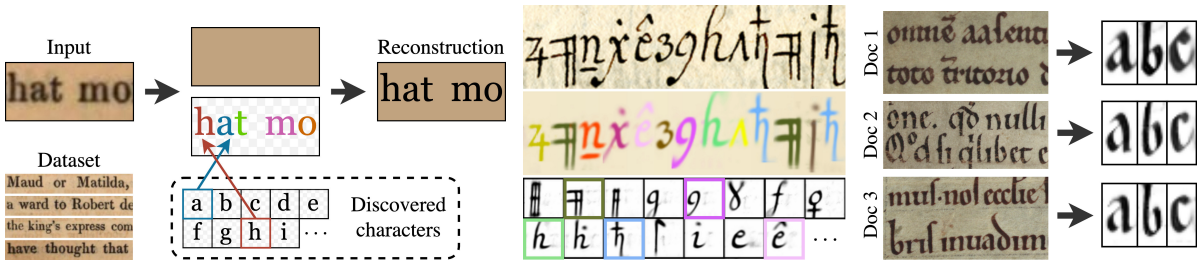
Chapter 3

The Learnable Typewriter: A Generative Approach to Text Analysis



(a) The Learnable Typewriter idea (b) Cipher analysis (c) Paleography

Figure 3.1: **The Learnable Typewriter. (a)** Given a text line dataset, we learn to reconstruct images by discovering the underlying characters. This generative approach can be used both **(b)** to analyze complex ciphers [Knight et al., 2011] and **(c)** as an automatic tool for the study of handwriting in historical documents [Camps et al., 2022].

3.1 Introduction

A popular approach to document analysis in the 1990s was to learn document-specific character prototypes, which enabled Optical Character Recognition (OCR) [Kopec and Lomelin, 1996, 1997; Xu and Nagy, 1999; Baird, 1999] but also other applications, such as font classification [Hochberg et al., 1997] or document image compression and rendering [Nolan and Filippini, 2010]. This idea culminated in 2013, with the Ocular system [Berg-Kirkpatrick et al., 2013] which proposed a generative model for printed text lines inspired by the printing process, and held the promise of achieving a complete explanation of their appearance. These document-specific generative

approaches were however overshadowed by discriminative approaches, whose sole purpose is to perform predictions, and which lead to higher performance at the cost of interpretability, e.g., [Graves and Schmidhuber, 2008; Li et al., 2023b]. In fact, [Simard et al., 2002] showed that using transformations of input exemplars while performing digit classification can allow learning with much fewer samples. Yet, acquiring such exemplars remains costly and ambiguous for more complicated tasks. Inspired by this, in this chapter, we explore how modern deep approaches enable revisiting and extending analysis by synthesis approaches for text line analysis. In particular, we demonstrate an approach that can deal with challenging examples of handwritten documents, enabling a quantitative perspective in analyzing the character morphology in historical handwriting as it is studied in the discipline of palaeography.

While discriminative approaches have been largely dominant in the current deep learning-based computer vision, a recent set of works revisited generative approaches for unsupervised multi-object object segmentation [Burgess et al., 2019; Emami et al., 2021; Greff et al., 2017, 2019b; Yang et al., 2020; Crawford and Pineau, 2019; Deng et al., 2020; Eslami et al., 2016; Jiang and Ahn, 2020; Smirnov et al., 2021; Monnier et al., 2021]. Most of them provide results on synthetic data or simple real images [Monnier et al., 2021], and sometimes demonstrate qualitative results on simple printed text images [Smirnov et al., 2021; Reddy et al., 2022]. Surprisingly, images of handwritten characters, which were notoriously used in the development of convolutional neural networks [LeCun et al., 1989, 1998] and generative adversarial networks [Goodfellow et al., 2014] were largely overlooked by these approaches.

We build on two recent sprite-based unsupervised image decomposition approaches [Smirnov et al., 2021; Monnier et al., 2021] that provide an interpretable decomposition of images into a vocabulary of small images, called sprites. These methods are trained to jointly optimize both the sprites and the neural networks that predict their position and color. Intuitively, we would like to adapt these methods so that from text lines that are extracted from any given document, they could learn sprites that correspond to each character. By adapting MarioNette [Smirnov et al., 2021] to perform text line analysis, we provide quantitative evaluation on real data and an analysis of the limitations of state-of-the-art approaches for unsupervised multi-object segmentation. We argue that text-line recognition should be used as a benchmark for this task in future work.

Because unsupervised performances are not completely satisfactory, we combine this approach with a weak supervision from line-level transcriptions. Transcriptions are widely available and easy to produce with dedicated software, e.g., [Kahle et al., 2017; Kiessling et al., 2019], and we show that this dramatically improves results, while

preserving their interpretability. Through this thesis we motivate the idea that similar weak (i.e., image-level) annotations should also be considered for future problems of image decomposition.

Contributions. To summarize, we present:

- a deep generative approach to text line analysis, inspired by deep unsupervised multi-object segmentation methods, adapted to work in both an unsupervised and a weakly supervised setting,
- a demonstration of the potential of our approach in challenging applications, particularly in ciphered documents and in palaeographic analysis,
- experiments on four different types of datasets: the printed volume of Google1000 [Vincent, 2007; Gupta et al., 2018], the historical fonts of MFGR [Seuret et al., 2023], the Copiale cipher [Baró et al., 2019; Knight et al., 2011], and two sets of historical handwritten manuscripts between the 12th-15th century [Camps et al., 2022] and Tab. F.1.

Our implementation can be found at github.com/ysig/learnable-typewriter.

3.2 Related Work

Text Analysis. Image Text Recognition, including Optical Character Recognition (OCR) and Handwritten Text Recognition (HTR), is a classic pattern recognition problem and one of the earliest successful applications of deep learning [LeCun et al., 1989, 1998]. The mainstream approaches for text line recognition rely on discriminative supervised learning. Typically, a Convolutional Neural Network (CNN) encoder will map the input image to a sequence of features, and a decoder will associate them to the ground truth, e.g., through a recurrent architecture trained with a Connectionist Temporal Classification (CTC) loss [Graves et al., 2006; Graves and Schmidhuber, 2008; Puigcerver, 2017; Bluche and Messina, 2017; de Sousa Neto et al., 2020], or a transformer trained with cross entropy [Kang et al., 2022; Li et al., 2023b].

More related to our work, ScrabbleGAN [Fogel et al., 2020] proposed a generative adversarial approach for semi-supervised text recognition, but their method is neither able to reconstruct an input text line nor to decompose it into individual characters. Also related are approaches that use already annotated sprites (referred to as exemplars or supports) to perform OCR/HTR in common fonts [Zhang et al., 2020], ciphers [Souibgui et al., 2020], and Cuneiform [Mikulinsky et al., 2025], by matching them to text lines. Recent unsupervised approaches, either cluster input

images embedded in a feature space [Baró et al., 2019] or rely on an existing text corpus of the recognized language [Gupta et al., 2018].

Closest to our work are classical prototype-based methods [Kopec and Lomelin, 1996, 1997; Xu and Nagy, 1999; Baird, 1999] and in particular the Ocular system [Berg-Kirkpatrick et al., 2013] which follows a generative probabilistic approach to jointly model text and character fonts in binarized documents and is optimized through Expectation Maximization (EM). Unlike us, it also relies on a pre-trained n-gram language model, originally from the English language and later extended to multiple languages [Garrette et al., 2015]. Other approaches rely on language models to identify characters [Kopec et al., 2001; Berg-Kirkpatrick et al., 2013; Gupta et al., 2018]. However, language models do not exist for unknown ciphers or historical manuscripts. Instead, we propose to disambiguate sprites by relying on line-level transcriptions.

Most related to our application in palaeography, [Goyal et al., 2020] proposes a probabilistic model for printed font analysis, [Srivatsan et al., 2021a] for linear scribal hands of linear-b. However, both works rely on single isolated and binarized characters as input, whereas the goal of our approach is to be directly applicable to colored text lines. Closer to us [Aiolli et al., 1999; Ciula, 2005] uses the tangent distance of [Simard et al., 2002] to learn prototypes from individual characters and cluster them into dendrograms of classes. Unlike us, this approach operates on cropped and segmented black and white characters.

Unsupervised multi-object segmentation. Unsupervised multi-object segmentation refers to a family of approaches that decompose and segment scenes into multiple objects in an unsupervised manner [Karazija et al., 2021]. Some techniques perform decomposition by computing pixel-level segmentation masks over the whole input image [Burgess et al., 2019; Emami et al., 2021; Greff et al., 2017, 2019b; Yang et al., 2020], while others focus on smaller regions of the input and learn to compose objects in an iterative fashion, mostly relying on a recurrent architecture [Crawford and Pineau, 2019; Deng et al., 2020; Eslami et al., 2016; Jiang and Ahn, 2020]. While all of these techniques can isolate objects from their backgrounds by producing segmentation masks, our goal is to summarize their recurring visual appearance.

We thus build on techniques that explicitly model the objects located inside the input image, by associating them to a set of image prototypes referred to as sprites through differentiable transformations [Monnier et al., 2021; Smirnov et al., 2021]. Sprites are color images with an additional transparency channel (RGBA), associated to networks that predict their spatial transformation [Jaderberg et al., 2015b] in order

to compose them onto a target canvas. DTI-Sprites [Monnier et al., 2021] provides good reconstruction fidelity, but can only predict a small amount of sprites for a collection of fixed-size images, and fails to scale when the number of objects within each image increases to those of real documents. At the same time, MarioNette [Smirnov et al., 2021] while being efficient, suffers from a high reconstruction error and fuzzy sprites that suboptimally reconstruct a toy text dataset.

3.3 The Learnable Typewriter

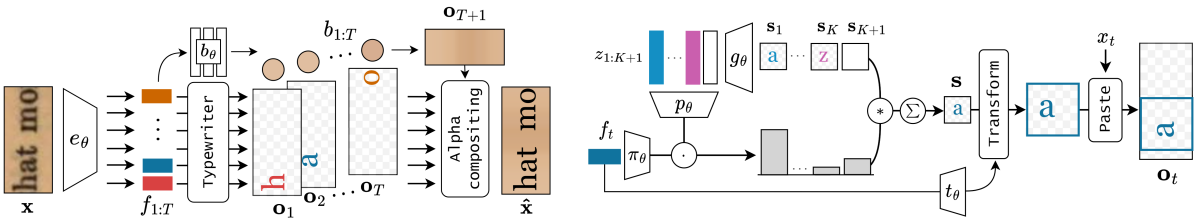
Given a collection of text lines that have a consistent font or script, our goal is to learn a representation of the average shape of all the characters it contains and a deep network that predicts how to transform them in order to reconstruct any input text line. Since complete supervision (i.e., the position and shape of every character found in a document) for would be extremely costly to obtain for our purposes, we propose to proceed in an analysis-by-synthesis fashion by building on sprite-based unsupervised image decomposition approaches [Smirnov et al., 2021; Monnier et al., 2021] which jointly learn a set of character images - called *sprites* - and a network that transforms and positions them on a canvas in order to reconstruct input lines. Due to the intrinsic ambiguity of decomposing a set of characters into sprites, we introduce a complementary weak-supervision from line-level transcriptions.

In this section, we first present an overview of our image model and approach (Sec. 3.3.1). Then, we describe the deep architecture we use (Sec. 3.3.2). Finally, we discuss our loss and training procedure (Sec. 3.3.3).

Notations. We write $a_{1:n}$ the sequence $\{a_1, \ldots, a_n\}$, and use bold letters $\mathbf{a}$ for images. An RGBA image $\mathbf{a}$ corresponds to an RGB image denoted by $\mathbf{a}^c$, alongside an alpha-transparency channel denoted by $\mathbf{a}^\alpha$. We use θ as a generic notation for network parameters and thus any character indexed by θ, e.g., a_θ, is a network.

3.3.1 Overview and image model

Fig. 3.2a presents an overview of our pipeline. An input image $\mathbf{x}$ of size $H \times W$ is fed to an encoder network e_θ generating a sequence of T features $f_{1:T}$ associated to uniformly-spaced locations $x_{1:T}$ in the image. Each feature f_t is processed independently by our *Typewriter* module (Sec. 3.3.2) which outputs an RGBA image $\mathbf{o}_t$ corresponding to a character. The images $\mathbf{o}_{1:T}$ are then composited with a canvas image we denote



(a) Overview of our full pipeline (b) Details of our Typewriter module

Figure 3.2: **Overview. (a)** An image is encoded into a sequence of features, each decoded by the Typewriter module into image layers. They are then fused by alpha compositing with a predicted low-res background. **(b)** The Typewriter module takes a feature as input, computes sprites and associated probabilities from learned latent codes, and composes them into a composite sprite that is then transformed and positioned onto an image-sized canvas.

$\mathbf{o}_{T+1}$. This canvas image $\mathbf{o}_{T+1}$ is a completely opaque image (zero transparency). Its colors are predicted by a Multi-Layer Perceptron (MLP) b_θ which takes as input the features $f_{1:T}$ and outputs RGB values $b_{1:T}$. All the resulting images $\mathbf{o}_{1:T+1}$ can be seen as ordered image layers and are merged using alpha compositing, as proposed by both [Monnier et al., 2021; Smirnov et al., 2021]. More formally, the reconstructed image $\hat{\mathbf{x}}$ can be written as:

$$\hat{\mathbf{x}} = \sum_{t=1}^{T+1} \left[\prod_{j<t} (1 - \mathbf{o}_j^\alpha) \right] \mathbf{o}_t^\alpha \mathbf{o}_t^c. \tag{3.1}$$

During training, the order of $\mathbf{o}_{1:T}$ in the compositing operation is randomized to reduce overfitting, as advocated by the MarioNette approach [Smirnov et al., 2021]. The full system is differentiable and can be trained end-to-end.

3.3.2 Typewriter Module

We now describe in detail the Typewriter module, which takes as input a feature f from the encoder and its position x, and outputs an image layer $\mathbf{o}$, to be composited. An overview of the module is presented in Fig. 3.2b. On a high level, it is similar to the MarioNette architecture [Smirnov et al., 2021], but handles blanks (i.e., the generation of a completely transparent image) differently and has a more flexible deformation model, similar to the one used in DTI-Sprites [Monnier et al., 2021]. More specifically, the module learns jointly RGBA images called *sprites* corresponding to character images, and networks that use the features f to predict a probability for each sprite and a transformation of the sprite. In the following, we detail how we obtain the

following three elements: the set of K parameterized sprites, the sprites compositing, and the transformation model.

