Weakly-supervised text-line analysis & Writing style modelling

Julien Gaubil, julien.gaubil@ens-paris-saclay.fr
M2 MVA, ENS Paris-Saclay
Deep Learning 2022-2023 – Project Report

Abstract

The use of Deep Learning models for Handwritten Text Recognition (HTR) applications led to progress in the automatic analysis of such documents. Some open research directions are however still opened, since advanced DL models often lack of explainability, yielding results that are hard to interpret. Despite these recent progress, these models also struggle with datasets that contain challenging handwritings such as historical documents, where there is an important variability in the aspect of the characters. This project aims at developing interpretable Deep Learning methods that could perform a detailed analysis of challenging handwritten documents. It builds on an existing method generative approach, the Learnable Typewriter [15] that performs character analysis and recognition in images of text lines in an interpretable way using little to no supervision. This document-specific approach performs multi-object segmentation by reconstructing input text lines using automatically extracted visual elements called sprites. The Learnable Typewriter however struggles at modelling different handwriting styles in a single dataset, and struggles at modelling characters that have different representations. Such aspects have important applications in paleography that is the study of ancient writings. In this project, we present two modifications of the original architecture that enable to improve its performances on these tasks, which we further demonstrate by an evaluation on the challenging handwritten dataset Alto-word [18] extracted from medieval manuscripts.

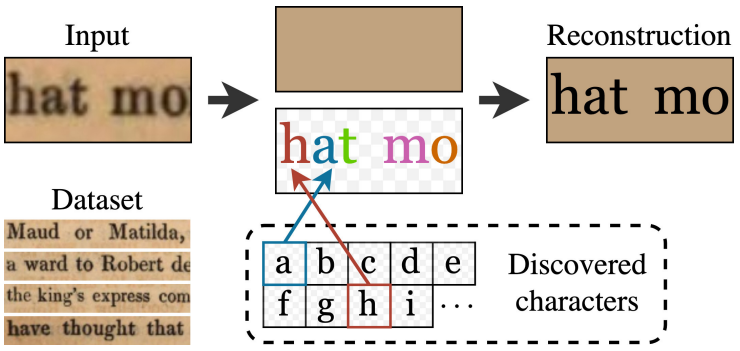


Figure 1: Overview of the Learnable Typewriter. Source [15].

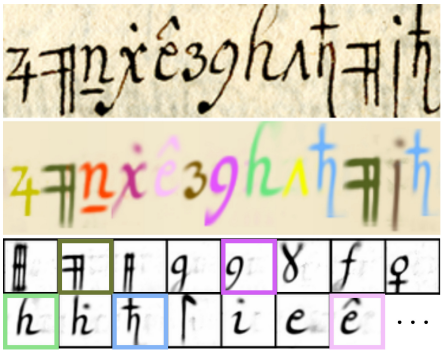


Figure 2: Results on the ciphered dataset Copiale [8]. An input text line (top) is reconstructed by transforming and placing automatically extracted characters (bottom), which enables a reconstruction segmented by character (middle). Source [15].

1. Introduction

1.1. Context and existing work

Context The purpose of this project is to develop Deep Learning methods that use little to no supervision for text recognition and handwriting analysis in images in an interpretable way on challenging handwritings. It follows an internship at ENPC under the supervision of Pr. Mathieu Aubry that was part of a collaboration with historians for automatic data analysis for paleography, that is the study of ancient writings.

Text recognition and handwriting analysis in images is challenging due to the high variability in the aspect of a same handwritten character. Interpretability is often seen as a major bottleneck of Deep Learning methods whose inference process is described as "black box".

The Learnable Typewriter This project builds on the existing model The Learnable Typewriter (LTW) [15] that was developed during the internship in collaboration with Pr. Aubry's group. This Deep Learning model performs character analysis and reconstruction in images of text lines with the purpose of dealing with challenging handwritten documents while being interpretable. Specifically, it builds on unsupervised multi-object segmentation approaches [13, 16] that learn a set of image prototypes called *sprites* to reconstruct the images. The LTW jointly extracts recurrent symbols (i.e. characters) from the dataset while training a network to reconstruct the input images of text lines by transforming and positioning the extracted characters on a background canvas. This process is illustrated figure 1. This approach is document-specific in the sense that the model is trained on a given set of documents and does not aim at generalizing at documents that are out of the scope of the corpus.

The model can perform in an unsupervised way, or can be guided by a widely available weak supervision, that are the transcriptions of the text lines. Intuitively, knowing the sequence of characters in a text line can indeed guide the selection of of the extracted characters to use to reconstruct the text lines.

Advantages and applications The Learnable Typewriter meets enables analysis of the text line images at the character level while being interpretable. Using the characters extracted from the dataset by transforming them and placing them indeed allows to know exactly which letter have been used to reconstruct a given text line. This is particularly useful in the case where there is an ambiguity on which sprite is used to reconstruct a character. It can also explain the bad reconstruction of a character, for instance if it has not been detected and modelled in the sprites. The model may then try to mix several extracted characters to reconstruct it, leading to a bad reconstruction. It is totally transparent on those predictions by offering an easy access to the sequence of transformed sprites used to reconstruct a text line, which enables to understand failure cases.

The combination of extracted characters and segmented reconstruction represented figure 2 enables an automatic analysis of the dataset as a whole through the study of the sprites. It also allows analysis at the character scale by the study of the reconstructions of text lines. To do so, it can perform in a fully unsupervised way or in a weakly-supervised way that requires limited annotations since transcriptions of the documents can often reasonably be obtained. In this setting, that we use for the project, it does not need a knowledge of the language (see figure 2 with the ciphered document), it only needs the sequence of characters.

1.2. Problem description

The existing model developed faces limitations on challenging handwritten datasets.

Variability of character representation First, such datasets from historical documents can have a lot of variability in the representation of a same character. A same character in the transcriptions can have several version in the dataset, as shown figure 3:

Figure 3: Two representations of the character 'd' in the dataset Alto-word [18].

This leads to difficulties in the weakly-supervised case where we use a single sprite to model a given character. This means that a single sprite has to learn the different representations of a character and average them, which leads to noisy sprites and bad reconstructions. Rarest representations can even not be captured in the sprites, leading to the use of the right sprite with the wrong appearance to reconstruct a this character.

Variability in the writing style Another challenging case is the case of historical documents with different handwritings. Figure 4 illustrates this case. In these samples, the last 't' has the same representation in both samples but still displays a different aspect due to different writing styles. The learned sprite however does not encode the style of the sample on the right. In both cases, the reconstructed sprite after transformation is similar to the learned sprite, since transformations can only modify colour and scale. The reconstructed character is therefore exactly the same in both cases despite the difference in the aspect of these 't' in the original image.

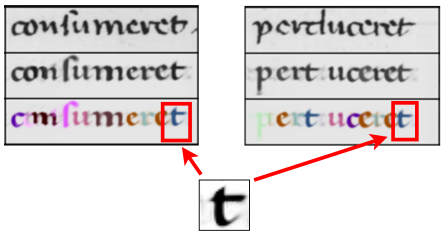


Figure 4: Samples from the dataset Alto-word [18] with two different writing styles. First line represents the original text line, second line the reconstructed text line using the learned sprite 't'. The third line is a segmentation per character of the reconstruction.

