of this dataset: varying paragraph

sizes (a paragraph can be a set of two lines or a full page) and unknown reading order (trained with different orientations which can impact feature extraction).

5.2 Impact of language models on speed

During inference, computations can be efficiently performed on the GPU up to the language model decoder, which due to current limitations is performed exclusively on the CPU. Consequently, the inclusion of a language model during decoding improves recognition accuracy, but at the cost of a slowdown in prediction speed. Specifically, for PyLaia, decoding with a language model results in a speed reduction by a factor of 10. On the other hand, in the case of DAN, where the decoding time is inherently higher, the proportional impact is less pronounced, as detailed in the table 5. We recommend the use of language models for batch processing of documents, since their application may not be suitable for real-time processing.

Table 5: Impact of language models on decoding time (in seconds/image).

Device	LM level	PyLaia Line	DAN Line	DAN Page
GPU	None	0.01	0.47	5.72
GPU + CPU	Character	0.12	0.69	8.13
GPU + CPU	Subword	0.09	0.67	7.78
GPU + CPU	Word	0.11	0.53	6.84

5.3 Comparison with state-of-the-art models

In this section, we compare the results obtained in this study with state-of-the-art models. Results are presented in Table 6 for models trained on text-lines and in Table 7 for models trained on paragraphs or pages. Several key observations can be made.

First, we establish a new baseline on the NorHand v2 dataset at line-level and page-level. As this dataset was released recently, we could not find any existing models for comparison. On this dataset, DAN systematically outperforms PyLaia, especially when combined with a character language model.

At page-level, the DAN with language modeling demonstrates state-of-the-art performance on RIMES and IAM. However, at line-level, the same model falls slightly below other methods, while achieving near state-of-the-art results. On these two datasets, the best model is the *S-Attn/CTC*, which uses a character-level language model and relies on additional data during training.

Table 6: Comparison with state-of-the-art models working at line-level. Models introduced in this article appear in bold. Best scores also appear in bold.

Dataset	Model	LM level	Add. data	CER	WER
NorHand v2	PyLaia-LM	None	No	8.2	21.7
	DAN-LM	Character	No	7.2	19.9
	DAN-LM	Subword	No	7.7	21.2
RIMES	SFR [32]	Character	No	2.1	9.3
	S-Attn / CTC [10]	Character	Yes	2.0	-
	DAN-LM	Character	No	3.2	10.1
	MDLSTM [30]	Word	No	2.8	9.6
	Laia [23]	Word	No	4.4	12.2
IAM	TrOCR large [17]	None	Yes	2.9	-
	VAN [9]	None	No	5.0	16.3
	S-Attn / CTC [10]	Character	Yes	2.8	-
	DAN-LM	Character	No	4.4	14.4
	MDLSTM [30]	Hybrid	No	3.5	9.3

Table 7: Comparison with state-of-the-art models working at page-level (RIMES) or paragraph-level (IAM, NorHand). Models introduced in this article appear in bold. Best scores also appear in bold.

Dataset	Model	LM level	Add. data	CER	WER
NorHand v2	DAN	None	No	12.3	28.2
	DAN-LM	Character	No	11.3	25.0
	DAN-LM	Subword	No	12.3	29.3
RIMES	Encoder Decoder [4]	None	No	2.9	12.6
	DAN [8]	None	No	4.5	11.9
	SFR [32]	Character	No	2.1	9.3
	DAN-LM	Character	No	2.1	8.0
IAM	SFR [32]	None	No	6.4	23.2
	OrigamiNet [34]	None	No	4.7	-
	VAN [9]	None	No	4.5	14.6
	DAN-LM	Character	No	4.0	12.7
	Encoder Decoder [4]	Word	No	5.5	16.4

6 Conclusion

In this article, we have explored the combination of modern neural networks and n-gram language models. N-gram language models were progressively abandoned, as recent architecture demonstrated impressive language modeling performance. The results of this study highlight that these technologies can be combined to further improve performance in Automatic Text Recognition. We have studied the impact of language modeling at character, subword, and word levels for two models: PyLaia [28] and DAN [8]. Since these models predict sequences of characters, our initial intuition was that subword-level language modeling would be optimal, as it would enrich the language context without introducing out-of-vocabulary issues [17]. Our results show that, despite subword language model yielding decent improvement, the optimal performance is achieved using character-level language modeling. Finally, we also show that language models improve ATR models whether they are trained on text-lines, paragraphs or full pages. As a result of our study, we set a new standard on IAM, RIMES and NorHand v2 at paragraph and page-level. One of the highlight of this work is that n-gram language models are auto-tuned: they can be trained directly on the training set without requiring any additional data.