Sprite Parametrization. We model characters as a set of K sprites which are defined using a generator network. More specifically, we learn K latent codes $z_{1:K}$ which are used as an input to a generator network g_θ in order to generate sprites $\mathbf{s}_{1:K} = g_\theta(z_{1:K})$. These sprites are images with a single channel that corresponds to their opacity. Similar to DTI-Sprites [Monnier et al., 2021], we model a variable number of sprites with an empty (i.e., completely transparent) sprite which we write $\mathbf{s}_{K+1}$. Instead of directly learning sprites in the pixel space as in DTI-Sprites [Monnier et al., 2021], we found that using a generator network yields faster and better convergence.

Sprite Probabilities and Compositing. To predict a probability p_k for each sprite $\mathbf{s}_k$, each latent code z_k is associated through a network p_θ to a probability feature $z^p_k = p_\theta(z_k)$ of the same dimension D as the encoder features ($D = 64$ in our experiments). We additionally optimize directly a probability feature z^p_{K+1} which we associate to the empty sprite. Given a feature f predicted by the encoder, we predict the probability p_k of each sprite $\mathbf{s}_k$ by computing the dot product between the probability features $z^p_{1:K+1}$ and a learned projection of the feature $\pi_\theta(f)$, and applying a softmax to the result:

$$p_{1:K+1}(f) = \text{softmax}\left(\lambda z^p_{1:K+1} \cdot \pi_\theta(f)^T\right), \tag{3.2}$$

where $\cdot$ is the dot product applied to each element of the sequence, $\lambda = 1/\sqrt{D}$ is a scalar temperature hyperparameter, and softmax is applied to the resulting vector. We use these probabilities to combine the sprites into the weighted average $\mathbf{s} = \sum_{k=1}^{K} p_k g_\theta(z_k)$. During inference, we simply select the sprite $g_\theta(z_k)$ with the highest probability p_k instead of computing a weighted average. Note that this compositing can be interpreted as attention operation [Vaswani et al., 2017]:

$$\mathbf{s} = \text{attention}(\bar{Q}, \bar{K}, \bar{V}) = \text{softmax}\left(\frac{\bar{Q}\bar{K}^T}{\sqrt{D}}\right)\bar{V}, \tag{3.3}$$

with $\bar{Q} = \pi_\theta(f)$, $\bar{K} = p_\theta(z_{1:K+1})$, $\bar{V} = g_\theta(z_{1:K+1})$, D the dimension of the features, and by convention $g_\theta(z_{K+1})$ is the empty sprite and $p_\theta(z_{K+1}) = z^p_{K+1}$. In fact, we show that directly optimizing $z^p_{1:K}$ instead of learning to predict the probability features $z^p_{1:K}$ from the sprite latent codes $z_{1:K}$, as in MarioNette [Smirnov et al., 2021], yields similar results. Note that we learn a probability code z^p_{K+1} to compute the probability of empty

sprites instead of having a separate mechanism as in MarioNette [Smirnov et al., 2021] because it is critical for our supervised loss (see Sec. 3.3.3).

Positioning and Coloring. The final step of our module is to position the selected sprite in a canvas of size $H \times W$ and to adapt its color. We implement this operation as a sequence of a spatial transformer [Jaderberg et al., 2015a] and a color transformation, similar to DTI-Sprites [Monnier et al., 2021]. More specifically, the feature f is given as input to a network t_θ that predicts three parameters for the color of the sprite and three parameters for isotropic scaling and 2D-translation that are used by a spatial transformer [Jaderberg et al., 2015a] to deform $\mathbf{s}$. Finally, using the location x associated with the feature f, we paste the deformed colored sprite onto a background canvas of size $H \times W$ at position x to obtain a reconstructed RGBA image layer $\mathbf{o}$. Positioning the sprites with respect to the position of the associated local features helps us obtain results co-variant to translations of the text lines and independent of the line size. To produce the background canvas, each of the features $f_{1:T}$ is first passed through a shared MLP b_θ, to produce a vector of T background colors $b_{1:T}$. We then use bilinear interpolation to upscale this vector to the full size of the input image x. Specific details concerning the parametrization of the transformation networks can be found in Appendix A (Sec. B.3).

3.3.3 Losses and training details

Our system is designed in an analysis-by-synthesis spirit and thus relies mainly on a reconstruction loss. This reconstruction loss can be complemented by a loss on the selected sprites in the supervised setting where each text line is paired with a transcription. In the following, we define these losses for a single text line image and its transcription, using the notations of the previous section.

Reconstruction loss. Our core loss is a simple mean square error between the input image $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$ predicted by our system as discussed in Sec. 3.3.1:

$$\mathcal{L}_{\text{rec}}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2. \tag{3.4}$$

In the unsupervised setting, we don't train with any additional regularization.

Weakly Supervised Loss. The intrinsic ambiguity of the sprite decomposition problem may result in sprites that do not correspond to individual characters.

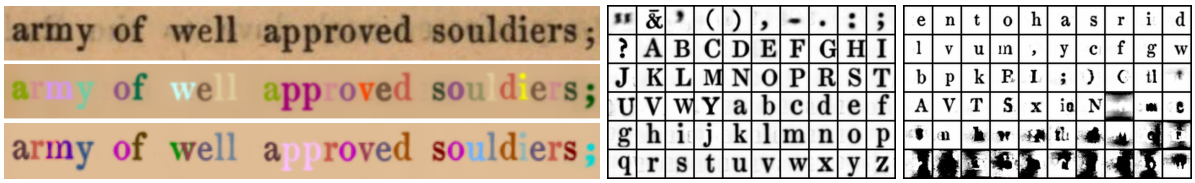
Using line-level annotation is an easy way to resolve this ambiguity. We find that simply adding the classical CTC loss [Graves et al., 2006] computed on the sprite probabilities to our reconstruction loss is enough to learn sprites that exactly correspond to characters. More specifically, we chose the number of sprites as the number of different characters and associate arbitrarily each sprite with a character and the empty sprite with the blank token of the CTC. Then given the one-hot line-level annotation y and $\hat{y} = (p_{1:K+1}(f_1), ..., p_{1:K+1}(f_T))$ the predicted sprite probabilities, we optimize our system's parameters by minimizing:

$$\mathcal{L}_{\text{sup}}(\mathbf{x}, y, \hat{\mathbf{x}}, \hat{y}) = \mathcal{L}_{\text{rec}}(\mathbf{x}, \hat{\mathbf{x}}) + \lambda_{\text{ctc}}\mathcal{L}_{\text{ctc}}(y, \hat{y}) \tag{3.5}$$

where λ_{ctc} is a hyperparameter and $\mathcal{L}_{\text{ctc}}(y, \hat{y})$ is the CTC loss computed between the ground-truth y and the predicted probabilities $\hat{y}$. We use $\lambda_{\text{ctc}} = 0.01$ in all our experiments, increased only for the less challenging Google1000 to $\lambda_{\text{ctc}} = 0.1$.

Implementation and training details. We train on the Google1000 [Vincent, 2007] and Fontenay [Camps et al., 2022] datasets with lines of height $H = 64$ and on the Copiale dataset [Knight et al., 2011] with $H = 96$. The generated sprites $\mathbf{s}_{1:K}$ are of size $H/2 \times H/2$. In the supervised setting, we use as many sprites as there are characters, and in the unsupervised, we set $K = 60$ for Google1000 and $K = 120$ for the Copiale cipher. We train for 100 epochs on Google1000 and for 500 epochs on Copiale with a batch size of 16, and we select the model that performs best on the validation set for evaluation. In the unsupervised setting, we use line crops of width $W = 2H$ and train for 1000 epochs on Google1000 and for 5000 on the Copiale cipher with a batch size of 32 and use the final model. The number of epochs is much higher in the unsupervised case than in the supervised case because the network sees only a small crop of each line at each epoch, but each epoch is much faster to perform. To always avoid learning sprites that reconstruct the background instead of the actual characters, we warm start the training process by only training the background MLP for 3000 gradient steps.

Our encoder network is a ResNet-32-CIFAR10 [He et al., 2016], that is truncated after layer 3 with a Gaussian feature pooling described in Appendix A (Sec. B.2). For our unsupervised experiments, we use as generator g_θ the U-Net architecture of Deformable Sprites [Ye et al., 2022] as it converged quickly, and for our supervised experiments a 2-layer MLP similar to MarioNette [Smirnov et al., 2021] which produces sprites of higher quality. The networks π_θ and p_θ are single linear layers followed by layer-normalization. We use the AdamW [Loshchilov and Hutter, 2019] optimizer with a learning rate of 10^{-4} and apply a weight-decay of 10^{-6} to the encoder parameters.



(a) Semantic Segmentation (↑ sup. ↓ unsup.) (b) Sprites (sup.) (c) Sprites (unsup.)

Figure 3.3: **Results on a printed document (Google1000).** In both the supervised and unsupervised setting our method produces meaningful sprites and accurate reconstructions **(a)**. We show the 60 most used sprites in alphabetic ordering in the supervised setting **(b)** and ordered by frequency in the unsupervised one **(c)**. See text for details and Appendix A for more reconstructions (Sec. A).

3.4 Experiments

In this section, we evaluate the performance of our method on challenging datasets of historical and modern fonts, as well as a handwritten cipher. We first introduce the datasets and metrics used in our evaluation in Sec. 3.4.1. Then we present qualitative results discussing the quality of the learned sprites with and without supervision in Sec. 3.4.2. Finally, in Sec. 3.4.3, to assess the performance of our method and ablate our architectural choices we present quantitative results both for reconstruction and for transcription quality, for both the supervised and the unsupervised setting.

3.4.1 Datasets and metrics

Datasets. We experiment with four datasets with different characteristics: Google1000 [Vincent, 2007], MFGR [Seuret et al., 2023], the Copiale cipher [Knight et al., 2011] and Fontenay manuscripts [Camps et al., 2022]:

• *Google1000.* The Google1000 dataset contains scanned historical printed books, arranged into Volumes [Vincent, 2007]. We use the English Volume 0002 which we process with the preprocessing code of [Gupta et al., 2018], using 317 out of 374 pages and train-val-test set with 5097-567-630 lines respectively. This leads to a total number of 83 distinct annotated characters. Although supervised printed font recognition is largely considered a solved problem, and the annotation for this dataset is actually the result of OCR, this document is still challenging for an analysis-by-synthesis approach, containing artifacts such as ink bleed, age degradation, as well as variance in illumination and geometric deformations due to digitization.

• *MFGR.* The ICDAR-2024 "Multi Font Group Recognition and OCR challenge" dataset [Seuret et al., 2023], contains text lines that were printed with a set of 8 distinct

typefaces: *antiqua, bastarda, fraktur, gotico-antiqua, italic, rotunda, schwabacher, textura*. We focus only on lines that contain fonts from a single group. This dataset is similar to Google1000, but comes with the challenges of older prints, such as non-fully printed letters and allographs. With 12K-45K training lines for each of the 8 different typefaces, it serves as an ideal benchmark to assess the robustness of our model in learning meaningful sprites across a variety of historical prints.

• *Copiale cipher.* The Copiale cipher is an oculist German text dating back to an 18th-century secret society [Knight et al., 2011]. Opposite to Baro et al. [Baró et al., 2019] which uses a binarized version of the dataset, we train our model on the original text-line images, which we segmented using docExtractor [Monnier and Aubry, 2020] and manually assigned to their annotations, respecting the train-val-test split of Baro et al. [Baró et al., 2019] with 711-156-908 lines each. The total number of distinct annotated characters is 112. This dataset is more challenging than printed text because of the handwritten variance of a historical manuscript, and its large number of characters.

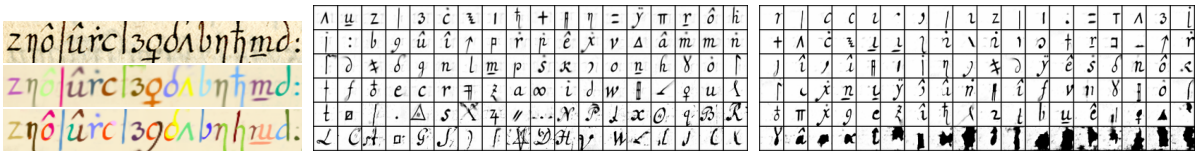
Metrics. Our goal is to capture the shape of all characters and position them precisely on each text line. Such fine-grained annotation is however not available in existing datasets. Instead, to provide a quantitative evaluation of our models, we report mean squared reconstruction error ("Rec." in our tables) and Character Error Rate (CER). CER is the standard metric for Optical Character Recognition (OCR). Given ground-truth and predicted sequences of characters, σ and $\hat{\sigma}$, it is defined as the minimum number of substitutions S, deletions D, and insertions I of characters needed to match the predicted sequence $\hat{\sigma}$ to the ground truth sequence σ, normalized by the size of the ground truth sequence $|\sigma|$:

$$CER(\sigma, \hat{\sigma}) = \frac{S + D + I}{|\sigma|}. \tag{3.6}$$

For simplicity, we ignore spaces. Predictions are obtained by selecting at every position the character associated to the most likely sprite. In the supervised setting, the association between sprites and characters is fixed at the beginning of training. In the unsupervised setting, we associate every sprite to a single character using a simple assignment strategy described in Appendix A (Sec. D).

3.4.2 Qualitative results

Examples of semantic segmentation and sprites in the supervised and unsupervised setting on Google1000 and Copiale are shown in Figs. 3.3,3.4 respectively. In the



(a) ↑ sup. ↓ unsup. (b) Sprites (sup.) (c) Sprites (unsup.)