We would like to be able to transform the learned sprites in a way that enables to adapt to each writing style. This could be useful to have a better reconstruction, but also to have characterise the transformations used to model a given handwriting. Having an automatic analysis of the handwriting style would be useful for historians in paleography, which was the first motivation of the project.

This could also lead to better results on challenging irregular handwritings. Being able to transform sprites in a refined manner could enable to adapt to each occurrence of a character, even if they have the same writing style.

1.3. Outline and contributions

To sum up, our contributions for this project are:

- contribution to the last developments of the Learnable Typewriter [15] for publication and submission at CVPR'23. Contribution to the writing of the submission,
- a generalization of the use of more than one sprite per character, which we demonstrate to effectively model different representations of the same character,
- an augmentation of the transformation pipeline to better model the different handwriting styles,
- an evaluation of the aforementioned modifications on the challenging handwritten dataset Alto-word [18] and its subsets.

After a literature review, we detail the architecture of the Learnable Typewriter on which we build. We will also present our modifications in the 'Methodology' section, and then evaluate them in the 'Experiment' section.

The implementations as part of this project can be found at: https://github.com/JulienGaubil/MVA/tree/main/Deep-Learning.

2. Related works

Text recognition The tasks of Optical Character Recognition (OCR) and Handwritten Text Recognition (HTR) have been amongst the first to demonstrate successful applications of Deep Learning models [9, 10]. More recently, discriminative approaches that use a Convolutionnal encoder followed by Bidirectionnal LSTM trained with the Connectionnist Temporal Classification (CTC) loss have been proposed [1, 2]. This CTC loss [4] aligns unsegmented inputs with their label and has important applications at the intersection of Computer Vision and Sequence Learning. One of the first applications to HTR has been demonstrated in [5] by training Multidimensional Recurrent Neural Networks with this loss function as an objective.

Closer to the approach of the Learnable Typewriter are approaches that use a set of handcrafted image prototypes/sprites to perform matching in the input text lines [17, 19]. ScrabbleGAN [3] also uses a Generative approachto synthetize handwritten text images with specific writing styles in a semi-supervised way. It however doesn't enable an analysis at the character level.

Sprite-based unsupervised image decomposition The Learnable Typewriter is built over two different lines of work that learn to model object in images with sprites, namely MarioNette [16] and DTI-Sprites [13]. These two approaches decomposes images in prototypes of objects called sprites, that are modeled by RGB images with an additional transparency channel. DTI-sprites directly learns sprites in the image space while MarioNette learns sprites in a latent space and train a generator to generate the RGBA sprites from the latent sprites. Both methods then use simple transformations to adapt the learned sprite to each specific instance of object in the image. To do so, they predict the parameters of parameterized transformations applied in a differentiable way by a Spatial Transformer Network [7]. The transformation pipeline is quite simple, but more complex transformations are used in [12] on which is based DTI-Sprites.

3. Methodology

3.1. The Learnable Typewriter

In this subsection, we will describe the model Learnable Typewriter [15] that was developed as part of the internship and at the beginning of this project for the CVPR submission. A set of elements $\{a_1, ..., a_n\}$ will be denoted as $a_{1:n}$ for simplicity.

Description The purpose of the Learnable Typewriter is, given a dataset of images of text lines, to learn the shape of the characters contained in the text lines and to learn a Deep Neural Network that can predict the transformations to apply to these learned characters to generate a given input text line. It is therefore a generative analysis by synthesis approach, that uses unsupervised image decomposition to jointly learn a representation of the characters called sprites, and the network. Specifically, the network learns both to predict transformations to apply to the learned sprites and how to position them on a background canvas to reconstruct the input text lines. Although it can be trained in an unsupervised way, we will introduce the weakly-supervised training that uses the transcriptions of the text lines as a supervision to guide the reconstruction.

Overview An input image of text line of size $H \times W$ is fed to a CNN encoder that outputs T feature vectors $f_{1:T}$ associated to uniformly spaced locations in the original image. The feature vectors are then fed to the Typewriter module, described below, that outputs T canvas $o_{1:T}$ of size $H \times W$ of transformed sprites positioned accordingly to the uniformly spaced locations, see figure 5a. The features are also fed to a network b_θ that uses the average feature vector to predict a single color for the opaque (0 transparency channel) background canvas o_{T+1}. The ordered layers $o_{1:T+1}$ are then merged using alpha-composition (see below) to form the reconstructed image $\hat{x}$. The image is iteratively defined by the following alpha-composition rule:

$$\hat{x}_t = o_t^\alpha \, o_t + (1 - o_t^\alpha) \, \hat{x}_{t-1} \quad \forall 1 \leq t \leq T \tag{1}$$

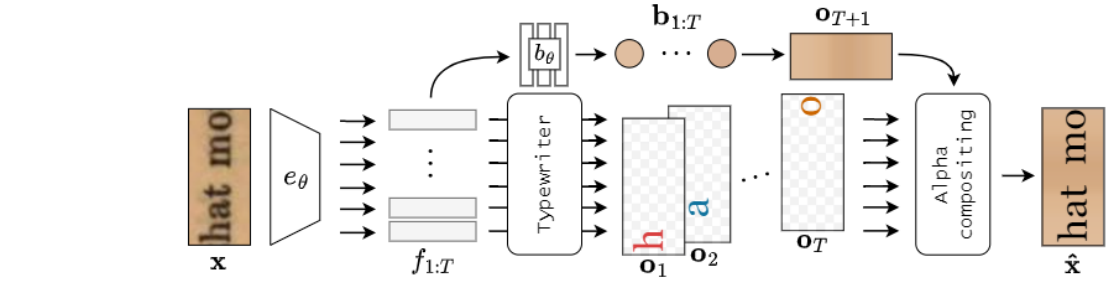
where o^α denotes the transparency channel of a layer o, and the initialization $\hat{x}_0 = o_{T+1}$. The final reconstruction is $\hat{x} = \hat{x}_T$. This equation shows that, at a given step t, the RGB channels of the layer o_t are masked by the transparency mask o_t^α while already composed layers $j < t$ are masked by the complementary transparency mask $1 - o_t^\alpha$ to obtain the reconstruction $\hat{x}_t$ at step t.

Sprites In the weakly-supervised case in which we set, we have access to the transcriptions of the images i.e. to the set of K characters in the dataset. We can therefore initialize exactly K sprites and set an explicit matching between sprites and characters. The Learnable Typewriter contains K sprites $z_{1:K}$ that are learned latent codes. An U-Net generator g_θ takes as an input latent sprite z_k and generates the transparency channel of an image $s_k = g_\theta(z_k)$. Along with an uniform color for the channel RGB, it will model a character. An empty sprite z_{K+1} is added to model the case where we don't want to place a sprite on a given canvas/layer.