For future works, we plan to explore how different granularities can be combined in language models. Additionally, we would like to study how language models can be leveraged to quickly adapt generic optical models to a specific style or period.

Acknowledgement

This work was supported by the Research Council of Norway through the 328598 IKTPLUSS HuginMunin project.

References

1. Arora, A., Chang, C.C., Rekabdar, B., BabaAli, B., Povey, D., Etter, D., Raj, D., Hadian, H., Trmal, J., Garcia, P., Watanabe, S., Manohar, V., Shao, Y., Khudanpur, S.: Using ASR Methods for OCR. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 663–668 (2019). <https://doi.org/10.1109/ICDAR.2019.00111>
2. Beyer, Y., Solberg, P.E.: Norhand v2 / dataset for handwritten text recognition in norwegian [data set] (2024). <https://doi.org/10.5281/zenodo.10555698>
3. Blecher, L., Cucurull, G., Scialom, T., Stojnic, R.: Nougat: Neural Optical Understanding for Academic Documents (2023), <https://arxiv.org/abs/2308.13418>
4. Bluche, T.: Joint line segmentation and transcription for end-to-end handwritten paragraph recognition. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. p. 838–846. NIPS’16, Curran Associates Inc., Red Hook, NY, USA (2016). <https://doi.org/10.5555/3157096.3157190>

5. Bluche, T., Louradour, J., Knibbe, M., Moysset, B., Benzeghiba, M.F., Kermorvant, C.: The A2iA Handwritten Arabic Text Recognition System at the OpenHART2013 Evaluation Campaign . In: Document Analysis Systems (2014). <https://doi.org/10.1109/DAS.2014.40>
6. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* **13**(4), 359–394 (1999). <https://doi.org/10.1006/csla.1999.0128>
7. Choi, H., Lee, J., Yang, J.: N-gram in swin transformers for efficient lightweight image super-resolution. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2071–2081. IEEE Computer Society, Los Alamitos, CA, USA (jun 2023). <https://doi.org/10.1109/CVPR52729.2023.00206>
8. Coquenat, D., Chatelain, C., Paquet, T.: DAN: a segmentation-free document attention network for handwritten document recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp. 1–17 (2023). <https://doi.org/10.1109/tpami.2023.3235826>
9. Coquenat, D., Chatelain, C., Paquet, T.: End-to-end handwritten paragraph text recognition using a vertical attention network. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **45**(1), 508–524 (2023). <https://doi.org/10.1109/TPAMI.2022.3144899>
10. Diaz, D.H., Qin, S., Ingle, R.R., Fujii, Y., Bissacco, A.: Rethinking text line recognition models. *CoRR* **abs/2104.07787** (2021)
11. Grosicki, E., El-Abed, H.: Icdar 2011 - french handwriting recognition competition. In: 2011 International Conference on Document Analysis and Recognition. pp. 1459–1463 (2011). <https://doi.org/10.1109/ICDAR.2011.290>
12. Jurafsky, D., Martin, J.: *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. (2020), <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>
13. Kozielski, M., Doetsch, P., Ney, H.: Improvements in RWTH’s system for off-line handwriting recognition. In: International Conference on Document Analysis and Recognition (2013). <https://doi.org/10.1109/ICDAR.2013.190>
14. Kudo, T.: Subword regularization: Improving neural network translation models with multiple subword candidates. In: Gurevych, I., Miyao, Y. (eds.) *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. pp. 66–75. Association for Computational Linguistics, Melbourne, Australia (Jul 2018). <https://doi.org/10.18653/v1/P18-1007>
15. Kudo, T., Richardson, J.: Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR* **abs/1808.06226** (2018). <https://doi.org/10.18653/v1/D18-2012>
16. Kumar, S., Nirschl, M., Holtmann-Rice, D., Liao, H., Suresh, A.T., Yu, F.: Lattice rescoring strategies for long short term memory language models in speech recognition (2017). <https://doi.org/10.1109/ASRU.2017.8268931>
17. Li, M., Lv, T., Cui, L., Lu, Y., Florêncio, D.A.F., Zhang, C., Li, Z., Wei, F.: TrOCR: Transformer-based optical character recognition with pre-trained models. In: AAAI Conference on Artificial Intelligence (2021). <https://doi.org/10.1609/aaai.v37i11.26538>
18. Maarand, M., Beyer, Y., Kåsen, A., Fosseide, K.T., Kermorvant, C.: A comprehensive comparison of open-source libraries for handwritten text recognition in norwegian. In: Document Analysis Systems: 15th IAPR International Workshop, DAS 2022, La Rochelle, France, May 22–25, 2022, Proceedings. p. 399–413. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-06555-2_27