Figure 3.4: **Results on the Copiale cipher [Knight et al., 2011].** Despite the high number of characters and their variability, our method learns meaningful sprites and performs accurate reconstructions in both settings **(a)**. We show the 108 most used sprites sorted by frequency in the supervised **(b)** and the unsupervised **(c)** settings.

unsupervised setting, several sprites (Figs. 3.3c,3.4c) can be used to reconstruct a single character. For example in Google1000, the "n" and "m" sprites are joined to better reconstruct "m". To account for appearance variation, multiple sprites are learned to reconstruct the most frequent character, e.g., "e" for Google and "c" in Copiale. These effects are even stronger in the handwritten Copiale dataset, where generic sub-character strokes are learned and used to better model characters' variations. In both datasets, some of the least used sprites do not correspond to characters, as they are never selected by the network, and thus are not properly optimized. These behaviors are expected in a completely unsupervised setting, because of the highly unbalanced statistics of the character frequencies and the ambiguity of reconstruction: without additional supervision, there is a clear benefit for the network to model well the variations of common characters, and to approximate or discard rare ones. This is a core limitation of existing unsupervised image decomposition approaches and a motivation for the introduction of our weakly supervised setting. In the (weakly) **supervised** setting, the sprites (Fig. 3.3b, 3.4b) closely correspond to the characters, except for very rare characters like the capital letter 'Z' for Google1000 (as can be seen in Fig. A.1 of Appendix A), while reconstruction is of very high quality and each character is reconstructed with the expected sprite.

Historical Font Reconstruction. In Fig. 3.5 we compare our learned sprites to manually extracted and binarized exemplars, where we observe that the learned sprites are mostly similar to the exemplars. Typefaces are sorted according to the average similarity between all the learned sprites and the manually extracted exemplars (between a-z and A-Z) that is computed using SSIM, as is a standard practice for font comparison [Srivatsan et al., 2021b]. SSIM is the highest for the antiqua font (0.745) and the lowest for the gotico-antiqua font (0.676). This seems correlated to the number of allographs that are present in each typeface. Antiqua is simple and standard, whereas

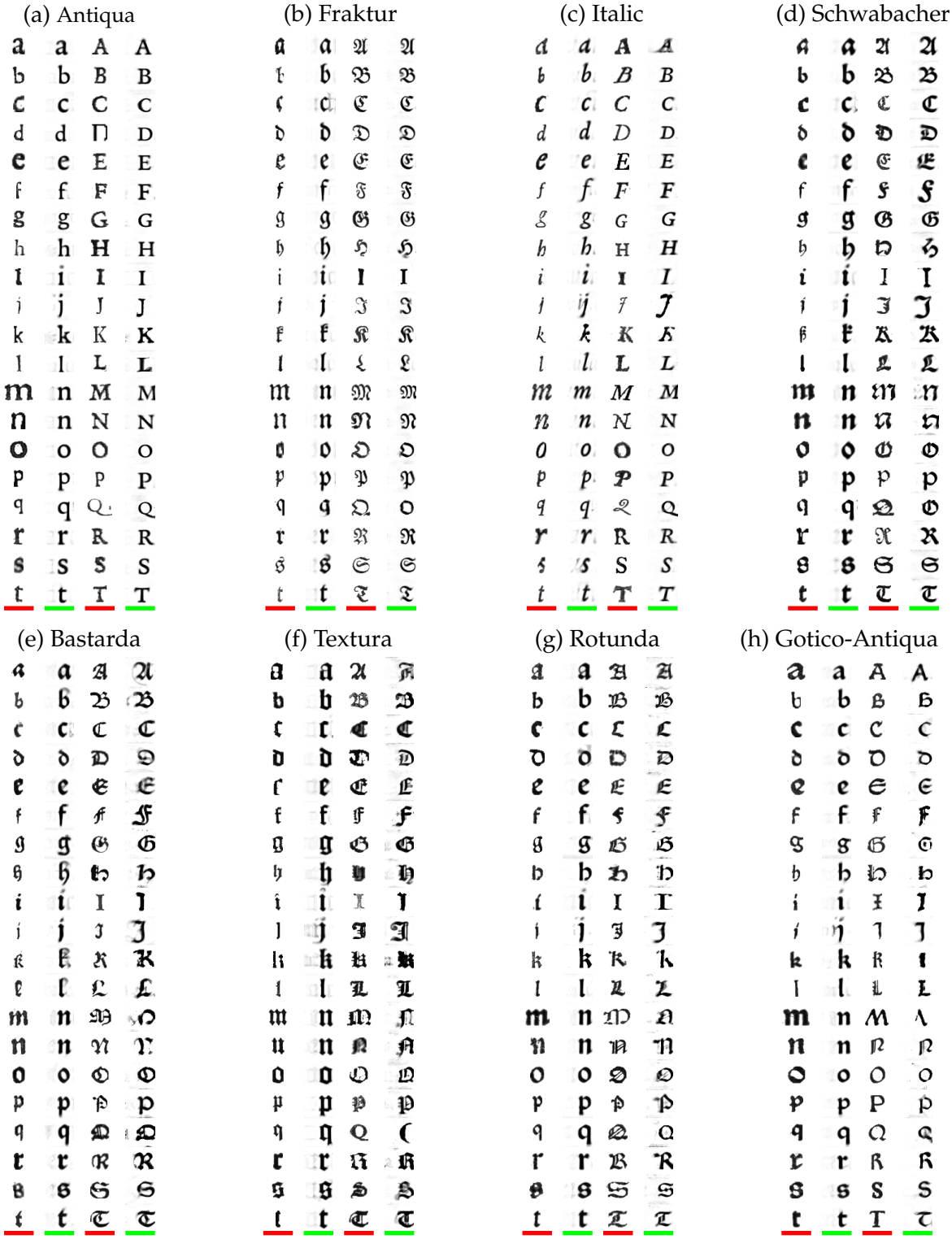


Figure 3.5: **Qualitative Evaluation on MFGR.** We compare a-t, A-T between learned sprites (ours), and manually extracted exemplars. Fonts are sorted by descending SSIM, computed on post-processed sprites (see Sec. C of Appendix A for more details). Note, that although fonts come from a single family, they may present *allographs*. For example, *Italic* contains different variants of "Q", where a random exemplar can significantly differ from the one summarized using our method.

gotico-antiqua is a hybrid between two visually distinct fonts, hence as our model learns a single sprite it fails to summarize them (e.g., see "T", "G", "I", "E", "F"). These results showcase the versatility of our approach, which is crucial for it to be applied for historical analysis.

3.4.3 Quantitative results

Our quantitative results and ablations for Google1000 and Copiale are reported in Tabs. 3.1,3.2 respectively.

For Google1000, the CER in the supervised setting is less than 1%, while it is 7.7% for the unsupervised setting. To provide baselines for these performances, we train and evaluate on our version of the dataset **(a)** ScrabbleGAN [Fogel et al., 2020], a supervised method with a standard recognizer and an additional generator module, **(b)** FontAdaptor [Zhang et al., 2020], a recent 1-shot method that learns to match single character exemplars to text lines, and **c)** an adaptation of the unsupervised DTI-Sprites [Monnier et al., 2020] to text lines which we detail in Appendix A (see Sec. E), where we also show that the vanilla MarioNette [Smirnov et al., 2021] has significantly worse results. Our unsupervised approach performs clearly better than our adaptation of DTI-Sprites and is almost on par with the 1-shot FontAdaptor, while our weakly supervised approach is almost on par with ScrabbleGAN. Our adaptation of DTI-Sprites is better at reconstructing images, but the learned sprites are much less meaningful, as shown by the poor CER performance. Interestingly, reconstruction is much better when using supervision, which hints that a better optimization scheme could improve unsupervised performances. We also evaluate the effect of varying the number of sprites K in the unsupervised setting. For K smaller than the actual number of characters (83), namely $K = 21$ and $K = 41$, we have a significant performance drop of 10% and 26% CER respectively, while increasing the number of characters to 166 and 332 doesn't significantly boost performances.

On the Copiale dataset, we compare our results with HTRbyMatching [Souibgui et al., 2020], a few-shot approach developed specifically for cipher recognition, using the same train/val/test splits. HTRbyMatching was evaluated on a wide range of few-shot scenarios, ranging from a scenario similar to FontAdaptor where a single exemplar is available for every character, to one where 5 exemplars are available for each character together with 5 completely annotated pages. Reported results are only for confident character predictions with different confidence thresholds, but summing the error rate of the predicted symbols and the percentage of non-annotated symbols, one can estimate the CER to vary between 10% and 47% depending on the

Method	Type	Rec. $\times 10^3$	CER
DTI-Sprites [Monnier et al., 2021]	unsup.	2.54	18.4 %
FontAdaptor [Zhang et al., 2020]	1-shot	-	6.7 %
ScrabbleGAN [Fogel et al., 2020]	sup.	-	0.6 %
Learnable Typewriter	sup.	3.5 ± 0.1	$0.85 \pm 0.03\%$
w\o shared z_k	sup.	3.3 ± 0.1	$0.89 \pm 0.06\%$
w\o p_θ	sup.	3.5 ± 0.1	$0.99 \pm 0.05\%$
w\o g_θ	sup.	3.4 ± 0.1	$0.88 \pm 0.04\%$
Learnable Typewriter	unsup.	7.1 ± 0.4	$7.7 \pm 0.6\%$
w\o shared z_k	unsup.	7.4 ± 0.4	$8.0 \pm 0.2\%$
w\o p_θ	unsup.	7.0 ± 0.3	$7.7 \pm 2.0\%$
w\o g_θ	unsup.	10.5 ± 0.7	$27.0 \pm 2.2\%$

Table 3.1: **Quantitative results and ablation on Google1000 [Vincent, 2007].** We report CER and mean squared reconstruction error for all the different approaches. For our method, we report the average of 5 runs and their standard deviation.

scenario[1]. This is consistent with the quantitative results we obtain with our approach, which are much better in the supervised setting (4.2%) and worse in the completely unsupervised one (52.6%). The low performance of the unsupervised approach is consistent with the qualitative results: given that many characters are reconstructed by sub-character sprites, one would have to associate sprite bi-grams to characters in order to obtain good CER performances. Interestingly, the reconstruction error is similar in the supervised and unsupervised setting, hinting that for this specific dataset, optimizing the reconstruction quality might not be enough to obtain relevant decomposition without additional priors. These results enable us to quantify and analyze a limitation of unsupervised image decomposition approaches on a more challenging dataset.

Note that the goal of our approach is not to boost CER performances - which in any case would be futile for Google1000 where the ground truth is already the result of an OCR model - but instead to learn character models and image decomposition. All these comparisons should be thus considered as sanity checks. Yet, it is possible to design post-processing algorithms to improve CER. We tested a simple algorithm where we assign a new sprite to the most frequent bi-grams and tri-grams, which leads to an improved CER for Copiale of 29.9%. However, we find this metric more

[1]Note, that the original paper calls SER our Character Error Rate which excludes spaces, corresponding to Symbol Error Rate, and not Sentence Error Rate.

Method	Type	Rec. $\times 10^2$	CER
HTRbyMatching [Souibgui et al., 2020]	few-shot	-	$10-47\%^*$
Learnable Typewriter	sup.	1.81 ± 0.01	$4.2 \pm 0.3\%$
w\o shared z_k	sup.	1.79 ± 0.01	$4.0 \pm 0.1\%$
w\o p_θ	sup.	1.77 ± 0.02	$4.7 \pm 0.1\%$
w\o g_θ	sup.	1.96 ± 0.07	$4.2 \pm 0.2\%$
Learnable Typewriter	unsup.	1.93 ± 0.02	$52.6 \pm 1.7\%$
w\o shared z_k	unsup.	1.89 ± 0.02	$47.6 \pm 2.8\%$
w\o p_θ	unsup.	1.81 ± 0.06	$51.9 \pm 2.0\%$
w\o g_θ	unsup.	3.99 ± 0.14	$80.6 \pm 0.9\%$

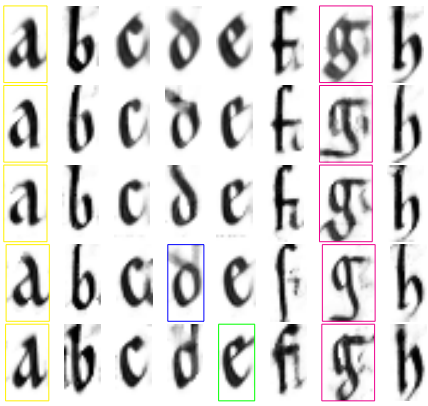
Table 3.2: **Quantitative results on Copiale [Knight et al., 2011].** We report CER and reconstruction error to evaluate both our selected baselines and our method. For our method, we report it across an average over 5 runs alongside its standard deviation. *See text for details.

informative when applied to the raw output of unsupervised image decomposition models.

In particular, we perform on both datasets an ablation of the architecture to better understand which design choices are critical. Interestingly, our results show that both in the supervised and the unsupervised setting, not sharing the latent codes z_k between the generation network and the sprite selection and even completely removing the probability network p_θ has limited influence on the performance clarifying that these design choices of MarioNette [Smirnov et al., 2021] are not of critical importance. Conversely, removing g_θ and directly learning prototypes as network parameters similar to DTI-Sprites [Monnier et al., 2021] has little impact in the supervised case, but leads to a significant drop in performance in the unsupervised one. A more detailed analysis of training curves reveals that in the unsupervised case, training is slower and leads to overfitting. While it might be possible to fix this issue by adapting the learning scheme for the prototypes, this shows that it is easier to learn the prototypes through a generator network than to optimize them directly.

3.5 Application to palaeography

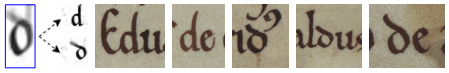
Understanding written character morphology, or *script type* is of central importance to palaeography, which seeks is to employ handwriting as historical evidence. Script types are for a handwritten manuscript what a font is for a printed one, yet as these prototypes don't exist outside the "scribe's mind" [Parkes, 1969; Stokes, 2011] and



(a) Sprites from document finetuning.

(b) Sprites and examples for “a” and “g”.

(c) Sprite and multiple examples for “e’.



(d) Learning dual sprites for “d” allographs.