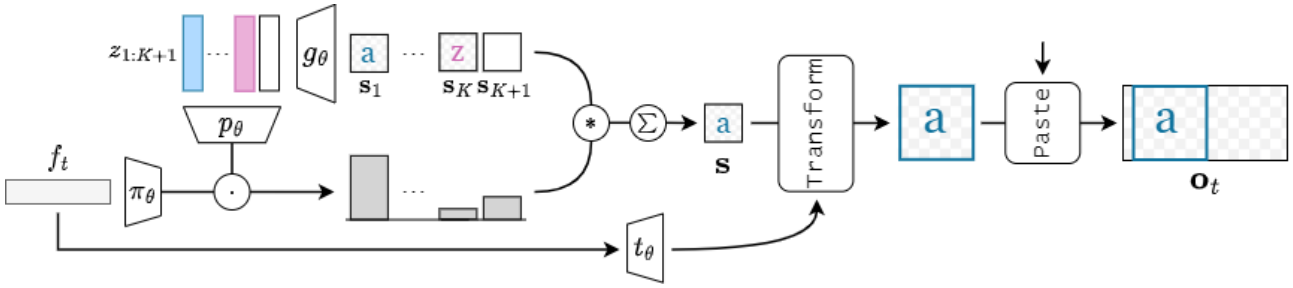
Typewriter Module We now describe the Typewriter module with a given feature vector f_t as input. It outputs a canvas o_t that is a RGB image of size $H \times W$ with an additional transparency channel. This module is represented figure 5b. The latent sprites $z_{1:K}$ are projected in a feature space by learned projection layers p_θ, which gives $p_\theta(z_{1:K}) \in \mathbb{R}^{D \times K}$. A sprite z_{K+1} is added to model the blank sprite in the feature space. The feature vector f_t is projected in the same feature space by learned projection layers, π_θ. Once in the same space, the similarity between the feature vector f_t and each of the latent sprites $z_{1:K+1}$ is computed by performing a dot product. After normalization, probabilities $p_{1:K+1}(f_t)$ over sprites are obtained by applying a softmax to the computed similarities. The overall operation is described below:

$$p_{1:K+1}(f_t) = \text{softmax}\Big(\frac{1}{\sqrt{D}} p_\theta(z_{1:K+1}) \, . \, \pi_\theta(f_t)\Big) \in \mathbb{R}^{K+1} \tag{2}$$

A weighted sprite s is then obtained by weighting the sprites $z_{1:K+1}$ (including the blank sprite $K+1$) with the computed probabilities $p_{1:K+1}(f)$: $s = \sum_k p_k s_k$. A network t_θ then predicts the parameters of a parameterized transformation to be applied to the



(a) Overall pipeline of The Learnable Typewriter. Source [15].



(b) The Typewriter Module. Source [15].

sprite s. t_θ takes as an input the feature vector f_t and outputs parameters $\beta_t \in \mathbb{R}^6$. Two of these parameters define a spatial translation that enable a refined positioning for the sprite s in the canvas o_t: a coarse positioning is defined by a window in the canvas positioned at the position attached to the canvas t. The predicted translation defines the position of the sprite s inside this window, refining its position to match exactly the position of a character in the input text line. One parameter is dedicated to scaling, and the last three parameters adapt the color of the sprite. These parameters β_t, as well as the sprite s are then fed to a transformation module (Spatial Transformer Network [7]) that outputs the canvas o_t. This canvas contains the sprite s transformed and positioned according to the prediction. The whole pipeline is differentiable and enables to train the model end-to-end.

Losses The loss used in the weakly-supervised setting (when using the transcriptions of the text lines) is composed of two terms. The first term is a reconstruction loss between the input images x and the reconstructed images $\hat{x}$. It pushes the the model to transform and place the sprites well to reconstruct the input text lines, and pushes sprites to model the characters in the dataset. The second term is the Connectionist Temporal Classification loss [4] that incorporates the weak supervision (transcriptions Y) in the training process by pushing the predicted probabilities $P = p_{1:K+1}(f_{1:T})$ to match the sequence of characters in the transcription. This enables to ensure that the sprite sequence matches the transcription sequence.

In the weakly supervised setting, we initialize one sprite per character with an explicit matching between them, and we use the blank sprite z_{K+1} as the separator for the CTC loss. The CTC loss pushes a sprite to effectively capture its associated character. The total loss is presented above, λ being an hyperparameter.

$$\mathcal{L}(x, \hat{x}, P, Y) = \|x - \hat{x}\|_2^2 + \lambda \, \mathcal{L}_{CTC}(P, Y) \tag{3}$$

3.2. Two sprites per character

The first problem that motivated this project was to model the different representations of a single character, see figure 3. A simple approach to solve this problem is to use more than one sprite per character. In this subsection, we describe how to use two sprites per character, which can be easily generalized to $N > 2$ sprites per character.

Formally, we now have $z_{2:2K+1}$ sprites, and the sprites z_{2k} and z_{2k+1} are associated to the k-th character. We still use the blank sprite, z_{2K+2}. Similarly to the original Learnable Typewriter, given an input feature f_t, the Typewriter module predicts probabilities for each sprites $p_{2:2K+1}(f_t)$, see equation 2. As represented figure 6, we then sum the probabilities p_{2k} and p_{2k+1} associated to a same character to obtain a single probability per character, $\tilde{p}_{1:K+1}(f_t)$, $\tilde{p}_k = p_{2k} + p_{2k+1}$ $\forall 1 \leq k \leq K$. The probabilities $\tilde{p}_{1:K+1}(f_t)$ can then be used in the CTC loss along with the transcriptions as in the original architecture. The weighted sprite s obtained will be the weighted sum of all the sprites, $s = \sum_{k=2}^{2K+2} p_k s_k$.

Using this sum of probabilities naturally enables to have a gradient backpropagated from the CTC loss to the latent sprites in the same way as in the original Learnable Typewriter. Indeed, the gradient is backpropagated from the CTC loss to the probabilities $\tilde{p}_{1:K+1}(f_t)$. After backpropagating the gradient from the probabilities $\tilde{p}_{1:K+1}(f_t)$ to the probabilities $p_{2:2K+1}(f_t)$ through the sum, the gradient can then be backpropagated to the sprites and the probabilities prediction network in the same way as the original Learnable Typewriter. An alternative solution that would consist in using a hard selection by only selecting the sprite that has the highest probability for each character (both for the CTC loss and the reconstruction) would lead to sprites never being selected, therefore never receiving gradient and being left non-optimized.