19. Marti, U.V., Bunke, H.: The iam-database: an english sentence database for offline handwriting recognition. vol. 5, pp. 39–46 (2002). <https://doi.org/10.1007/s100320200071>
20. Min, Z., Wang, J.: Exploring the integration of large language models into automatic speech recognition systems: An empirical study. In: Luo, B., Cheng, L., Wu, Z.G., Li, H., Li, C. (eds.) *Neural Information Processing*. pp. 69–84. Springer Nature Singapore, Singapore (2024). https://doi.org/10.1007/978-981-99-8181-6_6
21. Neto, A.F.S., Bezerra, B.L.D., Toselli, A.H., Lima, E.B.: A Robust Handwritten Recognition System for Learning on Different Data Restriction Scenarios. *Pattern Recognition Letters* **1**, 1–7 (4 2022). <https://doi.org/10.1016/j.patrec.2022.04.009>
22. Nguyen, T.T.H., Jatowt, A., Coustaty, M., Doucet, A.: Survey of post-ocr processing approaches. *ACM Comput. Surv.* **54**(6) (jul 2021). <https://doi.org/10.1145/3453476>
23. Puigcerver, J.: Are multidimensional recurrent layers really necessary for handwritten text recognition? In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). vol. 01, pp. 67–72 (2017). <https://doi.org/10.1109/ICDAR.2017.20>
24. Roy, A., Anil, R., Lai, G., Lee, B., Zhao, J., Zhang, S., Wang, S., Zhang, Y., Wu, S., Swavely, R., Tao, Yu, Dao, P., Fifty, C., Chen, Z., Wu, Y.: N-grammer: Augmenting transformers with latent n-grams (2022). <https://doi.org/10.48550/arXiv.2207.06366>
25. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. In: Erk, K., Smith, N.A. (eds.) *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. pp. 1715–1725. Association for Computational Linguistics, Berlin, Germany (Aug 2016). <https://doi.org/10.18653/v1/P16-1162>
26. Soper, E., Fujimoto, S., Yu, Y.Y.: BART for post-correction of OCR newspaper text. In: *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*. pp. 284–290. Association for Computational Linguistics, Online (Nov 2021). <https://doi.org/10.18653/v1/2021.wnut-1.31>
27. Tarride, S., Boillet, M., Kermorvant, C.: Key-value information extraction from full handwritten pages. In: Fink, G.A., Jain, R., Kise, K., Zanibbi, R. (eds.) *Document Analysis and Recognition - ICDAR 2023*. pp. 185–204. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-41679-8_11
28. Tarride, S., Schneider, Y., Generali, M., Boillet, M., Abadie, B., Kermorvant, C.: Improving automatic text recognition with language models in the pylaia open-source library. In: Submitted at ICDAR (2024)
29. Tassopoulou, V., Retsinas, G., Maragos, P.: Enhancing handwritten text recognition with n-gram sequence decomposition and multitask learning. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 10555–10560. IEEE Computer Society, Los Alamitos, CA, USA (jan 2021). <https://doi.org/10.1109/ICPR48806.2021.9412351>
30. Voigtlaender, P., Doetsch, P., Ney, H.: Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks. In: 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR). pp. 228–233 (2016). <https://doi.org/10.1109/ICFHR.2016.0052>
31. Wang, D., Raman, N., Sibue, M., Ma, Z., Babkin, P., Kaur, S., Pei, Y., Nourbakhsh, A., Liu, X.: DocLLM: A layout-aware generative language model for multimodal document understanding (2023)

32. Wigington, C., Tensmeyer, C., Davis, B., Barrett, W., Price, B., Cohen, S.: Start, follow, read: End-to-end full-page handwriting recognition. In: Proceedings of the European Conference on Computer Vision (ECCV) (September 2018). https://doi.org/10.1007/978-3-030-01231-1_23
33. Xu, H., Chen, T., Gao, D., Wang, Y., Li, K., Goel, N., Carmiel, Y., Povey, D., Khudanpur, S.: A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5929–5933 (2018). <https://doi.org/10.1109/ICASSP.2018.8461974>
34. Yousef, M., Bishop, T.E.: Origaminet: Weakly-supervised, segmentation-free, one-step, full page textrecognition by learning to unfold. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020). <https://doi.org/10.1109/CVPR42600.2020.01472>
35. Zhang, H., Liang, L., Jin, L.: SCUT-HCCDoc: A New Benchmark Dataset of Handwritten Chinese Text in Unconstrained Camera-captured Documents. Pattern Recognition p. 107559 (2020). <https://doi.org/10.1016/j.patcog.2020.107559>