Figure 3.6: **Sprites learned for similar documents of a Praegothica script.** Left: each line corresponds to a different document. Looking at any column, one can notice the small differences that characterize the handwriting in each document. Right: Colored boxed correspond to cases analyzed in more detail. Sprites summarize the key attributes of a character in each specific document, averaging its variations. Note the complexity of the documents: characters can overlap or be connected ligature, the parchment is often stained, and there are important intra-document character variations. *See text for more details.

are passed-by through a historical tradition, they come with a lot of written variation. Structuring this variation is crucial to adequately navigate such historical evidence [Stutzmann, 2018], because the analysis of script types cannot be reduced to a classification problem [Stutzmann, 2013, 2016; Hassner et al., 2015]. Moreover, such natural language descriptions may often be ambiguous in their process of being communicated across palaeographers and are often arbitrary as a means for comparison [Derolez, 2003]. In other words, as we discussed in Sec. 1.2, they are too Aristotelian [Studtmann, 2024]. Instead of a system of classification, what is more in need is a system for visual grounding [Ciula, 2017]. A simple solution, would be to choose an exemplar from a specific document or to ask a palaeographer to manually draw one that is “typical”. However, such processes are very time-consuming and might reflect their priors or biases, and lack interpretability on how they aggregate the existing variations of written morphology. Instead, we propose to continue previous research approaches such as the *System of Palaeographical Inspection* [Aiolli et al., 1999; Ciula, 2005], which tries to learn a hierarchy of “character prototypes”, similar to the ones of Eleanor Rosch as discussed Sec. 1.2, starting from the segmented characters of a document collection. Instead, in this Section we show how the Learnable Typewriter can extract prototypes directly from input text lines in a way that makes them visually

comparable, enabling a more systematic and interpretable quantitative palaeographic analysis.

Making Prototypes Comparable. Unlike Fig. 3.5 where prototypes are expected to have distinct differences, in palaeography, we expect to compare prototypes with subtle differences. Our goal is to analyze the written morphology across a granularity that can range from an individual document to a complete document collection. In order to properly compare their prototypes, our analysis would benefit from visual alignment. To approach this, our methodology is to first learn a *reference model* using a reference corpus and then finetune the model to reconstruct a subset of that corpus, up to the granularity of a specific document, by finetuning only the parameters that correspond to $g_\theta(z)$. Since the positioning, scaling, and coloring of the prototypes are shared, the prototypes will remain aligned, making them directly comparable. We first demonstrate our analysis approach in a small challenging dataset of Manuscripts from the Fontenay Abbey in Sec. 3.5.1, and then we move to a larger dataset of more established typology of Textualis Formata in Sec. 3.5.2 with the goal of performing a more systematic quantitative analysis.

3.5.1 Fontenay Manuscript

To first demonstrate the potential of our approach for palaeographic analysis, we apply it to a collection of 14 historical charters from the Fontenay abbey [Camps et al., 2022]. It contains digitized charters that originate from the Cistercian abbey of Fontenay in Burgundy (France) [Camps et al., 2022] and were created during the 12th and early 13th centuries. While they were carefully written and preserved, these documents are still very challenging (Fig. 3.6). Although they all use similar scripts from the *Praegothica* type, they also exhibit clear variations. Each of these documents has been digitized and each line has been manually segmented and transcribed. For our experiments, we selected a subset of 14 different documents sharing a similar script which falls into the family of praegothica scripts. These correspond to 163 lines, that use 47 distinct characters. They exhibit degradation, clear intra-document letter shape variations, and letters can overlap or be joined by ligature marks. Moreover, each document represents only a small amount of data, e.g., the ones used in Fig. 3.6 contain between 8 and 25 lines.

Qualitative Results Fig. 3.6a visualizes the sprites obtained for five different documents for the characters “a” to “h” and Fig. 3.6 highlights different aspects of these

results. Fig. 3.6b highlights the fact that the differences in the learned sprites correspond to actual variations in the different documents, whether subtle, such as for the "a" sprite, or clearer, such as for the descending part of the "g" sprite. Fig. 3.6c shows how a sharp sprite can be learned for the character "e", summarizing accurately its shape despite small variations throughout its different occurrences. Finally, Fig. 3.6d shows the case of a document in which two types of "d" co-exist. In this case, the learned sprite, shown on the left, reassembles an average of the two, where both versions of the ascending parts are visible on medium transparency. However, this limitation could be overcome by learning several sprites per character. Using our approach, we can learn two sprites per character, simply by summing their probabilities when optimizing the CTC-loss. We find that when allographs exist, i.e., different appearances of the same letter, these two different sprites do learn recover its two distinct appearances. In our example, we show the two different learned "d" sprites on the right of the original one.

3.5.2 Textualis Formata

Textualis formata is an established medieval gothic script which was continuously used for over three centuries (13th-15th century). To analyze the results of our approach, we adopt the taxonomy formalized by A. Derolez [Derolez, 2003], which provides a framework based on morphological criteria. Derolez makes a distinction between two subtypes of Northern and Southern Textualis (denoted as *NT* and *ST*) following their geographical location. However both types are followed by multiple distinct subtypes, that concern date, geographical origin, or language, which often intersect, questioning its fundamental distinction. To provide an analysis of this script type we compile a dataset of four train subfamilies and three test subfamilies for each class, providing a total of 892 lines. Our dataset selection is detailed in Appendix A (Sec. F.1).[2]

Quantitative Paleography We start by training multiple models as in Sec. 3.5.1 in order to obtain character prototypes at different levels of granularity: **(a)** a script type model for *Textualis*, **(b)** script subtype models for Northern and Southern *Textualis*, and **(c)** document level models for each document in our dataset. To quantitatively compare prototypes between **(b)** and **(c)**, we need to finetune from a common point of reference, for which we set the model **(a)** that was trained on all Textualis. Aligned

[2]Note, that this section is a reduction of [Vlachou-Efstathiou et al., 2024] focusing on its methodology which was my main contribution, as opposed to the palaeographic analysis which was the primary expertise of the first author.

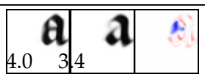
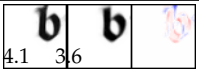
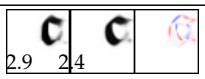
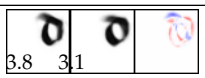
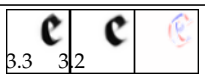
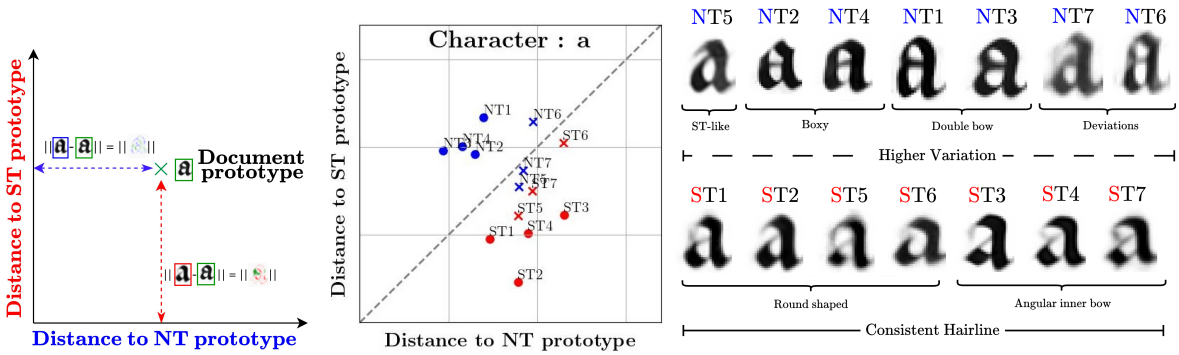
‹Ch.›	Derolez'criteria	NT | ST | diff. σ_{NT} σ_{ST}
‹a›	**NT**: Closed form with variations like "box-‹a›" **ST**: Open form or slightly closed with hairline	
‹b›	**NT**: Sloped or forked ascender tops **ST**: (i) Flat ascender tops, (ii) round lobe	
‹c›	**NT**: Angular or broken lobe curves **ST**: Semi-circular lobe	
‹d›	**NT**: (i) Lengthened and (ii) concave shaft **ST**: (i) Shorter shaft and (ii) almost horizontal, (iii) round bowl	
‹e›	**NT**: (i) Diagonal direction of the hairline and (ii) angular or broken lobe curves **ST**: (i) Horizontal or no hairline, (ii) semicircular lobe form	

Figure 3.7: Derolez' criteria for Northern and Southern *Textualis* and our subtype prototypes.

prototypes can now allow us to subtract them in pixel space and thus quantify their difference in an interpretable way. To optimally and reliably compare prototypes, we develop a post-processing procedure and quality evaluation which we describe in Appendix A (Sec. F.2). To make this difference easier to understand, we use a color map that represents zeros as white, and positive and negative values as two distinct colors, typically red and blue. By revealing pixel-wise differences, this method facilitates an initial qualitative examination of morphological disparities.

Consistency with classical Palaeography. In Fig. 3.7 we systematically check how the extracted observations from prototype comparisons conforms to Derolez's general morphological criteria for both Northern and Southern *Textualis* prototypes, highlighting their variations by visualizing their difference. While only ‹a›-‹e› are shown in the table, more complete results can be found on our original publication [Vlachou-Efstathiou et al., 2024]. In short, we find that Derolez's observations closely align with the variations that our prototypes enable us to visualize, which shows that our method conforms to the classical palaeographic analysis. Additionally, we report the standard deviation across prototypes for σ_{NT} and σ_{ST} for each letter, which were consistently higher for Northern *Textualis*, which is consistent with Derolez's claim that this script subtype generally exhibits higher intra-class variation.

Morphological EDA with Character Graphs. Having prototypes that characterize different groups of instances from certain manuscripts allows us to perform intra-class



(a) Graph interpretation (b) Character graph (c) Sprite Observations

Figure 3.8: **Comparison graph pipeline.** For a given document we can locate how far a manuscript prototype is to the prototypes of two reference families by computing their L2-distance **(a)**. Mapping this way allows us to locate sub-groups of prototypes **(b)** which have clear qualitative interpretation **(c)**. *See text for more details.

analysis. To allow quantitative exploratory data analysis, we introduce a comparison graph, illustrated in Fig. 3.8a. In this graph, each point represents a specific document character prototype, with its coordinates defined as its distance in pixel space to two selected prototypes computed by finetuning on the full manuscript that corresponds to each typology. Blue and red, markers signify Northern, and Southern *Textualis* documents, respectively. Each document's identifier is written near its marker.

Analyzing the Character ‹a›. The letter ‹a› is often considered as a discriminative criterion between script types, so much so that W. Oeser [Oeser, 1971] distinguished seven categories within the Northern *Textualis* script subtype mainly based on allographs of ‹a›. The dispersion of the characters on the graph in Fig. 3.8b provides insight into the variability of ‹a› in this subtype. The group associated to NT1-4 corresponds to the closed "box-a" form in NT2 ( ) and NT4 ( ) and the double-bow variant in NT1 ( ) and NT3 ( ). NT6 presents a more vertically elongated form which stands out ( ). Most striking in our ‹a› character graph is that the prototypes for NT5 ( ) and NT7 ( ) are actually closer to the ST prototypes. This is consistent with the observation that open ‹a› forms are standard for ST. While there are morphological variations across ST documents, with round shapes (ST1 ; ST2 ; ST5 ; ST6), or with more angular inner bows (ST3 ; ST4 ; ST7), the consistent use of an open form, or only closed with a hairline, distinguishes them from the NT subtype, and all ST documents prototypes are consistently closer to the ST prototype. Plotting this graph across documents in Fig. 3.9 allows us to easily identify that **NT5** stands out in the graphs as an outlier document, as seven character prototypes are closer to ST than to

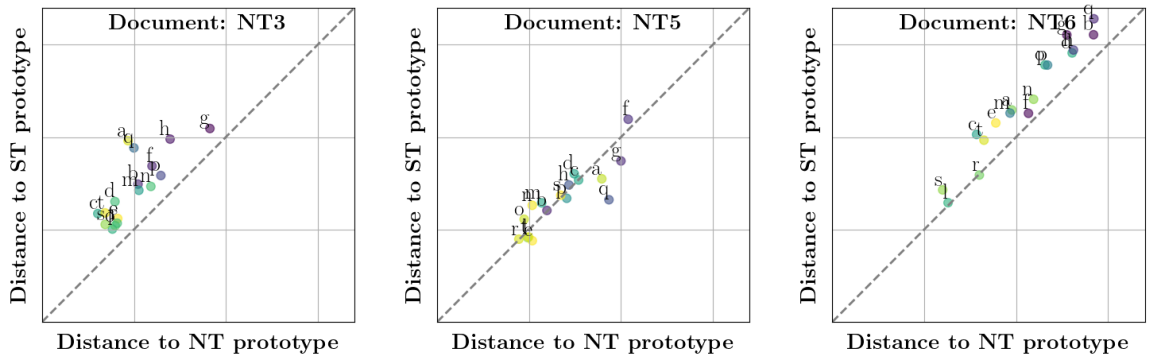


Figure 3.9: **Document comparison graphs.** Visualizing all letters for different documents allows us to identify the document NT5 as an outlier of the Northern Textualis family.

NT prototypes, with a particular difference for ‹a,g,q› ( ). These observations can be summarized in Fig. 3.8c.

3.6 Conclusion

We have presented a document-specific generative approach to document analysis. Inspired by deep unsupervised multi-object segmentation methods, we designed an end to end differentiable approach to accurately model text line images of manuscripts through a set of learned sprites. We outlined that a completely unsupervised approach suffers from the ambiguity of the decomposition problem and the imbalanced character distributions. Therefore, we extended these approaches using weak supervision to obtain robust, high-quality results. These allow us to learn prototypes for the characters of both standard printed documents and of much more complex ones, such as a handwritten ciphered manuscript or ancient charters. Finally, we demonstrated the potential of our approach for a novel application: palaeographic analysis. We extended our approach to a methodology for interpretable qualitative and quantitative palaeography, that can integrate and complement traditional historical approaches.