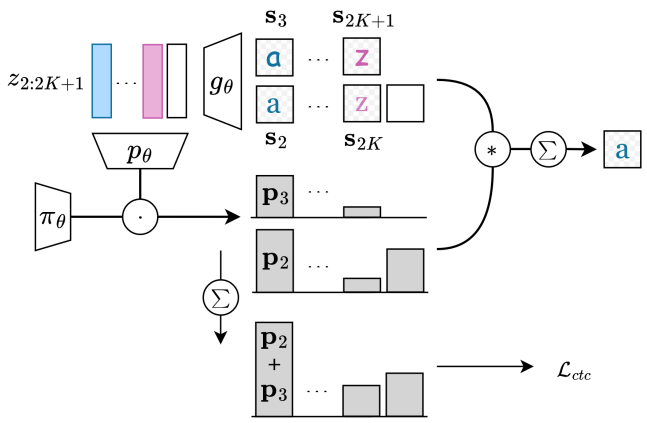


Figure 6: Modification of the Typewriter module using two sprites per character.

This method also has the advantage of being easily adaptable to the case with $N > 2$ sprites per character. It indeed only needs to sum through all the N predicted probabilities for a single character before the CTC loss, as represented in figure 6 for the case $N = 2$.

3.3. Adding transformations

The second problem was to model the transformations that characterize a handwriting style, and to have a transformation pipeline that enables to adapt to irregular handwritings. The transformation pipeline of the original Learnable Typewriter, represented on the top of the figure 7, only modifies the position and the scale of the sprite. It is therefore not able to achieve this task. The idea is to add transformations able to model such variability to the transformation pipeline. To do so, similar to [12], we add a Thin Plate Spline transformation at the end of the pipeline (see the bottom part of figure 7).

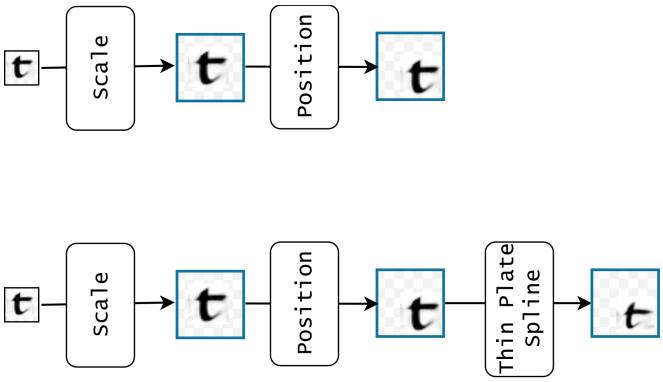


Figure 7: Original transformation pipeline (top) and augmented transformation pipeline (bottom).

The Thin Plate Splite is a data interpolation and smoothing method. In the 2D case, it uses K control points $(x_k, y_k)_{1:K}$, usually sampled in a grid in the image and interpolates exactly through these points while minimizing a regularization energy that is defined as the squares of the second derivatives:

$$I[f(x,y)] = \sum_{k=1}^{K} \|(x_k, y_k) - (x, y)\|^2 + \int_{\mathbb{R}^2} \left[\left(\frac{\partial^2 f}{\partial x^2} \right)^2 + \left(\frac{\partial^2 f}{\partial y^2} \right)^2 + 2 \left(\frac{\partial^2 f}{\partial x \partial y} \right)^2 \right] d_x d_y$$

In 2D, a Thin Plate Spline is often interpreted as the displacement of one of the coordinates (x or y) of the image plane. We therefore need two Thin Plate Splines f_x, f_y to have a 2D coordinate transformation/warping that will be defined by $(x', y') = (f_x(x, y), f_y(x, y)) \in \mathbb{R}^2$.

The closed-form solution of the previous equation in 2D has the following form:

$$f(x,y) = a_1 + a_x x + a_y y + \sum_{k=1}^{N} w_k \, U\left(\|(x,y) - (x_k, y_k)\|_2\right)$$

where U is the radial basis kernel $U(r) = r^2 log(r)$, $w_{1:K}$ is a set of mapping coefficients, and the coefficients (a_1, a_x, a_y) represent the linear plane that best approximates the mapped points $f(x_k, y_k)$. Each TPS has $K + 3$ parameters: the $w_{1:K}$ and the a_1, a_x, a_y. A TPS transformation/warping is therefore defined by $2(K + 3)$ parameters. Similar to the Learnable Typewriter, these parameters will be predicted by the predictor t_θ from the features $f_{1:T}$.

4. Experiments

4.1. Implementation and training details

We evaluate our approach on the dataset Alto-word with lines of height $H = 64$, and sprites of size $H/2 \times H/2$. We compare ourselves to the original method only (for comparison with other methods, see [15]). In the default weakly-supervised setting, we use as many sprites as characters in the dataset. We use the same training settings as the default Learnable Typewriter [15]: the optimizer AdamW [11] with a learning rate of 10^{-4} (with a weight-decay of 10^{-6} for the encoder). Similarly, the encoder e_θ used is a truncated ResNet-32 [6], and the generator is a U-Net [14]. The projection layers p_θ, π_θ are simple linear layers with layer normalization. The networks b_θ, p_θ used to predict the transformation parameters are 2-layers MLPs. On the full Alto-word, we train during 200 epochs with a batch size of 16.

For transformations, we adopt the same curriculum learning strategy as in [12] by iteratively training the network until convergence before adding another transformation to the pipeline and then keep training with the new pipeline. We train our models on a single GPU GTX 1080. The training time for 200 epochs with a batch size of 16 on Alto-word for the original architecture is around 8 hours.

4.2. Datasets & Metrics

Dataset Alto-word The dataset Alto-word [18] comes from 101 digitized images of dated medieval manuscripts and contains annotations at the word level. Each original image corresponds to a

single document that typically contains between 6 and 25 lines of text. The dataset contains a total of 12927 images of words segmented manually along with their transcription. The transcriptions include 99 different characters. Annotations also detail the writing style for each word, the three most common writing styles being Textualis, Praegothica and Humanistique. We create subsets for these writing styles to reduce the variability in the writing in these subsets. Alto-word is challenging due to an important variability inter and intra-handwriting styles. It also exhibits degradation, and characters from other lines can appear on the images.

Metrics We evaluate our models with the Character Error Rate metric. This metric counts the number of modifications required to transform a predicted sequence into a ground-truth sequence. Formally, it is defined as:

$$CER(s, \hat{s}) = \frac{S + D + I}{|s|} \tag{4}$$

where S is the number of substitutions, D is the number of deletions and I is the number of insertions required to match the ground-truth sequence $\hat{s}$. In our experiments, at every position in the image, the sprite that has the highest predicted probability will be selected. The sequence of predicted sprites obtained can then be translated in a sequence of predicted characters thanks to the explicit matching between sprites and characters. The CER will be applied to this sequence of predicted characters and will use the ground-truth transcription as the ground-truth sequence. The lower is the CER, the better is the model on this metric. Please note that although this metric is useful to evaluate the LTW and compare it to other models, generalizing on a test set and improving the CER is not the main purpose of the model.

4.3. Results

4.3.1 Experiments with two sprites per character

We first experiment the architecture using two sprites per character by training on the dataset Alto-word with this architecture and comparing to the baseline, the original LTW. The impact of this modification in terms of size of the model is negligible since the baseline contains 99 sprites and 3.863 millions parameters against 198 sprites and 3.964 millions parameters for the architecture when using two sprites per character. This has also little impact on the training time, 7h18min for the baseline against 7h38min with two sprites per character.