Chapter 4

Diffusion Models as Data Mining Tools

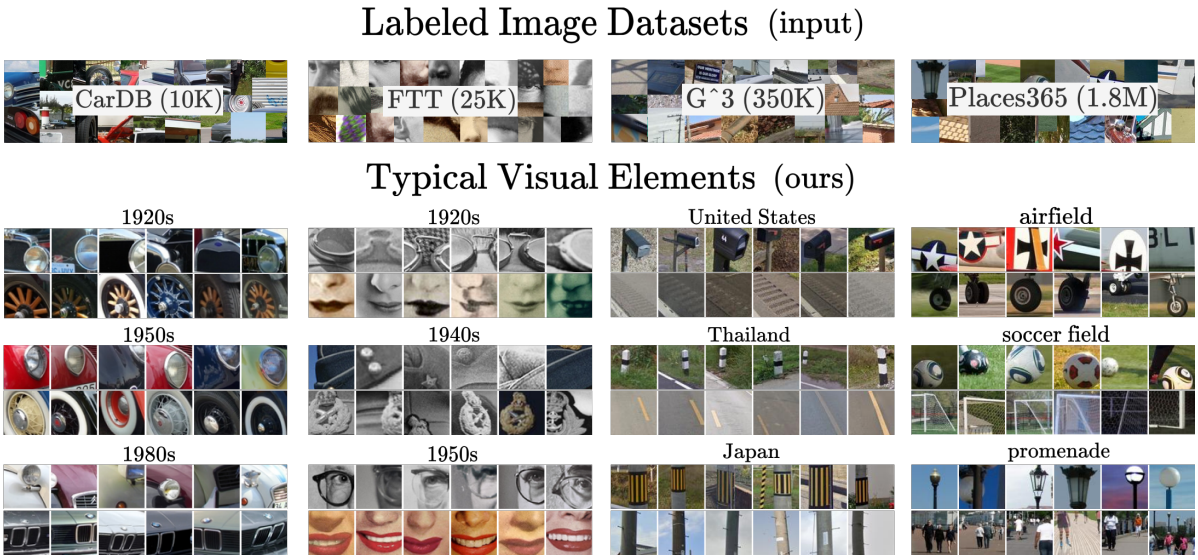


Figure 4.1: **Mining typical visual elements with diffusion models.** We demonstrate how to use diffusion models to mine visual data through a simple pixel-based score and a standard clustering approach. We present high-quality mining results for a diverse range of datasets (from left to right: 10,130 photographs of cars tagged with a creation year between 1920-1999 [Lee et al., 2013], 24,874 portraits from the 19th to the 21st century [Chen et al., 2023], 344,224 StreetView images tagged with country names [Luo et al., 2022], and 1,803,460 images of scenes images associated with descriptive names [Zhou et al., 2017a]). Our results highlight both expected elements and more unforeseen ones.

4.1 Introduction

As we discussed in Sec. 2.2.2 traditional image data mining aims to discover patterns within large visual corpora such as collections of StreetView panoramas [Doersch

et al., 2012; Lee et al., 2015b], historical images of faces [Ginosar et al., 2017; Chen et al., 2023] or photographs of cars [Lee et al., 2013; Dalens et al., 2019]. This would often be achieved through pairwise comparison of candidate patches in order to discover which ones of them are both frequent and discriminative [Singh et al., 2012]. This chapter proposes a novel idea: to turn generative models trained for image synthesis into a scalable method for large scale mining of image datasets. Generative models digest massive amounts of data, which they implicitly store in their weights. Our central insight is that we can use this learned summary of the visual input to identify the most typical image regions. This unconventional use of a diffusion model for studying its training data demonstrates that generative models are potent tools beyond synthesis—for data mining, summary, and understanding.

As we discussed in Sec. 1.3 our target task, mining for informative visual patterns, is challenging. Unlike text, where words act as discrete tokens that we can directly compare, the visual world seldom contains exactly repeating elements. Even common simple visual elements, such as windows, can have different colors and different numbers of panes; they may be seen from various viewpoints, and they may be located at multiple positions as part of different facades. The standard approach to image data mining [Doersch et al., 2012; Lee et al., 2013; Shen et al., 2021a] which we discussed in Sec. 2.2.2, involves learning data-specific similarities with relevant invariances (*e.g.*, such that different-looking windows will be similar) and using them to search for discriminative patterns. However, these techniques are not easily scalable since one must apply them across all pairs of visual elements within all pairs of images in the dataset. In other words, the similarity graph between visual elements scales quadratically with the size of the dataset. In contrast, our proposed analysis-by-synthesis approach does not require pairwise comparisons between different visual elements and thus scales to very large datasets.

The approach we propose takes as input a dataset with image-level tags, such as time [Lee et al., 2013; Chen et al., 2023], geography [Luo et al., 2022], or scene labels [Zhou et al., 2017a]. Our goal is to provide a visual summary of the elements typical of the different tags, such as the common elements that enable us to determine the location of a StreetView panorama. To arrive at this summary, we first finetune a conditional diffusion model on the target dataset. We then use the finetuned model to define a pixel-wise typicality measure by assessing the degree to which the label conditioning impacts the model's reconstruction of an image. We mine visual elements by aggregating typicality on patches, selecting the most typical ones, and clustering them using features extracted from the finetuned model [Tang et al., 2023]. As visu-

alized in Fig. 4.1, this leads to clusters of typical visual elements that summarize the most characteristic patterns associated with the tags available in the input dataset. For example, our face results highlight iconic elements, such as aviator glasses in the 1920s and military hats in the 1940s, and more subtle details, such as period-typical glasses or make-up. Interestingly, our results on StreetView data highlight details that are similar to the ones presented in geographical understanding websites [geodummy, 2023; geohints, 2023; Plonkit, 2023], popularized through the GeoGuessr game [geoguessr, 2023], such as typical parts of utility poles, bollards, or architecture. To our knowledge, no existing visual mining method has demonstrated such high-quality results on diverse datasets.

Contributions. To summarize, we present:

- a typicality score that can be formally derived from a diffusion model, allowing for an efficient extraction of the most typical visual elements of a dataset,
- a pipeline to extract and cluster typical elements in order to create typical summaries of different datasets, including cars [Lee et al., 2013], portraits [Chen et al., 2023], geographical data [Luo et al., 2022], and scenes [Zhou et al., 2017a],
- further applications of our method in locating elements which are typical across location, visualizing the bias of a diffusion model, and localizing abnormalities in chest X-ray images.

4.2 Related Work

Image data mining. Image data mining turned the manual and subjective process of comparing photographs (*e.g.*, [Kotchemidova, 2005]) into algorithmic methods for summarizing image data, such as architectural details [Doersch et al., 2012; Lee et al., 2015b], fashion [Ginosar et al., 2017; Matzen et al., 2017; Chen et al., 2023], industrial design [Jae Lee et al., 2013], and art [Shen et al., 2019, 2021b; Kaoua et al., 2021] by locating visual patterns. As we discussed in Sec. 2.2 this has mainly been achieved using techniques such as discriminative clustering. For example, [Lee et al., 2013] demonstrated how correspondence based mining across time can be achieved in a dataset of objects of similar parts, namely cars, and [Doersch et al., 2012] showed that geographically representative image elements can be automatically discovered from Google StreetView imagery in a discriminative manner. However, these traditional data mining approaches do not scale to large modern datasets. Indeed, they require pairwise comparisons between all the visual elements of each image to the entire

dataset in order to locate nearest neighbors and establish clusters. Notably, the discriminative clustering algorithm of [Doersch et al., 2012] requires training a separate linear SVM detector for each visual element- a computationally prohibitive approach when considering multiple possible visual elements for the purposes of analysis. In contrast, our approach is scalable to very large datasets. Closer to our work, generative models have been trained to analyze the evolution of faces [Chen et al., 2023] and cars [Dalens et al., 2019] across time, and the change in geography across GPS [Feng et al., 2025]. However, these works essentially focus on conditional image translation, and do not try to mine typical elements in their datasets.

Diffusion models. Diffusion models have gained popularity in recent years due to their stability in training and effectiveness in modeling complex multimodal distributions [Sohl-Dickstein et al., 2015; Ho et al., 2020; Dhariwal and Nichol, 2021; Ho et al., 2022; Karras et al., 2022]. These models are capable of generating high-quality imagery conditioned on input signals beyond categorical labels, like text [Rombach et al., 2022; Saharia et al., 2022; Ramesh et al., 2022], and can further incorporate additional modalities [Zhang et al., 2023; Li et al., 2023c]. In addition to generating images from scratch, diffusion models have been used extensively for instruction-driven image-to-image translation [Meng et al., 2022; Hertz et al., 2022; Mokady et al., 2023; Tumanyan et al., 2023; Brooks et al., 2023]. It has also been shown that pre-trained text-to-image diffusion models encode strong priors for natural scenes, allowing their internal features to be used for secondary tasks [Xu et al., 2023; Luo et al., 2023; Tang et al., 2023]. They can easily be adapted for new tasks or to new data distributions through minimal finetuning [Brooks et al., 2023; Zhang et al., 2023; Ruiz et al., 2023].

Beyond mere image synthesis, generative image models, and in particular diffusion models, have been studied as data augmentation engines. While most machine learning approaches treat the data as fixed and improve the learning algorithm, works such as [Jahanian et al., 2022; Chai et al., 2021; Azizi et al., 2023; Tian et al., 2023; Fan et al., 2024] fix the learning algorithm and augment the training data, using generative models to synthesize large amounts of synthetic training data.

In contrast, we present a new way to use generative models, with the goal of gaining insights about their training data.

4.3 Data Mining via Diffusion Models

Given a collection of images with assigned labels, our goal is to extract a small subset of visual elements from these images that best summarize the label inside the context of

the input dataset, or in other words that are highly typical of their label [Murphy, 2004]. The goal of our approach is to use generative probabilistic image synthesis models towards that end. We rely on finetuning a conditional stable-diffusion model [Rombach et al., 2022] trained for image synthesis, using it to extract a summary of the visual world. We start by reviewing diffusion models and the techniques we leverage in Sec. 4.3.1. In Sec. 4.3.2, we introduce our measure of typicality, which allows us to measure how the class label conditioning affects the synthesis of an image by the diffusion model. In Sec. 4.3.3, we describe how we aggregate typicality on patches to mine typical visual elements and cluster them to summarize the training data.

4.3.1 Preliminary

Diffusion models. Diffusion models are generative models trained to transform random noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}) \in \mathbb{R}^{H \times W}$ of height H and width W into a target prior distribution of images $p(\boldsymbol{x}) \in \mathbb{R}^{H \times W}$ [Sohl-Dickstein et al., 2015; Ho et al., 2020]. They achieve this by learning to revert a noising process called the *forward process* where noise is linearly interpolated, with a decreasing mixing strength $a_t \in [0, 1]$ to the input image $\boldsymbol{x}_0$ in different resolutions indexed by a fractional timestep index $t \in [0, 1]$:

$$\boldsymbol{x}_t^{\boldsymbol{\epsilon}} = \sqrt{a_t}\boldsymbol{x} + \sqrt{1 - a_t}\boldsymbol{\epsilon}. \tag{4.1}$$

The diffusion model $\epsilon_\theta(\boldsymbol{x}_t^{\boldsymbol{\epsilon}},\ t) \in \mathbb{R}^{H \times W}$ with parameters θ, is trained to predict the input noise ϵ added to that timestep, by minimizing the reconstruction loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{\epsilon}, t, \boldsymbol{x} \sim p(x)} \left[\left\| \epsilon_\theta(\boldsymbol{x}_t^{\boldsymbol{\epsilon}},\ t) - \boldsymbol{\epsilon} \right\|^2 \right], \tag{4.2}$$

At test time, the target distribution can be sampled by gradually passing an input noise through an iterative denoising process, known as the *backward process* [Ho et al., 2020; Song et al., 2021], which is based on an inversion of eq. 4.1:

$$\hat{\boldsymbol{x}}_{0,t} = \left(\boldsymbol{x}_t^{\boldsymbol{\epsilon}} - \sqrt{1 - a_t}\epsilon_\theta(\boldsymbol{x}_t^{\boldsymbol{\epsilon}},\ t) \right) / \sqrt{a_t}, \tag{4.3}$$

where $\hat{\boldsymbol{x}}_{0,t}$ is the denoised estimate at that timestep t. For this sampling procedure where noise is iteratively added and removed from $\hat{\boldsymbol{x}}_{0,t}$ from coarse to fine resolutions, various formulations have been proposed including DDPM [Ho et al., 2020] and

its non-markovian counterpart DDIM [Song et al., 2021] whose goal is to factor out stochasticity and speed up sampling.

Conditional Diffusion Models. In order to make diffusion models sample from a conditional distribution $p(\boldsymbol{x}|c)$, [Nichol and Dhariwal, 2021] train $\epsilon_\theta(\boldsymbol{z},\ t)$ on $p(\boldsymbol{x})$, and steer it during sampling towards c, using a trained classifier $d_\theta(c|\boldsymbol{x})$, by simply adding its gradient to ϵ_θ for a given class c according to $\boldsymbol{x}$: $\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t) + \lambda \nabla_{\boldsymbol{x}} d_\theta(c|\hat{\boldsymbol{x}}_{0,t})$, with a strength λ. In a later work [Ho and Salimans, 2021] showed that this is equivalent with training a diffusion model $\epsilon_\theta(z,\ t,\ c)$ on $p(\boldsymbol{x}|c)$, with c:

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{\epsilon},t,(\boldsymbol{x},c)\sim p(x|c)} \left[\|\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ c) - \boldsymbol{\epsilon}\|^2 \right], \tag{4.4}$$

while dropping it for a small percent of the time (10%) in order to learn an unconditional (or "null") distribution $p(\boldsymbol{x}|\varnothing)$, and during sampling replacing $\nabla_{\boldsymbol{x}} d_\theta(c|\hat{\boldsymbol{x}}_{0,t})$ with $\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ c) - \epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ \varnothing)$.

Latent diffusion models. Our work employs a variant of conditional diffusion models, trained with classifier free guidance, known as a *latent* diffusion model (LDM) [Rombach et al., 2022]. Instead of directly modeling the source data distribution $\boldsymbol{x}_0$ $p(\boldsymbol{x}|c)$, LDMs model the distribution of $\boldsymbol{x}_0$ in the latent space of a variational autoencoder $v_\phi(\boldsymbol{x})$ [Kingma and Welling, 2014]. Working in the latent space reduces the complexity of the data distribution. It thus significantly reduces both the number of parameters of the diffusion model and the amount of training samples necessary to learn a good model. As a conditioning they also use pretrained CLIP [Radford et al., 2021] text features $\tau_\phi(c)$, which are multimodal embeddings of text and images, trained on a large dataset of image-text pairs.