The sprites learned in both settings are presented figure 8 and figure 9. We highlight in red the sprites learned with the original architecture that are missing in the two sprites setting. We also highlight in green groups of two sprites that represent two versions of the same character in the two sprites setting.

As expected, adding more sprites did not harm the sprites learning since almost all the sprites learned in the original setting are also learned when using two sprites per character, apart from two sprites that represent very rare characters. Furthermore, using two sprites per character enabled to capture variations in the representation of characters, as highlighted in green. Several characters ('c', 'd', 'e', 's', 't') have two different versions encoded in

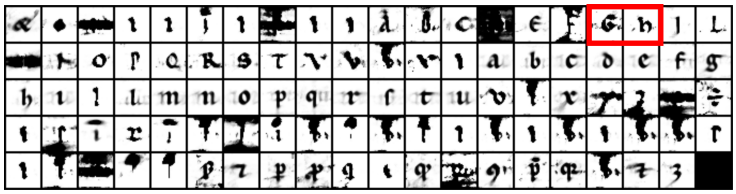


Figure 8: Sprites of the original LTW on Alto-word [18].

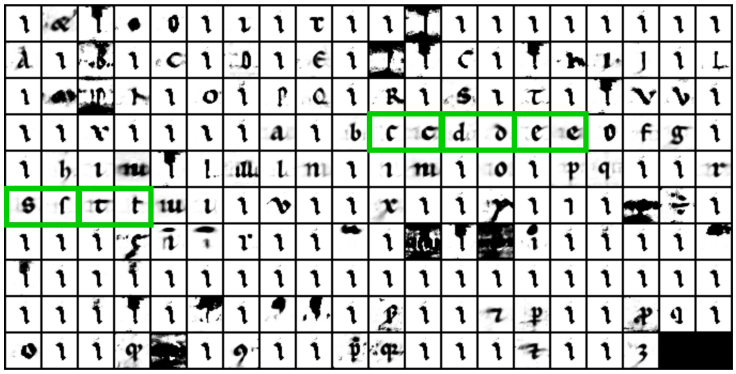


Figure 9: Learned sprites on Alto-word [18] when using two sprites per character.

the sprites. It can either be caused by different representations of the character like for the 's and the 'd', or by slight variations due to different writing styles like the 'e' or the 't'. Most characters only have one version represented, their second sprite not being optimized. As a result, sprites that represent characters that have two different representations do not need to capture all these representations in one sprite by averaging them. This leads to clearer sprites overall with less noise.

This improvement leads to better qualitative results but do not have an impact on the quantitative evaluation: the CER on the test set at the end of the training is close in the the two-sprites setting (28.2%) and in the baseline (28.1%). This is expected as the CER is influenced by the sequence of predicted sprites. As long as a character is modelled in the sprites, even if it encodes several versions, it may not prevent the model from using the right sprite and therefore to predict the right sequence. The visual aspect of the reconstruction will however be impacted since the sprite will be noisy or will not represent the right version of the character.

4.3.2 Experiments with more transformations

We now evaluate the pipeline augmented with the Thin Plate Spline transformation by training the baseline on Alto-word and its subsets with specific handwritings. After convergence, we train the models with the TPS transformation added to the pipeline. We first visualize results on the subset Humanistique that contains an highly irregular handwriting. Figures 10 and 11 illustrate the impact of the Thin Plate Spline transformation in such context. The first line contains the original text line, the second line contains the reconstruction and the third contains the segmentation of the reconstruction.

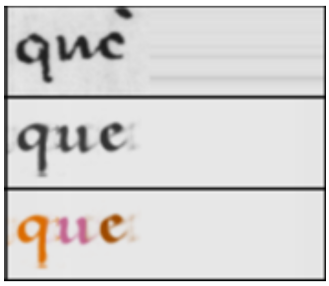


Figure 10: Reconstruction with the baseline pipeline.

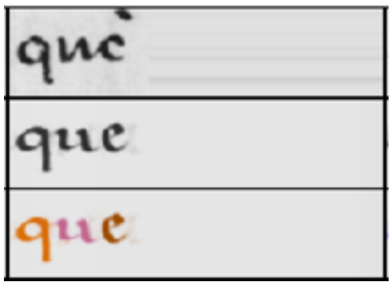


Figure 11: Reconstruction using TPS transformation.

We observe on the second line that using the TPS transformations enables slight deformations on the sprites (for instance the 'u') that enable to better match the instance of the character in the input image. These deformations give an overall aspect that is less rigid and imitate an handwritten style.

We further demonstrate the ability of the TPS transformation to closely match the variability in the aspect of a character in the figures 12 and 13. Figure 12 shows an example of two different versions of a 't' in the same subset, Alto-word Humanistique. The 't' on the left is thicker and upright while the 't' on the left is thinner and sloping.

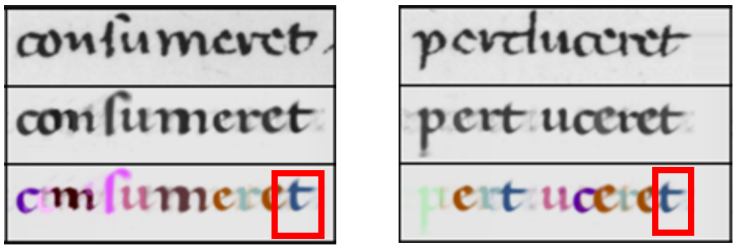


Figure 12: Reconstruction with the baseline pipeline.

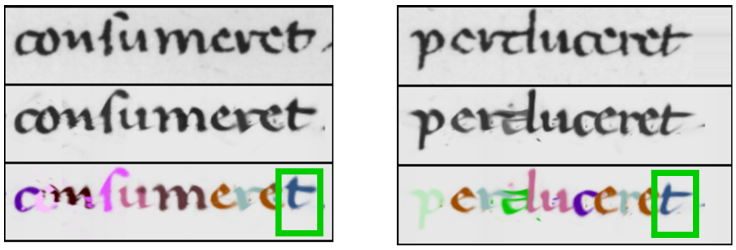


Figure 13: Reconstruction with the augmented pipeline using TPS transformation.

We observe that with the baseline pipeline, a similar sprite is composed in both samples to reconstruct the 't' (highlighted in red in the segmentation) despite the difference of their aspect in the original image cases. This is due to the the lack of transformation in the pipeline that is able to take into account this variability. However, when using the TPS transformation, the thickness, the deformation and the slope are indeed adapted to each sample (highlighted in green in the segmentation). The sprites nevertheless become a bit noisier when adding the TPS transformation which leads to small grains in the reconstruction.

Adding the TPS transformation to the pipeline also improves the Character Error Rate , as shown in table 1. Training the model with TPS after convergence of the baseline enables to improve the Character Error Rate by around 3% on three subsets of Alto-word (full dataset, Textualis, Humanistique). The Praegothica subset is the only one where adding the TPS transformation hurts the CER.