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{\epsilon},t,(\boldsymbol{x},c)\sim p(v_\phi(\boldsymbol{x})|\tau_\phi(c))} \left[\|\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ c) - \boldsymbol{\epsilon}\|^2 \right], \tag{4.5}$$

Diffusion Classifier. As conditional diffusion models are probabilistic models of the form $p(\boldsymbol{x}|y)$ they could be inverted in order to be used as classifier $p(y|x)$. Using the fact that the $-\mathcal{L}(\theta)$ is maximized as a lower bound of the log-likelihood, [Li et al., 2023a] through a standard application of the Bayes rule defined a classifier $p_\theta(c_i|x)$ for a set of labels c_i and an image $\boldsymbol{x}$ as:

$$p_\theta\left(c_i|\boldsymbol{x}\right) = \frac{1}{\sum_j \exp\left\{\mathbb{E}_{\epsilon,t}\left[L_t(\boldsymbol{x},\epsilon,c_i) - L_t(\boldsymbol{x},\epsilon,c_j)\right]\right\}}, \tag{4.6}$$

$$L_t(x,\epsilon,c) \equiv \|\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ c) - \boldsymbol{\epsilon}\|^2. \tag{4.7}$$

4.3.2 Typicality

For the purpose of mining we would like to define a typicality measure, that can enable us to sort visual elements coming from images of a specific class by how typical they are of that class. To define that measure we require that it can be computed both **(a)** efficiently, and **(b)** over different regions of the input image, so that we can discover visual structure. Given that $\varnothing$ is used to model the unconditional distribution, we can simply sum the denominator of eq. 4.6 only across the classes c and $\varnothing$ to:

$$p_\theta\left(c|\boldsymbol{x}\right) = \frac{1}{1 + \exp\left\{\mathbb{E}_{\epsilon,t}\left[L_t(\boldsymbol{x},\ \epsilon,\ c) - L_t(\boldsymbol{x},\ \epsilon,\ \varnothing)\right]\right\}}. \tag{4.8}$$

We can quickly observe that for the above expression to increase the denominator needs to maximize. Thus, for the purposes of only rank elements one only needs to compute the expression:

$$\mathbf{T}(\boldsymbol{x}|c) = \mathbb{E}_{\epsilon,t}[L_t(\boldsymbol{x},\ \epsilon,\ \varnothing) - L_t(\boldsymbol{x},\ \epsilon,\ c)], \tag{4.9}$$

where $\mathbf{T}$ is our derived measure of *typicality* of an image x given the ground truth class label conditioning c and the null conditioning $\varnothing$. Intuitively, an image is typical of a conditioning class label (*e.g.*, a country's name or a date) if the diffusion model is better at denoising the input image in the presence of that label than in its absence. However, note that instead of computing this measure across the whole image, one can compute it for any subregion π (down to the individual pixel), using a binary mask:

$$\mathbf{T}^\pi(\boldsymbol{x}|c) = \mathbb{E}_{\epsilon,t}[L_t^\pi(\boldsymbol{x},\ \epsilon,\ \varnothing) - L_t^\pi(\boldsymbol{x},\ \epsilon,\ c)], \tag{4.10}$$

$$L_t^\pi(\boldsymbol{x},\ \epsilon,\ c) \equiv \|\pi \cdot (\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ c) - \boldsymbol{\epsilon})\|^2. \tag{4.11}$$

While when computed for the whole image typicality ranks the same as a binary classifier, when computed per pixel it is equivalent to a measure known as pixel-wise mutual information, as we discuss in Appendix B (Sec. A). However, being able to compute typicality per patch is what enables us to discover visual elements.

4.3.3 Mining for Typical Visual Elements

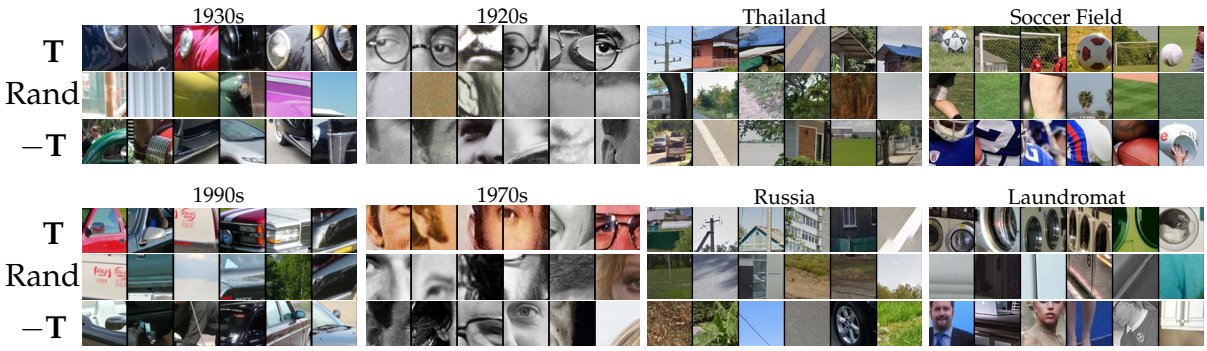
Patch-based analysis. To locate typical elements, we compute our typicality scores over all the patches π of an input image. This can be computed efficiently by computing $(\epsilon_\theta(\boldsymbol{x}_t^\epsilon,\ t,\ c) - \boldsymbol{\epsilon})$ and then using an average pooling operator with a fixed patch size. To extract "mid-level" patches we use a patch size of 50 for images with a base dimension of 256 and 64 for images with a base dimension of 512. Then, to identify the set of most typical visual elements for a dataset we pick the 5 most typical non-overlapping patches in each image according to the patch typicality, and select the 1000 most typical patches over all the dataset. Unlike [Li et al., 2023a], we find that reducing the sampled range of t to $[0.1, 0.7]$ improves the quality of our results, as the tails can contribute uninformative yet typical samples (see Sec. C of the Appendix B).

Clustering visual elements. We cluster the most typical patches using k-means [Lloyd, 1982] with 32 clusters. To cluster elements, we embed them with DIFT [Tang et al., 2023] features, computed at timestep $t = 0.161$ using our finetuned models. For visualization, we rank clusters by the median typicality of their elements in decreasing order and their elements by their distance to the centroid in increasing order.

Conditioning and finetuning. Given that our input dataset is labelled, we convert its labels to sentences: "A car/portrait from the {decade}s." for faces and cars ("A car/portrait." for the null conditioning $\varnothing$), "A Google StreetView image of {country}." for StreetView data ("A Google StreetView image." for the null conditioning $\varnothing$), and "An image of {scene}." for images of the Places dataset [Zhou et al., 2017a] (empty string for the null conditioning $\varnothing$). A latent diffusion model [Rombach et al., 2022] is then finetuned on the target dataset by optimizing the reconstruction loss (Equation 4.5) given the conditioning. We use Stable Diffusion V1.5 [Rombach et al., 2022] as a base model in all our experiments.

4.4 Experiments

We showcase the effectiveness of our approach in summarizing visual data in a wide variety of datasets. First, in Sec. 4.4.1, we introduce the datasets used in our experiments. Second, in Sec. 4.4.2, we evaluate the ranking produced by our typicality measure. Third, in Sec. 4.4.3, we discuss our main result, the mined visual summaries of the analyzed datasets, and compare it with [Doersch et al., 2012]. Finally, we discuss the limitations of our approach in Sec. 4.4.4.



(a) **CarDB** (b) **FTT** (c) **G^3** (d) **Places**

Figure 4.2: **Typical elements are informative of the conditioning label.** We visualize the top-6 patches ranked according to typicality (**T**) with respect to the conditioning class label, negative typicality (−**T**), and randomly (Rand.). The two rows correspond to different classes from each of the four datasets.

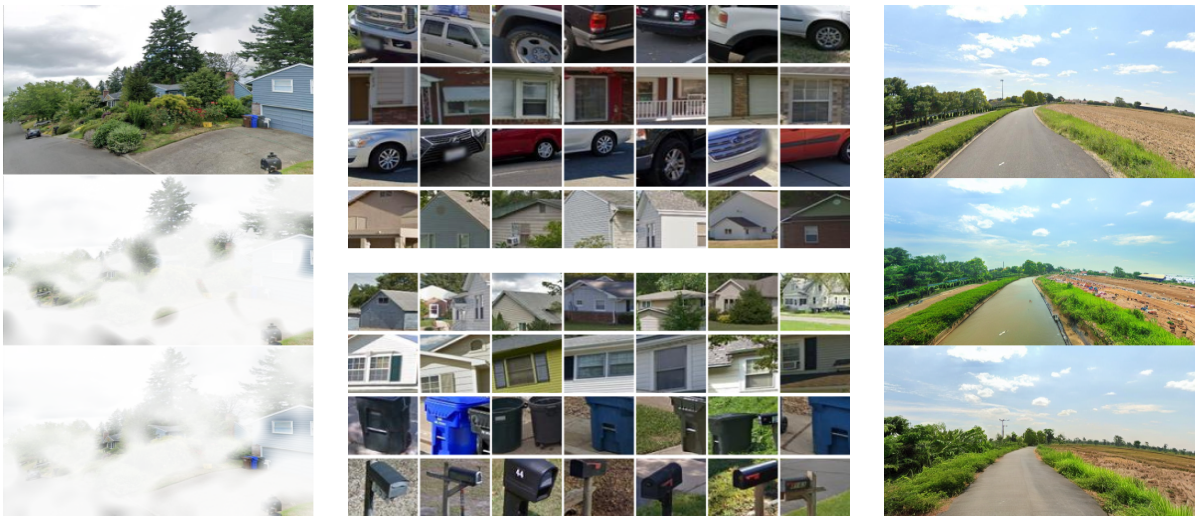
4.4.1 Datasets

We experiment with four diverse datasets. CarDB [Lee et al., 2013] and FTT [Chen et al., 2023] have already been used for image data mining and include a few tens of thousands of images. G^3 [Luo et al., 2022] and Places [Zhou et al., 2017a] are much larger with 344K and 1.8M images respectively, and to our knowledge have never been used for mining purposes.

Cars. The CarDB dataset [Lee et al., 2013] contains 10,130 photos of cars from 1920 to 1999, collected from cardatabase.net. They are labeled with creation years, which we bin into decades for our analysis. This dataset contains cars seen from various viewpoints and in diverse environments. As a result, extracting time-informative elements is challenging. We rescale all images to a height of 256 pixels while preserving their original aspect ratio.

Faces. The Faces Through Time (FTT) Dataset [Chen et al., 2023] contains 24,874 images of notable people from the 19th to 21st century, with roughly 1,900 images per decade, sourced from Wikimedia Commons. All photos come in 256x256 pixels.

Geo. The G^3 [Luo et al., 2022] dataset contains images obtained from crops of StreetView panoramas, diversely sampled worldwide, of which we selected 344,224 images, which we rescaled to 512x756 pixels. This dataset is challenging because of the small details that characterize a scene's appearance and scale. We focus on the 8 countries with the largest number of panoramas (United States, Japan, France, Italy, United Kingdom, Brazil, Russia, and Thailand) and select two countries with much



(a) Typicallity (b) Clusters (c) Parallel Translation

Figure 4.3: **Effect of finetuning**. **(a)** For the same USA image (top), finetuning changes the spatial allocation of typicality before (middle) and after (bottom) finetuning. **(b)** This results in different typical clusters (USA), which, after finetuning (bottom), select for more typical elements like mailboxes. **(c)** Translation (Sec. 4.5.1) of a picture of a road from France (top) to Thailand without finetuning (middle) suffers from data biases in the base model turning the road into a river and erasing utility poles. After finetuning on the G^3 dataset (bottom), the translated image is more consistent with the original.

fewer images (Nigeria and India). We finetune the network using all images from these countries, but we only mine a random subset of 1000 images for each country.

Places. The high-resolution version of the Places dataset [Zhou et al., 2017a] contains 1,803,460 million images from 365 place categories associated with their labels, with a minimum dimension of 512 pixels. For mining, we only use the validation dataset, which contains 100 images per scene category.

4.4.2 Typicality Measure Evaluation

Typicality score for patches. Fig. 4.2 shows the most and least typical patches according to our typicality measure and random patches from the four datasets. We note that the most typical patches are unique to each class and more discriminative than random patches, while the least typical patches are uninformative of the label c.

Effect of finetuning. Unsurprisingly, we found that finetuning the diffusion model on the dataset of interest was critical to the quality of our results. First, on a given image, finetuning changes the spatial distribution of typicality, prioritizing elements more

(a) **1920s** (b) **1930s** (c) **1980s**

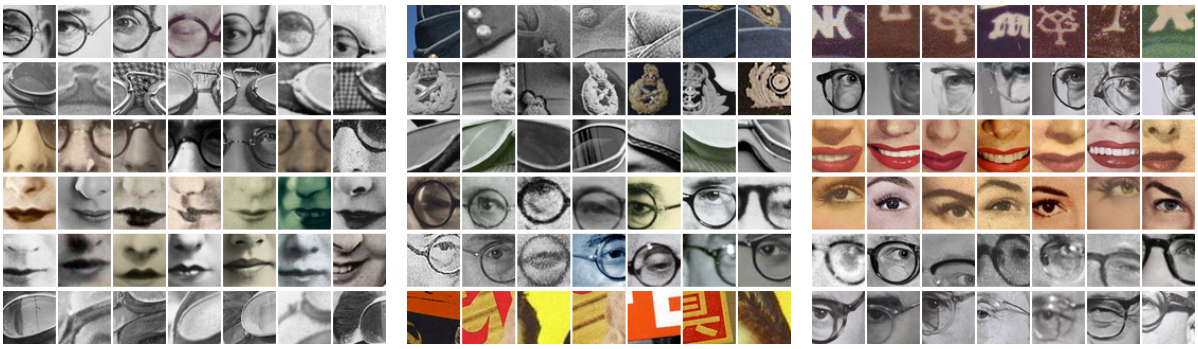
Figure 4.4: **Clusters of CarDB [Lee et al., 2013] visual elements.** Our visual summaries of typical car elements show elements unique to a period and elements that evolve with time. Evolving elements include the shapes of the car's body or headlights, which are parts of the 6 most typical clusters for most periods. More specific elements include running boards in the 1920s (**(a)**, 6th row) or large engine side grills in the 1930s (**(b)**, 3rd, 4th and 6th row). In the 1980s **(c)**, we observe two typical yet very discrete clusters of car design styles, of the curvy French 2CV (1-4 row) juxtaposed to the square American *chevy*-style cars (5-6 rows).

correlated with the training labels (see Fig. 4.3a). Second, in Fig. 4.3b, we show the most typical clusters identified before and after finetuning. The patches selected after finetuning avoid the biases in the training data of the base model and are more specific to the G ^3 dataset, identifying elements such as post-boxes. We also demonstrate this quantitatively in Sec. 4.5.3 for our application to X-ray images. Third, finetuning enables better translation between labels (see Sec. 4.5.1), as can be seen in Fig. 4.3c, allowing vegetation, roads, road tracks, and utility poles to be translated consistently across classes in the parallel dataset, which can be located in Appendix B (Sec. F).