Method	Subset	CER (%)	Epochs
Baseline	Full	28.1	200
TPS	Full	**25.9**	200+200
Baseline	Textualis	32.6	400
TPS	Textualis	**30.6**	400+400
Baseline	Praegothica	**33.3**	1200
TPS	Praegothica	37.2	1200+1200
Baseline	Humanistique	38.7	3600
TPS	Humanistique	**35.0**	3600+3600

Table 1: Character Error Rate on the subsets of the Alto-word dataset [18] (entire dataset denoted as Full).

4.4. Discussion

Limitations The main limitations in these experiments lie on the use of additional transformations. It indeed leads to sprites that are noisier which in turn impacts the quality of the reconstruction. It is also difficult to quantify the impact of the additional transformations apart from visual aspect in the reconstructions. The curriculum learning strategy is also hard to design and some parameters such as the choice of the learning rate or number of epochs can have an important impact on the results. Due to limited computational resources, we restrained ourselves to default values for the optimizer and to a fixed number of epochs for both stages, but a thorough investigation of those parameters might enable to significantly improve the results.

Future works Several other experiments can be done to further illustrate the interest of using more than one sprite per characters. First, using more than two sprites per character could be easy and interesting, although even on the challenging Alto-word, few characters have more than two significantly different representations on the dataset. Training with one sprite per character then using it as an initialization for a model that uses two sprites per character with the two sprites initialized by the previously learned sprite could also improve the results. It could indeed enable to have couples of sprites that present slight variations, while in our experiments only sprites that represented characters that have very different representations had their two sprites optimized. Most characters therefore had one sprite left non-optimized.

Adding more transformations to the transformation pipeline could enable a better modelling of the variability in the handwriting. For instance, using the Morphological transformation introduced in [12] could be interesting. Evaluating models trained with TPS on subsets of Alto-word with specific handwritings on a handwriting classification task and comparing it to the original LTW could also enable to assess the capacity of the model to better model the transformation that characterize a specific handwriting when using the TPS transformation.

5. Conclusion

Through this project, we have improved the performances of the model Learnable Typewriter on a challenging handwritten dataset. The use of an augmented transformation pipeline and the use of several sprites per characters proved through our evaluation to better model the variability in the aspect of the characters in such datasets. This could enable a better automatic analysis of historical documents and spread the use of this model to paleography applications, which was the purpose of this line of work.

References

[1] Théodore Bluche and Ronaldo Messina. Gated convolutional recurrent neural networks for multilingual handwriting recognition. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 646–651. IEEE, 2017. 3

[2] Arthur Flor de Sousa Neto, Byron Leite Dantas Bezerra, Alejandro Héctor Toselli, and Estanislau Baptista Lima. Htr-flor: a deep learning system for offline handwritten text recognition. In *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 54–61. IEEE, 2020. 3

[3] Sharon Fogel, Hadar Averbuch-Elor, Sarel Cohen, Shai Mazor, and Roee Litman. ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation. *arXiv:2003.10557 [cs]*, Mar. 2020. 3

[4] 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*, ICML '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery. 3, 4

[5] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in neural information processing systems*, 21:545–552, 2008. 3

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[7] Max Jaderberg, Karen Simonyan, and Andrew Zisserman. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015. 3, 4

[8] Kevin Knight, Beata Megyesi, and Christiane Schaefer. The Copiale Cipher. In *Proceedings of the ACL Workshop on Building and Using Comparable Corpora*, pages 2–9, 2011. 1

[9] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 3

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 5

[12] Tom Monnier, Thibault Groueix, and Mathieu Aubry. Deep Transformation-Invariant Clustering. In *NeurIPS*, Oct. 2020. 3, 5, 7

[13] Tom Monnier, Elliot Vincent, Jean Ponce, and Mathieu Aubry. Unsupervised Layered Image Decomposition into Object Prototypes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8640–8650, Apr. 2021. 1, 3

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv*, abs/1505.04597, 2015. 5

[15] Ioannis Siglidis, Nicolas Gonthier, Julien Gaubil, Tom Monnier, and Mathieu Aubry. The learnable typewriter: A generative approach to text line analysis. *arXiv:2302.01660 [cs.CV]*, Jan. 2023. 1, 2, 3, 4, 5

[16] Dmitriy Smirnov, Michael Gharbi, Matthew Fisher, Vitor Guizilini, Alexei A. Efros, and Justin Solomon. MarioNette: Self-Supervised Sprite Learning. *arXiv:2104.14553 [cs]*, Apr. 2021. 1, 3

[17] Mohamed Ali Souibgui, Alicia Fornés, Yousri Kessentini, and Crina Tudor. A Few-shot Learning Approach for Historical Ciphered Manuscript Recognition. *arXiv:2009.12577 [cs]*, Sept. 2020. 3

[18] Dominique Stutzmann. Dated and datable manuscripts: dataset, Apr. 2022. 1, 2, 5, 6, 7

[19] Chuhan Zhang, Ankush Gupta, and Andrew Zisserman. Adaptive Text Recognition through Visual Matching. *ECCV 2020*, Sept. 2020. 3