4.4.3 Clusters of Typical Visual Elements

In this section, we analyze the visual summary of each dataset, obtained by clustering the typical visual elements for the different class labels. We demonstrate the mined summaries of Cars, Faces, Geo, and Scenes in Figs. 4.4, 4.5, 4.6, 4.7 respectively. For all datasets we show for selected class labels, the top-6 clusters ranked by median typicality of their elements. Inside each cluster elements are ranked by their distance to the centroid. The resulting clusters are analyzed inside the figure captions for ease of viewing. Complete clusters can be found in the Appendix B (Sec. F).

Comparison to [Doersch et al., 2012]. As the Matlab implementation of [Doersch et al., 2012] is obsolete and hardware-specific, we reimplement their method in Python and release this reimplementation with our code. In Fig. 4.8, we show the results of



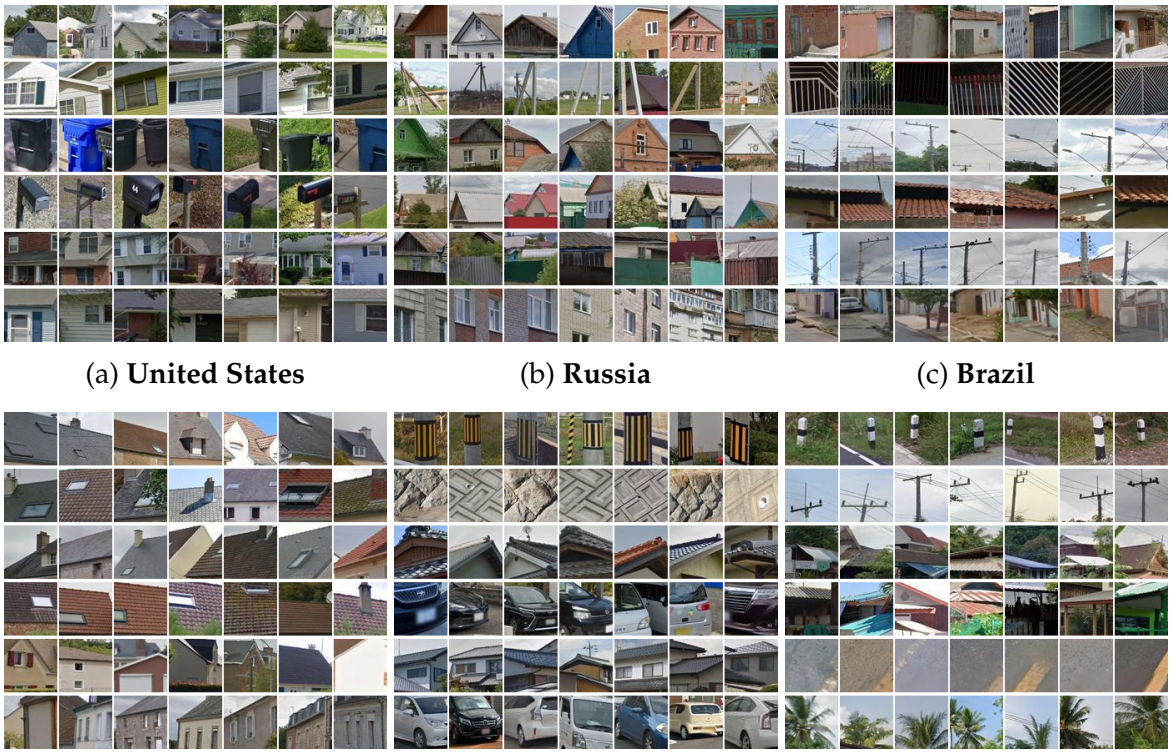
(a) **1920s** (b) **1940s** (c) **1950s**

Figure 4.5: **Clusters of FTT [Chen et al., 2023] visual elements.** Our cluster analysis of faces revealed that eyeglasses of varying designs are indicative of a portrait's decade throughout the history captured by FTT. Observing the 6 most typical clusters for the 1920s **(a)**, the 1940s **(b)**, and the 1950s **(c)**, we see how the shape of glasses is highly informative of each period. We also located fashion items that uniquely trended only in a particular period, such as aviator goggles in the 1920s (2nd row), military caps in the 1940s (1st and 2nd row), and baseball caps in the 1950s (1st row). Consistent with prior analysis [Ginosar et al., 2017], we also found clusters corresponding to smiles and makeup.

this approach when applied directly to the same mining subset of the G^3 dataset used by our approach. Similar to the original paper, we rank the trained detectors by *discriminativeness*, *i.e.*, the percentage of the top-50 final matches inside the positive set [Doersch et al., 2012], and for each we show its top 6 matches. The results produced with [Doersch et al., 2012] method demonstrate more textures, appear much less semantic, and contain much more similar elements than ours. Note that the results in the original [Doersch et al., 2012] paper do not show similar failures, and in particular much less vegetation, simply because the paper used a curated and non-publicly available dataset of images focused on selected cities extracted from Google StreetView.

4.4.4 Limitations

Although our method makes the first step towards utilizing generative models for data mining, it comes with limitations. We visualize our two main failure modes in Fig. 4.9. First, clustering elements using k-means can lead to mixed clusters containing different categories of samples (Fig. 4.9a) or produce repetitively similar clusters. Second, our method identified data artifacts (Fig. 4.9b) that are related to noisy printing or scanning of old photographs or post-processing artifacts of StreetView images, which are highly typical but irrelevant to our purpose. Interestingly, in the case of



(a) **United States** (b) **Russia** (c) **Brazil**

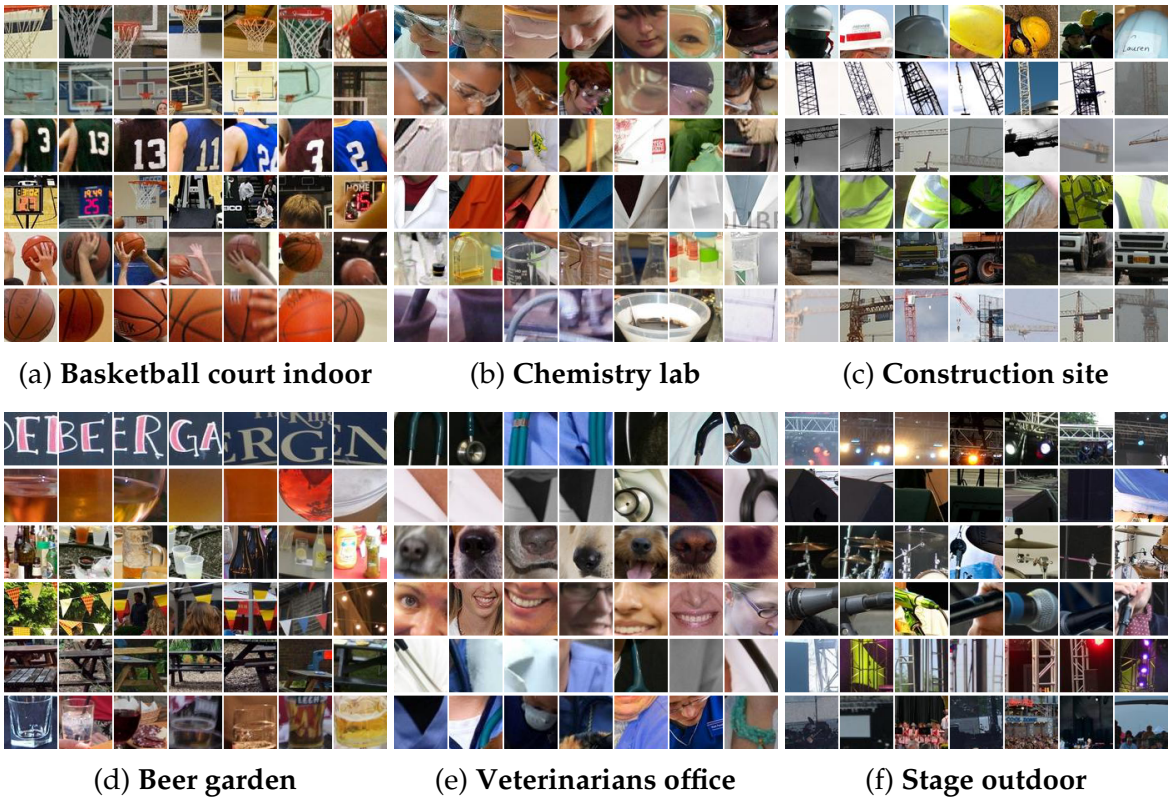
(d) **France** (e) **Japan** (f) **Thailand**

Figure 4.6: **Clusters of G^3 [Luo et al., 2022] visual elements.** Our geographic clusters show a wide diversity of typical elements across different countries. We found architectural elements such as roofs, facades, or windows among the most typical elements in all countries. For example, **(a)** the "double hung" American windows (2nd row), **(d)** French roof windows (1st-4th row), or **(f)** covered pathways in Thailand (4th row). Utility poles are ranked second in Russia and Thailand and 5th in Brazil. We also found typical objects that are unique to a single country, such as **(a)** American garbage cans and post boxes (3rd, 4th row), **(c)** protective guard rails in Brazil (2nd row), **(e)** Japanese electricity warning signs and exterior wall tiles (1st, 2nd row), and **(f)** Thai Bollards (1st row).

StreetView data similar artifacts are suggested in GeoGuessr [geoguessr, 2023] advice websites [geodummy, 2023; geohints, 2023; Plonkit, 2023], as shortcuts for geolocation.

4.5 Applications

Our typicality score allows us to explore three different applications. First, in Sec. 4.5.1, we translate geographical elements across locations and mine typical translations. Then, in Sec. 4.5.2, we show how our method can be used as a qualitative way of understanding bias in the sampled distribution of a diffusion model. Finally, in Sec. 4.5.3,



(a) **Basketball court indoor** (b) **Chemistry lab** (c) **Construction site**

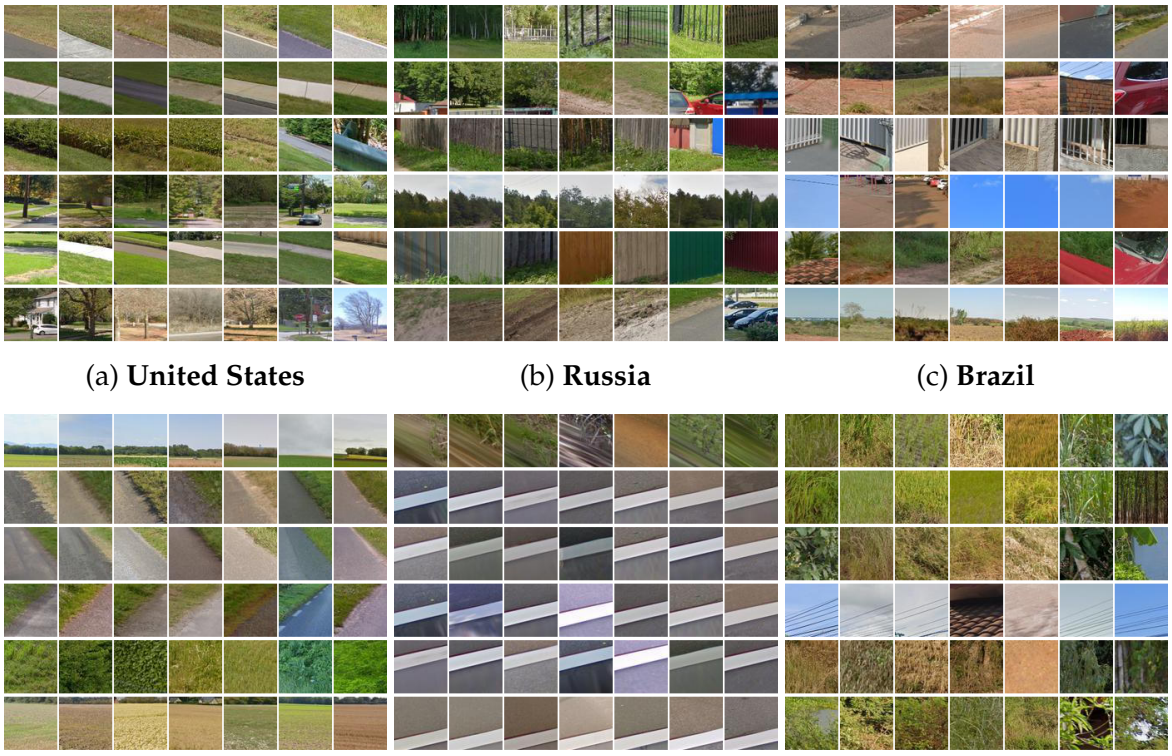
(d) **Beer garden** (e) **Veterinarians office** (f) **Stage outdoor**

Figure 4.7: **Clusters of Places365 [Zhou et al., 2017a] visual elements.** Unlike the other datasets we analyze, each class label correlates with objects of different categories in the scenes dataset, as different scenes contain objects of different categories. Yet, our approach can still summarize a large variety of complex scenes with their unique typical elements. For example, in basketball courts **(a)**, our approach locates the basket (1st row), the backboard (2nd row), the jersey numbers (3rd row), the shot clock (4th row), a shoot (5th row), and the ball (6th row). Our approach can still focus and summarize the most informative elements even in more cluttered scenes like an outdoor stage, chemistry labs, or beer gardens. For example, in the case of "outdoor stage" **(f)**, we see a lot of infrastructural elements, including lights and top rails (1st row), monitor speakers (2nd row), microphones (4th row), and side rails (5th row).

we show how disease localization emerges from typicality when training to generate frontal chest X-rays of patients, of various diseases.