Appendix

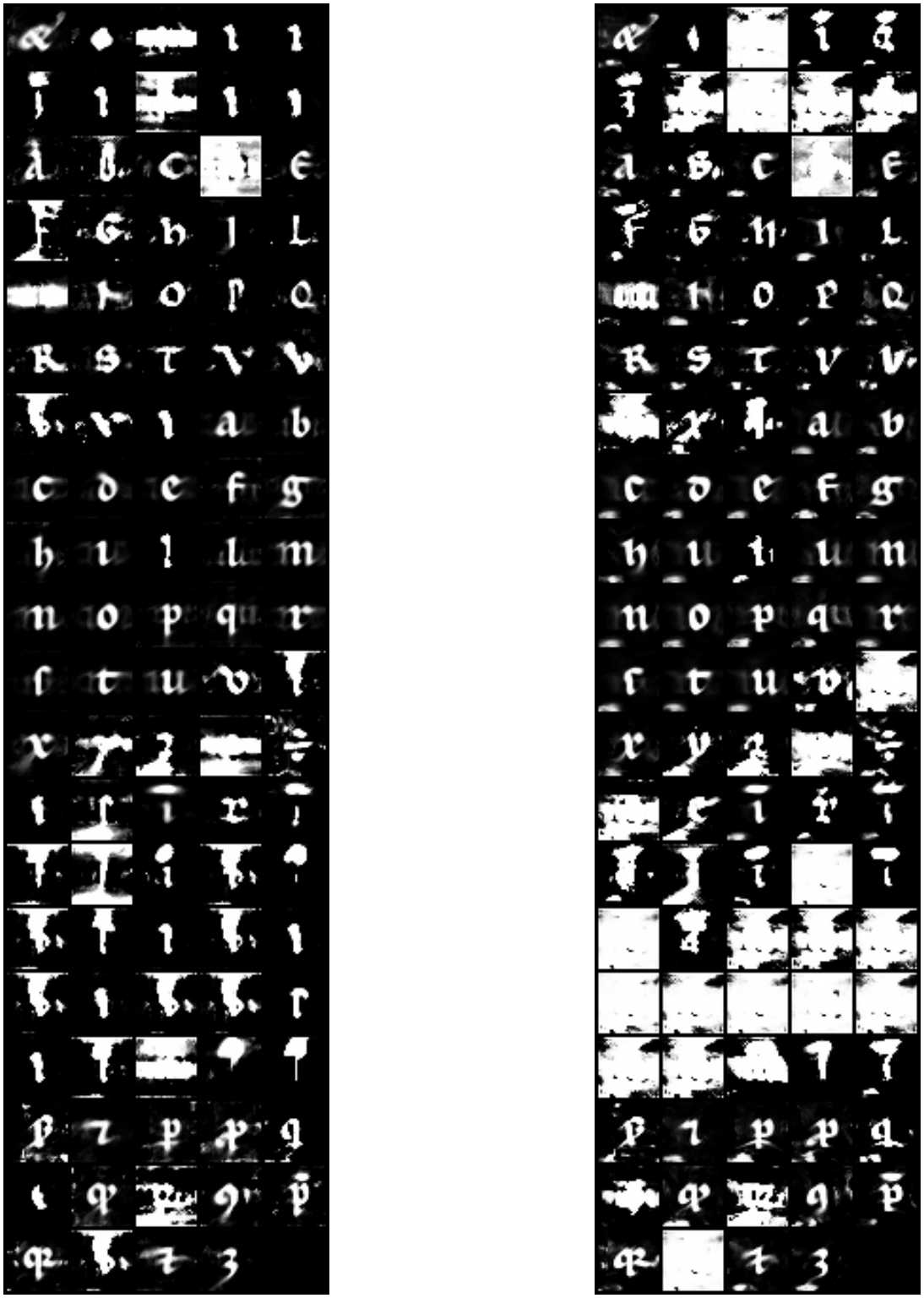


Figure 14: Sprites on **Alto-word** for baseline (left) and baseline trained during 200 more epochs with TPS transformations (right).

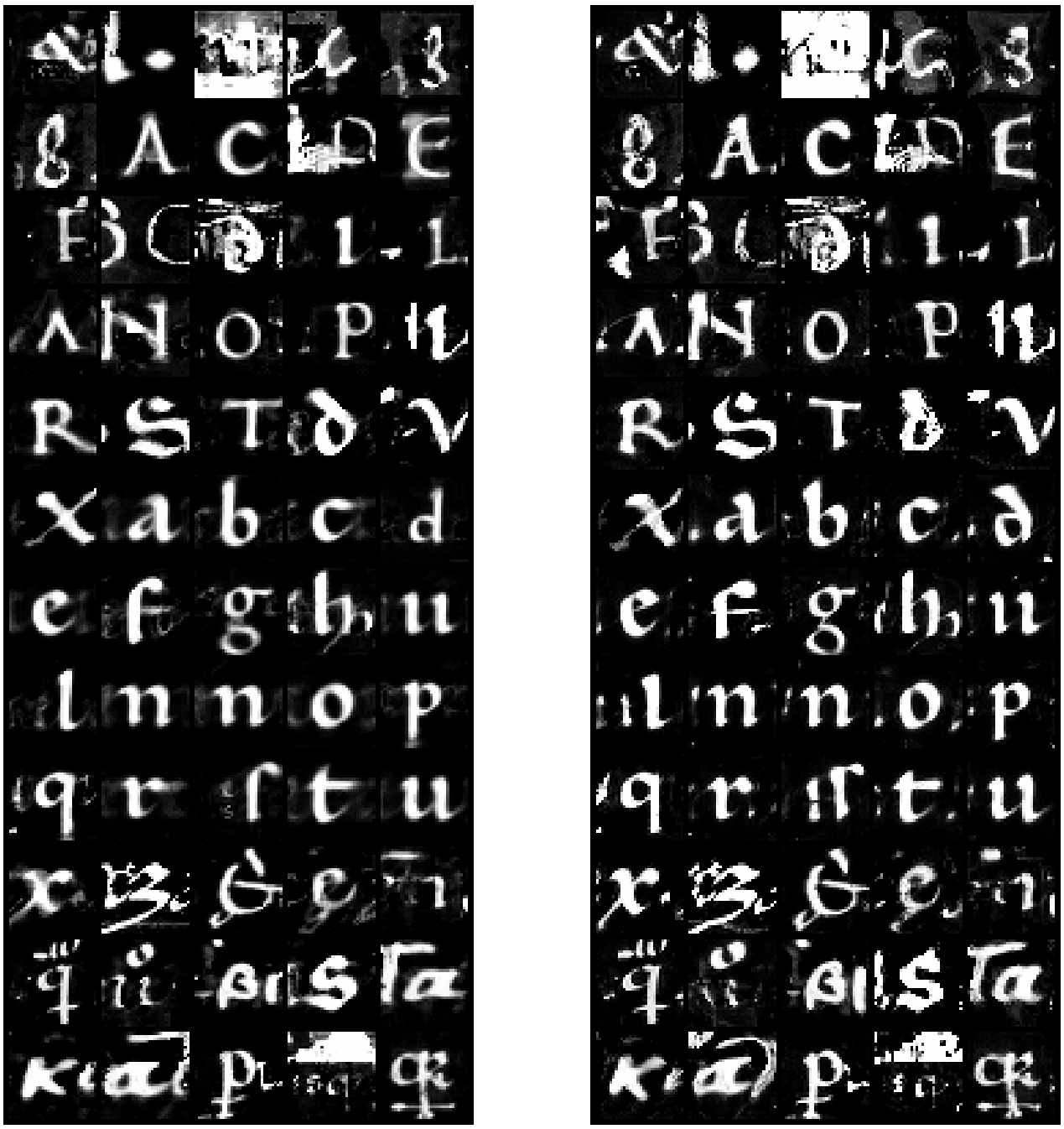


Figure 15: Sprites on **Humanistique** subset of Alto-word for baseline (left) and baseline trained during 3200 more epochs with TPS transformations (right).