4.5.1 Analyzing Trends of Visual Elements

Having a diffusion model finetuned on a dataset of interest enables further applications that were not possible with previous image mining approaches [Doersch et al., 2012; Lee et al., 2013; Chen et al., 2023; Ginosar et al., 2017]. One new application is the summary of variation of typical visual elements across different classes. As a case study, we use the G^3 dataset to discover and summarize how *co-typical* elements,



(a) **United States** (b) **Russia** (c) **Brazil**

(d) **France** (e) **Japan** (f) **Thailand**

Figure 4.8: **Doersch *et al.*, 2013 [Doersch et al., 2012] results on G^3 [Luo et al., 2022].** See text for details.

such as windows, roofs, or license plates, vary across locations. We start by using our finetuned diffusion model to create a "parallel dataset", by translating all the images in our mining dataset to all location, and then define a measure of co-typicality.

Generating a parallel dataset. We first use Plug and Play [Tumanyan et al., 2023] to translate input images from one location to another, which we denote by $x^{c_0 \to c}$, where c_0 is the initial country and c is the target country. We translate 1000 images for each of the 10 selected countries to all others, resulting in 100K images, which we refer to as our parallel dataset. Performing translation using our finetuned model is critical for keeping scene elements consistent, as seen in Fig. 4.3c. In Appendix B (Sec. E) we show how performing semantic segmentation for each image and its translations to different countries enables measuring statistical trends. For example, we can measure that translations to Thailand or Brazil add many potted plants, and translations to Nigeria add dirt roads and people. This trends can be visually confirmed on our parallel dataset (see Appendix B Sec. E)

(a) Mixed Clusters (b) Mining Dataset Artifacts

Figure 4.9: **Limitations.** The two most common failure modes we observe are: **(a)** issues in clustering, for example, clusters that contain diverse visual content, or multiple clusters that correspond to the same concept; **(b)** typicality highlighting artifacts of the dataset. Discovering artifacts is an expected behavior and can be useful for some applications.

Mining typical transformations across location. To further analyze our parallel dataset, we define a cross-location typicality measure to mine a parallel translation of patches across locations. We define the co-typicality $\bar{\mathbf{T}}$ as the median typicality across location:

$$\bar{\mathbf{T}}(x) = \underset{c \in C}{\text{med}} \left[\mathbf{T}(x^{c_0 \to c}, c) \right], \tag{4.12}$$

where c_0 is the true label of the patch x and the median is computed over all countries in our set of 10 analyzed countries, denoted as C.

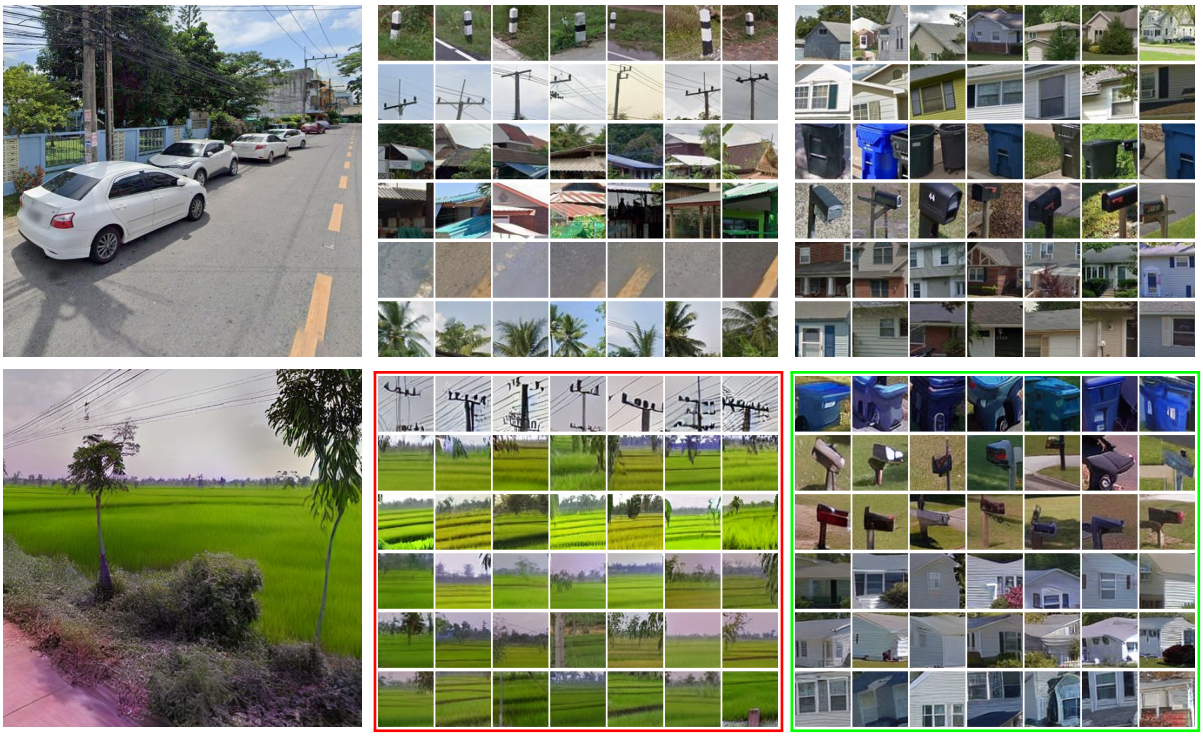
We can now ask: What visual elements are typical of a certain place and whose translation remains typical of another location? Instead of ranking single patches, we now rank a whole sequence of $|C|$ patches translated across locations according to $\bar{\mathbf{T}}$. We represent this sequence by concatenating the DIFT features of each patch [Tang et al., 2023]. To facilitate clustering, we first project the DIFT features of each patch from 1280 dimensions to 32 dimensions using UMAP [McInnes et al., 2018]. To keep the same proportion of typical patches to the number of analyzed images/sequences as in Sec. 4.4.3, we cluster the 10,000 visual elements with the highest co-typicality.

We display our results in Fig. 4.10, where for 6 selected clusters, we show in rows the four translated sequences closest to the cluster mean, highlighting in red the original image in each sequence. On the left column of Fig. 4.10, we show changes in typical architectural elements, such as gables, roofs, and windows. In contrast, on the right we show regulation-related elements, such as road tracks, utility poles, and license plates. Our approach allows us to both locate and visualize how common visual elements would vary from place to place, even though an exact match may not exist in the original data. For example, roofs typically turn dark brown when translated to the UK and black when translated to Japan.

Figure 4.10: **Clustering typical translations of elements across countries.** Ranking translated visual elements according to $\bar{\mathbf{T}}$ and clustering the translated sequences results in groups of elements with similar variations. We show elements from 6 selected clusters out of 32. The source image for each sequence is highlighted in red. See text for details.

4.5.2 Mining Bias in Generation

Our goal in this section is to use mining as a way of interpreting the generative performance of the diffusion model itself. There is a variety of metrics that concern measuring the performance of generative models, including for example Inception Score [Salimans et al., 2016], FID [Heusel et al., 2017], and Precision-Recall [Sajjadi et al., 2018; Kynkäänniemi et al., 2019]. Especially for diffusion models different sampling procedures can reveal different performance of aligning with the training data density for the same model, as demonstrated for two-dimensions in Fig. 1 of [Karras et al., 2025]. This indicates that while the diffusion model may already have knowledge of the true density, sampling may bias it towards certain outputs. In Fig. 4.11 we show how our typicality measure can be used to mine the sampling bias of a diffusion model. We start by sampling synthetic images of an equal amount of 1000 images per class to our real sampling dataset. While there is a clear difference between sampled and real images of Thailand (Fig. 4.11a), it's hard to ground it simply by comparing between



(a) Real vs Synthetic (b) Thailand (c) United States

Figure 4.11: **Mining Sampling Bias. (a)** Example of real image from Thailand (top) and synthetic (bottom). **(b)** Per row the corresponding visual summary for Thailand in the same amount of real and synthetic data. **(c)** The same visual summary but for the United States.

real and synthetic images. Instead, we can compare the mined visual summaries that are extracted using the same model that we used to generate synthetic data (Fig. 4.11b). This can reveal that when sampling Thailand (with c.f.g. of 7.5) the model will place a disproportionately higher amount of rice fields, while the same model is capable of detecting more diverse objects such as bollards, architecture and road-tracks in our real data. This procedure make the bias of the model highly interpretable. Note, that this observation is not common amongst all classes, as can be seen for example in the case of the US (Fig. 4.11c), where the typical visual summaries of real and synthetic images are more similar. Yet, the mined summaries of the real data contain much more distinct elements than the generated ones, which seems related to the diversity-fidelity trade-off of classifier-free-guidance (see Fig. 5 of [Ho and Salimans, 2021]). Examples across all of our ten countries, and different summaries for a range of c.f.g. λ values can be found in Appendix B (Sec. D). Interestingly, this experiment adds another dimension to the concept of *Latent Reading* which I introduced in [Siglidis, 2022], as the practice of understanding cultural data by interacting with a model trained to reproduce them.

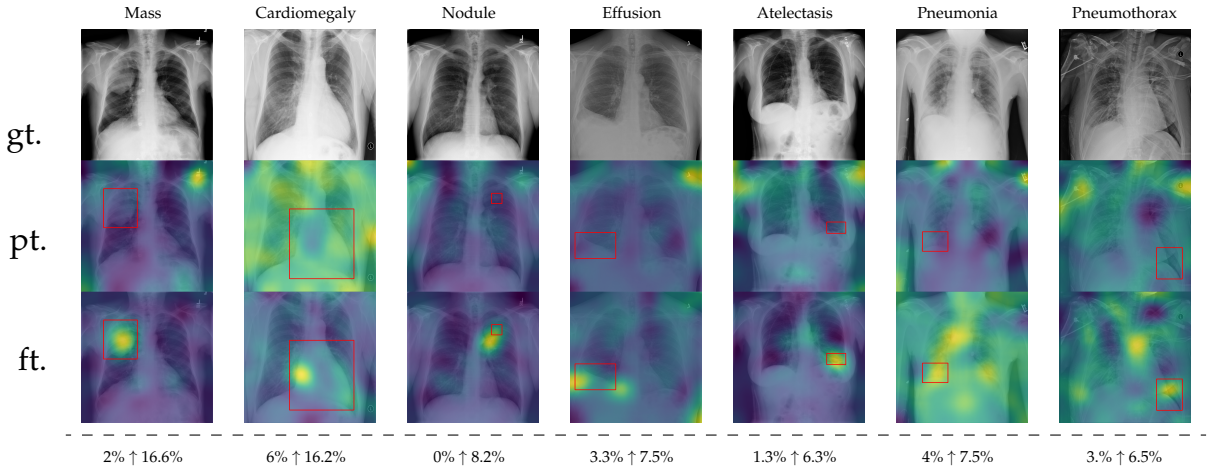


Figure 4.12: **Localizing abnormal areas in medical images.** We visualize typicality when finetuning our model on the CXR8 dataset of thorax diseases [Wang et al., 2017]. After finetuning (ft.), we can see a clear focus of the typicality score on expert annotated areas (red boxes) for each disease, while initial predictions from the pretrained Stable Diffusion V1.5 model (pt.) are mostly noise. Images are ordered by AUC-PR after finetuning [Arun et al., 2021]. With ↑ we delimitate performance before and after finetuning, in the last row.

On one hand, it shows how data is indeed summarized by generative models, which facilitates their analysis. On the other, it shows that the sampling procedure adds bias that doesn't simply truncate non-frequent elements but changes their density. Instead, our method could be used to apply generative models to real data, towards the same goal.

4.5.3 Analysis of Medical Images

In Sec. 4.4.2, we discussed how typicality helps locate relevant patches for an input label. In this section, we test this idea on completely different images: X-rays of patients who may suffer from a combination of various thorax diseases. We finetune Stable Diffusion on the ChestX-ray8 dataset [Wang et al., 2017] containing 108,948 frontal-view X-ray images annotated with 14 single-word disease-name labels. Experts annotated a test set of 879 images with 7 diseases with rectangular regions of interest (ROI) for each disease. For each image, we compute typicality per latent pixel, interpolate the resulting typicality to the input dimension, and blur the resulting typicality map for visualization. In Fig. 4.12, we show the resulting typicality maps together with the ROI annotation before and after finetuning. We can observe that finetuning clearly improves the localization. To quantify this effect on average, we compute the area under the precision recall-curve [Arun et al., 2021] (AUC-PR) of typicality associated

with the annotated ROIs. To do this we binarize typicality across 1000 uniformly (*i.e.*, log-linearly) spaced thresholds and for each count true (typical and inside the ROI) and false (typical and outside the ROI) positives pixels. As reported in Fig. 4.12, we see consistent improvement of this measure when finetuning the network (from 3.2% to 9.6%), ranging from +3.5% for Pneumonothorax (from 3% to 6%) to +14.6% for Mass (from 2% to 16.6%), which are respectively the least and most localized diseases. Similar to Sec. 4.4.3, finetuning uses only image labels without localization supervision.

4.6 Conclusion

This chapter, presented a novel use of diffusion models as visual mining tools. It defined a typicality measure using a pretrained stable diffusion model finetuned for conditional image synthesis. This measure of typicality was used to mine visual summaries of four datasets, tagged by year or location. We then showed that the same typicality measure can be extended in discovering trends when translating visual elements across location, making interpretable summaries of the sampling bias of diffusion models, and even localizing abnormalities in medical data. In summary, this chapter presented a novel approach to image data mining, enabling scaling to datasets significantly more extensive and diverse than those showcased in prior works as demonstrated by our experiments.