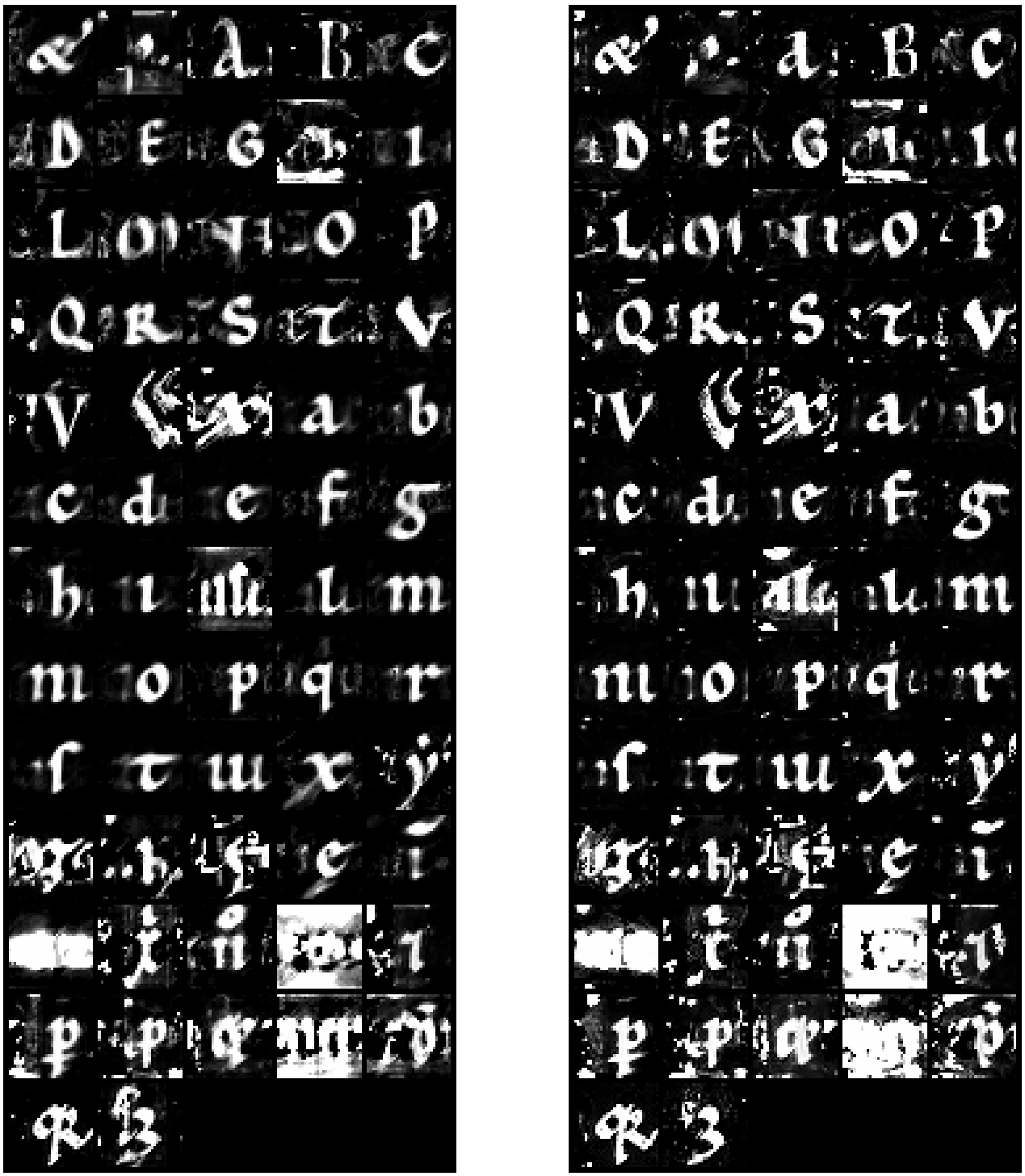


Figure 16: Sprites on **Praegothica** subset of Alto-word for baseline (left) and baseline trained during 1600 more epochs with TPS transformations (right).

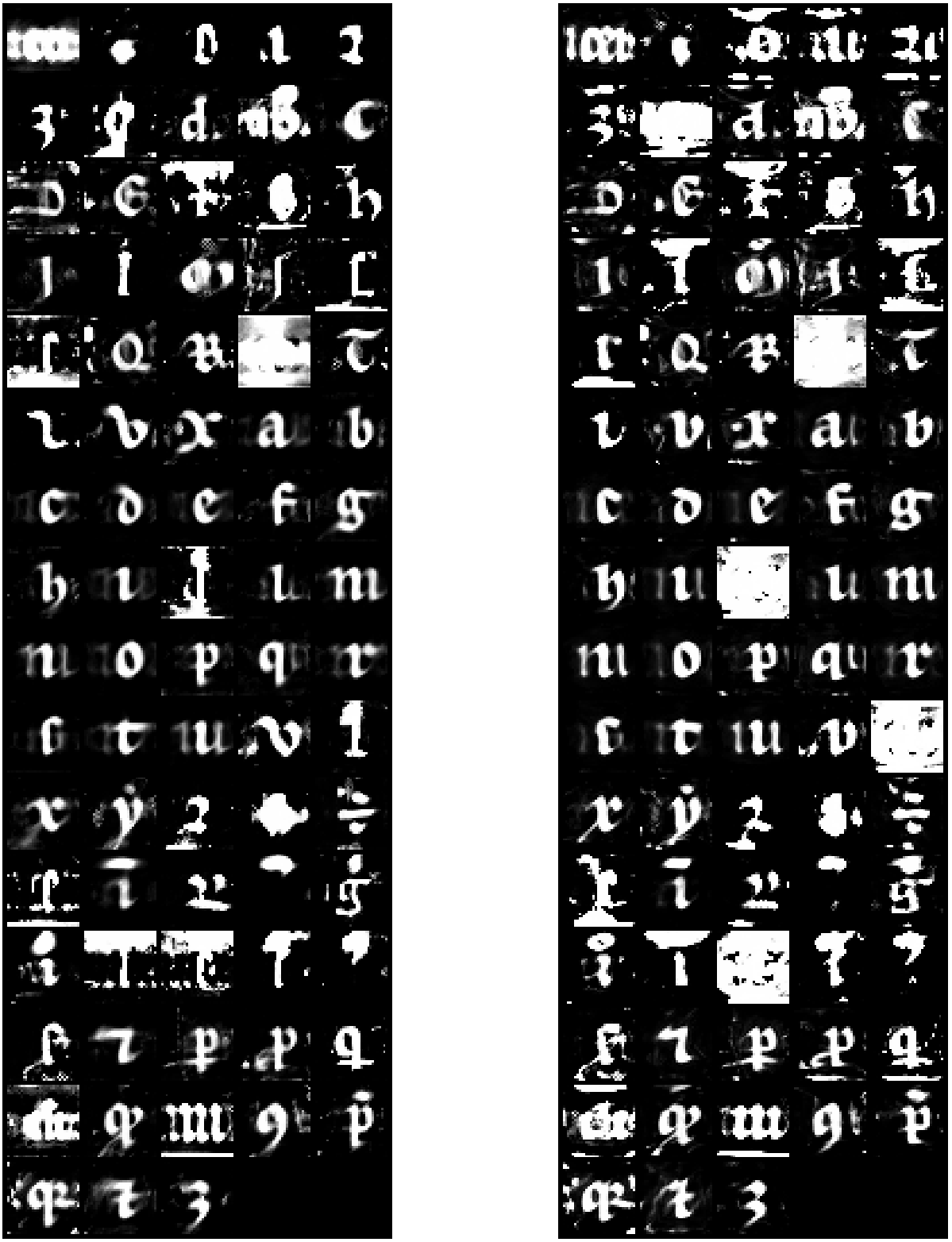


Figure 17: Sprites on **Textualis** subset of Alto-word for baseline (left) and baseline trained during 400 more epochs with TPS transformations (right).